HZ BOOKS
华章科技

Web开发
技术丛书

深入理解Bootstrap

徐涛 著

Understanding Bootstrap

- 资深Web技术专家经验结晶，根据Bootstrap最新版本撰写，内容系统、分析深入、实战性强，前端工程师必备
- 授人以鱼，全面讲解各功能组件的使用方法，以及对现有插件和扩展的二次开发；授人以渔，深入分析其架构思想和源码实现，以及如何开发自定义的完整插件和扩展

机械工业出版社
China Machine Press

Web开发技术丛书

深入理解Bootstrap

徐涛　著

ISBN：978-7-111-46479-2

本书纸版由机械工业出版社于2014年出版，电子版由华章分社（北京华章图文信息有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @研发书局

腾讯微博 @yanfabook

目　录

前言

为什么要写这本书

Bootstrap是一个非常受欢迎的前端开发框架，笔者在其1.0版本刚刚发布时就开始使用了。该框架极大地提高了我们团队的开发效率，同时也规范了团队成员在使用CSS和JavaScript方面的编写规范。

Bootstrap的强大之处在于它对常见的CSS布局小组件和JavaScript插件都进行了完整且完善的封装，使得开发人员（不仅是前端开发人员）信手拈来，轻松使用。它解决了广大后端开发人员的难题，学完以后，即使是那些只会.NET和Java的高手，也可以在没有前端开发人员的情况下独立开发一个比较精美的Web系统。当然，专业的前端还是很重要的，因为只有专业的前端才可解决更为专业的前端问题，比如封装库和框架等。

编写本书的初衷是培训公司内部所有的Web开发人员，以便所有的人都能够快速制作出精美的Web页面。在经历了两个比较大型的项目后（基于最新版Bootstrap　3），笔者对各种现实环境中曾经遇到过的问题和解决方法进行了整理，并融入了这本原本是公司内部培训教材的图书中。笔者希望以这种方式，把自己掌握的知识和在实战中总结的经验分享给大家，以便提高大家的学习效率。

本书的主要内容和特色

本书是基于最新的Bootstrap 3进行编写的，以实例讲解和源码分析为主要的讲解方式，所以在本书里列举了大量的示例以及与之相对应的源码，以便读者能够彻底了解每个小组件（以及各种用法）背后的原理。

在对CSS组件和JavaScript插件进行分析之后，本书提供了3章的实战内容，首先是对现有组件和插件进行了二次扩展，然后是根据Bootstrap架构思想开发了自己的完整插件，最后是一个组合应用的实例。

另外，本书还添加了很多Bootstrap的潜在用法，以及每个组件在日常使用时的注意事项。

关于本书的内容，这里再多说几句。首先，组件（或插件）是用来使用的，而架构思想是用来理解和创建组件（或插件）、解决疑难杂症的，所以读完本书以后，你可能得到两种结果。

结果1　组件和插件都会使用了，但是没有很好的架构思想。那就做一个制作网页的熟手吧。不是因为Bootstrap很难，而是因为你对相关知识（CSS3、jQuery、JavaScript代码）的掌握还不牢靠。如果你想完全理解它的思想，这就需要把第1章用于理解源码分析的必备知识完全吸收以

后（从别的书上再多学一点会更好），再回过头来阅读第2章，然后随便找个组件（或插件）研究一下，相信就没有问题了。

结果2　架构思想都完全理解了。那么你下一步的工作，除了指导你的小伙伴们使用Bootstrap以外，还要帮他们解决疑难杂症。若有机会，可尽量尝试创建具有自己风格的组件或插件。

目标读者

本书没有对目标读者做任何限制，初中高级读者均适合阅读。因为书中的内容由浅入深，涉及了各个层面的读者，相信各个层面的读者都能从本书中获益。

❑如果你是初级开发人员，本书丰富的示例会让你很快上手Bootstrap框架，并由此晋级到中高端的水平。

❑如果你是对CSS、JavaScript比较熟悉的中级开发人员，本书的源码分析部分将为你提供详细的分析步骤，包括设计思想、实现方式、弊端等，为你晋级高级水平提供详细的指导。

❑如果你已经是专业的前端开发人员，相信本书的源码分析、组件（插件）扩展、全新插件开发，以及实战部分会为你提供一些更开阔的思路。

想要学习如何编写出优雅而又结构化良好的代码吗？相信这本书就是为你准备的。

如何阅读本书

读者在阅读本书的过程中，需要注意以下几个事项：

❑如果你不太熟悉CSS和JavaScript（或jQuery），却想学习如何使用Bootstrap，建议你忽略1.5节～1.7节，这些小节是源码分析的基础。

❑如果你在阅读第2章Bootstrap架构思想时有点迷茫，请不要烦躁，可以在阅读完第3章～第5章以后，回过头来再次阅读第2章，相信那时你对于框架思想的理解就会有不同的效果了。

❑希望读者不要急于学习后面的实战章节，在完全熟悉现有Bootstrap组件的使用方法并理解了架构思想后，再进行实战部分的学习，效果比刚开始就学习实战会好得多。

本书约定

本书使用下列约定：

❑Bootstrap文件：一般默认是指普通的CSS文件或JS文件，而非压缩后的*.min.css或*.min.js文件，因为本书有大量的章节要进行源码分析。

❑Bootstrap CSS：一般情况下表示Bootstrap的CSS文件（bootstrap.css），特殊情况下指CSS框架集合。

❑Bootstrap JS：一般情况下表示Bootstrap的JS文件（bootstrap.js），特殊情况下是指与单个插件对应的JS文件（比如，实现Dropdown插件的dropdown.js文件）。

❑代码运行浏览器：用火狐浏览器20.0.1版本运行示例代码，特殊情况会使用IE，届时会做说明。

代码示例

本书的源码分析采用如下形式（第一行注释里的行号是该段代码在Bootstrap.css文件里的行号）：

```
// 源码307行

img {

vertical-align: middle; /*垂直居中*/

}
```

资源和勘误

第6~8章为实战部分，笔者将实战过程中的源代码进行了打包整理，读者可到笔者的博客上下载，下载地址是：http://files.cnblogs.com/TomXu/BootstrapInDepth.rar。当然，也可到华章网站上下载：http://www.hzbook.com。

读者在阅读的过程中，发现任何错误和表述不准确的地方，欢迎在笔者的博客上留言，以便再版时进行修订，在此多谢了。

另外，在阅读的过程中，有任何不够明白或者觉得难以理解的内容，可以随时留言给我，我们可以线上讨论。

Bootstrap是一个非常简单的框架，相信经过几周的学习，大家就会完全掌握它。如果你在学习的过程中有任何问题，都可以发邮件给笔者（或者直接在博客上留言），笔者会尽最大努力帮你们解决问题。

电子邮件：tomxu@outlook.com

博客地址：http://www.cnblogs.com/tomxu

致谢

首先要感谢机械工业出版社的杨福川先生，没有他的支持和鼓励，就不会有本书。在写本书的过程中，杨先生为我提供了很多重要的想法和灵感。同时还要感谢本书的编辑，正是因为他们的辛苦工作才使得本书能迅速推出。

最后，要感谢我的妻子韩梅，她非常支持我的写书工作，每天无怨无悔地照顾儿子，为我腾出了很多写作时间。

第1章　入门准备

本章简单介绍了Bootstrap的框架，以及各种入门的基本操作。另外还介绍了CSS、jQuery、JavaScript的一些基本用法，作为在后续的章节里对源码进行分析的基础。

1.1 框架简介

Bootstrap是目前最流行的前端开发框架，由Twitter的两位前员工Mark Otto和Jacob Thornton在2010年8月份创建。它是一套基于Less的前端开发库（最新版也包含了Sass源码），提供了很多常见并常用的各种CSS和JavaScript合集，以便开发人员随时上手。目前最新版本是3.1.0。

Bootstrap内置了非常多的漂亮样式，即便是非专业的前端开发人员也能轻易使用。它秉承了一切从简的风格，使得开发人员能够毫无顾虑，放心使用，而无须担心这个div的高度、那个span的宽度等细枝末节的问题。即使没有设计师的团队，也能够使用这套框架迅速构建一个网站原型，甚至是构建一个生产环境的网站。

截止到目前，Bootstrap在Github上已经有5142个Watch、64207个Star、23019个fork，在经受了千万用户的考验之后，如今的Bootstrap已经非常强大了，其如此受欢迎的原因是：首先，Bootstrap系出名门（Twitter），代码开源，久经考验，可减少测试的工作量（站在巨人的肩膀上，我们不需要再重复“造轮子”）；其次，Bootstrap的代码有着非常良好的代码规范（本书会用相当一部分章节来分析其源码），从中可以学到很多知识。所以使用Bootstrap作为前端框架创建项目，其日后代码的维护自然也会变得非常容易。

Bootstrap提供了如下重要的特性：

❑一套完整的基础CSS插件。

❑丰富的预定义样式表。

❑一组基于jQuery的JS插件集。

❑一个非常灵活的响应式（Responsive）栅格系统，并且崇尚移动先行（Mobile First）的思想。

Bootstrap默认提供了大量的插件和合集，代码非常简洁，并且易于修改，你完全可以在其基础之上修改成任何自己想要的样子。同时这也是Bootstrap真正强大的地方，这些非常不错的插件，包括对话框、下拉导航等，使得Bootstrap不但功能完善，而且十分精致，正在成为众多jQuery项目默认的设计标准。这使得工作效率得到了极大的提升。

从V3.1.0开始，Bootstrap的License授权改成了MIT协议。MIT是目前最为宽松的协议，大家可以放心地在各种商业环境中使用它。

另外，由于Bootstrap的火暴，很多贡献者围绕Bootstrap也开发了大量优秀的插件，其中最有名的就是Font Awesome。它是一套icon字体，提供了丰富的icon供你选择，最新的3.0版包含了多达369个icon图标（第9

章中讲解Font Awesome）。

由于框架的发展，该框架的开发团队也逐步扩大，其中Core Team已经从2人扩展到4人了。另外，随着Sass的不断强大，Sass Team也成立了。表1-1和表1-2所示是相关成员的基本信息。

表 1-1 Core Team 成员

姓 名	URL	Twitter
Mark Otto	https://github.com/mdo	@mdo
Jacob Thornton	https://github.com/fat	@fat
Chris Rebert	https://github.com/cvrebert	@cvrebert
Julian Thilo	https://github.com/juthilo	@juthilo

表 1-2 Sass Team 成员

姓 名	URL	Twitter
Thomas McDonald	https://github.com/thomas-mcdonald	@thomas-mcdonald
Gleb Mazovetskiy	https://github.com/glebm	@glebm
Tristan Harward	https://github.com/trisweb	@trisweb
Peter Gumeson	https://github.com/sporkd	@sporkd

注意

Mark Otto目前加盟了GitHub，而Jacob Thornton则加入了由Twitter的联合创始人Biz Stone和Evan Williams创立的Obvious孵化器公司。

Less是一种CSS预处理技术，它是一种动态样式语言，属于CSS预处理语言中的一种，它使用类似CSS的语法，为CSS赋予了动态语言的特性，如变量、继承、运算、函数等，更方便CSS的编写和维护。它可以在多种语言环境中使用，包括浏览器端、桌面客户端、服务端。由于本书不是专门讲解Less的书，所以在此不做过多的介绍，详情请访问Less中国官网（http://www.lesscss.net）。

1.2 新手入门

在下载Bootstrap框架文件之前，首先确保要安装一个好用并适合自己的代码编辑器。如果你已经是某个方面的开发人员，应该已经安装了相应的IDE开发工具了，比如.NET开发人员常用的Visual Studio、Java开发人员常用的Eclipse等。这些IDE开发工具都能够对HTML、CSS、JavaScript代码进行很好的编辑。

除此之外，如果非要推荐一些编辑器的话，我推荐大家使用EditPlus或者Sublime Text，这两个编辑器都有非常不错的高亮着色和智能提示功能。

Bootstrap框架的文件和源码可以在其官方网站（www.getbootstrap.com）下载。打开网站的首页，单击Download Bootstrap按钮，跳转到下载页面，可以看到3个下载链接，如图1-1所示。

图1-1 Bootstrap下载页面

❑Download Bootstrap。从该链接下载的内容是编译后可以直接使用的文件。默认情况下，下载后的文件分两种：一种是未经压缩的文件bootstrap.css，一种是经过压缩处理的文件bootstrap.min.css。一般网站正式运行的时候使用压缩过的min文件，以节约网站传输流量；而我们进行开发调试的时候往往使用原始的、未经压缩的文件，以便进行debug跟踪，就像jQuery的使用方式一样。

❑Download source。从该链接下载的是用于编译CSS的Less源码，以及各个插件的JS源码文件。默认情况下，本书将基于未压缩的bootstrap.css和bootstrap.js文件进行源码讲解，不会涉及与Less相关的内容，比如nav功能模板就单独建立一个navs.less文件，或者具有弹窗功能插件的modal.js文件。

❑Download Sass。从该链接下载的是用于编译CSS的Sass源码，以及各个插件的JS源码文件。默认情况下，本书也不会涉及与Sass相关的内容。

本书所有涉及源码的讲解和分析均基于Bootstrap V3.1.0版本的CSS和JS文件。

在CSS源码分析方面，会涉及CSS代码、Less源码、Sass源码，但不会针对Less源码和Sass源码进行过多的解释，读者大概能看懂就行了，主要还是阅读生成后的CSS源码。所以建议大家阅读完本书提及的bootstrap.css文件后，再进行Less源码和Sass源码的延伸阅读。

但是，有一点要记住：经常使用IDE的源码搜索功能，因为源码分析的时候不会处处提及行号。

在JavaScript插件方面，将以单一插件的功能源码为准进行讲解，读者可以下载第二个（或第三个）链接的源码进行阅读，或者可以直接在GitHub网站上在线阅读这些源码，目录分别是js和less：
https://github.com/twbs/bootstrap

1.3 文件结构

解压下载的压缩包（编译版），可以看到如下结构的文件夹和文件。这些文件按照类别放到了不同的文件夹内，并且提供了压缩与未压缩两种版本。

```
dist/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    └── glyphicons-halflings-regular.woff
```

由上述文件结构可以看出，Bootstrap3.x版和2.x系列版本有一个很大的区别，就是2.x系列版本用于展示icon小图标的.png图片不见了，取而代之的是fonts目录的4种类型的字体文件。我们称这种方式为@font-face版本的icon实现，该字体来自glyphicons.com网站，并得到免费授权。

所谓@font-face，其实是通过CSS里的@font-face语法，将安全的Web字体（Web Font）即时下载到客户端，从而进行引用显示。通过这种技术显示图标的好处是，图标可以被任意缩放、改变颜色，你需要做的只是像修改文字样式那样修改图标样式即可（关于@font-face的详细解释和用法，请阅读4.1节）。

其他文件结构和之前的2.x系列版本一样，bootstrap.min.css和bootstrap.min.js是压缩后的文件，用于生产环境使用，而普通的bootstrap.css和bootstrap.js是用于开发环境进行调试、分析的。可以将该目录结构拖拽到任何Web项目直接使用。

你也可以根据自己的程序结构，对上述的css、js文件夹名称进行随意重命名，但是不能对fonts文件夹进行重命名，因为CSS文件里的源码使用了相对路径（../fonts/），如果重命名了，那就读取不到这些字体文件了。

建议大家在阅读本书的过程中，使用未经压缩的文件，以便进行分析、阅读、学习。

注意

❑Bootstrap所有的JavaScript插件都依赖于jQuery库，要确保在使用这

些功能的时候引用相应的jQuery文件。

❑字体文件是使用Mac下的应用软件ImageOptim（http://imageoptim.com）对.png图片进行压缩得到的。

1.4 HTML标准模板

如下代码是使用Bootstrap框架的最基本HTML代码，可以在此基础上进行自己的扩展，只需要确保文件引用顺序一致即可。HTML标准模板如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>Bootstrap基础模板</title>
<!— Bootstrap —>
<link href="css/bootstrap.min.css" rel="stylesheet">
    <!— 以下2个插件是用于在IE8支持HTML5元素和媒体查询的，如果不用可移除 —>
    <!— 注意:Respond.js不支持file:// 方式的访问 —>
    <!—[if lt IE 9]>
      <script src="https:// oss.maxcdn.com/libs/html5shiv/3.7.0/html5shi
</script>
      <script src="https:// oss.maxcdn.com/libs/respond.js/1.4.2/responc
</script>
    <![endif]—>
</head>
<body>
    <h1>Hello, world!</h1>
    <!— 如果要使用Bootstrap的JS插件，则必须引入jQuery —>
    <script src="https:// code.jquery.com/jquery.js"></script>
    <!— Bootstrap的JS插件 —>
    <script src="js/bootstrap.min.js"></script>
</body>
</html>
```

由上述模板代码可以看出，3.x和2.x版本相比，有一个很大的区别，就是多了以下一行代码：

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

这就是我们前面提到的，Bootstrap从3.0版本开始全面支持移动平台了，并将贯彻移动先行（Mobile First）的宗旨。上述代码的意思是，默认情况下，UI布局的宽度和移动设备的宽度一致，缩放大小为原始大小。对于移动代码的处理，还有另外一种形式，代码如下：

```
<meta     name="viewport"     content="width=device-width,     initial-
scale=1, maximum-
        scale=1, user-scalable=no">
```

上述代码表示：强制让文档的宽度与设备的宽度保持1∶1，文档最大的宽度比例是1，且不允许用户单击屏幕放大浏览。尤其要注意的是，content里多个属性的设置一定要用分号和空格来隔开，如果不规范将不

会起作用。

官方网站还提供了各种布局的基本模板供大家使用，大家可以到如下地址进行访问查看：http://getbootstrap.com/getting-started/

通过对这些模板的模仿和练习，你将会发现Bootstrap的强大。

1.5　CSS基本语法

本节主要介绍Bootstrap里用到的CSS语法，以便在源码分析时更容易理解和学习，如果仅是为了使用Bootstrap框架（而非进行完整的源码分析，以便开发各种CSS和JS插件）的话，则可以忽略本节。但本节的语法都是比较基础的，巩固一下有好处。

Bootstrap的CSS组件的核心就是选择器的定义以及在各自优先级上的处理。由于大部分的选择器都很常见，所以就不做单独说明了，只列举一下Bootstrap用到的知识点。

1.5.1 优先级

如何确定CSS的优先级？这里我们要先引入一个机制，分别用4个数字（a，b，c，d）表示优先级组合，比如1，1，1，1和0，1，0，1。它们的意思分别是：

❑第一个数字（a）表示style属性，优先级最高。由于一般都是class样式，所以该值一般都是0。

❑第二个数字（b）是该CSS选择器上的id数量的总和，一般都是1个。

❑第三个数字（c）是用在该CSS选择器上的其他属性CSS选择器以及伪类的总和。这里包括class（.btn）和属性CSS选择器（比如li[id=red]）。

❑第四个数字（d）计算元素（就像table、p、div等）和伪元素（就像first-child等）。

❑通用CSS选择器（*）是0优先级。

❑如果两个CSS选择器有同样的优先级，在样式表中后面的那个起作用。

下面有几个例子和对应的优先级，如表1-3所示。

表 1-3 选择器和所对应的优先级

选 择 器	优 先 级
#menu h2	0，1，0，1
h2.title	0，0，1，1
h2+p	0，0，0，2

同理，计算下面两个选择器的优先级：

```
#leftbar li#first { color: red; }
#leftbar li:first-child{ color: blue; }
```

结果肯定是第1个比第2个优先级高，因为第1个的优先级是0，2，0，1，而第2个是0，1，0，2。

1.5.2 选择器

每一条CSS样式的定义都由两部分组成，形式如下：选择器{样式}。在{}之前的部分就是“选择器”。“选择器”指明了应用这些“样式”的网页元素。

1.属性选择器

Bootstrap的CSS组件里使用了很多属性选择器，比如[data-toggle^=button]、[data-toggle=toggle]等。属性选择器有多种用法，表1-4列出了这些用法的区别。

表 1-4 常见属性选择器用法

选 择 器	用 法
[att=value]	该属性有指定的确切值
[att~=value]	该属性值必须是多个用空格隔开的值，比如，class="title featured home"，而且这些值中的一个必须是指定的值 "value"
[att|=value]	属性的值就是 "value" 或者以 "value" 开始并立即跟上一个 "-" 字符，也就是 "value-"，比如 lang="zh-cn"
[att^=value]	该属性的值必须以指定的值开始
[att$=value]	该属性的值必须包含特定值（而无论其位置怎么样）
[att*=value]	该属性的值必须以特定值结束

2.子选择器

CSS里的子元素用符号“>”表示。如下示例是表示拥有table样式的表格，其thead元素内的tr元素如果有th的话，则应用该样式。

```
.table > thead > tr > th {
  vertical-align: bottom;
  border-bottom: 2px solid #dddddd;
}
```

3.兄弟选择器

兄弟元素分为两种，一种是临近兄弟，一种是普通兄弟。临近兄弟的选择符用“+”表示。比如导航条里要设置两个li之间的外边距，则需要如下定义：

```
.nav-pills > li + li {
  margin-left: 2px;       /* 加大左外边距 */
}
```

如果只想查找某一个指定元素后面的兄弟节点（而不限制于临近节点），可以使用普通兄弟节点的符号“～”。比如：

```
.article h1 ~ p {
	font-size: 13px;
}
```

1.5.3 伪类

CSS3提供了非常多的可用伪类，但是Bootstrap只用了常用的几个，这里我们只简单列一下常用的伪类和其对应的意思，具体如表1-5所示。

表 1-5 CSS3 中常见伪类

属 性	描 述
:hover	鼠标滑过时的状态
:focus	元素拥有焦点时的状态
:first-child	指定某一个元素的第一个子元素
:last-child	指定某一个元素的最后一个子元素
:nth-child()	指定某个元素的一个或多个特定的子元素，可以传入数字，也可以传入 even（奇数）或 odd（偶数）

举个例子，按钮组里，除第一个按钮、最后一个按钮和带有dropdown-toggle样式的元素外，其他btn样式的按钮都不能设置圆角。我们应该这么定义：

```
.btn-group   >   .btn:not(:first-child):not(:last-child):not(.dropdown-
toggle) {
      border-radius: 0;

}
```

注意

多个伪类可以一起使用。

1.5.4 display属性

这个属性用于定义建立布局时元素生成的框的类型。对于HTML等文档类型，如果使用display时不谨慎会很危险，因为可能违反HTML中已经定义的显示层次结构。表1-6是display支持的常见属性值和意义。

表 1-6 display 的场景属性值和意义

属 性 值	意 义
none	此元素不会被显示
block	此元素将显示为块级元素，此元素前后会带有换行符
inline	默认。此元素会被显示为内联元素，元素前后没有换行符
inline-block	行内块元素（CSS2.1 新增的值）
list-item	此元素会作为列表显示
run-in	此元素会根据上下文作为块级元素或内联元素显示
compact	CSS 中有 compact 值，不过由于缺乏广泛支持，已经从 CSS2.1 中删除
marker	CSS 中有 marker 值，不过由于缺乏广泛支持，已经从 CSS2.1 中删除
table	此元素会作为块级表格来显示（类似 <table>），表格前后带有换行符
inline-table	此元素会作为内联表格来显示（类似 <table>），表格前后没有换行符
table-row-group	此元素会作为一个或多个行的分组来显示（类似 <tbody>）
table-header-group	此元素会作为一个或多个行的分组来显示（类似 <thead>）
table-footer-group	此元素会作为一个或多个行的分组来显示（类似 <tfoot>）
table-row	此元素会作为一个表格行显示（类似 <tr>）
table-column-group	此元素会作为一个或多个列的分组来显示（类似 <colgroup>）
table-column	此元素会作为一个单元格列显示（类似 <col>）
table-cell	此元素会作为一个表格单元格显示（类似 <td> 和 <th>）
table-caption	此元素会作为一个表格标题显示（类似 <caption>）
inherit	规定应该从父元素继承 display 属性的值

1.5.5 媒体查询

媒体查询是进行响应式设计的核心要素，其功能非常强大。这里我们只列出Bootstrap用到的功能，具体可以访问http://www.w3.org/TR/css3-mediaqueries/进行查看。

Bootstrap主要用到min-width、max-width，以及and语法，用于在不同的分辨率下设置不同的CSS样式。示例如下：

```
@media (max-width: 767px) {
    /*在小于768像素的屏幕里,这里的样式才生效*/
}
@media (min-width: 768px) and (max-width: 991px) {
    /*在768和991像素之间的屏幕里,这里的样式才生效*/
}
@media (min-width: 992px) and (max-width: 1199px) {
    /*在992和1199像素之间的屏幕里,这里的样式才生效*/
}
@media (min-width: 1200px) {
    /*在大于1200像素的屏幕里,这里的样式才生效*/
}
```

1.6　JavaScript基本语法

本节和上节的CSS语法类似，主要介绍JavaScript的基本用法。理解这些用法以后，就可以按照Bootstrap的开发规范去开发自己的各种插件了。

1.6.1 ||和&&运算符

Java Script中的||和&&两个运算符，与其他语言相比略有不同，其符合如下规则：

❑||表示，如果第一个元素可以转换为true，则返回第一个元素的值，否则查询第二个元素的值。如果多个||一起用，则按顺序优先级判断。

❑&&则相反，如果第一个元素可以转换为false，才返回第一个元素的值，否则返回第二个元素的值，多个&&一起用时，也是按顺序优先级判断。

说得严谨一些就是：

❑a && b&& c&&d：返回第一个可转换为false的元素值。

❑a||b||c||d：返回第一个可转换为true的元素值。

上述运算符的转换规则如下：

❑对象为true

❑非零数字为true

❑非空字符串为true

❑其他为false

1.6.2 立即调用的函数表达式

在JS里，function在定义的时候就可以通过在后面加一个小括号的形式立即进行调用。比如：

```
(function () { /* code */ } ());            // 推荐使用这个
(function () { /* code */ })();                 // 这个也是可以用的
(function () { /* code */ } (1));               // 传入参数1
(function () { /* code */ })(2);                // 传入参数2
```

Bootstrap的所有的JS插件都使用了这个模式。比如在alert.js文件里有以下代码：

```
+function ($) {
“use strict”;
}(jQuery);
```

这个文件的意思是声明一个function，然后立即执行，并且在执行的时候传入jQuery对象作为参数。这么做的好处是，此时function内部的$已经是局部变量了，不会再受外部作用域的影响了。

function前面的+号和分号的功能是一样的，主要是防止定义了不符合规定的代码。比如上面的这段代码若没加+号，则代码连接在一起执行就会出错。这样就消除了出错的可能性。不习惯的话，改成这样也行：

```
;function ($) {
“use strict”;
}(jQuery);
```

关于立即执行的函数表达式，请访问笔者的博客“深入理解JavaScript系列”里的第4篇，地址如下：http://www.cnblogs.com/TomXu/archive/2011/12/31/2289423.html。

1.6.3 原型

在Bootstrap的JS插件里，所有的插件都是利用了类似下面的代码：

```
Alert.prototype.close = function (e) {
    /*…*/
}
```

上面的代码就是在Alert函数上定义一个名为close的原型方法。至于什么叫原型，原型有什么好处，这里我们不会深入讲解，只是举一个例子，简单理解一下就好。

```
var Calculator = function () {};
Calculator.prototype.add = function (a, b) {
    return a + b;
}
var cal = new Calculator();
var sum = cal.add(1, 2);
```

由于这些内容不是本书的重点，所以这里只需要知道Calculator的实例能调用原型上的add方法就可以了（就像Java和C#里定义的普通方法一样）。

关于原型和原型链的详细内容，请访问笔者的博客“深入理解JavaScript系列”里的第5篇，地址如下：http://www.cnblogs.com/TomXu/archive/2012/01/05/2305453.html。

1.7　jQuery基本用法

本小节主要是介绍与jQuery相关的用法，读者在深入了解这些特性以后，就可以按照Bootstrap的开发规范去开发自己的各种插件了，而不需要精通jQuery的全部用法。

在学习jQuery相关的用法之前，首先要确保所有的CSS选择器都能在jQuery里使用，比如$(‘[data-toggle^=button]’)。如果遇到任何不明白的选择器，请查询CSS相关的语法。

1.7.1 事件绑定

jQuery的on和off分别用于绑定和禁用事件。例如：

```
$('td').on("click", function (event) {      // 绑定abc元素上的click事件,单击时弹出提示
    alert(1);
});

$('td).off('click');                        // 禁用abc元素上的click事件
```

但是对于Bootstrap框架，它对jQuery的on和off的使用稍有不同。它使用了另外一种语法，例如：

```
$(document).on('click.bs.carousel.data-api','td',function (e){};
$(document).off('.carousel.data-api');
```

上述的on在使用时，中间多了一个参数，而且选择器变成了document。它的好处是只在document上绑定一个单击事件，利用冒泡的机制，在单击的时候检查是否是td元素，如果是才处理。而前面我们把td作为选择器的时候，一个页面有多少td元素就会绑定多少个click事件，这样性能会大大降低。这种3个参数的模式称为享元模式。

关于享元模式的详细信息，请访问笔者的博客“深入理解JavaScript系列”里的第37篇，地址如下：
http://www.cnblogs.com/TomXu/archive/2012/04/09/2379774.html

1.7.2 事件命名空间

另外，还需要注意的是，这里的事件后面都跟了一些字符串，我们简单称它们为带有命名空间的事件。比如，你声明如下这样的代码：

```
$(‘#abc’).on(“click.tomxu”, function (event) {
    alert(1);
});
```

一般别人触发click事件，都是这样做的：

```
$(‘#abc’).trigger(‘click’);
```

执行上述代码，在click时，所有绑定的click事件回调都会被执行。但是如果触发的时候，你不想影响其他click触发代码，这时候就可以只触发自己定义的click事件，以求不会对别人绑定的click回调产生影响，这时可以这样做。

```
$(“#abc”).trigger(‘click.tom.xu’);
```

1.7.3 $.data()

很多JS插件里都使用了$(selector).data()方法，它的意思是收集指定元素上的所有以data-开头的自定义属性，并合并成为一个对象字面量。

对于以data-开头的自定义属性，相信大家都知道它是HTML5新支持的语法。比如：

```
<div id="abc" data-role="aaa" data-toggle="toggle" data-xxx="tom">
</div>
```

如果要获取data-role里aaa这个值，则需要调用如下代码：

```
$("'#abc").data("role");
```

如果是不带参数的$("'#abc").data()；，则表示一次性将所有以data-开头的参数都收集起来，其结果和用如下方式声明一个value变量是一样的。

```
var value = {
    role: 'aaa',
    toggle: 'toggle',
    xxx: 'tom'
};
```

Bootstrap中的很多JS插件都是利用了这个特性，在HTML元素上定义了一些必要的参数，比如要不要使用动画、是否开启键盘事件等。大家在分析JS插件的option选项参数时即可看到各个参数的详细解释。

1.8 HTML5辅助设计

在Bootstrap组件里，很多示例里都出现了不是以data-开头的自定义属性，我们称之为辅助属性。这些属性是HTML5新提出的概念，用于支持残障人士、老年人、文化水平不高或暂时患病的人使用屏幕阅读器进行页面访问。这种逐渐增强和易访问的丰富Internet应用程序简称CSS。示例如下：

```
<div role="navigation" aria-labelledby="myModalLabel" aria-hidden="true"></div>
```

上述示例中的aria-hidden="true"表示对屏幕阅读器隐藏该div元素，我们称该aria-hidden为CSS状态属性。

上述示例中的aria-labelledby="myModalLabel"告诉屏幕阅读器，这是一个modal，一般用在区域元素上，它的值一般和标题或是标签元素的ID对应。

假定屏幕阅读器遇到包含role=navigation的页面且其上还有一个div元素，屏幕阅读器就会知道该div元素用于导航，用户将能直接使用导航功能而非通过所有链接选择标签。role支持的属性值如表1-7所示。

表 1-7　常用 role 属性列表

界标角色	结构性角色		小组件角色		
			独立小组件		复合小组件
application	article	region	alert	progressbar	combobox
banner	columnheader	row	alertdialog	radio	grid
complementary	definition	rowheader	button	scrollbar	listbox
contentinfo	directory	separator	checkbox	slider	menu
form	document	toolbar	dialog	spinbutton	menubar
main	group		gridcell	status	radiogroup
navigation	heading		link	tab	tablist
search	img		log	tabpanel	tree
	list		marquee	textbox	treegrid
	listitem		menuitem	timer	
	math		menuitemcheckbox	tooltip	
	note		menuitemradio	treeitem	
	presentation		option		

更多关于CSS的详细内容，可以阅读下面两篇文章：

- http://msdn.microsoft.com/zh-cn/magazine/jj863135.aspx
- http://www.zhangxinxu.com/wordpress/?p=2277

第2章　整体架构

对于本章，建议读者至少阅读两遍，第一遍是大概了解一下Bootstrap的设计思想，然后在阅读了后续章节，全面了解了Bootstrap的所有用法以后，回过头来再阅读一遍，这样就会对Bootstrap有更加深入的理解。如果想开发基于Bootstrap的插件的话，阅读第二遍是绝对有必要的。

当然，如果你一点都不了解Bootstrap，并且想先学习如何使用它，那么可忽略本章，从第3章开始继续阅读。

2.1 整体架构

大多数Bootstrap的使用者都认为Bootstrap只提供了CSS组件和JavaScript插件，其实CSS组件和JavaScript插件只是Bootstrap框架的表现形式而已，它们都是构建在基础平台之上的。在详细分析其架构之前，先来看看它的整体架构图，如图2-1所示。

图2-1总共分为6大部分，除了CSS组件和JavaScript插件以外，另外4部分都是基础支撑平台，下面对它们分别进行介绍。

1.CSS12栅格系统

要理解12栅格系统，首先要知道什么叫栅格系统。栅格系统没有官方的定义，但根据互联网上的各种描述，笔者认为可以这样定义：以规则的网格阵列来指导和规范网页中的版面布局以及信息分布。网页栅格系统是从平面栅格系统中发展而来的。对于网页设计来说，栅格系统的使用，不仅可以让网页信息的呈现更加美观、易读，更具可用性，而且对于前端开发来说，也让网页开发更加灵活与规范。

Bootstrap的12栅格系统也就是把网页的总宽度平分为12份，开发人员可以自由按份组合，以便开发出简洁方便的程序。另外Bootstrap也提供了更加灵活的栅格系统，即栅格系统所使用的总宽度可以不固定，你可以针对一个div元素使用12等分的栅格，因为Bootstrap是按照百分比进行12等分的（保留了15位小数点精度）。

12栅格系统是整个Bootstrap的核心功能，也是响应式设计核心理念的一个实现形式。

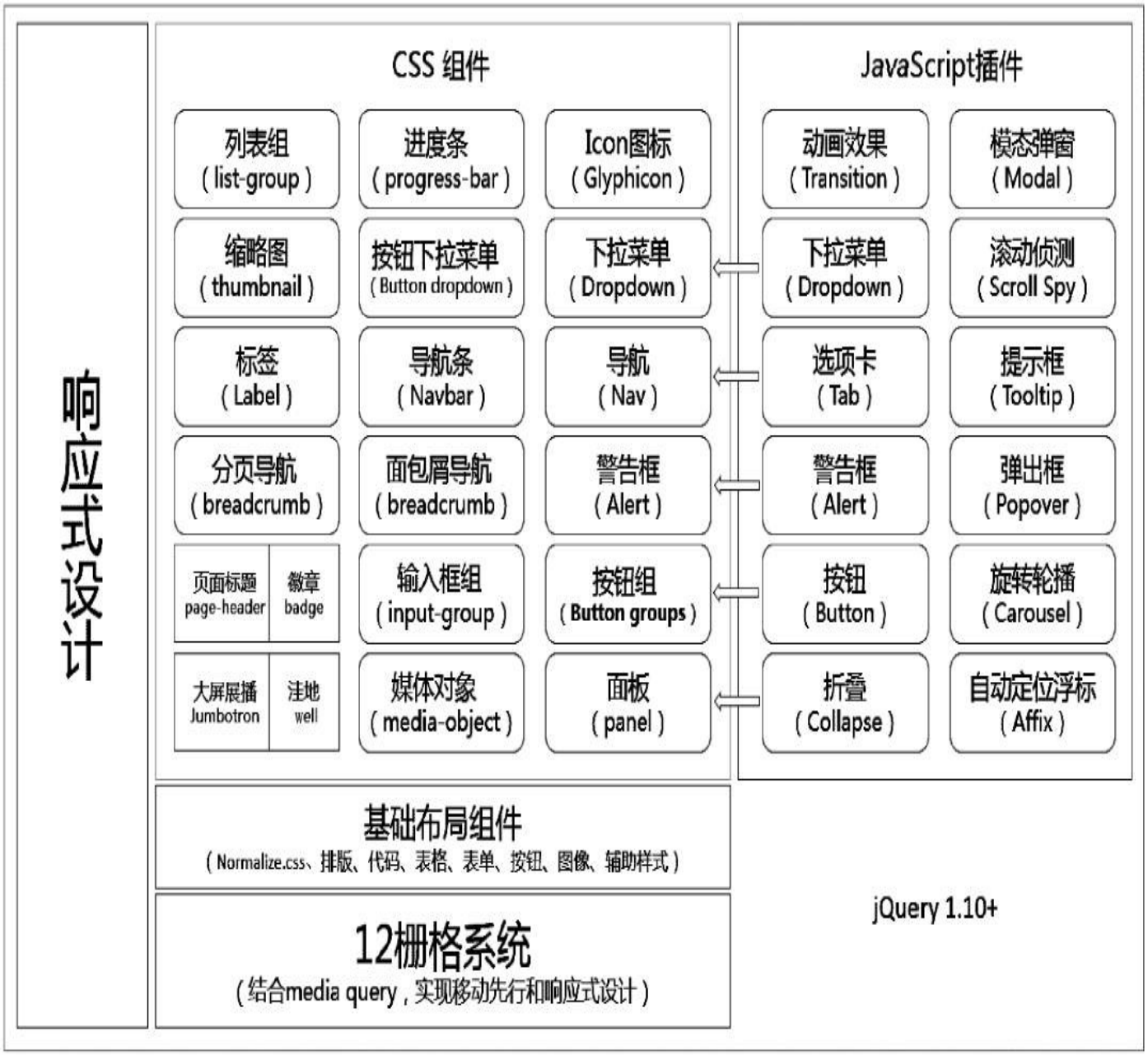


图2-1　Bootstrap整体架构图

2.基础布局组件

在12栅格系统的基础之上，Bootstrap提供了多种基础布局组件，比如排版、代码、表格、按钮、表单等，这些基础组件可以随意应用在任何页面的任何元素上，包括其顶部的CSS组件内部也可以任意使用这些基础组件。只能通过在成型的CSS组件上应用丰富多彩的自定义基础组件，才能制作出漂亮、精美、酷炫的网页来。

这些基础布局组件会在第3章进行详细讲解。在此只需要知道它们是最基础的、用于构建丰富多彩网页的基本组件即可。

3.jQuery

Bootstrap所有的JavaScript插件都依赖于jQuery1.10+，如果要使用这些插件，那就必须引用jQuery库。如果你只用CSS组件，那就可以不引用

它了。

4.响应式设计

页面的设计与开发应当根据用户行为以及设备环境（系统平台、屏幕尺寸、屏幕定向等）进行相应的响应和调整。具体的实践方式由多方面决定，包括弹性网格和布局、图片、CSS媒体查询（media query）的使用等。无论用户正在使用笔记本还是iPad，我们的页面都应该能够自动切换分辨率、图片尺寸及相关脚本功能等，以适应不同设备。换句话说，页面应该有能力自动响应用户的设备环境。响应式网页设计就是一个网站能够兼容多个终端，而不是为每个终端做一个特定的版本。这样就可以不必为不断到来的新设备做专门的版本设计和开发了。

响应式设计是一个理念，而非功能，这里为什么把它放在整个架构图的左边？是因为Bootstrap的所有内容，都是以响应式设计为设计理念来实现的。

5.CSS组件

最新的3.x版本里提供了20种CSS组件，而在原来的2.x版本里，Bootstrap只提供了14种CSS组件，分别是：下拉菜单（Dropdown）、按钮组（Button group）、按钮下拉菜单（Button dropdown）、导航（Nav）、导航条（Navbar）、面包屑导航（Breadcrumb）、分页导航（Pagination）、标签与徽章（Label&Badge）、排版（Typography）、缩略图（Thumbnail）、警告框（Alert）、进度条（Progress bar）、媒体对象（Media object）、其他（Well）。

对比一下，我们能够发现：Icon图标、大屏幕展播（Jumbotron）、页面标题（Page header）和洼地（Well）从基本布局里独立出来，成为了独立的组件；而标签（Label）、徽章（Badge）原来是一个组件，现在也成为了两个单独的组件了（并进行了增强）；最后，还新增了3个组件，分别是：输入框组（Input group）、列表组（List group）、面板（Panel）。另外，CSS和JavaScript插件中间有5个箭头，表示这5个相关的组件（插件）是有直接关系的。

6.JavaScript插件

新版的JavaScript插件总共12种，与2.x版本的13种相比少了一种输入提示（typeahead）插件。删除它的原因是该插件已经单独成为了Twitter的一个独立项目（https://github.com/twitter/typeahead.js）。另外，Bootstrap项目由原来的https://github.com/twitter转移到了https://github.com/twbs，其原因我们就不多说了。所以如果大家想继续使用typeahead插件，就去自行下载并安装使用吧。

2.2 栅格系统

本节就带领大家详细了解一下栅格系统。

2.2.1 实现原理

栅格系统的实现原理非常简单，仅仅是通过定义容器大小，平分12份，再调整内外边距，最后再结合媒体查询，就制作出了强大的响应式的栅格系统。Bootstrap默认的栅格系统平分为12份，在使用的时候大家也可以根据情况通过重新编译LESS源码来修改12这个数值。

栅格系统的主要工作原理如下：

❑一行数据（row）必须包含在.container中，以便为其赋予合适的对齐方式和内边距（padding）。

❑使用行（row）在水平方向创建一组列（column）。

❑具体内容应当放置于列（column）内，而且只有列（column）可以作为行（row）的直接子元素。

❑内置一大堆样式，可以使用像.row和.col-xs-4（占4列宽度）这样的样式来快速创建栅格布局。Bootstrap源码中定义的mixin也可以用来创建语义化的布局。

❑通过设置padding从而创建列（column）之间的间隔。然后通过为第一列和最后一列设置负值的margin从而抵消掉padding的影响。

❑栅格系统中的列是通过指定1到12的值来表示其跨越的范围的。例如，要让屏幕分为3个等宽的部分，可以使用3个.col-xs-4来创建。

为了方便大家理解，笔者画了一张示意图，如图2-2所示。我们来一步一步解释一下。

图2-2　CSS12栅格系统

1）最外层的边框。由于在设计栅格系统的时候，从头到尾都伴随着响应式设计的思想，所以作者在开发的时候，区分了4种类型的浏览器（超小屏、小屏、中屏、大屏），其像素的分界点分别是768px、992px以及1220px。

2）第二层的边框。该边框也就是我们所说的.container样式容器，针对4种不同宽度的浏览器，.container样式的宽度也不一样，就像示意图里所说的，其值分别是：自动、750px、970px、1170px。其中自动表示，如果你选择了全部12格填充，那就是100%充满这个超小屏幕。

3）1号横条。该横条内部总共有12个列表，根据源码的885～886行可以看出，每个列都有15px的左内边距（padding-left）和右内边距（padding-right）。这样就又导致一个问题，第一个列的padding-left和最后一列的padding-right占据了总共30px的宽度，从而导致页面不美观（当然，如果要留白，也没问题）。

4）2号横条，也就是row容器的定义，在row的CSS的定义里，将margin-left和margin-right都定义为-15px（源码810行），组合row和列以后，就会形成3号横条的效果。这也就是我们期望的左右宽度用满的效果。

5）对于4号横条，主要是想表达12个列可以自由组合，可以用完12

个，也可以只用一部分，只要符合“col-md-数字”的规则就可以（md是中型屏幕的样式，后面会讲到其他3种形式）。

最后，再来看一下.container样式的源码。

```
// 源码734行
.container {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
@media (min-width: 768px) {
  .container { width: 750px;    /*小型屏幕时,container样式的宽度*/  }
}
@media (min-width: 992px) {
   .container { width: 970px;     /*中型屏幕时,container样式的宽度,缩小min-
width范围*/}
}
@media (min-width: 1200px) {
   .container { width: 1170px;    /*大型屏幕时,container样式的宽度,再次缩小
min-width范围*/}
}
```

为什么这么设置？就是因为row样式定义了-15px的外边距，所以为了消除2号横条的溢出，需要用row的容器元素（也就是container样式元素）的padding值进行修补。

换句话，如果想让row充满全屏100%显示，那就不能用container样式了，因为container样式针对4个不同的屏幕大小固定了尺寸。这样，就得自己定义一个100%的样式，但是要注意的就是：别忘记修复-15px的外边距（即设置15px的左右padding值）。

除了上述原理，在实现细节上还有一些技巧，这部分的CSS分析，我们将结合基本用法进行讲解。

2.2.2 基本用法

说到栅格系统的用法，其实就是列的组合。在基本用法里有4种特性，再结合下一小节就要讲到的响应式用法，就能组合出功能强大的、各式各样的布局元素。先来看看这4种基本用法。由于不同屏幕的尺寸使用了不同的样式，这里我们以中型屏幕（md）为例进行介绍，其他类似的，可以参考下一小节。

1.列组合

刚才我们已经提到了列组合的用法，就是通过更改数字来合并列，类似于表格里的colspan。使用起来非常简单，示例代码如下：

```
<div class="container">
    <div class="row">
        <div class="col-md-1">.col-md-1</div>
        <!—由于这12个div都一样,所以这里省略了其他10个—>
        <div class="col-md-1">.col-md-1</div>
    </div>
    <div class="row">
        <div class="col-md-8">.col-md-8</div>
        <div class="col-md-4">.col-md-4</div>
    </div>
    <div class="row">
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4">.col-md-4</div>
        <div class="col-md-4">.col-md-4</div>
    </div>
    <div class="row">
        <div class="col-md-6">.col-md-6</div>
        <div class="col-md-6">.col-md-6</div>
    </div>
</div>
```

运行上述示例代码后会看到如图2-3所示的效果。

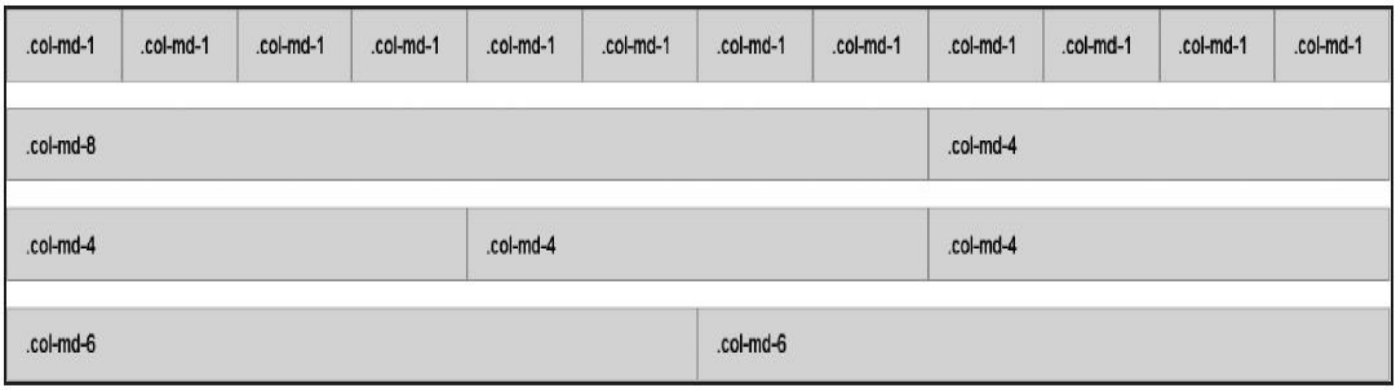


图2-3 CSS12栅格系统

列组合的实现方式非常简单，只涉及两个CSS特性：左浮动和宽度百分比。由于很简单，就不多讲述了，大家直接看代码注释即可。

```
// 源码1085行
.col-md-1,.col-md-2,.col-md-3,.col-md-4,.col-md-5,.col-md-6,
.col-md-7,.col-md-8,.col-md-9,.col-md-10,.col-md-11,.col-md-12 {
    float: left; /*确保12个列都是左浮动*/
}
 /*定义每个组合的宽度百分比*/
.col-md-12 {  width: 100%;}
.col-md-11 {  width: 91.66666666666666%;}
.col-md-10 {  width: 83.33333333333334%;}
.col-md-9 {  width: 75%;}
.col-md-8 {  width: 66.66666666666666%;}
.col-md-7 {  width: 58.333333333333336%;}
.col-md-6 {  width: 50%;}
.col-md-5 {  width: 41.66666666666667%;}
.col-md-4 {  width: 33.33333333333333%;}
.col-md-3 {  width: 25%;}
.col-md-2 {  width: 16.666666666666664%;}
.col-md-1 {  width: 8.333333333333332%;}
```

2.列偏移

有的时候，我们不想让两个相邻的列挨在一起，这时候利用栅格系统的列偏移（offset）功能来实现，而不必再定义margin值。使用.col-md-offset-*形式的样式就可以将列偏移到右侧。例如，.col-md-offset-4将.col-md-4向右移动了4个列的宽度，效果如图2-4所示。

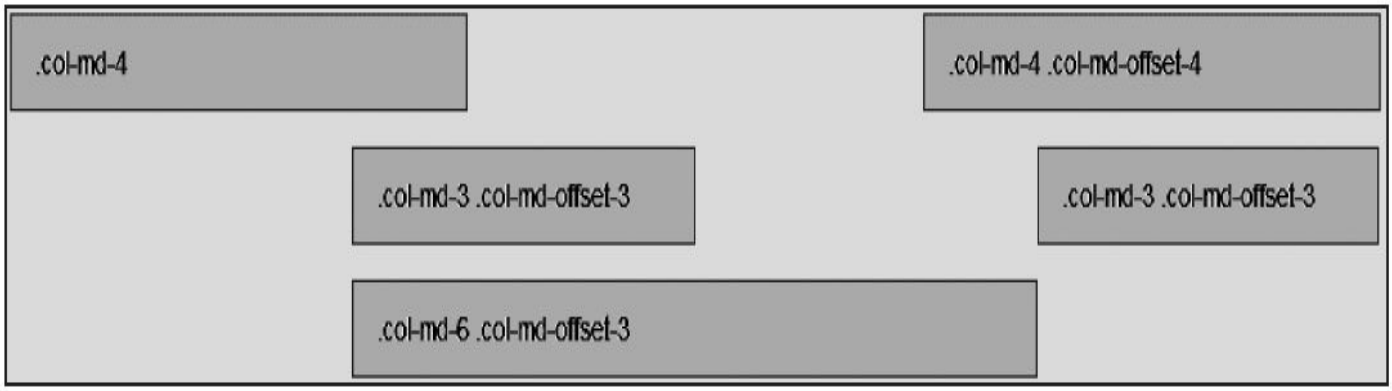


图2-4　列偏移运行效果

实现上述效果，只需要在下一个列上应用offset样式即可（因为该样式设置的是margin-left）。代码如下：

```
<div class=“row”>
    <div class=“col-md-4”>.col-md-4</div>
       <div  class=“col-md-4  col-md-offset-4”>.col-md-4  .col-md-offset-
4</div>
</div>
<div class=“row”>
       <div  class=“col-md-3  col-md-offset-3”>.col-md-3  .col-md-offset-
3</div>
       <div  class=“col-md-3  col-md-offset-3”>.col-md-3  .col-md-offset-
3</div>
</div>
<div class=“row”>
```

```
        <div class="col-md-6 col-md-offset-3">.col-md-6 .col-md-offset-3</div>
    </div>
```

源码也非常简单，只利用了1/12的margin-left而已，有多少个offset，就有多少个margin-left。

```
// 源码1203行
.col-md-offset-12 {  margin-left: 100%;}
.col-md-offset-11 {  margin-left: 91.66666666666666%;}
.col-md-offset-10 {  margin-left: 83.33333333333334%;}
.col-md-offset-9 {  margin-left: 75%;}
.col-md-offset-8 {  margin-left: 66.66666666666666%;}
.col-md-offset-7 {  margin-left: 58.333333333333336%;}
.col-md-offset-6 {  margin-left: 50%;}
.col-md-offset-5 {  margin-left: 41.66666666666667%;}
.col-md-offset-4 {  margin-left: 33.33333333333333%;}
.col-md-offset-3 {  margin-left: 25%;}
.col-md-offset-2 {  margin-left: 16.666666666666664%;}
.col-md-offset-1 {  margin-left: 8.333333333333332%;}
.col-md-offset-0 {  margin-left: 0;}
```

3.列嵌套

同样，栅格系统也支持列嵌套，即在一个列里再声明一个或者多个行（row）。但是注意，内部所嵌套的row的宽度为100%时，就是当前外部列的宽度。示例代码如下：

```
<div class="row">
    <div class="col-md-9">
        Level 1: .col-md-9
        <div class="row">
            <div class="col-md-6">Level 2: .col-md-6</div>
            <div class="col-md-6">Level 2: .col-md-6 </div>
        </div>
    </div>
<div class="col-md-3"></div>
</div>
```

运行效果如图2-5所示。

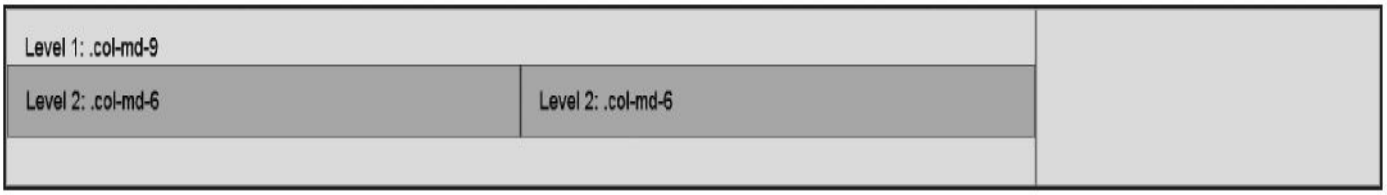


图2-5　列嵌套运行效果

可以看到，在第一个列（col-md-9）里面，嵌套了一个新行（row），然后在新行里，又放置了两个等宽列的col-md-6，这时候大家就会发现，两个col-md-6加起来是12，但是总宽度和外面的col-md-9列的宽度一样。也就是说row里的列宽度是按照百分比分配的，这也印证了上

面的源码。所以在任何一个嵌套列里，不管宽度是多少，都可以再进行12等分，并可以进一步组合。

4.列排序

列排序其实就是改变列的方向，也就是改变左右浮动，并且设置浮动的距离。在栅格系统里，可以通过.col-md-push-*和.col-md-pull-*来实现这一目的。先来看看效果示意图，如图2-6所示。

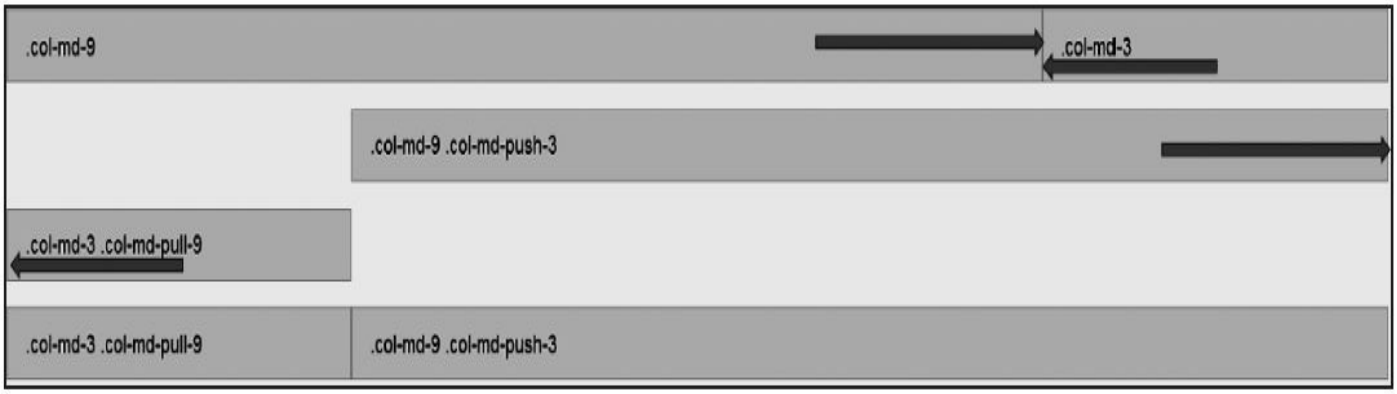


图2-6　列排序运行效果

默认情况下，col-md-9在左边，col-md-3在右边，如果要互换位置，需要将col-md-9列向右移动3个列的距离，也就是推3个列的offset，样式用col-md-push-3；而col-md-3则需要向左移动，也就是拉9个列的offset，样式用col-md-pull-9。读者可能会问，为什么不能用左右浮动呢？这是因为所有的列默认情况下都是左浮动，如果要使用左右浮动，那就不知道得修改多少列的左右浮动样式了。所以，作者在统一左浮动的基础上，通过设置left和right的值来实现定位显示。代码如下所示：

```
// 源码1125行
.col-md-pull-12 {  right: 100%;}
.col-md-pull-11 {  right: 91.66666666666666%;}
.col-md-pull-10 {  right: 83.33333333333334%;}
.col-md-pull-9 {  right: 75%;}
.col-md-pull-8 {  right: 66.66666666666666%;}
.col-md-pull-7 {  right: 58.333333333333336%;}
.col-md-pull-6 {  right: 50%;}
.col-md-pull-5 {  right: 41.66666666666667%;}
.col-md-pull-4 {  right: 33.33333333333333%;}
.col-md-pull-3 {  right: 25%;}
.col-md-pull-2 {  right: 16.666666666666664%;}
.col-md-pull-1 {  right: 8.333333333333332%;}
.col-md-pull-0 {  right: 0;}

.col-md-push-12 {  left: 100%;}
// 其他同理,上pull的设置类似,唯一不同的就是right和left的区别
.col-md-push-0 {  left: 0;}
```

只是利用左浮动、left、right　3个基础样式，就能实现这种效果，不能不说这真的很巧妙啊（笔者感叹：这3种样式，10年前就会了，为啥自

己想不到这种效果呢！）。

2.2.3 响应式栅格

刚才说到了，Bootstrap为不同的屏幕尺寸（4种类型）提供了不同的栅格样式，前面例子我们使用的一直是中型屏幕（md），其他3种样式分别是超小（xs）、小型（sm）、大型（lg），所以组合起来样式就多了，可以是.col-xs-1，也可以是.col-sm-offset-2，还可以是.col-lg-push-1或者.col-lg-pull-1。下面总结了一个表格用于记录不同类型浏览器的各种参数，如表2-1所示。

表 2-1 响应式栅格尺寸标准及实现设置

	超小屏幕设备手机 (<768px)	小屏幕设备平板 (≥ 768px)	中等屏幕设备桌面 (≥ 992px)	大屏幕设备桌面 (≥ 1200px)
栅格系统行为	总是水平排列	开始是垂直排列在一起，超过这些媒体查询的阈值将变为水平排列		
最大 .container 宽度	None（自动）	750px	970px	1170px
样式前缀	.col-xs-	.col-sm-	.col-md-	.col-lg-
列数	12			
最大列宽	自动	60px	78px	95px
padding	30px（每列左右均有 15px）			
可嵌套	Yes			
offsets	Yes			
列排序	Yes			

这些分界点是如何定义的?答案就是：它们是媒体查询定义的。前面源码已经提到了，下面再列一下：

```
// 超小型是默认实现
// 小型
@media (min-width: 768px) {
    .container { width: 750px;    }
}
// 中型
@media (min-width: 992px) {
    .container { width: 970px;    }
}
// 大型
@media (min-width: 1200px) {
    .container { width: 1170px;    }
}
```

1.跨设备组合定义

通过前面的示例和源码可以发现，一种样式（比如col-md-9）在其定义的尺寸范围以外是不起作用的，比如，在小型或者大型屏幕下，所有带有md的样式都不会生效，而且没有默认值。这就是说，我们可以在一个元素上应用不同类型的样式，以适配不同尺寸的屏幕。比如下面的示例：

```
<div class=“row”>
    <div class=“col-sm-12 col-md-8”>.col-sm-12 .col-md-8</div>
    <div class=“col-sm-6 col-md-4”>.col-sm-6 .col-md-4</div>
</div>
<div class=“row”>
    <div class=“col-sm-6 col-md-4”>.col-sm-6 .col-md-4</div>
    <div class=“col-sm-6 col-md-4”>.col-sm-6 .col-md-4</div>
    <div class=“col-sm-6 col-md-4”>.col-sm-6 .col-md-4</div>
</div>
<div class=“row”>
    <div class=“col-sm-6”>.col-sm-6</div>
    <div class=“col-sm-6”>.col-sm-6</div>
</div>
```

在上述示例中，每个div元素都应用了两种样式，分别是sm类型和md类型，用于适配小型屏幕和中型屏幕。在中型屏幕中的效果如图2-7所示。

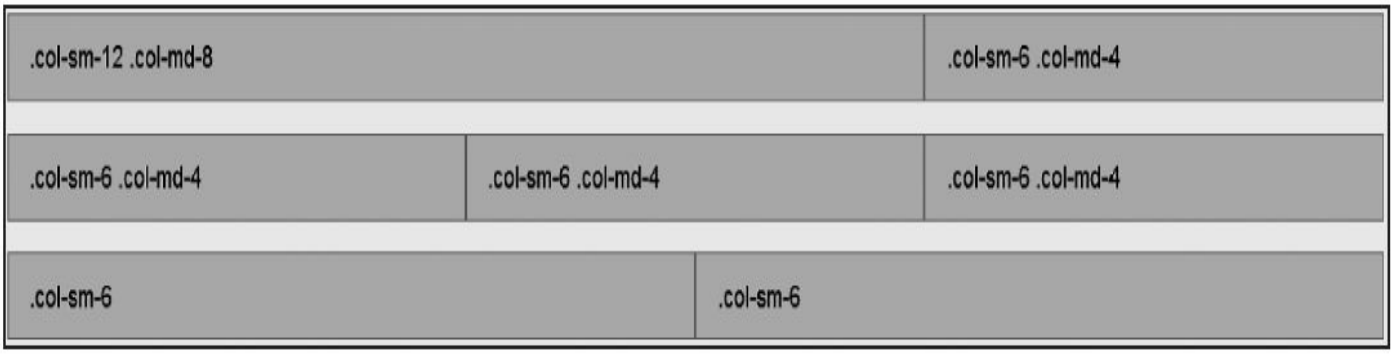


图2-7　中型屏幕的运行效果

而换到小型屏幕的浏览器上时，效果就完全不一样了，如图2-8所示。

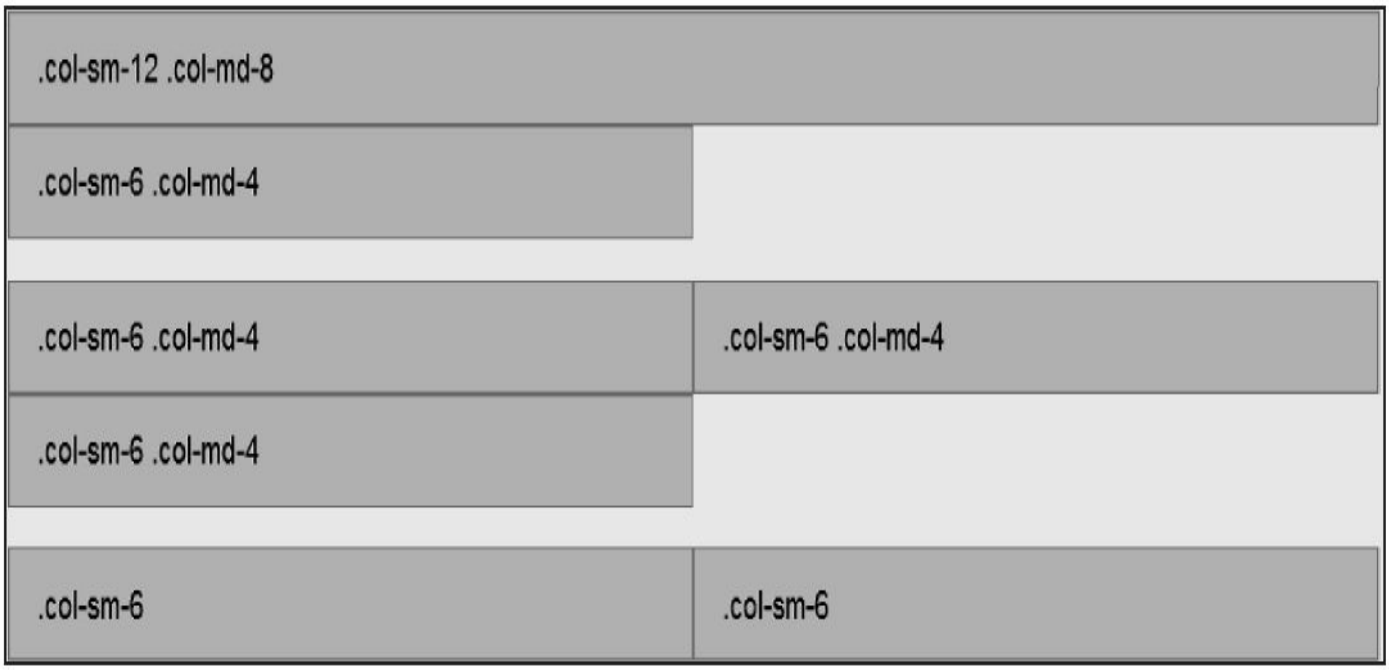


图2-8　小型屏幕的运行效果

为什么不一样？这就是因为第一行row里的第1个div使用了col-sm-12，一下子就占用了12个列，所以第2个div就自动换行了。第2行也是，两个div就占满了12个列，所以最后一个col-sm-6就换行了。

读者，可能会问，第2行用的col-sm-6，为什么在中型屏幕上也是各占50%啊？这是因为针对sm类型的媒体查询只用了@media(min-width: 768px)，而不是@media(min-width: 768px)and(max-width: 992px)。所以，如果没有定义md样式的话，sm样式默认情况下依然有效。总结一句，如果只用min-width，则表示向大兼容。

既然向大兼容，那向小肯定就不兼容了？没错，在小于768px的超小屏幕上浏览上述示例时，由于所有的样式都无效了，所以所有的div默认都垂直换行堆叠在一起了，效果如图2-9所示。

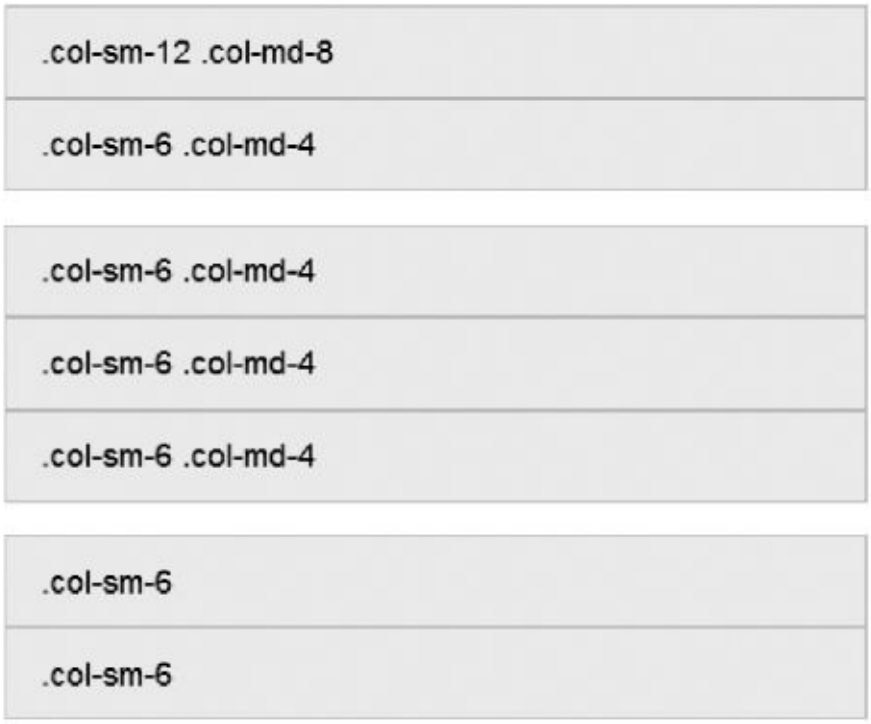


图2-9　跨设备组合在超小型屏幕下的运行效果

既然两种类型的样式可以组合在一起，那4种也就没什么问题了。所以可以在一个div上分别应用xs、sm、md、lg类型的样式，以便在所有的设备上都能进行正常浏览。

2.清除浮动问题

理想是美好的，但现实是残酷的。我们来举一个美好的例子。在小型屏幕上，希望实现如图2-10所示的效果。

图2-10　在小型屏幕下的预想效果

但是在超小型屏幕下，则希望每行只显示两个div元素，即如图2-11所示的效果。

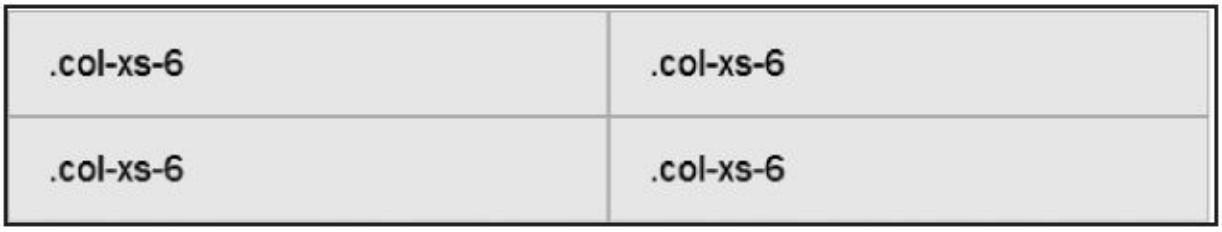


图2-11　在超小型屏幕下的预想效果

理想很美好的，按照上述响应式栅格的建议，可能觉得应该如下设计：

```
<div class="row">
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
    <div class="col-xs-6 col-sm-3">.col-xs-6 .col-sm-3</div>
</div>
```

把真实内容填到div里进入，看看实际效果，如图2-12所示。

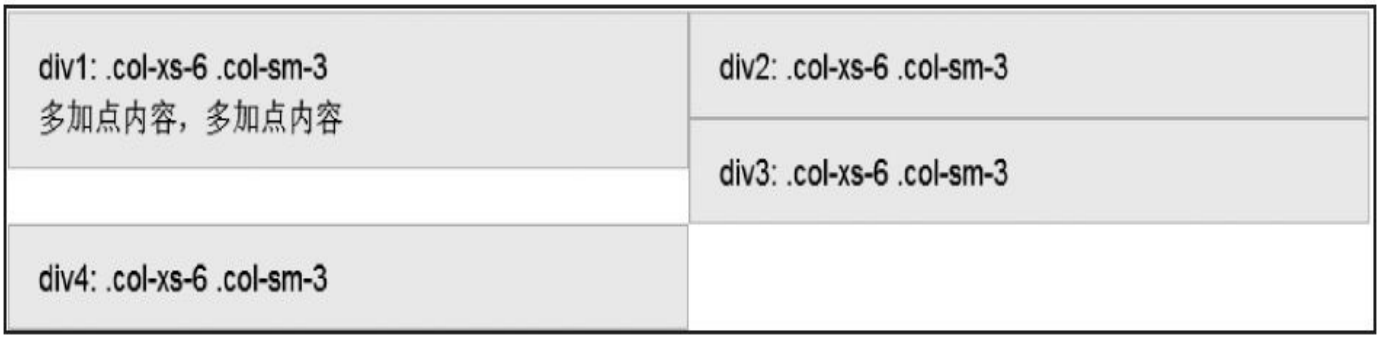


图2-12　在小型屏幕下的实际效果

你肯定会说："这是怎么回事啊！"怎么回事？所有的col-样式都是左浮动，很明显，这是因为没有清除浮动导致的。第3个div开始换行的时候，div1的内容高度过高，所以div3就在右边紧接着显示了。

要解决这个问题，需要用Bootstrap提供的clearfix样式。更新后的代码如下所示：

```
<div class="row">
<div class="col-xs-6 col-sm-3">div1: .col-xs-6 .col-sm-3</div>
<div class="col-xs-6 col-sm-3">div2: .col-xs-6 .col-sm-3</div>
<div class="clearfix visible-xs"></div>
<div class="col-xs-6 col-sm-3">div3: .col-xs-6 .col-sm-3</div>
<div class="col-xs-6 col-sm-3">div4: .col-xs-6 .col-sm-3</div>
</div>
```

利用clearfix样式清除浮动，但是前提条件是在超小型屏幕上能显示才行（因为其是用visible-xs样式控制的）。

3.列偏移和列排序

由于响应式设计是利用了媒体查询的特性，而所有以col-*开头的样式都是在媒体查询里定义的，所以像列偏移（offset）、列排序（pull和push）相关的样式也都可以用，以组合出不同的精美样式。以下的例子大家自行调试一下。

```
<div class="row">
    <div class="col-sm-5 col-md-6">.col-sm-5 .col-md-6</div>
      <div class="col-sm-5 col-sm-offset-2 col-md-6 col-md-offset-0">…
</div>
</div>
<div class="row">
     <div class="col-sm-6 col-md-5 col-lg-6">.col-sm-6 .col-md-5 .col-
```

```
lg-6</div>
      <div class="col-sm-6 col-md-5 col-md-offset-2 col-lg-6 col-lg-
offset-0">…</div>
</div>
```

怎么样？看到这么多特性，感觉到强大了吧？是的，我也觉得很强大，但是，请看下一节。

2.2.4　对重复代码的处理

本小节内容只代表我的个人观点，是在自身理解水平的基础上形成的，如有误导，请见谅。纵观整个栅格系统的源码，从854行到1635行绝大部分代码都是重复性的。比如给4种不同的类型设置width的时候，给pull和push设置left和right的时候，以及给offset设置的margin-left时候，设置的值都一样。只是样式名称不一样。除此之外不一样的地方就只有container样式的宽度设置了。那为什么不能这样设置呢？

所有重复性的内容都放一起（在媒体查询之外），那么和媒体查询有关的就只有这一项内容了，也就是如下代码中所列的width宽度。

```
.col-xs-12
.col-sm-12
.col-md-12
.col-lg-12 {
  width: 100%;
}
```

总结：这得节约多少行代码啊！当然，如果Bootstrap作者这么做是为了以后扩展成不同的值，那我就只能说：好，灵活性不错。

2.3 CSS组件架构的设计思想

对于CSS组件架构来说确定有点吓人了，但是我想给大家说明的就是CSS组件的一个设计思想，即Bootstrap的一个那么强大的组件是如何实现的（比如，btn按钮）。

我个人的理解，不管是设计思想，还是架构，都可总结为一个词：AO模式。A表示Append，即“附加”的意思；O表示Overwrite，即“重写”的意思。所有的CSS组件都是沿用这种思想来设计的。这也是CSS的特性，不同名的样式可以叠加在一起使用；同名的样式，后面的会覆盖前面的，从而达到组合应用的效果。

整个CSS组件，我总结出8大类型的样式，然后根据每个组件的特性，来组装这些类型的特性，从而达到丰富多彩的配置效果。这8种类型的样式分别如图2-13所示。

图2-13　CSS组件的8种基本类型

下面我们根据示例来具体分析这8种类型的样式是如何应用的。在分析过程中，如果需要查看细节，请自行参阅第3章和第4章中与此对应的插件的相关内容。

2.3.1 基础样式

任何一个CSS组件在刚开始都要先定义基本样式，就像一个新的HTML页面制作之前都要先定义统一的字号、背景色一样。在基本样式里，一般是定义该样式下的字体（font-）、外内边距（padding，margin）、显示方式（display）、边框（border）以及其他相关的内容。

比如在警告框组件（alert）里，其基本样式就只定义了内外边距、边框、圆角设置。源码如下：

```
// 源码4434行
.alert { /* 基本的警告框设置 */
  padding: 15px;
  margin-bottom: 20px;
  border: 1px solid transparent;
  border-radius: 4px;
}
```

而按钮组件（btn）的基本样式，除此之外还定义了文本、display、focus、hover的基本样式，详细内容请查看Bootstrap.css的第1988行。

2.3.2 颜色样式

如果在Bootstrap官方网站看过btn按钮或者alert警告框的样例，你会发现，Bootstrap默认为很多组件都提供了5种颜色的样式，这5种颜色分别是：primary（重点蓝）、success（成功绿）、info（信息蓝）、warning（警告橙）、danger（危险红）。定义规则是：组件名称-颜色类型，如btn-primary、alert-info。

在定义不同颜色样式的时候，主要是定义文本颜色、边框颜色、背景颜色等，具体定义什么颜色和该组件的特性有关。比如面板panel就只需要定义边框的颜色就可以了，而按钮不仅需要定义边框颜色，还需要定义背景色以及文本颜色。btn、panel、alert三个组件在定义颜色样式时的代码如下所示：

```
/* btn */
.btn-primary {
  color: #ffffff;  background-color: #428bca;  border-color: #357ebd;
}
/* panel */
.panel-success {
  border-color: #d6e9c6;
}
.panel-success > .panel-heading {
  color: #468847;  background-color: #dff0d8;  border-color: #d6e9c6;
}
/* alert */
.alert-success {
  color: #468847;  background-color: #dff0d8;  border-color: #d6e9c6;
}
.alert-success hr {
  border-top-color: #c9e2b3;
}
```

可以发现，不同的组件，定义的内容稍有不同，具体定义什么主要由组件特性来决定。另外还要定义其内部子样式的颜色以便形成统一风格，比如.panel-success内部的.panel-heading顶部元素也要定义一个相同风格的颜色。

而对于一些可单击元素，比如btn按钮，还要特殊处理按钮在hover、focus、active状态时的颜色，以便让这些状态在同一个风格下表现一致。其源码如下所示：

```
// 源码2081行
.btn-primary:hover,
.btn-primary:focus,
.btn-primary:active,
.btn-primary.active,
.open .dropdown-toggle.btn-primary {
  color: #fff;
```

```
  background-color: #3276b1;
  border-color: #285e8e;
}
```

使用方式就像Bootstrap宣传的那样，将两个样式叠加在一起使用即可。示例如下所示：

```
<button type=“button” class=“btn btn-primary”>Primary</button>
<button type=“button” class=“btn btn-success”>Success</button>
<div class=“alert alert-warning”>…</div>
<div class=“alert alert-danger”>…</div>
```

2.3.3 尺寸样式

Bootstrap也为大部分组件都提供了尺寸的快捷设置。一般组件都有4种尺寸：超小（xs）、小型（sm）、普通、大型（lg）。使用示例如下所示：

```
<button type=“button” class=“btn btn-xs”>Primary</button>
<button type=“button” class=“btn btn-lg”>Success</button>
<div class=“well well-lg”>…</div>
<div class=“well well-sm”>…</div>
```

调整尺寸主要是调整所对应元素的padding和圆角设置，有时候也会相应调整行间距和字体大小。btn按钮和well组件在尺寸方面的设置源码如下所示（截取的一部分）：

```
/* 源码2134行*/
.btn-lg {
  padding: 10px 16px;
  font-size: 18px;
  line-height: 1.33;
  border-radius: 6px;
}
/* 源码5060行*/
.well-lg {
  padding: 24px;                                  /*加大内边距*/
  border-radius: 6px;                        /*加大圆角设置*/
}
.well-sm {
  padding: 9px;                                  /*减少内边距*/
  border-radius: 3px;                        /*减少圆角设置*/
}
```

值得注意的是，同一个组件的不同类型的样式可以组合在一起使用，比如颜色样式btn-success和尺寸样式btn-lg一起使用完全没有问题，不会引起冲突。看下面的示例：

```
<button type=“button” class=“btn btn-success btn-lg”>Success</button>
```

2.3.4 状态样式

有一些可单击元素，经常需要根据状态来显示其效果，比如高亮可用的时候用active样式，禁用的时候用disabled样式或disabled属性。Bootstrap的一部分组件针对这种元素也都进行了处理，例如btn按钮的样式定义如下所示：

```
// 源码2018行
.btn:active,
.btn.active {
  background-image: none;
  outline: 0;                                  /* 消除outline*/
  -webkit-box-shadow: inset 0 3px 5px rgba(0, 0, 0, .125);      /*定义阴
影*/
          box-shadow: inset 0 3px 5px rgba(0, 0, 0, .125);
}
.btn.disabled,
.btn[disabled],
fieldset[disabled] .btn {
  pointer-events: none;
  cursor: not-allowed;
  filter: alpha(opacity=65);
  -webkit-box-shadow: none;                                    /* 去除
阴影*/
          box-shadow: none;
  opacity: .65;                                                /* 透明
处理*/
}
```

这种类型的样式一般是处理元素的阴影、鼠标形状、透明度、虚框等方面的内容。但是大家使用该机制做自定义组件开发时一定要注意，active和disabled样式和颜色样式有关联（因为牵涉到颜色），所以如果要定义这两种样式，就要为所有的颜色样式分别定义active和disabled对应的颜色。比如，将btn-success样式和active、disabled样式一起使用的时候，所定义的样式如下所示：

```
// 源码2122行
.btn-success:hover,.btn-success:focus,
.btn-success:active,.btn-success.active,
.open .dropdown-toggle.btn-success {
  color: #fff;
  background-color: #47a447;
  border-color: #398439;
}
.btn-success:active,.btn-success.active,
.open .dropdown-toggle.btn-success {
  background-image: none;
}
.btn-success.disabled,.btn-success[disabled],
/*此处省略部分选择器*/
.btn-success.disabled.active,.btn-success[disabled].active,
```

```
fieldset[disabled] .btn-success.active {
  background-color: #5cb85c;
  border-color: #4cae4c;
}
```

通过上述代码可以发现，不仅要注意当前元素的active、disabled样式，还要注意嵌套元素的相关样式。上述样式，可以直接和颜色样式一起使用，也可以和默认样式一起使用。

2.3.5 特殊元素样式

所谓特殊元素，即特定类型的组件一般只使用某一种或者几种固定的元素，比如alert警告框内一般只用警告标题、内容和关闭链接元素，再如导航（nav）里的经常用的li元素。在定义这些组件的时候，也要为这些常用的元素定义其相关的默认样式。这两个组件对常用特殊元素的样式定义如下所示：

```
// 源码 4444 行
.alert .alert-link {  font-weight: bold;        /* alert内的链接样式 */}
.alert > p,.alert > ul {  margin-bottom: 0;
                                                /* alert内的p元素和ul元素的底部外边
距设置 */}
.alert > p + p {  margin-top: 5px;              /*两个段落之间,增加一个顶部
外边距*/}
.alert-dismissable { padding-right: 35px;        /* 增大右内边距,以便关闭按
钮*/}
.alert-dismissable .close {                      /* 警告框内,如果设置关闭按
钮,特殊处理 */
  position: relative;
  top: -2px;
  right: -21px;                                 /* 关闭按钮,右对齐 */
  color: inherit;
}

// 源码 3462 行
.nav > li {
  position: relative;                            /*所有的菜单项都是相对定位
*/
  display: block;                                       /* 块级显示*/
}
.nav > li > a {
  position: relative;                           /* a链接相对定位*/
  display: block;                                       /* 块级显示*/
  padding: 10px 15px;                           /* 外边距保留稍微大一些*/
}
.nav > li > a:hover,.nav > li > a:focus {       /* 移动或焦点时链接的显示效
果*/
  text-decoration: none;
  background-color: #eeeeee;                     /* 链接移动或焦点时,背景色
变为灰色*/
}
.nav > li.disabled > a {  color: #999999;       /* li上禁用时的链接颜色*/}
.nav > li.disabled > a:hover,
.nav > li.disabled > a:focus {
/* li上禁用时,移动到链接上时的各种处理*/
  color: #999999;                                       /* 颜色变灰*/
  text-decoration: none;
  cursor: not-allowed;
  background-color: transparent;
}
```

特殊元素所定义的样式不固定，这取决于是什么样的特殊元素，有

的是定义内外边距，有的是自定颜色和背景色，具体元素具体对待。大家在分析源码的时候，一定要结合元素的特性进行分析，比如a链接元素，不仅可以设置内外边距，还可以设置文本颜色，甚至可以设置这些样式在hover、focus状态时的情况。

如果在这些组件内部使用的不是这些常用的特殊元素，而且其他相关的元素，那么就需要自己为这些特殊的情况定义相关的样式了，不然很有可能不符合默认组件的风格。定义的时候，还要考虑默认情况下是什么样式、和颜色样式搭配时又是什么样式，以及设置大小尺寸时要设置多少边距等。总之，要为各种情况设置相应的内容，以保持风格的统一。

2.3.6 并列元素样式

在很多情况下，一个组件内部需要放置多个子元素，比如导航组件（nav）里可以放置多个li元素，按钮组里可以放置多个button元素。如果有这种情况，就需要处理这样并列元素的间距问题了。一般来说，只需要考虑两方面即可：其一水平并列时候的左右内边距（padding-left、padding-right）和外边距（margin-left、margin-right）；其二：垂直并列时的上下内边距（padding-top、padding-bottom）和外边距（margin-top、margin-bottom）。

举个例子，alert组件里，有时候要提示两段内容，这时候就需要用到两个p元素，所以需要处理这种情况的样式。源码如下所示：

```
// 源码4447行
.alert > p {
  margin-bottom: 0;              /* alert内的p元素和ul元素的底部外边距设置 */
}
.alert > p + p {
  margin-top: 5px;               /*两个段落之间,增加一个顶部外边距*/
}
```

而在模态弹窗组件（modal）的底部，经常需要显示多个按钮（比如：“保存”、“取消”等），这时候就需要处理水平并列的情况了。源码如下所示。

```
// 源码5178行
.modal-footer .btn + .btn {
/* 底部区域内的按钮样式设置,如果有多个按钮,设置左部外边距 */
  margin-bottom: 0;
  margin-left: 5px;
}
```

2.3.7 嵌套子元素样式

有的时候，我们也需要将两个相同或不同的组件嵌套在一起使用，这时经常会出现一些特殊情况，比如，多个按钮组在一起使用，或者按钮和下拉菜单一起使用，效果如图2-14所示。

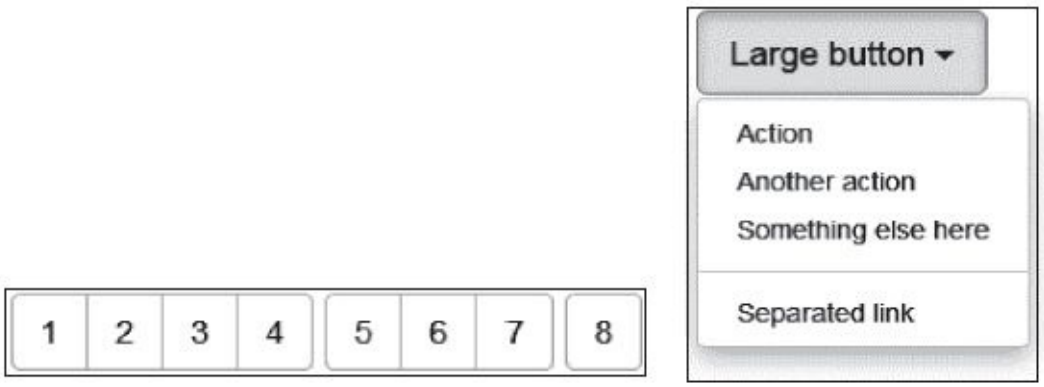


图2-14 嵌套元素运行效果

这个时候就需要考虑嵌套元素的使用情况了。比如多个分组按钮一起使用的时候，就需要考虑浮动方向和间距，其样式设置如下所示：

```
// 源码3147行
.btn-group .btn + .btn,
.btn-group .btn + .btn-group,
.btn-group .btn-group + .btn,
.btn-group .btn-group + .btn-group {
   margin-left: -1px;/* 消除2个按钮(或一个按钮和另外一个按钮组)之间的1像素细节引起的冲突 */
}
.btn-group > .btn-group {
  float: left;
}
```

而如果在分组按钮组件里使用下拉菜单，则需要考虑下拉菜单的圆角问题，因为默认情况下分组按钮里的元素，只有第一个和最后一个子元素才有圆角，其他都没有，所以需要特殊考虑下拉菜单的情况。部分源码如下所示：

```
// 源码3175行
.btn-group > .btn:last-child:not(:first-child),
.btn-group > .dropdown-toggle:not(:first-child) {
  border-bottom-left-radius: 0;
  /* 最后一个按钮或Dropdown(不是第一个按钮的情况下),左上和左下角不设置圆角 */
  border-top-left-radius: 0;
}
```

上述代码又多了一个维度，和颜色、尺寸、并列元素等组合在一起，更加复杂了。但是不用担心，毕竟这种嵌套使用的情况不是很多，

即便有，在测试的时候也会发现的，到时候就能很快修复了，毕竟只是最基本的样式修改而已。

2.3.8　动画样式

在Bootstrap里，动画样式应用得不是很多，只在进度条组件里会使用到。但是为了能够方便读者以后开发自定义组件，我们还是简述一下动画的定义，细节分析请阅读4.16节。

先来看一下动画是如何应用的：只需要在progress样式上应用一个active样式，即可开启动画过渡效果。示例代码如下所示：

```
<div class=“progress progress-striped active”>
    <div class=“progress-bar” style=“width: 45%”>
        <span class=“sr-only”>45% Complete</span>
    </div>
</div>
```

上述代码的运行效果如图2-15所示。

图2-15　progress进度条动画运行效果

在这里使用了active样式和progress样式叠加在一起产生动画过渡效果，我个人觉得active这个名称不太合适，因为状态样式里已经用过了。建议大家在做开发的时候设置一个别的名称，比如animation等。动画的定义也十分简单，只需要指定animation的名称和运行时间即可。本例的名称是progress-bar-stripes，关于progress-bar-stripes的详细定义请阅读4.16节。动画设置如下所示：

```
// 源码4551行
.progress.active .progress-bar {
  -webkit-animation: progress-bar-stripes 2s linear infinite;
                                                        /* 2秒过渡40像素,
无限循环 */
          animation: progress-bar-stripes 2s linear infinite;    /* 其他
浏览器 */
}
```

2.3.9 小结

前面我们介绍了8种类型的样式，可以发现一个问题：每一种类型的样式设置都是最基础的，都非常简单，但是这8种情况组合起来，就非常复杂了。比如，设置active状态的时候要兼顾颜色样式和尺寸样式；设置尺寸样式的时候，又要考虑并列元素的情况；根据不同的组件，还要考虑是垂直并列元素还是水平并列元素等。但是只要我们秉承“由繁入简易、由简入繁难”的思想就没问题，也就是说，先从高度层面上考虑这些问题，大概需要8种特性中的几种，然后考虑每一种之间有没有关系，最后再去针对每个细节进行样式的编写，这样就非常简单了。

另外，还有一个问题需要注意：由于这些样式是利用了CSS重载覆盖的概念，所以一定要注意其定义的顺序，以免出现重载顺序错误的问题。

下一章，我们将开始深入分析Bootstrap所有组件的使用，并进行源码分析。在分析源码的时候，脑子里时刻要有8种类型的概念，这样理解起来才能顺畅。

2.4 JavaScript插件架构

Bootstrap所有的插件在开发的时候都遵循了同样的规则，同时开发人员在使用的时候也遵循类似的规则，这些规则奠定了Bootstrap插件开发的基础，也为我们自定义插件提供了规范和依据。这些规则可以归纳为以下3种：

❑HTML布局规则：基于元素自定义属性的布局规则，比如使用类似于data-target的自定义属性等。

❑JavaScript实现步骤：所有的插件都遵循jQuery插件开发的标准步骤，所有的事件都保持了统一标准。

❑插件调用方法：所有插件的使用方式都非常类似，可以是HTML声明式，也可以是调用式（JavaScript代码），并且支持多种回调和可选参数。

2.4.1 HTML布局规则

默认情况下，所有的插件都可以通过设置特定的HTML代码和相应的属性（或自定义属性）来实现。也就是说，在网页加载的时候，JavaScript代码会自动检测这些标记，并自动绑定相应的事件，而无需再添加额外的JavaScript代码。示例如下：

```
<div class=“alert”>
                        <button type=“button” class=“close” data-
dismiss=“alert”>×</button>
        <strong>警告!</strong> 你输入的项目不合法！
    </div>
```

上述代码是警告框组件的HTML布局，只要在button元素上添加data-dismiss=“alert”属性，那么在单击该button的时候就会关闭该警告框。

同理，如下代码是下拉菜单的HTML布局，只要保证所单击的button按钮添加了data-toggle=“dropdown”属性，在单击按钮的时候，默认隐藏的下拉菜单就会显示出来。

```
<div class=“btn-group”>
        <button type=“button” class=“btn btn-default dropdown-toggle”
        data-toggle=“dropdown”>
            我的书籍 <span class=“caret”></span>
        </button>
        <ul class=“dropdown-menu”>
            <li><a href=”#”>JavaScript编程精解</a></li>
            <li><a href=”#”>JavaScript设计模式</a></li>
            <li><a href=”#”>JavaScript启示录</a></li>
            <li class=“divider”></li>
            <li><a href=”#”>深入理解Bootstrap3</a></li>
        </ul>
    </div>
```

再看一下更复杂的场景。如下代码是选项卡的HTML代码，并且在第三个选项卡（Dropdown）里加入了下拉菜单组件，也就是说单击第三个选项卡（Dropdown）的时候，下拉菜单会弹出来让用户再次选择一个菜单。但无论选择哪个li元素，插件都会根据配置的内容（如，data-toggle=“tab” href=”#home”），显示href所对应的id标签元素（即，单击首页，就会显示id=“home”的div）。

```
<ul class=“nav nav-tabs” id=“myTab”>
          <li class=“active”><a data-toggle=“tab” href=”#home”>首页</a>
</li>
          <li class=””><a data-toggle=“tab” href=”#profile”>个人资料</a>
</li>
        <li class=“dropdown”>
            <!— 单击”我的书籍”时,弹出下拉菜单 —>
             <a data-toggle=“dropdown” href=”#” class=“dropdown-toggle”>
我的书籍
```

```
                <b class="caret"></b></a>
            <ul class="dropdown-menu">
                <li><a data-toggle="tab" href="#dropdown1">JavaScript编程精解</a></li>
                <li><a data-toggle="tab" href="#dropdown2">JavaScript设计模式</a></li>
                <li><a data-toggle="tab" href="#dropdown3">JavaScript启示录</a></li>
                    <li><a data-toggle="tab" href="#dropdown4">深入理解Bootstrap</a></li>
            </ul>
        </li>
    </ul>
    <div class="tab-content" id="myTabContent">
        <div id="home" class="tab-pane fade active in"><p>单击"首页"时显示该区域
            </p></div>
        <div id="profile" class="tab-pane fade"><p>单击"个人资料"时显示该区域</p></div>
        <div id="dropdown1" class="tab-pane fade"><p>单击"JavaScript编程精解"
            时显示该区域</p></div>
        <div id="dropdown2" class="tab-pane fade"><p>单击"JavaScript设计模式"
            时显示该区域</p></div>
        <div id="dropdown3" class="tab-pane fade"><p>单击"JavaScript启示录"时
            显示该区域</p></div>
            <div id="dropdown4" class="tab-pane fade"> <p>单击"深入理解Bootstrap3"
            时显示该区域</p></div>
    </div>
```

通过上面的3个示例，大家是不是感觉很棒？一行JavaScript代码都不用就能实现非常炫的效果，这就是JavaScript插件统一布局规则的优势。

注意

上述代码data-toggle="tab" href="#home"所实现的功能，使用data-toggle="tab" data-target="#home"也可以实现，JavaScript默认先检测data-target属性，如果没有，再检测href属性，如果还没有，则默认为父元素。另外，其他所有的JavaScript插件也都遵循同样的检测规则。

2.4.2 JavaScript实现步骤

Bootstrap里提供的所有JavaScript插件都遵守了统一的实现步骤，好处自然就不用多说了，除了维护方便以外，学习和自定义其他插件也都很方便。要做到统一，需要有5个步骤，如图2-16所示。

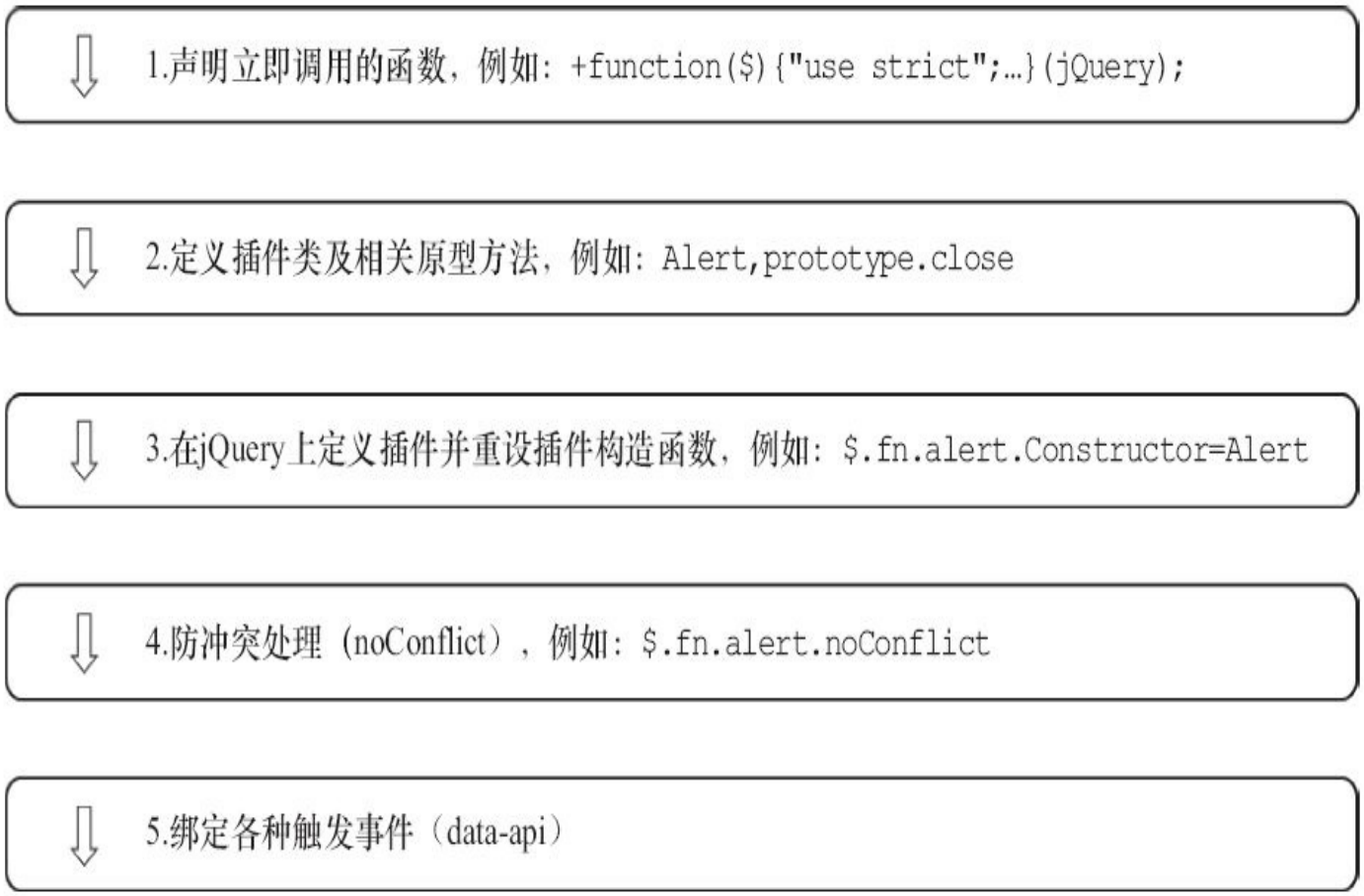


图2-16 Bootstrap的JS插件实现步骤示意图

下面以alert插件为例（alert.js），来讲述一下如何实现标准的插件。

步骤1 定义一个立即调用的函数声明，如代码清单2-1所示。在参数里传入jQuery对象，通过参数$引入变量，这样其所有代码在使用jQuery的时候，直接使用$符即可。这样的做法，有以下两个好处：

❑函数内部的$符变量代表了局部变量，而不是全局变量里代表jQuery的$符变量，以达到防止变量污染的目的。

❑内部的代码全部都是私有代码，外部代码无法访问，只有通过第三步，在$.fn上设置了插件（比如$.fn.alert=）的形式，通过$符变量才能将整个插件通过唯一的接口$.fn.alert暴露出去，从而保护了其内部代码。

代码清单2-1 立即调用的函数

```
+function ($) {
    “use strict”;// 1.使用严格模式ES5支持
    // 2.alert插件类及原型方法的定义
```

```
    // 3.在jQuery上定义alert插件,并重设插件构造器
    //  重设插件构造器,可以通过该属性获取插件的真实类函数
    // 4. 防冲突处理
    // 5. 绑定触发事件
}(window.jQuery);
```

注意

在function关键字前面有一个加号运算符（+），其主要目的是防止前面有未正常结束的代码（通常是遗漏了分号），导致前后代码被编译器认为是一体的，从而导致代码运行出错。

步骤2　定义该插件的核心代码，也就是在触发特定行为（通常是单击行为）后要进行处理的代码。示例如代码清单2-2所示。

代码清单2-2　插件核心代码示例

```
// alert插件类及原型方法的定义
// 定义选择器,所有符合该自定义属性的元素都可以触发下面的事件
var dismiss = '[data-dismiss="alert"]'
var Alert = function (el) {
    // 传入元素,如果元素内部有dismiss上设置的自定义属性,则click事件会触发原型上的close方法
    $(el).on('click', dismiss, this.close)
}
Alert.prototype.close = function (e) {
    // 关闭警告框的主要代码设置,每一行代码的细节注释,请参考5.8节
}
```

通过上述代码可以看出，主要是先定义了插件插件的类函数Alert，然后再定义需要用到的一些原型函数，比如close函数方法。Alert函数接收el参数，el表示DOM元素，一个DOM如果绑定了data-dismiss="alert"自定义属性，则在单击的时候就会触发close函数方法，从而达到关闭的目的。同样，Modal插件也是先定义Modal类函数，然后再在Modal的原型上定义toggle和show等方法，其内部根据相应的规则再细化处理。

步骤3　在jQuery上定义插件，以便通过jQuery.[插件名称]()的方式，也能够使用该插件，也就是在触发特定行为（通常是单击行为）后要进行处理的通用代码。最后在这里再调用插件类或原型方法。主要源码如代码清单2-3所示。

代码清单2-3　jQuery插件定义

```
// 在jQuery上定义alert插件,并重设插件构造器
var old = $.fn.alert
//  保留其他插件的$.fn.alert代码(如果定义),以便在noConflict之后,可以继续使用该旧代码
$.fn.alert = function (option) {
    return this.each(function () {
        // 根据选择器,遍历所有符合规则的元素,然后在元素上绑定插件的实例,以监控用户的事件行为
    })
}
```

```
$.fn.alert.Constructor = Alert;// 并重设插件构造器,可以通过该属性获取插件的真实类函数
```

jQuery插件的定义使用了标准的方法，在fn上进行扩展。在附加扩展之前，首先“备份”之前插件（或别的框架提供的同名插件）的旧代码，以方便在后面防冲突的时候使用。在附加扩展之后，重新设置插件的构造器（即Constructor属性）为内部定义的插件类函数自身，这样就可以通过Constructor属性查询到插件的真实类函数，使用new操作符实例化$.fn.alert的时候也不会出错。

注意

即便不声明第三步，HTML声明式的方式也是可以用的。所以说，第三步是专门为某些喜欢用JavaScript代码触发事件的人所准备的。但需要注意的是，如果不声明第三步，那第四步的防冲突的功能也就没法用了。

步骤4　防冲突处理，目的是让Bootstrap插件和其他UI库的同名插件共存。Bootstrap所有的插件都支持防冲突（noConflict）功能。源码如代码清单2-4所示。

代码清单2-4　防冲突处理

```
// 防冲突处理
$.fn.alert.noConflict = function () {
    $.fn.alert = old    // 恢复以前的旧代码
     return this            // 将$.fn.alert.noConflict()设置为Bootstrap的alert插件
}
```

这样一旦有了一个同名的插件，比如A库里有个同名$.fn.alert插件，则Bootstrap在执行之前就通过old先备份了，然后执行$.fn.alert.noConflict()后就会还原该old对象插件；而使用Bootstrap的alert插件的话，则通过var　alert=$.fn.alert.noConflict()的形式，将Bootstrap的alert插件转移到另外一个变量上，从而进行使用。

步骤5　在一切都就绪之后，绑定默认的触发事件。由于已经为jQuery提供了默认的$.fn.alert扩展插件功能，已经可以通过手工编写JavaScript代码来触发事件了。这里的第五步主要是为声明式的HTML触发事件，即：在HTML文档里已经按照布局规则声明了相关的自定义属性（比如data-dismiss=“alert”），然后通过这里的代码初始化默认的单击事件行为（或其他相关插件需要用到的行为）。绑定触发事件的源码如下所示：

```
// 绑定触发事件
// 为声明式的HTML绑定单击事件
// 在整个document对象上，检测是否有自定义属性data-dismiss=“alert”
// 如果有,则设置:单击的时候,关闭指定的警告框元素
```

```
$(document).on('click.bs.alert.data-
api', dismiss, Alert.prototype.close)
```

上述代码在整个document文档上检测自定义属性data-dismiss="alert"，如果有，则绑定click单击事件（在命名空间bs.alert.data-api上），事件回调函数则是原型方法Alert.prototype.close。这样，一旦单击了相应的元素，就会关闭特定的警告框。

Bootstrap通过这5个通用步骤，定义了所有的插件。每个插件在各个步骤里会有一些细节的不同，但思路都是一样的，那就是：绑定事件，触发行为，并在jQuery上扩展fn，同时解决防冲突的问题。大家在制作自定义插件的时候只要遵从这一思路，即可编写出既易于维护又高质量的插件代码。

2.4.3 通用技术

根据上述JavaScript插件的实现步骤，来总结一下JavaScript插件的通用技术，即Bootstrap的开发者在开发这些插件时所制定的规则和遵循的标准，同时也为我们制作自定义插件提供有力的参考。

首先，不同插件的JS代码都是单独放在一个JS文件中的，开发人员在使用的时候可以一次性编译到Bootstrap.js，也可以单独使用某一个或者多个JS插件文件。唯一需要注意的是：有些JS插件依赖于其他JS插件，所以不要遗漏了依赖引用。

Bootstrap所有的JavaScript插件都可以通过配置使用，即通过特定的HTML设置，而不需要任何JavaScript再次触发。但如果需要启用手动触发事件的行为，可以禁用默认的行为，禁用方法非常简单，只需要将body元素上的命名空间为data-api下的全部事件禁用即可。代码如下所示：

```
$(document).off('.data-api');
```

如果想禁用特定插件的默认行为，只需要禁用该插件所在命名空间下的事件即可。代码如下所示：

```
$(document).off('.alert.data-api');                    /* 禁用alert插件的所有默认行为 */
```

注意

off语法是jQuery提供的语法功能，用户在使用on进行绑定事件的时候，可以加命名空间，比如$().on('click.alert.data-api')，这样在卸载事件的时候，如果只想卸载该元素的该特定事件，可以使用off('click.alert.data-api')。如果不这样，仅仅使用off('click')，这样该元素上的所有click事件都将被卸载（导致该元素的其他click事件失效）。同理，如果执行off('.data-api')代码，则所有在data-api命名空间下的事件都会被卸载禁用，不管是该选择器内部的哪个元素、哪种事件。

1.可编程性

所有的插件不仅可以使用声明式定义（HTML），也可以通过JavaScript代码全部实现。利用jQuery的链式操作，编出的代码非常优美。示例如下所示：

```
$(".btn.btn-danger").button("toggle").addClass("fat");
```

所有的插件在使用JavaScript代码调用的时候，都提供多种调用方式：无参数传递（即默认方式）、传递对象字面量进行初始化参数设定、直接传入一个需要执行的方法名称字符串。示例如下所示：

```
$(“#myModal”).modal()                                    // 默认值进行初始化
$(“#myModal”).modal({ keyboard: false })          // 初始化时keyboard选项值是false
$(“#myModal”).modal(‘show’)                          // 初始化,然后立即调用show方法
```

每个插件都有一个Constructor属性，用于表示原始的构造函数，比如$.fn.alert.Constructor。另外也可以通过$（’选择符’）.data（‘bs.插件名称’）的形式（如，$(‘[data-dismiss=“alert”]’).data(‘bs.alert’)）获取该特定插件的实例。

2.防冲突

和jQuery一样，Bootstrap插件可以和其他同类插件共存，为了防止$.fn.下的Bootstrap插件被覆盖，Bootstrap也提供了防冲突功能（No conflict），用于定义别名。示例用法如下：

```
var bootstrapButton = $.fn.button.noConflict()
// 返回$.fn.button对象给bootstrapButton变量

$.fn.bootstrapBtn = bootstrapButton
//   将button对象给bootstrapButton变量赋予一个新插件名称$().bootstrapBtn,这时$().
// bootstrapBtn就拥有了先前button的所有功能了
```

3.自定义事件

Bootstrap为很多插件都提供了自定义事件功能，比如，modal弹窗里提供的show和shown事件，show事件在弹窗初始化（即将弹出）的时候触发，而shown事件则是在弹窗初始化完毕后（完全弹出）才触发。

在新版插件里，所有的事件都是命名空间化的，即单个事件都要放在某个命名空间下，比如，show.bs.modal。

所有的插件都提供了preventDefault功能，用于阻止继续执行后续的代码。例如，可以在modal弹窗的show事件里进行判断，如果不符合条件就拒绝显示弹窗。示例代码如下所示：

```
$(‘#myModal’).on(‘show.bs.modal’, function (e) {
    if (!data) return e.preventDefault()        // 拒绝显示弹窗
})
```

2.4.4 不足

虽然Bootstrap为我们提供了丰富多彩的插件，但是其插件代码在编程规范上却存在很多不足。这里所谓的不足是指：虽然编译器可以识别这些代码，但它不符合大多数JavaScript开发者所使用的规范。Bootstrap中不规范的代码示例如下：

```
var Modal = function (element, options) {
    this.options = options                          // 这里少了分号
                    this.$element     =     $(element).delegate('[data-
dismiss="modal"]', 'click.dismiss.
        modal', $.proxy(this.hide, this))           // 这里少了分号
     this.$backdrop =              // 这里使用了链式赋值,其值是this.isShown的
null值
    this.isShown = null                             // 这里少了分号

                if    (this.options.remote)    this.$element.find('.modal-
body').load(this.options.
        remote)
    // if语句应该用花括号括住
}
```

通过注释大家可以发现，其中省略了很多分号，并且使用了简便方法。虽然现代的JavaScript解释器会对这样的代码进行正常解释，但是其可读性变差。建议在编写自定义插件的时候参照如下示例代码：

