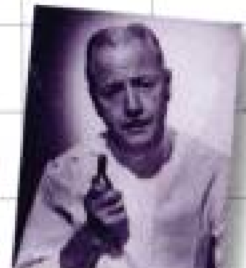
深入浅出JavaScript（中文版）

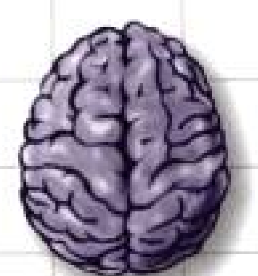
Head First JavaScript

Improve your user experience with web page interactivity

Whip up working JavaScript code in a snap



Stop fearing event handling—it won't hurt a bit



Load important JavaScript concepts directly into your brain

Slice and dice HTML with help from the DOM

Bend your mind around dozens of puzzles and exercises

Michael Morrison 著

O'Reilly Taiwan公司 编译

O'REILLY® 東南大學出版社

深入浅出JavaScript

如果有什么方式可以学会JavaScript，又不会中途就想放火烧了发明这个东西的人，也不会事后一辈子对万维网咒骂不休，该有多好？噢，或许只是我的白日梦吧…………

Michael Morrison 著

O'Reilly Taiwan公司 编译

O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

深入浅出 JavaScript/（美）莫里森（Morrison, M.）著；O'Reilly Taiwan 公司编译：－南京：东南大学出版社，2010.9

书名原文：Head First JavaScript
ISBN 978-7-5641-2416-8

I. ①深… II. ①莫… ② O… III. ① Java 语言－程序设计 IV. ① TP312

中国版本图书馆 CIP 数据核字（2010）第 169602 号

江苏省版权局著作权合同登记
图字：10-2010-273 号

©2007 by O'Reilly Media, Inc.

Simplified Chinese Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2010. Authorized translation of the English edition, 2010 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly Media, Inc. 出版 2007。

简体中文版由东南大学出版社出版 2010。英文原版的翻译得到 O'Reilly Media, Inc. 的授权。此简体中文版的出版和销售得到出版权和销售权的所有者 —— O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

深入浅出 JavaScript

出版发行：东南大学出版社
地　　址：南京四牌楼 2 号
出 版 人：江 汉
网　　址：http://www.seupress.com
电子邮件：press@seu.edu.cn
印　　刷：扬中市印刷有限公司
开　　本：787 毫米 × 980 毫米　12 开本
印　　张：53 印张
字　　数：887 千字
版　　次：2010 年 9 月第 1 版
印　　次：2010 年 9 月第 1 次印刷
书　　号：ISBN 978-7-5641-2416-8
印　　数：1~3000 册
定　　价：98.00 元（册）

本社图书若有印装质量问题，请直接与读者服务部联系。电话（传真）：025-83792328

献给 Netscape 的众位先行，早在上个世纪，他们对万维网的梦想就已不只是“一本虽然有许多链接但缺乏任何行动的大型在线书籍”。

当然，他们的梦想也制造出可怕的`<blink>`标签……勇于圆梦，但别冲过头了！

《深入浅出JavaScript》的作者

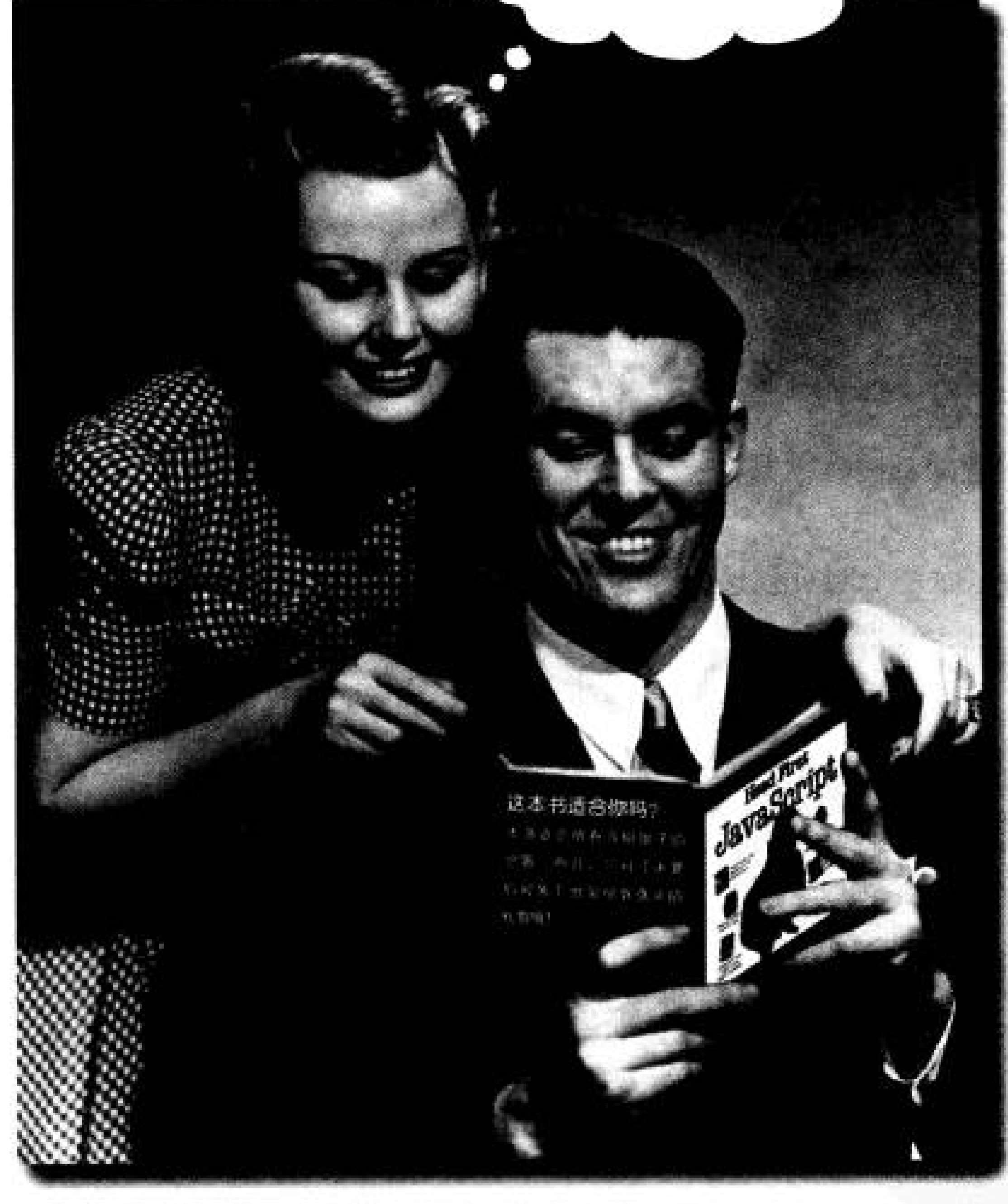
Michael Morrison与电脑世界结下不解之缘的开端，就是他的第一台个人电脑TI-99/4A，配件包括至高无上的人体工程学键盘、黑白TV“显示器”，还有那甜美的卡式带存储系统。往后的岁月里，他又曾拥有过其他电脑“玩具”，不过仍然时时怀念着当年在TI上把玩Parsec、在后院玩Nerf足球的日子。

现在的Michael已经长大成人，兴趣也变得比较成熟一点，例如建立交互网络应用程序……还有滑板。割伤、擦伤，大伤小伤不断，他面对技术挑战的方式其实与挑战极限运动时一样有勇无谋。在开发过几款电视游戏、发明过几种玩具、写了快50本书、建立许多在线课程后，Michael终于觉得可以面对《深入浅出JavaScript》的挑战……他以后再也不相信自己的感觉了。

事实上，面对“深入浅出”系列的写作时，不可能有准备好的那一天。选择墨菲斯手中的红色药丸，进入Head First黑客帝国（Matrix）的世界，已经是最佳准备途径了。当Michael从另一个世界中醒来后，身上也还残留着智力较量下的淤青，他看待学习（与教学）的眼光完全不一样了。这是多么令人兴奋的一件事啊！现在这个时候，他大概正与老婆一起在鲤鱼池边，反思着交互性万维网的神妙之处。

如何使用本书

序



本章回答了热门话题：
“为什么要把这些东西放到JavaScript相关书籍里？”

谁适合读这本书？

如果下列问题你都回答“是”：

1. 你使用的电脑安装了Web浏览器、文本编辑器，而且可以连接Internet吗？
2. 想要学习、了解、牢记如何创建活色生香的网页，把全球资讯网转变成真正的互动体验吗？
3. 喜欢香艳刺激的晚宴对话，胜过枯燥乏味的学术演讲？

我们将协助大家学习设计JavaScript代码，让网页多出各种只有HTML就办不到的炫目特技。

那么这本书正是为你而写。

谁或许应该远离这本书？

如果下列问题你都回答“是”：

1. 你是个网页创建世界的新手吗？（虽然不需是位 HTML 界的大师，但至少该对网页如何以 HTML 与 CSS组成、如何把网页放上网络有点基础的了解。）
2. 你已经是Script流派的一流高手，正在寻找JavaScript 的失传秘笈吗？
3. 你害怕尝试不同的事物吗？宁可接受牙根管治疗（抽神经），也不愿意混搭条纹与花格子布？你认为技术性书籍若为JavaScript代码赋予人性，则不够认真严肃吗？

本书非常适合作为《Head First HTML with CSS & XHTML》的延伸阅读，如果有人想好好地温习HTML技术，记得回顾一下。

那么这本书保证不符合你的期望。

［营销部补充：本书适合所有信用卡持卡者。］

我们知道你在想什么

“这怎么可能是一本正经的程序设计书籍？”

“这一堆图是干嘛的？”

“这样真的会让我学到东西吗？”

你的大脑认为“这”才重要。

我们也知道你的大脑在想什么

你的大脑渴望新奇的事物，它总是在搜索、扫描、期待着不寻常的事物。人类的大脑生来如此，正是这样的特质帮助我们常保活力，在竞争激烈的生命树上存活至今。

至于每天都要面对一成不变、平淡无奇的事物，你的大脑又作何反应？它会尽量阻止这些事去干扰大脑的真正工作——记录真正重要的事。大脑不会浪费脑细胞去储存无聊的事，它们绝对无法通过“显然不重要”的过滤器。

大脑究竟怎样知道什么是重要的事？假设你去郊游，突然有只老虎跳到你的眼前，你的大脑和身体会怎样反应？

神经元被触发、情绪激动、肾上腺素激增。

那就是大脑“知道”的方式……

这绝对重要，别忘了！

但是，想象你是在家里或图书馆，一个安全、温暖而且没有老虎出没的环境。你正在用功准备考试，或者研究某个技术——你的老板认为需要一周，顶多十天，就能完成的难题。

但是，有个问题。你的大脑正试图帮你忙，它试着确保这件“显然不重要”的事，不会占用有限的资源。毕竟，资源最好用来储存真正的大事，如老虎、火灾或者绝对不应该穿短裤玩滑雪板。

而且也没有简单的方法可以告诉你的大脑：“脑袋呀！拜托你啊……不管这本书多枯燥、多让我昏昏欲睡，还是请你把这些内容全都记下来。”

我们将“Head First”的读者视为<u>学习者</u>。

那么，要怎么“学习”呢？首先，必须“理解”，然后确定不会“忘记”学习内容。我们不会用填鸭的方式对待你。认知科学、神经生物学、教育心理学的最新研究指出：只利用书页上的文字，远不及“学习过程”的需要。我们知道如何帮你的脑袋“开机”、让大脑“兴奋”。

Head First 学习守则：

视觉化。图像远比纯文字更容易记忆，让学习更有效率（在知识的回想与转换上，提升度达到 89%）。图像也能让事情容易理解，将文字放进或靠近相关联的图像，而不是把文字放在图像下或后一页，可让学习者在解决相关内容的问题时，潜力最多提高两倍。

使用对话式与拟人化的风格。最新的研究发现，相较于一般正经八百的叙述方式，改以第一人称的角度、谈话式的风格直接与读者对话时，学员课后测验成绩的提升度可达 40%。以故事代替论述，以轻松的口语取代正式的演说，别太过严肃。你觉得晚宴伴侣的耳边细语和课堂上的演说，哪一种比较能够引起你的注意力？

onfocus!

onblur!

让学习者更深入地思考。换句话说，除非主动刺激神经，不然大脑就不会有所作为。读者必须受到刺激、亲身参与、感到好奇、接受启发，才会解决问题、得出结论并且形成新知识。为达此一目的，你需要挑战、需要习题、需要刺激思考的问题与活动，以同时运用左右脑与各种感官。

引起——并保持——读者的注意力。我们都有这样的经验：“我真的很想学会这个东西，但是还没翻完第一页，就已经昏昏欲睡了。”你的脑袋只会注意到特殊、有趣、怪异、引人注目且超乎预期的东西。新颖、困难、技术性的主题学起来未必枯燥乏味，如果不觉得无聊，你的大脑学习就会加快许多。

拨动心弦。现在，我们知道记忆的能力非常仰赖情感联系。你会记得自己在乎的事，当你心有所感时，就会“记住”。不不不……我不是在说灵犬莱西和小主人之间心有灵犀的故事，而是当你解开谜题、学会别人觉得困难的东西，或者发现自己比工程部的Bob懂得更多时，当下产生的惊讶、好奇、有趣、“哇……”、“我好棒！”这类的情绪与感觉。

元认知：想想如何思考（译注1）

如果真的想学习，想学得更快、更深入，那么，请注意自己如何“注意”，想想如何思考，学学如何学习。

大多数人在成长过程中并未修过元认知或学习理论的课程，师长们期望我们学习，却没有教导我们如何学习。

我们假设大家拿着这本书，是为了学习打造最热门、最发烧的交互式网站，但又不想花费太多学习时间。如果有意使用刚从书中得到的知识，你必须记住读过的东西，而在记住前，必须先理解阅读的内容。想要从本书（或任何书籍与学习经验）得到最多的利益，请为你的大脑负责，让你的大脑好好注意这些内容。

秘诀在于：让大脑认为你正在学习的新知识确实很重要，与你的生死存亡有关，就像跳到你面前的食人虎。否则，我们就会不断陷入与大脑的苦战，老是记不住新知识。

好吧！该如何让大脑将 JavaScript 视为一只饥饿的大老虎？

有慢又啰嗦的方法，也有快又有效的方法。慢的方法就是多读几次。各位想必听过“勤能补拙”，只要重复的次数够多，也能够学会并记住最乏味的知识，你的大脑会觉得：“虽然感觉起来不怎么重要，但他却一而再、再而三地苦读这个部分，所以应该是重要的吧！”

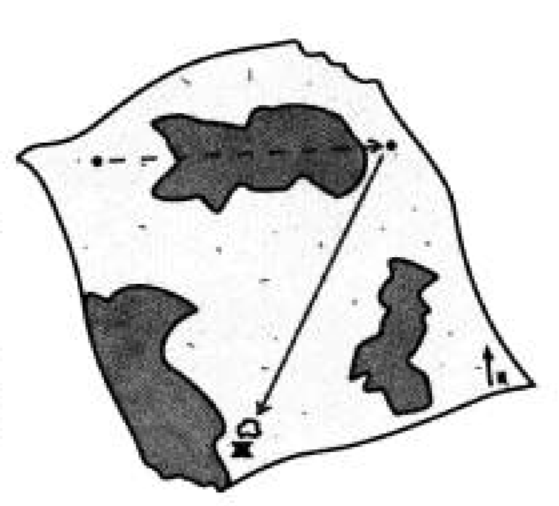
较快的方法则是想办法增进大脑活动，特别是不同类型的大脑活动。前页出现的素材就是解决方案，已证实有助于大脑运作。研究显示，文字若放在它所描述的图像内（而不是置于页面内其他地方，如图说或内文），有助于大脑建立图文的关联性，而且触发更多的神经元。越多的神经活动＝大脑越容易把这个内容视为值得注意的信息，也越可能记录下来。

对话式的风格也相当有帮助。意识到自己正在与人对话时，我们会付出更多的注意力，我们必须竖起耳朵，注意整个对话的进行，跟上双方的谈话内容。神奇的是，你的大脑根本不在乎那是人与书本间的“对话”！另一方面，如果写作风格既正式又枯燥，你的大脑会以为正在聆听一场演讲，自己只是一个被动的听众，根本不需要保持清醒。

然而，图像与对话式的风格只不过是一个开端。

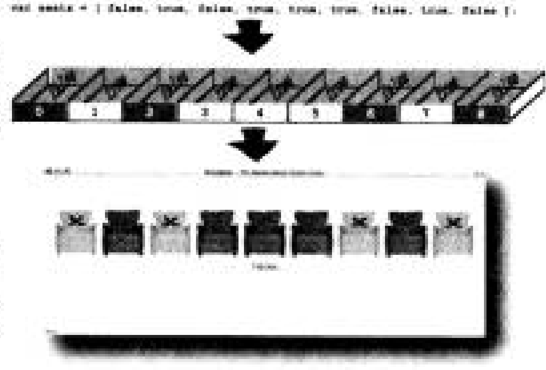
译注1：元认知（Metacognition），教育心理学专有名词。管理学习和认知的过程，增加学习的效果。

我们的做法：



我们使用图像，因为你的大脑对视觉刺激比较有感觉，而非文字。对大脑而言，一张图胜过千言万语。当文字和图像需要合作时，我们将文字嵌入图像里，因为文字若是位于相关图像中，而不是埋在图说或内文的某处，大脑将运作得比较有效率。

我们重复表现相同内容，以不同的表现方式、不同的媒介、多重的感知，一再叙述相同的事物。我们的策略目的就是：将内容烙印在大脑不同区域，进而增加记住的机会。

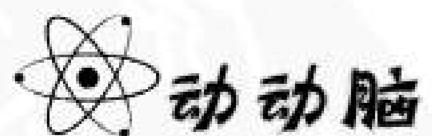


我们以超乎预期的方式使用概念和图像，因为大脑遇到新鲜有趣的事，波长才会与之同调。我们使用的图像与概念或多或少具有情绪内容，也是因为大脑的设计在于注意情绪带来的生物化学反应。让我们有情绪感觉的事物自然比较容易记住，即使那些感觉不过是“幽默”、“惊讶”、“有趣”、“笑话很难笑”等等。

我们使用拟人化、对话式的风格，因为当大脑相信你正处于对话中而不是被动地聆听演说时，便会付出更多注意力——即使你的交谈对象是一本书。也就是说，虽然实际上在“阅读”对话，大脑还是较为活跃。

我们包含了80个以上的练习活动，比起死读硬记，大脑从做中学的学习与记忆效果都更好。我们让习题维持在具有挑战性但又可以完成的程度，因为大多数人喜欢挑战后的成就感。

我们使用多种教学风格，有些人可能比较喜欢按部就班，有些人喜欢先了解整体大纲，有些人则喜欢直接看程序代码范例……然而，不管你是哪一类学习风格，都能够受益于本书以不同方式表现相同内容的手法。

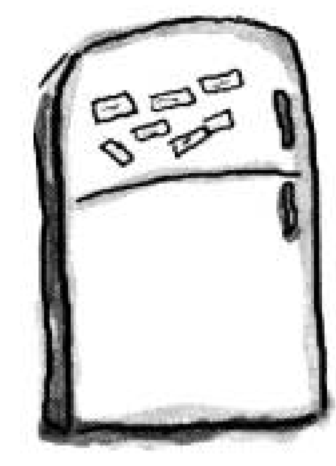


本书的设计同时考虑到左脑与右脑。越多脑细胞参与，越有可能学会并记住信息，而且保持更长时间的专注力。使用一边的大脑，往往意味着另一边的大脑有机会休息，遂得以延长专心学习的时间，也更有效率。

我们也会运用故事和练习，呈现多重观点，因为当大脑被迫进行评估与判断时，学习效果将更为深入。

书中也有相当多的挑战题，虽然提出问题，却不见得立刻提供答案。我们的用意是让大脑努力工作，才能学得更多、记得更牢。你想一想：只是看别人运动，有办法达到帮你塑身的效果吗？同时，我们尽力确保大脑的努力方向正确，以免让你花费大量脑力，却用于处理难以理解的范例，或者难以剖析、充满行话、咬文嚼字的论述。

我们还会使用人物。在故事、图像与范例中，处处都是人物。因为你也是人！你的大脑对人会比对事物更加注意。



驯服大脑的方法

好吧，该做的我们都做了，剩下的就靠你了。这里介绍一些技巧，但只是一个开端，你应该倾听你的大脑，看看哪些对你的大脑有效，哪些无效。试试看吧！

沿虚线剪下，用Hello Kitty磁铁贴在冰箱上。

---

① 慢慢来，理解越多，越不需要强记。

别只顾着翻页，记得停下来，好好思考。书中提出问题时，别完全不思考就直接看答案。想象是另外一个人面对面地向你提问，如果能够迫使大脑思考得更深入，就越有机会理解并记得更多的知识。

② 勤做练习，写下心得。

我们在书中安排了习题，如果你光看不做，就好像躺在床上看着电视中的美女姐姐卖力做着塑身运动……你这样是瘦不下来的。使用铅笔作答。大量证据显示，教学当中的实体活动可增加学习的效果。

③ 认真阅读“没有蠢问题”的单元。

详细阅读所有的“没有蠢问题”。这可不是无关紧要的说明，而是核心内容的一部分！千万别忽略！

④ 将阅读本书作为睡前最后一件事，至少是睡前最后一件具有挑战性的活动。

学习的一部分反应（特别是转化为长期记忆的过程）发生在放下书本之后。你的大脑需要进一步处理新知识的时间。如果在处理期间塞进其他新知识，某些刚学过的东西将会遗失。

⑤ 喝水，多喝水。

你的大脑需要浸泡在丰沛的液体（译注2）内，才能运作良好，脱水（往往发生在感觉口渴前）会减缓认知功能。

译注2：医学报告指出，大脑的重量有80%由水组成。

⑥ 念出声音，大声念出来。

“说话”将驱动大脑的不同部位。如果需要理解某项事物或试图增强记忆，请大声说出来，解释给别人听的效果更佳。你会学得更快，甚至触发许多新想法，光靠阅读无法有这种效果。

⑦ 倾听大脑的声音。

注意你的大脑是否过度负荷，如果你发现自己开始恍神，或者过目即忘，就是应该休息的时候。当你错过某些重点时，请放慢脚步，否则将失去更多。

⑧ 用心感受！

必须让大脑知道这一切都很重要。试着融入故事情境、为照片加上自己的说明，即使是抱怨笑话太不好笑，都比毫无感觉更好。任何情绪反应对学习效果都有帮助。

⑨ 动手吧！

学习JavaScript只有一种方式：试着多多设计JavaScript代码。这本书就是要你动手做。不要略过我们提出的JavaScript习题——“学习”经常发生在解决问题的时刻，就算书中解决的“火柴人大冒险”、“Mandango电影院划位程序”或“YouCube博客”这类情境平常好像不太常遇到……（干笑）。请坚持做完习题，再翻开下一页的内容。对了，如果你早就想做个交互性网站的话，不如一边看书，一边尝试刚学到的JavaScript技巧吧！

读我

这是一段学习经验，而不是一本参考用书。所有阻碍学习的东西，我们都会刻意排除。第一次阅读时，你必须从头开始，因为本书对读者的知识背景做了一些假设。

我们以“必知”的 JavaScript作为学习基础。

如果你想了解 JavaScript 的演进历程，请你另寻高明，我们不符合你的需求。本书的目标在于传授如何以 JavaScript 建立既酷炫又实用的元素，以扩增网页的互动性，让网页变成有应有答、大家想玩玩看的网络应用程序。我们放弃了正式礼节，只让各位看到必须知道的 JavaScript 概念——于实际层面、建立实际元素的 JavaScript 概念。我们非常实际地考虑过啰！

我们并未放入 JavaScript 语言里的所有细枝末节。

虽然可以巨细靡遗地涵盖所有 JavaScript语句、对象、事件或关键字，但各位应该不想用起重机把这本书从书桌上运到健身中心吧？本书是非常优秀的健脑读物，如果搭配铅笔，效果会更好。我们着重于各位必须知道、使用 JavaScript 时 95% 常用的知识。当你了解本书内容后，将能带着自信，寻找打造杀手级梦幻程序码的深奥方法。

因为 JavaScript 包含了大量内置函数库（可重复利用的程序代码），当你在处理标准JavaScript代码时，与处理你自己创建的自定义代码不同，理解它很重要。凡是看到“自定义”一词的出现，表示这段程序代码需由各位自行设计，而不是JavaScript内置的一部分。

我们鼓励大家多用几种浏览器搭配本书。

虽然市面上常见的浏览器都支持 JavaScript，但各家厂商处理 JavaScript代码的方式却有微妙的不同。所以，我们鼓励大家至少挑选两种最新的常用浏览器，以便检查脚本执行的效果。我们发现 Firefox 目前有助于追踪 JavaScript代码的错误，但总而言之，你的脚本必须在各种浏览器上都能产生一致的效果。讲到测试 JavaScript代码，别犹豫，请把亲朋好友、同事熟人、阿猫阿狗都拖下水一起测试吧！

不要略过任何活动。

习题与活动并非附加的装饰品，而是本书核心内容的一部分，有些协助记忆，有些促进理解，有些还可以帮助应用。所以，请不要略过这些习题。填字游戏是唯一非必要的部分，但是它们提供不同情境，帮助大脑用另一个部位回顾学过的关键字与术语，中文版虽然翻译了提示，但答案还是英文哦。如果你真的很不喜欢弄乱美美的书页，也可以不做 Page Bender 啦！但这样就不好玩了。

重复是刻意且必要的。

我们冀望“Head First”系列能让各位真正学到东西，希望你能够记住读过的内容。大部分参考用书的目标并不包括知识记忆的保存与触发，但本书的重点是**学习**，所以重要内容会一再出现以加深你的印象。

程序范例尽量精简。

我们的读者反映，不希望看到书中列出200行的程序码，其中却只有两行是主题的关键。本书尽量把程序代码缩短，让学习的过程清晰简单。请别以为所有的程序代码都很强健可靠，本书的程序代码仅作为辅助学习之用，功能不见得完整。

我们把所有范例的完整代码放上网站，方便大家复制与粘贴到文本编辑软件上。你也可以直接在线试用 JavaScript 的效果。请参考：

http://www.headfirstlabs.com/books/hfjs/

“动动脑”习题没有答案。

某些“动动脑”习题没有肯定的答案，另有某些习题所启发的学习经验，则在于自我判定正确应用的时机。在某些练习中，我们会提供暗示，指引正确的方向。需要转动的东西就是你的大脑……感受脑之威力吧！

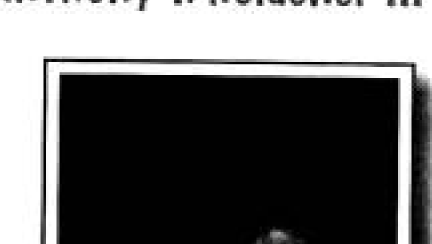
技术审阅团队

技术审阅：

Katherine St. John

Zachary Kessin

Anthony T. Holdener III



Alex Lee就读于休斯顿大学，主修管理信息系统。他喜欢跑步、电子游戏与三更半夜研究新的程序语言。

TW Scannell来自 Oregon 的 Sisters，从 1995 年就开始微调比特相关事物，目前是 Ruby on Rails 的开发者。

Elaine Nelson设计网站的经历已近十年。她曾对母亲大人说过，英国文学的文凭到哪里都很好用。她目前的作品与喜好，都能在*elainenelson.org*上看到。

Fletcher Moore是乔治亚科技学院的网络开发员与设计师。在休闲的时候，他热爱骑自行车、音乐、园艺，还是红袜队的球迷。他与妻子 Katherine、女儿 Sailor、儿子 Satchel 定居在亚特兰大城。

Anthony T. Holdener III是网络应用程序开发师，也是《Ajax: The Definitive Guide》的作者。

Zachary Kessin在全球资讯网刚开始萌芽时，就已开始在上面开发程序，大约有15年左右。他与妻子和三个小孩都住在以色列。

Katherine St.John是 City University of New York 的计算机科学与数学副教授，她的研究重心在于计算机生物学和随机结构。

Stephen Tallent在田纳西州的纳什维尔生活及工作，主要开发运动应用程序，并应付为人父母的混沌状态。当对前述任务稍有余力时，他喜欢玩滑雪板，而且正在准备第二项专长——快餐厨师。

致谢

给我的编辑：

还记得小学时候，学校会指派你和某个其他地方的小孩当笔友，然后你们就开始通过纸笔分享彼此的生活吗？项目开始后，**Catherine Nolan**自然而然地成为我的“深入浅出”笔友。不过，我们的交谈会通过电话、在线聊天客户端、电子邮件、传真机，以及任何能用OMG（Oh My God）、LOL（Laugh Out Loud）、BHH（Bless Her Heart，这句是我的最爱）的东西。在创作过程中，Catherine 渐渐不只是我的“JavaScript 认知学习的在线合作伙伴”，她成为我的朋友。关于 JavaScript 的“业务”会谈，可不会每次都逐渐扯到房屋整修……然后再扯回正题。在这次疯狂的著书过程中，很高兴能有这位经验老到的专家，陪我走过高潮与低潮。谢谢 Catherine！我还欠你好几个潜艇堡。

Catherine Nolan，Phish同好与杜威十进制图书分类法达人。

给O'Reilly小组：

Lou Barr，设计女神。

我对“深入浅出”小组的感谢有如滔滔江水绵延不绝，不过我会试着说得更具体一点。

Brett McLaughlin在Head First 训练营里，一开始就把我丢给疯狂的教学心理之狼，而且不让我退缩。这位先生对于学习过程的逆向工程学的认真程度，就像面对他的吉他一样。我想他在睡前一定至少自问一次：“我的动机是什么？”但因为他的不懈不怠，才造就这么好的一本书。谢谢 Brett！

Brett McLaughlin，“深入浅出”系列远征领队，而且很有戏剧天份。

Lou Barr是我在这个项目中的另一位虚拟笔友，也是关于美英差异的文化向导（她家在英国）。我觉得，设计之神真的只是暂时把她借给我们人类。如果没有她的魔法，本书的编排根本不可能实现。

Sanders Kleinfeld的运作比较静默，但在推动生产过程与及时提供“不可逃避”的大观念时，总是能感觉到他的存在。

在致谢的部分，当然不能遗忘其他O'Reilly团队成员。**Laurie Petrycki**准许这个项目通行，**Caitrin McCullough**负责管理一个杀手级的支持网站（http://www.headfirstlabs.com），还有准确地填满项目空隙的 **Keith McNamara**。谢谢大家！

最后，对**Kathy Sierra**与**Bert Bates**对于“深入浅出”系列的愿景，值得献上最大的谢意。我真的很高兴能成为其中的一分子……

目录（精要版）

	序	xxiii
1	交互式网络：感觉虚拟世界	1
2	存储数据：每项事物都有自己的位置	33
3	探索客户端：浏览器探索	85
4	决策：前有叉路，面对抉择	135
5	循环：自我重复的风险	189
6	函数：简化、重复利用、回收再利用	243
7	表单与验证：让用户全盘托出	289
8	驾驭网页：利用DOM分割HTML	343
9	为数据带来生命：科学对象怪人	393
10	创建自定义对象：自定义对象让你为所欲为	449
11	除错务尽：好脚本也会出错	485
12	动态数据：贴心易感的网络应用程序	537

目录（详实版）

序

你的大脑想专心于JavaScript。你正襟危坐地投入学习，但你的大脑一直催眠着你，它说这种学习一点也不重要。你的大脑说，大脑的空间应该储存更重要的事，如应该闪避的野生动物，或者绝对不应该穿短裤玩滑雪板。我们该如何欺骗大脑，让它觉得学习JavaScript与你的生命安危息息相关？

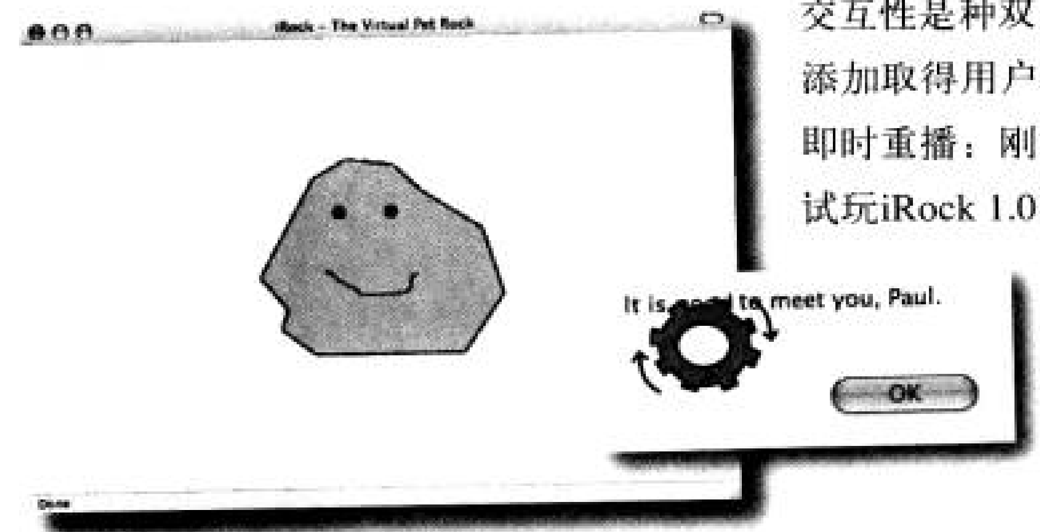
谁适合读这本书？	xxiv
我们知道你在想什么	xxv
元认知：想想如何思考	xxvii
驯服大脑的方法	xxix
读我	xxx
技术审阅团队	xxxii
致谢	xxxiii

1 交互式网络
感觉虚拟世界

不想只把万维网视为被动网页的同义词吗？

我们完全了解你的心情。被动的信息应该留给“书籍”这种形式。书籍适合阅读、学习，还有很多很多用途，但与**交互式**一点关系也没有。如果少了JavaScript，万维网就跟书没什么两样了。当然，没有JavaScript，还是可以提交表单，或许再利用HTML与CSS要些唬人的花招……不过，这只是在了无生趣的网页上操纵无生命的傀儡而已。真正让网页活起来的**交互性**，还需要**多一点智慧与多一点努力**……但保证带来**更多报酬**。

（在线）用户的需求 2
对墙弹琴……毫无反应 3
JavaScript在此为您服务 4
钢骨、烤漆、上路啰！ 6
利用 <script> 标签，向浏览器表示以下为JavaScript 11
你的浏览器可以处理HTML、CSS还有JavaScript 12
人类的虚拟好朋友……需要你的帮忙 15
让iRock动起来 16
创建iRock的网页 17
小试身手 17
JavaScript的事件：为iRock加上回答 18
以函数做alert 19
为iRock添加欢迎信息 20
让iRock真正动起来 22
交互性是种双向沟通 23
添加取得用户名字的函数 24
即时重播：刚才发生了什么事？ 27
试玩iRock 1.0 28

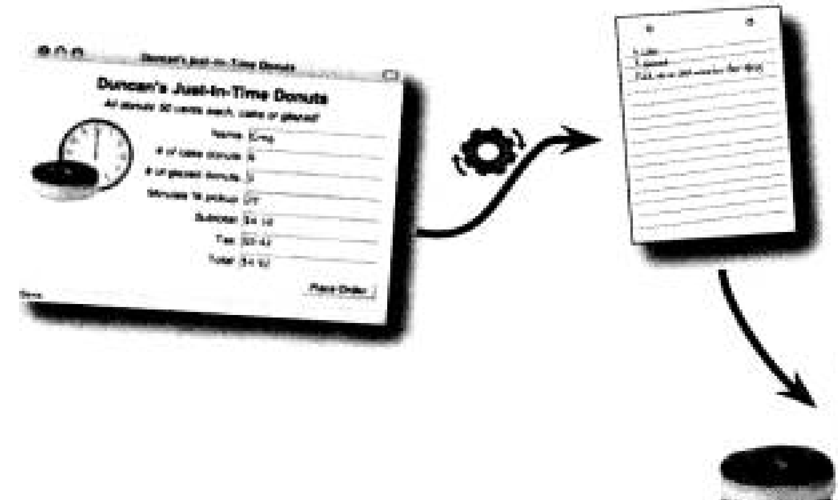


存储数据

2 每项事物都有自己的位置

现实生活中，我们经常把“有个地方储存物品”这件事看得太过重要。但在JavaScript中绝非如此。你不会有个足以藏人的更衣室，也不会有可以停放三辆保时捷的车库。在JavaScript的世界里，**每项事物都有自己的位置**，确认众多事物有所依归，则是你的责任。我们讲的事物就是数据——包括如何**呈现数据**、**存储数据**以及**找出数据**。身为JavaScript的存储专家，你将能面对满室杂乱无章的JavaScript数据，并用意志为它们贴上虚拟标签与储藏编号。

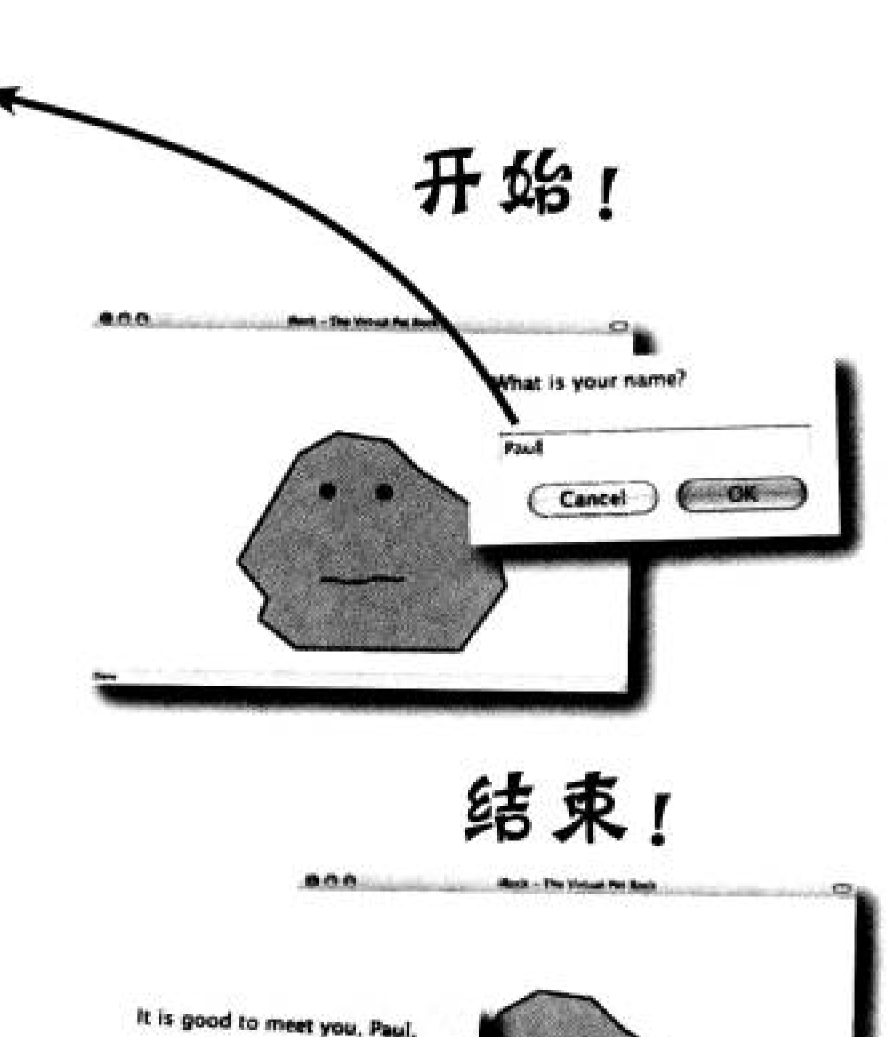
你的脚本也能存储数据	34
脚本满脑子都是数据类型	35
常量总是相同，变量可能有改变	40
变量刚开始没有值	44
用=初始化变量	45
常量拒绝改变	46
变量名称里卖什么药？	50
变量／常量名称的合法与违法	51
变量名称通常形似驼峰	52
规划Duncan's Donut 网站	56
甜甜圈计算首试	58
初始化你的数据……不然就……	61
NaN，非数字	62
加法不只能用在数字上	64
parseInt()与parseFloat()：文本转换为数字	65
为何订了多余的甜甜圈	66
Duncan 发现了甜甜圈间谍	70
用 getElementById()捕捉表单数据	71
验证网页表单的数据	72
努力贴近用户靠直觉输入的内容	77



探索客户端

浏览器探索

有时候，JavaScript 需要知道外在的真实世界如何运作。刚开始，你的脚本或许只是网页里的代码，但最后终将存活在浏览器（或称客户端）打造的世界里。**聪明的脚本**通常需要多了解外在的世界，才能与**浏览器沟通**，才能更加了解外在世界。如果和浏览器打好关系，无论是找出屏幕尺寸还是访问浏览器的打盹按钮（snooze button），脚本能利用的工具可多呢！



客户端、服务器端、JavaScript	86
先问浏览器能为你做什么	88
iRock 需要更有反应	90
定时器连接了行动与时间延迟	92
拆解定时器	93
设定时间 —— setTimeout()	94
再靠近一点：setTimeout()函数	95
多变的屏幕尺寸，永不消失的申诉电话	99
以document对象取得客户端窗口的宽度	100
以document对象的特性设置客户端窗口的宽度	101
设定iRock图像的宽度与高度	102
iRock应配合页面调整大小	103
浏览器改变大小时触发onresize事件	107
onresize事件调整了宠物石的大小	108
我们见过吗？记住用户	110
每个脚本都有大限	111
cookie延长脚本的生命周期	112
cookie记录名称与数据值……但也可能过期	117
你的JavaScript能活得比网页更长久	119
利用cookie欢迎用户	120
greetUser()有了cookie的神力加持	121
别忘记设定cookie	122
cookie 影响浏览器的安全性	124
没有cookie的世界	126
与用户沟通……讲了总比没讲好	129

决策

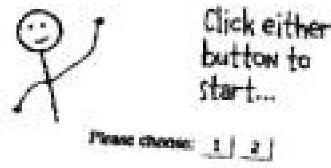
前有叉路，面对抉择

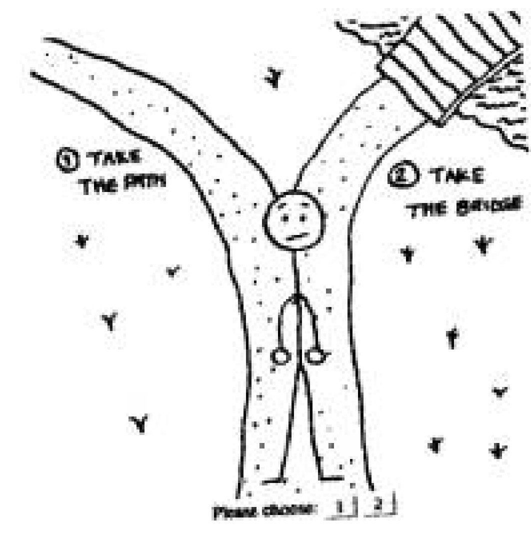
生命中需要太多抉择。该停车或继续开、奶酪或布丁、当个污点证人或俯首认罪……没有下决定的能力，就无法完成任何工作。JavaScript也有相同的状况——**决策（decision）让脚本在不同结果间作决定。作决定，为你的脚本带来"故事"**，即使是最世俗的脚本，或多或少都牵涉到故事。我是否信任用户输入的内容，并为她预订一趟寻找萨斯考奇大脚生物的神秘之旅？还是应该再检查一下，或许她想预订前往加拿大萨斯彻温省的巴士？决定权在你。

幸运的参赛者，来吧！ 136
"如果"是真的……就这样做…… 138
if语句负责评估条件……然后采取行动 139
利用if二选一 141
你可用 if作很多选择 142
在if语句里加个else 143
变量驱动故事 146
但部分故事不见了 147
综合你的JavaScript原力 148
if/else制造出阶梯式决策树 154
if可以放在别的if里 155
函数掌控你的网页 157
利用伪代码规划冒险蓝图 158
火柴人不平衡 162
!= 嘘，我没有什么能告诉你…… 163
由比较运算符精心制作的决策成品 164
注释、占位符、说明文档 166
JavaScript的注释以 // 起始 167
作用域与上下文：数据的生活 169
检查变量作用域 170
数据生活在哪里 171
5个选择 174
嵌套的if/else可能变得太复杂 175
switch语句能用很多"箱子" 177
在switch语句中 178

Welcome to

STICK FIGURE ADVENTURE

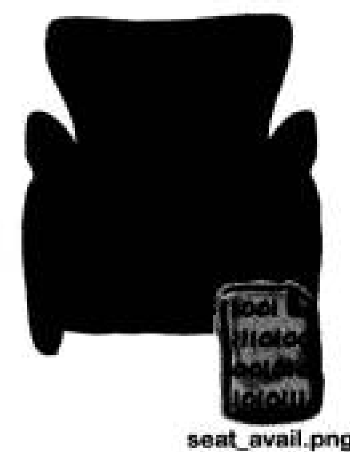




5 循环
自我重复的风险

有些人说，一点重复是生活中的香料。没错，尝试新鲜有趣的事物确实很刺激，但一点点重复真的有可能完成一天的工作。强迫性地消毒双手、让人神经紧张的嘀嗒声、每次收到邮件就按下“全部回复”……好吧，或许现实生活里的重复不见得都是好事。但在 JavaScript 的世界里，“一点重复”可能非常便利。我们需要脚本**重复运行代码**的程度很让人惊讶，循环功能因而闪闪发光。没有**循环**，你将浪费许多时间用于剪贴代码。

空位

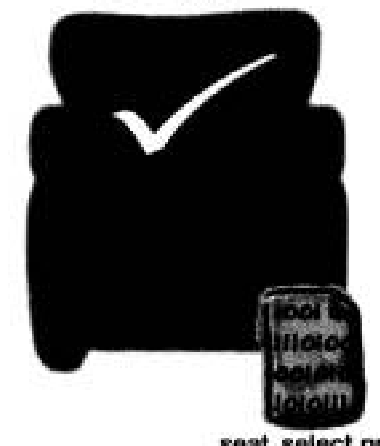


seat_avail.png

已售位

seat_unavail.png

已选择



seat_select.png

X 标示藏宝地点 190
似曾相识……循环 191
引入 for 循环的寻宝游戏 192
解剖循环 193
Mandango：壮汉专用电影院划位工具 194
先确认剩下的座位 195
循环、HTML与空位 196
电影院座位变量 197
数组收集了多块数据 198
数组值与键一起存储 199
从JavaScript到HTML 203
Mandango的划位状况：视觉版 204
试行：寻找单一座位的程序 209
太多好事也不好：结束不了的循环 210
循环总是需要一个（或两个）结束的信号！ 211
休息一下：break 212
boolean运算符的逻辑揭密 218
继续执行while循环，直到遇见条件句 222
停止while循环 223
选对循环 225
电影院座位数据建模 231
数组里的数组：二维数组 232
访问二维数组数据的两个键 233
2-D的Mandango 235
整间电影院的座位都很有男人味 238

6 函数

简化、重复利用、回收再利用

如果JavaScript里有整体环境移动的状况，将由函数领头。函数能让你把JavaScript代码变得更有效率，而且更能重复使用。函数是面向任务的，适用于组织代码，也是极佳的问题解决方案。听起来函数的履历表不错嘛！事实上，只有最简单的脚本才与函数重组代码的优势无关。虽然很难计算每个函数的二氧化碳排放量，但它们总能尽量帮助脚本更环保一点。

问题之母 244

函数，问题解决者 246

函数的中枢 247

你已经遇过的函数 248

建立数据更多的温度调节器 251

传递信息给函数 252

函数自变量作为数据 253

函数去除重复代码 254

创建设定座位的函数 257

setSeat()函数让Mandango更好 259

反馈的重要性 261

从函数返回数据 262

很多快乐的返回值 263

取得座位状态 267

显示划位状态 268

函数可与图像链接 269

重复的代码绝非好事 270

功能与内容分离 271

函数只是数据 272

调用或引用你的函数 273

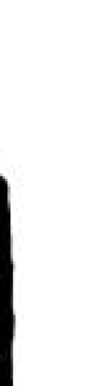
事件、回调与HTML属性 277

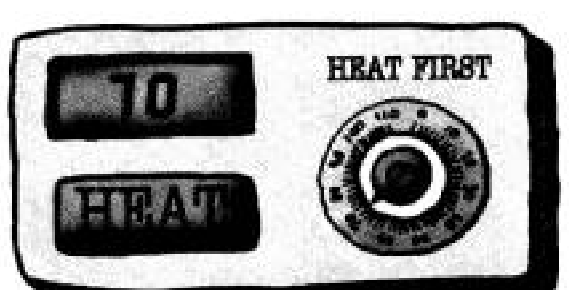
使用函数引用联结事件 278

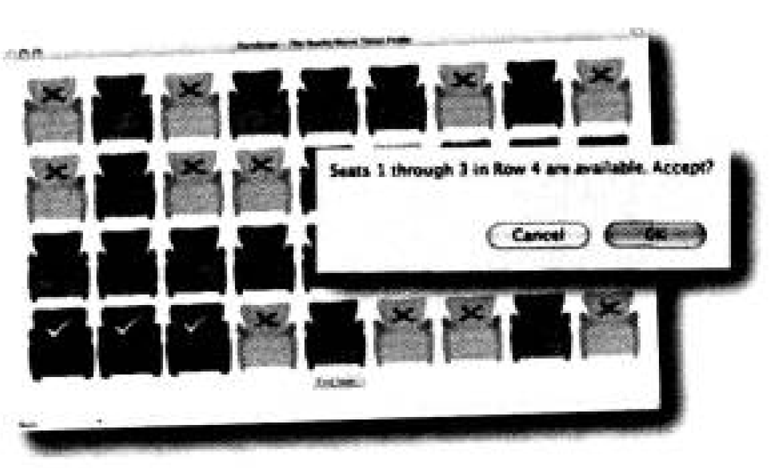
函数字面量拔刀相助 279

联结何在？ 280

以HTML网页为外壳 283







表单与验证

让用户全盘托出

有了JavaScript，你不用风度翩翩或鬼鬼祟祟，一样能成功取得用户的信息。但你必须千万小心。人类很容易犯错，也就是说，在线表单提供的数据不见得全都**准确**或合格。JavaScript能帮上一点忙。把输入的**表单数据传给JavaScript代码，可让网络应用程序更可靠，也能为服务器减少一些负担。珍贵的带宽应让给重要事物**，例如精彩的影片或可爱宠物的照片。

Bannerocity的HTML表单	291
当只有HTML还不够的时候	292
访问表单数据	293
表单域带来一连串事件	295
不再是焦点：onblur	296
可使用alert框呈现验证信息	297
确认域“并非无物”的验证	301
不使用烦人alert框的验证	302
更精密的非无物验证	303
尺寸很重要……	305
验证数据长度	306
验证邮政编码	311
验证日期	316
正则表达式一点都不“正规”	318
正则表达式定义用于匹配的模式	319
元字符不只表示一个字面量字符	321
深入正则表达式：限定符	322
利用正则表达式验证数据	326
匹配最少次数与最多次数	329
削除3位数的年份，该使用这个……或那个	331
不留下任何需要的改变	332
你听到了吗？电话号码的验证	333
你有一封新邮件：验证电子邮件地址	334
例外也是规则	335
从集合中匹配可选字符	336
构建邮件地址验证工具	337

8 驾驭网页

利用DOM分割HTML

利用JavaScript控制网页内容其实很像烹饪。 只是不用收拾残羹剩肴，但也没办法享受美味的成果。不过，你将能**完整访问**网页内的HTML构成要素，更重要的是，你将拥有**替换**网页成分的能力。**JavaScript让你可以随心所欲地操控网页内的 HTML代码**，从而打开各种有趣的机会之门，一切都是因为**标准对象**：DOM（Document Object Model）的存在。

能用但是粗糙简陋……界面很重要 344

不使用alert框描述情节 345

访问HTML元素 347

与内在的HTML建立接触 348

见树也见林：Document Object Model（DOM） 353

你的网页是DOM节点的集合 354

利用特性攀爬DOM树 357

使用DOM改变节点文本 360

与标准兼容的大冒险 365

设计更好、更干净的选项 367

重新思考节点文本替换功能 368

改用函数取代节点文本 369

动态选项真正好 370

更加改良的交互选项 371

样式的问题：CSS与DOM 372

交换样式类` 373

有格调的选项 374

测试加上样式的选项 375

选项有问题：空按钮 376

调整单项样式 377

不再出现假选项 379

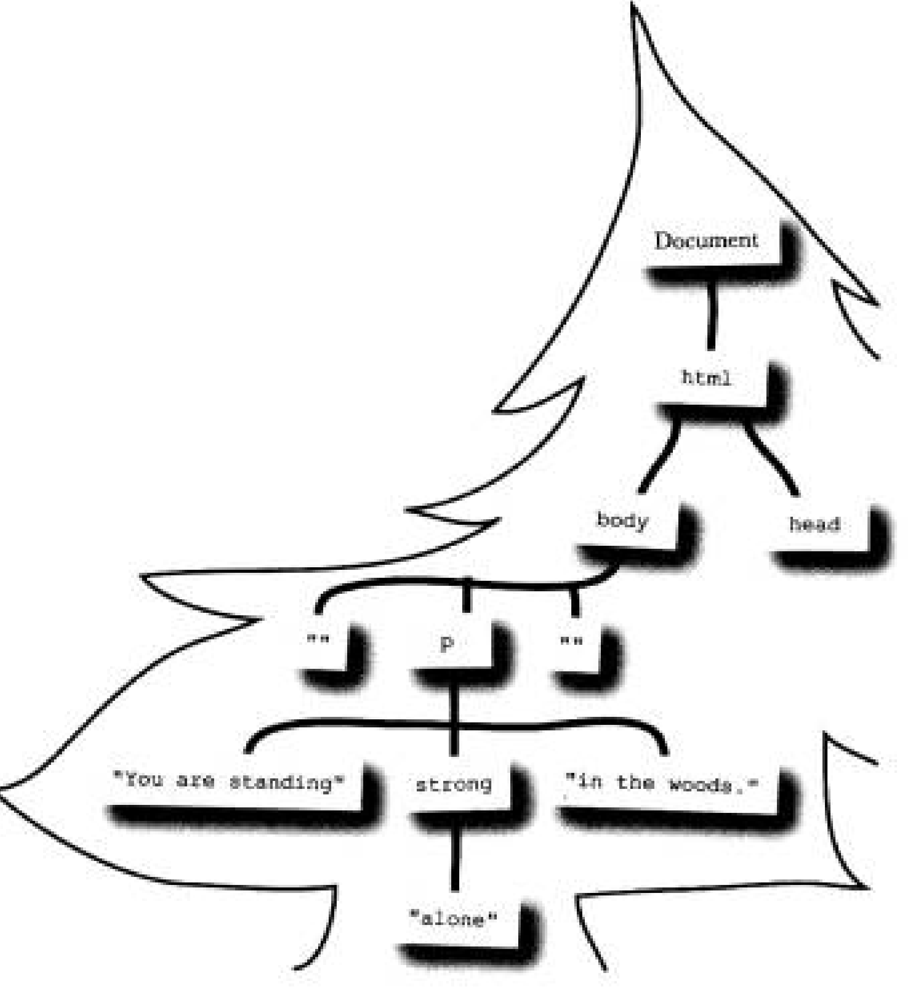
更多选项，更为复杂 380

追踪决策树的轨迹 382

把决策历程改为 HTML 383

操纵HTML代码 384

追溯冒险历程 387



为数据带来生命

科学对象怪人

JavaScript对象不见得总像医生说的一样阴森可怕。但它们拼凑起JavaScript语言的林林总总，所以它们团结在一起的力量比较大。**对象结合数据与行为**，以创建一种新的**数据类型**，比起我们到目前为止见过的其他数据更“活灵活现”。你将拿到**可以自行排序的数组、可以自我搜索的字符串**，还有会长毛、会在满月时咆哮的脚本！最后一项可能是我瞎说的，不过你了解我的意思了……

JavaScript 开派对 394
数据 + 行为 = 对象 395
对象拥有自己的数据 396
以点号引用对象成员 397
自定义对象扩展了JavaScript 401
构造你的自定义对象 402
构造函数里有什么？ 403
为博客对象带来生命 404
排序的需求 409
日期专用的JavaScript对象 410
计算时间 411
重新思考博客的日期 412
对象里的对象 413
对象转换成文本 416
访问日期格式的片段 417
视数组为对象 420
自定义数组排序方式 421
利用函数字面量，排序变得简单 422
搜索blog数组 425
搜索字符串内部：indexOf() 427
搜索blog数组 428
搜索功能也上线了！ 431
Math对象是个organizational object 434
使用Math.random产生随机数 436
把函数转变为方法 441
崭新的blog对象公开亮相 442
对象为YouCube做了什么？ 443

数据

```
var who;
var what;
var when;
var where;
```

\+ **行为**

```
function display(what, when, where) {
  ...
}
function deliver(who) {
  ...
}
```

= **对象**

```
var who;
var what;
var when;
var where;

function display() {
  ...
}
function deliver() {
  ...
}
```

10 创建自定义对象

自定义对象让你为所欲为

如果有这么简单，当然谁都想做。JavaScript没有退货保证，但你绝对可以让它照你的意思行动。自定义对象，就像JavaScript版的极热去奶泡不加鲜奶油要三份低咖啡因浓缩咖啡的大杯摩卡玛奇朵……好一杯特调咖啡啊！有了自定义JavaScript对象，你也能煮出任你操纵又利用了特性和方法优势的代码。最后甚至能制造出可重复使用的面向对象程序代码，既扩展了 JavaScript 语言的效率……而且又只为你服务！

重新查看YouCube的Blog方法 450

方法过载 451

类与实例 452

根据类创建实例 453

以this访问实例的特性 454

拥有一次，运行多次：类拥有的方法 455

在类层，使用prototype 456

类、prototype与YouCube 457

类特性也能共享 462

使用prototype创建类对象 463

签名且传递完毕 465

日期格式化方法 468

扩展标准对象 469

自定义的日期对象 = 改良的 YouCube 470

类能有自己的方法 471

检验排序比较函数 473

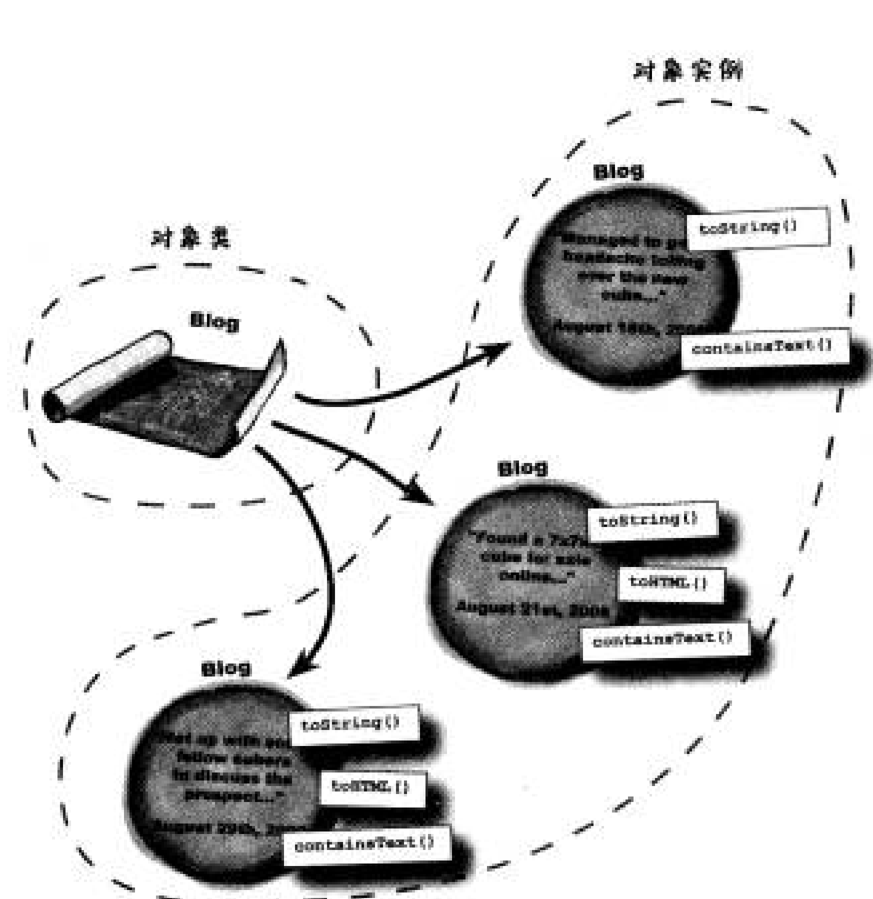
调用类方法 474

一张图胜过千言万语 475

把图像整合至 YouCube 476

为 YouCube 添加图像 478

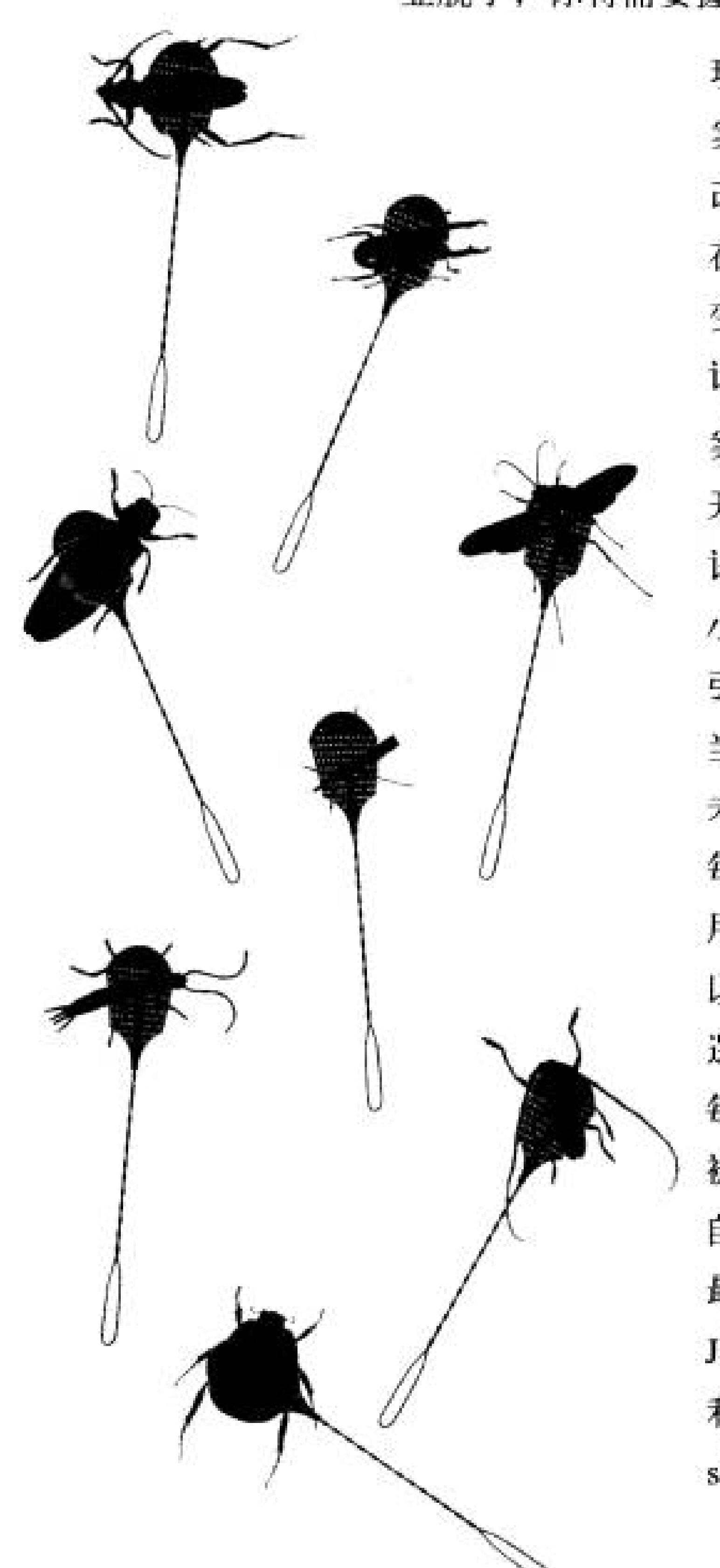
由对象驱动的 YouCube 480



11 除错务尽

好脚本也会出错

即使最佳规划的JavaScript蓝图也可能出错。当这种情形发生，事实上一定会发生，你的工作就是 —— 别惊慌。最好的JavaScript开发人员，不是从未造成程序出错的人 —— 这种人叫骗子。最好的JavaScript开发人员，是能够**成功猎捕并消灭**自己制造的**缺陷**（bug）的人。还有，一流的JavaScript缺陷终结者也会**发展出良好的编程习惯**，使最狡诈、最棘手的缺陷的出现率降到最低。**小心驶得万年船。**但缺陷如果登舰了，你将需要握紧自动机枪，准备战斗……



现实生活中的除虫大队	486
案件：爬满臭虫的 IQ 计算器	487
改用不同浏览器	488
在轻松街道上除虫	491
变量未定义	495
计算智商	497
案件：电台直播来电缺陷	498
开始调查	499
语法验证问题（Bug #1）	500
小心处理字符串	501
引号、撇号，还有一致性	502
当引号不是引号，使用转义符	503
未定义不只是变量的专利（Bug #2）	504
每个人都是赢家（Bug #3）	506
用 alert 框调试	507
以 alert 观察变量	508
逻辑差虽然合法却有缺陷	510
每个人都是输家！（Bug #4）	514
被烦人的 alert 淹没	515
自定义一个调试控制台	517
最麻烦的错误：运行时	524
JavaScript 三虫客	525
利用注释暂时禁用代码	528
shadow variable的危险	530

12 动态数据

贴心易感的网络应用程序

当代万维网是个非常有反应的世界，网页被期待着响应读者的每声呻吟，至少许多网站用户和开发者都有这个梦想。透过一个称为Ajax —— 提供戏剧性地改变网页"感觉"机制的编程技术，JavaScript在这个梦想中扮演了重要角色。有了Ajax，网页的行为比较像完全成熟的应用程序，因为它们能够**动态地快速下载并存储数据以及实时响应用户**，却不用重新载入网页或利用浏览器的小手段。

渴望着动态数据	538
由数据驱动的YouCube	539
Ajax献身沟通	541
XML让你用自己的标签标示数据	543
XML + HTML = XHTML	545
XML 与 YouCube 博客数据	547
为YouCube注入Ajax	550
援助Ajax的JavaScript：XMLHttpRequest	552
GET或POST？使用XMLHttpRequest的请求	555
理解Ajax请求	559
交互式网页就从请求对象开始	563
完工后再呼唤我	564
处理响应……天衣无缝	565
DOM 拔刀相助	566
YouCube 改由数据驱动了	571
不正常的按钮	573
按钮需要数据	574
省时的网络版新增日志方案	577
写出博客数据	578
PHP 也有需要	581
供应数据给 PHP脚本	582
组合代码：发表文章到博客服务器上	585
让 YouCube 更……好用	590
为用户自动填写域	591
重复的任务？不如来个函数吧？	592

1 交互式网络

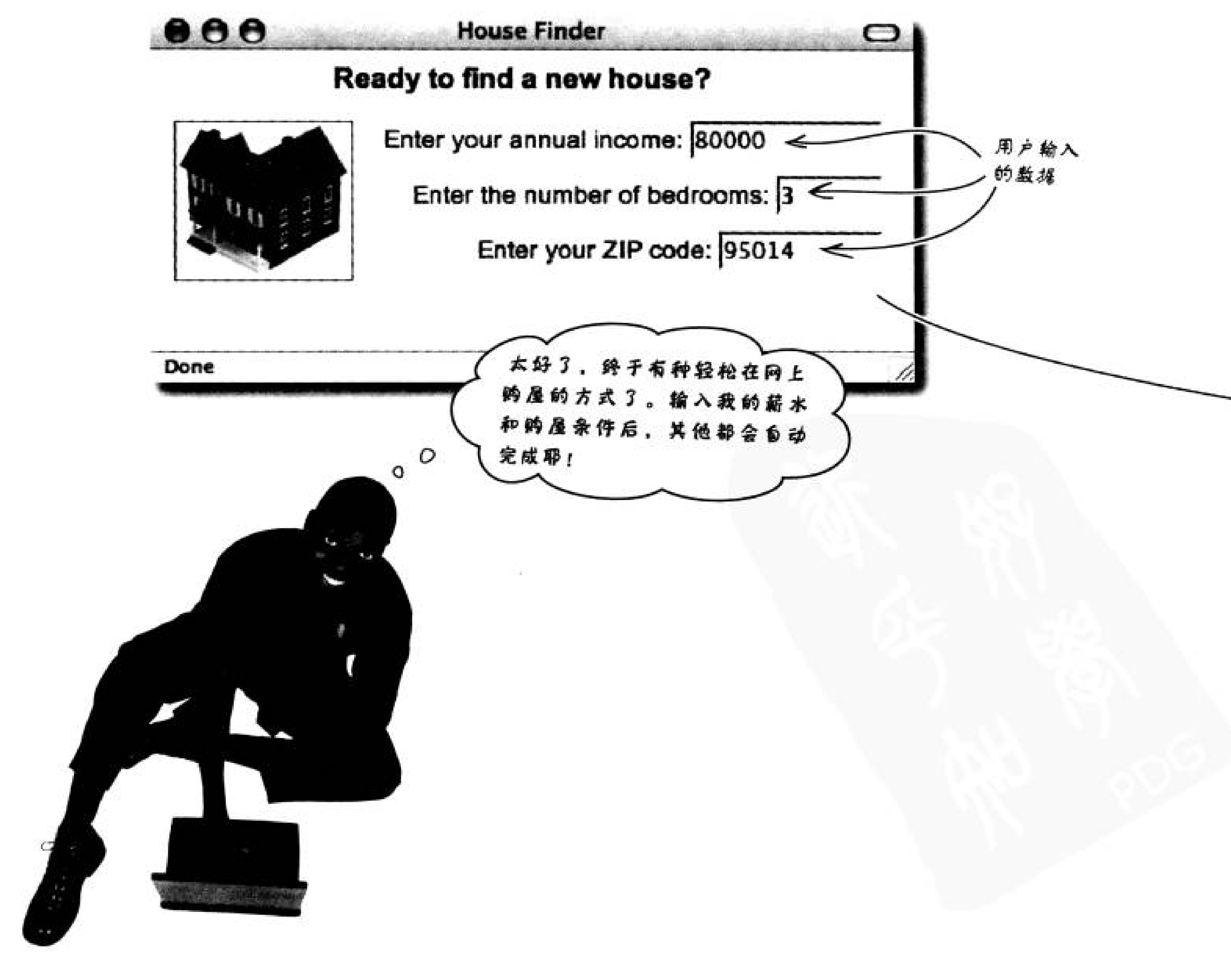
感觉虚拟世界

不想只把万维网视为被动网页的同义词吗？

我们完全了解你的心情。被动的信息应该留给“书籍”这种形式。书籍适合阅读、学习，还有很多很多用途，但与**交互式**一点关系也没有。如果少了JavaScript，万维网就跟书没什么两样了。当然，没有JavaScript，还是可以提交表单，或许再利用HTML与CSS耍些唬人的花招……不过，这只是在了无生趣的网页上操纵无生命的傀儡而已。真正让网页活起来的**交互性**，还需要**多一点智慧与多一点努力**……但保证带来**更多报酬**。

（在线）用户的需求

好，我们知道万维网（Web）是个虚拟世界，但使用万维网的人都是真正的人。人的需求可能包括从寻找超级美味的炖肉卷食谱、使用Meatloaf下载最喜欢的歌曲，到购买新家这类重大需求。幸好，万维网能针对你的优先需求量身打造。

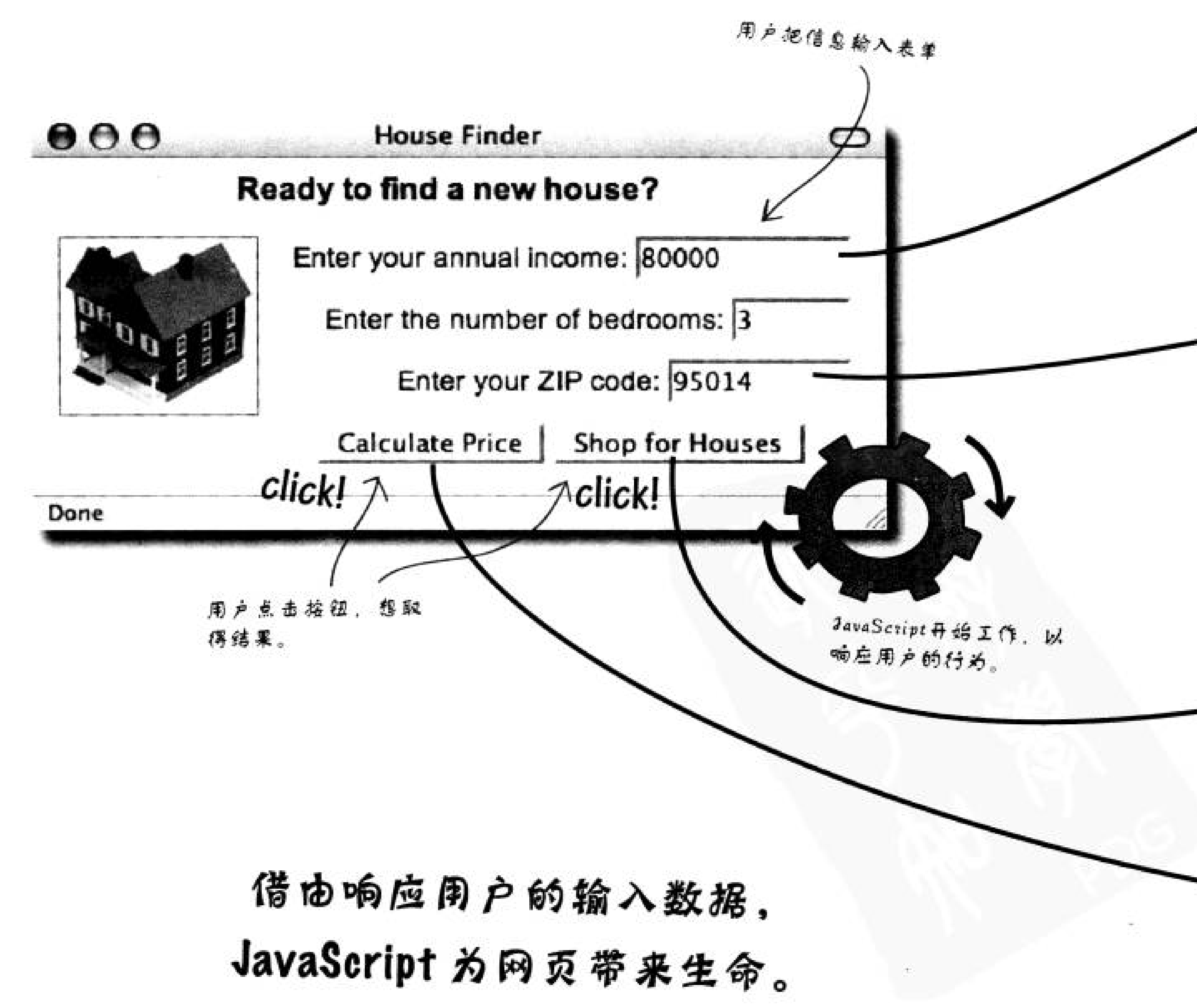


对墙弹琴……毫无反应

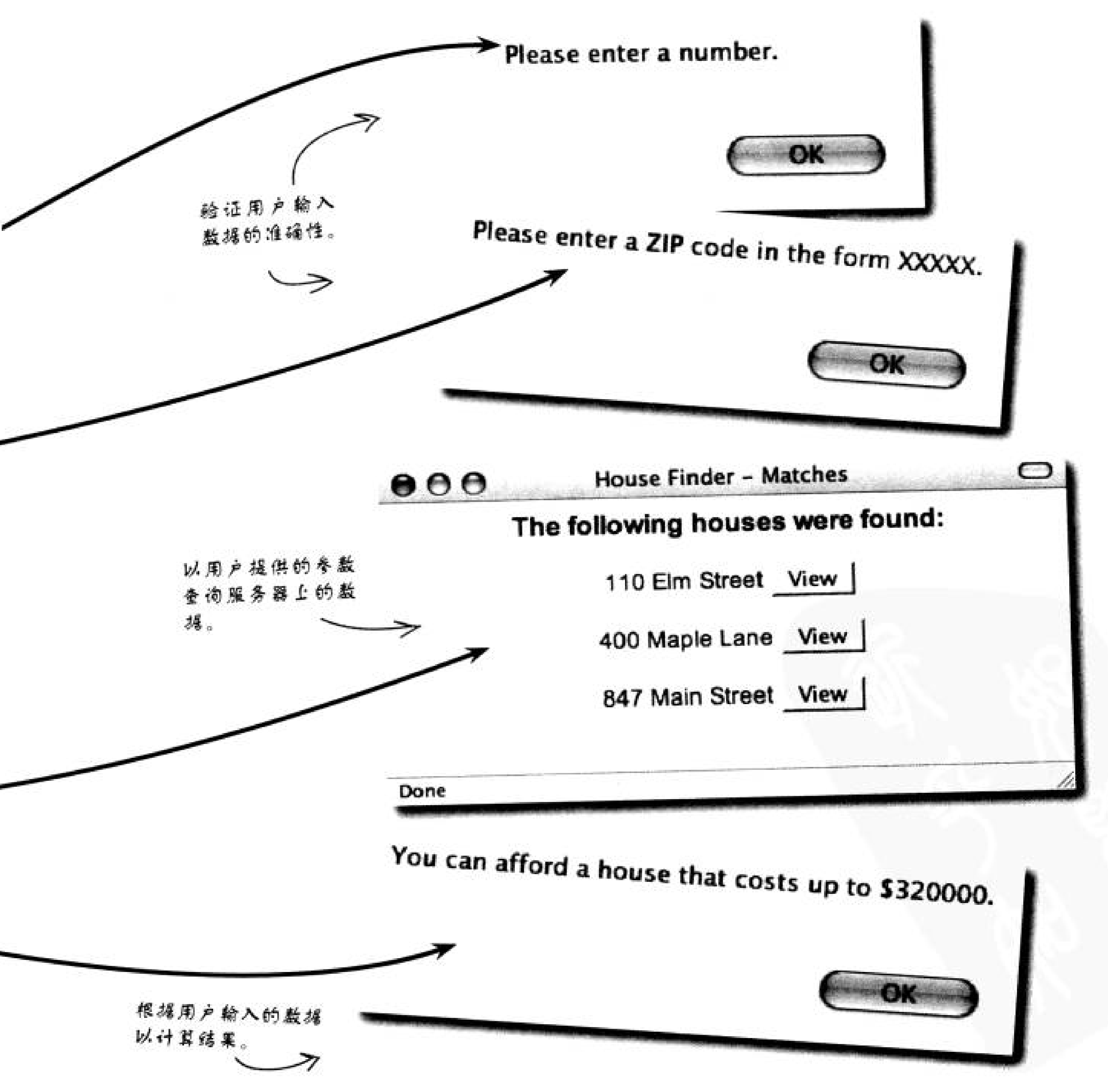
万维网的响应程度不见得都如想象中美好。事实上，它也有冷淡至极、毫无知觉、完全与外界隔离、对用户的需求不理不睬的时候。你以为输入一些数据就会跑出某些响应……结果居然什么都没有。别以为电脑或网络跟你有仇，那是因为静态万维网的能耐到此为止。

JavaScript 在此为您服务

JavaScript 的出现，使网页浏览一转身变成互动经验。它让冷冰冰的事物能听取我们的需求、处理输入数据，并响应我们最深层的渴望……好吧……我可能有点过度引申，不过 JavaScript 可以把网页转变为交互式应用程序，完全不同于无生命的静态网页，这就是好事！

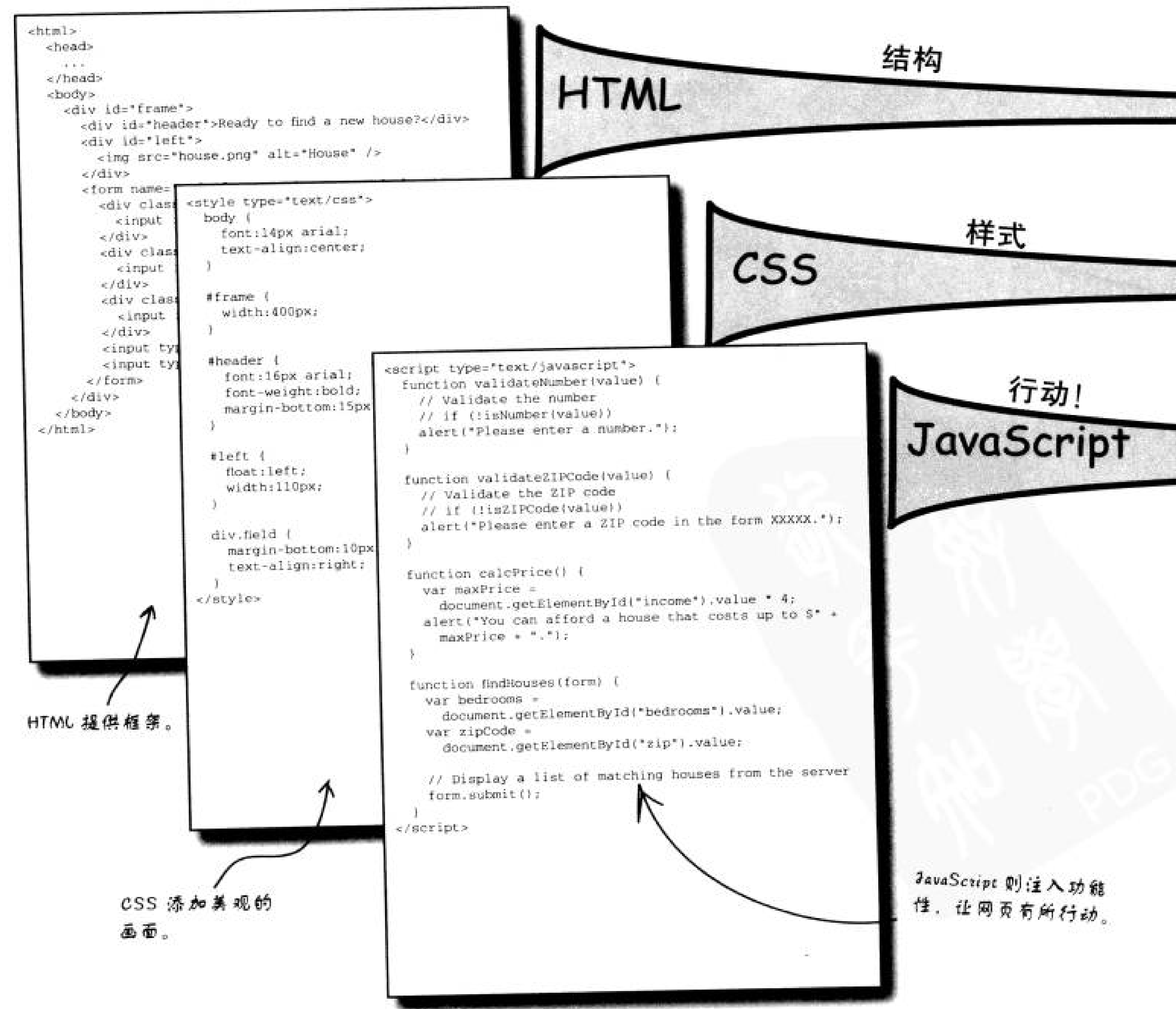


借由响应用户的输入数据，JavaScript 为网页带来生命。


Please enter a number.
OK
验证用户输入数据的准确性。
Please enter a ZIP code in the form XXXXX.
OK
House Finder - Matches
The following houses were found:
110 Elm Street View
400 Maple Lane View
847 Main Street View
Done
以用户提供的参数查询服务器上的数据。
You can afford a house that costs up to $320000.
OK
根据用户输入的数据以计算结果。

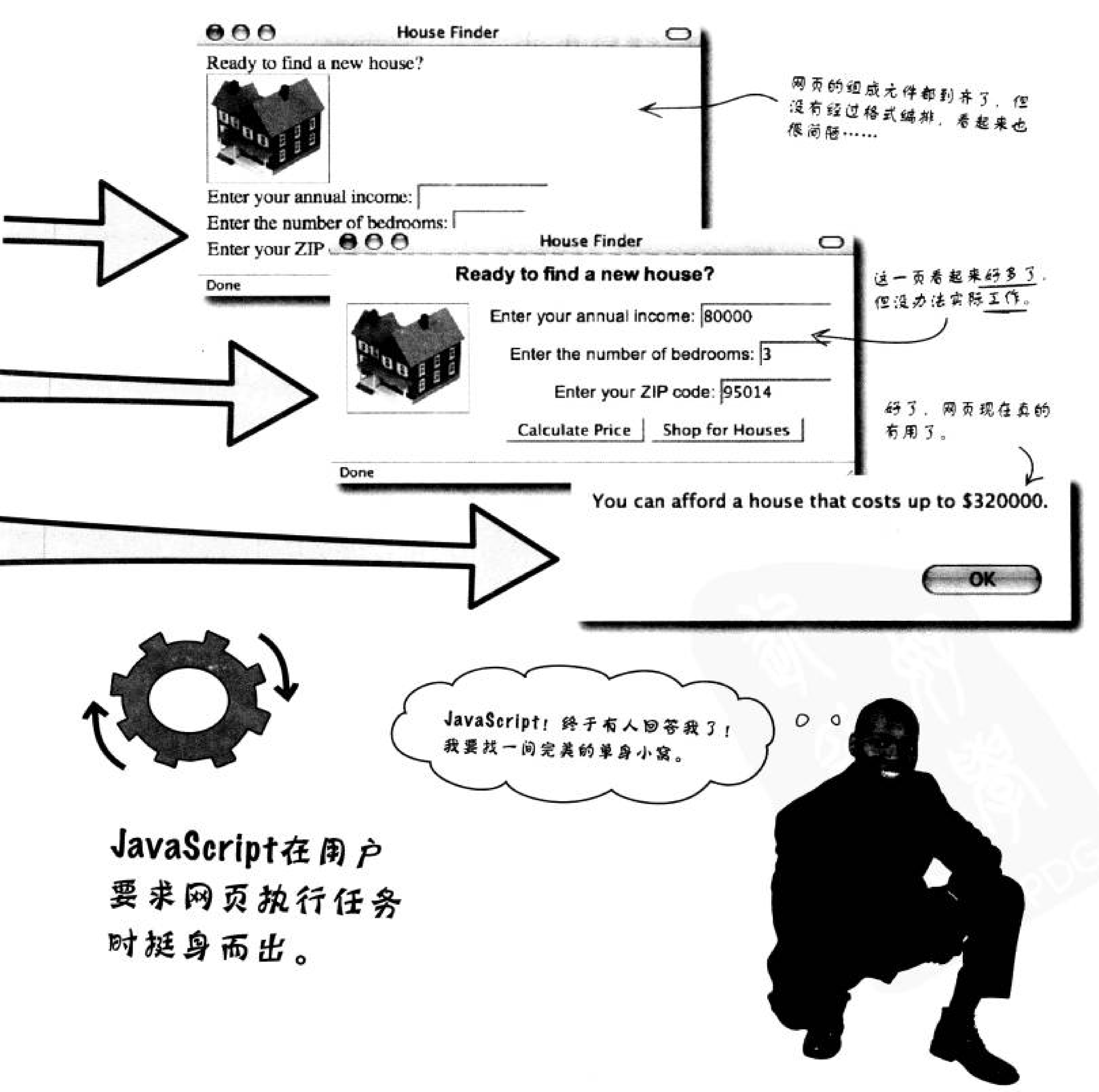
钢骨、烤漆、上路喽！

JavaScript与HTML、CSS同为构成网页的三大元素。HTML像是车身骨架，CSS是美化门面的烤漆，JavaScript则像轮胎，遵循**结构**（structure，HTML）、**样式**（style，CSS）、**行动**（action，JavaScript）的路径，它们同心协力把交互性推行上路。JavaScript代码与CSS相似，通常都直接放在网页中。




```
<style type="text/css">
  body {
    font:14px arial;
    text-align:center;
  }

  #frame {
    width:400px;
  }

  #header {
    font:16px arial;
    font-weight:bold;
    margin-bottom:15px
  }

  #left {
    float:left;
    width:110px;
  }

  div.field {
    margin-bottom:10px
    text-align:right;
  }
</style>
```

```
<script type="text/javascript">
  function validateNumber(value) {
    // Validate the number
    // if (!isNumber(value))
    alert("Please enter a number.");
  }

  function validateZIPCode(value) {
    // Validate the ZIP code
    // if (!isZIPCode(value))
    alert("Please enter a ZIP code in the form XXXXX.");
  }

  function calcPrice() {
    var maxPrice =
      document.getElementById("income").value * 4;
    alert("You can afford a house that costs up to $" +
      maxPrice + ".");
  }

  function findHouses(form) {
    var bedrooms =
      document.getElementById("bedrooms").value;
    var zipCode =
      document.getElementById("zip").value;

    // Display a list of matching houses from the server
    form.submit();
  }
</script>
```



JavaScript在用户要求网页执行任务时挺身而出。

不能只用HTML与CSS就好了吗？在JavaScript出现前，网络就已经很好了啊，你不是也看过吗？

HTML与CSS没有真正的交互性

问题在于HTML与CSS没有真正的交互性。CSS确实提供了许多诀窍，可于各种情况下（例如鼠标移过超链接时）操纵样式，但只使用HTML与CSS的话，其实你会觉得绑手绑脚，空有满怀壮志却无法施展。

JavaScript则几乎能让我们检测一切网页上的行动，例如用户点击按钮、调整浏览器窗口大小或在文本域里填入数据等等。而且，JavaScript是种脚本语言（scripting language），我们可以学着设计响应用户行动的程序码，例如执行计算、动态切换网页上的图像，甚至还能验证数据。

先别对JavaScript 的细节举白旗投降，现在还不到那个时候。

虽然 JavaScript 的能耐不小，但我知道各位才刚接触它。后续内容保证把事件（event）、函数（function）与众多 JavaScript 之谜及时奉上。另外，你可能已对 JavaScript 稍有了解，只是自己不知道罢了。

HTML + CSS + JavaScript = 真正的交互性

磨笔上阵

你对JavaScript的了解比你想象的更多。请看下列House Finder的网页，并在旁边注记各段JavaScript代码的用途。猜猜看嘛！猜错又不会少块肉！

```
<html>
  <head>
    <title>House Finder</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // Validate the number
           if (!isNumber(value))
        alert("Please enter a number.");
      }

      function validateZIPCode(value) {
        // Validate the ZIP code
           if (!isZIPCode(value))
        alert("Please enter a ZIP code in the form XXXXX.");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert("You can afford a house that costs up to $" + maxPrice + ".");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // Display a list of matching houses from the server
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">Ready to find a new house?</div>
      <div id="left">
        <img src="house.png" alt="House" />
      </div>
      <form name="orderform" action="..." method="POST">
        <div class="field">Enter your annual income:
          <input id="income" type="text" size="12"
          onblur="validateNumber(this.value)"/></div>
        <div class="field">Enter the number of bedrooms:
          <input id="bedrooms" type="text" size="6"
          onblur="validateNumber(this.value)"/></div>
        <div class="field">Enter your ZIP code:
          <input id="zip" type="text" size="10"
          onblur="validateZIPCode(this.value)"/></div>
        <input type="button" value="Calculate Price"
          onclick="calcPrice();" />
        <input type="button" value="Shop for Houses"
          onclick="findHouses(this.form);" />
      </form>
    </div>
  </body>
</html>
```

磨笔上阵解答

你对 JavaScript 的了解比你想象的更多。请看下列 House Finder 的网页，并在旁边注记各段 JavaScript代码的用途。猜猜看嘛！猜错又不会少块肉！

```
<html>
  <head>
    <title>House Finder</title>
    <script type="text/javascript">
      function validateNumber(value) {
        // Validate the number
           if (!isNumber(value))
        alert("Please enter a number.");
      }

      function validateZIPCode(value) {
        // Validate the ZIP code
           if (!isZIPCode(value))
        alert("Please enter a ZIP code in the form XXXXX.");
      }

      function calcPrice() {
        var maxPrice = document.getElementById("income").value * 4;
        alert("You can afford a house that costs up to $" + maxPrice + ".");
      }

      function findHouses(form) {
        var bedrooms = document.getElementById("bedrooms").value;
        var zipCode = document.getElementById("zip").value;

        // Display a list of matching houses from the server
        form.submit();
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div id="header">Ready to find a new house?</div>
      <div id="left">
        <img src="house.png" alt="House" />
      </div>
      <form name="orderform" action="..." method="POST">
        <div class="field">Enter your annual income:
          <input id="income" type="text" size="12"
          onblur="validateNumber(this.value)"/></div>
        <div class="field">Enter the number of bedrooms:
          <input id="bedrooms" type="text" size="6"
          onblur="validateNumber(this.value)"/></div>
        <div class="field">Enter your ZIP code:
          <input id="zip" type="text" size="10"
          onblur="validateZIPCode(this.value)"/></div>
        <input type="button" value="Calculate Price"
          onclick="calcPrice();" />
        <input type="button" value="Shop for Houses"
          onclick="findHouses(this.form);" />
      </form>
    </div>
  </body>
</html>
```

告知读者邮政编码应为5个数字，格式如 XXXXX。

用户薪水的4倍即为最高房价。

验证薪水（income）域（field），确认输入数字。

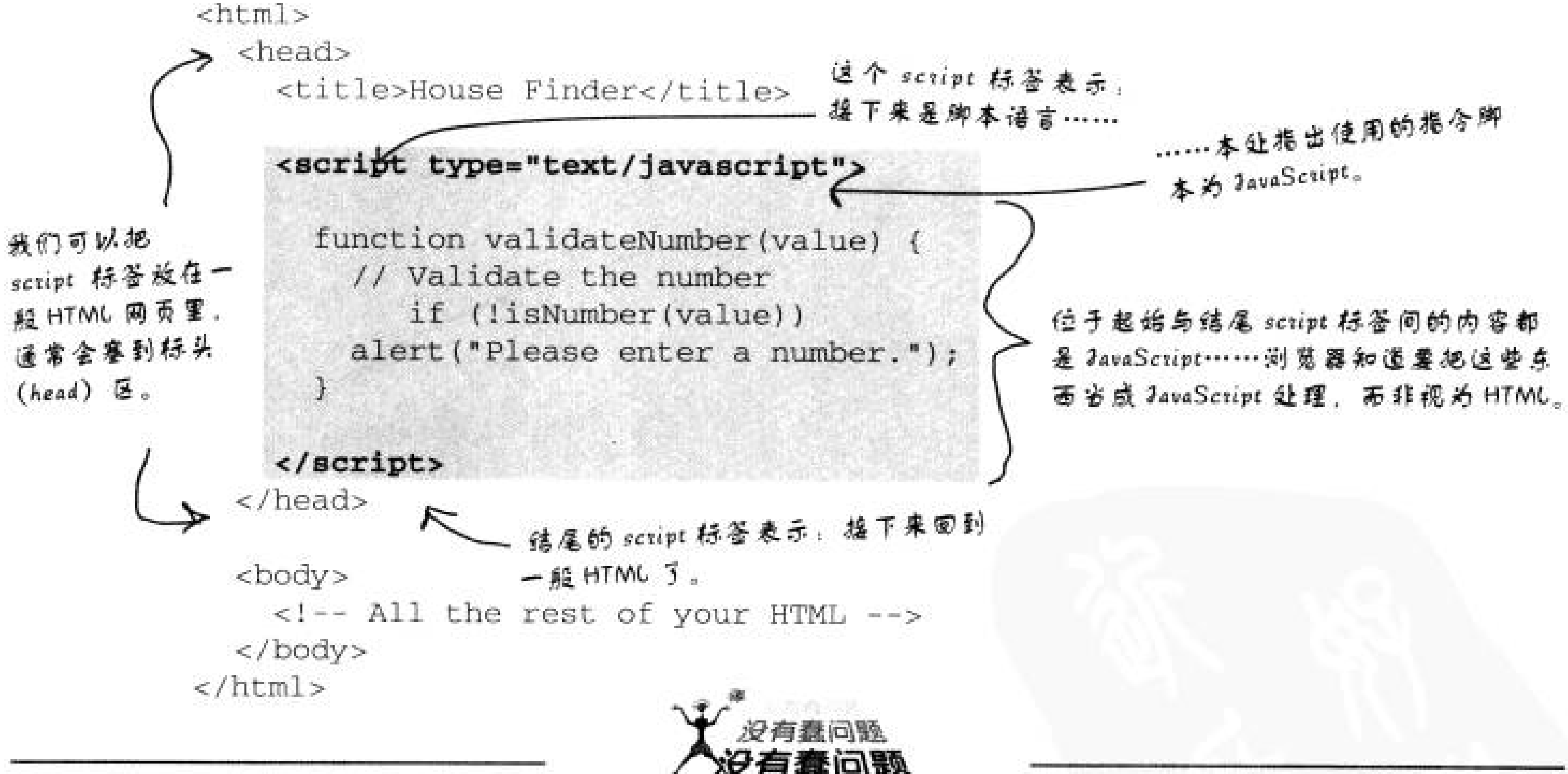
输入邮政编码域的值。

当用户按下 Calculate Price按钮时，计算最高房价。

利用 <script> 标签，向浏览器表示以下为 JavaScript

目前，我们将把 JavaScript直接放入HTML网页里，正如前页范例的格式。第一件事，就是让浏览器知道接下来不是 HTML，JavaScript 即将出现……`<script>` 标签的用处就在这里。

HTML 的任何地方都可以安插 `<script>` 标签，但最好放在网页的 `<head>`区域，如下所示：




```
<html>
  <head>
    <title>House Finder</title>

    <script type="text/javascript">

      function validateNumber(value) {
        // Validate the number
           if (!isNumber(value))
         alert("Please enter a number.");
      }

    </script>
  </head>

  <body>
    <!-- All the rest of your HTML -->
  </body>
</html>
```

没有蠢问题

问：这么说来，我放在 <script> 标签中的一切都是 JavaScript 喽？

答：不见得……`<script>` 标签只不过表示接下来是脚本语言，但不一定会是 JavaScript。说明 type 的属性，`type="text/javascript"` 才是对浏览器指定采用 JavaScript 的部分。

问：还有其他能用的脚本语言吗？

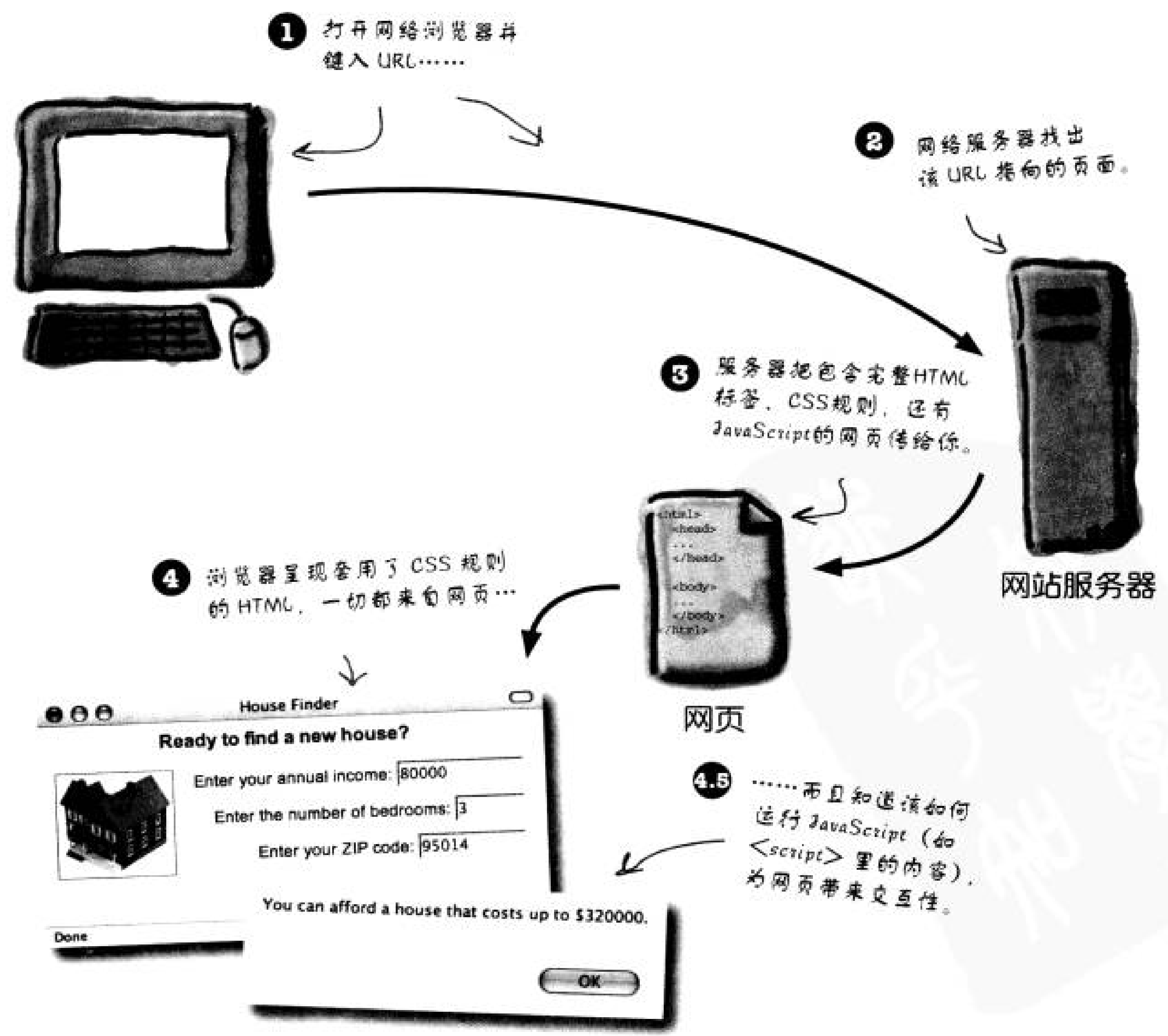
答：当然还有喽！Microsoft就拥有VBScript（脚本的Visual Basic）与ASP.NET AJAX等选择。我们也将于第12章讨论到一般的 Ajax。另外还有不少能用的脚本语言，但本书只会使用 `text/javascript`。

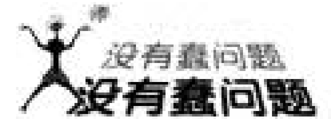
问：我的 <script> 元素一定要在 HTML 页面的 <head> 区域里吗？

答：好问题。`<script>`元素可以放在网页的任何地方，但放在`<head>`以外的区域都被视为坏习惯，这就跟大家不喜欢把CSS塞在网页中间一样……最好让JavaScript与网页的其他内容分离，`<head>`区域正是个绝佳的地方。

你的浏览器可以处理HTML、CSS 还有 JavaScript

各位已经知道，浏览器可以接受并呈现 HTML。各位也已使用 CSS，让浏览器以不同方式美化 HTML 的各个部分。接下来，请把 JavaScript 当成另一种与浏览器对谈的方式……但 JavaScript 并非要求浏览器呈现某些画面（这是 HTML 与 CSS 的工作），而是交给浏览器一些必须遵行的命令。





问：如何让浏览器运行 JavaScript 代码？

答：浏览器里具有一种特殊软件——JavaScript解释器，它的工作就是运行出现在网页中的JavaScript代码。所以你可能听过别人把 JavaScript 称为**解释语言**，与**编译语言**相对。C++ 或 C# 即为编译语言，需由编译器（compiler）转换为可执行的程序文件。因为JavaScript 程序的源代码可由浏览器解释，不一定需要编译器的转换。

问：我该如何让网页开始运行JavaScript代码？

答：大多数 JavaScript代码于网页内发生事件时运行，例如载入网页，或用户点击按钮。JavaScript 有个“事件”（event）机制，可以在网页上发生某个事件时，才触发某一段JavaScript代码。

问：万维网的安全问题不是很严重吗？JavaScript 安全吗？

答：嗯，大致上还算安全。JavaScript的基本设计就考虑到防止恶意代码的问题。例如，JavaScript不允许读/写用户硬盘上的文件。这项限制已抹消大部分病毒与邪恶代码的生存空间。当然，有限制不等于写出来的网页没有缺陷、不会造成用户的痛苦，有限制，只表示没办法用JavaScript陷用户于水深火热中。还有，浏览器的漏洞加上狡黠的黑客，早就找出侵害JavaScript安全性的方式，所以它并非绝对安全。

问：那个…House Finder 网页代码里的 <script> 标签……它是HTML还是JavaScript啊？

答：`<script>`标签本身是HTML，它的作用是提供混杂HTML代码与JavaScript代码的方式。出现在`<script>`标签中的源代码是JavaScript代码。但设计`<script>`标签时，其实希望它能支持多种脚本语言，所以要利用`type`属性指定使用 JavaScript。

问：我看过具有交互性的网页，例如检查输入数据是否正确的表单，看起来不太像需要 JavaScript 的样子。有可能不利用 JavaScript 吗？

答：不使用JavaScript也能获得网页的交互性，但多半缺乏效率，还常来沉重的负担。以数据的验证为例，其实可以在提交表单后由网络服务器处理。然而，这种方式代表要把整份表单提交、等待服务器的验证工作完毕，再把结果作为新网页返回……你干脆自己手动检查表单数据还快一点！JavaScript的交互性完全发生在浏览器里，不需载入新页面，因而减低了数据在服务器间非必要的来回传送。不仅如此，JavaScript提供的大部分交互性如果不用JavaScript的话，则一定要经过第三方浏览器附加程序才能达成。

请看下列片断JavaScript代码，从中分辨出标准 JavaScript 语言，以及程序设计师为House Finder网页自定义的程序代码。

`alert`	JavaScript / 自定义	`onblur`	JavaScript / 自定义
`calcPrice`	JavaScript / 自定义	`onclick`	JavaScript / 自定义
`zipCode`	JavaScript / 自定义	`findHouses`	JavaScript / 自定义
`var`	JavaScript / 自定义	`value`	JavaScript / 自定义

请看下列片断JavaScript代码，从中分辨出标准JavaScript语言，以及程序设计师为House Finder网页自定义的程序代码。

```
<head>
  <title>House Finder</title>
  <script type="text/javascript">
    function validateNumber(value) {
      // Validate the number
      // if (!isNumber(value))
      alert("Please enter a number.");
    }

    function validateZIPCode(value) {
      // Validate the ZIP code
      // if (!isZIPCode(value))
      alert("Please enter a ZIP code in the form XXXXX.");
    }

    function calcPrice() {
      var maxPrice = document.getElementById("income").value * 4;
      alert("You can afford a house that costs up to $" + maxPrice
+ ".");
    }

    function findHouses(form) {
      var bedrooms = document.getElementById("bedrooms").value;
      var zipCode = document.getElementById("zip").value;

      // Display a list of matching houses from the server
      form.submit();
    }
  </script>
</head>

<body>
  <div id="frame">
    <div id="header">Ready to find a new house?</div>
    <div id="left">
      <img src="house.png" alt="House" />
    </div>
    <form name="orderform" action="..." method="POST">
      <div class="field">Enter your annual income:
        <input id="income" type="text" size="12"
        onblur="validateNumber(this.value)"/></div>
      <div class="field">Enter the number of bedrooms:
        <input id="bedrooms" type="text" size="6"
        onblur="validateNumber(this.value)"/></div>
      <div class="field">Enter your ZIP code:
        <input id="zip" type="text" size="10"
        onblur="validateZIPCode(this.value)"/></div>
      <input type="button" value="Calculate Price"
        onclick="calcPrice();" />
      <input type="button" value="Shop for Houses"
        onclick="findHouses(this.form);" />
    </form>
  </div>
</body>
</html>
```

弹出警示无效数字的窗口。
alert — **JavaScript** / Custom

一段计算房价的自定义代码。
calcPrice — JavaScript / **Custom**

为一块数据保留存储位置。
var — **JavaScript** / Custom

寻找符合条件的房屋的自定义代码。
findHouses — JavaScript / **Custom**

zipCode — JavaScript / **Custom**
保留用户所输入邮政编码的存储位置。

onblur — **JavaScript** / Custom
指示用户移往下个输入域。

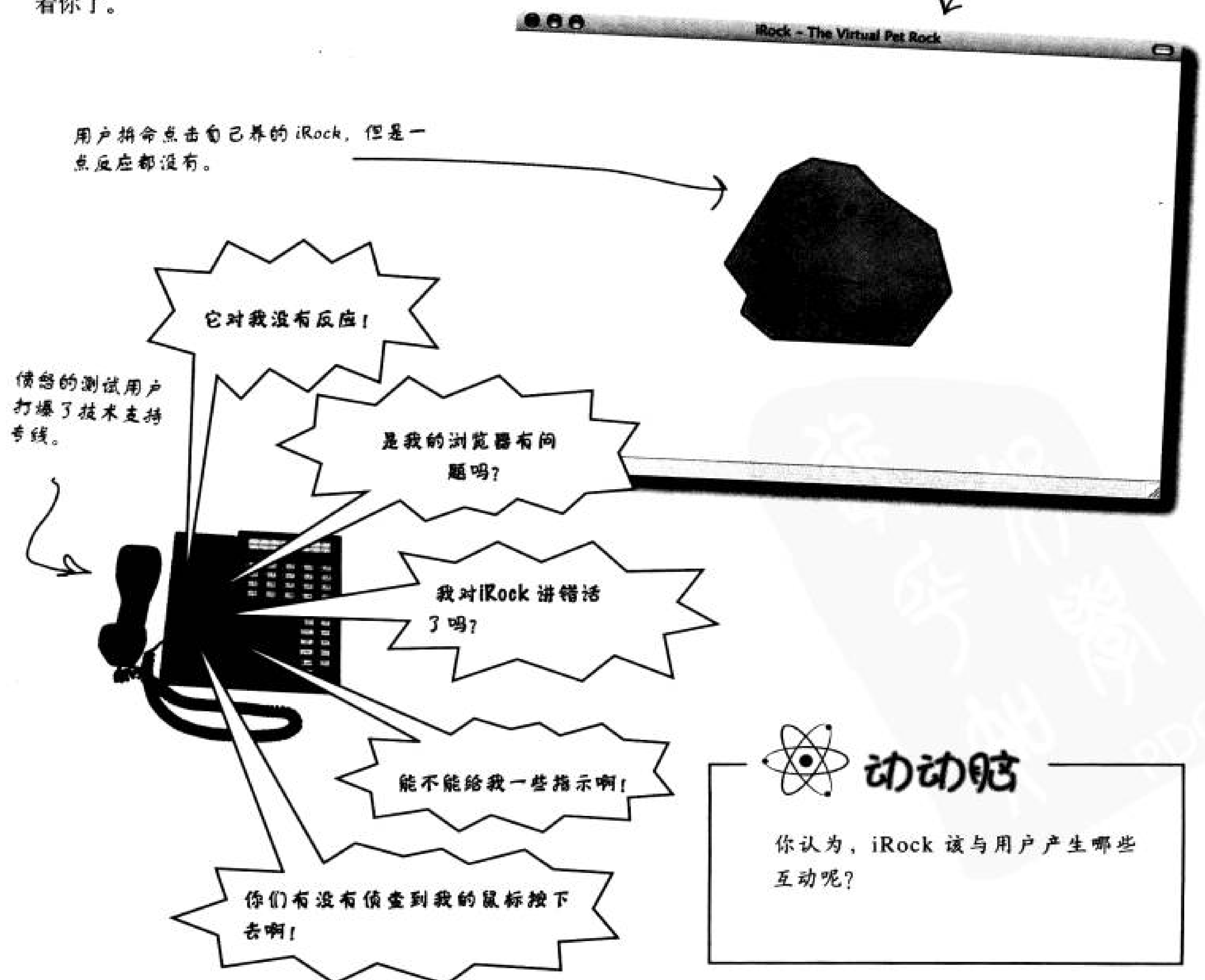
value — **JavaScript** / Custom
目前邮政编码程中的值。

onclick — **JavaScript** / Custom
指示Shop按钮已被点击。

人类的虚拟好朋友……需要你的帮忙

结束一场成功的 HTML 与 CSS 网页设计大会后，老板在办公室里召见你，让你看看他的最新在线发明：iRock。虚拟宠物在玩具业大会已经造成轰动，但我们的测试用户（beta user）却对这只在线宠物不太满意。

非常明显，用户看到石头，自然想点看看有什么新奇好玩的反应会出现……不过你的老板根本没想到要有“反应”这回事。现在，你得负责 iRock的**交互性**，这颗石头会升天成为五彩缤纷的仙石，还是会被丢到垃圾桶里，就全看你了。



动动脑

你认为，iRock 该与用户产生哪些互动呢？

让 iRock 动起来

为 iRock 赋予交互性的任务已经交到你的手上，但在建立交互性的同时，你也要一起学会 JavaScript。听起来还不错，最后你会变出一颗能说"哈啰！"的可爱宠物石。

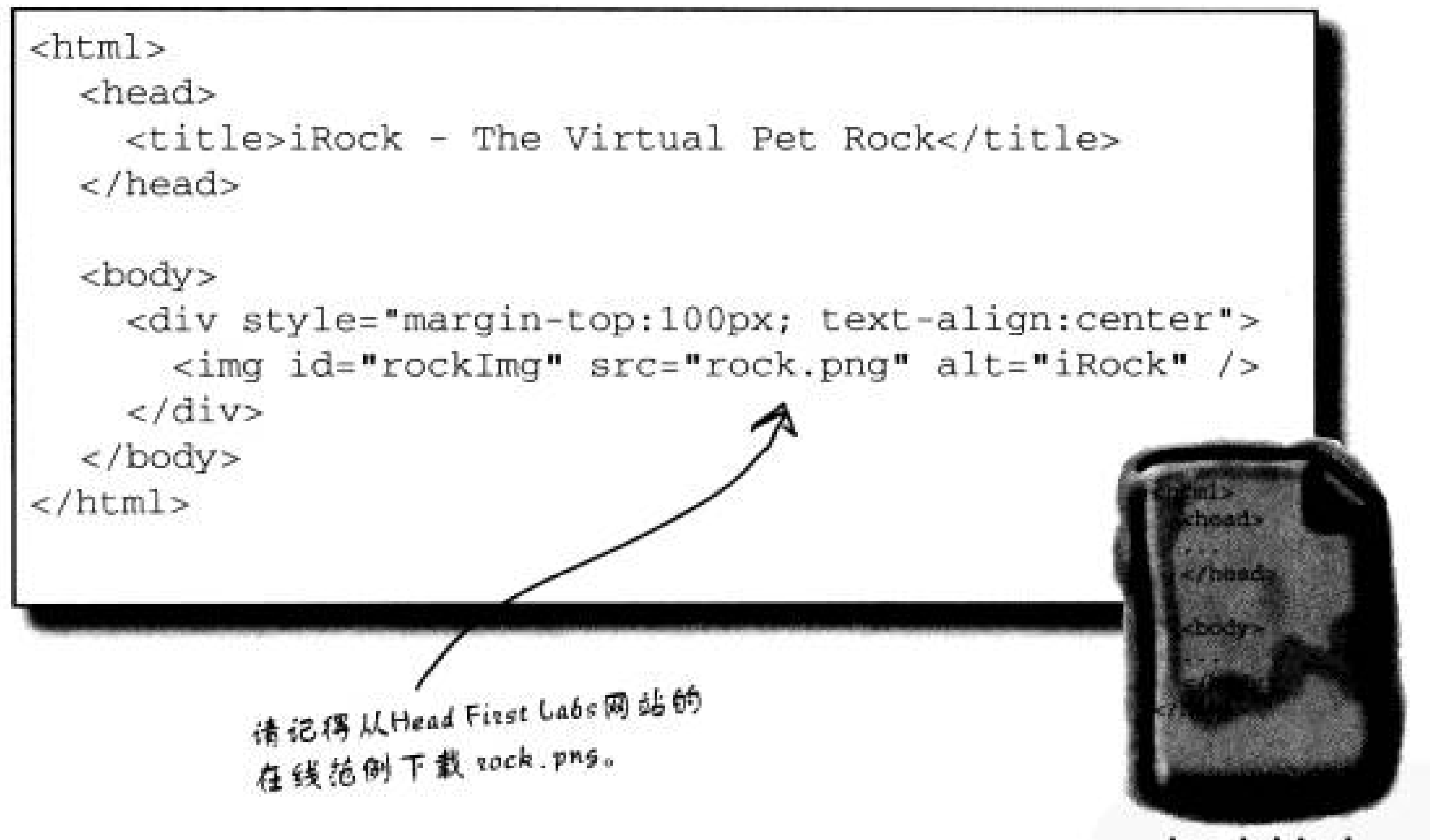
在本章接下来的过程中，你将完成下列事项：

1. 创建 iRock 的 HTML 网页。 你已经知道怎么做了。
2. 加入一个JavaScript的alert框，使iRock网页完全载入后，宠物石会欢迎用户。 alert，是弹出信息窗口的JavaScript方式。
3. 设计JavaScript代码，询问用户姓名，并呈现个性化的欢迎信息，还要让宠物石露出微笑。 这一步需要连接用户的行为，例如点击宠物石……
4. 加入一个事件处理器，于用户点击宠物石时，运行我们在第 3 步设计的代码。 ……与你设计的程序动作。
5. 赢得老板的赞赏和莫大的感激。

创建iRock的网页

iRock的网页大概是你看过最简单的HTML网页了。打开你最爱用的编辑器，键入下列HTML标签并保存为*iRock.html*。宠物石的图像可至Head First Labs网站下载：*http://www.headfirstlabs.com/books/hfjs/*。

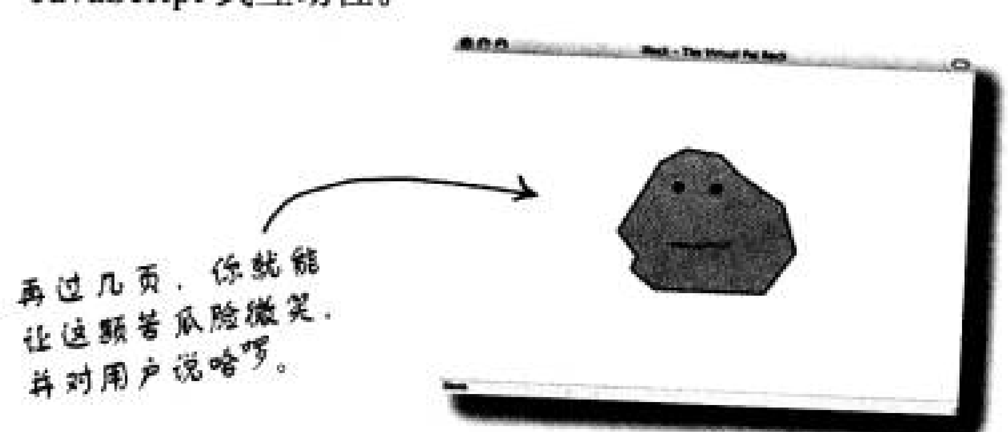
宠物石的HTML网页……简直比石头还没反应，难怪老板需要你的帮忙。



```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" />
    </div>
  </body>
</html>
```


小试身手

开始任何工作前，先保存并在浏览器上测试一下你的 iRock.html。检查一下网页外观是否与这一页的例图相似，接下来我们就要加入交互性元素啰－JavaScript 式互动性。



没有蠢问题

问：<div> 标签中的东西是 CSS 吗？

答：没错。你真内行。

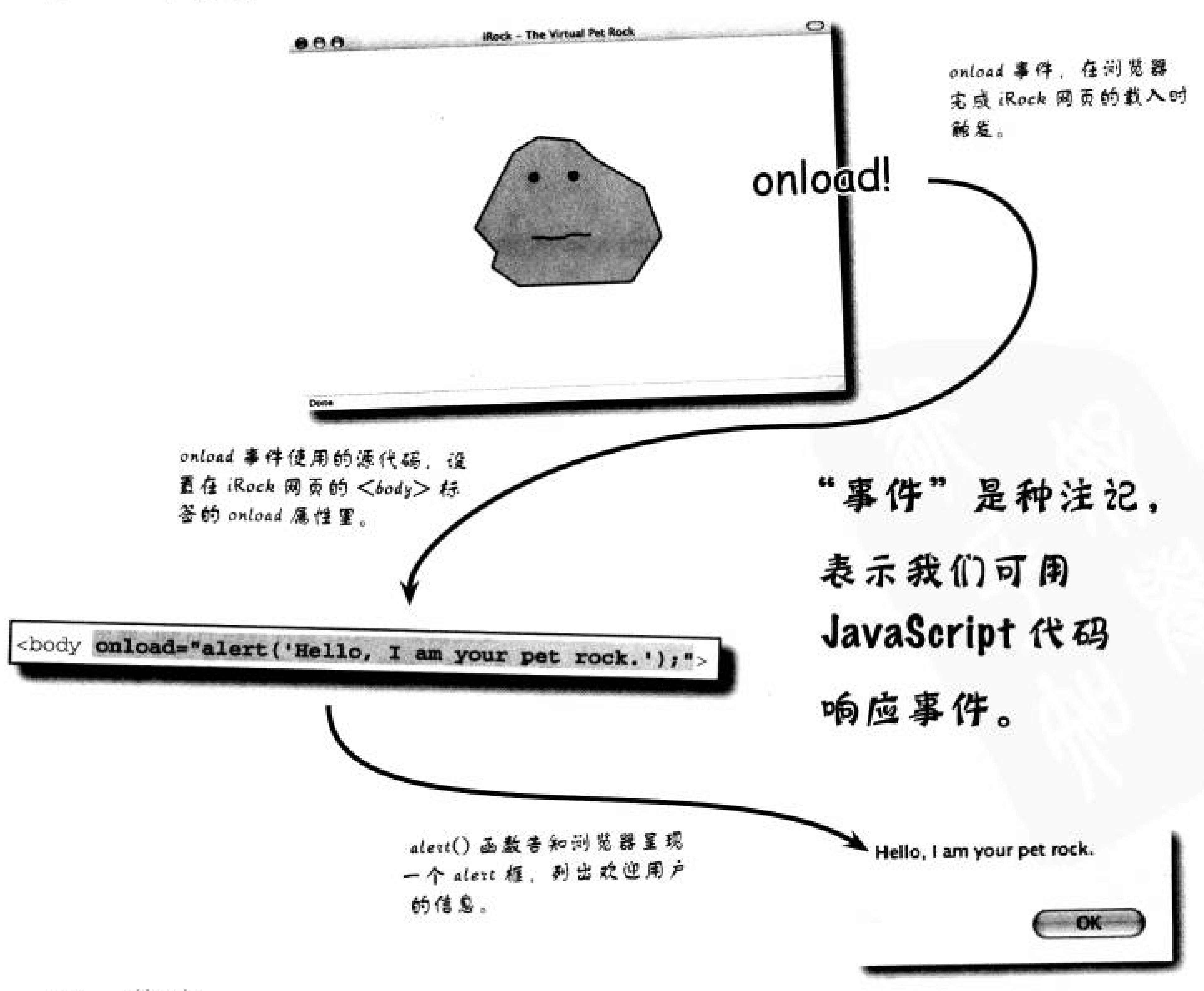
问：我还以为“CSS 直接放在 HTML 网页中”是个坏习惯，你为什么提供反面教材？

答：哎呀，想必你已熟读《Head First HTML with CSS》了吧？你说的没错，通常最好把 CSS 放在 <head> 区域的 <style> 标签里，或以外部样式表储存。不过，你的老板其实不太熟悉这些规则；另外，也能让第一个范例简单一点。话说回来，如果你想自己设计独立的外部样式表，我们绝对举双手赞成。

JavaScript 的事件：为 iRock 加上回答

想在初次载入网页时，用 JavaScript 欢迎用户，我们需要解决两项主要的 JavaScript 相关问题：网页何时下载完毕以及如何呈现用户能看到的欢迎方式。

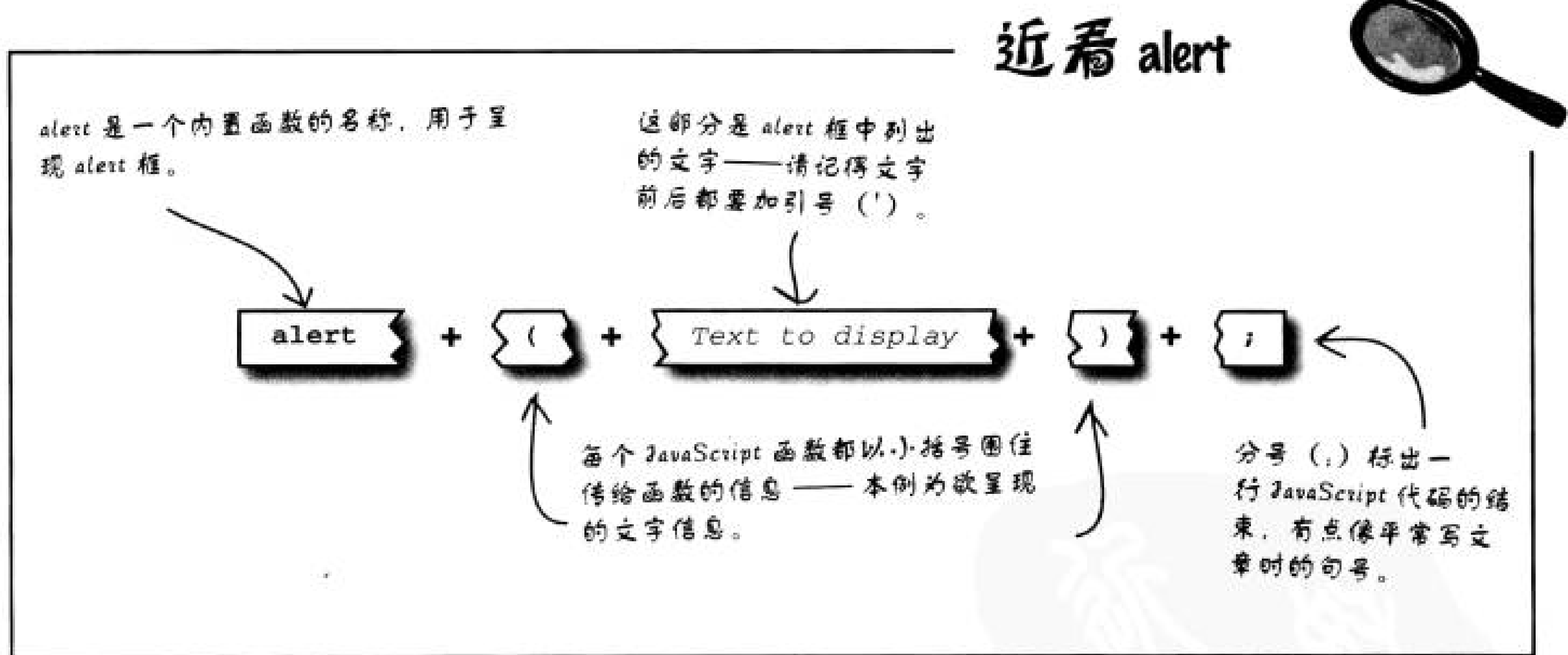
第一项问题关系到响应一个事件（本例为网页的载入），第二项问题则与使用JavaScript的内置功能 —— 消息提示框（alert box）有关。**事件**（event），其实是 JavaScript 在通知我们“有些值得注意的事情发生了”，例如网页的载入（onload）或按钮的点击（onclick）。你可以自己设计响应事件的 JavaScript 代码。



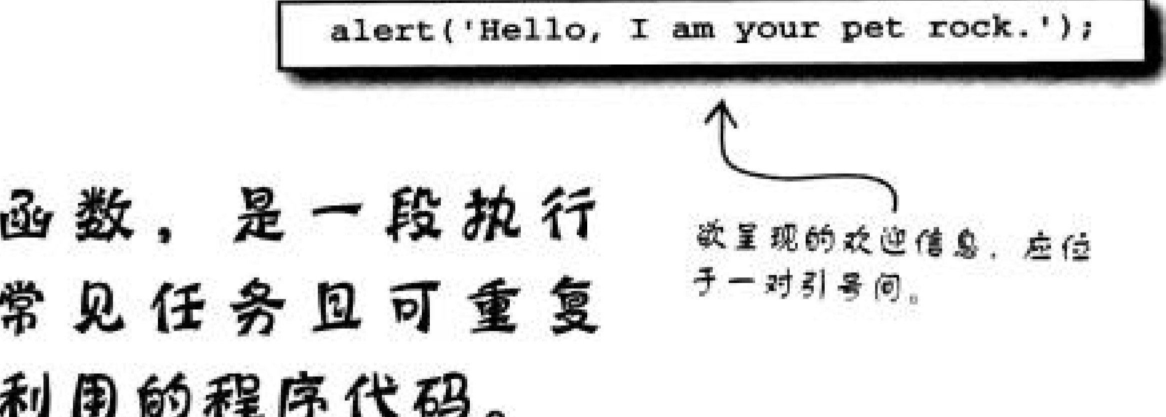
“事件”是种注记，表示我们可用 JavaScript 代码响应事件。

以函数做 alert

JavaScript 的 alert是一种弹出式的窗口（或称对话框），可对用户呈现信息。想呈现 alert 框，程序代码需调用JavaScript的 `alert()` 函数，且一并传送准备给用户看到的文字。**函数**（function），是一段执行常见任务且可重复利用的 JavaScript 代码，如在对话框中呈现信息就是个常见的函数任务。



结合前述内容，一行完整的 JavaScript 代码就这么变出来了。这行源代码将调用某个函数，并于 alert 框中呈现欢迎信息：



函数，是一段执行常见任务且可重复利用的程序代码。

别被事件搞昏头了。

如果你觉得我们刚讲到的“事件”似乎很煞风景，先别烦恼，随着本书的进行，事件的全貌也将越发明朗（而且更显合理）。

为 iRock 添加欢迎信息

想问候载入 iRock 网页的用户，我们需要添加事件处理器（eventhandler，或称事件处理程序）onload，还要透过 JavaScript 的alert()函数列出欢迎信息。请把下列这行加入 *iRock.html*：

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
  </head>

  <body onload="alert('Hello, I am your pet rock.');">
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" />
    </div>
  </body>
</html>
```

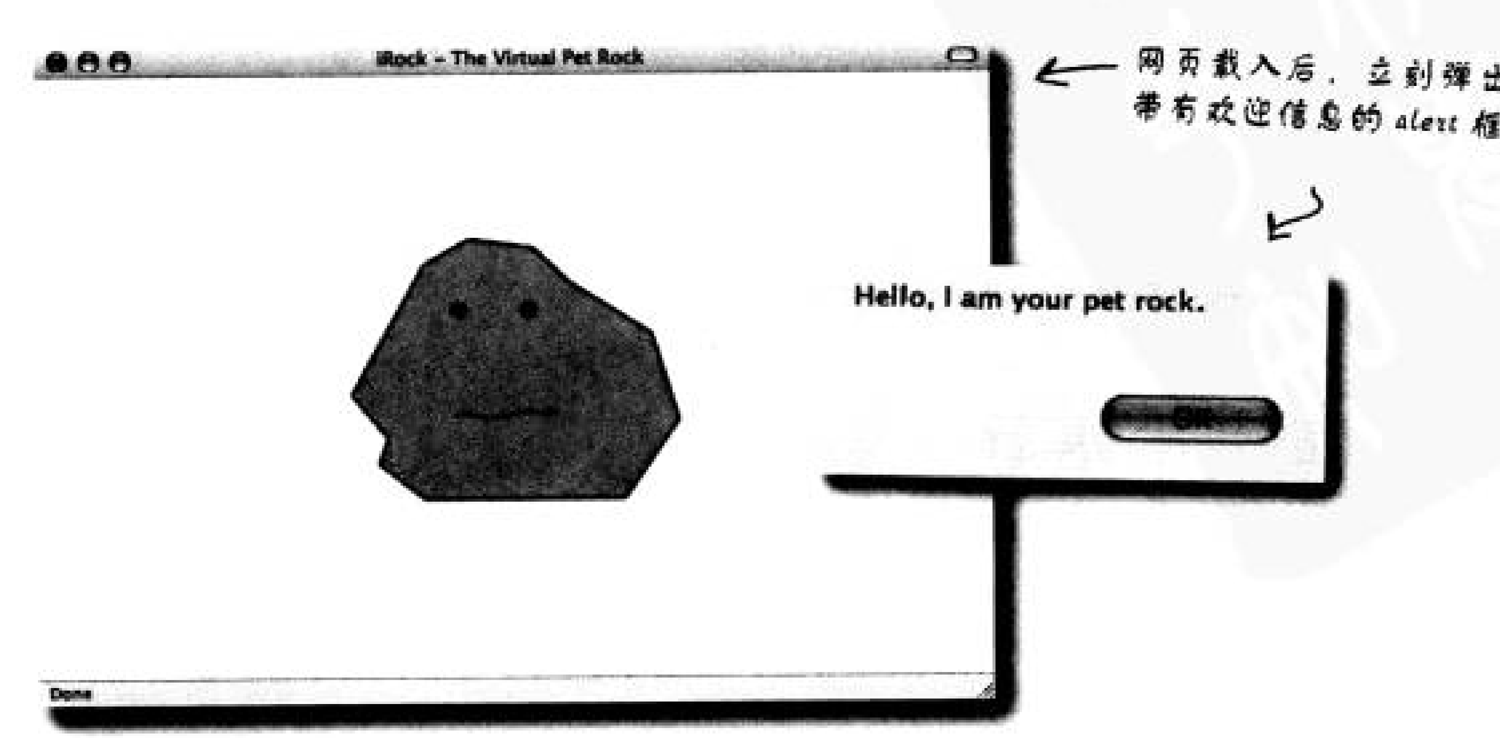
虽然 onload 事件将应用到整个网页上，但我们只需把它设为 <body> 标签的属性。因为对浏览器而言，<body>（网页主体）才是可见的内容。

请记得，你的 JavaScript 就位于网页里。网络浏览器知道如何处理 HTML 与 CSS，也知道如何处理 JavaScript。

irock.html

试玩加上交互性的宠物石

iRock现在比较有点交互性了，托了alert框信息的福（呈现alert框，乃是onload事件的响应）。请用你的浏览器打开*iRock.html*，看看有什么事情发生：



网页载入后，立刻弹出一个带有欢迎信息的 alert 框。

问："事件"从何处来？

答：虽然事件是由用户触发，但最终还是来自浏览器。就像"按（某个）键"，它是个被用户触发的事件，但浏览器必须包装关于事件的信息（例如哪个键被按下），再传递给负责响应这个事件的函数。

问：事件如果没有相应的源代码，会发生什么事吗？

答：如果杳无人烟的密林中有株大树倒了，你会听到它倒地的声响吗？事件也是如此。如果没有关于事件的响应，浏览器就会自己做自己的事，没有人听得到事件发生的信号。换句话说，不论 onload 是否有相关响应，都与网页的载入没有关系。

问：你不是说 JavaScript 代码应该放在 <script> 标签里吗？

答：的确如此。不过它们也能直接放在事件处理器中，一如前例的 onload事件。还有，当你只需要运行一行JavaScript代码时，像iRock这种直接嵌入HTML的方式比较简单。

问：还有其他类似 alert() 的内置函数吗？

答：当然，而且有很多呢！alert()只是一个 JavaScript 内置的可重复利用代码的例子。这趟 JavaScript 的旅程上，我们还会谈到很多标准（内置）函数。等各位看完本书，你甚至还可以自定义函数咧！

问：为什么 iRock 的 onload 代码要混用单双引号？

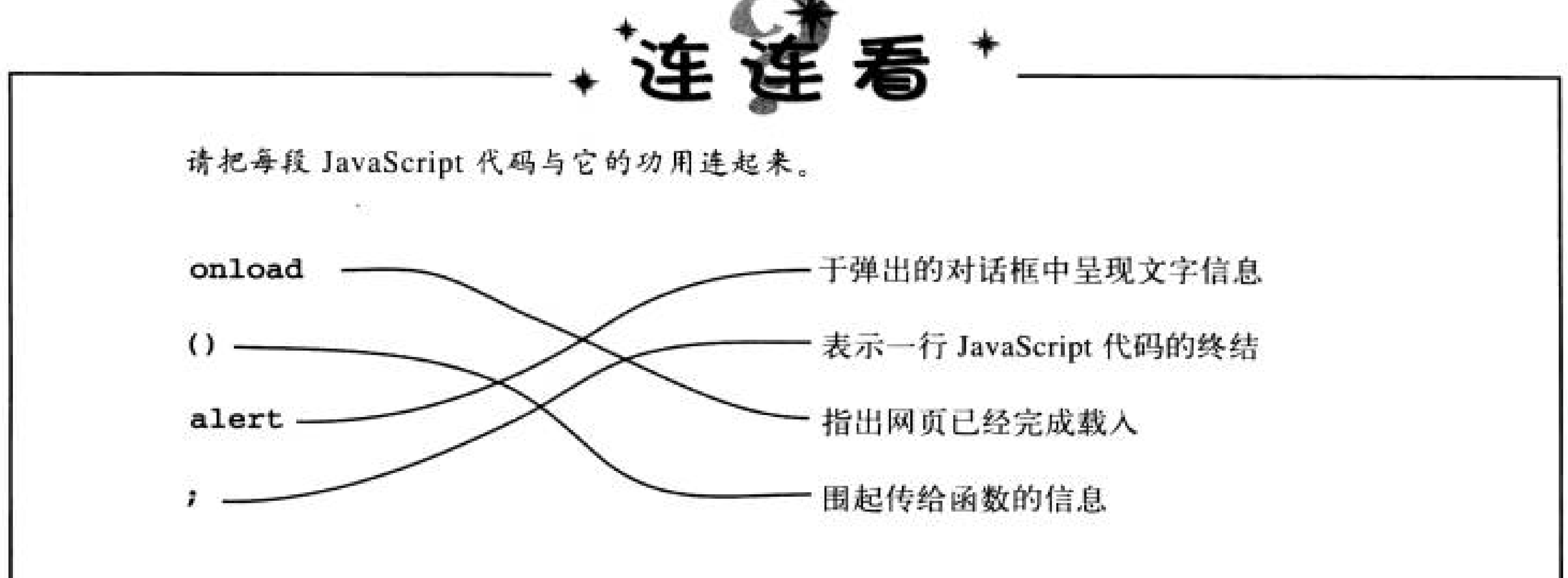
答：HTML与JavaScript都需要我们先标示一段文字的结束，才能开始下一段……除非使用不同的分界字符（delimiter，如单、双引号）。所以说，遇到JavaScript出现在HTML属性内的情况时（文字中又有文字），我们只好混合单双引号的使用，以回避标示文字结束的问题。无论你对HTML属性与JavaScript文字各采用何种分界字符，一旦选好了，就要从一而终。或许实际举个例子比较清楚 —— iRock的用法是："The user clicked and said, 'Hello, there.'"

复习要点

- 事件以 JavaScript 代码响应网页里发生的动静。
- onload 事件在网页载入完毕后触发。
- 设定 <body> 标签的 onload属性，即可对 onload 事件有所响应。
- 函数把 JavaScript 打包为可重复利用的模块。
- 有些函数需要我们把信息传过去，才能完成任务。
- alert()是个JavaScript内置函数，负责在弹出的对话框中呈现文字信息。

请把每段 JavaScript 代码与它的功用连起来。

onload	于弹出的对话框中呈现文字信息
()	表示一行 JavaScript 代码的终结
alert	指出网页已经完成载入
;	围起传给函数的信息



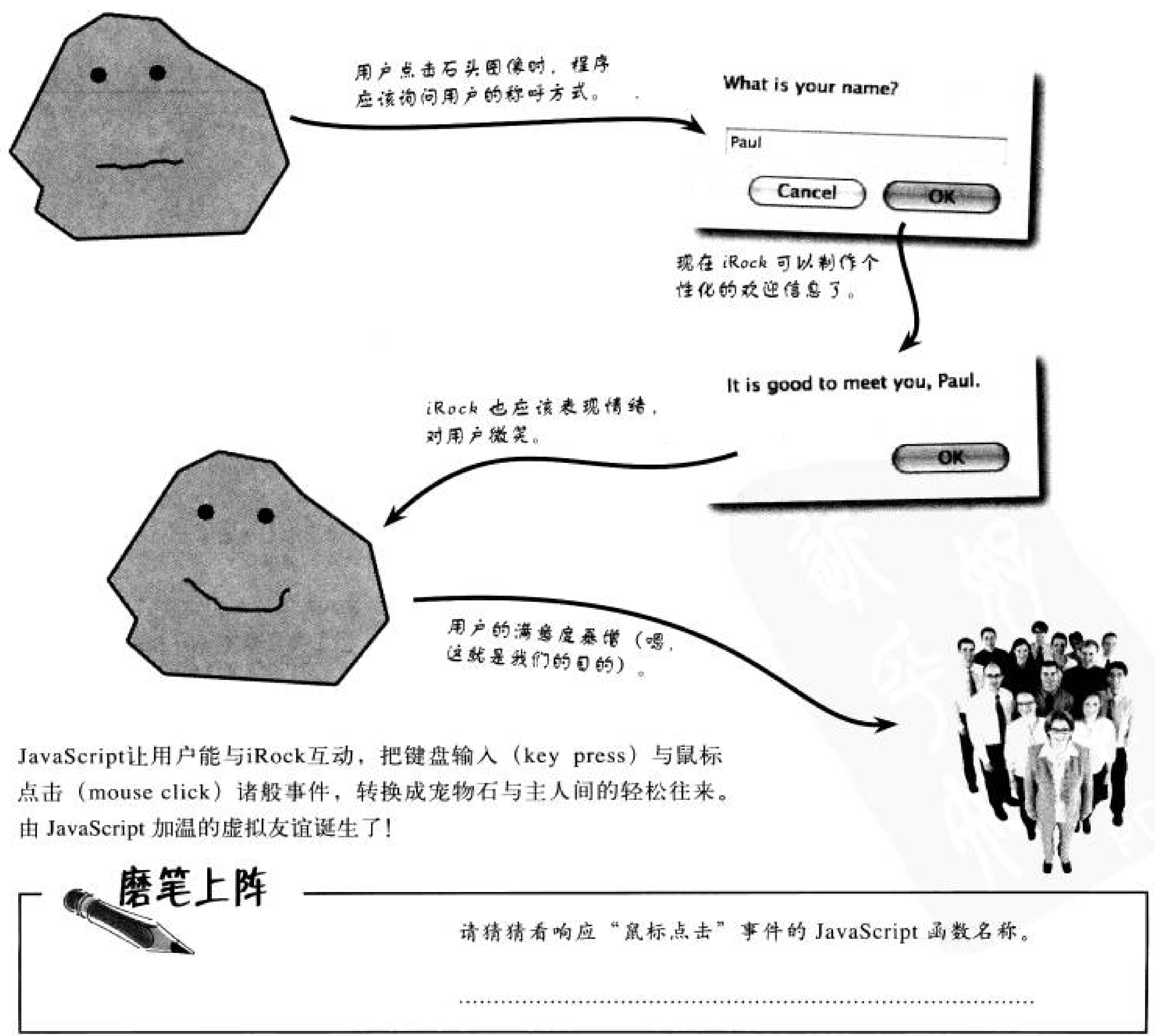
让 iRock 真正动起来

在设计交互性 iRock 的任务上，你已经有些小进展了，不过，想靠这么点东西赢取用户的欢心……你还记得那些待办事项吗？

1. ~~创建 iRock 的 HTML 网页。~~ ← 完成了！
2. ~~加入一个 JavaScript 的 alert框，使 iRock 网页完全载入后，~~宠物石会欢迎用户。 ← 这项也做完了！
3. 设计 JavaScript 代码，询问用户姓名，并呈现个性化的欢迎信息，还要让宠物石露出微笑。
4. 加入一个事件处理器，于用户点击宠物石时，运行我们在第 3 步设计的代码。
5. 赢得老板的赞赏和莫大的感激。

交互性是种双向沟通

到目前为止，我们的宠物石会打招呼，但用户没办法回答什么。其实，我们希望宠物石能响应用户的行为。利用一点 JavaScript 的小帮忙，改变 iRock 的表情，并在欢迎信息中加入用户的昵称……它就能变成非常活跃又友善、让人爱不释手的宠物石。



JavaScript让用户能与iRock互动，把键盘输入（key press）与鼠标点击（mouse click）诸般事件，转换成宠物石与主人间的轻松往来。由 JavaScript 加温的虚拟友谊诞生了！

磨笔上阵

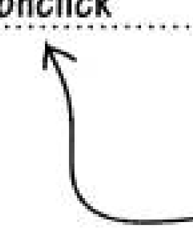
请猜猜看响应“鼠标点击”事件的 JavaScript 函数名称。

....

磨笔上阵 解答

请猜猜看响应“鼠标点击”事件的 JavaScript 函数名称。

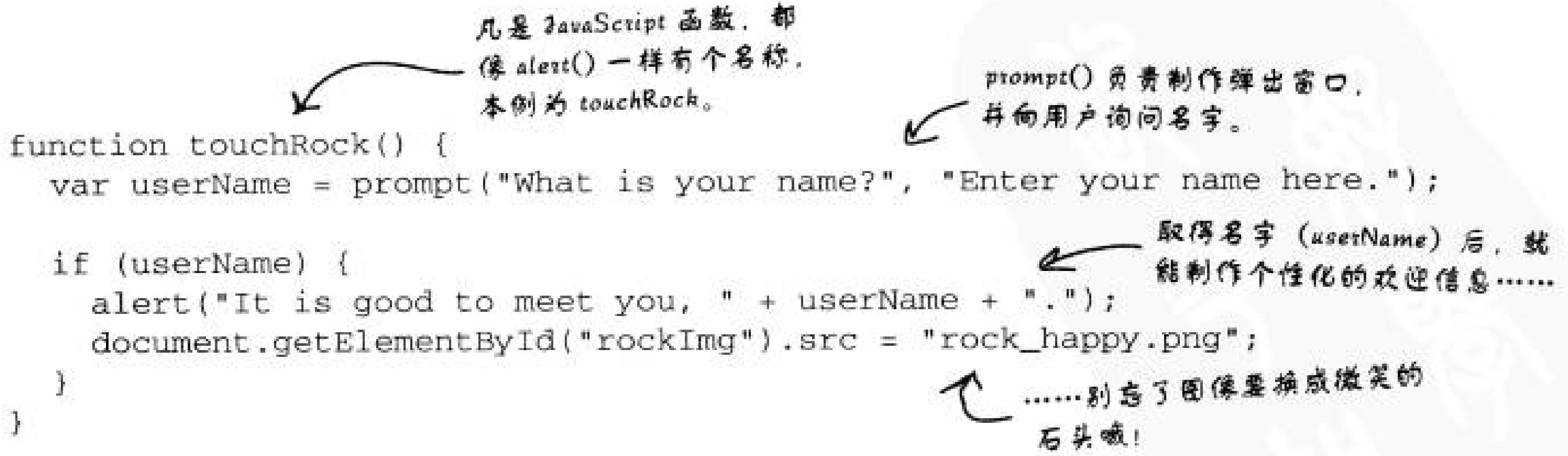
onclick



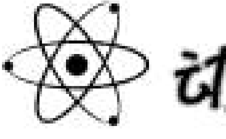
添加取得用户名字的函数

JavaScript 函数已经烹调完毕准备上桌。每次看到“JavaScript 上菜”的图示，表示各位只要一字不漏地照抄这部分代码就好了。请相信我们，你很快就会学到关于这些程序码的一切，而且连自己写函数都能上手。

下列程序代码是一个命名为 touchRock() 的自定义函数，用于敦促用户输入名字，接着利用 alert 框列出个性化的欢迎信息。这个函数也会把图像改成微笑的石头。为 iRock 加上个性化功能的方式就这么简单。




```
function touchRock() {
  var userName = prompt("What is your name?", "Enter your name here.");

  if (userName) {
    alert("It is good to meet you, " + userName + ".");
    document.getElementById("rockImg").src = "rock_happy.png";
  }
}
```



动动脑

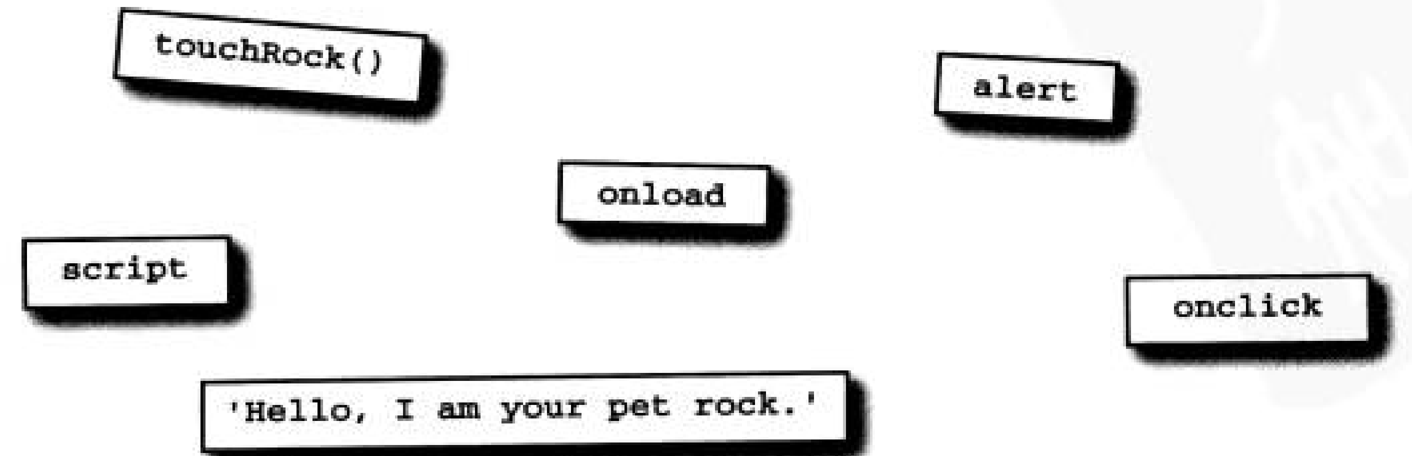
你知道这个函数应该放在 *irock.html* 的哪个位置吗？

JavaScript 冰箱磁铁

iRock 网页的原始程序少了几段关键代码。你能帮这个网页填入失落的片段吗？

提示：不确定答案吗？把你的答案填入你的 irock.html，试一试便知道！

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>

    < ................ type="text/javascript">
      function touchRock() {
        var userName = prompt("What is your name?", "Enter your name here.");

        if (userName) {
          alert("It is good to meet you, " + userName + ".");
          document.getElementById("rockImg").src = "rock_happy.png";
        }
      }
    </script>
  </head>

  <body .............. =" ..............( ..................................................... );">
    <div style="margin-top:100px; text-align:center">

      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"
        ..............=" .......................... ;" />
    </div>
  </body>
</html>
```



JavaScript 冰箱磁铁解答

iRock 网页的原始程序少了几段关键代码。你能帮这个网页填入失落的片段吗？

JavaScript 函数位于特殊的 <script> 标签里，这个标签又位于网页的 <head> 区域里。

<script> 标签的 type 属性，用于分辨使用的脚本语言，本例为 JavaScript。

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>

    <script type="text/javascript">
      function touchRock() {
        var userName = prompt("What is your name?", "Enter your name here.");

        if (userName) {
          alert("It is good to meet you, " + userName + ".");
          document.getElementById("rockImg").src = "rock_happy.png";
        }
      }
    </script>
  </head>

  <body onload="alert('Hello, I am your pet rock.');">
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"
      onclick="touchRock()" />
    </div>
  </body>
</html>
```

<body> 标签的 onload 事件属性，联系网页与 alert 框。

把原图改为微笑的石头。

石头原图的 onclick 事件属性，使得 touchRock() 函数于点击图像时受到调用。

当鼠标移过图像上方时，改变鼠标光标为手掌型。

即时重播：刚才发生了什么事？

一点点 JavaScript 就引发了许多改变，塑造出亲切可爱的全新 iRock。我们一起来回顾一下所有改变，以及它们影响网页的方式。

iRock – The Virtual Pet Rock

onclick!

点击 iRock 的图像，让事件触发自定义的 JavaScript 函数。