```
var Modal = function (element, options) {
    this.options = options;
                    this.$element     =     $(element).delegate('[data-
dismiss="modal"]', 'click.dismiss.modal',
            $.proxy(this.hide, this));          // 复杂的语句最好有点注释
    this.$backdrop = "";
    this.isShown = null;
    if (this.options.remote) {
        this.$element.find('.modal-body').load(this.options.remote);
    }
}
```

2.5　禁用响应式布局

Bootstrap是一个移动先行的框架，默认情况下，针对不同的屏幕尺寸，会自动地调整页面，使其在不同尺寸的屏幕上都表现得很好。但是，如果不想使用这种特性，也可以禁用它。下面列出了禁用响应式布局的步骤：

1）删除名称为viewpot的meta元素，例如：<meta name=“viewport”……/>。

2）为.container设置一个固定的宽度值，从而覆盖框架的默认width设置，例如width: 970px!important；。并且要确保这些设置全部放在默认的Bootstrap CSS后面。

3）如果使用了导航条组件，还需要移除所有的折叠行为和展开行为。

4）对于栅格布局，额外增加.col-xs-*样式，或替换.col-md-*和.col-lg-*样式。不要太担心，超小屏幕设备的栅格系统样式可以适应于所有分辨率的环境。

对于IE8来说，由于仍然需要媒体查询语法，所以还需要引入Respond.js文件，这样就禁用了Bootstrap对小屏幕设备的响应式支持。

第3章　CSS布局

本章的内容是Bootstrap三大核心内容的基础，即基础的CSS布局语法。其包括基础排版（Typography）、代码（Code）、表格（Tables）、表单（Forms）、按钮（Buttons）、图片（Images）、辅助类（Helper Classes）和响应式设计（Responsive　utilities）。这些基础的布局语法都是通过CSS最基础、最简单的组合来实现的，通过这些基础而又核心的布局语法，不需要太多时间，即可快速上手，制作出比较精美的页面。

3.1 概述

Bootstrap 3.x的目标是：简洁、直观、强悍的前端开发框架，让Web开发更迅速、简单。在使用之前，我们需要先了解一下Bootstrap的基础内容。

1.HTML5文档类型

由于Bootstrap使用了HTML5特定的HTML元素和CSS属性，所以使用Bootstrap的时候，所有的HTML文件都需要在其顶部引用HTML5的DOCTYPE属性，如下所示：

```
<!DOCTYPE html>
<html lang="en">
  …
</html>
```

2.移动先行

Bootstrap2.x系列版本中，在框架里为移动特性添加了一些可选支持。而在Bootstrap3.x版本中，作者重写了与移动特性相关的内容，与可选特性相比，新版本将移动特性加入了核心框架。实际上，新版Bootstrap是一个移动先行的框架集，移动先行的影子在整个框架集都可以发现，如下所示：

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

在移动设备浏览器上，通过为viewport meta标签添加的user-scalable=no可以禁用其缩放（zooming）功能。禁用缩放功能后，用户只能滚动屏幕，这样能让你的网站看上去更像原生应用。注意，我们并不推荐所有网站都使用这种方式，是否使用这种方式要视情况而定。

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
```

3.响应式图片

为了能对图片的大小进行自适应缩放，Bootstrap在3.x版里添加了一个.img-responsive样式，其实质是为图片设置了max-width: 100%；和height: auto；属性，以便可以让图片按比例缩放，并且不超过其父元素的尺寸，所以说该样式对响应式布局的支持更加友好了。使用的时候，只需要在相应图片元素上加一个.img-responsive样式即可（而默认情况下，如果不加该样式，则显示还是按照默认设置来进行的）。.img-responsive样式实现代码如下所示：

```
// 源码310行
.img-responsive {
```

```
  display: block;
  max-width: 100%;
  height: auto;
}
```

4.排版与链接

Bootstrap为网页设置了一些基本的全局样式、排版和链接风格，尤其是其具有如下特性：

❑在body元素上取消了margin设置，改为默认为0，margin: 0。

❑在body元素上设置了背景色为白色，background-color: #ffffff。

❑在布局排版方面，字体、字体大小、行间距使用的标准值分别如下：

```
font-family: “Helvetica Neue”, Helvetica, Arial, sans-serif;
            font-size: 14px;
line-height: 1.428571429;                    // 20除14
```

所有的默认基本样式都可以在normalize.less和scaffolding.less文件里找到。

5.Normalize.css

为了改进跨浏览器呈现，Bootstrap使用了第三方CSS库Normalize。Normalize.css是一个专门用于将不同浏览器的默认CSS特性设置为统一效果的CSS库，以便开发人员使用更标准的方式去访问或设置页面元素。该项目只专注于样式，比如button按钮在不同浏览器上效果都是一致的。详细内容可以查看源码里的Normalize.less文件。

项目地址如下：http: //necolas.github.io/normalize.css/。

6.居中容器

一个页面（或元素）要居中显示，可以在外部容器上简单应用.container样式即可。由于栅格系统依赖于外部容器的大小设置，所以默认情况下container样式有一个max-width属性用于限制栅格系统的最大宽度。示例代码如下所示：

```
<div class=“container”>
  …
</div>
```

container样式的源码如下：

```
// 源码734行
.container {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;                        /*左右居中的设置*/
```

```
  margin-left: auto;
}
```

注意

container的max-width大小和设备的最大分辨率有关，平板电脑一般为728像素，普通桌面浏览器一般为940像素，大型浏览器为1170像素。也可以通过重载覆盖的方式修改max-width。

3.2 基础排版

源码文件：type.less

CSS文件：bootstrap.css 349行开始

本节主要是讲述HTML页面基础排版（Typography）方面所需要的各种技能，比如head标题（h1至h6）、地址、列表、文本块等。在平时的工作中，大家可能已经在使用了，但是Bootstrap在此基础上做了不少优化工作，使其更方便使用。这里只是简单叙述一下各项语法的示例和优化方式。

3.2.1 标题

Bootstrap为传统的标题元素h1～h6重新定义了标准的样式，使得在所有浏览器下显示效果都一样，具体定义规则如表3-1所示。

表 3-1 h1 ~ h6 定义规则

元 素	字体大小	计算比例	其 他
h1	36px	14px x 2.60	margin-top:20px; margin-bottom:10px;
h2	30px	14px x 2.15	
h3	24px	14px x 1.70	
h4	18px	14px x 1.25	margin-top:10px; margin-bottom:10px;
h5	14px	14px x 1.00	
h6	12px	14px x 0.85	

标题元素的用法和平时的用法一致，示例如下所示：

```
<h1>Bootstrap权威指南</h1>
<h2>Bootstrap权威指南</h2>
<h3>Bootstrap权威指南</h3>
<h4>Bootstrap权威指南</h4>
<h5>Bootstrap权威指南</h5>
<h6>Bootstrap权威指南</h6>
```

Bootstrap还同步定义了6个class样式（.h1～.h6），以便在非标题元素下使用相同的样式，唯一的不同是class样式没有定义margin-top和margin-bottom。如下所示：

```
<span class=“h1”>Bootstrap权威指南</span><br />
<span class=“h2”>Bootstrap权威指南</span><br />
<span class=“h3”>Bootstrap权威指南</span><br />
<span class=“h4”>Bootstrap权威指南</span><br />
<span class=“h5”>Bootstrap权威指南</span><br />
<span class=“h6”>Bootstrap权威指南</span><br />
```

使用h元素和h样式进行显示时，两者的区别如图3-1所示。

a）h元素　　b）h样式

图3-1　h元素和h样式的运行效果比较

大部分情况下，在标题元素里可能会应用<small>元素，以便显示稍微小一点的字体。Bootstrap为此也特别定义了样式，如图3-2所示。具体内容如下：

图3-2　small元素分别在h1～h6元素内的效果

❑所有标题元素里的<small>内容的字体颜色都是灰色（#999999），行间距都为1。

❑<small>内的文本字体在h1、h2、h3内是当前元素所对应字体大小的65%；而在h4、h5、h6下则是75%（详细见448行和464行定义的small样式）。

3.2.2 页面主题

默认情况下，Bootstrap为全局设置的字体大小font-size为14像素，间距line-height为字体大小的1.428倍（即20像素）。该设置应用于<body>元素和所有的段落上。

```
// 源码275行
body {
  font-family: “Helvetica Neue”, Helvetica, Arial, sans-serif;
  font-size: 14px;
  line-height: 1.428571429;
  color: #333;
  background-color: #fff;
}
```

另外，<p>元素内的段落会有一个额外的margin-bottom，大小是行间距的一半（默认为10px），详细见CSS文件bootstrap.css中的497行。

```
// 源码464行
p {  margin: 0 0 10px; }
```

如果想让一个段落突出显示，可以使用.lead样式，其作用主要是增大字体大小、粗细、行间距和margin-bottom。用法如下：

```
<p class=“lead”>…</p>
```

lead样式的代码实现如下所示：

```
// 源码467行
.lead {
  margin-bottom: 20px;
  font-size: 16px;
  font-weight: 200;
  line-height: 1.4;
}
@media (min-width: 768px) {                    /*大中型浏览器字体稍大*/
  .lead {    font-size: 21px;  }
}
```

Bootstrap的排版设置默认值存储在variables.less文件里的两个LESS变量里：@font-size-base和@line-height-base。第一个用于设置字体大小，第二个用于设置行间距。系统默认使用这两个值产生整个页面相应的margin、padding和line-height。通过修改这两个值后，再重新编译，从而制定自己的Bootstrap版本。

3.2.3 强调文本

Bootstrap将默认的文本强调元素进行了轻量级实现，这些元素分别为：small、strong、em、cite。

```
// 源码56行
b,strong {  font-weight: bold;                    /* 粗体 */}
// 源码478行
small,.small {  font-size: 85%;                   /* 标准字体的85% */}
cite {  font-style: normal;                       /* 正常字体*/}
```

同样的原理，Bootstrap也为对齐方式定义了简单而又明了的4个样式以便使用。使用方式如下：

```
<p class=“text-left”>JavaScript编程精解</p>
<p class=“text-center”>JavaScript设计模式</p>
<p class=“text-right”>JavaScript启示录</p>
<p class=“text-justify”>Backbone应用开发实战</p>
```

3.2.4 缩略语

Bootstrap在abbr元素上实现了缩略词的功能，即鼠标移动到缩略词上时，就显示声明在title里的属性值。效果为：缩略词下面有虚横线，鼠标移动到缩略词上时有手形图标。其源码如下所示：

```
// 源码621行
abbr[title],
abbr[data-original-title] {
  cursor: help;
  border-bottom: 1px dotted #999999;
}
.initialism {
  font-size: 90%;  text-transform: uppercase;
}
```

从源码可以看出，有两种方式可以使用，一种是直接使用abbr，另外一种是应用.initialism样式以使字体稍微缩小一点。

```
<abbr   title=“JavaScript设计模式是一本专门讲解设计模式的专业书籍”>JavaScript设计模
        式</abbr>
<abbr title=“HyperText Markup Language” class=“initialism”>HTML</abbr>
```

3.2.5 地址元素

Bootstrap为地址元素address定义了一个简单通用的样式，其主要是行间距和底部的margin。源码如下：

```
// 源码682行
address {
  margin-bottom: 20px;
  font-style: normal;
  line-height: 1.428571429;
}
```

address的用法也比较简单，每一行用
结尾即可。

```
<address>
        <strong>Twitter, Inc.</strong><br>
        795 Folsom Ave, Suite 600<br>
        San Francisco, CA 94107<br>
        <abbr title=“Phone”>P:</abbr>
        (123) 456-7890
    </address>
    <address>
        <strong>汤姆大叔</strong><br>
        <a href=“mailto:#”>tomxu@outlook.com</a>
    </address>
```

3.2.6 引用

在<blockquote>元素里进行引用，可以引用任意HTML内容，但一般推荐使用<p>。Bootstrap也为此定义了一个通用的样式，用法如下所示：

```
<blockquote>
        <p>不愤不启,不悱不发。举一隅,不以三隅反,则吾不复也。</p>
</blockquote>
```

运行上述代码后会看到图3-3所示的效果。

图3-3 blockquote元素运行效果

如果想加上一些文字的出处作为注释，则可以配合使用small和cite元素，效果会更好，如图3-4所示。

```
<blockquote>
        <p>不愤不启,不悱不发。举一隅,不以三隅反,则吾不复也。</p>
        <small>出自 <cite title="论语•述而">论语</cite></small>
</blockquote>
```

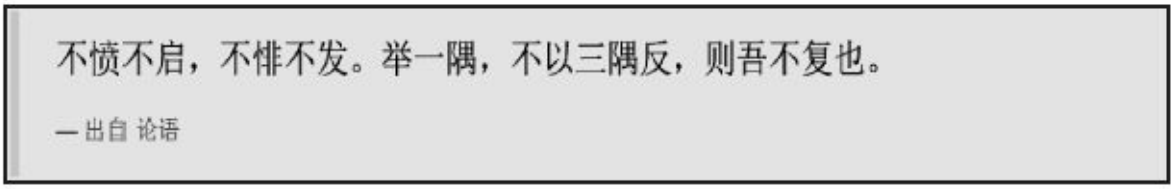


图3-4 配合small和cite的运行效果

另外，Bootstrap还提供了一个.pull-right样式用于右对齐，以适应不同的排版方式。

```
<blockquote class="pull-right">
        <p>不愤不启,不悱不发。举一隅,不以三隅反,则吾不复也。</p>
        <small>出自 <cite title="论语•述而">论语</cite></small>
  </blockquote>
```

上述代码运行后的效果如图3-5所示。

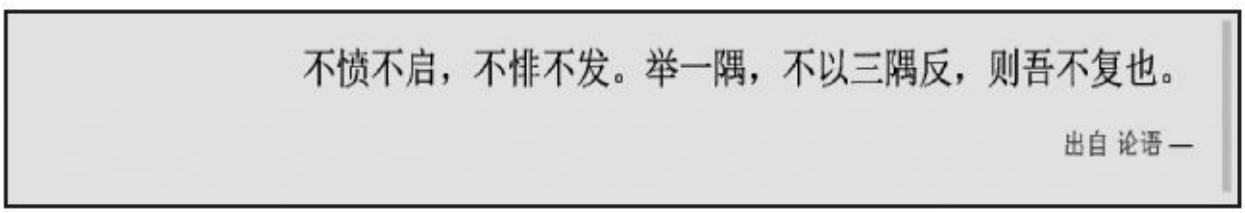


图3-5 右对齐运行效果

引用元素的主要样式代码如下：

```
// 源码630行
blockquote {
  padding: 10px 20px;
  margin: 0 0 20px;
  font-size: 17.5px;
  border-left: 5px solid #eee;
}
/* 此处省略部分代码 */
.blockquote-reverse,
blockquote.pull-right {
  padding-right: 15px;
  padding-left: 0;
  text-align: right;
  border-right: 5px solid #eee; /* 右边显示的竖线 */
  border-left: 0;
}
```

3.2.7　列表

Bootstrap提供了6种形式的列表，分别是：普通列表、有序列表、去点列表、内联列表、描述列表和水平描述列表。

1.普通列表

普通列表的使用方式和人们平时用的一样，Bootstrap只是在此基础上做了一些细微的优化。示例如下：

```
<ul>
  <li>…</li>
</ul>
```

运行效果如图3-6所示。

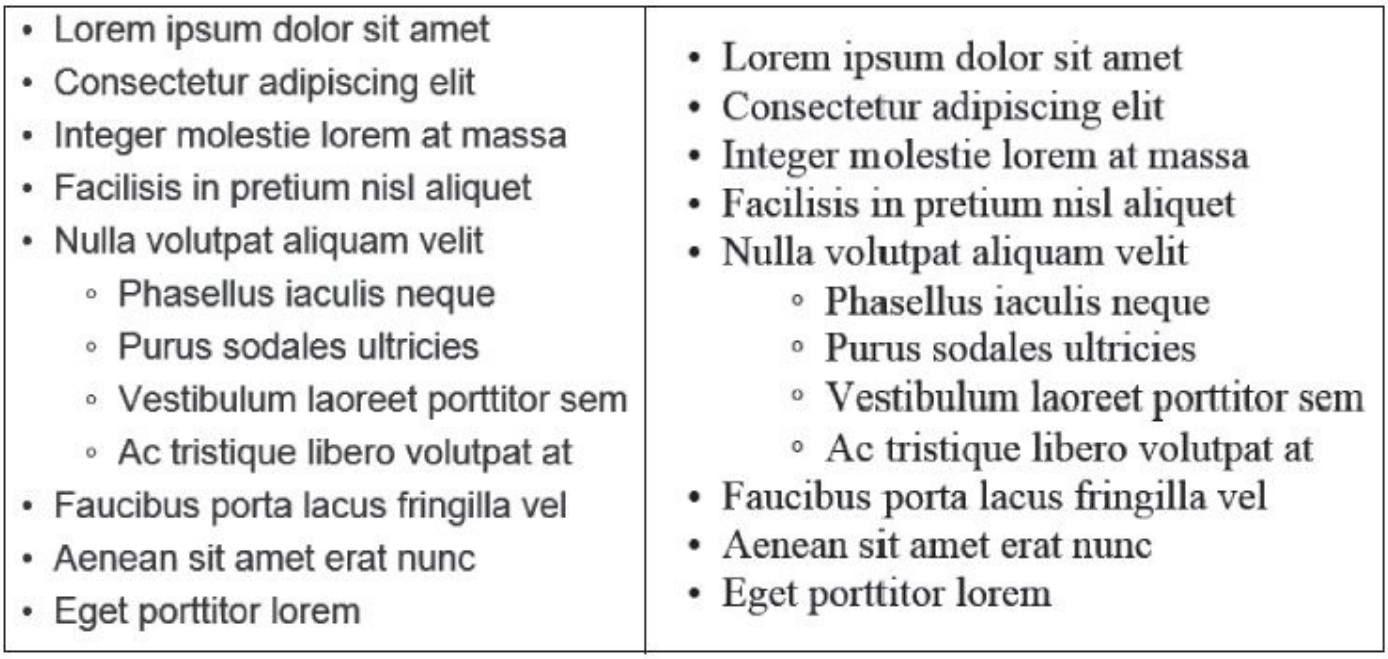


图3-6　普通列表在Bootstrap样式和默认样式下的运行效果

由图3-6所示的运行效果可以看出，Bootstrap在margin和行间距做了一些微调。源码如下：

```
// 源码566行
ul,ol {
  margin-top: 0;
  margin-bottom: 10px;
}
ul ul,ol ul,ul ol,ol ol {
  margin-bottom: 0;
}
```

2.有序列表

根据上述源码，可以看出使用了ol元素的有序列表，也是做了一些微调，看起来比原来的默认显示柔和了一些。示例用法如下：

```
<ol>
  <li>…</li>
</ol>
```

运行效果如图3-7所示。

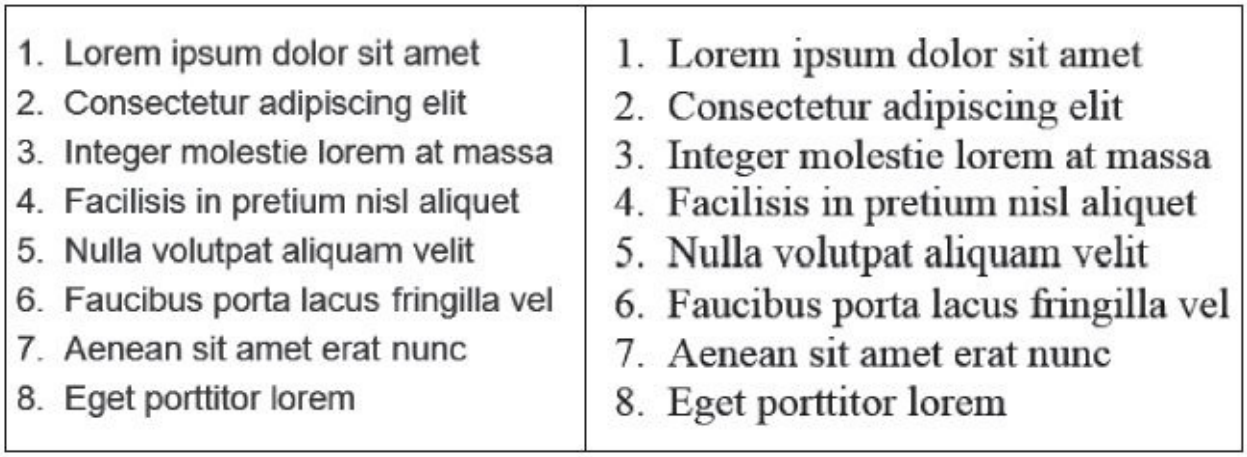


图3-7　有序列表在Bootstrap样式和默认样式下的运行效果

3.去点列表

Bootstrap提供了一个去除默认列表样式的风格，即去除普通列表项前面的圆点。用法如下：

```
<ul class="list-unstyled">
        <li>…</li>
        <li>
            <ul>
                <li>…</li>
                <li>
                    <ul>
                        <li>…</li>
                        <li>…</li>
                    </ul>
                </li>
            </ul>
        </li>
    </ul>
```

运行效果如图3-8所示。

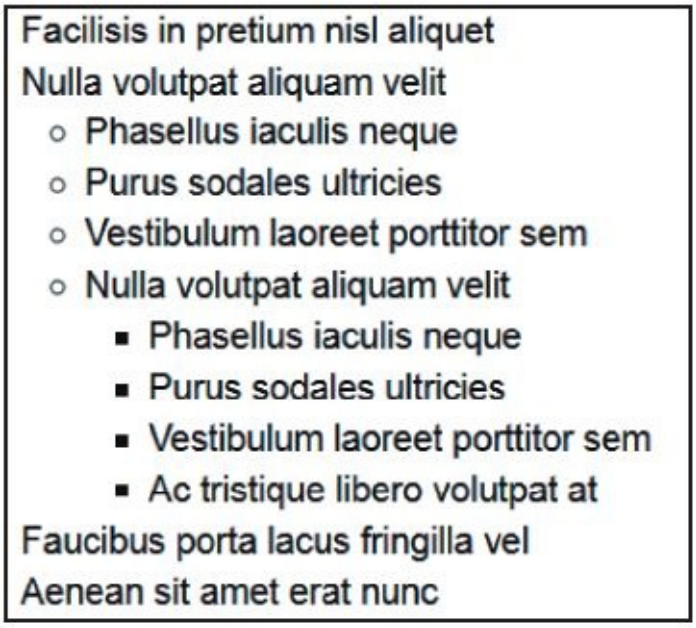


图3-8　去点列表的运行效果

去点列表的list-unstyled样式源码如下：

```
// 源码577行
.list-unstyled {
  padding-left: 0;
  list-style: none;
}
```

由运行效果图和源码可以看出，去除圆点只是去除了当前元素节点下的li旁边的圆点，如果li元素里有另外一个嵌套的ul或者ol，将不会起任何作用。也就是说如果要让嵌套的列表项也去除圆点（或其他标示符），则还需要单独应用.list-unstyled样式。

4.内联列表

Bootstrap定义了一个list-inline样式用于实现内联列表，也就是将列表项水平显示，实现效果如图3-9所示。

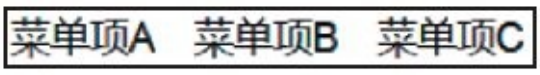


图3-9　水平列表的运行效果

list-inline样式的源码如下：

```
// 源码581行
.list-inline {
  padding-left: 0;
  list-style: none;
}
.list-inline > li {
  display: inline-block;
  padding-right: 5px;
  padding-left: 5px;
```

```
}
.list-inline > li:first-child {  padding-left: 0;}
```

5.定义列表

Bootstrap对默认的dl定义列表也做了一些轻微的调整，主要是调整行间距和字体加粗。用法和对比效果如下：

```
<dl>
  <dt>…</dt>
  <dd>…</dd>
</dl>
```

运用上述用法后的效果如图3-10所示。

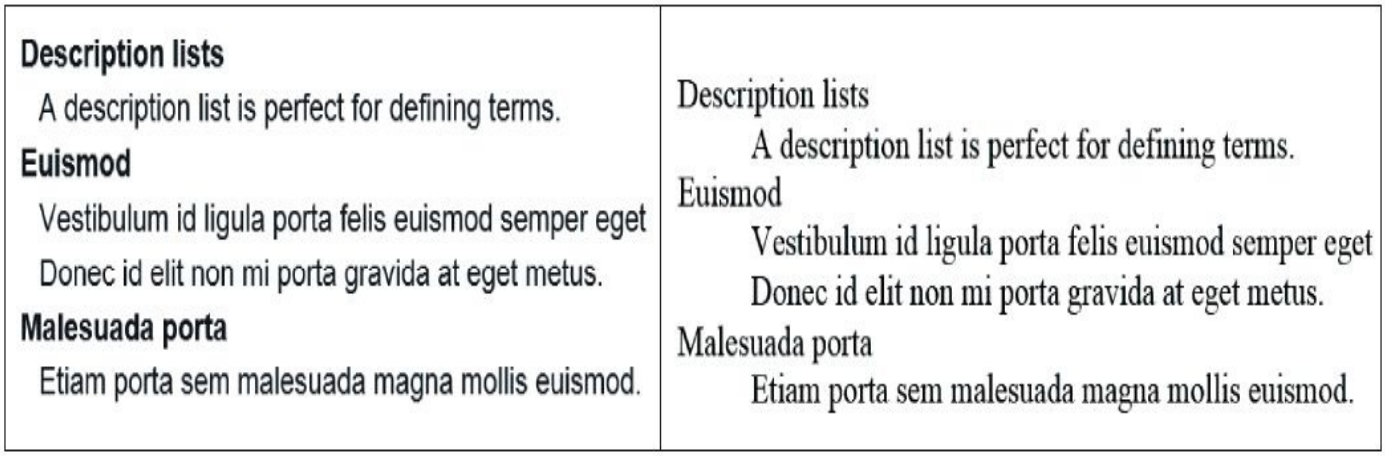


图3-10　定义列表在Bootstrap样式和默认样式下的运行效果

定义列表样式的处理源码如下：

```
// 源码593行
dl {  margin-top: 0;  margin-bottom: 20px;}
dt,dd {  line-height: 1.428571429;}
dt {  font-weight: bold;}
dd {  margin-left: 0;}
```

6.水平定义列表

Bootstrap提供了一个dl-horizontal样式，通过在dl元素上应用该class，实现了列表水平显示的效果。用法如下：

```
<dl class=“dl-horizontal”>
  <dt>…</dt>
  <dd>…</dd>
</dl>
```

运用上述用法后的效果如图3-11所示。

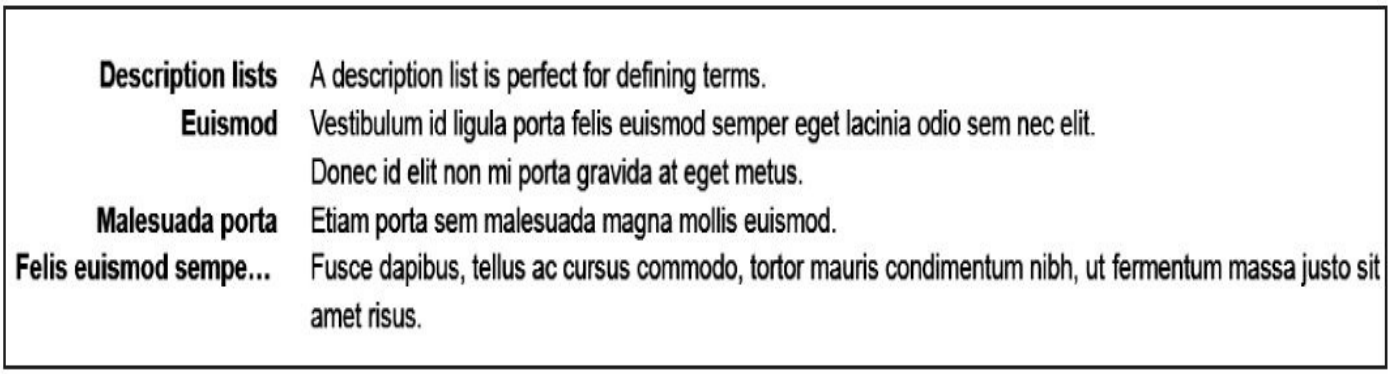


图3-11　运行效果

水平定义列表的主要实现方式是将dt进行left浮动，同时设置宽度为160像素，再将dd的margin-left设置为180像素，达到水平的效果。值得注意的是，如果dt标题超出长度将被自动隐藏。详细见效果图的第四项标题和源码里的text-overflow设置。

```
  // 源码607行
@media (min-width: 768px) {       /*这种样式只能在平板电脑或更大的浏览器上才能用 */
  .dl-horizontal dt {
    float: left;
    width: 160px;
    overflow: hidden;
    clear: left;
    text-align: right;
    text-overflow: ellipsis;            /* 超过160像素就自动隐藏 */
    white-space: nowrap;
  }
  .dl-horizontal dd {    margin-left: 180px;  }
}
```

3.3 代码

源码文件：code.less

CSS文件：bootstrap.css 103行开始

本小节的内容比较简单，主要是介绍代码显示的问题。Bootstrap对代码显示提供了以下3种方式：

❑使用<code>元素显示单行内联代码

❑使用<kbd>元素显示用户输入代码

❑使用<pre>元素新生多行代码块

代码样式的用法示例如下：

```
要对 <code>&lt;body&gt; &lt;/body&gt;</code> 进行设置。      <!-- 内联代码 -->
<pre>&lt;body&gt; &lt;/body&gt; </pre>     <!-- 多行代码 -->
```

3.3.1 内联代码

单行的内联代码需要使用code元素包含。用法如下：

```
要对 <code>&lt;body&gt; &lt;/body&gt;</code> 进行设置。
```

内联代码效果主要是设置code元素的背景颜色和其内部的文字颜色而已。源码如下：

```
// 源码693行
code {
  padding: 2px 4px;
  font-size: 90%;
  color: #c7254e;                    /* 文字颜色为深红*/
  white-space: nowrap;
  background-color: #f9f2f4;         /* 背景颜色为浅红*/
  border-radius: 4px;                /* 设置圆角*/
}
```

3.3.2 用户输入代码

<kbd>元素包含的内容是表示该内容需要用户键盘输入，所以一般是设置成input的效果，用法和code类似，其实现源码如下：

```
// 源码701行
kbd {
  padding: 2px 4px;
  font-size: 90%;
  color: #fff;                                    /* 文字颜色为白色*/
  background-color: #333;                         /* 背景颜色为深黑色*/
  border-radius: 3px;                            /* 设置圆角*/
  box-shadow: inset 0 -1px 0 rgba(0, 0, 0, .25);  /* 设置阴影*/
}
```

3.3.3 多行代码块

pre元素一般用于显示大块的代码段，并保证原有格式不变。另外可以在pre元素上应用.pre-scrollable样式，则可以控制该区域最大高度为340像素，并可以在y方向滚动。

```
// 源码709行
pre {
  display: block;
  /* 省略部分代码*/
  word-break: break-all;
  word-wrap: break-word;
  background-color: #f5f5f5;                                    /* 背景颜色为浅灰色*/
  border: 1px solid #ccc;                                       /* 边框为深灰色*/
  border-radius: 4px;
}
.pre-scrollable {
  max-height: 340px;                                            /* 固定高度*/
  overflow-y: scroll;                                           /* 竖向可滚动*/
}
```

3.4 表格

源码文件：table.less

CSS文件：bootstrap.css 1401行开始

在表格组件里，Bootstrap提供了1种基础.table样式、4种附加样式（.table-striped、.table-bordered、.table-hover、.table-condensed）以及一个支持响应式布局的.table-responsive容器样式。从第2章的设计思想我们可以知道，每种附加特效都是在.table样式的基础上联合应用才生效的。

3.4.1 基础样式

只需要在table元素上应用.table样式即可制作出比较漂亮的表格。示例如下：

```
<table class="table">
   …
</table>
```

运行效果如图3-12所示。

#	图书名称	出版社	作译者
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）
4	深入理解Bootstrap	机械工业出版社	汤姆大叔

图3-12　基础表格运行效果

.table样式的主要作用有3个：增加单元格的内边距、在thead的底部设置一条2像素的粗线，以及在每行记录的顶部都有1条1个像素的细线。主要源码如下：

```
// 源码1401行
table {  max-width: 100%;  background-color: transparent;}
th {  text-align: left;}
.table {  width: 100%;  margin-bottom: 20px;}
.table > thead > tr > th,
.table > tbody > tr > th,
.table > tfoot > tr > th,
.table > thead > tr > td,
.table > tbody > tr > td,
.table > tfoot > tr > td {
  padding: 8px;                              /*设置单元格的内边距*/
  line-height: 1.428571429;
  vertical-align: top;
  border-top: 1px solid #ddd;                /* 每行记录的顶部都有1条1个像素宽的
横线 */
}
.table > thead > tr > th {
  vertical-align: bottom;
  border-bottom: 2px solid #ddd;             /* thead有1条2个像素宽的横线 */
}
```

```
/*省略部分样式*/
.table > tbody + tbody {
  border-top: 2px solid #ddd;
                                    /* 如果表格里有2个tbody,两者之间也会有1条2个像素宽的横线 */
}
```

3.4.2　带背景条纹的表格

背景条纹表格，就是在现有.table样式的基础上再应用一个.table-striped样式，其添加了隔行加背景色的设置。使用方式如下：

```
<table class="table table-striped">
    …
</table>
```

运行效果如图3-13所示。

#	图书名称	出版社	作译者
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）
4	深入理解Bootstrap	机械工业出版社	汤姆大叔

图3-13　条纹背景表格的运行效果

隔行换色的实现方式是利用了CSS 3里的: nth-child选择器来实现的，所以不支持IE 8及以下版本。源码如下：

```
// 源码1464行
.table-striped > tbody > tr:nth-child(odd) > td,
.table-striped > tbody > tr:nth-child(odd) > th {
  background-color: #f9f9f9;
                          /* 如果需要更换颜色,需要重载覆盖才行(详情阅读第2章的2.3
节组件架构) */
}
```

3.4.3　带边框的表格

在该特效下，Bootstrap为表格所有的单元格提供了1条1像素宽的边框。使用方式如下：

```
<table class="table table-bordered">
    …
</table>
```

运行效果如图3-14所示。

#	图书名称	出版社	作译者
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录		汤姆大叔（译）
4	深入理解Bootstrap	机械工业出版社	汤姆大叔

图3-14　带边框的表格运行效果

v2.x版本的边框表格带有圆角，其实现方式非常复杂。而最新的3.x版本直接抛弃了圆角，这样其实现方式就简洁了许多，主要就是设置了整体表格和单元格的边框。实现源码如下：

```
// 源码1449行
.table-bordered {
  border: 1px solid #dddddd;                    /* 整体表格边框 */
}
.table-bordered > thead > tr > th,
.table-bordered > tbody > tr > th,
.table-bordered > tfoot > tr > th,
.table-bordered > thead > tr > td,
.table-bordered > tbody > tr > td,
.table-bordered > tfoot > tr > td {
  border: 1px solid #dddddd;                    /* 单元格边框 */
}
.table-bordered > thead > tr > th,
.table-bordered > thead > tr > td {
  border-bottom-width: 2px;                     /* 表头底部边框 */
}
```

3.4.4 鼠标悬停高亮的表格

在表格中的记录上，当鼠标移动上去时对应的部分能够换个颜色的话，那就会显得很有动感，并且有高亮显示的功能。Bootstrap当然也不会错过这个亮点，其提供了一个.table-hover样式。在Bootstrap下的使用方式如下：

```
<table class=“table table-hover”>
    …
</table>
```

悬停高亮的实现方式是在tr的hover事件中，设置tr元素内所有的td和th的背景色为新背景色。同样，如果需要更换颜色，需要对它进行重载覆盖设置。如下所示：

```
// 源码1468行
.table-hover > tbody > tr:hover > td,
.table-hover > tbody > tr:hover > th {
  background-color: #f5f5f5;
}
```

3.4.5　紧凑型表格

所谓紧凑型表格，即表格的显示比普通表格稍微紧凑一些，实现原理是减少单元格的内边距。使用方式如下：

```
<table class="table table-condensed">
    …
</table>
```

运行效果如图3-15所示。

#	图书名称	出版社	作译者
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）
4	深入理解Bootstrap	机械工业出版社	汤姆大叔

图3-15　紧凑型表格的运行效果

默认情况下表格的padding是8像素，紧凑型表格的padding值是5像素。源码如下：

```
// 源码1441行
.table-condensed > thead > tr > th,
.table-condensed > tbody > tr > th,
.table-condensed > tfoot > tr > th,
.table-condensed > thead > tr > td,
.table-condensed > tbody > tr > td,
.table-condensed > tfoot > tr > td {
  padding: 5px;
}
```

以上4种特效的样式可以混合在一起使用，从而制作出复杂而精美的表格。

3.4.6　行级元素样式

Bootstrap为表格的tr元素提供了5种额外的样式，用于控制tr的背景颜色。样式如表3-2所示。

表 3-2　行级元素样式

样　式	描　述	样　式	描　述
.active	表示当前活动的信息	.warning	表示警告，需要特别注意
.success	表示成功或者正确的行为	.danger	表示危险或者可能是错误的行为
.info	表示中立的信息或行为		

其使用方式非常简单，在tr元素（或者td元素）上应用相应的class样式即可。示例如下：

```
<tr class=“active”>
    <td>1</td>
    <td>JavaScript编程精解</td>
    <td>机械工业出版社</td>
    <td>汤姆大叔(译)</td>
</tr>
```

运行上述示例后的效果如图3-16所示。

1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）
4	Backbone应用开发实战	人民邮电出版社	汤姆大叔（译）
5	深入理解Bootstrap3	机械工业出版社	汤姆大叔（著）

图3-16　行级元素运行效果

除了active样式以外，其他4个样式在和.table-hover样式一起用的时候，Bootstrap也相应地设置了鼠标悬停高亮的颜色，所以如果需要添加额外颜色的tr行级样式，也需要注意这个设置。示例源码如下：

```
// 源码1563行
.table > thead > tr > td.danger,  .table > tbody > tr > td.danger,  .tab
```

```
    tfoot > tr > td.danger,
/* 此处省略了一些选择符*/
.table > thead > tr.danger > td,  .table > tbody > tr.danger > td,  .tab
    tfoot > tr.danger > td {
  background-color: #f2dede;            /* danger样式的tr背景色*/
}
.table-hover > tbody > tr > td.danger:hover,
/* 此处省略了一些选择符*/
.table-hover > tbody > tr.danger:hover > td {
   background-color: #ebcccc; /* table-hover和danger一起使用时,鼠标悬停时的
tr背景色*/
}
```

3.4.7　响应式表格

随着响应式设计的大量需求，Bootstrap为表格也提供了一个响应式设计的容器（.table-responsive样式），将.table-responsive样式包装在.table样式外部即可创建响应式表格，其会在小屏幕设备上（小于768像素）水平滚动。而当屏幕大于768像素宽度时，水平滚动条消失。示例如下：

```
<div class="table-responsive">
    <table class="table">
        …
    </table>
</div>
```

运行效果如图3-17所示。

响应式表格的实现原理是利用了CSS的媒体查询特性，在小于768像素的设备上应用@media(max-width:　767px)里的样式。该样式主要有3个方面的设置：

❑对.table-responsive容器边框以及滚动条进行了设置。

❑把原有.table样式的底部外边距margin-bottom从20像素改为了0像素，其目的是消除滚动条带来的上下高度差，并在.table-responsive样式上又设置了一个margin-bottom: 15px，以避免和容器外部的下一个元素重叠。

#	图书名称	出版社	作译者	出版时间	备注
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）	2012	
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）	2013	设计模式经典
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）	2013	
4	深入理解Bootstrap	机械工业出版社	汤姆大叔	2014	你正在阅读的

图3-17　响应式表格的运行效果

❑将所有单元格的文本设置成不自动换行，以保留原有样式。

实现响应式的详细源码如下：

```
// 源码1583行
@media (max-width: 767px) {
```

```
  .table-responsive {
    width: 100%;
    margin-bottom: 15px;                    /* 设置底部外边距，避免重叠 */
    overflow-x: scroll;          /* 超出范围，水平可滚动 */
    overflow-y: hidden;
    -webkit-overflow-scrolling: touch;
    -ms-overflow-style: -ms-autohiding-scrollbar;
    border: 1px solid #ddd;      /* 设置1像素宽的边框 */
  }
  .table-responsive > .table {    margin-bottom: 0;  }
  .table-responsive > .table > thead > tr > th,
  /* 省略部分样式 */
  .table-responsive > .table > tfoot > tr > td {
white-space:  nowrap;                    /*  确保单元格中的文本不会换行，直到遇到 <br> 标签为止*/
  }
  .table-responsive > .table-bordered {    border: 0;  }
```

通过上述代码，可以看到.table-responsive给自己加了一个1px的外边框。如果在.table上再使用.table-bordered样式的话，就会和表格的外边框重合了，变粗了，可能会想象成如图3-18所示的效果。

#	图书名称	出版社	作译者	出版时间	备注
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）	2012	
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）	2013	设计
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）	2013	
4	深入理解Bootstrap	机械工业出版社	汤姆大叔	2014	你正

图3-18　响应式样式和table-bordered样式叠加的运行效果

聪明的Bootstrap开发团队怎么会犯这样低级的错误呢？所以他们在该响应式样式的内部，针对.table-bordered样式又进行了去边框设置。源码如下：

```
// 源码1604行
  .table-responsive > .table-bordered {
    border: 0; /*将整个表格的外边框设置为0像素*/
  }
  .table-responsive > .table-bordered > tbody > tr > th:first-child,
  /*此处省略一些选择符*/
  .table-responsive > .table-bordered > tbody > tr > td:first-child,
  .table-responsive > .table-bordered > tfoot > tr > td:first-child {
```

```
    border-left: 0; /*将所有tr的第一个单元格(即最左边的一列)的左边框都置为0像素
*/
  }
  .table-responsive > .table-bordered > tbody > tr > th:last-child,
  /*此处省略一些选择符*/
  .table-responsive > .table-bordered > tbody > tr > td:last-child,
  .table-responsive > .table-bordered > tfoot > tr > td:last-child {
    border-right: 0; /*将所有tr的最后一个单元格(即最右边的一列)的右边框都设置为0
像素*/
  }
  .table-responsive > .table-bordered > tbody > tr:last-child > th,
  /*此处省略一些选择符*/
  .table-responsive > .table-bordered > tfoot > tr:last-child > td {
    border-bottom: 0; /*将最后一行tr里的单元格的底部边框设置为0像素*/
  }
```

运行效果如图3-19所示。

#	图书名称	出版社	作译者	出版时间	备注
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）	2012	
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）	2013	设计
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）	2013	
4	深入理解Bootstrap	机械工业出版社	汤姆大叔	2014	你正

图3-19　去边框处理后的运行效果

3.5　表单

源码文件：form.less

CSS文件：bootstrap.css 1630行开始

表单（Form）是HTML网页交互中最重要的部分，同时也是Bootstrap框架中的核心内容，表单提供了丰富的样式（基础、内联、横向）。结合各种各样的表单控件，利用各种表单控件不同的状态、大小、分组，可以组合出绚丽多彩的表单。下面详细介绍表单的各种功能。

3.5.1 基础表单

Bootstrap对基础表单未做太多的定制化效果设计，默认都使用全局设置，只是对表单内的fieldset、legend、label标签进行了设定，将这些元素的margin、padding、border等进行了细化设置。详细请参考源码1854行以后的代码。

如果在select、input、textarea元素上应用了.form-control样式，显示的宽度会变成100%，并且placeholder的颜色都设置成了#999999。主要源码如下：

```
// 源码1689行
.form-control {
  display: block;
  width: 100%;                              /* 设置宽度是100% */
  /* 省略部分设置 */
  border: 1px solid #ccc;                   /* 边框设置 */
  border-radius: 4px;                       /* 圆角设置*/
  -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
          box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
   -webkit-transition: border-color ease-in-out .15s, box-shadow ease-
in-out .15s;
           transition: border-color ease-in-out .15s, box-shadow ease-
in-out .15s;
}
.form-control:focus {
  border-color: #66afe9;                          /* 作用域得到焦点时的边框颜
色*/
  outline: 0;
                                                                -webkit-box-
shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102, 175,
      233, .6);
                                                                    box-
shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102, 175,
              233, .6);
}
.form-control:-moz-placeholder {  color: #999;
                                                    /* placeholder的文本颜
色:moz浏览器*/}
.form-control::-moz-placeholder {  color: #999;
                                          /* placeholder的文本颜色:moz浏览器
*/  opacity: 1;}
.form-control:-ms-input-placeholder {  color: #999;
                                                    /* placeholder的文本颜
色:IE浏览器*/}
.form-control::-webkit-input-placeholder {  color: #999;
                                                    /* placeholder的文本颜
色:webkit浏览器*/}
使用方式如下：
    <form>
       <fieldset>
         <legend>用户登录</legend>
```

```
            <div class="form-group">
                <label>登录账户</label>
                 <input type="email" class="form-control" placeholder="请输
入你
                    的用户名或Email">
            </div>
            <div class="form-group">
                <label>密码</label>
                 <input type="text" class="form-control" placeholder="请输入
你的密码">
            </div>
            <div class="checkbox">
                <label><input type="checkbox">记住密码</label>
            </div>
            <button type="submit" class="btn btn-default">登录</button>
        </fieldset>
    </form>
```

运行效果如图3-20所示。

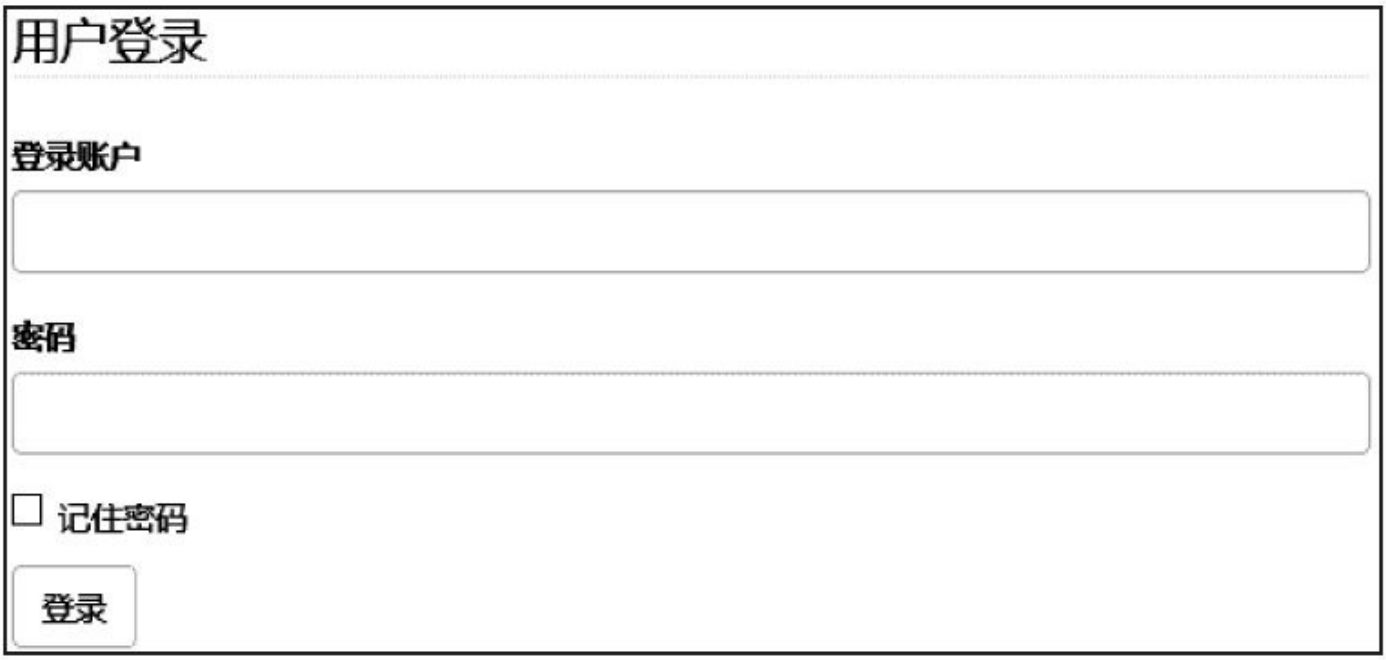


图3-20　普通表单运行效果

注意

在上例中，提示语label和input元素放在一个样式为.form-group的div里了。.form-group样式提供了一个margin-bottom:15px的底部外边距，所以可以很清晰地看到每一组控件。

3.5.2　内联表单

有的时候，我们可能需要一个所有控件都在一行中的表单，比如图3-21所示的登录。要实现这种内联样式效果，只需要在普通的form元素上应用一个.form-inline样式，即可将表单内的元素设置为内联样式。

图3-21　内联表单运行效果

水平线上的唯一因素就是要控制元素的显示方式为display: inline-block，所以只需要为相应的子元素设置display属性即可。但需要注意的是，该.form-inline样式只能在大于768像素的浏览器上才能应用。源码如下：

```
// 源码1927行
@media (min-width: 768px) {              /* 大于768像素的浏览器才生效*/
  .form-inline .form-group {
    display: inline-block;               /* 内联样式显示*/
    margin-bottom: 0;
    vertical-align: middle;
  }
  .form-inline .form-control {
    display: inline-block;
                 /* 内联样式显示,但由于form-control样式设置了100%的宽度,所以没什么用*/
    width: auto;
    vertical-align: middle;
  }
  .form-inline .radio,
  .form-inline .checkbox {
    display: inline-block;
    padding-left: 0;
    margin-top: 0;  /* 确保上下居中*/
    margin-bottom: 0;
    vertical-align: middle;
  }
  .form-inline .radio input[type=“radio”],
  .form-inline .checkbox input[type=“checkbox”] {
    float: none;  /* 不使用浮动定位*/
    margin-left: 0;
  }
  .form-inline .has-feedback .form-control-feedback {    top: 0;  }
}
```

要注意，由于默认的样式为.form-control，且其input、select和textarea的宽度都是100%，所以在使用内联表单的时候是无效的，需要对这些控件元素单独设置宽度width，或者外面再加上一层带有.form-group样式的

div元素。示例如下：

```
    <form class="form-inline">
        <div class="form-group">
             <input type="text" class="form-control" placeholder="请输入
你的用户名">
        </div>
        <div class="form-group">
             <input type="text" class="form-control" placeholder="请输入
你的密码">
        </div>
        <div class="checkbox">
            <label><input type="checkbox">记住密码</label>
        </div>
        <button type="submit" class="btn btn-default">登录</button>
    </form>
```

但这种情况下，如果再设置一个label的话，input又换行了。所以如果非要label的话，那就只能在input所在div元素的上边再加一个div元素用于包含label标签。比如：

```
    <div class="form-group">
        <label>用户名:</label>
    </div>
    <div class="form-group">
         <input type="text" class="form-control" placeholder="请输入你的用
户名">
    </div>
```

注意

如果没有为每个输入控件设置label，屏幕阅读器将无法正确识别。对于这种内联表单，可以通过为label设置.sr-only样式将其隐藏。比如：

```
<div class="form-group">
   <label class="sr-only" for="account">登录用户名</label>
    <input type="text" class="form-control" id="account" placeholder="请
输入你的用户名">
</div>
```

3.5.3 横向表单

横向表单与内联表单的使用方式不太一样，其不能在form元素上简单应用一个.form-horizontal样式，这是因为.form-horizontal样式本身没有做什么特殊的设置，只简单设置了一下相关的padding和margin值。源码如下：

```
// 源码1959行
.form-horizontal .control-label,
.form-horizontal .radio,
.form-horizontal .checkbox,
.form-horizontal .radio-inline,
.form-horizontal  .checkbox-inline  {          /*  简单设置了一下padding和
margin */
  padding-top: 7px;
  margin-top: 0;
  margin-bottom: 0;
}
.form-horizontal .radio,.form-horizontal .checkbox {  min-height: 27px;
                                                    /* 设置最小高度 */}
.form-horizontal .form-group {          /* 简单设置了一下margin */
  margin-right: -15px;  margin-left: -15px;
}
.form-horizontal .form-control-static {  padding-top: 7px; }
  @media (min-width: 768px) {
  .form-horizontal .control-label {    text-align: right;
                                                  /* 大屏幕下,label可以居右
显示 */  }
}
```

运行效果如图3-22所示。

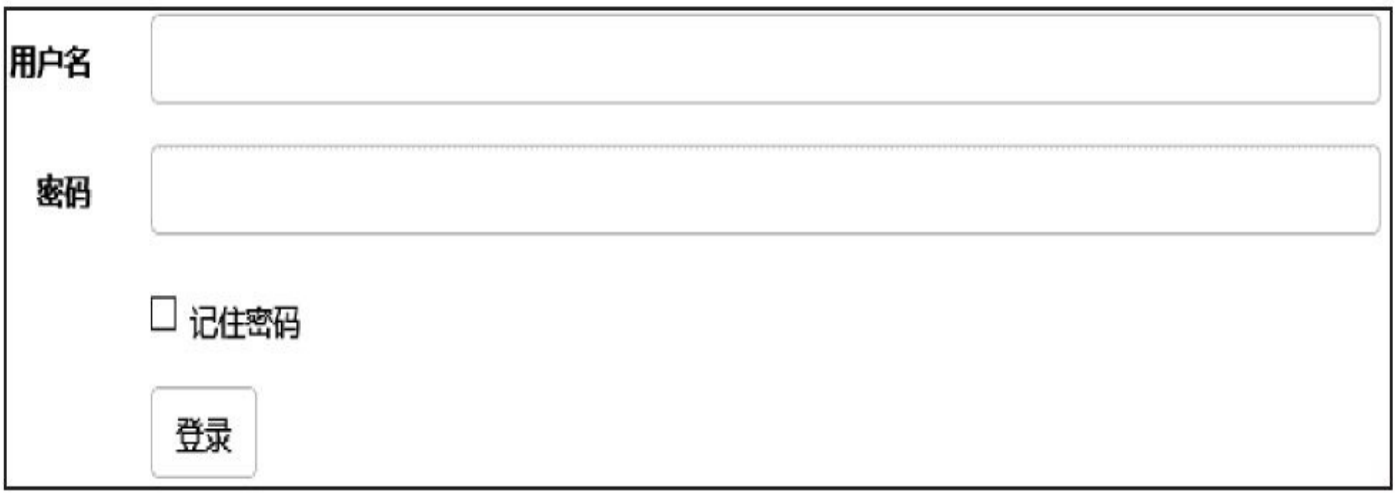


图3-22　横向表单运行效果

所以，要实现横向表单，不仅要应用上述样式，还要使用Bootstrap预置的栅格类，以便将label和控件水平并排布局。由于.form-horizontal样式改变了.form-group的行为，将其表现得像栅格系统中的行（row）一样，因此就无需再使用.row样式了。示例如下：

```
<form class="form-horizontal" role="form">
    <div class="form-group">
            <label  for="account"  class="col-sm-2  control-label">用户名
</label>
        <div class="col-sm-10">
          <input type="email" class="form-control" id=" account "
              placeholder="请输入你的用户名">
        </div>
    </div>
    <div class="form-group">
           <label  for="  password  "  class="col-sm-2  control-label">密码
</label>
        <div class="col-sm-10">
          <input type="password" class="form-control" id="password"
              placeholder="请输入你的密码">
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
          <div class="checkbox"><label><input type="checkbox">记住密码</
              label></div>
        </div>
    </div>
    <div class="form-group">
        <div class="col-sm-offset-2 col-sm-10">
            <button type="submit" class="btn btn-default">登录</button>
        </div>
    </div>
</form>
```

3.5.4 表单控件

在默认的Bootstrap源码里，对input、select、testarea都有良好的支持，尤其是对现有HTML5语法下的input都进行了支持（如type为text、password、datetime、datetime-local、date、month、time、week、number、email、url、search、tel和color的元素）。

1.input元素

在使用input元素的时候，必须声明type类型，否则可能会引起其他问题，因为Bootstrap在定义这些样式的时候都指定了type类型，比如，input[type=“text”]。

```
<input type=“text” placeholder=“文本输入框”>
```

2.select元素

下拉列表select元素的使用方式和原始的一致，多行选择设置mulitiple属性为multiple即可。Bootstrap会为这些默认的元素提供统一风格的显示。示例如下：

```
<select>
    <option>1</option>
</select>
<select multiple=“multiple”>
    <option>1</option>
    <option>2</option>
</select>
```

3.textarea元素

同样，在textarea元素里定义了rows数字即可定义大文本框的高度，定义cols可以定义大文本框的宽度。但是如果在该框架上应用了.form-control样式，则cols属性不起作用，因为.form-control样式的表单控件都设置了100%的宽度（或auto）。所以大家在使用时，一旦设置了该样式，就不需要再设置cols属性了。如：

```
<textarea class=“form-control” rows=“3”></textarea>
```

4.checkbox和radio

checkbox和radio是input元素里两个非常特殊的type，通常在使用的过程中和label文字搭配，但通常会出现左右边距对不齐的问题。为此，Bootstrap进行了标准设置，开发人员在使用的过程中遵循如下方式即可：

```
<div class=“checkbox”>
    <label><input type=“checkbox” value=””>是否想赚大钱？</label>
</div>
```

```
<div class=“radio”>
                                                                    <label>
<input type=“radio” name=“optionsRadios” value=“female” checked>请
        确保,您喜欢女人？</label>
</div>
<div class=“radio”>
    <label><input type=“radio” name=“optionsRadios” value=“male”>请确保,
您喜欢
        男人？</label>
</div>
```

即使用的时候，每个input外部都要用label包住，并且在最外层用容器元素包住，并应用相应的.checkbox和.radio样式。主要源码如下：

```
// 源码1741行
.radio,
.checkbox {             /* 让单选框和复选框都能左右和上下居中 */
  display: block;
  min-height: 20px;
  padding-left: 20px;
  margin-top: 10px;
  margin-bottom: 10px;
}
.radio label,
.checkbox label {               /* 内部有label的话,内联显示 */
  display: inline;
  font-weight: normal;
  cursor: pointer;
}
```

有些checkbox或者radio元素中文本很少，可能需要横向显示，为此Bootstrap也提供了相应的内联样式.checkbox-inline和.radio-inline，效果如图3-23所示。

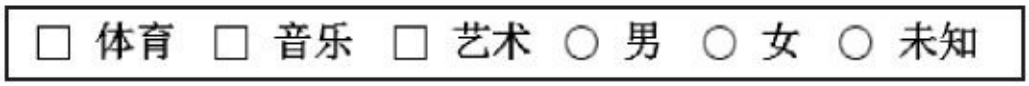


图3-23　checkbox和radio元素运行效果

使用方式如下：

```
<label class=“checkbox-inline”>
    <input type=“checkbox” value=”体育” id=“inlineCheckbox1”>体育
</label>
<label class=“checkbox-inline”>
    <input type=“checkbox” value=”音乐” id=“inlineCheckbox2”>音乐
</label>
…
<label class=“radio-inline”>
    <input type=“radio” value=”未知” id=“Radio3”>未知
</label>
```

实现源码如下：

```
// 源码1766行
```

```
.radio-inline,
.checkbox-inline {   /* 在其他元素上设置*-inline样式的话,也可以让多个控件水平在一行中显示 */
  display: inline-block;
  padding-left: 20px;
  margin-bottom: 0;
  font-weight: normal;
  vertical-align: middle;  /* 垂直居中 */
  cursor: pointer;
}
.radio-inline + .radio-inline,
.checkbox-inline + .checkbox-inline {
  margin-top: 0;
  margin-left: 10px;  /* 每个选项间距10px */
}
```

3.5.5　控件状态

每个控件（尤其是input）在使用的过程中，可能都会有很多种状态，比如输入内容的时候有焦点提示，输错的时候有出错提示等。Bootstrap提供了3种状态的样式可供使用，分别是：焦点状态、禁用状态、验证提示状态。

1.焦点状态

焦点状态的实现方式是在选择器: focus上删除默认的outline样式，重新应用一个新的box-shadow样式，从而实现焦点状态下，input出现柔和的阴影边框（注意，该效果必须使用.form-control样式才行）。运行效果如图3-24所示。

图3-24　焦点状态时的运行效果

该效果实现源码如下：

```
// 源码1706行
.form-control:focus {
  border-color: #66afe9;
  /* 作用域得到焦点时的边框颜色*/
  outline: 0;
                                                          -webkit-box-
shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102, 175, 233, .6
                                                              box-
shadow: inset 0 1px 1px rgba(0,0,0,.075), 0 0 8px rgba(102, 175,
              233, .6);
}
```

从图3-24中可以看出，radio和checkbox的焦点效果和普通input的不太一样。Bootstrap对file、radio、checkbox的焦点效果做了一些特殊处理，以便更圆形化。源码如下：

```
// 源码1675行
input[type=“file”]:focus,
input[type=“radio”]:focus,
input[type=“checkbox”]:focus {
  outline: thin dotted;
  outline: 5px auto -webkit-focus-ring-color;
  outline-offset: -2px;
}
```

2.禁用状态

禁用状态的实现方式主要是完善默认disabled状态的显示状态，使用方式和普通的disabled一样，只需要在禁用元素上使用disabled属性即可。使用方法如下：

```
<input type="text" placeholder="Disabled input here…" disabled>
```

其源码主要是设置了不准输入的鼠标样式和背景颜色（灰色）。注意，只有带.form-control样式的控件才会更改背景色。请看下面的源码：

```
// 源码1725行
.form-control[disabled],
.form-control[readonly],
fieldset[disabled] .form-control {
  /* 若form-control控件或fieldset元素被禁用，显示不允许输入手形图标*/
  cursor: not-allowed;
  background-color: #eee;
  opacity: 1;
}
```

而如果不使用上述样式，直接在普通的元素上使用disabled属性，则只会显示一个不能输入的手形图标。源码如下：

```
// 源码1780行
input[type="radio"][disabled],
input[type="checkbox"][disabled],
.radio[disabled],
.radio-inline[disabled],
.checkbox[disabled],
.checkbox-inline[disabled],
fieldset[disabled] input[type="radio"],
fieldset[disabled] input[type="checkbox"],
fieldset[disabled] .radio,
fieldset[disabled] .radio-inline,
fieldset[disabled] .checkbox,
fieldset[disabled] .checkbox-inline {
  cursor: not-allowed;
}
```

从上述源码可以看出，fieldset如果使用了disabled属性，则fieldset内部的input、select、textarea或应用了.form-control样式的其他控件也将为禁用状态。让我们来验证一下，示例如下：

```
<fieldset disabled>
     <legend><input></legend>
     <input type="text" placeholder="Disabled input">
     <select>
        <option>不可选择</option>
     </select>
     <div class="checkbox">
        <label><input type="checkbox">记住密码</label>
     </div>
```

```
        <button type="submit" class="btn btn-primary">提交</button>
    </fieldset>
```

运行效果如图3-25所示。我们发现大部分都被禁用了，但是legend内的input却没有被禁用，依然可以输入文本，这是因为这是HTML5的一个限制，即fieldset内的第一个legend元素不受disabled的影响。Bootstrap为了遵守这一规则（其实实现起来也很复杂，要用JavaScript代码进行实现）就继续保持原状了，大家使用过程中要特别注意。

图3-25　禁用状态时的运行效果

注意

由于IE不支持fieldset下的disabled属性（比如，IE10不支持input和button），所以IE下的开发者需要用JavaScript代码再进行特殊处理。

fieldset上应用disabled时，fieldset内部的a标签在所有浏览器下的单击行为都不能被禁用（比如：<a　class="btn　btn-default">），需要使用JavaScript代码再进行特殊处理。

3.验证提示状态

在填写表单的时候，经常要提示用户其输入内容是否合法，长度是否够用，再次输入的密码是否和第一次输入的密码一致，或者输入的用户名存在还是不存在等问题，不同的提示可能需要不同的提示状态（如，颜色、边框、提示语等）。Bootstrap提供了.has-warning、.has-error、.has-success三种样式用于分别表示警告（黄色）、错误（红色）、成功（绿色）语境的内容。代码如下：

```
    <div class="form-group has-warning">
        <label class="control-label" for="inputWarning">输入长度不够！
    </label>
        <input type="text" class="form-control" id="Text1">
    </div>
    <div class="form-group has-error">
        <label class="control-label" for="inputError">输入不符合要求！
    </label>
        <input type="text" class="form-control" id="Text2">
    </div>
    <div class="form-group has-success">
        <label class="control-label" for="inputSuccess">输入文本符合要求！
    </label>
```

```
        <input type="text" class="form-control" id="Text3">
    </div>
```

从示例代码可以看到，使用了form-group、control-label、form-control标签进行分组（关于控件分组，会在后续章节进行解释），在form-group平级的div元素上应用has-***样式，然后在input元素上应用form-control样式，实现效果如图3-26所示。

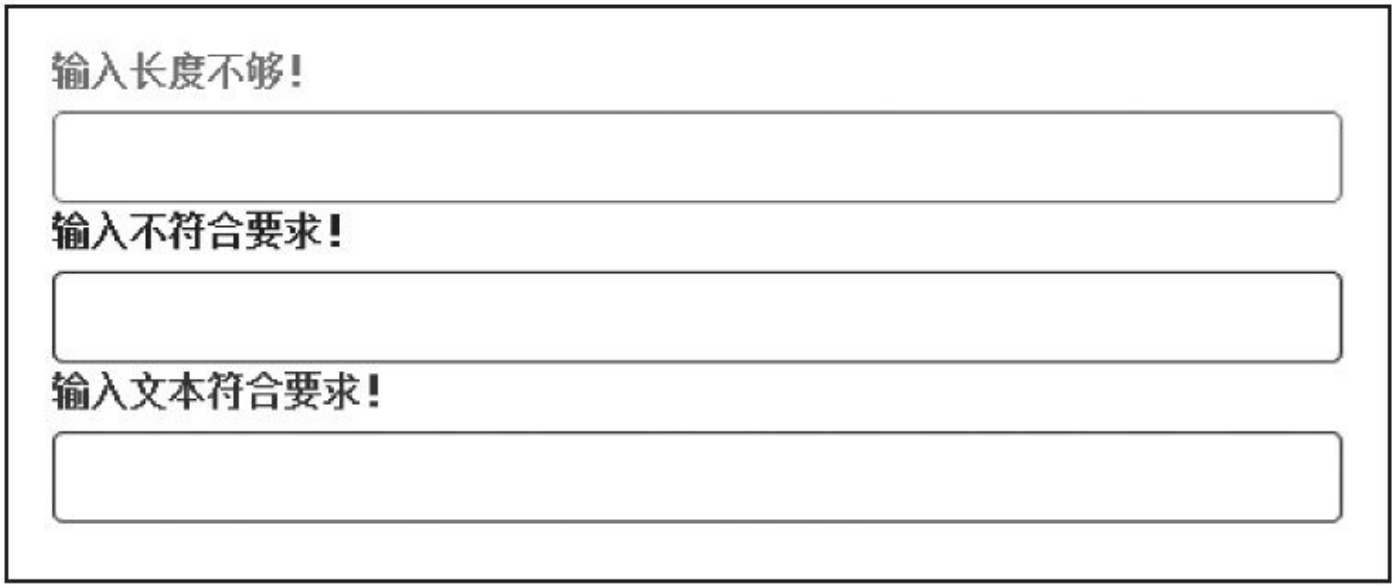


图3-26　验证提示的运行效果

has-error和has-success的实现方式相同，只是对文字、边框和阴影设置的颜色不同。has-warning的实现源码如下：

```
// 源码1866行
.has-warning .help-block,
.has-warning .control-label,
.has-warning .radio,
.has-warning .checkbox,
.has-warning .radio-inline,
.has-warning .checkbox-inline { /* has-warning容器内部的控件文本颜色统一设置
*/
  color: #8a6d3b;       /* 不使用control-label,使用其他的(如help-block)也可
以*/
}
.has-warning .form-control {
  /* 为has-warning容器内部的form-control控件设置边框和阴影效果*/
  border-color: #8a6d3b;
  -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
          box-shadow: inset 0 1px 1px rgba(0, 0, 0, .075);
}
.has-warning .form-control:focus {
  /* has-warning容器内部的控件在得到焦点时的效果颜色更深*/
  border-color: #66512c;
                                                          -webkit-box-
shadow: inset 0 1px 1px rgba(0, 0, 0, .075), 0 0 6px #c0a16b;
                                                              box-
shadow: inset 0 1px 1px rgba(0, 0, 0, .075), 0 0 6px #c0a16b;
}
```

```
.has-warning .input-group-addon {/* addon的背景色和字体颜色也要同步设置*/
  color: #8a6d3b;
  background-color: #fcf8e3;
  border-color: #8a6d3b;
}
```

注意

上述.input-group-addon样式的效果会在4.5节进行讲述。

有的时候，我们在验证状态时提供所对应状态的小图标，以便能够直观地显示，实现效果如图3-27所示。

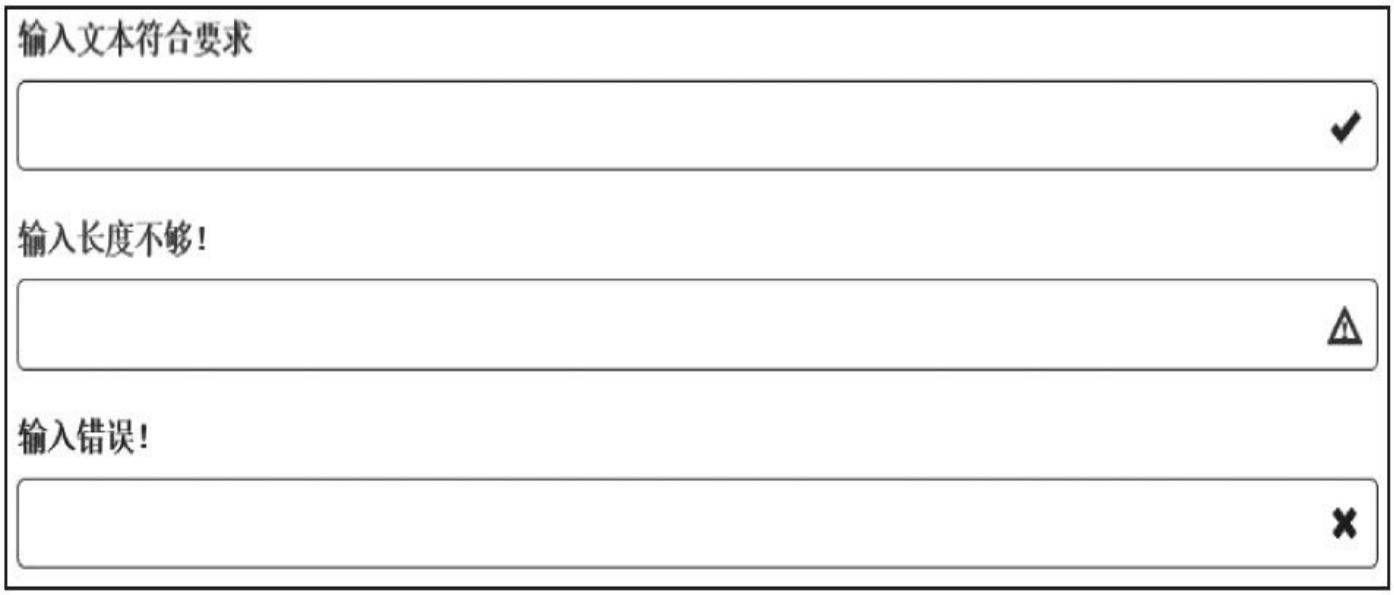


图3-27　小图标提示的运行效果

从运行效果可以看到，小图标肯定是固定在输入框的右边的。要实现这种效果，通常要解决两个问题：首先设置输入框父元素的定位方式为相对定位，然后设置这种小图标的定位方式为绝对定位（并设置right值）。

Bootstrap针对该功能提供了特殊的feedback样式，用于实现该效果。其用法如下：

```
<div class="form-group has-success has-feedback">
                                        <label          class="control-
label" for="inputSuccess2">Input with success</label>
    <input type="text" class="form-control" id="inputSuccess2" />
    <span class="glyphicon glyphicon-ok form-control-feedback"></span>
</div>
```

该效果运用了两个样式：父容器上的has-feedback样式用于设置定位方式；小图标元素上的form-control-feedback样式用于设置图标的显示大小等。两个样式的源码正好符合我们前面所说的要求，具体源码如下：

```
// 源码1824行
.has-feedback {  position: relative;     /* 相对定位,用于设置input元素的父容
```

```
器的定位方式*/}
.has-feedback .form-control {  padding-right: 42.5px;
                                                    /* 右内边距的设置,以便可以
显示小图标*/ }
.has-feedback .form-control-feedback {  /* 设置小图标的显示方式*/
  position: absolute;/* 绝对定位*/
  top: 25px;
  right: 0; /* 右对齐*/
  display: block;
  width: 34px;
  height: 34px;
  line-height: 34px;
  text-align: center;
}
.has-success .form-control-feedback {  color: #3c763d; /* 验证通过时图标的
显示颜色 */}
.has-warning .form-control-feedback {  color: #8a6d3b; /* 验证警告时图标的
显示颜色 */}
.has-error .form-control-feedback {  color: #a94442; /* 验证错误时图标的显
示颜色 */}
```

上述源码的最后3行分别设置了不同验证状态下的小图标的颜色。

3.5.6　控件大小

Bootstrap提供了两个样式用于设置稍大或者稍小的input输入框，分别是：.input-lg和.input-sm。使用方式如下：

```
<input class="input-lg form-control" type="text" placeholder="较大">
<input class="form-control" type="text" placeholder="正常大小">
<input class="input-sm form-control" type="text" placeholder="较小">
```

运行上述样式后的效果如图3-28所示。

图3-28　3种大小输入框的运行效果

控件大小的实现思路是给input输入框设定不同大小的padding、font-size、border-radius值。从如下源码也可以看出，.input-lg和.input-sm样式不仅适用于input，也适用于select和textarea元素。

```
/* 源码1794 行*/
.input-sm {                    /* 设置小的输入框input*/
  height: 30px;
  padding: 5px 10px;
  font-size: 12px;
  line-height: 1.5;
  border-radius: 3px;
}
select.input-sm {                    /* 设置小的选择框select*/
  height: 30px;
  line-height: 30px;
}
textarea.input-sm,
select[multiple].input-sm {     /* 设置小的文本框textarea*/
  height: auto;
}
.input-lg {                    /* 设置大的输入框input*/
    height:  46px;    padding:  10px  16px;    font-size:  18px;    line-
height: 1.33;
      border-radius: 6px;
```

```
}
select.input-lg {  height: 46px;  line-height: 46px;    /* 设置大的选择框
select*/}
textarea.input-lg,select[multiple].input-lg {  height: auto;    /* 设置
大的文本框
    textarea*/}
```

注意

根据上述源码，也可以按照类似规则，为上述3种元素定义其他大小的样式，比如，.input-mini。

再仔细看一下上述代码，发现上述代码在设置大小的时候只设置了高度，并没有设置宽度。所以，在实际设计过程中，对于不同尺寸的元素还要进行相应宽度的调整。好在这些元素默认都是100%显示，而且还有栅格系统提供的支持。所以对于控制宽度，可以按照如下方式设置：

```
    <div class=“row”>
        <div class=“col-xs-2”>
             <input type=“text” class=“form-control” placeholder=”.col-
xs-2”>
        </div>
        <div class=“col-xs-3”>
             <input type=“text” class=“form-control” placeholder=”.col-
xs-3”>
        </div>
        <div class=“col-xs-4”>
             <input type=“text” class=“form-control” placeholder=”.col-
xs-4”>
        </div>
    </div>
```

3.5.7 其他

Bootstrap还提供了一个help-block样式，用于在表单中显示块级的帮助提示。使用方式示例如下：

```
<span class="help-block">自己独占一行或多行的块级帮助文本。</span>
```

help-block样式的源码如下所示：

```
// 源码1921行
.help-block {
  display: block;
  margin-top: 5px;
  margin-bottom: 10px;
  color: #737373;
}
```

其实在v2.x版本里还提供了一个help-inline样式，用于将输入控件与帮助文本显示在一行。但不知道为什么，在v3.x系列中去掉了。如果想恢复这个功能，将如下代码添加到自己的CSS文件即可（使用方式和help-block一样）。

```
.help-inline {
  display: inline-block;
  padding-left: 5px;
  color: #737373;
}
```

3.6 按钮

源码文件：buttons.less

CSS文件：bootstrap.css 1988行开始

按钮是任何系统都不能缺少的组件，不同的系统需要的按钮多种多样，按钮的设置涉及按钮的大小、颜色、状态等。本节将对这些问题一一讲解。

3.6.1 按钮样式

按钮是网页交互过程中不可缺少的一部分，Bootstrap默认提供了7种样式的按钮风格，其效果如图3-29所示。

图3-29 按钮样式运行效果

使用方式如下：

```
<!— 标准button —>
<button type=“button” class=“btn btn-default”>Default</button>
<!— 提供视觉加重，表示在一组button中，该按钮是最主要的button —>
<button type=“button” class=“btn btn-primary”>Primary</button>
<!— 表示成功或正使用的button —>
<button type=“button” class=“btn btn-success”>Success</button>
<!— 表示提示信息的button —>
<button type=“button” class=“btn btn-info”>Info</button>
<!— 表示需要进行某种行为的button —>
<button type=“button” class=“btn btn-warning”>Warning</button>
<!— 表示危险或错误行为的button —>
<button type=“button” class=“btn btn-danger”>Danger</button>
<!— 让button的行为看起来像链接一样 —>
<button type=“button” class=“btn btn-link”>Link</button>
```

和第2章里的CSS设计思想讲述的一样，按钮风格首先定义了基础的.btn样式以及相关的hover、focus、active等行为特效，然后再为特殊的风格（如，btn-default）定义特殊的颜色（如，各种变化状态下的颜色）。.btn基础样式部分代码如下：

```
// 源码1988行
.btn {
  display: inline-block;                          /* inline模式*/
  padding: 6px 12px;
  margin-bottom: 0;
  font-size: 14px;
  font-weight: normal;
  line-height: 1.428571429;
  text-align: center;
  white-space: nowrap;
  vertical-align: middle;
  cursor: pointer;                          /* 手形图标 */
  -webkit-user-select: none;
     -moz-user-select: none;
      -ms-user-select: none;
       -o-user-select: none;
          user-select: none;
  background-image: none;
  border: 1px solid transparent;            /* 边框 */
  border-radius: 4px;                       /* 圆角*/
```

```
}
```

另外，基础样式也会对按钮默认的focus、active、hover、disabled行为定义相关的变化。详见源码2008～2034行。

特殊风格相关代码（以btn-default为例）如下：

```
// 源码2035行
.btn-default { /* 定义default风格相关的颜色 */
  color: #333;
  background-color: #fff;
  border-color: #ccc;
}
.btn-default:hover,
.btn-default:focus,
.btn-default:active,
.btn-default.active,
.open  .dropdown-toggle.btn-default  {   /*  hover、focus、active变化时的颜色 */
  color: #333;
  background-color: #ebebeb;
  border-color: #adadad;
}
.btn-default:active,
.btn-default.active,
.open .dropdown-toggle.btn-default {  /* active状态下,背景图设置为none */
  background-image: none;
}
.btn-default.disabled,
.btn-default[disabled],
/* 此处省略部分选择符 */
.btn-default[disabled].active,
fieldset[disabled] .btn-default.active {  /* 禁用状态下,各种变化时的颜色 */
  background-color: #fff;
  border-color: #ccc;
}
.btn-default .badge {  /* 按钮内部有徽章的话,设置徽章的显示颜色和背景色 */
  color: #fff;
  background-color: #333;
}
```

同理，其他6种按钮样式也是按照上述规则进行定义的，只是颜色变化不同而已。所以如果需要定义自己的风格，遵循上述规则即可。

注意

关于在按钮上应用不同主题的介绍，请参考第5章5.9节的内容。

3.6.2　按钮大小

像input一样，Bootstrap也提供了控制button按钮大小的CSS样式。在2.x系列时提供了3种样式：.btn-large、.btn-small、btn-mini。当前最新版本将这些样式改名为.btn-lg、.btn-sm、.btn-xs。这些样式可以和btn颜色样式组合使用。示例如下：

```
<p>
          <button   type=“button”   class=“btn   btn-default   btn-lg”>大型
button</button>
          <button   type=“button”   class=“btn   btn-primary   btn-lg”>大型
button</button>
</p>
<p>
    <button type=“button” class=“btn btn-success”>正常button</button>
    <button type=“button” class=“btn btn-info”>正常button</button>
</p>
<p>
          <button   type=“button”   class=“btn   btn-success   btn-sm”>小型
button</button>
           <button   type=“button”   class=“btn   btn-info   btn-sm”>小型
button</button>
</p>
<p>
          <button   type=“button”   class=“btn   btn-warning   btn-xs”>超小
button</button>
           <button   type=“button”   class=“btn   btn-danger   btn-xs”>超小
button</button>
</p>
```

运行上述示例后的效果如图3-30所示。

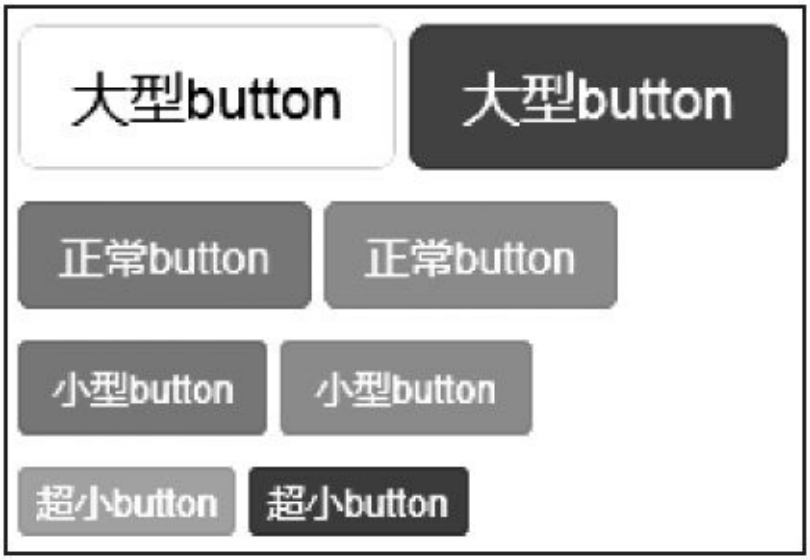


图3-30　不同尺寸和不同颜色按钮的运行效果

这些不同尺寸按钮的主要差别是padding、font-size、line-height和border-radius，具体源码里的数值如下：

```
/* 源码 2314 行 */
.btn-lg { /* 大按钮*/  padding: 10px 16px;  font-size: 18px;  line-
height: 1.33;
    border-radius: 6px; }
.btn-sm { /* 小按钮*/  padding: 5px 10px;  font-size: 12px;  line-
height: 1.5;
    border-radius: 3px; }
.btn-xs {/* 超小按钮*/ padding: 1px 5px;  font-size: 12px;  line-
height: 1.5;
    border-radius: 3px; }
```

讲述到这里，我们发现一个问题，所有按钮的宽度都是根据文本的长短（再加上padding值）来决定的，而如果我们需要一个充满父容器的100%宽度的按钮，则无法实现。好在Bootstrap又单独为我们提供了一个block级的样式——.btn-block按钮，它不以文本多少自动伸缩，它没有padding和margin值，而是充满父容器。具体源码实现如下：

```
// 源码2332行
.btn-block {  /* 取消padding*/
  display: block;
  width: 100%;
  padding-right: 0;
  padding-left: 0;
}
.btn-block + .btn-block {
  margin-top: 5px;  /* 多个按钮之间取消上下间隔*/
}
input[type=“submit”].btn-block,
input[type=“reset”].btn-block,
input[type=“button”].btn-block {
  width: 100%;  /* 各种input按钮的宽度也要充满父容器*/
}
```

.btn-block样式和普通按钮的区别如图3-31所示。可以看出块级元素充满父容器，而不随字符宽度变化而变化。

图3-31　.btn-block按钮的运行效果

3.6.3　多标签支持

btn相关元素的强大之处在于，它不仅能够支持普通的button元素，也能够支持a元素和input元素，这些元素应用btn样式也能够产生同样的显示效果。用法如下：

```
<a class=“btn btn-default” href=”#”>链接</a>
<button class=“btn btn-default” type=“submit”>按钮</button>
<input class=“btn btn-default” type=“button” value=”输入框”>
<input class=“btn btn-default” type=“submit” value=”提交”>
```

但是，强烈建议不管在任何情况下，都使用button元素，以免出现跨浏览器兼容性问题。比如，Firefox浏览器下就有一个bug，导致无法为<input>标签按钮设置line-height，其结果就是这种按钮和普通的按钮高度不一致。效果如图3-32所示。

图3-32　input元素在Firefox浏览器无法设置line-height

3.6.4 活动状态

当按钮处于活动状态时，其表现出的行为像被按压了一样（底色更深，边框颜色更深，内置阴影）。对于<button>元素，这是通过: active实现的；而对于<a>元素，则是通过.active实现的。但还可以联合使用：active和<button>，并通过编程的方式使其处于活动状态。

由于: active是伪类，因此无需添加<button>元素就默认支持。但是，如果想在页面加载的时候就表现出活动状态外观的话，则直接在上面添加.active样式即可。示例如下：

```
<button type="button" class="btn btn-primary btn-lg active">button</button>
<a href="#" class="btn btn-primary btn-lg active" role="button">link</a>
```

活动状态的实现方式是，首先在默认的.btn样式上定义: active伪类和.active样式，以提供阴影。源码如下：

```
// 源码2018行
.btn:active,
.btn.active {
  background-image: none;
  outline: 0;/* 消除outline*/
  -webkit-box-shadow: inset 0 3px 5px rgba(0, 0, 0, 0.125); /*定义阴影*/
          box-shadow: inset 0 3px 5px rgba(0, 0, 0, 0.125);
}
```

然后在各自风格的样式上，为活动状态定义不同的文字颜色、背景颜色。边框颜色。源码如下：

```
// 源码2040行
.btn-default:hover,
.btn-default:focus,
.btn-default:active,
.btn-default.active,
.open .dropdown-toggle.btn-default {  /* hover、focus、active变化时的颜色 */
  color: #333;
  background-color: #ebebeb;
  border-color: #adadad;
}
.btn-default:active,
.btn-default.active,
.open .dropdown-toggle.btn-default {  /* active状态下,背景图设置为none */
  background-image: none;
}
```

3.6.5 禁用状态

与活动状态一样，Bootstrap也提供了禁用状态按钮的设置，主要实现将按钮的背景色做65%的透明处理。有两种方式可以禁用按钮，一种是使用原始的disabled属性，一种是利用.disabled样式。两者的区别是.disabled样式不禁止按钮的默认行为（需要利用JavaScript代码来阻止）。

使用方式如下：

```
方式一:
<button type=“button” class=“btn btn-primary” disabled=“disabled”>大按钮
</button>
方式二:
<button type=“button” class=“btn btn-primary disabled”>大按钮</button>
```

注意

IE　9和以下版本下使用disabled属性会导致文本变成灰色，并产生阴影。

另外，.disabled样式作用在拥有.btn样式的链接a元素上时，也会产生同样的禁用状态效果，唯一的不足也是阻止不了链接的默认行为（需要JavaScript代码阻止其行为）。用法如下：

```
<a href=”#” class=“btn btn-primary btn-lg disabled”>大按钮</a>
<a href=”#” class=“btn btn-default btn-sm disabled”>小按钮</a>
```

相关源码如下：

```
// 源码2025行
.btn.disabled,
.btn[disabled],
fieldset[disabled] .btn {
  pointer-events: none;
  cursor: not-allowed;
  filter: alpha(opacity=65);
  -webkit-box-shadow: none;                    /* 去除阴影*/
          box-shadow: none;
  opacity: .65;                                /* 透明处理*/
}
```

同时，各自风格对禁用状态也进行了颜色的特殊处理。举例如下：

```
// 源码2054行
.btn-default.disabled,
.btn-default[disabled],
/* 此处省略部分选择符 */
fieldset[disabled] .btn-default.active {                    /* 禁用状态下,各
种变化时的颜色 */
 background-color: #fff;                                    /* 禁用状态下,背
景色 */
```

```
 border-color: #ccc;                                    /* 禁用状态下,边框颜色 */
}
```

3.7 图像

源码文件：scaffolding.less

CSS文件：bootstrap.css 270行开始

在图片显示方面，Bootstrap框架提供了3种风格效果，使用方式也非常简单，只需要分别在img标签上应用.img-rounded、.img-circle、.img-thumbnail样式即可，如图3-33所示。

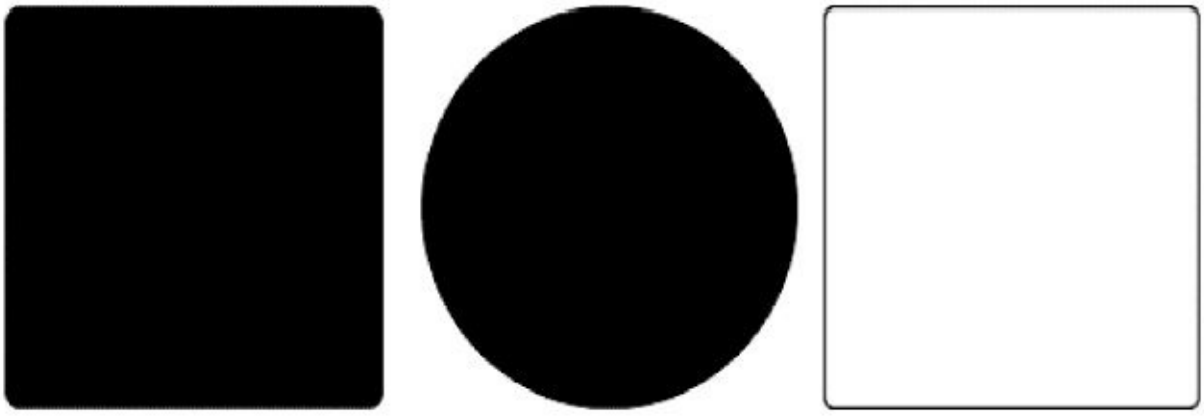


图3-33　图像在不同样式下的运行效果

3种效果的源码定义比较简单，通过如下源码，可以看出其区别。

```
/* 源码 307 行*/
/*默认设置和响应式设置*/
img {
  vertical-align: middle;                    /*垂直居中*/
}
.img-responsive {                                    /*响应式设计*/
  display: block;
  max-width: 100%;                          /*填充父元素*/
  height: auto;                             /*高度自适应*/
}
.img-rounded {                              /*圆角样式*/
  border-radius: 6px;                       /*圆角设置*/
}
.img-thumbnail {                                /*缩略图模式*/
  display: inline-block;                        /*inline模式*/
  max-width: 100%;
  height: auto;
  padding: 4px;                             /*内边距*/
  line-height: 1.428571429;
  background-color: #fff;                   /*白色背景和内边距配合显得有层次感*/
  border: 1px solid #ddd;                   /*边框*/
  border-radius: 4px;                       /*圆角*/
  -webkit-transition: all .2s ease-in-out;
          transition: all .2s ease-in-out;
}
.img-circle {                               /*圆形样式*/
  border-radius: 50%;
```

```
}
```

由源码可以看出一个问题需要我们使用过程中进行注意，那就是，上述样式没有控制图片的显示大小，所以需要额外应用样式或者限制父元素大小来控制图片显示大小。

注意

IE8及以下版本不支持.img-rounded和.img-circle样式特效。

还有一个单独的缩略图（thumbnail）组件效果，其实现稍微有点复杂，在后面讲述。

3.8 辅助样式

Bootstrap在布局方面还提供了一些细小的辅助样式，用于文字颜色及背景色设置、显示关闭图标、向左浮动、向右浮动、清除浮动、隐藏、显示等功能。

3.8.1　文本样式及背景样式

本节简单介绍一下文本样式及文本的背景样式。

1.文本样式

在文本显示方面，Bootstrap提供了6种统一的样式颜色，分别是：柔和灰（text-muted）、主要蓝（text-primary）、成功绿（text-success）、信息蓝（text-info）、警告黄（text-warning）、危险红（text-danger）。相关代码在Bootstrap.css的497～560行定义，效果如图3-34所示。

图3-34　文本颜色样式运行效果

文本样式的使用方式如下所示：

```
<p class="text-muted">JavaScript编程精解</p>
<p class="text-primary">JavaScript设计模式</p>
<p class="text-success">JavaScript启示录</p>
<p class="text-info">Backbone应用开发实战</p>
<p class="text-warning">深入理解Bootstrap3</p>
<p class="text-danger">编写可测试的JavaScript</p>
```

分析下面的样例源码可以看出，将鼠标移动到文本上，还会显示该色系的另外一种颜色。

```
// 源码500行
.text-primary {  color: #428bca;}
.text-primary:hover {  color: #3071a9;}
```

2.文本背景样式

与此同时，Bootstrap还提供了5种背景颜色和上面的5种文本颜色样式对应（muted样式除外），分别是：主要蓝（bg-primary）、成功绿（bg-success）、信息蓝（bg-info）、警告黄（bg-warning）、危险红（bg-danger），其运行效果如图3-35所示。

图3-35　文本背景样式运行效果

3.8.2 辅助图标

本小节主要讲述常用的关闭和向下箭头两种辅助图标。

1.关闭图标

关闭图标的初始代码在单独的close.less文件中定义，其有以下两种使用方式：

```
<!-- 第一种方式 --><button type="button" class="close">&times;</button>
<!-- 第二种方式 --><a class="close">&times;</a>
```

关闭图标的源码较为简单，这里就不详述了，请参见CSS文件的5068～5092行。

2.向下箭头

在下拉菜单中经常看到表示向下效果的小箭头，Bootstrap专门为此提供了一个caret样式，其使用方式非常简单，直接在相应的span（或其他元素）上应用该样式即可。

向下箭头的实现原理是在行内块级元素上，通过定义border-top、border-left、border-right来实现。具体源码如下：

```
// 源码2985行
.caret {                                                    /*向下箭头*/
  display: inline-block;
  width: 0;  height: 0;
  margin-left: 2px;
  vertical-align: middle;
  border-top: 4px solid;
  border-right: 4px solid transparent;
  border-left: 4px solid transparent;
}
```

3.8.3 内容浮动

在平时制作页面的过程中，对于内容浮动，一般会有3种需求（左浮动、右浮动、居中对齐），另外还有一种清除浮动需求。

对于左右浮动，分别定义了pull-left和pull-right样式。其源码如下：

```
/* 源码 5650 行*/
.pull-right {  float: right !important;          /*向右浮动*/ }
.pull-left {  float: left !important;            /*向左浮动*/}
```

而对于居中对齐，使用center-block样式即可，其定义原理是让左右外边距均为auto即可。源码定义如下：

```
/* 源码 5645 行*/
.center-block {
  display: block;                                          /*将页面元素设置为
块级元素*/
  margin-right: auto;                                      /*左右居中显示*/
  margin-left: auto;
}
```

一般某一个元素在浮动以后，在紧接着的元素使用之前可以清除浮动，以避免布局错乱。清除浮动一般是设置元素的before伪类display为table（且content为空），并设置after伪类的clear为both。Bootstrap提供了一个clearfix样式用于清除浮动，其定义如下：

```
/* 源码 5598 行*/
.clearfix:before,
.clearfix:after {  display: table;  content: " ";}
.clearfix:after {  clear: both;}
```

查看CSS文件的5598行以后的代码可发现，清除浮动的代码有关有50行左右，主要是对其他一些组件所使用的样式也定义了清除浮动代码（而不用再次应用clearfix），比如为nav样式和modal-footer样式清除浮动，就可以避免后续元素的布局错乱。

3.8.4 隐藏与显示

对于元素或者文本内容，通常我们要注意两种形式的隐藏和显示，一种是display的隐藏和显示，另一种是visibility的隐藏和显示。对于显示元素，Bootstrap框架仅定义了show样式，而对于隐藏则定义了hidden和invisible样式，其中hidden表示元素既不显示，也不占用文档流（会影响布局的调整），invisible则表示虽然不显示，但其占用文档流占位。这3种样式的源码定义如下：

```
/* 源码 5659 行*/
.show {  display: block !important;                    /*显示内容*/}
.invisible {  visibility: hidden;                      /*隐藏可视性*/}
.hidden {
  display: none !important;                            /*隐藏内容*/
  visibility: hidden !important;                       /*隐藏可视性*/
}
```

另外，如果一个元素对所有设备都要隐藏，而只对盲人可用的话，可以使用特殊的sr-only样式。

注意

虽然目前还有一个hide样式是可用的，但是未来版本将会删除它，请大家尽量不要再用该样式。

还有一种特殊情况，若想让一个元素的文本内容不显示，而只显示背景效果（也就是类似透明的效果）的话，可以使用text-hide样式。其实现原理是隐藏文本，设置背景色透明。源码如下：

```
/* 源码 5665 行*/
.text-hide {
  font: 0/0 a; /* 等价于 font-size: 0px;line-height: 0;font-family: a;*/
  color: transparent;
  text-shadow: none;
   background-color:  transparent;/*将页面元素所包含的文本内容替换为透明色,以显示背景图片*/
  border: 0;
}
```

3.9 响应式样式

源码文件：responsive-utilities.less

CSS文件：bootstrap.css 5679行开始

响应式设计可以适应不同尺寸的设备，它会根据不同尺寸的设备对特定的元素进行显示和隐藏设置，同时响应式设计也可以区别打印模式和普通浏览模式。

根据第2章介绍的响应式设计的思想，Bootstrap提供了8个样式，以适配不同尺寸的屏幕。如图3-36所示是每个样式在不同尺寸下的显示和隐藏情况。

上述样式的主要原理就是利用媒体查询的特性，对特定尺寸的屏幕进行隐藏或显示的设置。比如，.visible-开头的样式表示仅在某尺寸时显示，其他都隐藏；而.hidden样式则表示仅在某尺寸时隐藏，其他都显示。

	超小屏幕 手机 (<768px)	小屏幕 平板 (≥768px)	中等屏幕 桌面 (≥992px)	大屏幕 桌面 (≥1200px)
.visible-xs	可见	隐藏	隐藏	隐藏
.visible-sm	隐藏	可见	隐藏	隐藏
.visible-md	隐藏	隐藏	可见	隐藏
.visible-lg	隐藏	隐藏	隐藏	可见
.hidden-xs	隐藏	可见	可见	可见
.hidden-sm	可见	隐藏	可见	可见
.hidden-md	可见	可见	隐藏	可见
.hidden-lg	可见	可见	可见	隐藏

图3-36　响应式设计在不同尺寸下的隐藏和显示情况

关于具体的代码实现，请查看CSS文件5688行以后的源码。其主要就是利用了如下媒体查询语法，对上述8个样式进行分别设置。

```
// 源码5688行
@media (max-width: 767px) {
    /*在小于768像素的设备上的显示情况*/
}
```

```
@media (min-width: 768px) and (max-width: 991px) {
     /*在768像素和992像素之间的设备上的显示情况*/
}
@media (min-width: 992px) and (max-width: 1199px) {
     /*在992像素和1200像素之间的设备上的显示情况*/
}
@media (min-width: 1200px) {
     /*在大于1200像素的设备上的显示情况*/
}
```

另外，Bootstrap利用了@media print语法，提供了分别对浏览器和打印机进行隐藏和显示的设置，样式分别为：.visible-print和.hidden-print。其作用如图3-37所示。

样式	浏览器	打印机
.visible-print	隐藏	可见
.hidden-print	可见	隐藏

图3-37　对浏览器和打印机的隐藏和显示情况

具体实现源代码非常简单，读者可以自行阅读5804行后的源码。

注意

测试响应式的CSS样式可以在普通浏览器下进行（通过拖放浏览器改变浏览器可视大小，即可得到相应的效果）。

第4章　CSS组件

本章要介绍的内容是Bootstrap框架的三大核心之二：组件。组件也是最核心的地方，因为绝大部分的网页都必须利用组件才能构建出绚丽的页面。这些组件包括：Icon图标（Glyphicon）、下拉菜单（Dropdown）、按钮组（Button group）、按钮下拉菜单（Button dropdown）、输入框组（Input group）、导航（Nav）、导航条（Navbar）、面包屑导航（Breadcrumb）、分页导航（Pagination）、标签（Label）、徽章（Badge）、大屏幕展播（Jumbotron）、页面标题（Pageheader）、缩略图（Thumbnail）、警告框（Alert）、进度条（Progress bar）、媒体对象（Media object）、列表组（Listgroup）、面板（Panel）和洼地（Well），总计20种。本章将对这些组件进行全面分析，包括使用方法、注意事项、实现方式（源码分析）等。

4.1　小图标

源码文件：glyphicons.less

CSS文件：bootstrap.css 2367行开始

小图标（icon）是一个优秀网站不可缺少的一组元素，小图标的点缀可以使网站瞬间提升一个档次。新版Bootstrap提供了200个小图标（相应地，提供了200个CSS样式），这200个样式均可以在内联元素上应用，并显示出对应的图标。

4.1.1 基本用法

小图标的基本用法非常简单，在任何内联元素上应用所对应的样式即可。示例如下：

```
<!– 第一种方式 –><i class=“glyphicon glyphicon-search”></i>
<!– 第二种方式 –><span class=“glyphicon glyphicon-search”></span>
```

所有的icon样式都以glyphicon-开头，这是因为这些小图标都是http://glyphicons.com/网站提供，使用的时候必须同时使用两个样式，即.glyphicon和以.glyphicon-*开头的样式（稍后会讲解其原因）。

注意

glyphicons.com是一家提供商业icon图片集的网站，上述200个icon免费授权给了Bootstrap框架，所以大家可以随意免费试用。

小图标可以用于多种用途，比如编辑器里的小图标等。示例如下：

```
<div class=“btn-toolbar”>
  <div class=“btn-group”>
                          <a     class=“btn     btn-default”     href=”#”>
<span class=“glyphicon glyphicon-align-left”>
                </span></a>
                          <a     class=“btn     btn-default”     href=”#”>
<span class=“glyphicon glyphicon-align-center”>
                </span></a>
                          <a     class=“btn     btn-default”     href=”#”>
<span class=“glyphicon glyphicon-align-right”>
                </span></a>
                          <a     class=“btn     btn-default”     href=”#”>
<span class=“glyphicon glyphicon-align-justify”>
                </span></a>
  </div>
</div>
```

上述代码的运行效果如图4-1所示。

图4-1　icon运行效果

全部的Glyphicons小图标请查阅附录D。

4.1.2 实现方式

新版小图标的实现方式与之前的2.x系列版本完全不一样。老版本的实现方式是利用了图片精灵的特性，即所有的小图片都放在一张大图片上；每个小图片通过样式和坐标进行定位显示。

小图标的老版实现源码如下所示：

```
[class^=“icon-“],[class*=” icon-“] {
  display: inline-block;
   width: 14px;        height: 14px;          /*  每个图标的宽度和高度都设置为
14px */
  margin-top: 1px;  *margin-right: .3em;
  line-height: 14px;  vertical-align: text-top;
  background-image: url(“images/glyphicons-halflings.png”);
  background-position: 14px 14px;
  background-repeat: no-repeat;
}
.icon-music {
  background-position: -24px 0;          /* 定位开始坐标 */
}
```

老版的这种方式有以下两个缺点：

（1）图片大小被固定了，如果放大就会失真。

（2）图片的颜色只有黑色和白色两种，没办法扩展。

新版本的实现方式进行了革命性的更新，使用了CSS3中的@font-face特性。@font-face是CSS3中有关于字体设置的属性，通过它可以将服务器上的字体设置为Web页面字体，并且能完全兼容所有浏览器，使用这个属性则不必担心用户本地不具备这样的字体。因为把字体都上传在服务器上了，可能很多人会担心影响性能问题，但现代的浏览器都会进行样式缓存，有点影响所以影响不是很大，毕竟鱼和熊掌不能兼得。

新版icon就是利用这个@font-face特性，并结合一定的字体，来制作Web页面中的icon图标。原理就是，就是利用特殊的文字在特殊的字体上显示特殊的效果，比如字符A在特殊字体上显示成圆圈的形式，字符B显示为椭圆的形式等。这种形式不仅可以让字体无损失放大，还可以简单地通过color: #ff0的形式设置图标的颜色（因为它本身就是字体，字体当然可以设置颜色）。我们先来看一下@font-face的语法。

```
@font-face {
   font-family: <YourWebFontName>;
   src: <source> [<format>][,<source> [<format>]]*;
   [font-weight: <weight>];
   [font-style: <style>];
 }
```

上述语法的参数取值说明如下：

❑YourWebFontName：指的就是你自定义的字体名称，最好是使用自己下载的默认字体，它将被引用到你的Web元素中的font-family，如“font-family:“YourWebFontName”;”。

❑source：指的是自定义字体的存放路径，可以是相对路径也可以是绝路径。

❑format：指的是自定义字体的格式，主要用来帮助浏览器识别。其值主要有以下几种类型，truetype、opentype、truetype-aat、embedded-opentype、avg。

❑weight和style：这两个值大家一定很熟悉，weight定义字体是否为粗体，style主要定义字体样式，如斜体。

要让浏览器对@font-face进行全部兼容，就需要进行特殊设置，这涉及一个字体format的问题，因为不同浏览器对字体格式的支持是不一样的，不同版本的浏览器可支持不同的字体。下面就各种格式字体进行说明：

❑TureTpe（.ttf）格式：.ttf字体是Windows和Mac中的最常用的字体，是一种RAW格式，因此它不能被网站所优化。支持这种字体的浏览器有IE9+、Firefox3.5+、Chrome4+、Safari3+、Opera10+、iOS Mobile Safari4.2+。

❑OpenType（.otf）格式：.otf字体被认为是一种原始的字体格式，其内置在TureType的基础上，所以也提供了更多的功能。支持这种字体的浏览器有Firefox3.5+、Chrome4.0+、Safari3.1+、Opera10.0+、iOS Mobile Safari4.2+。

❑Web Open Font Format（.woff）格式：.woff字体是Web字体中的最佳格式，它是一个开放的TrueType/OpenType的压缩版本，同时也支持元数据包的分离。支持这种字体的浏览器有IE9+、Firefox3.5+、Chrome6+、Safari3.6+、Opera11.1+。

❑Embedded Open Type（.eot）格式：.eot字体是IE专用字体，可以从TrueType创建此格式字体。支持这种字体的浏览器有IE4+。

❑SVG（.svg）格式：.svg字体是基于SVG字体渲染的一种格式，支持这种字体的浏览器有Chrome4+、Safari3.1+、Opera10.0+、iOS Mobile Safari3.2+。

这就意味着在@font-face中我们至少需要.woff、.eot两种格式字体，才能适应所有主流的浏览器，甚至还需要.svg等字体以实现更多种浏览器版本的支持。为了使@font-face得到更多的浏览器支持，Paul Irish写了一个独特的@font-face语法叫Bulletproof @font-face。代码如下：

```
  @font-face {
     font-family: ‘YourWebFontName’;
     src: url(‘YourWebFontName.eot?’) format(‘eot’);            /*IE*/
     src:url(‘YourWebFontName.woff’) format(‘woff’), url(‘YourWebFontNam
         format(‘truetype’);/*非IE*/
  }
```

但为了获得更多的浏览器支持，也可以写成如下形式：

```
@font-face {
font-family: ‘YourWebFontName’;
src: url(‘YourWebFontName.eot’);                                  /* IE9 兼
容模式 */
src:         url(‘YourWebFontName.eot?#iefix’)         format(‘embedded-
opentype’),     /* IE6～IE8 */
     url(‘YourWebFontName.woff’) format(‘woff’),                  /* 现代
浏览器 */
     url(‘YourWebFontName.ttf’)  format(‘truetype’), /* Safari、 Android
       url(‘YourWebFontName.svg#YourWebFontName’) format(‘svg’);   /* 旧
版 iOS */
}
```

下面来看一下Bootstrap是如何定义这些代码的。

```
// 源码2367行
@font-face {
 font-family: ‘Glyphicons Halflings’;
 src: url(‘../fonts/glyphicons-halflings-regular.eot’);
          src:          url(‘../fonts/glyphicons-halflings-regular.eot?
#iefix’) format(‘embedded-opentype’),
     url(‘../fonts/glyphicons-halflings-regular.woff’) format(‘woff’),
                                    url(‘../fonts/glyphicons-halflings-
regular.ttf’) format(‘truetype’),
                                    url(‘../fonts/glyphicons-halflings-
regular.svg#glyphicons_halflingsregular’)
              format(‘svg’);
}
```

上述代码首先定义了字体名称为Glyphicons Halflings，然后根据语法设置了多个浏览器兼容的字体文件。而使用方式，也是利用了CSS的before和content特性进行展示，意思是在应用样式的元素前插入content设定的内容。源码如下：

```
// 源码2373行
.glyphicon {
 position: relative;    /*设置相对定位*/
 top: 1px;
 display: inline-block;
 font-family: ‘Glyphicons Halflings’;   /*设置字体名称*/
 font-style: normal;
 font-weight: normal;
 line-height: 1;
 -webkit-font-smoothing: antialiased;
 -moz-osx-font-smoothing: grayscale;
}
```

```
/*示例icon*/
.glyphicon-search:before {                          /* 所有的icon都在before之前设定content */
 content: ”\e003”; /* 表示\e003字符在Glyphicons Halflings字体里显示的是搜索图像*/
}
```

4.1.3 应用场景

icon样式可以用在各种元素容器内，比如button元素、nav列表元素等。例如，可以通过如下代码实现：

```
<div class="btn-group">
        <a    class="btn    btn-default    btn-primary"    href="#">
<span class="glyphicon
     glyphicon-user"></span>User</a>
    <a  class="btn  btn-default  btn-primary  dropdown-toggle"  data-
toggle=" href="#">
     <span class="caret"></span></a>
  <ul class="dropdown-menu">
          <li><a  href="#"><span  class="glyphicon  glyphicon-pencil">
</span>Edit</a></li>
           <li><a  href="#"><span  class="glyphicon  glyphicon-trash">
</span>Delete</a></li>
        <li><a  href="#"><span  class="glyphicon  glyphicon-ban-circle">
</span>Ban</a></li>
    <li class="divider"></li>
    <li><a href="#">Make admin</a></li>
  </ul>
</div>
```

上述代码的运行效果如图4-2所示。

图4-2　icon在按钮里的运行效果

或者结合btn样式，显示效果如图4-3所示。

图4-3　结合btn样式的icon运行效果

```
<a class="btn btn-default btn-large" href="#">
   <span class="glyphicon glyphicon-star"></span> Star
</a>
<a class="btn btn-default btn-small" href="#">
   <span class="glyphicon glyphicon-star"></span>
</a>
```

另外，一个非常常见的场景是，在每个操作菜单前显示代表该菜单意思的icon，效果如图4-4所示。

图4-4　显示代表菜单意见的icon

实现方式需要在具有nav、nav-pills和nav-stacked样式的ul元素下才行（nav样式组件在后面章节中详细介绍）。注意，图标的颜色和右边的文字保存一致，也就是说，该图标用起来和普通文字一样，可以随意设定颜色。

```
<ul class=“nav nav-pills nav-stacked”>
 <li class=“active”><a href=”#”><span class=“glyphicon glyphicon-home”>
</span>
         Home</a></li>
     <li><a     href=”#”><span     class=“glyphicon     glyphicon-book”>
</span> Library</a></li>
 <li><a href=”#”><span class=“glyphicon glyphicon-pencil”></span>
         Applications</a></li>
  <li><a  href=”#”><span  class=“glyphicon  glyphicon-“></span>  Misc</a>
</li>
</ul>
```

icon的最后一个常见场景是在表单的输入框前面或后面加上一个小图标，效果如图4-5所示。

图4-5　icon和输入框配合运行

也就是左边的addon显示的是小图标而不是字符，那么只需要在input-group-addon样式的元素内应用一个span或者i元素即可（然后在元素上应用glyphicon图标样式）。示例如下：

```
<div class=“control-group”>
 <div class=“controls”>
  <div class=“input-group”>
   <span class=“input-group-addon”>
   <span class=“glyphicon glyphicon-envelope”></span>
   </span>
   <input class=“col col-lg-2 form-control” id=“inputIcon” type=“text”>
  </div>
 </div>
</div>
```

4.1.4 其他

在阅读glyphicon样式源码的时候，可以发现所有的icon样式都是利用“content：‘\e001’”的形式设置的。如果不利用before特性，如何在普通的文本上进行显示呢？则需要在e001前面加上unicode前缀&#x，才能够作为正常文本进行显示。原因是什么呢？

content内的字符默认是保存在Unicode Private Use Area中的，即Unicode专用区，也就是保留给大家存放自定义字符的区域，可以简写为PUA。要想在HTML里像普通字符一样显示它们，则需要加前缀&#x。比如下面两个span显示的效果是一样的，都是玻璃杯。

```
<span class=“glyphicon glyphicon-glass”></span>
<span class=“glyphicon”>&#xe001</span>
```

注意

在第9章，我们会利用@font-face特性，再次全面扩展新的图标集，即Font Awesome 3.1所提供的强大功能——将近400个图标。

4.2　下拉菜单

源码文件：dropdowns.less

CSS文件：bootstrap.css 2985行开始

网页交互的时候经常会需要上下文菜单或者隐藏/展示菜单项，Bootstrap默认提供了通用的菜单显示效果，而且在各种交互状态下的菜单展示需要和JavaScript的Dropdown插件配合才能使用。本小节只讲述如何展示菜单（JavaScript交互插件在第5章讲解）。