```
<img id="rockImg" src="rock.png" alt="iRock"
  style="cursor:pointer" onclick="touchRock();"
/>
```

```
function touchRock() {
  var userName = prompt("What is your name?",
                        "Enter your name here.");
  if (userName) {
    alert("It is good to meet you, " + userName + ".");
    document.getElementById("rockImg").src = "rock_happy.png";
  }
}
```

函数询问用户的名字，根据名字制作个性化的欢迎信息。

What is your name?

Paul

Cancel OK

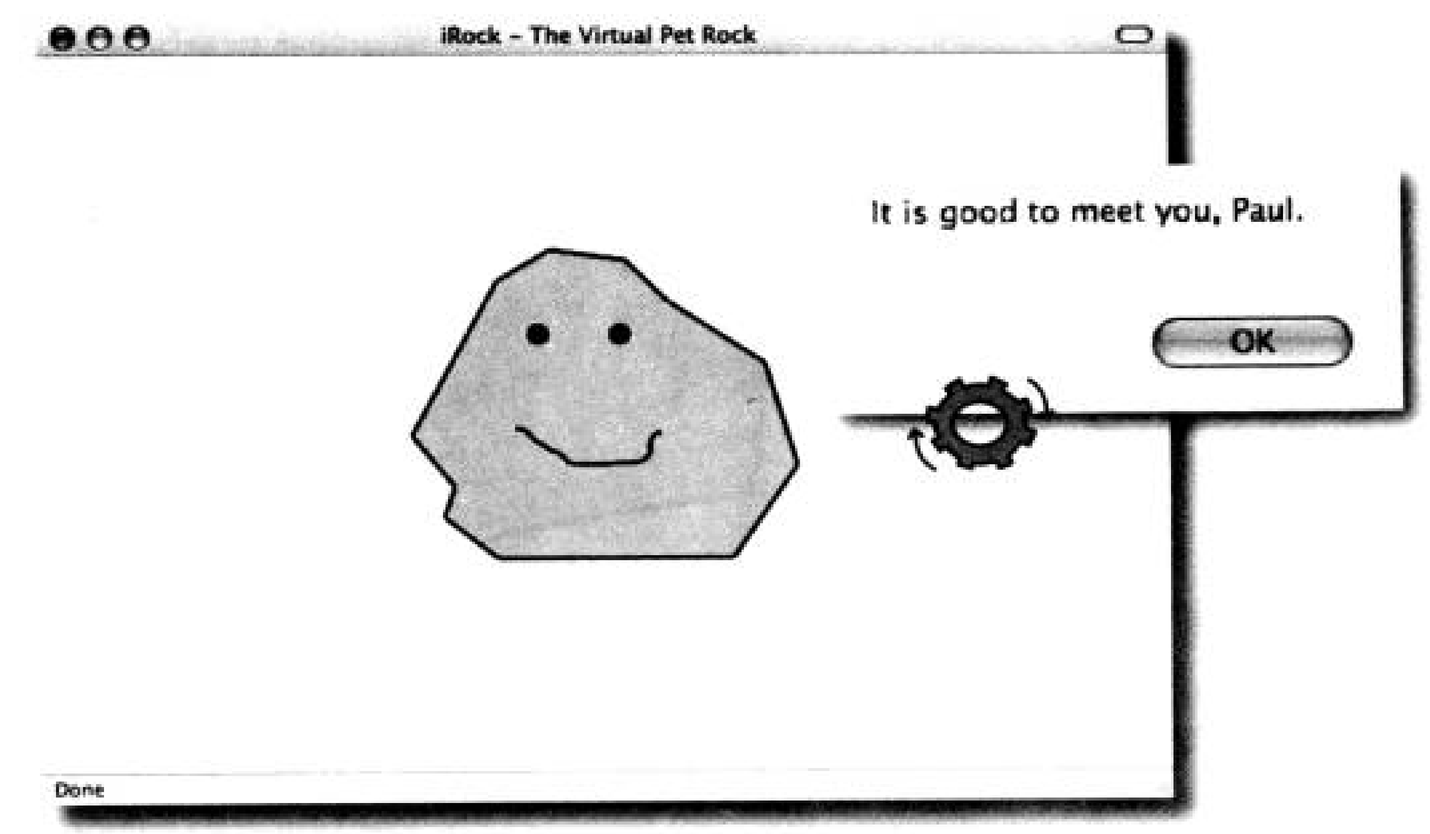
原本的图像换成微笑的石头。

It is good to meet you, Paul.

OK

试玩 iRock 1.0

大家务必自制 irock.html（参考26页），并记得到 Head First Labs（*http://www.headfirstlabs.com/books/hfjs/*）下载石头图像。然后，打开浏览器，试玩一下你的宠物石吧！

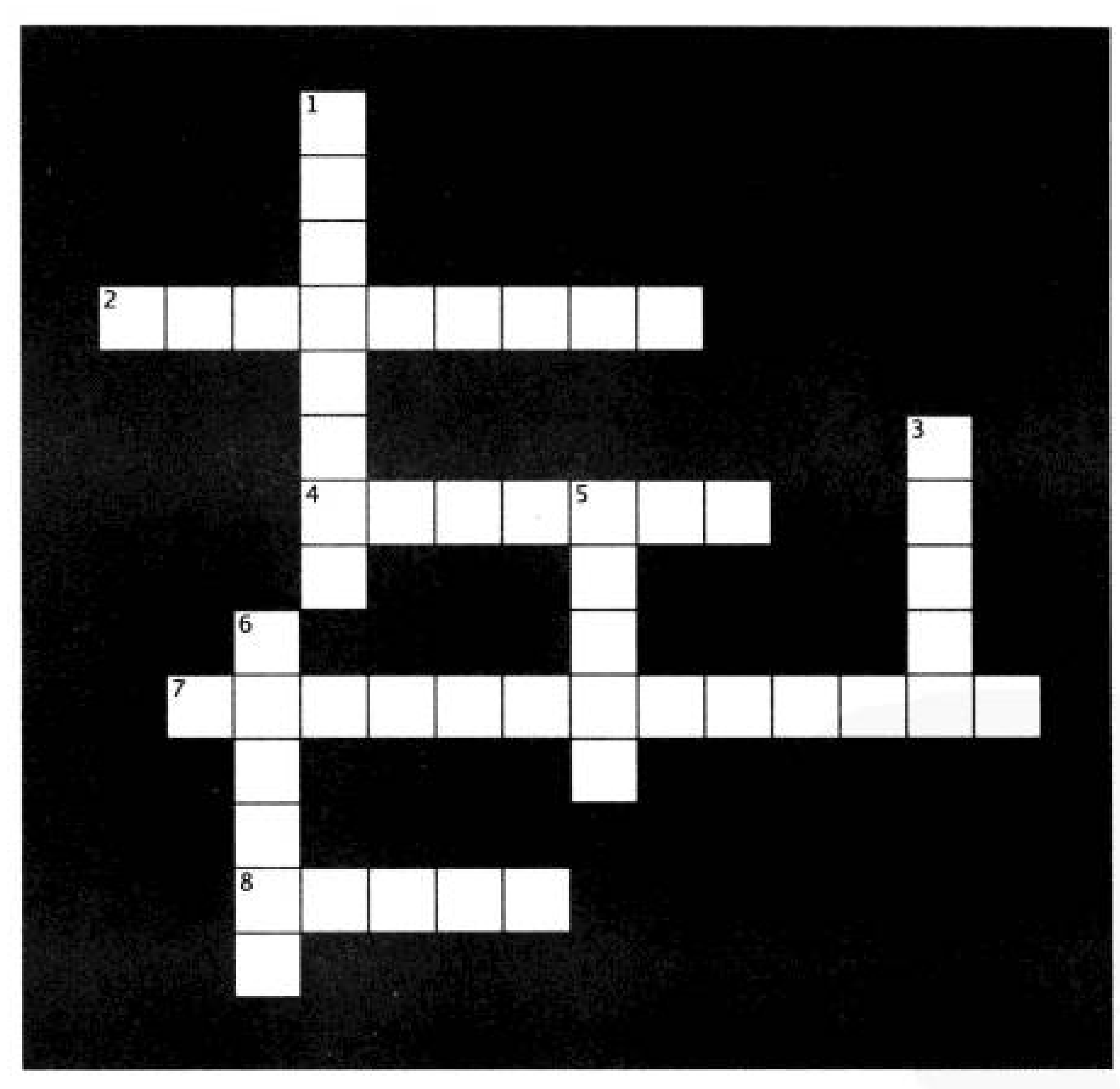


1. ~~创建 iRock 的 HTML 网页。~~ ← 完成了！
2. ~~加入一个 JavaScript 的 alert 框，使iRock 网页完全载入后，宠物石会欢迎用户。~~ ← 这项也做完了。
3. ~~设计 JavaScript 代码，询问用户姓名，并呈现个性化的欢迎信息，还要让宠物石露出微笑。~~ ← touchRock() 就是用在这里。
4. ~~加入一个事件处理器，于用户点击宠物石时，运行我们在第 3 步设计的代码。~~ ← onclick 事件用在这里。
5. 赢得老板的赞赏和莫大的感激。 ← 老板大人很满意……你的升迁与新的办公室是否就在眼前呢？

JavaScript 为网页加上行动能力，不再限于看图说故事。

JavaScript 填字游戏

坐下来让你的左脑也运动一下。下面是个典型的填字游戏，所有解答的词汇都曾在本章出现过。



横向提示

2. 有段代码为 iRock 提供个性化欢迎信息，它的名称是______。

4. 想响应鼠标点击的动作时，只需在 HTML 组件的 ______ 属性里设置一些 JavaScript 代码。

7. 若缺少这样特色，你干脆只用 HTML 与 CSS 就好了。

8. 想对用户呈现一段文字，请调用 ______ 函数。

纵向提示

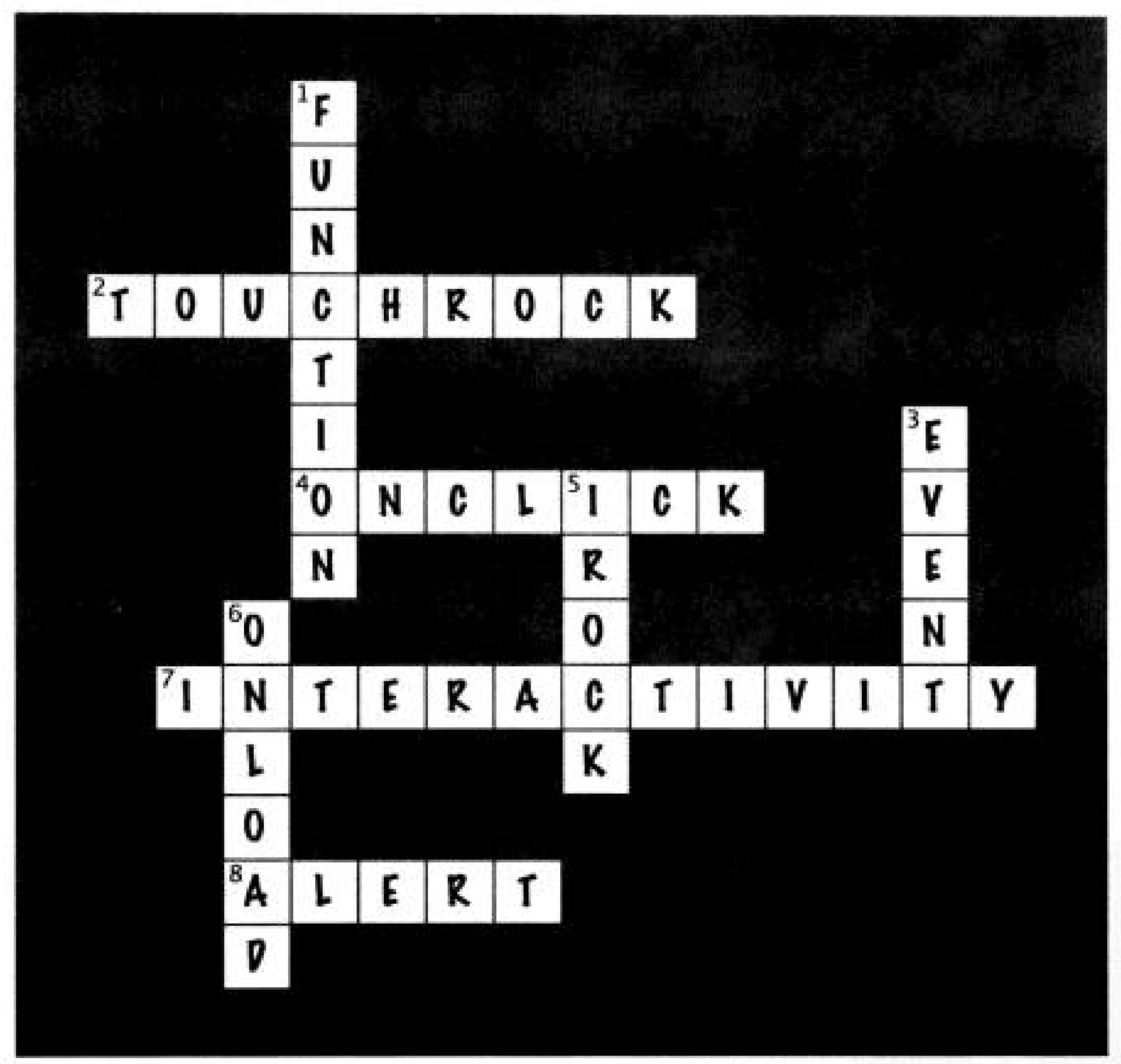
1. 一段可重复利用、执行常见任务的 JavaScript 代码，称为 ______。

3. 有 ______ 发生，而浏览器希望让我们知道。

5. “The feel good online toy of the season.”

6. ______ 让我们知道网页已载入完毕。

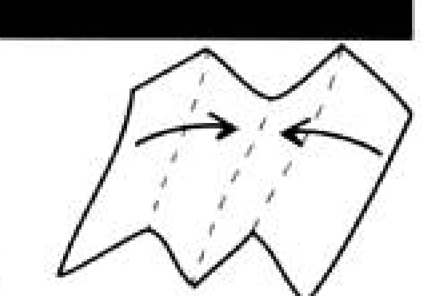
JavaScript 填字游戏解答



Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

JavaScript究竟为网页添加了什么东西？



这是左右脑的秘密会谈！

这世上有冷硬无情的石头……

……也有温暖贴心的石头。

onclick!

不管是哪种石头，都渴望着某件相同事情！

汪！

嘿，事情办得怎么样了？

喵……

现在，iRock与这些真实、非虚拟的宠物有些共通点了。是什么？

上网搜索，大概也找不到关于这个问题的好答案。
你应该在用户身上花点时间。
所有网页都会想知道用户的意见。

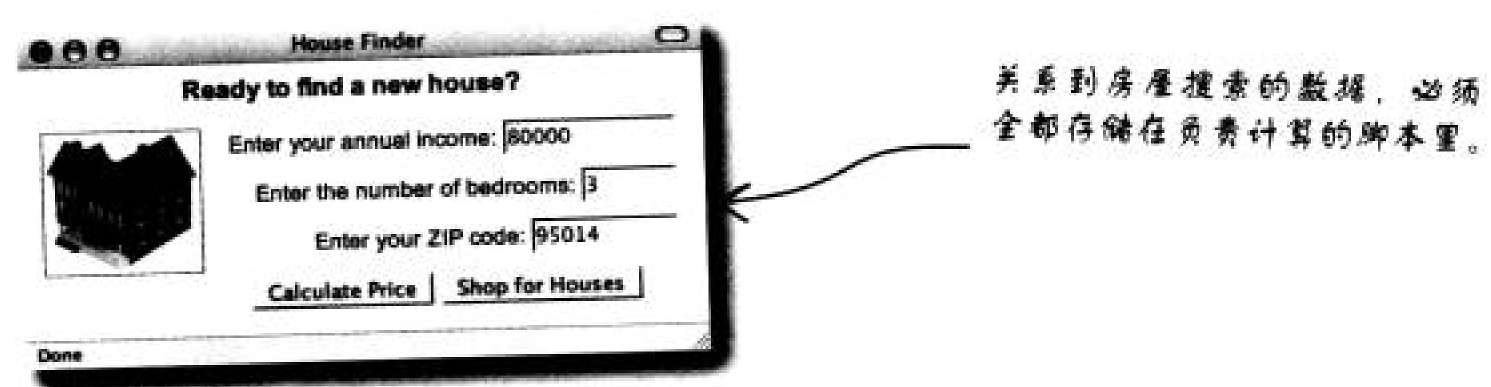
2 存储数据

每项事物都有自己的位置

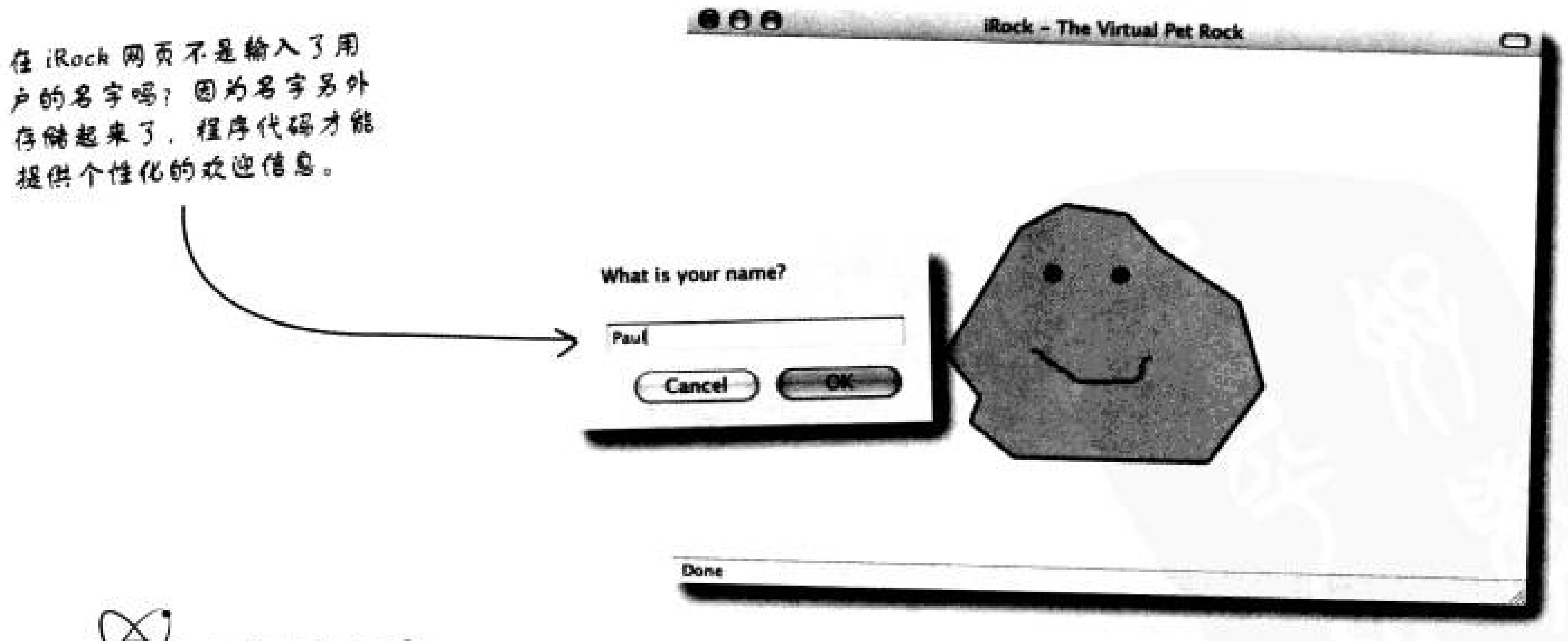
现实生活中，我们经常把“有个地方储存物品”这件事看得太过重要。但在JavaScript中绝非如此。你不会有个足以藏人的更衣室，也不会有可以停放三辆保时捷的车库。在JavaScript的世界里，**每项事物都有自己的位置**，确认众多事物有所依归，则是你的责任。我们讲的事物就是数据——包括如何**呈现数据**、**存储数据**以及**找出数据**。身为JavaScript的存储专家，你将能面对满室杂乱无章的JavaScript数据，并用意志为它们贴上虚拟标签与储藏编号。

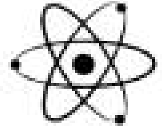
你的脚本也能存储数据

几乎所有脚本都会面对数据，通常是把数据存储于存储器里。住在网络浏览器里的JavaScript解释器，它负责清出一点存储JavaScript数据的空间。不过，你要负责指出**欲存储的数据，还有欲使用数据的方式。**



使用存储起来的数据，脚本才能继续计算，并**记忆**用户的**数据**。如果没有存储数据的能力，我们永远也找不到合适的房屋，也无法深入认识 iRock。



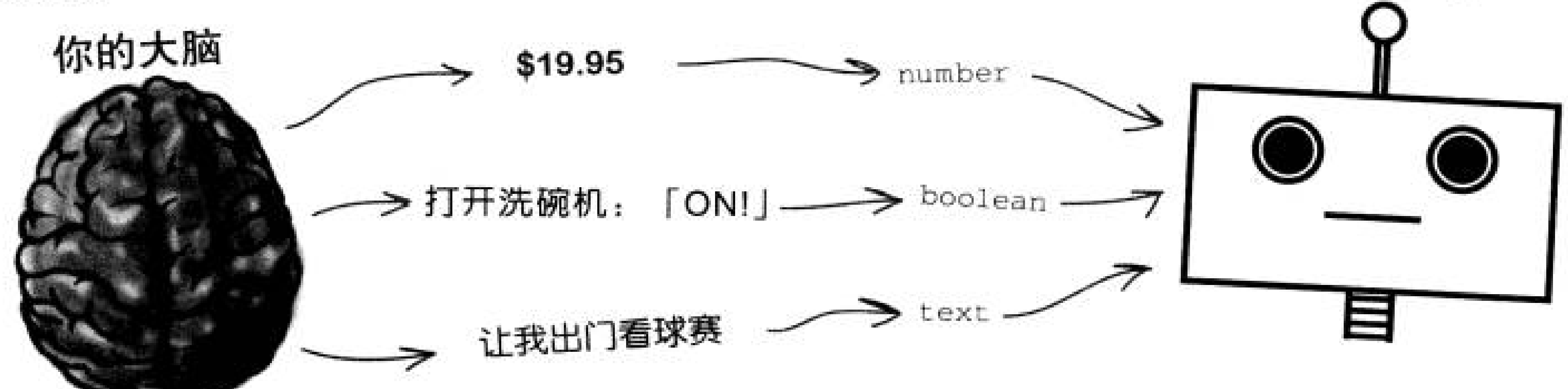


动动脑

想想日常生活中有哪些每天都在处理的信息。这些信息相似吗？有很大的差异吗？你会怎么组织各种不同的信息呢？

脚本满脑子都是数据类型

讲到日常生活的信息，我们都不自觉地把它们依种类区分：姓名、数字、声音……不胜枚举。JavaScript也会把数据依**数据类型**（data type）分类。数据类型是把我们脑中的信息与JavaScript的信息形成对应的重大关键。



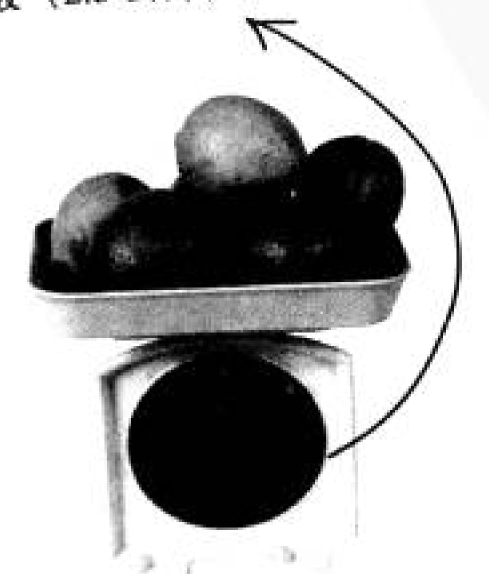
JavaScript使用三种基本数据类型：text、number、boolean。

Text

文本（text）数据只是一串字符，例如你最爱的咖啡品牌名称。文本通常是单一词汇或略长的句子，但不必然如此。**字符串**（string）也常用于称呼文本类的数据，在JavaScript里，文本数据常出现在双引号（" "）或单引号（''）中。

Number

数字（number）用于存储重量、物品数量等数值数据。JavaScript的数字可能是整数（2 公斤）或浮点数（2.5 公斤）。



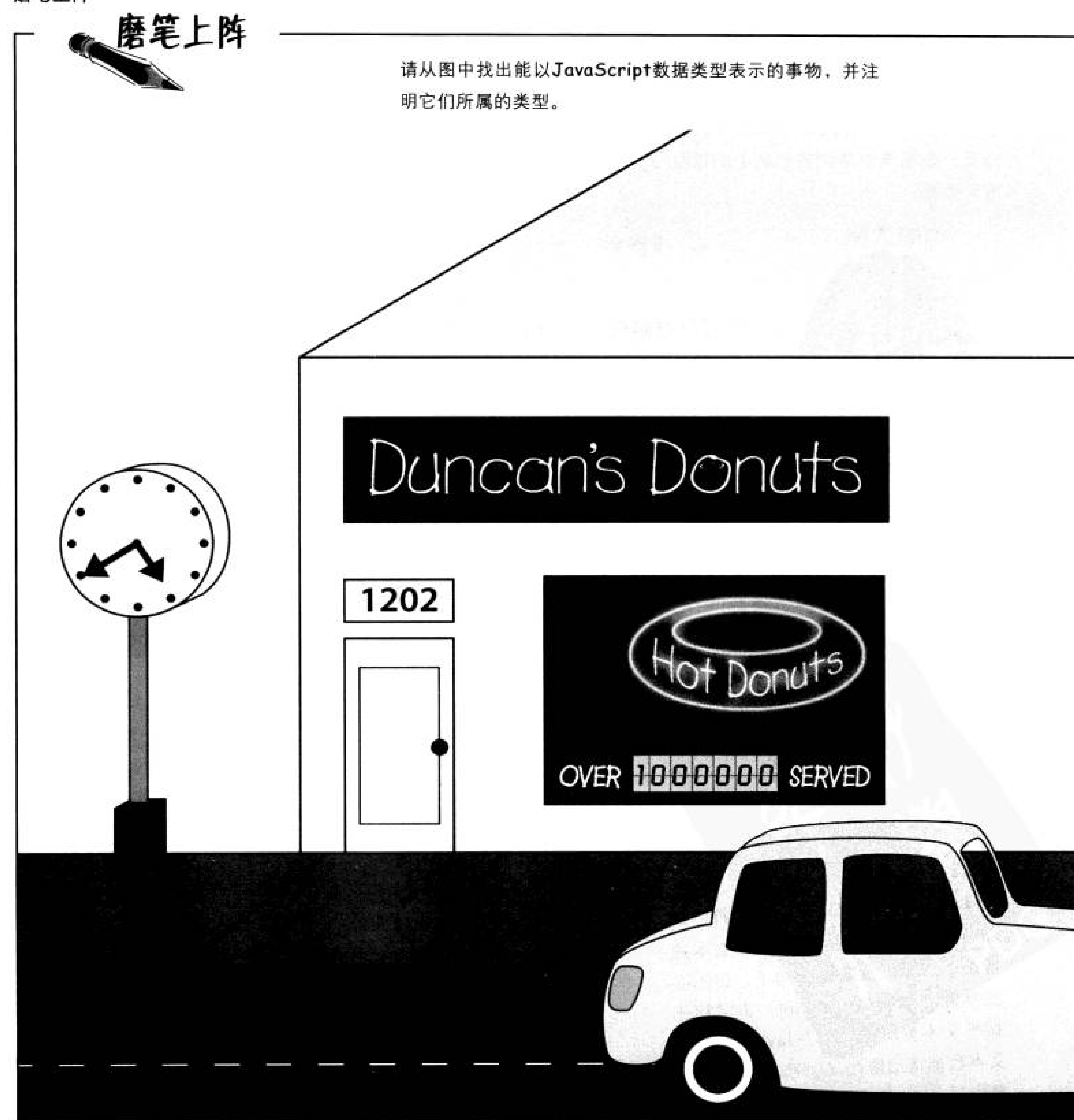
Boolean

布尔（Boolean）只有两种可能情况——**成立**（true）或**不成立**（false）。所以，你能用布尔值表达只有两种可能设定的状况，例如只能转到“开”或“关”的烤面包机。布尔随时随地都出现在我们身边，可协助我们作决定。第4章将更详细地讨论“作决定”这件事。

数据类型**直接影响**JavaScript代码里处理数据的方式。举例来说，alert框只能呈现文本，无法列出数字。所以数字将暗中转换为文本，而后才出现在我们面前。

磨笔上阵

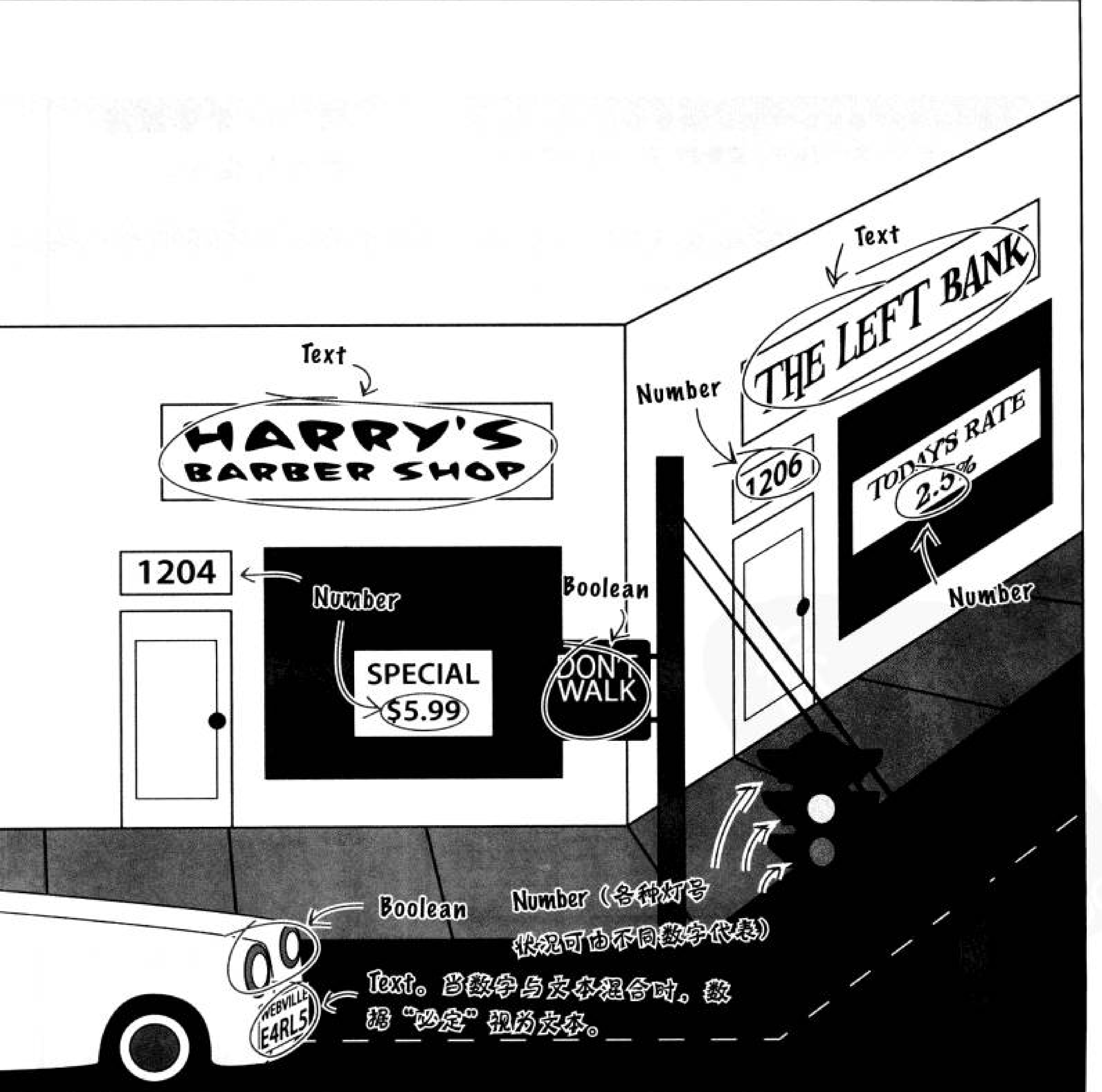
请从图中找出能以JavaScript数据类型表示的事物，并注明它们所属的类型。



HARRY'S
BARBER SHOP
1204
SPECIAL
$5.99
DON'T
WALK
THE LEFT BANK
1206
TODAY'S RATE
2.5%
WEBVILLE
E4RL5

磨笔上阵解答

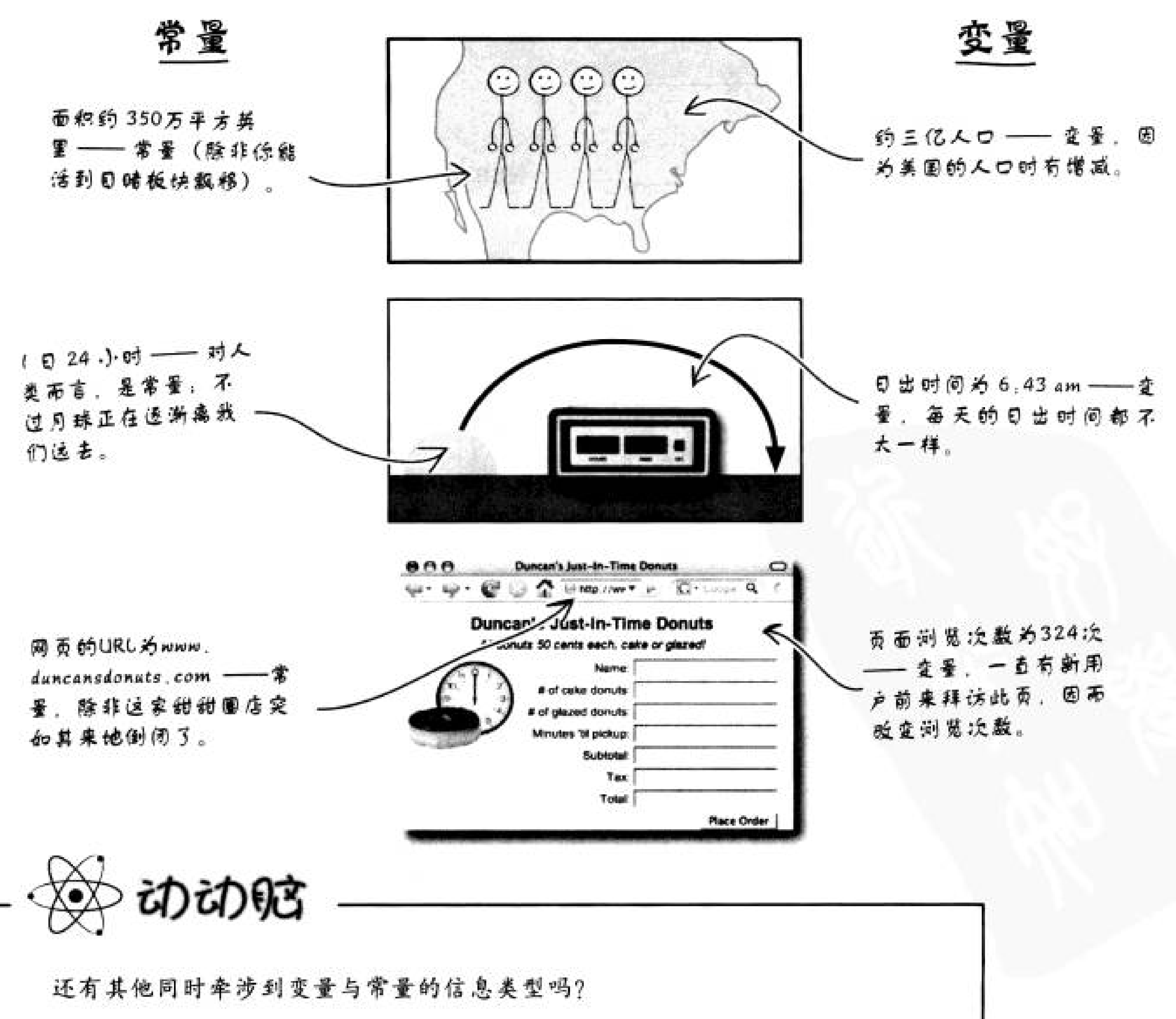
请从图中找出能以JavaScript数据类型表示的事物，并注明它们所属的类型。


Text
THE LEFT BANK
Number
1206
TODAY'S RATE
2.5%
Number
Text
HARRY'S
BARBER SHOP
1204
Number
SPECIAL
$5.99
Boolean
DON'T
WALK
Boolean
Number（各种灯号
状况可由不同数字代表）
Text。当数字与文本混合时，数
据“必定”视为文本。
WEBVILLE
E4RL5

常量总是相同，变量可能有改变

在 JavaScript 里存储数据不只要注意**类型**，还要注意数据的**用途**。你会如何使用数据？说得更仔细点，在脚本执行的过程中，数据会改变吗？你的回答将决定数据类型应设定为**变量**（variable）还是**常量**（constant）。**在脚本的执行过程中，变量会改变，常量的值则永不改变。**

变量数据可以改变——常量数据则是固定的。

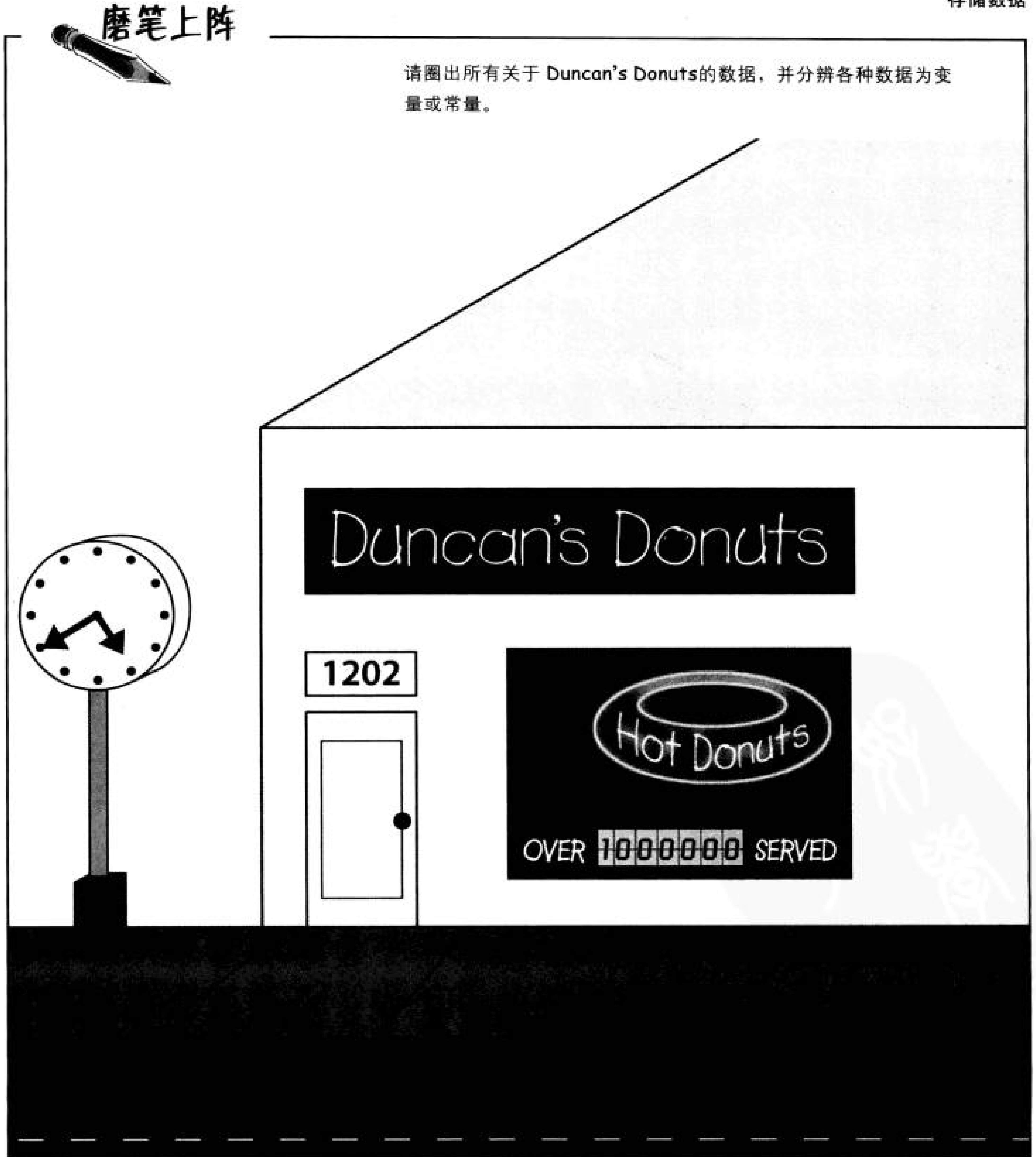


动动脑

还有其他同时牵涉到变量与常量的信息类型吗？

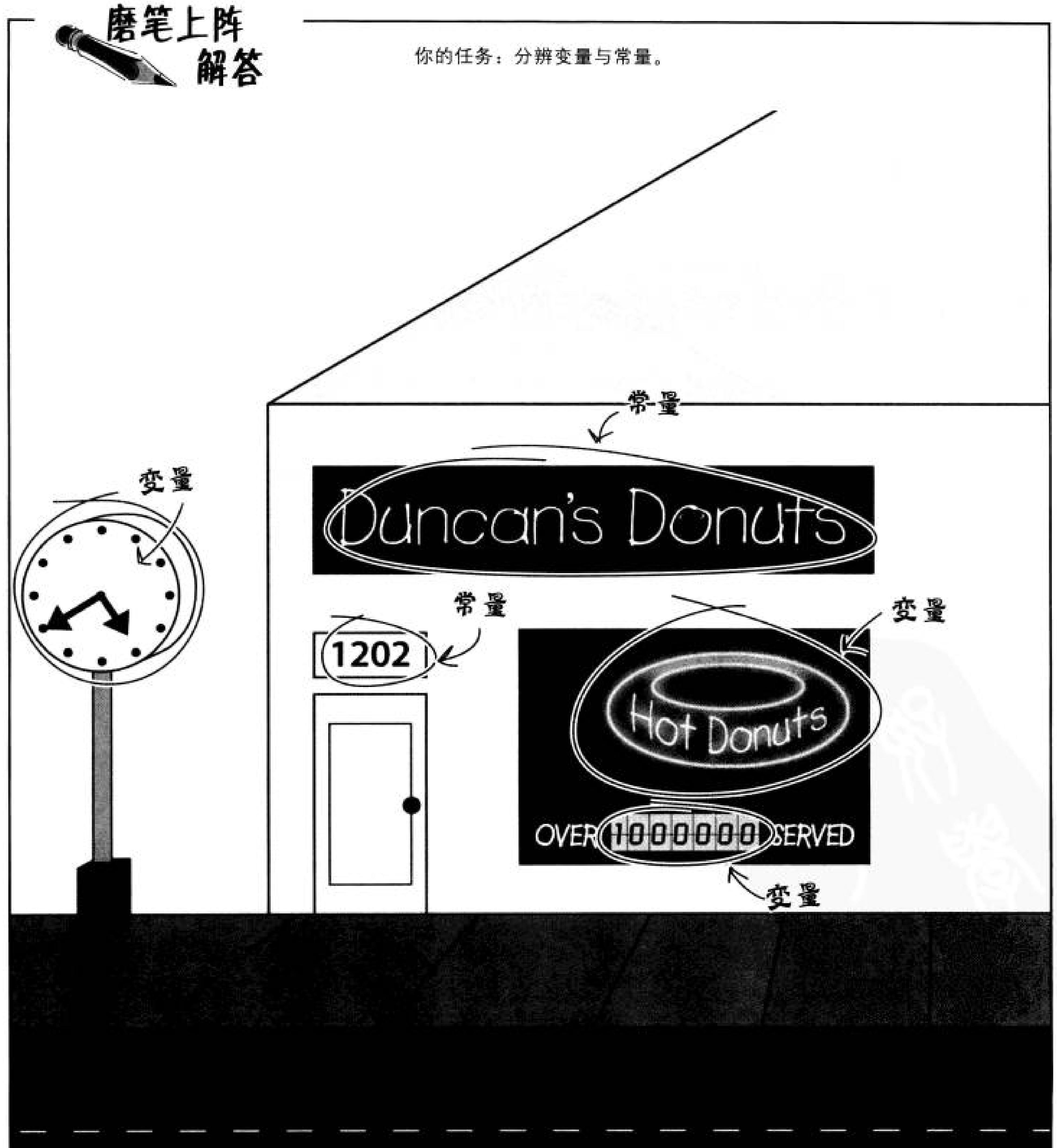
磨笔上阵

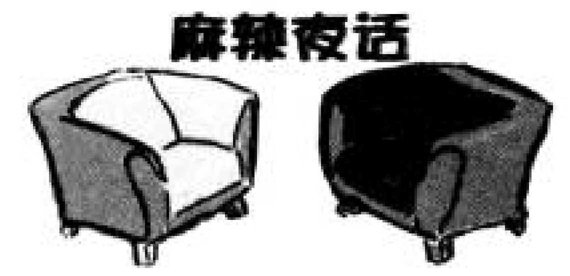
请圈出所有关于 Duncan's Donuts的数据，并分辨各种数据为变量或常量。



磨笔上阵解答

你的任务：分辨变量与常量。





今晚主题：变量与常量在数据存储领域的竞争。

变量：

讲到存储数据，我提供了最大的灵活性。你可以随意修改我的值。我可能现在是一个值，稍后又变成另一个值——这就是我所谓的自由啊！

常量：

我看是“墙头草，两边倒”吧！照我的意思，应该从一而终。就是坚决的一致性才使得我如此珍贵……脚本设计师很热爱数据在过程中的可预测性呢。

变量：

是是是……不过在数据必须与时并进的状况下，你那颗不知变通的水泥脑袋完全不适用嘛！像是火箭升空前要从十开始倒数，你来处理看看啊！

常量：

哟，你的意思是说只有“您”才是存储关键数据的选择啰？大错特错！你以为火箭要怎么运上发射台？那是因为有个聪明人懂得把发射日期设为常量。给我一个与时并进的截止日期，我就给你一直比预定行程慢一拍的项目。

变量：

好啦！随便啦！怎么有人能如此得意洋洋地把变异性视为邪恶呢？你不知道吗？改变也可能是好事，尤其是必须存储用户输入的数据、执行计算的时候。

常量：

事物越常改变、也越常维持不变。说真的，一开始为什么要有改变？选择对的值，然后就不要动它了。你想想，无论蓄意与否，某个值永远不会改变，知道有这么回事不是很舒畅吗？

变量：

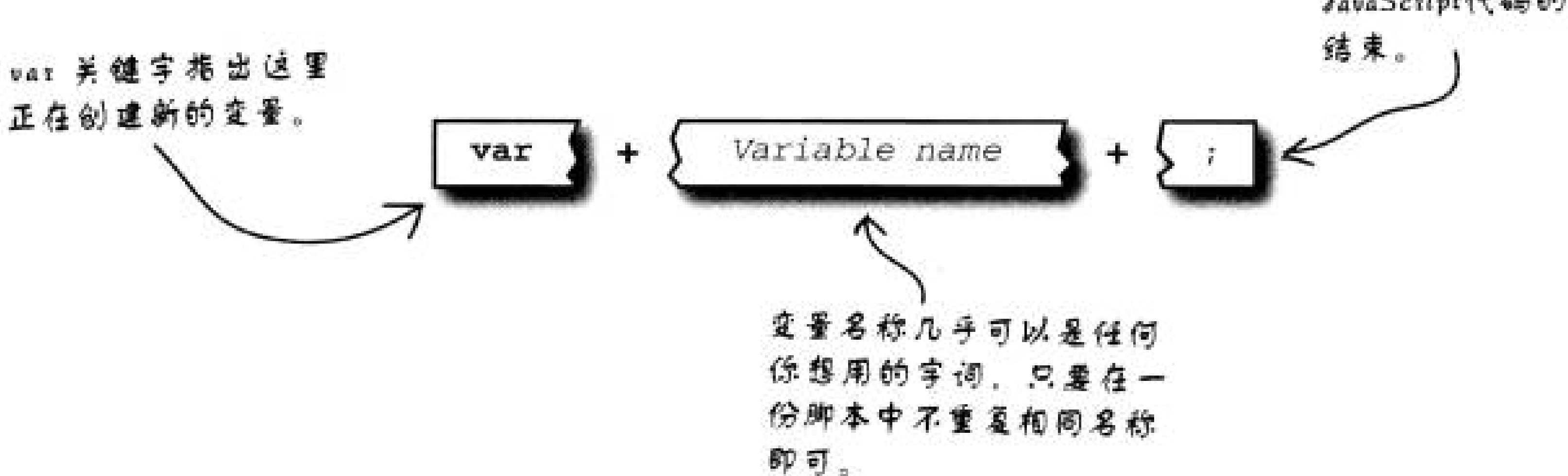
……我看我们同意彼此间歧见很深就好了。

常量：

事实上，我从头到尾都跟你意见不合。

变量刚开始没有值

变量，是存储器里附有独**一无二名称**（unique name）的**存储位置**（storage location），就跟我们为储藏箱贴上标签一样。创建变量，请使用 JavaScript 的特别关键字 var，并为新变量命名。**关键字**（keyword）是在 JavaScript 中设置为负责特殊任务（创建变量即为其中之一）的词汇。

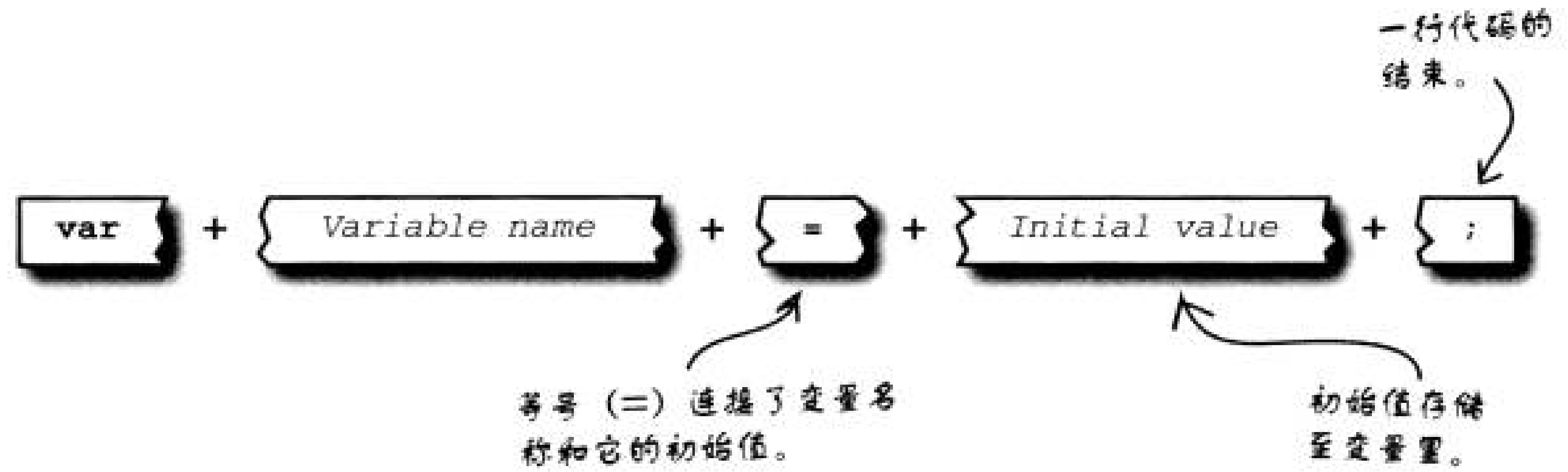


当你以 var 关键字创建变量时，变量刚开始是空的……没有值。变量刚开始是空的，没有关系，只要你在指派值给变量前别试着读取变量。下载 MP3 到播放器前，别试着播放音乐。

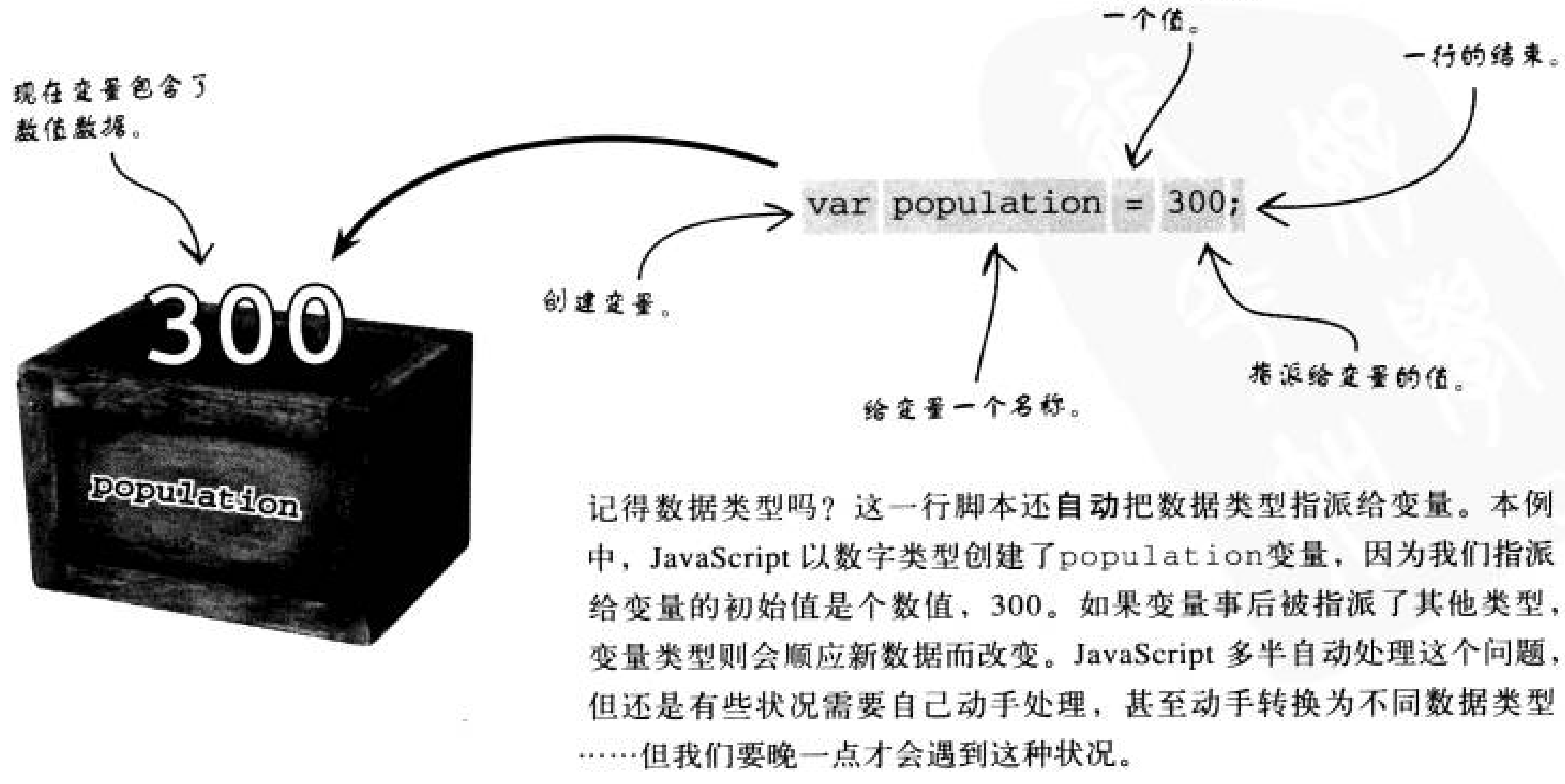
刚创建的变量都有（程序）拨出的保留空间，也准备好存储数据。存取与操纵变量存储数据的关键在于其名称。所以每个变量均有**“独特且有意义的变量名称”**才会这么重要。例如，pageHits 就是个提示变量存储内容的好名称。若把页面浏览次数命名为x或gerkin，描述性则荡然无存。

用=初始化变量

创建变量时其实已经可以加上初始值。事实上，创建一个有初始值的变量是种好习惯。这种行为称做“变量**初始化**(initializing)”，而且只要在一般创建变量的代码里稍微加点东西即可。



经过初始化的变量立刻就能投入战场，不像其他空荡荡的伙伴……变量里已经存储了值。就像买了一台预载 MP3 的音乐播放器——马上就能播放音乐。



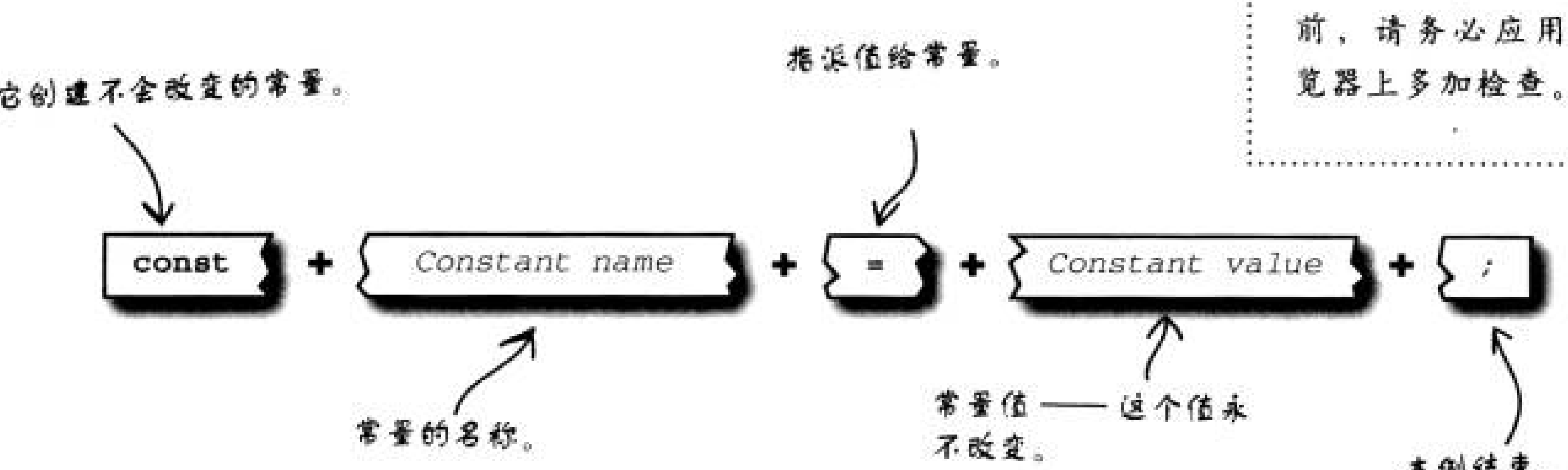
记得数据类型吗？这一行脚本还**自动**把数据类型指派给变量。本例中，JavaScript 以数字类型创建了`population`变量，因为我们指派给变量的初始值是个数值，300。如果变量事后被指派了其他类型，变量类型则会顺应新数据而改变。JavaScript 多半自动处理这个问题，但还是有些状况需要自己动手处理，甚至动手转换为不同数据类型……但我们要晚一点才会遇到这种状况。

常量拒绝改变

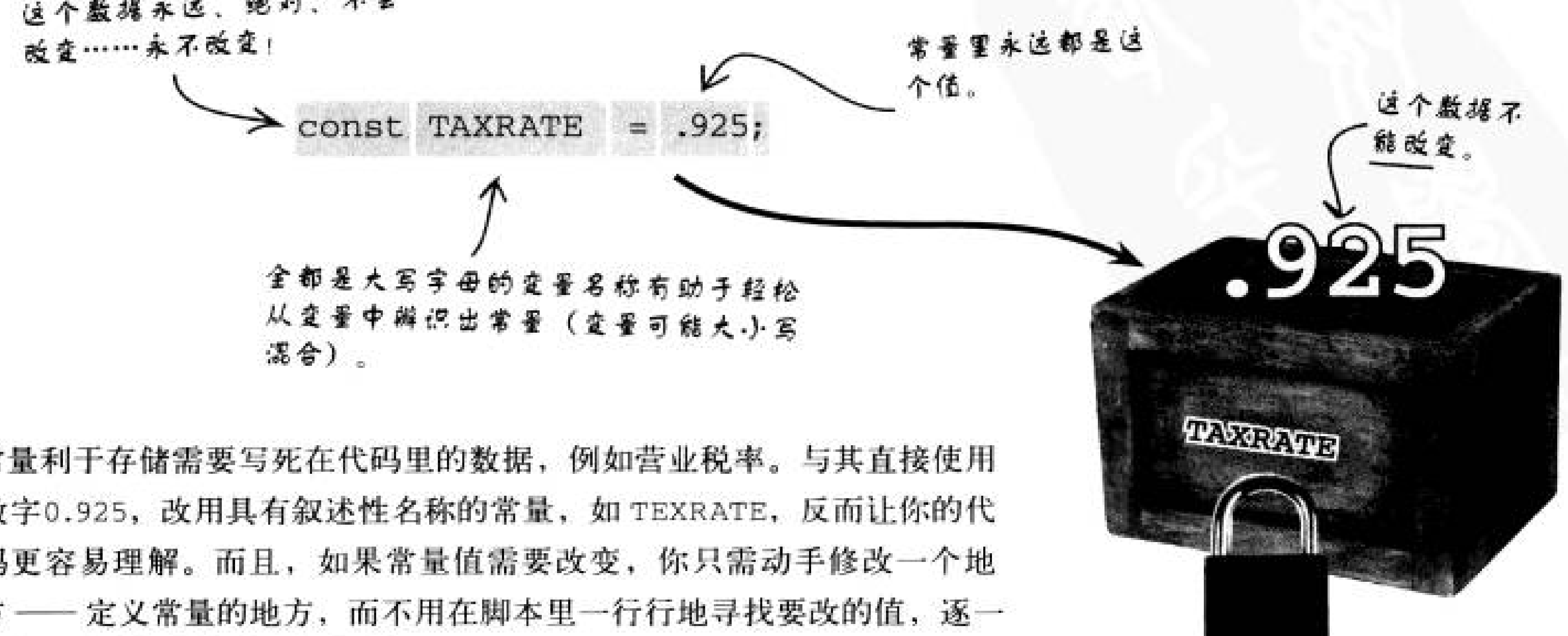
初始化一个变量只是设定它的**第一个**值 —— 不会阻止变量值**事后**发生变化。如果需要存储不会改变的数据，则需要常量的协助。常量的创建方式与初始化变量一样，但改用关键字const，而非var。此时的“初始”值变成“**永久**”值 —— 常量拿到手的东西，别想叫它放开！

注意！ **有些浏览器不支持关键字 const。**

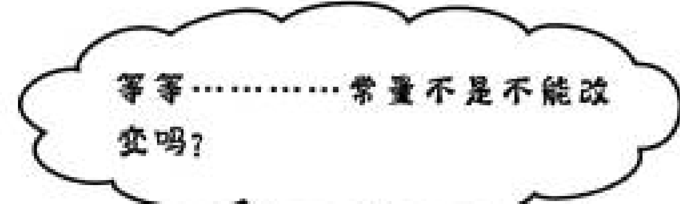
const关键字最近才加入 JavaScript，有些浏览器还不支持它。在发布使用const的 JavaScript代码前，请务必应用在常用浏览器上多加检查。



创建常量与创建变量间的最大不同，就在于使用的关键字——常量用const，变量用var。常量的语法与变量的初始化相同。不过，常量习惯以全都是大写字母的方式命名，作为与变量的区隔。



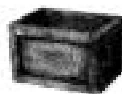
常量利于存储需要写死在代码里的数据，例如营业税率。与其直接使用数字0.925，改用具有叙述性名称的常量，如 TEXRATE，反而让你的代码更容易理解。而且，如果常量值需要改变，你只需动手修改一个地方 —— 定义常量的地方，而不用在脚本里一行行地寻找要改的值，逐一寻找可能会把事态变得很复杂。

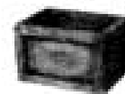


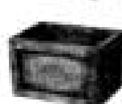
常量的确不能改变，至少要打开文本编辑器才能修改。

正在**运行**脚本时，常量确实不能改变……但我们没办法阻止你找到创建常量的地方，然后修改它的值。所以说，从脚本的观点来看，常量绝对是固定的；但从你的观点来看，只要回到创建常量的地方，就能改变它的值。在脚本运行时，税率常量不能改变，但我们可以在源代码中修改税率，此后的脚本运行结果则均以新税率为根据。

请判断下列信息应为常量或变量，然后写出创建它们的代码并予以初始化（如果需要初始化的状况）。

 目前的温度，刚开始并不知道

 从人类计算年龄的方式转变成狗计算年龄的方式（人的一年等于狗的7年）

 火箭发射的倒数计时（从10 到 0）

美味甜甜圈的价钱（50美分）

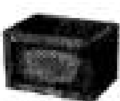
....

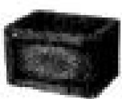
....

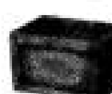
....

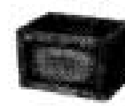
....

请判断下列信息应为常量或变量，然后写出创建它们的代码并予以初始化（如果需要初始化的状况）。

 目前的温度，刚开始并不知道

 从人类计算年龄的方式转变成狗计算年龄的方式（人的一年等于狗的7年）

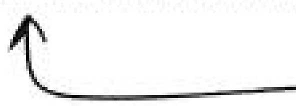
 火箭发射的倒数计时（从10 到 0）

 美味甜甜圈的价钱（50美分）

```
var temp;
```

温度随时都在变化，未经测量也不知道现在的温度，所以该把值空下来。

```
const HUMANTODOG = 7;
```



年龄的转换比例不会改变，常量是最合理的选择。

```
var countdown = 10;
```

倒数计时必须从 10 下降到 1，会改变就是变量，而且变量值也该初始化为开始倒数的数目（10）。

```
var donutPrice = 0.50; or const DONUTPRICE = 0.50;
```

甜甜圈的价格可能改变，所以设为具有初始值的变量比较合理。

……但如果甜甜圈的价格固定不变，这时改用常量存储价格比较好。

没有蠢问题

问：如果我不指定JavaScript数据的类型，它要怎么知道数据的类型？

答：JavaScript不是那种能够明确指定（变量或常量）类型的程序语言。设定JavaScript数据的值时，即已自动暗示类型。这种设计为JavaScript变量带来非常大的灵活性，因为它们的数据类型能随着不同的指派值而转变。举例来说，把数字17指派给变量x时，它的类型就是数字（number）。但如果你转变心意，改为指派“sixteen”给x，这个变量类型则自动转换为字符串。

问：既然JavaScript会自动处理数据类型，为什么我还要学习类型的相关内容啊？

答：因为这世上有很多无法依赖JavaScript自动处理类型的状况。假设有个数字存储为文本（text）类型，但你突然想把这个数字用于数学运算，这时就需要把文本转换为数字类型。如果是相反方向的转换，例如在alert框里列出数字，数据类型也同样要先转换为文本。JavaScript将自动执行数值一文字的转换，但转换结果可能跟你的预期有点差异。

问：如果我真的不知道变量的可能值，可以放着它，不做初始化吗？

答：当然没问题。初始化背后的概念，在于避免存取还没有存入值的变量。不过，总是会有不知道值却需要创建变量的状况。此时，请在使用变量前，先确定是否已设定变量值。还有，变量也可以用“什么都没有”当成初始值，例如“ ”（文本）、0（数字）、false（布林）等等。如此设定有助于减低不小心存取未初始化变量的风险。

问：有什么分辨变量与常量使用时机的诀窍吗？

答：虽然我可以简单回答“常量不会改变，但变量会改变”，可其实不只这样。我们经常先把所有东西当成变量，后来才慢慢转换其中一部分为常量。即使如此，把变量转换为常量的情形还是很少见的，反而常常花时间修正常用的文本或数字，例如欢迎信息或转换比例。与其让这类文本或数字的副本满布于代码中，应该把它们创建成常量并改用常量。如果以后还要修改这类值，就只需修改一个地方了。

问：重新载入网页时，脚本的数据怎么办？

答：此时数据重新设为初始值，回到脚本尚未执行前的状态。换句话说，重新载入网页，与第一次执行网页脚本的效果相同。

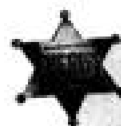
数据类型在设置变量值与常量值的同时建立。

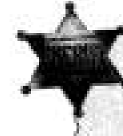
复习要点

- 脚本通常能以下列三种基本数据类型表示：**文本**（text）、**数字**（number）、**布尔**（boolean）。
- **变量**是可能在脚本执行过程中改变的数据。
- **常量**是不会改变的数据。
- 关键字var用于**创建变量**，关键字const用于创建常量。
- JavaScript数据的**数据类型**在**设定数据**为特定值的同时建立。另外，变量值的类型可以改变。

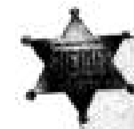
变量名称里卖什么药？

变量、常量及其他 JavaScript 语法元素在脚本里均靠独一无二的名称分辨，这种名称也叫做**标识符**（identifier）。JavaScript的标识符就像我们每个人都拥有的姓名，但比姓名缺乏灵活性（我们可能和别人同名同姓，但 JavaScript 的变量不能有这种情形）。除了在脚本中的唯一性（unique），标识符尚需承受几项 JavaScript 发布的命名法则：


标识符至少需有一个字符长。


标识符的第一个字符需为字母（letter）、下划线（_）或美元符号（$）。

第一个字符后可接字母、下划线（_）、美元符号（$）或数字（number）。


空格与特殊字符（下划线和美元符号除外）不可出现在标识符里。

为变量或常量创建JavaScript标识符时，你其实是在为一段信息命名，这段信息通常在脚本中有其**意义**（meaning）。所以，只是遵守标识符命名法则还不够，你还应该尽量为数据片段的名称赋予情境意涵，让其他人一眼就能辨识名称的意义。

当然，有时候只要 x 就能表达变量的需求 —— 有些资讯的用途就是很难描述。

标识符应具有描述性，才能轻易辨识数据，更别提还有合法性需要考虑……

变量/常量名称的合法与违法

5to10

违法：不可以数字起始。

firstName

合法：均为字母，没有问题。

top100

合法：数字未出现于首字符，可以接受。

ka_chow

合法：字母、下划线，没有问题。

!guilty

违法：除了下划线与美元符号，名称不能以其他特殊字符起始。

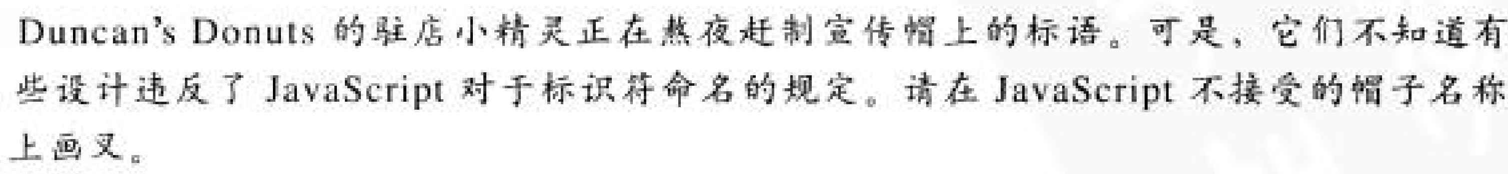
_topSecret

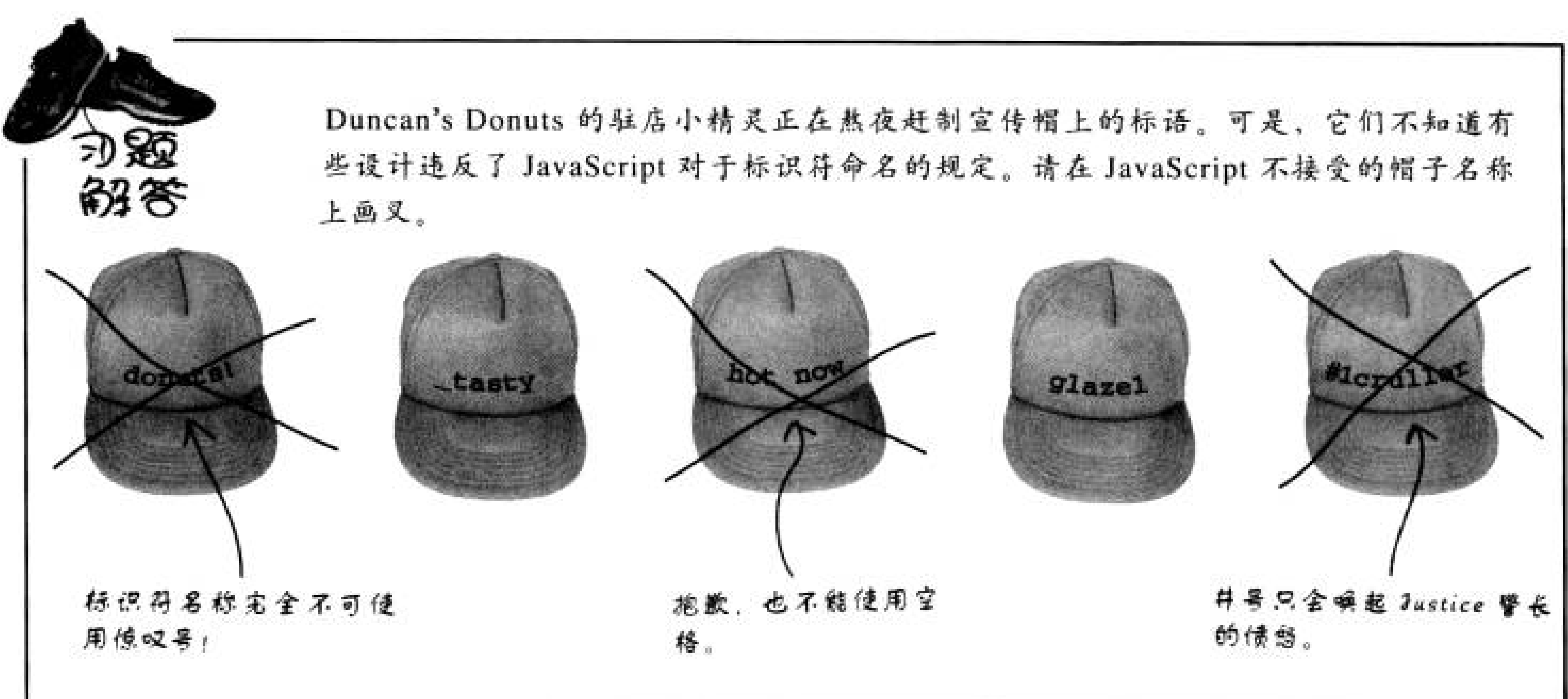
合法：以下划线起始，完全没有问题——甚至有人利用这项技巧，为特殊意义的变量命名。

$total

合法：虽然看起来有点怪，但以美元符号起始完全合法。

习题


Duncan's Donuts 的驻店小精灵正在熬夜赶制宣传帽上的标语。可是，它们不知道有些设计违反了 JavaScript 对于标识符命名的规定。请在 JavaScript 不接受的帽子名称上画叉。



变量名称通常形似驼峰

虽然没有白纸黑字指定JavaScript标识符的外形风格，但JavaScript社区有些**不成文规定**。其中一项，就是使用**驼峰型**（CamelCase），也就是当标识符由多个字词组成时，则以大小写混合的形式组成（再次提醒，变量名称不可使用空格）。变量通常使用小写驼峰造型，也就是第一个词汇全都小写，但后续的词汇则混合大小写。

小写驼峰型用于构成多个字词组成的变量名称。

JavaScript 冰箱磁铁

Duncan's Donuts 的标识符磁铁，与它们对应的变量或常量分散了。请把每个变量/常量与正确的磁铁联结起来，并确定它们的名称都符合JavaScript的规定。加分：标出数据类型。

今天卖出的咖啡数量（杯）

获得本月之星的员工姓名

制作一批甜甜圈所需的面粉量

用于决定商品含税额的营业税数字

泡芙夹心大胃王记录保持者

警示系统的状态

employee*of*the*Month

cups-o-coffee

Employee of the Month

FLOURPERBATCH

alarmStatus

Tax#

alarm _ status

eclairRECORDHOLDER

TAXNUM

numCups

eclairRecord

eclairWinner!

employeeOfMonth

ALARM-STATUS

flour quantity

#OfCups

JavaScript 冰箱磁铁解答

Duncan's Donuts 的标识符磁铁，与它们对应的变量或常量分散了。请把每个变量/常量与正确的磁铁联结起来，并确定它们的名称都符合JavaScript的规定。加分：标出数据类型。

今天卖出的咖啡数量（杯）

numCups

Number

获得本月之星的员工姓名

employeeOfMonth

Text

制作一批甜甜圈所需的面粉量

FLOURPERBATCH

Number

用于决定商品含税额的营业税数字

TAXNUM

Number

泡芙夹心大胃王记录保持者

eclairRecord

Text

警示系统的状态

alarmStatus

Boolean

剩下的JavaScript磁铁都是违规名称。

cups-o-coffee

Employee of the Month

alarm _ status

eclairWinner!

eclairRECORDHOLDER

employee*of*the*Month

Tax#

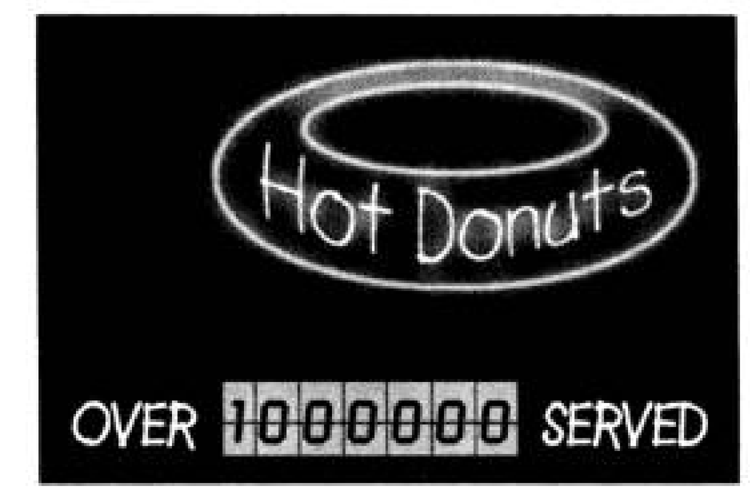
ALARM-STATUS

flour quantity

#OfCups

下一个（甜甜圈市场）头条新闻

你或许听过 Duncan's Donut，但你应该没见过 Duncan 本人，也没听过他对甜甜圈市场的惊天计划。Duncan 想把“热甜甜圈（Hot Dounts）”的市场带入下个层次……带入网络世界！他想做**即时甜甜圈**，当客户透过网络下了订单并输入指定的取货时间后，时间到了，客户就能拿到热腾腾的甜甜圈。**你的工作就是确认用户输入必需数据，同时还要计算税额与订购总额。**



Duncan's Just-In-Time Donuts
All donuts 50 cents each, cake or glazed!
Name: Paul
# of cake donuts: 0
# of glazed donuts: 12
Minutes 'til pickup: 45
Subtotal: $6.00
Tax: $0.55
Total: $6.55
Place Order

你好，我是 Duncan。这个热甜甜圈的在线订购系统将撼动世界！

JavaScript 捕捉用户的输入，并计算含税价与订购总额。

12 glazed
Pick up in 45 minutes for Paul

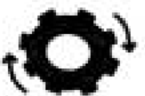
Blaster 3000，甜甜圈制造机。

x 12

热呼呼，刚出炉的甜甜圈！

规划 Duncan's Donut 网站

处理即时的甜甜圈订单，需要两种**检查**（或称**验证**），包括所需数据的订购表单，以及根据数据而计算的订购总额。小计与总价均于数据输入后**即时**计算，让用户**立刻收到**订购总额的响应。“Place Order”按钮用于送出“最终”订单，这不是JavaScript的责任……暂时先不考虑这个问题。



这些是订单需要的信息，所以应该交由 JavaScript 验证。

Duncan's Just-In-Time Donuts

Duncan's Just-In-Time Donuts

All donuts 50 cents each, cake or glazed!

Name: Paul

\# of cake donuts: 0

\# of glazed donuts: 12

Minutes 'til pickup: 45

Subtotal: $6.00

Tax: $0.55

Total: $6.55

Place Order

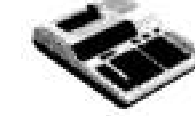
Done

这些信息需由JavaScript即时计算。

JavaScript 不负责最后的交付表单给网络服务器的动作。

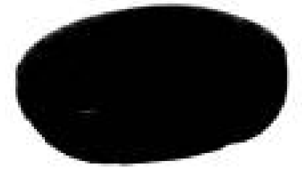
小计（subtotal）为甜甜圈数量乘上单价：

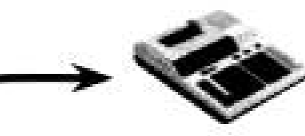
(# of cake donuts + # of glazed donuts) x price per donut



税额（tax）则是小计乘上税率：

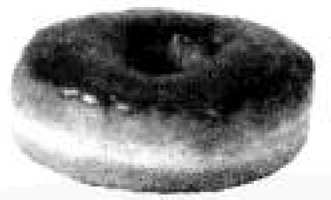
subtotal x tax rate



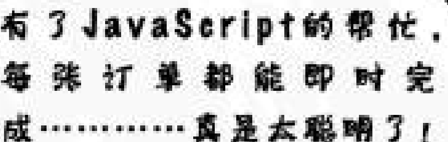


订单总额则是小计加上税额：

subtotal + tax



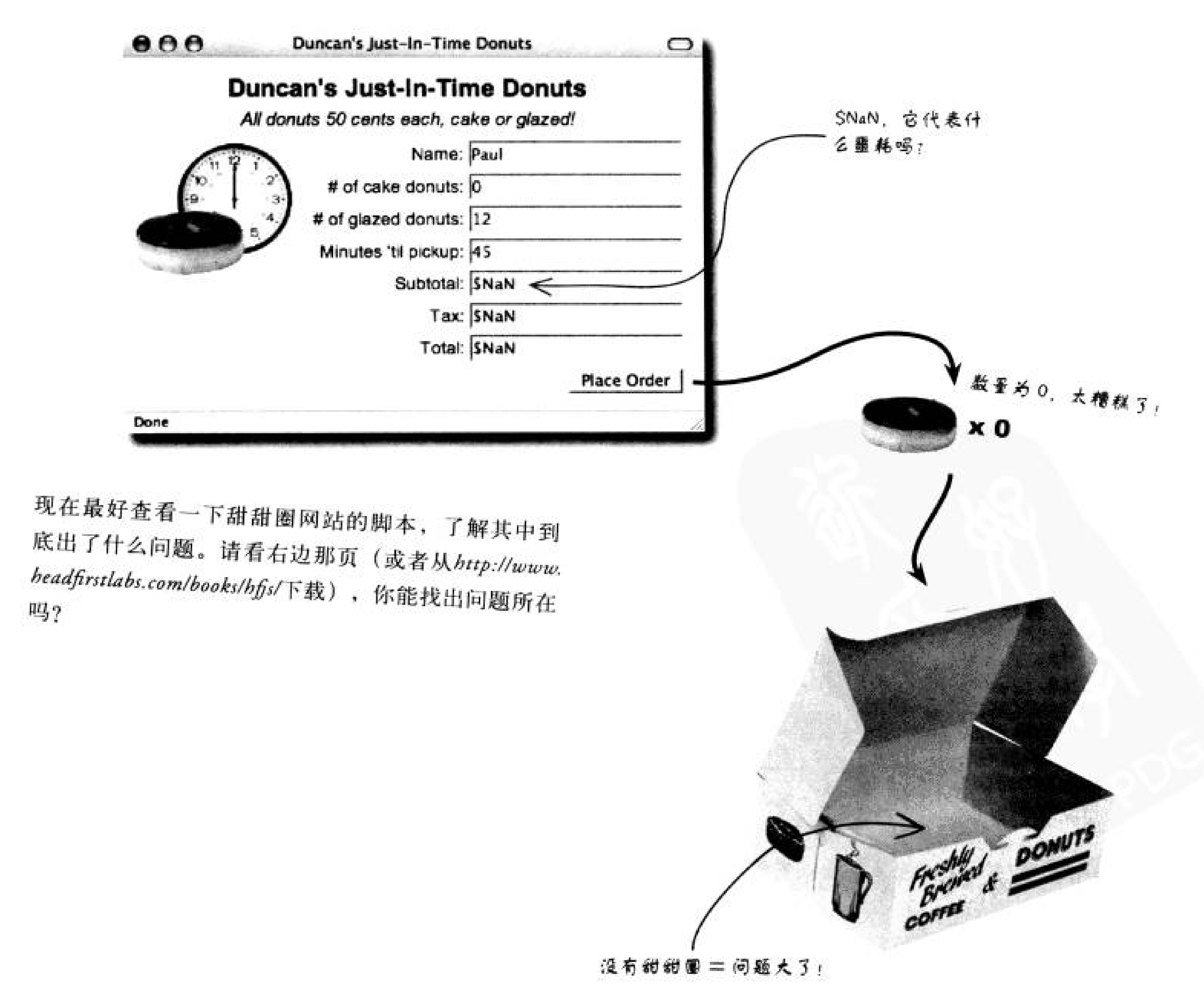
看来Duncan的订单里有不少需要追踪的数据。他不只需要保存用户输入的许多数据，还要把数据交给JavaScript代码计算。



上述计算需要哪些变量或常量？你会怎么命名？

甜甜圈计算首试

Duncan试着自己动手设计JavaScript的计算程序，但遇到了问题。用户输入甜甜圈数量后，应该在线即时计算的部分……错乱了。计算结果居然是 `$NaN`，也不知道代表什么意思。还有，订单根本无法传递给机器，用户对于 Duncan 的“技术革新”当然也不怎么捧场。



现在最好查看一下甜甜圈网站的脚本，了解其中到底出了什么问题。请看右边那页（或者从*http://www.headfirstlabs.com/books/hfjs/*下载），你能找出问题所在吗？

调用这段代码，立即开始计算甜甜圈的小计金额与总额，并更新订单。

因为用户输入的数据似乎没问题，那么问题应该出在常量身上。

这段代码把订单送交给网络服务器，并负责对用户确认订单无误。

只要甜甜圈数量有变，即更新订单。

订单于点击"Place Order"按钮后送出。

```
<html>
  <head>
    <title>Duncans' Just-In-Time Donuts</title>
    <link rel="stylesheet" type="text/css" href="donuts.css" />
    <script type=text/javascript>
      function updateOrder() {
        const TAXRATE;
        const DONUTPRICE;
        var numCakeDonuts = document.getElementById("cakedonuts").value;
        var numGlazedDonuts = document.getElementById("glazeddonuts").value;
        var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
        var tax = subTotal * TAXRATE;
        var total = subTotal + tax;
        document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
        document.getElementById("tax").value = "$" + tax.toFixed(2);
        document.getElementById("total").value = "$" + total.toFixed(2);
      }
      function placeOrder() {
        // Submit order to server...
        form.submit();
      }
    </script>
  </head>
  <body>
    <div id="frame">
      ...
      <form name="orderform" action="donuts.php" method="POST">
        ...
        <div class="field">
        # of cake donuts: <input type=text id=cakedonuts name=cakedonuts
          value="" onchange="updateOrder();" />
        </div>
        <div class="field">
        # of glazed donuts: <input type="text" id="glazeddonuts"
          name="glazeddonuts" value="" onchange="updateOrder();" />
        </div>
        ...
        <div class="field">
        <input type="button" value="Place Order"
        onclick="placeOrder(this.form);" />
        </div>
      </form>
    </div>
  </body>
</html>
```

磨笔上阵

请问，你觉得Duncan's Donut的即时甜甜圈脚本有哪些地方出错了？

....

....

请问，你觉得Duncan's Donut的即时甜甜圈脚本有哪些地方出错了？

常量 TAXRATE 与 DONUTPRICE 并未初始化，因此相关计算无法完成。

好吧……我了解常量要一直维持相同的值，但它们怎么会未初始化呢？

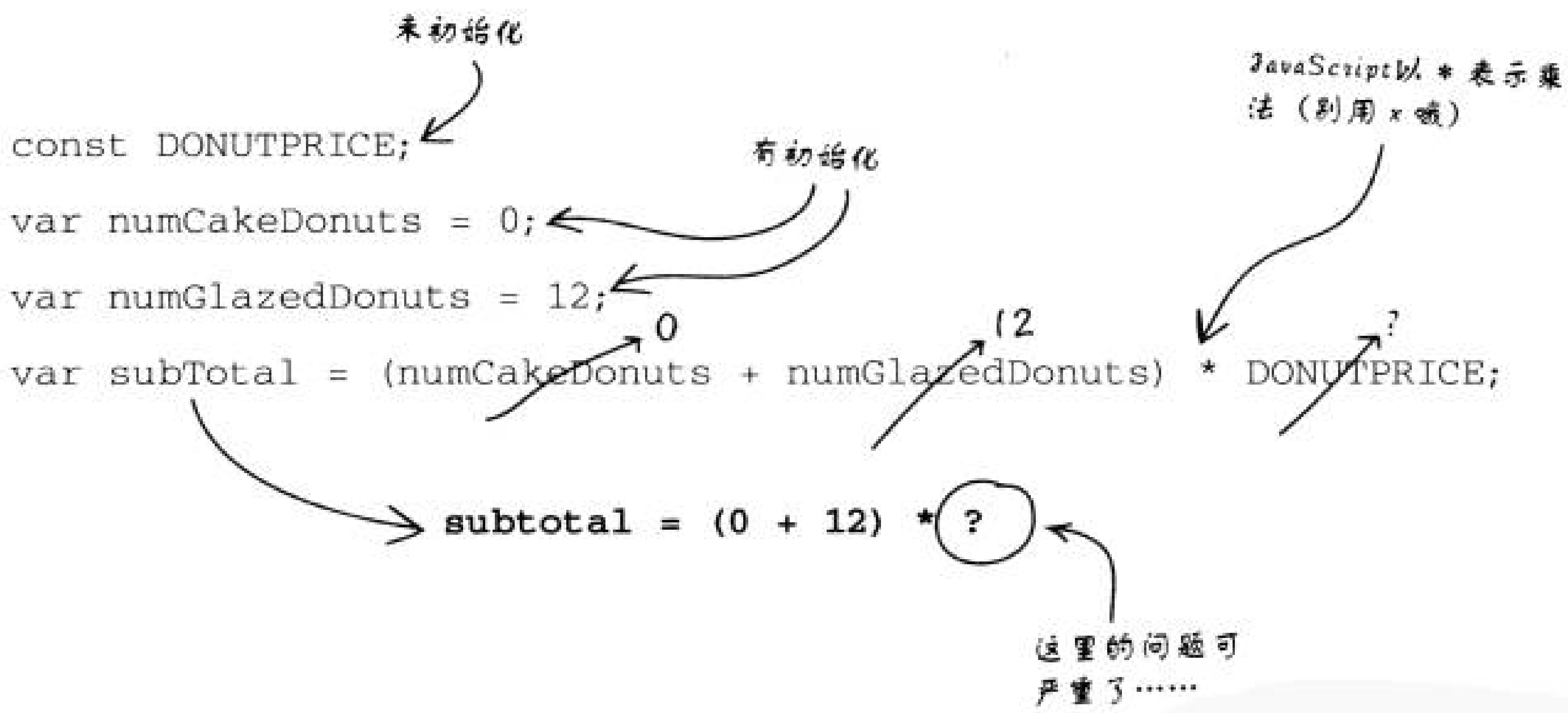
你不该做出未初始化的常量。

如果不指定常量值，当然会有未初始化的常量，但这种情况**非常糟糕，糟糕到极点**。若未于创建常量时初始化，常量将无所适从 —— 没有值，而且还**没办法指定它的值**。未初始化的常量最终成为**编码错误**，虽然浏览器通常不认识这个错误。

创建常量时务必初始化。

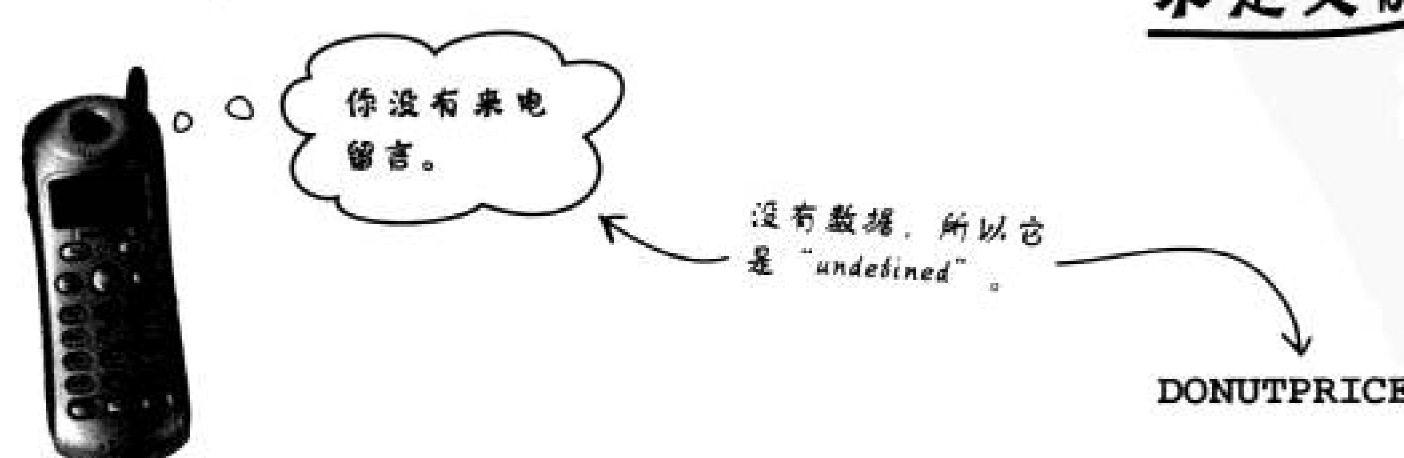
初始化你的数据……不然就……

如果某段数据并未初始化，就称为**未定义的**（undefined）——“没有值”的华丽形容方式。这可不是说它没有价值，只是表示其中没有信息……暂时没有。当你试着使用未初始化的变量或常量时，问题就出现了。



常量DONUTPRICE**未初始化**，表示它**没有值**。事实上，JavaScript有个特殊的、专门描述“没有值”状态的值：undefined。有点像你打开答录机时听到“没有留言”的信息——技术上而言，**“没有留言”**也是一则留言，但它的存在却是为了告知**“没有留言”**。undefined的用途也一样，它是为了告知**“没有数据”**。

一段数据如果没有值，就是未定义的。



NaN，非数字

一如 undefined 代表一种特殊的数据状态，JavaScript 还有一个很重要的值，用于表达变量的特殊状况：NaN，非数字（Not a Number）。前例中，因为信息不够，无法计算，subTotal遂被设为NaN。换句话说，某个应该是数字的值不见时，就会出现NaN。

数字

非数字啦！

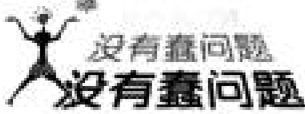
```
subtotal = (0 + 12) * ? = NaN
```

既然这个数据并未定义，计算无法继续下去。

NaN是个非数字的值，即使我们以为它会是个数字。

所以，解决NaN的问题，需要在创建常量DONUTPRICE 的同时初始化：

```
const DONUTPRICE = 0.50;
```



问：一份脚本中的标识符必须独一无二，这是什么意思？

答：“标识符”原本就是用于辨识脚本中的某段信息。在真实生活中，你我可能拥有相同的姓名……但人类有办法处理“同名同姓”的问题，分辨出是你或是我。JavaScript 则无法面对这种模棱两可，所以需要我们小心地使用不同的名称来分辨不同的信息。想达成这一点，就要确认脚本里的标识符都不一样。

问：我创建的所有标识符都必须独一无二吗？还是一份脚本里的标识符都有独特性就好了？

答：标识符的独特性只需在单一一份脚本里达成；有时甚至只要在脚本的某个部分有独特性就够了。不过，最好记得网络应用程序的脚本可能变得非常庞大、横跨多个文件。在这种情况下，确保每个标识符的独特性就变成一种挑战了。幸好，在自己的脚本中维持标识符的独特性不算太困难，只要在命名变量时尽量具有描述性。

问：我还是不太懂驼峰型与小写驼峰型的使用时机。有什么差别吗？

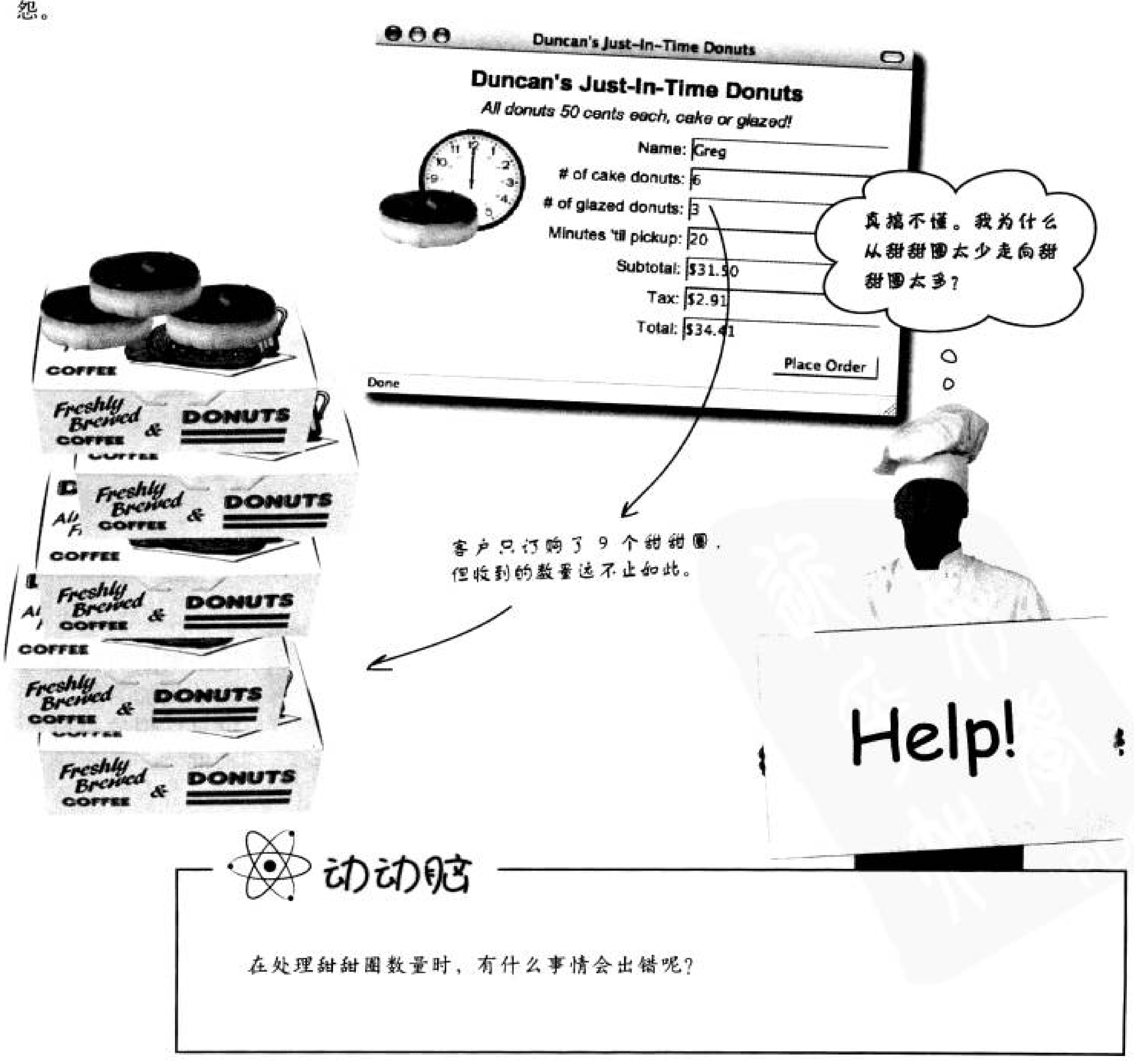
答：驼峰型（首字母也大写）只会用于对象，第9章将详细说明。小写驼峰型则用于变量和函数，它与驼峰型几乎一样，只不过差别在于首字母小写。所以，驼峰型表示我们可能把对象命名为Donut，而把函数命名为getDonut()，把变量命名为numDonuts。常量的名称则没有这些变化，全都大写就对了。

问：文本与布尔类型的数据会被视为 NaN 吗？

答：理论上，是的，因为它们决非数字。但在实际上，并非如此。NaN的设计目的，在于表示并非我们预期的数字数据。换句话说，NaN并非一般的 JavaScript 数据描述方式，它是数字数据类型有误时的指标。通常只在计算过程中需要数字，却收到非数值的数据时，才会看到NaN。

现在，回头讨论 Duncan 的问题

回头讨论 Duncan's Donut，事态越变越糟。原本是送出空的甜甜圈盒，现在变成制出过多甜甜圈 —— 每份订单的数量都计算得过多。Duncan 一直收到甜甜圈送得太多和面团填充得太多的抱怨。



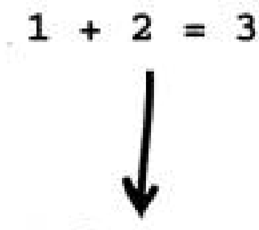
动动脑

在处理甜甜圈数量时，有什么事情会出错呢？

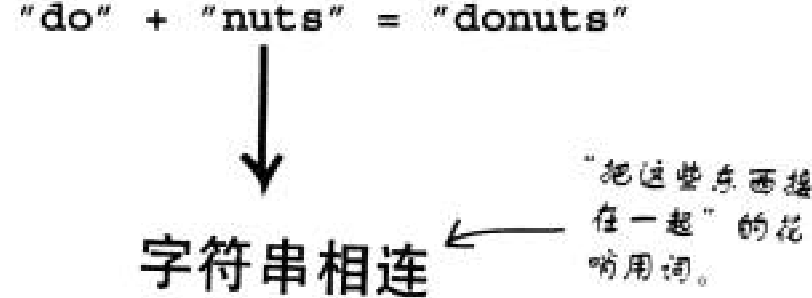
加法不只能用在数字上

JavaScript 里，**上下文**（context）就是一切。不只在使用数据**做事**（doing）时重要，上下文对于我们在代码里操纵**何种**（what kind）数据尤显重要。就连“两段信息的加法”这么简单的事情，都可能因为数据的**类型**而产生非常不同的结果。

```
1 + 2 = 3
```



```
"do" + "nuts" = "donuts"
```



数字相加

两个数字的加法 —— 产生两值应用**算数**加法后的结果 —— 跟你想的一样。

字符串相连

“把这些东西接在一起”的花哨用词。

两个字符串的加法也会产生各位或许意料到的结果，但与算数的加法非常不一样 —— 字符串将**首尾相连**（end-to-end）。

刚才说到字符串的加法与数字的不一样，如果把两个文本类型的数字应用加法呢？会发生什么变化？

```
"1" + "2" = ?
```

相加、相连，会是哪一种变化？

JavaScript 对于字符串里的内容并不在意 —— 它们对 JavaScript 而言都是字符。因此，字符串内容虽为数字，但对于“字符串相连”的规则并无影响 —— 如果你的**本意**是想做算数加法，此时的结果或许有点**出乎意料**。

注意！

应用加法前，请确认相加的数据是你想象中的数据

在试图相加数字时意外做了字符串相连，是种常见的JavaScript错误。如果想做数字相加，请确定先把字符串转换为数字了，再应用加法。

```
"1" + "2" = "12"
```

结果也是字串，怎么看都不像“数字相加”吧！

既然它们是字符串而非数字，就会使用“字符串相连”的规则做加法。

parseInt() 与 parseFloat()：文本转换为数字

虽然有相加／相连的问题，但对于应用算数运算至存储成字符串的数字，其实有正统的解决方法。这种时候，我们需要**先**把字符串**转换**（convert）为数字，再应用任何算数运算。JavaScript提供两个便利的转换函数：

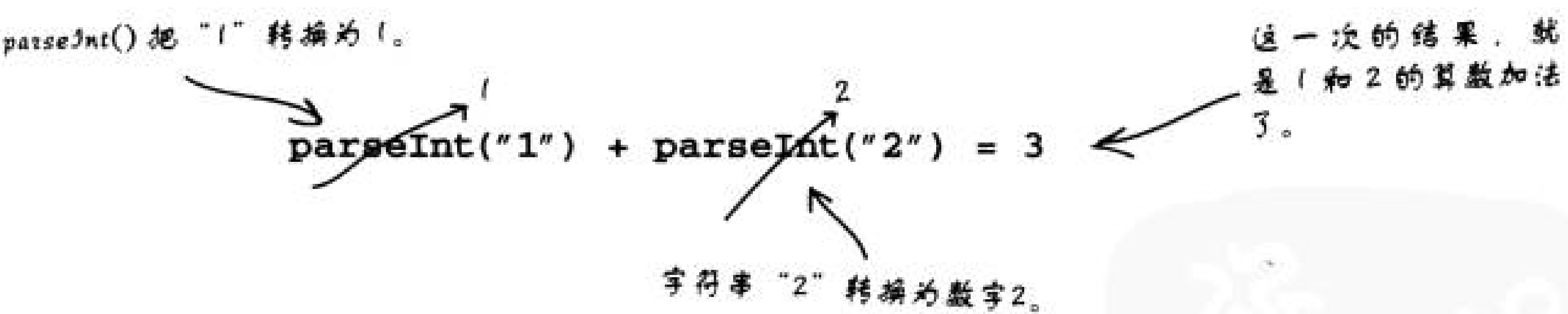
`parseInt()`

把字符串传给函数，函数就把字符串转换为整数

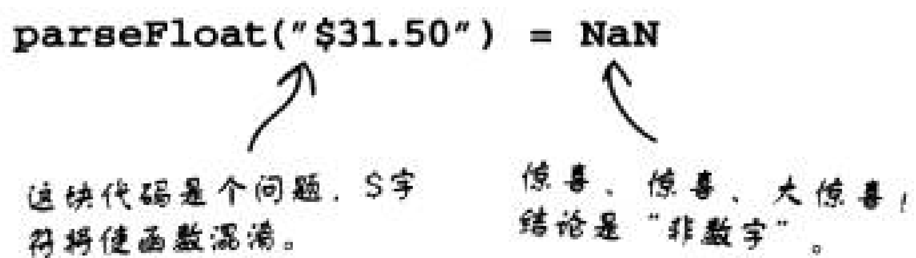
`parseFloat()`

把字符串传给函数，函数就把字符串转换为（十进位）浮点数

这两个内置函数都接受字符串，并在转换后返回数字：



请大家记得：parseInt()与parseFloat() 函数不保证转换一定成功。我们传给函数的信息决定它们的运作是否成功。函数会尽一切能力，把字符串转换为数字，但我们应该只提供"只包含数字"的字符串。



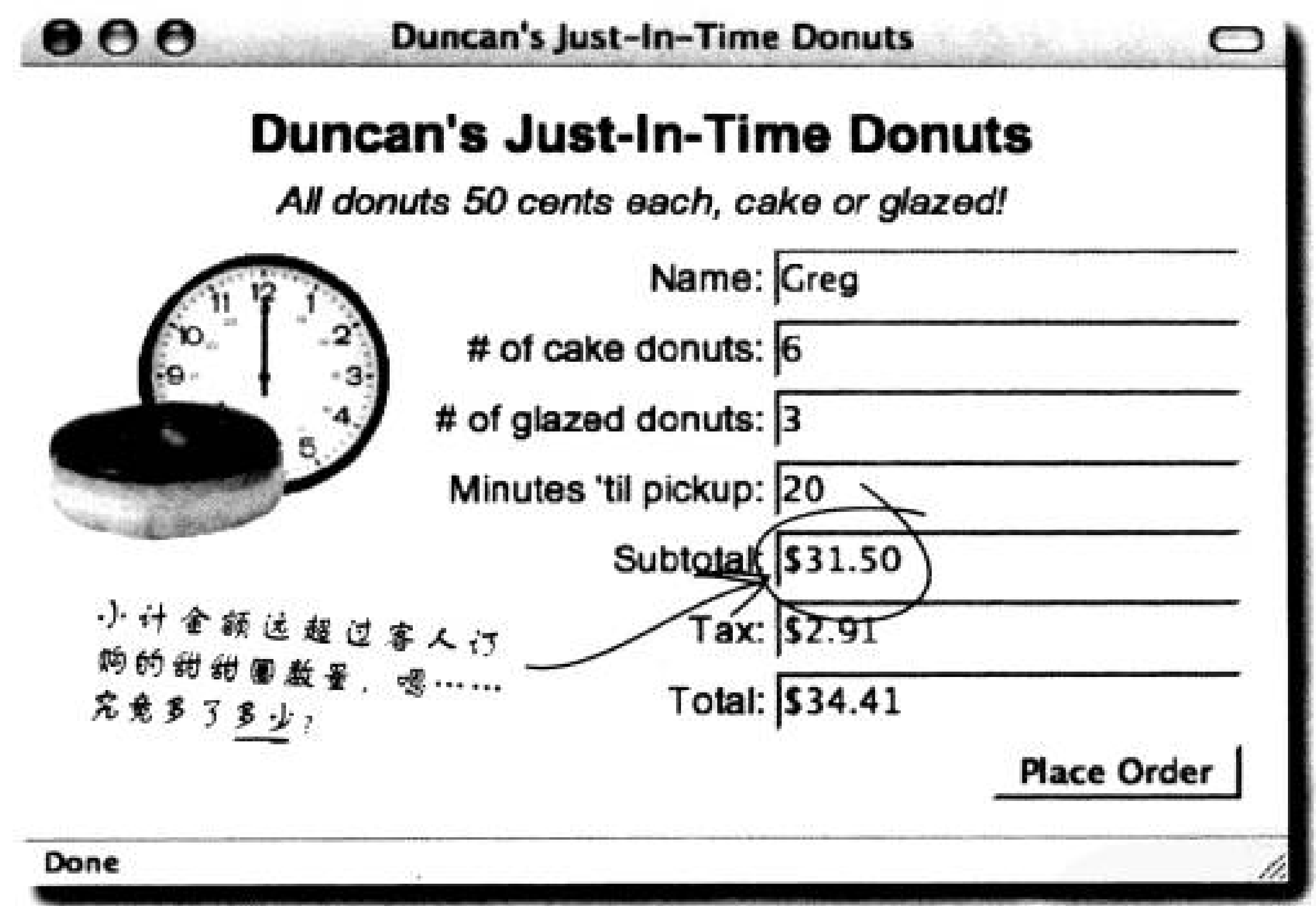
Fun 轻松

如果这里的函数还是让你觉得困惑，先别担心。

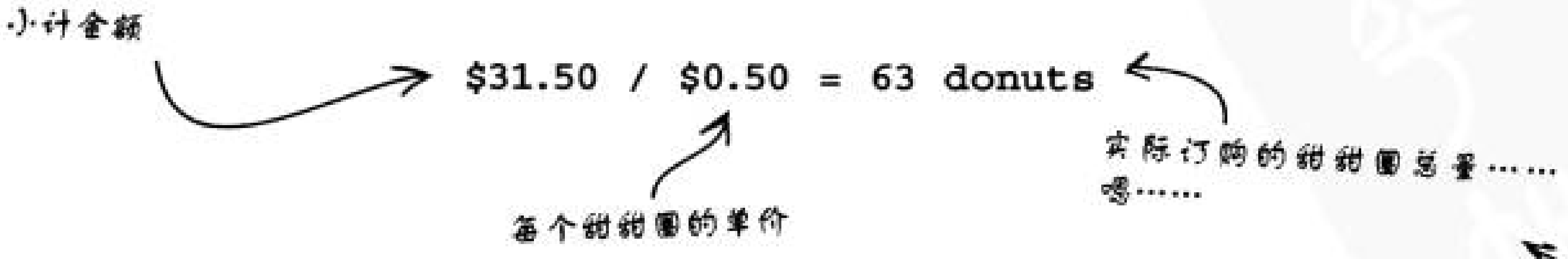
稍后，我们会提到关于函数的内幕——现在，你只要知道函数能让我们传入信息，而后返回某些东西就够了。

为何订了多余的甜甜圈

进一步查看即时甜甜圈的订购表单。我们应该能找出意外订了这么多甜甜圈的原因……



把小计金额除以甜甜圈单价……答案就是意外订购的甜甜圈数量。



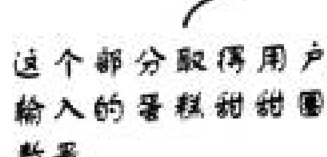
怎么看都像是数字字符串相连的问题，尤其在想到表单数据总是存储成字符串后（无论实际内容为何）。虽然我们输入表单域的数据是数字，但从 JavaScript 看来，它们只是字符。所以我们需要把字符串转换为正牌的数字，以免 JavaScript 误把数字相加当成字符串相连。

还记得"1" + "2" = "12"这回事吗？看起来这里发生了相同情况，不是吗？

磨笔上阵

使用下列代码片段，抓出填入甜甜圈数量的表单域的内容，为Duncan的updateOrder()补完不见的程序代码，让甜甜圈数量从字符串转换为数字。

```
document.getElementById("cakedonuts").value
```



这个部分取得用户输入的蛋糕甜甜圈数量。

这个部分取得用户输入的糖霜甜甜圈数量。

```
document.getElementById("glazeddonuts").value
```

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts =

    ..........................................................
  var numGlazedDonuts =

    ..........................................................
  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

使用下列代码片段，抓出填入甜甜圈数量的表单域的内容，为Duncan的 updateOrder() 补完不见的程序代码，让甜甜圈数量从字符串转换为数字。

```
document.getElementById("cakedonuts").value
```

```
document.getElementById("glazeddonuts").value
```

因为两个数字都是整数，遂采用parseInt()转换字符串。

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts =
    parseInt(document.getElementById( "cakedonuts" ).value);
  var numGlazedDonuts =
    parseInt(document.getElementById( "glazeddonuts" ).value);
  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

toFix()函数把金额四舍五入到小数点后两位。

复习要点

- 虽然 JavaScript 并未严格地要求，常量名称采用**全部大写**（UPPERCASE）、变量名称采用**小写驼峰型**（lowerCamelCase）是种良好的编程习惯。
- 创建常量时必定予以初始化，只要可能，最好也做变量初始化。
- 变量未初始化时，它将维持在**未定义的**（undefined）状态，直到我们指派任何值过去。
- NaN代表示**非数字**，也用于指出某段数据不是数字（当它期待中应该是数字时）。
- 字符串相连与算数的加法非常不同，虽然都使用加号（+）。

你找出了问题……

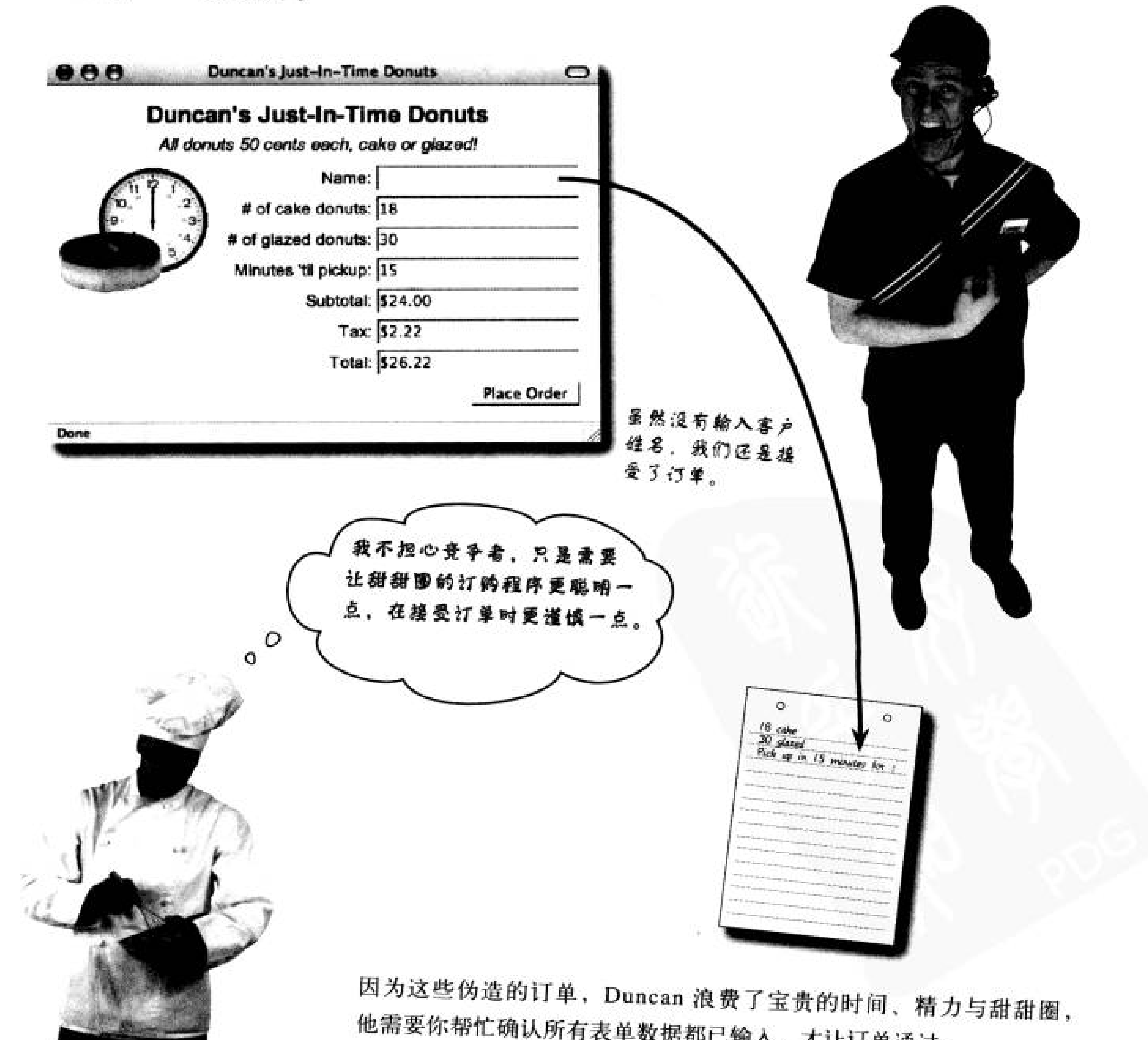
Duncan 看到你修复 JavaScript 程序代码的手法，大感佩服。他终于收到精准的订单……生意也蒸蒸日上了！

太好了，你让在线订购系统完美运作了！

当然，假设几个小修正就能永远解决所有问题，实在太冒险了。事实上，最麻烦的问题，有时候会因意料外的外在力量而显现……

Duncan发现了甜甜圈间谍

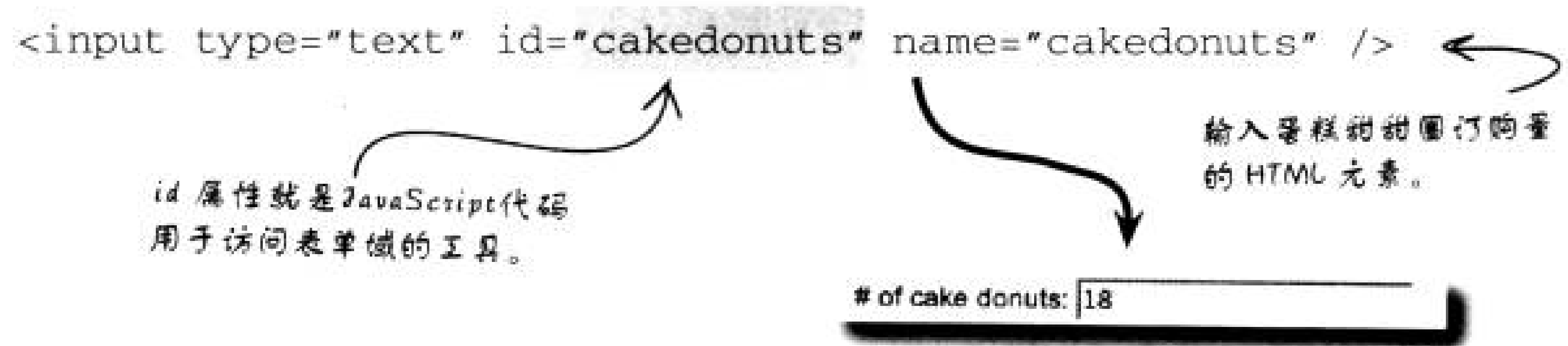
Duncan遇到了新的问题：狡猾的竞争者，Frankie。Frankie就在对街经营热狗摊，现在开始提供早餐服务“Breakfast Hound”。问题在于Frankie耍心机，他捏造了无名氏订购的甜甜圈订单。所以我们手边多了没有客户的订单——太糟糕了。



因为这些伪造的订单，Duncan浪费了宝贵的时间、精力与甜甜圈，他需要你帮忙确认所有表单数据都已输入，才让订单通过。

用 getElementById() 捕捉表单数据

为了检查表单数据是否合法（validity），你需要用于捕捉网页数据的方式。以 JavaScript 取得网页元素的关键，就是 HTML 标签里的`id`属性。



JavaScript 允许我们利用`getElementById()`函数以ID访问网页元素。这个函数不会直接捕捉元素的数据，而是以 JavaScript对象的形式提供 HTML域内容。我们则透过域的`value`性质（property）访问数据。

就技术而言，getElementById() 是个document对象里的方法（method），而不是函数。

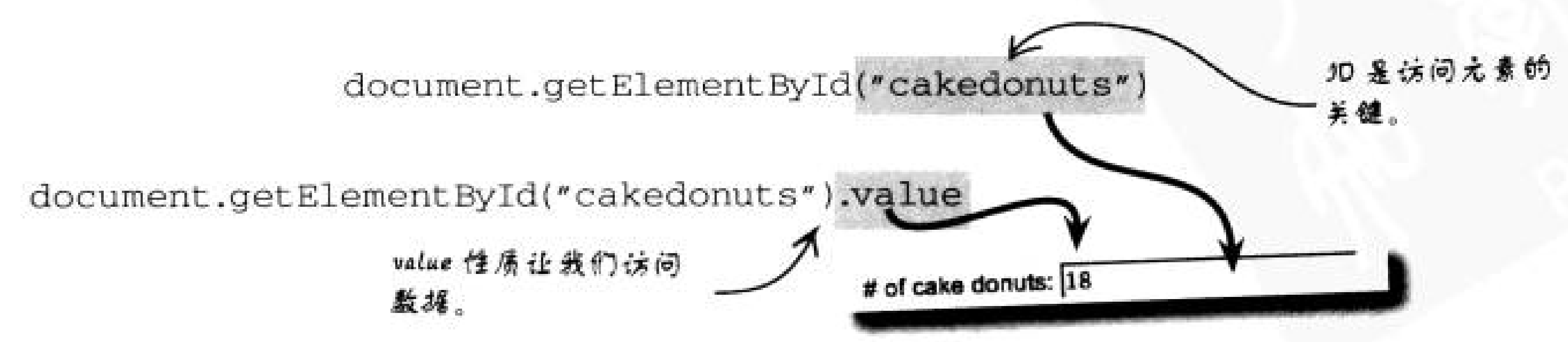
document.getElementById()

把网页上某个元素的 ID 传给这个函数，它就会把元素本身交给你，元素则可用于访问网页上的数据。

getElementById() 方法属于 document 对象。

先别对着对象、性质、方法等名词哭天抢地。

JavaScript支持一种先进的数据类型——对象（object）能达成一些非常酷的工作。事实上，整个 JavaScript 语言就是一堆对象。本书稍后将进一步讨论对象——现在，各位只要先知道"方法"很像函数，"性质"很像变量。

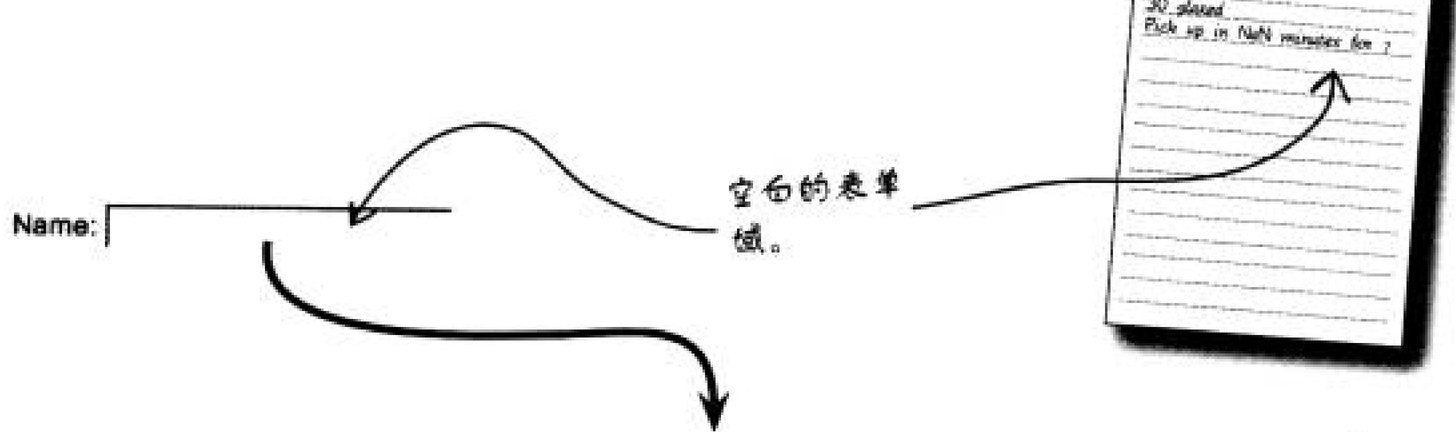


有了这段程序代码，你已经准备好检查 Duncan 的表单数据，以确保接受订单前每个表单域都有数据了。

验证网页表单的数据

你需要确认甜甜圈订单上是否输入了客户姓名。未输入取货时间，可能也是个问题，毕竟服务目的在于即时提供热腾腾的甜甜圈。所以说，至少你会想确认这两组数据都已输入而且合格（valid）。

检查表单域是否为空白，就是检查表单域值是否为空字符串（""）。

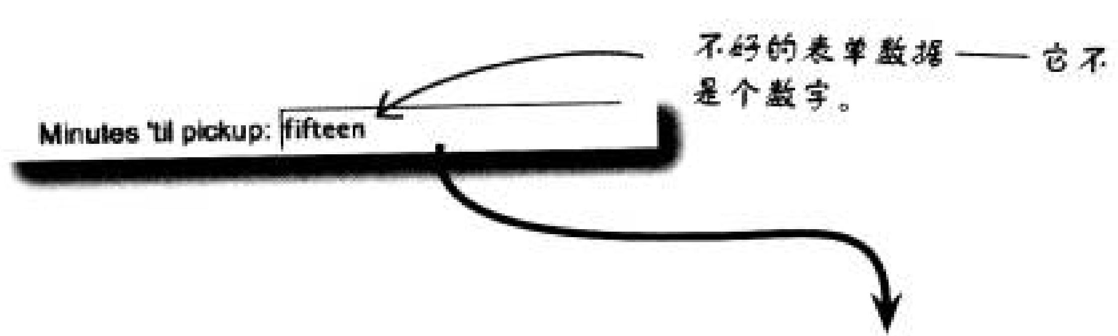



```
document.getElementById("name").value
```

""

如果name域值是个空字符串，你就知道订单需要暂缓处理，而用户应该收到询问姓名的要求。minutes域也应采用相同处理，不过或许应该进一步检查该域的数据是否为数字。内置函数isNaN()使这项检查成真——把值传给这个函数，然后它可能告诉你两种状况：1. 值不是数字（true）2. 值是数字（false）。

空字符串，是域里没有数据的线索。




```
isNaN(document.getElementById("pickupminutes").value);
```

isNaN()确认值是否为数值。

如果检查结果为true，表示数据不为数字，因此不可继续处理订单。 → true

JavaScript冰箱磁铁

我们用 placeOrder() 函数验证姓名与取货时间。使用以下提供的冰箱磁铁，完成检查姓名与取货时间数据是否存在的程序代码，还要确认取货时间是否为数字。你将需要每个磁铁，有些磁铁甚至不只使用一次。

"if" 用于检测条件式，然后根据条件采取行动——如果（if）这样，就做某事。

这里是个相等测试：A是否等于B？

两个条件之一可能造成某种行动——如果这样"或"那样，则做某件事。

```
function placeOrder() {
  if (.................................................................................................. == ..........)
    alert("I'm sorry but you must provide your name before submitting an order.");

  else if (........................................................................................ == .......... ||

    .................................................................................................................. )
    alert("I'm sorry but you must provide the number of minutes until pick-up" +
      "before submitting an order.");
  else
    // Submit the order to the server
    form.submit();
}
```

pickupminutes

"name"

.

)

""

(

document

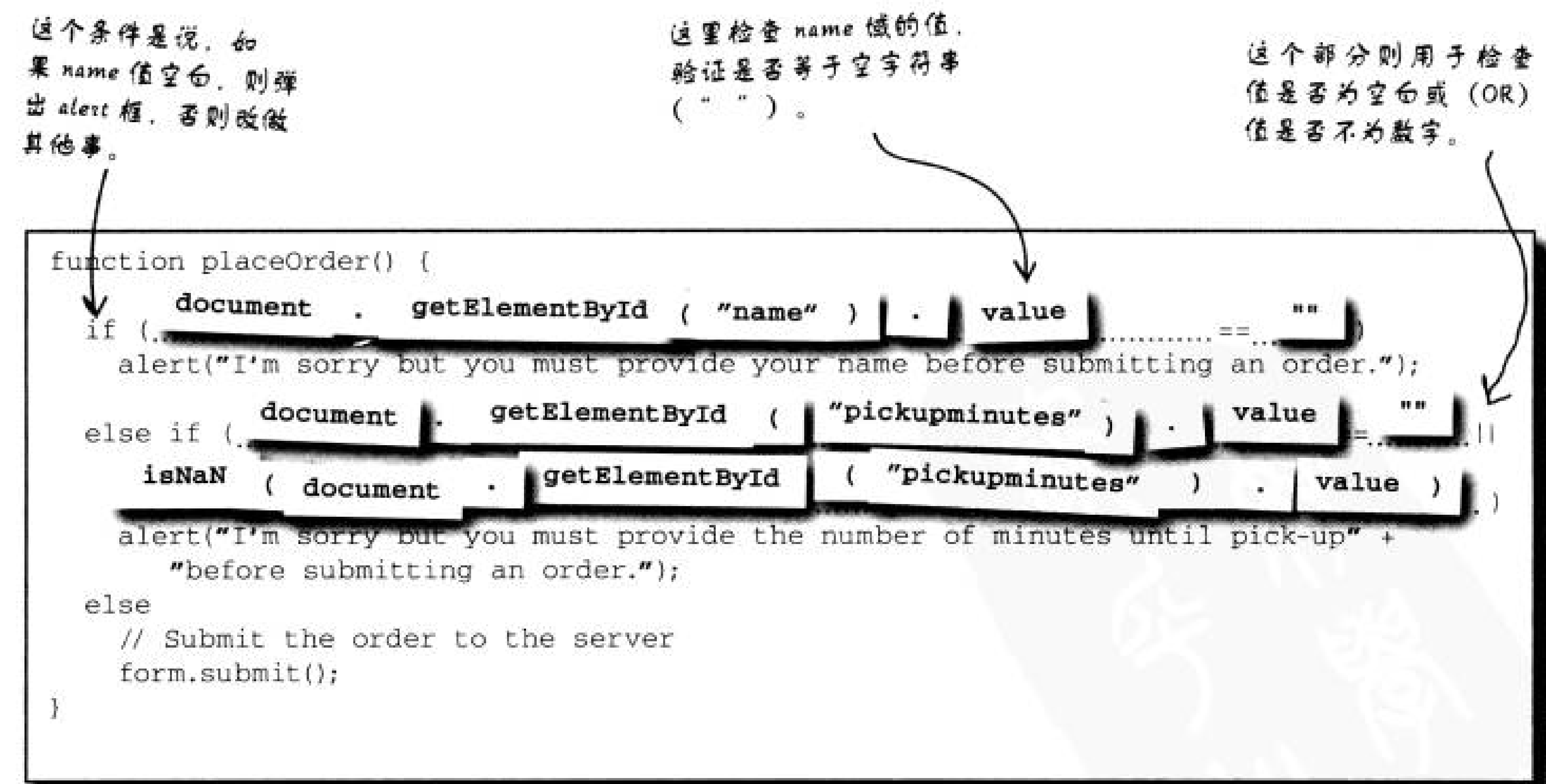
isNaN

value

getElementById

JavaScript 冰箱磁铁解答

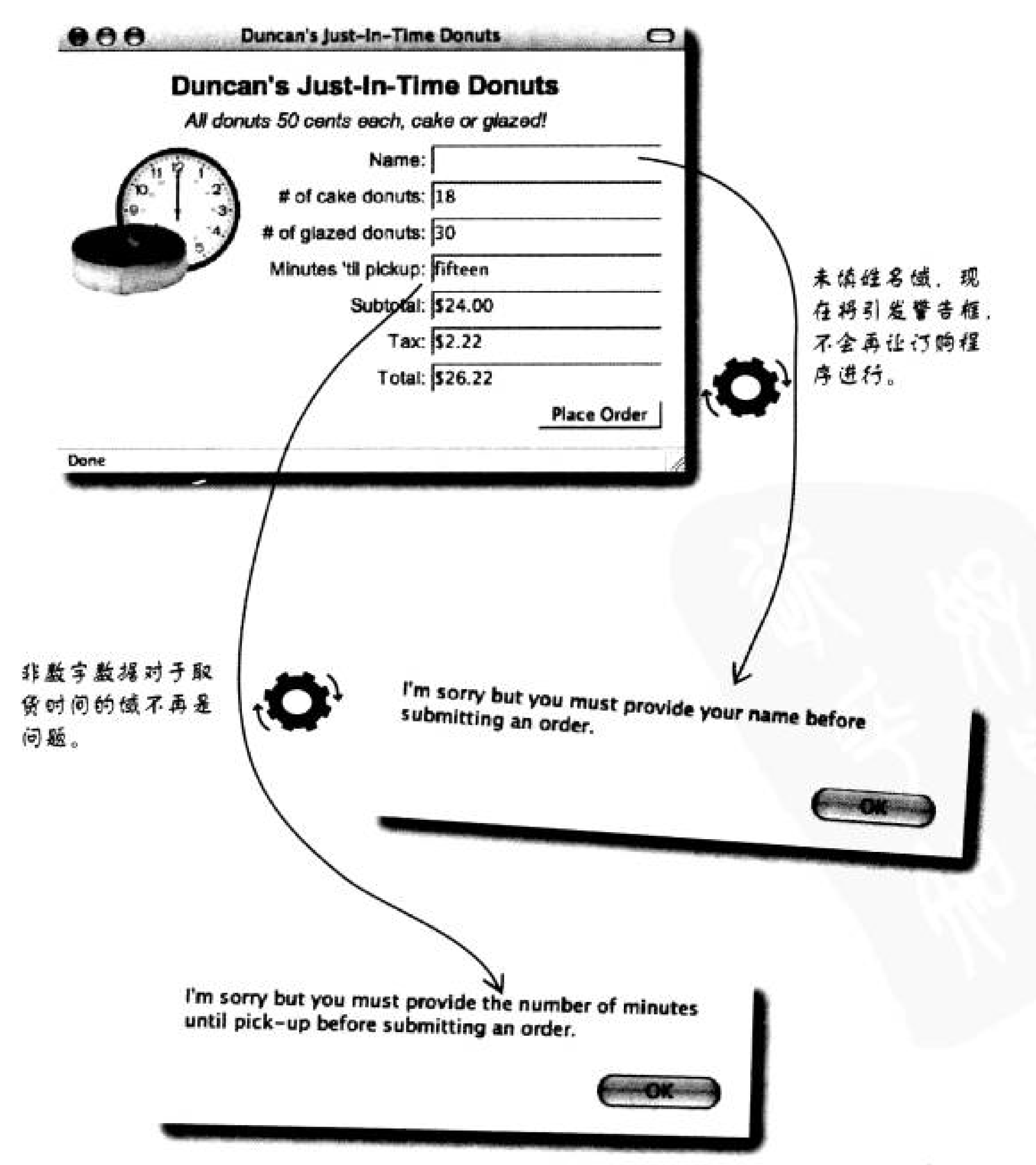
我们用 placeOrder() 函数验证姓名与取货时间。使用以下提供的冰箱磁铁，完成检查姓名与取货时间数据是否存在的程序代码，还要确认取货时间是否为数字。你将需要每个磁铁，有些磁铁甚至不只使用一次。




```
function placeOrder() {
  if (document.getElementById("name").value == "")
    alert("I'm sorry but you must provide your name before submitting an order.");
  else if (document.getElementById("pickupminutes").value == "" ||
    isNaN(document.getElementById("pickupminutes").value))
    alert("I'm sorry but you must provide the number of minutes until pick-up" +
      "before submitting an order.");
  else
    // Submit the order to the server
    form.submit();
}
```

你拯救了 Duncan's Donut……再一次！

经过你改良的全新即时甜甜圈订购表单（附数据验证），终结了 Frankie 的肮脏勾当，也让真正的客户能使用更可靠的网页。以 JavaScript 保护用户输入数据的健全性（integrity）是个双赢机制，尤其适用于竞争激烈的早餐市场里哦！！



没有蠢问题

问：加号（+）怎么知道该做相连或相加？

答：就像 JavaScript 里的许多事项，函数的功能需由上下文（context）决定。这表示加号看了要相加的东西后，才根据“数据类型”决定该做数值加法或字符串相连。你已经知道两个字词相加表示“首尾相连”。但问题可能发生在数据类型跟你的假设不一样的时候。所以，最好在数值加法前，先确认提供的运算对象都是数字，也确认字符串相连前都是提供文本。

问：如果试着做字符串与数字相加，会发生什么事？

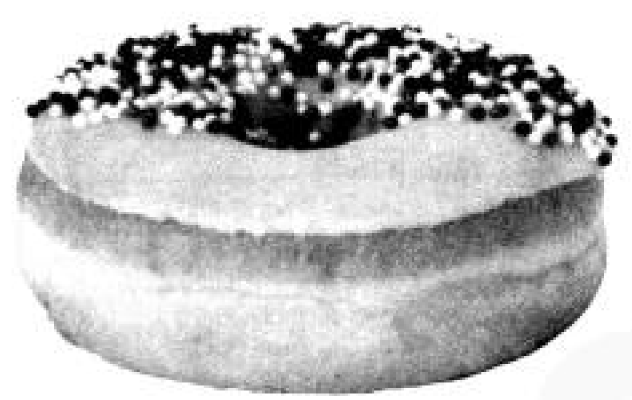
答：因为 JavaScript 会自动把数字转成字符串嘛……混合这两种类型的相加操作，必定造成字符串相连。所以说，数字先转换成字符串，系统再串连两个字符串。如果你原本想做数字加法，则需使用 `parseInt()`或`parseFloat()`，把字符串转换为数字。

问：如果使用`parseInt()`转换包含浮点数的字符串，会发生什么事？

答：别想太多，你的程序代码不会因此失事。JavaScript 只会假设你不在意小数点后的部分，因此仅返回小数点前的整数部分。

问：HTML 的`id`属性究竟如何联系网页元素与 JavaScript代码？

答：你可以把`id`属性当成一个入口，JavaScript代码利用这个入口访问HTML的内容。当别人说到JavaScript代码能在网页上运作时，其实并非真的在说网页本身——他们是指浏览器。实际上，JavaScript代码与HTML代码不怎么熟，必须透过非常特殊的机制才能访问。其中之一就是需要`id`属性，该属性能让 JavaScript访问 HTML 的元素。为网页元素加上附有 ID 的标签，JavaScript代码即可找到该元素，打开许多脚本编码（scripting）的可能性。



问：你讲得还真抽象。JavaScript代码到底如何访问HTML元素？请讲细节。

答：这个嘛……`document`对象的`getElementById()`方法，就是从JavaScript访问HTML元素的关键：这个方法则使用`id`属性来寻找网页上的某个元素。HTML 的 ID 就像 JavaScript的标识符——在某一个网页上，ID或标识符均应独一无二。否则，`getElementById()`方法将不知该返回哪个元素。

问：你说过我们会到第 9 章再谈对象，但对象已经出现过好几次了！它们是什么玩意？

答：你这是偷跑啊！小声点，别让其他人听见了。对象是种先进的 JavaScript数据类型，它能结合函数、常量与变量为一个逻辑实体（logical entity）。方法（method）其实是属于某对象的函数，特性（property）则是对象里的常量或变量。在实现层面上，JavaScript几乎使用对象表达一切 —— 浏览器窗口是个对象，网页文件也是个对象。所以`getElementById()`方法必须透过`document`对象才能调用 —— 因为它是某个表示整体网页的对象里的一部分。好了好了，赶快回来第 2 章……

问：我还是不明白网页元素和元素值的不同。到底有什么不同？

答：网页元素对 JavaScript 而言是种对象，也就是说，我们可以透过特性与方法操纵网页元素。这些对象都有个`value`特性，它保存了储藏在元素里的值。例如，表单域的 `value`，就是输入该域的数据。

问：为什么需要检查值是否“非”数字？检查是否“为”数字，不是更合理吗？

答：好问题。追根究底，这个问题其实在于检查值是否为数字。大多数假设状况下，你在处理数字，所以检查例外状况比较合理。借由检查 `NaN`，你用于处理数字的脚本会更可靠，希望也能减少计算非数字的怪异情况。

努力贴近用户靠直觉输入的内容

现在，Duncan 不再遇上紧急状况，他真的很想改善即时甜甜圈表单的用户体验。就像“热甜甜圈”的招牌，路过他店门口的行人看到这个招牌就会产生直觉影响，Duncan 也希望在线表单能拥有相似的直觉影响。Duncan 知道一般人喜欢以“打”为单位订购，也以相同单位提供。很少有人会订 12 或 24 个甜甜圈，他们会订“1 打”或“2 打”甜甜圈。他觉得甜甜圈表单的订购单位也该自然一点。

问题在于，目前的脚本并未考虑输入“dozen”一词的状况，程序并不知道这个词是种单位。

Duncan's Just-In-Time Donuts

Duncan's Just-In-Time Donuts

All donuts 50 cents each, cake or glazed!

Name: Dierdre

# of cake donuts: 3 dozen

# of glazed donuts:

Minutes 'til pickup: 60

Subtotal: $1.50

Tax: $0.14

Total: $1.64

Place Order

Done

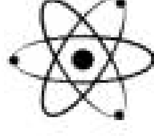
因为有parseInt()函数，“3 dozens”转换为数字3。

parseInt(3 dozen)

3

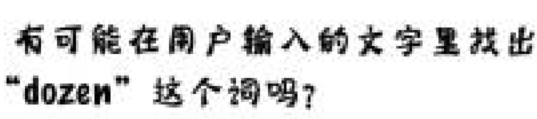
它是数字 3，不是字符串哦！

脚本不会抱怨用户在同一个域里输入了“dozen”和数字……parseInt() 直接忽视字符串里任何出现在数字后的文本。所以，“dozen”就这样被丢弃了，只有数字保存了下来。



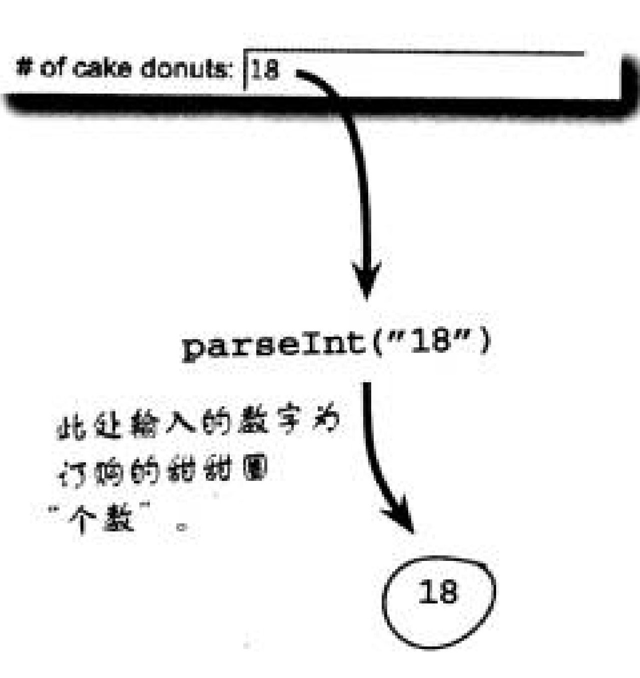
动动脑

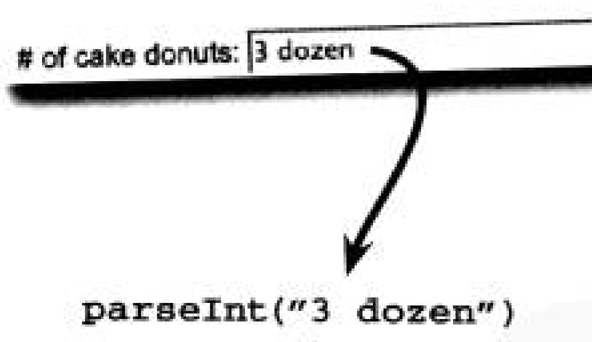
我们的甜甜圈脚本是否能允许用户输入数字，或输入数字与“dozen”（以“打”为订购单位）呢？该怎么做？

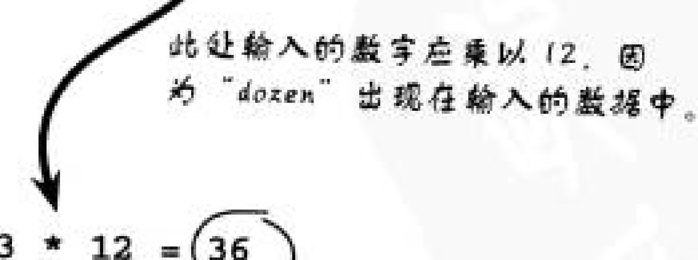


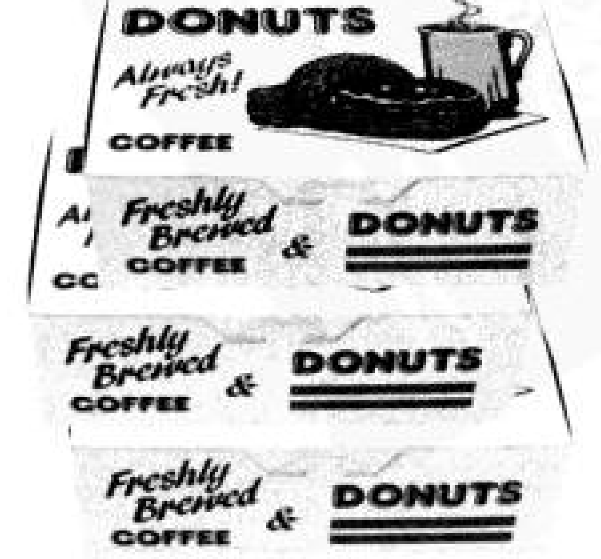
如果用户想要“dozen”，就乘以 12！

要以“打”为订购单位的选择，若在脚本里加入“小计”(subtotal) 前检查“dozen”一词的段落，就能提供。如果 (If) 出现了“dozen”，则把前面的数字乘以 12；如果没有，则把数字视为订购的甜甜圈“个数”。









我们自定义了parseDonuts()函数，负责处理甜甜圈的订购数量。函数先把数据转换为数字，然后检查“dozen”是否出现在输入的数据里。如果出现了，刚才得出的数字则乘以12。请上*http://www.headfirstlabs.com/books/hfjs/* 下载这段现成的程序代码。

```
function parseDonuts(donutString) {
  numDonuts = parseInt(donutString);
  if (donutString.indexOf("dozen") != -1)
    numDonuts *= 12;
  return numDonuts;
}
```

检查“dozen”是否出现在输入的数据里。

把甜甜圈的个数乘以12。

解析甜甜圈的数量

parseDonuts()函数被updateOrder()函数调用，也是根据用户输入的数据算出小计金额与总额的时候了。

两个常量的初始化。

从表单领取得甜甜圈的数量。