4.2.1 基本用法

2.x版本和最新的3.x版本的使用方式不一样了。原来老版本的使用方式如下：

```
<div class=“dropdown”>
 <ul class=“dropdown-menu” role=“menu” aria-labelledby=“dropdownMenu”>
  <li><a tabindex=”-1” href=”#”>Action</a></li>
  <li><a tabindex=”-1” href=”#”>Another action</a></li>
  <li><a tabindex=”-1” href=”#”>Something else here</a></li>
  <li class=“divider”></li>
  <li><a tabindex=”-1” href=”#”>Separated link</a></li>
 </ul>
</div>
```

上述代码的运行效果如图4-6所示。

图4-6　下拉菜单运行效果

从示例可以看出，.dropdown样式是大容器，.dropdown-menu是菜单li元素的容器，而.divider样式在li元素上的显示效果是分隔符。

但在新版里，.dropdown-menu样式的行为不一样，其默认是隐藏的。先看下面的源码：

```
// 源码3001行
.dropdown-menu {
 position: absolute;                          /* 绝对定位*/
 top: 100%; /* 如果该元素的父元素不是相对定位的话，那这个100%就是相对于body元素的
*/
 left: 0;
 z-index: 1000;                               /* 加大z-index，使其不被其他元素遮
盖*/
 display: none;                               /* 菜单默认是隐藏的*/
 float: left;
 min-width: 160px;
 padding: 5px 0;
 margin: 2px 0 0;
 font-size: 14px;
 list-style: none;
 background-color: #fff;
 background-clip: padding-box;
 border: 1px solid #ccc;                                  /* 边框效果*/
 border: 1px solid rgba(0, 0, 0, .15);  /* 透明度为0.15*/
 border-radius: 4px;                          /* 阴影效果*/
 -webkit-box-shadow: 0 6px 12px rgba(0, 0, 0, .175);
```

```
        box-shadow: 0 6px 12px rgba(0, 0, 0, .175);
}
```

在上述代码中，我们不仅看到了display: none;，还看到了为.dropdown-menu样式设置了绝对定位，并且还有一个top: 100%;设置，也就是说，该样式所在菜单必须有一个相对位置设置的父元素（即父元素由position: relative;设定）才正常（否则就显示到第二屏幕去了）。所以就出现了以下两个问题：

❑如何确保父元素是相对定位？

❑什么情况下能让菜单打开，即让.dropdown-menu样式的display设置为非none？

先看第一个问题。通过官方例子，可以看出父元素上应用了.dropdown样式，猜想这个样式可能就是设置相对定位的样式吧。来验证一下，查找该样式代码，果不其然。

```
// 源码2995行
.dropdown {
  position: relative; /* 由于只设置了相对定位，所以你也可以将该样式替换成其他样式，只要满足
                                    相对定位即可*/
}
```

再来看第二个问题：如何改变display方式？通过搜索所有与.dropdown-menu相关的样式，发现在源码的3067行有如下代码：

```
// 源码3067行
.open > .dropdown-menu {
 display: block;
}
```

莫非就是这一个？那就在.dropdown样式所在元素上再应用一个.open样式试试。正如我们所想的，菜单显示了。难道真的就是这个样式吗？为什么非要有一个.open样式？

通过了解得知，Bootstrap还提供了一个下拉菜单的JS插件（第5章5.3节详细介绍），以便让下拉菜单得到充分的应用。该插件提供了一个单击展开的效果，通过Firebug分析，如图4-7所示。可以看到，当单击“下拉菜单”按钮的时候，会自动在.dropdown样式的div元素上再次附加一个.open样式。这就验证了，.open样式确实是为了处理下拉菜单而设置的专用样式。

```
<div class="dropdown open">
    <button class="btn btn-default" href="#" data-toggle="dropdown">下拉菜单</button>
  ⊞ <ul class="dropdown-menu" aria-labelledby="dropdownMenu" role="menu">
</div>
```

图4-7　下拉菜单展开时会应用open样式

综合上述代码分析，我们的结论是，只有在菜单的父元素上应用.open样式并且该元素设定为position:　　relative;的时候（或者直接使用.dropdown样式），下拉菜单才会正常显示。即类似如下代码：

```
<div class=“open” style=“position:relative”>
 <ul class=“dropdown-menu” role=“menu” aria-labelledby=“dropdownMenu”>
…
 </ul>
</div>
```

分隔符的源码实现如下：

```
// 源码3025行
.dropdown-menu .divider {
 height: 1px;                    /*高1px*/
 margin: 9px 0;
 overflow: hidden;
 background-color: #e5e5e5;      /*灰色背景*/
}
```

除了分隔符以外，3.x新版还提供了一个菜单标题的功能（类似于分组标题），使用.dropdown-header样式，通过如下代码进行使用：

```
<div class=“dropdown open clearfix”>
     <ul  aria-labelledby=“dropdownMenu2”  role=“menu”  class=“dropdown-
menu”>
      <li class=“dropdown-header” role=“presentation”>翻译图书</li>
                                          <li       role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>JavaScript
              编程精解</a></li>
                                        <li          role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>JavaScript
              设计模式</a></li>
                                          <li        role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>JavaScript
              启示录</a></li>
                                          <li       role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>Backbone
              应用开发指南</a></li>
      <li role=“presentation” class=“divider”></li>
      <li role=“presentation” class=“dropdown-header”>原创图书</li>
                                          <li       role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>深入理解
               Bootstrap</a></li>
   </ul>
</div>
```

上述代码的运行效果如图4-8所示。

图4-8　菜单标题显示效果

通过下面的源码我们可以看到，.dropdown-header样式只是对字号、颜色和内边距进行了特殊设置。但是奇怪的是，该样式并没有像.divider一样把样式放在.dropdown-menu里面（即：.dropdown-menu　.divider），也许以后还有其他使用场景吧。

```
// 源码3081行
.dropdown-header {
 display: block;
 padding: 3px 20px;              /* 加大上下内边距*/
 font-size: 12px;                        /* 缩小字号*/
 line-height: 1.428571429;
 color: #999;                    /* 改变颜色*/
}
```

另外，由于菜单默认情况下都是左对齐的，Bootstrap又额外提供了一个.pull-right样式，可以使菜单相对于父容器右对齐（注意.pull-right要和.dropdown-menu一起应用）。

```
<ul     class=“dropdown-menu     pull-right”     role=“menu”     aria-
labelledby=“dLabel”>
 …
</ul>
```

同样，如果在菜单项li元素上应用.active样式后，就会有高亮显示的效果；而如果应用了.disabled样式，则该菜单看起来就像被禁用一样（但行为依然需要JavaScript特殊处理以便禁用）。示例如下：

```
<div class=“dropdown open clearfix”>
     <ul  aria-labelledby=“dropdownMenu2”  role=“menu”  class=“dropdown-
menu”>
                          <li     role=“presentation”     class=“active”>
<a href=”#” role=“menuitem”>JavaScript
             编程精解</a></li>
                        <li     role=“presentation”     class=“disabled”>
<a href=”#” role=“menuitem”>JavaScript
             设计模式</a></li>
      <li role=“presentation”><a href=”#” role=“menuitem”>JavaScript启示
录</a></li>
      <li role=“presentation”><a href=”#” role=“menuitem”>Backbone应用开
发指南</a></li>
```

```
    </ul>
</div>
```

上述代码的运行效果如图4-9所示。

图4-9　禁用菜单运行效果

关于上述样式和li元素的详细源码，请参见Bootstrap.css文件中的第3556行之后的代码。

4.2.2　多级嵌套

在很多情况下，我们可能需要多级菜单（也就是经常提到的嵌套菜单）。原来的2.x版里有这种功能，但在新版里已经被删除了，也许作者认为这个功能使用的频率不是特别多吧。老版是利用在.dropdown-meun样式中嵌套来实现类似的功能，做法是：只需要在子菜单项上应用.dropdown-submenu样式，然后在li元素里，和标准的菜单设置一样，内部嵌套一个新的ul元素即可（在这个ul上也需要设置标准的.dropdown-menu样式）。示例如下：

```
<ul class="dropdown-menu" role="menu" aria-labelledby="dropdownMenu">
   <!— 其他多个菜单项li元素 —>
   <li class="dropdown-submenu pull-left">
      <a tabindex="-1" href="#">More options</a>
      <ul class="dropdown-menu">
         <li><a tabindex="-1" href="#">二级菜单项</a></li>
         <li><a tabindex="-1" href="#">二级菜单项</a></li>
         <li><a tabindex="-1" href="#">二级菜单项</a></li>
         <li><a tabindex="-1" href="#">二级菜单项</a></li>
         <li><a tabindex="-1" href="#">二级菜单项</a></li>
      </ul>
   </li>
   <!— 其他多个菜单项li元素 —>
</ul>
```

上述代码的运行效果如图4-10所示。

图4-10　嵌套菜单运行效果

同理，根据上图的效果可以看出，如果要设置子菜单向上弹出，需要将dropdown样式换成dropup样式，居右的话则需要在菜单项上加pull-right样式。如果要在新版里使用上述功能，可以将V2.x版的如下代码，添加到自定义文件（或扩展文件）中即可。

```
.dropdown-submenu {
   position: relative;          /* 相对定位 */
}
```

```
.dropdown-submenu > .dropdown-menu {
      top: 0;
      left: 100%;                        /* 右侧开始对齐 */
      margin-top: -6px;        margin-left: -1px;
      -webkit-border-radius: 0 6px 6px 6px;
      -moz-border-radius: 0 6px 6px 6px;
      border-radius: 0 6px 6px 6px;
   }
.dropdown-submenu:hover > .dropdown-menu { display: block;    }
.dropup .dropdown-submenu > .dropdown-menu {
top: auto;    bottom: 0;                /* 向上方向 */
   margin-top: 0;    margin-bottom: -2px;
   -webkit-border-radius: 5px 5px 5px 0;
   -moz-border-radius: 5px 5px 5px 0;
   border-radius: 5px 5px 5px 0;
}
.dropdown-submenu > a:after {
   display: block;
   float: right;
   width: 0;    height: 0;
   margin-top: 5px;    margin-right: -10px;
   border-color: transparent;
   border-left-color: #cccccc;  /* 三角符号设置*/
   border-style: solid;
   border-width: 5px 0 5px 5px;
   content: ” ”;
}
.dropdown-submenu:hover > a:after {
    border-left-color: #cccccc;                /* 鼠标移动时的三角符号设置
*/ }
.dropdown-submenu.pull-left  {      float:  none;     /*  取消子菜单的向右浮
动 */ }
.dropdown-submenu.pull-left > .dropdown-menu {
      left: -100%;               /* 向右浮动的时候，子菜单应该从最左侧开始对齐
计算 */
      margin-left: 10px;
      -webkit-border-radius: 6px 0 6px 6px;
      -moz-border-radius: 6px 0 6px 6px;
      border-radius: 6px 0 6px 6px;
}
```

4.3 按钮组

源码文件：button-groups.less

CSS文件：bootstrap.css 3122行以后

Button按钮元素在基于现代浏览器的开发中，用得越来越多了。使用过程中，经常会将多个按钮组成一组进行使用，比如编辑器里的一组组小图标按钮等。本小节中，将针对按钮组展开详细的叙述。

4.3.1 基本用法

按钮组（Button group）的基本用法很简单，只需要在多个按钮外部使用一个容器元素（比如div），然后在容器元素上应用.btn-group样式即可。示例如下：

```
<!– 第一种方式 –>
<div class=“btn-group”>
   <button type=“button” class=“btn btn-default”>Left</button>
   <button type=“button” class=“btn btn-default”>Middle</button>
   <button type=“button” class=“btn btn-default”>Right</button>
</div>
<!– 第二种方式 –>
<div class=“btn-group”>
   <a class=“btn btn-default”>Left</a>
   <a class=“btn btn-default”>Middle</a>
   <a class=“btn btn-default”>Right</a>
</div>
```

容器里的按钮，可以使用button元素，也可以使用a元素，产生的效果都是一样的。系统展示和排列的方式主要是由.btn-group样式和.btn样式来控制的，运行效果如图4-11所示。

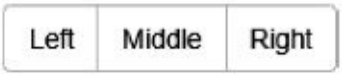


图4-11 按钮组运行效果

按钮组的实现源代码如下：

```
// 源码3122行
.btn-group,
.btn-group-vertical {   /* 多个分组的话，分组之间相对定位 */
 position: relative;    /* 内联块模式 */
 display: inline-block; /* 垂直居中 */
 vertical-align: middle;
}
.btn-group > .btn,
.btn-group-vertical > .btn {
 position: relative;    /* 多个按钮的话，按钮之间相对定位 */
 float: left;           /* 左浮动 */
}
.btn-group > .btn:hover,
/* 此处省略部分选择符 */
.btn-group-vertical > .btn.active {
  z-index: 2;           /* 加大z-index */
}
.btn-group > .btn:focus,
.btn-group-vertical > .btn:focus {
 outline: none;         /* 焦点时，取消轮廓显示 */
}
.btn-group .btn + .btn,
.btn-group .btn + .btn-group,
```

```
.btn-group .btn-group + .btn,
.btn-group .btn-group + .btn-group {
 margin-left: -1px;      /* 消除2个按钮（或一个按钮和另外一个按钮组）之间的1像素细节引起的冲突 */
}
```

另外，默认情况下，一组按钮的4个角都应该是圆角（支持CSS3的话）。其实现原理如下：

❑默认所有.btn样式的按钮都有圆角。

❑排除第一个按钮、最后一个按钮、Dropdown按钮以外，其他按钮都取消圆角设置。

❑第一个按钮的右上和右下角取消圆角设置。

❑最后一个按钮（或Dropdown）的左上和左下角，取消圆角设置。

```
// 源码3165行
.btn-group   >   .btn:not(:first-child):not(:last-child):not(.dropdown-toggle) {
 border-radius: 0;                  /* 除第一个、最后一个按钮和带有dropdown-toggle样式的元素外，其
                                他btn样式的按钮不设置圆角 */
}
.btn-group > .btn:first-child {
 margin-left: 0;                /* 第一个按钮左margin为0 */
}
.btn-group > .btn:first-child:not(:last-child):not(.dropdown-toggle) {
 /* 第一个按钮（不是最后一个按钮和带有dropdown-toggle样式的情况下），右上和右下角不设置圆角 */
 border-top-right-radius: 0;
 border-bottom-right-radius: 0;
}
.btn-group > .btn:last-child:not(:first-child),
.btn-group > .dropdown-toggle:not(:first-child) {
 border-top-left-radius: 0;     /* 最后一个按钮或Dropdown（不是第一个按钮的情况下），左上和左下角不设置圆角 */
 border-bottom-left-radius: 0;
}
```

4.3.2　按钮工具栏

按钮工具栏主要是将多个按钮组排列在一起，例如，1、2、3、4是一个分组，5、6、7是一个分组，8单独是一个分组，总共3个分组显示在一起，如图4-12所示。

图4-12　按钮组工具栏运行效果

按钮工具栏的用法和按钮组类似，即：在多个分组外部放一个大的容器元素，然后在容器元素上应用另外一个新的.btn-toolbar样式。示例如下：

```
<div class=“btn-toolbar”>
 <div class=“btn-group”>…</div>
 <div class=“btn-group”>…</div>
 <div class=“btn-group”>…</div>
</div>
```

源码实现主要是让容器的多个分组以table风格进行显示，并且每个分组之间保持5像素的left margin。代码如下：

```
// 源码5608行
.btn-toolbar:before,
.btn-toolbar:after {
 display: table;                /* table风格显示，等高 */
 content: ” ”;
}
.btn-toolbar:after {
  clear: both;           /* 结束以后，清除浮动 */
}
// 源码3156行
.btn-toolbar .btn-group,
.btn-toolbar .input-group {
 float: left;           /* 每组都是左浮动 */
}
.btn-toolbar > .btn,
.btn-toolbar > .btn-group,
.btn-toolbar > .input-group {
 margin-left: 5px;      /* 每个分组之间有5像素的left margin */
}
```

4.3.3 按钮尺寸设置

在按钮组里也可以使用.btn样式的3个额外的尺寸即.btn-lg、.btn-sm、.btn-xs来调整按钮大小，以便形成如图4-13所示的效果。可以在每一组的所有按钮上重复应用上述样式。

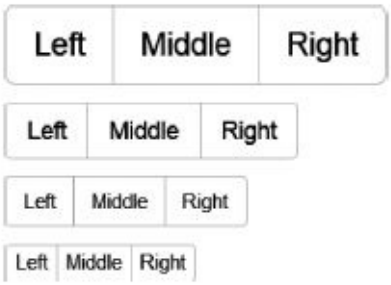


图4-13 不同尺寸的按钮组运行效果

但是Bootstrap专门又为按钮组提供了额外的样式，以便不用在多个按钮上重复应用样式。其用法是：在.btn-group样式的元素上再次应用.btn-group-lg、.btn-group-sm、.btn-group-xs样式即可提供同样的效果，一次设置，全组都有效。

```
<div class="btn-group btn-group-lg">…</div>
<div class="btn-group">…</div>
<div class="btn-group btn-group-sm">…</div>
<div class="btn-group btn-group-xs">…</div>
```

按钮组的实现原理和普通按钮的尺寸设置一样，即修改相关的padding、font-size、line-height以及border-radius。

```
/* 源码 3199 行 */
.btn-group-xs > .btn { padding: 1px 5px; font-size: 12px; line-height: 1.5;
        border-radius: 3px;}
.btn-group-sm > .btn { padding: 5px 10px; font-size: 12px; line-height: 1.5;
        border-radius: 3px;}
.btn-group-lg > .btn { padding: 10px 16px; font-size: 18px; line-height: 1.33;
        border-radius: 6px;}
```

有的读者可能有疑问，如果设置了.btn-group-lg样式，并且在里面的按钮上又设置了.btn-sm样式，效果是怎样的呢？里面的按钮肯定不会变小的，因为.btn-group-lg样式是在.btn-sm样式之后定义的，在这种场景下，会使用后面覆盖的样式。

同时，在btn-group组里，如果在普通按钮旁边放置下拉菜单的话，也进行了padding值的微调（仅普通按钮和大型按钮），以修正显示效果，详细分析见下一小节。

```
// 源码3217行
.btn-group > .btn + .dropdown-toggle {  padding-right: 8px;  padding-left: 8px;}
.btn-group  >  .btn-lg  +  .dropdown-toggle  {  padding-right: 12px;  padding-left: 12px;}
```

4.3.4 嵌套分组

在平时使用过程中，我们经常将下拉菜单和普通的按钮组排列在一起，实现如图4-14所示的效果。

图4-14　嵌套分组运行效果

使用的时候，只需要将dropdown下拉菜单外部包装一个div容器元素，并在div元素上重新应用.btn-group样式，并且和普通的1、2、3按钮放在同一级即可。示例如下：

```
<div class="btn-group">
   <button class="btn btn-default" type="button">首页</button>
   <button class="btn btn-default" type="button">个人简介</button>
   <button class="btn btn-default" type="button">作品</button>
   <div class="btn-group">
        <button data-toggle="dropdown" class="btn btn-default dropdown-
toggle"
                type="button"> 图书<span class="caret"></span>
      </button>
      <ul class="dropdown-menu">
         <li><a href="#">JavaScript编程精解</a></li>
         <li><a href="#">JavaScript设计模式</a></li>
         <li><a href="#">JavaScript启示录</a></li>
      </ul>
   </div>
</div>
```

通过上述HTML代码可以发现，并没有在dropdown-menu父容器的div上设置dropdown样式（因为该样式有position: relative;设置），这是因为btn-group里也设置了该相对定位，所以可以省略dropdown样式。

圆角设置的规则和多组按钮的设置类似，唯一的不同就是嵌套的btn-group样式（以btn-group>btn-group开头）。主要源码如下：

```
// 源码3734行
.btn-group > .btn-group {
  float: left;
}
.btn-group > .btn-group:not(:first-child):not(:last-child) > .btn {
 border-radius: 0;      /* 嵌套分组：除第一个分组、最后一个分组外，其他
                           分组内的btn样式的按钮不设置圆角 */
}
.btn-group > .btn-group:first-child > .btn:last-child,
.btn-group > .btn-group:first-child > .dropdown-toggle {
```

```
 border-top-right-radius: 0; /* 嵌套分组内的第一个分组的最后一个按钮的右上和右下角不设置圆角 */
 border-bottom-right-radius: 0;
}
.btn-group > .btn-group:last-child > .btn:first-child {
 border-top-left-radius: 0;
 border-bottom-left-radius: 0; /* 嵌套分组内的最后一个分组的第一个按钮的左上和左下角不设置圆角 */
}
.btn-group .dropdown-toggle:active,
.btn-group.open .dropdown-toggle {      /* 嵌套dropdown，在打开的情况下，设置阴影*/
 outline: 0;
}
/* 源码 3225 行 */
.btn-group.open .dropdown-toggle {      /* 嵌套dropdown，在打开的情况下，设置阴影*/
 -webkit-box-shadow: inset 0 3px 5px rgba(0, 0, 0, .125);
       box-shadow: inset 0 3px 5px rgba(0, 0, 0, .125);
}
.btn-group.open .dropdown-toggle.btn-link {
 -webkit-box-shadow: none; /* 嵌套dropdown，在打开的情况下，如果是link按钮，就取消阴影*/
       box-shadow: none;
}
```

4.3.5　垂直分组

有些特殊情况下，需要显示垂直分组的按钮效果，如图4-15所示。

图4-15　垂直分组运行效果

使用的时候只需要将基础用法里的.btn-group样式替换为.btn-group-vertical样式即可。示例如下：

```
<div class=“btn-group-vertical”>
   <button class=“btn btn-default” type=“button”>首页</button>
   <button class=“btn btn-default” type=“button”>个人简介</button>
   <button class=“btn btn-default” type=“button”>作品</button>
</div>
```

注意

按钮分组样式.btn-group的设计思想和普通的按钮样式.btn的设计思想不太一样，普通和垂直分组是通过直接替换成不同的class样式实现的，而不是在现有.btn-group样式的基础上，再附加一个新样式实现的（比如：class=“btn-group btn-group-vertical”）。

复杂的垂直分组按钮和普通按钮组在实现方法上基本一样：不同点如下：

❑垂直按钮组的宽度是自适应的，所以有些情况需要控制父容器的宽度。

❑普通分组的圆角是第一个按钮的左上角和左下角，最后一个按钮的右上角和右下角。

❑垂直分组的圆角是第一个按钮的左上角和右上角，最后一个按钮的左下角和右下角。

复杂的垂直分组按钮和普通按钮组的效果比较如图4-16所示。

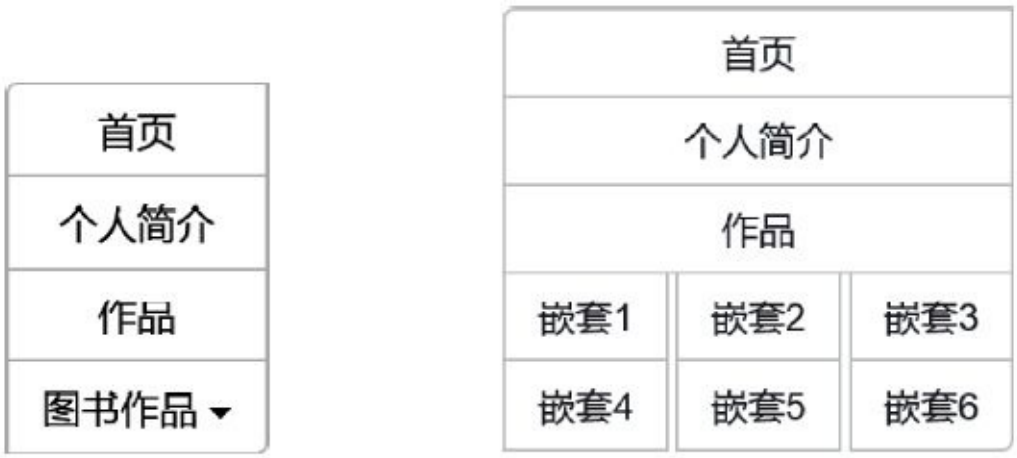


图4-16　复杂的垂直分组按钮

图4-16所示是两个比较复杂的垂直按钮组，尤其是第二个。下面结合该图一步一步分析其实现源码。

```
// 源码3243行
.btn-group-vertical > .btn,
.btn-group-vertical > .btn-group,
.btn-group-vertical > .btn-group > .btn {/* 该样式在V3.1.0 版才更新，下面会叙述相关的bug */
 display: block;                /* 作为块级元素显示 */
 float: none;
 width: 100%;           /* 100%显示，如果图片过大，需要控制父元素的大小 */
 max-width: 100%;
}
.btn-group-vertical > .btn-group > .btn {
  float: none;          /* 如果有嵌套的btn-group组，将其内部按钮的浮动清除 */
}
.btn-group-vertical > .btn + .btn,
.btn-group-vertical > .btn + .btn-group,
.btn-group-vertical > .btn-group + .btn,
.btn-group-vertical > .btn-group + .btn-group {
 margin-top: -1px;      /* 消除2个按钮上下之间的1像素细节引起的冲突 */
 margin-left: 0;                /* 消除2个按钮之间外左边距 */
}
.btn-group-vertical > .btn:not(:first-child):not(:last-child) {
 border-radius: 0;              /* 除第一个按钮、最后一个按钮外，其他btn样式的按钮不设置圆角 */
}
.btn-group-vertical > .btn:first-child:not(:last-child) {
 border-top-right-radius: 4px; /* 第一个按钮（并且不是最后一个按钮）的左下角和右下角不设置圆角 */
 border-bottom-right-radius: 0;
 border-bottom-left-radius: 0;
}
.btn-group-vertical > .btn:last-child:not(:first-child) {
 border-top-left-radius: 0; /* 最后一个按钮（并且不是第一个按钮）的左上角和右上角不设置圆角 */
 border-top-right-radius: 0;
 border-bottom-left-radius: 4px;
```

```
}
.btn-group-vertical      >      .btn-group:not(:first-child):not(:last-
child) > .btn {
  border-radius:  0;/*  如果有嵌套的btn-group组，将其内部按钮（除第一个和最后一个
以外）的圆角清除*/
/* 如图4-17中的嵌套2按钮 */
}
.btn-group-vertical         >         .btn-group:first-child:not(:last-
child) > .btn:last-child,
.btn-group-vertical         >         .btn-group:first-child:not(:last-
child) > .dropdown-toggle {
 /* 如果有嵌套的btn-group组，将第一组（且不是最后一组）的最后一个按钮或者下拉菜单的
左下角和右下
          角的圆角清除*/
 border-bottom-right-radius: 0;
  border-bottom-left-radius:  0;/*  以便和下一组的按钮平衡，比如按钮“嵌套3”和按
钮“嵌套6”
                                        之间没有圆角 */
}
.btn-group-vertical         >         .btn-group:last-child:not(:first-
child) > .btn:first-child {
 /* 如果有嵌套的btn-group组，将最后一组（但不是第一组）的第一个按钮的左上角和右上角
的圆角清除*/
 border-top-left-radius: 0;
 border-top-right-radius: 0;
}
```

对于最新的V3.1.0版，对这个功能的使用一般都没有问题，但在V3.1.0之前的版本，使用的过程中，在两个特殊的场景下会产生bug，如图4-17所示。第一个bug是将第一张图里的“图书作品”下拉按钮修改成“图书”按钮就会显现出来了（下拉菜单和普通按钮没对齐）。第二个bug通过如下示例代码即可发现（按钮“嵌套6”的右上角的圆角没有消除）。

```
<div class=“btn-group btn-group-vertical”>
<button class=“btn btn-default” type=“button”>首页</button>
<button class=“btn btn-default” type=“button”>个人简介</button>
<button class=“btn btn-default” type=“button”>作品</button>
<div class=“btn-group”>
      <button class=“btn btn-default” type=“button”>嵌套1</button>
      <button class=“btn btn-default” type=“button”>嵌套2</button>
      <button class=“btn btn-default” type=“button”>嵌套3</button>
</div>
<div class=“btn-group”>
      <button class=“btn btn-default” type=“button”>嵌套4</button>
      <button class=“btn btn-default” type=“button”>嵌套5</button>
      <button class=“btn btn-default” type=“button”>嵌套6 </button>
</div>
</div>
```

a）简单的垂直分组按钮　　b）复杂的垂直分组按钮

图4-17　出现bug的效果

第一个Bug产生的原因在3424行，作者为相关的元素设置了100%的宽度，但只考虑了以下两种情况：

❑.btn-group-vertical样式的直接子按钮（带有.btn样式的按钮）

❑.btn-group-vertical样式中的分组（带有.btn-group样式的分组）

该样式没有考虑第三种情况，也就是分组中的按钮的宽度设置。所以，如果想让“图书”按钮也和上面的按钮对齐的话，则也要设置其宽度为100%，选择符是.btn-group-vertical > .btn-groupl > .btn，将该选择符和上述两个选择符放在一起即可（目前最新的V3.1.0已经解决了该bug）。

第二个Bug产生的原因在于V3.1.0之前的代码（如V3.0.3），作者只消除了第一组.btn-group按钮的最后一个按钮（“嵌套3”）的圆角，和最后一组.btn-group按钮的第一个按钮（“嵌套4”）的圆角，但是没有消除最后一组.btn-group按钮的最后一个按钮（“嵌套6”）的右上角的圆角。原来的代码如下：

```
.btn-group-vertical > .btn-group:first-child > .btn:last-child,
.btn-group-vertical > .btn-group:first-child > .dropdown-toggle {
 border-bottom-right-radius: 0;
 border-bottom-left-radius: 0;
}
.btn-group-vertical > .btn-group:last-child > .btn:first-child {
 border-top-right-radius: 0;
 border-top-left-radius: 0;
}
```

而新版的V3.1.0中的代码则很巧妙地利用了两个伪类not来限制选择符，比如：.btn-group: first-child: not(: last-child)和.btn-group: last-child: not(: first-child)。

4.3.6　自适应分组

.btn-group-justified样式提供了一个特殊的功能，就是在一个.btn-group容器上，如果使用了该样式，则所有的按钮都会100%充满容器元素，即容器元素宽高有多少，按钮的宽高就会变成多少，效果如图4-18所示。

图4-18　图4-18　自适应分组运行效果

```
<div class="btn-group btn-group-justified"><!—这回，又回到了附加样式的设计思想了—>
<a role="button" class="btn btn-default">左</a>
    <a role="button" class="btn btn-default">中</a>
    <a role="button" class="btn btn-default">右</a>
</div>
```

自适应分组的原理主要是利用了CSS的display: table和display: table-cell特性，这两个特性正如第一章CSS基础语法里讲的，主要是模拟实现table的功能，能够解决所有在使用绝对定位和浮动定位进行多列布局时所遇到的问题。

```
// 源码3286行
.btn-group-justified {
 display: table;                    /* 应用CSS表格样式 */
 width: 100%;              /* 100%填充 */
 table-layout: fixed;/* 边框分开，也就是相邻的2个按钮间的线是双倍border像素粗线（即2像素） */
 border-collapse: separate;
}
.btn-group-justified > .btn,
.btn-group-justified > .btn-group {
 display: table-cell;   /* 每个按钮，代表一个表格单元格样式 */
 float: none;
 width: 1%;               /* 设置一个小值，以便等分按钮，如果不设置会导致每个按钮宽度不一致，
                             如果过大，按钮多时也会变形 */
}
.btn-group-justified > .btn-group .btn {
 width: 100%;              /* 但在自适应分组内，按钮分组里的按钮100%显示 */
}
```

注意

该样式仅完美地支持a元素的按钮，因为有些浏览器对button元素的支持不太好（比如Firefox）。

4.4　按钮下拉菜单

按钮下拉菜单（Button　dropdown）是在普通的下拉菜单的基础上封装了.btn样式的效果，就类似于单击一个button按钮，然后显示隐藏的下拉菜单。

由于是结合下拉菜单组件使用的，所以本章讨论的功能需要Dropdown的Javascript插件，请读者在练习的时候确保引用了该插件的js文件或者完整的Bootstrap.js文件。

由于本章节的功能是按钮（第3章）、下拉菜单（第4章）、按钮组（第4章）这三部分内容的组合，所以大家在学习源码的时候，请参考上述相关的内容。

4.4.1 组合式下拉菜单

组合式下拉菜单其实就是普通的下拉菜单，只不过将触发元素从a元素换成了button元素（其实也可以在a元素上应用.btn样式实现）。唯一不同的是外部容器的样式从.dropdown换成了.btn-group。

通过下拉菜单章节的内容可以知道，下拉菜单需要满足一个特殊的需求，那就是必须有position: relative;样式的定义（如.dropdown样式）。而通过源码可以看出，新的.btn-group正好也满足该需求，所以简单替换成.btn-group样式即可实现的需要的效果。

示例源码如下：

```
<div class="btn-group">
   <button type="button" class="btn btn-success dropdown-toggle" data-
toggle=
           "dropdown">
    Success <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <li><a href="#">Action</a></li>
    <li><a href="#">Another action</a></li>
    <li><a href="#">Something else here</a></li>
    <li class="divider"></li>
    <li><a href="#">Separated link</a></li>
  </ul>
</div>
```

上述代码的运行效果如图4-19所示。

图4-19 组合式下拉菜单运行效果

在上述示例中，button元素内的span元素上有一个.caret样式，是表示向下箭头。具体实现源码如下：

```
// 源码2985行
.caret { /*向下箭头*/
  display: inline-block;
  width: 0;  height: 0;
  margin-left: 2px;
  vertical-align: middle;
  border-top: 4px solid;
  border-right: 4px solid transparent;
  border-left: 4px solid transparent;
```

```
}
```

如果需要向上箭头，则需要改变一下border-bottom样式。源码如下：

```
// 源码3100行
.dropup .caret,
.navbar-fixed-bottom .dropdown .caret {
  content: ””;
  border-top: 0;
  border-bottom: 4px solid;
}
```

4.4.2　分离式下拉菜单

大家在阅读上一小节讲解的组合式下拉菜单的时候，可能会发现，箭头是包含在按钮内部的，即：不管是单击按钮还是箭头，都会触发弹出事件。但是，往往有时候开发人员可能需要按钮和箭头分离的功能，即单击箭头的时候弹出菜单，而单击按钮的时候可以做其他的事情。这就要求我们实现如图4-20所示效果的菜单。

图4-20　分离式下拉菜单运行效果

通过前面学到的知识，我们可以猜想，既然单击事件是通过设置data-toggle=“dropdown”和data-target=””来触发执行的，所以完全可以让箭头单独成为一个按钮，而原来的按钮继续保持，这样两个按钮排放在一起，只需要处理按钮间的圆角即可。而在上一节讲解按钮组的时候就已经知道多个按钮间是没有圆角的了。

所以HTML代码就自然产生了。

```
    <div class=“btn-group”>
            <button type=“button” class=“btn btn-danger”>保持原来的按钮
</button>
             <button type=“button” class=“btn btn-danger dropdown-
toggle” data-
                    toggle=“dropdown”>
                <span class=“caret”></span>
        </button>
        <ul class=“dropdown-menu”>
            <li><a href=”#”>…</a></li>
            <li class=“divider”></li>
            <li><a href=”#”>…</a></li>
        </ul>
    </div>
```

4.4.3　按钮大小

不管是组合式下拉菜单还是分离式下拉菜单，都可以使用.btn原来的附加样式.btn-lg、.btn-sm、.btn-xs来对按钮大小进行控制，因为组合之间不会破坏各种附加的样式，除非又重新进行了定义。

```
    <div class="btn-group">
                    <button  class="btn  btn-success  btn-lg  dropdown-
toggle" type="button"
                data-toggle="dropdown">
              Large button <span class="caret"></span>
          </button>
          <ul class="dropdown-menu">
              …
          </ul>
    </div>
```

不同尺寸的下拉菜单运行效果如图4-21所示。

图4-21　不同尺寸的下拉菜单运行效果

4.4.4 向上弹起的下拉菜单

在有些特殊情况下，网页上的下拉菜单需要设置向上弹出的方法（暂且称为上弹项），使用的时候和普通的下拉菜单相比只要多附加一个.dropup样式即可。示例代码如下：

```
    <div class="btn-group dropup">
        <button type="button" class="btn btn-success">Dropup</button>
              <button  type="button"  class="btn  btn-success  dropdown-
toggle" data-toggle=
            "dropdown">
            <span class="caret"></span>
        </button>
        <ul class="dropdown-menu">
            <!– 具体菜单项 –>
        </ul>
    </div>
```

上述代码的运行效果如图4-22所示。

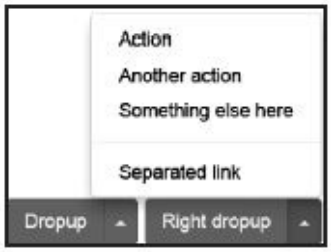


图4-22 向上弹起的下拉菜单运行效果

CSS实现方式主要是设置.dropdown-menu样式容器的bottom为100%（即靠着上边框，向上方向弹出）。

```
// 源码3106行
.dropup .dropdown-menu,
.navbar-fixed-bottom .dropdown .dropdown-menu {
  top: auto;     /* 高度自适应 */
   bottom: 100%; /* 距离dropup样式的元素底部100%的高度，即靠着上边框，向上方向弹
出 */
  margin-bottom: 1px;
}
```

4.5 输入框组

源码文件：input-groups.less

CSS文件：bootstrap.css 3305行以后

4.5.1 基本用法

有些时候，我们需要将文本输入框（Input group）和文字或者小icon组合在一起进行显示（我们称之为addon）。例如经常需要如图4-23所示效果的表单输入框。

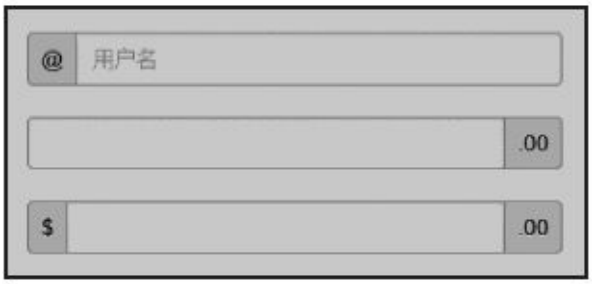


图4-23 输入框组运行效果

Bootstrap在3.x版将其独立出来了，成为了一个单独的输入框组（Input group）。使用方式也非常简单，只需要在容器上应用.input-group样式，然后对需要在input前后显示的个性元素上应用.input-group-addon样式即可。

```
<div class="input-group">
    <span class="input-group-addon">$</span>
    <input type="text" class="form-control"><!— 这里的input, 必须使用form-control
            样式才行 —>
    <span class="input-group-addon">.00</span>
</div>
```

具体实现方式有几个重要的知识点，步骤如下：

步骤1 将各元素无缝地拼接在一起，即设置margin为0。

步骤2 将各元素设置为等高显示，即设置display为table-cell。

步骤3 设置addon元素的显示方式。

步骤4 不管前后有没有addon或有几个，按照如下规则进行圆角处理。

❑设置input-group元素里的第一个input或input-group-addon元素的左上角和左下角为圆角

❑设置input-group元素里的最后一个input或input-group-addon元素的右上角和右下角为圆角

其中步骤1~步骤3的主要源码如下：

```
// 源码3305行
.input-group {
  position: relative;   /* 相对定位 */
```

```
  display: table;                /* 表格布局 */
  border-collapse: separate;
}
.input-group[class*=“col-“] {    /* input-group如果也同时应用了col样式，则取消浮动和左右
                               padding */
  float: none;
  padding-right: 0;
  padding-left: 0;
}
.input-group .form-control {
  float: left;           /* 左浮动，防止由于IE9的占位符和select元素的箭头而导致变形*/
   width: 100%;             /* input-group里的form-control组件的宽度都设为100% */
  margin-bottom: 0;
}
/* 此处代码从3982行开始 */
.input-group-addon,
.input-group-btn {
  width: 1%;             /* 设置最小值，以便表格模式进行等分 */
  white-space: nowrap;
  vertical-align: middle;        /* 垂直居中 */
}
.input-group-addon {    /* 设置addon*/
  padding: 6px 12px;
  font-size: 14px;
  font-weight: normal;
  line-height: 1;
  color: #555;
  text-align: center;
  background-color: #eee;
  border: 1px solid #ccc;
   border-radius: 4px;    /* 圆角设置（结合上面代码，可判断出是第一个和最后一个）*/
}
```

关于第4步圆角的处理，可能认为很简单，不就是简单地把第一个和最后一个元素的圆角设置一下就行了，即先把所有的圆角都去除，然后再恢复第一个和最后一个元素的圆角设置，就可以了。

是的，一般情况下是没问题的，但如果内部有多个input，多个addon，可能就不灵了。在2.x版本时作者也是这么实现的，可是带来了很多问题，后续在3.x版又进行了重构。先来看看老版的代码，和我们想象的一样。

```
.input-group-addon,.input-group-btn,.input-group input {
  display: table-cell;
  margin: 0;
  border-radius: 0;                        /* 取消设置圆角 */
}
…
.input-group input:first-child,
.input-group-addon:first-child {
```

```
/*   设置input-group里第一个input或者input-group-addon元素的左上方和左下方的圆角 */
  border-bottom-left-radius: 4px;
  border-top-left-radius: 4px;
}
…
.input-group input:last-child,
.input-group-addon:last-child {
/*  设置input-group里最后一个input或者input-group-addon元素的右上方和右下方的圆角 */
  border-top-right-radius: 4px;
border-bottom-right-radius: 4px;
}
```

新版的实现方式很精妙，我们分析的时候，一定要在纸上多画几个input或者addon，因为这里面牵涉了很多所有input（但不包括第一个和最后一个）这样的设置，多画几个容易理解。另外只看一遍也不容易看懂，希望大家多画几个场景，多读几遍下面的源码（尽力做了注释）。

```
// 源码3407行
/* input-group容器内的第一个form-control样式的组件 */
.input-group .form-control:first-child,
/* 第一个addon组件 */
.input-group-addon:first-child,
/* 第一个.input-group-btn容器里的按钮 */
.input-group-btn:first-child > .btn,
/* 第一个.input-group-btn容器里的所有.btn-group组里的按钮 */
.input-group-btn:first-child > .btn-group > .btn,
/* 第一个.input-group-btn容器里的下拉菜单 */
.input-group-btn:first-child > .dropdown-toggle,
/* 最后一个.input-group-btn容器里的按钮（不包括最后一个按钮以及下拉菜单） */
.input-group-btn:last-child    >    .btn:not(:last-child):not(.dropdown-toggle),
/*     最后一个.input-group-btn容器里的所有.btn-group组（但不是最后一个）中的按钮 */
.input-group-btn:last-child > .btn-group:not(:last-child) > .btn {
  border-top-right-radius: 0;            /*清除右上角圆角*/
  border-bottom-right-radius: 0;         /*清除右下角圆角*/
}
.input-group-addon:first-child {
  border-right: 0; /* 第一个addon组件的右边框设置为0，以便和右边的input靠近 */
}
/* input-group容器内的最后一个form-control样式的组件 */
.input-group .form-control:last-child,
/* 最后一个addon组件 */
.input-group-addon:last-child,
/* 最后一个.input-group-btn容器里的按钮 */
.input-group-btn:last-child > .btn,
/* 最后一个.input-group-btn容器里的.btn-group子容器里的按钮 */
.input-group-btn:last-child > .btn-group > .btn,
/* 最后一个.input-group-btn容器里的下拉菜单 */
.input-group-btn:last-child > .dropdown-toggle,
/* 第一个.input-group-btn容器里的按钮（不包括第一个按钮） */
.input-group-btn:first-child > .btn:not(:first-child),
/* 第一个.input-group-btn容器里的所有.btn-group组（但不是第一个）中的按钮 */
```

```
.input-group-btn:first-child > .btn-group:not(:first-child) > .btn {
  border-top-left-radius: 0;             /*清除左上角圆角*/
  border-bottom-left-radius: 0;          /*清除左下角圆角*/
}
```

注意

❑请尽量避免在select元素上使用该功能，因为WebKit浏览器不完全支持input-group组件的特性。

❑不要直接将.input-group和.form-group混合使用，因为.input-group是一个独立的组件。

❑上述代码有很多关于.input-group-btn样式的设置，将在4.5.4小节进行讲解。

4.5.2 尺寸大小设置

和按钮组的.btn-group-lg样式一样，Bootstrap在.input-group-addon样式容器上，也可以通过应用类似的样式达到设置尺寸的目的，并提供了大型和小型样式（.input-group-lg和.input-group-sm），未提供超小型的样式（也许作者认为不需要）。使用方式如下：

```
<div class="input-group input-group-lg">
    <span class="input-group-addon">@</span>
    <input type="text" class="form-control" placeholder="Username">
</div>
<div class="input-group">
    <span class="input-group-addon">@</span>
    <input type="text" class="form-control" placeholder="Username">
</div>
<div class="input-group input-group-sm">
    <span class="input-group-addon">@</span>
    <input type="text" class="form-control" placeholder="Username">
</div>
```

上述代码的运行效果如图4-24所示。

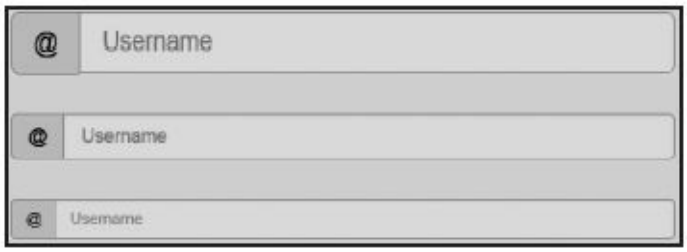


图4-24 不同尺寸的输入框组运行效果

尺寸实现的源码和其他组件的尺寸设置原理一样，非常简单，就不讲述了，直接看下面的源码即可。

```
/* 源码 3320 行 */
.input-group-lg > .form-control,
.input-group-lg > .input-group-addon,
.input-group-lg > .input-group-btn > .btn {
  height: 46px;
  padding: 10px 16px;
  font-size: 18px;
  line-height: 1.33;
  border-radius: 6px;
}
.input-group-sm > .form-control,
.input-group-sm > .input-group-addon,
.input-group-sm > .input-group-btn > .btn {
  height: 30px;
  padding: 5px 10px;
  font-size: 12px;
  line-height: 1.5;
  border-radius: 3px;
```

```
}
```

注意

上述尺寸源码的下面，还有如下样式的设置：select.input-group-lg和textarea.input-group-lg，也就是说在select元素或者textarea元素上应用.input-group-lg样式的设置。但是好像根本就没这种需求。

上述方式有一个弊端，就是该容器元素内的addon和input元素都统一设置成了相同的大小，如果不是这种需求的话，就没办法满足了。所以Bootstrap在实现上述方式以外，还又提供了另外一种方式进行设置，即重用input元素的.input-lg和.input-sm样式。因为input本来就可以用，所以只需要在.input-group-addon样式上设置一下就行了。查看一下源码，的确如此。

```
// 源码3393行
.input-group-addon.input-sm {  /* 在addon上应用input-sm样式时的设置 */
  padding: 5px 10px;
  font-size: 12px;
  border-radius: 3px;
}
.input-group-addon.input-lg {  /* 在addon上应用input-lg样式时的设置 */
  padding: 10px 16px;
  font-size: 18px;
  border-radius: 6px;
}
```

使用方式也就不用说了，像在input元素应用该样式的方式一样。示例如下：

```
<div class=“input-group”>
  <span class=“input-group-addon input-lg”>@</span>
  <input type=“text” class=“input-large” placeholder=”用户名”>
</div>
<div class=“input-group”>
  <span class=“input-group-addon input-sm”>@</span>
  <input type=“text” class=“input-small” placeholder=”用户名”>
</div>
```

另外，由于上述两个特殊设置是在.input-group-lg样式之后设置的，所以两种样式即便是混用在一起，对里面的单个元素个性化设置也没有影响。当然，如果要全部统一大小，那就需要把里面的个性化设置样式去除才行，因为不能既让大的不影响小的，又让小的不影响大的。

4.5.3　复选框与单选框作为addon

在.input-group-addon样式里，不仅可以放置label和icon，也可以放置复选框（checkbox）和单选框（radio），以达到如图4-25所示的效果。

图4-25　单选框和复选框的运行效果

使用方式也非常简单，只需要在.input-group-addon样式内放置带有checkbox和radio类型的input即可。示例如下：

```
<div class=“row”>
    <div class=“col-lg-3”>
        <div class=“input-group”>
                <span class=“input-group-addon”><input type=“checkbox”>
</span>
            <input type=“text” class=“form-control”>
        </div>
    </div>
    <div class=“col-lg-3”>
        <div class=“input-group”>
            <span class=“input-group-addon”><input type=“radio”></span>
            <input type=“text” class=“form-control”>
        </div>
    </div>
</div>
```

由于.input-group-addon样式在设置的时候没有限制元素，所以对上述两个类型的input，作者只是对其margin-top值进行了消除，以方便对齐。源码如下：

```
// 源码3403行
.input-group-addon input[type=“radio”],
.input-group-addon input[type=“checkbox”] {
  margin-top: 0; /* 如果addon是单选框或复选框，则取消margin-top */
}
```

4.5.4　按钮作为addon

输入框组上的addon支持复选框和单选框，是不是也支持普通的button以addon的形式出现呢？Bootstrap开发者当然也考虑到这一点了，但是由前面的章节我们知道，.btn按钮样式上定义了各种各样的样式（比如大小、颜色、内外边距等），其不像checkbox、radio、label等直接放到.input-group-addon样式里就行的，所以为了避免冲突，作者为.btn样式又单独设置了一个.input-group-btn样式（基础用法里提到过），用其替换.input-group-addon作为新的addon容器。示例用法如下：

```
<div class="row">
    <div class="col-lg-6">
        <div class="input-group">
                <span class="input-group-btn"><button class="btn btn-
default" type=
              "button">Go!</button></span>
            <input type="text" class="form-control">
        </div>
    </div>
    <div class="col-lg-6">
        <div class="input-group">
            <input type="text" class="form-control">
                <span class="input-group-btn"><button class="btn btn-
default" type=
             "button">Go!</button></span>
        </div>
    </div>
</div>
```

上述代码的运行效果如图4-26所示。

图4-26　按钮组作为addon的运行效果

就如基本用法里源码分析的一样，.input-group-btn样式共用了.input-group-addon样式的部分设置，然后又单独处理了各种内外边距。源码如下：

```
// 源码3433行
.input-group-btn {
  position: relative;   /*相对定位*/
  font-size: 0;
  white-space: nowrap;
}
.input-group-btn > .btn {
  position: relative;   /*所有.input-group-btn容器内的按钮都相对定位*/
}
.input-group-btn > .btn + .btn {
  margin-left: -1px;              /*所有.input-group-btn容器内的按钮之间的左
```

```
外边距都有-1像素，以便靠近*/
}
.input-group-btn > .btn:hover,
.input-group-btn > .btn:focus,
.input-group-btn > .btn:active {
  z-index: 2;            /*高亮时，加大z-index*/
}
.input-group-btn:first-child > .btn,
.input-group-btn:first-child > .btn-group {
   margin-right: -1px;    /*第一个.input-group-btn容器内的按钮或按钮组，右外边
距设置为
                              -1，以便都可以靠近*/
}
.input-group-btn:last-child > .btn,
.input-group-btn:last-child > .btn-group {
   margin-left: -1px; /*最后一个.input-group-btn容器内的按钮或按钮组，左外边距
设置为-1，
                              以便都可以靠近*/
}
```

4.5.5　下拉菜单按钮作为addon

很自然，能支持普通按钮，也就能支持下拉菜单按钮，不需要额外的样式支持，只需要在普通的.btn按钮上应用一个data-toggle=“dropdown”属性即可。示例代码如下：

```
<div class=“row”>
    <div class=“col-lg-2”>
        <div class=“input-group”>
            <div class=“input-group-btn ”>
                 <button type=“button” class=“btn btn-default dropdown-
toggle”
                         data-toggle=“dropdown”>
                    Action<span class=“caret”></span>
                </button>
                <ul class=“dropdown-menu”>…</ul>
            </div>
            <input type=“text” class=“form-control”>
        </div>
    </div>
    <div class=“col-lg-2”>
        <div class=“input-group”>
            <input type=“text” class=“form-control”>
            <div class=“input-group-btn ”>
                 <button type=“button” class=“btn btn-default dropdown-
toggle”
                         data-toggle=“dropdown”>
                    Action<span class=“caret”></span>
                </button>
                 <ul class=“dropdown-menu pull-right”><!— 这里使用了pull-
right,
                         以便可以右对齐 —></ul>
            </div<
        </div<
    </div<
</div<
```

上述代码的运行效果如图4-27所示。

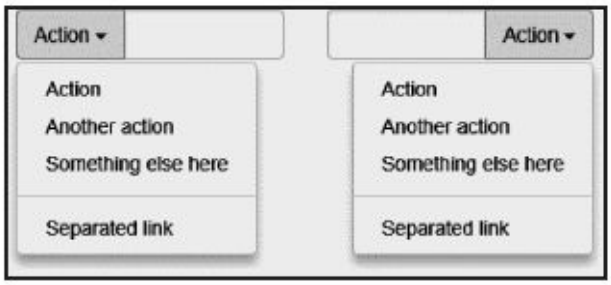


图4-27　下拉菜单作为addon的运行效果

4.5.6 分段按钮作为addon

在基本用法小节的源码分析里，我们看到了Bootstrap对.input-group-btn样式容器内的多个.btn样式按钮也都做了处理，而且在按钮作为addon小节的源码分析里也对多个按钮的外边距进行了消除设置，所以也就是说，通过这两个方面的设置，在.input-group-btn样式里，也是可以放置多个按钮的，并且中间按钮的圆角也都会消除的。示例代码如下：

```
<div class="row">
    <div class="col-lg-4">
        <div class="input-group">
            <div class="input-group-btn">
                    <button class="btn btn-default" type="button">按钮
A</button>
           <button data-toggle="dropdown" class="btn btn-default
                            dropdown-toggle" type="button">
                          <span class="caret"></span><span class="sr-
only">Toggle
                            Dropdown</span>
                </button>
                    <button class="btn btn-default" type="button">按钮
B</button>
                    <button class="btn btn-default" type="button">按钮
C</button>
                <ul role="menu" class="dropdown-menu">…</ul>
            </div>
            <input type="text" class="form-control">
        </div>
    </div>
    <div class="col-lg-2">
        <div class="input-group">
            <input type="text" class="form-control">
            <div class="input-group-btn open">
                                          <button   class="btn   btn-
default" type="button">Action</button>
                <button data-toggle="dropdown" class="btn btn-default
                            dropdown-toggle" type="button">
                          <span class="caret"></span><span class="sr-
only">Toggle
                            Dropdown</span>
                </button>
                <ul role="menu" class="dropdown-menu pull-right">…</ul>
            </div>
        </div>
    </div>
</div>
```

上述代码的运行效果如图4-28所示。

图4-28　分段按钮作为addon的运行效果

只是有一点需要注意，所有的按钮都必须放置在.input-group-btn样式容器内。

4.6 导航

源码文件：navs.less

CSS文件：bootstrap.css 4070行以后

导航（Nav）是一个网站最重要的组成部分，可以便于用户查找网站所提供的各项功能服务。导航的制作方法五花八门，本节我们要讲述最常用的导航功能。本节提供的nav样式和btn样式一样，可以通过应用多种附加样式实现多种特效。

和默认的.btn样式不同，默认的.nav样式不提供默认的导航，必须通过附加另外一个样式才行，比如.nav-tabs样式表示选项卡导航。

.nav基础样式主要是设置布局方式（相对）、块级显示、padding、active、disabled状态下的颜色等基础设置。主要代码如下：

```
// 源码3457行
.nav {
  padding-left: 0;
  margin-bottom: 0;
  list-style: none;              /*消除list圆点*/
}
.nav > li {
  position: relative;            /*所有的菜单项都是相对定位*/
  display: block;                        /* 块级显示*/
}
.nav > li > a {
  position: relative;            /* a链接相对定位*/
  display: block;                        /* 块级显示*/
  padding: 10px 15px;            /* 外边距保留稍微大一些*/
}
.nav > li > a:hover,
.nav > li > a:focus {            /* 移动或焦点时链接的显示效果*/
  text-decoration: none;
  background-color: #eee;                /* 移动或焦点时，背景色变为灰色*/
}
.nav > li.disabled > a {
  color: #999;                   /* li上禁用时的链接颜色*/
}
.nav > li.disabled > a:hover,
.nav > li.disabled > a:focus {   /* li上禁用时，移动到链接上时的各种处理*/
  color: #999;                   /* 颜色变灰*/
  text-decoration: none;
  cursor: not-allowed;
  background-color: transparent;
}
/* 省略部分样式*/
.nav > li > a > img {
  max-width: none;               /* 如果a链接里是img图片，则不设置最大宽度*/
}
```

4.6.1 选项卡导航

选项卡导航是最常用的一种导航方式，尤其是在多内容编辑的时候，需要通过选项卡进行分组显示，其效果如图4-29所示。

图4-29 选项卡运行效果

上述效果的第一个菜单“主页”是当前高亮菜单项（.active样式），“作品”菜单项是禁用状态（.disabled样式）。实现方式如下：

```
<ul class=“nav nav-tabs ”>
    <li class=“active”><a href=”#”>主页</a></li>
    <li><a href=”#”>个人信息</a></li>
    <li class=“disabled”><a href=”#”>作品</a></li>
    <li><a href=”#”>图书</a></li>
</ul>
```

其实现原理主要是让每个li项按照块级元素显示，然后定义非高亮菜单的样式和鼠标触发行为，最后定义高亮菜单项的样式和鼠标行为。代码如下：

```
// 源码3501行
.nav-tabs {
  border-bottom: 1px solid #ddd;          /* 所有菜单下都有横线，代表整体导航为水平方向 */
}
.nav-tabs > li {
  float: left;                    /* 所有菜单都左浮动 */
  margin-bottom: -1px;
}
.nav-tabs > li > a {
  margin-right: 2px;
  line-height: 1.428571429;
  border: 1px solid transparent;          /* 透明边框 */
  border-radius: 4px 4px 0 0;    /* 每个菜单项上面都有圆角 */
}
.nav-tabs > li > a:hover {
   border-color: #eee #eee #ddd; /* 非高亮状态下，菜单项在鼠标触发时候的背景颜色 */
}
.nav-tabs > li.active > a,
.nav-tabs > li.active > a:hover,
.nav-tabs > li.active > a:focus {
  color: #555;
  cursor: default;
  background-color: #fff;         /* 高亮状态下，背景颜色固定为白色 */
  border: 1px solid #ddd;
  border-bottom-color: transparent;       /* 高亮菜单项下的横线设置为透明，即不
```

```
显示横线 */
}
```

注意，如果需要处理单击菜单，以便立即显示所对应tab的内容区的话，需要配合JS组件里的Tab组件。

4.6.2　胶囊式选项卡导航

同样是上述示例代码，将.nav-tabs样式替换为.nav-pills样式，即可变换成完全不同的效果，当前.active的菜单会进行背景色高亮显示。示例代码如下：

```
<ul class=“nav nav-pills ”>
    <li class=“active”><a href=”#”>主页</a></li>
    <li><a href=”#”>个人信息</a></li>
    <li class=“disabled”><a href=”#”>作品</a></li>
    <li><a href=”#”>图书</a></li>
</ul>
```

运行效果如图4-30所示。

图4-30　胶囊式选项卡运行效果

这种风格的CSS设置比tab稍简单，只要加大每个li元素的圆角设置和当前li元素的文字颜色及背景色即可。

```
// 源码3570行
.nav-pills > li {
  float: left;                    /* 左浮动 */
}
.nav-pills > li > a {
  border-radius: 4px;                     /* 圆角设置 */
}
.nav-pills > li + li {
  margin-left: 2px;                       /* 加大左外边距 */
}
.nav-pills > li.active > a,
.nav-pills > li.active > a:hover,
.nav-pills > li.active > a:focus {        /* 高亮菜单设置文字颜色和背景色 */
  color: #fff;                            /* 文字为白色*/
  background-color: #428bca;              /* 背景为蓝色*/
}
```

4.6.3　堆叠式导航

所谓堆叠式导航，是将普通的导航菜单垂直堆放而已。比如，高亮导航的堆叠效果如图4-31所示。

图4-31　堆叠式导航运行效果

应用该功能的时候，只需要在nav-pills的基础上再附加一个.nav-stacked样式即可。示例如下：

```
<ul class="nav nav-pills nav-stacked">
    <li class="active"><a href="#">主页</a></li>
    <li><a href="#">个人信息</a></li>
    <li><a href="#">作品</a></li>
    <li><a href="#">图书</a></li>
</ul>
```

堆叠式导航的原理主要是，去除.nav-pills样式的浮动显示（即默认不设置浮动，让其垂直摆放），然后设置上下两个li元素的间距即可实现图4-31的效果。

```
// 源码3585行
.nav-stacked > li {
  float: none;                          /* 不设置浮动，垂直摆放 */
}
.nav-stacked > li + li {
  margin-top: 2px;                      /* 设置两个li元素上下之间的top值 */
  margin-left: 0;
}
```

在分析源码的时候，笔者还发现一个.nav-divider样式，但官方并没有讲解过该样式。源码如下：

```
// 源码3492行
.nav .nav-divider {                     /* 分隔符，只适用于堆叠式导航*/
  height: 1px;
  margin: 9px 0;
  overflow: hidden;
  background-color: #e5e5e5;
```

```
}
```

于是猜想，该样式是不是和.dropdown-menu样式里的.divider样式一样，是作为分隔线来用的。于是和.nav-stacked样式一起试用一下，的确如此。运行示例如下：

```
<ul class="nav nav-pills nav-stacked">
    <li class="active"><a href="#">主页</a></li>
    <li><a href="#">个人信息</a></li>
    <li><a href="#">作品</a></li>
    <li class="nav-divider"></li>
    <li><a href="#">图书</a></li>
</ul>
```

上述代码的运行效果如图4-32所示。

图4-32　堆叠式导航中的分隔线运行效果

可能大家要问，如果在.nav-tabs样式上附加.nav-stacked样式，会不会也显示垂直的风格？答案是：

老版本v2.x系列可以，3.x新版本不可以。这主要是tab本身就是选项卡，如果垂直可能就不太好看，于是作者就放弃了这种支持。不过如果想用，那么将v2.x系列的源码拿过来引用一下就可以了。源码如下：

```
.nav-tabs.nav-stacked {
    border-bottom: 0 none;
}
.nav-tabs.nav-stacked > li > a {
    border: 1px solid #DDDDDD;
    border-radius: 0 0 0 0;                /* 默认先取消所有的圆角 */
    margin: 0;
}
.nav-tabs.nav-stacked > li:first-child > a {
    border-top-left-radius: 4px;           /* 第一个菜单项设置左上角和右上角的圆角 */
    border-top-right-radius: 4px;
}
.nav-tabs.nav-stacked > li:last-child > a {
    border-bottom-left-radius: 4px;        /* 最后一个菜单项设置左下角和右下角的
```

```
圆角 */
    border-bottom-right-radius: 4px;
}
.nav-tabs.nav-stacked    >    li    >    a:hover,    .nav-tabs.nav-
stacked > li > a:focus {
    border-color: #DDDDDD;
    z-index: 2;
}
```

运行以后，则又支持了之前的效果了，如图4-33所示。虽然不太好看，但却可以满足一些特殊需求（其实大家可以使用按钮组里的垂直分组，也可以实现类似的效果）。

图4-33　垂直风格的选项卡运行效果

注意

上述效果其实就是4.18节讨论的列表组，只不过删除了两个li之间的间隔。

4.6.4　自适应导航

对于自适应导航，同按钮组的.btn-group-justified样式一样，可以将li元素充满整个父容器，其样式为. nav-justified。使用时只需要在.nav-tabs或.nav-pills样式基础上再附加上它就可以了。示例代码如下：

```
<ul class=“nav nav-tabs nav-justified”>
  …
</ul>
<ul class=“nav nav-pills nav-justified”>
  …
</ul>
```

上述代码的运行效果如图4-34所示。

图4-34　自适应导航运行效果

实现方式也一如既往，在宽度为100%的基础上，设置每个元素的display风格是table-cell。源码如下：

```
// 源码3592行
.nav-justified {
  width: 100%;           /* 宽度充满父元素 */
}
.nav-justified > li {
  float: none;           /* 取消浮动 */
}
.nav-justified > li > a {
  margin-bottom: 5px;
  text-align: center;    /* 文本居中 */
}
.nav-justified > .dropdown .dropdown-menu {
  top: auto;
  left: auto;
}
@media (min-width: 768px) {
  .nav-justified > li {
    display: table-cell;          /* 表格风格显示，一个按钮代表一个表格单元格样式 */
    width: 1%;
  }
  .nav-justified > li > a {
    margin-bottom: 0;
  }
```

```
}
```

上述代码中有一句@media (min-width: 768px)媒体查询，也就说只有在大于768像素的浏览器上才有这样的显示效果，小于该像素的话，将分别显示两种堆叠式导航，如图4-35所示。

图4-35　自适应导航在小分辨率下的运行效果

由于.nav-tabs样式的导航带有灰色边框，所以在进行处理的时候（也就是同时应用.nav-tabs.nav-justified样式的时候），作者为了应对这些边框和圆角又单独设置了很多内容。由于内容很简单，大家可以自行参考源代码的3541～3569行。

另外，在Bootstrap.css源码里，还定义了一种.nav-tabs-justified样式，看名称可能是又对选项卡导航专门定义了一种样式，可是笔者试了多种使用方式，显示都很凌乱，而且作者的官方网站也没有提及这个样式，所以很有可能是作者还未完善的一种样式吧。

4.6.5 禁用链接

导航里的一个li元素菜单项需要禁用的话，可以和button元素一样，应用disabled样式即可。同样，这个样式仅仅是实现了禁用的外观效果，还不能真正防止单击以后的行为，所以还需要用JavaScript代码对单击事件进行特殊处理（阻止冒泡，e.preventDefault()、e.stopPropagation()、return false之类的代码）。示例代码如下：

```
<ul class=“nav nav-tabs”>
  …
  <li class=“disabled”><a href=”#”>Disabled link</a></li>
  …
</ul>
```

4.6.6　二级导航实现

一般网站可能都会有二级导航菜单，而前面小节讲解的都是一级菜单。那如何实现二级导航菜单呢？

可能会有部分读者会发现，将普通导航里的li元素作为父元素容器，内部包含Dropdown下拉菜单的内容是否就可以实现类似的功能？我们来试试在nav-tabs上是否可以实现。示例如下：

```
<ul class="nav nav-tabs">
    <li class="active"><a href="#">Home</a></li>
    <li><a href="#">Profile</a></li>
    <li><a href="#">Messages</a></li>
    <li class="dropdown">
                                    <a    class="dropdown-toggle"    data-
toggle="dropdown" href="#">Dropdown <span
           class="caret"></span></a>
        <ul class="dropdown-menu">
            <li><a tabindex="-1" href="#">二级菜单1</a></li>
            <li><a tabindex="-1" href="#">二级菜单2</a></li>
        </ul>
    </li>
</ul>
```

上述代码的运行效果如图4-36所示。

这正是我们想要的效果，没有添加新的CSS样式，只是在最后一个li元素上应用.dropdown样式，内部元素按照下拉菜单的规则布局HTML代码，即可将两种效果组合在一起。同样的道理，如果想做三级菜单，那在二级菜单里的li元素上加.dropdown-submenu样式即可（.dropdown-submenu样式在3.x版已经被删除，请参考下拉菜单里的多级嵌套方式）。

同样的道理，在.nav-pills样式上也可以实现类似的效果，如图4-37所示。

图4-36　二级导航运行效果

图4-37　胶囊式导航中的二级导航运行效果

比较一下图4-36和图4-37，发现在下拉菜单的显示上有以下两个差异：

❑.nav-tabs样式的Dropdown按钮在单击的时候有边框，而.nav-pills样式的没有。

❑.nav-tabs样式的下拉菜单弹出的菜单，左上角和右上角没有圆角，而.nav-pills样式则全部4个圆角都有。

通过Firebug跟踪，我们发现了单独的设置，其差异分别如下：

在单击下拉菜单的时候，JS插件给所单击元素应用了我们前面讲解过的.open样式，而Bootstrap开发者又对.nav下的.open样式单独进行了设置，所以就出现了边框。源码如下：

```
// 源码3468行
.nav .open > a,
.nav .open > a:hover,
.nav .open > a:focus {
  background-color: #eee;
  border-color: #428bca;
}
```

由于.nav-tabs样式设置了边框线，而.nav-pills样式没有设置边框线，这就是出现问题1的原因。

而第二个问题，则是由于作者又单独为.nav-tabs元素里的.dropdown-menu样式消除了两个圆角和上外边距，而对.nav-justified样式元素里的下拉菜单则没有特殊设置什么内容。源码如下：

```
// 源码3602行
.nav-justified > .dropdown .dropdown-menu {
  top: auto;                    /* 默认上边缘计算 */
  left: auto;                   /* 默认左边缘计算 */
}
/*此处代码从3644行开始*/
.nav-tabs .dropdown-menu {      /* 如果nav-tabs导航中有下拉菜单，则进行如下设置*/
  margin-top: -1px;             /* 消除外边距，让ul元素和tab按钮靠近*/
  border-top-left-radius: 0;    /* 去除左上角的圆角设置*/
  border-top-right-radius: 0;   /* 去除右上角的圆角设置*/
}
```

在.nav-tabs和.nav-pills样式中，如果在附加了.nav-stacked样式的情况下实现二级导航，虽然能实现，但是垂直方式的显示风格很不友好，一般网站不会使用。读者如果有兴趣，可以自行完善之。

注意

无论是.nav-tabs还是.nav-pills样式的导航，都可以通过再附加一个.pull-left或者.pull-right样式，来控制整个导航向左浮动或向右浮动。

4.7　导航条

源码文件：navbar.less

CSS文件：bootstrap.css 3649行以后

导航条（Navbar）和上一节导航的区别就是多了一个“条”，也就是菜单项背后有一个可以看得到的背景条，以方便访问用户明显地识别出这是一条导航块，里面的内容可以选择，效果如图4-38所示。

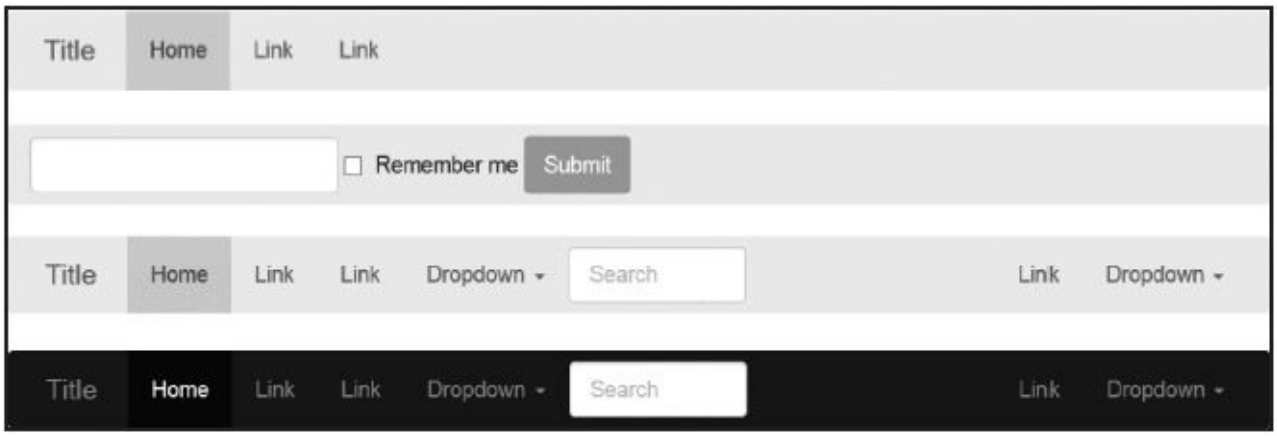


图4-38　导航条运行效果

4.7.1　基础导航条

基础导航条是在普通导航的基础上进行改进实现的，但实现原理复杂得多。我们先来看一个普通的例子，首先在普通导航的ul元素上应用.navbar-nav样式，然后在外部父元素容器上应用.navbar样式以及.navbar-default样式即可实现。示例代码如下：

```
<nav class="navbar navbar-default" role="navigation">
    <div class="navbar-header">
        <a class="navbar-brand" href="#">Brand</a>
    </div>
    <ul class="nav navbar-nav">
        <li class="active"><a href="#">active</a></li>
        <li><a href="#">Link</a></li>
        <li class="disabled"><a href="#">disabled</a></li>
        <li><a href="#">Link</a></li>
    </ul>
</nav>
```

运行效果如图4-39所示。上述代码里有一个.navbar-brand样式的链接，表示该元素是导航条的名称，起到提醒的目的。

图4-39　基础导航条运行效果

注意

要增强可访问性，一定要给每个导航条加上role="navigation"。

在讲解源码之前，我们先大概说一下整个navbar的原理。在整个设计中，.nav、.navbar、

.navbar-nav等是控制大小、内外边距、行距等方面的样式，而颜色则是由.navbar-default和.navbar-inverse这两大风格所控制（本节会讲述.navbar-default风格样式，而.navbar-inverse会放到另一小节讲）。另外由于Bootstrap新版是一个移动先行的前端框架，所以该导航条不仅支持普通的宽屏浏览器，也支持手机的窄屏浏览器，也就是说上述所有的样式，在窄屏和宽屏下都有不同的设置（通过媒体查询实现的）。所以在讲解的时候，先讲解普通的宽屏，然后在另外一个小节"响应式导航条"里再单独讲解窄屏下的实现方式。

.navbar样式的主要功能是设置左右padding、圆角等，但不设置与颜色相关的内容，因为与颜色相关的内容是由.navbar-default样式来控制的。.navbar样式主要源码如下：

```
// 源码3649行
```

```
.navbar {
  position: relative;                    /* 相对定位 */
  min-height: 50px;                      /* 最小高度 */
  margin-bottom: 20px;                   /* 底部的外边距 */
  border: 1px solid transparent;         /* 边框透明 */
}
@media (min-width: 768px) {
  .navbar {    border-radius: 4px;       /* 宽屏时，才设置圆角 */    }
}
@media (min-width: 768px) {
  .navbar-header {    float: left;       /* 宽屏时，header为左浮动 */  }
}
```

.navbar-nav样式，是在原有.nav样式的基础上（nav和navbar-nav要一起使用），重新调整设置了菜单项链接的浮动和内外边距的设置（不包括颜色，颜色依然是由.navbar-default和.navbar-inverse两大样式设置）。下面是.navbar-nav样式的源码：

```
// 源码3822行
@media (min-width: 768px) {              /*宽屏情况下*/
  .navbar-nav {
    float: left;                                  /* 左浮动*/
    margin: 0;
  }
  .navbar-nav > li {
    float: left;                                  /*菜单项左浮动*/
  }
  .navbar-nav > li > a {                          /*加大上下方向的内边距*/
    padding-top: 15px;
    padding-bottom: 15px;
  }
  .navbar-nav.navbar-right:last-child {
     margin-right: -15px;                             /*向右对齐时最后一个子元
素，设置负的margin值*/
  }
}
```

而作为提醒功能的navbar-brand样式，则主要是加大了字体设置，并控制最大宽度显示。

```
// 源码3746行
.navbar-brand {
  float: left;                           /*左浮动*/
  height: 20px;
  padding: 15px 15px;
  font-size: 18px;
  line-height: 20px;                     /*增大行间距*/
}
.navbar-brand:hover,
.navbar-brand:focus {
  text-decoration: none;
}
```

基础的导航条功能，不仅能看到各种各样的菜单定义，还包括当前

活动的菜单（active样式）、当前已禁用的菜单（disabled样式），下拉菜单也可以放在里面，如图4-40所示。这是一个比较复杂的基础导航条，我们就这个效果，来一步一步分析.navbar-default样式风格是如何定义的。

图4-40　基础导航条的综合应用

完整的.navbar-default风格的源码和注释如下，请大家一定要仔细看注释，并对照查看示例效果。

```
// 源码3944行
.navbar-default {                                    /*设置背景颜色和边框颜色*/
  background-color: #f8f8f8;
  border-color: #e7e7e7;
}
.navbar-default .navbar-brand {  color: #777;  /*设置navbar-brand的文本颜色*/}
.navbar-default .navbar-brand:hover,
.navbar-default .navbar-brand:focus {
  color: #5e5e5e;               /*navbar-brand移动或焦点时的文本颜色和背景颜色*/
  background-color: transparent;
}
.navbar-default .navbar-text {  color: #777;  /*设置navbar-text的文本颜色，下节讲*/}
.navbar-default .navbar-nav > li > a {  color: #777;  /*设置菜单项链接的文本颜色*/}
.navbar-default .navbar-nav > li > a:hover,
.navbar-default .navbar-nav > li > a:focus {
  color: #333;          /*设置菜单项链接在移动或焦点时的文本颜色*/
  background-color: transparent;
}
.navbar-default .navbar-nav > .active > a,
.navbar-default .navbar-nav > .active > a:hover,
.navbar-default .navbar-nav > .active > a:focus {
  color: #555;          /*设置当前活动的菜单项链接的文本颜色和背景颜色*/
  background-color: #e7e7e7;
}
.navbar-default .navbar-nav > .disabled > a,
.navbar-default .navbar-nav > .disabled > a:hover,
.navbar-default .navbar-nav > .disabled > a:focus {
  color: #ccc;          /*设置当前已禁用的菜单项链接的文本颜色和背景颜色*/
  background-color: transparent;
}
```

```
.navbar-default .navbar-toggle {  border-color: #ddd;  /*响应式按钮的边框颜色,
                                                                在4.7.6一节会讲*/}
.navbar-default .navbar-toggle:hover,
.navbar-default .navbar-toggle:focus {
  background-color: #ddd;       /*响应式按钮在移动或者焦点时的背景颜色*/
}
.navbar-default .navbar-toggle .icon-bar {  background-color: #888;
                                /*响应式按钮里的icon的背景颜色*/}
.navbar-default .navbar-collapse,
.navbar-default .navbar-form {
  border-color: #e7e7e7;        /*折叠边框或者表单样式的边框颜色*/
}
.navbar-default .navbar-nav > .open > a,
.navbar-default .navbar-nav > .open > a:hover,
.navbar-default .navbar-nav > .open > a:focus {
  color: #555;          /*下拉菜单打开时，链接项的文本颜色和背景颜色*/
  background-color: #e7e7e7;
}
@media (max-width: 767px) {
  .navbar-default .navbar-nav .open .dropdown-menu > li > a {
    color: #777;                /*下拉菜单打开时，其里面的子菜单项链接的文本颜色*/
  }
  .navbar-default .navbar-nav .open .dropdown-menu > li > a:hover,
  .navbar-default .navbar-nav .open .dropdown-menu > li > a:focus {
    color: #333;                /*下拉菜单打开时，其里面的子菜单项链接的文本颜色（移动或焦点时）*/
    background-color: transparent;
  }
  .navbar-default .navbar-nav .open .dropdown-menu > .active > a,
  .navbar-default .navbar-nav .open .dropdown-menu > .active > a:hover,
            .navbar-default      .navbar-nav      .open      .dropdown-menu > .active > a:focus {
    color: #555; ;      /*下拉菜单打开时，其里面的当前活动的子菜单项链接的文本颜色*/
    background-color: #e7e7e7;
  }
  .navbar-default .navbar-nav .open .dropdown-menu > .disabled > a,
            .navbar-default      .navbar-nav      .open      .dropdown-menu > .disabled > a:hover,
            .navbar-default      .navbar-nav      .open      .dropdown-menu > .disabled > a:focus {
    color: #ccc;                /*下拉菜单打开时，里面的当前已禁用的子菜单项链接的文本颜色*/
    background-color: transparent;
  }
}
.navbar-default .navbar-link {  color: #777;   /*内部链接的文本颜色*/ }
.navbar-default .navbar-link:hover {  color: #333; /*内部链接在鼠标移动时的文本颜色*/ }
```

4.7.2 导航条中的表单

很多情况下，我们想把表单元素放在导航条里，以起到导航效果，比如支持搜索等。Bootstrap专门提供了一个附件样式.navbar-form来实现图4-41所示的效果。

图4-41 表单在导航条中的运行效果

使用方式是，在.navbar容器内放置form元素，然后在form元素上应用.navbar-form样式即可。同时左右浮动的样式（.navbar-left和.navbar-right）也可以使用，以便控制左右对齐。示例如下：

```
<nav class="navbar navbar-default" role="navigation">
                    <div     class="navbar-header"><a     class="navbar-
brand" href="#">Brand</a></div>
    <form class="navbar-form navbar-left" role="search">
              <div  class="form-group"><input  type="text"  class="form-
control"
                    placeholder="Search"></div>
        <button type="submit" class="btn btn-default">左按钮</button>
    </form>
    <form class="navbar-form navbar-right" role="search">
              <div  class="form-group"><input  type="text"  class="form-
control"
                    placeholder="Search"></div>
        <button type="submit" class="btn btn-default">右按钮</button>
    </form>
</nav>
```

在源码实现上，主要是确保所有的表单控件元素都设置成内联块（inline-block）的形式显示，并且确保特殊的radio和checkbox元素垂直居中即可。源码如下：

```
// 源码3846行
.navbar-form {
  padding: 10px 15px;
  margin-top: 8px;       /* 减少margin值 */
  margin-right: -15px;
  margin-bottom: 8px;
  margin-left: -15px;
  border-top: 1px solid transparent;
  border-bottom: 1px solid transparent;
                                                        -webkit-box-
shadow: inset 0 1px 0 rgba(255, 255, 255, .1), 0 1px 0 rgba(255,
           255, 255, .1);
                                                              box-
shadow: inset 0 1px 0 rgba(255, 255, 255, .1), 0 1px 0 rgba(255,
                    255, 255, .1);
}
```

```
@media (min-width: 768px) {        /* input、select、textarea、radio、checkbox都设置为
                                      内联块显示 */
    .navbar-form .form-group {       display: inline-block;    margin-bottom: 0;
           vertical-align: middle;  }
         .navbar-form    .form-control    {        display:    inline-block;  width: auto;
           vertical-align: middle;  }
   .navbar-form .control-label {      margin-bottom: 0;      vertical-align: middle;  }
  .navbar-form .radio,
  .navbar-form .checkbox {
    display: inline-block;    padding-left: 0;
    margin-top: 0;    margin-bottom: 0;    vertical-align: middle;
  }
  .navbar-form .radio input[type="radio"],
  .navbar-form .checkbox input[type="checkbox"] {
    float: none;    margin-left: 0;
  }
  .navbar-form .has-feedback .form-control-feedback {    top: 0;  }
}
@media (max-width: 767px) {             /* 窄屏浏览器下，设置底部外边距*/
  .navbar-form .form-group {    margin-bottom: 5px;  }
}
@media (min-width: 768px) {             /* 普通宽屏浏览器下*/
  .navbar-form {
    width: auto;
    padding-top: 0;             /* 去除上下内边距*/
    padding-bottom: 0;
    margin-right: 0;            /*去除左右外边距*/
    margin-left: 0;
    border: 0;
    -webkit-box-shadow: none;   /* 取消阴影*/
            box-shadow: none;
  }
  .navbar-form.navbar-right:last-child {
    margin-right: -15px;                /* 最后一个元素，设置负的margin，以便对齐*/
  }
}
```

注意

如果在表单内没有为输入框添加label标签，屏幕阅读器将会遇到问题。对于导航条内的表单，可以通过.sr-only样式来隐藏label标签。

4.7.3 导航条中的按钮、文本、链接

在普通导航条里，除了可以使用navbar-brand样式的a元素和navbar-nav的ul和form以外，也可以使用其他元素，比如button（navbar-btn）、文本（navbar-text）、普通链接（navbar-link）等。但Bootstrap对另外这些元素有些限制，那就是如果单独应用button、文本或普通链接，将只支持一个，或者最多两个（需要设置左右浮动），并且需要配合ul或者navbar-brand样式的a元素一起使用。我们来解释一下原因。

首先了解一下Bootstrap对这3种特殊元素是如何支持的。源码如下：

```
// 源码3918行
.navbar-btn {  margin-top: 8px;  margin-bottom: 8px; /* 仅设置了上下外边距值 */ }
.navbar-btn.btn-sm {  margin-top: 10px;  margin-bottom: 10px;}
.navbar-btn.btn-xs {  margin-top: 14px;  margin-bottom: 14px;}
.navbar-text {   margin-top: 15px;  margin-bottom: 15px;}
@media (min-width: 768px) {                    /* 普通宽屏浏览器下，增加左右外边距*/
   .navbar-text {     float: left;     margin-right: 15px;     margin-left: 15px;  }
  .navbar-text.navbar-right:last-child {    margin-right: 0;  }
}
/*以下代码在3956行*/
.navbar-default .navbar-text {  color: #777777; /*设置navbar-text的文本颜色*/ }
/*以下代码在4021行*/
.navbar-default .navbar-link {  color: #777;    /*内部链接的文本颜色*/ }
.navbar-default .navbar-link:hover {  color: #333;      /*内部链接在鼠标移动时的文本颜色*/ }
```

可以发现，CSS的设置项非常少，只是象征性地设置了margin和颜色，但威力却非常强大（大家再仔细看一遍.navbar-default的源码就能知道，其实该风格控制了很多东西），所以不管怎么应用，都不会出现问题。来看看如下几个示例：

```
<nav class=“navbar navbar-default” role=“navigation”>
                    <div     class=“navbar-header”><a     class=“navbar-brand” href=”#”>Brand</a></div>
    <ul class=“nav navbar-nav”>
        <li class=“active”><a href=”#”>Home</a></li>
        <li><a href=”#”>Link</a></li>
        <li><a href=”#”>Link</a></li>
    </ul>
    <button class=“btn btn-success navbar-btn” type=“button”>和ul一起使用的按钮</button>
</nav>
<nav class=“navbar navbar-default” role=“navigation”>
                    <div     class=“navbar-header”><a     class=“navbar-brand” href=”#”>Brand</a></div>
    <div class=“nav navbar-nav”>
```

```
          <button class="btn btn-warning navbar-btn" type="button">和navbar-
                  brand一起使用的按钮</button>
         <button class="btn btn-warning navbar-btn" type="button">另外一个按钮</button>
    </div>
</nav>
<nav class="navbar navbar-default" role="navigation">
                    <div    class="navbar-header"><a    class="navbar-brand" href="#">Brand</a></div>
    <div class="nav navbar-nav">
         <button class="btn btn-danger navbar-btn" type="button">2个单独的按钮A</button>
         <button class="btn btn-danger navbar-btn" type="button">2个单独的按钮B</button>
    </div>
</nav>
```

上述代码的运行效果如图4-42所示。

图4-42　导航条中的按钮运行效果

而对于文本和链接，如果连续放置两个相同的元素，则会出现一些问题。示例代码如下：

```
<nav class="navbar navbar-default" role="navigation">
                    <div    class="navbar-header"><a    class="navbar-brand" href="#">Brand</a></div>
    <div class="nav navbar-nav">
        <label class="navbar-text">文本A</label>
        <label class="navbar-text">文本A</label>
    </div>
</nav>
<nav class="navbar navbar-default" role="navigation">
                    <div    class="navbar-header"><a    class="navbar-brand" href="#">Brand</a></div>
    <div class="nav navbar-nav">
        <p class="navbar-text">文本A</p>
        <p class="navbar-text">文本A</p>
    </div>
</nav>
<nav class="navbar navbar-default" role="navigation">
                    <div    class="navbar-header"><a    class="navbar-brand" href="#">Brand</a></div>
    <div class="nav navbar-nav">
        <a href="#" class="navbar-link">汤姆大叔</a>
        <a href="#" class="navbar-link">汤姆大叔</a>
```

```
    </div>
</nav>
```

上述代码的运行效果如图4-43所示。

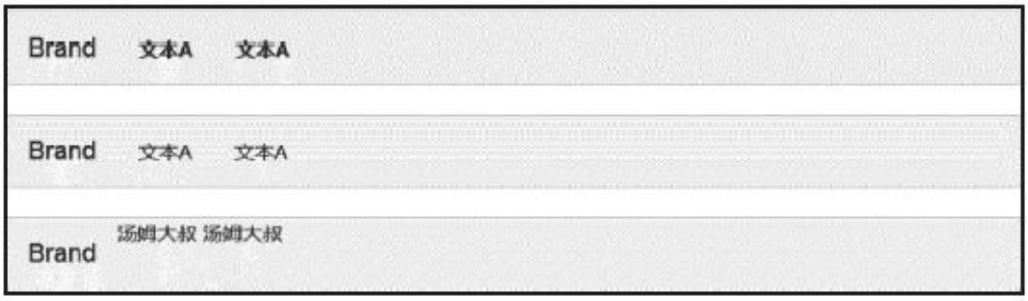


图4-43　导航条中的文本运行效果

可以看出，navbar-text在label和p元素上使用的时候还比较正常，但是navbar-link就有问题了，因为没有设置padding值。解决方法是，在navbar-link样式上再应用navbar-text样式就可以了（或者在外面的容器上应用navbar-text样式）。

Button方面也有问题，如果将多个button放到navbar-nav样式的多个li元素里或者放到navbar-form样式的form元素会正常吗？也不会，不信看如图4-44所示的运行效果。

图4-44　导航条中的按钮运行bug

在li元素里放button，就会让多个button重叠在一起，因为li默认设置了margin-left为负值。而在form里放的话，左右间距是正常了，但是上下间距又不正常了。但上述方式内放置文本和链接却是正常的，因为li本身就支持，而且-1像素的margin-left对文本和链接没有太大影响。

总结，要尽量避免使用这些元素，如果非要用，就自己实现一个灰色背景，自己控制间距，没有必要在导航条的基础上做。按照上面的建议设置各个元素所需要改进的地方。或者如果需要显示两个，就使用navbar-left和navbar-left进行浮动显示吧。

4.7.4　导航条中的项进行左右浮动

在导航条的各种菜单项容器（如ul、p等元素）上设置.navbar-left样式或navbar-right样式，即可让该容器元素左右浮动。这些样式是.pull-left和.pull-right的mixin版本，但是它们被限定在了媒体查询中，这样可以更容易地在各种尺寸的屏幕上处理导航条组件。示例代码如下：

```
<nav class=“navbar navbar-default” role=“navigation”>
    <div class=“navbar-header”>
        <a class=“navbar-brand” href=”#”>Brand</a>
    </div>
    <p class=“navbar-text navbar-left”>左浮动</p>
    <p class=“navbar-text navbar-right”>右浮动</p>
</nav>
```

上述代码的运行效果如图4-45所示。

图4-45　导航条中的左右浮动运行效果

但是这两个样式只在普通的宽屏浏览器（大于768像素）上才能使用，在窄屏浏览器上是无效的。源码如下：

```
// 源码3838行
@media (min-width: 768px) {      /*宽屏情况下*/
  .navbar-left {
    float: left !important;      /*左浮动*/
  }
  .navbar-right {
    float: right !important;     /*右浮动*/
  }
}
```

4.7.5 顶部固定或底部固定

很多情况下，设计师都想让导航条固定在某个位置上。比如最顶部或者最底部。Bootstrap提供了两个强有力的样式支持这一特性，分别是：.navbar-fixed-top支持最顶部固定，.navbar-fixed-bottom支持最底部固定。示例用法如下：

```
<!– 顶部固定 –>
<div class=“navbar navbar-default navbar-fixed-top”>
  …
</div>
<!– 底部固定 –>
<div class=“navbar navbar-default navbar-fixed-bottom”>
  …
</div>
```

功能虽然强大，但实现原理却非常简单，主要是利用了position属性为fixed（绝对定位）的特性，然后设置元素容器的top或者bottom值为0即可。具体源码如下：

```
// 源码3724行
.navbar-fixed-top,
.navbar-fixed-bottom {
  position: fixed;                          /* 固定定位 */
  right: 0;                        /* 左右不留空隙 */
  left: 0;
  z-index: 1030;                              /* 加大z-inedx，确保导航条在最上
边，但小于modal弹窗
                                             的1040/1050 */
}
@media (min-width: 768px) {      /*普通宽屏浏览器下才取消圆角*/
  .navbar-fixed-top,
  .navbar-fixed-bottom {
    border-radius: 0; ;          /* 无圆角 */
  }
}
.navbar-fixed-top {
  top: 0;                                   /* 固定在最顶部显示 */
  border-width: 0 0 1px;                    /* 左右1像素边框 */
}
.navbar-fixed-bottom {
  bottom: 0;                       /* 固定在最底部显示 */
  margin-bottom: 0;                /* 底部不留空隙 */
  border-width: 1px 0 0;                    /* 上下1像素边框 */
}
```

运行之后，读者可能会发现，导航条位置虽然固定了，但却带来一个问题，就是在顶部固定的情况下，页面最顶部的其他内容被遮盖住了（或者底部固定的情况下，网页最底部的内容也被遮盖住了）。原因很简单，就是fixed定位，网页最开头的元素（不管是否在导航条后面）在显示的时候都会从最顶部开始（0，0），然后导航条通过z-index覆盖在

上面。要解决这个问题，需要在body上设置padding值。由于默认的navbar高度是50px，所以一般padding值设置为70px即可。示例如下：

```
<!— 顶部固定的时候设置如下 —>
body { padding-top: 70px; }
<!— 底部固定的时候设置如下 —>
body { padding-bottom: 70px; }
```

注意

❑上述两行代码不需要同时设置，只需要根据顶部固定（或底部固定）选择其一即可。

❑上述代码需要在调用bootstrap.css之后才能使用，以便覆盖bootstrap的默认设置。

另外还有一个.navbar-static-top样式，用于表示设置一个100%充满父元素容器的导航条。其实默认情况下使用navbar样式的时候也是100%充满宽度的，.navbar-static-top样式的唯一功能是去掉导航条的圆角设置。

```
// 源码3715行
.navbar-static-top {
  z-index: 1000;
  border-width: 0 0 1px;                                    /* 左右1像素边
框 */
}
@media (min-width: 768px) {                          /*普通宽屏浏览器下才取消圆
角*/
  .navbar-static-top {    border-radius: 0;     /* 取消圆角设置 */  }
}
```

4.7.6　响应式导航条

一个导航条默认情况下都是全屏100%显示的，所以通常都会有很多菜单，如图4-46所示。

图4-46　大于768像素的宽屏导航条

但在一些小屏幕下可能就不会显示全，通常我们需要根据屏幕尺寸自动调整，隐藏或去除一部分菜单内容，这就是我们所说的响应式设计的一部分内容，如图4-47所示。

图4-47　小于768像素的导航条

Bootstrap提供了这种功能，屏幕大小的分界点是768像素，在小于768像素的时候，所有的菜单默认会隐藏，单击右边的icon图标，所有默认的菜单就会展示出来，如图4-48所示。

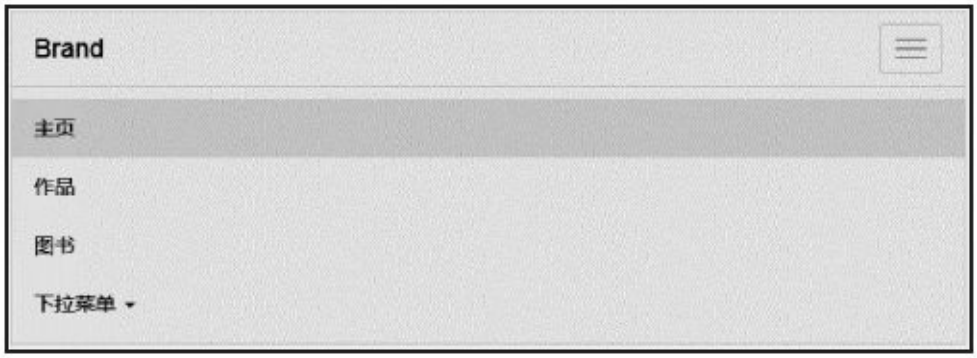


图4-48　单击图标显示菜单

这种效果不是很好看，因为这里用了input输入框，并且最后两个按钮进行了右浮动。如果去除这些元素，而保持普通链接，效果就会很好看。上述效果的HTML代码如下：

```
<div class="navbar navbar-default">
    <div class="navbar-header">
                    <!—   .navbar-toggle样式用于toggle收缩的内容,即：nav-collapse collapse
                样式所在的元素 —>
                 <button  type="button"  class="navbar-toggle"  data-toggle="collapse"
                data-target=".navbar-responsive-collapse ">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
```

```
        </button>
        <!— 确保无论是宽屏还是窄屏，navbar-brand都会显示 —>
        <a class=“navbar-brand” href=”#”>Brand</a>
    </div>
     <!— 屏幕宽度小于768像素时，该div内的内容默认都会隐藏（通过单击icon-bar所在的图标，
            可以再展开）；大于768像素时默认显示 —>
    <div class=“collapse navbar-collapse navbar-responsive-collapse ”>
        <ul class=“nav navbar-nav”>
            <li class=“active”><a href=”#”>主页</a></li>
            <li><a href=”#”>作品</a></li>
            <li><a href=”#”>图书</a></li>
            <li class=“dropdown”>
            <a data-toggle=“dropdown” class=“dropdown-toggle” href=”#”>
                        下拉菜单 <b class=“caret”></b></a>
                <ul class=“dropdown-menu”>
                    <li><a href=”#”>子菜单1</a></li>
                    <li><a href=”#”>子菜单2</a></li>
                    <!—省略菜单 —>
                </ul>
            </li>
        </ul>
    </div>
</div>
```

上述示例有几个需要注意的重要知识点。

（1）Toggle图标

右上角的button图标（icon）必须包含在.navbar-toggle样式里，这是作者定下的规矩。相关源码如下：

```
// 源码3763行
.navbar-toggle {                          /*窄屏时，响应式按钮，用于单击展开其他菜单*/
  position: relative;           /*相对定位*/
  float: right;                 /*右浮动*/
  padding: 9px 10px;
  margin-top: 8px;
  margin-right: 15px;           /*通过3个方向的margin，保证该按钮居中显示*/
  margin-bottom: 8px;
  background-color: transparent;        /*透明背景*/
  background-image: none;
  border: 1px solid transparent;
  border-radius: 4px;           /*圆角设置*/
}
.navbar-toggle:focus {
  outline: none;                        /* 焦点状态时取消轮廓 */
}
.navbar-toggle .icon-bar {      /*响应式按钮里的icon设置*/
  display: block;                       /*块级显示*/
  width: 22px;
  height: 2px;                  /*限制高度，因为一般都是显示3个*/
  border-radius: 1px;           /*限制圆角*/
}
.navbar-toggle .icon-bar + .icon-bar {
```

```
  margin-top: 4px;                /*多个icon之间的垂直间隔为4像素*/
}
```

根据响应式导航的要求，浏览器在大于768像素时，该icon图标是不应该显示的。通过分析下面的代码可以看出，navbar-toggle在大于768像素的宽屏下是不显示的，这正好符合我们的期望。

```
// 源码3787行
@media (min-width: 768px) {
   .navbar-toggle {     display: none;     /*宽屏下，隐藏响应式按钮，因为所有菜单默认都显示*/  }
}
```

（2）收缩容器

分析上述代码，可以看出在窄屏下，默认隐藏收缩的代码都在一个样式为.navbar-responsive-collapse的div里，并且该div应用了navbar-collapse和collapse两个样式。首先看一下collapse样式，默认情况下，应用collapse样式的容器会隐藏显示。源码如下：

```
// 源码2354行
.collapse {
  display: none;                              /*默认情况下，隐藏显示*/
}
```

而对于大于768像素的宽屏，通过媒体查询语句，特殊设置其高度为auto，也就是不隐藏。源码如下：

```
// 源码3677行
@media (min-width: 768px) {     /*宽屏时，折叠区域的显示设置*/
  .navbar-collapse {
    width: auto;                              /*不限制宽度*/
    border-top: 0;
    box-shadow: none;           /*取消阴影*/
  }
  .navbar-collapse.collapse {
    display: block !important;  /*块级显示*/
    height: auto !important;
    padding-bottom: 0;
    overflow: visible !important;
  }
  .navbar-collapse.in {
    overflow-y: visible;                      /*允许显示滚动条*/
  }
  .navbar-fixed-top .navbar-collapse,
  .navbar-static-top .navbar-collapse,
  .navbar-fixed-bottom .navbar-collapse {
    padding-right: 0;           /*固定位置时，取消左右的padding值*/
    padding-left: 0;
  }
}
```

再继续分析小于768像素的窄屏。由于默认情况下是隐藏的，通过单

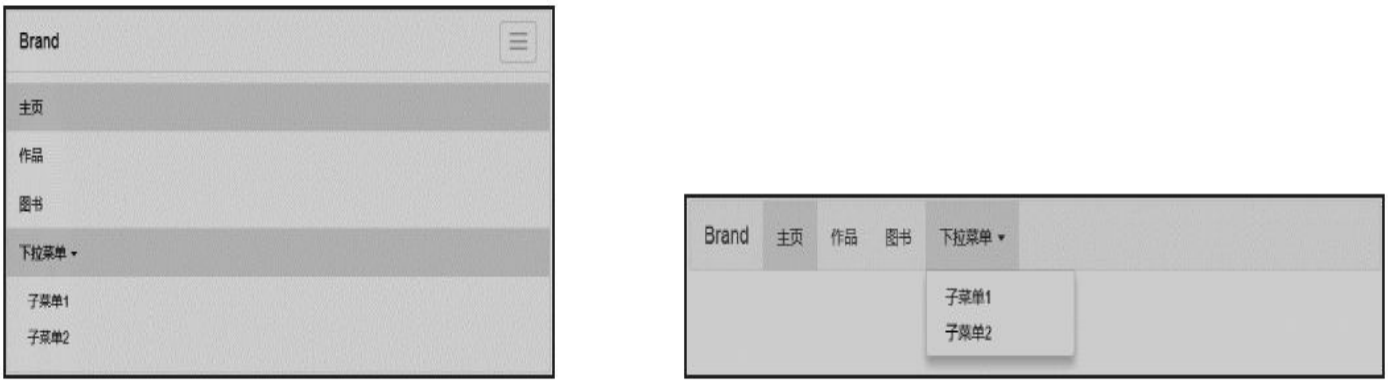
击Toggle图标，预期的结果是展开该收缩内容。那如何实现呢？通过上面的代码可以得出结果：很简单，单击以后，设置div的容器的高度height也为auto或者设置为块级显示方式就可以了。

通过单击事件检测HTML变换，JavaScript的collspase插件在单击的时候给div元素多附加了一个.in样式，也就是说默认的collapse样式和in样式组合在一起就可以展开内容了。去源码里找答案，正如我们所预期的，Bootstrap作者为这两个组合样式定义了块级显示方式。源码如下：

```
// 源码2357行
.collapse.in {
  display: block;                              /* 块级显示*/
}
```

至此就真相大白了。

让我们来运行示例看看效果。随便单点，突然发现单击导航条里的dropdown下拉菜单菜单的时候，和宽屏的弹出行为不同，在窄屏下，直接在下面显示了同样宽度的横条。效果如图4-49所示。



a）导航条在窄屏下的运行效果　　b）导航条在宽屏下的运行效果

图4-49　同一导航条在窄屏和宽屏下的运行效果

怎么回事？一般情况下dropdown下拉菜单是没问题的，现在为什么有问题?看看它的父元素容器有何限制？分析发现，作者在.navbar-nav样式下，又对.dropdown-menu样式进行了设置，主要目的就是让其和普通100%宽度的菜单有一样的效果，然后对内边距多设置了一点，分别是：padding: 5px 15px 5px 25px;。源码如下：

```
// 源码3800行
@media (max-width: 767px) {      /*在窄屏情况下*/
   .navbar-nav .open .dropdown-menu {      /*如果菜单项是个下拉菜单，对下拉菜单里的子菜单进行设置*/
     position: static;
     float: none;                                /*取消浮动*/
     width: auto;                                /*宽度自适应定*/
     margin-top: 0;                  /*取消上外边距*/
     background-color: transparent;       /*背景透明*/
```

```
    border: 0;                    /*取消边框*/
    box-shadow: none;             /*取消阴影*/
  }
  .navbar-nav .open .dropdown-menu > li > a,
  .navbar-nav .open .dropdown-menu .dropdown-header {
    padding: 5px 15px 5px 25px; /*设置子菜单项或者菜单项标题的内边距*/
  }
  .navbar-nav .open .dropdown-menu > li > a {
    line-height: 20px;            /*设置行距*/
  }
  .navbar-nav .open .dropdown-menu > li > a:hover,
  .navbar-nav .open .dropdown-menu > li > a:focus {
    background-image: none;
  }
}
```

而且在.navbar-default样式风格下，对下拉菜单的子菜单项的鼠标行为和颜色也都进行了相关的调整。具体源码如下：

```
// 源码3999行
@media (max-width: 767px) {
  .navbar-default .navbar-nav .open .dropdown-menu > li > a {
    color: #777;         /*下拉菜单打开时，其里面的子菜单项链接的文本颜色*/
  }
  .navbar-default .navbar-nav .open .dropdown-menu > li > a:hover,
  .navbar-default .navbar-nav .open .dropdown-menu > li > a:focus {
    color: #333;         /*下拉菜单打开时，其里面的子菜单项链接的文本颜色（移动或
焦点时）*/
    background-color: transparent;
  }
  .navbar-default .navbar-nav .open .dropdown-menu > .active > a,
  .navbar-default .navbar-nav .open .dropdown-menu > .active > a:hover,
            .navbar-default      .navbar-nav      .open      .dropdown-
menu > .active > a:focus {
    color: #555; ;       /*下拉菜单打开时，其里面的当前活动的子菜单项链接的文本颜
色*/
    background-color: #e7e7e7;
  }
  .navbar-default .navbar-nav .open .dropdown-menu > .disabled > a,
            .navbar-default      .navbar-nav      .open      .dropdown-
menu > .disabled > a:hover,
            .navbar-default      .navbar-nav      .open      .dropdown-
menu > .disabled > a:focus {
    color: #ccc;         /*下拉菜单打开时，其里面的当前已禁用的子菜单项链接的文本
颜色*/
    background-color: transparent;
  }
}
```

这也就是我们在上边看到，在两种尺寸下下拉菜单显示不同效果的原因了。

注意

示例HTML里的toggle按钮上的data-target的设置为.navbar-responsive-

collapse，而不是先前我们用的#或者元素#+id选择器。它的意思是使用样式进行定位，即查找父容器下的子元素，如果有navbar-responsive-collapse样式就进行toggle，也就是我们所说的默认隐藏收缩的div容器。可以回头检查一下，该div上应用了navbar-responsive-collapse样式。

4.7.7　反色导航条

反色导航条，其实就是上面我们说的导航条两大风格中的第二个风格.navbar-inverse。和第一个风格相比，除了各种背景色、文本颜色以外，其他都相同。运行效果如图4-50所示。

图4-50　反色导航条运行效果

反色导航的源码和.navbar-default几乎一样，不一样的只是各种颜色设置，详细不同请自行阅读源码（4027行之后的代码）。

如果要扩展自己的风格，请务必按照与.navbar-inverse一样的代码进行颜色改变。

4.8　面包屑导航

源码文件：breadcrumbs.less

CSS文件：bootstrap.css 4116行之后

面包屑（Breadcrumb）一般用于导航，表示当前页面所在的位置（或功能插件）。使用方式如下：

```
<ul class="breadcrumb">
  <li><a href="#">Home</a></li>
  <li><a href="#">Library</a></li>
  <li class="active">Data</li>
</ul>
```

上述示例的运行效果如图4-51所示。

图4-51　面包屑导航运行效果

2.x版本的设计思路如下：

❑设置面包屑的背景颜色和相关的圆角等。

❑循环显示多个li，每个li表示一个元素链接。

❑每个li后面要显示分隔符，可利用CSS里的after和content实现。

❑最后一个li后面不需要显示分隔符。

❑高亮当前li元素的内容。

旧版代码如下：

```
.breadcrumb {                                    /* 基础样式，背景、圆角、list-
style */
  padding: 8px 15px;
  margin: 0 0 20px;
  list-style: none;
  background-color: #f5f5f5;
  border-radius: 4px;
}
.breadcrumb > li {                          /* li设置为内敛块模式 */
  display: inline-block;
  text-shadow: 0 1px 0 #fff;
}
.breadcrumb > li:after {                         /* 分隔符设置：每个li后面设
```

```
置分隔符 */
  display: inline-block;
  padding: 0 5px;
  color: #ccc;
  content: ”\00a0 /”;
}
.breadcrumb > li:last-child:after {       /* 最后一个li后面的分隔符不需要设
置 */
  display: none;
}
.breadcrumb > .active {                  /* 设置当前项的颜色 */
  color: #999999;
}
```

看起来是不是很不错？是的，很不错。但在3.x版本，作者改变了设计思想，其方式更为精妙。源码如下：

```
// 源码4116行
.breadcrumb {                                /* 基础样式，背景、圆角、list-
style */
  padding: 8px 15px;
  margin-bottom: 20px;
  list-style: none;
  background-color: #f5f5f5;
  border-radius: 4px;
}
.breadcrumb > li {
  display: inline-block;                          /* 所有的li项都是内联块方
式 */
}
.breadcrumb > li + li:before {           /* 并列li项才起作用 */
  padding: 0 5px;
  color: #ccc;
  content: ”/\00a0”;
}
.breadcrumb > .active {
  color: #999;                           /* 设置当前项的颜色 */
}
```

新版本代码在对分隔符进行设置的时候，使用了li + li: before语法，也就是说只有两个li在一起显示的时候，才在后面一个li的before的content上设置分隔符。由于所有的li都在一起，所以都可以认为是两两相连的，也就说除了第一个li以外，其他所有的li项都满足条件，在它的before的content上设置分隔符，正好符合我们的预期。

是不是很精妙？老版3段代码实现的功能，新版用2段就实现了，而且代码更少。看来Bootstrap的作者对CSS3的理解又进步了不少。

注意

面包屑内可以设置其他相关的小标记内容，比如标签、徽章标记等。

4.9　分页导航

几乎所有的网站内容都需要分页显示（比如新闻列表、订单记录等），一个用户体验良好的分页组件会得到访问用户的良好评价。Bootstrap为大家提供了两种分页组件，一种是带多个页码的组件（pagination），一种是只有上一页、下一页的翻页组件（pager）。

4.9.1　页码分页

源码文件：pagination.less

CSS文件：bootstrap.css 4134行之后

普通页码分页（pagination）的效果如图4-52所示。中间是页码，两头分别是上一页和下一页的链接。

图4-52　普通分页效果

页码分页的HTML代码和样式设置非常容易，只需要在ul上设置pagination样式，在li元素上设置页码链接即可。示例如下：

```
<ul class=“pagination”>
  <li><a href=”#”>«</a></li> <!– 上一页 –>
  <li><a href=”#”>1</a></li>
  <li><a href=”#”>2</a></li>
  <li><a href=”#”>3</a></li>
  <li><a href=”#”>4</a></li>
  <li><a href=”#”>5</a></li>
  <li><a href=”#”>»</a></li> <!– 下一页 –>
</ul>
```

页码分页实现原理可以分解为如下两个步骤。

步骤1　设置li元素的内联显示和边框属性。源码如下：

```
// 源码4134行
.pagination {
  display: inline-block;                    /* 内联块显示 */
  padding-left: 0;                  /* 取消左内边距块显示 */
  margin: 20px 0;                           /* 上下margin为20像素 */
  border-radius: 4px;               /* 容器元素的圆角设置 */
}
.pagination > li {
  display: inline;                  /* 内联显示 */
}
.pagination > li > a,
.pagination > li > span {
  position: relative;
  float: left;
  padding: 6px 12px;
  margin-left: -1px;
  line-height: 1.428571429;
  color: #428bca;
  text-decoration: none;
  background-color: #fff;
  border: 1px solid #ddd;                   /* 设置所有的边框都为1像素 */
}
```