```
function updateOrder() {
  const TAXRATE = 0.0925;
  const DONUTPRICE = 0.50;
  var numCakeDonuts = parseDonuts(document.getElementById("cakedonuts").value);
  var numGlazedDonuts = parseDonuts(document.getElementById("glazeddonuts").value);
  if (isNaN(numCakeDonuts))
    numCakeDonuts = 0;
  if (isNaN(numGlazedDonuts))
    numGlazedDonuts = 0;
  var subTotal = (numCakeDonuts + numGlazedDonuts) * DONUTPRICE;
  var tax = subTotal * TAXRATE;
  var total = subTotal + tax;
  document.getElementById("subtotal").value = "$" + subTotal.toFixed(2);
  document.getElementById("tax").value = "$" + tax.toFixed(2);
  document.getElementById("total").value = "$" + total.toFixed(2);
}
```

如果输入的甜甜圈数量不为数字，则设为0。

算出小计金额、税额（tax）与总计。

在网页上列出金额。

把金额四舍五入至小数点后两位（以“美分”为单位）。

即时甜甜圈获得空前成功！

Duncan 的即时热甜甜圈构想，完全被一个由 JavaScript 驱动、小心验证用户输入数据的网页了解，生活真是太顺利了！

热爱甜甜圈的广大客户们，可以在线订购热甜甜圈，而且即时取货了。

Hot Donuts

OVER 1000000 SERVED

Hot Donuts
Just in time!

Duncan's Just-In-Time Donuts

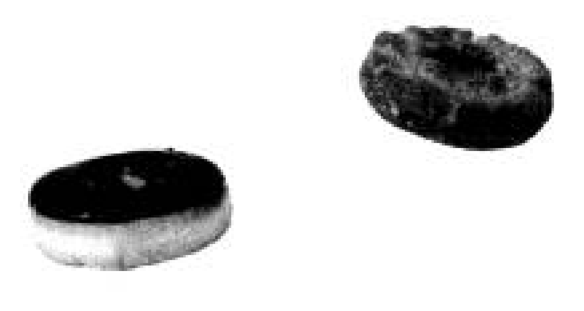
Duncan's Just-In-Time Donuts

All donuts 50 cents each, cake or glazed!

Name: Alan
# of cake donuts: 15
# of glazed donuts: 4 dozen
Minutes 'til pickup: 10
Subtotal: $31.50
Tax: $2.91
Total: $34.41

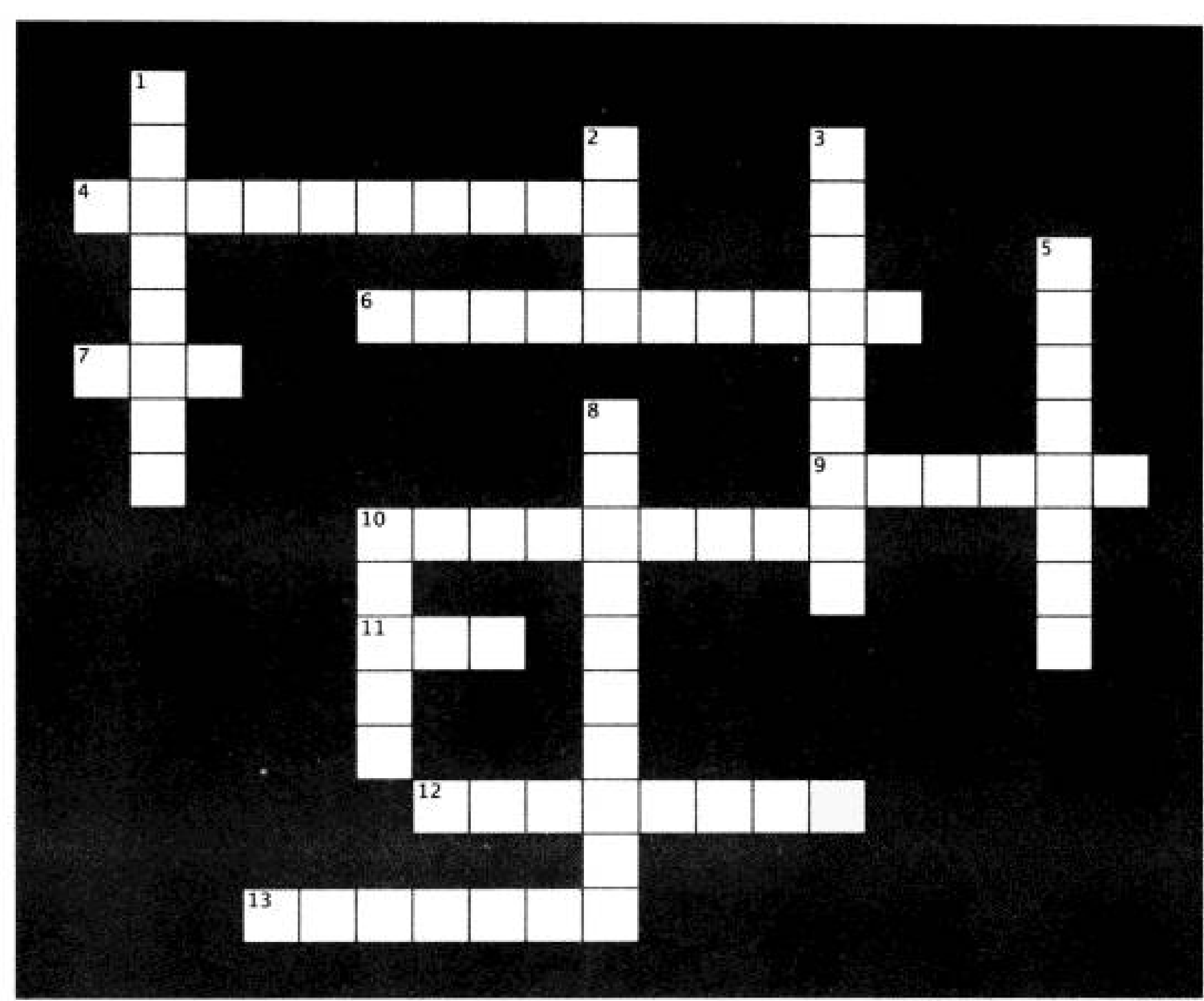
Place Order

Done



JavaScript 填字游戏

数据不见得都以 JavaScript 码储存。有时候，它们也会藏在填字游戏的字里行间，正等着各位发掘呢。

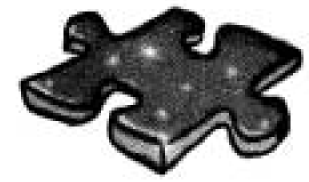


横向提示

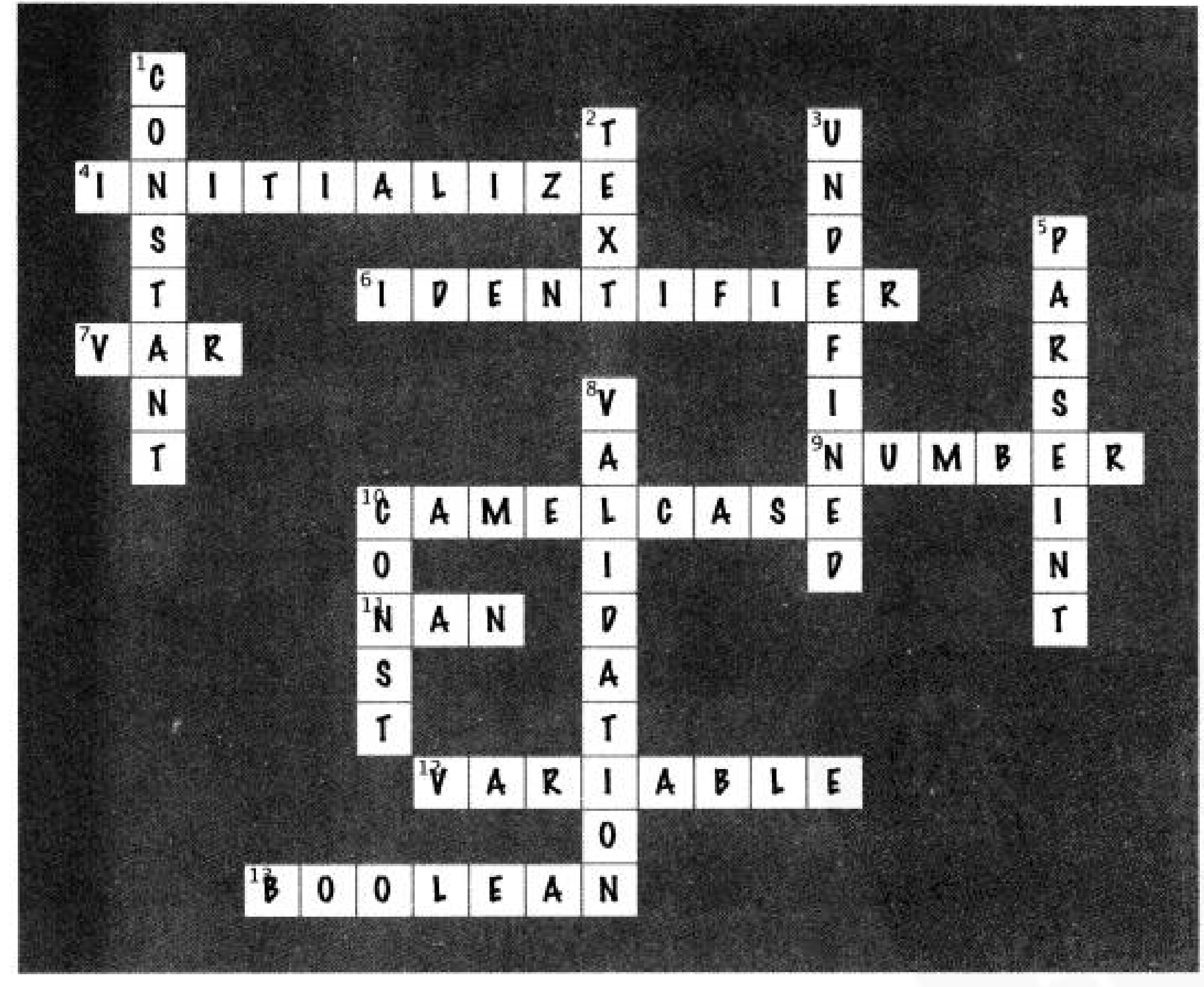
4. 当你在创建一段数据时，也指定它的值，就是________。
6. 用于提及一段数据的独特名称。
7. 用于创建变量的 JavaScript 关键字。
9. 3、14、11、5280 都属于这种数据类型。
10. 关于使用大小写命名标识符的编程惯例，例如 ThisIsMyName。
11. 非数字。
12. 值能改变的一段信息。
13. 具有“是/否”值的一段信息，将存储为这种数据类型。

纵向提示

1. 值不能改变的一段信息。
2. 用于存储字符、字词和短语的数据类型。
3. 当变量或常量未设定值时，这段数据被视为________
5. 用于转换字符串为整数的内置JavaScript函数。
8. 确认用户输入的数据正如其调用用途的检查过程，称为________
10. 用于创建常量的 JavaScript 关键字。



JavaScript 填字游戏解答



Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

我们想对脚本的数据要求什么？

这是左右脑的秘密会谈！

我对脚本的数据有许多要求，但有一项特别重要。

好吃！

Hot Donuts

OVER 1000000 SERVED

用户输入的数据，

是种你不该无条件信任的数据。

假设用户会输入数据，还会检查数据是否正确，

完全是种不安全的假设。更安全的存储方案，

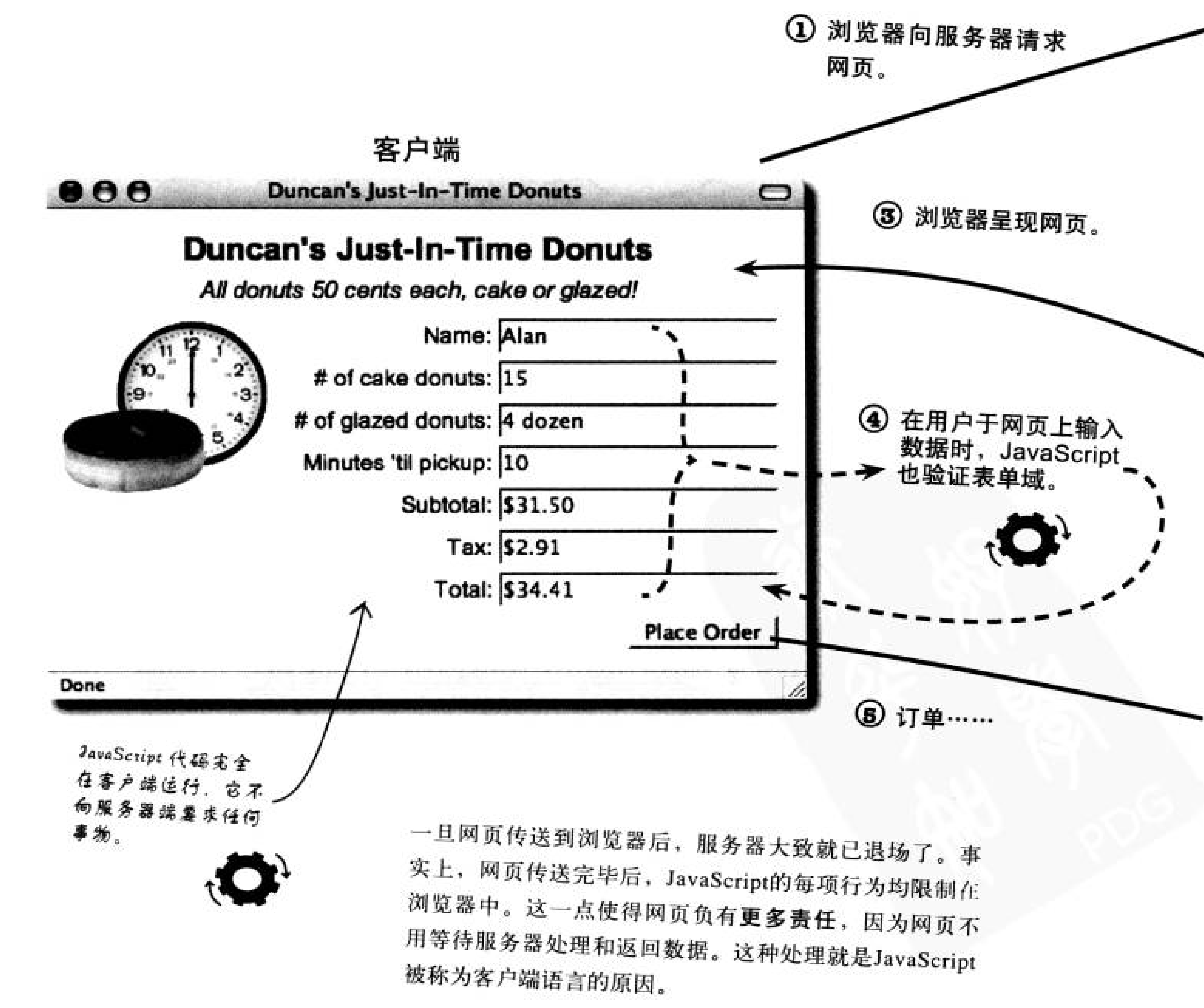
将需要JavaScript。

3 探索客户端

有时候，JavaScript 需要知道外在的真实世界如何运作。刚开始，你的脚本或许只是网页里的代码，但最后终将存活在浏览器（或称客户端）打造的世界里。**聪明的脚本**通常需要多了解外在的世界，才能与**浏览器沟通**，才能更加了解外在世界。如果和浏览器打好关系，无论是找出屏幕尺寸还是访问浏览器的打盹按钮（snooze button），脚本能利用的工具可多呢！

客户端、服务器端、JavaScript

当你点击超链接或在浏览器里输入一段URL，浏览器都会向某个网站服务器请求网页。直到浏览器呈现网页的前一刻，JavaScript代码才加入动作行列，负责与浏览器合作响应**用户的交互**，并于需要时**调整网页画面**。浏览器里运行 JavaScript 代码的部分称为JavaScript解释器。



一旦网页传送到浏览器后，服务器大致就已退场了。事实上，网页传送完毕后，JavaScript的每项行为均限制在浏览器中。这一点使得网页负有**更多责任**，因为网页不用等待服务器处理和返回数据。这种处理就是JavaScript被称为客户端语言的原因。

请求网页

```
GET / HTTP/1.1
Host: www.duncansdonuts.com
Connection: close
Accept-Encoding: gzip
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ...
Accept-Language: en-us
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X; en-US; rv:1.8.1.7) ...
```

HTML 告知网页上的内容。

提供网页

```
<html>
  <head>
    <title>Duncan's Just-In-Time Donuts</title>
    <link rel="stylesheet" type="text/css" href="donuts.css" />
    <script type="text/javascript">
      function updateOrder() {
        ...
      }

      function parseDonuts(donutString) {
        ...
      }

      function placeOrder() {
        ...
      }
    </script>
  </head>

  <body>
    <div id="frame">
      <div class="heading">Duncan's Just-In-Time Donuts</div>
      <div class="subheading">All donuts 50 cents each, cake or glazed!</div>
      <div id="left">
        <img src="donuttime.png" alt="Just-In-Time Donuts" />
      </div>
      <div id="right">
        ...
      </div>
    </div>
  </body>
</html>
```

CSS 让网页看来漂亮。

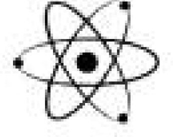
JavaScript 让网页有交互性，本例为验证用户输入的数据。

服务器端

② 服务器以网页提供响应。

⑤ ……送交至……

⑤ ……服务器。



动动脑

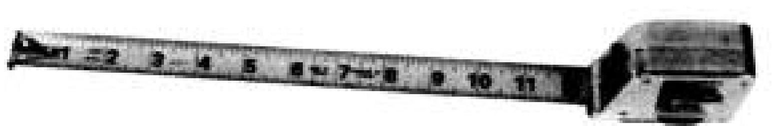
你能想到其他适合由客户端而非服务器端负责的任务吗？

先问浏览器能为你做什么

你的客户端网络浏览器负责执行JavaScript代码，脚本因此得以访问客户端环境。例如：取得浏览器窗口的宽度与高度、类似闹钟的计时机制、对cookie的访问……这些有趣的浏览器功能都将对JavaScript敞开大门，因此能在离开网页甚至关掉浏览器后存储数据。

浏览器度量单位

浏览器的度量单位，包括各种关于“尺寸”的测量，像浏览器窗口、可视网页范围，甚至还有制造商与版本编号等信息。



浏览器历史记录

浏览器的历史记录，即最新参观过的网页。你可以使用JavaScript访问网页清单，并把浏览器导向其中一页，有效率地创建自己的浏览器导览控制。

定时器

定时器可以在特定时间过去后触发一段JavaScript代码。

Cookie

cookie就像变量，浏览器把它存储在用户的硬盘里，存活期间超过单一web session（会话）。换句话说，你可以放着某个网页稍后再回来，数据仍然等着你。

客户端能为脚本服务的功能不只这些，但你应该看得出来，JavaScript的用途远大于控制网页中的元素。事实上，若能**看得更远、超越网页**并取得浏览器的一点帮助将对很多情况大有帮助。

问：所以说，JavaScript 是客户端程序的一部分啰？

答：没错。支持JavaScript的网络浏览器附有JavaScript解释器，解释器负责在网页上解读JavaScript代码，然后运行。

问：如果 JavaScript 代码在客户机运行，它为什么跟服务器有关？

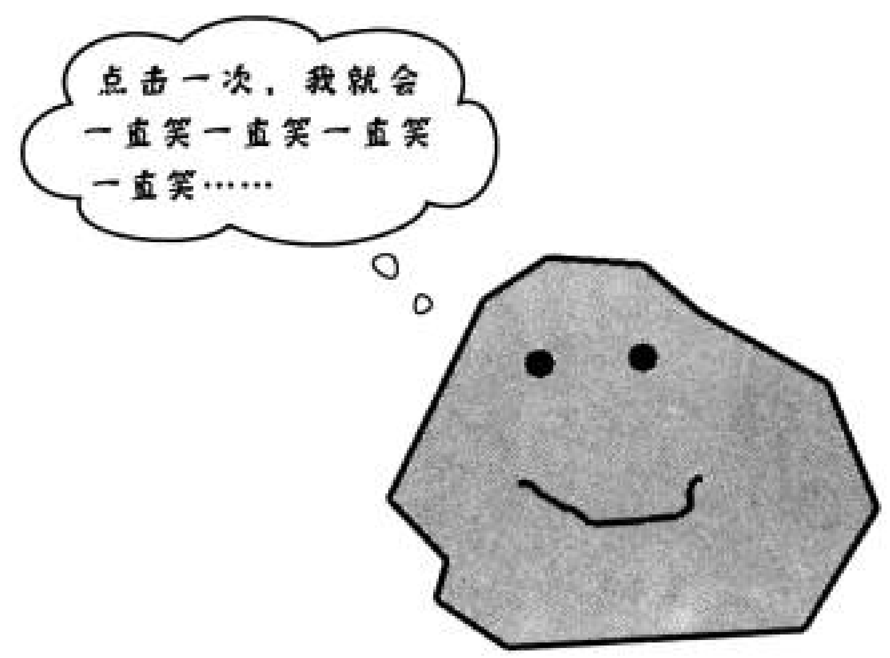
答：JavaScript 多半不会直接与网络服务器有关系，因为它在客户机上独立运行。JavaScript 通常用于拦截从服务器传递到浏览器的网络数据；但也可以编写向网站请求信息的脚本，然后于网页上呈现信息。这种脚本技术称为 Ajax，我们会在第12章讨论如何使用 Ajax。

问：JavaScript 能让我们控制客户端吗？

答：可以……但又不行。虽然网络浏览器允许 JavaScript 访问客户端环境的特定部分，但为了安全考量，浏览器不会放任 JavaScript 拥有无限的自由。例如在没有用户的同意时，大多数浏览器都不准脚本擅自开启或关闭。

iRock 乐过头了

还记得 iRock 吗？你写的 JavaScript 代码实在非常成功，有位年轻的创业家 Alan 买下了 iRock。有一天，他找你到办公室喝咖啡……因为用户对于 iRock **恒久不变的笑容**感到毛骨悚然。没错，我们都希望宠物开心快乐，但 iRock 的情绪范围似乎极度有限。



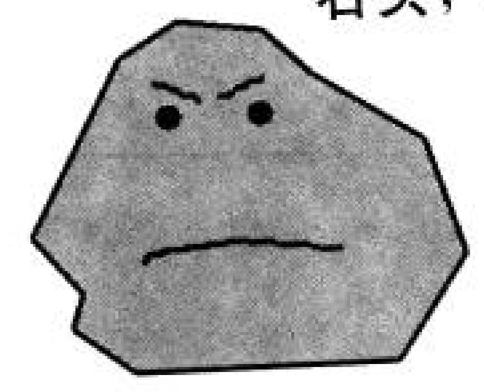
我喜欢一直都很高兴的宠物，不过用户希望多一点真实性。所以我在想，你需要重新看一下源代码…………

iRock 的新问题与**用户的期望**有关。虚拟宠物的概念在于应该尽可能接近真实的宠物。你的挑战就是找出如何**改善iRock的行为**，让宠物石更真实。客户端的网络浏览器或许握有解决这个问题的线索……

iRock 需要更有反应

让我们一起为iRock集思广益，想想一些可能会用到的行为，让宠物更具**真实性**，与用户更为**亲密**，当然也要更有**交互性**。理想中，iRock增加情绪范围后，应该更能**响应**用户。

请到 http://www.headfirstlabs.com/books/hfjs 下载 iRock 2.0 的最后一段程序代码。

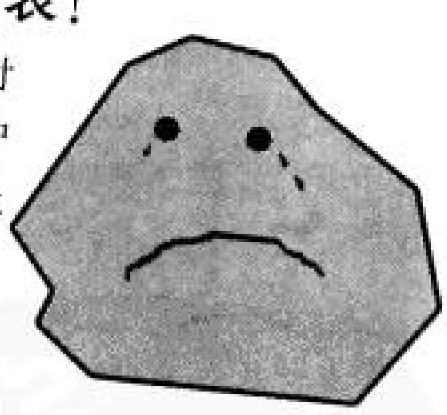


石头，怒！

iRock 将随机、毫无理由地生气，用户必须想办法安抚。

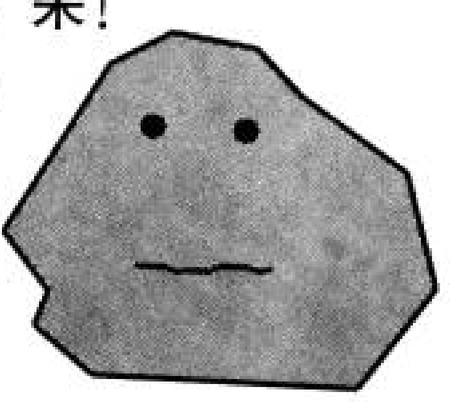
石头，哀！

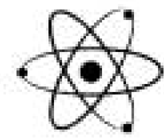
iRock 在你关闭网页时露出哭脸，并要求用户继续开着浏览器，别让宠物石情绪低落。



石头，呆！

如果太久没玩 iRock，它将回到发呆般的表情，用户可以不定时点击一下，唤起宠物石的注意力。



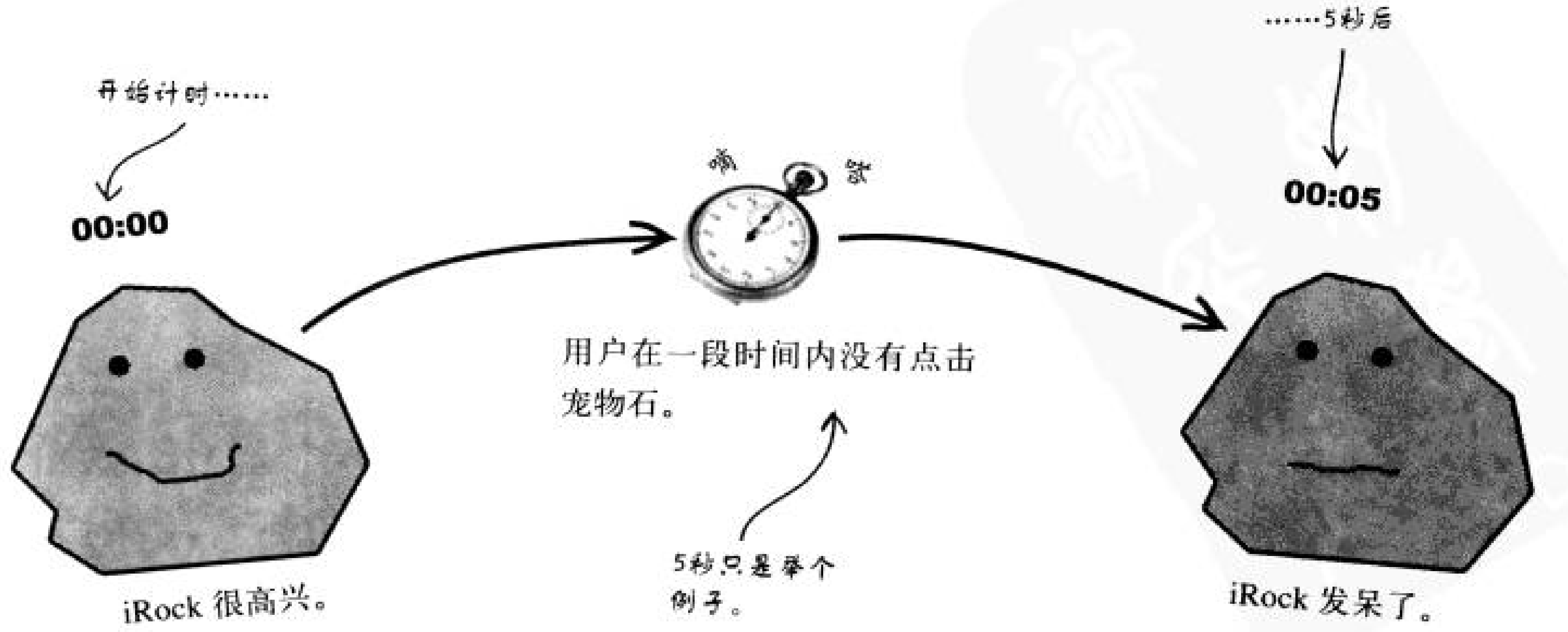


动动脑

iRock 该具备哪些行为才合理？你会如何利用JavaScript 在 iRock的脚本里实现这些行为？

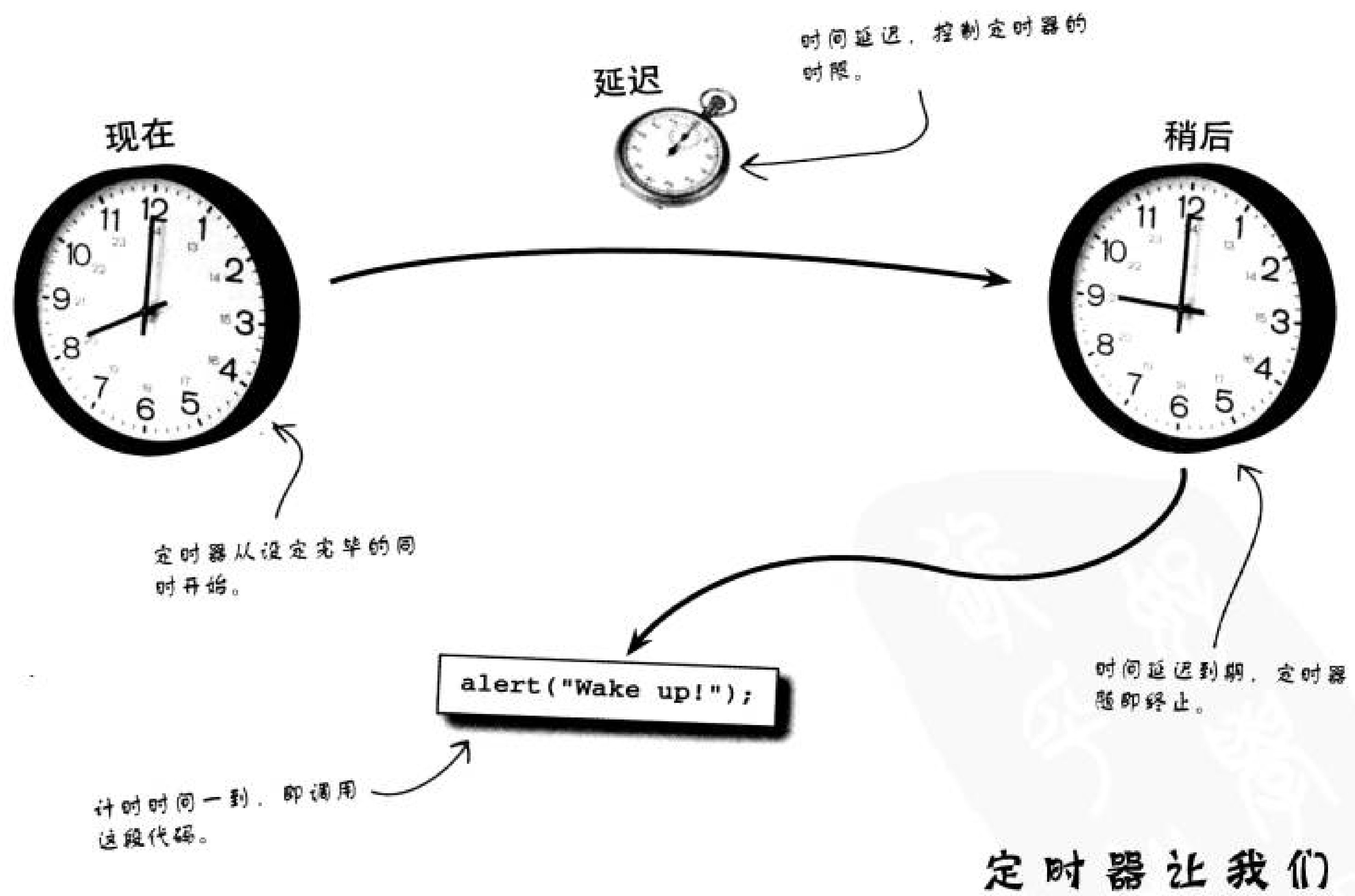
JavaScript让你知道用户做了“某事”或没做“某事”的时间点。

“宠物石开始发呆”的想法十分有趣，想必能刺激用户与宠物石互动，又不会增加心理负担，还能为**用户**带来iRock**积极响应的奖励**。你面对的挑战就是：以时间为根据，利用JavaScript改变iRock的呈现图像。我们预计等待一段时间，如果用户一直没有点击iRock，则改变iRock的状态。



定时器连接了行动与时间延迟

JavaScript能设定**定时器**（timer）。JavaScript定时器的工作方式很像闹钟：先设定闹钟醒来的时间；设定的时间来临，即触发某段代码或某个组件。但是，JavaScript定时器并非**在指定时间点**触发程序代码，而是**在一段时间流逝后**，才触发程序代码。这一点都不是问题，你只要改成思考**时间延迟**（time delay）的问题（而非注意时间是否为标准时间十二点零分）。



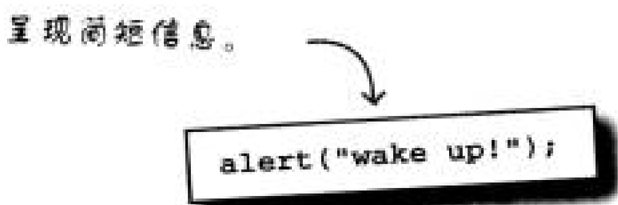
定时器让我们能在一段时限流逝后，才运行JavaScript代码。

JavaScript 定时器让我们在计时终止时，几乎可运行任何程序代码，厉害吧！有些定期改变数据的网页正是利用定时器于一段时间延迟后自我更新，有些网页则会定期检测用户是否与网页交互。

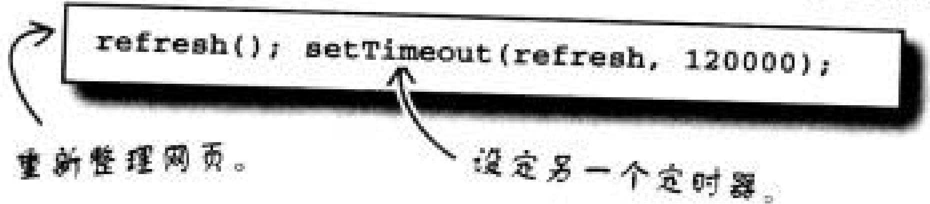
拆解定时器

在 JavaScript 里设定定时器有两个关键：1) 建立**时间延迟**（time delay）；2) 让定时器知道时限来临时**该运行的程序代码**。从你设定完成的那一刻起，定时器马上开始走动。

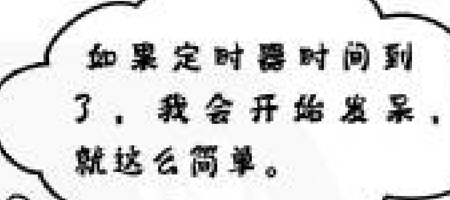
时间延迟以**毫秒**（millisecond）即千分之一秒表示。把理想的延迟秒数乘上 1,000，就是你需要的毫秒数。例如 2 秒等于 2,000 毫秒。

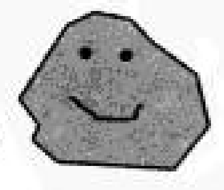


定时器终止时运行的程序代码，可以是任何你想用的 JavaScript 代码，例如一行语句（statement）、许多语句（每句均以分号结尾），甚至调用自定义或内置的函数。



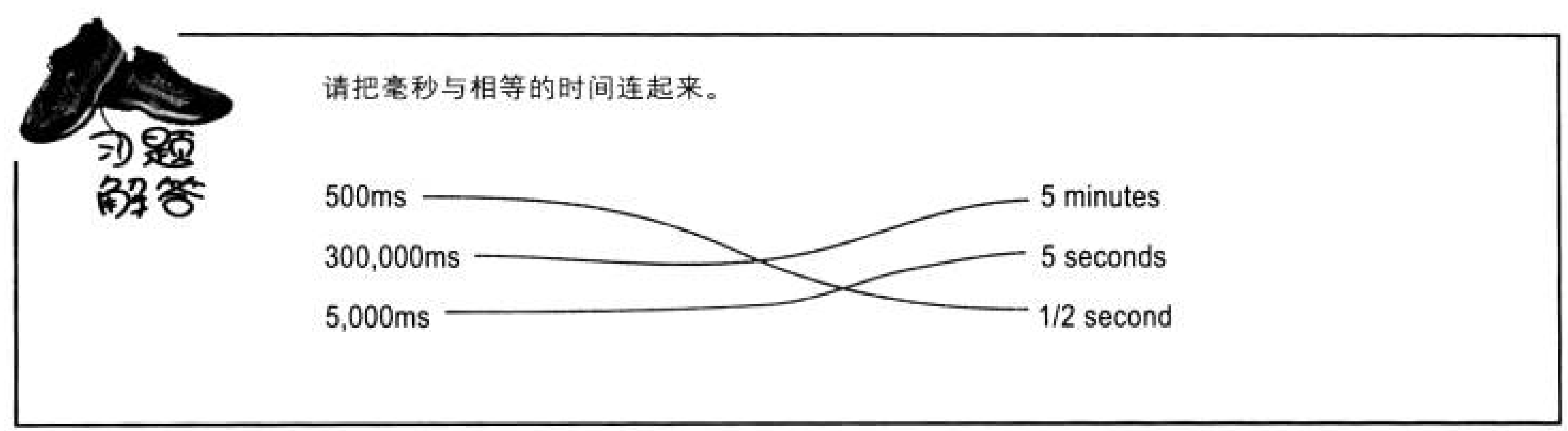
普通的 JavaScript 定时器在终止并运行指定程序代码后，定时器就完了、结束了。这种定时器称为**单次定时器**（one-shot timer），顾名思义，它只会触发**一次**程序代码。另外亦可建立**间隔定时器**（interval timer），设定多段间隔，而不只设定一次时间延迟。间隔定时器每隔一段时间，即**重复**调用"定时器代码"，直到我们让它停止。虽然间隔定时器一定有它的舞台，但 iRock 显然需要单次定时器。





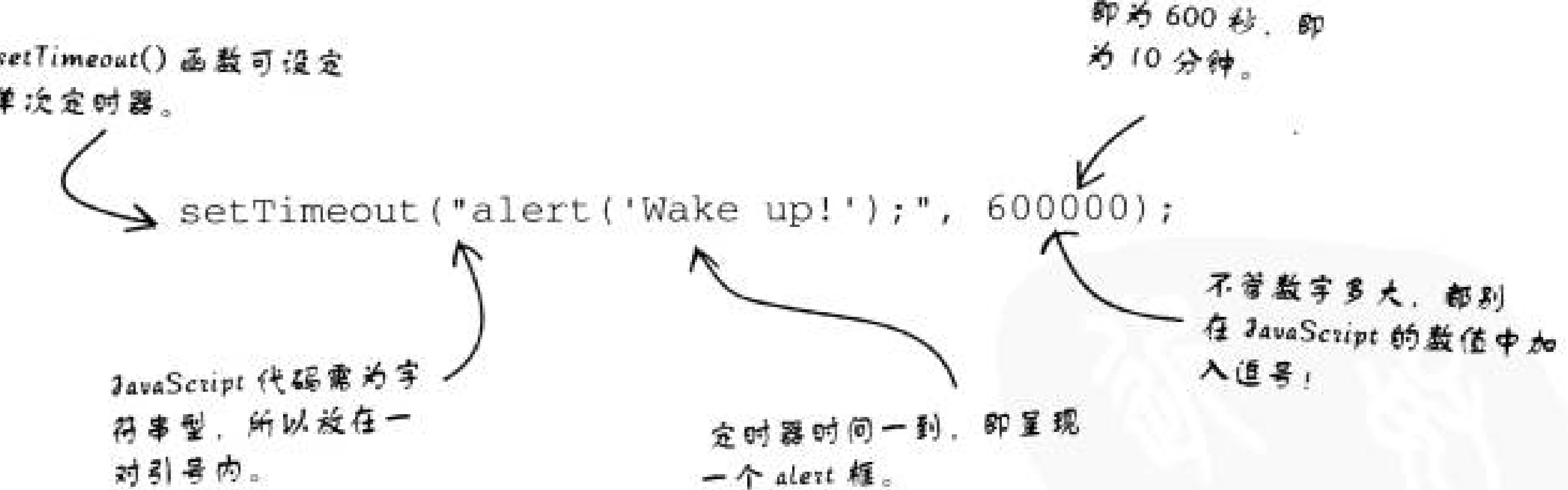
请把毫秒与相等的时间连起来。

500ms	5 minutes
300,000ms	5 seconds
5,000ms	1/2 second

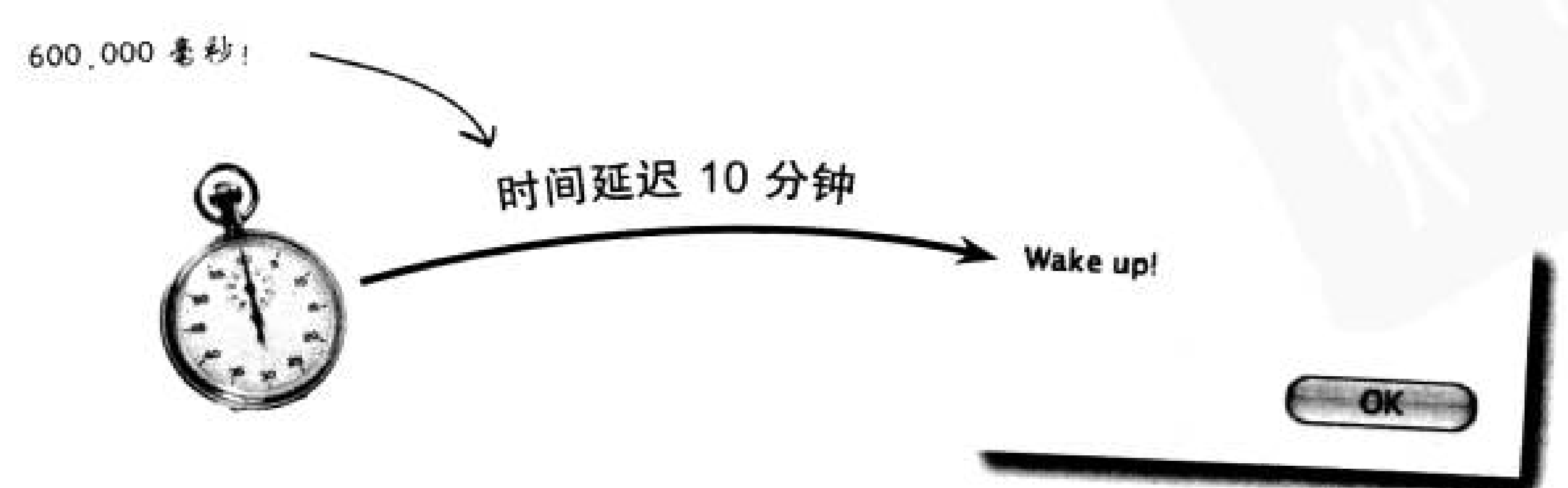


设定时间——setTimeout()

制作（单次）定时器的 JavaScript 内置函数被称为`setTimeout()`。函数需要的两项信息，分别是**时间延迟**（time delay）与定时器停止时**欲运行的程序代码**（请至 *http://www.headfirstlabs.com/hfjs* 下载）。提供这两项信息时没有顺序问题。如下例所示：

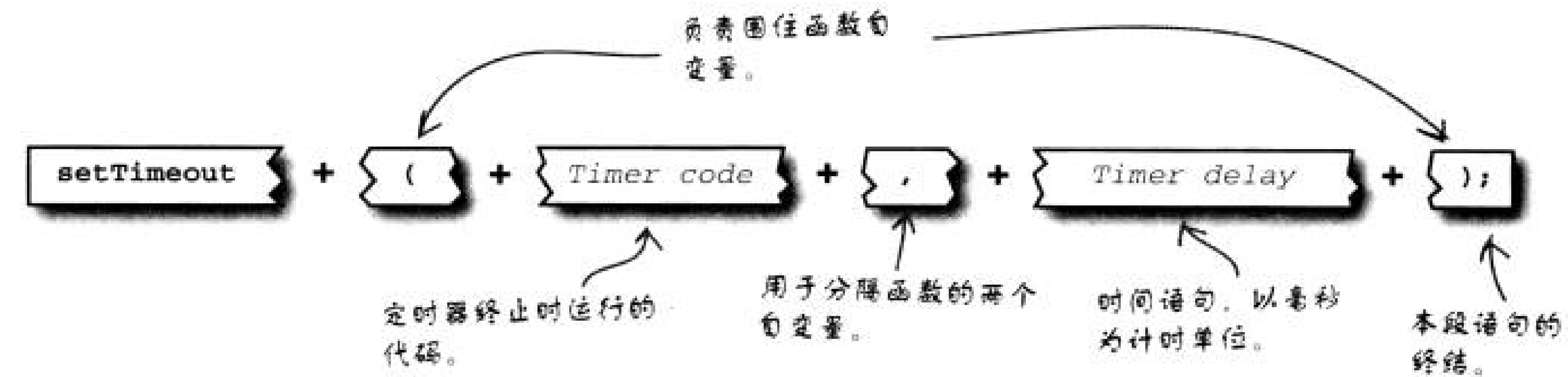


上例调用 `setTimeout()` 函数创建一个等待 10 分钟，而后弹出 alert 框的定时器。



再靠近一点：setTimeout()函数

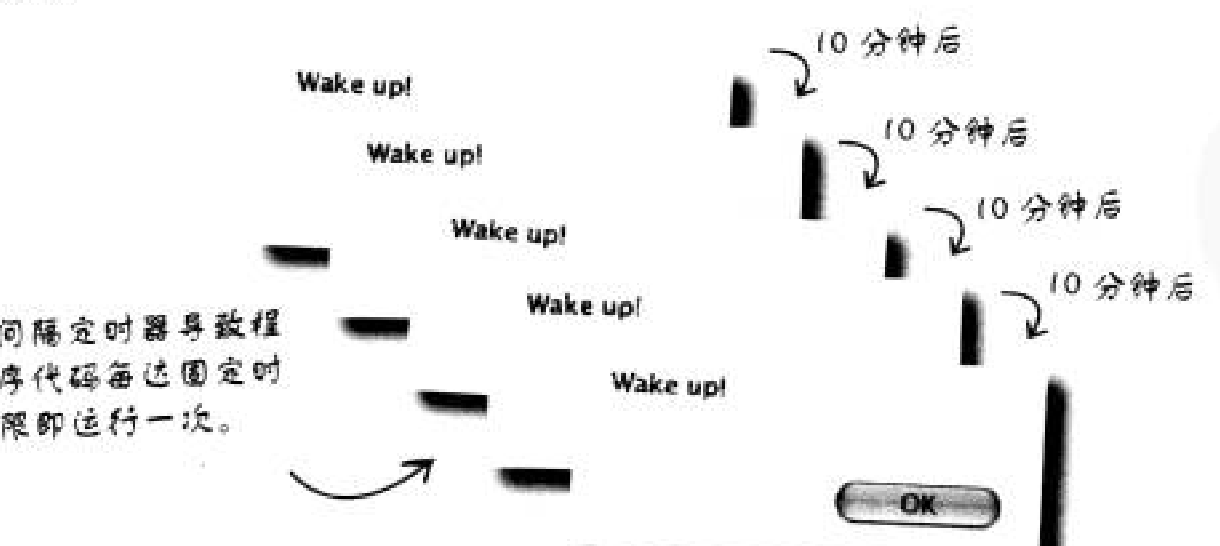
这是 setTimeout()函数的一般形式：



间隔定时器的设定大致与单次定时器的设计相去不远，只不过改为调用 setInterval()函数（单次定时器需调用setTimeout()）。设定间隔定时器，即可于每次间隔延迟到期时**反复**运行程序代码：

```
var timerID = setInterval("alert('Wake up!');", 600000);
```

另行存储定时器的ID。　设定间隔定时器。　以毫秒表示时间长度。



间隔定时器导致程序代码每达固定时限即运行一次。

注意！

延迟时间必以毫秒设定。

毫秒只是千分之一秒，如果你忘记该以毫秒为单位，定时器到期的时间将短得很夸张。

磨笔上阵

请动笔设计改变 iRock 图像的程序代码：让宠物石等待 5 分钟后，就从笑脸换成发呆的表情。提示：iRock 图像元素的 ID 是 rockImg，发呆图像的文件名是 *rock.png*。*

在翻页前，请把你设计的代码输入 irock.html，看看你的成果是否可用。

....

....

* 从http://www.headfirstlabs.com/books/hfjs 下载文件。

磨笔上阵 解答

请动笔设计改变 iRock 图像的程序码：让宠物石等待 5 分钟后，就从笑脸换成发呆的表情。提示：iRock 图像元素的 ID 是 rockImg，发呆图像的文件名是 *rock.png*。*

```
setTimeout("document.getElementById('rockImg').src = 'rock.png';",
 5 * 60 * 1000);
```

交替使用双引号和单引号，为函数制造适当的嵌套层次。

延迟时间为5分钟。乘上60，换算为秒数，再乘上1000，换算为毫秒数。

在 img 元素的 src 属性里设定新图像文件，即可改变 iRock 的图像。

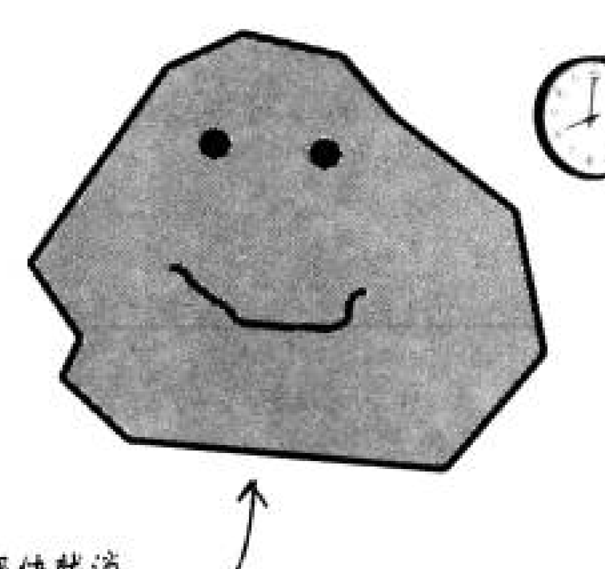
图像元素的 ID

发呆中的宠物石图像。

iRock开始发呆了

请确认你已经修改过 irock.html（细节如解答所示），再测试一下 iRock 的行动吧！现在，iRock 独处 5 分钟后（其间用户没有点击），就会进入发呆状态。这种时间延迟或许让宠物看来有点需要人类的疼爱，不过我们就是希望抓住用户的注意力嘛！再说，需要主人关心的宠物，才算得上是宠物啊……

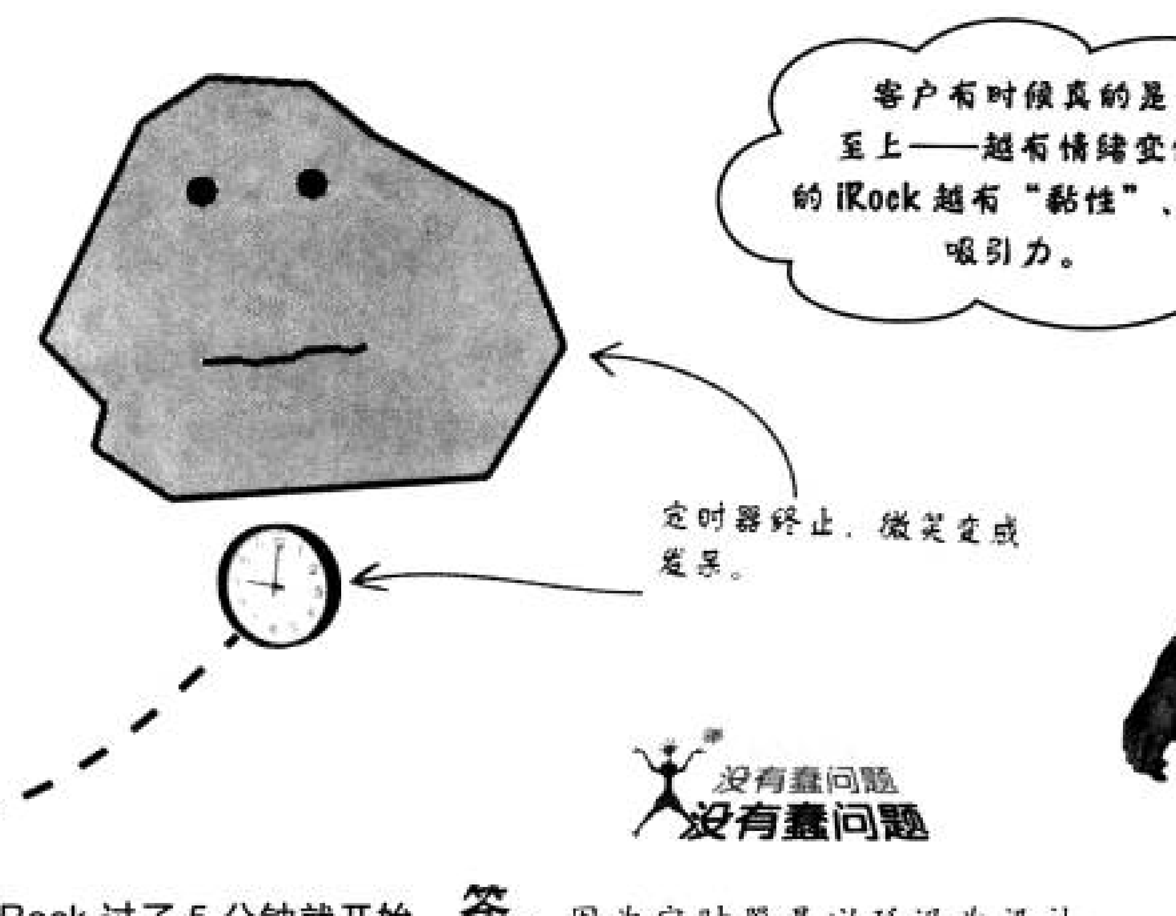
有个定时器真的在为 iRock 的"微笑时间"倒数计时。



iRock 的微笑很快就消失了，这点显得真实许多。

技客新知

调用setTimeout()函数时，若采用较短的延迟期限，就能加快iRoc的情绪改变速度。适合快速测试脚本，又不用每次真的都要等个 5 分钟。



没有蠢问题

问：如果 iRock 过了 5 分钟就开始发呆，为什么不用间隔定时器？

答：答案与单次定时器的使用方式有关。虽然宠物石每隔一段时间就要发呆，但只在原本状态是“微笑”时，才开始计算这个时间。点击宠物石的同时，定时器开始倒数，经过5分钟后，改为“发呆”状态，且直到下次点击，都维持在发呆状态。听起来不像间隔定时器的工作方式——间隔定时器每隔 5 分钟就会触发，与用户的行为无关。

问：如果用户在定时器终止前就关闭了浏览器呢，会发生什么事？

答：没事。JavaScript 解释器随着浏览器一同关闭，所有JavaScript代码也停止运行，包括任何未结束的定时器。

问：我该如何创建一个每天在固定时间启动的定时器？

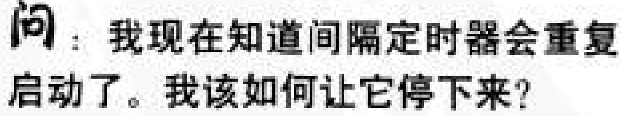
答：因为定时器是以延迟为设计基础，而不是根据确切的时间，我们必须把时间转换为时间延迟，所以必须从预计的启动时间里减去现在的时间。这个计算需要向 JavaScript 的 `Date` 对象借用力量，我们会在第 9 章仔细认识它。

问：我有个数据会改变的网页需要每隔 15 分钟更新一次，该怎么做呢？

答：请使用 `setInterval()` 函数，设定一个每隔 15 分钟启动一次的间隔定时器，15 分钟为900,000毫秒（15 × 60 × 1000）。你需要定时器帮忙更新页面，可调用`location`对象的`reload()`方法为你服务，如下所示：

```
location.reload();
```

这个定时器每隔 15 分钟就会触发网页的更新。当然，你也能利用 Ajax（第 12 章）动态载入数据，而不是重新载入整个网页。



问：我现在知道间隔定时器会重复启动了。我该如何让它停下来？

答：`clearInterval()`，这个函数用于清除`setInterval()`设置的间隔定时器。`clearInterval()`函数需要我们传入间隔定时器的ID，ID则于创建定时器的同时，由`setInterval()`返回。没错，函数也可以返回信息。存储了`setInterval()`返回的值（例如存到变量`timerID`），再把这个值传给`clearInterval()`，就能清除定时器了，如下所示：

```
clearInterval(timerID);
```

JavaScript真情指数——浏览器篇

本周主题：网络客户端的告白

Head First：先谢谢你在百忙中抽空接受访问。

Browser（浏览器）：真的够忙！要搞定 HTML与CSS的大牌脾气，以及它们造成的网页排版问题，原本就已经让我忙到分身乏术……现在又加入了新角色JavaScript，它完全是另一个空间的动物。

Head First：咦？为什么这样说？难道JavaScript习性狂野，很难管教吗？

Browser：呃，不是啦。"动物"只是种比喻。我的意思是：JavaScript带来一套特有问题，我一定要另外处理。我现在多了看懂JavaScript代码的新工作，祈祷这些脚本千万不要撰写得太糟糕，在运行JavaScript代码的同时，还要盯着HTML与CSS的表现。

Head First：听来真是辛苦啊。它们相处得好不好呢？

Browser：幸好，它们的相处是众多问题中最不严重的一项。它们三个常常一起工作，不过 JavaScript 偶尔会有点恶作剧，把 HTML 的外表弄得面目全非。真正的问题在于我无力阻止这些恶作剧，我只是个办事跑腿的小角色而已。

Head First：你的角色只能唯命是从吗？

Browser：这也是一种说法啦……更精确地说，我极度重视一致性（consistency）。我非常地照本宣科，一切任务就是接收服务器给我的程序代码，并依照上面的指示工作。

Head First：就算你知道指示有问题，还是照做不误吗？

Browser：我会尽已所能解决发现的问题，不过这项任候很艰巨。再说，这是下一次的主题（第11章）。我们今天不是该讨论"网络客户端"的角色吗？

Head First：说得对，我离题了，回题是岸。身为"客户端"又是什么意思呢？

Browser：嗯，客户端主要是指我位于网页传递管道的"接收端"，在发出对网页的请求后，等着服务器把网页传给我。

Head First：这项工作跟 JavaScript 有什么关系吗？

Browser：岂止有关系，大有关系咧！在我辛苦地呈现网页排版并处理用户输入的数据时，JavaScript就跟在我的背后，偷偷伸手乱动版面。但JavaScript也有好的一面，它带来很多我一个人根本办不到的魔术。

Head First：例如？

Browser：以前，用户的鼠标光标移过图片或调整浏览器窗口时，我都不敢乱想加上什么特殊举动。但JavaScript则不然，它就是为了"特殊举动"而生。以脚本改变网页外观，或基于客户端的改变而重新调整网页内容，这些对JavaScript 而言都是家常便饭。不过我可以接受，因为 JavaScript 代码仍以网页为基础，只会影响特定网页或网站。

Head First：你讲得 JavaScript 好像是另一个人。这世上真的另有 JavaScript，还是一切都是你？

Browser：两种答案都对。JavaScript确实是我的一部分，但你可以想成它有自己的实体，因为它能透过有限的接口访问客户端——就是我。我不会赋予JavaScript访问"我的一切"的无限权力，这样就太不负责了，毕竟我已经没办法控制谁会设计脚本叫我运行了。

Head First：原来如此。谢谢你过来说明客户端的相关疑问。

Browser：很高兴能帮上忙。

多变的屏幕尺寸，永不消失的申诉电话

Alan 刚把“为 iRock 增添情绪”的设计费付给你，马上又涌现一波 iRock 用户的夺命连环call。这次的问题似乎出在 iRock 的大小非常不一致上，有些用户反馈出现“缩小石综合病”，但有些用户却严重害怕“只看得到小部分结构的巨大石头”。Alan 最信任的人就是你，所以，~~赚钱的机会又来了！~~ 再度为他调整 JavaScript 吧！

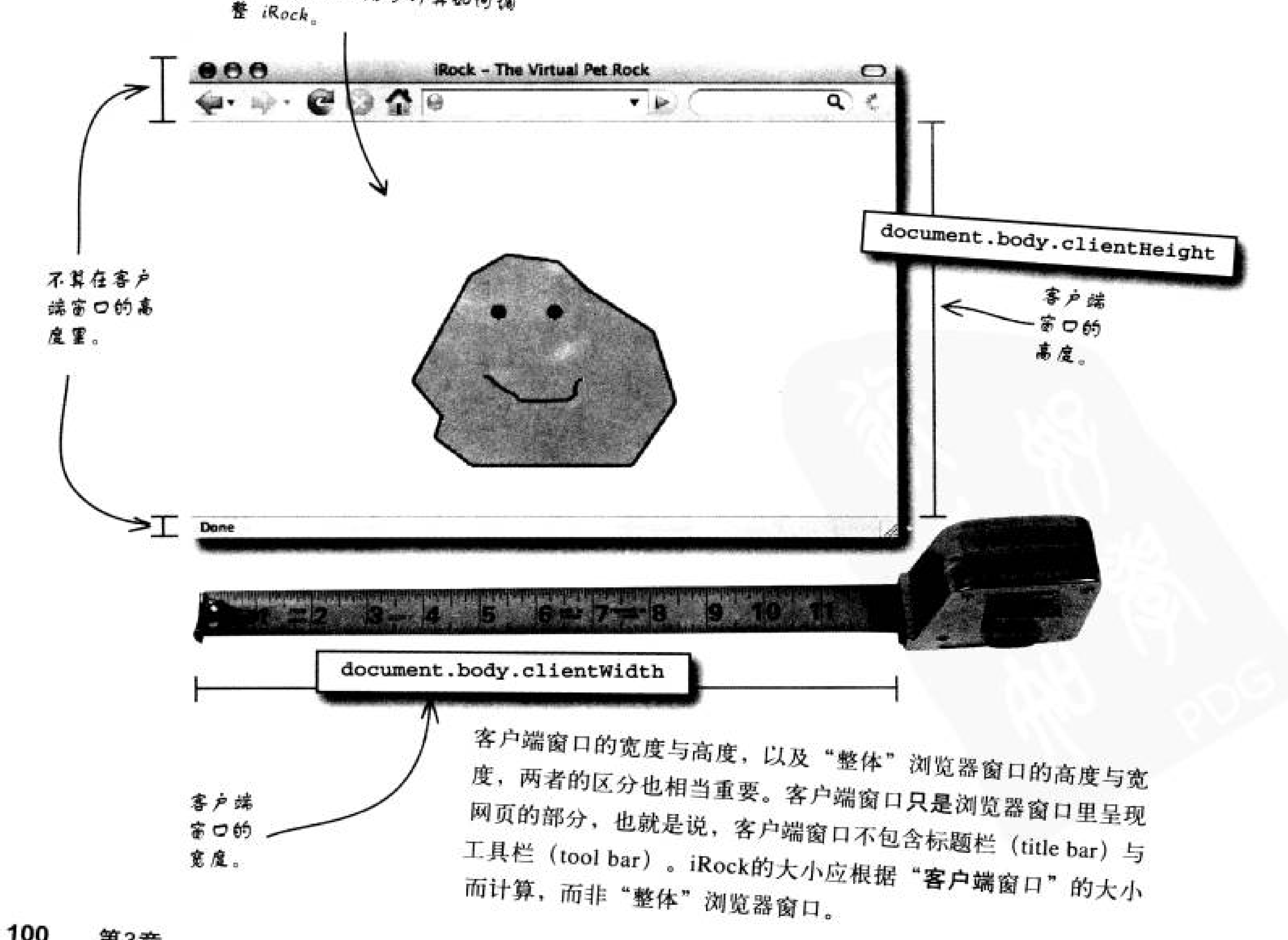
动动脑

为什么在不同浏览器上，会看到不一样大小的 iRock？

以document对象取得客户端窗口的宽度

iRock 的问题，想必在于 iRock 的大小并未随着浏览器窗口的大小而改变。听起来不像是个问题，除非你开始计算在**所有**可以浏览万维网的设备上，一共拥有多少种**尺寸差异剧烈**的浏览器……包括**娇小的手持设备**与桌上型计算机的**巨大屏幕**。你需要检查浏览器窗口尺寸的方式，同样的方式稍后可用于调整宠物石图像的大小。

客户端窗口只是浏览器窗口里呈现网页的部分。



客户端窗口的宽度与高度，以及“整体”浏览器窗口的高度与宽度，两者的区分也相当重要。客户端窗口**只是**浏览器窗口里呈现网页的部分，也就是说，客户端窗口不包含标题栏（title bar）与工具栏（tool bar）。iRock的大小应根据“**客户端窗口**”的大小而计算，而非“整体”浏览器窗口。

以document对象的特性设置客户端窗口的宽度

客户端窗口的大小需与网页协调，JavaScript可透过document对象取得这项信息。利用同一个对象的getElementById()方法，则可取得网页元素。document对象的body.clientWidth与body.clientHeight特性里存储了客户端窗口的宽度与高度。

document

document 对象代表网页本身。

```
<html>
  <head>
    <title>iRock - The Virtual Pet Rock</title>
    <script type="text/javascript">
      var userName;
      function greetUser() {
        alert('Hello, I am your pet rock.');
      }
      function touchRock() {
        if (userName) {
          alert("I like the attention, " + userName + ". Thank you.");
        }
        else {
          userName = prompt("What is your name?", "Enter your name here.");
          if (userName)
            alert("It is good to meet you, " + userName + ".");
        }
        document.getElementById("rockImg").src = "rock_happy.png";
        setTimeout("document.getElementById('rockImg').src = 'rock.png';",
          5 * 60 * 1000);
      }
    </script>
  </head>

  <body onload="greetUser();">
    <div style="margin-top:100px; text-align:center">
      <img id="rockImg" src="rock.png" alt="iRock" style="cursor:pointer"
        onclick="touchRock();" />
    </div>
  </body>
</html>
```

document.body

文档主体（body）是网页里能被看见的部分，包括客户端窗口的高度与宽度。

没有蠢问题

问：我只是想弄清楚……网络客户端、浏览器、客户端窗口和浏览器窗口……有什么不同？

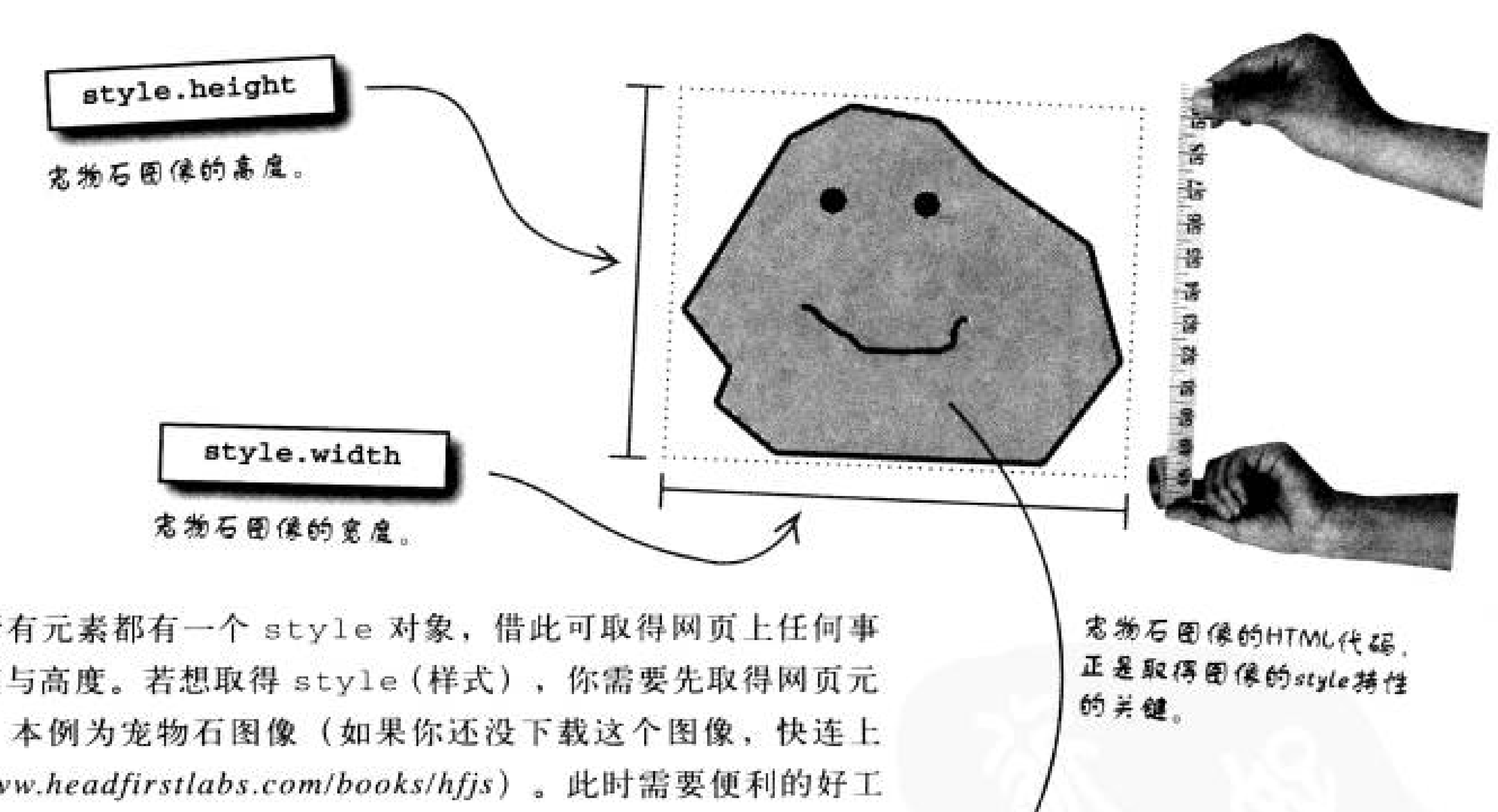
答： 嗯，听起来是有点让人混乱。在万维网的通用语里，“浏览器”（browser）意指网络客户端（web client），因为它在网页的提供路径上处于客户端。但在浏览器环境里，“客户端”（client）则有不同意义，此时它表示浏览器窗口里**呈现网页**的特定区域。所以，客户端窗口位于浏览器窗口里，并未包含标题栏、工具栏、滚动条等其他事物。

问：为什么建议根据客户端窗口调整 iRock 尺寸呢？

答： 客户端窗口提供较佳的图像尺寸调整基准，因为它反映了图像呈现处的实际大小，遂可减少无法考虑的变量，例如附加的工具栏、不同平台与不同厂商而造成的浏览器窗口尺寸差异等等。举个例子，Mac平台上的Safari与Windows版的Firefox，两者的浏览器窗口尺寸并不一样，但它们的网页呈现区域（即客户端窗口）大小相同。

设定iRock图像的宽度与高度

如果无法**调整宠物石的图像大小**，只知道客户端窗口大小也是枉然。幸好，借用一点CSS的能力，JavaScript就能调整图像尺寸（就算CSS不是你的风格）。图像元素的`width`与`height`特性不只决定图像刚开始的大小，也能于必要时动态调整图像尺寸。



网页上所有元素都有一个 `style` 对象，借此可取得网页上任何事物的宽度与高度。若想取得 `style`（样式），你需要先取得网页元素本身，本例为宠物石图像（如果你还没下载这个图像，快连上 *http://www.headfirstlabs.com/books/hfjs*）。此时需要便利的好工具——`document` 对象的 `getElementById()` 方法：

```
<img id="rockImg" src="rock.png" alt="iRock" ... />
```

```
document.getElementById("rockImg").style.height
```

这一小块代码可取得宠物石图像的高度。

若想改变宠物石图像的尺寸，只要指定图像宽度或高度的值。**只设定其中一个值就能产生作用，因为另一个值会配合图像比例自动缩放。**

设定宠物石图像高度为100 pixel。

```
document.getElementById("rockImg").style.height = "100px";
```

各位不用设置宽度值……它将根据新设定的高度自行缩放，以维持图像的比例。

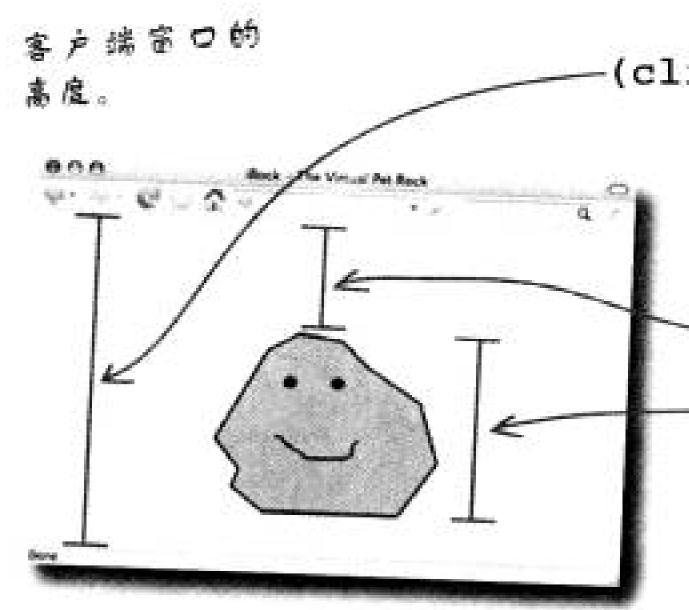
iRock应配合页面调整大小

目前，我们还没提到如何**根据客户端窗口大小**而计算图像的缩放。缩放必须与客户端窗口的大小成比例，我们希望用**百分比**（percentage）设定iRock的大小。

话虽如此，宠物石图像不是应该根据客户端窗口的宽度或高度（二选一）而计算吗？基于浏览器的**垂直高度**较受限制，我们采用客户端窗口的**高度**（height）比较保险。

客户端窗口的高度。

```
(clientWindowHeight - 100) * 0.9 = rockImageHeight
```



100 pixel。

剩余高度的90%。

这段计算可为宠物石图像设定垂直位置（距顶端100 pixel），再把图像尺寸设为剩余高度的90%（0.9）。有时候，这类计算需要多一点尝试，才能找出最恰当的数值。你应该用你的iRock试试，看看效果如何……不过，先利用“磨笔上阵”练习一下吧！

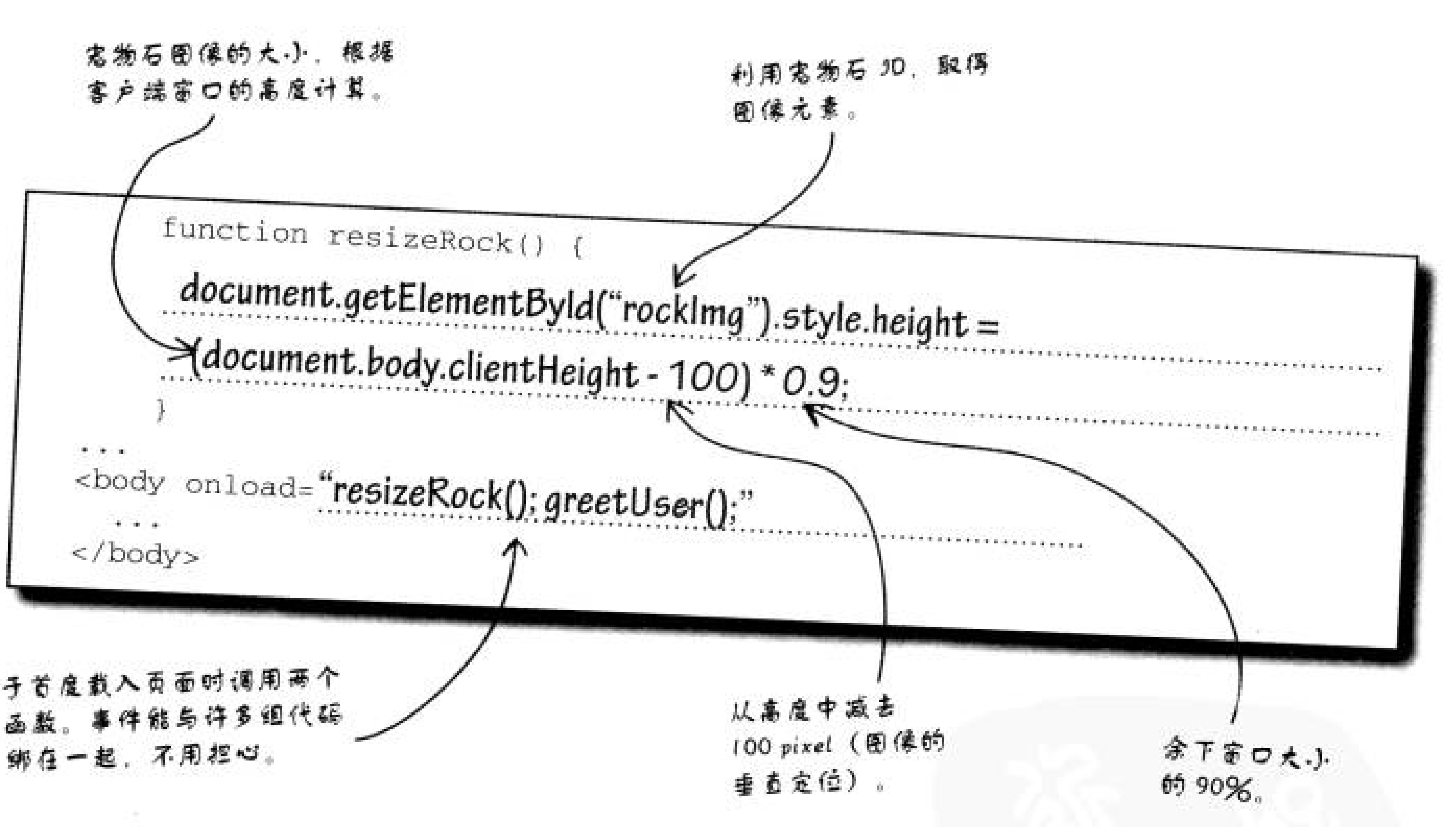
磨笔上阵

请完成 resizeRock() 的程序代码。你应该根据客户端窗口的尺寸，缩放宠物石的大小。另外，在 onload 事件处理器中，除了 greetUser()，还要加入对 resizeRock() 的调用。