运行效果如图4-53所示。

图4-53　设置边框

步骤2　设置第一个元素和最后一个元素的圆角，即可得到最终效果。源码如下：

```
// 源码4155行
.pagination > li:first-child > a,
.pagination > li:first-child > span {
  margin-left: 0;                                    /* 取消左外边距*/
  border-top-left-radius: 4px;              /* 设置左上方圆角 */
  border-bottom-left-radius: 4px;           /* 设置左下方圆角 */
}
.pagination > li:last-child > a,
.pagination > li:last-child > span {
  border-top-right-radius: 4px;             /* 设置右上方圆角 */
  border-bottom-right-radius: 4px;          /* 设置右下方圆角 */
}
```

运行效果如图4-54所示。

图4-54　设置圆角

1.状态支持

对于分页，一般来说当前页面会高亮显示，并且不允许单击。而在第一页的时候，上一页链接也不允许单击。为此Bootstrap提供了两个通用的样式来实现，分别是：active和disabled（源码在4166行之后）。示例代码如下：

```
<ul class="pagination">
  <li class="disabled"><a href="#">«</a></li> <!– 上一页 –>
  <li class="active"><a href="#">1</a></li>
  <li><a href="#">2</a></li>
  <li><a href="#">3</a></li>
  <li><a href="#">4</a></li>
  <li><a href="#">5</a></li>
  <li><a href="#">»</a></li> <!– 下一页 –>
</ul>
```

上述示例的运行效果如图4-55所示。

图4-55　当前页面禁用状态

上述代码，即便应用了active和disabled，也阻止不了单击效果，还是需要JavaScript来阻止。所以一般情况下，最终的完美解决方案是在上述两种样式的li元素里使用span元素而不是a元素。因为从上述2个步骤的分析源码可以看出，pagination在li元素里支持a和span两种元素标签。示例代码如下：

```
<ul class=“pagination”>
  <li class=“disabled”><span>«</span></li>
  <li class=“active”><span>1</span></li>
</ul>
```

2.大小支持

同button的样式btn一样，分页也支持增大和缩小风格，其样式分别是：.pagination-lg和.pagination-sm。使用方式如下：

```
<ul class=“pagination pagination-lg”>…</ul>
<ul class=“pagination”>…</ul>
<ul class=“pagination pagination-sm”>…</ul>
```

大小样式主要是增加相应的padding大小、字体大小、圆角大小。源码如下：

```
// 源码4197行
.pagination-lg > li > a,
.pagination-lg > li > span {
  padding: 10px 16px;                    /* 增大padding值 */
  font-size: 18px;                       /* 增大字体大小 */
}
.pagination-lg > li:first-child > a,
.pagination-lg > li:first-child > span {         /*第一个li项，增大圆角弧度
*/
  border-top-left-radius: 6px;
  border-bottom-left-radius: 6px;
}
.pagination-lg > li:last-child > a,
.pagination-lg > li:last-child > span { /*最后一个li项，增大圆角弧度*/
  border-top-right-radius: 6px;
  border-bottom-right-radius: 6px;
}
```

.pagination-sm样式的设置类似，详情请阅读源码4212行。

4.9.2 翻页

源码文件：pager.less

CSS文件：bootstrap.css 4227行之后

在一些简单的网站上（比如博客、杂志），一般不会展示很多页码，而是使用“上一页”、“下一页”的简单分页方式。默认情况下，Bootstrap提供的这种组件是居中显示的。示例代码如下：

```
<ul class=“pager”>
    <li><a href=”#”>上一页</a></li>
    <li><a href=”#”>下一页</a></li>
</ul>
```

上述示例的运行效果如图4-56所示。

图4-56　翻页效果

翻页效果的实现方式，有如下几个要点：

❑整体居中显示。

❑li元素有圆角。

❑设置鼠标移动上去的效果。

❑支持a元素和span元素。

支持以上几个要点的主要源码如下：

```
// 源码4227行
.pager {
  padding-left: 0;
  margin: 20px 0;
  text-align: center;   /* 居中显示 */
  list-style: none;     /* li无标记显示 */
}
.pager li {
  display: inline;      /* 内联显示 */
}
.pager li > a,
.pager li > span {
  display: inline-block;         /* 内联块，支持a元素和span元素 */
  padding: 5px 14px;
  background-color: #fff;
  border: 1px solid #ddd;        /* 边框设置1像素 */
  border-radius: 15px;  /* 加大圆角弧度 */
}
.pager li > a:hover,
```

```
.pager li > a:focus {
  text-decoration: none;          /* 鼠标触发或焦点时无下划线 */
  background-color: #eee;
}
```

如果需要将两个按钮连接，分别进行左右对齐，则需要在按钮的li容器元素上分别应用previous和next样式，效果如图4-57所示。

图4-57　左右对齐的翻页运行效果

两个样式的实现很简单，就是我们平常所用的左右浮动。源码如下：

```
// 源码4249行
.pager .next > a,
.pager .next > span {
  float: right;          /* 下一页，右浮动 */
}
.pager .previous > a,
.pager .previous > span {
  float: left;           /* 上一页，左浮动 */
}
```

注意

在li元素上分别应用.pull-left和.pull-right样式也能实现上述效果，因为原理是一样的。

另外，和普通的链接（或其他button元素）一样，应用disabled样式可实现禁用效果，运行效果如图4-58所示。

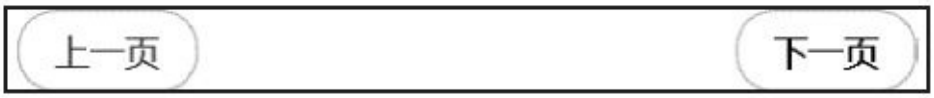


图4-58　翻页中的禁用效果

和以前一样需要注意，disabled样式只是实现禁用效果，并没有禁用a链接的单击功能。所以如果要禁用单击功能，要么使用JavaScript禁用a的单击行为，要么干脆使用span元素，因为分页组件对a和span两种元素都同样支持。分页组件里的disabled样式源码如下：

```
// 源码4257行
.pager .disabled > a,
.pager .disabled > a:hover,
.pager .disabled > a:focus,
.pager .disabled > span {
  color: #999;           /* 被禁用时，改变文本颜色 */
  cursor: not-allowed;   /* 被禁用时，改回默认的鼠标手形图标 */
```

```
  background-color: #fff;
}
```

4.10　标签

源码文件：label.less

CSS文件：bootstrap.css 4265行之后

在网页排版的时候，我们经常要高亮一些标题里的特殊字符或者整个字符，Bootstrap提供了一个.label样式用于实现高亮的功能。默认情况下高亮效果如图4-59所示。

图4-59　高亮运行效果

应用方式也非常简单，只需要在高亮的地方应用label样式即可。示例如下：

```
<h1>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h1>
<h2>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h2>
<h3>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h3>
<h4>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h4>
<h5>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h5>
<h6>10个最好的JavaScript动画库和开发框架<span class=“label”>New</span></h6>
```

标签样式的实现代码如下（主要是设置背景色和背景方框）：

```
// 源码4265行
.label {
  display: inline;
  padding: .2em .6em .3em;
  font-size: 75%;
  font-weight: bold;
  line-height: 1;
  color: #fff;
  text-align: center;
  white-space: nowrap;
  vertical-align: baseline;
```

```
  border-radius: .25em;
}
```

另外，label如果作用在链接元素上，hover和focus时的背景颜色会稍微加深10%。源码如下：

```
// 源码4277行
.label[href]:hover,
.label[href]:focus {
  color: #fff;
  text-decoration: none;
  cursor: pointer;
}
.label:empty {
  display: none;  /*如果元素内没有内容，则隐藏该元素*/
}
```

和按钮元素Button类似，label样式也提供了多种颜色的支持，分别是成功绿，警示黄、危险红、信息蓝，如图4-60所示。

图4-60　不图颜色样式的标签

.label样式的使用方式如下：

```
<span class=“label label-default”>Default</span>
<span class=“label label-primary”>Primary</span>
<span class=“label label-success”>Success</span>
<span class=“label label-warning”>Warning</span>
<span class=“label label-danger”>Danger</span>
<span class=“label label-info”>Info</span>
```

label样式的实现方式是修改背景颜色。主要源码如下：

```
// 源码4290行
.label-default {  background-color: #999999;}
.label-primary {  background-color: #428bca;}
.label-success {  background-color: #5cb85c;}
.label-info {  background-color: #5bc0de;}
.label-warning {  background-color: #f0ad4e;}
.label-danger {  background-color: #d9534f;}
```

另外，对于作用域在href链接上的label样式，hover和focus时的背景色也会相应地加深10%。主要代码见Bootstrap4293行。

其实，也可以像.btn一样扩展label的尺寸大小，例如label-lg、label-sm等类似的样式。

4.11 徽章

源码文件：badges.less

CSS文件：bootstrap.css 4332行之后

在开发交互式系统或者信息系统时，经常要显示一些最新收到的消息、需要有多少审批的消息等。Bootstrap的.badge样式提供了很好的效果，只需要在span或者label上应用该样式即可。示例如下：

```
<a href="#">Inbox <span class="badge">42</span></a>
```

上述示例的运行效果如图4-61所示。

图4-61　徽章运行效果

如下源码主要是设置具有圆角的背景椭圆框。

```
// 源码4332行
.badge {
  display: inline-block;
  /* 此处省略部分设置 */
  vertical-align: baseline;
  background-color: #999;        /* 背景色 */
  border-radius: 10px;           /* 超大圆角 */
}
```

另外，如果.badge样式应用的内部无内容时（例如：没有新消息），.badge样式会自动消失。示例和源码实现分别如下。

示例：

```
<span class="badge"></span>
```

源码：

```
// 源码4346行
.badge:empty {
  display: none;
}
```

注意

由于IE8和以下版本不支持empty选择器，所以在这些浏览器下没有上述效果。

另外，badge样式在按钮元素button或list-group-item和nav-pills也可以有类似的样式，只是颜色稍有不同。示例如下。

```
<ul class="nav nav-pills nav-stacked">
```

```
  <li class="active">
    <a href="#">
      <span class="badge pull-right">42</span>
      Home
    </a>
  </li>
  …
</ul>
```

上述示例的运行效果如图4-62所示。

图4-62　徽章在按钮及下拉菜单中的运行效果

徽章样式badge的源码如下：

```
// 源码4349行
.btn .badge {
  position: relative;           /*在.btn样式里的元素上应用badge时，设置为相对定位*/
  top: -1px;
}
a.badge:hover,
a.badge:focus {                  /*在a链接上设置badge时，在hover和focus时的颜色设置*/
  color: #ffffff;
  text-decoration: none;
  cursor: pointer;
}
a.list-group-item.active > .badge,
.nav-pills > .active > a > .badge {
  color: #428bca;                        /*将文本颜色和胶囊式按钮的背景色设为一致*/
  background-color: #ffffff;    /*背景设为白色*/
}
.nav-pills > li > a > .badge {
  margin-left: 3px;             /*设置左外边距*/
}
```

.badge样式有个遗憾就是，它没有定义多风格颜色的设定。不过根据label样式的多颜色设置，我们可以很简单为.badge样式扩展类似的功能，使用方式也和label类似。扩展源码如下：

```
.badge-danger {    background-color: #d9534f;}
.badge-success {    background-color: #5cb85c;}
.badge-warning {    background-color: #f0ad4e;}
.badge-info {    background-color: #5bc0de;}
```

上述代码的运行效果如图4-63所示。

图4-63　扩展后的彩色徽章

类似的，我们也可以为特殊元素内的badge定义特殊的字体颜色（而保持背景色不变）。扩展源码如下：

```
.badge-content-default {    background-color: #fff;    color: #a7a9aa;}
.badge-content-danger {    background-color: #fff;    color: #d9534f;}
.badge-content-success {    background-color: #fff;    color: #5cb85c;}
.badge-content-warning {    background-color: #fff;    color: #f0ad4e;}
.badge-content-info {    background-color: #fff;    color: #5bc0de;}
```

上述代码的运行效果如图4-64所示。

图4-64　扩展后的徽章在特殊字体时的颜色

4.12　大屏幕展播

源码文件：jumbotron.less

CSS文件：bootstrap.css 4371行之后

在设计网页布局的时候，经常会有一些大屏（或大块头）内容的显示，.jumbotron样式提供的展示效果正是我们所需要的，如图4-65所示。

图4-65　大屏幕展播效果

非常炫吧，使用方式只需要在容器元素上应用.Jumbotron样式即可。示例如下：

```
<div class=“jumbotron”>
  <h1>Hello, world!</h1>
  <p>…</p>
  <p><a class=“btn btn-primary btn-large”>Learn more</a></p>
</div>
```

实现该效果的jumbotron样式源码如下：

```
// 源码4371行
.jumbotron{                                          /*默认100%填充*/
  padding: 30px;                                            /*加大内边距*/
  margin-bottom: 30px;                               /*加大外边距*/
  color: inherit;
  background-color: #eee;                            /*背景颜色*/
}
.jumbotron h1,.jumbotron .h1 {  color: inherit; /*窄屏下，字体设置*/}
.jumbotron p {                                       /*窄屏下，行距设置*/
  margin-bottom: 15px;
  font-size: 21px;                                   /*加大字体*/
  font-weight: 200;                                  /*超大粗体*/
}
.container  .jumbotron  {   border-radius:  6px;       /*如果jumbotron放在
container样式
                                                          内，则显示圆角*/ }
.jumbotron  .container  {   max-width:  100%;         /*如果jumbotron内部有
container样
                                                                式，则最大宽度为
100%*/ }
```

```
@media screen and (min-width: 768px) {          /*宽屏下的设置*/
  .jumbotron { padding-top: 48px;    padding-bottom: 48px;
                                                        /*宽屏下，加大上下
内边距*/    }
  .container .jumbotron { padding-right: 60px;    padding-left: 60px;
                                                        /*宽屏下，加大左右
内边距*/    }
  .jumbotron h1,  .jumbotron .h1 {    font-size: 63px;  /*宽屏下，字体加大
*/  }
}
```

通过源码可以看出，在.jumbotron内部应用h1和p元素时，这两种元素也会进行相应的调整。从前面的HTML示例也可以看出这点。

另外，上述源码中有一个.container .jumbotron设置，也就是说，如果jumbotron放在container样式内，则显示圆角；如果不放在里面，就不会显示圆角。所以可以将.jumbotron样式从所有的.container样式容器里拿出来，然后单独放在一个地方（内部可以包含.container样式），就不再显示圆角了。示例如下：

```
<div class=“jumbotron”>
    <div class=“container”>
        …
    </div>
</div>
```

最后，通过上面的后半部分的媒体查询代码，可以得知在屏幕宽度大于768像素的时候，.jumbotron样式的padding也加大了，其内部字体也增大到了63像素。而如果在窄屏下，则按上面的默认设置样式显示。

4.13　页面标题

源码文件：type.less

CSS文件：bootstrap.css 561行之后

页面标题（Page　header）在2.x版是一个基础排版样式，在新版里划成了CSS组件，但没有进行功能的扩展。对于该组件，Bootstrap中的.page-header样式提供了类似h1元素的显示效果，只不过margin和底部padding距离稍大一些，底部有1像素的边框稍大一些。使用方式如下：

```
<div class=“page-header”>
  <h1>Example page header <small>Subtext for header</small></h1>
</div>
```

上述代码的运行效果如图4-66所示。

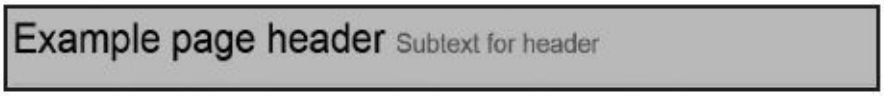


图4-66　页面标题运行效果

下面页面标题的源码非常简单，请自行理解。

```
// 源码561行
.page-header {
  padding-bottom: 9px;                    /* 设置底部内边距*/
  margin: 40px 0 20px;                    /* 设置上下外边距*/
  border-bottom: 1px solid #eee;          /* 设置底部边框为1像素,并设置颜色*/
}
```

4.14　缩略图

源码文件：thumbnails.less

CSS文件：bootstrap.css 4406行之后

在第3章中我们提到了图像展示的几种方式，这里将详细说明缩略图.thumbnail样式。结合12栅格系统，并使用.thumbnail样式，可以将图像、视频、文本展示得更加漂亮。使用方式如下：

```
<div class=“row”>
    <div class=“col-sm-6 col-md-3”>
        <a href=”#” class=“thumbnail”>
            <img data-src=”图像地址” alt=”…”>
        </a>
    </div>
    …
</div>
```

运行效果如图4-67所示。

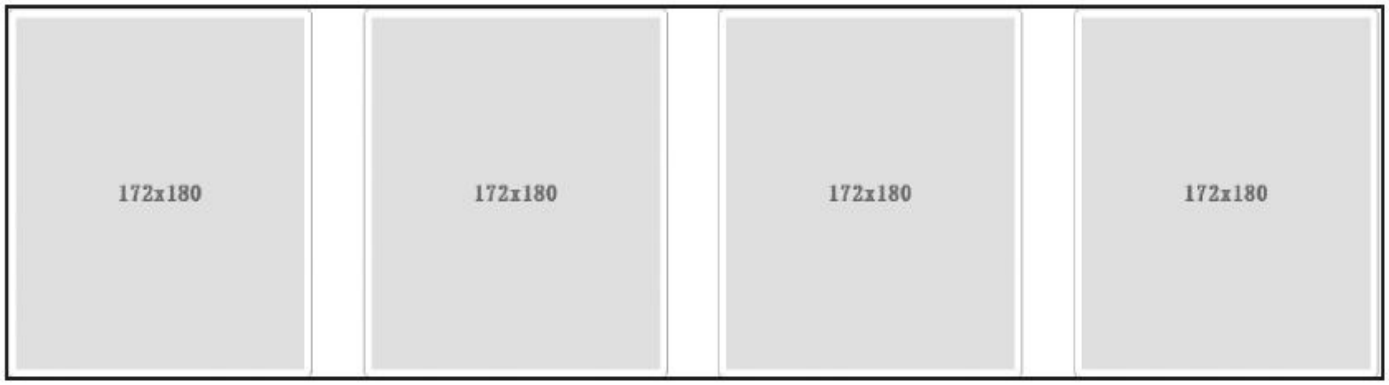


图4-67　缩略图运行效果

thumbnail样式的源码如下：

```
// 源码4406行
.thumbnail {
  display: block;                /*块级显示*/
  padding: 4px;          /* 图片外边设置4像素的边框间距 */
  margin-bottom: 20px;   /* 底部保留20像素的外边距 */
  line-height: 1.428571429;
  background-color: #fff;        /* 白色背景 */
  border: 1px solid #ddd;        /* 灰色、1像素的边框 */
  border-radius: 4px;    /* 圆角设置 */
  -webkit-transition: all .2s ease-in-out;
          transition: all .2s ease-in-out;
}
.thumbnail > img,
.thumbnail a > img {
  display: block;                /*块级显示*/
  max-width: 100%;       /* 如果img位于thumbnail样式内部，则自动填充100% */
  height: auto;          /*高度自适应*/
```

```
  margin-right: auto;   /*水平居中*/
  margin-left: auto;
}
a.thumbnail:hover,
a.thumbnail:focus,
a.thumbnail.active {
  border-color: #428bca;         /*鼠标移动或焦点时（也包括高亮时）边框颜色设置
*/
}
.thumbnail .caption {
  padding: 9px;          /* caption样式一般用于显示缩略图名称，再增加一点边框间
距 */
  color: #333;
}
```

根据上述代码可知，如果想像label和badge一样扩展.thumbnail的样式，修改border: 1px solid #dddddd;代码即可。效果如图4-68所示。怎么样，简单吧！

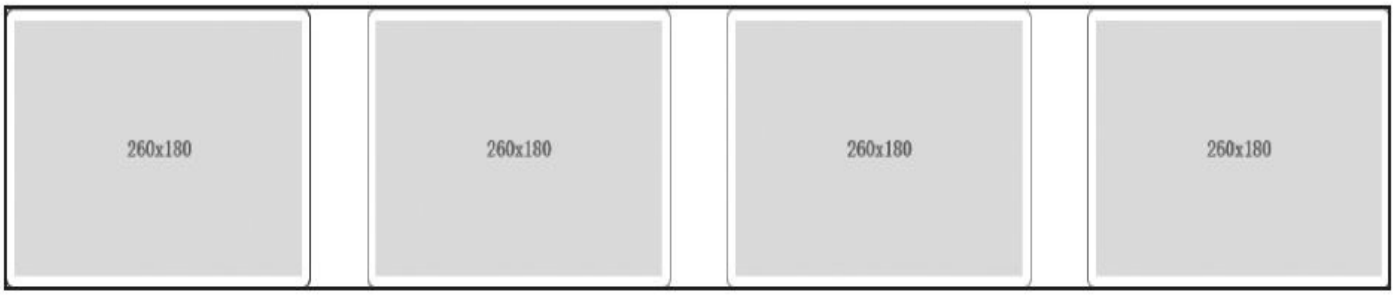


图4-68　边框颜色扩展后的运行效果

另外，根据上述源码的定义，缩略图有两种使用方式，一种用于仅显示图片，另外一种利用容器将图片和标题显示在一起。示例如下：

```
<img src="…" class="thumbnail">
<div class="row">
    <div class="col-sm-6 col-md-4">
        <div class="thumbnail">
            <img data-src="图片地址" alt="…">
            <div class="caption">
                <h3>Thumbnail label</h3>
                <p>…</p>
                <p>
                    <a href="#" class="btn btn-primary">Action</a>
                    <a href="#" class="btn btn-default">Action</a>
                </p>
            </div>
        </div>
    </div>
</div>
```

可以在.caption样式的元素容器内，添加任意风格的元素，比如按钮、链接等，效果如图4-69所示。

图4-69　带有caption样式的运行效果

注意

由于示例代码使用了col-sm-6和col-md-4样式，该样式在可视宽度大于768px的浏览器中才生效；如果浏览器是窄屏的话，该样式将失效，同时thumbnail样式的元素容器将充满全屏100%。

4.15 警告框

源码文件：alerts.less

CSS文件：bootstrap.css 4434行之后

在交互式网页中，经常要根据用户操作的上下文为用户提供灵活的提示消息，比如操作成功、警告提示、错误信息等。.alert样式为此提供了基础的操作样式。

4.15.1 默认警告框

默认的警告框是用带有alert样式的div元素容器包含的（p元素也可以），内部可选地添加一个关闭按钮button元素，只不过关闭按钮要确保设置button元素的属性值data-dismiss=“alert”，原因是警告框的关闭功能是通过JavaScript检测该属性实现的（具体原理在5.8节会讲到）。示例如下：

```
<div class=“alert”>
  <button type=“button” class=“close” data-dismiss=“alert”>×</button>
                                                    <strong>Warning!
</strong> Best check yo self, you’re not looking too good.
</div>
```

上述示例的运行效果如图4-70所示。

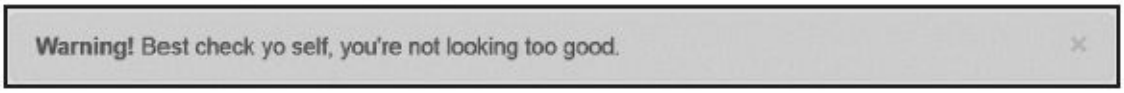


图4-70 默认警告框运行效果

alert样式的源码主要是设置了背景色、边框、角度和文字颜色，然后特殊处理了两种内部元素h4和hr，分别用于显示警告框中的标题和分隔符。源码如下：

```
// 源码4434行
.alert {                                                    /* 基本的警告框设
置 */
  padding: 15px;
  margin-bottom: 20px;
  border: 1px solid transparent;
  border-radius: 4px;
}
.alert h4 {  margin-top: 0;  color: inherit;}
.alert .alert-link {  font-weight: bold;        /* alert内的链接样式 */}
.alert > p,
.alert > ul {
  margin-bottom: 0; /* alert内的p元素和ul元素的底部外边距设置 */
}
.alert > p + p {  margin-top: 5px;              /*两个段落之间，增加一个顶
部外边距*/}
```

一个完整的警告框，在应用了h4和hr元素以后，看起来就有了一种酷炫的效果，如图4-71所示。

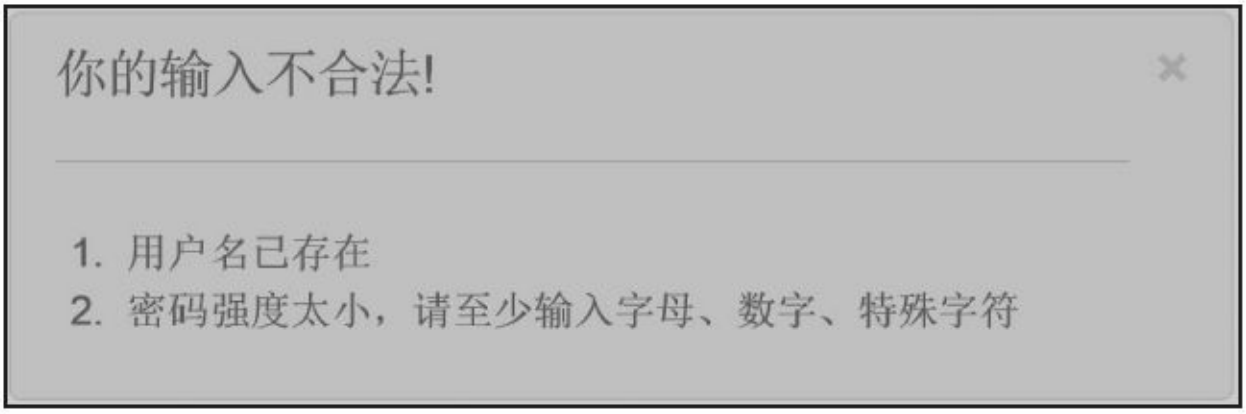


图4-71　完整的警告框运行效果

4.15.2　可关闭的警告框

在.alert样式上，再应用一个.alert-dismissable样式即可实现一个可关闭的警告框。使用方式如下：

```
<div class="alert alert-warning alert-dismissable">
<button   type="button"   class="close"   data-dismiss="alert"   aria-
hidden="true">×</
    button>
  <strong>Warning!</strong>
    <p>Best check yo self, you're not looking too good.</p>
</div>
```

运行效果如图4-72所示。

图4-72　可关闭的警告框

可关闭的警告框的实现方式主要是增加了.alert-dismissable容器右边的内边距，以便放置关闭符号。源码如下：

```
// 源码4454行
.alert-dismissable {              /* 警告框内，如果是可关闭的话 */
  padding-right: 35px;            /* 增大右内边距，以便放置关闭按钮*/
}
.alert-dismissable .close {       /* 警告框内，如果放置关闭按钮，特殊处理 */
  position: relative;
  top: -2px;
  right: -21px;                   /* 关闭按钮，再向右，以便右对齐 */
  color: inherit;
}
```

4.15.3　多彩样式警告框

和btn、label样式一样，警告框也提供了多种颜色的功能，在默认淡黄色的基础之上，另外提供了3种样式，分别是：alert-danger（危险红）、alert-success（成功绿）、alert-info（信息蓝）。示例代码如下：

```
<div class="alert alert-danger">…</div>
<div class="alert alert-success">…</div>
<div class="alert alert-info">…</div>
```

上述示例的运行效果如图4-73所示。

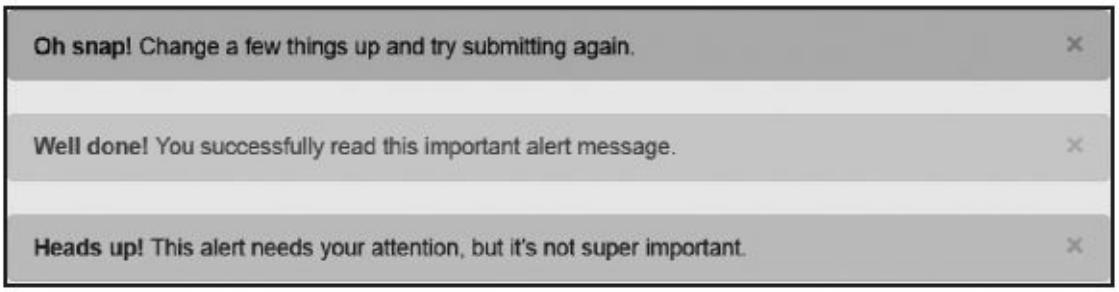


图4-73　彩色警告框

上述的3种样式分别对当前样式的文字颜色、背景颜色、边框颜色和内部分隔符颜色进行了相应的设置。其中alert-success的源码如下：

```
// 源码4463行
.alert-success {
  color: #3c763d;                                    /* 设置文字为绿色 */
  background-color: #dff0d8;                         /* 设置背景颜色 */
  border-color: #d6e9c6;                             /* 设置边框颜色 */
}
.alert-success hr {
  border-top-color: #c9e2b3;                         /* 设置分隔符颜色 */
}
.alert-success .alert-link {
  color: #2b542c;                                    /* 设置内部链接的文本颜色 */
}
```

大家可以模仿上述规则对其进行其他风格、颜色的设置。

4.15.4　警告框中的链接

警告框中的内容，如果放了链接的话，通常我们都希望链接高亮显示。Bootstrap在4种不同颜色的风格内都给链接颜色提供了高亮，应用的方式都一样，在alert容器元素内的a链接上应用.alert-link样式，从而达到想要的效果。示例代码如下：

```
<div class="alert alert-danger">
  <a href="#" class="alert-link">…</a>
</div>
<div class="alert alert-success">
  <a href="#" class="alert-link">…</a>
</div>
<div class="alert alert-info">
  <a href="#" class="alert-link">…</a>
</div>
```

上述示例的运行效果如图4-74所示。

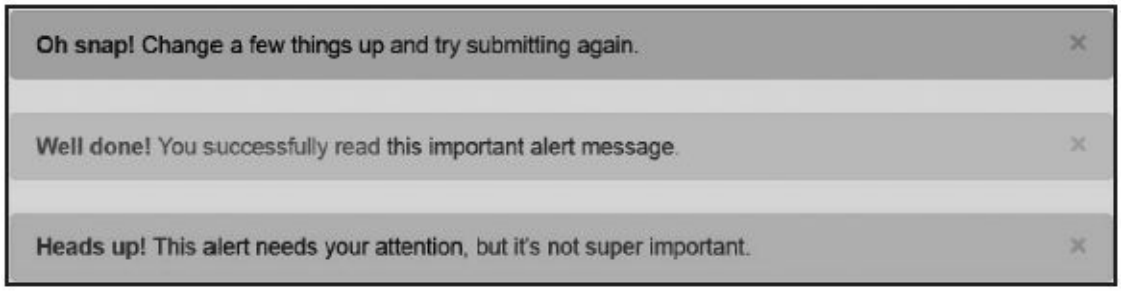


图4-74　带链接的警告框

Bootstrap在4种不同颜色的风格下，对链接的颜色都进行了相应的加深以及文字加粗。具体源码如下：

```
// 源码4471行
.alert .alert-link {  font-weight: bold;          /*文字加粗*/}
/* 以下代码对应4种颜色的风格 */
.alert-warning .alert-link {  color: #a47e3c;    /* 加深颜色 */}
.alert-success .alert-link {  color: #356635;    /* 加深颜色 */}
.alert-danger .alert-link {  color: #953b39;     /* 加深颜色 */}
.alert-info .alert-link {  color: #2d6987;       /* 加深颜色 */}
```

注意

根据前面的介绍可知、如果需要扩展自定义颜色的风格，除了设置新风格的文字颜色、背景颜色、边框颜色以外，还要设置分隔符（hr）元素的颜色和加深的链接颜色（alert-link）。

4.16　进度条

源码文件：progress-bars.less

CSS文件：bootstrap.css 4507行之后

进度条（Progress　bar）是一个比较常见的网页效果，一般用于表示加载、跳转或动作正在执行中的状态。针对进度条效果，Bootstrap提供了丰富多彩的样式可供选择，我们来一一讲解。

4.16.1　基本样式

基本的进度条在使用的时候只需要应用两个样式：progress和progress-bar。progress用于设置进度条的容器样式，而progress-bar用于限制进度条的进度（颜色进度）。示例代码如下：

```
<!—一般定义—>
<div class=“progress”>
    <div class=“progress-bar” style=“width: 60%;”></div>
</div>
<!— 带辅助阅读的定义—>
<div class=“progress”>
    <div class=“progress-bar” style=“width: 60%;”
        role=“progressbar” aria-valuenow=“60” aria-valuemin=“0” aria-
                    valuemax=“100”>
        <span class=“sr-only”>60% Complete</span>
    </div>
</div>
```

上述示例的运行效果如图4-75所示。

图4-75　进度条效果

progress样式主要是设置进度条容器的背景色、容器高度、间距等。源码如下：

```
// 源码4523行
.progress {
  height: 20px;                                          /* 进度条容器高
度 */
  margin-bottom: 20px;                                  /* 设置20像素的底
部外间距 */
  overflow: hidden;                                      /* 超出容器就自动
隐藏 */
  background-color: #f5f5f5;                             /* 容器背景色为淡
灰色 */
  border-radius: 4px;                                   /* 容器圆角 */
  -webkit-box-shadow: inset 0 1px 2px rgba(0, 0, 0, .1);         /* 容器
阴影 */
          box-shadow: inset 0 1px 2px rgba(0, 0, 0, .1);
}
```

progress-bar样式在设置进度方面，重要的是设置了进度条的颜色（即样式的背景色）和过渡效果。源码如下：

```
// 源码4532行
.progress-bar {
  float: left;                                          /* 左浮动 */
  width: 0;                                             /* 默认宽度是0 */
  height: 100%;                                          /* 高度100%自适
```

```
应 */
  font-size: 12px;
  line-height: 20px;
  color: #fff;
  text-align: center;
  background-color: #428bca;                                  /* 进度条颜色 */
  -webkit-box-shadow: inset 0 -1px 0 rgba(0, 0, 0, .15);          /* 阴影
处理 */
          box-shadow: inset 0 -1px 0 rgba(0, 0, 0, .15);
  -webkit-transition: width .6s ease;                           /* 0.6秒的过
渡 */
          transition: width .6s ease;                   /* 其他浏览器 */
}
```

4.16.2　多彩样式

不能配置多种颜色的进度条不是一个好的进度条，Bootstrap作者肯定也意识到了这一点，所以也提供了多颜色扩展功能。默认添加了4种颜色：　　progress-bar-info（信息蓝）、progress-bar-success（成功绿）、progress-bar-warning（警告黄）、progress-bar-danger（危险红）。示例代码如下：

```
<div class="progress">
    <div class="progress-bar progress-bar-success" style="width: 40%">
</div>
</div>
<div class="progress">
     <div class="progress-bar progress-bar-info" style="width: 20%">
</div>
</div>
<div class="progress">
    <div class="progress-bar progress-bar-warning" style="width: 60%">
</div>
</div>
<div class="progress">
    <div class="progress-bar progress-bar-danger" style="width: 80%">
</div>
</div>
```

上述示例的运行效果如图4-76所示。

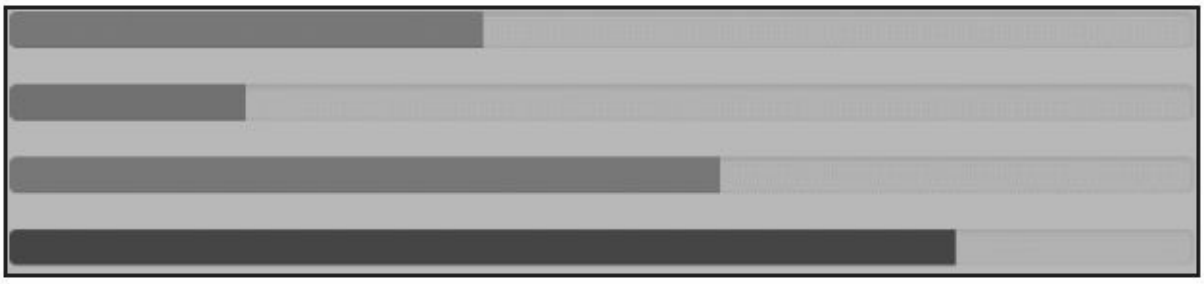


图4-76　彩色进度条效果

多彩样式的实现方式，只是改变了进度条的颜色（背景颜色）而已。源码如下：

```
// 源码4555行
.progress-bar-danger {  background-color: #d9534f;      /* 危险红 */}
.progress-bar-success {  background-color: #5cb85c;     /* 成功绿 */}
.progress-bar-warning {  background-color: #f0ad4e;     /* 警告黄 */}
.progress-bar-info {  background-color: #5bc0de;        /* 信息蓝 */}
```

4.16.3　条纹样式

与其他组件只提供多种颜色不同，Bootstrap还为进度条提供了一种条纹样式，并且不同的颜色可以显示不同的条纹。示例用法如下：

```
<div class="progress progress-striped">
      <div class="progress-bar progress-bar-info" style="width: 20%">
</div>
</div>
<div class="progress progress-striped">
     <div class="progress-bar progress-bar-success" style="width: 40%">
</div>
</div>
<div class="progress progress-striped">
     <div class="progress-bar progress-bar-warning" style="width: 60%">
</div>
</div>
<div class="progress progress-striped">
      <div class="progress-bar progress-bar-danger" style="width: 80%">
</div>
</div>
```

上述示例的运行效果如图4-77所示。

图4-77　条纹进度条效果

通过示例代码和运行效果可以看出，只是在容器元素上附加了一个新的progress-striped样式，该样式的实现是利用CSS3的线性渐变特性linear-gradient来实现的。源码如下：

```
// 源码4546行
.progress-striped .progress-bar {       /* 下面是根据不同的浏览器进行线性渐变的设置，6个间隔的设置 */
                              background-image:             -webkit-linear-gradient(45deg, rgba(255, 255, 255, .15)
      25%, transparent 25%, transparent 50%, rgba(255, 255, 255, .15) 50
      rgba(255, 255, 255, .15) 75%, transparent 75%, transparent);
                                  background-image:                linear-gradient(45deg, rgba(255, 255, 255, .15) 25%,
      transparent 25%, transparent 50%, rgba(255, 255, 255, .15) 50%,
      rgba(255, 255, 255, .15) 75%, transparent 75%, transparent);
  background-size: 40px 40px;   /* 以每40像素为一个块进行背景显示 */
}
```

而其他颜色下的条纹样式，因为都已经设置了颜色，所以也会相应地变化。

Bootstrap在上述4个背景颜色设置的时候也重复了线性渐变的代码，但其实是没必要的，因为.progress-striped .progress-bar样式已经设置了，而有了其他颜色设置的时候，这两个颜色也是存在的，所以不需要多此一举。

4.16.4 动画样式

在提供了条纹样式以后，Bootstrap还提供了一个更炫的动画样式，即：让条纹动起来。所需要做的只是在进度条容器元素上再多附加一个active样式即可。示例如下：

```
<div class="progress progress-striped active">
  <div class="progress-bar" style="width: 45%"></div>
</div>
```

实现动画样式的源码如下：

```
// 源码4551行
.progress.active .progress-bar {
   -webkit-animation: progress-bar-stripes 2s linear infinite;/* 2秒过渡40像素，无限循环 */
           animation: progress-bar-stripes 2s linear infinite;/* 其他浏览器 */
}
```

上述代码在应用时定义了动画过渡，而通过线性渐变可以看出每40个像素为一个区间，所以该动画过渡也是40个像素，2秒完成，然后无限循环。

不同的浏览器定义的动画过渡的代码如下：

```
// 源码4507行
@-webkit-keyframes progress-bar-stripes {       /* Webkit浏览器 */
  from {    background-position: 40px 0;  }
  to {    background-position: 0 0;  }
}
@keyframes progress-bar-stripes {               /* 其他浏览器 */
  from {    background-position: 40px 0;  }
  to {    background-position: 0 0;  }
}
```

注意

.active样式必须和.progress-striped样式一起应用才行。而IE浏览器只有IE10以上才支持动画过渡效果。

4.16.5　堆叠样式

何为堆叠样式？先看以下示例代码：

```
<div class=“progress”>
   <div  class=“progress-bar  progress-bar-success”  style=“width:  35%”>
</div>
   <div  class=“progress-bar  progress-bar-warning”  style=“width:  20%”>
</div>
    <div  class=“progress-bar  progress-bar-danger”  style=“width:  10%”>
</div>
</div>
```

上述示例的运行效果如图4-78所示。

图4-78　堆叠式进度条

奇怪，没有特殊的附加样式，却可以实现这么好看的效果，为什么呢？再次回到progress-bar样式的源码。

```
// 源码4532行
.progress-bar {
  float: left; /* 左浮动 */
   /* 这里省略了其他代码 */
}
```

明白了吧？已经做好铺垫了，在左浮动的基础上就可以实现上述效果。不过总宽度不要超过100%。

注意

进度条依赖于CSS3的渐变、过渡和动画特性，这些都是不被IE7～IE9或较老版本的Firefox所支持的。另外IE10和Opera　12之前的版本均不支持动画特性。

4.17　媒体对象

源码文件：media.less

CSS文件：bootstrap.css 4583行之后

媒体对象（Media　object）这是一个抽象的样式，用以构建不同类型的组件，这些组件都具有在文本内左对齐或右对齐的图片（就像blog内容或Tweets等），默认样式是在内容区域的左侧或右侧浮动一个媒体对象（图片、视频、音频）。

4.17.1 默认样式

一组媒体的默认使用方式通常包括如下几个样式：media、media-object、media-body、media-heading 4个样式，和一个用于控制左右浮动的pull-left（或pull-right）样式。示例代码如下：

```
<div class="media">
    <a class="pull-left" href="#">
        <img class="media-object" src="…" />
    </a>
    <div class="media-body">
        <h4 class="media-heading">Media heading</h4>
        …
        <!— 嵌套的media对象 —>
        <div class="media">
            …
        </div>
    </div>
</div>
```

上述示例的运行效果如图4-79所示。

媒体对象的样式相对比较简单，主要是设置各个样式的间距大小和左右浮动。源码如下：

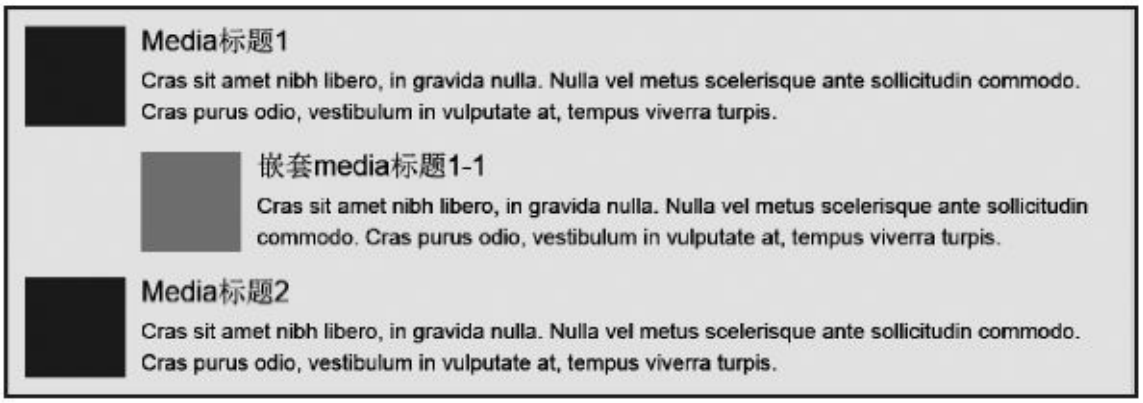


图4-79 媒体对象运行效果

```
// 源码4583行
.media,
.media-body {
  overflow: hidden;                    /* 超出显示区域则隐藏 */
  zoom: 1;
}
.media,
.media .media {
  margin-top: 15px;                    /* 设置每个媒体对象（或嵌套媒体）的顶部间距 */
}
.media:first-child { margin-top: 0;     /* 第一个媒体对象，不设置顶部间距 */ }
.media-object { display: block;          /* 针对的媒体对象object设置为块状显示 */ }
```

```
.media-heading {  margin: 0 0 5px;        /* 媒体对象标题，设置5像素的底部间距 */ }
.media > .pull-left {  margin-right: 10px;       /* 左浮动，保留10像素的右间距 */ }
.media > .pull-right {  margin-left: 10px;       /* 右浮动，保留10像素的左间距 */ }
```

4.17.2　媒体列表

如果需要将多个媒体进行列表展示的话，则可以利用在ul上应用media-list样式、li上应用media样式来实现。示例代码如下：

```
<ul class=“media-list”>
    <li class=“media”></li>
    <li class=“media”></li>
    <li class=“media”></li>
</ul>
```

上述示例的运行效果如图4-80所示。

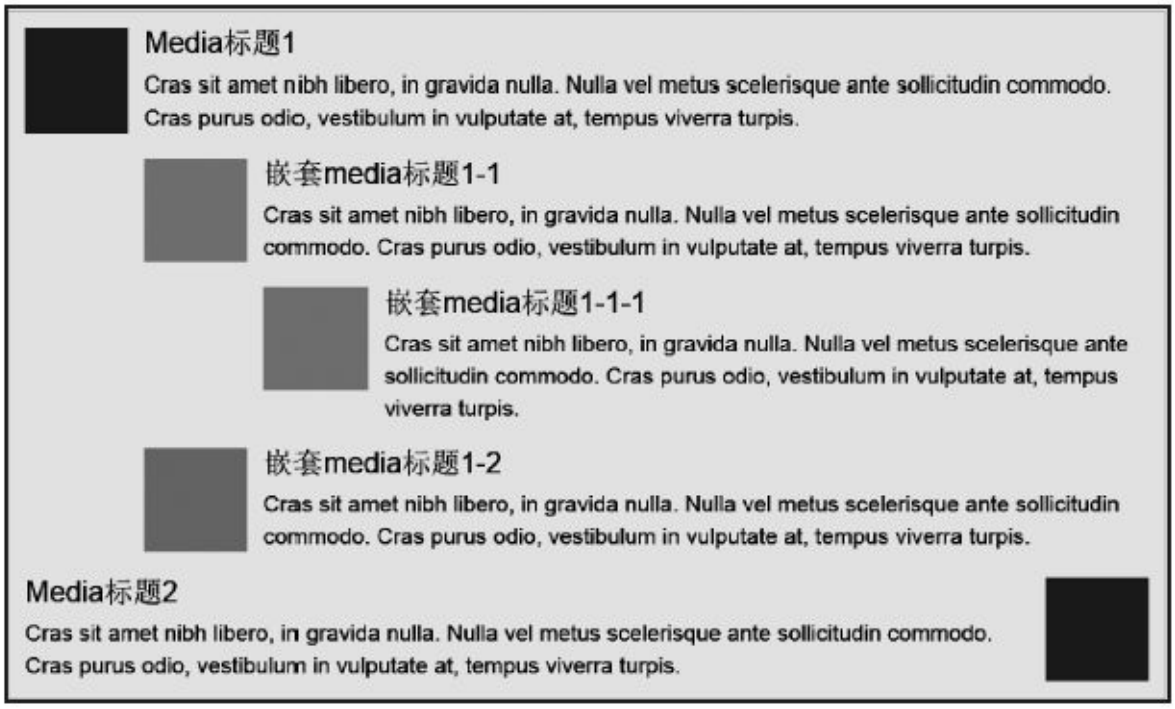


图4-80　媒体对象嵌套显示效果

媒体列表样式进行了两项最简单的设置：不保留左间距，并且去除li标记符。源码如下：

```
// 源码4607行
.media-list {
  padding-left: 0;       /* 不保留左间距 */
  list-style: none;      /* 媒体列表，去除li标记符 */
}
```

4.18　列表组

源码文件：list-group.less

CSS文件：bootstrap.css 4611行之后

在4.6.3小节，我们在讲解导航的时候，曾经在nav-stacked样式的基础上，实现了堆叠式导航。这种堆叠式导航样式就是本小节要讲解的列表组（List group），其是Bootstrap在新版提供的新组件。

4.18.1　基础列表组

大部分的列表组都是由无标记的列表（ul/li）来实现，然后通过应用特定的样式实现效果。先看以下示例代码：

```
<ul class=“list-group”>
  <li class=“list-group-item”>JavaScript编程精解</li>
  <li class=“list-group-item”>JavaScript设计模式</li>
  <li class=“list-group-item”>JavaScript启示录</li>
  <li class=“list-group-item”>当前你正在读的书</li>
  <li class=“list-group-item”>正在构想的书</li>
</ul>
```

上述示例代码的运行效果如图4-81所示。

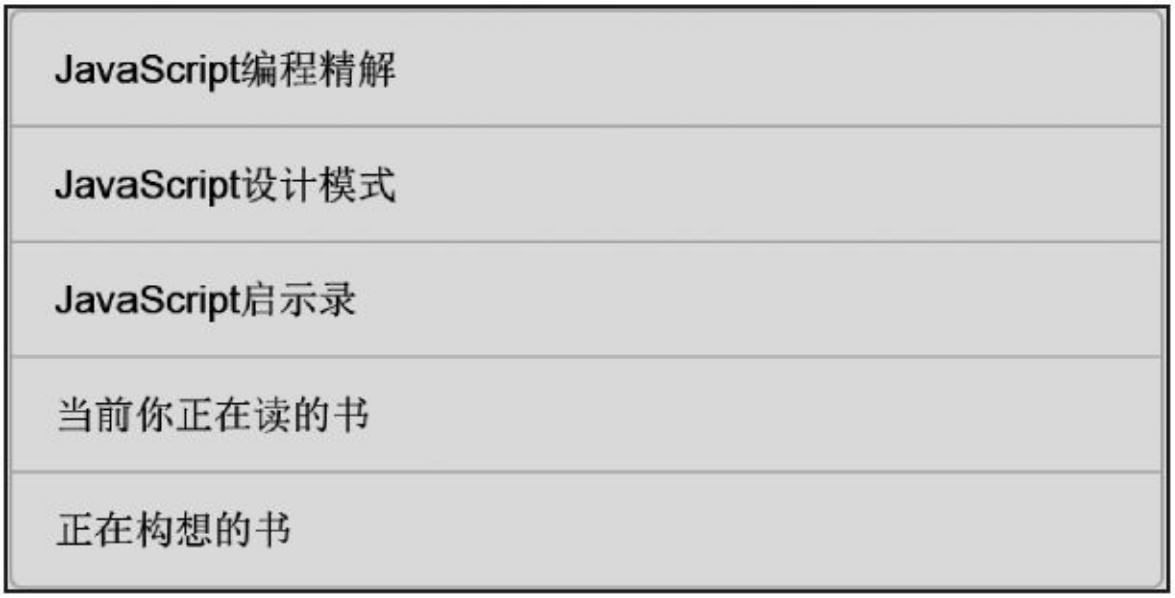


图4-81　列表组运行效果

根据上述的示例可以看出，列表组有2个基本的样式：list-group和list-group-item。这两个样式主要是设置了基本的显示和布局内容，比如间距、上下的圆角等。源码如下：

```
// 源码4611行
.list-group {
  padding-left: 0;                /* 消除左边距*/
  margin-bottom: 20px;            /* 底部间距保留20像素 */
}
.list-group-item {
  position: relative;             /* 相对布局 */
  display: block;                       /* 块状显示 */
  padding: 10px 15px;
  margin-bottom: -1px;
  background-color: #fff;               /* 默认全部列表的背景颜色为白色 */
  border: 1px solid #ddd;               /* 1像素边框 */
}
.list-group-item:first-child {   /* 设置最上边的一个li元素的左上角和右上角为圆角 */
```

```
  border-top-left-radius: 4px;
  border-top-right-radius: 4px;
}
.list-group-item:last-child {
  margin-bottom: 0;              /* 取消最下边的一个li元素底部外边距 */
   border-bottom-right-radius: 4px;       /* 设置最下边的一个li元素左下角和右下角为圆角 */
  border-bottom-left-radius: 4px;
}
```

4.18.2　应用徽章标记或导航箭头

在列表组的li元素上，我们也可以应用一些徽章图标。具体的做法是在li元素内部放置标记小图标的span元素（或其他内敛元素label、i）即可。示例代码如下：

```
<!– 徽章标记 –>
<ul class=“list-group”>
  <li class=“list-group-item”>
    <span class=“badge”>14</span>
    Cras justo odio
  </li>
</ul>
```

上述示例的运行效果如图4-82所示。

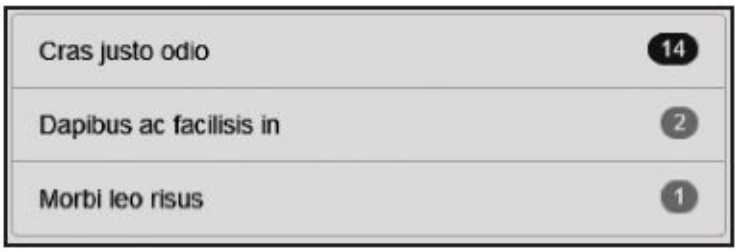


图4-82　列表组上应用徽章

图4-82所示是列表组上应用徽章的运行效果，主要是要对小图标设置浮动方向及右间距。源码如下：

```
// 源码4632行
.list-group-item > .badge {
  float: right;          /* 设置小图标 右浮动 */
}
.list-group-item > .badge + .badge {
  margin-right: 5px;     /* 如果两个图标放在一起显示，则保留5像素的右间距 */
}
```

在2.x版本里还提供了导航箭头图标，效果如图4-83所示。但这个图标在新版里被去除了，如果还想保留这个功能，可以将旧版的代码移植过来，再做少许修改即可。

图4-83　列表组上应用导航箭头

示例代码如下：

```
<!– 徽章标记 –>
```

```
<!— 导航标记 —>
<ul class=“list-group”>
    <li class=“list-group-item”>
        <span class=“glyphicon glyphicon-chevron-right”></span>
        Cras justo odio
    </li>
</ul>
```

相应的导航箭头图标实现源码如下：

```
.list-group-item > .glyphicon-chevron-right {
    float: right;                /* 设置小图标 右浮动 */
    margin-right: 5px;  /* 小图标在span、label、i元素上设置-15像素的右间距 */
}
.list-group-item > .glyphicon + .badge {
    margin-right: 5px;  /* 如果两个图标放在一起显示，则保留5像素的右间距 */
}
```

根据上述源码，可以看出，一个列表项里，两个或者多个图片是可以放在一起的，但是要注意两者之间的间距。读者可以自行运行查看相应的效果。

4.18.3　可链接的列表组

虽然可以在列表项的文字上链接，但有时候需要整个列表项元素都可以被单击（比如li元素），这就需要另外一种使用方式了。

主要的不同方式是用普通的div元素代替ul元素，用a链接元素代替li元素，并在a元素上应用list-group-item样式，即可达到所想的效果。添加附加样式active的话，则会让该列表项高亮显示。示例代码如下：

```
<div class="list-group">
        <a  href="#"  class="list-group-item  active">JavaScript编程精解
<span class=
        "badge">1</span></a>
    <a href="#" class="list-group-item">JavaScript设计模式 <span class=
        "badge">2</span></a>
    <a href="#" class="list-group-item">JavaScript启示录 <span class=
        "badge">3</span> </a>
                <a    href="#"    class="list-group-item">当前你正在读的书
<span class="badge">4</span></a>
                  <a    href="#"    class="list-group-item">正在构思的书
<span class="badge">5</span></a>
</div>
```

上述示例的运行效果如图4-84所示。

图4-84　列表组内放置链接时的运行效果

与默认的ul、li元素不同，使用a元素链接时，样式需要有特殊的设置，比如：不设置下划线，高亮时设置文字颜色和背景色。源码如下：

```
// 源码4638行
a.list-group-item {
  color: #555;                                /*链接的文本颜色*/
}
a.list-group-item .list-group-item-heading {
  color: #333;                                /*如果链接内有heading元素，则设置文
本颜色*/
}
a.list-group-item:hover,
a.list-group-item:focus {                     /*链接在移动（或焦点）时的设置*/
  text-decoration: none;                              /* 链接不设置下划线 */
  background-color: #f5f5f5;                  /* 背景色 */
```

```
}
a.list-group-item.active,
a.list-group-item.active:hover,
a.list-group-item.active:focus {          /*链接被设置为当前活动（高亮）时的设
置*/
  z-index: 2;
  color: #fff;                            /* 高亮的文字是白色，背景是深色 */
  background-color: #428bca;
  border-color: #428bca;
}
```

4.18.4　多彩列表项

自V3.1.0开始，列表组中的每项也可以自定义背景颜色了。该组件提供了类似list-group-item-succes的样式用于指定背景颜色，同时也支持高亮样式active。示例用法如下：

```
<ul class="list-group">
                    <li     class="list-group-item     list-group-item-
success active">JavaScript编程精解</li>
   …
</ul>
<div class="list-group">
              <a    href="#"    class="list-group-item    list-group-item-
success active">JavaScript
       编程精解</a>
    …
</div>
```

上述示例的运行效果如图4-85所示。

图4-85　列表项的彩色效果

列表项彩色效果的实现原理相对比较简单，分为两种情况：一种是使用ul和li元素的列表组，只需要设置文字颜色和背景色即可；另外一种是a链接作为列表项时的设置，这种设置有点特殊，就是需要处理链接在hover和focus焦点时的样式。详细源码如下：

```
// 源码 4667 行
.list-group-item-success {
  color: #3c763d;                          /* 文字颜色 */
  background-color: #dff0d8;     /* 背景颜色 */
}
a.list-group-item-success {
  color: #3c763d;                         /* a元素作为列表项时的文字颜色 */
}
a.list-group-item-success .list-group-item-heading {
  color: inherit;                          /* a元素作为列表项时，heading的文字
颜色集成上述颜色 */
}
a.list-group-item-success:hover,
a.list-group-item-success:focus {         /* a元素作为列表项时，鼠标移动和焦点
时的文字和背景色设置 */
```

```
  color: #3c763d;
  background-color: #d0e9c6;
}
a.list-group-item-success.active,
a.list-group-item-success.active:hover,
a.list-group-item-success.active:focus {
                                        /* a元素作为列表项，同时又是高亮元素
时的相关颜色设置 */
  color: #fff;
  background-color: #3c763d;
  border-color: #3c763d;
}
```

从最后一段源码可以看出，a元素作为列表项时，如果是高亮状态（应用了active样式），则会重新再设置一种比较深的同色系颜色，其运行效果如图4-86所示。

图4-86　列表项为a元素时的效果（第一项是高亮状态）

4.18.5　自定义列表组

在可链接的列表组基础上，Bootstrap又提供了list-group-item-heading和list-group-item-text这两个样式，用于开发人员自定义列表项里的具体内容，其表示的意思分别是：列表项条目的头部（heading）和主要内容（text）。示例代码如下：

```
<div class="list-group">
    <a class="list-group-item active" href="#">
        <h4 class="list-group-item-heading">JavaScript编程精解</h4>
        <p class="list-group-item-text">……</p>
    </a>
    <a class="list-group-item" href="#">
        <h4 class="list-group-item-heading">JavaScript设计模式</h4>
        <p class="list-group-item-text">……</p>
</a>
<a class="list-group-item" href="#">
        <h4 class="list-group-item-heading">JavaScript启示录</h4>
        <p class="list-group-item-text">……</p>
    </a>
</div>
```

上述示例的运行效果如图4-87所示。

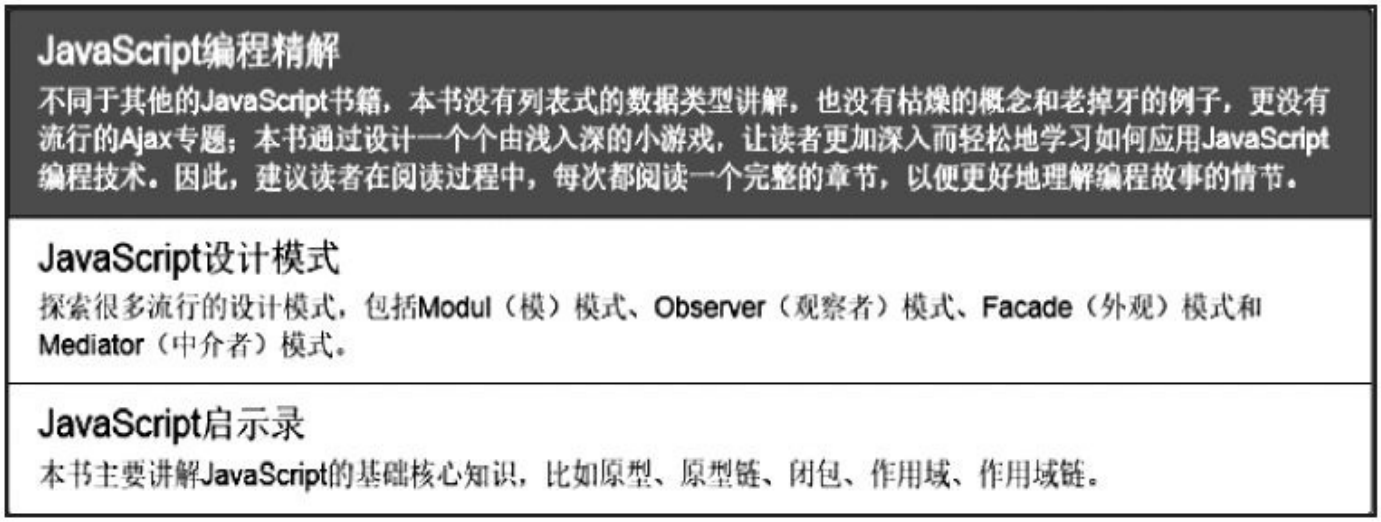


图4-87　列表组内放置heading和text时的运行效果

两个样式主要是控制在普通状态和高亮状态时的文字（heading）颜色和内容（text）颜色。源码如下：

```
// 源码4657行
a.list-group-item.active .list-group-item-heading,
a.list-group-item.active:hover .list-group-item-heading,
a.list-group-item.active:focus .list-group-item-heading {
  color: inherit;         /*链接被设置为当前活动（高亮）时，内部的heading元素继承父元素的设置*/
}
a.list-group-item.active .list-group-item-text,
a.list-group-item.active:hover .list-group-item-text,
```

```
a.list-group-item.active:focus .list-group-item-text {
  color: #e1edf7;         /*链接被设置为当前活动（高亮）时，内部的text元素的文本颜色*/
}
// 源码4755行
.list-group-item-heading {      /*默认情况下的heading设置*/
  margin-top: 0;
  margin-bottom: 5px;
}
.list-group-item-text {         /*默认情况下的text设置*/
  margin-bottom: 0;
  line-height: 1.3;
}
```

4.19　面板

源码文件：panels.less

CSS文件：bootstrap.css 4763行之后

面板（Panel）组件是Bootstrap在新版中提供的另一个新组件，主要是用于处理其他通用组件处理不了的功能。先来看一下一般面板所实现的效果，如图4-88所示。

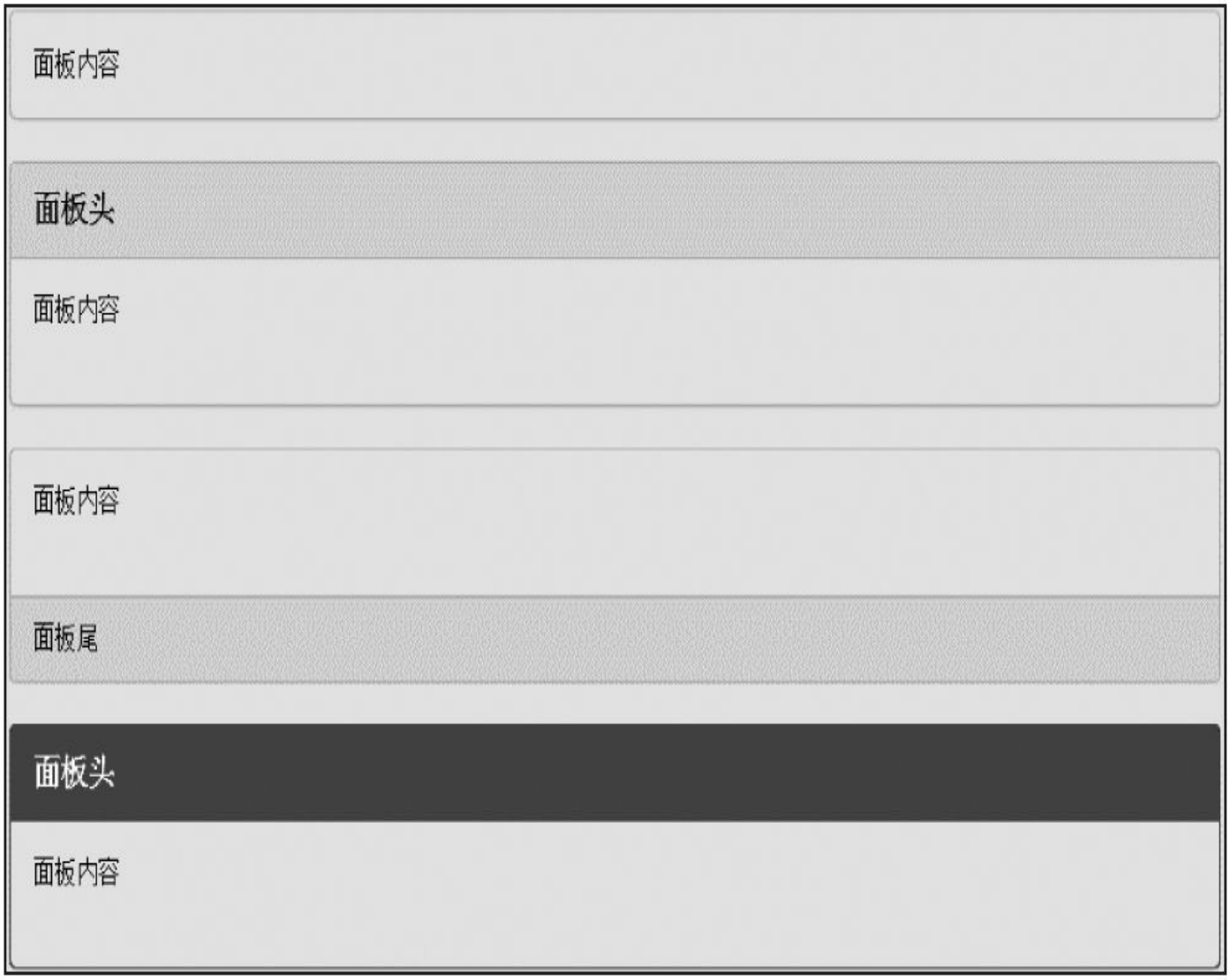


图4-88　面板效果

4.19.1　基础面板

基础的面板很简单，就是简单在div上应用panel，产生一个具有边框的文本显示框。示例代码如下：

```
<div class=“panel panel-default”>
    <div class=“panel-body”>
        这里是面板内容
    </div>
</div>
```

上述示例的运行效果如图4-89所示。

图4-89　基础面板效果

由示例可以看出，pannel样式不控制主题颜色，pannel-default才是控制颜色的主题，而且还多了一个pannel-body样式，专门用于存放文本内容，这一点和2.x版本的不太一样了。先来看看panel样式，主要是对边框、外部边距、圆角进行了设置。源码如下：

```
// 源码4763行
.panel {
  margin-bottom: 20px;           /* 设置底部边距 */
  background-color: #fff;        /* 白色背景 */
  border: 1px solid transparent;          /* 透明边框 , 颜色在后面根据主题进行
设置*/
  border-radius: 4px;            /* 边框圆角弧度 */
  -webkit-box-shadow: 0 1px 1px rgba(0, 0, 0, .05);  /* 阴影 */
          box-shadow: 0 1px 1px rgba(0, 0, 0, .05);
}
.panel-body {
  padding: 15px;                           /* 设置正文的内边距 */
}
```

4.19.2　带有头和尾的面板

基础面板看着稍微有点寒酸，所以Bootstrap作者又为其定义了面板头（panel-heading）和面板尾（panel-footer）样式，其功能是高亮显示相对应的面板头和面板尾。示例代码下：

```
<div class=“panel panel-default”>
    <div class=“panel-heading”>面板header</div>
<div class=“panel-body”>
        这里是面板内容
    </div>
    <div class=“panel-footer”>面板footer</div>
</div>
```

上述示例的运行效果如图4-90所示。

图4-90　带头部和尾部的面板效果

两个样式分别对边框、内外边距、背景色、圆角、字体大小、字体粗细进行了相应的设置。源码如下：

```
// 源码4914行
.panel-heading {                          /* 面板头部相关设置 */
  padding: 10px 15px;
  border-bottom: 1px solid transparent;
  border-top-left-radius: 3px;
  border-top-right-radius: 3px;
}
.panel-heading > .dropdown .dropdown-toggle {
  color: inherit;                         /* 面板头部，如果有下拉菜单，继承父元
素设置 */
}
.panel-title {                    /* 面板头部，放title标题，去除上下外边距，加
大字体 */
  margin-top: 0;
  margin-bottom: 0;
  font-size: 16px;
  color: inherit;
}
.panel-title > a {
  color: inherit;
```

```
}
.panel-footer {                 /* 面板底部相关设置 */
  padding: 10px 15px;
  background-color: #f5f5f5;    /* 面板底部背景色 */
  border-top: 1px solid #ddd;
  border-bottom-right-radius: 3px;
  border-bottom-left-radius: 3px;
}
```

通过上述代码，我们可以看出，底部footer设置了背景色，但是head却没有设置，这是因为头部head是跟随主题风格来设置的（比如pannel-default）。

4.19.3　多彩面板

学习了那么多组件，发现几乎所有的组件都提供了多彩风格的显示，面板也不例外，分别提供了5种颜色：panel-primary（重点蓝）、panel-success（成功绿）、panel-warning（警告黄）、panel-danger（危险红）、panel-info（信息蓝）。

使用方式也非常简单，只需要在现有panel的样式上，附加一个颜色样式即可。示例代码如下：

```
<div class="panel panel-primary">…</div>
<div class="panel panel-success">…</div>
<div class="panel panel-warning">…</div>
<div class="panel panel-danger">…</div>
<div class="panel panel-info">…</div>
```

上述示例的运行效果如图4-91所示。

首先对整个面板的边框进行设置，然后对面板头部进行相关的设置：比如头部内的文字颜色、背景颜色和头部本身的边框颜色。panel-default样式的源码如下（其他样式都类似）：

```
// 源码4962行
.panel-default {
  border-color: #ddd;            /* 设置面板边框颜色 */
}
.panel-default > .panel-heading {        /* 面板头部相关的设置 */
  color: #333;                   /* 文字颜色 */
  background-color: #f5f5f5;     /* 背景颜色 */
  border-color: #ddd;            /* 面板头的边框颜色 */
}
.panel-default > .panel-heading + .panel-collapse .panel-body {
  border-top-color: #ddd;        /* 如果是折叠面板的话，设置顶部边框的颜色*/
}
.panel-default > .panel-footer + .panel-collapse .panel-body {
  border-bottom-color: #ddd;     /* 如果是折叠面板的话，设置footer的底部边框颜色*/
}
```

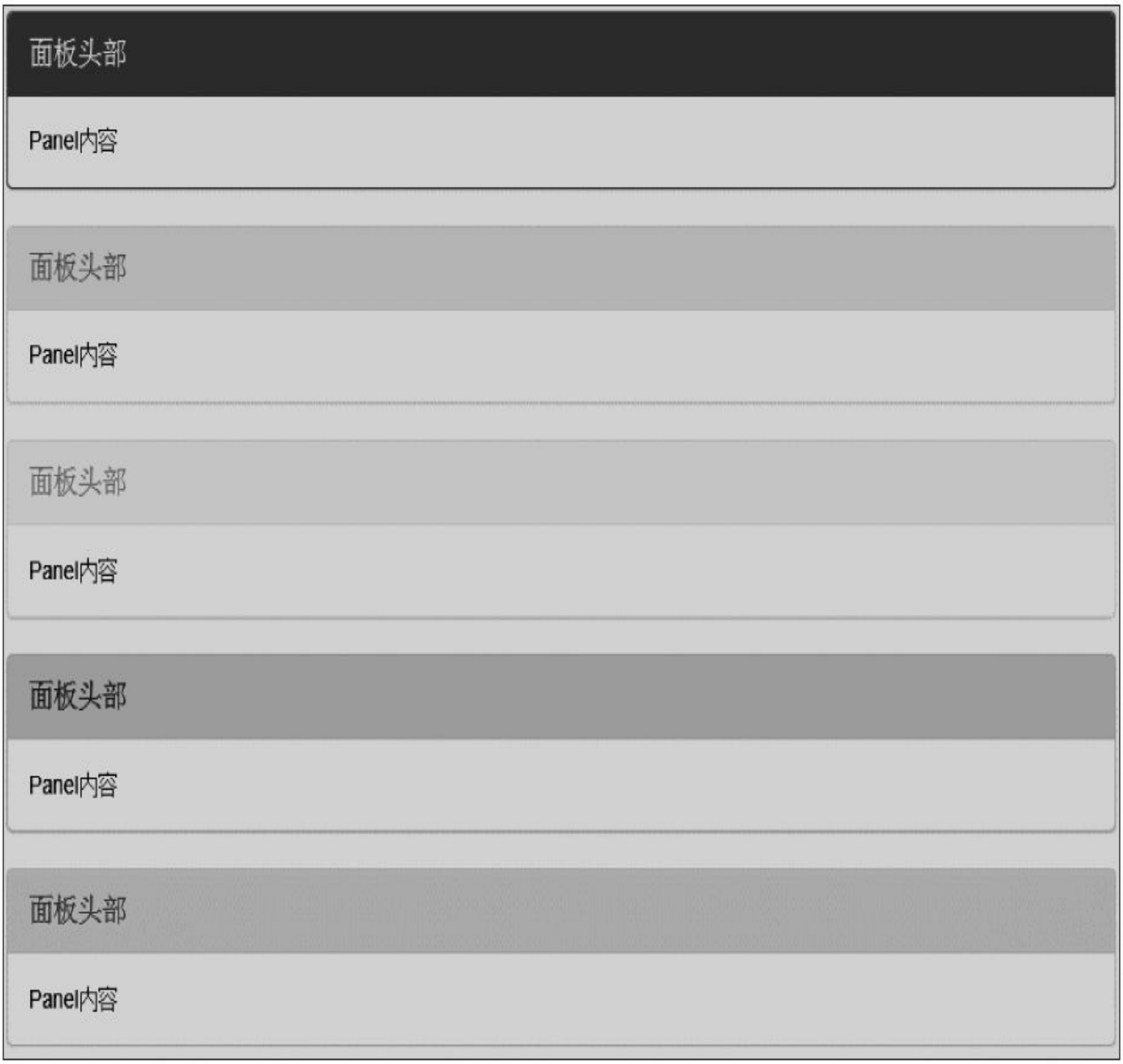


图4-91　彩色面板效果

折叠面板将在第5章的5.10节细讲。

4.19.4　面板与表格进行嵌套

一般在使用面板的时候，往往可能会在主区域（panel-body）内放很多内容，比如图片、表格等。来看一下放表格的效果，示例代码如下：

```
<div class=“panel panel-default”>
    <div class=“panel-heading”>Panel heading</div>
    <div class=“panel-body”>
        <p>…</p>
        <table class=“table table-bordered”>
            …
        </table>
    </div>
</div>
```

上述示例的运行效果如图4-92所示。

图4-92　面板与表格嵌套时的效果

可以看到，由于panel-body有15像素的内边距，所以内部的表格边框并没有和外部的panel紧靠在一起，如果想紧靠在一起，则只要将table元素和panel-body所在的div元素并列放就可以了。示例代码如下：

```
<div class=“panel panel-default”>
    <div class=“panel-heading”>Panel heading</div>
    <div class=“panel-body”>
        <p>…</p>
```

```
    </div>
    <table class="table table-bordered">
        …
    </table>
</div>
```

上述示例的运行效果如图4-93所示。

这样看起来是不是很完美？你可能会有疑问：table-bordered样式本来是有边框的，这样放的话，应该会有边框重合的粗边框才对啊。是的，原本是有粗线的，但是Bootstrap作者专门又对这种场景进行了优化（即：去除一个边框）。源码如下：

```
// 源码4798行
.panel > .table,
.panel > .table-responsive > .table {
  margin-bottom: 0;      /*使用panel时，将表格的底部外边距设置为0，以便重合靠近
*/
}
.panel > .panel-body + .table,
.panel > .panel-body + .table-responsive {
  border-top: 1px solid #ddd;
    /*使用panel时，如果表格和panel-body并列放，将表格的顶部边框设置为1像素，以便和
panel-body
    有分隔*/
}
.panel > .table > tbody:first-child > tr:first-child th,
.panel > .table > tbody:first-child > tr:first-child td {
  border-top: 0;                  /*将tbody区域的第一个th或td的顶部边框设置为
0，避免有水平粗线*/
}
.panel > .table-bordered,
.panel > .table-responsive > .table-bordered {
  border: 0;              /*使用panel时，将表格的table-bordered样式的外边框设
置为0，
                            但tr和td依然有边框*/
}
.panel > .table-bordered > thead > tr > th:first-child,
.panel > .table-responsive > .table-bordered > thead > tr > th:first-
child,
.panel > .table-bordered > tbody > tr > th:first-child,
.panel > .table-responsive > .table-bordered > tbody > tr > th:first-
child,
.panel > .table-bordered > tfoot > tr > th:first-child,
.panel > .table-responsive > .table-bordered > tfoot > tr > th:first-
child,
.panel > .table-bordered > thead > tr > td:first-child,
.panel > .table-responsive > .table-bordered > thead > tr > td:first-
child,
.panel > .table-bordered > tbody > tr > td:first-child,
.panel > .table-responsive > .table-bordered > tbody > tr > td:first-
child,
.panel > .table-bordered > tfoot > tr > td:first-child,
.panel > .table-responsive > .table-bordered > tfoot > tr > td:first-
child {
```

```
  border-left: 0;                    /*将左边框设置为0，以便和左边的panel重合靠紧
*/
}
……
/*将右边框设置为0，以便和panel的右边重合靠紧*/
/*将下边框设置为0，以便和panel的下边框重合靠紧*/
```

我的图书

Some default panel content here. Nulla vitae elit libero, a pharetra augue. Aenean lacinia bibendum nulla sed consectetur. Aenean eu leo quam. Pellentesque ornare sem lacinia quam venenatis vestibulum. Nullam id dolor id nibh ultricies vehicula ut id elit.

#	图书名称	出版社	作译者
1	JavaScript编程精解	机械工业出版社	汤姆大叔（译）
2	JavaScript设计模式	人民邮电出版社	汤姆大叔（译）
3	JavaScript启示录	人民邮电出版社	汤姆大叔（译）
4	深入理解Bootstrap	机械工业出版社	汤姆大叔

图4-93　表格充满面板时的运行效果

而如果去除了.table-bordered样式，仅仅使用.table样式的话，效果也是很好的，如图4-94所示。

图4-94　面板与table-bordered样式的完美融合

最后，还有一个场景就是，如果不用panel-body样式的元素，只放head元素和表格元素，效果也是极好的。示例代码如下：

```
<div class=“panel panel-default”>
    <div class=“panel-heading”>我的图书</div>
    <table class=“table”>
        …
    </table>
</div>
```

上述示例的运行效果如图4-95所示。

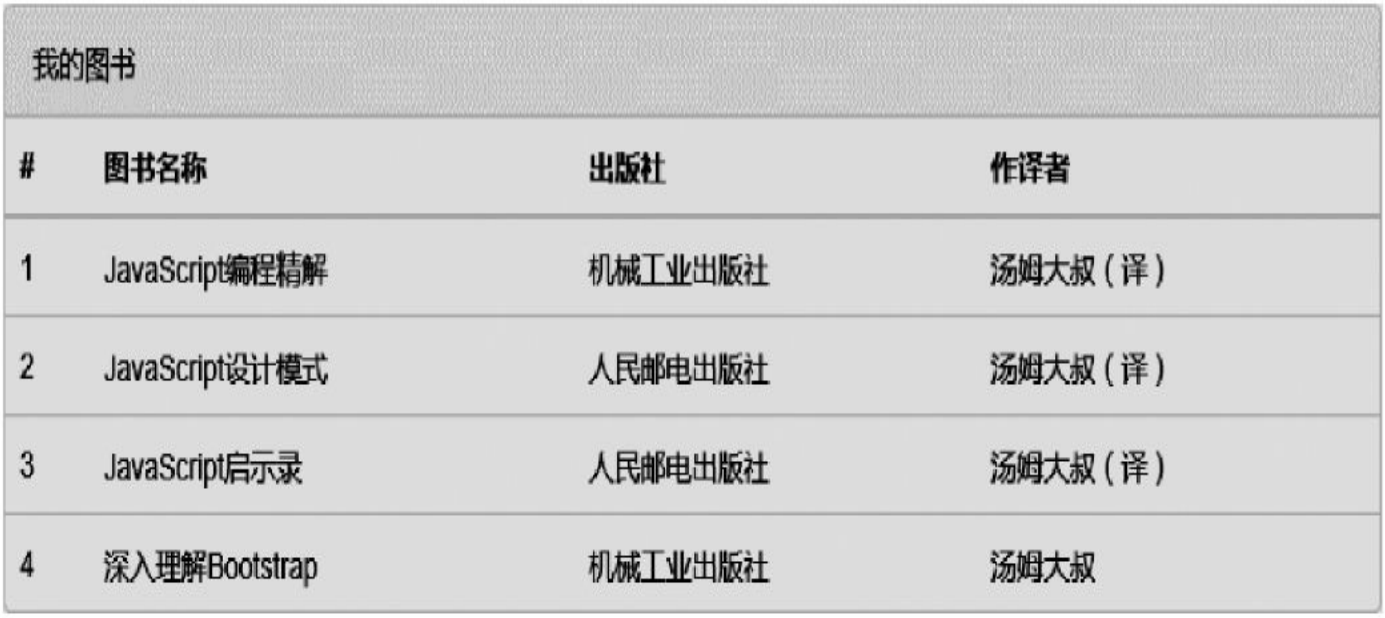


图4-95　只有head和表格时的运行效果

4.19.5　面板和列表组进行嵌套

了解了面板和表格的嵌套之后，结合前面学到的列表组，发现如果将两者结合在一起的话，会产生另外一种类似百叶窗的效果。示例用法如下：

```
<div class="panel panel-success">
    <!— 默认面板 —>
    <div class="panel-heading">汤姆大叔的书目</div>
    <p>
        <br />以下列表是汤姆大叔在2012、2013年所出版的图书，敬请大家阅读，多提建
议。<br /><br />
    </p>
    <!— 列表组 —>
    <ul class="list-group ">
        <li class="list-group-item">JavaScript编程精解</li>
        <li class="list-group-item">JavaScript设计模式</li>
        <li class="list-group-item">JavaScript启示录</li>
        <li class="list-group-item">您正在读的本书</li>
        <li class="list-group-item">正在构思的书</li>
    </ul>
</div>
```

上述示例的运行效果如图4-96所示。

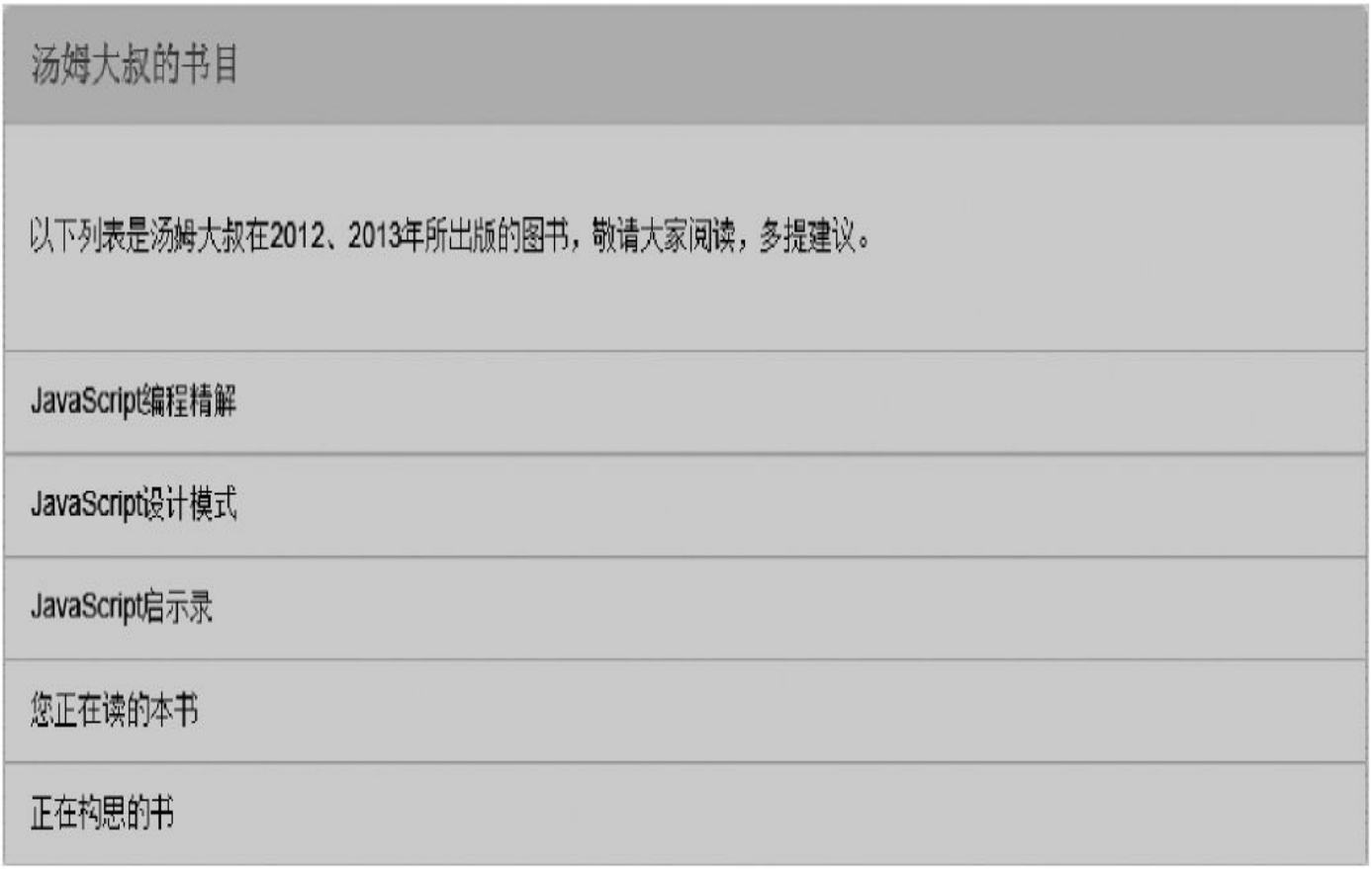


图4-96　面板和列表组结合的运行效果

和表格的处理方式一样，列表组在面板里使用时，Bootstrap作者也

进行了微调，主要是调整边框重合的问题，即如果两个组件都有底部边框，那么就取消一个。相关源码如下：

```
// 源码4774行
.panel > .list-group {
  margin-bottom: 0;     /*将底部的外边距设置为0，以便和panel的底部重合靠紧*/
}
.panel > .list-group .list-group-item {
  border-width: 1px 0;  /*将列表组里的item项的上下边框设置为1像素*/
  border-radius: 0;     /*将列表组里的item项的圆角取消*/
}
.panel > .list-group .list-group-item:first-child {
  border-top: 0;                          /*设置所有list group内的第一个item
的顶部边框为0
                                        像素，以便和panel的顶部重合靠紧*/
}
.panel > .list-group .list-group-item:last-child {
  border-bottom: 0;              /*设置所有list group内的最后一个item的底部边
框为0
                                        像素，以便和panel的底部重合靠紧*/
}
.panel > .list-group:first-child .list-group-item:first-child {
   border-top-left-radius: 3px;             /*设置第一个list group内的第一个
item的顶部左右圆角*/
  border-top-right-radius: 3px;
}
.panel > .list-group:last-child .list-group-item:last-child {
  border-bottom-right-radius: 3px;
                                            /*设置最后一个list group内的最后一
个item的底部左右圆角*/
  border-bottom-left-radius: 3px;
}
.panel-heading + .list-group .list-group-item:first-child {
  border-top-width: 0;           /*如果列表组和head并列紧靠，则将第一个item的
上边框
                                        的宽度设置为0，以防产生重合的粗线*/
}
```

注意

面板容器在应用了多彩样式之后，内部的列表组的边框的颜色依然是灰色，这是因为一般情况下不会使用这种组合做百叶窗效果，而是拿面板当一个容器使用，所以一般不会对列表项里的内容设置得和面板多彩样式一模一样，而是将面板内部增加内边距（padding）值，然后在里面显示多块内容。

4.20　洼地

源码文件：wells.less

CSS文件：bootstrap.css 5046行之后

洼地（Well）样式的效果和大屏幕展播Jumbotron样式类似，不同点是well样式有了边框设置，并且默认高度是自适应文本的高度。源码如下：

```
// 源码5046行
.well {
  min-height: 20px;              /*最小高度设置为20像素*/
  padding: 19px;                          /*内边距设置*/
  margin-bottom: 20px;           /*底部外边距为20像素*/
  background-color: #f5f5f5;     /*背景色为灰色*/
  border: 1px solid #e3e3e3;     /*设置边框*/
  border-radius: 4px;            /*设置圆角*/
  -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, .05);  /*阴影*/
          box-shadow: inset 0 1px 1px rgba(0, 0, 0, .05);
}
.well blockquote {
  border-color: #ddd;            /*如果有引用的话，加深引用的边框颜色*/
  border-color: rgba(0, 0, 0, .15);
}
```

well样式的使用方法也非常简单。示例代码如下：

```
<div class="well">
    <p>Some default panel. </p>
    <blockquote>注意，这里是引用内容！</blockquote>
</div>
```

上述示例的运行效果如图4-97所示。

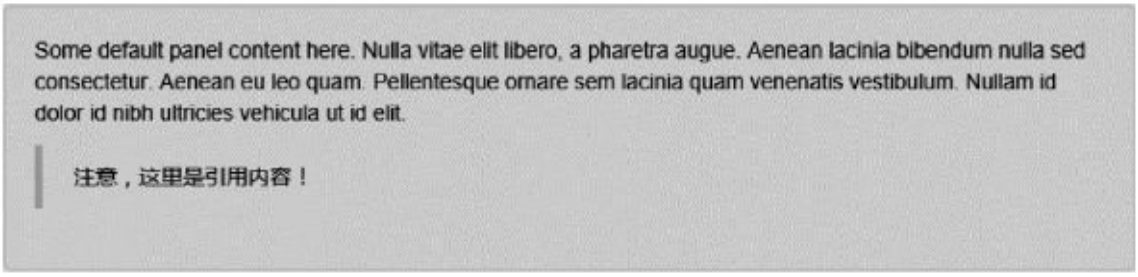


图4-97　well运行效果

另外，well样式也提供了不同大小的样式（主要是panding和圆角大小），分别是：well-lg和well-sm。使用时，直接和well一起应用在同一个元素上即可。

```
// 源码5060行
.well-lg {
  padding: 24px;                          /*加大内边距*/
```

```
  border-radius: 6px;           /*加大圆角设置*/
}
.well-sm {
  padding: 9px;                 /*减少内边距*/
  border-radius: 3px;           /*减少圆角设置*/
}
```

类似的，读者也可以改造well，让well支持不同颜色的显示（比如边框和背景色的变化），样式如well-success、well-info一样，请读者自行练习。

4.21 主题

在Bootstrap新版——3.x版本中还新增了一个bootstrap-theme.css文件，该文件并没有提供什么新的功能，仅是针对一些常用的CSS组件进行了增强。其原因是，在2.x版里的默认主题得不到用户的细化，所以在3.x版本里对所有的CSS组件都默认给出了一个非常简单的样式，然后通过theme文件，提供了一个增强的样式风格。也就说，如果要按照theme声明的方式改造自己的风格，完全可以在新版bootstrap.css后面附加一个和theme.css类似的文件来实现。但是theme默认并没有对所有的CSS组件都进行增强，而是针对9个方面的组件进行了增强：btn按钮、缩略图、下拉菜单、导航条、警告框、进度条、列表组、面板和well。

4.21.1 btn按钮样式主题

在theme里，与btn相关的样式主要对按钮阴影和背景颜色进行了增强（或改变）。默认状态下的按钮的阴影调整代码如下：

```
// 源码7行
.btn-default,.btn-primary,
.btn-success,.btn-info,
.btn-warning,.btn-danger {
  text-shadow: 0 -1px 0 rgba(0, 0, 0, 0.2);
                                                    -webkit-box-
shadow: inset 0 1px 0 rgba(255, 255, 255, 0.15), 0 1px 1px rgba
              (0, 0, 0, 0.075);
                                                          box-
shadow: inset 0 1px 0 rgba(255, 255, 255, 0.15), 0 1px 1px rgba(0,
              0, 0, 0.075);
}
```

高亮状态下的按钮阴影调整代码如下：

```
// 源码17行
.btn-default:active,.btn-primary:active,
.btn-success:active,.btn-info:active,
.btn-warning:active,.btn-danger:active,
.btn-default.active,.btn-primary.active,
.btn-success.active,.btn-info.active,
.btn-warning.active,.btn-danger.active {
  -webkit-box-shadow: inset 0 3px 5px rgba(0, 0, 0, 0.125);
          box-shadow: inset 0 3px 5px rgba(0, 0, 0, 0.125);
}
.btn:active,.btn.active {  background-image: none;}
```

另外，在新版本中针对按钮的每一种风格，对其相关的边框、背景图，以及不同状态下的背景色和边框颜色都进行了增强和改变。以.btn-default样式为例，其源码如下：

```
// 源码36行
.btn-default {
  text-shadow: 0 1px 0 #fff;
                            background-image:             -webkit-linear-
gradient(top, #fff 0%, #e0e0e0 100%);
        background-image:                                   linear-
gradient(to bottom, #fff 0%, #e0e0e0 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#fff
      Colorstr='#ffe0e0e0', GradientType=0);
  filter: progid:DXImageTransform.Microsoft.gradient(enabled = false);
  background-repeat: repeat-x;
  border-color: #dbdbdb;
  border-color: #ccc;
}
.btn-default:hover,.btn-default:focus       {              background-
color: #e0e0e0;  background-
    position: 0 -15px;}
```

```
.btn-default:active,.btn-default.active      {             background-
color: #e0e0e0;  border-color:
    #dbdbdb;}
```

4.21.2 缩略图样式主题

针对缩略图样式，3.x版本中也只是加强了一下阴影的设置。具体如下：

```
// 源码146行
.thumbnail,.img-thumbnail {
  -webkit-box-shadow: 0 1px 2px rgba(0, 0, 0, 0.075);
          box-shadow: 0 1px 2px rgba(0, 0, 0, 0.075);
}
```

4.21.3　下拉菜单样式主题

在下拉菜单里，针对li里的a链接进行了两种类似的增强，分别如下。

❑默认的a链接。在hover和focus状态下的背景色和背景图标设置如下：

```
// 源码151行
.dropdown-menu > li > a:hover,
.dropdown-menu > li > a:focus {
  background-color: #e8e8e8;
                          background-image:              -webkit-linear-
gradient(top, #f5f5f5 0%, #e8e8e8 100%);
         background-image:                                      linear-
gradient(to bottom, #f5f5f5 0%, #e8e8e8 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#fff
      endColorstr='#ffe8e8e8', GradientType=0);
  background-repeat: repeat-x;
}
```

❑高亮的a链接。在hover和focus状态下的背景色和背景图标设置如下：

```
// 源码159行
.dropdown-menu > .active > a,
.dropdown-menu > .active > a:hover,
.dropdown-menu > .active > a:focus {
  background-color: #357ebd;
                          background-image:              -webkit-linear-
gradient(top, #428bca 0%, #357ebd 100%);
         background-image:                                      linear-
gradient(to bottom, #428bca 0%, #357ebd 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#ff4
      endColorstr='#ff357ebd', GradientType=0);
  background-repeat: repeat-x;
}
```

4.21.4 导航条样式主题

由于导航条有两种——navbar-default和navbar-inverse，所以在增强的时候，分别对这两种样式进行修改。以navbar-default为例，增强修改如下。

针对导航条的背景、圆角、阴影的修改如下：

```
// 源码168行
.navbar-default {
                            background-image:             -webkit-linear-
gradient(top, #fff 0%, #f8f8f8 100%);
         background-image:                                        linear-
gradient(to bottom, #fff 0%, #f8f8f8 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#fff
      endColorstr='#fff8f8f8', GradientType=0);
  filter: progid:DXImageTransform.Microsoft.gradient(enabled = false);
  background-repeat: repeat-x;
  border-radius: 4px;
                                                                -webkit-box-
shadow: inset 0 1px 0 rgba(255, 255, 255, .15), 0 1px 5px rgba
      (0, 0, 0, .075);
                                                                      box-
shadow: inset 0 1px 0 rgba(255, 255, 255, .15), 0 1px 5px rgba
             (0, 0, 0, .075);
}
```

针对导航条内的高亮元素链接的背景、阴影的修改如下：

```
// 源码178行
.navbar-default .navbar-nav > .active > a {
                            background-image:             -webkit-linear-
gradient(top, #ebebeb 0%, #f3f3f3 100%);
         background-image:                                        linear-
gradient(to bottom, #ebebeb 0%, #f3f3f3 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffe
      endColorstr='#fff3f3f3', GradientType=0);
  background-repeat: repeat-x;
  -webkit-box-shadow: inset 0 3px 9px rgba(0, 0, 0, .075);
          box-shadow: inset 0 3px 9px rgba(0, 0, 0, .075);
}
```

针对Brand和其他相关的阴影的修改如下：

```
// 源码186行
.navbar-brand,.navbar-nav > li > a {
  text-shadow: 0 1px 0 rgba(255, 255, 255, 0.25);
}
```

另外，针对固定状态的几个特殊样式的圆角，也进行了消除设置。源码如下：

```
// 源码209行
```

```
.navbar-static-top,
.navbar-fixed-top,
.navbar-fixed-bottom {
  border-radius: 0;
}
```

4.21.5 警告框样式主题

针对警告框样式，先是统一调整了阴影设置，源码如下：

```
// 源码214行
.alert {
  text-shadow: 0 1px 0 rgba(255, 255, 255, 0.2);
                                                    -webkit-box-
shadow: inset 0 1px 0 rgba(255, 255, 255, 0.25), 0 1px 2px rgba
      (0, 0, 0, 0.05);
                                                          box-
shadow: inset 0 1px 0 rgba(255, 255, 255, 0.25), 0 1px 2px rgba
      (0, 0, 0, 0.05);
}
```

然后又对所有风格的边框颜色和背景设置进行了调整，以success为例，源码如下：

```
// 源码219行
.alert-success {
                            background-image:          -webkit-linear-
gradient(top, #dff0d8 0%, #c8e5bc 100%);
                                 background-image:              linear-
gradient(to bottom, #dff0d8 0%, #c8e5bc 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffc
      endColorstr='#ffc8e5bc', GradientType=0);
  background-repeat: repeat-x;
  border-color: #b2dba1;
}
```

4.21.6　进度条样式主题

对进度条的调整都与背景相关，形式和警告框alert类似，这里不赘述，请读者自行阅读247～282行源码。

4.21.7　列表组样式主题

对于列表组，首先是调整了列表组容器的圆角和阴影设置，源码如下：

```
// 源码283行
.list-group {
  border-radius: 4px;
  -webkit-box-shadow: 0 1px 2px rgba(0, 0, 0, 0.075);
          box-shadow: 0 1px 2px rgba(0, 0, 0, 0.075);
}
```

其次是对当前高亮的列表项的阴影、背景以及边框颜色进行了调整。

```
// 源码288行
.list-group-item.active,
.list-group-item.active:hover,
.list-group-item.active:focus {
  text-shadow: 0 -1px 0 #3071a9;
                          background-image:              -webkit-linear-
gradient(top, #428bca 0%, #3278b3 100%);
         background-image:                                        linear-
gradient(to bottom, #428bca 0%, #3278b3 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#ff4
      endColorstr='#ff3278b3', GradientType=0);
  background-repeat: repeat-x;
  border-color: #3278b3;
}
```

4.21.8 面板样式主题

面板样式的调整有两个：一个是容器的阴影，一个是每种风格下的head元素的背景调整。示例源码如下：

```
// 源码298行
.panel {
  -webkit-box-shadow: 0 1px 2px rgba(0, 0, 0, .05);
          box-shadow: 0 1px 2px rgba(0, 0, 0, .05);
}
.panel-default > .panel-heading {
                            background-image:             -webkit-linear-
gradient(top, #f5f5f5 0%, #e8e8e8 100%);
                                  background-image:                linear-
gradient(to bottom, #f5f5f5 0%, #e8e8e8 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#fff
      endColorstr='#ffe8e8e8', GradientType=0);
  background-repeat: repeat-x;
}
```

4.21.9 well样式主题

well样式的调整也是背景、边框颜色这几样。源码如下：

```
// 源码338行
.well {
                              background-image:              -webkit-linear-
gradient(top, #e8e8e8 0%, #f5f5f5 100%);
                                    background-image:                 linear-
gradient(to bottom, #e8e8e8 0%, #f5f5f5 100%);
  filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#ffe
      endColorstr='#fff5f5f5', GradientType=0);
  background-repeat: repeat-x;
  border-color: #dcdcdc;
                                                                  -webkit-box-
shadow: inset 0 1px 3px rgba(0, 0, 0, .05), 0 1px 0 rgba(255, 255,
      255, .1);
                                                                        box-
shadow: inset 0 1px 3px rgba(0, 0, 0, .05), 0 1px 0 rgba(255,
              255, 255, .1);
}
```

注意，theme文件不是必须要引用的，如果喜欢这种风格才引用它；如果要定制自己的风格，则可以引用自己的theme名称，比如bootstrao-theme-flatui.css。但是一定要注意，该文件一定要在bootstarp.min.css文件之后才能引用，否则会覆盖默认的效果。

第5章　JavaScript插件

新版Bootstrap提供了12种JavaScript插件，分别是：动画过渡（Transition）、模态弹窗（Modal）、下拉菜单（Dropdown）、滚动侦测（Scrollspy）、选项卡（Tab）、提示框（Tooltip）、弹出框（Popover）、警告框（Alert）、按钮（Button）、折叠（Collapse）、旋转轮播（Carousel）、自动定位浮标（Affix）。

除了第一种动画插件以外，其他11种插件都是我们日常开发中需要用到的功能，除了细述所有的用法以外，本章将深入对这些插件进行源码分析。

5.1 动画过渡效果

源文件：transition.js

Bootstrap默认给各组件提供了基本的动画过渡效果，在使用过程中，如果使用了统一编译的Bootstrap.js，则动画过渡效果默认就有了。如果单个组件分开使用，则需要先引用transition.js，再引用插件的JS文件。

值得注意的是，Bootstrap使用的动画过渡效果全部使用了CSS3的语法，所以IE6、IE7、IE8是不能用动画过渡效果了。

5.1.1 使用方法

默认情况下，以下组件使用了动画过渡效果：

❑模态弹窗（Modal）的滑动和渐变效果；

❑选项卡（Tab）的渐变效果；

❑警告框（Alert）的渐变效果；

❑旋转轮播（Carousel）的滑动效果。

5.1.2 源码分析

动画过渡效果不是标准的插件，而是一个判断动画的工具方法，所以没有遵守JavaScript插件的标准开发步骤。Bootstrap的transition.js文件用于将浏览器是否支持动画的标记赋值给$.support.transition属性，其他插件在使用的时候可以直接从$.support.transition属性获取相应的值。transition.js源码如下所示：

```
+function ($) {
    ‘use strict’;                                                    // 严格
模式
    function transitionEnd() {
        var el = document.createElement(‘bootstrap’)    // 创建一个自定义
标签做测试

        var transEndEventNames = {       // 用于检测CSS3 transition结束时
的回调事件
            ‘WebkitTransition’: ‘webkitTransitionEnd’,// Webkit浏览器事件
            ‘MozTransition’: ‘transitionend’,            // Mozlia浏览器
事件
            ‘OTransition’: ‘oTransitionEnd otransitionend’,     // Opera
浏览器事件
            ‘transition’: ‘transitionend’                // 其他浏览器事件
        }
        for (var name in transEndEventNames) {
            if (el.style[name] !== undefined) {
                return { end: transEndEventNames[name] }
                       //  如果支持动画事件，则返回对象字面量，如果没有，默认返回
undefind
            }
        }
        return false                                          // 解决 IE8下的
bug
}

    $.fn.emulateTransitionEnd = function (duration) {   // 触发动画事件
        var called = false, $el = this
        $(this).one($.support.transition.end, function () { called = tru
        var callback = function () { if (!called) $($el).trigger($.suppo
        setTimeout(callback, duration)                  // 根据传入的延迟
毫秒数，再触发
        return this
}

    $(function () {
        $.support.transition = transitionEnd()  // 全局赋值
    })
}(jQuery);
```

通过源码可以看出，$.support.transition属性的值是{end:事件名称}，插件在确定支持动画的情况下，给该事件绑定其他额外的处理代码，以便在触发时进行调用。示例如下：

```
$.support.transition && this.$element.hasClass('fade')?
          this.$backdrop.one($.support.transition.end, callback) : callb
```

该代码的意思是如果浏览器支持动画，并且元素上也设置了启用动画（fade），则在动画结束的时候才继续执行callback代码（即在$.support.transition.end上绑定callback），否则直接执行callback代码。

有很多效果都用到了动画，动画特效本身的代码都在CSS里设置，一般都是通过元素内部定义fade样式，fade的源码如下：

```
// 源码2346行
.fade {
  opacity: 0;                                /*100%透明*/
  -webkit-transition: opacity .15s linear;      /*透明度，匀速变化0.15秒*/
          transition: opacity .15s linear;
}
.fade.in {                                     /*应用in样式，表示全部打开
*/
  opacity: 1;                                /*完全不透明*/
}
```

fade样式在弹窗弹出以后的源码设置如下：

```
// 源码5109行
.modal.fade .modal-dialog {
  -webkit-transition: -webkit-transform .3s ease-out;  /*减速运行*/
     -moz-transition:    -moz-transform .3s ease-out;
       -o-transition:      -o-transform .3s ease-out;
          transition:         transform .3s ease-out;
  -webkit-transform: translate(0, -25%);         /*向下移动25%的距离，从而产
生向下的动画视觉*/
       -ms-transform: translate(0, -25%);
          transform: translate(0, -25%);
}
.modal.in .modal-dialog {                      /*页面完整显示时，回归到原始位置*/
  -webkit-transform: translate(0, 0);
       -ms-transform: translate(0, 0);
          transform: translate(0, 0);
}
```

5.2 模态弹窗

源文件：modal.js、modals.less

CSS文件：bootstrap.css 5096行之后

模态弹窗（也叫popup，Bootstrap里称为Modal）是绝大部分交互式网站都需要的一种功能，一般有以下几种用法：

❑提示信息、警告信息、大文本等；

❑确认提示（多按钮）；

❑显示表单元素（一般使用ajax操作等功能）；

❑其他需要特殊显示的信息（如单击缩略图时放大图片）。

而伴随着这些功能要求，也会产生其他的功能需求，比如按钮触发事件、表单验证、关闭事件等。Bootstrap中的Modal插件为此提供了强大的支持，本节我们将详细讲述Modal插件的使用方法和源码分析。

5.2.1 弹窗布局与样式

在讲解用法之前先看一下一般弹窗的运行效果，如图5-1所示。其中包括了3个部分：头部（包括标题和关闭符号），中间部分（主要是内容），底部（主要是放置操作按钮）。

图5-1 普通的弹窗效果

根据上述效果，我们先来了解一下弹窗HTML代码的布局定义和相关的样式。和之前的老版本不同，新版的弹窗组件使用了3层div容器元素，其分别应用了modal、modal-dialog、modal-content样式。然后在真正的内容容器modal-content内包括了弹窗的头（header）、内容（body）、尾（footer）3个部分，分别应用了3个样式：modal-header、modal-body、modal-footer。示例代码如下：

```
<div class="modal show">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                            <button type="button" class="close" data-
dismiss="modal"
aria-hidden="true">×</button>
                <h4 class="modal-title">Modal标题</h4>
            </div>
            <div class="modal-body">
                <p>这里是弹窗里的具体内容……</p>
            </div>
            <div class="modal-footer">
                    <button type="button" class="btn btn-default" data-
dismiss="modal">
                    关闭</button>
                     <button type="button" class="btn btn-primary">保存
</button>
            </div>
        </div>
    </div>
```

```
</div>
```

modal样式和之前的功能有所不同，在这里只是做一个背景容器，100%充满全屏（其底部还有一个100%充满全屏的modal-backdrop样式，稍后再说）。另外还有一个功能是，当弹窗中内容很多的时候，可以在modal容器里进行滚动操作，避免了弹窗不能移动的弊端。源码如下：

```
// 源码5096行
.modal {
  position: fixed;                                        /* 固定布局 */
  top: 0;                                                  /* 上下左右都是0，表示充满全屏 */
  right: 0;
  bottom: 0;
  left: 0;
    z-index:  1050;/*  提升z-index，防止其他元素溢出，modal-dialog的z-index是1050 */
  display: none;                                          /* 默认不显示 */
  overflow: auto;
  overflow-y: scroll;                                    /* 竖向可以滚动 */
   -webkit-overflow-scrolling: touch;      /*  支持移动设备上使用触摸方式进行滚动 */
  outline: 0;                                             /* 消除虚边框 */
}
```

另外在modal样式的基础上，Bootstrap还提供一个动画的功能，即在弹窗完全显示之前执行一段动画，其动画内容是：从-25%的top值位置到top:0的位置。相关代码如下：

```
// 源码5109行
.modal.fade .modal-dialog {
  -webkit-transition: -webkit-transform .3s ease-out;  /*减速运行*/
     -moz-transition:    -moz-transform .3s ease-out;
       -o-transition:      -o-transform .3s ease-out;
          transition:         transform .3s ease-out;
  -webkit-transform: translate(0, -25%);         /*向下移动25%的距离，从而产生向下的动画视觉*/
       -ms-transform: translate(0, -25%);
          transform: translate(0, -25%);
}
.modal.in .modal-dialog {                                 /*页面完整显示时，回归到原始位置*/
  -webkit-transform: translate(0, 0);
       -ms-transform: translate(0, 0);
          transform: translate(0, 0);
}
```

在modal完全充满全屏的情况下，默认在其内部（也就是内部容器）放置了一个宽度自适应、左右水平居中的modal-dialog样式的div容器。modal-dialog样式有两段代码，其源码如下：

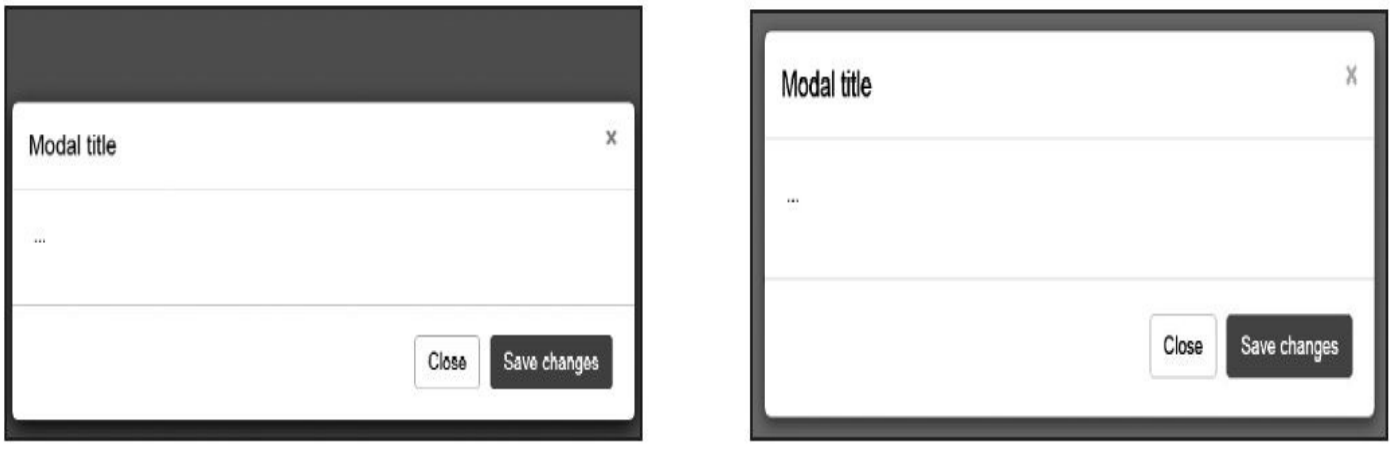
```
// 源码5123行
.modal-dialog {                                          /*默认设置（包括窄屏）*/
```

```
  position: relative;                    /* 相对于modal元素，进行相对定位 */
  width: auto;                           /*宽度自适应*/
  margin: 10px;                          /*10像素外边距*/
}
```

分辨率大于768像素时的样式设置如下：

```
// 源码5188行
@media (min-width: 768px) {                    /* 可视范围大于768像素的普
通浏览器 */
  .modal-dialog {
    width: 600px;                              /* 普通宽度为600像素 */
    margin: 30px auto;            /* 上下各30像素外边距，这里才是设置弹窗看起来
居中的地方，
                                                这不是真正居中，而是30像素的高
度 */
  }
  .modal-content {
    -webkit-box-shadow: 0 5px 15px rgba(0, 0, 0, .5);
            box-shadow: 0 5px 15px rgba(0, 0, 0, .5);
 }
  .modal-sm {    width: 300px;  /* 小型弹窗设置其宽度为300像素 */  }
  .modal-lg {    width: 900px;  /* 大型弹窗设置其宽度为900像素*/  }
}
```

通过上述两段代码可以看出来，在正常的桌面浏览器（可视区域大于768像素）上，弹窗并不是上下居中，而只是设置了30像素的padding-top值而已，因为从视觉上看和上下居中区别不大；而在移动设备（小于768像素）的浏览器上，则只有10像素的外边距（看起来几乎是满屏显示了），效果如图5-2所示。



a）30 像素　　b）10 像素

图5-2　不同尺寸浏览器下的运行效果

另外，modal-sm和modal-lg样式是两个新样式，在新版V3.1.0中才推出，用于在768像素以上的浏览器上，对弹窗进行强制缩小（宽度为300像素）和加大设置（宽度为900像素）。

对于第三层嵌套div上的modal-content样式，才是对弹窗进行设置的主要代码，主要是边框、边距、背景色、阴影的处理。源码如下：

```
// 源码5128行
.modal-content {                                /*默认设置（包括窄屏）*/
  position: relative;           /* 相对定位 */
  background-color: #fff;       /* 白色背景 */
  background-clip: padding-box; /* 背景的裁剪区域设置，从padding区域向外 */
  border: 1px solid #999;       /* 边框设置 */
  border: 1px solid rgba(0, 0, 0, .2); /* 透明度为0.2 */
  border-radius: 6px;           /* 圆角设置 */
  outline: none;                            /* 取消轮廓显示 */
  -webkit-box-shadow: 0 3px 9px rgba(0, 0, 0, .5); /* 阴影设置 */
          box-shadow: 0 3px 9px rgba(0, 0, 0, .5);
}
```

同样，作者对于不同宽度的浏览器也进行了自适应设计，对于桌面浏览器（或可视范围大于768像素的浏览器），加大了阴影设置。

```
// 源码5193行
@media (min-width: 768px) {     /* 可视范围大于768像素的普通浏览器 */
  .modal-content {
    -webkit-box-shadow: 0 5px 15px rgba(0, 0, 0, .5);
            box-shadow: 0 5px 15px rgba(0, 0, 0, .5);
 }
}
```

内容区域上中下3个部分的样式，主要是用于控制3个地方的内外边距和对齐方式等。主要代码如下：

```
// 源码5156行
.modal-header {                             /* 弹窗头部位置设置 */
  min-height: 16.428571429px;   /* 最小高度设置 */
  padding: 15px;                            /* 内边距 */
   border-bottom: 1px solid #e5e5e5;       /* 底部设置细线，和modal-body区
分 */
}
.modal-header .close {
  margin-top: -2px;             /* 关闭按钮 */
}
.modal-title {                  /* 头部位置内的标题样式 */
  margin: 0;
  line-height: 1.428571429;
}
.modal-body {                   /* 中间内容区域 */
  position: relative;           /* 相对位置 */
  padding: 20px;                            /* 内边距 */
}
.modal-footer {                             /* 底部区域设置 */
  padding: 19px 20px 20px;      /* 内边距 */
  margin-top: 15px;             /* 顶部外边距 */
  text-align: right;            /* 居右对齐，因为一般都是按钮 */
   border-top: 1px solid #e5e5e5;          /* 上边框设置，以便和modal-body分
隔 */
```

```
}
```

另外，由于底部区域一般都是放置各种按钮，所以在底部样式内，也对各种按钮进行了特殊设置。源码如下：

```
// 源码5178行
.modal-footer .btn + .btn {      /* 底部区域内的按钮样式设置，如果有多个按钮，
设置左部外边距 */
  margin-bottom: 0;
  margin-left: 5px;
}
.modal-footer .btn-group .btn + .btn {
  margin-left: -1px;/* 底部区域内的按钮组样式设置，如果按钮都在一个组内，则减小左
部外边距值 */
}
.modal-footer .btn-block + .btn-block { /* 底部区域内的按钮块样式设置 */
  margin-left: 0;       /* 取消左部外边距*/
}
```

最后，关于弹窗，还有一个隐性的样式（modal-backdrop），也就是大背景，一般设置为灰色背景。除了弹窗内容以外，对于背景下面的元素的所有单击事件一律阻止。该样式是通过JavaScript代码在弹窗弹出之前附加到body元素内的，其z-index(1040)在modal(1050)之下。相关源码如下：

```
// 源码5139行
.modal-backdrop {
  position: fixed;              /* 固定定位 */
  top: 0;                       /* 上下左右充满全屏 */
  right: 0;
  bottom: 0;
  left: 0;
  z-index: 1040;                 /* 小于modal的1050，阻止鼠标单击背景下的其他
元素 */
  background-color: #000;       /* 默认黑色背景 */
}
.modal-backdrop.fade {  filter: alpha(opacity=0);  opacity: 0;
                                /* 动画过渡之前，保持原样，即透明度为0 */}
.modal-backdrop.in {  filter: alpha(opacity=50);  opacity: .5;
                                /* 动画过渡之后，保持50%的透明度 */}
```

通过源码，我们可以想象出整个模态弹窗的盒子模型，如图5-3所示。如果大家还不理解，请再次阅读上面的源码分析。

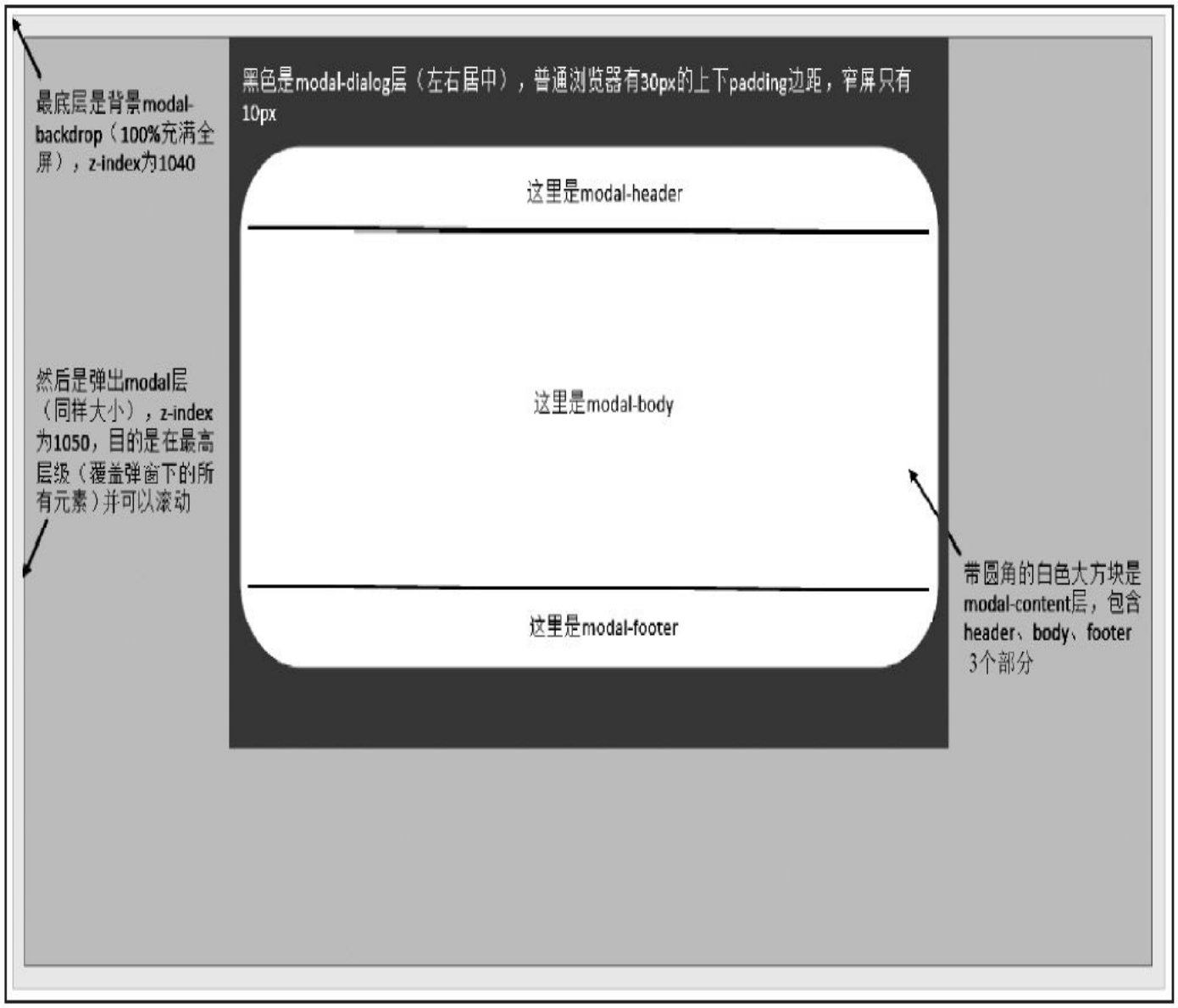


图5-3　模态弹窗的盒子模型

5.2.2 声明式用法

在了解了弹窗的基本HTML代码和样式原理之后，我们开始学习弹窗的声明式用法。在2.4节里讲到，所有的JavaScript组件都提供了声明式的语法，要想触发某一插件就需要对特定元素设置触发属性，一般都是自定义的，比如data-toggle=或者data-dismiss=。弹窗所用的触发属性是第一种。

一个基本的弹窗声明，只需要两个必需的自定义属性：data-toggle=和data-target=。示例如下：

```
<!— 触发元素 —>
<button  data-toggle=“modal”  data-target=“popupCSS”  class=“btn  btn-primary”>
    单击弹出</button>
<!— 弹窗内容 —>
<div class=“modal fade popupCSS”>
    <!— 嵌套div和具体内容 —>
</div>
```

对于data-toggle，必须将其值设置为modal；data-target的值则可以设置为CSS选择符，可以是要弹出弹窗（modal）的id值，也可以是唯一的CSS样式。

如果触发元素为a元素，则也可以使用href的值作为data-target，这都是对modal弹窗进行定位用的，但一般建议都使用data-target，以便统一。

```
<!— 触发元素 —>
<a  data-toggle=“modal”  href=”#myModal”  class=“btn  btn-primary  btn-large”>Launch
    demo modal</a>
<!— 弹窗内容 —>
<div class=“modal fade” id=“myModal”>
    <!— 嵌套div和具体内容 —>
</div>
```

另外，在进行声明式定义的时候，该组件还支持默认的参数选项，比如是否有灰色背景遮罩、按Esc键时是否关闭弹窗等。表5-1所示是可以用的属性和属性解释。

表 5-1　Modal 插件的声明式选项

属性名称	类　型	默 认 值	描　述
data-backdrop	布尔值	true	是否包含一个背景 div 元素，如果为 true，则单击该背景（不是弹窗本身）时关闭弹窗；如果设置 data-backrop="static"，则单击背景 div 元素时不关闭弹窗
data-keyboard	布尔值	true	按 Esc 键时关闭弹窗；如果设置为 false，则不关闭弹窗
data-show	布尔值	true	初始化时是否显示
href	URL 路径	空值	如果 href 的值不是以 # 开头，则表示是一个 url 地址，弹窗会先加载其内容，然后替换原有的 modal-content 样式的 div 元素。示例如下： <a data-toggle="modal" href="remote.html" data-target="#modal">触发按钮 （设置 href 为地址时，data-target 则必须填写制定 id）

上述属性，除了href外，其他3个data-开头的属性，一般情况下可以设置在触发元素（带有[data-toggle=“modal”]属性的元素）上，也可以设置在弹窗最外层的div元素（带有modal，并且是由data-target指定的div元素）上。但特殊情况除外，就是在设置了href为url远程获取内容时，如果该弹窗实例已经运行过一次，则不会加载远程url的内容（依然保持modal-body内容不变），而且在触发元素上设置的可选参数也无效，但弹窗div元素容器上设置的参数依然有效。所以为了防止出现类型问题，开发的时候可以统一在触发元素容器上设置表5-1中3个以data-开头的参数。示例代码如下：

```
<!— 触发元素 —>
<a                    data-toggle=“modal”                    data-
target=”#myModal” href=“remote.html” class=“btn
    btn-primary”>触发元素</a>
<!— 弹窗内容 —>
<div class=“modal fade” id=“myModal” data-modal=” ”>
    <!— 嵌套div和具体内容 —>
</div>
```

注意

为了能够支持辅助性工具，可以为.modal添加role=“dialog” aria-labelledby=“myModalLabel”属性；aria-hidden=“true”告诉辅助性工具略过模态框的DOM元素。另外，还应该为模态框添加描述性信息，即为.modal添加aria-describedby属性。

5.2.3 JavaScript用法

模态弹窗组件在JavaScript方面的一般用法是，传入可以拥有一部分可选参数的JavaScript对象字面量，用于初始化弹窗的一些自定义属性，以便控制个性化的弹窗效果。示例如下：

```
$('#myModal').modal({
    backdrop: true,
    keyboard: false
})
```

1. 属性

该组件默认支持的自定义属性如表5-2所示。其中和表5-1所示的声明式的自定义属性类似（以data-开头的文字去掉了，并且声明式里的href属性换成了remote属性）。

表 5-2 Modal 插件的 JavaScript 用法选项

属性名称	类 型	默 认 值	描 述
backdrop	布尔值	true	是否包含一个背景 div 元素，如果为 true，则单击该背景（不是弹窗本身）时关闭弹窗；如果设置为 backrop: "static"，则单击背景 div 元素时不关闭弹窗
keyboard	布尔值	true	按 Esc 键时候关闭弹窗；如果设置为 false，则不关闭弹窗

（续）

属性名称	类 型	默 认 值	描 述
show	布尔值	true	初始化时是否显示
remote	URL 路径	空值	远程获取 remote 指定 url 的内容来填充弹窗（modal-content 样式内）

2. 参数

根据前面章节提到的，所有的JavaScript组件都支持传入特定字符串执行其内部方法的用法，字符串名称和用法描述如表5-3所示。

表 5-3　Modal 插件的 JavaScript 用法参数

参数名称	使用方式	描　　述
toggle	$('#myModal').modal('toggle')	触发时，反转弹窗的状态，即如果当时是开启，则关闭，反之则开启
show	$('#myModal').modal('show')	触发时，显示弹窗
hide	$('#myModal').modal('hide')	触发时，关闭弹窗

一般来说，参数字符串show用得比较多，因为都是要打开弹窗。特殊情况下，如果在弹窗上添加某些元素的触发事件，以便用户关闭弹窗，也可以使用hide字符串参数。toggle用得很少，因为弹窗一旦弹出就利用背景div元素阻止了其他非弹窗元素的单击事件，所以一般用不上。

需要注意的是，即便是使用了JavaScript的用法实现弹窗功能，也必须按照嵌套格式设置好弹窗所需要的div结构和样式，否则会出现意想不到的效果。

3. 事件

模态弹窗支持4种类型的事件订阅，分别对应模态弹窗的弹出前、弹出后、关闭前、关闭后，解释和用法如表5-4所示。

表 5-4　Modal 插件的事件类型

事件类型	描　　述
show.bs.modal	在 show 方法调用时立即触发（尚未显示之前）；如果单击了一个元素，那么该元素将作为事件的 relatedTarget 属性
shown.bs.modal	该事件在模态弹窗完全显示给用户之后（并且等 CSS 动画完成之后）触发；如果单击了一个元素，那么该元素将作为事件的 relatedTarget 属性
hide.bs.modal	在 hide 方法调用时（但还未关闭隐藏）立即触发
hidden.bs.modal	该事件在模态弹窗完全隐藏之后（并且等 CSS 动画完成之后）触发

调用方式也很简单，和普通的jQuery代码并无二样。

```
$('#myModal').on('hidden.bs.modal', function (e) {
    // 处理代码…
})
```

注意

弹窗的标准用法要求开发人员必须事先把所需弹窗的div结构放在页面里，并且保持隐藏状态，以便调用时弹出，很多时候这样做不是很方

便。在实战章节，我们会提供其他的方式，可以自己临时组装弹窗div代码，然后再弹出。

5.2.4 源码分析

在源码分析之前，确保我们已经记住了弹窗Modal的基本元素，这样在阅读JavaScript代码时才能对照元素进行理解。示例如下：

```
<!— 触发元素 —>
<button  data-toggle=“modal”  data-target=”#myModal”  class=“btn  btn-
primary”>
    触发元素</button>
<!— 弹窗内容 —>
<div class=“modal fade” id=“myModal”>
    <div class=“modal-dialog”>
        <div class=“modal-content”>
            <div class=“modal-header”>…</div>
            <div class=“modal-body”>…</div>
            <div class=“modal-footer”>…</div>
        </div>
    </div>
</div>
```

步骤1　立即调用的函数。回顾2.4节，JavaScript插件架构里的步骤，第一步是最通用的步骤，也就是使用立即调用的函数，防止插件内代码外泄，从而形成一个闭环，并且只能从jQuery的fn里进行扩展。

```
    // 1.定义立即调用的函数
+function ($) {
    “use strict”;                          // 使用严格模式 ES5支持
    // 后续步骤
    // 2.Modal插件类及原型方法的定义
    // 3.在jQuery上定义Modal插件，并重设插件构造器
    // 可以通过该属性获取插件的真实类函数
    // 4. 防冲突处理
    // 5. 绑定触发事件
}(window.jQuery);
```

步骤2　插件核心代码。主要是Modal核心类函数的定义、默认参数的定义和8个原型方法的定义，这8个原型方法主要是处理弹窗的反转、打开、关闭和弹窗背景设置、取消等操作。核心代码如下：

```
var Modal = function (element, options) {
                                          // element表示modal弹出框容器及内
部元素，options是设置选项
    this.options = options       // 传进来的各种参数
    this.$element = $(element)
    this.$backdrop =             // modal下面的背景对象
    this.isShown = null                   // 默认情况下，不设置是否已经显示弹窗

       //  如果设置了remote，就加载remote指定url的内容到modal-content样式的元素
内，并触发
         // loaded.bs.modal事件
    if (this.options.remote) {
        this.$element
```

```
            .find('.modal-content')
            .load(this.options.remote, $.proxy(function () {
                        this.$element.trigger('loaded.bs.modal') //  触发
loaded.bs.modal事件
            }, this))
    }
}

Modal.DEFAULTS = {{              // 默认设置
    backdrop: true,              // 默认单击弹窗以外的地方时自动关闭弹窗
    keyboard: true,              // 默认设置，按Esc键关闭弹窗
    show: true                   // 默认设置，单击触发元素时打开弹窗
}
// 反转弹窗状态
Modal.prototype.toggle = function () { };
// 打开弹窗
Modal.prototype.show = function () { };
// 关闭弹窗
Modal.prototype.hide = function (e) { };
// 确保当前打开的弹窗处于焦点状态
Modal.prototype.enforceFocus = function () { };
// 按Esc键是否退出的处理
Modal.prototype.escape = function () { };
// 关闭弹窗
Modal.prototype.hideModal = function () { };
// 删除背景，关闭弹窗时触发
Modal.prototype.removeBackdrop = function () { };
// 添加背景，打开弹窗时触发
Modal.prototype.backdrop = function (callback) { };
```

注意

上面的this.$element.load(this.options.remote)语句表示将内容加载至modal-content样式的元素内，而不是modal-body样式元素内，这和2.x版完全不一样了。

下面的代码是上述8个原型方法的详细注释，阅读的时候主要是要注意元素焦点、动画、回调函数等主要代码的执行原理。

```
// 反转弹窗状态
Modal.prototype.toggle = function (_relatedTarget) {
    return this[!this.isShown ? 'show' : 'hide'](_relatedTarget)
                                        // 如果是关闭状态，则打开弹窗，否则就
关闭
}
// 打开弹窗
Modal.prototype.show = function (_relatedTarget) {
    var that = this                      // 当前modal对象赋值为that，防止作
用域冲突
    var e = $.Event('show.bs.modal', { relatedTarget: _relatedTarget })
                                                // 定义弹窗前的触发事件

    this.$element.trigger(e)            // 打开弹窗前，触发事件

    // 如果已经打开了（或者曾经被阻止过），则退出执行，后续代码不做处理
```

```
    if (this.isShown || e.isDefaultPrevented()) return

    this.isShown = true                              // 设置状态为打开

    this.escape()                              // 处理键盘事件，主要是设置按Esc键的
时候是否关闭弹窗

                  this.$element.on('click.dismiss.bs.modal',     '[data-
dismiss="modal"]',
        (this.hide, this))
    // 如果单击了元素内的子元素（带有[data-dismiss="modal"]属性），则关闭弹窗

    this.backdrop(function () {          // 绘制弹窗背景以后，处理以下代码
        var transition = $.support.transition && that.$element.hasClass(
           //  判断浏览器是否支持动画，并且弹窗是否设置了动画过渡效果（是否有fade样
式）

        if (!that.$element.parent().length) {
            that.$element.appendTo(document.body)
                                                    // 如果modal弹窗没有父容
器，则将它附加到body上
        }

        that.$elementshow().scrollTop(0)         // 显示modal弹窗

        if (transition) {
            that.$element[0].offsetWidth         // 如果支持动画，强制刷新UI
现场，重绘弹窗
        }

        that.$element
             .addClass('in')               // 给modal弹窗添加in样式，和modal
样式一起
             .attr('aria-hidden', false) // 设置aria-hidden为假（告诉阅读器
该元素是非
                                                    // 隐藏状态）

        that.enforceFocus()              // 强制给弹窗设定焦点
        var e = $.Event('shown.bs.modal', { relatedTarget: _relatedTarge

                                                    // 打开弹窗显示后的触发事件
        transition ?
            that.$element.find('.modal-dialog')   // 找到弹窗元素
            .one($.support.transition.end, function () {
                    //  如果支持动画，则动画结束以后给弹窗内的元素设置焦点，并触发
shown事件
                that.$element.focus().trigger(e)
            })
            .emulateTransitionEnd(300) :
            that.$element.focus().trigger(e)    // 否则直接设置焦点，并触发
shown事件
    })
}
// 关闭弹窗
Modal.prototype.hide = function (e) {
    if (e) e.preventDefault()            // 先阻止冒泡行为
```

```
    e = $.Event('hide.bs.modal')        // 关闭弹窗前的触发事件
    this.$element.trigger(e)            // 关闭弹窗前触发事件

    // 如果已经关闭了（或者曾经被阻止过），则退出执行后续代码不做处理
    if (!this.isShown || e.isDefaultPrevented()) return

    this.isShown = false                // 设置状态为关闭
    this.escape()                        // 处理键盘事件，主要是设置按Esc键的时候是否关闭弹窗
        $(document).off('focusin.bs.modal')                  //  取消所有的focusin.bs.modal事件

    this.$element
        .removeClass('in')              // 删除in样式
        .attr('aria-hidden', true)      // 设定aria-hidden为true（告诉阅读器该元素是隐藏状态）
        .off('click.dismiss.bs.modal')  // 取消dismiss的单击事件

    // 如果支持动画，则动画结束以后再关闭，否则直接关闭
    $.support.transition && this.$element.hasClass('fade') ?
        this.$element
        .one($.support.transition.end, $.proxy(this.hideModal, this))
        .emulateTransitionEnd(300) :
        this.hideModal()
}
// 确保当前打开的弹窗处于焦点状态
Modal.prototype.enforceFocus = function () {
    $(document)
         .off('focusin.bs.modal')         // 禁用所有的focusin事件，防止无限循环
        .on('focusin.bs.modal', $.proxy(function (e) {
            if (this.$element[0] !== e.target && !this.$element.has(e.ta
                 // 如果处于焦点的元素不是当前元素（或不包含当前元素），则强制给当前元素设置焦点
                this.$element.focus()
            }
        }, this))
}
// 按Esc键是否退出的处理
Modal.prototype.escape = function () {
    if (this.isShown && this.options.keyboard) {
         // 如果弹窗是打开状态，并且keyboard选项不为false，则说明允许按Esc键关闭弹窗
        this.$element.on('keyup.dismiss.bs.modal', $.proxy(function (e)
            // 检测键盘事件，如果是Esc（keycode=27）键，则关闭
            e.which == 27 && this.hide()
        }, this))
    } else if (!this.isShown) {         // 否则，取消键盘事件检测
        this.$element.off('keyup.dismiss.bs.modal')
    }
}
// 关闭弹窗
Modal.prototype.hideModal = function () {
    var that = this
    this.$element.hide()                // 关闭弹窗
    this.backdrop(function () {
```

```
        that.removeBackdrop()           // 清除背景
        that.$element.trigger('hidden.bs.modal')
                                    // 关闭以后，触发hidden事件（hide事件是在关闭前执行的）
    })
}
// 删除背景，关闭弹窗时触发
Modal.prototype.removeBackdrop = function () {
    this.$backdrop && this.$backdrop.remove()   // 删除背景元素
    this.$backdrop = null               // 设置背景对象为null
}
// 添加背景，打开弹窗时触发
Modal.prototype.backdrop = function (callback) {
    var animate = this.$element.hasClass('fade') ? 'fade' : ''
                                                // 是否设置了动画过渡效果，如果是则设置为fade
    if (this.isShown && this.options.backdrop) {
                                                // 如果是打开状态，并且设置了backdrop参数
        var doAnimate = $.support.transition && animate // 定义动画标识

        // 在body上定义背景div元素，并附加fade标识以支持动画
                            this.$backdrop    =    $('<div    class="modal-backdrop ' + animate + '" />')
            .appendTo(document.body)


          // 背景被单击时进行判断：如果backdrop参数为static，则强制将弹窗设置为焦点；否则，关闭弹窗
        this.$element.on('click.dismiss.bs.modal', $.proxy(function (e)
            if (e.target !== e.currentTarget) return
            this.options.backdrop == 'static'
                ? this.$element[0].focus.call(this.$element[0])
                : this.hide.call(this)
        }, this))

        if (doAnimate) this.$backdrop[0].offsetWidth
                                                // 如果支持动画，则强制刷新UI现场，重绘弹窗
        this.$backdrop.addClass('in')   // 添加in样式
        if (!callback) return           // 如果没有回调，则直接返回

        // 如果支持动画，则动画结束后执行回调函数；否则，直接执行回调函数
        doAnimate ?
            this.$backdrop
            .one($.support.transition.end, callback)
            .emulateTransitionEnd(150) :
            callback()

    } else if (!this.isShown && this.$backdrop) {
                                                // 如果是关闭状态，但背景对象依然还存在
        this.$backdrop.removeClass('in')        // 去除in样式

        // 如果支持动画，则动画结束后执行回调函数；否则，直接执行回调函数
        $.support.transition && this.$element.hasClass('fade') ?
```

```
            this.$backdrop
            .one($.support.transition.end, callback)
            .emulateTransitionEnd(150) :
            callback()

    } else if (callback) {                  // 如果不是以上两种情况，但回调函数却存在
        callback()                          // 直接执行回调函数
    }
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.modal插件，有点特殊的代码是options参数的收集和合并，主要收集了3部分：插件的默认参数Defaults、modal元素上的data-属性、执行插件时候传入的option对象，三部分的优先级依次升高。源码如下：

```
var old = $.fn.modal
//  保留其他库的$.fn.modal代码（如果定义的话），以便在noConflict之后可以继续使用该老代码
$.fn.modal = function (option, _relatedTarget) {
    return this.each(function () {      // 根据选择器，遍历所有符合规则的元素
        var $this = $(this)
        var data = $this.data('bs.modal')       // 获取自定义属性bs.modal的值

        // 将默认参数、选择器所在元素的自定义属性（data-开头）和option参数，这三种值合并在
        一起，作为options参数
        // 优先级：后面的参数优先级高于前面的参数
        var options = $.extend({}, Modal.DEFAULTS, $this.data(), typeof
            'object' && option)

        // 如果值不存在，则将Modal实例设置为bs.modal的值
        if (!data) $this.data('bs.modal', (data = new Modal(this, optior

        // 如果option传递了string，则表示要执行某个方法
            // 比如传入了show，则要执行Modal实例的show方法，data["show"]相当于data.show()
        if (typeof option == 'string') data[option](_relatedTarget)
        else if (options.show) data.show(_relatedTarget)
    })
}
$.fn.modal.Constructor = Modal   // 重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。源码如下：

```
// 防冲突处理
$.fn.modal.noConflict = function () {
    $.fn.modal = old    // 恢复以前的旧代码
     return this                        // 将$.fn.modal.noConflict()设置为Bootstrap的Modal插件
}
```

步骤5　绑定触发事件。源码如下：

```
// 绑定触发事件
$(document).on('click.bs.modal.data-api',                    '[data-
toggle="modal"]', function (e) {
    // 监测所有拥有自定义属性data-toggle="modal"的元素上的单击事件

    var $this = $(this)
    var href = $this.attr('href')              // 获取href属性值

    // 获取data-target属性值，如果没有，则获取href值，该值是所弹出元素的id
var $target = $($this.attr('data-target') || (href && href.replace(/.*
(?=#[^\s]+$)/, '')))

       //  如果弹窗元素上已经有该弹窗实例（即弹出过一次了），则设置option值为字符串
toggle
    // 否则将remote值（如果有的话）、弹窗元素上的自定义属性值集合、触发元素上的自定
义属性值集合，
     合并为option对象
    var option = $target.data('bs.modal') ? 'toggle' : $.extend({ remote
        test(href) && href }, $target.data(), $this.data())

     if ($this.is('a')) e.preventDefault()()      // 如果是a链接元素，则还要
阻止默认行为

    $target
            .modal(option,  this)   //  给弹窗元素绑定Modal插件（也就是实例化
Modal），并传入option参数
        .one('hide', function () {
            $this.is(':visible') && $this.focus()       // 定义一次hide事
件，给所单击元素加上焦点
        })
})

$(document)
      //  给所有的弹窗元素绑定shown事件，一旦该弹窗打开以后，就在body上添加modal-
open样式
    .on('show.bs.modal', '.modal', function () { $(document.body).addCla
open') })
      //  同理，给所有的弹窗元素绑定hidden事件，一旦该弹窗关闭以后，就删除body上的
modal-open样式
    .on('hidden.bs.modal', '.modal', function () { $(document.body).remo
open') })
```

在绑定触发事件上，最后在body元素上添加、删除modal-open样式的操作时需要注意，因为该样式控制了整个body样式（超出边界是自动隐藏），然后给其内部的modal元素（或backdrop背景）做样式设置的时候提供了参考，但切勿通过此样式影响其他的内容布局等，所以要确保该样式不被其他功能或者插件所引用。

```
// 源码5093行
.modal-open {
  overflow: hidden;
}
```

至此，模态窗体的适应方法和源码分析就结束了。通过分析我们可

以看出它的设计精妙的部分，尤其是几个元素的z-index设置，以及在触发事件时候的各种样式的操作。最后还有可扩展的事件，比如在弹出前触发的show，弹出后触发的shown，以及关闭前触发的hide，和关闭后触发的hidden，都为我们做自定义插件提供了良好的参考。

5.3 下拉菜单

源文件：dropdown.js、dropdowns.less

CSS文件：bootstrap.css 2985行之后

第4章讲解下拉菜单组件的时候，只提到了菜单的展示，本节将详细讲述怎么结合JavaScript代码进行交互。

5.3.1 声明式用法

一般下拉菜单都是在导航条（navbar）和选项卡（tab）上实现，如图5-4所示的效果就是navbar的效果，单击每个Dropdown链接都可以弹出事先隐藏设置的菜单。

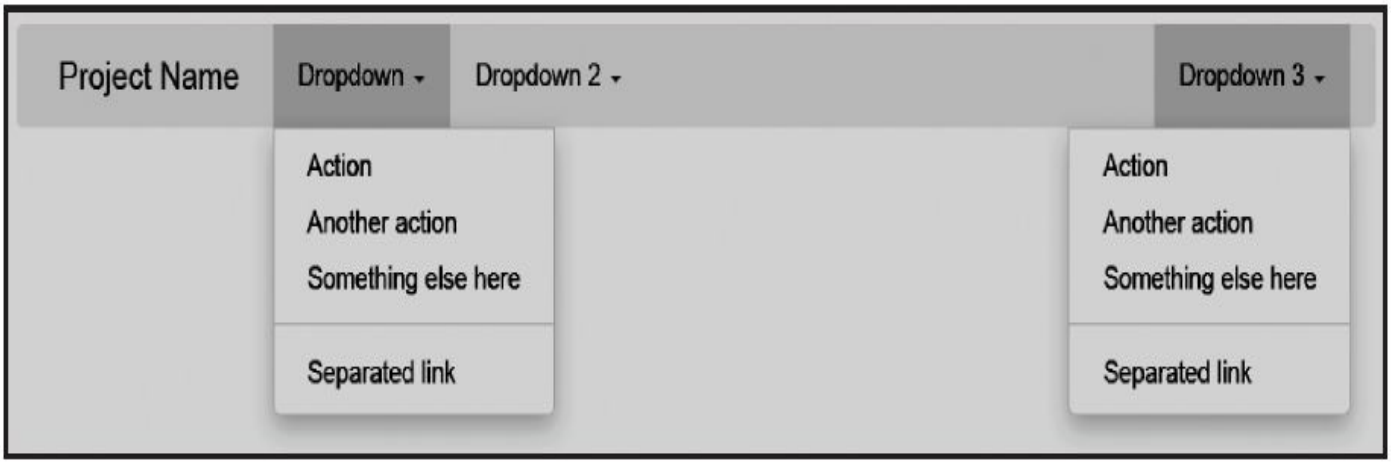


图5-4 下拉菜单运行效果

上述效果的示例代码如下：

```
<div class="navbar navbar-default" id="navbar-example">
    <a href="#" class="navbar-brand">Project Name</a>
    <ul role="navigation" class="nav navbar-nav">
        <li class="dropdown">
                            <a  data-toggle="dropdown"  class="dropdown-
toggle" role="button"
                href="#" id="drop1">Dropdown <b class="caret"></b></a>
               <ul aria-labelledby="drop1" role="menu" class="dropdown-
menu">
                <li role="presentation"><a href="#" tabindex="-1" role=
                    "menuitem">Action</a></li>
                …
            </ul>
        </li>
        <li class="dropdown">
                            <a  data-toggle="dropdown"  class="dropdown-
toggle" role="button"
                   id="drop2" href="#">Dropdown 2 <b class="caret"></b>
</a>
               <ul aria-labelledby="drop2" role="menu" class="dropdown-
menu">
                <li role="presentation"><a href="#" tabindex="-1" role=
                    "menuitem">Action</a></li>
                …
            </ul>
        </li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
        <li class="dropdown" id="fat-menu">
                            <a  data-toggle="dropdown"  class="dropdown-
```

```
toggle" role="button"
                    id="drop3" href="#">Dropdown 3 <b class="caret"></b>
</a>
                <ul aria-labelledby="drop3" role="menu" class="dropdown-
menu">
                 <li role="presentation"><a href="#" tabindex="-1" role=
                     "menuitem">Action</a></li>
                 …
             </ul>
         </li>
     </ul>
</div>
```

注意上述代码，首先navbar的父容器上要应用.navbar样式，其次顶级ul块上要应用.nav和.navbar-nav样式。Dropdown3应用了.navbar-right样式后，就变成了向右浮动对齐了。

另外一种运行在选项卡（tab）上的效果如图5-5所示。

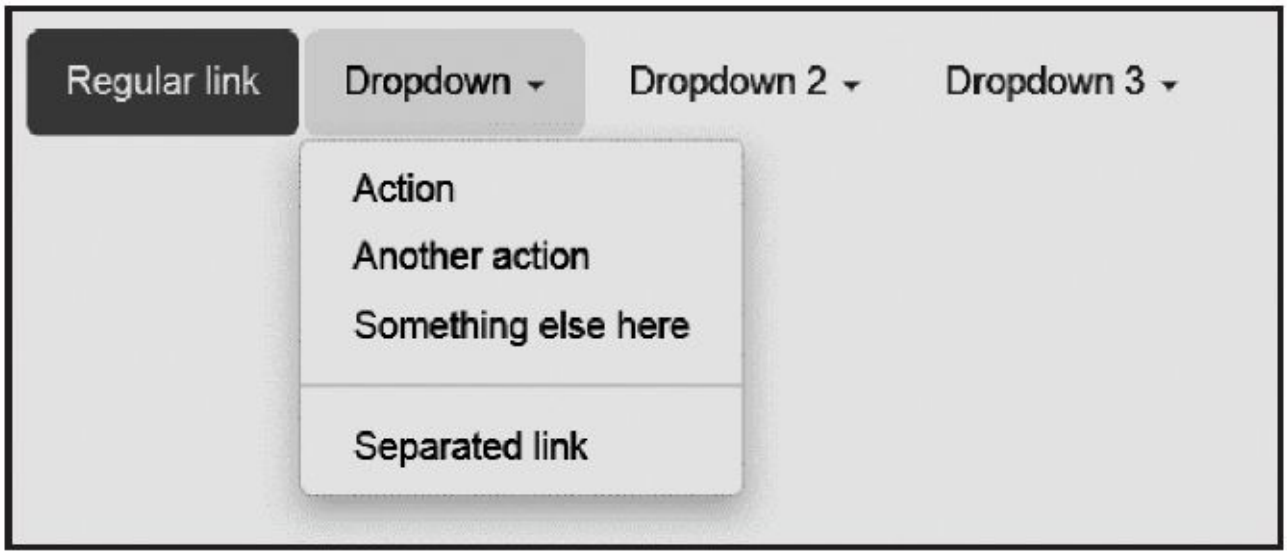


图5-5　选项卡上的下拉菜单运行效果

图5-5的实现HTML代码如下：

```
<ul class="nav nav-pills">
    <li class="active"><a href="#">Regular link</a></li>
    <li class="dropdown">
        <a href="#" data-toggle="dropdown" class="dropdown-toggle"
            role="button" id="drop4">Dropdown <b class="caret"></b></a>
              <ul  aria-labelledby="drop4"  role="menu"  class="dropdown-
menu" id="menu1">
                                                  <li    role="presentation">
<a href="#" tabindex="-1" role="menuitem">
                Action</a></li>
            …
        </ul>
    </li>
    <li class="dropdown">
                 <a  href="#"  data-toggle="dropdown"  class="dropdown-
toggle" role=
```

```
            “button” id=“drop5”>Dropdown 2 <b class=“caret”></b></a>
              <ul  aria-labelledby=“drop5”  role=“menu”  class=“dropdown-
menu” id=“menu2”>
                                                    <li    role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”>
                Action</a></li>
            …
          </ul>
      </li>
      <li class=“dropdown”>
                 <a  href=”#”  data-toggle=“dropdown”  class=“dropdown-
toggle” role=
            “button” id=“A1”>Dropdown 3 <b class=“caret”></b></a>
              <ul  aria-labelledby=“drop5”  role=“menu”  class=“dropdown-
menu” id=“menu3”>
                                                    <li    role=“presentation”>
<a href=”#” tabindex=”-1” role=“menuitem”
                >Action</a></li>
            …
          </ul>
      </li>
</ul>
```

根据上面的两种运行效果，可以发现，用户单击Dropdown链接或者小箭头的时候，隐藏的菜单会弹出来。在HTML编写方面只需要满足如下规则即可：

❑菜单样式和菜单项保持一致（即按照第4章的要求编写）。

❑被单击的链接或者按钮上需要应用data-toggle=“dropdown”样式，以便在初始化的时候JavaScript可以监控单击事件。

其实现原理如下：

❑Dropdown插件在网页加载的时候，对所有带有data-toggle=“dropdown”样式的元素进行事件绑定。

❑用户单击带有data-toggle=“dropdown”样式的链接或按钮时，触发JavaScript事件代码。

❑Javascript事件代码在父容器（本例是：<li class=“dropdown”>）上加一个.open样式。

❑默认隐藏的.dropdown-menu菜单在外部有了.open样式后，即可显示出来，从而达到预期的效果。

结合上面的例子和原理，其基本用法示例如下：

```
<div class=“dropdown”>
  <a class=“dropdown-toggle” data-toggle=“dropdown” href=”#”>下拉菜单触发
元素</a>
  <ul class=“dropdown-menu” role=“menu” aria-labelledby=“dLabel”>
    …
```

```
  </ul>
</div>
```

为了避免href=”#”被单击时页面跳到顶部，可以使用data-target=”#”代替href=”#”。

```
<div class=“dropdown”>
        <a    class=“dropdown-toggle”    data-toggle=“dropdown”    data-
target=”#” id=“dLabel”
      role=“button” href=”/page.html”>下拉菜单触发元素<span class=“caret”>
</span>
  </a>
  <ul class=“dropdown-menu” role=“menu” aria-labelledby=“dLabel”>
    …
  </ul>
</div>
```

关于data-target属性，还有一个有趣的用法，那就是触发元素可以放在菜单的父容器外部。如下示例中，触发元素是一个button按钮，而菜单在nav样式内的li元素里，这样单击button也能弹出默认隐藏的菜单项。

这种用法有两点需要注意：一是要设置父容器的id值（本例是menutest1），二是要设置button元素的data-toggle属性和data-target属性，而且data-target属性值是#id，即#menutest1。

```
<button  id=“btn”  data-toggle=“dropdown”  data-target=”#menutest1”>外部按
钮</button>
<ul class=“nav nav-pills”>
  <li class=“dropdown” id=“menutest1”> <!—必须在父元素上设置和data-target一
致的id值,
      即menutest1 —>
        <ul class=“dropdown-menu”>
            <li><a href=”#”>动作</a></li>
            …
            <li class=“divider”></li>
            <li><a href=”#”>被间隔的链接</a></li>
        </ul>
   </li>
</ul>
```

5.3.2 JavaScript用法

Bootstrap的Dropdown插件也支持JavaScript手动触发的方式，这样的好处是，可以按照自己的方式省略一些元素或者样式。示例如下：

```
<ul class="nav nav-pills">
    <li>
                        <a   href="#"   id="btnLink"   class="dropdown-
toggle">Dropdown <b class=
            "caret"></b></a>
        <ul class="dropdown-menu">
            <li><a href="#" tabindex="-1">Action</a></li>
            …
            <li class="divider"></li>
            <li><a href="#" tabindex="-1">Separated link</a></li>
        </ul>
    </li>
</ul>
```

按照上述布局，我们只需要对链接元素设置dropdown绑定即可，可以使用id，也可以使用class样式，本例中是对dropdown-toggle样式进行绑定。源码如下：

```
<script type="text/javascript">
    $(document).ready(
    function () {
                            $('.dropdown-toggle').dropdown();   //   使用
$('#btnLink').dropdown();也可以
    });
</script>
```

和Modal一样，dropdown也接收字符串作为参数进行传递，例如dropdown('toggle')。但这非常不好用，因为一旦执行了该语句，插件就在系统再次注册了单击事件进行toggle，这样如果再执行dropdown('toggle')语句，每次单击都要两次togle，就会一直是一个不变的状态（下一章节的源码会提到这个问题）。所以，一般情况下，使用示例中不带参数的dropdown方法进行注册，即便非要使用带toggle参数的，建议使用jQuery的one方法（即第一次单击的时候注册，以后就不管了，因为系统已经自动注册了）。

```
$("#btnDropdown").one("click", function () {
    $(this).dropdown('toggle');
});
```

同模态弹窗一样，下拉菜单也支持4种类型的事件订阅，分别对应下拉菜单的弹出前、弹出后、关闭前、关闭后，如表5-5所示。

表 5-5　下拉菜单组件的事件类型

事件类型	描　述
show.bs.dropdown	在 show 方法调用时立即触发（尚未显示之前）
shown.bs.dropdown	在下拉菜单完全显示给用户之后（并且等 CSS 动画完成之后）触发
hide.bs.dropdown	在 hide 方法调用时（但还未关闭隐藏）立即触发
hidden.bs.dropdown	在下拉菜单完全隐藏之后（并且等 CSS 动画完成之后）触发

调用方式也很简单，和普通的jQuery代码并无二样。

```
$(‘#myDropdown’).on(‘show.bs.dropdown’, function () {
    // do something…
})
```

5.3.3 源码分析

分析源码之前，我们先来回顾一下下拉菜单的HTML布局规则的定义。示例如下：

```
<div class="dropdown">
                              <a       class="dropdown-toggle"       data-
toggle="dropdown" href="#">Dropdown trigger</a>
    <ul class="dropdown-menu" role="menu" aria-labelledby="dLabel">
        …
    </ul>
</div>
```

上述示例可以看出下拉菜单的几个重要属性：触发属性（data-toggle="dropdown"）、目标父容器属性（href="#"）、按钮元素容器（带dropdown-menu样式的<ul>）。

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2 插件核心代码。Dropdown插件的核心源码如下：

```
var backdrop = '.dropdown-backdrop'          // 弹出下拉菜单时的背景样式
var toggle = '[data-toggle=dropdown]'    // dropdown触发元素的自定义属性
var Dropdown = function (element) {
    var $el = $(element).on('click.bs.dropdown', this.toggle)
    // 插件类函数定义，一旦触发，就在click事件上绑定toggle，所以不能再用自定义代码
进行toggle了
}
// 控制下拉菜单的打开、关闭操作
Dropdown.prototype.toggle = function (e){};
// 利用箭头控制下拉菜单（例如，按向下箭头的时候，打开下拉菜单）
Dropdown.prototype.keydown = function (e){};
// 关闭所有的下拉菜单
function clearMenus(){};
// 获取下拉菜单的父元素容器
function getParent($this) {};
```

上述源码，除了定义Dropdown插件类函数外，还定义了两个原型方法和两个内部函数。原型方法toggle和keydown分别是处理下拉菜单的展开、关闭操作，以及支持按键对下拉菜单的选择操作的功能。clearMenus函数用于在打开特定的下拉菜单之前，关闭所有其他的下拉菜单；getParent函数则用于查找目标元素（即下拉菜单的父元素用于在上面判断是否有open样式，不过通常触发元素和菜单项都同属于一个父元素容器，就像示例中的代码一样）。代码如下：

```
// 控制下拉菜单的打开、关闭操作
Dropdown.prototype.toggle = function (e) {
    var $this = $(this)
```

```
    if ($this.is('.disabled, :disabled')) return          // 如果有禁用标记，则不处理

    var $parent = getParent($this)                 // 获取触发元素的父元素
    var isActive = $parent.hasClass('open')       // 判断父元素是否有open样式，有，
                                                                  // 则代表当前下拉菜单正在打开状态

    clearMenus()  // 关闭所有其他的下拉菜单，即同一个页面，只允许同时出现一个下拉菜单

    // 判断：单击时当前下拉菜单是否是打开状态
    // 如果是，在clearMenus阶段就已经关闭了，所以就不需要再次关闭了
    // 如果不是，说明默认是关闭状态，则需要展开下拉菜单
    if (!isActive) {
        // 如果是移动设备，则使用dropdown-backdrop样式，因为移动设备不支持click单击委托
        if ('ontouchstart' in document.documentElement && !$parent.close
            ('.navbar-nav').length) {
                                                $('<div    class="dropdown-backdrop"/>').insertAfter($(this)).on
                ('click', clearMenus)
        }

        var relatedTarget = { relatedTarget: this }
        $parent.trigger(e = $.Event('show.bs.dropdown', relatedTarget))
                                                                  // 展开下拉菜单前，触发show事件

         if (e.isDefaultPrevented()) return        // 如果已经阻止了默认行为，就不再处理了

        $parent
            .toggleClass('open')                  // 添加open样式，打开下拉菜单，因为
                                                                   // dropdown-menu在open样式内会自动显示
            .trigger('shown.bs.dropdown', relatedTarget)
                                                                  // 展开下拉菜单后，触发shown事件

        $this.focus()           // 给当前触发元素加上焦点
    }

    return false                            // 阻止该元素后续的默认行为
}
// 利用箭头控制下拉菜单（例如，按向下箭头的时候，打开下拉菜单）
Dropdown.prototype.keydown = function (e) {
    // 如果按键不是Esc、或上下方向箭头，则忽略处理
    if (!/(38|40|27)/.test(e.keyCode)) return

    var $this = $(this)
    e.preventDefault()          // 阻止默认行为
    e.stopPropagation()         // 阻止冒泡
```

```
     if ($this.is('.disabled, :disabled')) return // 如果有禁用标记，则不做处理

    var $parent = getParent($this)      // 获取触发元素的父元素
    var isActive = $parent.hasClass('open')
                                    // 判断父元素是否有open样式，有，则代表当前下拉菜单正在打开状态

    // 如果有open样式，或者没有open样式（但按键是向下箭头的话），也打开下拉菜单
    if (!isActive || (isActive && e.keyCode == 27)) {
        if (e.which == 27) $parent.find(toggle).focus()
                                            // 如果按了向下箭头，则给触发元素加上焦点
        return $this.click()            // 默认单击事件，打开下拉菜单
    }
    // 返回可以利用箭头选择的下拉菜单项
            //   打开下拉菜单时，上下箭头只操作（选中）设置了role   =   menu（或role=listbox）的链接项
    // 必须是可见的a链接，并且不包括分隔符
    var desc = ' li:not(.divider):visible a'
    var $items = $parent.find('[role=menu]' + desc + ', [role=listbox]'

    if (!$items.length) return  // 如果没有，则不做处理
    var index = $items.index($items.filter(':focus'))
                                            // 找出当前处于焦点状态的第一个下拉菜单项的索引

    if (e.keyCode == 38 && index > 0) index—   // 按向上箭头的话，index减1
    if (e.keyCode == 40 && index < $items.length - 1) index++
                                            // 按向下箭头的话，index加1
    if (!~index) index = 0              // 特殊意外情况，设置为第一个菜单项

    $items.eq(index).focus()    // 给所选择的菜单项设置焦点
}
// 关闭所有的下拉菜单
function clearMenus(e) {
    $(backdrop).remove()                // 删除用于移动设备的背景
    $(toggle).each(function () {         // 根据选择器，遍历所有的dropdown标记，然后全部关闭
        var $parent = getParent($(this))
        var relatedTarget = { relatedTarget: this }
        if (!$parent.hasClass('open')) return
        $parent.trigger(e = $.Event('hide.bs.dropdown', relatedTarget))
                                                                // 关闭前，触发hide事件
        if (e.isDefaultPrevented()) return
        $parent.removeClass('open').trigger('hidden.bs.dropdown', relate
                                                                // 关闭后，触发hidden事件
    })
}
// 获取下拉菜单的父元素容器
function getParent($this) {
    var selector = $this.attr('data-target')
                                    // 如果设置了target，就针对该traget元素进行处理
```

```
    if (!selector) {
        selector = $this.attr('href')            // 如果没有target，则查找
href里设置的值
                                selector    =    selector    &&    /#[A-Za-
z]/.test(selector) && selector.replace
            (/.*(?=#[^\s]*$)/, '')
    }
    var $parent = selector && $(selector)        // 如果上述两个步骤满足其
一，设置其为
                                                          parent元素（下
拉菜单的容器元素）
    // 如果都不满足，就使用当前触发元素（$this）的父元素
    return $parent && $parent.length ? $parent : $this.parent()
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.dropdown插件的代码和Modal插件类似，也是遍历元素、实例化或触发动作。源码如下：

```
var old = $.fn.dropdown
//  保留其他库的$.fn.dropdown代码（如果定义的话），以便在noConflict之后，可以继续
使用该老代码
$.fn.dropdown = function (option) {
    return this.each(function () {              // 根据选择器，遍历所有符合
规则的元素
        var $this = $(this)
        var data = $this.data('bs.dropdown')    // 获取自定义属性dropdown
的值

        // 如果值不存在，则将Dropdown实例设置为data-dropdown的值
        if (!data) $this.data('bs.dropdown', (data = new Dropdown(this)))
        // 如果option传递了string，则表示要执行某个方法
                    //   比如传入了toggle，则要执行Dropdown实例的toggle方法，
data["toggle"]相当于
         data.toggle();
        if (typeof option == 'string') data[option].call($this)
    })
}
$.fn.dropdown.Constructor = Dropdown // 重设插件构造器,可以通过该属性获取插件
的真实类函数
```

步骤4　防冲突处理。源码如下：

```
// 防冲突处理
$.fn.dropdown.noConflict = function () {
    $.fn.dropdown = old                          // 恢复以前的旧代码
    return this  // 将$.fn.dropdown.noConflict()设置为Bootstrap的dropdown
插件
}
```

步骤5　绑定触发事件。源码如下：

```
// 绑定触发事件
$(document)
    // 为声明式的HTML绑定单击事件，在单击以后先关闭左右的下拉菜单
    .on('click.bs.dropdown.data-api', clearMenus)
```

```
    // 如果内部有form元素，则阻止冒泡，不做处理
                                                .on('click.bs.dropdown.data-
api', '.dropdown form', function (e) { e.stopPropagation() })
    // 默认行为，一般都要进行toggle操作（打开或关闭）
                                                .on('click.bs.dropdown.data-
api', toggle, Dropdown.prototype.toggle)
    // 为触发元素和下拉菜单项绑定keydown按钮事件，以便可以进行打开或选择操作
          //  toggle  =  [data-toggle=dropdown]表示所有带有自定义属性data-
toggle="dropdown"的元素
                                              .on('keydown.bs.dropdown.data-
api', toggle + ', [role=menu], [role=listbox]',
         Dropdown.prototype.keydown)
```

在绑定触发事件上，dropdown插件做了很多操作，首先是确保关闭所有的下拉菜单，如果有form元素，则进行特殊忽略处理，接着才绑定真正的单击事件所要触发的内容（Dropdown.prototype.toggle），最后也提供了keydown事件的绑定，用于让键盘方向键也可以选择下拉菜单项。

5.4　滚动侦测

源文件：scrollspy.js

滚动侦测（ScrollSpy）插件是根据滚动的位置自动更新导航条中相应的导航项，如图5-6所示。拖动右面区域的滚动条，当滚动区域到达@mdo时，上面的@mdo菜单就会高亮，这是因为该插件可自动检测到达哪个位置了，然后在需要高亮的菜单父元素上加了一个active样式。

图5-6　滚动侦测效果

如果导航里有下拉菜单，并且滚动区域的内容到达下拉菜单子项所对应的区域，除了子菜单高亮以外，子菜单的父元素（DropDown按钮）也会高亮，效果如图5-7所示。

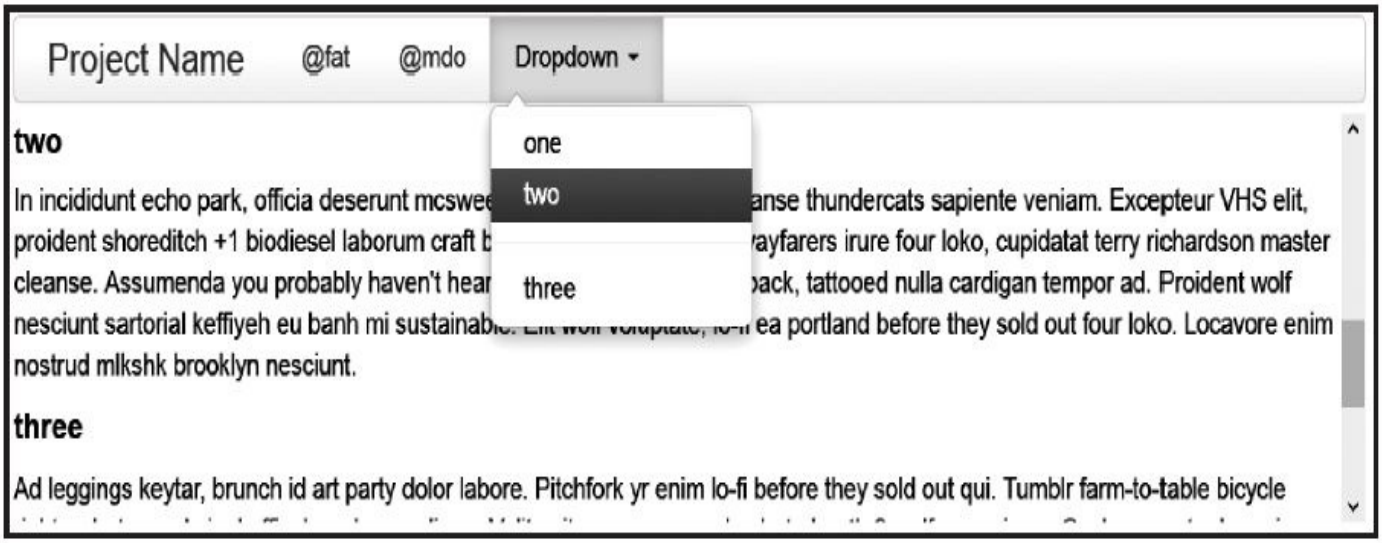


图5-7　滚动侦测支持子菜效果

注意

❑子菜单虽然会高亮，但不会打开，为了展示效果，需要手动打开下

拉菜单查看。

❑该插件所需要的导航条只能在拥有.nav样式的ul/li元素上起作用。

5.4.1 声明式用法

在平时使用过程中，滚动侦测一般有两种用法，一种是像上面的那样固定一个元素的高度，进行滚动，然后对相应的菜单进行高亮显示；另外一种是对整个页面（body）进行滚动侦测。两种方式的用法一样，都需要有如下3个步骤：

1）设置滚动容器，即在所要侦测的元素上设置data-target=”#selector” data-spy=“scroll”属性。

2）设置菜单链接容器，该容器的id（或样式）和data-target属性所对应的选择符应一致。

3）在菜单容器内，必须有.nav样式的元素，并且在其内部有li元素，li内包含的a元素才是可以侦测高亮的菜单链接，即符合.nav li > a这种选择符的条件。

第一种用法示例如下：

```
<div id=“selector”><!— 菜单容器 —>
                <nav    class=“navbar    navbar-default    navbar-fixed-
top” role=“navigation”>
        <ul class=“nav navbar-nav”>
              <li class=“active”><a href=”#fat”>@fat</a></li><!— 第1个链
接 —>
            <li class=””><a href=”#mdo”>@mdo</a></li><!— 第2个链接 —>
            <li class=“dropdown”>
                <a data-toggle=“dropdown” class=“dropdown-toggle” href=
                    ”#”>Dropdown <b class=“caret”></b></a>
                <ul class=“dropdown-menu”>
                     <li class=””><a href=”#one”>one</a></li><!— 第3个链
接 —>
                     <li class=””><a href=”#two”>two</a></li><!— 第4个链
接 —>
                    <li class=“divider”></li>
                     <li class=””><a href=”#three”>three</a></li><!— 第5
个链接 —>
                </ul>
            </li>
        </ul>
    </nav>
</div>
<!—滚动侦测区域 —>
<div data-offset=“0” data-target=”#selector” data-spy=“scroll” style=”
    确保设置了固定高度，并且overflow: auto”>
    <h4 id=“fat”>@fat</h4><p>具体内容</p>
    <h4 id=“mdo”>@mdo</h4><p>具体内容</p>
    <h4 id=“one”>one</h4><p>具体内容</p>
    <h4 id=“two”>two</h4><p>具体内容</p>
    <h4 id=“three”>three</h4><p>具体内容</p>
</div>
```

第二种用法示例和第一种类似，只不过将滚动侦测容器挪到了body上，而nav一定要在body内部。

```
<body data-target=”#selector” data-spy=“scroll”>
                <nav    class=“navbar    navbar-default    navbar-fixed-
top” role=“navigation” id=” selector ”>
        …
    </nav>
    <h4 id=“fat”>@fat</h4>
    <p>具体内容</p>
    <!— 省略其他链接 —>
</body>
```

从上述两个例子可以看出，除了两个容器以外，滚动侦测容器内必须有上述菜单项链接所对应的id选择符，即如果要高亮显示href=”#fat”所在的菜单，就必须有id=“fat”的元素，其显示范围超过当前滚动区域才行。

5.4.2 JavaScript用法

滚动侦测的JavaScript用法相对比较简单，只需要指定两个容器的名称即可。示例如下：

```
用法：$('滚动侦测容器选择符').scrollspy({ target: '#菜单容器的选择符' })
示例：$('body').scrollspy({ target: '#selector' })
```

如果需要经常在滚动侦测的容器内部进行DOM更新，则更新以后为了让该滚动侦测功能继续能用的话，需要使用如下代码进行刷新绑定相关的事件。示例代码如下：

```
$('[data-spy="scroll"]').each(function () {
    var $spy = $(this).scrollspy('refresh')
})
```

需要注意的是，这种refresh只对声明式用法有效，如果使用的是JavaScript方式，并且需要刷新DOM，那就需要重新应用该插件；或者从data-scrollspy属性上获取该实例，然后再调用refresh方法了。

第一个示例中的target是作为options参数传递进去的。除了该参数以外，滚动侦测容器还支持另外一个offset参数，用法如表5-6所示。

表 5-6 Scrollspy 组件的 JavaScript 用法描述

属性名称	类　型	默 认 值	描　述
offset	数字	10	默认的 offset，即滚动内容距离滚动容器 10 像素以内的话，就高亮显示所对应的菜单

滚动侦测也支持事件的订阅和触发功能，目前只支持一个activate事件，描述如表5-7所示。

表 5-7 Scrollspy 组件的 JavaScript 用法描述

属性名称	描　述
activate.bs.scrollspy	当滚动侦测插件将某个元素设置为 active 样式时，触发此事件

事件用法如下：

```
$('#fat').on('activate.bs.scrollspy', function () {
                    // 处理代码…
})
```

注意

分析源码以后，笔者发现在触发该事件的时候使用的是trigger('activate')，没有使用全名称的命名空间，所以也可以像如下代码这样绑定事件：

```
$(‘#fat’).on(‘activate, function () {
                    // 处理代码…
})
```

5.4.3 源码分析

在源码分析之前，再来确保一下已经理解了滚动侦测组件的HTML结构。首先要满足如下几点：

❑有滚动容器，设置了属性data-target=”#selector” data-spy=“scroll”，并且有相对的id元素。

❑有菜单容器，设置id为滚动容器里设置的data-target属性selector。

❑菜单容器里的菜单必须是nav组件，即有ul上有nav样式，li元素里有a元素或者下拉菜单。

❑如果有下拉菜单，则要将nav样式的li元素作为下拉菜单的父容器。

步骤1 立即调用的函数。此步骤与Modal插件一样，此处不赘述。

步骤2 插件核心代码。主要是Scrollspy核心类函数的定义、滚动容器、菜单容器的确定、计算当前页面内菜单容器的所有data-target（或href）值，并且确保这些值和滚动容器内的id值都匹配，然后进行第一次位置调整处理。核心代码如下：

```
function ScrollSpy(element, options) {
    var href
var process = $.proxy(this.process, this)        // 代理对象，防止this作用域失效

    // 如果元素是body，则用window对象选择器，如果不是，则保持
    this.$element = $(element).is('body') ? $(window) : $(element)

    this.$body = $('body')
    // 给滚动容器绑定滚动事件
     this.$scrollElement = this.$element.on('scroll.bs.scroll-spy.data-api', process)

    // 将默认值和传进来的option参数合并，后者优先级高
    this.options = $.extend({}, ScrollSpy.DEFAULTS, options)

    // 设置菜单容器（也就是滚动到一个点时候，那个菜单要变化）
    // 如果option里设置了traget，即data-target有值，则优先使用
    // 如果没有，则查找滚动容器上有没有href值，如果有的话，也可使用
     //  如果以上都没有，则查找通用.nav样式的子元素，即.nav样式内的li子元素内的a链接，作为菜单容器
    this.selector = (this.options.target
          || ((href = $(element).attr('href')) && href.replace(/.*(?=#[^\s]+$)/, '')) // strip for ie7
        || '') + ' .nav li > a'
    this.offsets = $([])
    this.targets = $([])
    this.activeTarget = null            // 高亮显示的菜单
    this.refresh() // 计算当前页面内所有滚动容器内的id集合和每个id元素距离浏览器顶部的像素距离
```

```
    this.process()                          // 开始正式处理
}
ScrollSpy.DEFAULTS = {
    offset: 10                                      // 默认offset
}
// 计算当前页面内所有滚动容器内的id集合和每个id元素距离浏览器顶部的像素距离
ScrollSpy.prototype.refresh = function (){};
// 每次滚动都要对滚动高度进行判断，然后设置高亮菜单元素
ScrollSpy.prototype.process = function (){};
// 设置高亮菜单元素
ScrollSpy.prototype.activate = function (target) {};
```

上面代码里的3个原型方法的作用分别如下：

❑refresh方法用于获取页面内所有滚动容器的id值（也可以说是菜单对应的target值（或href））的集合，以及每个id元素的offset值。

❑prcess用于在滚动以后判断所有id元素的offset，然后定位需要高亮显示的菜单。

❑activate是具体设置菜单高亮的方法，主要是在菜单的父元素（通常是li）上加上active样式。

上述原型方法的详细源码如下：

```
// 计算当前页面内所有滚动容器内的id集合和每个id元素距离浏览器顶部的像素距离
ScrollSpy.prototype.refresh = function () {
    var offsetMethod = this.$element[0] == window ? 'offset' : 'positior

    this.offsets = $([])
    this.targets = $([])

    var self = this
    var $targets = this.$body
        .find(this.selector)
        .map(function () {
            var $el = $(this)
            var href = $el.data('target') || $el.attr('href')
            var $href = /^#./.test(href) && $(href)

            // 返回一个二维数组，每个滚动容器内的id对象到页面顶部的距离
            // 以及高亮菜单容器里所对应的href的值（也就是滚动容器里的id值）
            return ($href
            && $href.length
            && $href.is(':visible')
                                                &&    [[$href[offsetMethod]
().top + (!$.isWindow(self.$scrollElement.get(0))
                  && self.$scrollElement.scrollTop()), href]]) || null
        })
        .sort(function (a, b) { return a[0] - b[0] })
        .each(function () {
            self.offsets.push(this[0])  // 收集所有的偏离值（距离top的距离）
             self.targets.push(this[1])  // 收集菜单容器里的所有href值（也就
是滚动容器里的id值）
// 两者必须都满足才行，而且包含#号
```

```
        })
}
// 每次滚动都要对滚动高度进行判断，然后设置高亮菜单元素
ScrollSpy.prototype.process = function () {
    // 获取滚动容器距离顶部的距离，再加上默认设置的offset（10像素）
    var scrollTop = this.$scrollElement.scrollTop() + this.options.offse

    // 获取滚动容器里的内容高度
    var scrollHeight = this.$scrollElement[0].scrollHeight || this.$body

    // 最大可以滚动的高度=总内容高度-滚动容器设置的高度
    // 比如，内容高度300像素，而滚动容器设置了100像素，则最大滚动高度为200像素
     // 如果滚动容器是body，则内容高度一般是超过一屏幕的高度，滚动容器高度则是浏览器
屏幕的高度

    var maxScroll = scrollHeight - this.$scrollElement.height()
    var offsets = this.offsets          // 所有的偏离值
    var targets = this.targets          // 所有的id值（或href值），包含#号
    var activeTarget = this.activeTarget        // 当前高亮对象
    var i
    // 如果超过最大滚动高度，说明已经滚动到底了
    if (scrollTop >= maxScroll) {
        // 如果还没有设置最后一个元素高亮，则设置最后一个元素高亮
                    return   activeTarget   !=   (i   =   targets.last()
[0]) && this.activate(i)
    }
    if (activeTarget && scrollTop <= offsets[0]) {
        return activeTarget != (i = targets[0]) && this.activate(i)
    }
    for (i = offsets.length; i—;) {    // 遍历所有元素的offset，从最后一个元
素开始循环
        activeTarget != targets[i]      // 如果i元素不等于当前高亮元素
            && scrollTop >= offsets[i]  // 并且滚动高度大于i元素的offset
        // 并且i+1元素不存在，或者存在但大于滚动高度
            && (!offsets[i + 1] || scrollTop <= offsets[i + 1])
            && this.activate(targets[i])        // 则设置i元素为高亮元素
    }
}
// 设置高亮菜单元素
ScrollSpy.prototype.activate = function (target) {
    this.activeTarget = target          // 赋值实例属性

    $(this.selector)
        .parentsUntil(this.options.target, '.active')
         .removeClass('active')  // 删除其他高亮元素的active样式（active样式
控制高亮显示）

    // 查找菜单容器内符合[data-target="#' + 所高亮元素的id + '"]属性的元素
    // 或者href值是#' + 所高亮元素的id + '的话，也可以
    var selector = this.selector +
        '[data-target="' + target + '"],' +
        this.selector + '[href="' + target + '"]'

    var active = $(selector)
        .parents('li')
        .addClass('active')             // 查找父元素li，然后添加active高亮
```

样式

```
    // 如果li元素的父元素有dropdown-menu样式，则表示是一个dropdown下拉菜单
    if (active.parent('.dropdown-menu').length) {
        active = active
            .closest('li.dropdown')
             .addClass('active')  // 则需要给dropdown的li元素也加上active高亮样式
    }
    active.trigger('activate.bs.scrollspy')     // 触发自定义高亮事件
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.　　scrollspy插件，和Modal插件一样，在实例化之前，需要收集option参数。源码如下：

```
var old = $.fn.scrollspy
//  保留其他库的$.fn.scrollspy代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.scrollspy = function (option) {
    return this.each(function () {      // 根据选择器，遍历所有符合规则的元素
        var $this = $(this)
            var  data  =  $this.data('bs.scrollspy')     //  获取自定义属性bs.scrollspy的值（当前实例）

        // 如果option参数是对象，则作为ScrollSpy的参数传入
        var options = typeof option == 'object' && option

        // 如果值不存在，则将ScrollSpy实例设置为bs.scrollspy的值
        if (!data) $this.data('bs.scrollspy', (data = new ScrollSpy(this

        // 如果option传递了string，则表示要执行某个方法
                    //   比如传入了refresh，则要执行ScrollSpy实例的refresh方法，data["refresh"]相当于
        // data.refresh()
        if (typeof option == 'string') data[option]()
    })
}
$.fn.scrollspy.Constructor = ScrollSpy           // 重设插件构造器,可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。源码如下：

```
// 防冲突处理
  $.fn.scrollspy.noConflict = function () {
    $.fn.scrollspy = old                 // 恢复以前的旧代码
        return  this    //  将$.fn.scrollspy.noConflict()设置为Bootstrap的scrollspy插件
}
```

步骤5　绑定触发事件。源码如下：

```
// 绑定触发事件
$(window).on('load', function () {
     $('[data-spy="scroll"]').each(function () {  // 遍历所有符合条件的滚动容器
```

```
        var $spy = $(this)
         $spy.scrollspy($spy.data())      // 执行scrollspy插件，并传入滚动容器上设置的
                                                                    // 自定义参数（data-开头）
    })
})
```

绑定触发事件相对比较简单，主要是查找符合条件的元素，然后遍历实例化scrollspy插件。大家可能有点疑惑，能不能给一个比较完整的绑定，比如如下这样的代码：

```
$(document).ready(
    function () {
        $(‘[data-spy=“scroll”]’).scrollspy();
    }
);
```

这样其实也不是不可以，但是需要修改第三步的插件代码，在实例化之前，需要获取当前滚动容器实例所需要的option，也就是data()参数。读者可以自行修改。

5.5　选项卡

源文件：tab.js

选项卡（Tab）组件是Bootstrap提供的一种非常常用的功能，其就像我们平时使用的Windows操作系统里的选项卡设置一样，单击一个选项，下面就显示对应的选项卡面板。而对应于网页，其行为也是非常类似的，一个示例效果如图5-8所示。

图5-8　选项卡效果

通过效果我们可以看出，选项卡由两部分组成：CSS选项卡组件+底部可以切换的选项卡面板。由于CSS选项卡组件在第4章已经讲述了，所以本节将着重在JavaScript代码实现上。

5.5.1 声明式用法

声明用法也比较简单，除了CSS导航选项卡的声明方式保持不变以外，下面添加选项卡面板，然后再设置对应的id即可。声明式用法要满足如下几点要求：

❑选项卡导航和选项卡面板要同时有（但不一定要放在一起）。

❑导航链接里要设置data-toggle=“tab”，并且还要设置data-target=”选择符”（或href=”选择符”），以便单击的时候能找到该选择符所对应的tab-pane面板。

❑tab-pane要放在tab-content里面，要有id（或者CSS样式）并与data-target=”选择符”（或href=”选择符”）一致。

选项卡的HTML代码示例如下，按照如下规则设置好以后，不再需要调用任何JavaScript代码，这一切都由Bootstrap.js全部帮你初始化好了。

```
<!– 导航选项卡–>
<ul class=“nav nav-tabs”>
    <li><a href=”#home” data-toggle=“tab”>Home</a></li>
    <li><a href=”#profile” data-toggle=“tab”>Profile</a></li>
    <li><a href=”#messages” data-toggle=“tab”>Messages</a></li>
    <li><a href=”#settings” data-toggle=“tab”>Settings</a></li>
</ul>
<!– 选项卡面板 –>
<div class=“tab-content”>
    <div class=“tab-pane active” id=“home”>…</div>
    <div class=“tab-pane” id=“profile”>…</div>
    <div class=“tab-pane” id=“messages”>…</div>
    <div class=“tab-pane” id=“settings”>…</div>
</div>
```

关于面板的隐藏和显示样式控制，其实Bootstrap只是提供了两个样式而已，分别用于控制隐藏和显示。

默认情况下，.tab-content下的.tab-pane是隐藏的，一旦应用了.active样式，该元素就会以块级元素进行显示。源码如下：

```
// 源码3638行
.tab-content > .tab-pane {
  display: none;                /*隐藏*/
}
.tab-content > .active {
  display: block;       /*块级显示*/
}
```

为了让隐藏/显示的切换效果更流畅，可以在面板上使用fade样式产生渐入的效果。不过，别忘记了，如果页面初始化的时候就有一个默认显示的面板（并且需要渐入效果），则还要加上in样式。示例如下：

```
<div class=“tab-content”>
    <div class=“tab-pane fade in active” id=“home”>…</div>
    <div class=“tab-pane fade” id=“profile”>…</div>
    <div class=“tab-pane fade” id=“messages”>…</div>
    <div class=“tab-pane fade” id=“settings”>…</div>
</div>
```

Bootstrap不仅支持tabs的导航，还支持胶囊式选项卡导航，使用的时候需要把nav-tabs替换为nav-pills，还要把data-toggle的tab替换为pill。示例如下：

```
<ul class=“nav nav-pills”>
    <li><a href=”#home” data-toggle=“pill”>Home</a></li>
    <li><a href=”#profile” data-toggle=“pill”>Profile</a></li>
    <li><a href=”#messages” data-toggle=“pill”>Messages</a></li>
    <li><a href=”#settings” data-toggle=“pill”>Settings</a></li>
</ul>
<div class=“tab-content”>
    …
</div>
```

注意

❑笔者在使用过程中，发现不管使用nav-tabs还是使用nav-pills，其data-target里的值都可以随意设置，即：tab和pill设置哪一个都行，效果都是一样的。

❑笔者还发现，选项卡插件对堆叠式导航（nav-stack）也是支持的，只不过一般情况下，需要自行处理一下，要把面板放在右边，以做成左右形式的布局。

5.5.2 JavaScript用法

如果不使用HTML声明式绑定（即声明data-toggle），选项卡组件也是支持JavaScript代码直接初始化的。代码示例如下：

```
$(‘.nav a’).click(function (e) {
    e.preventDefault()
    $(this).tab(‘show’)
})
```

使用JavaScript代码的效果与在nav里面的a链接上设置data-toggle=“tab”属性是一样的，最终效果都是查询所单击的元素（a元素），然后查找其data-target（或者href）所对应的选择符，然后显示该选择符对应的面板（并隐藏其他面板）。

所以根据上述原理，如果要对单个的选项卡进行调用的话，也是可以的。对单个选项卡调用的方法有如下几种：

```
$(‘.nav a[href=”#profile”]’).tab(‘show’)   // 通过元素名称查询
$(‘.nav a:first’).tab(‘show’)                  // 查询第一个tab
$(‘.nav a:last’).tab(‘show’)                   // 查询最后一个tab
$(‘.nav li:eq(2) a’).tab(‘show’)               // 查询第3个tab（索引从0开始）
```

选项卡组件目前只支持两种类型的事件订阅，分别对应下拉菜单的弹出前、弹出后。解释如表5-8所示。

表 5-8 选项卡组件的事件类型

属性名称	描　述
show.bs.tab	该事件在 tab 即将显示，但是还未显示之前触发。如果可能的话，将 event.target 设置为当前活动 tab，将 event.relatedTarget 设置为上一个 tab
shown.bs.tab	该事件在 tab 完全显示之后（CSS 动画也要结束）才触发。如果可能的话，将 event.target 设置为当前活动 tab，将 event.relatedTarget 设置为上一个 tab

事件的调用方式也很简单，但需要注意，一个事件e的两个属性分别代表两个不同的tab对象。

```
$(‘a[data-toggle=“tab”]’).on(‘shown.bs.tab’, function (e) {
    e.target                                    // 当前单击的tab
    e.relatedTarget                             // 前一个tab
})
```

5.5.3 源码分析

在源码分析之前，我们再来回顾以下Tab选项卡组件的原理要点：

❑单击一个元素时，首先将原来高亮（active）的元素取消高亮（即去除.active样式）。

❑然后给被单击元素添加.active样式，使之高亮。

❑如果该单击元素是下拉菜单里的子元素，则下拉菜单所在的父容器li也要高亮设置。

❑如果定义了动画，或者回调函数，就相应地进行执行。

步骤1 立即调用的函数。此步骤与Modal插件一样，此处不赘述。

步骤2 插件核心代码。主要是Tab核心类函数的定义、事件触发、新旧元素的高亮设置等。核心代码如下：

```
var Tab = function (element) {
    this.element = $(element)            // 指定当前元素
}
// 显示逻辑，触发show和shown事件
Tab.prototype.show  = function (){};
// active样式的应用，面板的隐藏和显示，以及tab的高亮与反高亮都使用了该方法
Tab.prototype.activate = function (){};
```

上面代码里的两个原型方法的作用分别如下：

❑show方法主要是触发show事件，然后调用activate原型方法，最后再触发shown事件。

❑activate用于取消原来元素的高亮设置，对现有元素进行高亮设置，然后触发动画和回调函数。

上述原型方法的详细源码如下：

```
// 显示逻辑，触发show和shown事件
Tab.prototype.show = function () {
    var $this = this.element              // 当前tab
    var $ul = $this.closest('ul:not(.dropdown-menu)')
                                                  // 找到最近的ul（但不是下拉
菜单），其实就是父元素
     var selector = $this.data('target') // 找到当前tab上的data-target属性
值

    if (!selector) {                      // 如果data-target属性不存在
        selector = $this.attr('href')    // 再查找href值
                   selector   =   selector   &&   selector.replace(/.*(?=#
[^\s]*$)/, '')  // IE7特殊处理
    }

    // 如果当前tab已经是活动状态了，即父元素li上有active样式的话，直接返回，不做任
```

```
何处理
    if ($this.parent('li').hasClass('active')) return

    // 找到上一个元素，即上一个带有active样式的li里的a元
    var previous = $ul.find('.active:last a')[0]

      //  设置show事件，注意此时的relatedTarget是上一个元素（自定义回调时使用，见
JavaScript用法里的示例）
    var e = $.Event('show.bs.tab', {
        relatedTarget: previous
    })

    $this.trigger(e)                        // 触发show事件

    if (e.isDefaultPrevented()) return  // 如果自定义回调里阻止了默认行为，则
不再继续处理

    var $target = $(selector)  // 要激活显示的面板，即target（或href）里的值
所对应的元素

    this.activate($this.parent('li'), $ul)  // 高亮显示当前tab
    this.activate($target, $target.parent(), function () {
                                                    // 显示对应的面板，并在回调
里触发shown事件
        $this.trigger({
            type: 'shown.bs.tab',
            relatedTarget: previous
        })
    })
}
// active样式的应用，面板的隐藏和显示，以及tab的高亮与反高亮都使用了该方法
Tab.prototype.activate = function (element, container, callback) {

    // 查找当前容器（nav里的ul或者tab-content）里所有有active样式的元素
    var $active = container.find('> .active')
    var transition = callback
        && $.support.transition
        && $active.hasClass('fade')     // 判断是使用回调还是动画

    function next() {
        $active
            .removeClass('active')      // 去除其他元素的active样式
            .find('> .dropdown-menu > .active')
                                                    // 包括li元素里面的下拉菜单
里的active样式也要去除
            .removeClass('active')

        element.addClass('active')      // 给当前被单击的元素添加active高亮
样式

        if (transition) {
            element[0].offsetWidth      // 如果支持动画，就重绘页面
            element.addClass('in')      // 并添加in样式，去除透明
        } else {
            element.removeClass('fade') // 否则删除fade
        }
```

```
        if (element.parent('.dropdown-menu')) {
                                                    // 如果单击的是下拉菜单里的
颜色，则查找父元素（即ul）
            // 将离下拉菜单最近的li.dropdown元素（一般都是其父元素）进行高亮
            element.closest('li.dropdown').addClass('active')
        }

        callback && callback()          // 最后，如果有回调，就执行回调

    }

    transition ?                                    // 如果支持动画
        $active
        .one($.support.transition.end, next)  // 就在动画结束后，执行next
        .emulateTransitionEnd(150) :
        next()                                      // 否则，直接执行next

    $active.removeClass('in')
}
```

步骤3　jQuery插件定义。是在jQuery上定义$.fn.　tab插件，和Modal插件一样，在实例化之前，需要收集option参数。源码如下：

```
var old = $.fn.tab
//  保留其他库的$.fn.tab代码（如果定义的话），以便在noConflict之后，可以继续使用该
老代码
$.fn.tab = function (option) {
    return this.each(function () {      // 根据选择器，遍历所有符合规则的元素
        var $this = $(this)
          var data = $this.data('bs.tab') // 查询当前元素上是否已经有了tab实
例
        if (!data) $this.data('bs.tab', (data = new Tab(this)))
                                                    // 如果没有，就初始化一个，
并传入this
        if (typeof option == 'string') data[option]()
                                                     // 如果option是字符，说明
传入的是方法，直接调用该方法
    })
}
$.fn.tab.Constructor = Tab       // 重设插件构造器，可以通过该属性获取插件的真
实类函数
```

步骤4　防冲突处理。源码如下：

```
// 防冲突处理
  $.fn.tab.noConflict = function () {
    $.fn.tab = old                          // 恢复以前的旧代码
    return this                              // 将$.fn.tab.noConflict()设置为
Bootstrap的tab插件
}
```

步骤5　绑定触发事件。源码如下：

```
// 绑定触发事件
```

```
// 查询所有组合条件（带data-toggle=“tab”或者data-toggle=“pill”属性）的元素
$(document).on(‘click.bs.tab.data-api’,  ’[data-toggle=“tab”],  [data-toggle=
    “pill”]’, function (e) {
    e.preventDefault()                    // 阻止默认行为
    $(this).tab(‘show’)                          // 在单击事件上触发tab组件的show方法
})
```

如果要使用自己风格的data-toggle的话，就要自己写代码触发事件了。比如要使用一个自定义属性data-toggle=“switch”，那绑定代码就应该如下：

```
$(document).on(‘click.bs.tab.data-api’,                    ’[data-toggle=“switch”]’, function (e) {
    e.preventDefault()
    $(this).tab(‘show’)
})
```

注意

从下一节开始，我们将忽略步骤4（防冲突）的叙述，因为所有的插件在这一步的代码都一样，只是换个名字而已。也就是说，以后的源码分析，将只分析步骤2（类和原型方法定义）、步骤3（fn插件制作）和步骤5（事件绑定）。

5.6　提示框

源文件：tooltip.js、tooltips.less

CSS文件：bootstrap.css 5204行之后

提示框（Tooltip）也是一个比较常见的功能，一般来说是鼠标移动到特定的元素上时，显示相关的提示语，具体效果如图5-9所示。

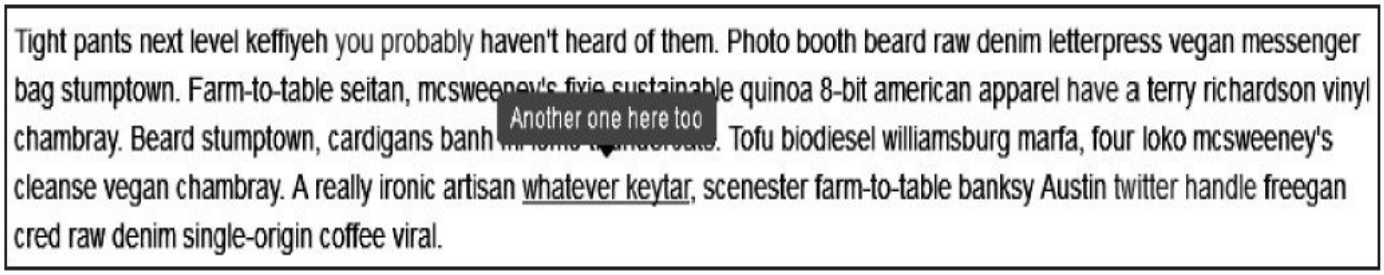


图5-9　提示框效果

上述效果是鼠标移动到href链接上时，显示对应的提示语。并且不管是href链接还是按钮，只要按照特定的规则设置，就可以在鼠标移动的时候显示提示语。

该组件是受Jason　Frame开发的jQuery.tipsy插件的启发而开发的，并且比那个插件更强大。而且此插件不依赖于图片，只是使用CSS3来实现动画过渡效果，并使用data属性设置标题。

5.6.1 声明式用法

提示框组件默认没有提示声明式的用法，作者对此的解释是，出于性能的问题才不提供的，所以如果要使用的话，需要自己使用JavaScript代码来初始化才行。尽管需要初始化，但HTML代码依然需要按照一定的规则来设置。首先，假设已经初始化了tooltip组件的功能。代码如下：

```
$('[data-toggle=tooltip]').tooltip();
```

其HTML定义非常简单，必须要定义的属性就是初始化代码里指定的data-toggle="tooltip"属性，然后再定义要显示的内容。示例如下：

```
<a  href="#"  data-toggle="tooltip"  data-original-title="这里是提示语的内容">链接</a>
<a href="#" data-toggle="tooltip" title="设置title也可以">链接</a>
```

由上述示例可以看出，提示语的内容可以通过两种形式设置，第一种是data-original-title，第二种是title。该组件将第一种的内容的优先级设置为最高，如果该属性没有值，才取title的值。

另外在声明式用法里还提供了7种自定义属性，这里先举个例子：placement，可以设置提示语相对于触发元素的显示位置。这次我们使用的是button按钮元素来触发行为。示例代码如下：

```
<button data-original-title="Tooltip on left" class="btn btn-default"
data-toggle="tooltip" data-placement="left">左侧Tooltip</button>

<button data-original-title="Tooltip on top" class="btn "
data-toggle="tooltip" data-placement="top">上方Tooltip</button>

<button data-original-title="Tooltip on bottom" class="btn "
data-toggle="tooltip" data-placement="bottom">下方Tooltip</button>

<button data-original-title="Tooltip on right" class="btn "
data-toggle="tooltip" data-placement="right">右侧Tooltip</button>
```

上述示例的运行效果如图5-10所示。

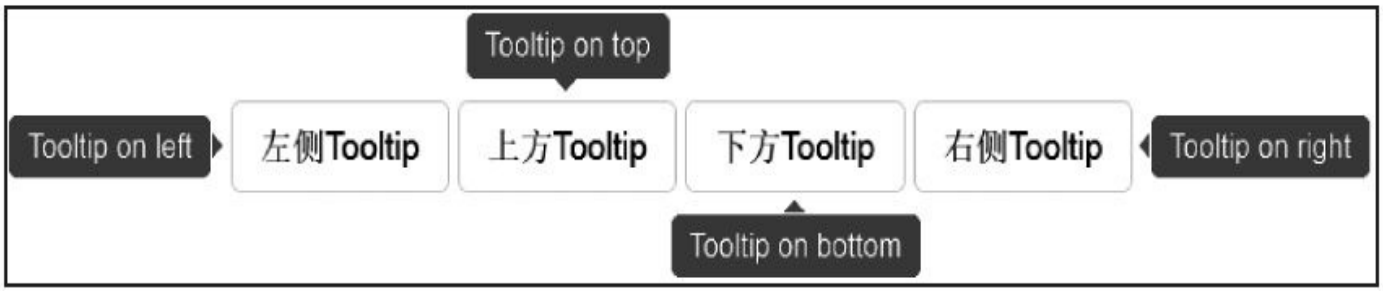


图5-10　不同方位的提示框效果

所有可用的自定义属性和解释如表5-9所示。

表 5-9　提示框组件的声明式属性

属性名称	类　型	默 认 值	描　述
data-animation	布尔值	true	在 tooltip 上应用一个 CSS fade 动画
data-html	布尔值	false	将 HTML 代码作为 tooltip 提示语，如果是 false，jQuery 将使用 text 方法将 HTML 代码转换为文本作为提示语。如果担心 XSS 攻击，请输入一般的文本
data-placement	string|function	top	tooltip 的显示位置，选项是：top | bottom | left | right | auto 如果使用 "auto"，将会再次调整，比如如果声明 "auto left",tooltip 提示语将尽量显示在左边（left），实在不行，就显示在右边
data-selector	字符串	false	如果声明了 selector，在触发该 selector 时才显示 tooltip
data-original-title	string|function	''	提示语的内容
title	string|function	''	如果没有定义 data-original-title，则取这个 title 的值
data-trigger	字符串	hover focus	如何触发 tooltip，选项是：click | hover | focus | manual。如果要传入多个触发器，使用空格隔开，比如 hover focus
data-delay	number|object	0	延迟多久才显示或关闭 tooltip（毫秒），不适用于 manual 触发器 如果传入的是数字，则说明 hide/show 都延迟这个毫秒数 传入对象的话，结构是：{ show: 500, hide: 100 }
data-container	string|false	false	将 tooltip 附加到特定的元素上，比如：container: 'body'
data-template	字符串		提示语的 HTML 模板，可以自定义指定。默认值是：

注意

❑在.btn-group　或　.input-group内的元素上使用tooltip时，需要指定

container:　‘body’选项，以避免不必要的副作用（例如，当tooltip显示之后，与其相关的页面元素可能变得更宽或去圆角）。

❑为了给disabled　或.disabled元素添加tooltip，将需要增加tooltip的页面元素包裹在一个 中，然后对这个 元素应用tooltip。

5.6.2 JavaScript用法

tooltip组件的JavaScript用法有很多种，除了通用方式以外，其他都是直接调用相关的方法（后面源码里会提到），其初始化代码就像前面所说的，最简单的一种如下：

```
$('[data-toggle=tooltip]').tooltip();
```

当然，也可以单独指定一个元素，在该元素上调用Tooltip组件，并且还可以提供各种JavaScript形式的自定义参数，而无须使用以data-开头的元素自定义属性。用法如下：

```
// 通用方式，options可以定义其他属性
$('#element').tooltip(options);
// 例子：指定提示语内容和显示位置
$('#element').tooltip({
    title: "提示语内容",
    placement: 'top'
});
```

如果想改变提示语显示的模板风格，也可以自己再指定一个template属性。

另外，如果要操作tooltip的各种其他效果，比如：显示、关闭、反转等，则可以使用如下代码：

```
$('#element').tooltip('show');          // 显示提示语
$('#element').tooltip('hide');          // 关闭提示语
$('#element').tooltip('toggle');        // 反转提示语
$('#element').tooltip('destroy');       // 隐藏并销毁提示语
```

1.选项

在JavaScript用法下，所有可用的自定义属性和描述如表5-10所示。

表 5-10　提示框组件的 JavaScript 用法属性

属性名称	类　型	默 认 值	描　述
animation	布尔值	true	在 tooltip 上应用一个 CSS fade 动画
html	布尔值	false	将 HTML 代码作为 tooltip 提示语，如果是 false，jQuery 将使用 text 方法将 HTML 代码转换为文本作为提示语。如果担心 XSS 攻击，请输入一般的文本

（续）

属性名称	类　型	默 认 值	描　述
placement	string|function	top	tooltip 的显示位置，选项是：top | bottom | left | right | auto 如果使用 "auto"，将会再次调整，比如如果声明 "auto left"，tooltip 提示语将尽量显示在左边（left），实在不行，就显示在右边
selector	字符串	false	如果声明了 selector，在触发该 selector 时才显示 tooltip
original-title	string|function	''	提示语的内容
title	string|function	''	如果没有定义 data-original-title，则取这个 title 的值
trigger	字符串	hover focus	如何触发 tooltip，选项是：click | hover | focus | manual。如果要传入多个触发器，使用空格隔开，比如 hover focus
delay	number|object	0	延迟多久才显示或关闭 tooltip（毫秒），不适用于 manual 触发器 如果传入的是数字，则说明 hide/show 都延迟这个毫秒数 传入对象的话，结构是：{ show: 500, hide: 100 }
container	string|false	false	将 tooltip 附加到特定的元素上，比如：container: 'body'
template	字符串		提示语的 HTML 模板，可以自定义指定。默认值是：

调用tooltip的时候，以上任何属性都可以在options里设置，以便制作出各种各样的效果。

2.事件

同Modal弹窗一样，下拉菜单也支持4种类型的事件订阅，分别对应下拉菜单的弹出前、弹出后、关闭前，关闭后，解释如表5-11所示。

表 5-11　提示框组件的事件类型

属性名称	描　　述
show.bs.tooltip	在 show 方法调用时立即触发（尚未显示之前）
shown.bs.tooltip	在提示框完全显示给用户之后（并且等 CSS 动画完成之后）触发
hide.bs.tooltip	在 hide 方法调用时（但还未关闭隐藏）立即触发
hidden.bs.tooltip	在提示框完全隐藏之后（并且等 CSS 动画完成之后）触发

调用方式也很简单，和普通的jQuery代码并无二样。

```
$('#myTooltip').on('hidden.bs.tooltip', function () {
    // 处理代码…
})
```

其实，tooltip组件还有其他一些方法可以调用，但是一般使用的机会不多，就不详细解释了。具体的方法在下面的源码分析小节都会有注释的。

5.6.3 源码分析

分析源码之前来思考一下，通过前面的两种用法可以看出，tooltip组件其实在HTML方面并没有特别的要求，可以定义data-toggle=“tooltip”，也可以什么都不定义，而使用jQuery的选择器来查找元素，然后应用tooltip函数。

在源码分析之前，一定要熟悉前面的9个属性，因为源码里针对这9个属性做了大量的代码工作。下面来一一分析一下。

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不再赘述。

步骤2 插件核心代码。tooltip的原型方法比较多，主要源码如下：

```
var Tooltip = function (element, options) { // 实例化tooltip时，传入当前元素和options参数
    this.type =
    this.options =
    this.enabled =
    this.timeout =
    this.hoverState =
this.$element = null

    // 根据传入的选项进行初始化事件绑定
       //  注意：这里传入tooltip，因为popover插件也使用了tooltip的原型方法，所以init提供了一个type参数
    this.init('tooltip', element, options)
}

Tooltip.DEFAULTS = {
    animation: true,                    // 是否开启动画
    placement: 'top',                   // 显示位置，默认在上方显示
    selector: false,                    // 触发器的选择符
     template: '<div class="tooltip"><div class="tooltip-arrow"></div><div class="tooltip-
        inner"></div></div>',            // tooltip显示的内容模板
    trigger: 'hover focus',             // 设置触发tooltip的事件
title: '',                             // 标题
delay: 0,                               // 延迟
    html: false,                        // tooltip内容是否是html
    container: false // tooltip容器设置，如果有直接赋值，没有则默认false
}
// 初始化，事件绑定
// 注意：这里传入的是tooltip，因为popover插件也使用了tooltip的原型方法，所以init提供了一个type参数
//  popover插件传入的就是popover
Tooltip.prototype.init = function (type, element, options){};
// 获取默认配置
Tooltip.prototype.getDefaults = function (e){};
// 获取配置参数（将传入的参数和默认参数合并）
Tooltip.prototype.getOptions = function (options) {};
```

```
// 手动触发的时候，添加额外的options参数
Tooltip.prototype.getDelegateOptions = function (){};
// 移除元素时的处理，主要是找到元素实例，然后调用show方法
Tooltip.prototype.enter = function (obj) {};
// 离开元素时的处理，主要是找到元素实例，然后调用hide方法
Tooltip.prototype.leave = function (obj) {};
// 显示tooltip提示语
Tooltip.prototype.show = function (){};
// 再次应用更新的placement样式和位置
Tooltip.prototype.applyPlacement = function (offset, placement) {};
// 更新小箭头的位置
Tooltip.prototype.replaceArrow = function (delta, dimension, position) {
// 在模板里设置title内容，以便正式组装tooltip的内容
Tooltip.prototype.setContent = function (){};
// 关闭tooltip提示框
Tooltip.prototype.hide = function (){};
// 修复title提示，既有title，又有tooltip
Tooltip.prototype.fixTitle = function (){};
// 判断触发元素是拥有内容（及title值），其调用了getTitle
Tooltip.prototype.hasContent = function (){};
// 获取tooltip的当前位置
Tooltip.prototype.getPosition = function (){};
// 计算更新placement样式后的位置
Tooltip.prototype.getCalculatedOffset = function (placement, pos, actual
    actualHeight) {};
// 获取触发元素的title内容
Tooltip.prototype.getTitle = function (){};
// 获取tooltip的模板内容
Tooltip.prototype.tip = function (){};
// 查找小箭头元素
Tooltip.prototype.arrow = function (){};
// 验证元素释放合法
Tooltip.prototype.validate = function (){};
// 设置tooltip可用
Tooltip.prototype.enable = function (){};
// 设置tooltip不可用
Tooltip.prototype.disable = function (){};
// 反转可用状态
Tooltip.prototype.toggleEnabled = function (){};
// 反转tooltip显示/隐藏状态
Tooltip.prototype.toggle = function (e) {};
// 去除tooltip插件的绑定
Tooltip.prototype.destroy = function (){};
```

上述源码，除了定义tooltip插件类函数和默认参数以外，其他的原型方法非常多，有的是用于显示、关闭、反转提示语的，还有对提示语进行位置计算、调整的，甚至还有调整提示语里的小箭头的以及获取和设置相关的内容的。所以读者在阅读这几页源码的时候，一定要静下心来，否则很容易混乱的。细节代码和注释如下：

```
// 初始化，事件绑定
// 注意：这里传入的是tooltip，因为popover插件也使用了tooltip的原型方法，所以init
提供了
// 一个type参数
```

```
//  popover插件传入的就是popover
Tooltip.prototype.init = function (type, element, options) {
    this.enabled = true                          // 可用状态
    this.type = type                     // 默认是tooltip
    this.$element = $(element)           // 当前元素
    this.options = this.getOptions(options)     // 获取配置参数（将传入的参
数和默认参数合并）

    var triggers = this.options.trigger.split(‘ ’)
                                                  // 将多个触发事件转换成数
字，例如hover click

    for (var i = triggers.length; i—;) {
        var trigger = triggers[i]       // 针对每个事件，单独处理如下的内容

        if (trigger == ‘click’) {       // 如果是click事件
             // 则在click.tooltip上绑定toggle回调，即每单击一次，就会反转显示状
态
            this.$element.on(‘click.’ + this.type, this.options.selector
                (this.toggle, this))
        } else if (trigger != ‘manual’) {       // 如果不是手动触发
            // 如果是hover事件，则进入事件是mouseenter，否则是focusin
            var eventIn = trigger == ‘hover’ ? ‘mouseenter’ : ‘focusin’
            // 如果是hover事件，则移出事件是mouseleave，否则是focusout
            var eventOut = trigger == ‘hover’ ? ‘mouseleave’ : ‘focusout

            // 给进入事件绑定enter回调，如mouseenter.tooltip
            this.$element.on(eventIn + ’.’ + this.type, this.options.sel
                (this.enter, this))
            // 移出事件绑定leave回调，如focusout.tooltip
            this.$element.on(eventOut + ’.’ + this.type, this.options.se
                $.proxy(this.leave, this))
        }
    }

    // 如果指定了内部选择符
     // 则合并原来的options到一个新的_options对象上，并添加trigger和selector两
个选项
    this.options.selector ?
        (this._options = $.extend({}, this.options, { trigger: ‘manual’,
            selector: ” })) :
        this.fixTitle()                          // 否则，修复title提示
}
// 获取默认配置
Tooltip.prototype.getDefaults = function () {
    return Tooltip.DEFAULTS
}
// 获取配置参数（将传入的参数和默认参数合并）
Tooltip.prototype.getOptions = function (options) {
    // 将传入的参数和默认参数合并
    options = $.extend({}, this.getDefaults(), this.$element.data(), opt
    // 如果传入的delay是数字，则表示show和hide的延迟时间都是该数字
    if (options.delay && typeof options.delay == ‘number’) {
        options.delay = {
            show: options.delay,
            hide: options.delay
```

```
        }
    }
    return options
}
// 手动触发的时候，添加额外的options参数
Tooltip.prototype.getDelegateOptions = function () {
    var options = {}
    var defaults = this.getDefaults()           // 默认配置参数

    this._options && $.each(this._options, function (key, value) {
                                                                // 如果_options
可用
        // 如果默认参数里指定key的值和_options指定key的值不一样，则将_options里
的值赋
        // 值给options，也就是用新值
        if (defaults[key] != value) options[key] = value
    })
    return options                                  // 返回更新后的选项参数
}
// 移除元素时的处理，主要是找到元素实例，然后调用show方法
Tooltip.prototype.enter = function (obj) {
    // 如果obj是当前tooltip的实例就赋值给self
                        //      否则就取该元素上的实例（data-bs.tooltip属性），
即：// $(‘#id’).tooltip(options).data
    // (‘bs.tooltip’)
    var self = obj instanceof this.constructor ?
                                obj    :    $(obj.currentTarget)[this.type]
(this.getDelegateOptions()).data(‘bs.’
            + this.type)

    clearTimeout(self.timeout)                  // 去除timeout
    self.hoverState = ‘in’                      // 设置移入状态是in

    // 如果delay没提供，后者delay.show没提供，直接返回self.show
    if (!self.options.delay || !self.options.delay.show) return self.sho

    self.timeout = setTimeout(function () {     // 重新设置timeout
        if (self.hoverState == ‘in’) self.show()        // 如果移入状态是
in，调用show方法
    }, self.options.delay.show)
}
// 离开元素时的处理，主要是找到元素实例，然后调用hide方法
Tooltip.prototype.leave = function (obj) {
    // 如果obj是当前tooltip的实例就赋值给self
                        //      否则就取该元素上的实例（data-bs.tooltip属性），
即：// $(‘#id’).tooltip(options).data
    // (‘bs.tooltip’)
    var self = obj instanceof this.constructor ?
                                obj    :    $(obj.currentTarget)[this.type]
(this.getDelegateOptions()).data
            (‘bs.’ + this.type)

    clearTimeout(self.timeout)                  // 去除timeout
    self.hoverState = ‘out’                     // 设置移入状态是out

    // 如果delay没提供，后者delay.hide没提供，直接返回self.hide
```

```
    if (!self.options.delay || !self.options.delay.hide) return self.hic

    self.timeout = setTimeout(function () {     // 重新设置timeout
        if (self.hoverState == 'out') self.hide()       // 如果移入状态是
out，调用hide方法
    }, self.options.delay.hide)
}
// 显示tooltip提示语
Tooltip.prototype.show = function () {
    var e = $.Event('show.bs.' + this.type)     // 设置tooltip完全显示之前
触发的show事件

    if (this.hasContent() && this.enabled) {    // 如果有要显示的内容，并且
tooltip可用
        this.$element.trigger(e)                // 首先触发show事件

        if (e.isDefaultPrevented()) return      // 如果show回调里阻止了继
续操作，则返回
        var that = this;
        var $tip = this.tip()                   // 获取tooltip的模板内容

        this.setContent()  // 在模板里设置title内容，以便正式组装tooltip的内
容
        if (this.options.animation) $tip.addClass('fade')
                                                            // 如果设置了动
画，则在tooltip上添加fade样式

          // 如果显示位置参数设置的是function，则直接调用该函数，否则直接利用该位
置，比如
        // left、right
        var placement = typeof this.options.placement == 'function' ?
            this.options.placement.call(this, $tip[0], this.$element[0])
            this.options.placement

        // 如果位置设置里有auto关键字，先删除auto关键字，比如只剩left；如果什么都
没剩余，默认为top
        var autoToken = /\s?auto?\s?/i
        var autoPlace = autoToken.test(placement)
        if (autoPlace) placement = placement.replace(autoToken, '') || 't

        $tip
            .detach()
            .css({ top: 0, left: 0, display: 'block' }) // 默认左上角块级
显示（相对定位）
            .addClass(placement) // 设置显示方向是left、right、top、bottom
中的一种

        // 如果指定了container容器，则将tooltip附加到该容器，否则附加到当前元素
的最末尾（内部）
        this.options.container ? $tip.appendTo(this.options.container) :
            insertAfter(this.$element)

        var pos = this.getPosition()            // 获取tooltip的当前位置
        var actualWidth = $tip[0].offsetWidth   // 获取实际宽度
        var actualHeight = $tip[0].offsetHeight         // 获取实际高度
```

```
        if (autoPlace) {                            // 如果定义了auto方向
            var $parent = this.$element.parent()        // 获取触发元素的
父元素
            var orgPlacement = placement        // 将原来的方向先临时存到一
个临时变量里
            // 获取当前页面的距离顶端的top高度
            var docScroll = document.documentElement.scrollTop || docume
            // 获取父元素的整个宽度
            var parentWidth = this.options.container == 'body' ? window.
                $parent.outerWidth()
            // 获取父元素的整个高度
            var parentHeight = this.options.container == 'body' ? windo
                $parent.outerHeight()
            //  获取父元素的left值
            var parentLeft = this.options.container == 'body' ? 0 : $par

            // 如果位置是bottom，但是超出了父元素的高度，则使用top
            // 如果位置是top，但是超出了浏览器顶部，则使用bottom
            // 如果位置是right，但是超出了浏览器右边栏，则使用left
            // 如果位置是left，但是超出了浏览器左边栏，则使用right
            placement = placement == 'bottom' && pos.top + pos.height +
                actualHeight - docScroll > parentHeight ? 'top' :
                        placement == 'top' && pos.top - docScroll -
                            actualHeight < 0 ? 'bottom' :
                        placement == 'right' && pos.right + actualWidth
                            parentWidth ? 'left' :
                        placement == 'left' && pos.left - actualWidth <
                            parentLeft ? 'right' :
                        placement               // 否则，还使用原来的位置

            $tip
                .removeClass(orgPlacement)      // 删除原来设置的位置样式
                .addClass(placement)            // 使用新计算的位置样式
        }

        // 计算更新placement样式后的位置
        var calculatedOffset = this.getCalculatedOffset(placement, pos,
            actualWidth, actualHeight)
        this.applyPlacement(calculatedOffset, placement)
                                                        // 再次应用更新的
placement样式和位置
        this.hoverState = null

        var complete = function () {
            that.$element.trigger('shown.bs.' + that.type)
        }

        $.support.transition && this.$tip.hasClass('fade') ?
            $tip
            .one($.support.transition.end, complete)
            .emulateTransitionEnd(150) :
            complete()
    }
}
// 再次应用更新的placement样式和位置
Tooltip.prototype.applyPlacement = function (offset, placement) {
```

```
    var replace
    var $tip = this.tip()
    var width = $tip[0].offsetWidth
    var height = $tip[0].offsetHeight

    // 手动获取margin值，因为使用getBoundingClientRect在不同的浏览器不准
    var marginTop = parseInt($tip.css('margin-top'), 10)
    var marginLeft = parseInt($tip.css('margin-left'), 10)

    // 在IE8、IE9下必须要判断值为NaN的情况
    if (isNaN(marginTop)) marginTop = 0
if (isNaN(marginLeft)) marginLeft = 0

    // 将margin值更新到offset
    offset.top = offset.top + marginTop
    offset.left = offset.left + marginLeft

          //   设置新的offset，但由于$.fn.offset不能彻底设置像素值，所以这里使用
setOffset方法直接设置
    $.offset.setOffset($tip[0], $.extend({
        using: function (props) {
            $tip.css({
                top: Math.round(props.top),
                left: Math.round(props.left)
            })
        }
    }, offset), 0)

    $tip.addClass('in')                                    // 并添加in样式，
用于显示

    // 检查tooltip在重新应用后，是否又自己重绘并改变大小了
    var actualWidth = $tip[0].offsetWidth
    var actualHeight = $tip[0].offsetHeight

    // 如果选择的是top，并且高度改变了，则重新更新offset
    if (placement == 'top' && actualHeight != height) {
        replace = true
        offset.top = offset.top + height - actualHeight
    }
     if (/bottom|top/.test(placement)) {          // 如果使用了bottom或top
位置
        var delta = 0
        if (offset.left < 0) {                        // 如果超出了浏览器的
左边框，则设置为最小值0
            delta = offset.left * -2
            offset.left = 0

            $tip.offset(offset)                  // 重新更新offset

                actualWidth = $tip[0].offsetWidth    // 再次获取重绘以后的
offset
            actualHeight = $tip[0].offsetHeight
        }

        this.replaceArrow(delta - width + actualWidth, actualWidth, 'lef
```

```
                                                        // 更新小箭头的位
置
    } else {
        this.replaceArrow(actualHeight - height, actualHeight, 'top')
                                                        // 更新小箭头的位
置
    }

    if (replace) $tip.offset(offset)    // 重新应用offset
}
// 更新小箭头的位置
Tooltip.prototype.replaceArrow = function (delta, dimension, position) {
    this.arrow().css(position, delta ? (50 * (1 - delta / dimension) + '
}
// 在模板里设置title内容，以便正式组装tooltip的内容
Tooltip.prototype.setContent = function () {
    var $tip = this.tip()
    var title = this.getTitle()

    // 如果支持HTML，就设置HTML，否则设置text
      $tip.find('.tooltip-inner')[this.options.html ? 'html' : 'text']
(title)
    $tip.removeClass('fade in top bottom left right')
                                                    // 如果有多余的样式，全部删
除，后面会根据状态再添加
}
// 关闭tooltip提示框
Tooltip.prototype.hide = function () {
    var that = this
    var $tip = this.tip()
    var e = $.Event('hide.bs.' + this.type)     // 设置tooltip在开始关闭时
触发的hide事件

    function complete() {
        if (that.hoverState != 'in') $tip.detach()
                                                    // 如果当前元素没有in样式，
直接移除tooltip
        that.$element.trigger('hidden.bs.' + that.type)
    }

    this.$element.trigger(e)            // 触发hide事件
     if (e.isDefaultPrevented()) return  // 如果hide回调里阻止了继续操作，则
返回

    $tip.removeClass('in')              // 删除in样式

    $.support.transition && this.$tip.hasClass('fade') ?
                                                        // 如果设置了动画，则在
tooltip上添加fade样式
        $tip
        .one($.support.transition.end, complete)
                                                        // 设置动画以后，调用
complete函数关闭tooltip
        .emulateTransitionEnd(150) :    // 延迟150毫秒才开始动画
        complete()                      // 否则直接关闭
```