```
    function resizeRock() {
    ..............................................
    ..............................................
    }
...
<body onload= ................................ >
  ...
</body>
```

磨笔上阵解答

请完成 resizeRock()的程序代码。你应该根据客户端窗口的尺寸，缩放宠物石的大小。另外，在onload事件处理器中，除了 greetUser()，还要加入对 resizeRock()的调用。



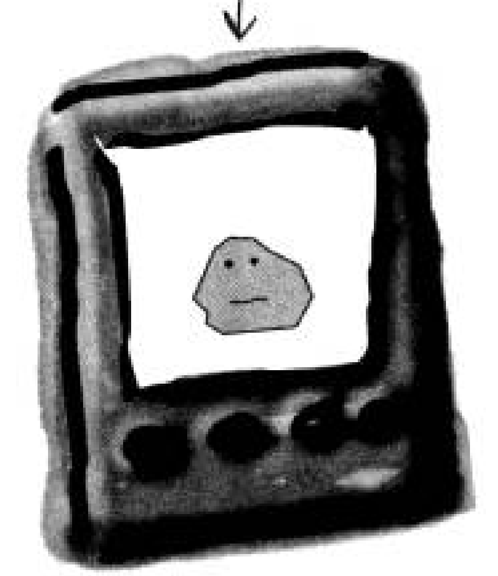
复习要点

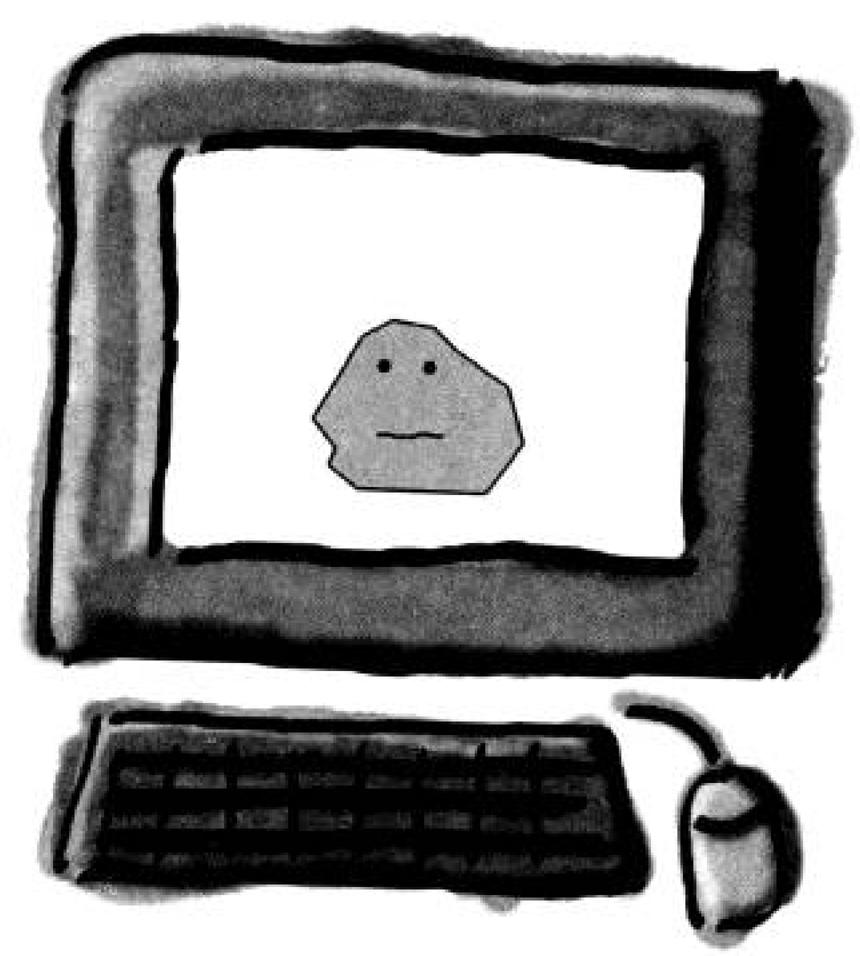
- setTimeout()函数让我们创建**单次定时器**，这种定时器将于一段时限后触发JavaScript代码。
- 设定依固定间隔启动的定时器，请使用setInterval()创建**间隔定时器**。
- 必以毫秒（millisecond）为指定定时器时限的单位，1 毫秒为千分之一秒。
- 网页元素具有设定样式的style对象，width、height 都是样式。
- 客户端窗口是浏览器窗口的一部分，只负责呈现网页，其他不关它的事。
- 你可以透过document对象的body.clientWidth与body.clientHeight特性，取得客户端窗口的宽与高。

你的 iRock……进化了！

修改代码后，iRock 进化为适应各种浏览器环境的新一代物种。请确认你也更新了自己的 *iRock.html*，与第104 页的解答相符（可至 *http://www.headfirstlabs.com/books/hfjs* 下载），记得多采用几种浏览器与窗口大小测试网页载入结果。如果你愿意，也可以在 iPhone 上试试哦！

宠物石的大小能随着浏览器窗口的大小而变化了。





用户的宠物石不再有问题，Alan 正准备给你丰厚的入股权。大家都过得幸福美满……暂时如此……

没有蠢问题

问：我还是不知道 iRock 图像尺寸计算中出现的“100”有何作用。它是何方神圣？

答： 请看iRock网页里的HTML/CSS代码，它们把图像放在距离网页顶端100 pixel的位置，才不会显得太窘迫。先减去100 pixel的位移量，再以比例（90%）计算图像高度，即可维持设计美学。100 pixel一点都不神秘，只是大多数浏览器都很适合加上这段距离。

问：我能利用`style`的`width`与`height`特性，改变任何元素的尺寸吗？

答： 几乎。希望借由稍早的举例，大家都能了解 JavaScript 对于网页内容的强大操控力量。以 iRock 的脚本为例，先用到查询客户端窗口大小的能力，然后把查到的尺寸，当成改变图像大小的根据。

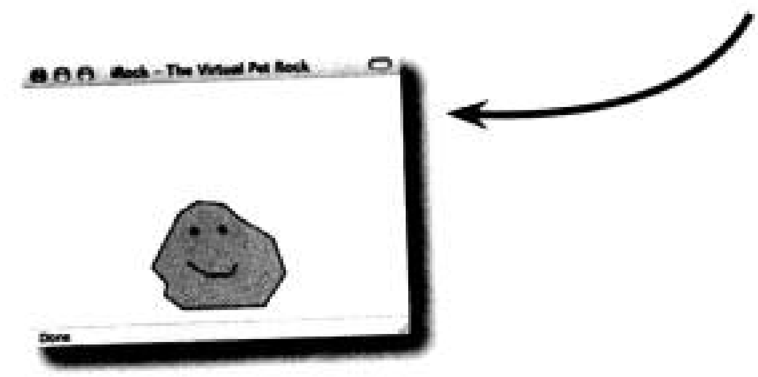
问：为什么不在标头（head）部分的JavaScript代码里直接改变 iRock的图像尺寸，反而要用`onload`事件呢？

答： 因为网页在`onload`事件触发后才会载入。如果你的JavaScript代码（就像iRock）会取用网页元素，你就没办法在`onload`事件发生前运行任何程序代码。

什么都不会发生，我们的石头不会动态改变大小。

有些用户可能会调整浏览器窗口的尺寸，但iRock不会随之变化大小，想必用户对此不太满意。因为，宠物石图像的大小只在网页首次载入（在onload事件中）后调整。此后，没有任何行动能导致图像调整自己的大小。太糟了，我们又回到问题的原点：

图像在浏览器窗口调整大小时维持原状。



浏览器改变大小时触发 onresize 事件

为了让宠物石图像能随（浏览器的）客户端窗口等比例缩放，你的脚本需要知道用户何时调整了浏览器窗口。浏览器大小的改变需透过 onresize 事件沟通。onresize 事件于浏览器有任何尺寸修改时被触发，此时若想一并调整网页里的图像，你得捕捉这个事件。

iRock – The Virtual Pet Rock

Done

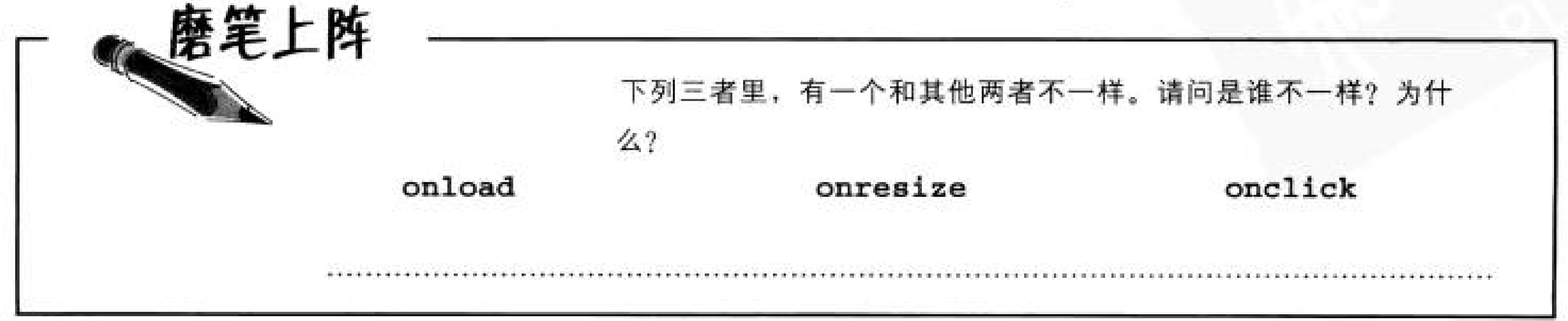
调整浏览器窗口尺寸将触发 onresize 事件……

onresize!

……导致事件处理代码的运行。

```
<body onresize="doSomething();">
```

想在 onresize 事件发生时提供响应，只要在<body>标签的 onresize 属性里填入 JavaScript 代码即可。



磨笔上阵

下列三者里，有一个和其他两者不一样。请问是谁不一样？为什么？

onload **onresize** **onclick**

下列三者里，有一个和其他两者不一样。请问是谁不一样？为什么？

onload（圈出） **onresize** **onclick**

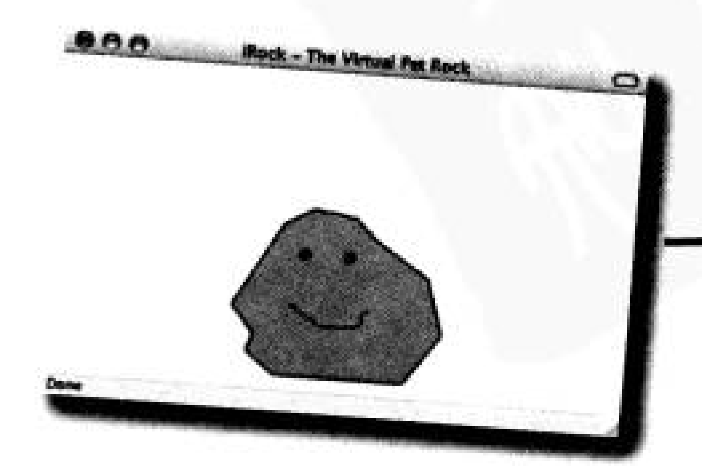
onresize 与 onclick 均由用户触发，但 onload 则不然。

onresize 事件调整了宠物石的大小

好了，我们已经创建了调整宠物石图像尺寸的函数，享受成果的时刻到了。为了配合浏览器窗口尺寸的调整而改变宠物石图像的大小，你必须调用 resizeRock() 函数以响应 onresize 事件：

浏览器窗口调整尺寸时，iRock 终于会自动修正图像大小了。

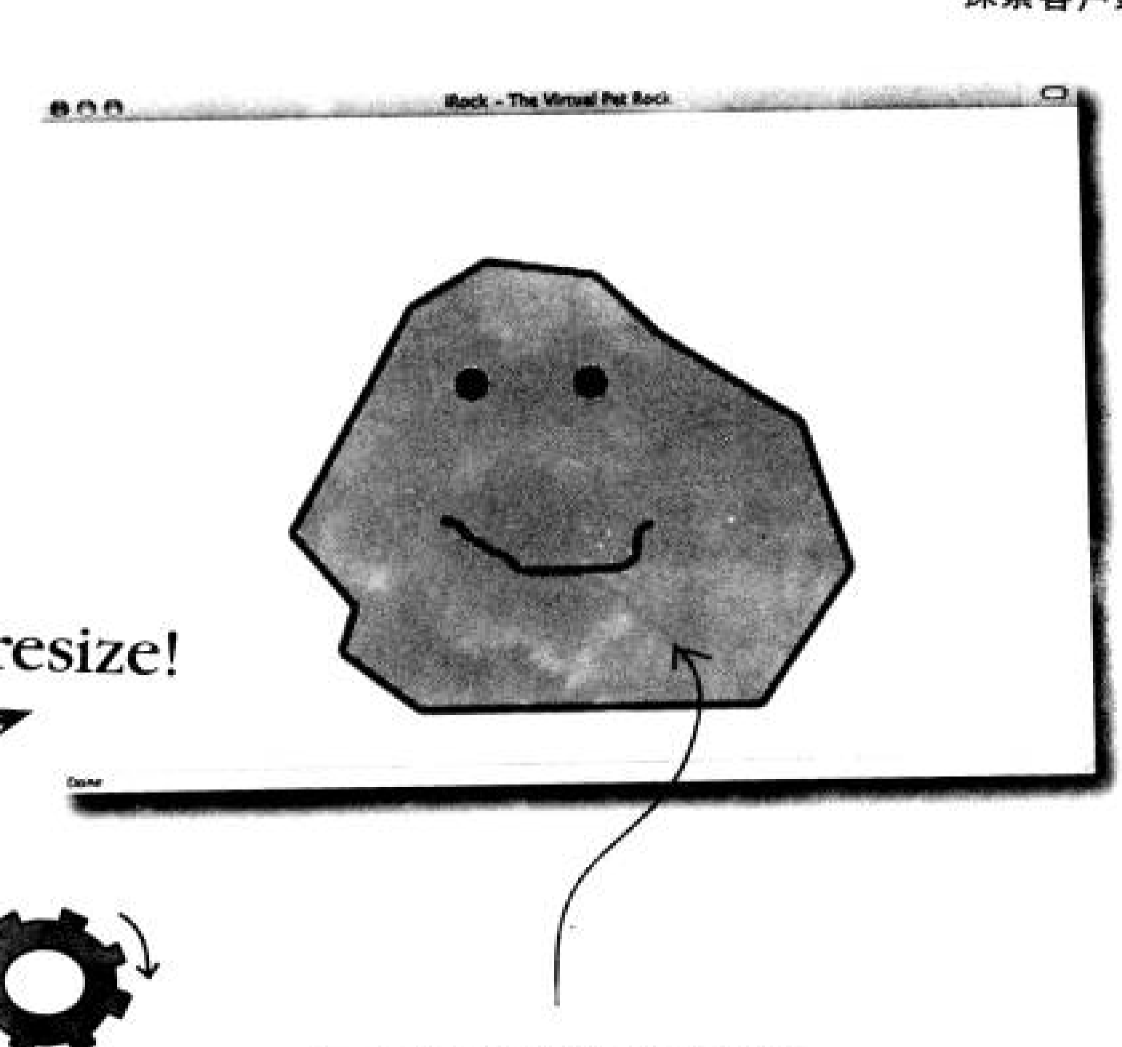
onresize 事件能让我们检测并响应浏览器窗口尺寸的调整。



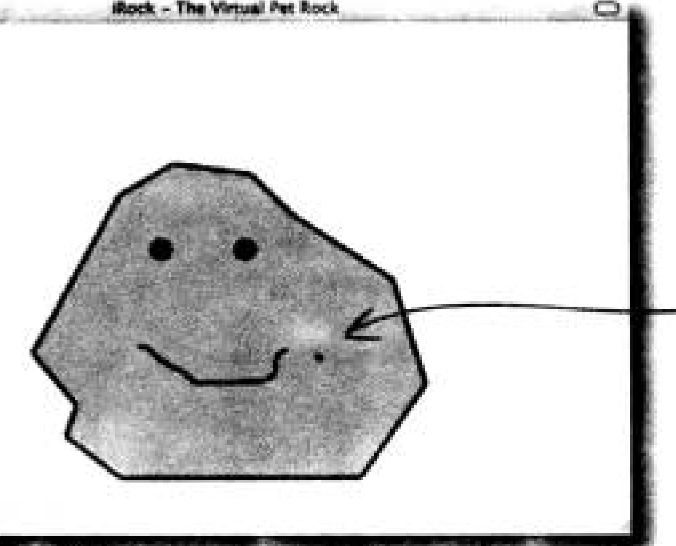
注意！

以 JavaScript 调整图像尺寸时请小心。

尤其在放大很小的图像时更要注意。图像的品质将蒙受缩放之害。



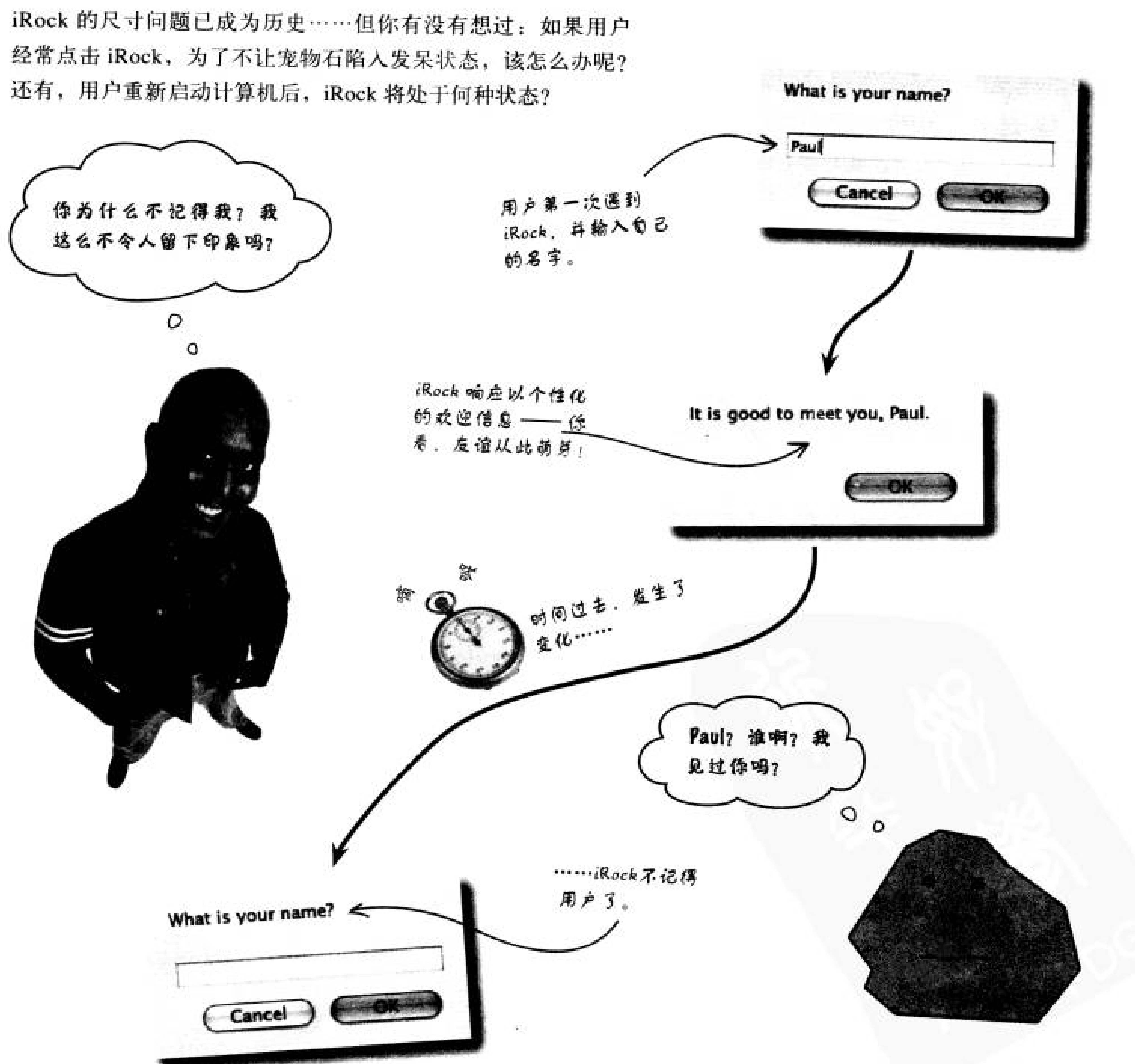
JavaScript 检测到客户端的改变，随即因应改变而动态调整网页内容。



因为 iRock 不再受困于五花八门的浏览器尺寸，Alan 简直可以感受到用户此后的心花怒放！iRock 不只在网页首次载入时配合浏览器的客户端窗口而调整大小，还会在用户改变浏览器大小时一起修正。

我们见过吗？记住用户

iRock 的尺寸问题已成为历史……但你有没有想过：如果用户经常点击 iRock，为了不让宠物石陷入发呆状态，该怎么办呢？还有，用户重新启动计算机后，iRock 将处于何种状态？



虽然 iRock 以前真的见过它的主人，但却忘记了主人的名字……

每个脚本都有大限

iRock 的失忆症其实与脚本的生命周期（life cycle）有关，生命周期将影响存储在脚本的变量里的数据。

JavaScript 于浏览器关闭或网页重新载入时摧毁“所有”变量。

动动脑

该如何调整程序码，以修正 iRock 忘记使用者名称的问题呢？

cookie 延长脚本的生命周期

我们在 iRock 遇到的问题，有个专有名词：**持久性**（persistence），或者该说是“缺乏持久性”。有时候，我们真的需要不会消失的数据。但 JavaScript 变量的生命却如昙花一现，于浏览器关闭或网页更新的弹指间消逝无踪。浏览器的 cookie 提供了持久地存储数据，让数据比脚本生命周期更长的方式。

cookie，是浏览器存储在用户计算机里的一块数据。cookie很类似于JavaScript变量，只不过在你关闭浏览器、重新载入网页、关掉计算机的电源、整修房屋后，cookie 依然待在你的计算机里。所以，cookie 成了存储 iRock 用户名称的便利选择。

碰！

"Paul"

关闭浏览器时，脚本把用户名称写成一个 cookie。

onload!

"Paul"

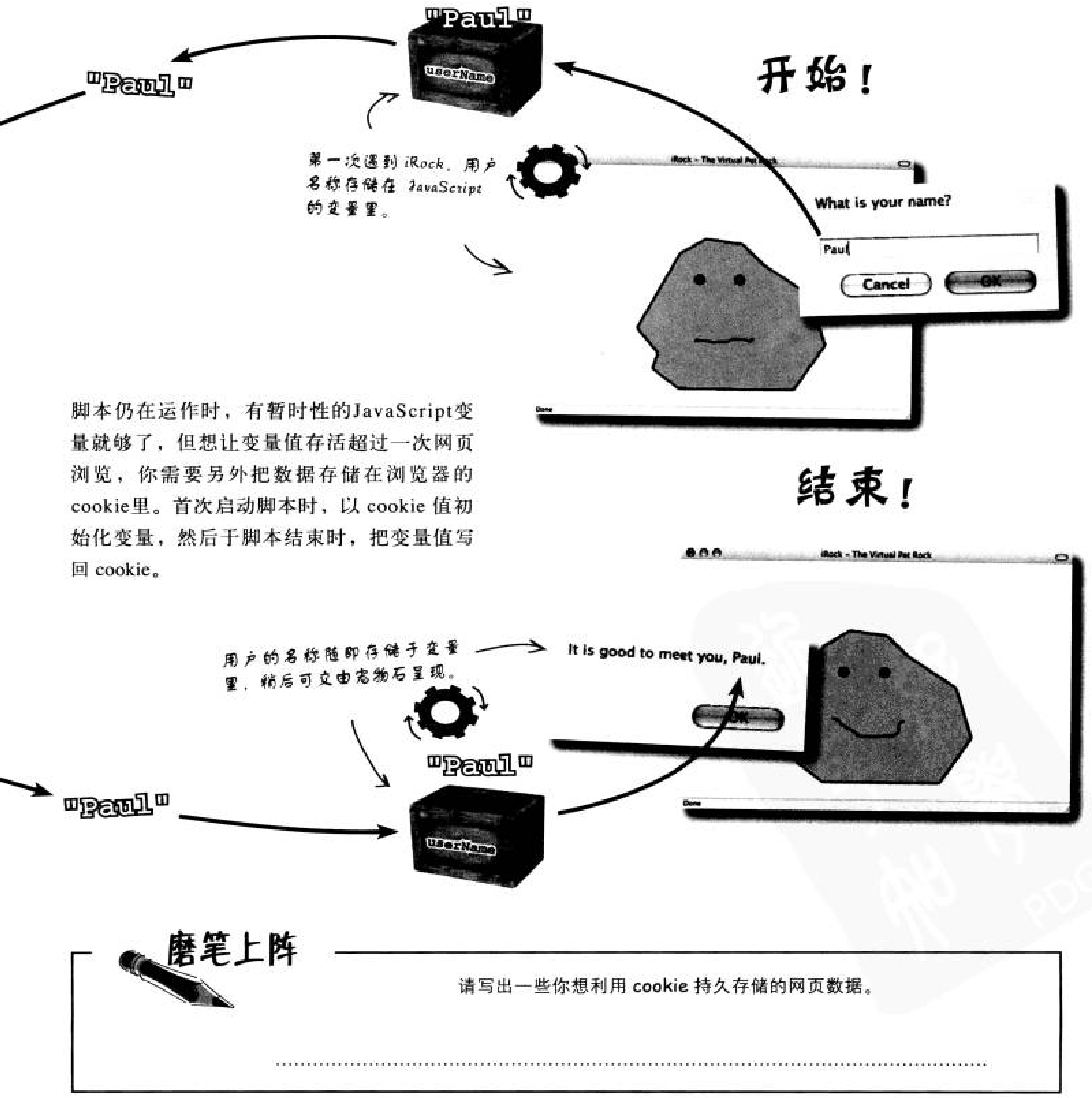
当网页再度被开启时，从硬盘上读取“用户名称”的 cookie。

浏览器保存着不同网页创建的各种 cookie。

"Paul"

硬盘

浏览器存储cookie（用户名称）于硬盘上，以策安全。

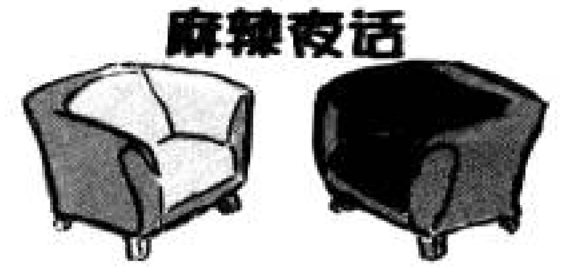


脚本仍在运作时，有暂时性的JavaScript变量就够了，但想让变量值存活超过一次网页浏览，你需要另外把数据存储在浏览器的cookie里。首次启动脚本时，以 cookie 值初始化变量，然后于脚本结束时，把变量值写回 cookie。

磨笔上阵

请写出一些你想利用 cookie 持久存储的网页数据。

....



今晚主题：变量与cookie评估持久性数据数据存储的重要性

变量：

真不知道为什么把我们摆在一起——你和JavaScript根本没有关系嘛！

Cookie:

这话只算说对了一半……没有 JavaScript，我还是能完成份内工作，但我与 JavaScript 也有重要的关系。我提供了长久保存数据的方式。你最近大概也发现了，JavaScript 的持久性实在不怎么样。

变量：

哦……你这话的意思……不就是说我不算够格的数据存储方案吗？每次浏览器关闭或网页重新载入，我就无能为力了……哼！我可不像某人，变量的取得可是**非常**容易呢！

Cookie:

不容易取得？我一直都跟浏览器待在一起，准备随时被调用啊！

变量：

或许吧……咦？你不是和其他饼干一起堆在那个狭小的房间里吗？

Cookie:

是啦……怎么样？

变量：

如果谣言属实，寻找 cookie 是项很花时间的行动……因为你们全都堆在一张庞大的列表里。光想到就觉得头痛……这就是我说的“不容易取得”。

Cookie:

呃，是啦，我们 cookie 都存储在一张庞大列表里，但我们的名称都不一样，要找出我们其实没那么难。只要知道如何分解列表，就能找出名称了。

请写出其他你想利用 cookie 持久存储的网页数据。

用户 ID、购物车内容、住址、语言

变量：

Cookie:

你说的没错，但也正是问题所在。我完全不需要什么列表，只要叫我一声……我就出现了！

少在那边自鸣得意。接下来才是真正重要的问题。如果用我存储数据，我永远都记得，不管浏览器是否已关闭，或网页是否重新载入。我是永恒……除非用户选择清除所有cookie。但这不是今晚的讨论主题。

持久性是很好，但无法解决每天发生的问题。仔细想想，哪来那么多需要永久存储的数据。事实上，暂时存储数据、使用完毕就丢弃数据，通常反而更有效率。这就是我的舞台。我是脚本数据的终极暂时性存储媒介。

随便啦……我觉得你太小看持久性数据存储的重要性了。你想想看，数天后再度造访网络购物，而所有待买物品都还在购物车上，这不是太神奇了吗？这种魔术都要靠我啊！

有趣的观点，不过你以为那些物品是怎么放入购物车的？大多数购物车都依赖我存储购物时的暂时性数据。我也很重要……说不定比你更重要，虽然我是有点健忘啦。

听起来我们好像能够互补耶！我一直以为你是敌人。

说得好。我们分别解决不同的问题，实在没必要互相竞争。不过，我还是要坚持一点，我的易存取性还是优于你的持久存储能力。

你就是忍不住要占点上风，是吧？我会找个地方好好休息，反正这页一结束，你就忘记整段谈话了。

什么谈话？

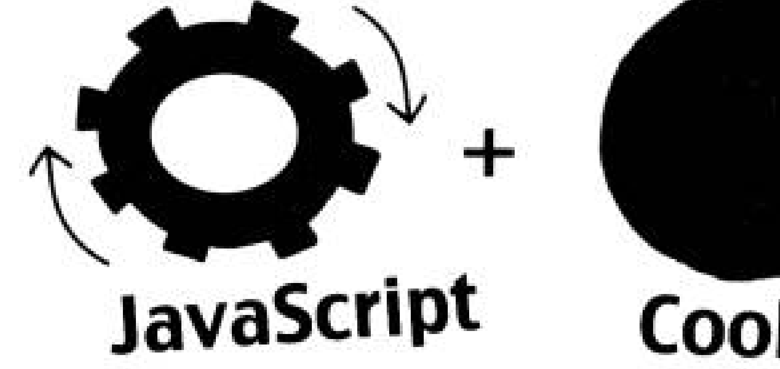
我就说吧！

为什么不直接把持久性网络数据存储在服务器上呢？

你不需要用服务器存储琐碎的信息，例如用户的名称。

没错，还有服务器。服务器的确是个持久存储数据的选择，但它可能被过度琐碎的数据折磨至死。在服务器上存储数据，需要在服务器上编写程序代码，还要准备存储媒介（如数据库）。服务器编程与数据库存储，对于存储持久用于简单客户端脚本的数据（例如 iRock 需要用户的名字），好像有点小题大做。

cookie 让我们能在客户端长久存储数据，又不用把服务器拖下水。不仅如此，如果想摆脱网页存储的持久性信息，用户也拥有清除 cookie 的能力。如果数据存储在服务器上，用户就无能为力了。

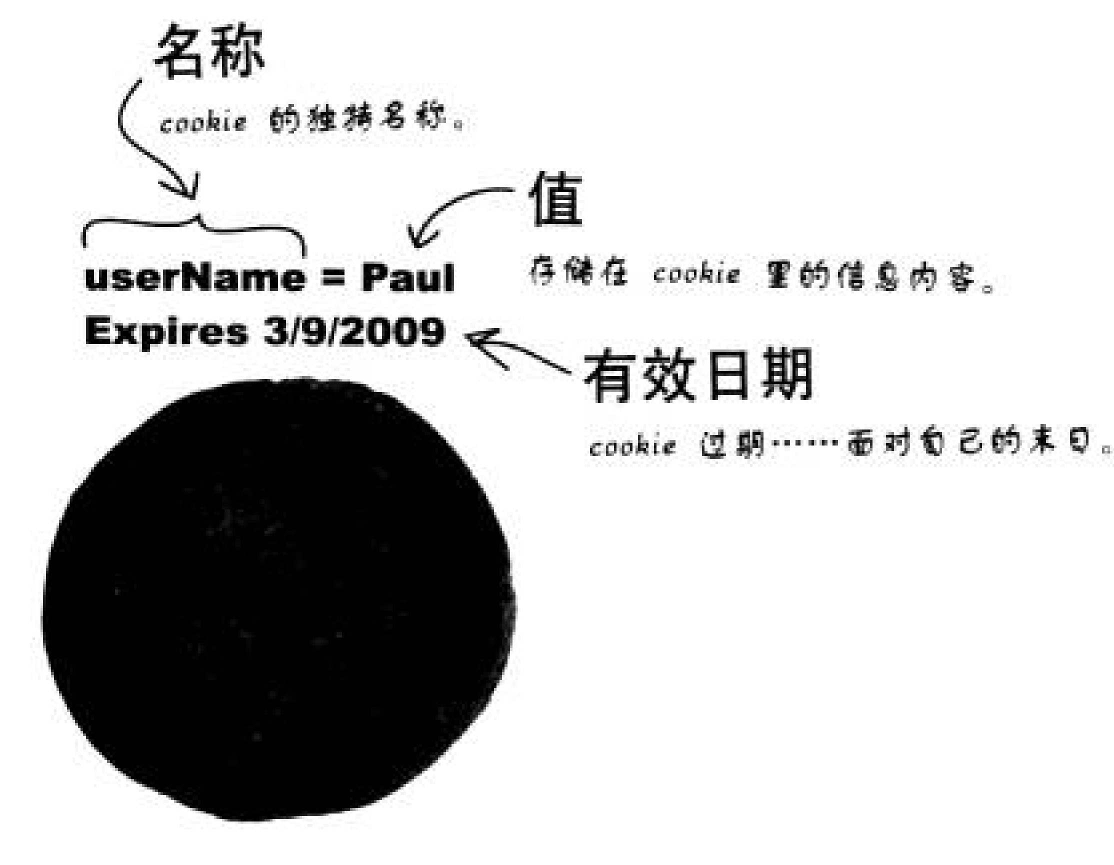


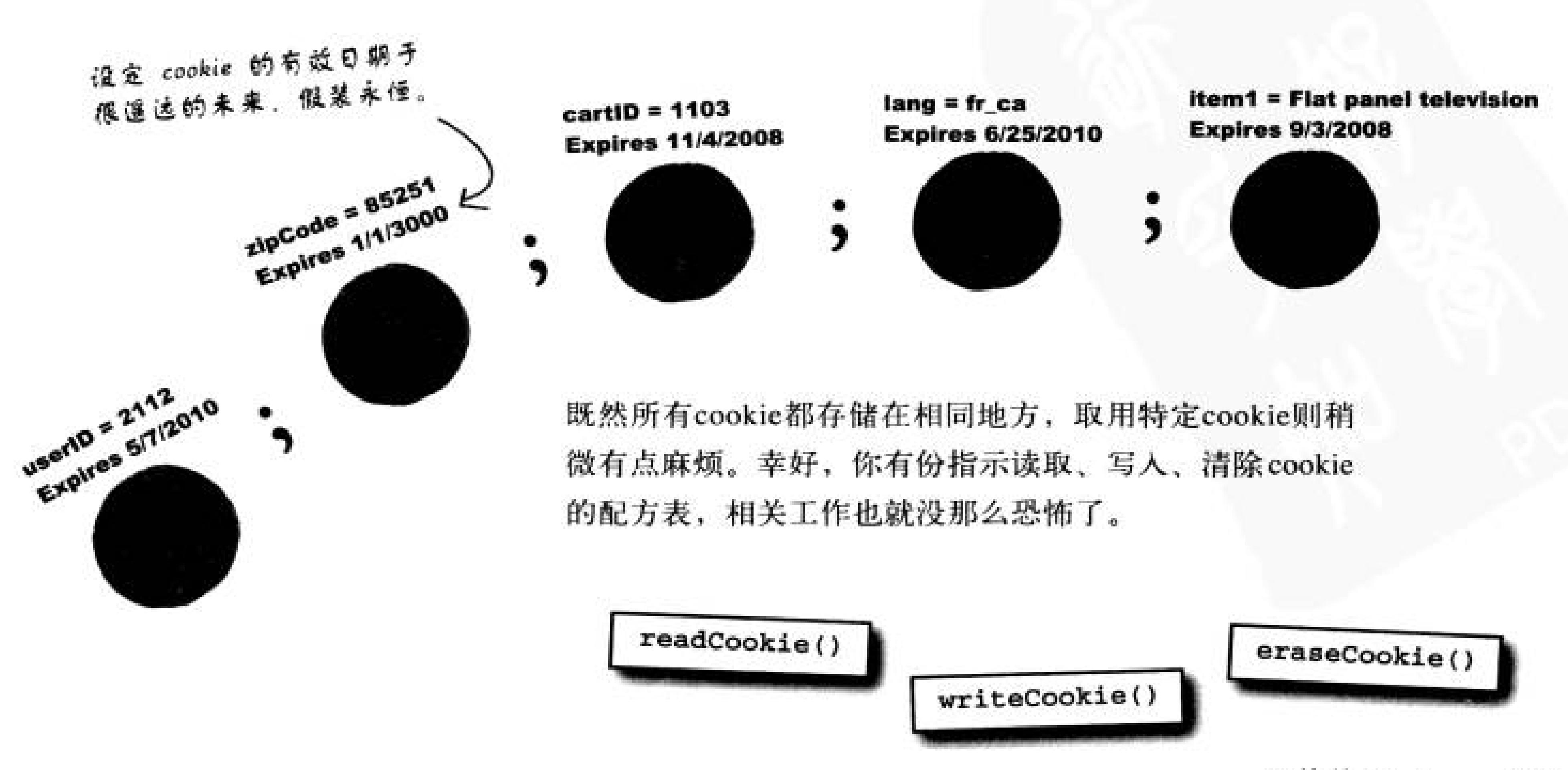
适合客户端、便利的持久性数据存储方案！

cookie 记录名称与数据值……但也可能过期

cookie 以独一无二的名称存储一段数据，这点很像变量。但是，cookie 可以设定有效日期（expiration date），时限一到，则销毁 cookie，这点跟变量不一样。实际上，cookie 并非真的永恒不变，它们只是比变量的寿命**更长**。你也可以创建未设定有效日期的 cookie，但是它将与一般 JavaScript 的变量没有两样 —— 于浏览器关闭时清除。

cookie 以长文本字符串的形式存储于用户的计算机里，该字符串的内容与网站（或网域）相关。分号（;）区隔各个cookie，读取 cookie 列表及取用某个cookie 时，分号将是关键。





既然所有cookie都存储在相同地方，取用特定cookie则稍微有点麻烦。幸好，你有份指示读取、写入、清除cookie 的配方表，相关工作也就没那么恐怖了。

`readCookie()` `writeCookie()` `eraseCookie()`

下例包含三个处理 cookie 的协助函数，让各位轻松地写入、读取、清除 cookie。有时候，最聪明的处理方式在于借用其他人的努力成果。请取用我们的函数（可至 *http://www.headfirstlabs.com/books/hfjs* 下载），然后尽量用在你的自制 cookie 函数中。

如果你还没办法完全了解这些范例，没有关系……慢慢看下去，重点在后面！

```
function writeCookie(name, value, days) {
  // By default, there is no expiration so the cookie is temporary
  var expires = "";

  // Specifying a number of days makes the cookie persistent
  if (days) {
    var date = new Date();
    date.setTime(date.getTime() + (days * 24 * 60 * 60 * 1000));
    expires = "; expires=" + date.toGMTString();
  }

  // Set the cookie to the name, value, and expiration date
  document.cookie = name + "=" + value + expires + "; path=/";
}

function readCookie(name) {
  // Find the specified cookie and return its value
  var searchName = name + "=";
  var cookies = document.cookie.split(';');
  for(var i=0; i < cookies.length; i++) {
    var c = cookies[i];
    while (c.charAt(0) == ' ')
      c = c.substring(1, c.length);
    if (c.indexOf(searchName) == 0)
      return c.substring(searchName.length, c.length);
  }
  return null;
}

function eraseCookie(name) {
  // Erase the specified cookie
  writeCookie(name, "", -1);
}
```

有效日期，表示为 cookie 可以存在的天数。

计算有效日期时，需把天数换算为毫秒数，再把毫秒数加到当前时间上。

cookie 列表使用分号区隔各个 cookie。

清除 cookie 的方式，就是把它改成没有值，而且有效日期已经过去（-1 天）。

创建一个空白文件，把它命名为 cookie.js，并把上述范例代码填入文件中。

cookie.js

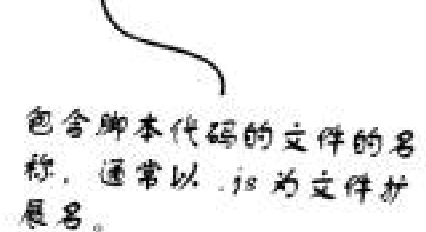
只包含 JavaScript 代码的文件，通常以 .js 作为文件扩展名。

你的 JavaScript 能活得比网页更长久

当 JavaScript 代码另外存储成一个文件后，你必须把文件导入（import）任何需要JavaScript 代码的网页。以 *cookie.js* 为例，它需要被导入至 *iRock.html* 网页里。只要对 `<script>` 做点小修改，就能导入：

别忘了结尾的 </script> 标签。

```
<script type="text/javascript" src="cookie.js"></script>
```



请你跟我这样做！把这里的修改加到你的 iRock.html，记得检查 cookie.js 是否也在相同目录下。

导入外部脚本代码至网页时，文件里的所有 JavaScript 代码将嵌入 HTML 的 `<script>` 标签，就像把这些代码直接放在网页里。只要是能用很多次的代码，最好都另存为新文件，而后再导入网页。

```
<html>
  <head>
   <title>iRock - The Virtual Pet Rock</title>
   <script type="text/javascript" src="cookie.js"></script>
   <script type="text/javascript">
     var userName;

     function resizeRock() {
       document.getElementById("rockImg").style.height =
         (document.body.clientHeight - 100) * 0.9;
     }

     function greetUser() {
```

导入的脚本代码将于网页载入时加到网页里。

习题

为什么最好把可再利用的代码存成外部文件？请写下原因。

....

为什么最好把可再利用的代码存成外部文件？请写下原因。

代码的再利用度；因存放在单一位置而容易维护；文件的组织度较佳。

利用 cookie 欢迎用户

我们需要一个经过 cookie 加持的 iRock 脚本。若用户名称已经存入 cookie，新脚本可以提供用户专属的问候语；如果没有名称，则采用一般、未个性化的问候语。真是个两全其美的好办法！

userName = Paul
Expires 3/9/2009

找 cookie 取得用户名称，并存储在某个变量里。

JavaScript 变量。

userName =

这里可能是用户名称……或者什么都没有！

找到用户名称了吗？

找到了！

"Paul"

userName

个性化

It is good to meet you, Paul.

OK

没找到

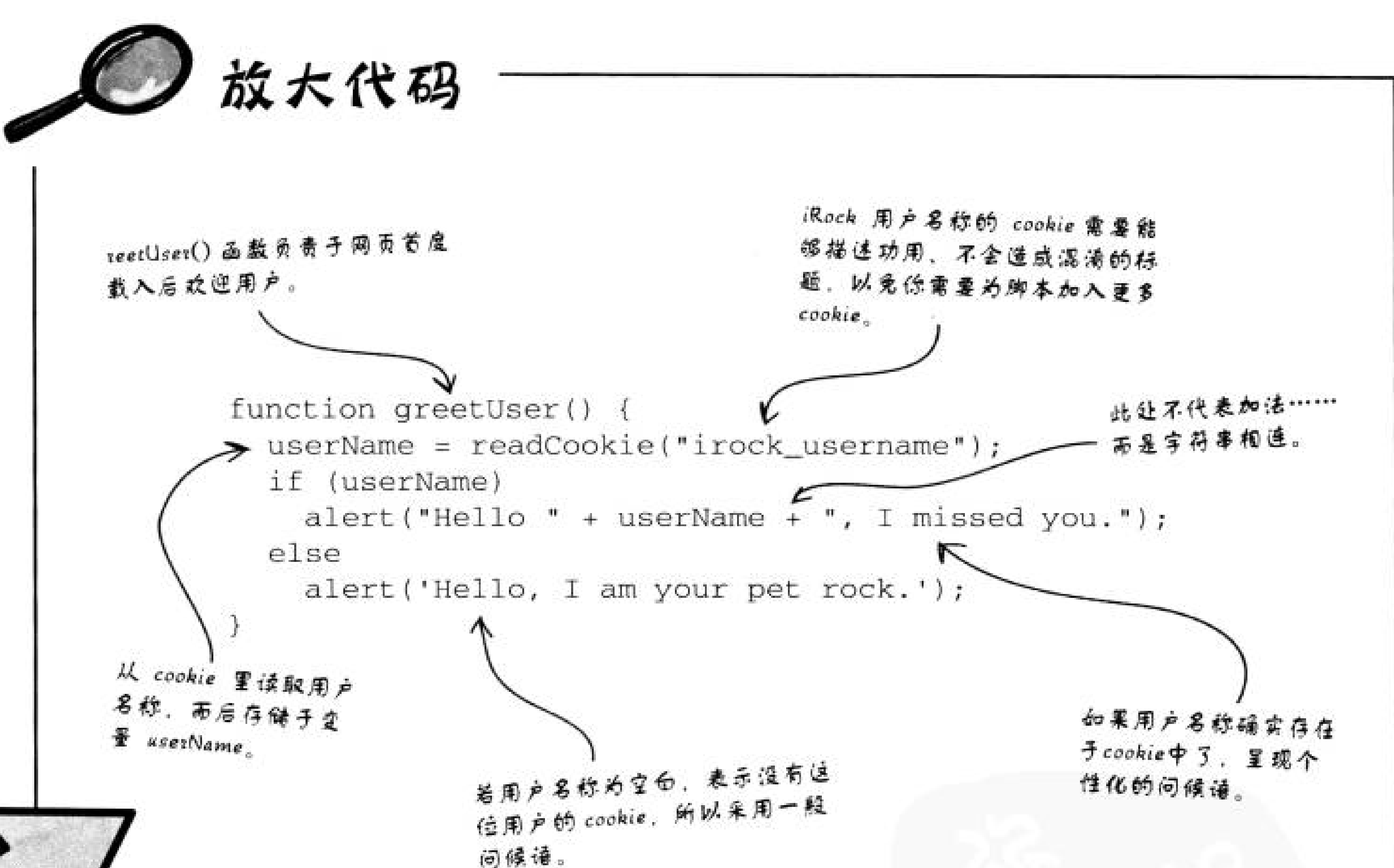
userName

一般用语

Hello, I am your pet rock.

OK

放大代码

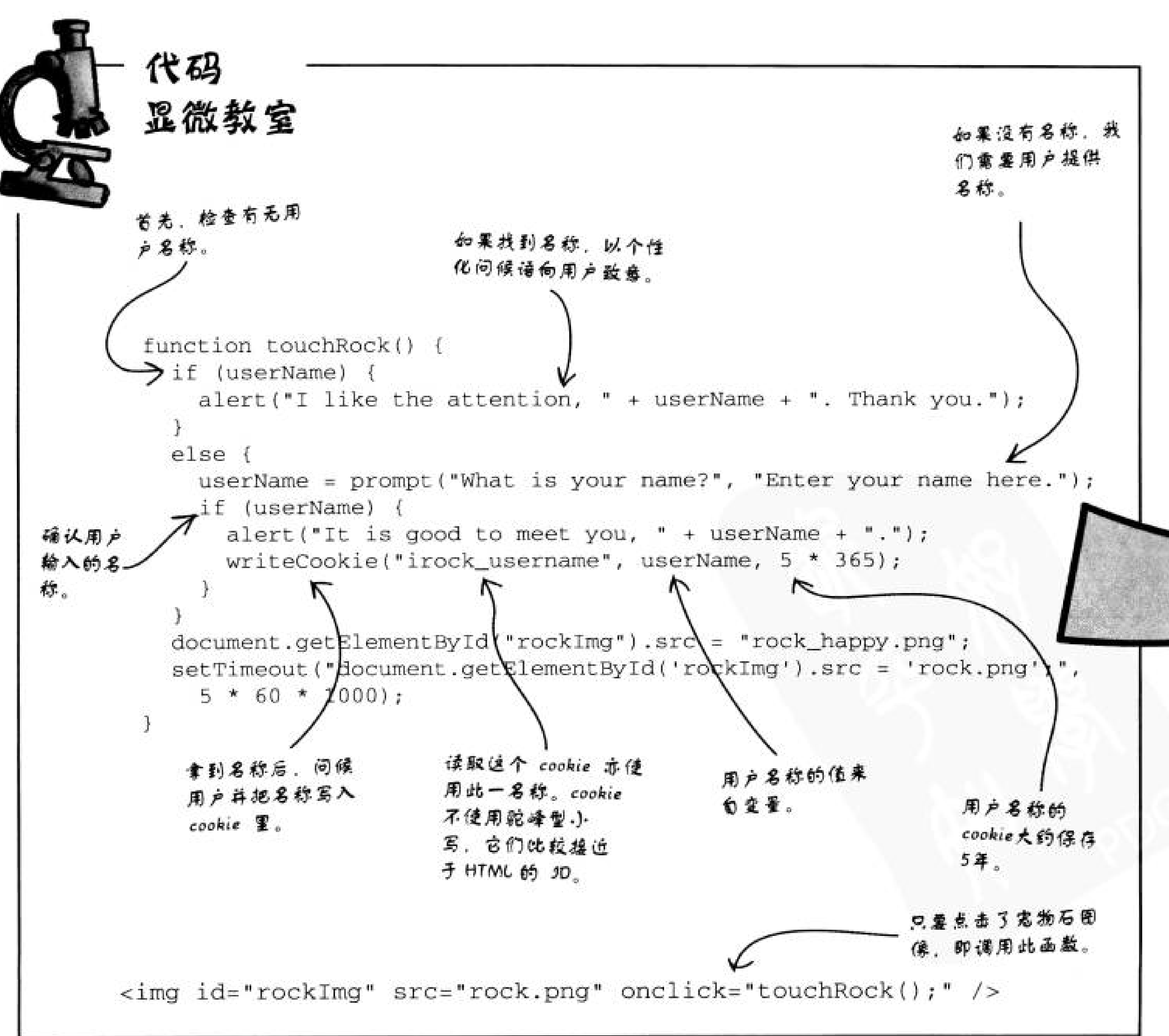


greetUser() 有了 cookie 的神力加持

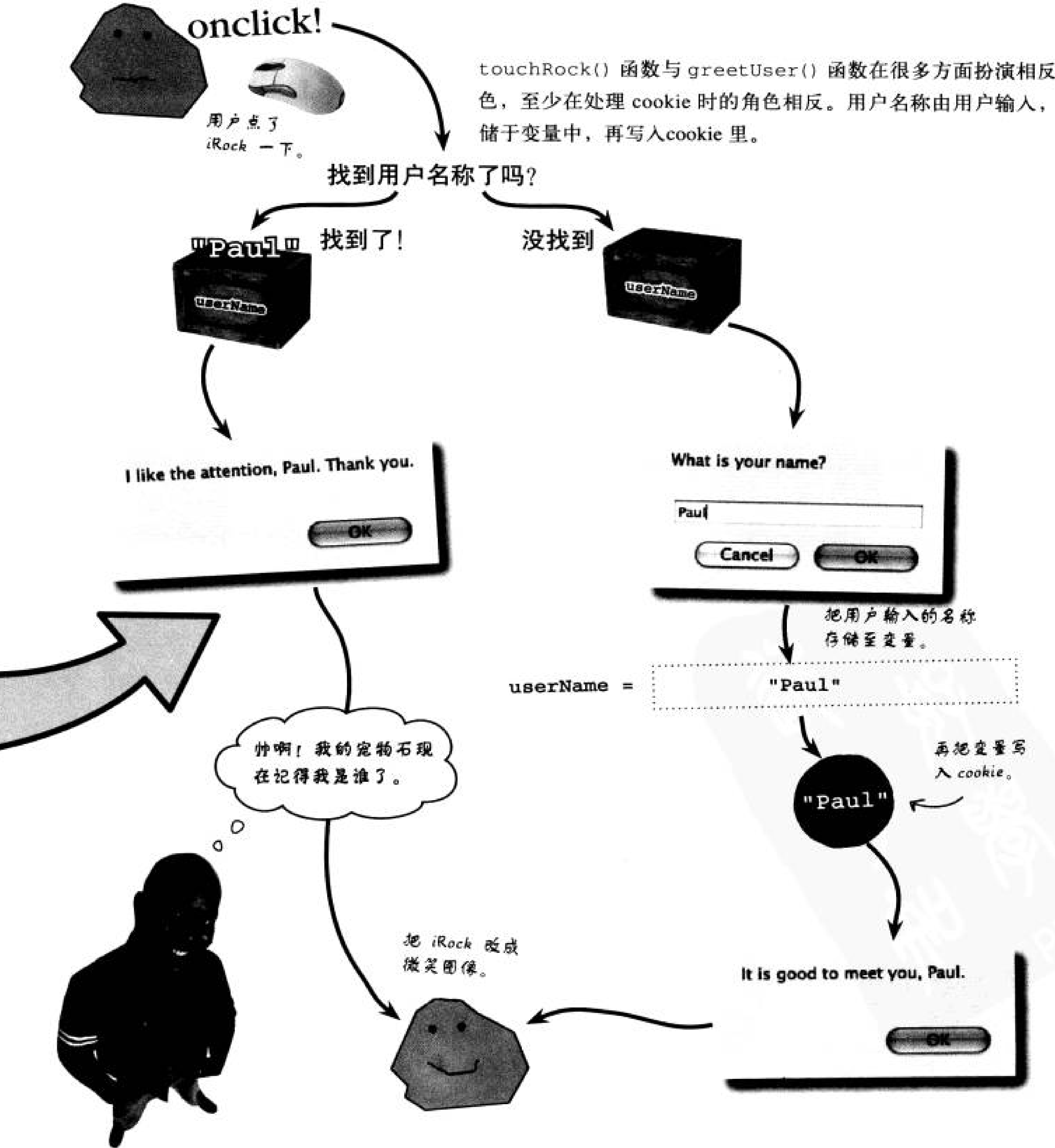
函数 `greetUser()` 就像一首变量与 cookie 的重唱曲。从cookie 中找到的用户名称将存储在变量里，但我们不能全盘指望 cookie 存有一切名称……如果用户第一次造访脚本，未曾输入名称，该怎么办呢？所以，范例代码检查变量是否存储了cookie 提供的名称 —— 这项检查决定采用个性化或一般的问候语。

别忘记设定 cookie

读取 iRock 的 cookie，还不错，是个好主意，但别忘了一开始要设定 cookie。写入 cookie 的工作应该由 `touchRock()` 函数处理（它是用户点击宠物石图像时调用的函数）。`touchRock()` 已提供了输入用户名称的方式 —— 现在只需要把输入的名称写入cookie 里。



touchRock() 函数与 greetUser() 函数在很多方面扮演相反角色，至少在处理 cookie 时的角色相反。用户名称由用户输入，存储于变量中，再写入cookie 里。



cookie影响浏览器的安全性

虽然大多数 iRock 用户都很高兴 cookie 能治好失忆症，但也有少数人质疑 cookie 的安全风险。这是个好问题。个人数据常用 cookie 存储，但 cookie 实际上**并未**导致安全风险……至少与访问计算机存储的敏感数据无关。不过，cookie 本身也非安全的存储场所，最好别把**敏感**数据存储在 cookie 里。

注意！

可以，不等于应该。

虽然可用cookie 存储任何东西，但它实在不算极度安全的数据存储方式。以 cookie 存储敏感数据，决非好主意。

cookie 只是一块文本数据，存储在计算机上的文件里。

广播暂停，向各位紧急宣布 JavaScript 的安全问题……

虽然 cookie 通常存储于硬盘上，但无权碰触硬盘上的其他东西。

因为 cookie 不是可执行的程序，它们不会散布病毒或蠕虫。

cookie 能存储个人数据，但用户必须特意上网输入数据。

有虫啊！！

没有蠢问题

问：cookie 总是存储在用户的硬盘上吗？

答：不见得。绝大多数浏览器都用硬盘存储 cookie，但有些浏览器无法这么做。例如某些移动设备采用特殊存储器（不是硬盘）作为持久性的数据存储方案，设备上的浏览器即以持久性存储器存储 cookie。即使如此，就浏览器（和脚本）的角度而言，cookie 只要记得数据值，背后存储方式为何都无关紧要。

问：我怎么知道 cookie 名称是否独特（unique）？

答：只有在指定的网页里，cookie 名称才需要独特。因为 cookie 存储时会依创建它们的网页加上区隔，其中包含网页所在的网站信息。也就是说，网页实际上是 cookie 名称的一部分，至少就独特性而言。

底线：确认你的 cookie 在单一网站或网页里具有独特性。

问：不同浏览器能共享 cookie 的数据吗？

答：不行。各个浏览器负责维护自己独有的 cookie 数据库，所以 Firefox 或 Opera 其实看不到 Internet Explorer 设置的 cookie。

问：如果 cookie 这么方便，为什么有人会想把数据存上服务器呢？

答：简单地说，cookie 只适合存储相对较少的文本数据（少于 4 Kb）—— 很重大的限制。更重要的一点，cookie 实在不太有效率，你不会想浪费时间一直读写大量 cookie 数据。大量数据应是数据库的用途，而数据库通常留在服务器上。虽然 cookie 非常适合存储小量、不需占用服务器空间的数据片段，但并非适合所有网络数据的存储方案。而且，cookie 也不是存储敏感数据的理想方式，当初设计 cookie 时，并未把安全性列入考量。

问：有其他方式创建真正永远存在的 cookie 吗？

答：没有。不管你怎样想，每个 cookie 都有有效日期。构思 cookie 时，并非用于长期数据存储，甚至也不算中期的保存方案。换言之，cookie 适合把数据保存个几天、几周或几月。如果你面对需要存储很久的数据，或许改为存储在服务器上比较好。cookie 虽然也能保存几年，但很难保证用户在这几年内不会更新计算机、重新安装，或者干脆把 cookie 清空。

问：讲 cookie 也讲够了。JavaScript 代码另外存储于外部文件，会带来什么坏处吗？

答：不太有。不过，请记得外部代码是在程序代码需要出现在复数网页时，让程序代码的共享与维护更容易。如果是只出现在特定某页的程序代码，把它存到外部文件也不会有什么好处；除非是网页特别混乱，为了你自己着想才把 JavaScript 代码独立出来。

复习要点

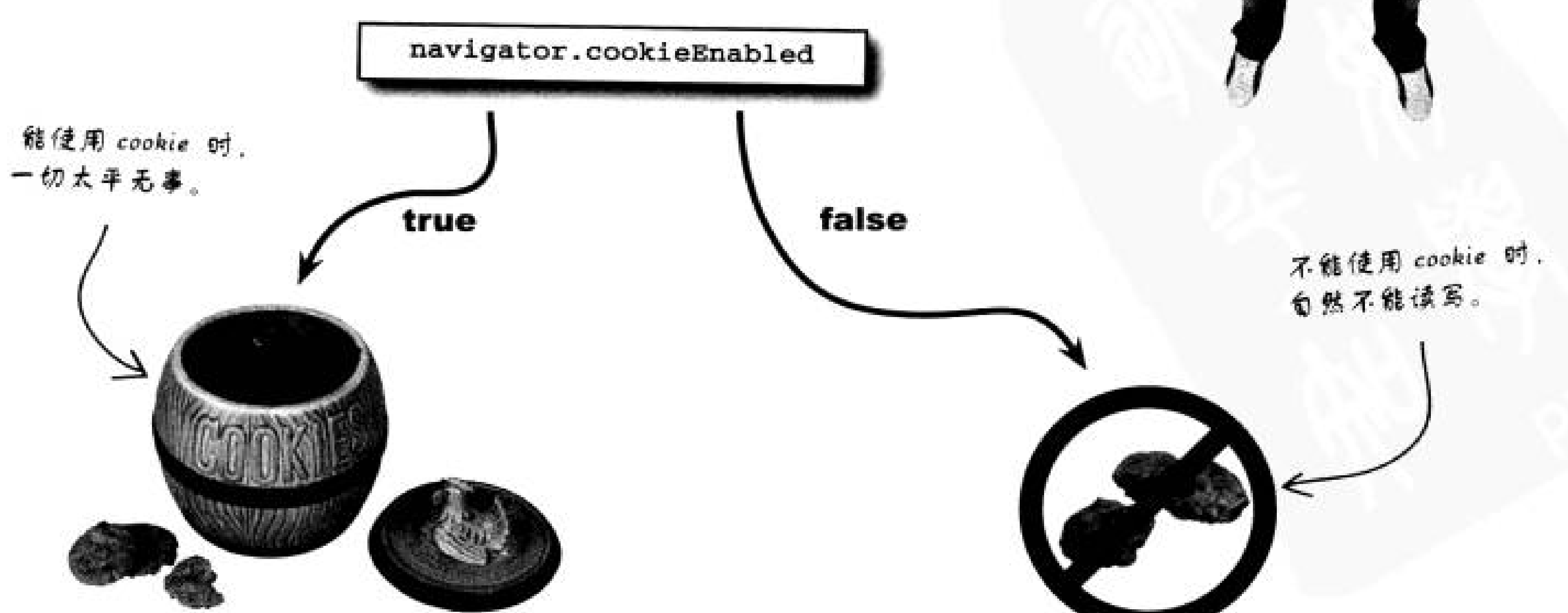
- cookie 是一块**文本数据**，由用户计算机上的浏览器下令存储。
- cookie 让脚本**存储的数据**能存活**超过单一网络** session。
- 每个 cookie 都有有效日期，超过期限，浏览器随即清除 cookie。
- 把脚本移动到**外部文件**，是个让程序更容易重复利用和维护的便利方式。
- cookie **不能访问用户的硬盘**或**散布病毒**，但可以存储输入网页上的个人数据。

没有 cookie 的世界

无论是安全问题还是浏览器的限制，却有少数 iRock 用户并未享受到 cookie 带来的好处，因为他们的浏览器不支持 cookie。这件事突显了大问题 —— iRock 假设大家都支持 cookie。iRock 依赖 cookie 记忆数据是一回事，但不能让用户不知道自己错失了完整的 iRock 体验。

有一个用户被遗忘，就等于少了一名顾客…………不可接受。

幸好浏览器有个可供检查 cookie 支持的特性。`cookieEnable` 特性是 `navigator` 对象的一部分，该对象为 JavaScript 提供浏览器本身的信息。



磨笔上阵

请帮 greetUser() 与 touchRock() 函数补上不见的代码，检查浏览器是否支持 cookie。还要帮 touchRock() 函数加上告知“无法使用 cookie”的代码。

```
function greetUser() {

  ........................................
    userName = readCookie("irock_username");
  if (userName)
    alert("Hello " + userName + ", I missed you.");
  else
    alert('Hello, I am your pet rock.');
}

function touchRock() {
  if (userName) {
    alert("I like the attention, " + userName + ". Thank you.");
  }
  else {
    userName = prompt("What is your name?", "Enter your name here.");
    if (userName) {
      alert("It is good to meet you, " + userName + ".");

      ........................................
        writeCookie("irock_username", userName, 5 * 365);
      else

      ........................................
    }
  }
  document.getElementById("rockImg").src = "rock_happy.png";
  setTimeout("document.getElementById('rockImg').src = 'rock.png';",
    5 * 60 * 1000);
}
```

磨笔上阵解答

请帮 greetUser() 与 touchRock() 函数补上不见的代码，检查浏览器是否支持 cookie。还要帮 touchRock() 函数加上告知“无法使用 cookie”的代码。

```
function greetUser() {

  if (navigator.cookieEnabled)
    userName = readCookie("irock_username");
  if (userName)
    alert("Hello " + userName + ", I missed you.");
  else
    alert('Hello, I am your pet rock.');
}

function touchRock() {
  if (userName) {
    alert("I like the attention, " + userName + ". Thank you.");
  }
  else {
    userName = prompt("What is your name?", "Enter your name here.");
    if (userName) {
      alert("It is good to meet you, " + userName + ".");
      if (navigator.cookieEnabled)
        writeCookie("irock_username", userName, 5 * 365);
      else
        alert("Sorry. Cookies aren't supported/enabled in your browser. I won't remember you later.");
    }
  }
  document.getElementById("rockImg").src = "rock_happy.png";
  setTimeout("document.getElementById('rockImg').src = 'rock.png';",
    5 * 60 * 1000);
}
```

如果支持 cookie，则从 iRock 的 cookie 里读取用户名称。

如果支持 cookie，则写入记录用户名称的 cookie。

告知用户，缺少 cookie 的支持将限制 iRock 的功能。

你的 iRock.html 网页应该像这样……实际试试吧！

问：**能够根据浏览器或其版本，而得知客户端是否支持 cookie 吗？**

答：检测浏览器是种不可靠的编程捷径，最后只会造成不可靠的结果。浏览器自己回报的版本不可信赖，唯一可靠的 cookie 支持检查途径只有 `navigator.cookieEnable` 特性。

与用户沟通……讲了总比没讲好

虽然在没有 cookie 可用时，也没有模拟 cookie 的好方式，幸好，向用户报告坏消息就能稍稍满足用户了。

太好了，有办法处理缺少 cookie 的浏览器。

iRock - The Virtual Pet Rock

Cookies aren't supported/enabled in your browser, which means I won't remember you later. I'm sorry.

OK

Done

比起清楚说明用户缺少的东西，还有很多更糟的状况。

Alan 的心情又好多了……你也收到另一张支票啦！

JavaScript 之王也会满意 iRock 的表现

走过这一趟帮助 iRock 成功的旅程，这双刚入手的 JavaScript 新鞋真的留下不少足迹。加上一点客户端的协助，iRock 现在更具有人性，不再死板地维持一致大小，而且记忆力也改良了！

浏览器的度量单位和 CSS 的 style 特性，赋予 iRock 确认周遭环境的能力。

iRock – The Virtual Pet Rock

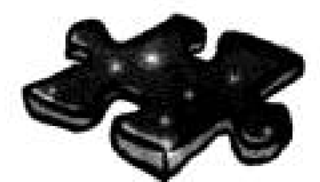
Done

多谢你的辛勤工作，iRock 现在是个坚若磐石的好宠物了！

cookie让iRock记得数据，超越脚本的生命周期。

userName = "Paul"

Paul！很高兴能认识你。



JavaScript 填字游戏

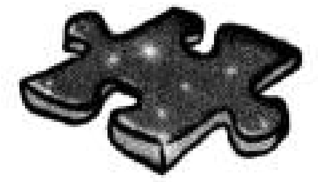
坐下来，换你的右脑运动一下。这是个标准的填字游戏，一切答案都来自本章的内容。

横向提示

5. 千分之一秒。
7. 我负责管理 cookie 列表。
8. 一种 JavaScript 机制，能在时限到临时运行程序代码。
10. cookie 包含名称、值与________。
11. ________于客户端存储稍后可能需要的信息。
12. 从网页引用外部 JavaScript 代码的行为，称做________。

纵向提示

1. 重复运行一段程序代码的定时器。
2. 用于创建单次定时器的函数。
3. 在脚本结束运行后，数据仍然存在，这是________。
4. 浏览器窗口调整尺寸时，触发________事件。
6. Web浏览器的另一个称呼。
9. cookie 无法散布这些东西。



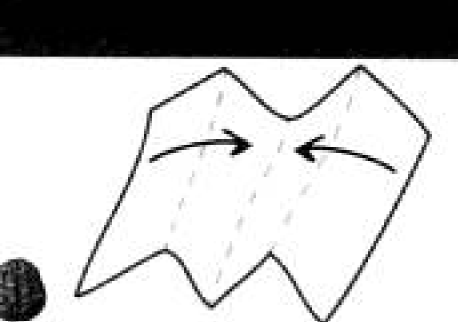
JavaScript 填字游戏解答

		1 I																
		N																
2 S		T																
E		E																
T		R																
T		V																
I		A											3 P					4 O
5 M	I	L	L	I	S	E	6 C	O	N	D			E					N
E							L					7 B	R	O	W	S	E	R
O						8 T	I	M	E	R			S					E
U					9 V		E						I					S
T					I		N						S					I
	10 E	X	P	I	R	A	T	I	O	N	D	A	T	E				Z
					U								E					E
					S								N					
11 C	O	O	K	I	E			12 I	M	P	O	R	T					
					S													

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

JavaScript为何该在意客户端呢？



这是左右脑的秘密会谈！

客户端才是JavaScript运行的地方，
也就是说，JavaScript活在浏览器里。
这是件好事，因为服务器
可以少担心一些事，
例如存储cookie！

4 决策

前有叉路，面对抉择

生命中需要太多抉择。该停车或继续开、奶酪或布丁、当个污点证人或俯首认罪……没有下决定的能力，就无法完成任何工作。JavaScript也有相同的状况——**决策（decision）让脚本在不同结果间作决定。作决定，为你的脚本带来“故事”**，即使是最世俗的脚本，或多或少都牵涉到故事。我是否信任用户输入的内容，并为她预订一趟寻找萨斯考奇大脚生物的神秘之旅？还是应该再检查一下，或许她想预订前往加拿大萨斯喀彻温省的巴士？决定权在你。

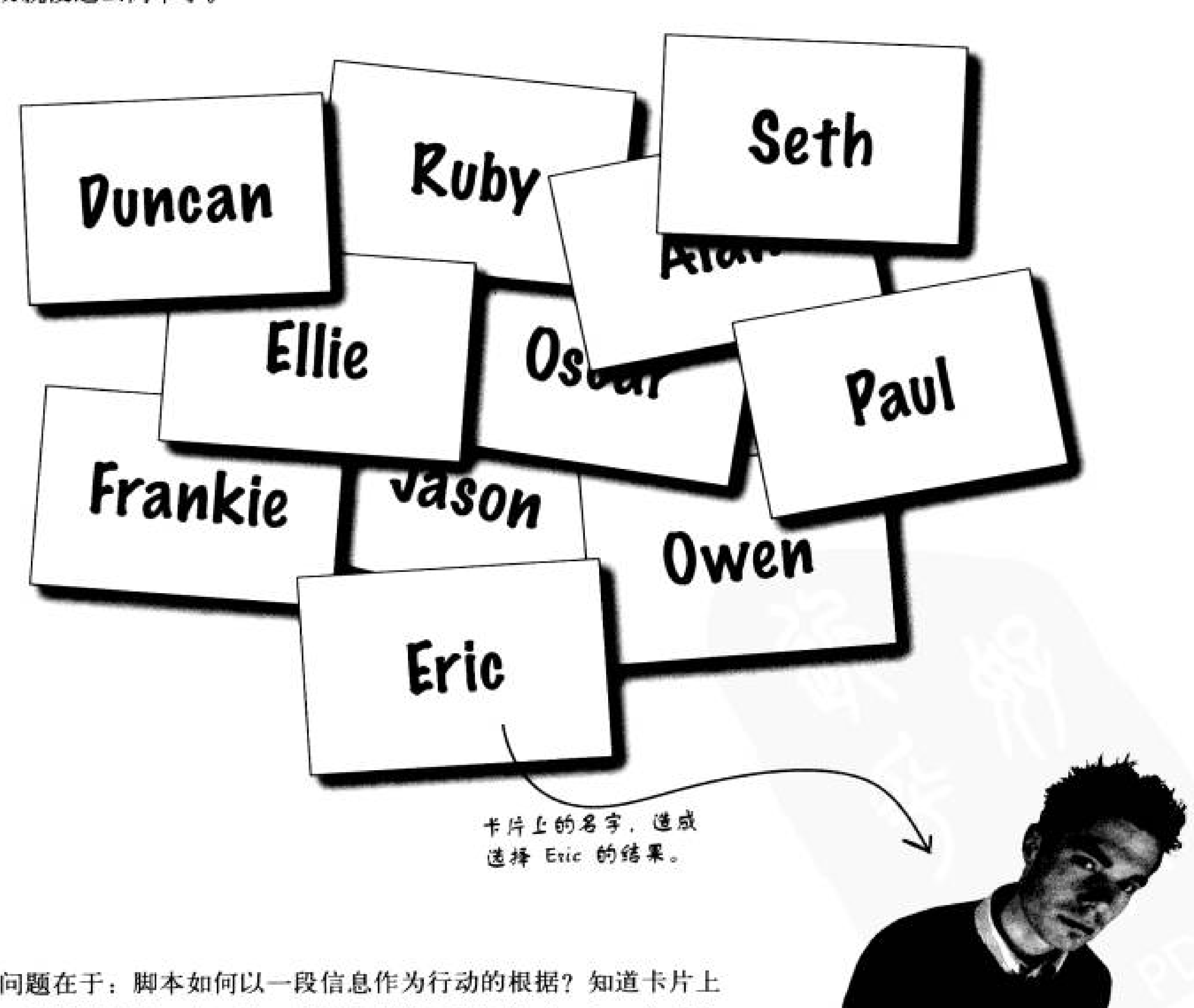
幸运的参赛者，来吧！

今天的节目是刺激精彩的《全民猜猜乐》，马上就要选出幸运的参赛者……

虽然你在家中等着见识Eric的决策本领，真正的问题却是：游戏节目主持人怎么知道要宣布Eric为参赛者？

选择，都是决策的一部分

拜托，他的名字就写在卡片上啊！还用问吗？没错，有名字的卡片，但你已经把主持人根据卡片上的名字作**决策**（decision）的事实视为理所当然。因为主持人是个人，而人类擅于处理信息与作出决定。如果主持人是份脚本，一切就没这么简单了。

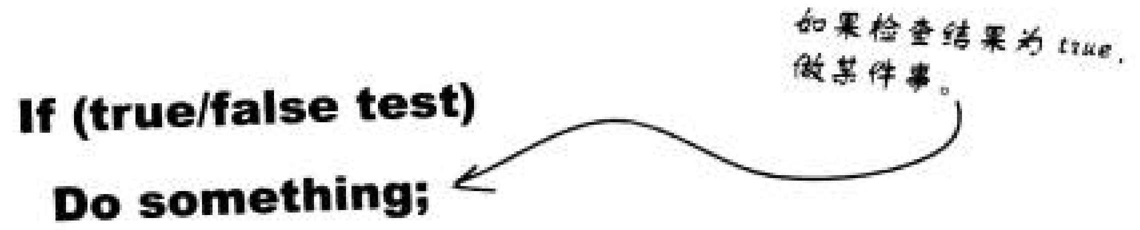


卡片上的名字，造成选择 Eric 的结果。

真正的问题在于：脚本如何以一段信息作为行动的根据？知道卡片上的名字，只是答案的一半。另一半答案则关系到**评估**（evaluae）卡片上的名字，而后**选择**（choose）游戏参赛者，本例选中了Eric。

“如果”是真的……就这样做……

JavaScript 其实已经引入处理信息与决策的需求，其中一种应对方式即为 if 语句。if语句能根据评估结果为 true/false，作出简易决定，而后有条件地运行 JavaScript代码。



戴上 JavaScript 版if语句的镜头，再看一次游戏节目的范例，将看到如下程序代码：

确认两方事物是否相等。

找到了！

Eric