```
    this.hoverState = null
    return this
}
// 修复title提示，既有title，又有tooltip
Tooltip.prototype.fixTitle = function () {
var $e = this.$element
// 触发元素有title值，或者data-original-title属性不是字符串时
    // 将title（或者空）赋值给data-original-title属性，然后再将title属性设置为空
          if  ($e.attr('title')  ||  typeof  ($e.attr('data-original-title')) != 'string') {
                                                $e.attr('data-original-title', $e.attr('title') || '').attr('title', '')
    }
}
// 判断触发元素是拥有内容（及title值），其调用了getTitle
Tooltip.prototype.hasContent = function () {
    return this.getTitle()
}
// 获取tooltip的当前位置
Tooltip.prototype.getPosition = function () {
    var el = this.$element[0]
    return $.extend({}, (typeof el.getBoundingClientRect == 'function')
        getBoundingClientRect() : {
        // 如果系统支持getBoundingClientRect函数，直接用该函数获取位置
                width:  el.offsetWidth,    //  否则使用el.offsetWidth和el.offsetHeight来获取位置
        height: el.offsetHeight
    }, this.$element.offset())  // 如果都获取不了，则返回默认的offset
}
// 计算更新placement样式后的位置
Tooltip.prototype.getCalculatedOffset = function (placement, pos, actual
    actualHeight) {
    return placement == 'bottom' ? { top: pos.top + pos.height, left: po
        pos.width / 2 - actualWidth / 2 } :
            placement == 'top' ? { top: pos.top - actualHeight, left: po
                pos.width / 2 - actualWidth / 2 } :
            placement == 'left' ? { top: pos.top + pos.height / 2 -
                actualHeight / 2, left: pos.left - actualWidth } :
        /* placement == 'right' */ { top: pos.top + pos.height / 2 -
            actualHeight / 2, left: pos.left + pos.width }
}
// 获取触发元素的title内容
Tooltip.prototype.getTitle = function () {
    var title
    var $e = this.$element
    var o = this.options

    // 第一优先级：data-original-title
    // 第二优先级：如果options.title传入的是function，将其调用结果作为title；如果不是直
    // 接返回options.title
    title = $e.attr('data-original-title')
        || (typeof o.title == 'function' ? o.title.call($e[0]) : o.title

    return title
```

```
}
// 获取tooltip的模板内容
Tooltip.prototype.tip = function () {
    return this.$tip = this.$tip || $(this.options.template)
}
// 查找小箭头元素
Tooltip.prototype.arrow = function () {
      return  this.$arrow  =  this.$arrow  ||  this.tip().find('.tooltip-
arrow')
}
// 验证元素释放合法
Tooltip.prototype.validate = function () {
    if (!this.$element[0].parentNode) {          // 如果元素没有父节点，比如
document
        this.hide()                                  // 隐藏该元素
        this.$element = null
        this.options = null
    }
}
// 设置tooltip可用
Tooltip.prototype.enable = function () {    this.enabled = true}
// 设置tooltip不可用
Tooltip.prototype.disable = function () {    this.enabled = false}
// 反转可用状态
Tooltip.prototype.toggleEnabled = function () {    this.enabled = !this.
// 反转tooltip显示/隐藏状态
Tooltip.prototype.toggle = function (e) {
    // 如果调用对象时触发元素，就查找该触发元素上的实例，否则就是this
                var    self    =    e    ?    $(e.currentTarget)[this.type]
(this.getDelegateOptions()).
        data('bs.' + this.type) : this

      //  如果tip上有in样式，就触发移出操作（leave）关闭tooltip；否则触发移入操作
（enter）开启tooltip
    self.tip().hasClass('in') ? self.leave(self) : self.enter(self)
}
// 去除tooltip组件的绑定
Tooltip.prototype.destroy = function () {
    clearTimeout(this.timeout)
            //   去除.tooltip命名空间中所有的事件绑定，并移除tooltip实例（data-
bs.tooltip）
    this.hide().$element.off('.' + this.type).removeData('bs.' + this.ty
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.tooltip插件的代码，和前面几个插件几乎一样：遍历元素、实例化或触发动作。源码如下：

```
var old = $.fn.tooltip
//   保留其他库的$.fn.tooltip代码（如果定义的话），以便在noConflict之后，可以继续
使用该老代码
$.fn.tooltip = function (option) {
    return this.each(function () {              // 根据选择器，遍历所有符合
规则的元素
        var $this = $(this)                      // this赋值一个变量，防止
作用域改变
```

```
        var data = $this.data('bs.tooltip')      // 查询当前元素上是否已经有了tooltip实例
        // 如果option是对象，说明是参数集合，在new tooltip的时候传入
        var options = typeof option == 'object' && option

        if (!data && option == 'destroy') return
        // 如果没有实例，就初始化一个，并传入this
        if (!data) $this.data('bs.tooltip', (data = new Tooltip(this, op

        // 如果option是字符串，说明传入的是一个方法，直接调用该方法
        if (typeof option == 'string') data[option]()
    })
}
$.fn.tooltip.Constructor = Tooltip // 重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。在前面我们已经说了，tooltip插件默认没有提供触发声明式的代码，所以也就没有此步代码了。

5.7　弹出框

源文件：popover.js、popovers.less

CSS文件：bootstrap.css 5302行之后

在讲述弹出框插件之前，先来看看该组件的两个运行效果，如图5-11和5-12所示。

图5-11　弹出框效果1

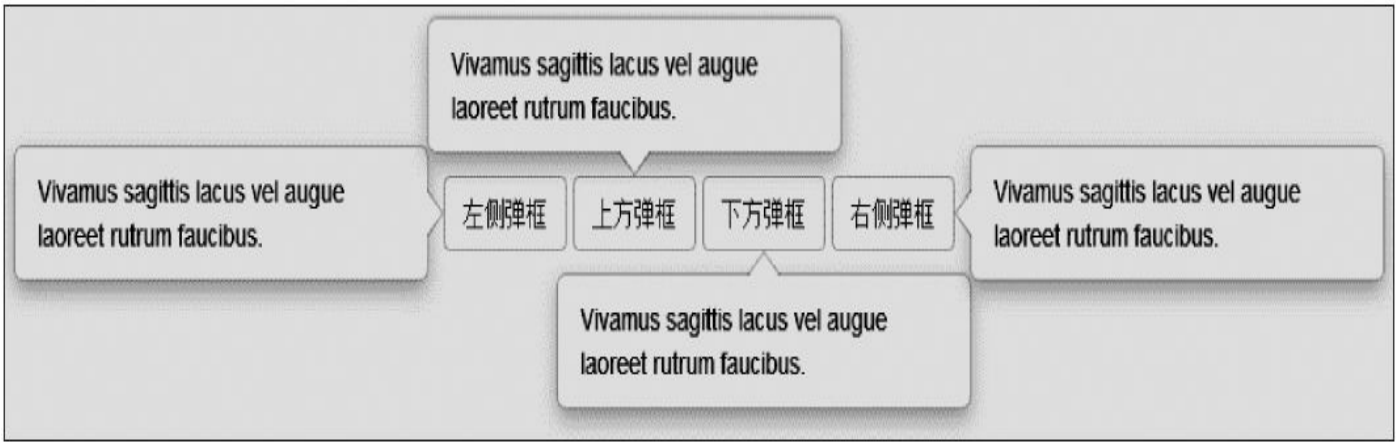


图5-12　弹出框效果2

有人可能会说，这不就是上一节的提示框（tooltip）插件嘛，就是样式不一样而已，可以通过template属性来定制。

是的，弹出框其实就是一种特殊的提示框，相比来说只是多了一个title标题而已，不带标题的效果2完全可以通过自定义tooltip来实现。那作者为何还提供这一个弹出框插件呢？我想，是因为两个用户体验不太一样吧，一个是提示，一个是弹出，弹出就意味着可以操作里面的内容，相对比较正式。

通过分析源码，笔者发现，两个插件只有以下两个不同点：

❑提示框的默认触发事件是hover和focus，而弹出框的默认触发事件是click。

❑提示框只有一个内容（即title），而弹出框不仅可以设置标题title，

还可以设置内容content（也就是说模板不一样）。

两个插件的显示模板分别如下：

```
// 提示框效果
<div class=“tooltip”>
    <div class=“tooltip-arrow”></div>
    <div class=“tooltip-inner”></div>
</div>
// 弹出框效果
<div class=“popover”>
    <div class=“arrow”></div>
    <h3 class=“popover-title”></h3>
    <div class=“popover-content”></div>
</div>
```

至此，你应该可以看出点名堂了吧。其实popover插件就是通过继承tooltip来实现的，源码分析小节会详细讲述。

5.7.1 声明式用法

弹出框插件的用法和提示框组件几乎一模一样，唯一不一样的就是插件名称以及多余的content字段。而关于自动初始化，也和提示框插件一样，默认没有提供，需要自己使用Javascript代码来初始化。代码如下：

```
$('[data-toggle=popover]').popover();
```

声明式用法的HTML定义也非常简单，和提示框（tooltip）相比，只多了一个data-content自定义属性。示例如下：

```
<a data-toggle="popover"
    data-original-title="A Title" title=""
                                                    data-
content="And here's some amazing content. It's very engaging. right?"
    class="btn btn-lg btn-danger" href="#">单击反转弹出框
</a>
```

弹出框插件所支持的元素，也是和提示框一样，支持任何元素，只要设置data-toggle="popover"即可。对这方面的内容就不多说了，如果不明白，可以先去阅读提示框的内容。

尽管弹出框的自定义属性和提示框的几乎都一样（只多了一个data-content），可以将所有可用的自定义属性和解释都列出来，具体如表5-12所示。

表 5-12 弹出框组件的声明式选项

属性名称	类 型	默 认 值	描 述
data-animation	布尔值	true	在 popover 上应用一个 CSS fade 动画
data-html	布尔值	false	将 HTML 代码作为 popover 提示语，如果是 false，jQuery 将使用 text 方法将 HTML 代码转换为文本作为提示语。如果担心 XSS 攻击，请输入一般的文本

（续）

属性名称	类　型	默 认 值	描　述
data-placement	string|function	top	Popover 的显示位置，选项是：top | bottom | left | right | auto 如果使用 "auto"，将会再次调整，比如如果声明 "auto left"，popover 提示语将尽量显示在左边（left），实在不行，就显示在右边
data-selector	字符串	false	如果声明了 selector，在触发该 selector 时才显示 popover
data-original-title	string|function	''	提示语的内容
title	string|function	''	如果没有定义 data-original-title，则取这个 title 的值
data-trigger	字符串	hover focus	如何触发 popover，选项是：click | hover | focus | manual。如果要传入多个触发器，使用空格隔开，比如 hover focus
data-delay	number|object	0	延迟多久才显示或关闭 popover（毫秒），不适用于 manual 触发器 如果传入的是数字，则说明 hide/show 都延迟这个毫秒数 传入对象的话，结构是：{ show: 500, hide: 100 }
data-container	string|false	false	将 popover 附加到特定的元素上，比如：container: 'body'
data-template	字符串		提示语的 HTML 模板，可以自定义指定。默认值是： <h3 class="popover-title"></h3>

注意

❑在.btn-group 或 .input-group内的元素上使用popover时，需要指定container: ‘body’选项以避免不必要的副作用（例如，当popover显示之后，与其相关的页面元素可能变得更宽或去圆角）。

❑为了给disabled 或.disabled元素添加popover，将需要增加popover的页面元素包裹在一个 中，然后对这个 元素应用popover。

5.7.2 JavaScript用法

弹出框插件JavaScript的用法和提示框的也是一模一样的，把tooltip换成popover即可，这里只列出以下几个示例。

```
// 通用方式，options可以定义其他属性
$(‘#element’).popover(options);
// 例子：指定提示语的标题、内容以及显示位置
$(‘#element’). popover ({
title: ”提示语的标题”,
content：”提示语的内容”,
    placement: ‘top’
});
$(‘#element’).popover(‘show’);              // 显示提示语
$(‘#element’).popover(‘hide’);              // 关闭提示语
$(‘#element’).popover(‘toggle’);            // 反转提示语
$(‘#element’).popover (‘destroy’);          // 隐藏并销毁提示语
```

1.选项

除了content，其他选项与声明式选择里以data-开头的自定义属性一样。

表 5-13　弹出框组件的 JavaScript 用法选项

属性名称	类　型	默 认 值	描　述
animation	布尔值	true	在 popover 上应用一个 CSS fade 动画
html	布尔值	false	将 HTML 代码作为 popover 提示语，如果是 false，jQuery 将使用 text 方法将 HTML 代码转换为文本作为提示语。如果担心 XSS 攻击，请输入一般的文本
placement	string|function	top	Popover 的显示位置，选项是：top | bottom | left | right | auto 如果使用 "auto"，将会再次调整，比如如果声明 "auto left"，popover 提示语将尽量显示在左边（left），实在不行，就显示在右边
selector	字符串	false	如果声明了 selector，在触发该 selector 时才显示 popover
original-title	string|function	''	提示语的内容
title	string|function	''	如果没有定义 data-original-title，则取这个 title 的值
trigger	字符串	hover focus	如何触发 popover，选项是：click | hover | focus | manual。如果要传入多个触发器，使用空格隔开，比如 hover focus
delay	number|object	0	延迟多久才显示或关闭 popover（毫秒），不适用于 manual 触发器 如果传入的是数字，则说明 hide/show 都延迟这个毫秒数 传入对象的话，结构是：{ show: 500, hide: 100 }
container	string|false	false	将 popover 附加到特定的元素上，比如：container: 'body'
template	字符串		提示语的 HTML 模板，可以自定义指定。默认值是： <h3 class="popover-title"></h3>

调用popover的时候，以上任何属性都可以在options里设置，以便制作出各种各样的效果。

2.事件

同提示框tooltip一样，弹出框也支持4种类型的事件订阅，分别对应弹出框的弹出前、弹出后、关闭前、关闭后。解释和用法如表5-14所示。

表 5-14　弹出框组件的事件类型

属性名称	描　述
show.bs.popover	在 show 方法调用时立即触发（尚未显示之前）
shown.bs.popover	在提示框完全显示给用户之后（并且等 CSS 动画完成之后）触发
hide.bs.popover	在 hide 方法调用时（但还未关闭隐藏）立即触发
hidden.bs.popover	在提示框完全隐藏之后（并且等 CSS 动画完成之后）触发

调用方式就不多说了。

5.7.3　源码分析

我们前面说了，弹出框和提示框几乎一模一样，而且弹出框是从提示框继承而来的，所以继承以后唯一要处理的不同的内容，就是上面说的两个不同点：

❑将默认事件设置为click。

❑模板设置步骤要重新写，因为设置了两个内容。

步骤1　立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2　插件核心代码。主要是popover核心类函数的定义、默认参数设置、继承tooltip的原型方法、重载要处理不同逻辑的个别方法。核心摘要代码如下：

```
var Popover = function (element, options) {
    this.init('popover', element, options)      // 调用了从tooltip继承过来的原型方法
                                                 // init，并传入了popover类型
}
// 如果tooltip没引用，抛错，因为其依赖于tooltip
if (!$.fn.tooltip) throw new Error('Popover requires tooltip.js')

// 除了继承tooltip的默认值外，下面的默认值覆盖了tooltip的默认值
Popover.DEFAULTS = $.extend({}, $.fn.tooltip.Constructor.DEFAULTS, {
    placement: 'right',                  // 显示位置，默认在右方显示
    trigger: 'click',                    // 设置触发popover的事件
    content: '',                                 // 默认显示内容
          template:  '<div  class="popover"><div  class="arrow"></div><h3 class="popover-
title"></h3><div class="popover-content"></div></div>'  // popover显示的内容模板
})

// 继承tooltip的原型，含所有原型方法
Popover.prototype = $.extend({}, $.fn.tooltip.Constructor.prototype)

Popover.prototype.constructor = Popover // 恢复构造函数，以免使用tooltip的构造函数
// 重载tooltip里的getDefaults方法：获取默认配置
Popover.prototype.getDefaults = function () {};
// 重载tooltip里的setContent方法：设置popover要显示的内容
Popover.prototype.setContent = function (){};
// 重载tooltip里的hasContent方法：判断popover是否有要显示的内容
Popover.prototype.hasContent = function (){};
// 重载tooltip里的getContent方法：获取要显示的content内容
Popover.prototype.getContent = function (){};
// 重载tooltip里的arrow方法：获取显示的小箭头
Popover.prototype.arrow = function (){};
```

```
// 重载tooltip里的tip方法：获取模板内容
Popover.prototype.tip = function (){};
```

通过上述概要代码可以看出，就像我们从其他文章里看到的一样，首先继承了tooltip的原型，然后重新赋值其构造函数，最后再重载6个原型方法，分别用于处理：默认值、内容设置、小箭头设置等。详细源码如下：

```
// 重载tooltip里的getDefaults方法：获取默认配置
Popover.prototype.getDefaults = function () {
    return Popover.DEFAULTS
}
// 重载tooltip里的setContent方法：设置popover要显示的内容
Popover.prototype.setContent = function () {
    var $tip = this.tip()                // 获取模板内容
    var title = this.getTitle()          // 获取title内容
    var content = this.getContent()      // 获取要显示的内容

    // 给模板里的popover-title样式的元素添加内容，如果支持HTML，就设置HTML，否则设置text
      $tip.find('.popover-title')[this.options.html ? 'html' : 'text'](title)
    $tip.find('.popover-content')[       // 对于HTML对象使用append，用于维护JS事件
        this.options.html ? (typeof content == 'string' ? 'html' : 'appe
    ](content)

    // 正式显示之前，如果有多余的样式，全部删除，后面会根据状态再添加
    $tip.removeClass('fade top bottom left right in')

     // 由于IE8不支持empty伪类选择器，所以需要手动判断：如果title没有值，则隐藏该元素
        if (!$tip.find('.popover-title').html()) $tip.find('.popover-title').hide()
}
// 重载tooltip里的hasContent方法：判断popover是否有要显示的内容
Popover.prototype.hasContent = function () {
    return this.getTitle() || this.getContent()  // title和content任何一个有内容,
        即认为popover有内容
}
// 重载tooltip里的getContent方法：获取要显示的content内容
Popover.prototype.getContent = function () {
    var $e = this.$element               // 所单击的触发元素
    var o = this.options                 // 传入的选项

    // 如果data-content属性有值，就使用它作为内容
    // 否则，再判断如果options里的content属性是function，就将其调用结果作为内容
    // 如果上述两者都不是，直接将options里的content属性作为内容返回
    return $e.attr('data-content')
        || (typeof o.content == 'function' ?
            o.content.call($e[0]) :
            o.content)
}
// 重载tooltip里的arrow方法：获取显示的小箭头
```

```
Popover.prototype.arrow = function () {
    return this.$arrow = this.$arrow || this.tip().find('.arrow')
}
// 重载tooltip里的tip方法：获取模板内容
Popover.prototype.tip = function () {
    if (!this.$tip) this.$tip = $(this.options.template)
            // 如果$tip不存在，则用options里的template模板
    return this.$tip
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.tpopoverab插件，没有什么特殊的地方。代码如下：

```
var old = $.fn.popover
//   保留其他库的$.fn.popover代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.popover = function (option) {
    return this.each(function () {                  // 根据选择器，遍历所有符合规则的元素
        var $this = $(this)                          // this赋值一个变量，防止作用域改变
        var data = $this.data('bs.popover')      // 查询当前元素上是否已经有了popover实例
        // 如果option是对象，说明是参数集合，在new popover的时候传入
        var options = typeof option == 'object' && option

        if (!data && option == 'destroy') return

        // 如果没有popover实例，就初始化一个，并传入this
        if (!data) $this.data('bs.popover', (data = new Popover(this, op

        // 如果option是字符串，说明传入的是一个方法，直接调用该方法
        if (typeof option == 'string') data[option]()
    });
}
$.fn.popover.Constructor = Popover // 重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。与提示框插件一样，由于性能问题，没有定义绑定触发事件的代码。

5.8　警告框插件

源文件：alert.js、alerts.less

CSS文件：bootstrap.css 4434行之后

警告框（Alert）插件，就是在警告框组件的基础上，提供了单击x号关闭警告框的功能，如图5-13所示。单击右上角的x符号，则该警告框会关闭（当然，关闭功能肯定是JavaScript实现的）。

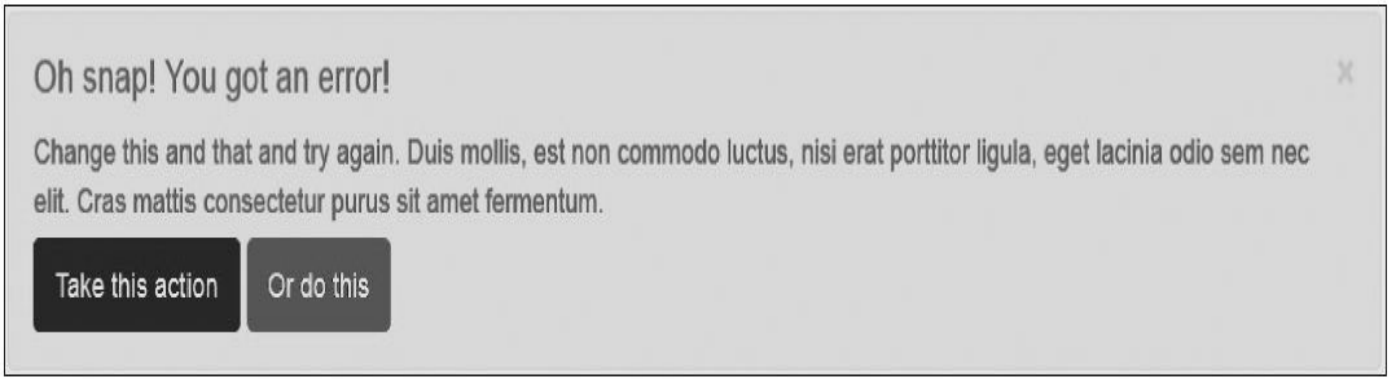


图5-13　带x号的警告框效果

5.8.1 声明式用法

警告框的一般使用方法需要注意一点，那就是要在关闭x号元素上设置自定义属性data-dismiss=“alert”。示例源码如下：

```
<div class=“alert alert-warning fade in”>
<button data-dismiss=“alert” class=“close” type=“button”>×</button>
    <h4>警告标题</h4>
    <p>Change this and that and try again. …</p>
</div>
```

默认情况下，如果不设置data-target属性，则会关闭x号按钮元素所在的父元素容器，即外部的div元素。关闭按钮不一定非要用x号，也可以是普通的按钮元素或者链接元素，只需要保证设置了data-dismiss=“alert”即可。示例代码如下：

```
<div class=“alert alert-warning fade in”>
    <h4>警告标题</h4>
    <p>Change this and that and try again. …</p>
<a href=”#” class=“btn btn-danger” data-dismiss=“alert”>关闭</a>
</div>
```

上述示例的运行效果如图5-14所示。

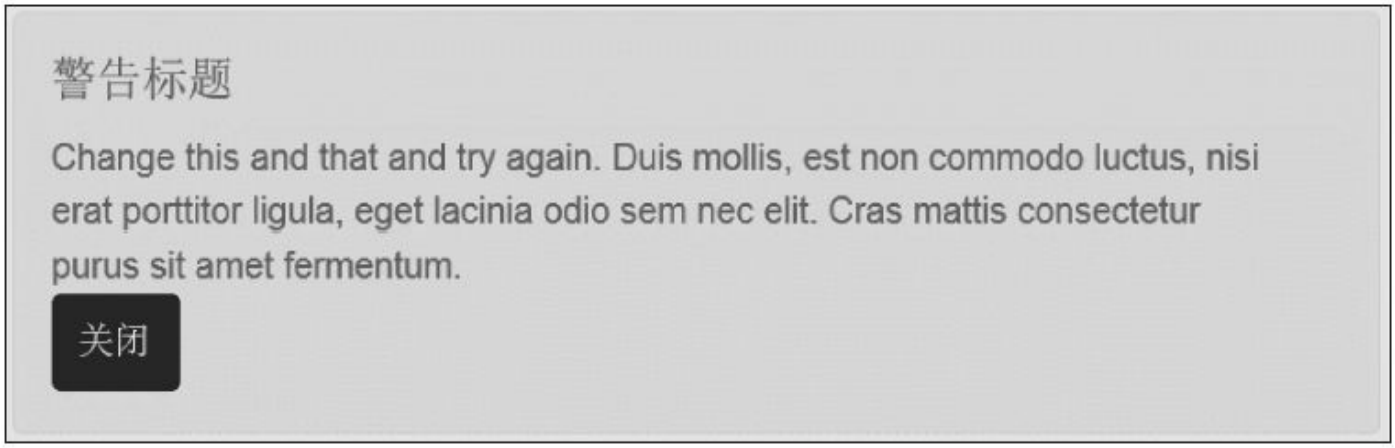


图5-14　不用x号的警告框效果

通过上面的代码，单击“关闭”按钮（其实是a链接）以后，也能关闭该警告框，因为该a链接元素同样也符合标准，即设置了data-dismiss=“alert”。

通过前面的插件可以得知，一般插件在操作一个元素时都是按照先查找data-target属性，再查找href属性，如果都没有，则操作父元素的规则。我们可以发现，即便把“关闭”按钮定义在警告框外部，同时定义好data-target属性，也能关闭该警告框，示例代码如下：

```
<a href=”#” class=“btn btn-danger” data-dismiss=“alert” data-target=”#alert2”>关闭</a>
<div id=“alert2” class=“alert alert-warning fade in”>
```

```
<h4>警告标题</h4>
<p>Change this and that and try again. …</p>
</div>
```

上述示例的运行效果如图5-15所示。

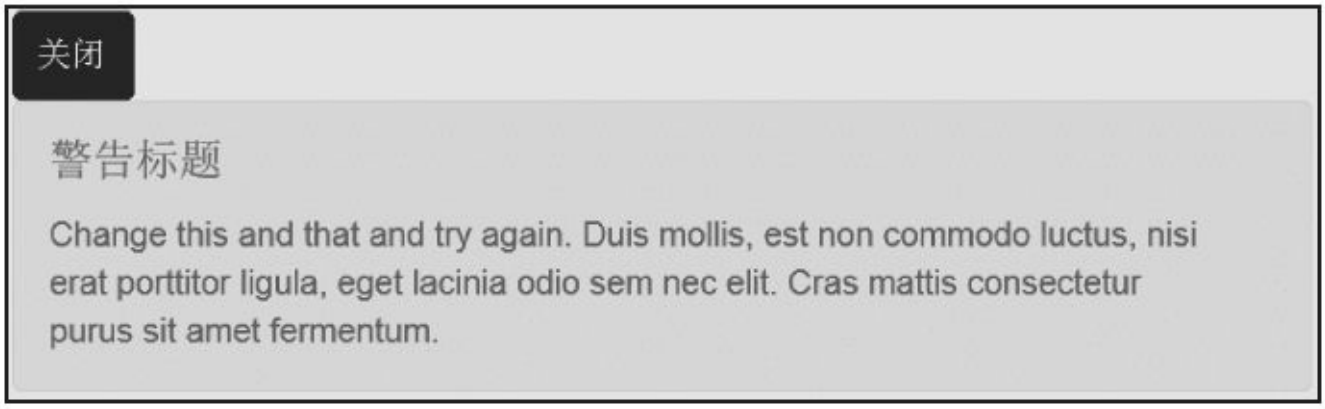


图5-15　触发外部元素关闭警告框

上面的代码和其他示例代码不同的地方是，a元素不仅设置了data-dismiss=“alert”，还设置了data-target=”#alert2”，而alert2则必须是要关闭警告的id值，也就是div设置的id必须是alert2。这种用法的效果有个弊端：按钮本身关闭不了，因为它已经不在警告框内了。

5.8.2 JavaScript用法

警告框插件，在JavaScript手工用法上提供了如下几种用法：

1）当页面容器全部禁用了警告框之后，如何恢复该功能？通过第2章的内容可以得知，禁用一个插件很容易，只需禁用该插件所在命名空间的触发事件即可。示例如下：

```
$(document).off('.alert.data-api');          // 禁用所有的警告框功能
```

这样就禁用了整个警告框alert插件的功能了。要恢复该插件有两种形式，一种是使用jQuery中fn上的alert函数进行恢复。示例如下：

```
$('[data-dismiss="alert"]').alert();          // 恢复所有的警告框功能
```

还有一种复杂一点的用法如下：

```
$(document).on('click.bs.alert.data-api', '[data-dismiss="alert"]',
$.fn.alert.Constructor.prototype.close)
// 恢复所有的警告框功能
```

上述方法是alert插件的原始绑定方法，即绑定整个DOM元素上所有的警告框事件，并使用bs.alert.data-api命名空间。

2）某些时候，可能不需要恢复所有的警告框功能，这时候需要开启警告框的那个按钮元素（或x号，或a链接）值的一个特殊的id值即可，然后针对该id选择器进行绑定。示例代码如下：

```
$('#btnClose').alert();                       // 开启id为btnClose元素上的警告
框功能
```

注意，以上所有的警告框恢复（或开启）功能，需要在"关闭"按钮元素上设置自定义属性data-dismiss="alert"，否则不会触发关闭功能。读者可能要问了，不设置自定义属性，如何触发警告框事件？可利用插件传递可选参数这一特性来实现这一功能。示例如下：

```
$("#btnClose").on("click", function () {
   $(this).alert('close');       // 单击时，关闭警告框
});
```

上述代码是在"关闭"按钮元素被单击时，直接调用alert插件的close方法进行关闭。这种方式是直接的关闭方法，我们会在下一小节源码分析的时候讲述关闭原理。

最后，alert插件还提供了两个自定义事件，以便开发人员可以在此事件上进行绑定，进行特殊代码的处理，事件内容如表5-15所示。

表 5-15　警告框组件的事件类型

属性名称	描　　述
close.bs.alert	alert 插件内部的 close 方法执行刚开始时、但还未真正关闭时，触发该事件
closed.bs.alert	当警告框被关闭之后（同时 CSS 过渡效果执行完毕），触发该事件

警告框的事件用法如下：

```
$(‘#btnClose’).on(‘close.bs.alert’, function () {
    // alert(“即将关闭”);
})

$(‘#btnClose’).on(‘closed.bs.alert’, function () {
    // alert(“关闭成功”);
})
```

5.8.3 源码分析

步骤1 立即调用的函数。第2章讲述插件设计思想的时候，曾经以警告框插件为例进行了说明，所以这里就不详细说了，直接从下一步开始。

步骤2 插件核心代码。由于警告框插件比较简单，所以这里直接给出详细的代码。代码如下：

```
// alert插件类及原型方法的定义
// 定义选择器，所有符合该自定义属性的元素都可以触发下面的事件
var dismiss = '[data-dismiss="alert"]'
var Alert = function (el) {
     // 传入元素，如果元素内部有dismiss上设置的自定义属性，则click事件会触发原型上的close方法
    $(el).on('click', dismiss, this.close)
}

Alert.prototype.close = function (e) {
     var $this = $(this)           // 被单击元素的jQuery对象，临时赋值，防止this污染
       var selector = $this.attr('data-target')   // 获取自定义属性data-target的值

     if (!selector) {              // 如果自定义属性data-target无值，则获取href属性的值
         selector = $this.attr('href')
         selector = selector && selector.replace(/.*(?=#[^\s]*$)/, '')
                                          // IE7特殊处理，比如#hom
    }
    // 执行选择器，获取jQuery对象，该对象元素通常就是要被删除的元素
    // $parent名称有点不太合适，但一般都表示包含关闭按钮和alert内容的父元素容器
    // （除非data-target或href有特定值）
    var $parent = $(selector)

     if (e) e.preventDefault()    // 防止冒泡，阻止默认行为，比如按钮或链接的其他行为

     if (!$parent.length) {       // 如果该元素对象不存在，继续判断，以进行特殊处理
         // 被单击的元素如果有alert样式的话，要删除的元素$parent就是自身
         // 如果没有alert样式，就将要删除的元素$parent设置为被单击元素的父元素
         $parent = $this.hasClass('alert') ? $this : $this.parent()
    }
    // 删除元素前执行close事件，可以通过自定义绑定来执行定义代码
    $parent.trigger(e = $.Event('close.bs.alert'))

      // 如果自定义事件里的回调函数调用过e.preventDefault()方法，则返回（避免重复执行）
    if (e.isDefaultPrevented()) return
     $parent.removeClass('in')    // 删除$parent元素上的in样式（如果没有，则不做处理）
```

```
     function removeElement() {   // 通用内部函数，在触发closed事件后，再删除$parent元素
         $parent.trigger('closed.bs.alert').remove()
     }
     // 如果支持动画，并且设置了fade样式，则在执行动画过渡效果后（从有in样式，变化到无in样式），
     // 再删除元素；否则，直接删除元素
     $.support.transition && $parent.hasClass('fade') ?
         $parent
         .one($.support.transition.end, removeElement)
         .emulateTransitionEnd(150) :
         removeElement()
}
```

上述源码比较简单，主要是查找关闭按钮，然后触发相应的单击事件，最后通过原型方法close，关闭data-target指定的元素（或父元素）。

步骤3　jQuery插件定义。具体代码如下：

```
// 在jQuery上定义alert插件，并重设插件构造器
var old = $.fn.alert
// 保留其他库的$.fn.alert代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.alert = function (option) {
                //    遍历所有符合规则的元素，例如在btnClose按钮上绑定alert插件：$("#btnClose").alert();
    return this.each(function () {
        var $this = $(this)
         var data = $this.data('bs.alert')  // 获取自定义属性data-bs.alert的值（其实是
             alert实例）

        // 如果值不存在，就创建一个alert实例，然后设置为data-bs.alert的值
        if (!data) $this.data('bs.alert', (data = new Alert(this)))

        // 如果option传递了string，则表示要执行某个方法，比如传入了close，则要执行alert
        // 实例的close方法
        // data["close"]相当于data.close();
        if (typeof option == 'string') data[option].call($this)
    })
}
$.fn.alert.Constructor = Alert // 并重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。代码如下：

```
// 为声明式的HTML绑定单击事件
// 即在整个document对象上，检测是否有自定义属性data-dismiss="alert"
// 如果有，则设置：单击的时候，关闭指定的警告框元素
$(document).on('click.bs.alert.data-
api', dismiss, Alert.prototype.close)
```

5.9 按钮

源文件：button.js、buttons.less

CSS文件：bootstrap.css 1988行之后

按钮插件提供了一组可以控制按钮多种状态的功能，比如，一个按钮有禁用状态、正在加载状态、正常状态等。举例来说，单击一个按钮请求ajax数据的时候，想让该按钮上的文本变成“正在加载”；一旦ajax返回数据，再次将该按钮上的文本变为“获取数据”。按钮插件就可以提供类似的功能，当然还有其他很多的功能，比如控制单选框、复选框的状态等。下面一一讲述一下使用方法。

5.9.1 声明式用法

按钮插件的声明式用法支持3种元素，分别是按钮（button）、复选框（input）、单选项（radio）。其实后面通过源码可以看出，对按钮判断的时候并没有限制button元素，所以应用了btn样式的a链接元素也是可以用的。我们先来看看以下例子：

```
<button type="button" class="btn btn-primary" data-toggle="button">反转状态</button>
```

只需要一个data-toggle="button"属性，就可以在单击的时候将按钮状态进行反转，其原理是在该元素上应用（或取消）.active样式。

而对于input元素来说，还有点不太一样，data-toggle="button"属性变成 data-toggle="buttons"属性。另外，除了反转.active样式以外，还反转了.disabled样式和disabled属性，即如果当前元素是活动的，则添加一个.disabled样式以及disabled=disabled属性，反转则去除disabled相关的内容。读者可以查看下面的两个示例，并查看单击以后的HTML代码。

下面的示例是复选框例子，即可以同时反转多个按钮。示例如下：

```
<div class="btn-group" data-toggle="buttons">
    <label class="btn btn-primary"><input type="checkbox">选项1</label>
    <label class="btn btn-primary"><input type="checkbox">选项2</label>
    <label class="btn btn-primary"><input type="checkbox">选项3</label>
</div>
```

而单选框的话，顾名思义，反转的时候只能反转一个，同组radio按钮的状态同时只能选择一个。示例如下：

```
<div class="btn-group" data-toggle="buttons">
                              <label       class="btn       btn-primary">
<input type="radio" name="options" id=
        "option1">男</label>
                              <label       class="btn       btn-primary">
<input type="radio" name="options" id=
        "option2">女</label>
                              <label       class="btn       btn-primary">
<input type="radio" name="options" id=
        "option3">未知</label>
</div>
```

上述3种元素的状态反转，是通过两种属性的设置来实现的，即data-toggle="button"和data-toggle="buttons"。使用这种效果的时候不需要使用JavaScript代码，因为默认Bootstrap就已经为用户初始化好了。

但是还有另外一种情况不需要设置这种data-toggle属性的额外功能，这也是笔者最喜欢的一种功能，那就是可以设置自定义状态。举例如下：

```
<button type="button" id="btn1" class="btn btn-default" data-loading-text="
        正在加载...">获取内容</button>
// 触发代码
$(function () {
    $('#btn1').click(function () {
        var btn = $(this)
        btn.button('loading')
    });
});
// 或者，可以直接调用button插件，不需要click事件
$('#btn1').button('loading');
```

运行上述代码，可以看到，如果在该按钮的click事件上触发button组件，并传入loading参数，则自定义属性data-loading-text的值就会替换成按钮的文本内容（也就是“获取内容”更改为“正在加载...”）。而同样，如果传入abc参数，则该插件也会查询有没有data-abc-text这个自定义属性，如果有，也会将它的值设置成按钮的文本。示例如下：

```
<button type="button" id="btn2" class="btn btn-default" data-complete-text="已完
        成!">完成</button>
$(function () {
$('#btn2').click(function () {
    var btn = $(this)
        btn.button('complete')
    })
});
// 或者，可以直接调用button插件，不需要click事件
$('#btn2').button('complete');
```

因为这个参数可以随意定义，所以Bootstrap并没有提供任何初始化的操作，大家在用的时候，自行使用JavaScript代码调用即可。但是要注意，一旦修改了自定义状态，记得也要设定一个机制来恢复原来的状态（使用reset参数）。例如上述示例，要将按钮在3秒以后恢复状态（或者也可以在ajax的成功回调里恢复状态）。使用如下代码即可：

```
$(function () {
    $('#btn1').click(function () {
        var btn = $(this);
        btn.button('loading');
        setTimeout(function () {
            btn.button('reset');        // 重置按钮状态，文本也会被恢复原状
        }, 3000);
    })
});
```

注意

在Firefox浏览器上，如果一个按钮被禁用了（并且没有恢复），那在刷新页面以后，禁用状态不会改变。所以为了解决这个问题，只需要在按钮元素（或者按钮所在的表单元素）上，添加一个自定义属性

autocomplete=“off”即可。

5.9.2 JavaScript用法

按钮插件的用法从上面的初始化代码就可以看出来了，就是调用button函数，然后传入参数即可。但是有两个参数是固定的，即toggle和reset，其他都可以随意定义。示例如下：

```
$(‘#id’).button(‘toggle’)  // 反转按钮状态
$(‘#id’).button(‘reset’)                    // 重新恢复按钮原来的状态
$(‘#id’).button(‘任意字符’)          // 任意参数，替换data-参数名-text的属性值为按钮文本值

// 另外，不传任何参数的调用，等同于传loading参数，因为源码里的默认值是loading，即
// $(‘#id’).button()和$(‘#id’).button(‘loading’)是相同的
```

就这么简单，所以大家使用的时候，唯一需要注意的就是，找到合适的时机恢复按钮状态即可。另外，该插件没有提供其他的回调事件。

5.9.3 源码分析

因为功能简单，所以源码自然就简单。在按钮插件里，源码主要就体现在如下几个方面：

❑toogle的特殊处理。

❑其他自定义参数的特殊处理。

❑jQuery的fn插件定义。

❑初始化绑定。

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2 插件核心代码。首先是button插件类及原型方法的定义，主要源码如下：

```
// button插件类及原型方法的定义
var Button = function (element, options) { // 传入要触发的元素和调用options选项参数
     this.$element = $(element)
this.options = $.extend({}, Button.DEFAULTS, options)   // 合并参数
this.isLoading = false                    // 是否加载状态
}
Button.DEFAULTS = {
    loadingText: 'loading…'            // 默认loading时的文本内容
}

// 设置按钮状态的方法，所有的自定义参数都使用这个方法
Button.prototype.setState = function (state) {};
// 切换按钮状态
Button.prototype.toggle = function (){};
```

setState方法是最关键的，主要是根据传入的参数state（例如loading），来进行相关的设置。详细源码如下：

```
// 设置按钮状态的方法
Button.prototype.setState = function (state) {
    var d = 'disabled'                              // 按钮需要禁用时使用它，先赋值一个临时变量
    var $el = this.$element                  // 当前元素
    var val = $el.is('input') ? 'val' : 'html'
                                                    // 如果是input，则使用val获取值，否则使用html获取值
     var data = $el.data()  // 获取当前元素的自定义属性，即所有以data-开头的属性

    state = state + 'Text'
     // 组装下面需要用到的属性，比如传入了loading，则组装成loadingText，在下面就查找
    // data-loading-text属性值
```

```
    if (!data.resetText) $el.data('resetText', $el[val]())
     // 如果data里不包含data-reset-text值，则将当前元素的值赋给data-reset-text
临时存
    // 放，以便过后再恢复使用它

    $el[val](data[state] || this.options[state])
      // 给元素赋新值，先从自定义属性里查询（比如data-complete-text），如果没有，
就从options
    // 默认值里查询

    // 不阻止事件，以允许表单的提交
    setTimeout($.proxy(function () {
        if (state == 'loadingText') {   // 如果传入的是loading,
            this.isLoading = true       // 设置加载状态为true
             $el.addClass(d).attr(d, d)  // 禁用该元素（即添加disabled样式和
disabled属性）
        } else if (this.isLoading) {
            this.isLoading = false
            $el.removeClass(d).removeAttr(d)
                                                  // 如果不是loading，则删除
disabled样式和disabled属性
        }
    }, this), 0)
}

// 切换按钮状态
Button.prototype.toggle = function () {
    var changed = true                  // 设置change标记
var $parent = this.$element.closest('[data-toggle="buttons"]')
// 查找带有[data-toggle="buttons"]属性的最近父元素

    if ($parent.length) {               // 如果父元素存在
        var $input = this.$element.find('input')        // 查看触发元素内
是否有input
         if ($input.prop('type') == 'radio') {            // 如果查找到的
input是radio
            if ($input.prop('checked') && this.$element.hasClass('active
                 changed = false        // 判断如果该radio已经是高亮且是被选
中的，则不需
                                        // 要再改变（change为false）
            }else{ // 否则，查找同级元素是否有为active的，如果有，删除之
                 $parent.find('.active').removeClass('active')
            }
        }
        if (changed) $input.prop('checked', !this.$element.hasClass('act
            trigger('change')
          // 如果有，继续判断是否活动（有active样式），否则的话则设置为禁用；不活
动，就设置为启用，
        // 并触发change事
    }

if (changed) this.$element.toggleClass('active')
   // 在change标记为true时，将自身元素的状态进行反转，即反转active样式
}
```

setState原型方法主要是结合参数，组合xxxText这样的名称，然后设置自定义属性（data-xxx-text），最后再调整disabled状态和样式，而toggle就不多说了。

步骤3　jQuery插件定义。具体代码如下：

```
// 在jQuery上定义alert插件，并重设插件构造器
var old = $.fn.button
//  保留其他库的$.fn.button代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.button = function (option) {
    return this.each(function () {      // 遍历所有符合规则的元素
        var $this = $(this)
          var data = $this.data('bs.button')        // 获取自定义属性data-bs.button的值
                                                    // （其实是button实例）

        // 如果option是对象，说明是参数集合，在new Button的时候传入
        var options = typeof option == 'object' && option

        // 如果没有button实例，就初始化一个，并传入this
        if (!data) $this.data('bs.button', (data = new Button(this, opti

        // 如果option是toggle字符，说明传入的是一个方法，直接调用该方法
        // 否则调用setState方法
        if (option == 'toggle') data.toggle()
        else if (option) data.setState(option)
    })
}
$.fn.button.Constructor = Button        // 并重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。绑定触发事件的源码如下：

```
// 绑定触发事件
// 查询所有以button开头为值的data-toggle属性，绑定click事件
$(document).on('click.bs.button.data-api',                          '[data-toggle^=button]', function (e) {
    var $btn = $(e.target)      // 当前单击对象
    if (!$btn.hasClass('btn')) $btn = $btn.closest('.btn')
                                        // 如果没有btn样式，就查找最近带有btn样式的元素
    $btn.button('toggle')       // 反转状态
    e.preventDefault()          // 组织默认事件
})
```

从上述代码中可以看到，该初始化是设置了toggle参数，也就说只提供了状态反转功能，没有提供loading这样的功能。

另外还有一个问题，在上述代码中查询元素用选择符[data-toggle^=button]的时候，其意思是指data-toggle属性，并且其值是以button

开头（比如声明式用法里的data-toggle=“button”和data-toggle=“buttons”），这就有了一个隐性的bug，如果自己想定义别的插件，比如data-toggle=“buttonABC”，就会受这个插件的影响，而导致出错。所以还是选项卡插件的方式比较好，如下面的示例。

笔者给出的建议代码如下：

```
$(document).on(‘click.bs.button.data-api          ’,          ’[data-
toggle=“button”], [data-
         toggle=“buttons”]’, function (e) {
     // 处理逻辑
})
```

5.10　折叠

源文件：collapse.js

折叠插件实现的效果是：当单击一个触发元素时，在另外一个可折叠区域进行显示（或隐藏），再次单击时可以反转显示状态。经典的场景是多个折叠区域的手风琴风格（accordion），以及单一title/content的风格，其中手风琴效果如图5-16所示。

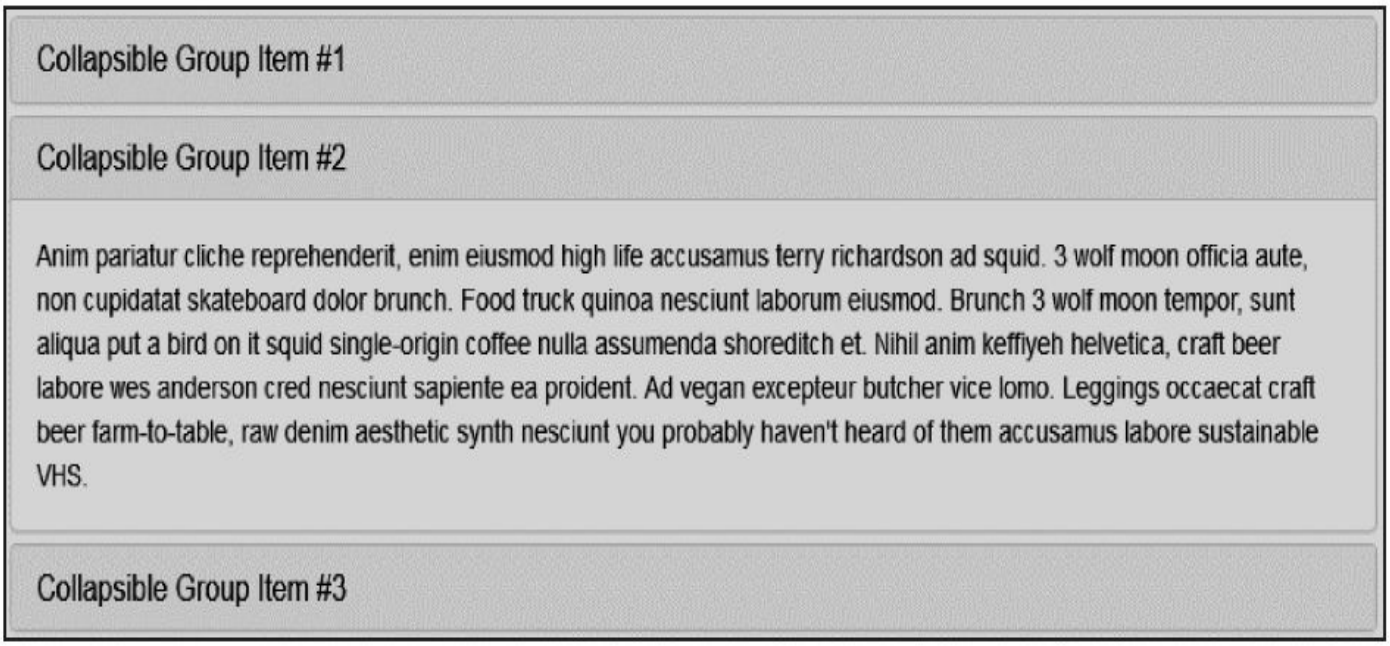


图5-16　手风琴折叠效果

5.10.1 声明式用法

最简单的用法就像其他功能一样，声明一个触发按钮和一个折叠区域。先看以下默认显示折叠区域的示例。

```
<!– 默认显示折叠区域–>
<button class=“btn btn-danger” data-toggle=“collapse” data-target=”#demo”>触发按钮</button>
<div id=“demo” class=“collapse in” >折叠区域…</div>
```

默认隐藏折叠区域则有所不同，触发元素要添加一个.collapsed样式，而且折叠区域的.in样式也要去除。

```
<!– 默认隐藏折叠区域–>
<button class=“btn btn-danger collapsed” data-toggle=“collapse” data-target=”#demo”>触发
    按钮</button>
<div id=“demo” class=“collapse ” >折叠区域…</div>
```

.collapsed样式是表示该插件所对应的demo区域已经处于隐藏状态了，而collapse和in两个样式一起用的时候是块级显示状态，单独collapse样式则是隐藏状态。另外，该触发元素还可能有一个collapsing样式，该样式是在折叠区域进行隐藏显示动画过程中附加的。

通过上述两个例子和样式分析，我们知道了显示和隐藏的两个状态，其实该插件操作的也就是这些内容。可以为该插件总结出以下两个基本的要点：

❑要标识插件的data-toggle=“collapse”以及表示折叠区域的data-target=”#demo”。

❑要正确设置折叠区域的显示/隐藏状态，以及触发元素的collapsed样式标记。

读者可能会问，这只是单个折叠区域的情况，针对手风琴风格，一般里面都会有多个折叠区域，而且显示一个折叠区域的时候，要关闭所有其他的折叠区域。

针对这个情况，可根据前面几个插件的经验猜想出，要实现上述效果，就得为手风琴定义一个大的容器元素来包含所有的触发元素和折叠区域，在单击其中一个元素的时候，先关闭所有的折叠区域，再打开所单击的区域（如果所单击的区域原来就是打开的，则关闭它），这样一来就实现了我们想要的效果了。

但是如何通过声明式用法来使用呢？答案是：Bootstrap给触发元素提供了一个data-parent属性，用于设置一个父容器，在处理的时候，会把这个容器里的所有折叠区域都关闭掉，从而实现我们所说的手风琴效

果。示例代码如下：

```
<div class=“panel-group” id=“accordion”>
    <div class=“panel panel-default”>
        <div class=“panel-heading”>
            <h4 class=“panel-title”>
                                        <a  data-toggle=“collapse”  data-
parent=”#accordion” href=
                    ”#collapseOne”>触发元素#1</a>
            </h4>
        </div>
        <div id=“collapseOne” class=“panel-collapse collapse in”>
            <div class=“panel-body”>
                折叠内容…
            </div>
        </div>
    </div>
    <!– 其他两个pannel格式和上边的一样 –>
</div>
```

通过示例，我们可以看出以下几点：

❑使用panel的panel-title作为触发元素，使用panel-body的父元素作为折叠区域。

❑使用一个panel-group样式的元素包含多个panel，从而实现手风琴效果。

❑每个panel里的触发元素都要指定data-parent属性（这是重复工作，希望作者改进一下）。

❑data-parent属性的值（本例是accordion）对应panel-group样式元素的id（或者样式选择符也行）。在源码分析的时候记住这几点就行了。

5.10.2 JavaScript用法

折叠插件的JavaScript用法和普通的jQuery插件的使用方式一样，如果要手动触发一个折叠区域进行反转显示，可以使用如下代码：

```
$(“.collapse”).collapse();
```

但这种用法一般都不常用，因为平时都是事先全部触发好，然后在某个过程中，可能再需要一些强制显示或者隐藏的方法。

1.选项

除了在声明式里讲到的parent属性以外，该插件还支持一个toggle的参数，属性和解释如表5-16所示。

表 5-16 折叠组件的 JavaScript 用法选项

属性名称	类 型	默 认 值	描 述
parent	selector	false	如果指定了 parent，在单击一个特定触发元素的时候，该 parent 下的所有折叠区域都会隐藏，从而达到手风琴效果
toggle	boolean	true	是否开启反转功能，以便在多次单击的时候进行状态反转

其他可支持的方式如下：

```
// 通用方式，options可以定义其他属性
$(‘#element’).collapse(options);
// 例子：指定parent和toggle参数
$(‘#element’).collapse({
    parent: ”#accordion”,
    toggle: false
});
$(‘#element’).collapse(‘show’);          // 显示折叠区域
$(‘#element’).collapse(‘hide’);          // 隐藏折叠区域
$(‘#element’).collapse(‘toggle’);        // 反转折叠区域
$(‘#element’).collapse();                // 效果和toggle一样
```

2.事件

事件照旧是4种类型，分别对应下拉菜单的弹出前、弹出后、关闭前、关闭后，解释和用法如表5-17所示。

表 5-17　折叠组件的事件类型

事件类型	描　述
show.bs.collapse	在 show 方法调用时立即触发（尚未显示之前）
shown.bs.collapse	该事件在折叠区域完全显示给用户之后（并且等 CSS 动画完成之后）触发
hide.bs.collapse	在 hide 方法调用时（但还未关闭隐藏）立即触发
hidden.bs.collapse	该事件在折叠区域完全隐藏之后（并且等 CSS 动画完成之后）触发

调用方式也很简单，和普通的jQuery代码并无二样。

```
$('#myCollapsible').on('hidden.bs.collapse', function () {
    // 处理代码…
})
```

手风琴风格使用到的panel-group样式主要是多个panel之间的外边距、边框和圆角等，具体细节请自行查看bootstrap.css的5765～5789行源码。

5.10.3 源码分析

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2 插件核心代码。虽然折叠的功能比较简单（就是隐藏和显示元素），但里面也还包括了一些令人欣喜的内容，我们一步一步来看。核心代码如下：

```
// Collapse类的定义
var Collapse = function (element, options) {
    this.$element = $(element)          // 当前折叠区域的元素
    this.options = $.extend({}, Collapse.DEFAULTS, options) // 合并参数
    this.transitioning = null           // 是否正在执行显示/隐藏操作

    if (this.options.parent) this.$parent = $(this.options.parent)
                                                    // 如果参数里指定了
parent，则赋值它
    if (this.options.toggle) this.toggle()      // 如果支持toggle，则直接
调用toggle方法
}

Collapse.DEFAULTS = {
    toggle: true                                // 默认值，是否支持折叠区域
的显示状态反转
}
// 获取折叠区域的显示动画的打开方向，是从左向右(width)，还是从上向下(height)，默认
为height
Collapse.prototype.dimension = function () {};
// show方法用于显示折叠区域
Collapse.prototype.show = function () {};
// hide方法用于隐藏折叠区域
Collapse.prototype.hide = function (){};
```

其实通过上面的注释就可以看到，有一个dimension不同寻常，从注释可以看出，元素的隐藏和显示并不是一卡一卡的状态，而是一个模拟动画，比如高度（height）从0到原有高度，或者宽度（width）从0到原有宽度，通过这样的效果，达到平滑的目的。那么我们就来仔细看看它是如何实现这个效果的。代码如下：

```
// 获取折叠区域的显示动画的打开方向，是从左向右(width)，还是从上向下(height)，默认
为height
Collapse.prototype.dimension = function () {
    var hasWidth = this.$element.hasClass('width')      // 折叠区域元素上
是否有width样式
    return hasWidth ? 'width' : 'height'  // 有，则返回width，没有，则返回
height
}
// show方法用于显示折叠区域
Collapse.prototype.show = function () {
    // 如果正在执行collapse操作，或者该折叠元素已经显示，就不做处理了
```

```
    if (this.transitioning || this.$element.hasClass('in')) return

    var startEvent = $.Event('show.bs.collapse')         // 定义要触发的事
件命名空间
    this.$element.trigger(startEvent)   // 在显示之前，先触发show事件
    if (startEvent.isDefaultPrevented()) return
                                                         // 如果show事件的回调里阻
止了继续操作，则直接返回

    // 如果parent存在（手风琴风格），则查找所有该元素内已经打开的折叠区域
    var actives = this.$parent && this.$parent.find('> .panel > .in')

    if (actives && actives.length) {    // 如果找到的已打开的折叠区域存在
        var hasData = actives.data('bs.collapse')        // 查找该折叠区域
上面的实例

        // 如果实例存在，并且正在执行相关的collapse操作，则直接返回
        if (hasData && hasData.transitioning) return

        actives.collapse('hide')        // 关闭所有找到的已打开的折叠区域
        hasData || actives.data('bs.collapse', null)    // 并且消除其上面
的所有实例
    }

    var dimension = this.dimension()     // 获取显示展开元素的方向，默认是
height（上下）

    this.$element
        .removeClass('collapse')        // 删除折叠区域上的collapse样式
        .addClass('collapsing')         // 然后再添加collapsing样式
        [dimension](0)    // 将height设置为0，表示上下展开，如果是width，则表
示左右展开

    this.transitioning = 1              // 表示正在处理collapse插件的显示工
作

    // 回调函数，用于处理完成状态
    var complete = function () {
        this.$element
            .removeClass('collapsing')  // 删除collapsing样式
            .addClass('collapse in')    // 添加in样式，表示已显示
            [dimension]('auto')         // 将height（或width）设置为auto
        this.transitioning = 0          // 表示已经处理完成
        this.$element.trigger('shown.bs.collapse')      // 触发shown事件
    }

    if (!$.support.transition) return complete.call(this)
                                                        // 如果不支持动画，直接调用
complete函数

    // 获取表示折叠元素的scroll大小的方向，结果是scrollHeight或者scrollWidth
    var scrollSize = $.camelCase(['scroll', dimension].join('-'))

    // 延迟350毫秒才执行动画,动画结束以后，调用complete回调函数
       //  并设置正常的高度或宽度，例如this.$element[height](this.$element[0]
[scrollHeight])
```

```
    this.$element
        .one($.support.transition.end, $.proxy(complete, this))  //
        .emulateTransitionEnd(350)
        [dimension](this.$element[0][scrollSize])
}
// hide用于隐藏折叠区域
Collapse.prototype.hide = function () {
    // 如果正在执行collapse操作，或者该折叠元素已经隐藏，就不做处理了
    if (this.transitioning || !this.$element.hasClass('in')) return

     var startEvent = $.Event('hide.bs.collapse') // 定义要触发的事件命名空间
    this.$element.trigger(startEvent)   // 在关闭之前，先触发hide事件
    if (startEvent.isDefaultPrevented()) return
                                                    // 如果hide事件的回调里阻止了继续操作，则直接返回

      var dimension = this.dimension()    //  获取显示展开元素的方向，默认是height（上下）
    this.$element
        [dimension](this.$element[dimension]())
        [0].offsetHeight                // 重绘该折叠区域，得到实际高度

    this.$element
        .addClass('collapsing')         // 添加collapsing样式
        .removeClass('collapse')        // 删除collapse样式
        .removeClass('in')              // 删除in样式，表示隐藏

    this.transitioning = 1               // 表示正在处理collapse插件的隐藏工作

    // 回调函数，用于处理完成状态
    var complete = function () {
        this.transitioning = 0          // 表示已经处理完成
        this.$element
            .trigger('hidden.bs.collapse')      // 触发hidden事件
            .removeClass('collapsing')  // 删除collapsing样式
            .addClass('collapse')       // 添加collapse样式表示隐藏
    }

    if (!$.support.transition) return complete.call(this)
                                                    // 如果不支持动画，直接调用complete函数

    this.$element
        [dimension](0)   // 根据展开的方向，直接将height（或者width）设置为0
        .one($.support.transition.end, $.proxy(complete, this))
         .emulateTransitionEnd(350)       // 350毫秒后，执行动画，然后再调用complete回调函数
}
Collapse.prototype.toggle = function () {
    this[this.$element.hasClass('in') ? 'hide' : 'show']()
                                                    // 有in样式则表示已经显示了，应该隐藏，否则显示
}
```

如果是我们自己写类似的插件，估计都是hide或者show一下就结束了，看到上述代码，才知道Bootstrap的作者为了产生一点平滑的效果，确实花了很多工夫。

通过上述代码，我们也发现可以利用这个功能的方法（因为官方网站并没说明该功能的用法），那就是，默认情况下是上下方向显示（也就是height），如果要左右方向怎么办呢？答案肯定是在折叠区域元素上添加width样式就可以了。示例如下：

```
<!—  默认显示折叠区域—>
<button    class=“btn    btn-danger”    data-toggle=“collapse”    data-
target=”#demo”>触发
        按钮</button>
<div id=“demo” class=“collapse in width” >折叠区域…</div>
```

步骤3　jQuery插件定义。具体代码如下：

```
// 在jQuery上定义collapse插件，并重设插件构造器
var old = $.fn.collapse
//  保留其他库的$.fn.collapse代码（如果定义的话），以便在noConflict之后，可以继续
使用该老代码
$.fn.collapse = function (option) {
    return this.each(function () {      // 遍历所有符合规则的元素
        var $this = $(this)
        var data = $this.data(‘bs.collapse’)
                                                    // 获取自定义属性data-
bs.collapse的值（其实是
                                                // collapse实例）
        // 合并参数
        var options = $.extend({}, Collapse.DEFAULTS, $this.data(), type
            ‘object’ && option)

        // 如果实例不存在，并且options.toggle存在，且传入的option为show
        // 则设置show为false，避免在string判断的时候执行data[option]()
        if (!data && options.toggle && option == ‘show’) option = !optic

        // 如果没有collapse实例，就初始化一个，并传入this
        if (!data) $this.data(‘bs.collapse’, (data = new Collapse(this,
        if (typeof option == ‘string’) data[option]()
                                                // 如果传入的是字符串，则调
用对应的方法
    })
}
$.fn.collapse.Constructor = Collapse    // 并重设插件构造器，可以通过该属性获
取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。该步骤的绑定触发代码比较复杂，和其他的插件有所不同，该插件在触发元素上绑定事件，但并没有将触发元素作为Collapse类的实例。具体如下：

```
// 绑定触发事件
// 在带有data-toggle=collapse属性的元素上绑定事件
$(document).on('click.bs.collapse.data-api', '[data-toggle=collapse]',
    function (e) {
    var $this = $(this), href

     // 查找target，即所指定的折叠地区的id或者选择符，如果没有target，就使用href
里的值
    var target = $this.attr('data-target')
        || e.preventDefault()
             || (href = $this.attr('href')) && href.replace(/.*(?=#
[^\s]+$)/, '')
                                                    // IE7特殊处理
    var $target = $(target)             // 包装jQuery对象
    var data = $target.data('bs.collapse')// 查找上面是否已经有了collapse实
例

    // 如果有，就将toggle作为option（也就是反转状态）
    // 如果没有，收集该触发元素上的所有data-属性作为option配置参数
    var option = data ? 'toggle' : $this.data()
    var parent = $this.attr('data-parent')      // 查看该触发元素上是否设置
parent元素容器
     var $parent = parent && $(parent)   // 如果设置了parent，获取其jQuery
对象

         if (!data || !data.transitioning) { // 如果实例不存在，或者
transitioning不存在
        // 如果parent存在，关闭所有该容器内的折叠区域
               if ($parent) $parent.find('[data-toggle=collapse][data-
parent="' +
           parent + '"]').not($this).addClass('collapsed')
          // 如果当前折叠区域已经打开了，则在触发元素上应用collapsed样式做标记，否
则，删除
        // collapsed样式
           $this[$target.hasClass('in') ? 'addClass' : 'removeClass']
('collapsed')
    }
    $target.collapse(option)   // 调用collapse，开始执行
})
```

通过上述代码可以看到，上述代码是使用$target作为调用collapse的实例对象，而不是触发按钮，这也是为了和其他插件有所不同。

5.11　旋转轮播

源文件：carousel.js、carousel.less

CSS文件：bootstrap.css 5421行之后

旋转轮播（Carousel）这个插件有很多种译法，有人叫轮播，有人叫传送带，本书为了形象而译为“旋转轮播”（其实我个人私下都是叫它幻灯片），也简称轮播。其主要显示效果就像各大网站的400×600的多幅滚动广告一样，比如京东首页的大图片，默认情况下是循环向左轮播，如果单击某一小方块的话，会直接显示所单击的那张图。本插件不仅要实现这种效果，还实现定制化（也就是可以用JavaScript代码手动触发显示或者隐藏任意一张图片），效果如图5-17所示。

图5-17　旋转轮播幻灯片效果

5.11.1 声明式用法

轮播的声明式用法比较复杂（应该是所有Bootstrap插件里最复杂的），但理解了它的HTML结构以后，也就不觉得复杂了。来看一下一般的声明结构和运行效果：

```
<div data-ride="carousel" class="carousel slide" id="carousel-container">
    <!— 图片容器 —>
    <div class="carousel-inner">
        <div class="item"><img alt="第一张图" src="A" /></div>
        <div class="item active"><img alt="第二张图" src="B" /> </div>
        <div class="item"><img alt="第三张图" src="C" /> </div>
    </div>
    <!— 圆圈指示符 —>
    <ol class="carousel-indicators">
        <li data-slide-to="0" data-target="#carousel-container"></li>
        <li data-slide-to="1" data-target="#carousel-container"></li>
                    <li data-slide-to="2" data-target="#carousel-container" class="active"></li>
    </ol>
    <!— 左右控制按钮 —>
                        <a data-slide="prev" href="#carousel-container" class="left carousel-control">
        <span class="glyphicon glyphicon-chevron-left"></span>
    </a>
                        <a data-slide="next" href="#carousel-container" class="right carousel-control">
        <span class="glyphicon glyphicon-chevron-right"></span>
    </a>
</div>
```

乍一看，有很多粗体的代码，但我们来一一分析一下，就会去除畏惧心理。

首先，任何插件一般都有一个父容器，本插件也不例外，带有data-ride="carousel"的div就是轮播插件的容器，容器的id为carousel-container，稍后会用到；还有两个样式，其中carousel样式做样式容器，而slide样式和fade类似，用来定义轮播图片的时候是否有特效。

然后再看其内部结构，分为以下3个部分：

❑carousel-inner样式div内部包含多个含有item的div样式，在这些内部的div里，定义我们要显示的幻灯图片。

❑carousel-indicators样式ol内部定义了一组标示符，用户单击这些标示符可以直接显示相应的图片，而这些标示符上面都定义了类型data-slide-to="2"这样的属性，其表示单击该标识符显示第3张图片（索引从0开始）。

❑另外两个a链接是一组，用于显示左右箭头，可以改变轮播的方向。注意一下，这两个元素上定义的data-slide属性值只能是prev或者next（表示上一张、下一张）。

需要特别说明的是，ol指示符元素在3个部分的位置可以任意定义，左右控制链接（a元素）可以放在ol前面，也可以放在ol后面，但是千万不能放在carousel-inner样式div的前面（会被其遮盖住）。

另外，还有几点需要再解释一下（在源码分析时容易理解）：item样式上如果有active样式的话，则表示该图正在显示，其他图片则都隐藏；提示符上的data-target和左右控制链接href里的值都一样，其表示容器元素的id（或者样式），以便后期这些元素被单击的时候，可以很方便地找到容器元素。

针对图片，轮播插件还提供了一个字幕说明样式（carousel-caption），紧接着img元素定义即可。示例如下：

```
<div class="item active">
    <img src="…" alt="…">
    <div class="carousel-caption">
        <h3>标题</h3>
        <p>描述…</p>
    </div>
</div>
```

除了data-ride="carousel"、data-slide、data-slide-to以外，轮播组件还支持其他3个自定义属性，如表5-18所示。

表 5-18　旋转轮播组件的自定义属性

属性名称	类　型	默 认 值	描　述
data-interval	number	5000	幻灯片轮换的等待时间（毫秒）。如果为 false，轮播将不会自动开始循环
data-pause	string	hover	默认鼠标停留在幻灯片区域即暂停轮播，鼠标离开即启动轮播
data-wrap	boolean	true	轮播是否持续循环

上述3个自定义属性可以在容器元素上定义，也可以在标示符（或左右控制链接）上定义，但是后者定义的优先级比较高，比如如果你在容器元素上定义时间间隔（interval）为5秒，而标示符A上定义的是3秒，那一旦你单击该标示符A以后，后续的轮播就会使用3秒作为默认设置，如果你再单击了一个没有设置时间的标示符B，那后续的轮播则又会继续使用5秒作为默认设置（后续JavaScript源码分析时请注意这一点）。

注意

Bootstrap基于CSS3实现动画过渡效果，所以IE 8、IE 9不支持这些效果。

5.11.2　CSS源码分析

在这一节，我们将只分析主要的一部分CSS样式，其他的样式因为和JavaScript代码关系比较大，所以会在JavaScript源码分析小节进行分析。首先，先看一下轮播插件的HTML结构示意图，如图5-18所示。

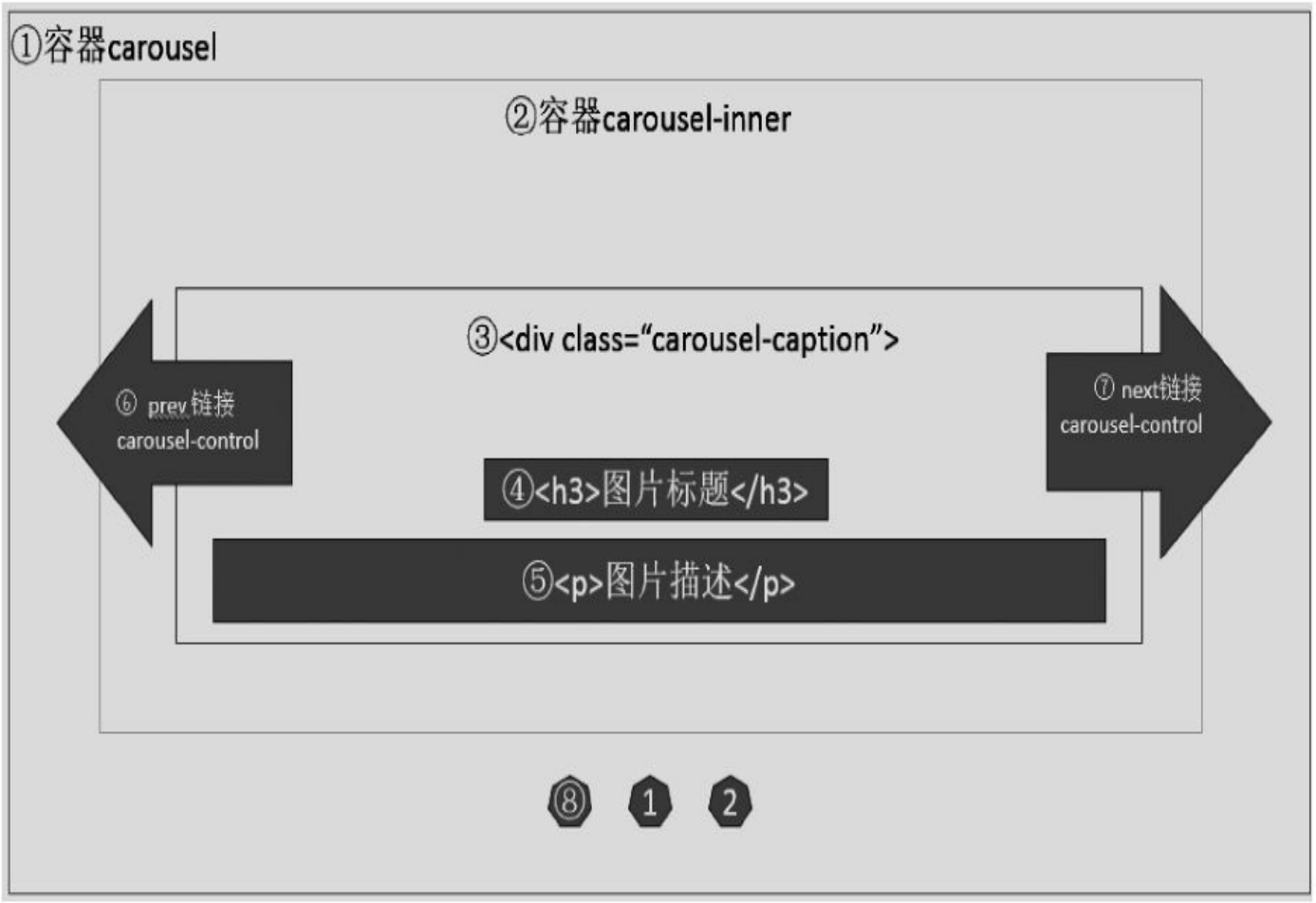


图5-18　旋转轮播元素布局示意图

结构示意图中有8个序号，我们来逐一分析一下相关的CSS样式。首先是容器样式carousel，它只定义了一个相对定位position: relative;，所以在使用的过程中需要再次定义该元素的高度和宽度。

其次是2号carousel-inner样式，该样式是图片集的容器，其内部放置了多个以item为样式的div元素，主要定义了定位和默认的隐藏与显示状态。主要源码如下：

```
// 源码5424行
.carousel-inner {                                                    /*图片集容器*/
  position: relative;                                          /*相对定位*/
  width: 100%;                                                 /*充满父元素*/
   overflow: hidden; /*超出部分隐藏，这一点非常重要，即将显示的图片将在超出部分，然后再以动画形式滑入*/
}
.carousel-inner > .item {                                      /*每个图片item容器的样式*/
  position: relative;                                          /*相对定位*/
```

```
  display: none;                                          /*默认都不显示*/
  -webkit-transition: 0.6s ease-in-out left;     /*过渡动画*/
          transition: 0.6s ease-in-out left;
}
.carousel-inner > .item > img,
.carousel-inner > .item > a > img {              /*item内的图片元素上应用的
样式*/
  display: block;                                 /*块级显示*/
  height: auto;                                           /*高度自适应*/
  max-width: 100%;                                /*最大宽度100%*/
  line-height: 1;
}
.carousel-inner > .active,
.carousel-inner > .next,
.carousel-inner > .prev {
/*inner容器内的任何元素，如果有acrive、prev、next样式，都设置为可见，主要用于上一
张、下一张图的过渡*/
  display: block;                                 /*块级显示*/
}
.carousel-inner > .active {                       /*如果是高亮图片，则左对齐
*/
  left: 0;
}
```

通过上述代码可以得知，默认情况下，所有的图片都是隐藏的，只有设置了.active样式的图片才以块级元素的方式显示出来，其位置是通过left: 0来达到左对齐的目的。

对于3、4、5号的图片说明样式，禁用一个.carousel-caption样式即可。源码如下：

```
// 源码5563行
.carousel-caption {                               /*图片说明，窄屏情况下*/
  position: absolute;                             /*绝对定位*/
  right: 15%;                                     /*右边留15%间距*/
  bottom: 20px;                                   /*底部外边距*/
  left: 15%;                                      /*左边留15%间距*/
  z-index: 10;                                     /*加大z-index，高于其他元
素*/
  padding-top: 20px;                              /*顶部内间距*/
  padding-bottom: 20px;                           /*底部内间距*/
  color: #fff;                                    /*文本为白色*/
  text-align: center;                             /*居中*/
  text-shadow: 0 1px 2px rgba(0, 0, 0, .6);     /*阴影设置*/
}
.carousel-caption .btn {
  text-shadow: none;                               /*有按钮的话，取消按钮的阴
影*/
}
```

6号和7号分别是左右箭头，这两个箭头的z-index高度图片容器，其定义原理主要是，在记录大容器左边和右边的15%的距离，分别显示两个a链接，然后在a链接内部定义需要显示的小图标。a链接的位置和行为

定义如下：

```
// 源码5472行
.carousel-control {
  position: absolute;                                  /*绝对定位*/
  top: 0;
  bottom: 0;                                        /*默认距离父元素的顶部、底部、左边都
是0距离*/
  left: 0;
  width: 15%;                                            /*宽度设置为整体父元素的
15%*/
  /*省略部分代码 */
  opacity: .5;
}
.carousel-control.left {                                    /*左箭头的背景设置
*/
  background-repeat: repeat-x;
                                        background-image:          linear-
gradient(to right, rgba(0, 0, 0, .5) 0%, rgba(0, 0, 0, .0001) 100%);
}
.carousel-control.right {                            /*右箭头的背景设置*/
  right: 0;
  left: auto;
    background-repeat: repeat-x;
                                        background-image:          linear-
gradient(to right, rgba(0, 0, 0, .0001) 0%,
        rgba(0, 0, 0, .5) 100%);
}
.carousel-control:hover,
.carousel-control:focus {                             /*鼠标移动上去的时候加深透
明度*/
  color: #fff;
  text-decoration: none;
  filter: alpha(opacity=90);
  outline: none;
  opacity: .9;
}
```

然后定义a链接内部的小图标。关于小图标，作者定义了两种样式，一种是glyphicon表示左右方向字体图标，另外一种是icon的字符图标。通用定义如下：

```
// 源码5503行
.carousel-control .icon-prev,
.carousel-control .icon-next,
.carousel-control .glyphicon-chevron-left,
.carousel-control .glyphicon-chevron-right {    /*内部小图标设置*/
  position: absolute;                           /*绝对定位*/
  top: 50%;                                      /*距carousel-control元素
的顶部50%*/
  z-index: 5;
  display: inline-block;                                 /*内联块级显示*/
}
.carousel-control .icon-prev,
.carousel-control .glyphicon-chevron-left {     /*向左的小图标*/
```

```
  left: 50%;  /*距carousel-control元素的左边50%，即15%的一半*/
}
.carousel-control .icon-next,
.carousel-control .glyphicon-chevron-right {    /*向右的小图标*/
  right: 50%; /*距carousel-control元素的右边50%，即15%的一半*/
}
```

字体样式glyphicon-chevron-left和glyphicon-chevron-right分别代表左右箭头；同理，字符样式icon-prev和icon-next分别代表另外一种形式的左右箭头，其字符是通过在before伪类上定义的。代码如下：

```
// 源码5520行
.carousel-control .icon-prev,
.carousel-control .icon-next {                  /*字符小图标*/
  width: 20px; margin-top: -10px;
  height: 20px;margin-left: -10px;
  font-family: serif;
}
.carousel-control .icon-prev:before {           /*左箭头*/
  content: '\2039';
}
.carousel-control .icon-next:before {           /*右箭头*/
  content: '\203a';
}
```

8号圆圈是代表指示符，其carousel-indicators样式最大一个注意点就是它的z-index是15，比图片的z-index还高，以防止被图片说明所遮盖。

```
// 源码5534行
.carousel-indicators {                          /*指示符*/
  position: absolute;                           /*绝对定位*/
  bottom: 10px;                                 /*距离底部10像素*/
  left: 50%;                                    /*距离左边50%*/
  z-index: 15;                                   /*加大z-index，以防被遮盖
*/
  /*省略一部分*/
  list-style: none;
}
.carousel-indicators li {                       /*指示符项*/
  display: inline-block;                                /*内联块显示*/
  width: 10px;                                  /*宽度高度均为10像素*/
  height: 10px;
  margin: 1px;
  text-indent: -999px;                          /*隐藏内部文本*/
  cursor: pointer;                          /*鼠标手型*/
  /*省略一部分*/
   border-radius: 10px; /*因为高度和宽度都是10像素，角度为10像素就意味着它就是一
个圆圈*/
}
.carousel-indicators .active {          /*高亮的圆圈指示符背景显示为白色*/
  width: 12px;                          /*宽度和高度稍微加大一些*/
  height: 12px;
  margin: 0;
  background-color: #ffffff;
}
```

上述1～8个区域的样式均为默认的普通样式，即无论是在窄屏还是在宽屏都可以。但作者为普通桌面浏览器又单独设置了一下，都稍微加大了一些，使其看起来更平滑。代码如下：

```
// 源码5578行
@media screen and (min-width: 768px) {  /*如果是普通桌面浏览器屏幕*/
  .carousel-control .glyphicons-chevron-left,
  .carousel-control .glyphicons-chevron-right,
  .carousel-control .icon-prev,
  .carousel-control .icon-next {                  /*加大左右控制链接图片的大小*/
    width: 30px;
    height: 30px;
    margin-top: -15px;
    margin-left: -15px;
    font-size: 30px;
  }
  .carousel-caption {                   /*加大距离父容器的左右距离*/
    right: 20%;
    left: 20%;
    padding-bottom: 30px;               /*加大底部内边距*/
  }
  .carousel-indicators {
    bottom: 20px;                       /*加大指示符的底部内边距*/
  }
}
```

也就是说，可以像如下这样定义左右箭头：

```
<!— 左右控制按钮 —>
<a data-slide=“prev” href=”#carousel-container” class=“left carousel-control”>
    <span class=“icon-prev”></span>
</a>
<a data-slide=“next” href=”#carousel-container” class=“right carousel-control”>
    <span class=“icon-next”></span>
</a>
```

5.11.3 JavaScript用法

默认情况下，如果在容器元素上已经定义了data-ride="carousel"属性，页面加载以后，就会自动加载该轮播插件进行图片切换。如果没有这个属性的话，可以使用JavaScript代码来触发这个行为。用法如下：

```
$('.carousel').carousel();                    // 所有带carousel样式的容器元素都开启轮播
// 或者
$('#carousel-container').carousel();          // 针对特定元素，开启轮播
```

1.选项

在开启轮播的时候，可以像自定义属性一样，使用另外3个额外的option选项，不同的只是去除data-前缀，具体如表5-19所示。

表 5-19 旋转轮播组件的 JavaScript 用法选项

属性名称	类 型	默 认 值	描 述
interval	number	5000	幻灯片轮换的等待时间（毫秒）。如果为 false，轮播将不会自动开始循环
pause	string	hover	默认鼠标停留在幻灯片区域即暂停轮播，鼠标离开即启动轮播
wrap	boolean	true	轮播是否持续循环

使用时，在初始化插件的时候传入对象字面量参数即可。例如：

```
$('.carousel').carousel({
    interval: 2000
})
```

2.方法

除了在初始化时支持自定义属性以外，该插件还支持外部调用，即，当你获取一个轮播插件的实例以后，可以在该实例上应用如下几个方法，以达到自己想要的效果，使用方法和用途如表5-20所示。

表 5-20 旋转轮播组件的方法和用途

方法名称	描 述
.carousel('cycle')	循环各帧（默认从左到右）
.carousel('pause')	停止轮播
.carousel(number)	轮播到指定的图片上（下标以 0 开始，类似数组）
.carousel('prev')	返回到上一张
.carousel('next')	转到下一张

关于插件实例的获取，所有的插件都一样，那就是，在初始化插件以后，通过一个自定义属性（data-bs-插件名称）即可获取该插件的实例，然后就可以调用其方法了。示例如下：

```
var data = $('#carousel-container').data('bs.carousel');
if (data instanceof Carousel) {          // 确保是轮播插件的实例
    data.carousel(0);
    data.carousel('next');
}
```

3.事件

该插件只提供两种事件类型，解释和用法如表5-21所示。

表 5-21　旋转轮播组件的事件类型

事件名称	描　述
slide.bs.carousel	此事件在 slide 方法被调用之后（但还未开始处理下一张图片之前）立即触发
slid.bs.carousel	在一张图片结束轮播之后触发（注：与 show 和 shown 相比，slid 应该改为 slidden）

调用方式也很简单，和普通的jQuery代码并无二样。但两个事件的接收参数却不一样，区别如下：

```
$('#myCarousel').on('slide.bs.carousel', function (relatedTarget, direct
       console.log(relatedTarget.html());    //  输出：<div  class="item">
<img alt="第一张
                                                    // 图" src="A" /></div>
    console.log(direction);              // left
    // 处理代码…
})

$('#myCarousel').on('slid.bs.carousel', function () {
    // 处理代码…
})
```

可以看出，slide事件回调里，可以获取当前即将要显示的图片元素的父元素和轮播方向。获取这两个对象以后，就可以很灵活地定义自己的处理逻辑了。

5.11.4 源码分析

在写这一小节的时候，我有一点小担心，我个人认为，旋转轮播的JavaScript代码是所有Bootstrap插件里最复杂的。所以在分析的时候，将配合一些HTML和实时的CSS以及图片来讲解。

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2 插件核心代码。旋转轮播的核心代码比较复杂，在理解了CSS原理之后，先看一下概要代码。

```
var Carousel = function (element, options) {
    this.$element = $(element)          // 容器元素，因为不管单击哪个，最终都会转换到
                                                                    // data-ride= “carousel”容器元素
    this.$indicators = this.$element.find(‘.carousel-indicators’)
                                                        // 查找小圆圈指示符元素集合
    this.options = options
     // 插件运行参数，优先级最高的是所单击元素上的data-属性，然后是容器上的data-属性，最后才是默认值
    this.paused =                       // 暂停标记
    this.sliding =                      // 轮播标记
    this.interval =                     // 轮播间隔标记
    this.$active =                      // 当前活动图片的对象
    this.$items = null                  // 所有的图片元素对象

    this.options.pause == ‘hover’ && this.$element                  // 如果设置鼠标移动上去就暂停的话
        .on(‘mouseenter’, $.proxy(this.pause, this))    // 鼠标进入时，执行pause方法进行暂停
        .on(‘mouseleave’, $.proxy(this.cycle, this))    // 鼠标移出时，执行cycle方法重启开启
}

Carousel.DEFAULTS = {
    interval: 5000,                                     // 默认间隔5秒
    pause: ‘hover’,                                     // 默认设置，鼠标移动上去图片就暂停
    wrap: true                                                      // 轮播是否持续循环
}

// 开启轮播（默认从右向左）
Carousel.prototype.cycle = function (e) {};
// 判断当前图片在整个轮播图片集的索引
Carousel.prototype.getActiveIndex = function (){};
// 直接轮播指定索引的图片
Carousel.prototype.to = function (pos) {};
// 暂停轮播
Carousel.prototype.pause = function (e) {};
// 轮播下一张图片
```

```
Carousel.prototype.next = function (){};
// 轮播上一张图片
Carousel.prototype.prev = function (){};
// 轮播的具体操作方法
Carousel.prototype.slide = function (type, next) {};
```

上述7个原型方法，前6个都是为第7个slide方法做辅助工作的，比如处理上一张、下一张、找到当前图片在图片集中的索引、判断移动方向、直接定位到指定的图片上等类似工作，而slide是真正执行轮播动画过渡效果的方法。先看前6个方法的详细代码：

```
// 开启轮播（默认从右向左）
Carousel.prototype.cycle = function (e) {
    e || (this.paused = false)                  // 如果没传e，将paused设置为false

     this.interval && clearInterval(this.interval)         // 如果设置了interval间隔，就清除它

    // 如果设置了options.interval间隔，并且没有暂停
    // 就将在下一个间隔之后，执行next方法（播放下一张图片）
    this.options.interval
        && !this.paused
        && (this.interval = setInterval($.proxy(this.next, this), this.c

    return this                                          // 返回this，以便链式操作
}
// 判断当前图片在整个轮播图片集中的索引
Carousel.prototype.getActiveIndex = function () {
      this.$active = this.$element.find('.item.active')     // 找到当前active图片元素（其实是元素
                                                           // 外部的div容器）

     // 在找到该元素的父容器（即carousel-inner样式容器）的子集合（即所有的item元素集合）
    this.$items = this.$active.parent().children()

    return this.$items.index(this.$active)      // 判断当前图片元素在集合中的索引位置，并返回
}
// 直接轮播指定索引的图片
Carousel.prototype.to = function (pos) {
    var that = this
    var activeIndex = this.getActiveIndex()     // 查找当前图片的索引位置

    // 如果传入的pos值大于图片总数，或者小于0，则直接返回不做任何操作
    if (pos &gt; (this.$items.length - 1) || pos &lt; 0) return
     // 如果正在执行其他图片轮播，则在其结束以后再跳转到指定的pos图片（通过触发一次性的slid事件来实现）
    if (this.sliding) return this.$element.one('slid.bs.carousel', funct
    // 如果当前活动图片正好是指定的pos图片，则先暂停，然后继续执行
    if (activeIndex == pos) return this.pause().cycle()
```

```
    // 如果pos大于当前活动图片的索引，则传入next方法，否则是prev方向
    // 第二个参数是将pos对应的item元素对象传进去（具体作用查看下面的slide方法）
    return this.slide(pos &gt; activeIndex ? ‘next’ : ‘prev’, $(this.$it
}
// 暂停轮播
Carousel.prototype.pause = function (e) {
    e || (this.paused = true)                    // 如果没传e，将paused设置
为true（说明要暂停）

    // 如果有next或prev元素，并且支持动画，则触发动画
    if (this.$element.find(‘.next, .prev’).length && $.support.transitio
        this.$element.trigger($.support.transition.end) // 触发动画
         this.cycle(true)                              // 开始执行（注意传入了
true参数）
    }
    this.interval = clearInterval(this.interval)
    return this                                          // 返回this，以便
链式操作
}
// 轮播下一张图片
Carousel.prototype.next = function () {
     if (this.sliding) return                        // 如果正在轮播（还没结
束），直接返回
    return this.slide(‘next’)                    // 否则，轮播下一张图片
}
// 轮播上一张图片
Carousel.prototype.prev = function () {
     if (this.sliding) return                        // 如果正在轮播（还没结
束），直接返回
    return this.slide(‘prev’)                    // 否则，轮播上一张图片
}
```

上述6个原型方法里，主要是利用了jQuery的基础操作方法，来对各个图片元素进行查找，并设置各种状态，但最终都会调用slide方法。在分析slide方法之前，我们先来理一下思路。

首先假定一个场景，有3张图片需要轮播，方向是从右向左，并且假定当前第2张图片B正在处理显示状态，效果如图5-19所示。

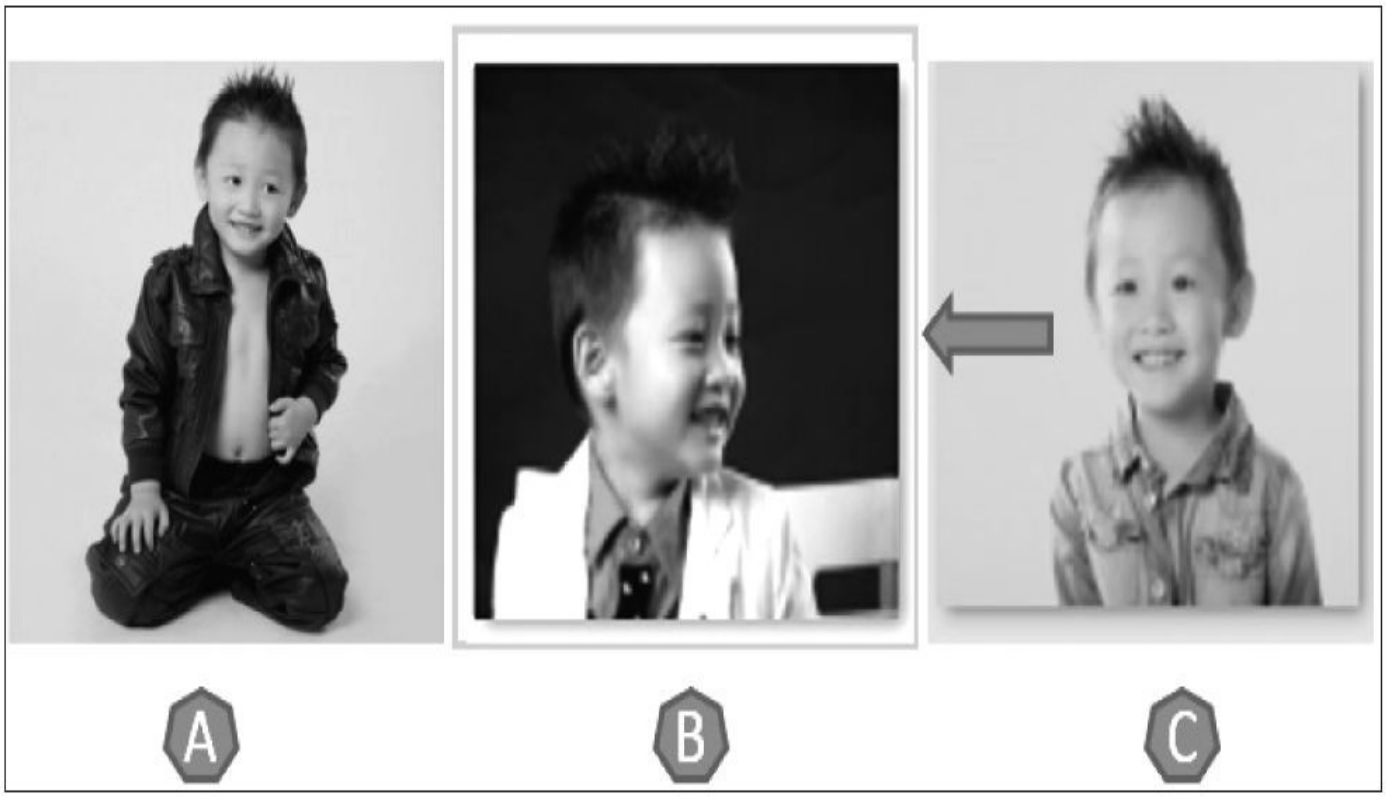


图5-19　图B在高亮状态

这时候，C要向左移动，也就意味着A和B都要向左移动（或者是隐藏掉），这时候C才能达到中间的显示区域，效果如图5-20所示。

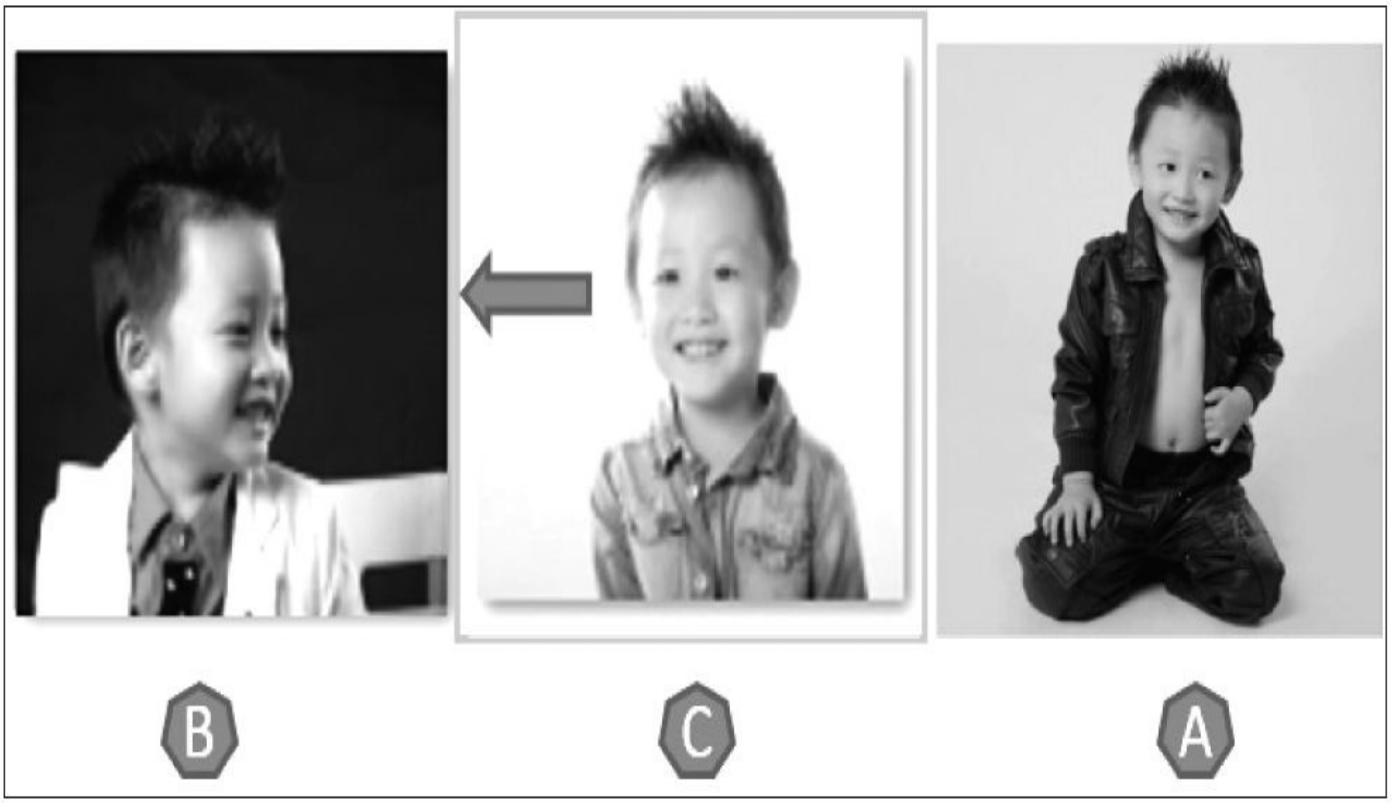


图5-20　图B左移，图C移动到高亮状态

C一旦达到中间的显示区域以后，由于只有3张图片，所以C的下一张就应该是第二轮的A图片。当然，默认参数里有个wrap来设置是否循环

滚动，如果是false，那C显示以后，就停止不动了。剩余几个原型方法的详细代码如下：

```
// 轮播的具体操作方法
Carousel.prototype.slide = function (type, next) {
    var $active = this.$element.find('.item.active')    // 找到当前活动的图片对象条目

    // 如果提供了next参数，就使用这个参数，如果没提供，就使用当前活动条目的下一个图片条目
    var $next = next || $active[type]()
    var isCycling = this.interval

    // 获取移动的方向：如果是next，则是向左移动，否则是向右移动
    var direction = type == 'next' ? 'left' : 'right'

    // 如果获取失败，指定一个元素进行特殊处理，如果再传next，则指向下一轮的图片
    // 即如果最后一个图片显示以后，还要next，那就是下一轮的first
    var fallback = type == 'next' ? 'first' : 'last'
    var that = this                         // 获取当前调用者的this对象，防止作用域污染

    if (!$next.length) {                            // 如果下一个对象不存在
        if (!this.options.wrap) return  // 判断wrap是否为假，如果是，则直接返回
        $next = this.$element.find('.item')[fallback]()
                                                    // 否则，使用fallback指定的元素当做 $next对象元素
    }
    // 如果下一个元素已经是高亮了，则设置轮播标记为false
    if ($next.hasClass('active')) return this.sliding = false

    // 设定轮播后要触发的事件，以及要暴露的参数
    var e = $.Event('slide.bs.carousel', { relatedTarget: $next[0], dire
    this.$element.trigger(e)                // 触发slide事件
    if (e.isDefaultPrevented()) return  // 如果要轮播的对象已经是active高亮了，直接返回不做处理

    this.sliding = true                     // 标记轮播正在进行
    isCycling && this.pause()               // 如果有间隔，则暂停自动执行

    // 处理小圆圈的高亮状态
    if (this.$indicators.length) {      // 如果有小圆圈指示符
        this.$indicators.find('.active').removeClass('active')
                                                    // 去除原来高亮指示符的active样式
        // 设置一次性slid事件，以便在轮播后执行该事件，从而设置高亮指示符
        this.$element.one('slid.bs.carousel', function () {
                    var  $nextIndicator  =  $(that.$indicators.children()[that.getActiveIndex()])
            // 获取当前高亮图片的索引，按照该索引找到对的指示符
            $nextIndicator && $nextIndicator.addClass('active')
                                                    // 如果找到的话，就添加active样式使其高亮
        })
```

```
    }
// 如果支持动画，并且设置了slide样式（注意，这里不是fade效果）
if ($.support.transition && this.$element.hasClass('slide')) {
         $next.addClass(type)              // 给要轮播的元素添加type类型样式（比如：next、prev）
        $next[0].offsetWidth             // 重绘UI
           $active.addClass(direction)       // 给当前活动的对象添加方法（如left、right）
         $next.addClass(direction)        // 给要轮播的元素添加方法（如left、right）

        // 给当前活动元素绑定一次性动画事件，在该事件回调里执行以下操作
        $active
            .one($.support.transition.end, function () {
                 // 在将要轮播的元素上，删除对应type和方向样式（如next left或者prev right），
                $next.removeClass([type, direction].join(' ')).addClass(
                 // 删除当前活动元素（即将隐藏）上的active样式和方向样式（如left或right）
                $active.removeClass(['active', direction].join(' '))
                that.sliding = false    // 设置轮播状态结束
                 // 然后触发slid事件，这里使用了setTimeout是确保UI刷新线程不被阻塞
                setTimeout(function () { that.$element.trigger('slid.bs.
            })
                           .emulateTransitionEnd($active.css('transition-duration').slice(0, -1) * 1000)
    } else {                                // 如果不支持动画
        $active.removeClass('active')   // 删除当前高亮元素上的active样式
         $next.addClass('active')         // 给要轮播的元素上添加高亮active样式
        this.sliding = false             // 设置轮播状态结束
        this.$element.trigger('slid.bs.carousel')  // 触发slid事件
    }
    isCycling && this.cycle()            // 如果有间隔，则重新开始（间隔后）自动执行
              return   this   //   返回this，以便链式操作（这里的this是data-ride="carousel"容器元素）
}
```

根据上述步骤可以得知，前面几行代码都是处理方向的，然后在执行的时候先判断是否支持动画（以及有否slide样式），如果不支持就直接把原来图片元素上的active删除，在新显示图片上加上active样式即可；而如果支持动画，则明显很复杂。我们来用下面的HTML运行结果来分析，首先B处于active状态，HTML代码如下：

```
<div class="item"><img alt="A图" src="A" /></div>
<div class="item active"><img alt="B图" src="B" /></div>
<div class="item"><img alt="C图" src="C" /></div>
```

在触发slide.bs.carousel事件后，如果不阻止默认工作，就执行如下操作：

```
$next.addClass(type)                    // 给要轮播的元素添加type类型样式（比如：next、prev）
$next[0].offsetWidth                    // 重绘UI
$active.addClass(direction)             // 给当前活动的对象添加方法（如left、right）
$next.addClass(direction)               // 给要轮播的元素添加方法（如left、right）
```

也就是，首先给元素C添加一个next样式，此时是第一次变化。

```
<div class="item"><img alt="A图" src="A" /></div>
<div class="item active"><img alt="B图" src="B" /></div>
<div class="item next"><img alt="C图" src="C" /></div>
```

然后通过调用$next[0].offsetWidth，重绘UI，以便能够生效，再给元素C添加left样式，并且也给原来的图片B也添加一个left样式，此时HTML结构变成如下这样（此时是第二次变化）：

```
<div class="item"><img alt="A图" src="A" /></div>
<div class="item active left"><img alt="B图" src="B" /></div>
<div class="item next left"><img alt="C图" src="C" /></div>
```

此时，我们再来看一下CSS里所定义的相关样式（我们这里只看next和left相关的，如果是反方向的prev和right相关的，请自行查看CSS源码）。

```
// 源码5450行
.carousel-inner &gt; .next,
.carousel-inner &gt; .prev {         /*如果是即将轮播的元素*/
  position: absolute;           /*绝对定位*/
  top: 0;                               /*顶部间距为0*/
  width: 100%;
}
.carousel-inner &gt; .next {         /*如果是下一张图片要显示*/
  left: 100%;   /*则left为100%，表示在当前显示图片的右边，利用上面的overflow:
                          hidden;先隐藏起来*/
}
.carousel-inner &gt; .prev {         /*如果上一张图片即将显示了*/
      left:   -100%;      /*则left为-100%，表示在当前显示图片的左边，利用上面的overflow:
                          hidden;先隐藏起来*/
}
.carousel-inner &gt; .next.left,
.carousel-inner &gt; .prev.right {  /*如果下一张图片即将显示了*/
  left: 0;                              /*则设置左对齐，然后利用动画滚动到左边*/
}
.carousel-inner &gt; .active.left { /*如果滚动方向是向左*/
  left: -100%;  /*则上一张高亮显示的照片向左移动100%的距离，以便隐藏起来*/
}
.carousel-inner &gt; .active.right {         /*如果滚动方向是向右*/
  left: 100%;   /*则上一张高亮显示的照片向右移动100%的距离，以便隐藏起来*/
}
```

在重绘UI线程之前，元素C的left值是100%；再次添加left样式后，left值变成了0，而原来的元素B的left值则变成了-100%，表示已经移出显示区域了。

与此同时，定义一个一次性动画事件，并在600毫秒以后执行。

```
$active.one($.support.transition.end, function () {
      // 给当前活动元素绑定一次性动画事件，在该事件回调里执行如下操作
      $next.removeClass([type, direction].join(‘ ’)).addClass(‘active’)
                //   在将要轮播元素上，删除对应type和方向的样式（如next   left或者
prev right），然后添加active样式
      $active.removeClass([‘active’, direction].join(‘ ’))
      // 删除当前活动元素（即将隐藏）上的active样式和方向样式（如left或right）
      that.sliding = false                    // 设置轮播状态结束
      setTimeout(function () { that.$element.trigger(‘slid’) }, 0)
      // 然后触发slid事件，这里使用了setTimeout是确保UI刷新线程不被阻塞
 }).emulateTransitionEnd(600)
```

在回调函数里，我们可以看出，首先删除了元素C上的next和left样式，添加了active样式；然后删除了元素B上的active和left样式；最后设置轮播状态结束，触发slid事件。其结果HTML结构如下（最终状态）：

```
<div class=“item”><img alt=“A图” src=“A” /></div>
<div class=“item ”><img alt=“B图” src=“B” /></div>
<div class=“item active ”><img alt=“C图” src=“C” /></div>
```

和初始HTML代码相比，其实就是把active样式从B的div上，拖到了C的div上，就这么简单。当然，有的人可能还会问到为什么最后一步触发slid事件用了setTimeout函数。

```
setTimeout(function () { that.$element.trigger(‘slid’) }, 0)
// 然后触发slid事件，这里使用了setTimeout是确保UI刷新线程不被阻塞
```

这主要是因为，在默认情况下，浏览器上执行的JavaScript代码和UI更新是属于同一个现场，所以如果直接触发slid的话，可能会导致阻塞UI更新。UI线程的阻塞很多时候是由于我们要在代码里进行长时间的脚本运算，超过了浏览器限制，导致浏览器失去响应，造成假死的状态。使用setTimeout则表示，暂时放开线程的控制器，以便让UI能够立即开始更新，然后再执行setTimeout内部的代码，从而达到平滑的效果。

Slide方法分析完了，理解以后，大家可能就觉得真不错（或者是：也就这么回事嘛）。是的，正是这么简单的代码才构成了这么炫的特效。

步骤3　jQuery插件定义。jQuery插件的定义和其他插件类似。这里我们需要注意一下，当option传入为数字（number）的时候，是表示直接显示特定的图片，也就是调用该实例的.to(number)方法。源码如下：

```
var old = $.fn.carousel
```

```
//  保留其他库的$.fn.carousel代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.carousel = function (option) {
    return this.each(function () {      // 遍历所有符合规则的元素
        var $this = $(this)             // 当前触发元素的jQuery对象
        var data = $this.data('bs.carousel')
                                                    // 获取自定义属性data-bs.carousel的值 （其实是
                                                // carousel实例）

        // 合并参数，优先级依次递增
       // var options = $.extend({}, Carousel.DEFAULTS, $this.data(), ty
            // 'object' && option)
        // 如果option参数是字符串，直接使用，否则使用options里的slide参数
        var action = typeof option == 'string' ? option : options.slide
        // 如果没有carousel实例，就初始化一个，并传入this和参数
        if (!data) $this.data('bs.carousel', (data = new Carousel(this,

        // 如果option是数字，表示是想直接切换到某张图上，所以直接使用.to()方法
        if (typeof option == 'number') data.to(option)
          else if (action) data[action]()           //     否则，再判断如果action存在，就执行action
                                                // 所对应的方法
        else if (options.interval) data.pause().cycle()
                                                    // 最后，如果指定了interval参数，先暂停然后重新循环
    })
}
$.fn.carousel.Constructor = Carousel    // 并重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。该步骤的触发代码比较复杂，主要是和其他的插件有所不同，该插件在触发元素上绑定事件，但并没有将触发元素作为Carousel类的实例。具体如下：

```
// 绑定触发事件
// 在带有data-slide或data-slide-to属性的元素上绑定事件
$(document).on('click.bs.carousel.data-api',    '[data-slide],    [data-slide-to]', function (e) {
    var $this = $(this), href

      // 查找target，即所指定的折叠地区的id或者选择符，如果没有target，就使用href里的值
                        var      $target      =      $($this.attr('data-target') || (href = $this.attr('href')) &&
        href.replace(/.*(?=#[^\s]+$)/, ''))             // strip for ie7

    // 合并target上的data-属性和触发元素上的data-属性
    var options = $.extend({}, $target.data(), $this.data())

    // 查找单击元素上是否有data-slide-to属性
     // 如果存在，则取消间隔设置（因为单击data-slide-to意味着是手动触发行为，后续
```

```
是不会循环播放的）
    var slideIndex = $this.attr('data-slide-to')
    if (slideIndex) options.interval = false

    $target.carousel(options)                                    // 实例化插件

    // 再次判断如果单击的是小圆圈data-slide-to,
    // 则直接跳转到那张图上（$target.data('bs.carousel')是绑定的插件实例）
    if (slideIndex = $this.attr('data-slide-to')) {
        $target.data('bs.carousel').to(slideIndex)
    }
    e.preventDefault()                                           // 阻止默认行为
})

$(window).on('load', function () {
    $('[data-ride="carousel"]').each(function () {       // 遍历所有符合规
则的元素
        var $carousel = $(this)
          $carousel.carousel($carousel.data())// 实例化插件（并收集data-参
数），以便自动运行
    })
})
```

通过上述代码可以看到，该触发代码有两部分。第一部分是对左右控制链接和圆圈指示符（也就是带data-slide、data-slide-to属性的元素）绑定click事件；并且判断如果单击了data-slide-to，则直接显示特定的图片。这里有一点需要注意的是：一旦选择了特定图片，options.interval参数就会被设置为false，也就是不自动轮播，但这取决于初始的轮播是否设置了interval参数，如果设置的话，则还是会自动轮播；如果没有设置，在显示指定的图片之后就会停止。第二部分是通用的插件初始化代码，就不多说了。

5.12　自动定位浮标

源文件：affix.js、utilities.less

CSS文件：bootstrap.css 5676行之后

自动定位浮标（Affix）这个插件名字，有多种译法，图灵出版社的李先生译为粘条，还有人称作固定定位。为了形象，我将它命名为自动定位浮标，但在段落里引用的时候依然叫Affix原有词。Affix的效果就像其官方网站左边的导航链接一样（注意不是菜单的高亮和自动展示功能），功能如下：

❑在顶部的时候该菜单和第二部分的蓝色部分可以一起向上滚动。

❑一旦蓝色部分滚动完毕，该菜单导航就固定在左边不动了，其效果就像浮标一样。

其实还有第三种效果，当整个网页都滚动到底的时候，如果出现底部的footer和左侧菜单重叠的情况，该菜单又会自动向上滚动。例外一个拥有上述3个效果的导航是苹果在线购物商城的购物车摘要描述，如图5-21所示。


顶部显示（设置 offsetTop）
中间显示
底部显示（设置 offsetBottom）
新标签页
<Ctrl+K>
Developing Backbo...
我的帐户
购物车
企业版 Mac
摘要
RMB 42,433 包含增值税
分期每月最低约RMB 1,884
更多信息
预计发货时间：
1-3 个工作日
免费送货
添加到购物车
提供礼品包装
欢迎随时提问
400-666-8800
规格
2.9GHz 双核 Intel Core i7，Turbo Boost 高达 3.6GHz
8GB 1600MHz DDR3 SDRAM - 2x4GB
512GB 固态硬盘
SuperDrive 8 倍速光驱 (DVD±R DL/DVD±RW/CD-RW)
背光键盘 (简体) 与使用手册 (中文)
MacBook Air/13 英寸
MacBook Pro 的 AppleCare
显示更多
选择国家/地区

图5-21　苹果在线购物商城订购页面的自动定位效果

比如，网页刚打开的时候，订购了一台Mac book以后，导航里的“摘要”方框与上面的“企业版Mac”图标基本上是在一起的，页面向上滚动，等“企业版Mac”图标消失以后，该“摘要”方框就悬停不动了；这时候继续向上滚动，一旦底部的footer链接出来以后，和“规格”方框（底部是显示更多链接）快要重合的时候，“规格”方框以及上面的“摘要”方框又可以一起向上滚动了，目的是不影响底部footer的向上滚动（不然两者都重叠在一起）。这种效果一般用于一块比较重要的内容（比如导航、购物车、比价网的商品选择等），其目的是让用户在浏览整个网页的时候，总能看到这一块重要的内容。

在3个不同的状态下，Affix插件给该元素3个不同的CSS样式，分别是affix-top、affix、affix-bottom。根据场景，可猜出3个样式的主要设置如下：

❑affix-top是在正常文档流中的状态定义时的状态。

❑affix是以fixed方式定位时的状态。

❑affix-bottom是以absolute方式定位时的状态。

3个样式中只有affix给了默认的定义，其他两个样式需要我们在开发的时候自己实现。在弄明白这些之前，先来了解一下声明式用法，然后再来讲一下实现原理。

5.12.1 声明式用法

Affix插件在声明式用法上只需要设置两种类型的参数即可。示例如下：