```
if (chosenContestant == "Eric")
  alert("Eric, come on down!");
```

true/false 的检查必置于括号间。

并非 if 语句的真正结尾，没有分号。

if 语句到此结束。true/false 的检查必置于括号间。

Eric, come on down!

OK

if语句，乃是有条件地运行一段程序码的绝佳方式。

if语句负责评估条件…然后采取行动

每个if语句都依循相同格式。在我们帮iRock添加cookie时，就已使用过这个格式，现在拆解开来看个详细：

"必须"评估这段程序代码为true或false。

if + (+ Test condition +)

Statement + ;

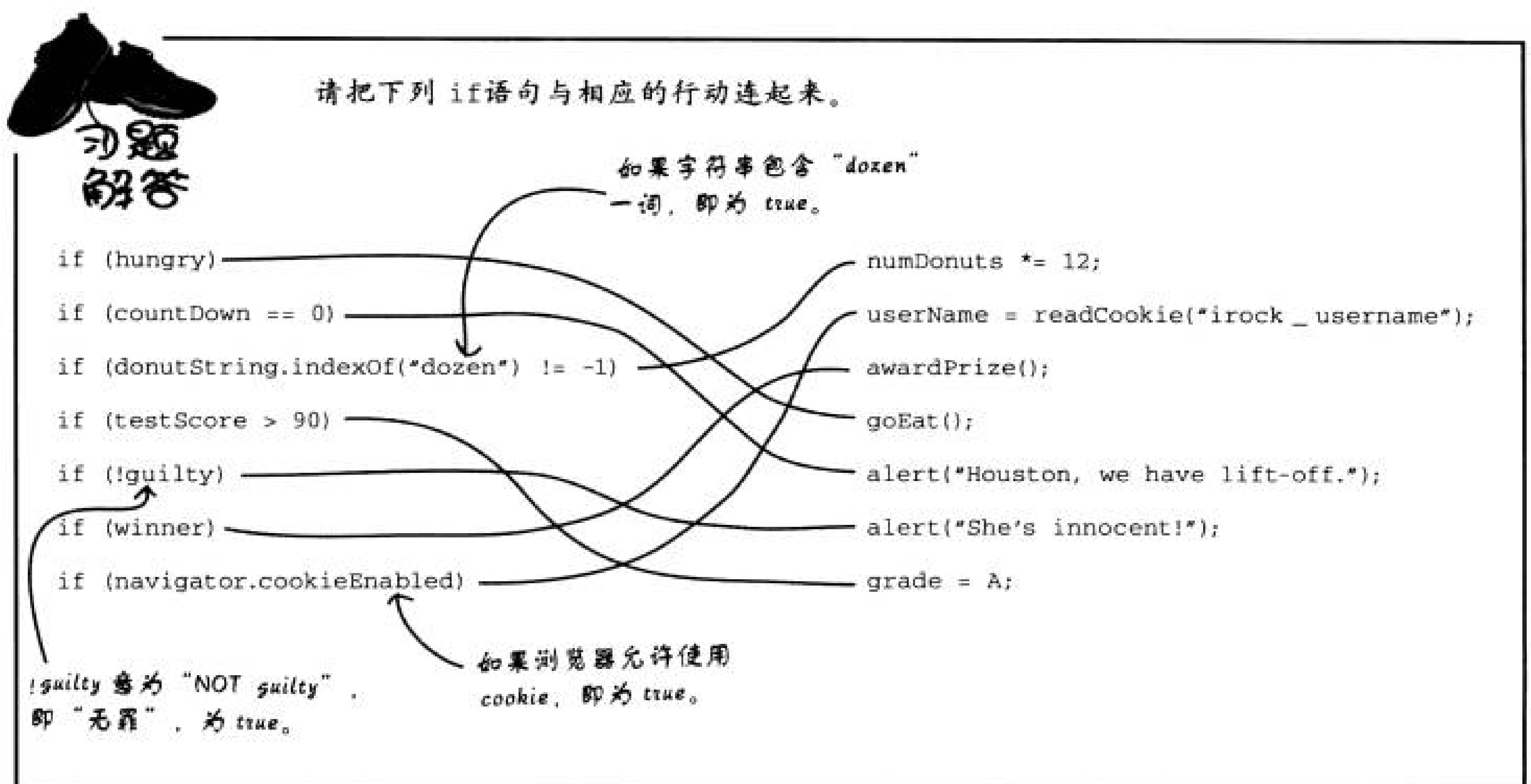
缩排，为了让程序代码更容易阅读。缩排的语句也是"if"的一部分。

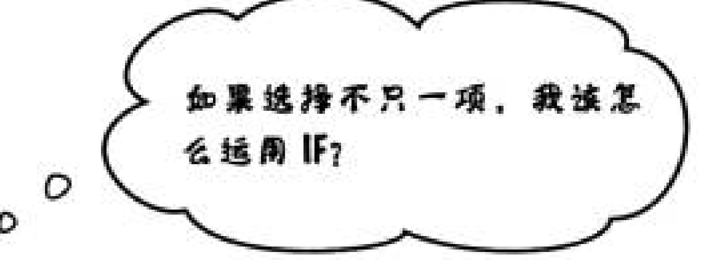
你应该记住关于if语句格式的一些细节。首先，只能利用if运行一段代码，而且这段代码应该缩排，列于if和条件测试句下。虽然不是严格规定，但最好都缩排代码，才容易看出是否身为if语句的一部分。以下是利用if语句作决策的步骤：

1. 以括号围起true/false条件测试句（test condition）。
2. 下一行的程序代码要缩排，在行首加上几个空格即可。
3. 编写条件测试句为true时运行的代码。

请把下列if语句与相应的行动连起来。

```
if (hungry)                                   numDonuts *= 12;
if (countDown == 0)                           userName = readCookie("irock_username");
if (donutString.indexOf("dozen") != -1)       awardPrize();
if (testScore > 90)                           goEat();
if (!guilty)                                  alert("Houston, we have lift-off.");
if (winner)                                   alert("She's innocent!");
if (navigator.cookieEnabled)                  grade = "A";
```





做这件事……或另一件事。

原本以为 JavaScript 妥妥当当地照顾了大小事宜，但有不寻常的事情发生了。实际上，从多个结果中选择一点都不奇怪……巧克力还是香草口味、低咖啡因或一般咖啡豆，看起来很多选择都牵涉到做第一件事"或"另一件事。所以 if 语句可调整为作了决定后，再采取一或两种可能行动……甚至是更复杂的行动。

利用if二选一

调整一下if语句，即可从两种可能结果中二选一。再回到《全民猜猜乐》节目，Eric也面对了相似的天人交战。他面前有两个不同选择，必须二选一。

A

B

A箱是Eric的第一个选择……

……B箱是他的第二个选择。

```
if (true/false test)
  Do something;
else
  Do something else;
```

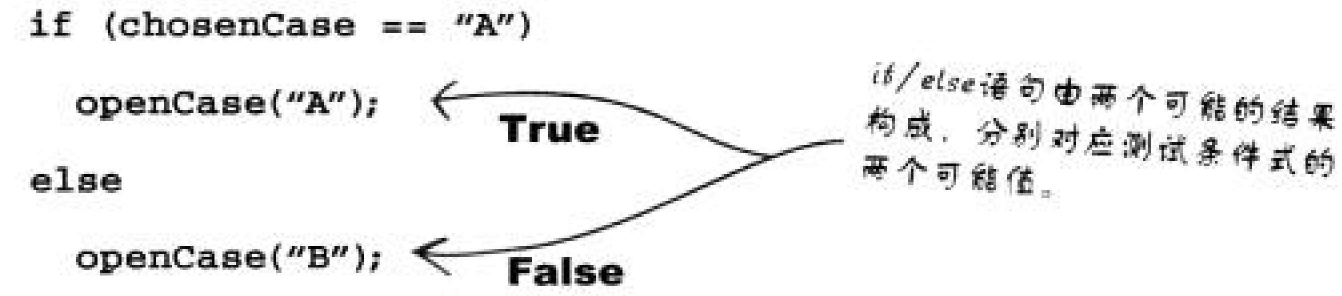
我们的生活很少只会简单地面对**一件事**。if语句为我们（和Eric）提供了选择A箱（Case A）或B箱（Case B）的能力。但JavaScript究竟怎么看待这件事？

```
if (chosenCase == "A")
  openCase("A");

?
```

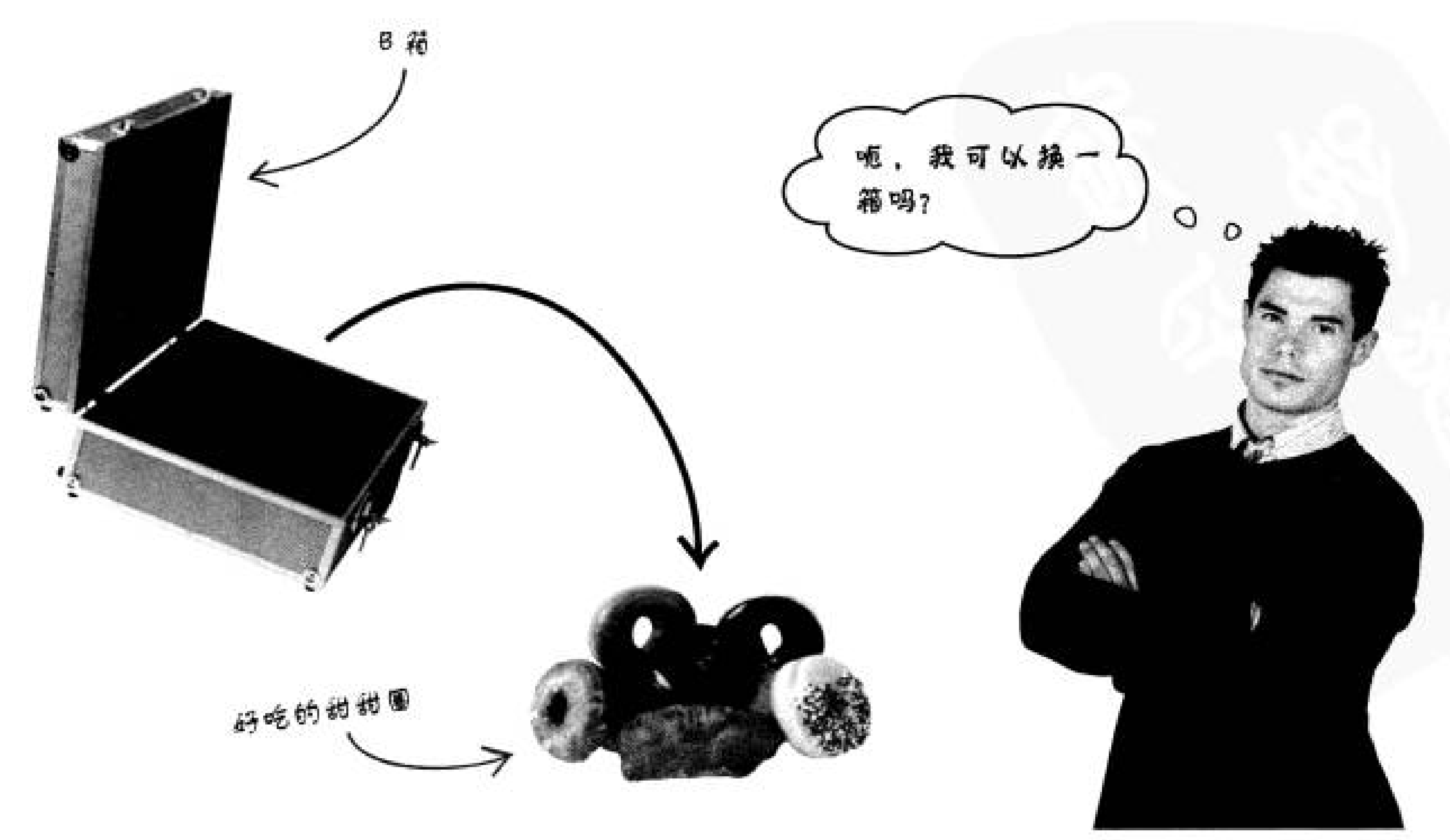
现在呢？如果 chosenCase 不等于"A"，我们该怎么下指示？

你可用if作很多选择

使用if语句从多种行动中选择，就变成了if/else语句，可于true/false条件测试失败时，运行另一段程序代码。也等于指示：如果（if）测试为true，则运行第一段程序代码；否则（else）运行另一段程序代码。

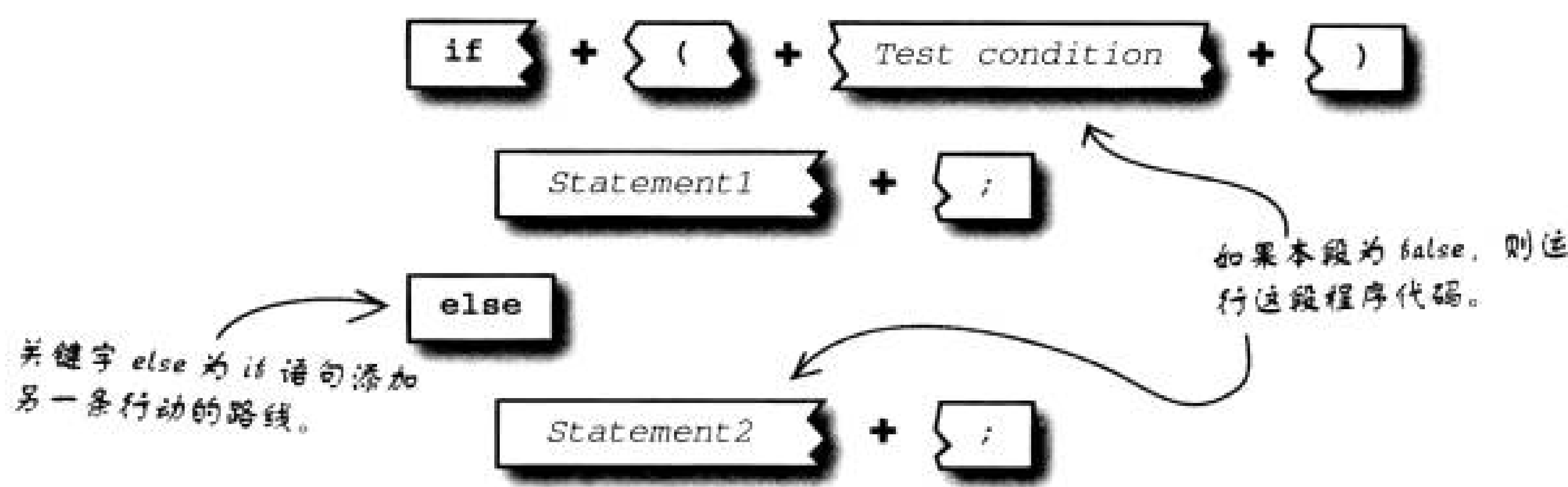


Eric选择B箱，所以chosenCase变量应为“B”。既然第一项测试失败了，即触发if/else语句运行else部分的程序代码。很可惜，Eric选的B箱是一箱甜甜圈，而不是他想要的满箱钞票。



在 if 语句里加个 else

if/else语句的格式与 if语句非常相近，只需添加关键字 else以及条件为 false 时运行的程序代码：



为 if 语句增加第二条行动路线，请依循下列步骤：

1. 把关键字 else 安置于第一组行动的语句（statement）后。
2. 缩排它的下一行程序代码，以便阅读。
3. 编写条件句测试为 false 时欲运行的程序代码。

问：为什么在 if 语句的括号后没有分号？

答： JavaScript规定，每条语句须以分号结尾，if语句亦不例外。然而，if语句不只是if(Test Condition)，也包括条件句为true时运行的程序代码，此时就需加上分号。所以，if语句的确以分号结尾，只要你了解它的正确结构。

问：如果条件句测试结果为 false，而if语句没有else子句，会发生什么问题吗？

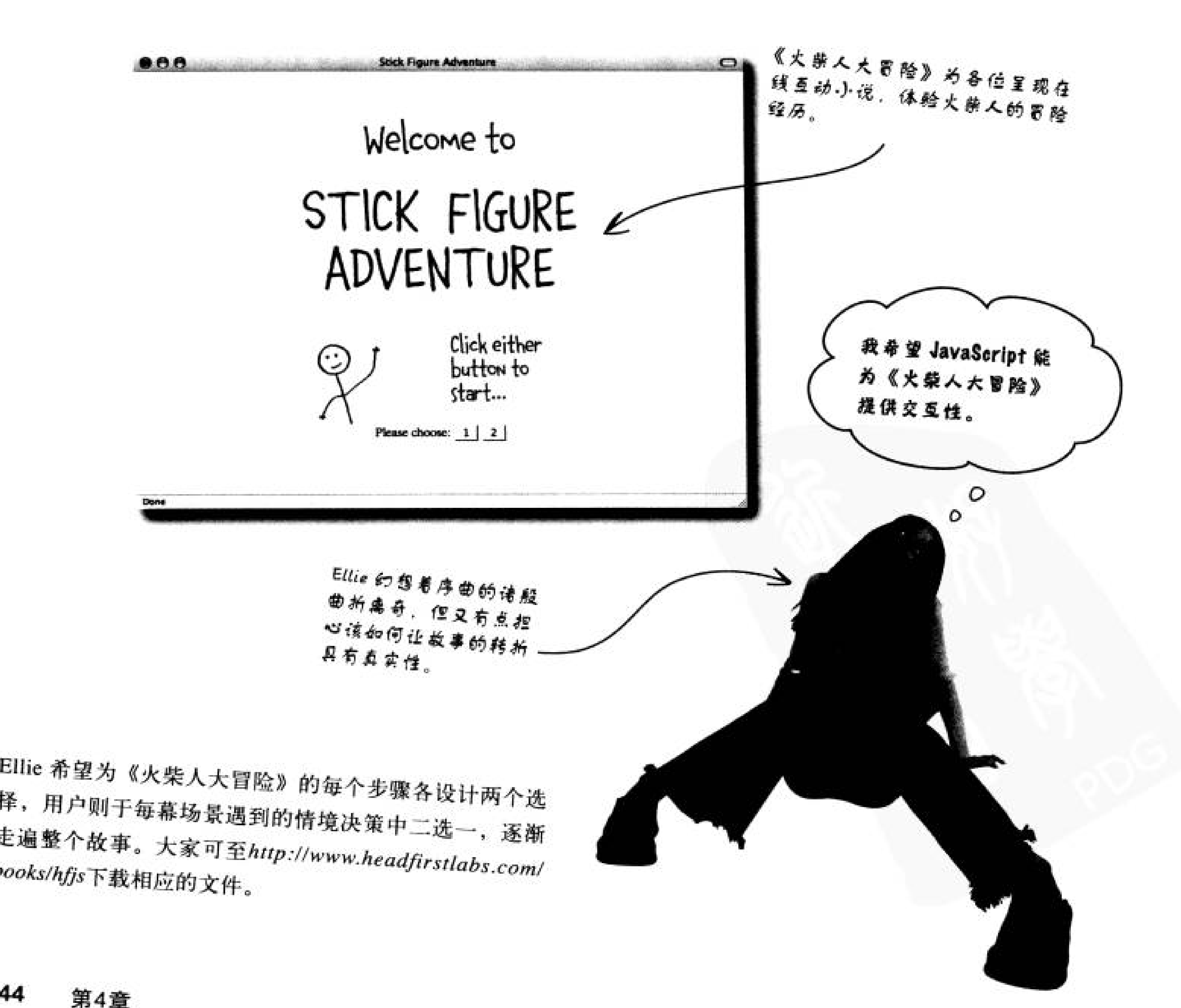
答： 什么问题都不会发生。此时，测试条件句的结果完全不会导致任何行动。

问：有可能使用多个else子句，从更多可能结果里选择吗？

答： 是的。当然可以构筑支持两个以上选项的 if/else语句，但可不是增加另一个 else 子句而已。你需把另一组 if/else语句嵌套在原本的语句里，如果有很多选择，这种做法很快就变得夹缠不清。错不在if/else语句，而是JavaScript对这种需求提供了更好的方式 ——switch/case，本章稍后就会提到。

壮丽的史诗级冒险传奇

Ellie 正在构思《火柴人大冒险》（Stick Figure Adventure）的故事。她计划在故事发展中的每个转折都加上一个选择时刻，所以她希望 JavaScript的决策功能，能为她解决故事走向抉择的问题，这样用户才能在网络上享受这趟有趣的冒险。



Ellie 希望为《火柴人大冒险》的每个步骤各设计两个选择，用户则于每幕场景遇到的情境决策中二选一，逐渐走遍整个故事。大家可至*http://www.headfirstlabs.com/books/hfjs*下载相应的文件。

冒险开始

《火柴人大冒险》由一连串场景构成，每幕场景都有一帧图像与一段情境说明。更重要的一点，每幕场景都设计了两条路（两个选择），作了选择才会移到下幕场景。

磨笔上阵

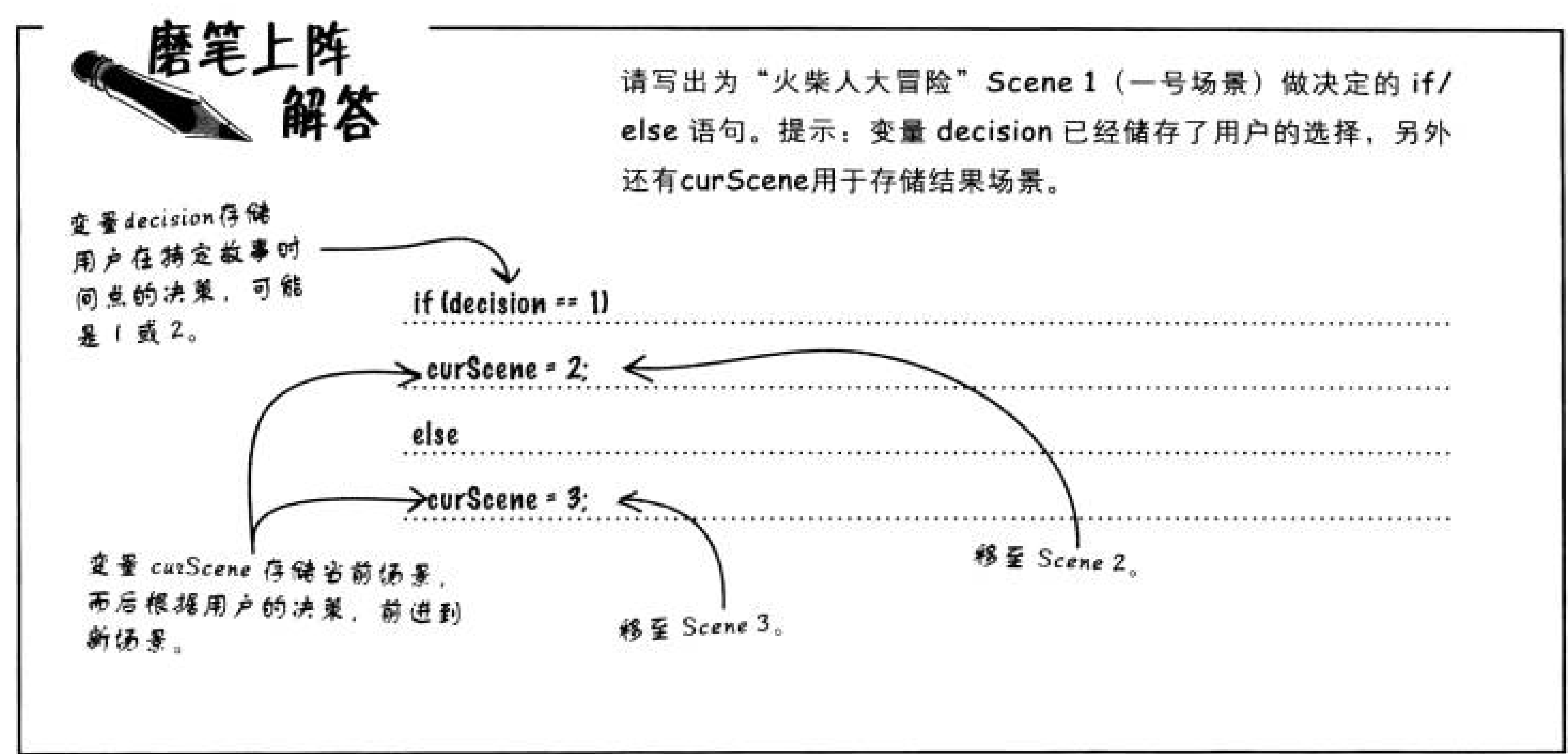
请写出为《火柴人大冒险》Scene 1（一号场景）作决定的 if/else 语句。提示：变量 decision 已经存储了用户的选择，另外还有curScene用于存储结果场景。

....

....

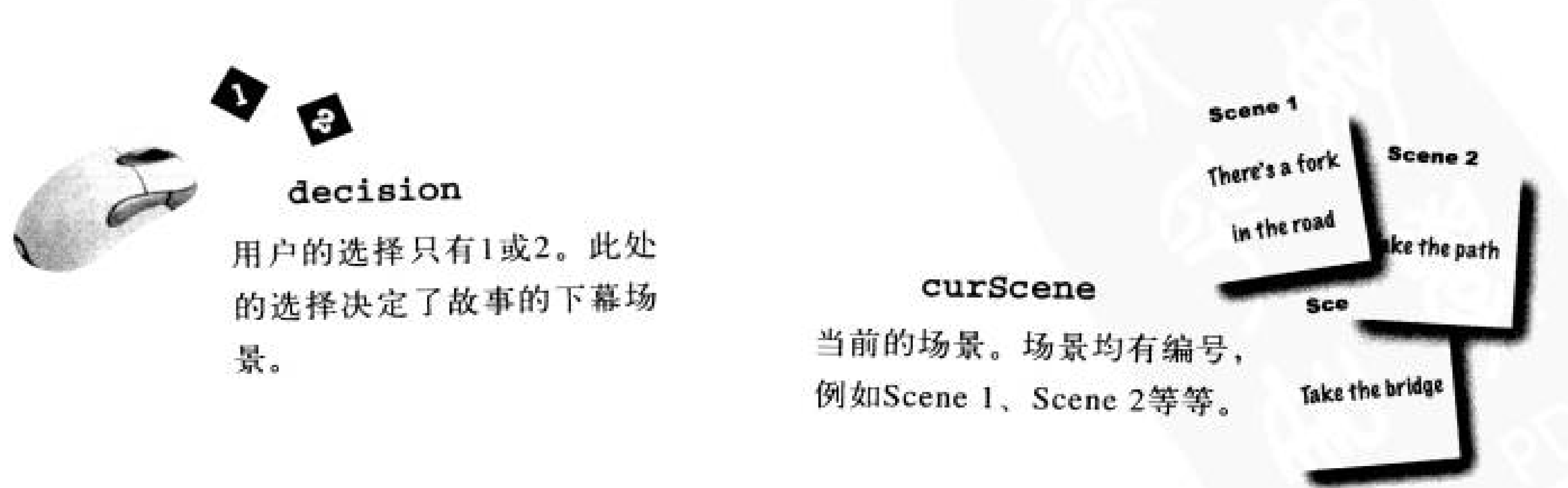
....

....



变量驱动故事

让我们仔细研究《火柴人大冒险》使用的两个变量，这两个变量对于响应用户的决策以及根据决定前进到下幕场景，均十分重要。



变量decision与curScene一起负责存储用户的决策，以作为故事发展的根据。从一幕场景到下幕场景，这段过程均重复一次，故事则随着一个个决策而逐渐展开，一切都因if/else语句才能完成。

但部分故事不见了

if/else语句是《火柴人大冒险》的绝佳决策引擎，但整个故事不是到此就结束了。每幕场景都有特定的影像与文字说明，随着剧情进展而呈现。改变场景编号即足以改变图像，但对文字说明没有助益。

"新"当前场景。

Scene 2

Little house in the woods

用户选择决策1，前进到Scene 2。

Scene 1

Fork in the road

場景图像

场景说明

"You have arrived at a cute little house in the woods."

PNG图像与GIF图像非常接近，但PNG是广获接受的新标准。

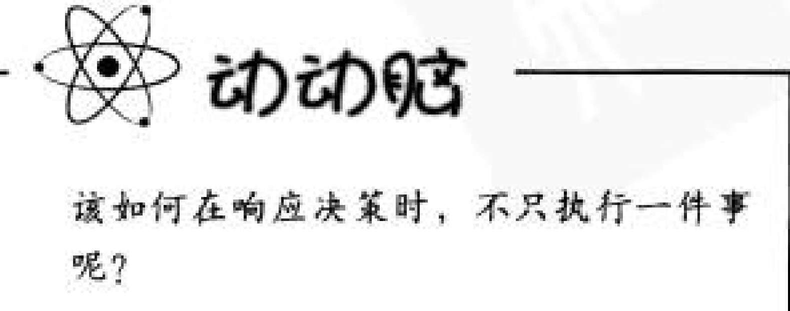
场景编号就足以设定图像，因为图像的文件名包含了场景编号。

场景说明却被搁下了，因为响应换场景的决策时，只采取了一个动作。

sfa_scene2.png

```
document.getElementById("sceneimg").src = "scene" + curScene + ".png";
```

在if/else语句的每个部分只能运行一段程序代码，你受到只能采取一项行动的限制。换句话说，你不能同时呈现图像与显示文字说明。



综合你的JavaScript原力

Ellie需要能在每个if/else语句代码出现分岔时执行多个行动。她需要同时改变场景的编号与文字说明，接下来的两行程序代码才能推动故事：

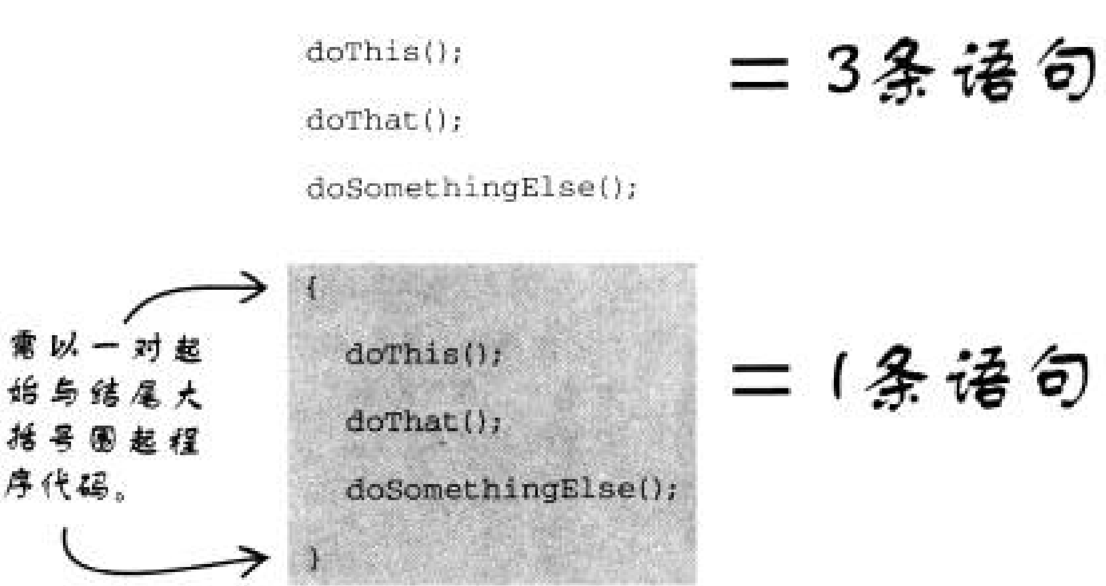
改变场景图像为新选择的场景。

```
document.getElementById("sceneimg").src = "scene" + curScene + ".png";
alert(message);
```

为用户呈现新场景的说明文字。

复合式的语句让我把一大块程序代码视为同一段。

尽管JavaScript只允许我们运行一段程序代码，挑战就在于“一次做到好几样工作”。解决方案称为**复合语句**（compound statement），它把一大块代码包起来，在脚本中看来就像同一段程序代码。要创建复合语句，只需在语句前后加上大括号（{}）。



有了一个复合语句，就可能建立新的、每个行动分支下可执行多种行动的if/else语句。

```
if (chosenDoor == "A") {
  prize = "donuts";
  alert("You won a box of donuts!");
}
else {
  prize = "pet rock";
  alert("You won a pet rock!");
}
```

在if/else语句的分支下，不只执行一种行动。

问：《火柴人大冒险》究竟如何利用变量推动故事？

答：在任意时间点，均由变量curScene存储当前的场景编号。每幕场景则呈现该场景图像及说明，还要提供后续场景的选项，以供用户二选一。变量decision存储用户的选择：1或2。选择完毕后，decision结合curScent即可判定新场景的编号。说得更详细一点，各场景的图像均利用curScent值而改变，各场景的说明则使用alert框呈现。

问：为什么“复合语句里包含多段语句”这点很重要呢？

答：因为JavaScript语言里有很多地方以单一语句为其结构重心。有点像航空公司只准许乘客携带两件行李上机的理由：只要你能塞得进两个行李箱里，没人管你到底塞了多少东西。复合语句有点像登机箱，允许我们把多段语句塞入单一“容器”，该容器对脚本的其他部分而言，宛如单一语句。

问：为什么复合语句未以分号结尾？

答：分号保留给单一语句使用，复合语句不需要。所以，复合语句里出现的单一语句，还是需要加上分号，但复合语句本身则不需要。

问：函数是复合语句吗？

答：好问题！是的，没错。可能有人注意到了，函数代码均以大括号围起。目前，各位可以先把函数当成一个大型复合语句，而且可以把数据传入或传出。

磨笔上阵

重新设计《火柴人大冒险》的第一个if/else决策。现在，使用复合语句，同时设定场景编号与说明信息。

....

....

....

....

....

....

....

....

磨笔上阵解答

重新设计《火柴人大冒险》的第一个if/else决策。现在，使用复合语句，同时设定场景编号与说明信息。

```
if (decision == 1) {
  curScene = 2;
  message = "You have arrived at a cute little house in the woods." ;
}
else {
  curScene = 3;
  message = "You are standing on the bridge overlooking a peaceful stream." ;
}
```

根据用户的决策，调整当前的场景编号。

场景说明的设定，与新的场景相对应。

起始大括号，表示复合语句的开始。

复合语句内的程序代码最好采取缩排格式。

结尾大括号，表示复合语句的结束。

为 Scene 3 设定新的辅助说明。

复习要点

- 使用if语句，有条件地运行一段 JavaScript 代码。
- if 语句的条件测试句，结果必须只能为 true 或 false。
- 使用if/else语句，有条件地运行一或两段JavaScript代码。
- 使用复合语句，如运行单一语句般运行多段JavaScript代码。
- 以一对大括号（{}）围住多个单一语句，即可创建复合语句。
- 复合语句让 if 和 if/else 语句的行动部分能执行多项行动。

冒险开始了

一些复合语句，加上if/else，《火柴人大冒险》踏上成为在线互动故事的第一步了。它有意发展为功能完整的在线冒险，看来前景不错喔。

Scene 1.

① TAKE THE PATH

② TAKE THE BRIDGE

Please choose: 1 2

Your journey begins at a fork in the road.

OK

场景的文字语句以alert框呈现。

按钮1即为选项1，引导用户走向Scene 2。

按钮2即为选项2，引导用户走向Scene 3。

Scene 2.

① WALK AROUND BACK

② KNOCK ON DOOR

Please choose: 1 2

You have arrived at a cute little house in the woods.

OK

Scene 3.

① WALK ACROSS BRIDGE

② GAZE INTO STREAM

Please choose: 1 2

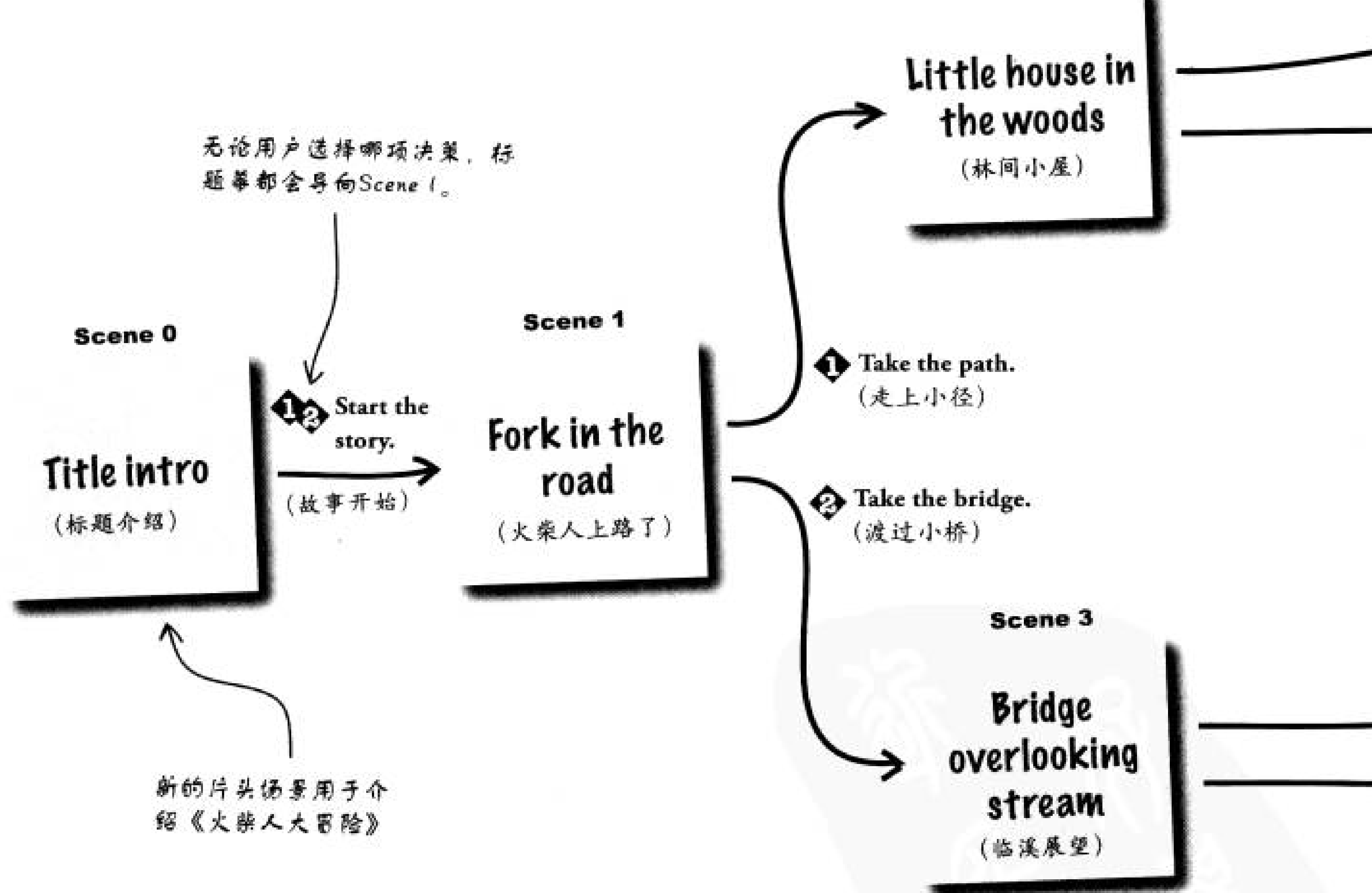
You are standing on the bridge overlooking a peaceful stream.

OK

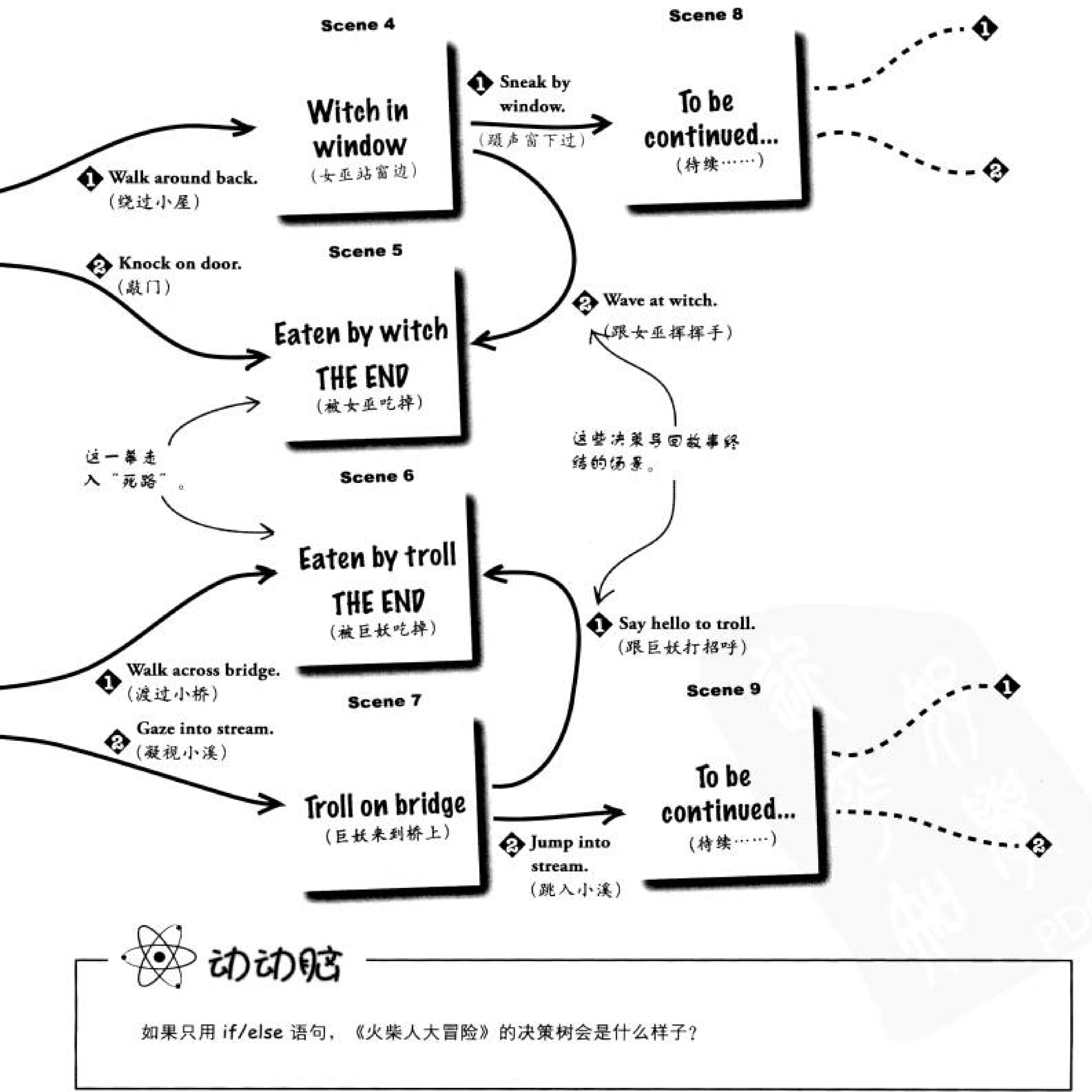
太神奇了！故事的前几个场景看来不错耶！

后续的冒险

单调的决策很难变成有趣的互动故事。但 Ellie 早有计划，她打算加入更多故事场景，让《火柴人大冒险》的情节更为错综复杂。众多场景合起来构成了**决策树**（decision tree），可用于画出故事情节的不同路径。



除了加入更多场景，制造新的剧情转折，Ellie也创建了新的标题介绍幕，准备放在Scene 1前。片头场景Scene 0比较特殊，无论我们选择1或2，都会导向Scene 1。换句话说，Scene 0不是故事的分支点，只是开场的片头。新的剧情幕与开场幕可至*http://www.headfirstlabs.com/books/hfjs*下载。

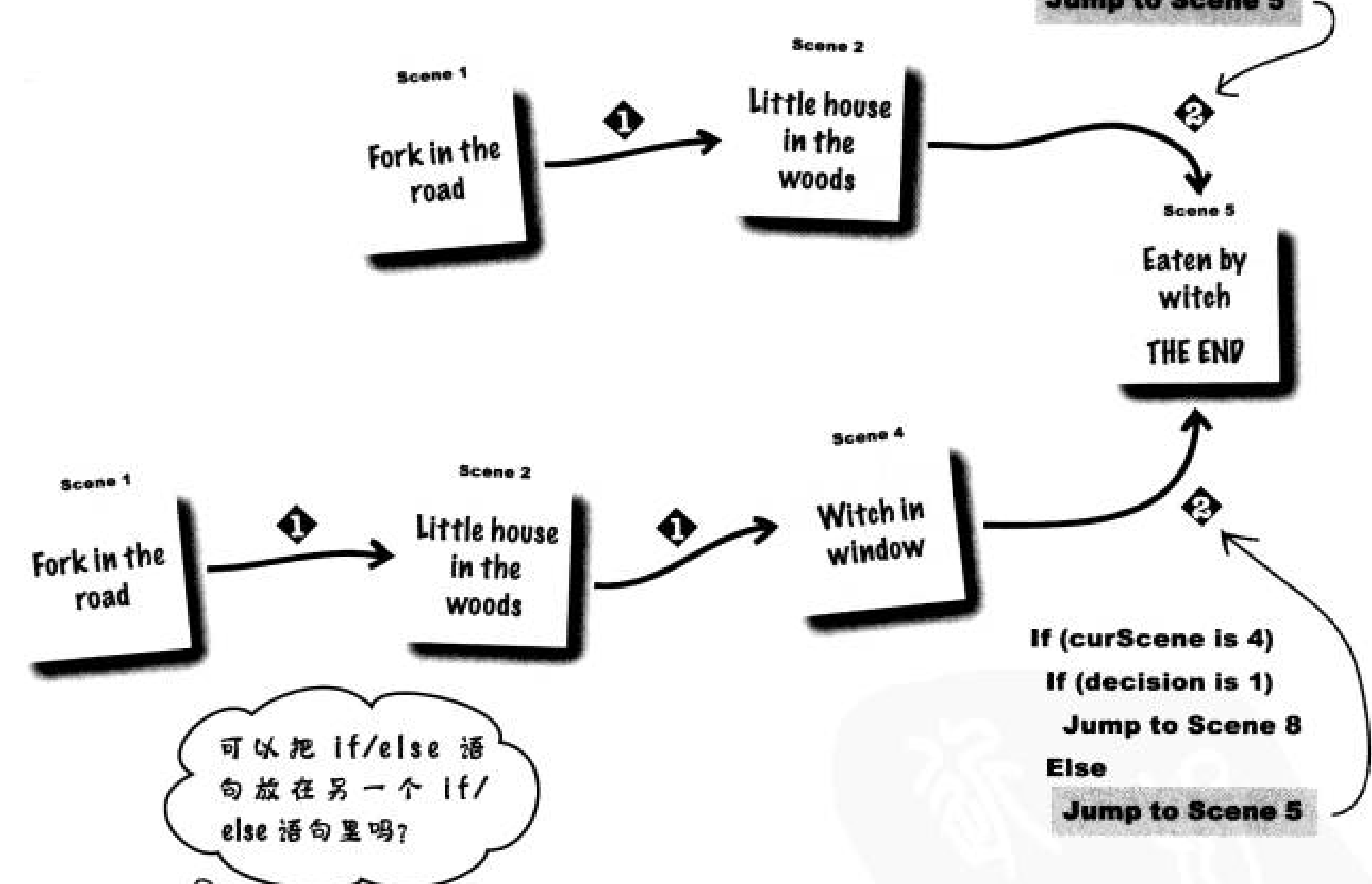


动动脑

如果只用 if/else 语句，《火柴人大冒险》的决策树会是什么样子？

if/else制造出阶梯式决策树

虽然《火柴人大冒险》决策树里的每个决策点都只有两个选项，但Ellie知道后续的每项决策都与前面的决策有关。例如，想进入Scene 5的情节，用户必须遵循特定路线：

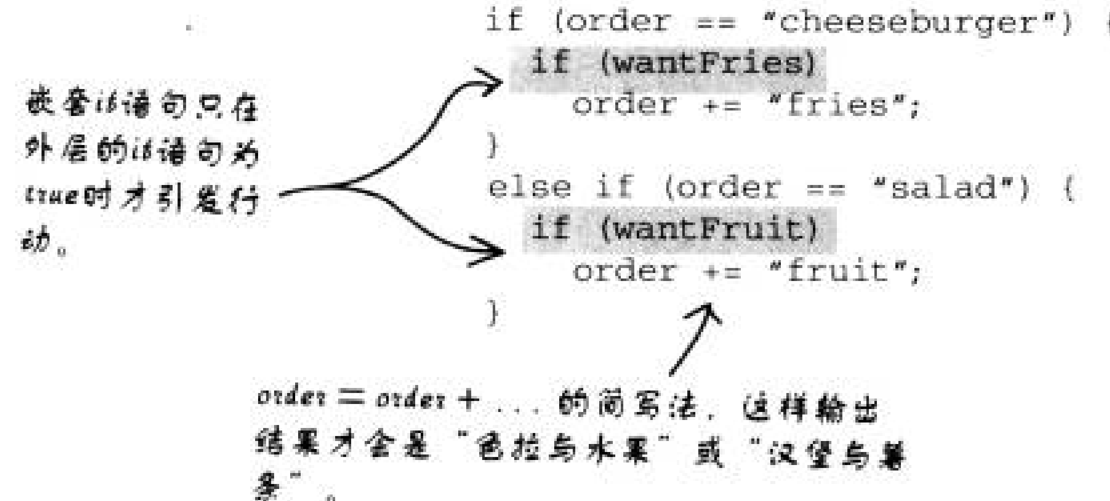


知道用户的选择还不足以决定下个场景。Ellie也需要把当前这一幕的情节纳入考量。使用多个if/else语句，先检查目前的场景编号，再根据用户的决策而行动——这是方案之一。但阶梯式的决策将使if里包着另一个if、再另一个if……似乎是种奇怪的想法。

看起来好像很奇怪，但我们平常也会作阶梯式决策啊！“你想加点一份薯条吗？”这个问题很少跟在点了生菜色拉后出现，对不对？“薯条问题”根据前一个问题的答案而出现，例如“我要一个奶酪汉堡”。这就是阶梯式决策的例子，因为“要薯条吗？”取决于前一个问题“要点奶酪汉堡或色拉？”的答案。

if可以放在别的if里

在JavaScript里，把if放在另一个if里完全合法。但请记得，if语句仍只是语句，只是作为另一个if语句的行动工具。换句话说，问题可以接在另一个问题后。这种if放在另一个if里的语句，称为**嵌套if语句**（nested if statement）。



磨笔上阵

请为《火柴人大冒险》的Scene 0与Scene 1设计决策代码，并确认于必要的时机使用if和if/else语句。

....

....

....

....

....

....

....

....

....

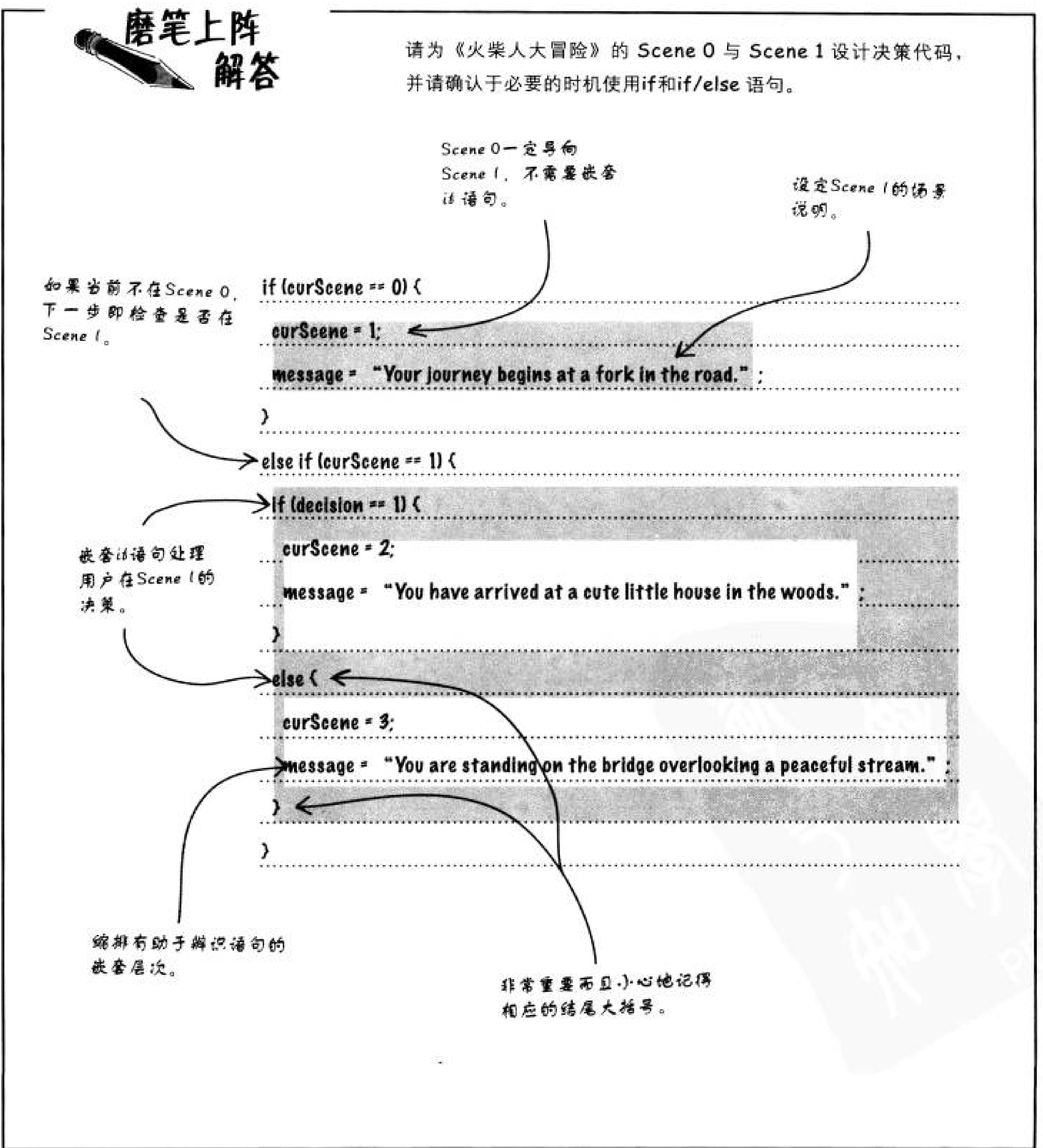
....

....

....

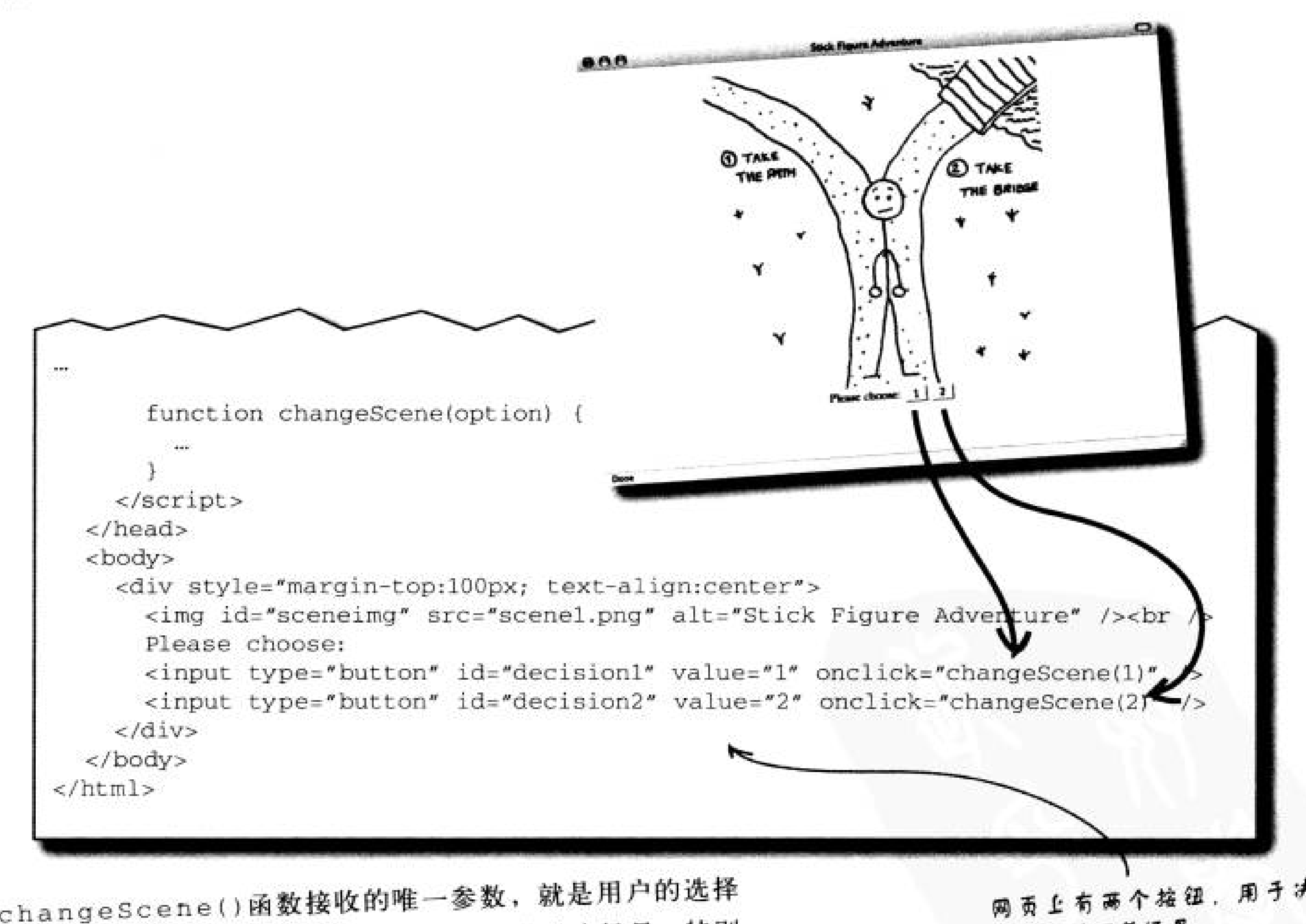
....

....


磨笔上阵
解答
请为《火柴人大冒险》的 Scene 0 与 Scene 1 设计决策代码，并请确认于必要的时机使用if和if/else 语句。
Scene 0一定导向 Scene 1，不需要嵌套 if 语句。
设定Scene 1的场景说明。
如果当前不在Scene 0，下一步即检查是否在 Scene 1。
if (curScene == 0) {
curScene = 1;
message = "Your journey begins at a fork in the road." ;
}
else if (curScene == 1) {
if (decision == 1) {
curScene = 2;
message = "You have arrived at a cute little house in the woods." ;
}
else {
curScene = 3;
message = "You are standing on the bridge overlooking a peaceful stream." ;
}
}
嵌套if语句处理用户在Scene 1的决策。
缩排有助于辨识语句的嵌套层次。
非常重要而且小心地记得相应的结尾大括号。

函数掌控你的网页

《火柴人大冒险》网页上的按钮（“1”与“2”），就是用户前往故事下幕场景的方式。当用户决定按下某个按钮时，changeScene()函数即获得调用，并根据按钮代表的选择而换幕。



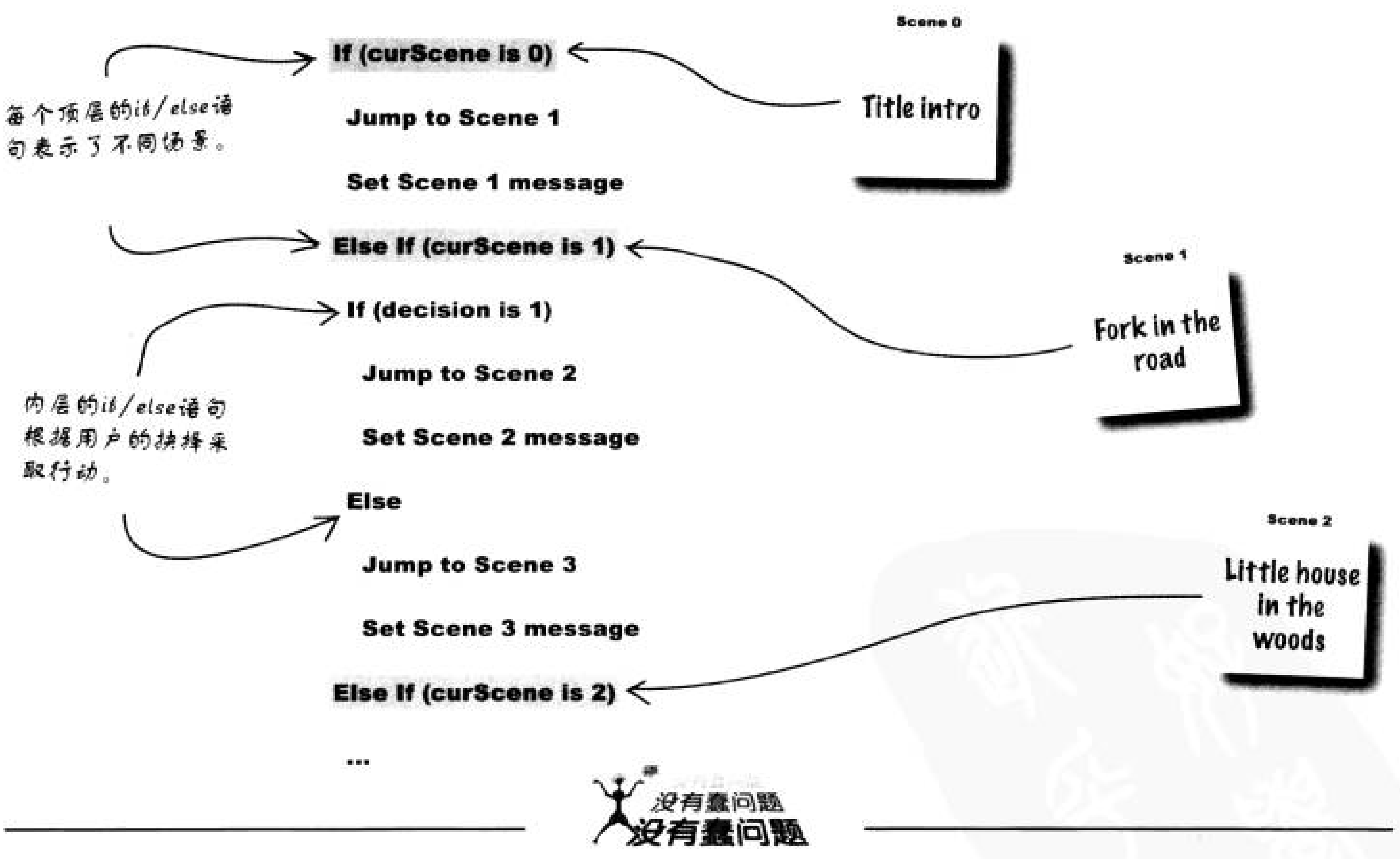
```
...
      function changeScene(option) {
        ...
      }
    </script>
  </head>
  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="sceneimg" src="scene1.png" alt="Stick Figure Adventure" /><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```


changeScene()函数接收的唯一参数，就是用户的选择（“1”或“2”）。函数只需要这个信息就能改变场景。特别是处理三项工作的changeScene()函数：

1. 设定变量 curScene 为下幕场景的编号。
2. 设定变量 message 为下幕场景的文字说明。
3. 根据变量 curScene 改变场景的图像，并呈现文字说明。

利用伪代码规划冒险蓝图

对于如何以JavaScript代码建立changeScene()函数，从而实现《火柴人大冒险》的决策树，Ellie已经很有概念了。但这么多决策可能使得实际编程时一团混乱。有时候，先用伪代码写下决策树也是个有益的手法。伪代码（pseudocode）是种较不严谨、较易阅读，也是非常不正式的脚本代码表达方式。击败伪代码后，JavaScript代码将更为清晰易懂，编码时也较少产生错误。



没有蠢问题

问：伪代码看起来很像JavaScript代码。为何多此一举啊？

答：不做伪代码也没关系，我们的用意是想把复杂的过程——决策树转换为JavaScript代码的过程予以简化，同时又把制造错误的风险降到最底。因为伪代码的细节不如真正的JavaScript代码详尽，所以我们可以先把精力放在每幕场景连接时的逻辑上，而不用分心注意大括号和分号的位置是否正确。等你熟悉伪代码后，将之转换为JavaScript代码其实相当轻而易举。

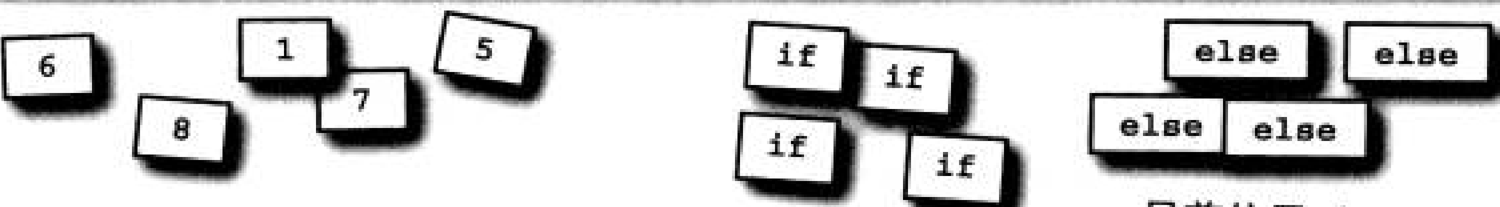
问：制作嵌套if语句时，一定要用大括号吗？

答：不一定。事实上，如果只要在 if 语句里塞入另一个 if 语句，而不包括其他代码，的确可以不加上大括号，因为此时在技术上不需要复合语句。不过，若是复杂的嵌套if语句，使用大括号则会带来清楚表示嵌套结构的优势（即使并非严格需要）。

JavaScript 冰箱磁铁

《火柴人大冒险》的 changeScene() 函数少了一些代码。请利用本页的小磁铁，制作出第 152 页的决策树。对了，有些场景决策代码并未列出来 —— 有几幕场景故意留下来不写。

```
function changeScene(option) {
  var message = "";

 ...........(curScene == 0) {
    curScene =........;
    message = "Your journey begins at a fork in the road.";
  }
  ...
  ......................(curScene == 3) {
    .......(option == 1) {
      curScene = ........;
      message = "Sorry, a troll lives on the other side of the bridge and you" +
        "just became his lunch.";
    }
    .......      {
      curScene = ........;
      message = "Your stare is interrupted by the arrival of a huge troll.";
    }
  }
  ......................(curScene == 4) {
    if (option == 1) {
      curScene = ........ ;
    }
    .............{
      curScene = ........ ;
      message = "Sorry, you became part of the witch's stew.";
    }
  }
  ...

  document.getElementById("sceneimg").src = "scene" + curScene + ".png";
  alert(message);
}
```



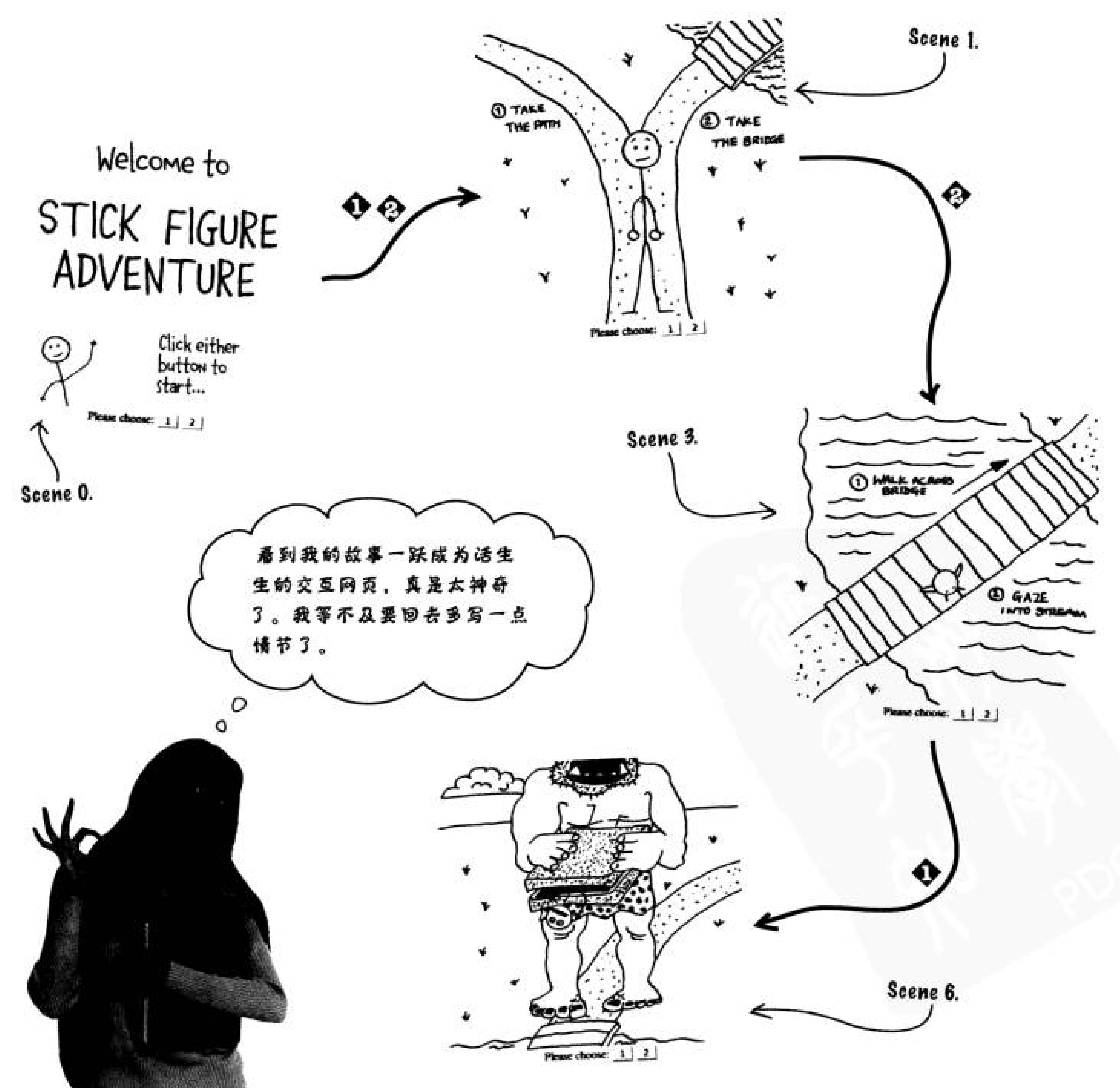
JavaScript 冰箱磁铁解答

《火柴人大冒险》的changeScene()函数少了一些代码。请利用本页的小磁铁，制作出第152 页的决策树。对了，有些场景决策代码并未列出来 —— 有几幕场景故意留下来不写。

```
function changeScene(option) {
  var message = "";

  if (curScene == 0) {
    curScene = 1;
    message = "Your journey begins at a fork in the road.";
  }
  ...
  else if (curScene == 3) {
    if (option == 1) {
      curScene = 6;
      message = "Sorry, a troll lives on the other side of the bridge and you" +
        "just became his lunch.";
    }
    else {
      curScene = 7;
      message = "Your stare is interrupted by the arrival of a huge troll.";
    }
  }
  else if (curScene == 4) {
    if (option == 1) {
      curScene = 8;
    }
    else {
      curScene = 5;
      message = "Sorry, you became part of the witch's stew.";
    }
  }
  ...

  document.getElementById("sceneimg").src = "scene" + curScene + ".png";
  alert(message);
}
```

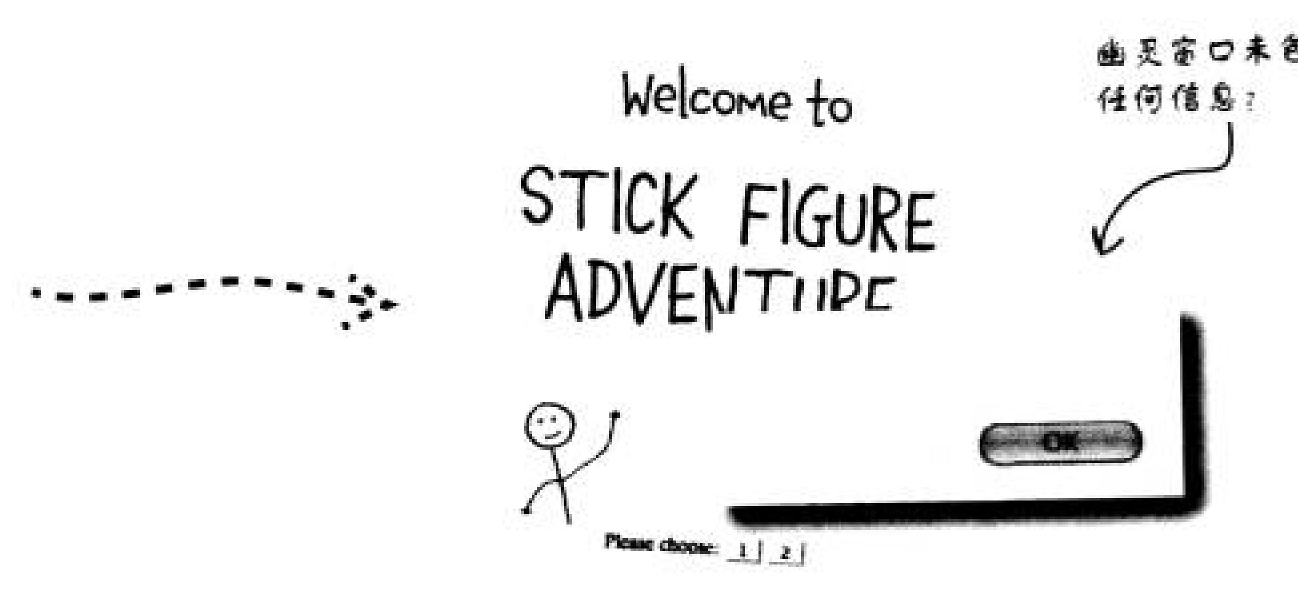
踏上火柴人的冒险旅程

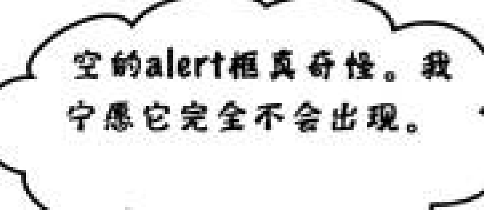
《火柴人大冒险》的脚本现已反映整个决策树，我们可以踏上几段不同路线，体验故事。举例来说：



火柴人不平衡

糟糕，Ellie 的《火柴人大冒险》已经遇到了问题。她拜托朋友们帮忙测试刚做好的几页故事，结果，不少人遇到了没有任何信息的窗口。这个“幽灵窗口”只在上一场冒险结束，准备重新开始新冒险时出现。所以问题出在从其他幕移向 Scene 0 的时候。





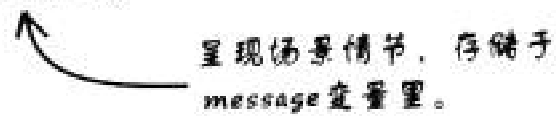
调查结果发现，只有两幕场景能通向Scene 0：Scene 5与Scene 6。因为这两幕都是故事的结局，走到结局后，能够从头开始 —— 回到Scene 0 —— 才算合理。所以Ellie在changeScene()函数中另外处理这两个场景导回Scene 0的部分。

```
else if (curScene == 5) {
  curScene = 0;
}
else if (curScene == 6) {
  curScene = 0;
}
```

Scene 5与Scene 6均设定了变量curScene，但并未处理message变量。

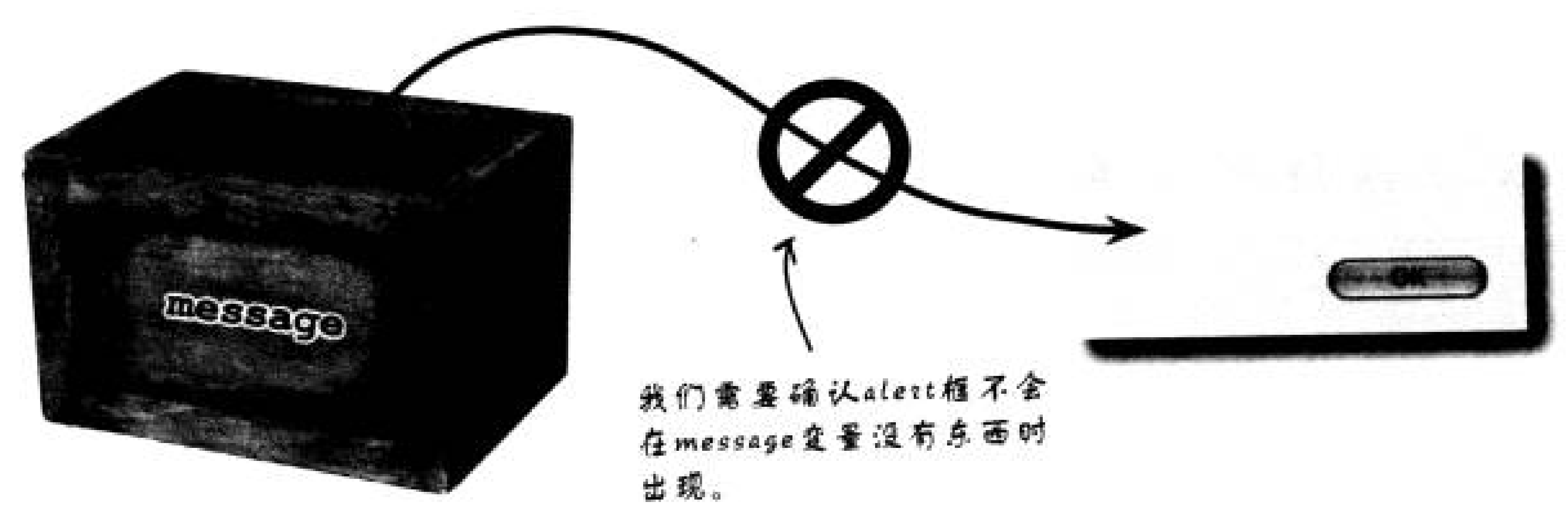
虽然在这么简单的代码中看不出什么，但如果看看changeScene()函数最底端的处理场景图像改变与呈现场景说明文字的部分……

```
document.getElementById("sceneimg").src = "scene" + curScene + ".png";
alert(message);
```



!= 嘘，我没有什么能告诉你……

《火柴人大冒险》代码的问题，在于它必定呈现每幕场景的说明文字与alert框，就算没有信息可供呈现，如重新由Scene 0开始冒险时的情况。不过，该如何检查变量message是否存储了情节的文字说明呢？



解决方案需要在弹出alert框前检查message变量是否为空字符串（"　"）。换句话说，只在 message 变量不等于（not equal）空字符串时，才呈现alert框。没错，听起来好像是个退步的解决方式，但请记得，现在的状况需要一个true/false测试，以判断是否呈现alert框。

相等运算符（==）用于检查两项事物是否相等，不相等运算符（!=）则可检查两项事物是否不相等。

除了Scene 6，其他幕都使这个条件判定为true。

```
if (curScene != 6)
  alert("Thankfully, you haven't been eaten by the troll.");
```

磨笔上阵

请为《火柴人大冒险》重新编写弹出alert框（及每幕场景说明文字）的代码，这一次只在message变量确实存有文本数据时，才呈现alert框。

....