```
<div data-spy=“affix” data-offset=“60” >导航内容</div>
<!–或者分开设置offset – >
<div data-spy=“affix” data-offset-top=“60” data-offset-bottom=“100” >导航内容</div>
```

第一个参数data-spy=“affix”是固定不变的；第二个参数offset有两种方式，一种是直接设置data-offset，表示top和bottom都设置100像素，另一种是data-offset-top，这样的方式则可以分别设置。那offset又是什么作用呢？

data-offset-top表示一个完整的新网页，从顶部向下拖动滚动条（也就是网页向上滚动）这个像素以后，affix元素就不再滚动了（固定不动）；而data-offset-bottom则表示距离最底部还有多少距离（本例60像素）以后，就要开始继续滚动了，以免和底部重叠。

为了更能直观地描述该场景，再来看一下示意图，如图5-22所示。

图5-22中有3个状态，第一张图是Affix随着顶部一起滚动的状态，第二张是Affix固定在页面中间的状态，第三张是底部的bottom快要和Affix重叠在一起的状态（此时Affix就会随着Bottom一起向上滚动）。为了更好地解释，同样一个元素在3个状态下分别叫Top1、Top2、Top3。

1）新页面打开以后，Top1显示，但是Affix1没显示全（底部还留一点），所以这时候开始滚动，在进入第2种状态下，默认Affix元素上的样式为affix-top。

2）在中间状态的时候，Top2已经全部滚动出浏览器界面了，这时候Affix2全部进入了视野，而且下面还留了一点白色间距（通过浏览器高度比较可以看到），而这时候Main2的区域还没有完全滚动上去，所以Bottom2也还没有进入浏览器的视野。整个滚动过程中，Affix元素上的样式为affix。

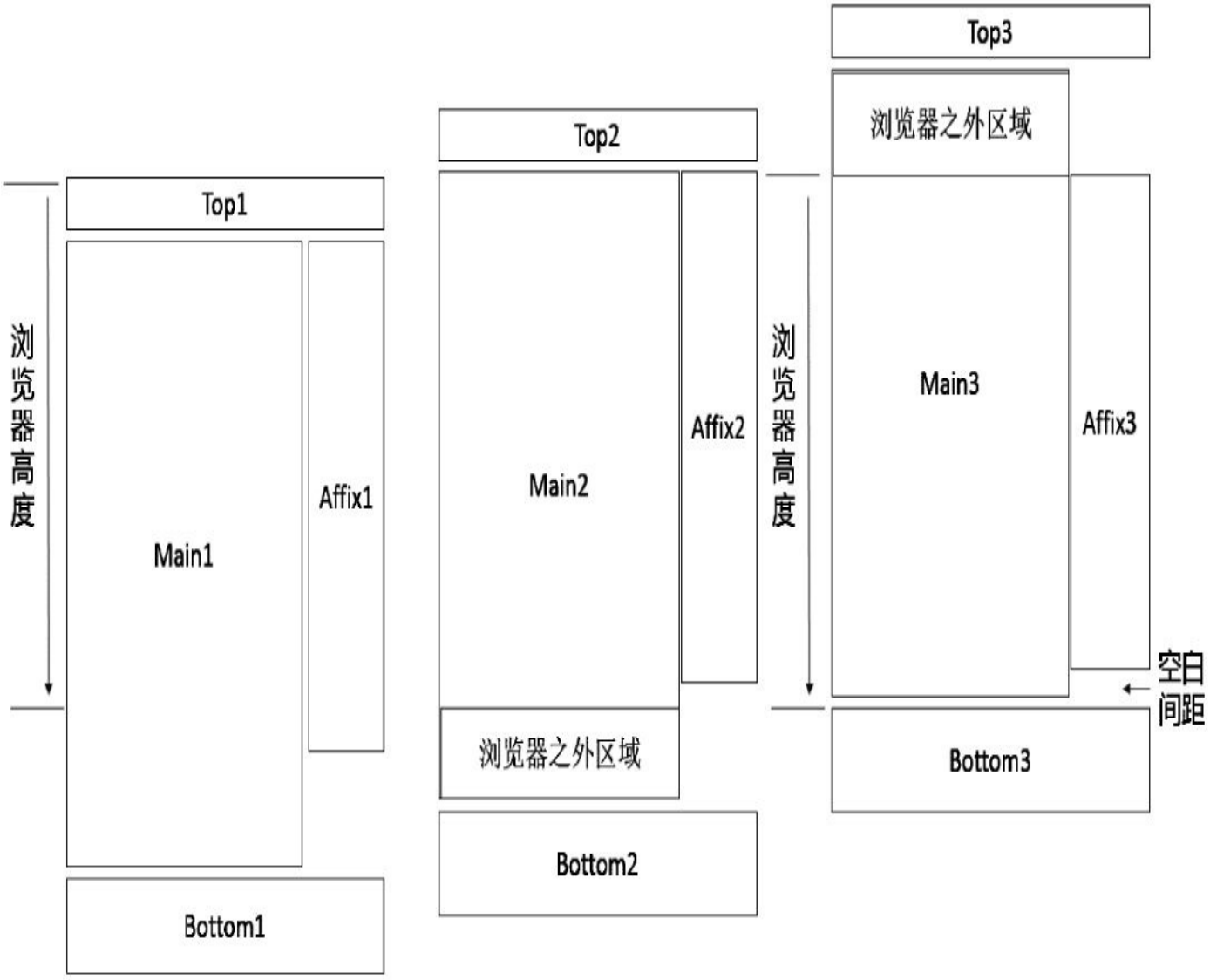


图5-22　自动定位浮标示意图

3）继续滚动，Main3区域的底部完全进入视野了（此时Main3区域的顶部已经滚动出视野了），但Bottom3还没有进入视野，这时候Affix3和Bottom3还有一点距离，我们称之为“空白间距”。然后，继续滚动，在Bottom3和Affix3的边框重合的一瞬间，Affix3就又可以开始滚动了。此时，Affix元素上的样式为affix-bottom。

通过效果，我们可以看到，offset-top的值是从开始移动top，到affix固定不动，期间所移动的距离，本例中也就是top元素的高度60px。而offset-bottom的值，则比较特殊，本例中，我们设置的是Bottom3的高度100px，但是实际上，在Bottom3开始进入之后，要等Bottom3和Affix3重叠时才开始移动。也就是说，虽然你设置了100px，我可以在100px进入视野时开始检测（但未必移动Affix3），还得看你是否和我重叠了，如果还有一段距离，就继续，一直到快要重叠了，我才会让Affix3开始向上移动。

所以结论是：

❑对于offset-top，想在距离顶部多少的地方固定，那就设置多少（注意内外边距）。

❑对于offset-bottom，需要计算affix即将滚动时，其底部到整个页面的底部之间的距离，也就是bottom3的高度+空白高度（其实一般设置为bottom3的高度也没问题）。

所以，使用该插件，除了需要affix-top和affix-bottom样式以外，还需要精确设置这两个offset值，这样才能达到平滑效果，不然滚动的时候会一卡一卡的。另外还需要注意margin-top、margin-bottom、padding-top以及padding-bottom的值，因为这些值也是占用屏幕空间的，所以也会参与到计算中。

5.12.2 JavaScript用法

有的时候，使用该插件，其顶部和底部的高度不一定是固定的，所以在初始化的时候使用声明式用法就不太合适了。作者对Affix插件提供的JavaScript用法就显得更为灵活了，因为它不仅支持传入数字型的offset，还支持传入能够动态计算offset的function函数。示例如下：

```
$('#myAffix').affix({
    offset: {
        top: 100,
        bottom: function () {
            return (this.bottom = $('.bs-footer').outerHeight(true))
        }
    }
})
```

这样，如果底部的footer元素有动态变化的情况，通过给bottom设置function动态计算监控就可以了。

总结一下offset的设置规则如下：

❑如果只设置一个数字，则表示top和bottom都使用该数字。例如：offset:100。

❑如果设置了一个object，则表示分别进行设置。如：offset: { top: 10 }或者offset: { top: 10, bottom: 5 }。

❑如果设置了function，则表示要动态计算。就像上面的示例一样。

Affix组件自V3.1.0开始提供了6种事件，即affix和affixed各对应于3种状态（普通、top、buttom）时的事件，如表5-22所示。

表 5-22 自动定位浮标的事件类型

事件名称	描 述
affix.bs.affix	此事件在定位结束之前立即触发
affixed.bs.affix	此事件在定位结束之后立即触发
affix-top.bs.affix	此事件在定位元素应用 affixed-top 效果之前立即触发
affixed-top.bs.affix	此事件在定位元素应用 affixed-top 效果之后立即触发
affix-bottom.bs.affix	此事件在定位元素应用 affixed-bottom 效果之前立即触发
affixed-bottom.bs.affix	此事件在定位元素应用 affixed-bottom 效果之后立即触发

调用方式也很简单，和普通的jQuery代码并无二样。

5.12.3 源码分析

步骤1 立即调用的函数。此步骤与Modal插件的步骤1一样，此处不赘述。

步骤2 插件核心代码。通过对Affix插件的实现原理分析，结合其他一些插件的源码分析，我们可以看出，该插件在其他地方大同小异，唯一有难度的就是如何做到实时监控滚动，以及如何实时计算设置的offset值，以便对affix-top、affix以及affix-bottom样式进行及时的切换。只有及时切换，才能控制相应的滚动/固定状态。先来看一下摘要代码：

```
// 定义Affix类
var Affix = function (element, options) {
    this.options = $.extend({}, Affix.DEFAULTS, options)         // 合并参数，options优先级高于默认值
    this.$window = $(window)                 // 顶级对象window上监控scroll和click事件
        // scroll事件发生时，调用checkPosition方法
                                                .on('scroll.bs.affix.data-api', $.proxy(this.checkPosition, this))
        // click事件发生时，调用checkPositionWithEventLoop方法
                                                 .on('click.bs.affix.data-api', $.proxy(this.checkPositionWithEventLoop, this))

    this.$element = $(element)                                  // 要固定粘住的元素
    this.affixed =
    this.unpin =
    this.pinnedOffset = null

    this.checkPosition()                                        // 默认调用一次，初始化一下位置
}

Affix.RESET = 'affix affix-top affix-bottom'
Affix.DEFAULTS = {    offset: 0 }
// 获取固定定位元素的offset
Affix.prototype.getPinnedOffset = function () {}
// click事件时，调用此方法调整位置
Affix.prototype.checkPositionWithEventLoop = function () {}
// 重新计算位置的方法
Affix.prototype.checkPosition = function (){};
```

通过上面的代码可见，checkPositionWithEventLoop原型方法最终还是调用了checkPosition原型方法，所以checkPosition方法是最核心的内容，所有计算高度margin-top、padding-top的代码都在该函数里。代码和注释如下：

```
// 获取固定定位元素的offset
Affix.prototype.getPinnedOffset = function () {
    if (this.pinnedOffset) return this.pinnedOffset
```

```
this.$element.removeClass(Affix.RESET).addClass('affix')
var scrollTop = this.$window.scrollTop()
    var position = this.$element.offset()
    return (this.pinnedOffset = position.top - scrollTop)
}
// click事件时，调用此方法调整位置
Affix.prototype.checkPositionWithEventLoop = function () {
            //   使用setTimeout的目的，是让事件循环都处理结束（1毫秒）后，才调用
checkPosition
    setTimeout($.proxy(this.checkPosition, this), 1)
}
// 重新计算位置的方法
Affix.prototype.checkPosition = function () {
    if (!this.$element.is(':visible')) return           // 如果元素不可见
的话，直接返回

    var scrollHeight = $(document).height()             // 整个文档的高度
    var scrollTop = this.$window.scrollTop()    // 窗口向上滚动的偏移量（单
位像素）
    var position = this.$element.offset()       // 返回该元素相对滚动条顶部
的偏移量（单位像素）
    var offset = this.options.offset            // 默认的偏移量设置
    var offsetTop = offset.top                  // 顶部top的偏移量设置
    var offsetBottom = offset.bottom            // 底部bottom的偏移量设置

    // 判断如果affix形式是top，则将scrollTop加到原来的top上
    if (this.affixed == 'top') position.top += scrollTop

    // 因为offset支持不同的方式传值，所以需要判断它是数字还是对象或函数
       //  如果offset不是对象，则表明是一个数字，则将offset赋值于offsetBottom和
offsetTop
    if (typeof offset != 'object') offsetBottom = offsetTop = offset
    // 如果offsetTop是函数，就将其执行结果赋值给offsetTop
    if (typeof offsetTop == 'function') offsetTop = offset.top(this.$ele
    // 如果offsetBottom是函数，就将其执行结果赋值给offsetBottom
if (typeof offsetBottom == 'function') offsetBottom = offset.bottom(this

    // 计算affix当前应该属于什么状态？top、正常、bottom（如果看不明白，下面有改造
后的if/else代码）
    var affix = this.unpin != null && (scrollTop + this.unpin &lt;= posi
                offsetBottom != null && (position.top + this.$element.he
                    scrollHeight - offsetBottom) ? 'bottom' :
                offsetTop != null && (scrollTop &lt;= offsetTop) ? 'top'

    if (this.affixed === affix) return  // 如果原来的状态和现在计算的状态一致
的话，就不需要处理了
    if (this.unpin) this.$element.css('top', '')        // 如果为unpin，就
清空top值

    var affixType = 'affix' + (affix ? '-' + affix : '')        // 判断
affix类型
    var e = $.Event(affixType + '.bs.affix')    // 设置要触发的affix事件

    this.$element.trigger(e)                    // 触发affix事件

    if (e.isDefaultPrevented()) return
```

```
this.affixed = affix                                    // 将最新的affix状态赋值给
affixed
    // 如果是bottom模式，则通过getPinnedOffset获取
    this.unpin = affix == 'bottom' ? this.getPinnedOffset() : null
    this.$element
         .removeClass(Affix.RESET)                                  // 删除所有的
affix样式
        .addClass(affixType)                              // 再添加最新的样式，如果
affix模式不为空，
                                                                    // 则添加两个样
式，如affix或affix-bottom
        .trigger($.Event(affixType.replace('affix', 'affixed')))
                                                                  // 根据类型，触发
相应的affixed事件

    if (affix == 'bottom') {     // 如果是bottom模式，则重新设置元素offset里
的top值
        this.$element.offset({ top: scrollHeight - offsetBottom - this.
            $element.height() })
    }
}
```

上述代码中，最复杂的就是连续嵌套的三目表达式。为了更容易理解，改成if/else代码再来看一下：

```
var affix;
//  如果unpin不为空，计算（屏幕滚动的高度+unpin），如果其和小于affix元素的top值，
则表示不需
 要固定位置
if (this.unpin != null && (scrollTop + this.unpin <= position.top)) {
    affix = false;
}
else {
        //  如果offsetBottom不为空，并且（元素的top值+元素的高度）>=（滚动高度-
offsetBottom）
    if (offsetBottom != null && (position.top + this.$element.height() >
        scrollHeight - offsetBottom)) {
        // 则表示affix模式为bottom
        affix = "bottom";
} else {
         // 如果offsetTop不为空，如果（滚动高度）<=（设置的offsetTop），则表示
affix模式为top
        // （正常模式）
        if (offsetTop != null && (scrollTop <= offsetTop)) {
             affix = "top";
        }
        else {
            affix = false;
        }
    }
}
```

这样看起来是不是好理解一点？如果还不理解，请结合前面的原理示意图（图5-22），再理解一遍。看的时候最好也写代码自己练习一

下。

步骤3　jQuery插件定义。Affix插件在jQuery上的定义和其他插件没有什么不同。

```
// 在jQuery上定义affix插件，并重设插件构造器
var old = $.fn.affix
// 保留其他库的$.fn.affix代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.affix = function (option) {
    return this.each(function () {      // 遍历所有符合规则的元素
        var $this = $(this)             // 当前触发元素的jQuery对象
          var data = $this.data('bs.affix')         // 获取自定义属性data-bs.affix的值（其实是affix实例）
        var options = typeof option == 'object' && option  // 合并参数

        // 如果没有Affix实例，就初始化一个，并传入this和参数
        if (!data) $this.data('bs.affix', (data = new Affix(this, optior

        // 如果option是字符串，则表示直接调用该实例上的同名方法
        if (typeof option == 'string') data[option]()
    })
}
$.fn.affix.Constructor = Affix // 并重设插件构造器，可以通过该属性获取插件的真实类函数
```

步骤4　防冲突处理。此步骤与Modal插件的步骤4一样，此处不赘述。

步骤5　绑定触发事件。绑定触发事件的源码如下：

```
// 绑定触发事件
$(window).on('load', function () {
    $('[data-spy="affix"]').each(function () {  // 遍历所有符合规则的元素
        var $spy = $(this)              // 临时赋值变量
        var data = $spy.data()          // 收集该元素上的自定义属性（data-开头）

         data.offset = data.offset || {} // 如果设置了offset就使用它，否则传一个默认空值

                        //    如果设置了data-offset-bottom属性，则将它的值赋给data.offset.bottom
        if (data.offsetBottom) data.offset.bottom = data.offsetBottom

        // 如果设置了data-offset-top属性，则将它的值赋给data.offset.top
        if (data.offsetTop) data.offset.top = data.offsetTop
        $spy.affix(data)                        // 实例化插件（并收集data-参数），以便自动运行
    })
})
```

通过上述代码可以看到，上述代码offset做了特殊处理，即先检测自定义属性data-offset，临时保存一下；然后再判断有没有data-offset-

top（或data-offset-bottom），如果有，就使用它，如果没有，就使用普通的offset。也就是说了，如果同时声明了data-offset=100和data-offset-top=60，最终的结果就是：top用60，bottom用100。

第6章　实战：扩展现有组件

本章将对Bootstrap现有的组件（分页组件、modal弹窗组件）进行扩展方面的实战，这些内容都是在平时开发过程中非常常用的场景。熟悉现有组件的扩展步骤以后，我们就可以非常简单地扩展其他组件了。

6.1　扩展分页组件

网页列表是每个Web网站不可或缺的一部分，而分页又是列表不可或缺的一部分。但这些内容根据不同风格的网站，却又不太一样，有的是布局不一样。有的是颜色不一样。本小节，我们将在现有分页组件的基础之上进行扩展。

6.1.1 形状扩展

有些特殊的情况下，我们可能不太需要Bootstrap默认提供的风格。比如我们想让所有的页码都放在单独的方格里，显示出如图6-1所示的分页效果，该如何做呢？

图6-1 设置的分页效果

通过默认分页的源码分析我们可以知道，如果要单个元素都要分开显示，则最简单的方式就是单个元素的边框都设置为1px，然后每个元素之间设置一定像素的margin值。源码如下：

```
.pagination > li > a,
.pagination > li > span {
    margin: 0 5px;
    border: 1px solid #dddddd;          /* 设置所有的边框都为1像素 */
}
```

然后再去除两边的圆角设置即可。由于默认分页单独通过first-child和last-child特性对两边的圆角进行了设置，所以需要重设圆角的值为0。源码如下：

```
.pagination > li:first-child > a,
.pagination > li:first-child > span,
.pagination > li:last-child > a,
.pagination > li:last-child > span {
    border-radius: 0px;                 /* 不管是第一个还是最后一个元素，都禁用圆角 */
}
```

在默认的Bootstrap.css之后应用上述两段样式可将分页效果设置成如图6-1所示的效果，从而达到我们的要求。

在自定义样式的时候，为了避免覆盖Bootstrap默认的样式或行为，建议通过附加样式的形式来实现。比如上述示例，可以额外添加一个square样式。源码和使用示例如下：

```
.pagination.square > li > a,
.pagination.square > li > span {
    margin: 0 5px;
    border: 1px solid #dddddd;          /* 设置所有的边框都为1像素 */
}

.pagination.square > li:first-child > a,
.pagination.square > li:first-child > span,
.pagination.square > li:last-child > a,
.pagination.square > li:last-child > span {
```

```
    padding-left: 10px;
    padding-right: 10px;
    border-radius: 0px;                    /* 取消圆角 */
}
```

上述用法，也利用了Bootstrap在CSS方面的设计思想，再多应用一个square样式即可。

```
<ul class=“pagination square”>
…
</ul>
```

注意

❑同理，如果想让pager里的两个链接变成长方形，则只需把圆角角度设置为0px即可。

❑同样的道理，如果要想每个页码都设置成圆形，则只需将所有li元素圆角的border-radius值设置大一些就可以了，比如15px。

6.1.2　颜色扩展

分页组件默认是没有颜色效果设置的，在大多数情况下，都要对它进行改造，以便该组件适应我们设计的页面风格。本节就来扩展一下它的颜色样式。先看一下具体效果，如图6-2所示。

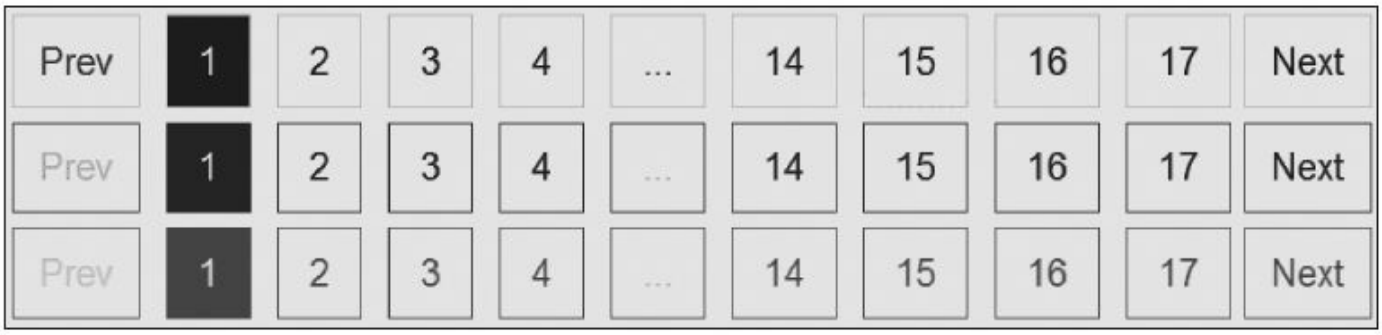


图6-2　不同颜色的分页样式

要更改颜色首先要注意以下几个方面：

❑li元素的边框颜色

❑li元素内的链接文本颜色

❑li元素在active高亮时的背景色和文本颜色

❑li元素在disabled禁用时的背景色和文本颜色

知道了这几点，我们就可以开始设置一个红色（red）风格的分页样式了。首先red样式必须和.pagination样式一起使用才能生效。相对应的详细代码如下：

```
/* 所有li元素内的a链接和span边框和文本颜色设置 */
.pagination.red > li > a,
.pagination.red > li > span {
    color: #f00;
    border: 1px solid #f00;
}

/* a链接和span在鼠标移动或焦点时的背景色设置 */
.pagination.red > li > a:hover,
.pagination.red > li > span:hover,
.pagination.red > li > a:focus,
.pagination.red > li > span:focus {
    background-color: #fdd5d5;
}

/* active高亮时，设置文本颜色、背景色、边框颜色 */
.pagination.red > .active > a,
.pagination.red > .active > span,
.pagination.red > .active > a:hover,
.pagination.red > .active > span:hover,
.pagination.red > .active > a:focus,
```

```
.pagination.red > .active > span:focus {
    color: #ffffff;
    background-color: #f00;
    border-color: #f00;
}

/*disabled禁用时，设置文本颜色、背景色、边框颜色 */
.pagination.red > .disabled > span,
.pagination.red > .disabled > span:hover,
.pagination.red > .disabled > span:focus,
.pagination.red > .disabled > a,
.pagination.red > .disabled > a:hover,
.pagination.red > .disabled > a:focus {
    color: #fdd5d5;
    background-color: #ffffff;
    border-color: #f00;
}
```

这样，通过如下样式，就可以使用该样式了。同样的道理，也可以扩展出其他各种各样的颜色样式。

```
<ul class=“pagination red”>
…
</ul>
```

6.2　扩展Modal弹窗

Modal插件的模态弹窗是个常用的插件，但是实际工作中，它可能还不能完全满足我们的需求。一般来说我们经常使用两种弹窗：一种是Info弹窗（信息确认框，带有“确认”和“取消”按钮），一种是Form弹窗（表单提交框，但“提交”按钮通常是利用Ajax技术提交）。本节，我们将在现有Modal插件的基础上，对这两种新需求进行扩展。

6.2.1 扩展点探讨

在扩展插件之前，再仔细回顾一下Bootstrap所有的JS插件的开发步骤，总共分5步，分别是：定义立即调用的函数表达式、声明组件类函数及原型方法、jQuery的fn插件定义、防冲突处理、DATA-API事件绑定。

利用现有的组件进行扩展，通常意味着重用现有的代码。在JavaScript开发过程中，重用代码一般都是利用原型继承来实现的。关于原型继承，第1章我们已经简单介绍过，如果想进一步了解原型继承，请访问我的以下两篇博客：

❑强大的原型和原型链：
http://www.cnblogs.com/TomXu/archive/2012/01/05/2305453.html

❑代码复用模式：
http://www.cnblogs.com/TomXu/archive/2012/04/24/2438050.html

了解了原型继承的原理，正当我们跃跃欲试时，却发现利用这种思路扩展Modal插件根本不可行。为什么？就是因为所有的Bootstrap JavaScript插件都是利用立即调用的函数表达式来定义的，所有的组件类和原型方法都在局部作用域里，这就使得我们无法利用现有的Modal类和相关的原型方法。

重新回顾JavaScript插件的架构设计，不管是声明式还是JavaScript调用，所有的对外接口都是通过jQuery来触发的，所以如果想扩展Modal插件，只能在jQuery上找突破口了。

在前面章节对Modal插件进行源码分析时，看到过如下代码：

```
var  data  =  $this.data('bs.modal')  //  获取自定义属性bs.modal的值，也就是Modal插件的实例
var options = $.extend({}, Modal.DEFAULTS, $this.data(), typeof option =
if (!data) $this.data('bs.modal', (data = new Modal(this, options)))
```

我们讲过，基本上所有的插件在实例化之后，都会将该实例绑定到模板元素的一个自定义属性上，本例中是通过$.data('bs.modal',实例对象）的方式进行设置的，所以在将插件实例化之前，首先要获取该属性实例data对象，如果存在就直接用，如果不存在，就实例化一个新的（紧接着就保存到自定义属性上以便缓存，下次使用）。这就是我们的突破口，也就是说，只要能够知道目标元素，就能获取它的实例。

另外一个扩展点是，原有Modal插件所暴露出来的自定义事件回调（如：show、shown、hide、hidden）。通过结合这两大突破口就可以实现我们在本节所说的Info弹窗和Form弹窗。

另外，所谓扩展，就意味着不能影响原来的功能，也就是说不能影

响原来的自定义事件回调、即不能影响原有实例上的原型方法、声明式和JavaScript的用法。

6.2.2 Info弹窗扩展

首先，我们先来明确一下UI的设计。Info弹窗和普通弹窗的唯一不同点是：Info弹窗里有“确认”和“取消”按钮，分别对应确认和取消操作。相应的HTML部分如下：

```
<div class=“modal-footer”>
    <button type=“button” class=“btn btn-success”>确认</button>
    <button type=“button” class=“btn btn-danger”>取消</button>
</div>
```

在普通的Modal弹窗里，如果我们要实现Info弹窗效果，就需要在“确认”和“取消”按钮上分别绑定事件，而且如果Modal内容是从远程（remote）获取的，那只能在shown.bs.modal的回调里绑定上述两个按钮的click事件，然后再处理相应的逻辑。所以，这也就意味着，要为这两个按钮分别添加id属性。

通过对data-dismiss=“modal”的了解可以看出，只需要以类似的方式，给按钮声明一个自定义属性，然后在新的Info弹窗里处理上述两个按钮的单击行为就可以了，然后再暴露两个事件回调出来。自定义属性的修改如下：

```
<div class=“modal-footer”>
     <button type=“button” class=“btn btn-success” data-info=“confirm”>
确认</button>
    <button type=“button” class=“btn btn-danger” data-info=“cancel”>取消
</button>
</div>
```

添加一个data-info按钮，在弹窗显示的时候，绑定事件，然后执行操作（处理回调），整个思路就是这样的。也就是说，声明式用法里只多了两个自定义属性，其他的和Modal插件一模一样，我们就一步一步地在Modal插件基础上扩展吧。

首先，我们将自己新扩展的插件取名为InfoModal，所以要先定义整个插件类函数：

```
// InfoModal类定义
var InfoModal = function (element, options) {
    this.$element = $(element);
        this.super  =  this.$element.data(‘bs.modal’);  //  获取自定义属性
bs.modal的值
    this.options = options;

                  this.$element.on(‘click.confirm.infomodal’,    ’[data-
info=“confirm”]’,
        $.proxy(this.confirm, this));
    this.$element.on(‘click.cancel.infomodal’, ’[data-info=“cancel”]’,
        $.proxy(this.cancel, this));
```

```
    var that = this;                                    // 防止污染作用域，用临
时变量that
}

// 默认设置
InfoModal.DEFAULTS = {
    closeAfterConfirm: false,
    closeAfterCancel: true
}
```

上述代码和Modal类似，但有以下几个地方不一样：

1）多了一个this.super对象，它的值指向前面所说的Modal弹窗的示例，以便可以使用该实例上的一些原型方法。

2）在modal元素内部的子元素（带有data-info属性）上分别绑定了confirm和cancel方法，以便在单击的时候调用它们。

3）还定义了两个默认值，分别设置在单击完“确认”和“取消”按钮后，使用的过程中可以随意设置要不要自动关闭，也可以不关闭（利用回调函数手动进行关闭）。

基本的类定义完了，开始定义原型方法和自定义事件。要思考到底要定义多少原型方法：首先confirm和cancel肯定得要，其次show、hide和toggle这3个也应该要。而对于自定义事件，首先要有和Modal弹窗一样的show、shown、hide、hidden事件，另外，又添加了confirm.bs.infomodal和cancel.bs.infomodal事件。

先来定义原型方法：

```
// 反转弹窗状态
InfoModal.prototype.toggle = function (_relatedTarget) {
    // 如果是关闭状态，则打开弹窗，否则关闭
    return this[!this.super.isShown ? ‘show’ : ‘hide’](_relatedTarget)
}
// 打开弹窗
InfoModal.prototype.show = function (_relatedTarget) {
    this.super.show(_relatedTarget);
}
// 关闭弹窗
InfoModal.prototype.hide = function (e) {
    if (e) e.preventDefault();                          // 先阻止冒泡行为
    this.super.hide();
}
// 单击“确认”按钮的行为
InfoModal.prototype.confirm = function (e) {
    if (e) e.preventDefault();  // 先阻止冒泡行为

    var e = $.Event(‘confirm.bs.infomodal’);
    this.$element.trigger(e);   // 确认前触发事件，主要用于处理相关代码

    if (e.isDefaultPrevented()) return;
```

```
    if (this.options.closeAfterConfirm) {
            this.hide();                    //  如果设置了data-close-after-
confirm=true参数,
                                            // 则关闭弹窗
    }
}
// 单击“取消”按钮的行为
InfoModal.prototype.cancel = function (e) {
    if (e) e.preventDefault();  // 先阻止冒泡行为

    var e = $.Event('cancel.bs.infomodal');
    this.$element.trigger(e);   // 取消前先触发事件

    if (e.isDefaultPrevented()) return;
    if (this.options.closeAfterCancel) {
        this.hide(e);           // 如果设置了data-close-after-cancel=true
参数,
                                            // 则关闭弹窗
    }
}
```

上述几个原型方法，首先是利用modal实例（this.super）里已有的方法，然后在confirm和cancel里分别触发了所对应的自定义事件（其实两个方法里也没处理什么逻辑，主要就是触发事件）。

笔者在这里是通过绑定modal现有的事件来触发这些新的事件。代码的位置放置在InfoModal类里的最底部。代码如下：

```
var that = this;                    // 防止污染作用域，用临时变量that

this.$element.on("show.bs.modal", function (e) {
    that.$element.trigger(e = $.Event('show.bs.infomodal'));
    if (e.isDefaultPrevented()) return;
});

this.$element.on("shown.bs.modal", function (e) {
    that.$element.trigger(e = $.Event('shown.bs.infomodal'));
    if (e.isDefaultPrevented()) return;
});

this.$element.on("hide.bs.modal", function (e) {
    that.$element.trigger(e = $.Event('hide.bs.infomodal'));
    if (e.isDefaultPrevented()) return;
});

this.$element.on("hidden.bs.modal", function (e) {
    that.$element.trigger(e = $.Event('hidden.bs.infomodal'));
    if (e.isDefaultPrevented()) return;
});
```

也就是，modal触发自己的事件时，就会利用这些回调触发InfoModal类的事件，而且这样做有个好处：那就是我如果阻塞了事件（调用e.preventDefault()），modal后续就不能再触发了。所以如果分别对这8

个事件进行定义，它们的执行顺序应该如下：

- ❑show.bs.modal
- ❑show.bs.infomodal
- ❑shown.bs.modal
- ❑shown.bs.infomodal
- ❑hide.bs.modal
- ❑hide.bs.infomodal
- ❑hidden.bs.modal
- ❑hidden.bs.infomodal

读者可能会问：不在show和hide方法里，是否也可以触发新的show、shown、hide、hidden自定义事件呢？这是可以的，大家可以自行试验。

另外，由于这种机制是因为对新的事件进行绑定，肯定得先触发Modal的事件，才能触发Infomodal的事件。如果想要改变show.bs.modal和show.bs.infomodal的执行顺序，也必须在新的show和hide原型方法里改变触发顺序的设置，大家可以自行试验一下。

在jQuery.fn.infomodal插件定义的时候，和modal几乎一样，除了要把对应的名字全换成InfoModal以外，唯一需要注意的就是，InfoModal默认应该是关闭的，每次触发时都应该打开，也就是说原来定义的toggle方法基本上用不上。具体代码如下：

```
$.fn.infomodal = function (option, _relatedTarget) {
    return this.each(function () {
        // 根据选择器，遍历所有符合规则的元素

        var $this = $(this)
             var  data  =  $this.data('bs.infomodal')     //  获取自定义属性
bs.infomodal的值
        var options = $.extend({}, InfoModal.DEFAULTS, $this.data(),
            typeof option == 'object' && option)
        // 将默认参数、选择器所在元素的自定义属性（data-开头）和option参数，
        // 这3种的值合并在一起，作为options参数
        // 优先级：后面的参数优先级高于前面的参数
        options.show = false;                          // 默认先关闭，然后在后面手
动打开
        var modal = $this.modal(options, _relatedTarget);

        if (!data) $this.data('bs.infomodal', (data = new InfoModal(this
        // 如果值不存在，则将InfoModal实例设置为bs.infomodal的值
        if (typeof option == 'string') {
            data[option](_relatedTarget)
        }
```

```
        else {
            data.show(_relatedTarget);
        }
        // 如果option传递了string，则表示要执行某个方法
          // 比如传入了show，则要执行InfoModal实例的show方法，data[“show”]相当于data.show()
    })
}
```

其他关于防冲突的和DATA-API的，和modal除了名字不一样，其他都一样（InfoModal是通过[data-toggle=“infomodal”]触发的）。运行效果如图6-3所示。

图6-3　Info弹窗运行效果

使用时的示例代码如下：

```
<div    class=“modal    fade”    id=“DisabledPopup”    data-close-after-confirm=“true”>
    <div class=“modal-dialog”>
        <div class=“modal-content”>
            <div class=“modal-header”>…  </div>
            <div class=“modal-body”>…   </div>
            <div class=“modal-footer”>
                    <button type=“button” class=“btn btn-success” data-info=“confirm”>
                    确认</button>
                     <button type=“button” class=“btn btn-danger” data-info=“cancel”>
                    取消</button>
            </div>
        </div>
    </div>
</div>
<button    class=“btn    btn-warning”    data-toggle=“infomodal”    data-target=”#DisabledPopup”
data-backdrop=“static”>Disable</button>
```

由示例代码也可以看到，我们没有改动原有modal的任何内容，只是添加了一些自定义属性。同样的道理，InfoModal既支持声明式用法，也支持JavaScript用法，具体的实例读者可以自行试验。完整的源代码如下：

```
+function ($) {
    “use strict”;

    // InfoModal类定义
    var InfoModal = function (element, options) {
        this.$element = $(element);
        this.super = this.$element.data(‘bs.modal’);
                                                                    // 获取自定义属性
bs.modal的值
        this.options = options;

                    this.$element.on(‘click.confirm.infomodal’,  ’[data-
info=“confirm”]’,
            $.proxy(this.confirm, this));
                     this.$element.on(‘click.cancel.infomodal’,  ’[data-
info=“cancel”]’,
            $.proxy(this.cancel, this));

         var that = this;                                    // 防止污染作用
域，用临时变量that

        this.$element.on(“show.bs.modal”, function (e) {
            that.$element.trigger(e = $.Event(‘show.bs.infomodal’));
            if (e.isDefaultPrevented()) return;
        });

        this.$element.on(“shown.bs.modal”, function (e) {
            that.$element.trigger(e = $.Event(‘shown.bs.infomodal’));
            if (e.isDefaultPrevented()) return;
        });

        this.$element.on(“hide.bs.modal”, function (e) {
            that.$element.trigger(e = $.Event(‘hide.bs.infomodal’));
            if (e.isDefaultPrevented()) return;
        });

        this.$element.on(“hidden.bs.modal”, function (e) {
            that.$element.trigger(e = $.Event(‘hidden.bs.infomodal’));
            if (e.isDefaultPrevented()) return;
        });
    }

    // 默认设置
    InfoModal.DEFAULTS = {
        closeAfterConfirm: false,
        closeAfterCancel: true
    }
    // 反转弹窗状态
    InfoModal.prototype.toggle = function (_relatedTarget) {
```

```
                return this[!this.super.isShown ? 'show' : 'hide']
(_relatedTarget)
        // 如果是关闭状态，则打开弹窗，否则关闭
    }
    // 打开弹窗
    InfoModal.prototype.show = function (_relatedTarget) {
        this.super.show(_relatedTarget);
    }
    // 关闭弹窗
    InfoModal.prototype.hide = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为
        this.super.hide();
    }
    // 单击“确认”按钮的行为
    InfoModal.prototype.confirm = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为

        var e = $.Event('confirm.bs.infomodal');
        this.$element.trigger(e);       // 确认前触发事件，主要用于处理相关代
码

        if (e.isDefaultPrevented()) return;
        if (this.options.closeAfterConfirm) {
             this.hide(e);                // 如果设置了data-close-after-
confirm=true
                                          // 参数，则关闭弹窗
        }
    }
    // 单击“取消”按钮的行为
    InfoModal.prototype.cancel = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为

        var e = $.Event('cancel.bs.infomodal');
        this.$element.trigger(e);       // 取消前先触发事件

        if (e.isDefaultPrevented()) return;
        if (this.options.closeAfterCancel) {
             this.hide(e);                // 如果设置了data-close-after-
cancel=true
                                          // 参数，则关闭弹窗
        }
    }
    // InfoModal 插件定义
    var old = $.fn.infomodal
    // 如果定义了其他infomodal，则保留它（以便在noConflict（解决防冲突）之后，
    // 可以继续使用该旧插件代码）

    $.fn.infomodal = function (option, _relatedTarget) {
        return this.each(function () {
            // 根据选择器，遍历所有符合规则的元素

            var $this = $(this)
            var data = $this.data('bs.infomodal')
                                                  // 获取自定义属性
bs.infomodal的值
            var options = $.extend({}, InfoModal.DEFAULTS, $this.data(),
```

```
                typeof option == 'object' && option)
            // 将默认参数、选择器所在元素的自定义属性（data-开头）、option参数这3种的值合
            // 并在一起，作为options参数
            // 优先级：后面的参数优先级高于前面的参数
            options.show = false;       // 默认先关闭，然后在后面手动打开
            var modal = $this.modal(options, _relatedTarget);

            if (!data) $this.data('bs.infomodal', (data = new InfoModal(
            // 如果值不存在，则将InfoModal实例设置为bs.infomodal的值
            // 如果option传递了string，则表示要执行某个方法
            // 比如传入了show，则要执行InfoModal实例的show方法，data["show"]相当于data.show();
            if (typeof option == 'string') {
                data[option](_relatedTarget)
            }
            else {
                data.show(_relatedTarget);
            }
        })
    }
    $.fn.infomodal.Constructor = InfoModal;
                                                    // 重设插件构造器，可以通过该属性获取插件的真实类函数

    // InfoModal防冲突
    $.fn.infomodal.noConflict = function () {
        $.fn.infomodal = old
        return this
    }
    // InfoModal DATA-API
$(document).on('click.bs.infomodal.data-api',                    '[data-toggle="infomodal"]',
    function (e) {
    // 监测所有拥有自定义属性data-toggle="modal"的元素上的单击事件
        var $this = $(this)
        var href = $this.attr('href')   // 获取href属性值
        var $target = $($this.attr('data-target') || (href &&
            href.replace(/.*(?=#[^\s]+$)/, '')))
        // 获取data-target属性值，如果没有，则获取href值，该值是所弹出元素的id

        var option = $target.data('infomodal') ? 'toggle' : $.extend
            ({ remote: !/#/.test(href) && href }, $target.data(), $this.
        // 如果弹窗元素上设置了data-infomodal属性值，则option值是字符串toggle
         // 否则将remote值（如果有）、弹窗元素上的自定义属性值集合、触发元素上的自定义属性
        // 值集合进行合并，作为option选项对象

        e.preventDefault()              // 阻止默认行为
        $target
            .infomodal(option, this)
               // 给弹窗元素绑定infomodal插件（也就是实例化InfoModal），并传入option参数
            .one('hide', function () {
                $this.is(':visible') && $this.focus()
                                                    // 定义一次hide事件，给所
```

```
单击元素加上焦点
            })
    })
}(jQuery);
```

6.2.3 Form弹窗扩展

在平时的Web开发中，除了用弹窗提示信息以外，另外一个最常用的方式就是在弹窗里进行表单处理，如添加账户、修改密码等。但一般很少用于处理字段比较多的表单（其实也可以，只不过弹窗里要设置固定高度，以便在内容多的时候呈现滚动条）。本章我们就来看看，在Modal插件的基础上，如何扩展该功能。

弹窗里的表单一般有如下几个特点：

❑添加类表单在弹出时，往往需要表单是空的，如果添加过一次（再关闭弹窗），再次弹出时需要手工清空表单内容。

❑表单内容往往可能是通过ajax加载的html表单代码。

第1个特点带来的问题的解决方案比较简单，直接在shown.bs.formmodal事件上注册一个回调事件，清空表单里的内容即可。

第2个特点带来的问题比较多，我们分别来说一下。

1）默认的Modal插件请求ajax时会有缓存，如果在弹窗里修改多条记录，那每次ajax请求都是第一次的内容。解决方式是通过某种方式强制重新加载。

2）远程利用remote属性加载html内容将会替换掉整个modal-content元素的所有内容（包括modal-header、modal-body、modal-footer），这是与Bootstrap 2.x系统特别不一样的地方。所以ajax请求返回的表单内容单必须是如下格式：

```
<div class=“modal-content”>
    <div class=“modal-header”>
        … 此处是标题提示
    </div>
    <div class=“modal-body”>
        <form class=“form-horizontal” role=“form”>
            … 此处放置表单元素
        </form>
    </div>
    <div class=“modal-footer”>
        … 此处可以放置按钮
    </div>
</div>
```

3）一般表单都有验证，大部分都会使用类似jQuery.validate这样的插件，但这种类型的插件一般都只是在DOM加载结束后，对当前页面所有的form元素进行检测并绑定相应事件。由于弹窗里的表单是后来通过ajax加载的，所以一般都不会起作用。其解决办法就是，在弹窗表单显示以后，手工对弹窗里的form元素绑定验证事件（在shown.bs.formmodal上绑

定事件回调）。

下面，我们就带着这些问题，来一步一步扩展Form弹窗。首先定义FormModal类函数。

```
// FormModal类定义
var FormModal = function (element, options) {
    this.$element = $(element);
    this.super = this.$element.data('bs.modal');         // 获取自定义属性bs.modal的值
    this.options = options;

    this.$element.on('click.submit.formmodal', '[data-form="submit"]',
        $.proxy(this.submit, this));
    this.$element.on('click.reset.formmodal', '[data-form="reset"]',
        $.proxy(this.reset, this));
    this.$element.on('click.cancel.formmodal', '[data-form="cancel"]',
        $.proxy(this.cancel, this));

    var that = this;                                    // 防止污染作用域，用临时变量that

    this.$element.on("show.bs.modal", function (e) {
        that.$element.trigger(e = $.Event('show.bs.formmodal'));
        if (e.isDefaultPrevented()) return;
    });
    this.$element.on("shown.bs.modal", function (e) {
        that.$form = that.$element.find('form');
        that.$element.trigger(e = $.Event('shown.bs.formmodal'));
        if (e.isDefaultPrevented()) return;
    });
    this.$element.on("hide.bs.modal", function (e) {
        that.$element.trigger(e = $.Event('hide.bs.formmodal'));
        if (e.isDefaultPrevented()) return;
});
this.$element.on("hidden.bs.modal", function (e) {
        that.$element.trigger(e = $.Event('hidden.bs.formmodal'));
        if (e.isDefaultPrevented()) return;
    });
}
// 默认设置
FormModal.DEFAULTS = {
    cacheForm: false,                                   // 默认不用缓存，每次都加载最新的内容
    closeAfterCancel: true                              // 默认取消后，直接关闭弹窗
}
```

上述代码和InfoModal类似，但有两个地方不太一样。一个地方是在触发shown.bs.infomodal自定义事件中，多了一句that.$form = that.$element.find('form');代码，该代码是确保在弹窗显示之后，将找到的form元素赋值给一个临时$form变量，以便后面的reset原型方法使用（用于重置表单）；另外一个地方是默认参数新加了一个cacheForm，用于设置（如果是ajax请求时）是否要缓存数据，如果不缓存，则每次请

求时都获取最新的html内容。

所以Form弹窗插件在声明式用法时，可以定义两个参数，分别是data-cache-form和data-close-after-cancel。

我们来看原型方法。由于很多内容也是和InfoModal类似，所以toggle、show、hide、cancel原型方法都是一样的，不一样的是去除了confirm，增加了submit和reset原型方法。具体代码如下：

```
// 单击“确认”按钮的行为
FormModal.prototype.submit = function (e) {
    if (e) e.preventDefault();  // 先阻止冒泡行为

    this.$element.trigger(e = $.Event('beforeSubmit.bs.formmodal'));
                                        // 提交前触发事件，主要用于处理相关代
码
    if (e.isDefaultPrevented()) return;

    this.$form.submit();
    this.$element.trigger(e = $.Event('afterSubmit.bs.formmodal'));
                                        // 提交后触发事件，主要用于处理相关代
码
    if (e.isDefaultPrevented()) return;
}
// 单击“重置”按钮的行为
FormModal.prototype.reset = function (e) {
    if (e) e.preventDefault();  // 先阻止冒泡行为

    var resetAction = function () {
        this.$element.trigger(e = $.Event('beforeReset.bs.formmodal'));
                                        // 重置前触发事件
        if (e.isDefaultPrevented()) return;
        this.$form.each(function () {
            this.reset();       // jQuery不支持reset，需要转换为DOM对象
                                        // 再调用原生reset方法
        });
        this.$element.trigger(e = $.Event('afterReset.bs.formmodal'));
                                        // 重置后触发事件
        if (e.isDefaultPrevented()) return;
    }
if (this.super.isShown) return resetAction.call(this);

    this.$element.one("shown.bs.formmodal", $.proxy(resetAction, this));
    this.show();
}
```

submit原型方法主要就是触发beforeSubmit、afterSubmit事件以及提交表单。而reset原型实现相对比较复杂：首先，jQuery不支持reset方法，所以需要将符合条件的form元素转换为DOM对象；然后调用DOM上的reset方法；在重置的时候，首先要先判断弹窗内容是不是重置了，只有显示了，才能直接重置，否则在shown上注册一个回调（回调内容是reset操作），在显示以后，才执行reset操作，其主要目的是当开发人员在调

用reset方法时，可以直接传入reset参数（如$.formmodal(‘reset’）；而不至于出错。

FormModal暴露的所有事件如表6-1所示。

表 6-1　FormModal 弹窗的事件类型

事件名称	描　述
show.bs.formmodal	在 show 方法调用时立即触发（尚未显示之前）；如果单击了元素，那么该元素将作为事件的 relatedTarget 属性

（续）

事件名称	描　述
shown.bs.formmodal	在模态弹窗完全显示给用户之后（并且 CSS 动画完成之后）触发；如果单击了一个元素，那么该元素将作为事件的 relatedTarget 属性
beforeSubmit.bs.formmodal	提交表单之前触发
afterSubmit.bs.formmodal	提交表单之后触发
beforeReset.bs.formmodal	表单重置之前触发
afterReset.bs.formmodal	表单重置之后触发
cancel.bs.formmodal	表单弹窗取消时触发
hide.bs.formmodal	在调用 hide 方法时（但还未关闭隐藏）立即触发
hidden.bs.formmodal	在模态弹窗完全隐藏（并且 CSS 动画完成）之后触发

jQuery.fn.formmodal插件在定义的时候，和infomodal稍有不同，其目的是在这个地方处理缓存问题，也就是判断是否使用绑定在元素上的原有实例（本例中是bs.formmodal和bs.modal），需要通过参数来判断。具体代码如下：

```
$.fn.formmodal = function (option, _relatedTarget) {
    return this.each(function () {                    // 根据选择器，遍历所有符合规则的元素
        var $this = $(this)
        var options = $.extend({}, FormModal.DEFAULTS, $this.data(),
            typeof option == ‘object’ && option)
        // 将默认参数、选择器所在元素上的自定义属性（data-开头）和option参数的值
        // 合并在一起，作为options参数
        // 优先级：后面的参数优先级高于前面的参数

        var data = options.cacheForm && $this.data(‘bs.formmodal’)
                                                          // 使用缓存时，才获取原来绑定的实例
        options.show = false;
```

```
        if (!options.cacheForm) {                // 如果不用缓存，基本modal实例也要先
                                                 // 清空，然后重新加载远程的html内容
            $this.data('bs.modal', null);
        }

        $this.modal(options, _relatedTarget);   // 重新加载内容

        // 如果值不存在，则将formmodal实例设置为bs.formmodal的值
        if (!data) $this.data('bs.formmodal', (data = new FormModal(this

        // 如果option传递了string，则表示要执行某个方法
         // 比如传入了show，则要执行formmodal实例的show方法，data["show"]相当于data.show()
        if (typeof option == 'string') {
            data[option](_relatedTarget)
        }
        else {
            data.show(_relatedTarget);
        }
    })
}
```

其他内容没有特别的地方，可以参考本节最后的完整源代码。我们来看一个使用示例：

```
<button   class="btn   btn-warning"   data-toggle="formmodal"   data-target="#AddPopup"
    data-backdrop="static">添加</button>
<div  class="modal  fade"  id="AddPopup"  data-cache-form="true"  data-remote="form.html">
</div>
```

要记得，ajax获取的内容，必须是完整的内容（以modal-dialog样式开始）。示例如下：

```
<div class="modal-dialog">
<div class="modal-content">
     <div class="modal-header">
                          <button  type="button"  class="close"  data-dismiss="modal" aria-
                hidden="true">×</button>
            <h4 class="modal-title" id="H1">Email设置</h4>
        </div>
        <div class="modal-body">
            <form class="form-horizontal" role="form">
                <div class="form-group">
                    <label class="col-sm-2 control-label">Email</label>
                                              <div  class="col-sm-8">
<input type="text" name="test"
                        class="form-control" /></div>
                     <span class="col-sm-1 control-label" id="messages">
</span>
                </div>
```

```
            </form>
        </div>
        <div class="modal-footer">
                <button type="button" class="btn btn-success" data-
form="submit">
                保存</button>
                 <button type="button" class="btn btn-danger" data-
form="reset">
                重置</button>
                <button type="button" class="btn btn-default" data-
form="cancel">
                取消</button>
        </div>
    </div>
</div>
```

上述示例的运行效果如图6-4所示。

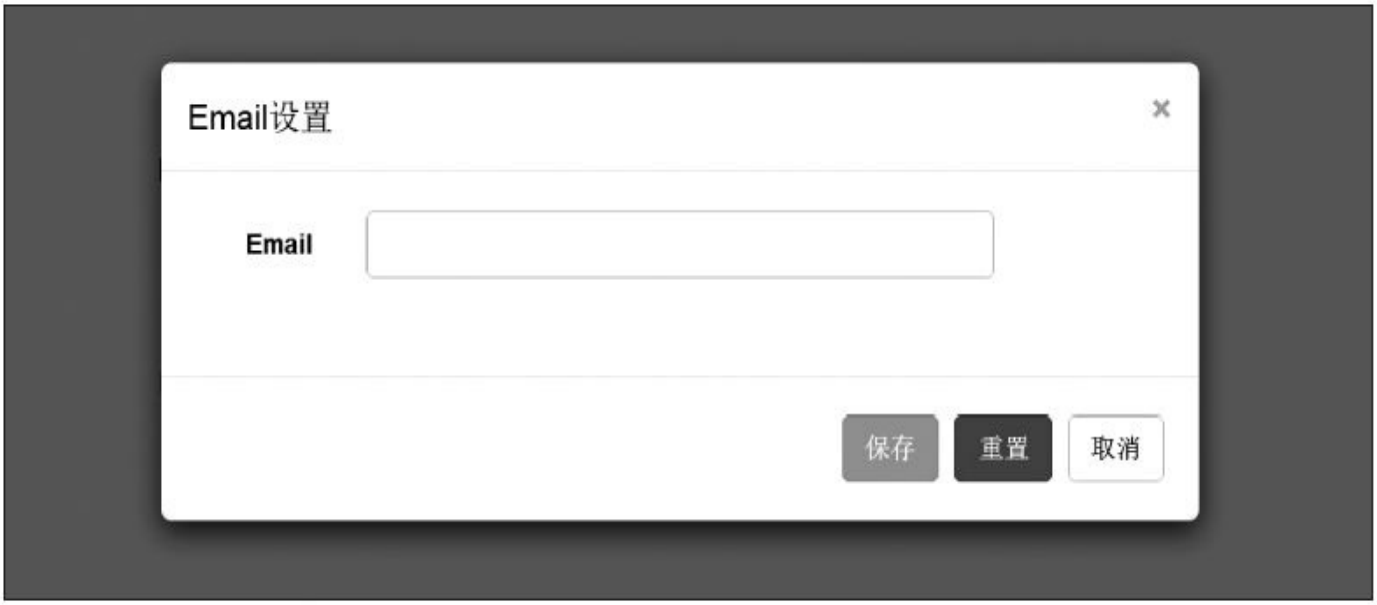


图6-4　ajax获取弹窗内容的运行效果

记住，如果要在弹窗里的表单上应用验证控件，需要在它的shown事件上重新应用一下验证触发事件。以jQuery.validate插件为例，其触发代码如下：

```
$('#AddPopup').on('shown.bs.formmodal', function (e) {
    $('#AddPopup').find('form').validate({
        sendForm: false,
        description: {
            test: {
                required: '<div class="label label-danger">Email必填！
</div>',
                pattern: '<div class="label label-
danger">Pattern</div>',
                conditional: '<div class="label label-
danger">Conditional</div>',
                valid: '<div class="label label-success">输入合法！
</div>'
            }
```

```
        },
        valid: function () {
            // console.log("valid"); 验证失败时的操作
        },
        invalid: function () {
            // console.log("invalid");验证成功时的操作
        }
    });
});
```

注意

❑上述示例是以ajax请求为例的，如果弹窗不请求ajax，则直接在声明式代码里写全表单就可以了，没有什么影响。注意，如果要在弹出前重置表单内容，直接调用$.formmodal('reset')即可。

❑上述示例中，ajax加载的form元素是放在modal-body样式里（即按钮在form元素外）的，如果把form元素放在modal-dialog外，全部包含了按钮和表单控件，则也可以直接使用原生的submit和reset按钮，而不需要使用data-form="submit"和data-form="reset"自定义属性。

Form弹窗的完整源代码如下：

```
+function ($) {
    "use strict";

    // FormModal类定义
    var FormModal = function (element, options) {
        this.$element = $(element);
          this.super = this.$element.data('bs.modal'); // 获取自定义属性bs.modal的值
        this.options = options;

        this.$element.on('click.submit.formmodal',
            '[data-form="submit"]', $.proxy(this.submit, this));
        this.$element.on('click.reset.formmodal',
            '[data-form="reset"]', $.proxy(this.reset, this));
        this.$element.on('click.cancel.formmodal',
            '[data-form="cancel"]', $.proxy(this.cancel, this));

        var that = this;                        // 防止污染作用域，用临时变量that

        this.$element.on("show.bs.modal", function (e) {
            that.$element.trigger(e = $.Event('show.bs.formmodal'));
            if (e.isDefaultPrevented()) return;
        });
        this.$element.on("shown.bs.modal", function (e) {
            that.$form = that.$element.find('form');
            that.$element.trigger(e = $.Event('shown.bs.formmodal'));
            if (e.isDefaultPrevented()) return;
        });
        this.$element.on("hide.bs.modal", function (e) {
            that.$element.trigger(e = $.Event('hide.bs.formmodal'));
```

```
            if (e.isDefaultPrevented()) return;
        });
        this.$element.on("hidden.bs.modal", function (e) {
            that.$element.trigger(e = $.Event('hidden.bs.formmodal'));
            if (e.isDefaultPrevented()) return;
        });
    }
    // 默认设置
    FormModal.DEFAULTS = {
        cacheForm: false,
        closeAfterCancel: true
    }
    // 反转弹窗状态
    FormModal.prototype.toggle = function (_relatedTarget) {
                return  this[!this.super.isShown  ?  'show'  :  'hide']
(_relatedTarget)
        // 如果是关闭状态，则打开弹窗，否则关闭
    }
    // 打开弹窗
    FormModal.prototype.show = function (_relatedTarget) {
        this.super.show(_relatedTarget);
    }
    // 关闭弹窗
    FormModal.prototype.hide = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为
        this.super.hide();
    }
    // 单击"确认"按钮的行为
    FormModal.prototype.submit = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为

        this.$element.trigger(e = $.Event('beforeSubmit.bs.formmodal'));
        // 提交前触发事件，主要用于处理相关代码

        if (e.isDefaultPrevented()) return;

        this.$form.submit();
        this.$element.trigger(e = $.Event('afterSubmit.bs.formmodal'));
                                                    // 提交后触发事件，用于处理
相关代码
        if (e.isDefaultPrevented()) return;
    }
    // 单击"重置"按钮的行为
    FormModal.prototype.reset = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为
        var resetAction = function () {
            this.$element.trigger(e = $.Event('beforeReset.bs.formmodal'
            // 重置前触发事件
            if (e.isDefaultPrevented()) return;

            this.$form.each(function () {
                 this.reset();            // jQuery不支持reset，需要转换为
DOM对象
                                                    // 再调用原生reset方法
            });
```

```
            this.$element.trigger(e = $.Event('afterReset.bs.formmodal')
                                                    // 重置后触发事件
            if (e.isDefaultPrevented()) return;
        }
        if (this.super.isShown) return resetAction.call(this);

        this.$element.one("shown.bs.formmodal", $.proxy(resetAction, thi
        this.show();
    }
    // 单击"取消"按钮的行为
    FormModal.prototype.cancel = function (e) {
        if (e) e.preventDefault();      // 先阻止冒泡行为

        var e = $.Event('cancel.bs.formmodal');
        this.$element.trigger(e);       // 取消前，先触发事件

        if (e.isDefaultPrevented()) return;
        if (this.options.closeAfterCancel) {
              this.hide(e);                 // 如果设置了data-close-after-
cancel=true
                                                    // 参数，则关闭弹窗
        }
    }
    // formmodal 插件定义
    var old = $.fn.formmodal
     // 如果定义了其他的formmodal插件，则保留它（以便在noConflict（解决防冲突）之
后，
    // 可以继续使用该旧插件代码）

    $.fn.formmodal = function (option, _relatedTarget) {
        return this.each(function () { // 根据选择器，遍历所有符合规则的元素
            var $this = $(this)
            var options = $.extend({}, FormModal.DEFAULTS, $this.data(),
                typeof option == 'object' && option)
            // 将默认参数、选择器所在元素的自定义属性（data-开头）、
            // option参数这3种的值合并在一起，作为options参数
            // 优先级：后面的参数优先级高于前面的参数

            var data = options.cacheForm && $this.data('bs.formmodal')
                                        // 获取自定义属性bs.modal的值

            options.show = false;
            if (!options.cacheForm) {
                                        // 如果不用缓存，则先清空实例，然后重
新加载远程的html内容
                $this.data('bs.modal', null);
            }

            $this.modal(options, _relatedTarget);
            // 如果值不存在，则将formmodal实例设置为bs.formmodal的值
            if (!data) $this.data('bs.formmodal',
                (data = new FormModal(this, options)))
            // 如果option传递了string，则表示要执行某个方法
            // 比如传入了show，则要执行formmodal实例的show方法，
            // data["show"]相当于data.show();
            if (typeof option == 'string') {
```

```
                data[option](_relatedTarget)
            }
            else {
                data.show(_relatedTarget);
            }
        })
    }
    $.fn.formmodal.Constructor = FormModal;
                                          // 重设插件构造器，可以通过该属性获取
插件的真实类函数

     // formmodal防冲突
    $.fn.formmodal.noConflict = function () {
        $.fn.formmodal = old
        return this
}

    // formmodal DATA-API
    $(document).on('click.bs.formmodal.data-api',
        '[data-toggle="formmodal"]', function (e) {
        // 监测所有拥有自定义属性data-toggle="modal"的元素上的单击事件
        var $this = $(this)
        var href = $this.attr('href')   // 获取href属性值
        var $target = $($this.attr('data-target') ||
            (href && href.replace(/.*(?=#[^\s]+$)/, '')))
        // 获取data-target属性值，如果没有，则获取href值，该值是所弹出元素的id

        // 如果弹窗元素上设置了data-formmodal属性值，则option值是字符串toggle
         // 否则将remote值（如果有的话）、弹窗元素上的自定义属性值集合、触发元素上
的
        // 自定义属性值集合进行合并，作为option选项对象
        var option = $target.data('formmodal') ? 'toggle' : $.extend({
            remote: !/#/.test(href) && href }, $target.data(), $this.dat

        e.preventDefault()              // 阻止默认行为
        $target
             .formmodal(option, this)    // 给弹窗元素绑定formmodal插件（即
实例化
                                                  // formmodal），并传入
option参数
            .one('hide', function () {
                $this.is(':visible') && $this.focus()
                                                 // 定义一次hide事件，使所
单击元素获得焦点
            })
    })
}(jQuery);
```

第7章　实战：Win8磁贴组件开发

Win8的成功很大一部分功劳是属于非常耀眼的磁贴（Tile），各种方块的不同形状、不同颜色，非常绚丽。在本章，我们将开发一个Tile组件，用于模拟磁贴片大部分的功能，使得在Web上也能实现如此酷炫的效果。

回顾一下Bootstrap的组件和插件开发流程，一个标准的CSS插件要考虑8个方面的样式，分别是：基础样式、颜色样式、尺寸样式、状态样式、特殊元素样式、并列元素与嵌套子元素样式，以及动画样式（配合JavaScript代码）。

下面就按照这8种样式，分别对Tile组件进行开发定义。

7.1 基本样式

一个Tile的基本内容其实就只有一个带有颜色的方块，由于颜色样式会在下节讲解，所以默认最基本的方块会设置一个灰色的背景。还有就是一个移动上去出现的边框了。定义完这两项内容，其实就完成了基本样式的定义。

7.1.1 方块定义

在分析Bootstrap所有的CSS组件时都可以看到，一个可重用的组件（尤其是可以并列多个显示的元素），在定义其样式的时候需要考虑到定位方式、显示方式、默认大小、内外边距等，如果是元素容器还要考虑到容器溢出的情况。

针对方块样式的定义，比较简单，没有太特殊的地方，所以请大家直接看代码吧。

```
.tile {
    position: relative;          /* 相对布局*/
    overflow: hidden;            /* 内部元素溢出时，将其隐藏*/
    display: block;              /* 块级显示*/
    float: left;                       /* 左浮动*/
    width: 120px;                      /* 标准的tile大小是120×120*/
    height: 120px;
    margin: 0 10px 10px 0;       /* 设置右外边距和底部外边距，稍后会讲解原因*/
    padding: 0;                  /* 不设置内边距*/
    cursor: pointer;             /* 默认鼠标移动上去时显示手形图标 */
    background-color: #eeeeee;   /* 默认背景颜色是灰色 */
    box-shadow: inset 0px 0px 1px #FFFFCC;      /* 设置少许的阴影 */
}
```

上述代码中有一个地方需要说明一下，就是margin的定义，这里只定义了右边（right）和底部（bottom）的margin值，而没有定义左边（left）和顶部（top），主要是因为margin在一些浏览器上有重叠的问题，也就是说，如果你为一个方块定义了左右外边距分部是10px，那如果这两个方块在一起显示的话，其中间的间隔只有10px，而不是20px。为了消除这个bug，我们不采取对称的声明方式，以防止出现上述问题。

关于margin重叠的详细内容，请访问http://www.zhangxinxu.com/study/200908/margin-overlap.html。

7.1.2　边框定义

定义了基本的方块，下面就需要定义鼠标移动时要显示的边框了。这里的“边框”其实不是真正的border，而是outline轮廓，其原因是outline不占用布局空间，并且也能产生同样的效果，所以，我们就使用outline来定义所谓的“边框”。代码如下：

```
.tile:hover {
    outline: #999999 solid 3px; /* 鼠标移动时，设置轮廓颜色和边框 */
}

.tile.no-outline {              /* 提供一个no-outline样式用于专门禁用轮廓*/
    outline-color: transparent; /* 不使用轮廓时，设置透明色 */
}
```

至此，基本的样式就定义完了，我们通过在一个div上应用tile样式即可制作出一个默认的磁贴。代码如下：

```
<div class=“tile”></div>
```

上述示例的运行效果如图7-1所示。

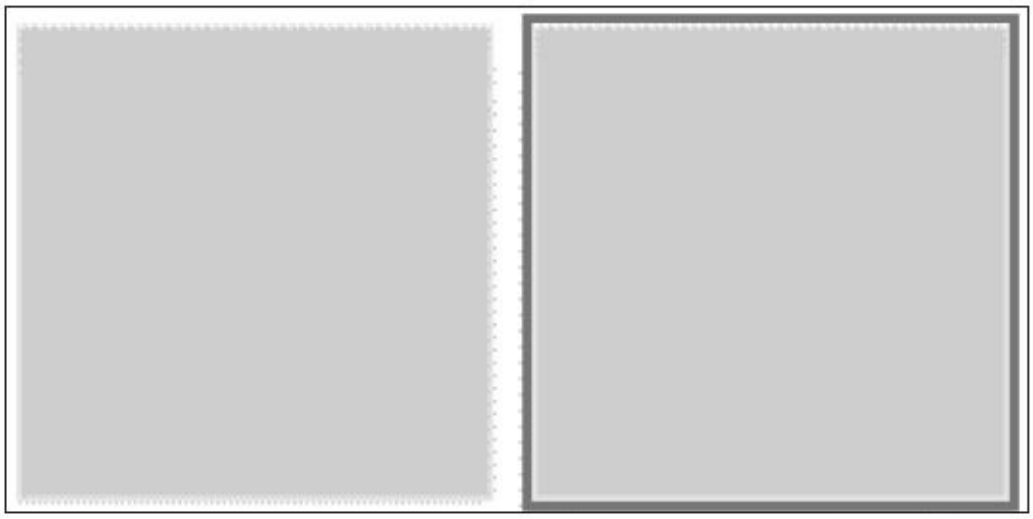


图7-1　基础磁贴效果

注意

上述效果中的右图是鼠标移动上去时的效果。

7.2 颜色样式

要设计tile的颜色样式，需要考虑的内容很多，有背景颜色、轮廓颜色，以及以后内部要放置的文本颜色，我们要为此设置一套规则。初步建议如下：

```
.tile-alert {…}
.tile-info {…}
```

但是，正当想实现这些样式的时候，发现这样定义几个风格，不足以实现Win8上那些所有的酷炫效果。所以应该再设计一套更灵活的风格，以便能够让开发人员自由选择背景颜色、轮廓颜色，以及文本颜色。

由于Win8本身就提供了很多安全颜色，所以我们可以直接通过在这些颜色名称前面加上一个前缀来实现相应的效果。同时，为了不局限于只在tile上使用，我们去除了tile前面的限制。示例如下：

```
/*背景色样式*/
.bg-red { background-color: #e51400 !important;}
.bg-green { background-color: #60a917 !important;}
.bg-darkBlue { background-color: #16499a !important;}
/*其他省略*/
```

同样的道理，也可以定义轮廓颜色（outline）、文本颜色（foreground），以及有可能用到的边框颜色（border）。示例如下：

```
/*内部文本样式*/
.fg-red { color: #e51400 !important;}
.fg-green { color: #60a917 !important;}
.fg-darkBlue { color: #16499a !important;}

/*轮廓样式*/
.ol-red { outline-color: #e51400 !important;}
.ol-green { outline-color: #60a917 !important;}
.ol-darkBlue { outline-color: #16499a !important;}

/*边框颜色样式*/
.bd-red { border-color: #e51400 !important;}
.bd-green { border-color: #60a917 !important;}
.bd-darkBlue { border-color: #16499a !important;}
```

使用上述规则，可以将所有Win8的安全颜色都定义成相应的样式，如表7-1所示是Win8支持的安全颜色。

表 7-1　Win 8 安全颜色列表

颜色名称	颜色名称	颜色名称
*-black	*-yellow	*-darkTeal
*-white	*-brown	*-darkEmerald
*-lime	*-olive	*-darkGreen
*-green	*-steel	*-darkOrange
*-emerald	*-mauve	*-darkRed
*-teal	*-taupe	*-darkPink
*-cyan	*-gray	*-darkViolet
*-cobalt	*-dark	*-darkBlue
*-indigo	*-darker	*-lightBlue
*-violet	*-transparent	*-lightTeal
*-pink	*-darkBrown	*-lightOlive
*-magenta	*-darkCrimson	*-lightOrange
*-crimson	*-darkMagenta	*-lightPink
*-red	*-darkIndigo	*-lightRed
*-orange	*-darkCyan	*-lightGreen
*-amber	*-darkCobalt	

定义完上述样式，就可以通过组合制作出不同风格的方块了。比如，定义如下示例：

```
<div class=“tile bg-red”></div>
<div class=“tile bg-green”></div>
<div class=“tile ol-orange”></div>
<div class=“tile bg-yellow”></div>
```

定义4个tile，4个不同的颜色，然后给第三个tile也定义了一个橙色的轮廓，效果如图7-2所示。看起来是不是好看很多？

图7-2　不同颜色的磁贴效果

7.3　尺寸样式

这一步，我们要定义tile的尺寸样式了。标准的Win8磁贴片有5个不同的大小，分别是：标准大小（120×120）、50%大小（55×55）、2倍大小（120×250）、3倍大小（120×380）、4倍大小（120×510）。另外，针对放大磁贴，还提供了垂直方向的显示效果，分别是：垂直2倍（250×120）、垂直3倍（380×120）、垂直4倍（510×120）。

读者可能比较奇怪，2倍大小的尺寸应该是240px才对，为什么是250px？这是因为，在Win8里，所有的磁贴片不论大小如何，都应该进行对齐，所以如果是2倍，就应该把两个标准磁贴片之间的margin间隔也应该算上，才能对齐，对齐效果如图7-3所示。2倍公式为：120×2+10=250px；同理3倍尺寸的话，应该加2个margin间隔，公式为：120×3+10×2=380px。

图7-3　不同尺寸的磁贴对齐效果

尺寸样式的实现方式比较简单，就只是改变所需要的高度和宽度而已。代码如下：

```
/* 50%大小 */
.tile.half { width: 55px; height: 55px;}

/* 2倍大小 */
.tile.double { width: 250px;}

/* 3倍大小 */
.tile.triple { width: 380px;}

/* 4倍大小 */
.tile.quadro { width: 510px;}
```

同理，如果要定义垂直倍数的样式，只需要修改高度即可。代码如下：

```
/* 垂直2倍大小 */
.tile.double-vertical { height: 250px;}
```

```
/* 垂直3倍大小 */
.tile.triple-vertical { height: 380px;}

/* 垂直4倍大小 */
.tile.quadro-vertical { height: 510px;}
```

至此，有了基本样式，也有了颜色样式，而且尺寸样式也定义了，应该可以组合出任意酷炫的磁贴片效果了。

7.4　状态设置

普通的组件在定义状态的时候主要是定义高亮和禁用，但是在Tile组件里，只需要定义一个选中状态就可以了，效果如图7-4所示。

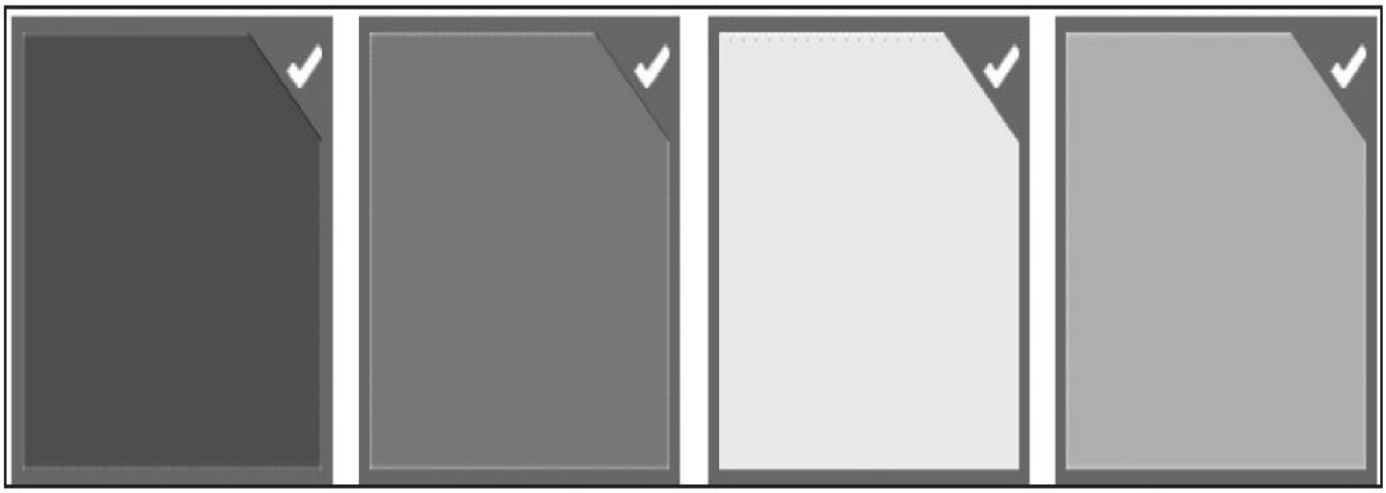


图7-4　选中状态的磁贴效果

由于选中效果是在右上角显示一个对号，该对号又是真实的内容，所以实现的时候要么叠加新的元素，比如在标签里定义一个小图标；要么使用CSS的before和after特性。叠加新元素太麻烦了，所以，在这里利用CSS特性来实现。

先定义方块在选中状态时的边框（默认为蓝色）。代码如下：

```
.tile.selected {
   border: 4px #4390df solid;
}
```

再定义右上角的蓝色三角形（不包括对号图标）。注意一下三角形是如何通过boder-top和border-left来实现的。

```
.tile.selected:after {
    position: absolute;                 /* 绝对定位 */
    display: block;                     /* 块级显示 */
    border-top: 28px solid #4390df;     /* 三角形定义 */
    border-left: 28px solid transparent;
    right: 0;                           /* 三角形靠右上角 */
    top: 0;
    content: ””;
    z-index: 101;
}
```

最后才是对小图标的设置。关于小图标，由于Bootstrap已经提供了很多样式，符合该图标的样式为glyphicon-ok样式，尽量不要叠加新的样式进来，所以这里重用了glyphicon-ok样式里的content内容来定义图标。具体代码如下：

```
.tile.selected:before {
    position: absolute;                  /* 绝对定位 */
    display: block;                      /* 块级显示 */
    content: ”\e013”;                    /* 小图标内容 */
    color: #fff;                                 /* 颜色为白色 */
    right: 0;                            /* 右上角显示 */
    top: 0;
    font-family: Glyphicons Halflings;  /* 设置图标字体*/
    font-size: 10pt;
    font-weight: normal;
    z-index: 102;
}
```

至此，我们仅通过一个selected样式就可以表示磁贴片的选中状态了。示例如下：

```
<div class=“tile selected”></div>
```

7.5　特殊元素样式

到此，可能觉得磁贴片的主要功能都已经完成了。其实还有很多呢，先看看如图7-5所示的效果。

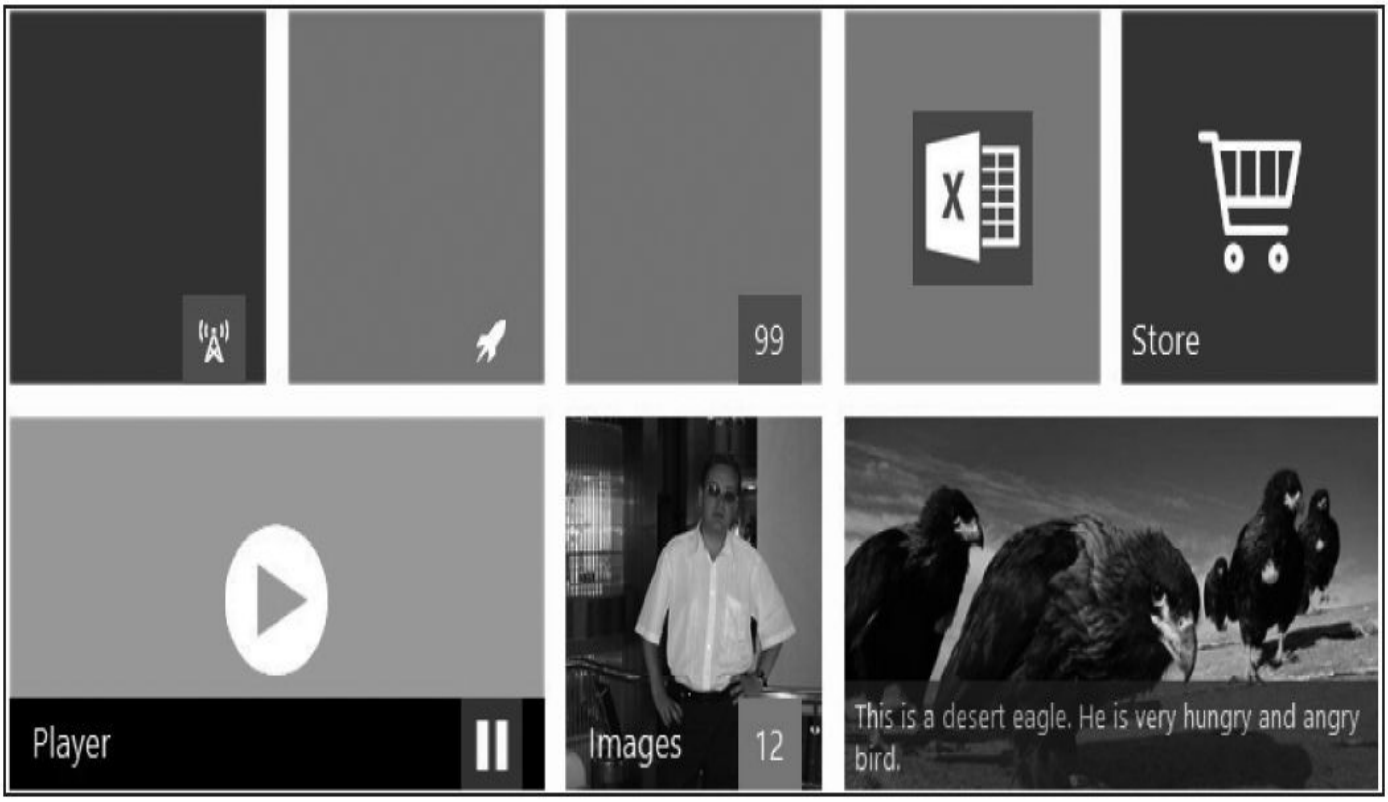


图7-5　磁贴里的特殊元素效果

上述内容其实和磁贴片本身没有多大关系，都是磁贴片内部的显示内容。我们在第2章的CSS架构里说过，一个组件内部或多或少都会有一些固定格式的内容进行显示，比如图7-5中磁贴片中的Excel图标、右下角的提示信息（称为Badge）、底部带有透明度的横条等。本节，我们就针对这些特殊元素，设置另外一组新的样式。

根据上述效果，可将元素内容设置为两部分，分别是主内容区（tile-content）和Brand底部横条区，然后底部横条内部又分3种类型的样式，分别是左下角的标题（label）、右下角的徽章（badge），以及只显示文本的text。完整的布局样式如下：

```
<div class=“tile ”>
    <div class=“tile-content”>
        主区域内容
    </div>
    <div class=“brand ”>
        <span class=“label”>Player</span>
        <span class=“text”>text内容</span>
          <div class=“badge”><i class=“glyphicon glyphicon-pause”> </i>
</div>
            <!— 小图标设置 —>
```

```
    </div>
</div>
```

另外，如果磁贴的内部只显示图标的话，我们可以做特殊处理（图标统一为56×56），并且居中显示，在title-content样式的基础上，添加一个icon标记样式即可。示例代码如下：

```
<div class=“tile ”>
    <div class=“tile-content icon”>
        <i class=“glyphicon glyphicon-play”></i>
</div>
</div>
```

7.5.1 tile-content样式定义

大概设置了布局以后，就分别来实现这些样式。首先对于tile-content样式，只需要让其充满tile容器的全屏即可。代码如下：

```
.tile .tile-content {
    width: 100%;                                    /* 充满父元素*/
    height: 100%;
    padding: 0;                                 /* 内外边距都取消*/
    margin: 0;
    display: block;                             /* 块级显示*/
    position: absolute;                         /* 绝对定位*/
    left: 0;
    top: 0;
    overflow: hidden;
}
```

上面的代码取消了内外边距的设置。如果确实想使用一些内边距，则在tile-content上附加一个padded样式即可。实现代码如下：

```
.tile .tile-content.padded {
    padding: 10px;                              /* 10像素足够了*/
}
```

7.5.2 tile-content子元素的样式定义

但是，如果要在主区域单独显示一个充满100%的图片的话，还得对它特殊设置一下。本例中，通过再次附加一个image样式的方式实现。

```
.tile .tile-content.image {
    width: 100%;                                    /* 图标100%充满容器 */
    min-height: 100%;
    max-height: 100%;
}
/* 示例：<div class="tile-content image"><img src=""/></div>*/
```

注意一下，图7-5效果中的Excel图标只是一个图标，不能使用image样式进行设置，应该使用icon样式。针对该样式，定义如下：

```
.tile .tile-content.icon [class*="icon-"],
    /* tile-content上应用icon，并且子元素应用以icon-开头的样式时 */
.tile .tile-content.icon [class*="glyphicon-"],
    /* tile-content上应用icon，并且子元素应用以glyphicon-开头的样式时 */
.tile .tile-content.icon img { /* tile-content上应用icon，并且子元素是imag
元素时 */
    line-height: 56px;
    height: 56px;                    /* 强制设置图标大小是56×56 */
    width: 56px;
    font-size: 48px;
    color: #ffffff;          /* 白色字体或图标 */
    text-align: center;
    position: absolute;      /* 绝对定位 */
    left: 50%;
    top: 50%;
    margin-top: -28px;
    margin-left: -28px;
}

/* 示例：<div class="tile-content icon"><img src=""/></div>*/
```

上述样式说明，只要在tile-content上应用icon样式，在内部的子元素上，应该是可以有3种方式：Bootstrap默认的glyphicon、普通icon-开头的图标以及元素的img元素。无论使用哪种方式都可以。

所以，根据这些样式，可以通过glyphicon-play样式来实现上述效果图的显示效果。示例代码如下：

```
<div class="tile-content icon">
    <i class="glyphicon glyphicon-play"></i>
</div>
```

在上述样式中有几个属性是实现图标的上下左右居中的，其实现方式比较灵活。具体代码如下：

```
left: 50%;
top: 50%;
```

```
margin-top: -28px;
margin-left: -28px;
```

其原理是，首先将图标元素进行校对定位以后，将其左顶点从磁贴的中心区域开始（通过left和top的50%来实现），然后再将其左（并向上）移动28像素（56像素的一半），从而实现上下左右居中。这个所谓的“移动”其实是通过设置负的margin值来实现的，也就是margin-top和margin-left，效果如图7-6所示。

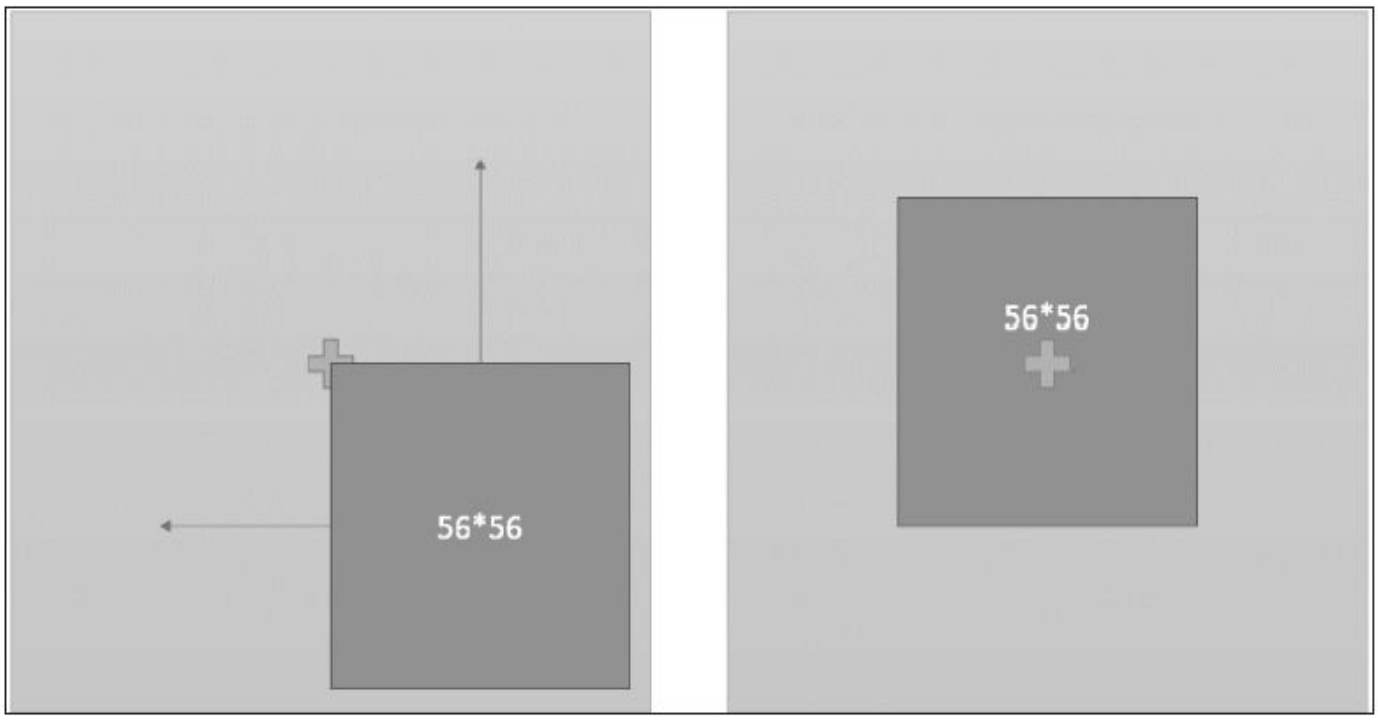


图7-6　磁贴图片居中显示的原理图

7.5.3 Brand样式定义

根据前面的布局，我们知道，Brand区域主要包括brand容器样式和3个子样式（label、text、badge），其中text样式一般不和label（以及badge）同时出现。

先来看一下brand容器样式。根据磁贴效果可以知道，该部分内容要绝对定位在磁贴的底部，并且带有一些透明度。该样式的设置如下：

```
.tile .brand {
    position: absolute;                 /* 绝对定位 */
    bottom: 0;                          /* 靠近磁贴底部 */
    left: 0;                            /* 靠近磁贴左边 */
    right: 0;                           /* 靠近磁贴右边 */
    min-height: 29px;                   /* 最小高度29像素 */
    margin: 0;                          /* 不设置margin */
}

.tile .brand.opacity { opacity: .7;}

.tile .brand:before,
.tile .brand:after { display: table;    content: "";}
.tile .brand:after { clear: both;}
```

接着，我们再来看label和text样式。两个样式差不多，不一样的地方有字体大小、行间距以及外边距。所以我们先设置一个通用的样式，然后再设置另外一个不同的样式。代码如下：

```
.tile .brand .label,
.tile .brand .text {
    float: left;
    display: block;
    font-size: 10pt;
    margin: 5px 20px 2px 5px;
    background: transparent;
    color: #ffffff;
}

.tile .brand .text {
    margin: 4px 5px;
    font-size: 8pt;
    line-height: 10pt;
}
```

在右下角显示更小的图标所用的徽章样式设置也很简单，主要是利用绝对定位将其定位到右下角即可。代码如下：

```
.tile .brand .badge {
    position: absolute; /*绝对定位*/
    bottom: 0;          /*靠近底部*/
    right: 10px;                /*离最右边保持10像素距离*/
    width: 29px;
```

```
    height: 29px;                /*确保和brand横条高度一致*/
    text-align: center;
    color: #ffffff;     /*默认文本为白色*/
    padding: 5px;
    margin: 0;
    vertical-align: middle !important;
    display: block;
    font-size: 10pt;
    border-radius: 0px; /* 特别重要，要覆盖Bootstrap默认badge样式的圆角设置*/
}
```

上述源码可以看到，由于Bootstrap内置的徽章样式是带有圆角的，所以为了让该样式在Win8风格上实现小方块的效果，就需要取消圆角的设置。

7.6　并列元素与嵌套样式

本小节将实现将一组磁贴片放在一起的效果，类似于按钮的btn-group样式，在这里取名为tile-group样式，效果如图7-7所示。

图7-7　一组磁贴在一起的显示效果

通过图7-7所示效果可以看出来，要实现一组容器的效果，除了在容器内添加一个放置标题的元素以外，左右两边还要保留10像素的内边距。由于每个tile本身在右边和底部已经有了10像素的margin值了，所以此时只需要设置tile-group样式的padding-left为10px就可以了。实现代码如下：

```
.tile-group {
    float: left;
    display: block;
     padding: 40px 0px 0px 10px; /* 40像素时给title标题留的位置，10像素时左边的间隙*/
    position: relative;
    color: #ffffff;
}

.tile-group .tile-group-title {
    font-size: 18px;
    line-height: 20px;
    position: absolute;          /* 绝对定位*/
    top: 10px;
    color: inherit;
}
```

这样一来，实现上述效果图的HTML代码就简单多了。代码如下：

```
<div class=“tile-group bg-dark”>
    <div class=“tile-group-title”>title</div>
    <div class=“tile bg-red”></div>
    <div class=“tile bg-green”></div>
    <div class=“tile ol-orange”></div>
</div>
```

由于Win8风格的磁贴组，一般默认是水平显示为2个、4个、6个，一般不都不会超过7个，所以可以通过单独的样式来控制水平方向的宽度，以便让多余的磁贴tile换行显示。

```
.tile-group.two,
.tile-group.double { width: 270px; /* 250+10 */}

.tile-group.three,
.tile-group.triple { width: 400px; /* 380+10 */}

.tile-group.four,
.tile-group.quadro { width: 530px; /* 510+10 */}

.tile-group.five { width: 660px;}
.tile-group.six { width: 790px;}
.tile-group.seven { width: 920px;}
```

使用时只需在tile-group样式上附加一个要限制的个数样式即可。比如总共有6个tile，每行只能显示3个，此时再附加一个three样式即可。示例如下：

```
<div class=“tile-group three bg-dark”>
    <div class=“tile-group-title”>title</div>
    <div class=“tile bg-red”></div>
    <div class=“tile bg-green”></div>
    <div class=“tile ol-orange”></div>
    <div class=“tile bg-red”></div>
    <div class=“tile bg-green”></div>
    <div class=“tile ol-orange”></div>
</div>
```

7.7 动画插件

整个tile插件基本上就算完成了，但是还差一个动画部分。Win8上经常有一些这样的效果：磁贴片里的内容有规律地滚动，比如向上逐条显示新闻或邮件。本节，我们就将一步一步实现这些效果。

先分析一下该插件都需要设置什么内容。由于该插件和旋转轮播效果类似，所以通过回顾轮播插件，可以分析到，需要以下几个方面的内容：

❑可以在上下左右4个方向进行秩序滚动，如果能有左右（或上下）弹球式滚动就更好了，要求能在声明式用法里设置滚动效果，比如slideLeft、slideUp等。

❑要求能设置滚动时间、滚动以后的暂停时间。

❑是否能支持更炫的动画特效。

经过考虑，我们发现前两点比较容易实现，第3点的更炫特效由于已经偏离了本书的内容，所以只能利用第三方的jQuery的easing插件进行调用，也就是暴露一个接口，让开发人员自行选择。

首先要考虑一下，多个滚动的内容如何定义。前面我们在容器内定义了一个主区域内容tile-content，可以在里面连续放置多个充满tile-content元素的子元素用于滚动。可是，再设置一个层级的子元素，又要处理很多相关的CSS样式，这里，就大胆使用多个tile-content元素试试，此时只需要显示一个tile-content，其他的tile-content都默认隐藏即可，因为其他的tile-content可以通过tile动画插件来显示。具体结构如下：

```
<div class="tile double bg-amber" data-toggle="tile" data-direction="slideLeftRight"
    data-period="700" data-duration="3000" data-easing="doubleSqrt">
    <div class="tile-content bg-red">a区</div>
    <div class="tile-content bg-green">b区</div>
    <div class="tile-content bg-orange ">c 区</div>
    <div class="tile-content bg-grayDark">d区 </div>
</div>
```

这里，除了在tile元素放置多个tile-content子容器以外，还定义了5个参数，分别表示：插件标识、滚动方向、单个tile-content滚动时间、滚动结束后的停留时间、动画过渡效果。

此时，我们需要解决一个问题，那就是在一个tile里，默认只能显示一个tile-content容器，这个内容可以使用CSS来控制，默认设置只显示第一个tile-content容器（此时，也不影响原先定义的CSS样式）。具体代码如下：

```
.tile .tile-content {    display: none;}
.tile .tile-content:first-child {    display: block;}
```

声明式布局定义好以后，就开始定义插件内容。

步骤1　立即调用的函数。

```
// 1.定义立即调用的函数
+function ($) {
    “use strict”;                                    // 使用严格模式 ES5支持
    // 后续步骤
    // 2.tile插件类及原型方法的定义
    // 3.在jQuery上定义tile插件，并重设插件构造器
    // 并重设插件构造器,可以通过该属性获取插件的真实类函数
    // 4. 防冲突处理
    // 5. 绑定触发事件
}(window.jQuery);
```

步骤2　插件核心代码。定义基础的Tile类函数和相关原型方法，Tile类函数主要是接收options值、设置各种默认值等。

Tile类函数的定义代码如下：

```
// Tile类函数定义
var Tile = function (element, options) {
    this.$element = $(element);          // 容器元素
this.options = options;
// 插件运行参数，根据初始化代码，优先级最高的是所单击元素上的data-属性，
// 然后是容器上的data-属性，最后才是默认值

    this.frames = this.$element.children(“.tile-content”);
                                                    // 查找有多少个主区域需要滚
动
    this.currentIndex = 0;                  // 当前所显示主区域的索引
    this.interval = 0;                      // 触发设置
    this.size = {                                   // 获取当前磁贴的高度和宽度
        ‘width’: this.$element.width(),
        ‘height’: this.$element.height()
    };

    // 确保默认的参数都是正常传值，如果是undefined，就使用默认值
    if (this.options.direction == undefined) { this.options.direction =
        Tile.DEFAULTS.direction; }
    if (this.options.period == undefined) { this.options.period =
        Tile.DEFAULTS.period; }
    if (this.options.duration == undefined) { this.options.duration =
        Tile.DEFAULTS.duration; }
    if (this.options.easing == undefined) { this.options.easing =
        Tile.DEFAULTS.easing; }

    // 定义一个默认的动画过渡效果，可以使用jQuery的easing插件
    $.easing.doubleSqrt = function (t) { return Math.sqrt(Math.sqrt(t));
}

// 默认值定义
Tile.DEFAULTS = {
```

```
    direction: ‘slideLeft’,      // 默认滚动方向
    period: 3000,                        // 默认暂停间隔
    duration: 700,               // 默认滚动时间
    easing: ‘doubleSqrt’                 // 默认动画过渡效果
}
```

关于原型方法，能公开进行调用的，我们只暴露start和pause，其他的一些原型方法都是用于处理滚动效果的，先来看一下核心摘要代码：

```
// 启动执行动画
Tile.prototype.start = function () {
    var that = this;
    this.interval = setInterval(function () {
        that.animate();
    }, that.options.period);
}

// 暂停动画
Tile.prototype.pause = function () {
    var that = this;
    clearInterval(that.interval);
}

// 动画处理入口，再分别调用各自方向的动画处理效果
Tile.prototype.animate = function () {
    var that = this;
    var currentFrame = this.frames[this.currentIndex], nextFrame;
    this.currentIndex += 1;
    if (this.currentIndex >= this.frames.length) this.currentIndex = 0;
    nextFrame = this.frames[this.currentIndex];

    // 根据滚动方向，分别调用相应的内部方法，参数是：
    // 当前要滚动的tile-content、下一个要滚动的tile-content
    switch (this.options.direction) {
        case ‘slideLeft’: this.slideLeft(currentFrame, nextFrame); break
        case ‘slideRight’: this.slideRight(currentFrame, nextFrame); bre
        case ‘slideDown’: this.slideDown(currentFrame, nextFrame); break
        case ‘slideUpDown’: this.slideUpDown(currentFrame, nextFrame); b
        case ‘slideLeftRight’: this.slideLeftRight(currentFrame, nextFra
        default: this.slideUp(currentFrame, nextFrame);
    }
}

// 左右来回滚动效果
Tile.prototype.slideLeftRight = function (currentFrame, nextFrame) {…}
// 上下来回滚动效果
Tile.prototype.slideUpDown = function (currentFrame, nextFrame) {…}
// 一直向上滚动效果
Tile.prototype.slideUp = function (currentFrame, nextFrame) {…}
// 一直向下滚动效果
Tile.prototype.slideDown = function (currentFrame, nextFrame) {…}
// 一直向左滚动效果
Tile.prototype.slideLeft = function (currentFrame, nextFrame) {…}
// 一直向右滚动效果
Tile.prototype.slideRight = function (currentFrame, nextFrame) {…}
```

这里，在start里使用了setInterval函数来触发动画执行函数，而在pause里使用了clearInterval函数清除动画执行函数。其他方面都很好理解了，看注释即可明白。

而关于4个方向的滚动效果则更简单，主要就是利用jQuery的animate函数将原有的tile-content移出后，再将下一个tile-content移入并显示。代码如下：

```
// 左右来回滚动效果
Tile.prototype.slideLeftRight = function (currentFrame, nextFrame) {
    if (this.currentIndex % 2 == 1)  // 用奇偶数来决定滚动方向
        this.slideLeft(currentFrame, nextFrame);
    else
        this.slideRight(currentFrame, nextFrame);
}

// 上下来回滚动效果
Tile.prototype.slideUpDown = function (currentFrame, nextFrame) {
    if (this.currentIndex % 2 == 1)  // 用奇偶数来决定滚动方向
        this.slideUp(currentFrame, nextFrame);
    else
        this.slideDown(currentFrame, nextFrame);
}

// 一直向上滚动效果
Tile.prototype.slideUp = function (currentFrame, nextFrame) {
    var move = this.size.height;
    var options = {
        'duration': this.options.duration,
        'easing': this.options.easing
    };

    $(currentFrame).animate({ top: -move }, options);
    $(nextFrame)
        .css({ top: move })
        .show()
        .animate({ top: 0 }, options);
}

// 一直向下滚动效果
Tile.prototype.slideDown = function (currentFrame, nextFrame) {
    var move = this.size.height;
    var options = {
        'duration': this.options.duration,
        'easing': this.options.easing
    };

    $(currentFrame).animate({ top: move }, options);
    $(nextFrame)
        .css({ top: -move })
        .show()
        .animate({ top: 0 }, options);
}
```

```
// 一直向左滚动效果
Tile.prototype.slideLeft = function (currentFrame, nextFrame) {
    var move = this.size.width;
    var options = {
        'duration': this.options.duration,
        'easing': this.options.easing
    };

    $(currentFrame).animate({ left: -move }, options);
    $(nextFrame)
        .css({ left: move })
        .show()
        .animate({ left: 0 }, options);
}

// 一直向右滚动效果
Tile.prototype.slideRight = function (currentFrame, nextFrame) {
    var move = this.size.width;
    var options = {
        'duration': this.options.duration,
        'easing': this.options.easing
    };

    $(currentFrame).animate({ left: move }, options);
    $(nextFrame)
        .css({ left: -move })
        .show()
        .animate({ left: 0 }, options);
}
```

步骤3　jQuery插件定义。在jQuery上定义$.fn.tile插件，有点特殊的代码是options参数的收集和合并，主要收集了3部分：插件的默认参数Defaults、modal元素上的data-属性、执行插件时候传入的option对象。三者的优先级依次升高。源码如下：

```
// Tile插件定义
var old = $.fn.tile;
//  保留其他库的$.fn.tile代码（如果定义的话），以便在noConflict之后，可以继续使用该老代码
$.fn.tile = function (option) {
    return this.each(function () {      // 遍历所有符合规则的元素
        var $this = $(this)             // 当前触发元素的jQuery对象
        var data = $this.data('bs.tile')
                                                    // 获取自定义属性data-bs.tile的值（其实是tile实例）

        var options = $.extend({}, Tile.DEFAULTS, $this.data(),
            typeof option == 'object' && option)
       // 合并参数，优先级依次递增

        if (!data) $this.data('bs.tile', (data = new Tile(this, options)
       // 如果没有tile实例，就初始化一个，并传入this和参数

        option === 'pause' ? data.pause() : data.start();
    })
```

```
}
```

步骤4　防冲突处理。防冲突处理的源码如下：

```
$.fn.tile.Constructor = Tile;       // 并重设插件构造器，可以通过该属性获取插件的真实类函数
// Tile 防冲突
$.fn.tile.noConflict = function () {
    $.fn.tile = old
    return this
}
```

上述代码非常简单了，可以像其他插件一样接收option参数（参数里可以设置滚动方向、时间等）。唯一一个需要注意的就是，默认的执行方式只提供了start和pause，而且，只有在option参数是pause字符串的时候才执行pause方法。否则一律执行start方法。大家在进行JavaScript调用的时候需要注意一下。

步骤5　绑定触发事件。DATA-API绑定触发事件则更简单了，由于不需要绑定相关的click事件，只需要在data-toggle=“tile”属性的tile元素上执行tile插件即可。

```
// Tile DATA-API
// 由于不需要click相关的时间，所以这里只是简单触发tile插件
$(window).on('load', function () {
    $('[data-toggle="tile"]').each(function () {          // 遍历所有符合规则的元素
        $(this).tile();                                   // 实例化插件以便自动运行
    })
})
```

一般情况下，该插件使用声明式用法即可。关于插件的JavaScript用法，我们来举两个例子。代码如下：

```
// 触发所有的tile磁贴
$(function () {
    $(".tile").tile();
});
// 对特定磁贴启用特殊参数
$(function () {
    $("#firstTile").tile({
        direction: "slideRight",
        duration: 700
    });
});
```

如果需要在其他元素上作用单击事件，来手动触发tile插件的滚动效果，则利用现有插件的实例，在停止的时候传入pause参数即可。代码如下：

```
// 通过开始和暂停按钮手动触发磁贴动画
```

```
$(function () {
    $(“#btnStart”).click(function () {
        $(“#firstTile”).tile({
            direction: “slideRight”,
            duration: 700
        });
    });
    $(“#btnPause”).click(function () {
        $(“#firstTile”).tile(‘pause’);
    });
});
```

7.8　更全的Win8风格样式

除了磁贴组件以外，Win8还有很多其他风格的组件，乌克兰的大神Sergey　Pimenov编写了一个全功能的Metro　UI框架，用于在Web上模拟Win8上的所有风格组件，拥有数十种组件。笔者在编写本章实战举例的时候，也大量参考了该框架中的Tile组件部分。其官方网站是http://metroui.org.ua/，目前该框架的最新版本是2.0，可以独立于Bootstrap运行。

Sergey　Pimenov在Tile组件中还添加了其他方面的特殊元素样式，比如email样式，并改进了Tile分组的设置，读者可以再自行研究一下他的Tile组件。

第8章　实战：组合应用开发

本章的实战过程是一个真实的“逆袭”事件，主人公是笔者的亲戚韩先生，原本是做测试的，一直想转做前端开发但没有机会。在美国客户举办的一次限时技术比赛上（3小时），通过利用Bootstrap做出的精美网页获得客户的高度赞赏，客户最终同意他转为Web前端开发。

8.1 任务描述

事情的起源是这样的，韩先生在一家外包公司做测试，其客户是美国一家很有名的产品公司。该公司使用了多家外包公司分别做不同的项目以降低风险，其中韩先生这家公司只做C/S客户端开发和测试，而另外一家公司做UI和前端。而且该美国公司有一个习惯，每周五都会在各个外包公司举行一个限时比赛（3个小时），比如比赛编写一个网络小应用、自动化脚本等，目的是提高团队成员的技术。第一名奖金是500美金。

机会就这样来了。周五的竞赛题目，周一就定下范围了，但是没给细节，只说是HTML编写网页方面的。这下韩先生高兴得不得了，因为他一直想转前端，但从未获得过领导的赏识，所以很珍惜这个机会。他找到我，一定得让我帮忙。于是，我们进行了一番沟通。

笔者：“你们客户挺大方的啊，每周搞比赛，还发500美元奖金。”

韩先生：“是的，有时候没东西可比了，就比赛跳绳呢。”我心想，客户有那么傻吗，肯定是怕你们身体垮了，才让你们体育锻炼的。

韩先生：“哥，这次你一定得帮我，除了测试，我就喜欢做前端，一定要赢得这次比赛。”

笔者：“我很好奇，你们一般技术比赛的相关技术，之后在项目上都用得上吗？”

笔者：“听说好像是都有用，我是做测试的，不太清楚。”我心想，这也许就是客户的目的，通过奖金让员工进步、挑战极限，然后节约培训成本，从而提供高质量的业绩。

笔者：“你们公司不是只做客户端和测试吗？让你们比赛网页制作干啥？”

韩先生：“不知道啊，可能是因为最近开发和测试的工作不多吧。”

笔者：“那你们要小心了，外包公司没活就意味着不景气。不过，等等！莫非因为活不多了，你们客户想在你们公司设置几个前端的职位？或者想把另外一个外包公司的活转给你们公司做，所以要先考核一下你们？”

韩先生：“谁知道呢，反正我就想拿奖金。哥，你一定得帮我。”

笔者：“做事之前一定得弄清楚目的。我觉得很有可能给你们开新职位，你一定要把握这次机会。你现在已经是测试组长了，如果前端项目给你们公司的话，说不定可以让你带队做前端呢，这可太好了。”

笔者：“先来分析分析。首先，客户有什么限制吗？如果能让用现成的框架就很快，如果不让用，4个小时完成一个好看的网页也是很困难的哦。”

韩先生：“允许用jQuery，还说CSS的框架随便选，比如Bootstrap、Foundation等。”

笔者：“那我估计，主要就是想考察这几个框架方面的内容。”

韩先生：“jQuery没问题，可是Bootstrap我只大概看了一下，不太会。”

笔者：“Bootstrap很简单，用心看一周绝对没问题，而且一般考查做网页，主要多注意几个点就行了。比如：菜单（多级菜单）、导航、弹窗、表格等，其中表格和弹窗要求的技术相对较多，你在这方面要多花一些时间。有问题，随时问我”

韩先生：“好的，我知道了。”

到了周五，还没下班，韩先生就打电话给我：“哥，告诉你个好消息：我拿奖金了，500美金！哈哈，我请你吃饭。”

晚上，韩先生就给我讲述了他的获奖过程。题目是这样的：制作一个用户列表页面，在该页面可以对用户信息进行查询、添加、删除、禁用等操作。

最终结果如图8-1所示。

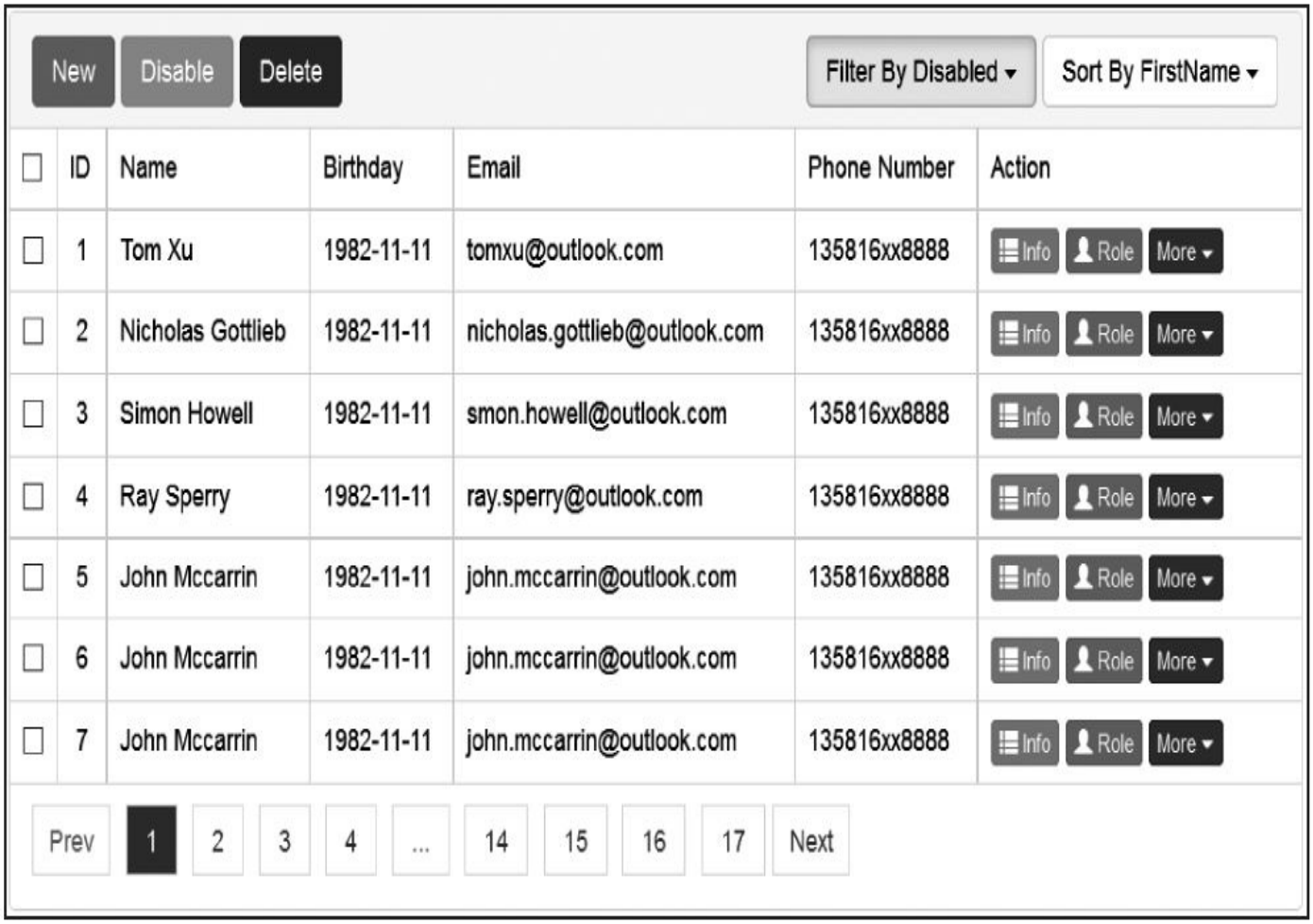


图8-1　测试题综合显示效果

8.2 实战过程

首先，看到题目的要求是制作一个列表管理，将它分为3部分：顶部的操作按钮和筛选条件、中间的列表内容、底部的分页码。

刚开始的时候，本来想只用一个table里的thead、tbody、tfoot实现这3个部分，但是后来在将第一列的checkbox居中对齐的时候，发现会对thead和tfoot有影响，所以就换成了panel组件、table以及list-group组件的组合了。示例代码如下：

```
<div class=“panel panel-default ”>
    <div class=“panel-heading”>
        这里是head区
    </div>
    <table class=“table table-bordered table-hover”>
        <thead></thead>
        <tbody></tbody>
        <tfoot></tfoot>
    </table>
    <ul class=“list-group”>
        <li class=“list-group-item”>
            这里放分页码
        </li>
    </ul>
</div>
```

panel里有3部分，panel-heading里放置操作按钮和筛选条件，中间放表格，并且在表格上应用了table-bordered样式，以便都有边框，另外还加了一个鼠标移动变化tr背景色的功能，显得富有动感。底部的list-group用于放置分页码，这里用list-group是因为该组件和panel一起用的时候，可以很好地处理横线和底部边框的效果，具体可以参考该组件的详细使用方法。上述代码的效果如图8-2所示。

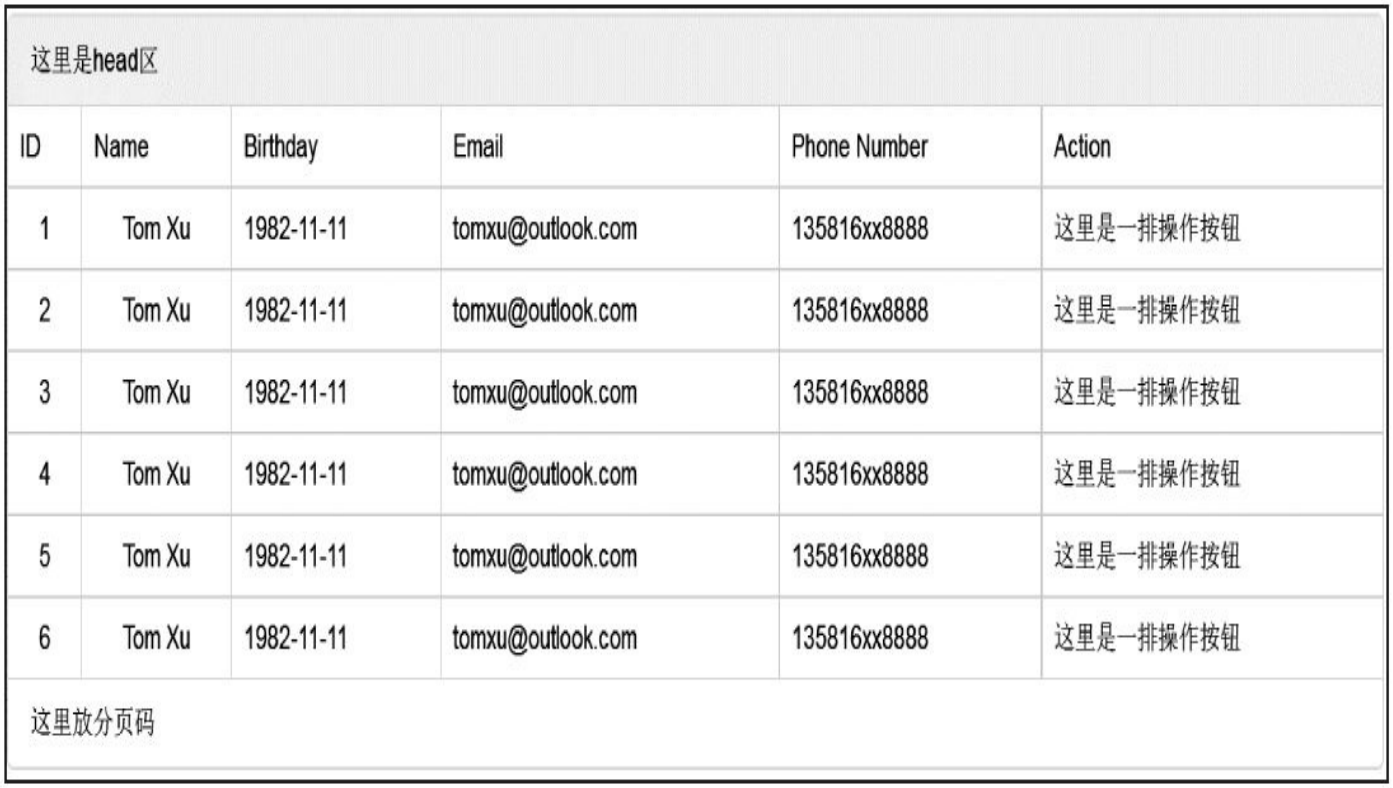


图8-2　pannel里设置表格显示

表格设计得很不错。下一步，因为我很喜欢分页码，所以先处理这一块，把相关代码放到list-group-item样式元素里。代码如下：

```
<li class="list-group-item">
    <ul class="pagination ">
        <li class="disabled"><span>Prev</span></li>
             <li  class="active"><a  href="#">1  <span  class="sr-only">
(current)
            </span></a></li>
        <li><a href="#">2</a></li>
        <li><a href="#">3</a></li>
        <li><a href="#">4</a></li>
        <li class="disabled"><span>…</span></li>
        <li><a href="#">14</a></li>
        <li><a href="#">15</a></li>
        <li><a href="#">16</a></li>
        <li><a href="#">17</a></li>
        <li><a href="#">Next</a></li>
    </ul>
</li>
```

这里先设置第一页为当前页，那么上一页（prev）就得被禁用掉。另外，在分页码很多的情况下，中间经常要省略一部分页面，所以在它上面也使用了disabled样式，运行效果如图8-3所示。

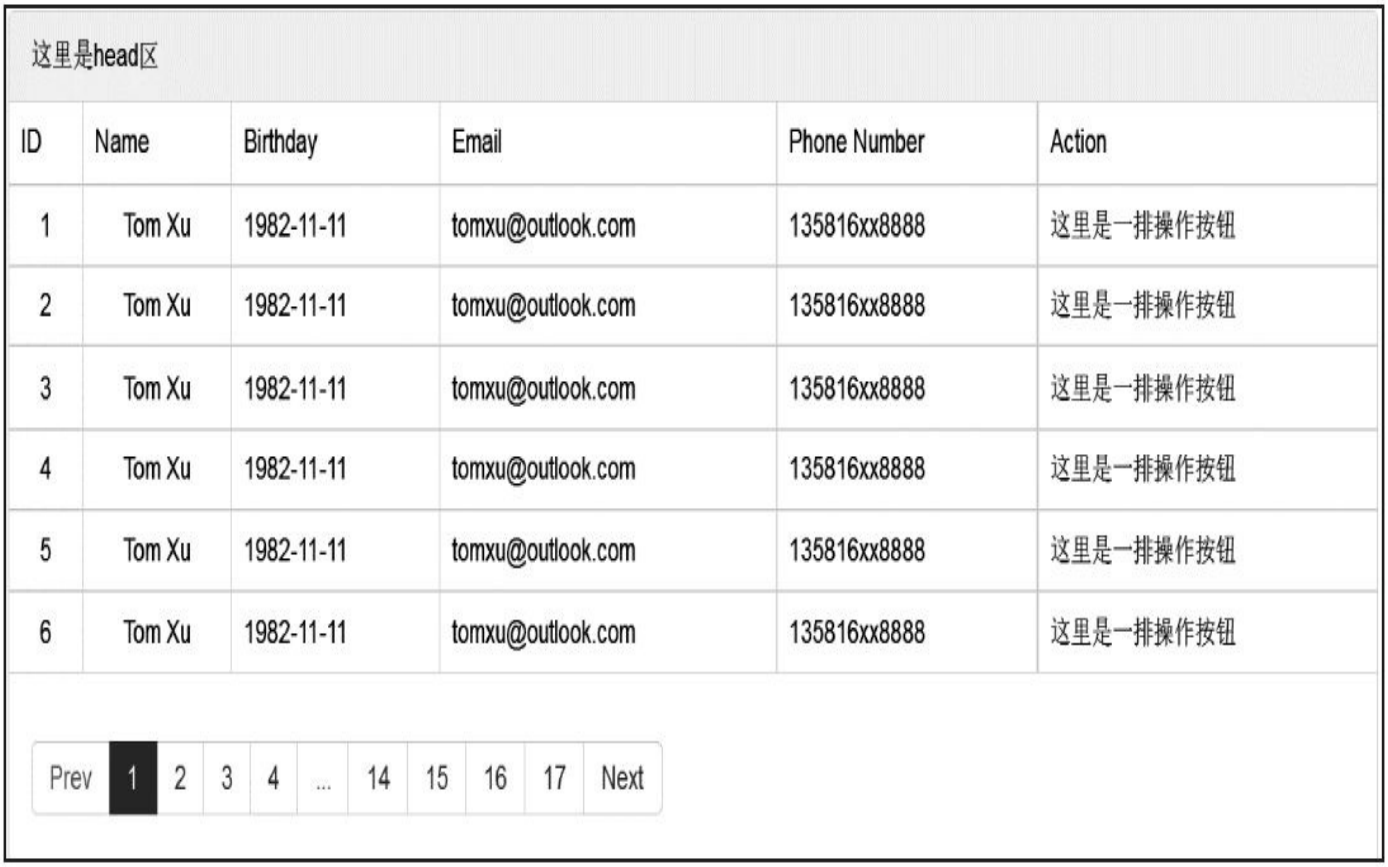


图8-3　分页组件效果

分页显示不是很好看，上下边距太大了，而且我想让每个页码都单独分开，这样显得更有个性。那么就来改造一下pagination样式。由于不能影响原有的分页功能，所以我们按照再附加一个square样式的情况进行改造，主要是要改造上下边距，以及每个li元素的外边距。代码如下：

```
.pagination.square {
    margin: 0;/* 去除外边距 */
}

.pagination.square > li > a,
.pagination.square > li > span {
    margin: 0 5px;
    border: 1px solid #dddddd; /* 设置所有的边框都为1像素 */
}

.pagination.square > li:first-child > a,
.pagination.square > li:first-child > span,
.pagination.square > li:last-child > a,
.pagination.square > li:last-child > span {
    margin-left: 0px;
    padding-left: 10px;
    padding-right: 10px;
    border-radius: 0px; /* 取消圆角 */
}
```

不仅把外边边距都调整了，还把前后元素的圆角也都去除了，形成一个很干净的方形效果，如图8-4所示。

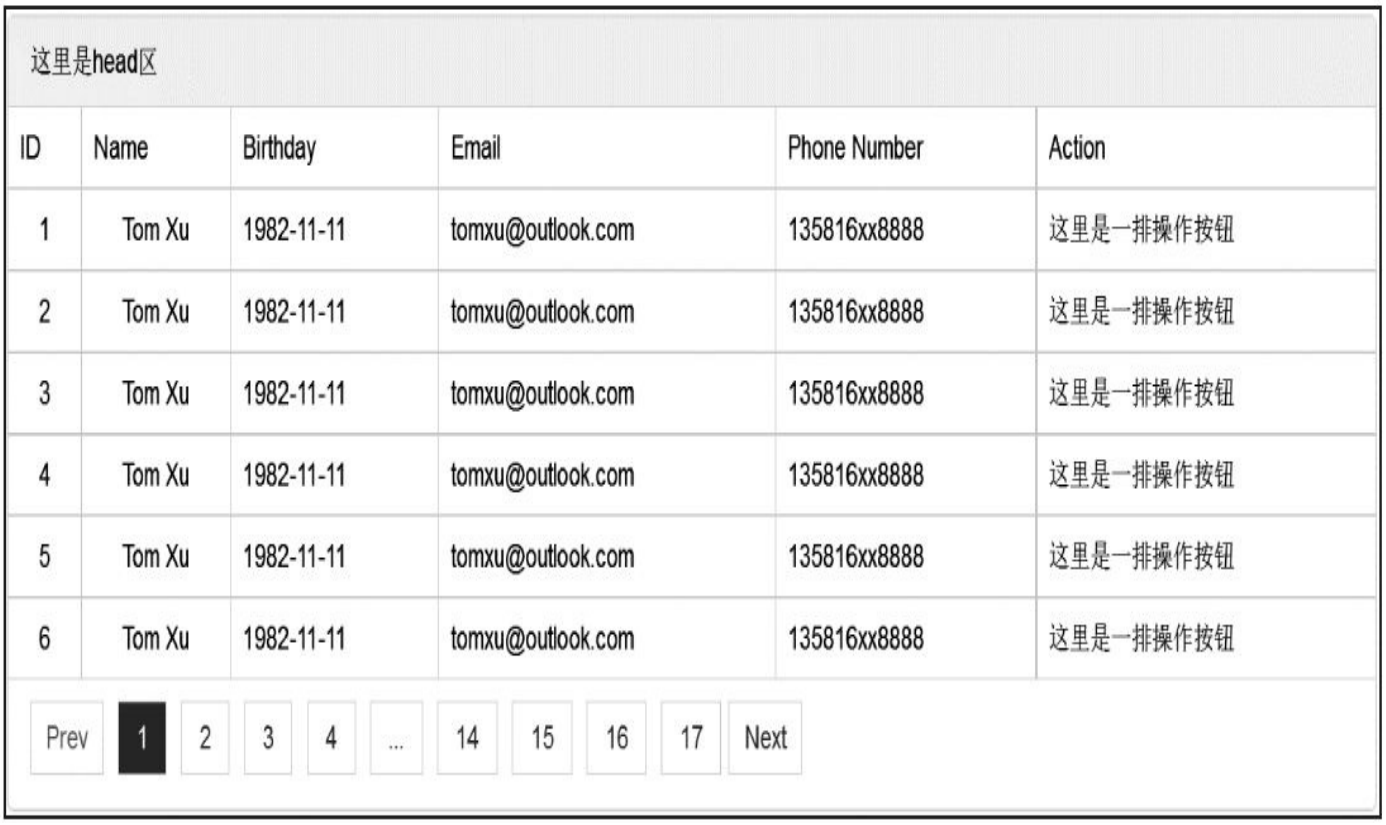


图8-4　分页数字方框间隔显示

现在可以处理head头部的内容了。在这里想将它分为左右两部分，左边放功能按钮，右边放筛选和排序选择菜单，所以加了两个div，分别是pull-left和pull-right，来实现左右浮动。代码如下：

```
<div class=“panel-heading”>
    <div class=“pull-right”>
        左边区域
    </div>
    <div class=“pull-left”>
        右边区域
    </div>
</div>
```

运行看效果，发现效果变形了（和table重叠了），原因是没有清除浮动。记得Bootstrap里有清除浮动的样式，查了一下手册，找到了clearfix，将该样式应用到panel-heading上即可，效果如图8-5所示。

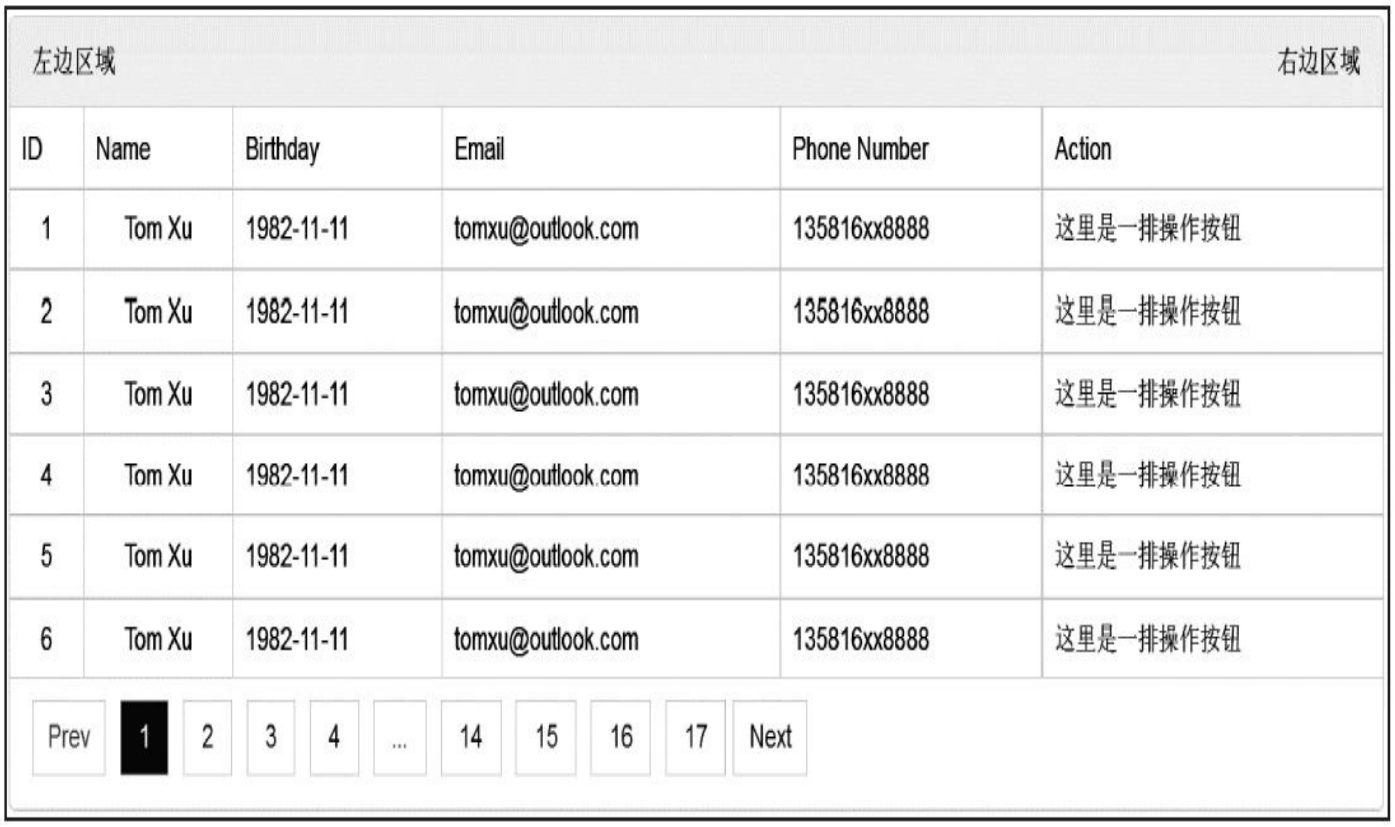


图8-5　将pannel头部分为左右两部分

左边的按钮很简单，在里面直接放3个button元素即可，然后用颜色样式稍微修饰一下。

```
<div class=“pull-left”>
    <button class=“btn btn-success”>New</button>
    <button class=“btn btn-warning”>Disable</button>
    <button class=“btn btn-danger”>Delete</button>
</div>
```

右边的排序和筛选条件就比较复杂了。刚开始的时候，想直接用select元素就可以了，但是该元素在不同的操作系统和浏览器下显示效果不一样，而且想体现一下对Bootstrap的了解，所以就用了下拉菜单（Dropdown）组件。排序和筛选分别用两个下拉菜单，其外部用按钮组（btn-group）样式包住，这样就差不多了。代码如下：

```
<div class=“pull-right”>
    <div class=“btn-group”>
        <button type=“button” class=“btn btn-default
            dropdown-toggle active” data-toggle=“dropdown”>
            Filter By Disabled <span class=“caret”></span>
        </button>
        <ul class=“dropdown-menu” role=“menu”>
            <li><a href=”#”>Filter by Account Status</a></li>
            <li><a href=”#”>Filter by Department</a></li>
            <li class=“divider”></li>
            <li><a href=”#”>Reset</a></li>
        </ul>
    </div>
```

```
        <div class="btn-group">
            <button type="button" class="btn btn-default
                dropdown-toggle" data-toggle="dropdown">
                Sort By FirstName <span class="caret"></span>
            </button>
            <ul class="dropdown-menu pull-right" role="menu">
                <li><a href="#">Sort by FirstName</a></li>
                <li><a href="#">Sort by Email</a></li>
                <li class="divider"></li>
                <li><a href="#">Reset</a></li>
            </ul>
        </div>
    </div>
```

看到两段代码有点一样了吧，筛选条件下拉菜单里，我将button的样式多加了一个active样式，主要是告诉后端集成的开发人员，如果他想把（已经选择的条件）作为高亮显示就加active样式就可以了，如果没有选择条件，就不用加active样式。

另外一个排序下拉菜单，我多用了一个pull-right，这是因为，刚开始预览的时候发现，如果下拉菜单里的文字太长，就会超出屏幕右侧（溢出了），加了pull-right样式后就可以让它右对齐了，效果如图8-6所示。

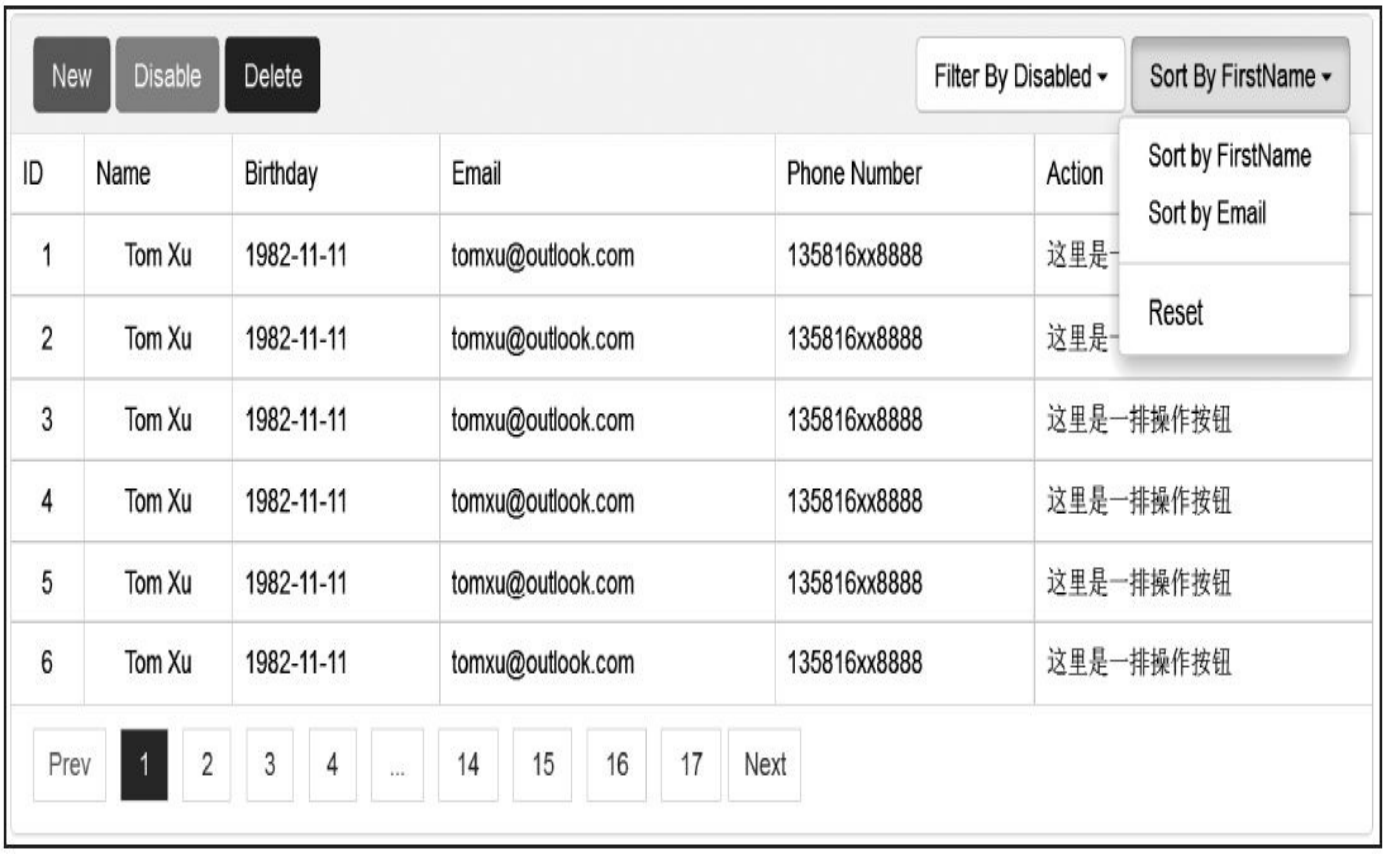


图8-6　在Pannel头部添加操作按钮和筛选条件

有点做前端开发的感觉了。看看再增加点什么？最后一列的按钮还没做，应该可以做至少3个按钮：查看详细信息、查看该用户的角色、更多操作（再弹出一个下拉菜单）。还要选择合适的颜色（由于使用默认

的btn样式会增加行高，所以使用了超小型按钮）。代码如下：

```
<button class=“btn btn-xs btn-info”>Info</button>
<button class=“btn btn-xs btn-success”>Role</button>
<div class=“btn-group”>
       <button  type=“button”  class=“btn  btn-xs  btn-primary  dropdown-
toggle”
        data-toggle=“dropdown”>
        More <span class=“caret”></span>
    </button>
    <ul class=“dropdown-menu” role=“menu”>
        <li><a href=”#”>Projects</a></li>
        <li><a href=”#”>Tasks</a></li>
        <li><a href=”#”>Timesheet</a></li>
        <li class=“divider”></li>
        <li><a href=”#”>Others</a></li>
    </ul>
</div>
```

上述示例的运行效果如图8-7所示。

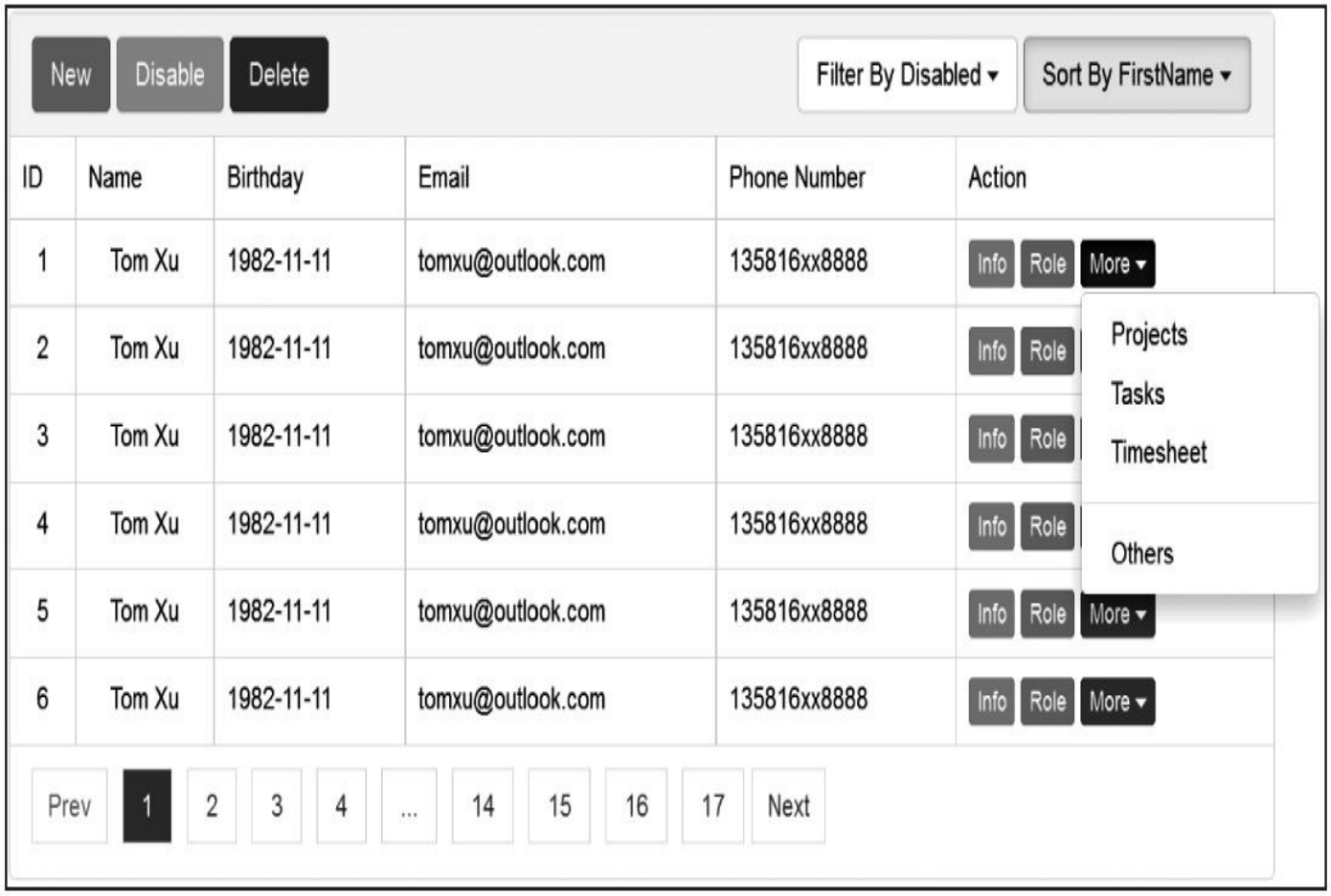


图8-7　表格行数据里的操作按钮显示

图8-7所示的效果看起来还行，但是稍微有点单调。添加一个小图标icon试试怎么样，示例如下：

```
<button class=“btn  btn-xs btn-info”>
<span class=“glyphicon glyphicon-list”> </span>Info
</button>
```

```
<button class="btn  btn-xs btn-success">
<span class="glyphicon glyphicon-user"> </span>Role
</button>
```

添加完图标以后，发现小图标和文字靠在一起了，所以又重新定义了一下小图标的右外边距。代码如下：

```
.btn > .glyphicon {
    margin-right: 3px;
}
```

这样，完成以后，发现运行效果真的比之前好太多了，效果如图8-8所示。

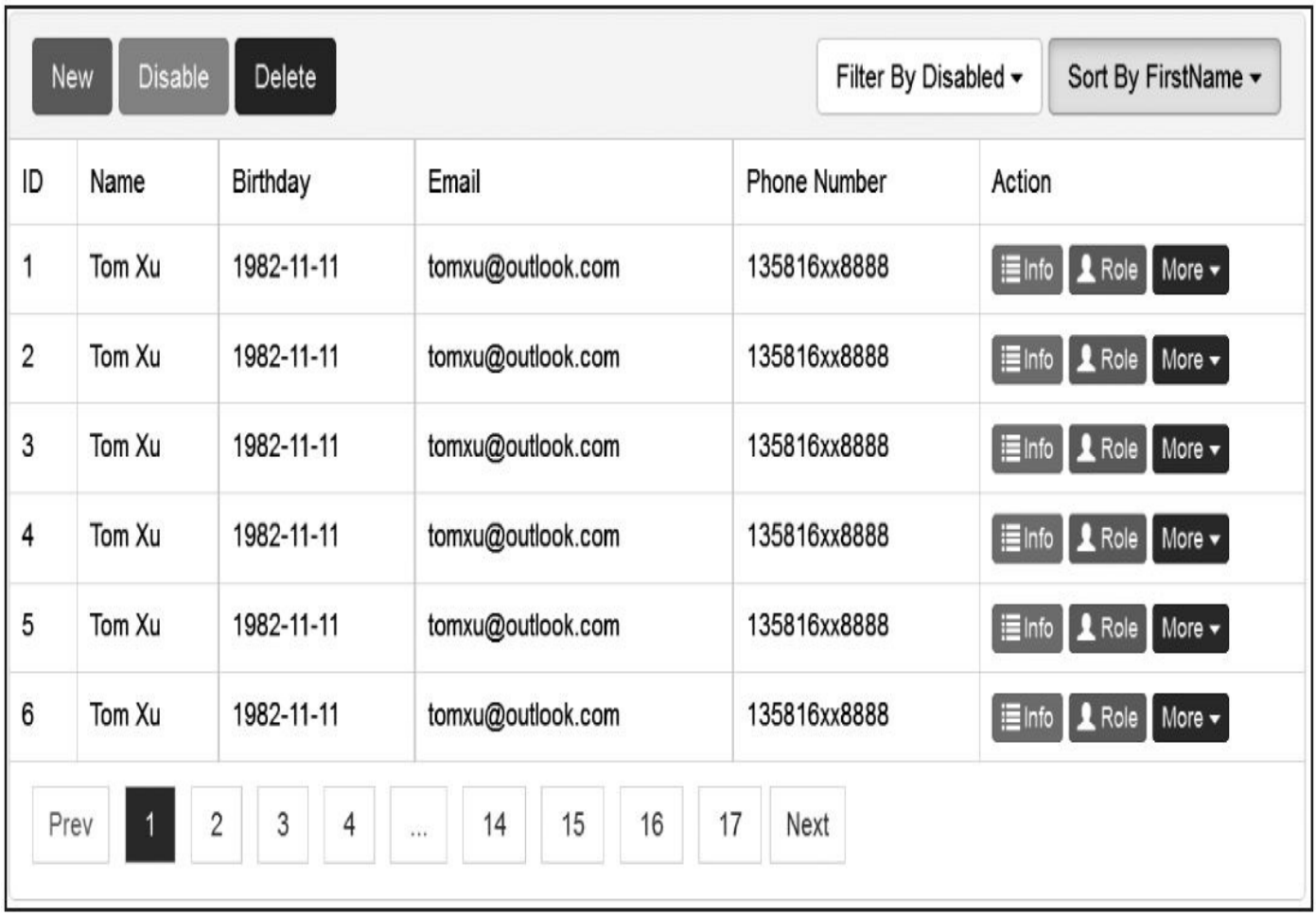


图8-8　综合显示效果

应该能给自己打80分了。但还有问题。第一列的全选按钮checkbox忘记加了，而且ID也没有居中对齐。

于是在第一列和第二列的所有的td上都加了一个text-center样式，但是发现这种方式添加，样式重复出现的次数太多了，为了用最少的代码实现最多的功能，决定使用CSS的选择器来实现（虽然还不太熟），试验了很多次才搞定（看来CSS3还得再强化才行）。代码如下：

```
thead > tr:not(:first-child) > td:nth-child(1),
thead > tr:not(:first-child) > td:nth-child(2),
```

```
tbody > tr > td:nth-child(1),
tbody > tr > td:nth-child(2) {
    text-align: center;
}
```

平时犯懒的时候肯定就只用一个样式，在所有的td上附加，但这次是竞赛，为了体现我还是懂一点CSS3的，所以使用了nth-child、first-child、not这样的选择器条件，保存刷新，效果如图8-9所示。

看着这样的效果，我自己都觉得挺满意了。但是时间还有1个多小时呢，应该再多做一点东西。想一下还要体现哪些能力。对了，弹窗还没用过，这个页面好多地方都可以用到弹窗，比如单击Info按钮的时候查看用户信息，单击Role按钮的时候查看角色信息，单击New按钮时也可以添加新用户。

由于Info按钮和Role按钮都是显示信息，所以我做了一个固定的弹窗，所有的单击事件都调用这一个弹窗（以后开发人员开发的时候再换id就行了）。首先，在所有的Role按钮上都添加了data-toggle和data-target，以便在单击的时候可以找到对应的id。代码如下：

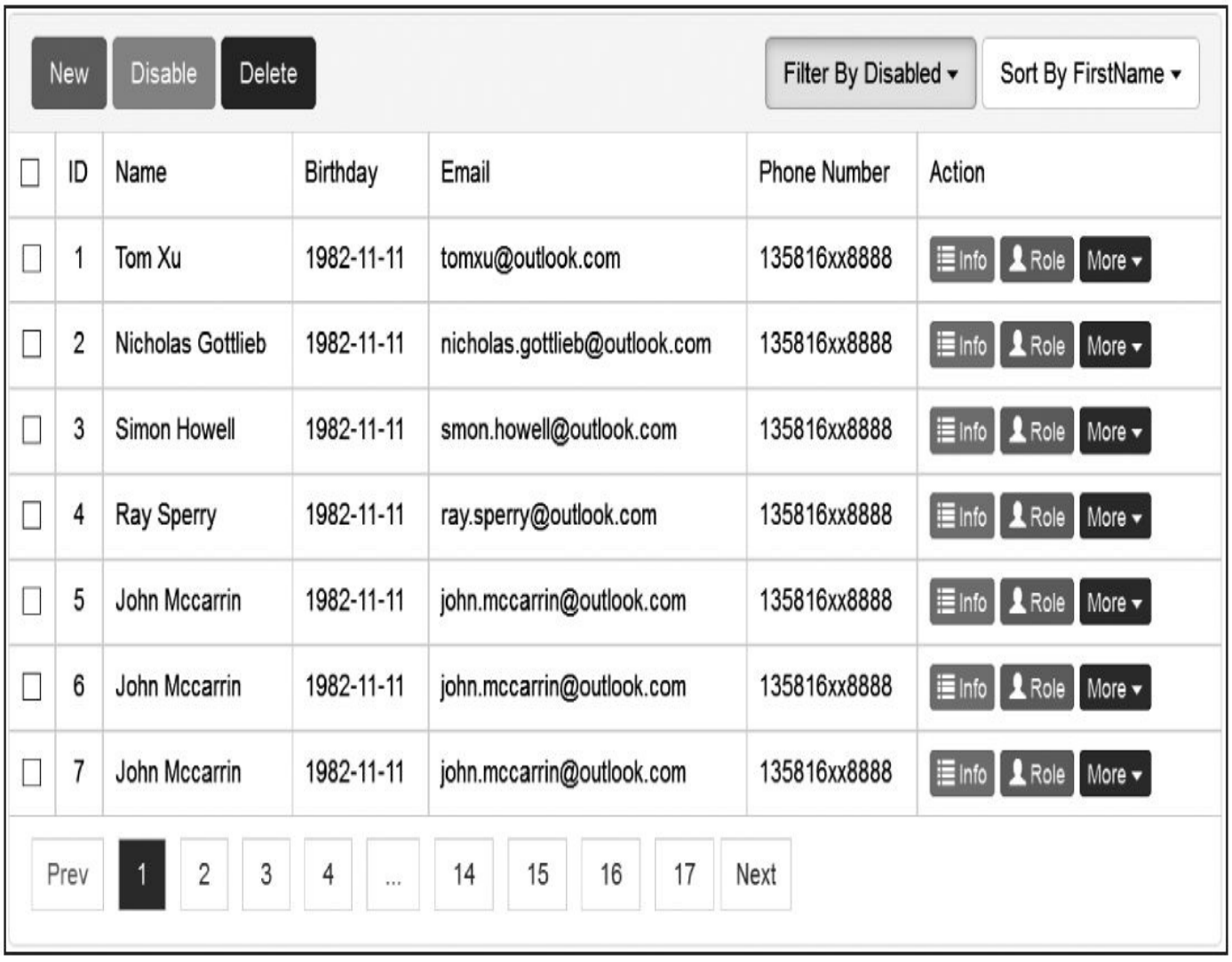


图8-9　复选框显示效果

```
<button class="btn  btn-xs btn-info" data-toggle="modal"
```

```
    data-target="#rolePopUp"> </button>
```

然后按照Modal插件的要求，设置了一个隐藏的div，并在其中modal-body里设置角色列表，最后还提供一个“关闭”按钮。代码如下：

```
<div class="modal fade" id="rolePopUp">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                            <button  type="button"  class="close"  data-
dismiss="modal"
                    aria-hidden="true">×</button>
                                                    <h4   class="modal-
title" id="myModalLabel">Roles of Tom Xu</h4>
            </div>
            <div class="modal-body">
                <table class="table table-hover">…</table>
            </div>
            <div class="modal-footer">
                     <button type="button" class="btn btn-default"data-
dismiss=
                    "modal">Close</button>
            </div>
        </div>
    </div>
</div>
```

上述示例的运行效果如图8-10所示。

同样的道理，在New按钮上也绑定一个弹窗，id为newPopup，只不过这一次弹窗的内容变成了表单。对于表单的字段，不用做太多工作，所以只用了email、password、Username这3个字段和一个复选框，但是按钮没有少，提供了3个，分别用于保存、重置和取消操作。表单的代码如下：

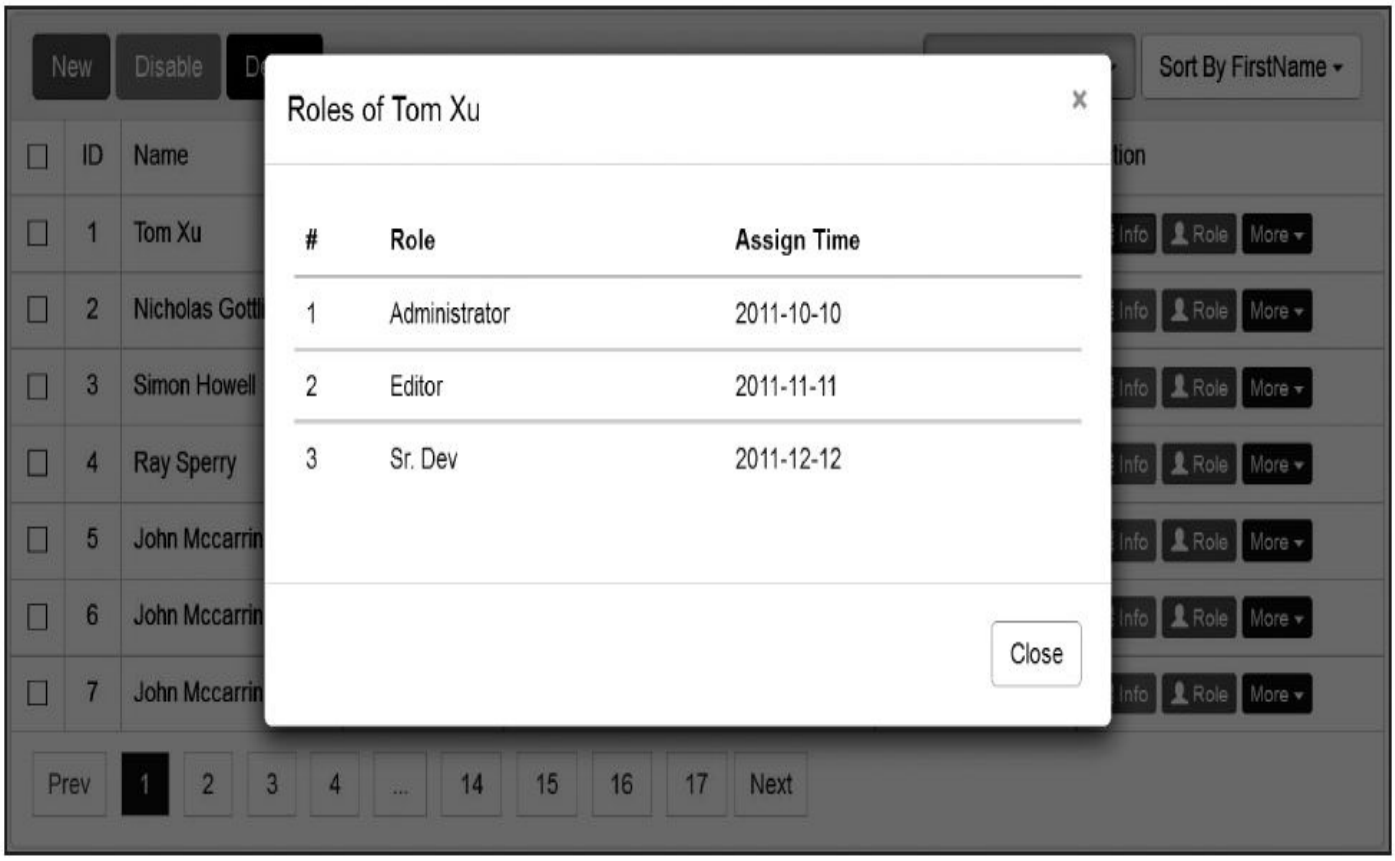


图8-10　弹窗显示效果

```
<form class=“form-horizontal” role=“form”>
    <div class=“form-group”>
                    <label  for=“inputEmail3”  class=“col-sm-2  control-
label”>Email</label>
        <div class=“col-sm-10”>
            <input type=“email” class=“form-control” id=“Email”
                placeholder=“Email”>
        </div>
    </div>
    <div class=“form-group”>
                <label  for=“inputPassword3”  class=“col-sm-2  control-
label”>Password</label>
        <div class=“col-sm-10”>
            <input type=“password” class=“form-control” id=“Password”
                placeholder=“Password”>
        </div>
    </div>
    <div class=“form-group”>
                <label  for=“inputPassword3”  class=“col-sm-2  control-
label”>Username</label>
        <div class=“col-sm-10”>
            <input type=“text” class=“form-control” id=“Username”
        placeholder=“Username”>
        </div>
    </div>
    <div class=“form-group”>
        <div class=“col-sm-offset-2 col-sm-10”>
            <div class=“checkbox”><label><input type=“checkbox”>Enable
                </label></div>
        </div>
```

```
    </div>
</form>
```

运行效果如图8-11所示。但是又发现鼠标单击灰色区域的时候，弹窗就关闭了，这说明缺少禁用这个事件了。于是又在New按钮上加了一个data-backdrop=“static”属性，这样就没问题了。

图8-11　带表单的弹窗显示效果

做到这时候，时间还余半个小时，突然想起来，对JavaScript方面的能力还没有体现过呢。正好Modal插件提供了好几个事件回调可以使用。

首先，我想在单击Create按钮的时候对表单进行验证，所以我引入了jQuery的validate插件。在定义JS代码的时候，引入validate库以后，首先进行了设置，然后对单击事件进行监控。代码如下：

```
$(‘#newPopUp’).find(‘#btnCreate’).on(‘click’, function (e) {
    $(‘#newPopUp’).find(“form”).validate({
        submitHandler: function (form) {
            // 处理内容
        }
    });
})
```

发现这样做一点儿用都没有，弄了10多分钟也没找出问题，后来只有10多分钟了。所以我就放弃了。但是还有最后5分钟的时候突然发现，可以先使用popover插件伪造一个验证效果代码如下：

```
$(‘#Username’).popover({
    title: “Alert”,
    content: “Username must be input.”,
    placement: “bottom”,
    trigger: “manual”
})

$(‘#newPopUp’).find(‘#btnCreate’).on(‘click’, function (e) {
    $(‘#Username’).popover(“show”);
})
```

这样在单击按钮的时候，会弹出提示，说用户输入不合法。这样做的目的只是给客户展示我想实现这样的效果（只不过不完善而已）。

最后又发现了一个问题，popover信息提示以后关闭不了，所以我就在input的focus上添加了一个回调，手动关闭整个信息提示。代码如下：

```
$(‘#Username’).focus(function () {
    $(‘#Username’).popover(“hide”);
});
```

就这样，总算完成了。最后在评奖的时候，我居然是全公司的第一名，是做得最好的。其他同事大部分都只完成一半，很多人都抱怨说，仅全选、反选的JavaScript代码都浪费了很长时间。我当初没有做这个就是怕代码麻烦，其实说实话，我也没有好的办法哦。

8.3 笔者的建议

总地来说，3个小时完成这么多内容，还是很不错的，但是真正了解Bootstrap以后，其实还可以做得更好，但是也需要一些相关的技巧。

举个例子，在使用modal的时候，对New按钮的弹窗内容，如果默认隐藏在当前页，使用jQuery的validate控件的话，就需要给Create按钮的click事件做一个回调来触发表单的submit事件，部分validate验证是没办法起作用的。但是如果这个表单的内容是采用ajax获取的（例如使用了MVC的model绑定），就算添加了回调也是没用的，原因是validate插件只对现有DOM对象里的表单进行验证，ajax获取的表单属于后来附加上去的，根本就没有绑定相关的validate事件。

所以上述问题的解决方案是，在modal弹出的show事件里重新绑定应用的插件。举个例子，笔者之前在Asp.net MVC项目里使用到了jquery.validate.unobtrusive.js插件，修改后的绑定代码如下：

```
$('#newPopUp').on('show.bs.modal', function (e) {
    $.validator.unobtrusive.parse($(this).find("form"));  // 对modal内的form重新
                                                          // 绑定validate事件
})
```

几乎绝大部分插件在modal上使用的时候都要进行这种重新绑定，比如BootSwitch、Date Time Picker等。它们的重新绑定代码如下：

```
$('#newPopUp').on('show.bs.modal', function (e) {
    var popup = $(this);

    if ($.fn.bootstrapSwitch) {
        popup.find("form .switch")['bootstrapSwitch']();
    }

    if ($.fn.datepicker) {
        popup.find('form .datepicker').datepicker().on('changeDate', fur
            $(this).datepicker('hide');
        });
    }
})
```

另外，针对上述表格的制作，笔者还有一些建议。相信在考虑了这些建议以后，制作出的网页，可能会让老板（或客户）更加满意。那就来看看都是什么样的小建议吧。

❑分页数字的方框宽度不一样，如果是3位数字，就会变得大小不一，需要根据实际情况调节。

❑鼠标hover时每个tr的背景色要换成自定义的颜色的话，还得再改造

一下。

❑如果要做响应式表格，又该如何呢？

❑目前只是一个表格，怎么融合到整个页面里？比如左边的菜单和顶部的菜单，如果有冲突该怎么办？

❑如果表格里没有记录，如何提示“无记录”这样的显示，并且优化相应的背景色。

❑表头的背景颜色能否自定义？

❑如果需要搜索功能，又该如何做？

❑如果需要设置每页显示的记录数（pagesize），该怎么放合适？

❑是否能找个地方添加一个跳转页码的功能？

❑如果提示“添加成功”、“删除成功”之类的信息的话，如何做带有动画过渡效果的提示？

❑如果modal使用ajax获取内容，是自己组装，还是使用remote参数？

❑如果是自己组装内容，可以使用模板绑定插件，比如使用Underscore插件进行模板处理。

❑如果使用remote参数的话，还要注意它会替换modal内的所有内容（包括head、body、foot）。

相信，针对这些小细节调整以后，会获得意想不到的效果。大家有时间的话，可以慢慢完善这个表格，相信大部分的Web系统都会使用到表格，届时你就会非常轻松了。

8.4 实战成果

虽然不是本书的重点，但还是想说一下韩先生最后的战果。除了500美元奖金以外，还有更让人羡慕的事情。

两个月后，客户新设置了两个head count的UI职位，其中韩先生任lead（似乎另外一个外包公司减少了2个UI职位）。韩先生实现了自己的愿望。同时韩先生的公司也很聪明，赶紧又给韩先生加薪，又请其他部门的高手专门加强UI的再培训工作，目的很明显，想把UI这一块做好，争取获得更多的head count，从而为公司带来更多的利润。

笔者认为，其实收益更大的是客户，因为客户通过这种方式降低了风险（主要是很多外包公司在客户面试的时候会作弊），确保了新增加的人有能力做一件事情，然后才确定给外包公司新增加职位。

8.5 所用技术总结

我们来总结一下这次实战用到的Bootstrap的功能。主要功能如下：栅格系统、基础排版、表格、表单、按钮、辅助样式、icon图标、下拉菜单、按钮组、按钮下拉菜单、输入框组、分页导航、标签、列表组、面板、动画、弹窗、弹出框。

如下组件或插件还没有用到，大家可以自己制作一个完整的页面，争取将下面的功能都使用一遍：代码、图像、导航、导航条、面包屑、徽章、大屏幕展播、缩略图、警告框、进度条、媒体对象、well、选项卡、提示框、折叠、旋转轮播、affix。

第9章　第三方扩展

本章主要讲解目前比较常用的一些Bootstrap扩展，比如字体图标（Font　Awesome）、时间选择器、按钮等，这些扩展确实非常强大，能在Bootstrap的基础上为我们增加不少光彩。本章主要介绍Font Awesome，其他的只进行概要介绍。

9.1 Font Awesome

9.1.1 介绍

Font Awesome是一款强大的icon图标集，可以进行矢量缩放，支持任意CSS对大小、颜色、阴影等的控制操作。目前最新版本是V4.0.3，它拥有如下9大特性：

❑一个字体文件，369个图标。

❑完全免费，可以商用。

❑完全兼容Bootstrap。

❑可以直接利用CSS控制图标的大小、颜色、阴影等，只要CSS支持的都可以。

❑支持IE7（需要应用兼容性文件：font-awesome-ie7.css）。

❑友好的使用方式，样式和普通文本（unicode格式）均可。

❑无限的可伸缩性。

❑Retina屏上的完美显示。

❑完全兼容屏幕阅读器。

其下载地址是http://fontawesome.io/assets/font-awesome.zip。

因为和Bootstrap完全兼容（Bootstrap以glyphicon-开头，Font Awesome以icon-开头），所以使用方式非常简单，在Bootstrap之后直接引用CSS文件即可。示例如下：

```
<link rel="stylesheet" href="/css/font-awesome.min.css">
<!–[if IE 7]>
  <link rel="stylesheet" href="/css/font-awesome-ie7.min.css">
<![endif]–>
```

如果要支持IE7版本浏览器，则需要加上上述内的兼容性文件代码。

注意

❑所有的icon都可以在http://fontawesome.io/icons/上找到。

❑所有icon对应的样式和Unicode编码都可以在http://fontawesome.io/cheatsheet/上找到。

9.1.2 常规用法

其常规用法和Bootstrap里的icon一样，只需要在内联元素上应用相应的样式即可。

```
<i class="icon-camera-retro"></i> icon-camera-retro
```

不同的是，对于所有的图标，Font Awesome提供了4种缩放大小的设置样式，分别是：.icon-large、icon-2x、icon-3x和icon-4x，主要是对图标放大相应的倍数。

```
<p><i class="icon-camera-retro icon-large"></i> 放大1.33倍</p>
<p><i class="icon-camera-retro icon-2x"></i> 放大2倍</p>
<p><i class="icon-camera-retro icon-3x"></i> 放大3倍</p>
<p><i class="icon-camera-retro icon-4x"></i> 放大4倍</p>
```

icon图标集也支持Bootstrap里的左右浮动功能：pull-left、pull-right。用法如下：

```
<i class="icon-quote-left icon-4x pull-left icon-muted"></i>
```

上述示例的运行效果如图9-1所示。

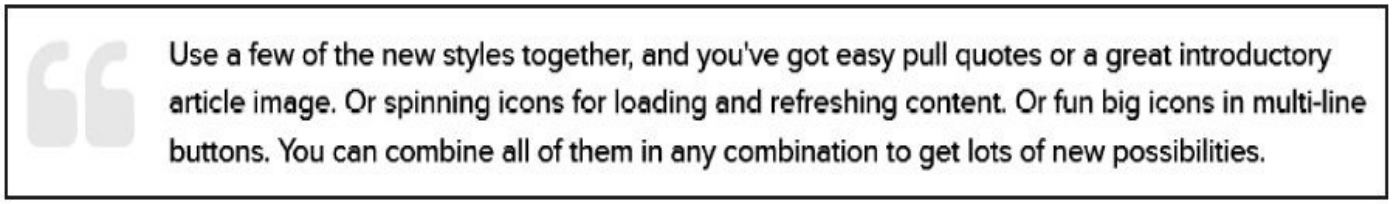


图9-1 引用文本效果

同时，通过应用icon-border样式，也可以为icon图标添加一个带圆角的方框。用法如下：

```
<i class="icon-flag icon-4x pull-left icon-border"></i>
```

上述示例的运行效果如图9-2所示。

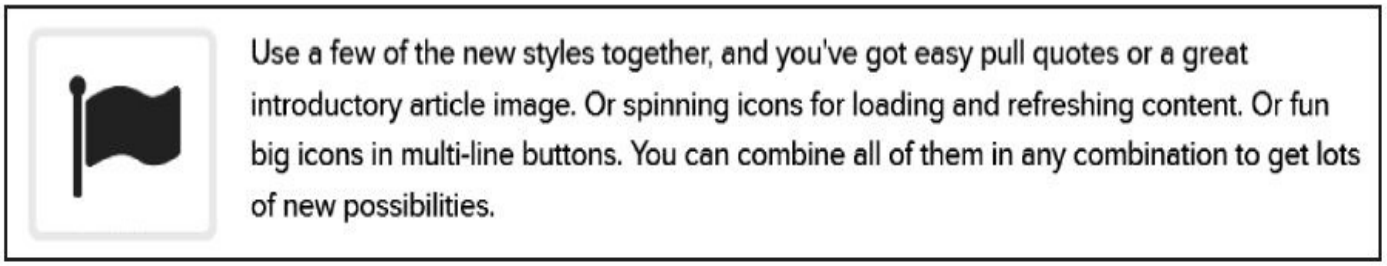


图9-2 icon图标修改后效果

icon样式在其他元素（比如Button、链接或者add-on）上的使用方式和Bootstrap提供的glynhicon样式用法一致，可以一同使用。

9.1.3　List列表上的图标

经常使用数字（或者圆点）显示列表li元素的界面已经太古老了，Font Awesome提供了令人欣喜的新功能，即将自定义图标作为li元素的标示符显示，将icons-ul和icon-li分别应用在ul和li元素上。示例如下：

```
<ul class=“icons-ul”>
     <li><i class=“icon-li icon-ok”></i>Bulleted lists (like this one)
</li>
    <li><i class=“icon-li icon-ok”></i>Buttons</li>
    <li><i class=“icon-li icon-ok”></i>Button groups</li>
    <li><i class=“icon-li icon-ok”></i>Navigation</li>
    <li><i class=“icon-li icon-ok”></i>Prepended form inputs</li>
                          <li><i      class=“icon-li      icon-ok”></i>…
and many more with custom CSS</li>
</ul>
```

上述示例的运行效果如图9-3所示。

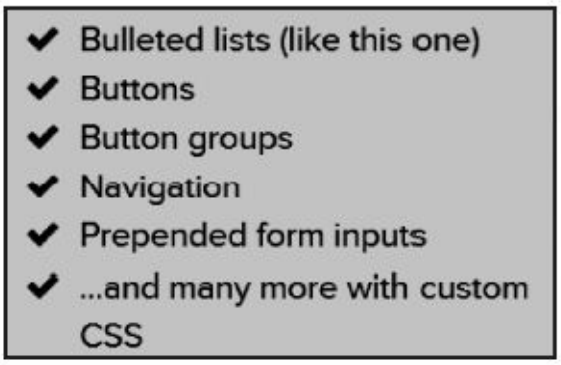


图9-3　列表上的icon显示效果

icon相关的样式源码如下：

```
// 源码74行
ul.icons-ul {
    list-style-type: none;
    text-indent: -0.7142857142857143em;
    margin-left: 2.142857142857143em;
}
ul.icons-ul > li .icon-li {
    width: 0.7142857142857143em;
    display: inline-block;
    text-align: center;
}
```

9.1.4　导航上的图标

在导航菜单上显示icon图标的方式与4.1节Bootstrap默认的使用方式一样（即使用带有nav　nav-pills　nav-stacked样式的ul列表），唯一不同的是，Font　　Awesome也可以在普通的nav样式上使用。示例如下（.icon-fixed-width用于增大字体，和图标保持一致）：

```
<ul class="nav ">
      <li class="active"><a href="#"><i class="icon-fixed-width icon-
home">
    </i> Home</a></li>
            <li><a  href="#"><i  class="icon-fixed-width  icon-book">
</i> Library</a></li>
    <li><a href="#"><i class="icon-fixed-width icon-pencil"></i>
        Applications</a></li>
    <li><a href="#"><i class="icon-fixed-width icon-cogs"></i> Settings
        </a></li>
</ul>
```

上述示例的运行效果如图9-4所示。

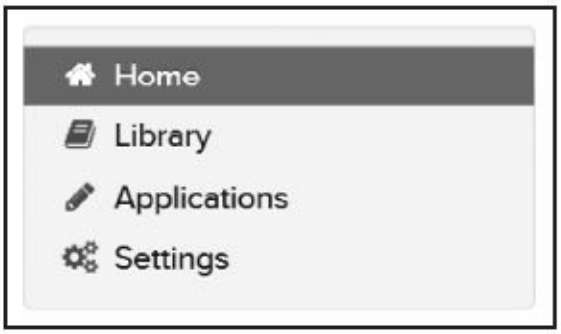


图9-4　导航上的icon显示效果

9.1.5　固定角度旋转

Font　Awesome提供了一组样式，用于旋转特定的图标，从而达到“一种图标六种用法”的目的，其样式和旋转角度如表9-1所示。

表 9-1　icon 旋转样式

额外样式	显示效果	额外样式	显示效果
只保留原有的图标样式	正常显示	icon-rotate-270	旋转 270 度
icon-rotate-90	旋转 90 度	icon-flip-horizontal	水平翻转
icon-rotate-180	旋转 180 度	icon-flip-vertical	垂直翻转

旋转的用法如下：

```
<i class=“icon-shield”></i>  正常<br>
<i class=“icon-shield icon-rotate-90”></i>  旋转90度<br>
<i class=“icon-shield icon-rotate-180”></i>  旋转180度<br>
<i class=“icon-shield icon-rotate-270”></i>  旋转270度<br>
<i class=“icon-shield icon-flip-horizontal”></i>  水平翻转<br>
<i class=“icon-shield icon-flip-vertical”></i>  垂直翻转
```

上述示例的运行效果如图9-5所示。

图9-5　旋转效果

上述5个旋转样式的实现原理是基于CSS3的transform特性实现的。源码如下：

```
// 源码316行
/* Icon rotations and mirroring */
.icon-rotate-90:before { /* 旋转90度 */
    -webkit-transform: rotate(90deg);
    -moz-transform: rotate(90deg);
    -ms-transform: rotate(90deg);
    -o-transform: rotate(90deg);
    transform: rotate(90deg);
    filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=1);
}
.icon-rotate-180:before { /* 旋转180度 */
```

```
    -webkit-transform: rotate(180deg);
    -moz-transform: rotate(180deg);
    -ms-transform: rotate(180deg);
    -o-transform: rotate(180deg);
    transform: rotate(180deg);
    filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=2);
}
.icon-rotate-270:before { /* 旋转270度 */
    -webkit-transform: rotate(270deg);
    -moz-transform: rotate(270deg);
    -ms-transform: rotate(270deg);
    -o-transform: rotate(270deg);
    transform: rotate(270deg);
    filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=3);
}
.icon-flip-horizontal:before { /* 水平翻转 */
    -webkit-transform: scale(-1, 1);
    -moz-transform: scale(-1, 1);
    -ms-transform: scale(-1, 1);
    -o-transform: scale(-1, 1);
    transform: scale(-1, 1);
}
.icon-flip-vertical:before { /* 垂直翻转 */
    -webkit-transform: scale(1, -1);
    -moz-transform: scale(1, -1);
    -ms-transform: scale(1, -1);
    -o-transform: scale(1, -1);
    transform: scale(1, -1);
}
```

如果大家需要，也可以根据自身情况定义任意角度的旋转样式。

注意

由于使用了CSS3的特性，所以该旋转功能不支持IE7及以下版本。

9.1.6　360度旋转

Font Awesome不仅提供了固定角度的旋转功能，还提供了360度持续旋转的功能，开发刷新、加载功能的时候特别有用，用的时候只需要在所应用图标样式上再加一个icon-spin样式即可。示例如下：

```
<i class="icon-spinner icon-spin"></i> 360度持续旋转，loading内容<br />
<i class="icon-refresh icon-spin"></i> 360度持续旋转，刷新内容<br />
<i class="icon-twitter icon-spin"></i> twitter小鸟也能360度持续旋转<br />
```

上述示例的运行效果如图9-6所示。

图9-6　持续旋转效果

再配合.icon-large样式，就可以放大显示了，这比Bootstrap自带的icon功能集强大多了。spin样式的源码如下：

```
// 源码269行
/* Animated rotating icon 主要源码 */
.icon-spin {
    display: inline-block;
    -moz-animation: spin 2s infinite linear;
    -o-animation: spin 2s infinite linear;
    -webkit-animation: spin 2s infinite linear;
    animation: spin 2s infinite linear;
}
@-moz-keyframes spin {
    0% {    -moz-transform: rotate(0deg);  }
    100% {    -moz-transform: rotate(359deg);  }
}
/* 针对特殊浏览器的设定省略，请查阅相应源码 */
```

注意

由于使用了CSS3的spin特性，所以该旋转功能不支持IE9及以下版本。

9.1.7　多图叠加

另外一个比较令人赞叹的特性是，Font　Awesome的图标支持多个叠加在一起组成复合特效的图标，而使用这个特性只需要多应用两个样式即可。示例如下：

```
<span class="icon-stack">
    <i class="icon-check-empty icon-stack-base"></i>
    <i class="icon-twitter"></i>
</span>在icon-check-empty上应用icon-twitter<br>

<span class="icon-stack">
    <i class="icon-circle icon-stack-base"></i>
    <i class="icon-flag icon-light"></i>
</span>在icon-circle上应用icon-flag，并设置icon-flag为icon-light白色<br>

<span class="icon-stack">
    <i class="icon-sign-blank icon-stack-base"></i>
    <i class="icon-terminal icon-light"></i>
</span>在icon-sign-blank上应用icon-terminal，并设置
    icon-terminals为icon-light白色
```

上述代码通过组合6个不同的图标，可以形成3个新的复合图标，如图9-7所示。看起来很不错吧？

图9-7　多图叠加效果

上述效果用到了两个样式：icon-stack（作为容器样式）、icon-stack-base（作为底部样式）。源码如下：

```
// 源码244行
/* Stacked and layered icon */
.icon-stack {
    position: relative;
    display: inline-block;
    width: 2em;
    height: 2em;
    line-height: 2em;
    vertical-align: -35%;
}
.icon-stack [class^="icon-"],
.icon-stack [class*=" icon-"] {
    display: block;
```

```
    text-align: center;
    position: absolute;                 /* 相对于icon-stack-base元素绝对定
位 */
    width: 100%;
    height: 100%;
    font-size: 1em;
    line-height: inherit;
    *line-height: 2em;
}
.icon-stack .icon-stack-base {
    font-size: 2em;
    *line-height: 1em;
}
```

9.2 BSIE扩展

由于最新的Bootstrap已经不支持IE8以及低版本了，所以在这些版本的浏览器上进行开发就会有很多困难，尤其是IE6浏览器在中国还有人在用，所以有人在网上开发了一个针对IE6的兼容库，名称为BSIE（Bootstrap IE6 Compatible Library），官方网址是https://github.com/ddouble/bsie。

有点遗憾的是，该扩展还没有完全支持最新的Bootstrap 3.x版，模板支持比较好的是2.2.1版本，希望作者尽快升级，解决还在为IE6努力的开发者们。BSIE目前支持的组件和功能如表9-2所示。

表 9-2 BSIE 组件功能列表

组件名称	功 能	组件名称	功 能
grid	fixed、fluid	badges	所有
navbar	top、fixed	Code	所有
nav	list、tabs、pills	Modal	大部分
dropdown	dropdown (two level)	Tooltip	所有
buttons	button、group color、size、dropdown-button,（disable 状态不是自动的）	popover	所有
		Alert	所有
form	default、horizontal、inline、all controls、validation state	typeahead	所有
		progressbar	大部分
tables	Hover	Media	所有
breadcrumbs	所有	Well	所有
pagination	所有	hero unit	所有
Labels	所有	Icon	所有

该插件在引用的时候需要特别注意，CSS需要引用两个特别的文件。示例如下：

```
<link rel="stylesheet" type="text/css" href="bootstrap/css/bootstrap.mir
<!—[if lte IE 6]>
<!— BSIE CSS 补丁文件 —>
<link  rel="stylesheet"  type="text/css"  href="bootstrap/css/bootstrap-
```

```
ie6.css”>
<!– BSIE 额外的 CSS 补丁文件 –>
<link rel=“stylesheet” type=“text/css” href=“bootstrap/css/ie.css”>
<![endif]–>
```

而对于JS文件，则只需要引入一个即可。示例如下：

```
<script type=“text/javascript” src=“js/jquery-1.10.1.min.js”></script>
<script     type=“text/javascript”     src=“bootstrap/js/bootstrap.js”>
</script>

<!–[if lte IE 6]>
<!– BSIE JS 补丁只在IE6中才执行 –>
<script type=“text/javascript” src=“js/bootstrap-ie.js”></script>
<![endif]–>
```

另外一个特殊情况是，如果页面里有ajax加载的内容，而且这些内容又要支持IE6的特殊功能，那么需要调用如下语句，才能让IE6工作正常。代码如下：

```
// 让el容器中的所有元素都能兼容IE6
$.bootstrapIE6(el);
```

其他在IE6下使用的相关注意事项，请参考官方网站的文档。

9.3 Buttons

Buttons是一个基于Sass和Compass构建的CSS按钮（button）样式库（官方网站是http://alexwolfe.github.io/Buttons/），其对按钮的支持度远远高于Bootstrap默认的按钮。笔者在写作的过程中本想以btn样式为例进行扩展写一篇实战，但是后来发现这个Buttons以后，就放弃了，因为它真的很强大，拥有各种各样的按钮、各种各样的样式、各种各样的效果。

首先在官方网站下载该插件，然后引入。由于Buttons插件使用了Font Awesome，所以引入的示例如下：

```
<link rel="stylesheet" href="css/font-awesome.css">
<link rel="stylesheet" href="css/buttons.css">

<script type="text/javascript" src="js/jquery-1.10.1.min.js"></script>
<script type="text/javascript" src="js/buttons.js"></script>
```

注意，只有在需要使用小图标的时候才引入font-awersome.css文件。两个JS文件也是可选的，只有在使用该插件里的下拉菜单功能时，才需要引入这两个JS文件。

对于该插件的功能，我们只打算列一下功能列表和使用方法，不分析其源码（因为这可以单独写一本书了）。常见的运行效果如图9-8所示。

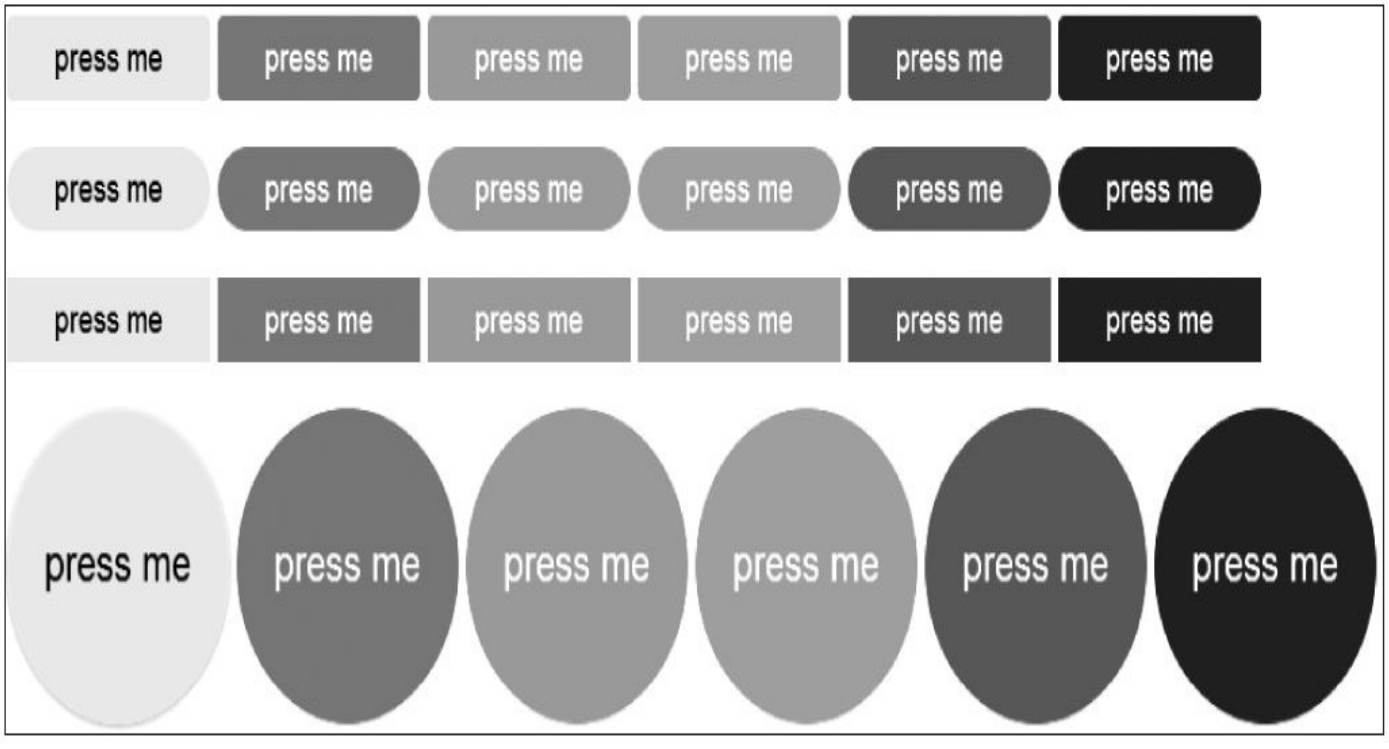


图9-8　扁平化按钮

上述效果的基本用法如下：

```
<a href=”#” class=“button button-flat”>press me</a>
<a href=”#” class=“button button-flat-primary”>press me</a>
<a href=”#” class=“button button-flat-action”>press me</a>
<a href=”#” class=“button button-flat-highlight”>press me</a>
<a href=”#” class=“button button-flat-caution”>press me</a>
<a href=”#” class=“button button-flat-royal”>press me</a>
```

上述示例是默认效果，也就是图9-8中的第三种效果。从示例中也可以看出，颜色风格和Bootstrap并不一致，而是使用了primary、action、highlight、caution以及royal名称。

而对于第1、2、4种效果，则只需要分别再多应用一个风格样式即可，它们是：button-rounded、button-pill以及button-circle。部分示例如下：

```
<a href=”#” class=“button button-rounded button-flat”>press me</a>
<a href=”#” class=“button button-rounded button-flat-primary”>press me</a>

<a href=”#” class=“button button-pill button-flat-action”>press me</a>
<a href=”#” class=“button button-pill button-flat-highlight”>press me</a>

<a href=”#” class=“button button-circle button-flat-caution”>press me</a>
<a href=”#” class=“button button-circle button-flat-royal”>press me</a>
```

而如果需要发光效果的按钮，则需要添加一个glow样式。示例代码如下：

```
<a href=”#” class=“button glow ”>press me</a>
<a href=”#” class=“button glow button-primary”>press me</a>
<a href=”#” class=“button glow button-action”>press me</a>
<a href=”#” class=“button glow button-highlight”>press me</a>
<a href=”#” class=“button glow button-caution”>press me</a>
<a href=”#” class=“button glow button-royal”>press me</a>
```

上述示例的运行效果如图9-9所示。

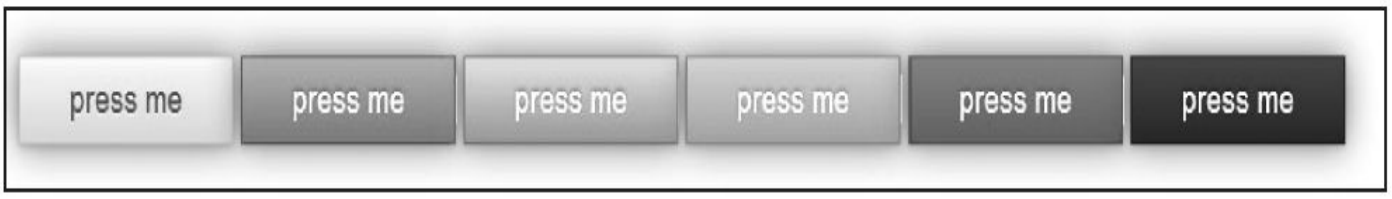


图9-9　发光效果的按钮

如果在第一个例子中，取消button-flat开头的样式，保留button-rounded样式，则会是默认的立体按钮。示例如下：

```
<a href=”#” class=“button button-rounded”>press me</a>
<a href=”#” class=“button button-rounded button-primary”>press me</a>
<a href=”#” class=“button button-rounded button-action”>press me</a>
```

```
<a href=”#” class=“button button-rounded button-highlight”>press me</a>
<a href=”#” class=“button button-rounded button-caution”>press me</a>
<a href=”#” class=“button button-rounded button-royal”>press me</a>
```

上述示例的运行效果如图9-10所示。

图9-10　立体效果按钮

如果需要3D按钮，则如下定义样式就可以了：

```
<a href=”#” class=“button button-3d”>press me</a>
<a      href=”#”      class=“button      button-3d-primary      button-
rounded”>press me</a>
<a href=”#” class=“button button-3d-action button-pill”>press me</a>
<a      href=”#”      class=“button      button-3d-highlight      button-
circle”>press me</a>
<a href=”#” class=“button button-3d-caution”><i class=“fa fa-camera”>
    </i> press me</a>
<a href=”#” class=“button button-3d-royal button-rounded”>press me</a>
```

上述示例的运行效果如图9-11所示。

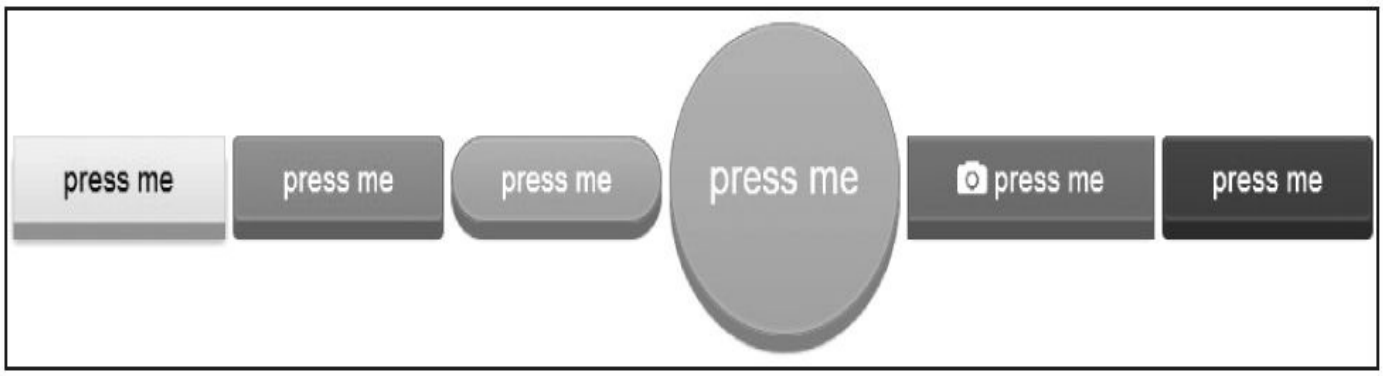


图9-11　3D效果按钮

如果想保留边框效果，那可以如下定义：

```
<a href=”#” class=“button button-border”>press me</a>
<a     href=”#”     class=“button     button-border-primary     button-
rounded”>press me</a>
<a     href=”#”     class=“button     button-border-action     button-
pill”>press me</a>
<a    href=”#”    class=“button    button-border-highlight    button-
circle”>press me</a>
<a  href=”#”  class=“button  button-border-caution”><i  class=“fa  fa-
camera”></i> press me</a>
<a href=”#” class=“button button-border-royal button-pill”>press me</a>
```

上述示例的运行效果如图9-12所示。

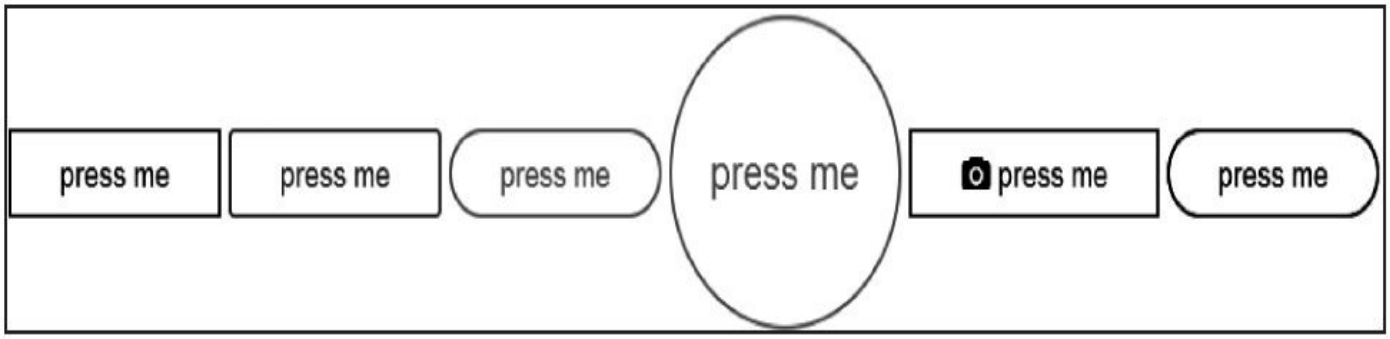


图9-12　带边框效果的按钮

如果要定义按钮的大小，则可以像如下示例这样使用：

```
<a href=”#” class=“button button-rounded button-flat-primary
    button-large”>press me</a>
<a     href=”#”     class=“button     button-rounded     button-flat-
primary”>press me</a>
<a href=”#” class=“button button-rounded button-flat-primary
    button-small”>press me</a>
<a  href=”#”  class=“button  button-rounded  button-flat-primary  button-
tiny”>
    press me</a>
```

上述示例的运行效果如图9-13所示。

图9-13　不同尺寸的按钮

而且还可以在按钮上添加小图标。示例如下：

```
<a href=”#” class=“button button-rounded button-flat”><i class=
    “fa fa-github”></i> press me</a>
<a href=”#” class=“button button-rounded button-flat-primary”><i class=
    “fa fa-refresh”></i> press me</a>
<a href=”#” class=“button button-rounded button-flat-action”><i class=
    ““fa fa-cloud”></i> press me</a>
<a   href=”#”   class=“button   button-rounded   button-flat-highlight”>
<i class=
    “fa fa-camera”></i> press me</a>
<a href=”#” class=“button button-rounded button-flat-caution”><i class=
    “fa fa-code”></i> press me</a>
<a href=”#” class=“button button-rounded button-flat-royal”><i class=
    “fa fa-envelope”></i> press me</a>
```

上述示例的运行效果如图9-14所示。

图9-14　按钮图标

上面这些功能是Buttons插件中的很少一部分，详细的功能列表，请访问官方网站进行查询。我个人觉得，一般系统的按钮用这一套插件就足够了。

9.4 DateTime Picker

之前一直认为jQuery的datepicker插件是日期选择插件里最强大的，见过Bootstrap的这个DateTime Picker插件（官方网站为http://www.malot.fr/bootstrap-datetimepicker/）以后才发现，它和jQuery的插件一样强大，而且还能保持和Bootstrap一样的风格，（目前是2.x版本风格，以后就要支持新版3.x默认风格了）。

关于该插件的细节，我们不做过多的介绍，这里先列一下几个功能截图，如图9-15所示。

我们这里简单罗列一下该插件的使用方法，具体的细节请查看官方详细文档。

图9-15　时间和日期选择效果

1）绑定输入框，并设置format选项。示例如下：

```
<input type=“text” value=“2012-05-15 21:05” id=“datetimepicker”>
// JS代码
$(‘#datetimepicker’).datetimepicker({
    format: ‘yyyy-mm-dd hh:ii’
});
```

2）绑定输入框，并设置format标记。示例如下：

```
<input type=“text” value=“2012-05-15 21:05” id=“datetimepicker”
    data-date-format=“yyyy-mm-dd hh:ii”>
// JS代码
$(‘#datetimepicker’).datetimepicker();
```

3）作为声明式组件使用时，示例如下：

```
<div  class=“input-append  date”  id=“datetimepicker”  data-date=“12-02-
2012”
     data-date-format=“dd-mm-yyyy”>
    <input size=“16” type=“text” value=“12-02-2012” readonly>
    <span class=“add-on”><i class=“icon-th”></i></span>
</div>
// JS代码
$(‘#datetimepicker’).datetimepicker();
```

4）作为声明式组件使用时，示例如下：

```
<div id=“datetimepicker”></div>
// JS代码
$(‘#datetimepicker’).datetimepicker();
```

它几乎和jQuery的datepicker插件一模一样，非常好用。另外，该插件还支持国际化，但是需要按照固定的格式提供国际化的翻译内容才行，具体步骤请阅读官方文档。

9.5 Cikonss

Cikonss是纯CSS实现的响应式Icon插件，兼容IE8+（官方网站是http://www.bootcss.com/p/cikonss/），直接引用一个CSS文件即可使用。该插件提供了43个icon，5种尺寸，3种样式，总计可以生成645个icon组合。

每个icon都由两个元素构成：一个父元录，一个子元素，其中父元素是容器元素。父元素定义如下：

❑普通样式：.icon（必选）；

❑尺寸样式：.icon-small、.icon-mid、.icon-large、.icon-extra-large、.icon-huge（必选）；

❑变形样式：.icon-square、.icon-rounded（可选）。

父元素的使用方式如下：

```
<span class=“icon icon-small icon-square”>…</span>
```

但是，子元素才是真正的icon实现。每个icon的组织形式必须按如下格式使用：

```
<span class=“icon icon-mid”><span class=“icon-mail”></span></span>
```

当然，也可以在父元素上添加一个可选样式icon-square。示例如下：

```
<span class=“icon icon-small icon-square”><span class=“icon-mail”>
</span></span>
```

该插件的所有icon图标如图9-16所示。

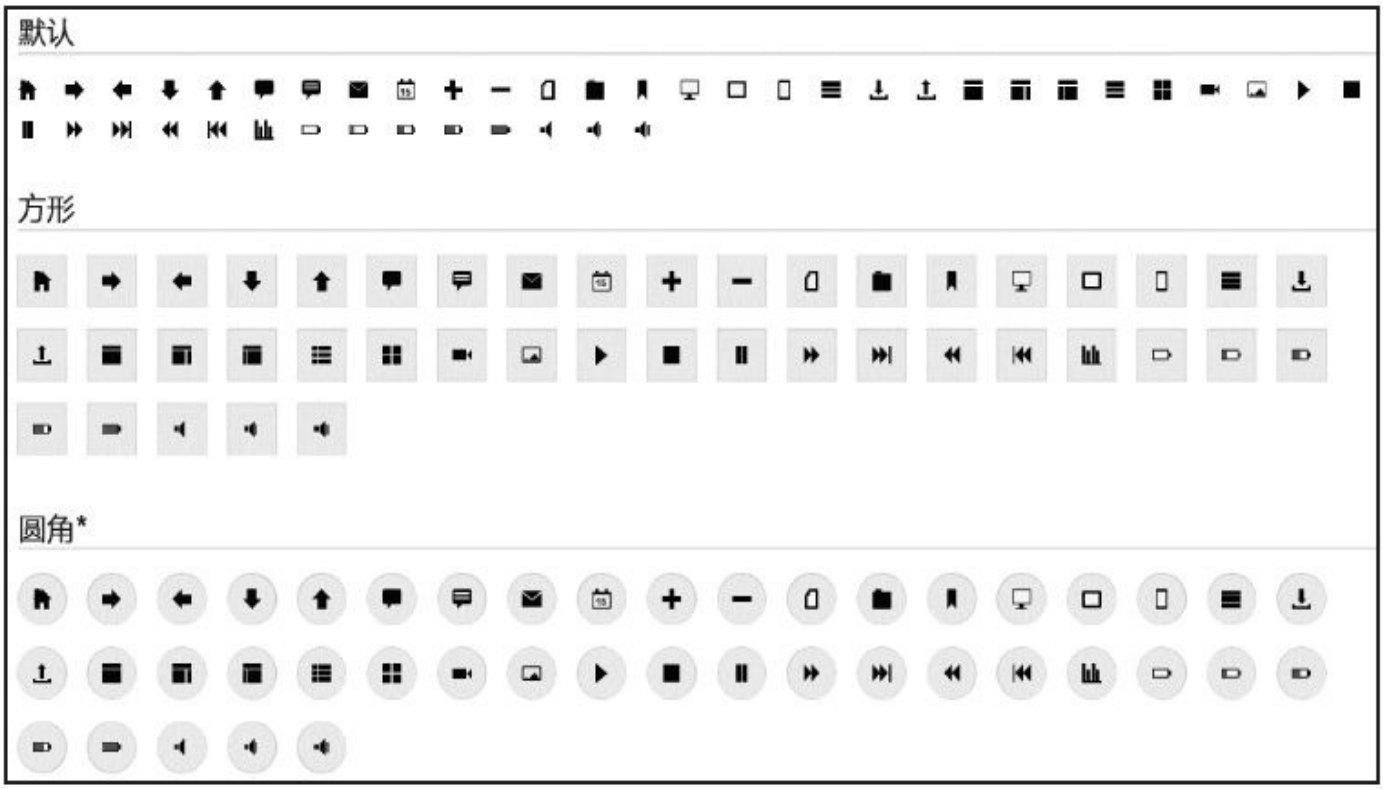


图9-16　Cikonss完整图标列表

9.6 Flat UI

Flat UI基于Bootstrap进行了扁平化风格改造，由Designmodo提供。Flat UI包含了很多Bootstrap提供的组件，但是外观比Bootstrap默认的更加漂亮。官方网站是http://designmodo.com/flat-free/。但其Pro已经开始收费了，不过免费版也很不错，大家可以借鉴一下，部分样式截图如图9-17所示。

Buttons

Default Button | Primary Button | Info Button | Danger Button

Success Button | Warning Button | Inverse Button | Disabled Button

Menu

Menu Item | Messages 1 | About Us

X-Men

Input

Inactive | Error | Success | Disabled

Progress bars & Sliders

Navigation

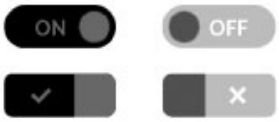
Checkboxes

- Unchecked
- Checked
- Disabled unchecked
- Disabled checked

Radio Buttons

- Radio is off
- Radio is on
- Disabled radio is off
- Disabled radio is on

Toggles



Tags

Clean | Fresh | Modern | Unique | +

Share

Facebook OFF

Twitter ON

Pinterest OFF

Share

Tooltips

Tooltip under the text.

Here is the sample of talltooltip that contains three lines or more. More.

Text Box

Lato is free web-font designed by **Lukasz Dziedzic** from Warsaw.

Here you can feel the color, size, line height and margins between paragraphs. Don't forget to underline your links, they are an important visual marker for users.

Also, to attract attention you can mark some important words using **bold weights**.

Headings

Header

Header 2

Header 3

Header 4

HEADER 5

HEADER 6

Color Swatches

#1ABC9C TURQUOISE	#2ECC71 EMERLAND	#3498DB PETER RIVER	#9B59B6 AMETHYST	#34495E WET ASPHALT
#16A085 GREEN SEA	#27AE60 NEPHRITIS	#2980B9 BELIZE HOLE	#8E44AD WISTERIA	#2C3E50 MIDNIGHT BLUE

SWATCHES

Colors — is almost the most important part of the **Flat UI**. Better to use different shades of provided colors than new.

For your convenience we

图9-17　FlatUI显示效果

下载网址是https:// github.com/designmodo/Flat-UI/。

9.7 更多插件

还有一些其他有名的插件也非常常用，这里我们就不做过多介绍了，只是列在表9-3里供大家参考。

表 9-3 常用插件列表

插件名称及下载网址	功能介绍
Bootstrap Switch *http://www.bootstrap-switch.org/*	用于模拟 iPhone 开关效果
Responsive Nav *http://responsive-nav.com/*	响应式导航的设计
jQuery.Pin *https://github.com/webpop/jquery.pin*	任意页面元素“钉”在某个容器顶部，而且在尺寸小的屏幕上能够自动禁用这种效果
jQuery UI Bootstrap *http://addyosmani.github.io/jquery-ui-bootstrap/*	将 Bootstrap 和 jQuery UI 完美地融合在一起
Metro UI CSS *http://metroui.org.ua/*	是一套用来创建类似于 Windows 8 Metro UI 风格网站的样式。现在，Metro UI CSS 项目在 Bootstrap 的基础上被开发成一个独立的解决方案
Bootstrap Form Builder *https://github.com/minikomi/Bootstrap-Form-Builder*	Bootstrap 在线表单构造器能够以鼠标拖拽的方式迅速生成一个基于 Bootstrap 的完整表单，减轻了手写 HTML 代码的劳动，而且不会出错
Messager *http://github.hubspot.com/messenger/*	是一个非常酷的弹框（Alert）组件，能够非常好地与 Bootstrap 融合，当然，单独使用效果也非常棒。Messenger 自带 4 套皮肤

更多插件，请参考http://www.bootcss.com/网站上推荐的列表。

附录A　浏览器兼容性支持

Bootstrap的目标是在最新的桌面浏览器和移动浏览器上有最好的表现，也就是说，在一些比较老旧的浏览器上，某些组件渲染的样式可能会有些不同，但是其功能是完整的。

如表A-1所示的浏览器在不同操作系统下的最新版均支持Bootstrap。

表 A-1　不同浏览器在不同操作系统下对 Bootstrap 的支持情况

浏览器 / 操作系统	Chrome	Firefox	IE	Opera	Safari
Android	支持	不支持	N/A	不支持	N/A
iOS	支持	N/A ⊖		不支持	支持
Mac OS X	支持	支持		支持	支持
Windows	支持	支持	支持	支持	不支持

表示该操作系统上没有这种浏览器。

1.IE8和IE9

Bootstrap默认是支持IE8和IE9的，但是由于很多CSS3属性和HTML5元素（例如，圆角矩形和投影）都不被支持，所以一些特性无法发挥。另外，IE8还需要使用Respond.j文件，两者配合起来才能实现对媒体查询（media query）的支持。如表A-2所示是一些不被支持的功能。

表 A-2　CSS3 中的 IE8/IE9 支持对照表

插件名称	IE8	IE9
border-radius	不支持	支持
box-shadow	不支持	支持
Transform	不支持	支持，需要带 -ms 前缀
Transition	不支持	不支持
Placeholder	不支持	不支持

注：关于 CSS3 和 HTML5 特性在各个浏览器上的详细支持情况，请访问网站 http://caniuse.com/。

2.IE8和Respond.js

如果在IE8下使用Respond.js进行项目开发工作的话，需要注意以下内容：

❑如果和Respond.js一起使用的CSS文件分布在不同的子域名（或者CDN网络）的话，则需要特殊的设置才行，详情访问以下网站：https://github.com/scottjehl/Respond/blob/master/README.md#cdnx-domain-setup

❑由于浏览器安全的原因，Respond.js不支持file:// 协议的引用。

❑Respond.js也不支持@import引入的CSS文件。

3.IE8和box-sizing

由于IE8不能完全支持box-sizing: border-box;与min-width、max-width、min-height或max-height一同使用，所以Bootstrap开发团队从v3.0.1版本开始，在.container上不再使用max-width了。

4.IE兼容性模式

由于Bootstrap不支持IE的兼容性模式，所以为了让IE浏览器运行最新的渲染模式，建议将如下标签加入项目的所有页面中，确保每个页面在IE浏览器中都能保持最好的渲染效果。

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

5.Windows 8中的IE10和Windows Phone8

IE10并没有将屏幕宽度和视口（viewport）宽度区别出来，这就导致Bootstrap中的媒体查询并不能很好地发挥作用。要解决这个问题，可以引入下面列出的这段CSS暂时修复该问题。

```
@-ms-viewport       { width: device-width; }
```

然而，上述代码会导致Windows Phone 8设备按照桌面浏览器的方式渲染页面，而不是较窄的"手机"渲染方式。为了解决这个问题，还需要加入以下CSS和JavaScript代码，直到微软修复该问题为止。

CSS代码如下：

```
@-webkit-viewport   { width: device-width; }
@-moz-viewport      { width: device-width; }
@-ms-viewport       { width: device-width; }
@-o-viewport        { width: device-width; }
@viewport           { width: device-width; }
```

JavaScript代码如下：

```
if (navigator.userAgent.match(/IEMobile\/10.0/)) {
    var msViewportStyle = document.createElement("style")
    msViewportStyle.appendChild(
      document.createTextNode(
        "@-ms-viewport{width:auto!important}"
      )
```

```
    )
                                        document.getElementsByTagName("head")
[0].appendChild(msViewportStyle)
}
```

6.Safari对百分比数字凑整的问题

从OS X版Safari v6.1和iOS v7.0.1版Safari开始，其浏览器的绘制引擎在处理.col-*-1所对应的百分比小数时存在bug。也就是说，如果在一行（row）之中定义了12个单独的列（.col-*-1），就会看到这一行比其他行要短一些。官方团队目前还未解决这个问题，但可以通过如下方式进行避免：

❑为最后一列添加.pull-right，将其强制向右对齐。

❑手动调整百分比数字，让其针对Safari表现更好（这比第一种方式更困难）。

官方团队将继续跟踪这个问题，如果有更方便的解决方案，就会更新官方代码。

7.模态框（Modal）、导航（Navbar）和虚拟键盘

（1）溢出和滚动

在iOS和Android上，<body>元素对overflow: hidden的支持很有限。其结果就是，在这两种设备上的浏览器中，当滚动屏幕超过模态框的顶部或底部时，<body>中的内容将开始随着滚动。

（2）虚拟键盘

如果正在模态框上面输入内容（iOS上的绘制bug），当触发虚拟键盘之后，模态框不会更新fixed元素的位置。这有几种解决方案，如将fixed元素转变为position: absolute或启动一个定时器手工修正其位置。官方没有将这些方案加入Bootstrap中，因此，需要自己选择最好的解决方案应用到项目中。

（3）导航下拉菜单

在iOS平台上没有应用.dropdown-backdrop样式，是因为其平台上的z-indexing太复杂了。因此，如果要在导航栏上关闭下拉菜单，需要单击下拉菜单元素来关闭（或者单击可以在iOS平台触发事件的其他元素也行）。

8.浏览器缩放

不可避免地，页面的缩放功能可能会将某些组件搞得很凌乱，不仅仅是Bootstrap，整个互联网上的所有网站都是这样。官方更倾向于忽略这些问题，因为这些问题除了一些hack手段，一般没有直接的解决方

案。

9.打印机视窗

即便是最新版本的浏览器，在打印方面也可能出现问题。特别是Chrome　v32，在打印页面使用媒体查询语法时，不管怎么设置margin，其视窗的宽度总是小于物理纸张大小。其可能导致超小型屏幕上的栅格系统在打印时被意外激活。我们的建议解决方法如下：

❑拥抱超小型屏幕上的栅格系统，确保页面可接受。

❑定义Less变量@screen - *，确保打印纸比超小型网格大。

❑为打印媒体（print　media）添加自定义媒体查询，强制改变网格的大小。

10.Android stock浏览器

Android从4.1开始，默认情况下是使用stock浏览器，可惜该浏览器对HTML和CSS的支持不是很完美，比如，该浏览器下的select元素不支持border-radius和border语法样式。所以在使用的时候，需要对此进行特殊处理。如下代码，判断如果不是Chrome、afari、Mozilla浏览器，就删除样式设置。

```
<script>
var nua = navigator.userAgent;
var isAndroid = (nua.indexOf('Mozilla/5.0') > -1 && nua.indexOf('Androic
     -1 && nua.indexOf('AppleWebKit') > -1 && nua.indexOf('Chrome') ===
if (isAndroid) {
                                $('select.form-control').removeClass('form-
control').css('width', '100%');
    }
</script>
```

附录B　第三方插件开发建议

虽然官方并不支持任何第三方插件，但还是提供一些建议，以避免在项目中可能会出现的问题。

某些第三方软件，例如Google地图和Google定制搜索引擎，都会由于* { box-sizing: border-box; }的设置而产生一些冲突，该设置将使得页面元素最终宽度的计算不受padding的影响。更多信息请参考盒子模型与尺寸计算（http://css-tricks.com/box-sizing/）。

为了避免Bootstrap设置的全局盒子模型所带来的影响，在使用的过程中需要根据不同情况，覆盖（方案1）或重置整个区域（方案2）。示例代码如下：

方案1：

```
/* 方案1: 通过CSS覆盖单个页面元素的盒模型 */
.element {
  -webkit-box-sizing: content-box;
     -moz-box-sizing: content-box;
          box-sizing: content-box;
}
```

方案2：

```
/* 方案2: 通过CSS重置整个区域 */
.reset-box-sizing,
.reset-box-sizing *,
.reset-box-sizing *:before,
.reset-box-sizing *:after {
  -webkit-box-sizing: content-box;
     -moz-box-sizing: content-box;
          box-sizing: content-box;
}
```

附录C　从2.x迁移到3.x

最新的3.x版，并不兼容老版的2.x，所以如果你现有的项目已经完工（或快完工）的话，建议不用浪费时间迁移了；如果你的项目刚开始（或即将开始），真的需要使用最新版的话，请参考两个版本之间的详细变更情况以及具体增删的内容。

为了移动先行和灵活性，作者对Bootstrap 3.x的基本样式、关键风格以及行为都做了大量的修改，这些修改并没有直接的表现，所以这里我们列出了一些关键的内容。

❑默认情况下，对文本型的表单控件仅设置了最少的样式。在这些元素上应用.form-control样式可以实现高亮颜色（在获得焦点时）和圆角样式。

❑应用了.form-control样式的文本表单控件默认宽度为100%。可以使用 包含（input）达到控制宽度的目的。

❑徽章样式（.badge）不再保留彩色样式（如-success、-primary等）。

❑.btn样式必须加用.btn-default才能产生默认效果。

❑.container和.row目前是基于百分比定义的宽度。

❑默认情况下，图片不具有响应式特性，需要使用.img-responsive才能让<img>实现响应式。

❑图标样式变成了.glyphicon，是新型的字体图标。每个图标都需要一个基本样式和一个代表特定图标的样式（例如，.glyphicon .glyphicon-asterisk）。

❑Typeahead组件被移除，建议使用Twitter Typeahead组件（http://twitter.github.io/typeahead.js/）。

❑模态框组件的HTML结构发生了很大的改变。.modal-header、.modal-body和.modal-footer部分目前包含在了.modal-content和.modal-dialog中，为的是增强移动设备上的样式和行为特性。

❑通过remote属性加载远程HTML内容的时候，是直接附加到.modal样式元素内，而非.modal-body样式元素。这么做的目的，是为了让用户可以自由控制.modal-header和.modal-footer，而不仅仅是.modal-body。

❑JavaScript事件目前全部都应用了命名空间。例如，模态弹窗的show事件的名称为show.bs.modal，选项卡组件的shown事件名称为shown.bs.tab，还有很多其他事件名称也是类似的形式。

1.变更样式

表 C-1　变更样式详细列表

Bootstrap 2.x 样式（旧）	Bootstrap 3.x 样式（新）	Bootstrap 2.x 样式（旧）	Bootstrap 3.x 样式（新）
.container-fluid	.container	.hidden-desktop	.hidden-md
.row-fluid	.row	.input-small	.input-sm
.span*	.col-md-*	.input-large	.input-lg
.offset*	.col-md-offset-*	.control-group	.form-group
.brand	.navbar-brand	.checkbox.inline .radio.inline	.checkbox-inline .radio-inline
.nav-collapse	.navbar-collapse	.input-prepend .input-append	.input-group
.nav-toggle	.navbar-toggle	.add-on	.input-group-addon
.btn-navbar	.navbar-btn	.img-polaroid	.img-thumbnail
.hero-unit	.jumbotron	ul.unstyled	.list-unstyled
.icon-*	.glyphicon .glyphicon-*	ul.inline	.list-inline
.btn	.btn .btn-default	.muted	.text-muted
.btn-mini	.btn-xs	.text-error	.text-danger
.btn-small	.btn-sm	.table .error	.table .danger
.btn-large	.btn-lg	.bar	.progress-bar
.alert-error	.alert-danger	.bar-*	.progress-bar-*
.visible-phone	.visible-xs	.accordion	.panel-group
.visible-tablet	.visible-sm	.accordion-group	.panel .panel-default
.visible-desktop	.visible-md	.accordion-heading	.panel-heading
.hidden-phone	.hidden-xs	.accordion-body	.panel-collapse
.hidden-tablet	.hidden-sm	.accordion-inner	.panel-body

2.新增样式

表 C-2　新增样式详细列表

元　素	描　述
面板	.panel .panel-default .panel-body .panel-title .panel-heading .panel-footer .panel-collapse
列表组	.list-group .list-group-item .list-group-item-text .list-group-item-heading
Glyphicon 图标	.glyphicon
大屏幕展播	.jumbotron
超小栅格（<768px）	.col-xs-*
小型栅格（≥ 768px）	.col-sm-*
中型栅格（≥ 992px）	.col-md-*

（续）

元　素	描　述
大型栅格（≥ 1200px）	.col-lg-*
响应式设计（≥ 1200px）	.visible-lg .hidden-lg
Offset	.col-sm-offset-* .col-md-offset-* .col-lg-offset-*
Push	.col-sm-push-* .col-md-push-* .col-lg-push-*
Pull	.col-sm-pull-* .col-md-pull-* .col-lg-pull-*
输入框组	.input-group .input-group-addon .input-group-btn
表单控件	.form-control .form-group
按钮组尺寸	.btn-group-xs .btn-group-sm .btn-group-lg
导航条文本	.navbar-text
导航条 header	.navbar-header
自适应导航	.nav-justified
响应式图像	.img-responsive
彩色的表格行	.success .danger .warning .active
彩色的面板	.panel-success .panel-danger .panel-warning .panel-info
模态弹窗	.modal-dialog .modal-content
缩略图	.img-thumbnail
well 尺寸	.well-sm .well-lg
警告框链接	.alert-link

3.移除样式

表 C-3　移除样式详细列表

元　　素	移除的样式	Bootstrap 3.x 中的等效样式
表单行为	.form-actions	无
搜索表单	.form-search	无
Fluid 容器	.container-fluid	.container 没有固定栅格（都是百分比）
Fluid 行	.row-fluid	.row 没有固定栅格（都是百分比）
控件包装器	.controls	无
控件行	.controls-row	.row 或 .form-group
导航条的内部容器	.navbar-inner	无
导航条垂直分隔符	.navbar .divider-vertical	无
下拉菜单的子菜单	.dropdown-submenu	无
选项卡对齐	.tabs-left .tabs-right .tabs-below	无
导航列表	.nav-list .nav-header	没有直接等效样式，但列表组、面板组与此类似

附录D　Glyphicons图标全集

.glyphicon-adjust
.glyphicon-align-center
.glyphicon-align-justify
.glyphicon-align-left
.glyphicon-align-right
.glyphicon-arrow-down
.glyphicon-arrow-left
.glyphicon-arrow-right
.glyphicon-arrow-up
.glyphicon-asterisk
.glyphicon-backward
.glyphicon-ban-circle
.glyphicon-barcode
.glyphicon-bell
.glyphicon-bold
.glyphicon-book
.glyphicon-bookmark
.glyphicon-briefcase
.glyphicon-bullhorn
.glyphicon-calendar
.glyphicon-camera
.glyphicon-certificate
.glyphicon-check
.glyphicon-chevron-down
.glyphicon-chevron-left
.glyphicon-chevron-right
.glyphicon-chevron-up
.glyphicon-circle-arrow-down
.glyphicon-circle-arrow-left
.glyphicon-circle-arrow-right
.glyphicon-circle-arrow-up
.glyphicon-cloud
.glyphicon-cloud-download
.glyphicon-cloud-upload
.glyphicon-cog
.glyphicon-collapse-down
.glyphicon-collapse-up
.glyphicon-comment
.glyphicon-compressed
.glyphicon-copyright-mark
.glyphicon-credit-card
.glyphicon-cutlery
.glyphicon-dashboard
.glyphicon-download
.glyphicon-download-alt
.glyphicon-earphone
.glyphicon-edit
.glyphicon-eject
.glyphicon-envelope
.glyphicon-euro
.glyphicon-exclamation-sign
.glyphicon-expand
.glyphicon-export
.glyphicon-eye-close
.glyphicon-eye-open
.glyphicon-facetime-video
.glyphicon-fast-backward
.glyphicon-fast-forward
.glyphicon-file
.glyphicon-film
.glyphicon-filter
.glyphicon-fire
.glyphicon-flag
.glyphicon-flash
.glyphicon-floppy-disk
.glyphicon-floppy-open
.glyphicon-floppy-remove
.glyphicon-floppy-save
.glyphicon-floppy-saved
.glyphicon-folder-close
.glyphicon-folder-open
.glyphicon-font
.glyphicon-forward
.glyphicon-fullscreen
.glyphicon-gbp
.glyphicon-gift
.glyphicon-glass
.glyphicon-globe
.glyphicon-hand-down
.glyphicon-hand-left
.glyphicon-hand-right
.glyphicon-hand-up
.glyphicon-hd-video
.glyphicon-hdd
.glyphicon-header
.glyphicon-headphones
.glyphicon-heart
.glyphicon-heart-empty
.glyphicon-home
.glyphicon-import
.glyphicon-inbox
.glyphicon-indent-left
.glyphicon-indent-right
.glyphicon-info-sign
.glyphicon-italic
.glyphicon-leaf
.glyphicon-link
.glyphicon-list
.glyphicon-list-alt
.glyphicon-lock
.glyphicon-log-in
.glyphicon-log-out
.glyphicon-magnet
.glyphicon-map-marker
.glyphicon-minus
.glyphicon-minus-sign
.glyphicon-move
.glyphicon-music
.glyphicon-new-window
.glyphicon-off
.glyphicon-ok
.glyphicon-ok-circle
.glyphicon-ok-sign
.glyphicon-open
.glyphicon-paperclip
.glyphicon-pause
.glyphicon-pencil
.glyphicon-phone
.glyphicon-phone-alt
.glyphicon-picture
.glyphicon-plane
.glyphicon-play
.glyphicon-play-circle
.glyphicon-plus
.glyphicon-plus-sign
.glyphicon-print
.glyphicon-pushpin
.glyphicon-qrcode
.glyphicon-question-sign
.glyphicon-random
.glyphicon-record
.glyphicon-refresh
.glyphicon-registration-mark
.glyphicon-remove
.glyphicon-remove-circle
.glyphicon-remove-sign
.glyphicon-repeat
.glyphicon-resize-full
.glyphicon-resize-horizontal
.glyphicon-resize-small
.glyphicon-resize-vertical
.glyphicon-retweet
.glyphicon-road
.glyphicon-save
.glyphicon-saved
.glyphicon-screenshot
.glyphicon-sd-video
.glyphicon-search
.glyphicon-send
.glyphicon-share
.glyphicon-share-alt
.glyphicon-shopping-cart
.glyphicon-signal
.glyphicon-sort
.glyphicon-sort-by-alphabet
.glyphicon-sort-by-alphabet-alt
.glyphicon-sort-by-attributes
.glyphicon-sort-by-attributes-alt
.glyphicon-sort-by-order
.glyphicon-sort-by-order-alt
.glyphicon-sound-5-1
.glyphicon-sound-6-1
.glyphicon-sound-7-1
.glyphicon-sound-dolby
.glyphicon-sound-stereo
.glyphicon-star
.glyphicon-star-empty
.glyphicon-stats
.glyphicon-step-backward
.glyphicon-step-forward
.glyphicon-stop
.glyphicon-subtitles
.glyphicon-tag
.glyphicon-tags
.glyphicon-tasks
.glyphicon-text-height
.glyphicon-text-width
.glyphicon-th
.glyphicon-th-large
.glyphicon-th-list
.glyphicon-thumbs-down
.glyphicon-thumbs-up
.glyphicon-time
.glyphicon-tint
.glyphicon-tower
.glyphicon-transfer
.glyphicon-trash
.glyphicon-tree-conifer
.glyphicon-tree-deciduous
.glyphicon-unchecked
.glyphicon-upload
.glyphicon-usd
.glyphicon-user
.glyphicon-volume-down
.glyphicon-volume-off
.glyphicon-volume-up
.glyphicon-warning-sign
.glyphicon-wrench
.glyphicon-zoom-in
.glyphicon-zoom-out