....

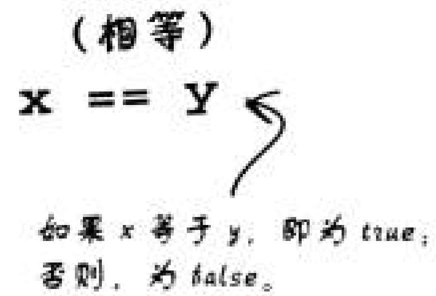
请为《火柴人大冒险》重新编写弹出alert框（及每幕场景说明文字）的代码，这一次只在message变量确实存有文本数据时，才呈现alert框。

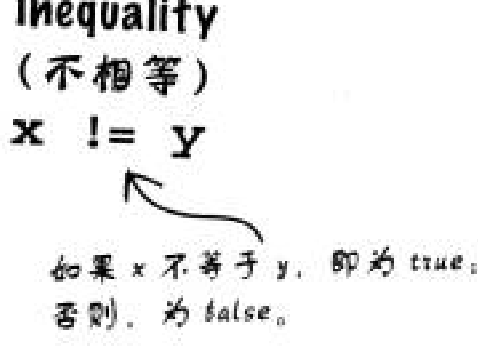
只在 message 包含非空字符串的文本时，条件判断才为 true。

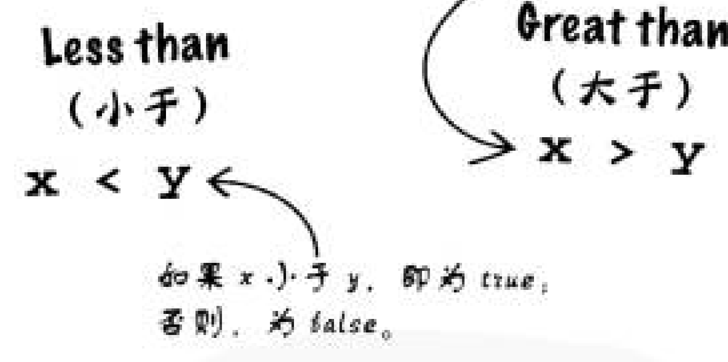
```
if (message != " ")
  alert(message);
```

由比较运算符精心制作的决策成品

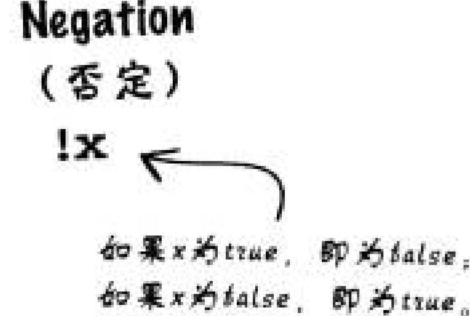
相等与不相等运算符，它们在JavaScript代码里建立条件测试，也是作决策的好帮手，但能够助你一臂之力的比较运算符可不只它们两个而已哦！

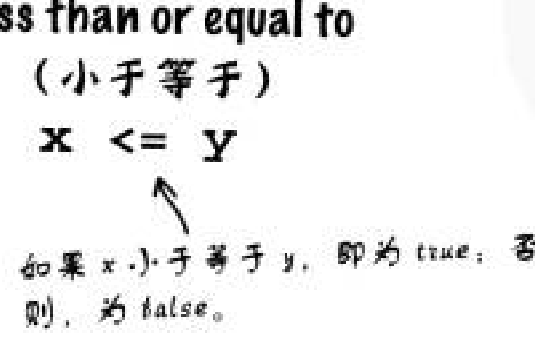


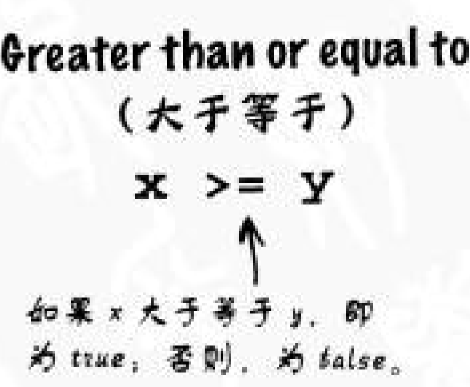




如果 x 大于 y，即为 true；否则，为 false。



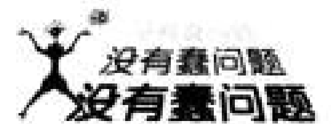




JavaScript 的运算符（operator），例如这些比较运算符，可用于建立表达式（expression）——一块最终结合成单一值的JavaScript代码。表达式由产生boolean值（true/false）的比较运算符组成，因而适合构造使用if/else语句的决策逻辑。

= 与 == 非常不一样。

在比较两个值前，务必确认你使用了 ==，而非 =。否则将变成指派数据值，有可能建立任何全新、不寻常的程序错误。



问：为什么否定运算符只用到一个值？

答：大多数比较运算符需要两个值，但否定运算符只需要一个。它的工作也很简单：反转运算符的意义。所以，true变成false，false变成true。

问：我看过否定运算符应用于并非进行比较运算的值。这一切如何运作？

答：于非比较运算中，使用否定运算符的代码，其实巧妙利用了 JavaScript 判断值是否为 true 的细部设定。如果在预期为比较运算的情境中使用了非比较值，只要不是null、0、空字符串（"　"），或"未定义"，任何值都会被解读为true。换句话说，从比较运算的观点来看，数据的出现就代表true。所以，当你看到否定运算符用于非比较运算值时，null、0、""的意义均反转为 true，其他值则被反转为false。

问：等一下，null 是何方神圣？

答：null是个特殊的 JavaScript 值，表示"缺少数据"。在对象的情境中比较容易理解，所以等到第 9 章与第 10 章再来讨论吧！

下例代码能呈现献给《火柴人大冒险》的**正面**鼓励信息。请问，a、b、c、d 各应为什么值，才能成功写出信息呢？

```
var quote = "";

if (a != 10)
  quote += "Some guy";
else
  quote += "I";
if (b == (a * 3)) {
  if (c < (b / 6))
    quote += "don't care for";
  else if (c >= (b / 5))
    quote += "can't remember";
  else
    quote += "love";
}
else {
  quote += "really hates";
}
if (!d) {
  quote += "Stick Figure";
}
else {
  quote += "Rock, Paper, Scissors";
}

alert(quote + "Adventure!");
```

a = ...

b = ...

c = ...

d = ...

下例代码能呈现献给《火柴人大冒险》的正面鼓励信息。请问，a、b、c、d 各应为什么值，才能成功写出信息呢？

```
var quote = "";

if (a != 10)
  quote += "Some guy";
else
  quote += "I";
if (b == (a * 3)) {
  if (c < (b / 6))
    quote += "don't care for";
  else if (c >= (b / 5))
    quote += "can't remember";
  else
    quote += "love";
}
else {
  quote += "really hates";
}
if (!d) {
  quote += "Stick Figure";
}
else {
  quote += "Rock, Paper, Scissors";
}

alert(quote + "Adventure!");
```

a必须等于10

b必须等于10×3

c必须等于5

a = 10

b = 30

c = 5

d = false

d必为false，才能使!d为true。d也可以是null。

我们需要的鼓励信息。

I love Stick Figure Adventure!

OK

注释、占位符、说明文档

《火柴人大冒险》也是脚本程序里具有未完成代码的好例子，因为这个故事还没说完。例如Scene 8与Scene 9都是“待续”，等着Ellie提出更多创意。如果可以标明这个地方有未完成的代码就好了，这样才不会忘记日后补充细节。JavaScript的**注释**（comment）即可在代码里加上注记，完全不会影响程序运行。

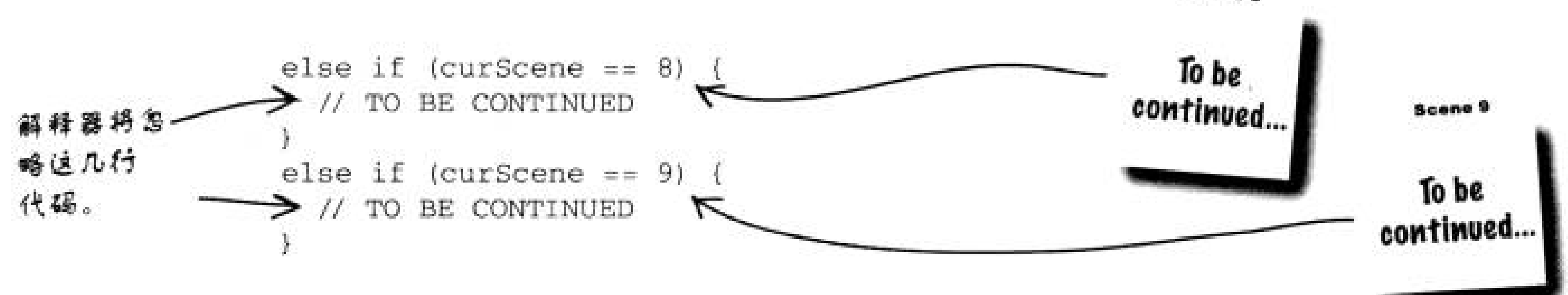
注释的起始方式为一对斜线//。

// + Comment

注释可为任何文本——JavaScript解释器将忽略所有注释文本。

JavaScript的注解以//起始

注释以//起始，延伸至一行的结尾。创建注释作为占位符时，只需在斜线后注记还有更多程序代码待补。



注释不只可用于占位，其实更常用于制作程序代码的说明，使程序更有组织且容易了解。**现在**知道某段程序代码的用途，不代表以后一直都记得它的用途，而且很可能将由其他人接管你设计的程序代码。事先说明程序的用途，对大家都有好处。

这段注释说明变量的初始化。

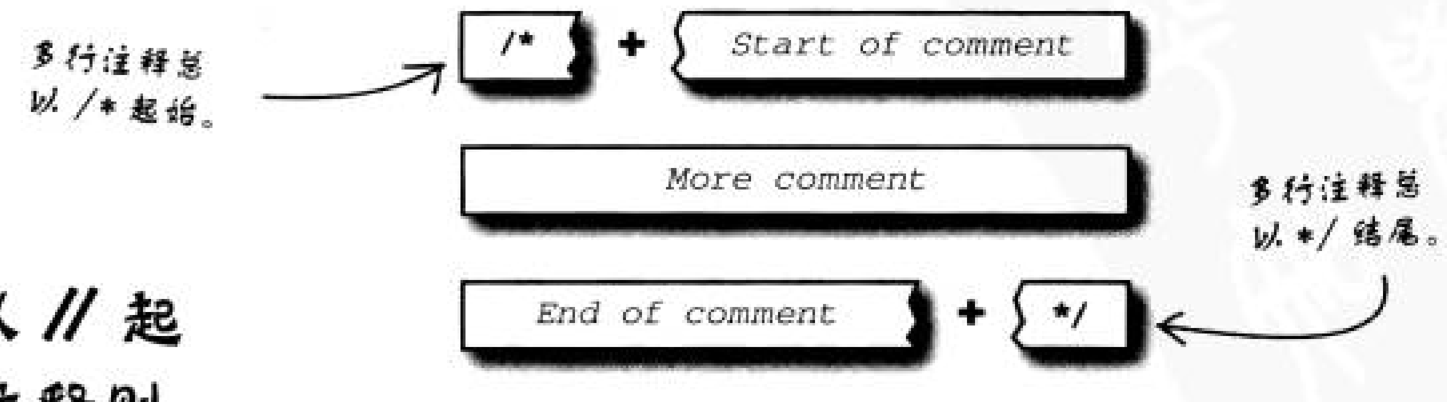
```
// Initialize the current scene to Scene 0 (Intro)
var curScene = 0;
```

变量curScene的初始化，因为有了详细的注释而显得更清楚。相似的注释可用于清楚说明变量message的初始化。

再一次，注释澄清了后续程序代码的用途。

```
// Clear the scene message
var message = "";
```

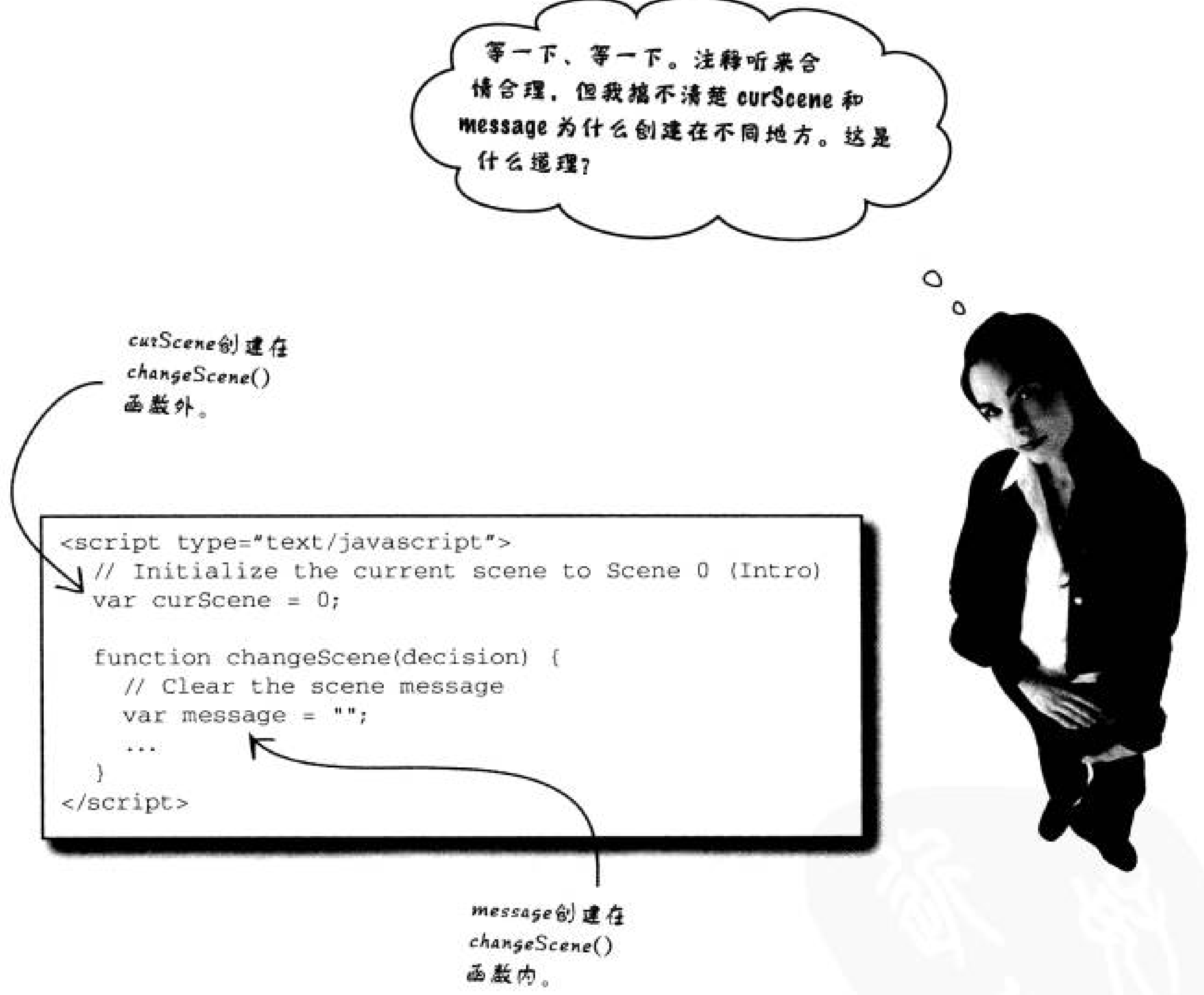
如果你需要跨越多行的注释，也可以创建多行注释。



单行注释以//起始，多行注释则用/*与*/围起来。

多行注释想有多长，就可有多长，只需以/*起始并以*/结尾。

```
/* All three of these lines of code are one
   big comment. Seriously, Im not kidding.
   No joke, this is still part of the comment. */
```



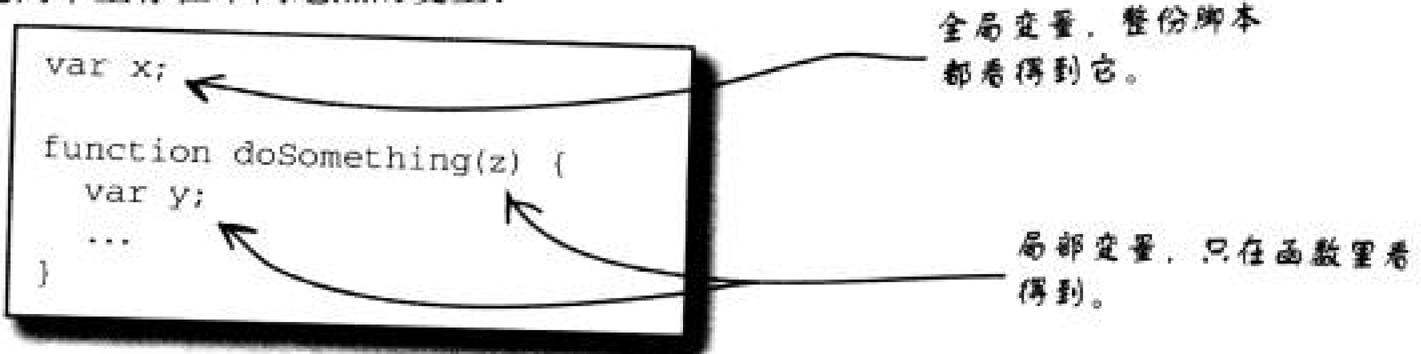

```
<script type="text/javascript">
  // Initialize the current scene to Scene 0 (Intro)
  var curScene = 0;

  function changeScene(decision) {
    // Clear the scene message
    var message = "";
    ...
  }
</script>
```

变量的……地点、地点，重点就是地点

在 JavaScript 的世界里，地点就像现实世界的房地产一样重要。在《火柴人大冒险》的范例中，变量创建的位置非常重要。换句话说，curScene创建在changeScene()函数外，message却创建在同一函数内，这并非意外。原因在于**作用域**（scope），作用域掌控变量的生命周期，还有程序代码可以访问变量的时间。

作用域与上下文：数据的生活

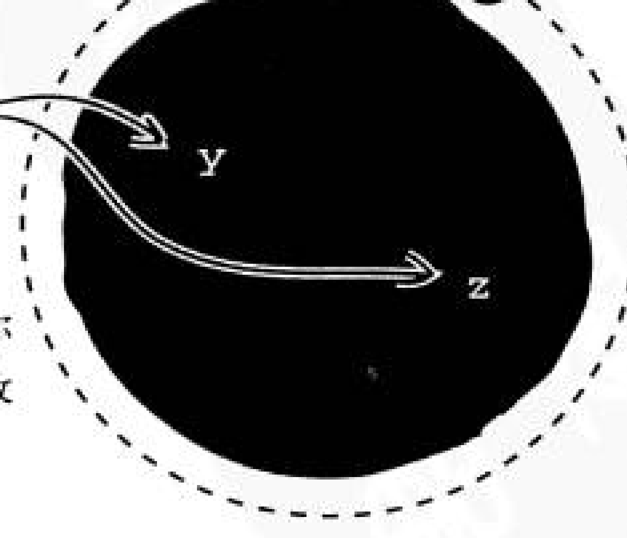
在JavaScript里，作用域（scope）代表**数据的上下文**（context of data），例如数据生存的地点以及访问数据的方式。根据作用域的不同，有些数据能被脚本的其他部分看到，但有些数据则只限某块代码使用，例如某个函数。下例就是两个生存在不同地点的变量：



在这段范例代码中，x称为全局变量（global variable），因为它创建在任何函数或其他程序代码块外，因此能被整份脚本看到。更重要的一点，只要脚本正在运行，x都还“活着”。但y不像x，它是个局部变量（local variable），这个变量的可见度只限于函数 doSomething()里。还有，y只于doSomething()函数运行时“存在”——当函数运行时，y被创建，函数结束时它则被摧毁。

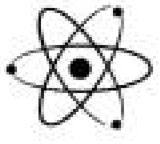
目前看来还不错，不过，doSomething()的参数z又处于什么位置？从结果而言，函数参数的行为就像已经初始化的局部变量。所以，z的作用域与y的相同，只能从函数内访问。

在脚本的整个生命周期中，全局变量都一直存在。



局部变量依其作用域的指示而创建或摧毁。

数据可见度属于“有需要才要知道”的知识，也就是说只要可能，你都应该限制访问的程度。这有助于预防数据意外被不该访问它的代码改变。换成实践用语，就是尽可能使用局部变量。



动动脑

全局变量与局部变量，如何加入《火柴人大冒险》的代码中？

检查变量作用域

有了何谓局部变量与全局变量的知识，我们回头看看《火柴人大冒险》的变量，现在更能理解变量为何创建在不同地方了。

每次跑过changeScene()函数都会重设 message 变量，所以它如局部变量般运作。

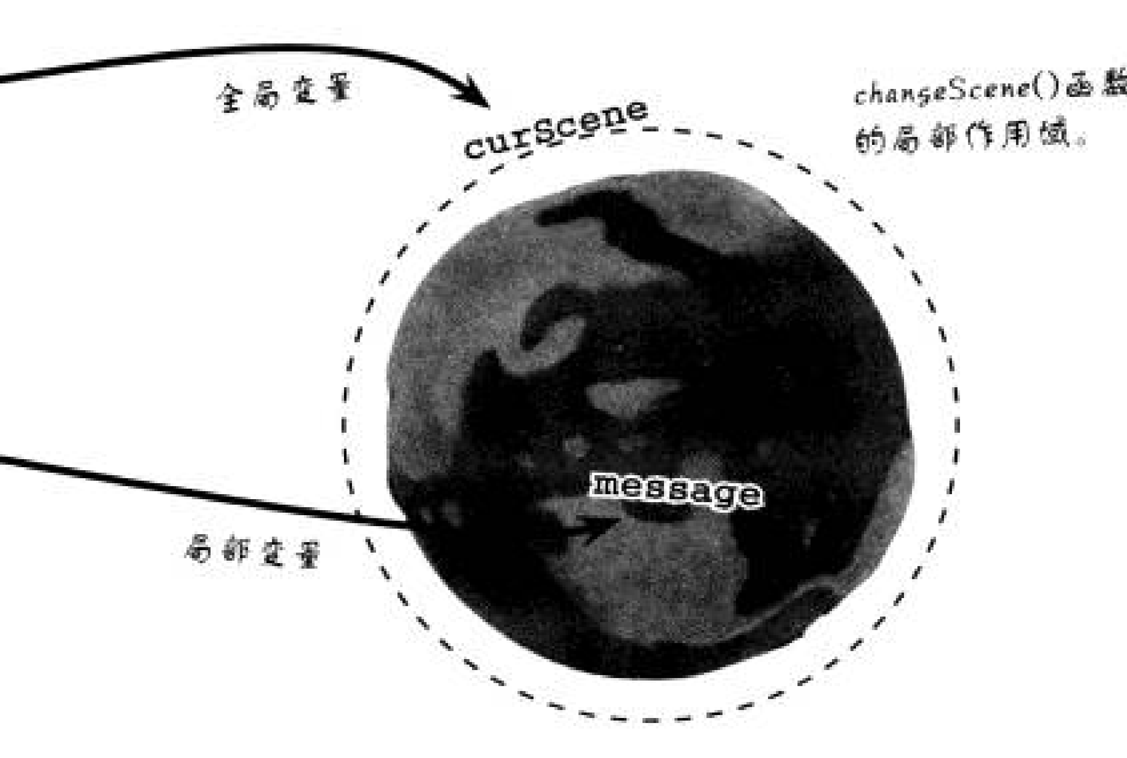
在 changeScene() 函数的调用间必须维持curScene 的变量值，所以它必须是全局变量。

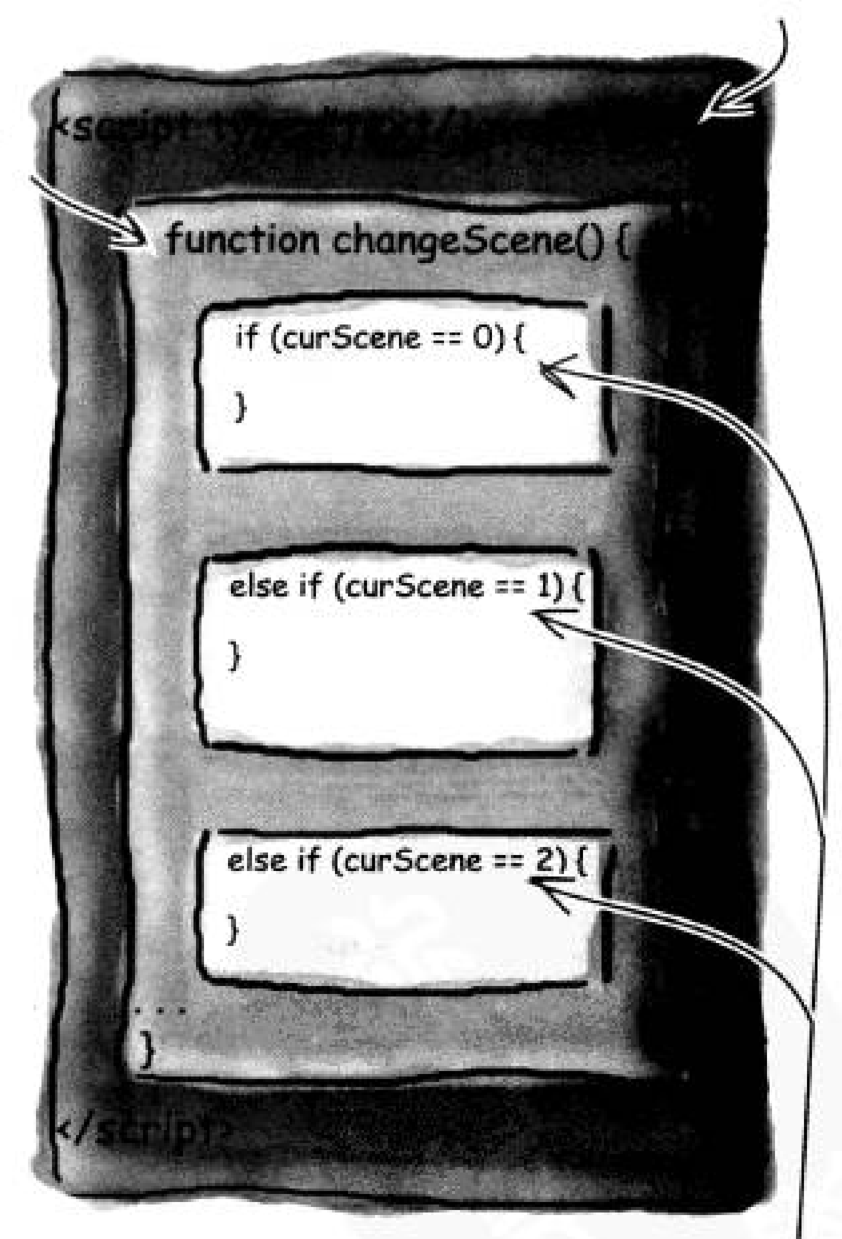
```
<script type="text/javascript">
  // Initialize the current scene to Scene 0 (Intro)
  var curScene = 0;

  function changeScene(decision) {
    // Clear the scene message
    var message = "";

    if (curScene == 0) {
      curScene = 1;
      message = "Your journey begins at a fork in the road.";
    }
    else if (curScene == 1) {
      if (decision == 1) {
        curScene = 2;
        message = "You have arrived at a cute little...";
      }
      else {
        curScene = 3;
        message = "You are standing on the bridge...";
      }
    }
    else if (curScene == 2) {
      ...
  }
</script>
```

问题在于需要在 changeScene() 函数的作用域外保留变量值。函数起始时的 message 变量十分清楚，不需保留在函数外。但是，变量 curScene 却在 if/else 语句的条件句里被测试了很多次，所以必须在函数调用间维持这个变量。**底线：message 可创建为局部变量，但本例的 curScene 需为全局变量。**

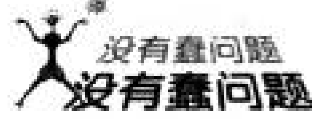
数据生活在哪里

如果作用域的概念还是让你有点困惑，或许，把脚本的各个部分当成数据可在其中生活、自给自足的区域会有点帮助。以《火柴人大冒险》为例，脚本中就有好几个可用于存储数据的作用域。





只有一个用于创建全局变量的全局作用域，其他均为局部作用域。任何创建于全局层次的事物，都能被整份脚本看见，但局部数据只能在它的限定作用域里被看见与使用。



问：我有些数据。我该用哪种变量存储数据呢？全局或局部？

答：使用数据的方式取决于数据需为全局的或局部的。但你既然问了，一般规则是尽量都创建为局部变量，只在局部变量不敷使用时，才采用全域变量。

今晚主题：局部变量与全局变量针对帮数据找家一事，讨论地点的重要性

局部变量：我发现只专心看待自己周围的环境还蛮有用的。事实上，我根本没办法说出我的邻居家发生了什么事，不过我喜欢这种感觉。

全局变量：老兄，你真应该扩展一下世界观。出门来趟小旅行，看一下脚本世界的其他地方。

局部变量：听起来很诱人，但我喜欢舒适空间的安全性。当我悠闲地待在家里，我知道外界没有人能够打扰我。

全局变量：或许如此。但你知道吗？从脚本模式的全局看来，你的小小生活实在无足轻重。每次需要你的小世界，你就随之创建，但不需要时，你也跟着被摧毁，而我仍在此屹立不摇。只要脚本存在，我就存在。

局部变量：噢！轮回转世啊！我不太确定是否该相信，但在存储数据这方面，我完全和你一样方便。我只是没有调用全世界前来观赏的暴露欲。

全局变量：好吧。我承认有时候我是被误用或滥用了那么几次，但是我有不畏艰难险阻，始终紧握着值的优点，这样应该可以平衡缺点了。当脚本需要一块总是记得自己的值而且随时可取用的数据时，它们就需要找我了。

局部变量：至于需要对某部分代码维持某些信息的隐秘性时，脚本就会来找我，因为我很懂得保密防谍。

全局变量：听来不错，不过我每天都把可得性和持久性看得比隐私更重要。

局部变量：所以大家都觉得我们很好用。

没有蠢问题

问：如果在注释里放了真正的JavaScript代码，会发生什么事？

答：什么都不会发生！JavaScript解释器完全忽略注释，任何放入注释的内容，在解释器运行脚本时都会被忽略。知道了这一点，注释即可于追踪问题或尝试不同的编码策略时，暂时停止某段程序代码的活动。

问：一行JavaScript代码结束后，可以在它后面加上单行注释吗？

答：可以。此时，代码一样会运行，因为它不是注释里的一部分。单行注释不需要占据一整行——注释只是从//到该行结尾。如果//出现在代码后，代码仍会运行。

问：为什么注释不需以分号表示结束？

答：因为注释不是一种JavaScript语句。注释是种标示，用途仅限于叙事，或提供关于代码的附加信息，有点像书中的注脚。重点就是记得JavaScript解释器忽略所有注释——注释乃为人类而存在，并非为了JavaScript而存在。

问：在创建全局变量时，"脚本层次"又是什么意思？

答："脚本层次"（script level）是脚本的最上层，就在<script>标签里。"脚本层次"的特殊，在于它位于任何函数或任何代码块之外，因此，任何建立在"脚本层次"的事物均被视为全局的。也就是说，任何创建在"脚本层次"的事物，生命周期与脚本一样，而且能被任何网页内的代码访问。

问：如果我在复合语句里创建变量，它是局部变量吗？

答：或许是。目前的JavaScript版本（1.7）对变量并不支持真正的局部作用域，而是支持函数作用域（function scope），也就是说，某个函数内的变量被视为该函数内的局部变量。但是，只把变量塞到复合语句里，它不会自动属于局部作用域。未来的JavaScript或将调整这种状况。这样记忆比较简单：创建在函数内的变量为局部的，其他均为全局的。

问：作用域、流程（flow）、执行……这些和局部或全局变量有关的东西，听起来真是够复杂。真的很难吗？

答：不会。重点只要记得：局部变量即足以存储暂时信息（即不需用于函数或其他代码块的信息）。如果你需要数据在脚本的生命中一直存在，才设定为全局变量。有一点很让人吃惊，大部分脚本比我们的想象更为暂时，所以你使用局部变量的时机可能比全局变量多很多。

局部变量存储暂时性信息，全局变量存储的时间则与脚本的生命相同。

复习要点

- **注释**（comment）适合提醒还有未写完的代码。
- 别怕在代码中加上很多注释说明，如此才易懂。
- 使用一对斜线（//）起始**单行注释**。
- **多行注释**以 /* 起始、以*/ 结尾。
- 全局变量创建于脚本层次，超越任何函数或其他代码块，而且在**脚本的终生**都会存在。
- 局部变量创建（和摧毁）于代码块内，只能在该代码内被访问。
- 我们**比较**建议使用局部变量，因为它们的访问权受到较严格的控制。

5个选择

还记得我们的游戏参赛者 Eric 吗？他急忙吃完甜甜圈，参加下一轮《全民猜猜乐》。问题是，他现在面对非常有挑战性的抉择……他必须从5个箱子里选一个。

A B C D E

天哪，真希望有人能帮我作决定。

动动脑

你会如何设计一个牵涉到5个不同选项的JavaScript代码？

难道不能使用许多 if/else 语句，从5个选项里挑选吗？

5个选项

好主意！虽然if/else语句原意在于两物的选择，但重叠好几个 if/else 的嵌套语句，就能在任意多个选项中选择了。

```
if (chosenCase == "A")
  openCase("A");
else if (chosenCase == "B")
  openCase("B");
else if (chosenCase == "C")
  openCase("C");
else if (chosenCase == "D")
  openCase("D");
else if (chosenCase == "E")
  openCase("E");
```

这份代码可行，但选择最后一个箱子时，需要等到每个条件都被评估，实在不太有效率。

嵌套的 if/else 可能变得太复杂

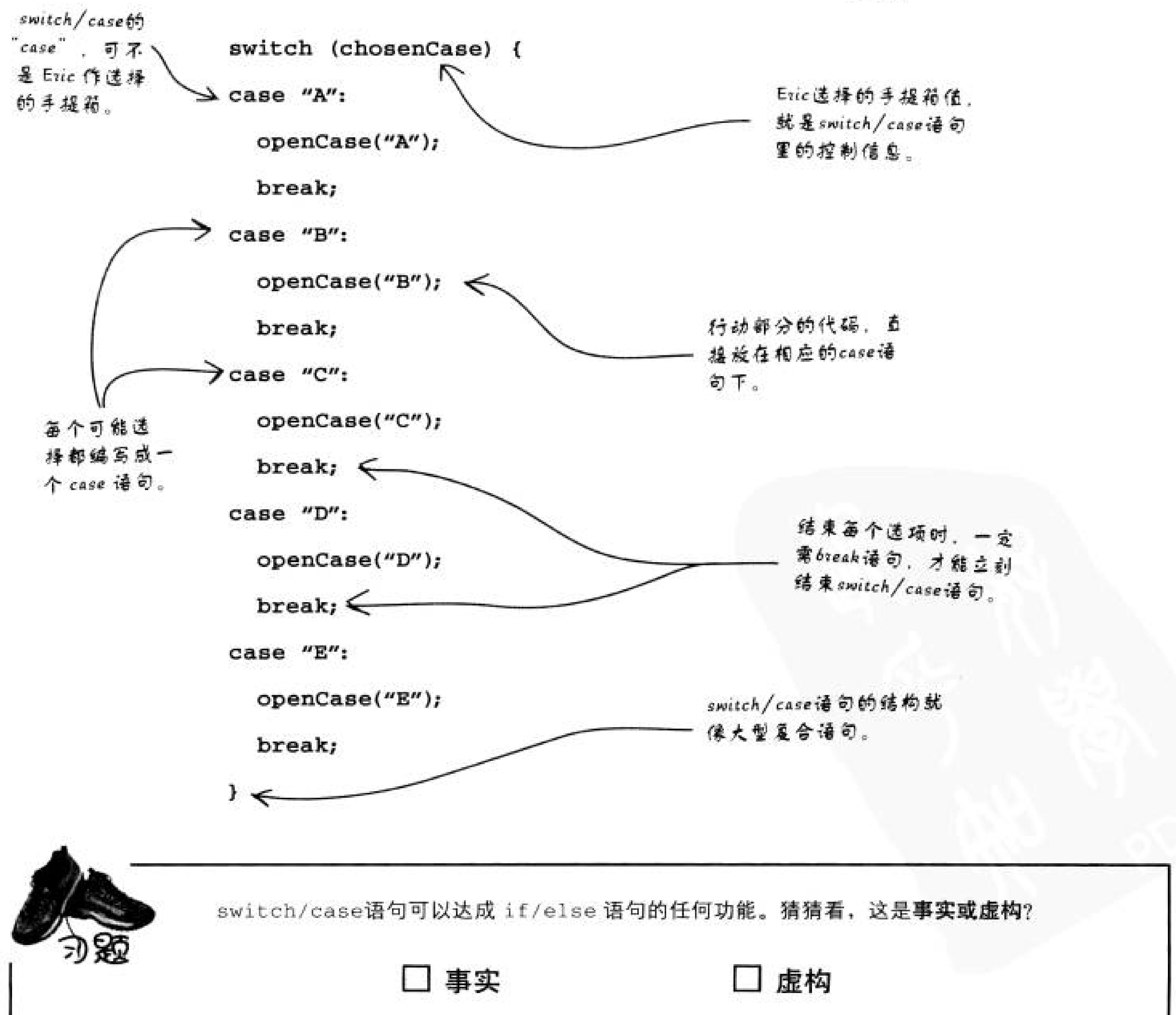
嵌套的if/else语句的确可行……但效率有点差，因为if/else语句并非为了超过两个选择的状况而设计。原因嘛……你想想要经过多少次boolean测试，才能选择 E 箱就知道了。5项条件都需评估，实在不太有效率。

如果可以从两个以上的项目里选择，而又不需使用无效率的if/else语句，该有多好啊？真希望这不是我的黄粱大梦…………

switch 语句能用很多“箱子”

JavaScript 有个多重选项专用的决策语句。前面提到的 if/else适合在**两样事物**间二选一，而 switch/case 语句则让我们能在**许多事物**间有效率地多选一。让我们一同从 switch/case 的角度观看 Eric 遇到的困境。

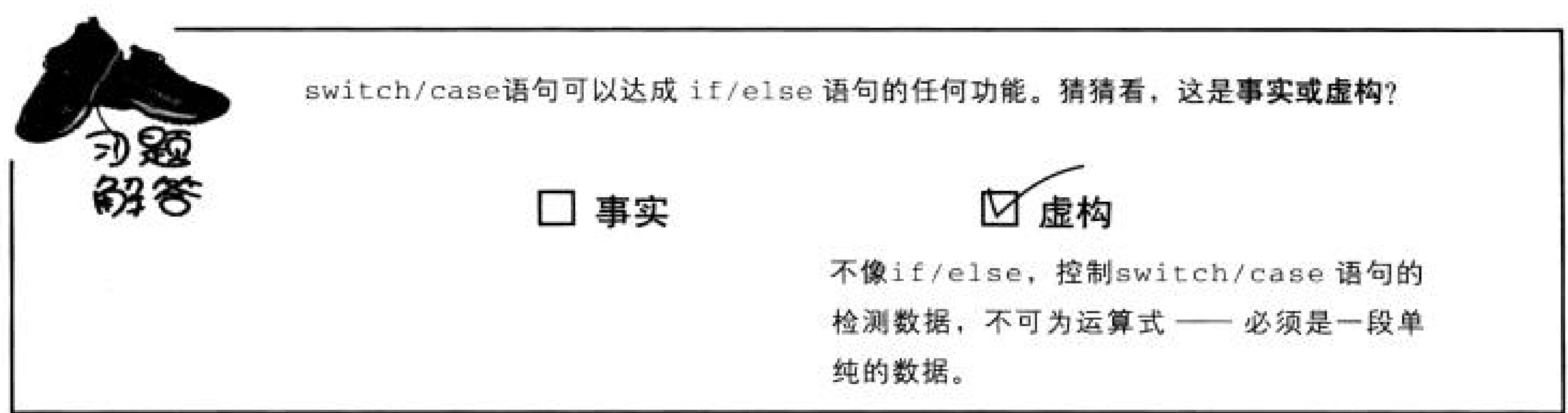
switch/case 语句有效率地在两个以上选项的状况中作决定。



习题

switch/case语句可以达成 if/else 语句的任何功能。猜猜看，这是事实或虚构?

☐ 事实　　☐ 虚构



习题解答

switch/case语句可以达成 if/else 语句的任何功能。猜猜看，这是**事实或虚构**？

☐ 事实

☑ 虚构

不像if/else，控制switch/case 语句的检测数据，不可为运算式 —— 必须是一段单纯的数据。

在 switch 语句中

见识过switch/case语句的行动后，接下来分析这个语句的一般格式。

```
switch + ( + Test data + ) + {
case + Match 1 + :
    Statement + ;
    ...
    break + ;
...
default + :
    Statement + ;
    ...
    break + ;
}
```

测试数据必须为一块数据，不能是一段表达式 —— 此处并非用于评估true/false。

每个决策分支均以关键字case起始，后接匹配案例（match）。

每个匹配案例的结尾为冒号，并非分号。

break 语句避免运行到其他决策分支的程序代码。

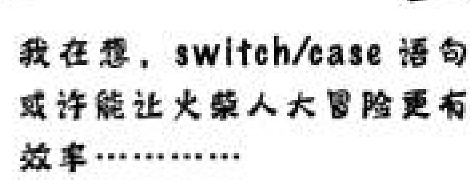
选用的“default”分支里包含的程序代码，于没有匹配案例时运行。

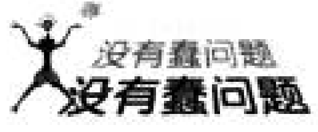
整个语句主体需以一对大括号围起。

switch/case 语句：换你写写看

创建switch/case语句的确比创建if/else语句需要更多工作，但在处理两个以上的结果时，前者比较有效率。以下为过程：

1. 以括号围起测试数据（test data），并开始一段复合语句（{）。
2. 编写相应的 case，后随冒号（:）。
3. 设计匹配案例（match）时运行的语句（statement）。这部分可以是很多行代码 —— 不见得需为复合语句。
4. 加入break语句 —— 别忘了分号（;）。
5. 可选择引入默认分支（default），用于没有匹配案例的状况。
6. 结束复合语句（}）。





问：**所以说，switch/case 语句并非利用 true/false 运算作决定啰？**

答：没错。并不像 if 或 if/else 语句，switch/case 语句只使用一块测试数据作决定，因此才能支持多个结果的状况。

有 break 比较安全。

注意！ 为免代码意外运行，请在每个相应的switch/case后加上break语句。

问：**每个 case 的匹配案例都只会对应至某个测试数据而已吗？**

答：是的。此时的概念是以变量作为测试数据，然后使用字面值与匹配案例相比较。

问：**如果不在switch/case语句中加上break语句，会发生什么事？**

答：可能发生意外结果哦！break语句具有区隔switch/case语句中每段行动代码的功用。少了break语句，所有行动代码将如一大块代码般运行，如此便失去了不同决策的用意。在switch/case语句中找到匹配案例时，即运行相应case下的代码，直到遇见break语句为止。此时，才会离开 switch/case 语句。

本周主题：
行动者、决定者

HeadFirst：欢迎你今天来上我们的节目！如果要用一个词来形容你自己，你会选什么词呢？

Switch：挑剔（choosy）。

HeadFirst：呃……能不能解释得更详细一点？

Switch：我让“在许多项目间作选择”一事成真。虽然有些情况只需要单纯的黑白二分法决定，但有更多状况需要……嗯……差别较为细致的选项。此时就轮到我上场了。

HeadFirst：但大家都说 If 能做到相同的工作，需要的代码甚至更少一点。

Switch：或许是这样没错，只要敲得够久，用槌子也能截断木头啊。就个人而言，我觉得用锯子比较好。其实，每个人都有自己的专长，我的专长就是有效率地在许多不同事物间作选择。我并不讨厌 If，但他比较适合不同工作。

HeadFirst：你提到了效率。告诉我们，效率在你的工作里扮演什么角色？

Switch：嗯，我的构造在于根据数据值作决策，而我的工作只是拿数据与可能的选择相比对，决定该运行哪一段代码。只是这样而已。我不会烦恼评估表达式的问题，我也不需为了从许多可能结果中作决策，而利用嵌套的语句或其他花哨手段。如果打算根据某个数据快速决策，我就是你最好的朋友！

HeadFirst：我们来聊一下你的好朋友—— Break。没有他的话，听说你的日常工作将无法结束？

Switch：确实如此。没有 Break，我的麻烦就大了；没有他，我将无法分隔不同段的行动代码。Break 让我知道哪段代码已经结束运行，我可以结束工作，不会意外地运行其他代码。

HeadFirst：原来如此。Case 呢？他也是跟你关系很亲近的好朋友，对不对？

Switch：当然了。Case 和我的关系很密切，主要在于 Case 告诉我所有可能的选择。没有 Case，我就失去了作决策的基础。

HeadFirst：这么说来，Case 呈现可能的匹配案例，你则利用匹配的案例，决定该做什么事。不过，测试数据如果没找到匹配案例该怎么办？

Switch：看状况。如果没有特别处理“无符合案例”的代码，什么都不会发生。不过，我还有个好朋友 Default，可以在“无匹配案例”时运行特定代码。

HeadFirst：哇，我倒是不知道有这么回事。Default 与 Case 的相处情形如何？

Switch：还不错，事实上。他们不会抢着曝光率，也不会侵犯对方的地盘。Case 处理所有匹配案例的事项，Default 则处理没有匹配案例时的状况。偷偷跟你讲，我想Case其实很高兴有 Default 的帮忙，因为它很不会处理没有匹配案例的状况。

HeadFirst：哦，我了解了。看来我们的时间也快到了，你还想补充什么吗？

Switch：嗯……迟疑不决是最糟糕的事。没人喜欢含糊其辞的家伙。外面有很多种可能性，不表示你必须举手投降。调用我，我会尽一切可能帮你作出对脚本最有利的决策。

请转换《火柴人大冒险》前两幕场景的代码，让它们使用switch/case语句取代 if/else 语句。

```
...
if (curScene == 0) {
  curScene = 1;
  message = "Your journey begins at a fork in the road.";
}
else if (curScene == 1) {
  if (decision == 1) {
    curScene = 2
    message = "You have arrived at a cute little house in the woods.";
  }
  else {
    curScene = 3;
    message = "You are standing on the bridge overlooking a peaceful
stream.";
  }
}
...
```

这里是使用 if/else 的原始版。

磨笔上阵解答

请转换《火柴人大冒险》前两幕场景的代码，让它们使用**switch/case**语句取代 **if/else** 语句。

```
...
if (curScene == 0) {
  curScene = 1;
  message = "Your journey begins at a fork in the road.";
}
else if (curScene == 1) {
  if (decision == 1) {
    curScene = 2
    message = "You have arrived at a cute little house in the woods.";
  }
  else {
    curScene = 3;
    message = "You are standing on the bridge overlooking a peaceful
stream.";
  }
}
...
```

这里是使用if/else的原始版。

设定新的场景编号与说明文字，与if/else版一样。

每个case的选择均与场景编号相应。

在每个case里，利用if/else处理用户的故事情节选择仍然合理。

接下来的场景也采用相似结构。

以}结束switch/case语句。

```
switch (curScene) {
case 0:
  curScene = 1;
  message = "Your journey begins at a fork in the road." ;
  break;
case 1:
  if (decision == 1) {
    curScene = 2
    message = "You have arrived at a cute little house in the woods." ;
  }
  else {
    curScene = 3;
    message = "You are standing on the bridge overlooking a peaceful stream." ;
  }
  break;
...
}
```

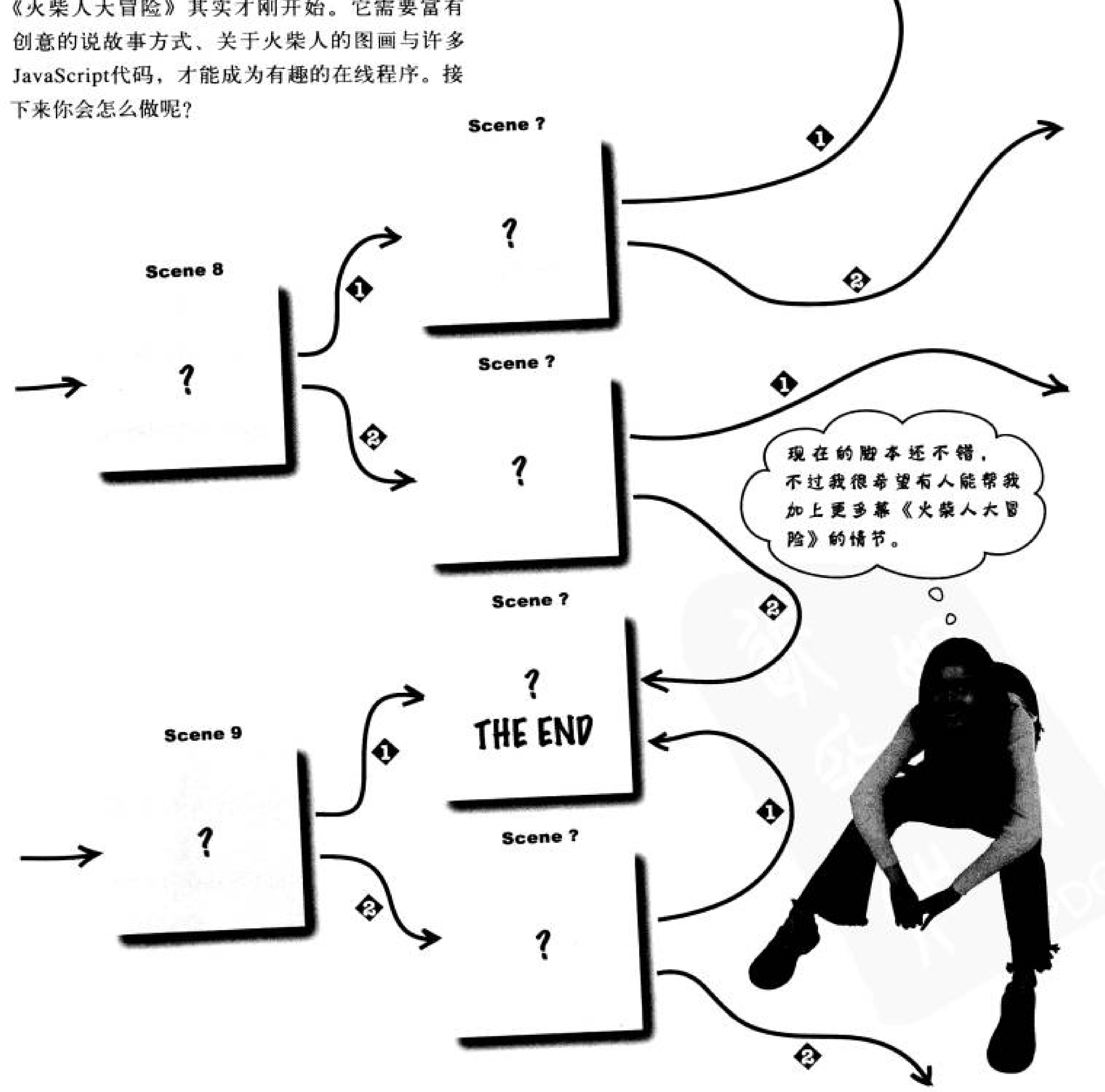
switch 版火柴人大冒险：试用

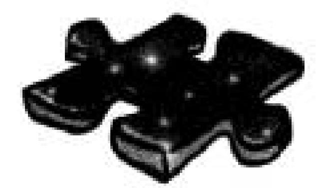
完整地重新编写过《火柴人大冒险》的决策逻辑后，Ellie 真的很想看看结果。如果你走一遍故事流程，两者的差别很快就会看出来……唔……

等一下，一切看来都一样！因为switch/case语句对《火柴人大冒险》的改变，只影响了代码的结果，而不是呈现的外观。这是个代码的改变单纯发生在幕后的范例……彻头彻尾地只在“幕后”！

故事继续……

《火柴人大冒险》其实才刚开始。它需要富有创意的说故事方式、关于火柴人的图画与许多JavaScript代码，才能成为有趣的在线程序。接下来你会怎么做呢？





JavaScript 填字游戏

来点简单的决策吧！现在休息一下，玩一盘填字游戏如何？来吧！

横向提示

1. 先写下______、能让设计复杂的JavaScript代码更容易。
4. 一种能用于说明代码的东西。
5. 这类语句实际由许多语句组成。
6. 当某个 if 语句置于另一个 if 语句中时，称为________。
8. != 运算符用于测试________。
10. 利用________，有助于把复杂决策视觉化。
13. 整份脚本都可访问这类变量。

纵向提示

2. 做这件事，________，做另一件事。
3. 这类运算符的结果为 true/false。
4. 当代码是部分 if/else 语句时的运作方式。
7. 可根据一段数据的值作决策的语句。
9. 作用域受限的变量。
11. 这种语句让我们有条件地运行一段代码。
12. 在 switch 语句下的每个决策分支都有一个_______。

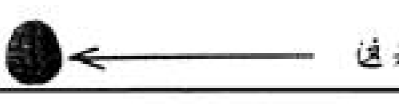
JavaScript 填字游戏解答

					[1]P	S	E	U	D	O	C	O	D	[2]E									
														L			[3]B						
							[4]C	O	M	M	E	N	T	S		[5]C	O	M	P	O	U	N	D
							O							E			O						
							N										L						
							D						[6]N	E	[7]S	T	E	D					
							I								W		A						
							T								I		N						
							[8]I	N	E	Q	U	A	[9]L	I	T	Y							
							O						O		C								
[10]D	E	C	[11]I	S	I	O	N	T	R	E	E		C		H								
			F				A						A			[12]C							
							L					[13]G	L	O	B	A	L						
							L									S							
							Y									E							

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

当 if/else 还不够的时候……



Scene 0
Title intro

Scene ?
Swamp

global

local

Scene ?
Ditch

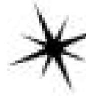
Scene ?
Troll on bridge

Scene 1
Fork in the road

Scene 3
Bridge overlooking stream

虽然 if/else 语句非常方便，
但还是有其限制，
例如，不能在两个选项中切换。
如果你不相信，
请自己试试看。

5 循环

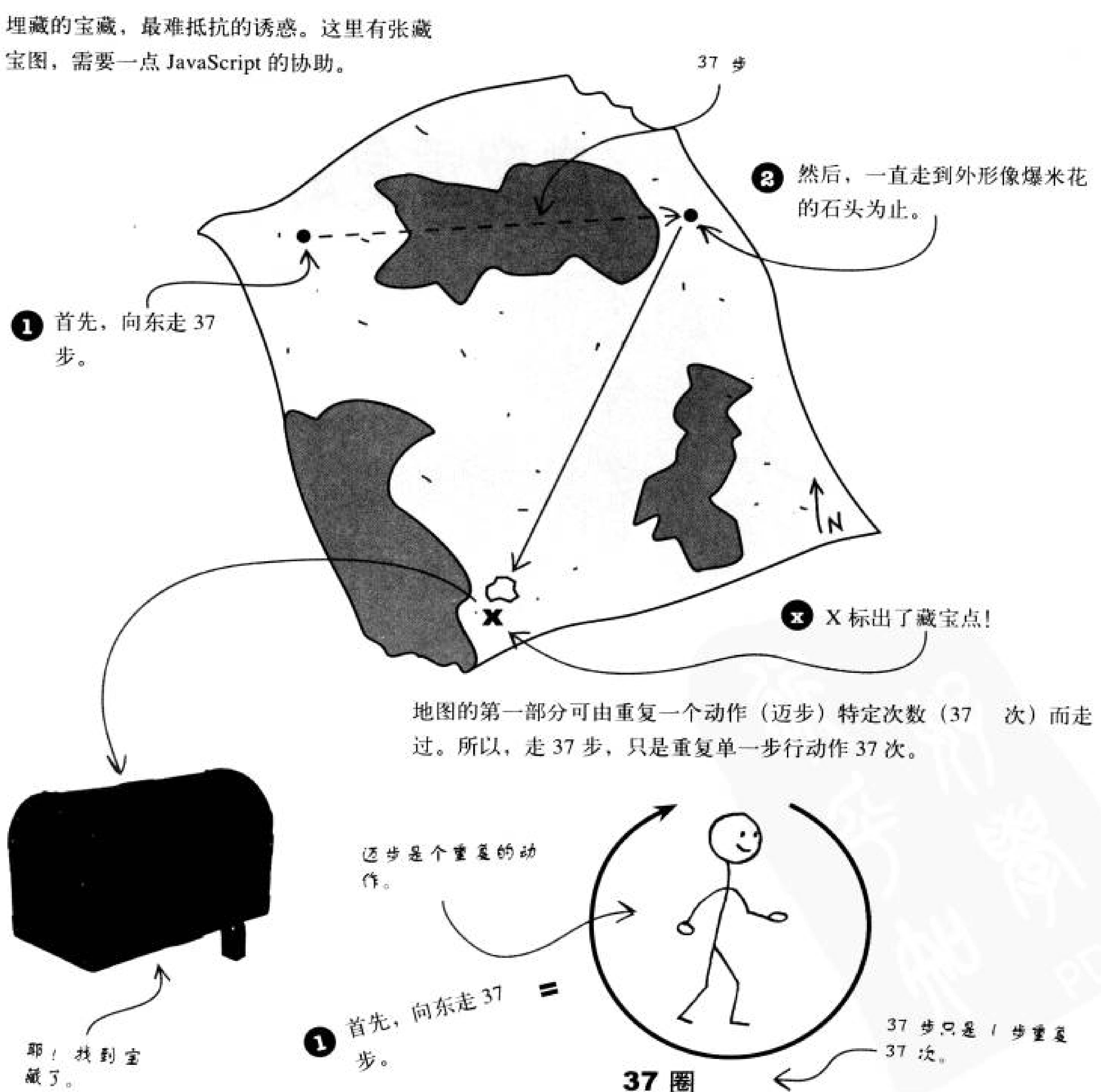


自我重复的风险

有些人说，一点重复是生活中的香料。没错，尝试新鲜有趣的事物确实很刺激，但一点点重复真的有可能完成一天的工作。强迫性地消毒双手、让人神经紧张的嘀嗒声、每次收到邮件就按下“全部回复”……好吧，或许现实生活里的重复不见得都是好事。但在JavaScript的世界里，“一点重复”可能非常便利。我们需要脚本**重复运行代码**的程度很让人惊讶，循环功能因而闪闪发光。没有**循环**，你将浪费许多时间用于剪贴代码。

X 标示藏宝地点

埋藏的宝藏，最难抵抗的诱惑。这里有张藏宝图，需要一点 JavaScript 的协助。



地图的第一部分可由重复一个动作（迈步）特定次数（37 次）而走过。所以，走 37 步，只是重复单一步行动作 37 次。

所以问题是……JavaScript 如何重复动作？

似曾相识……循环

JavaScript 的重复机制为**循环**（loop），这个机制让我们重复代码。尤其以 for 循环特别适合重复动作**已知**次数。举例而言，for 循环适合计数的任务，如倒数至零或倒数至某个数。

for 循环由4个部分组成：

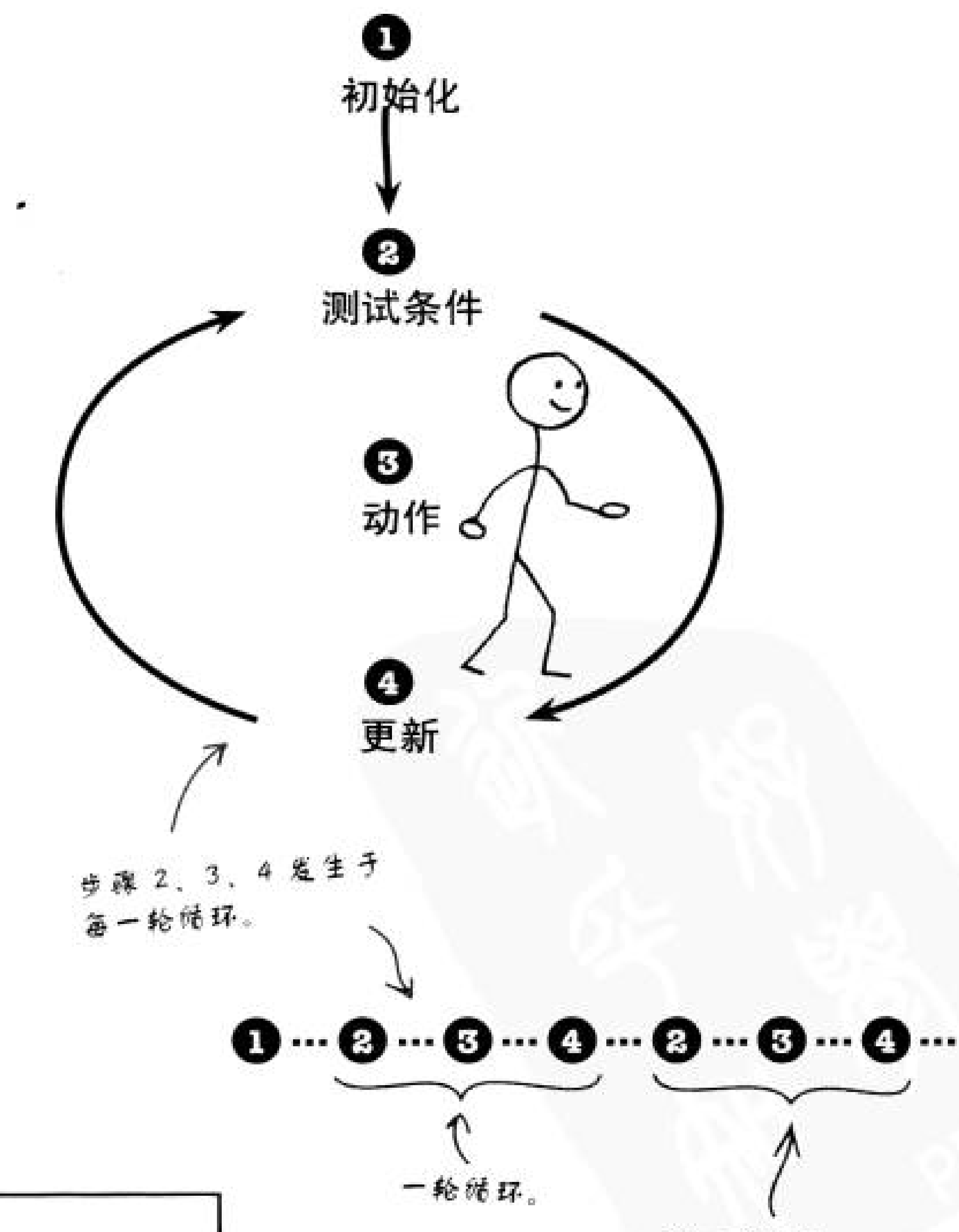
循环能重复运行程序代码特定次数。

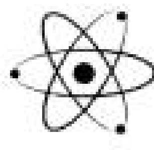
❶ **初始化**（initialization）
初始化只于 for 循环起始时发生一次。

❷ **测试条件**（test condition）
测试条件检查循环是否要再继续跑一轮。

❸ **动作**（action）
循环里的动作就是每一轮循环实际重复的代码。

❹ **更新**（update）
循环里的更新负责于每一轮结尾更新任何循环变量。



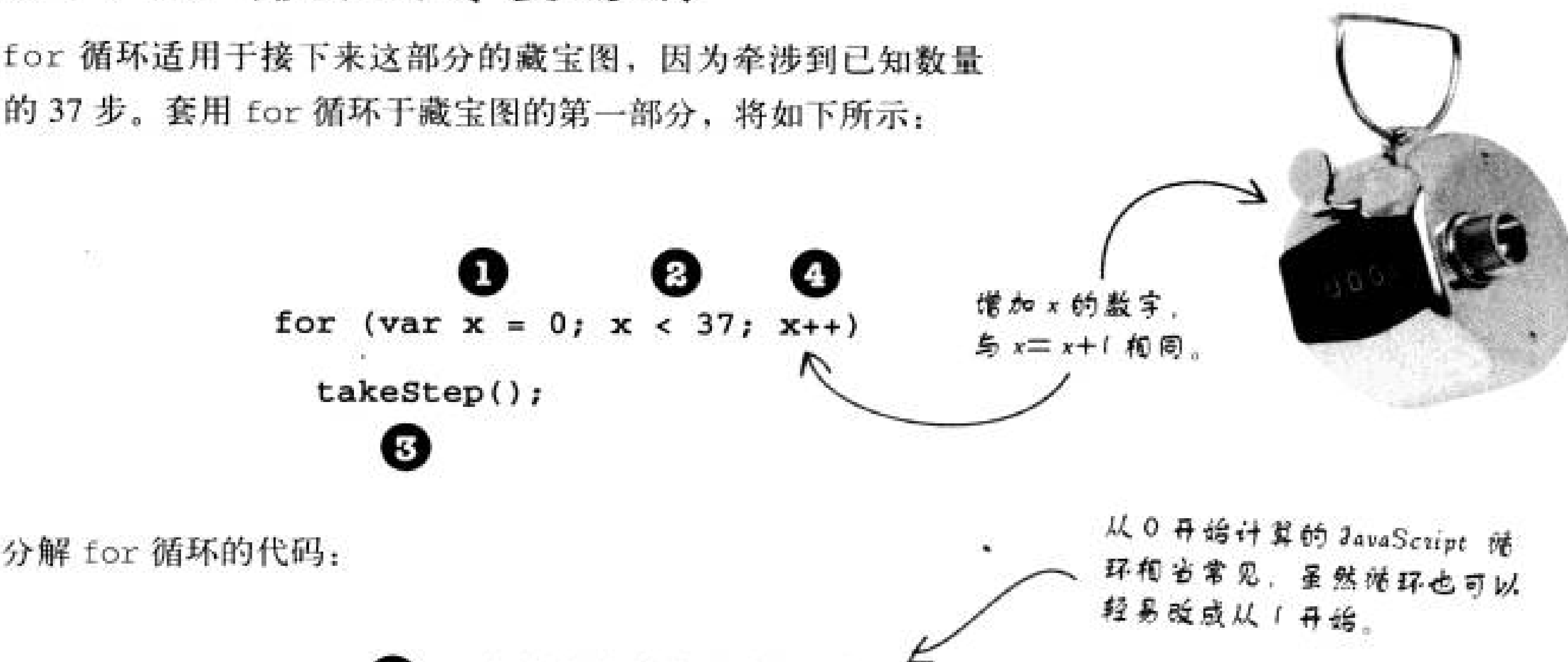


动动脑

for 循环的4个部分，如何与藏宝图产生联系？

引入 for 循环的寻宝游戏

for 循环适用于接下来这部分的藏宝图，因为牵涉到已知数量的 37 步。套用 for 循环于藏宝图的第一部分，将如下所示：



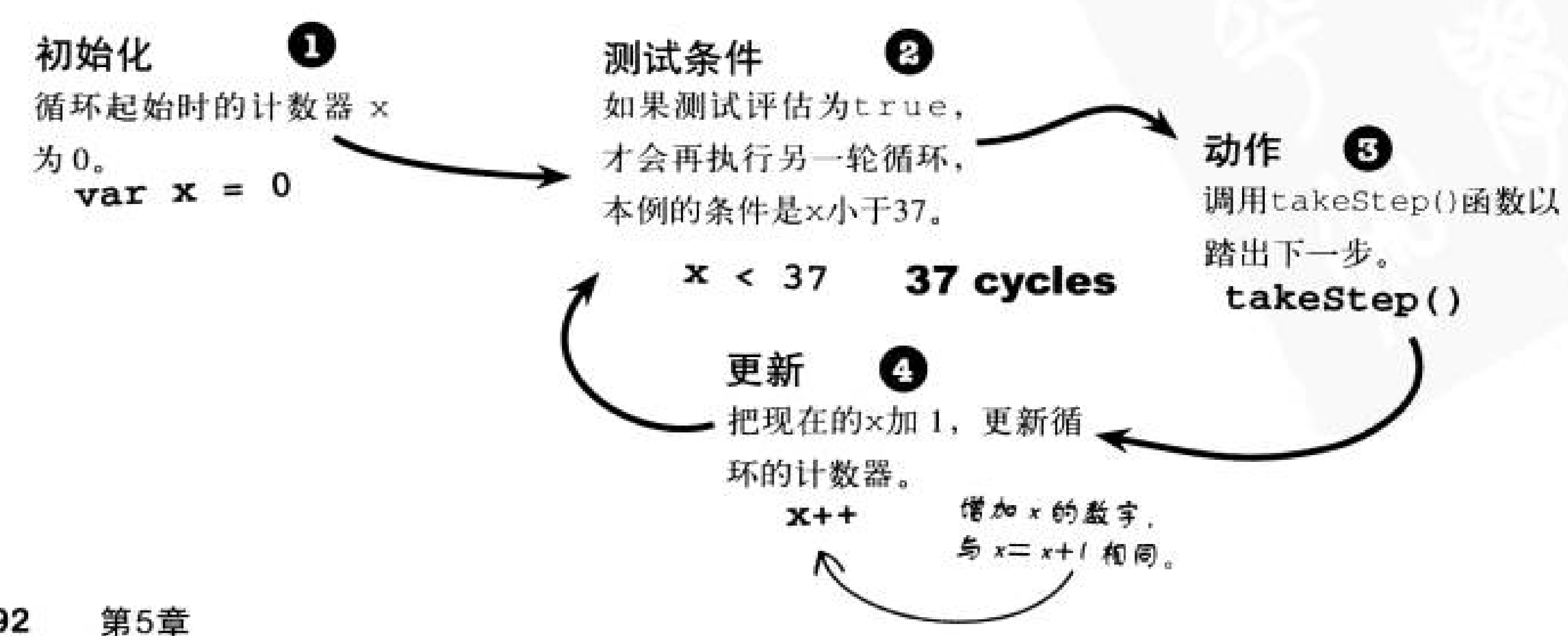
分解 for 循环的代码：

从 0 开始计算的 JavaScript 循环相当常见，虽然循环也可以轻易改成从 1 开始。

1. 初始设定计数器变量x为 0。
2. 确认 x 是否小于 37。
3. 运行循环的动作部分代码，本例为运行 takeStep() 函数。
4. 增加 x 的计数，回到第 2 步，准备开始下一轮可能循环。

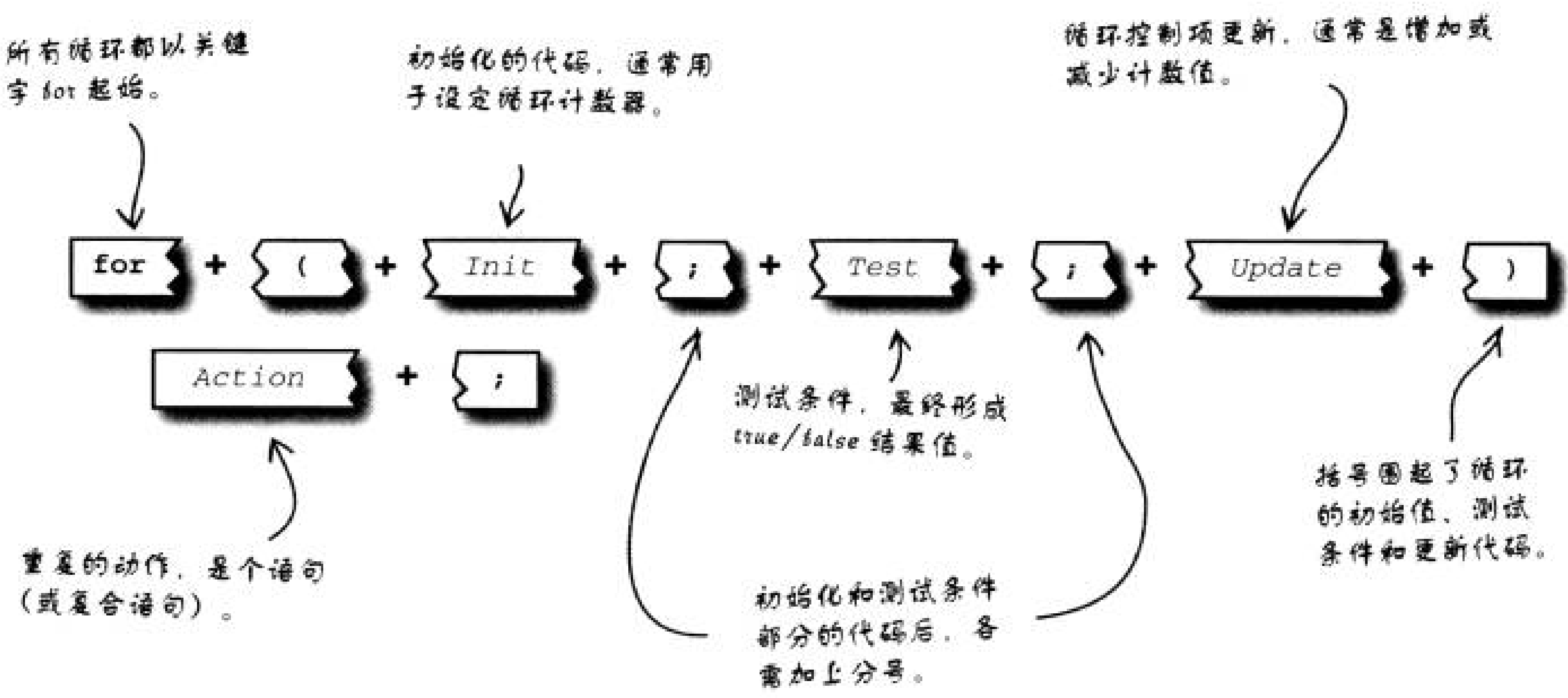
37 圈

跑过 37 轮循环后，结束在 x 等于 37。一切都归功于 for 循环4个部分的通力合作，一起建立了 JavaScript 的重复机制。



解剖循环

所有 for 循环都维持一致公式，需要它的4个组成部分待在特定位置。幸好这个公式保留了很多灵活性，能制作我们自己的循环。



习题

完成下列代码，首先提醒用户输入一个大于0的数字，然后使用该数字作为 for 循环的倒数计时，就像老电影片头的字幕一样（4、3、2、1，开始！）。还有，记得验证数字确实大于 0，才开始倒数计时。

数字存储于 count 变量。

提醒用户输入数字。

```
var count = prompt("Enter a number greater than 0:", "10");
```

....

....

....

....

....

....

....

完成下列代码，首先提醒用户输入一个大于0的数字，然后使用该数字作为for循环的倒数计时，就像老电影片头的字幕一样（4、3、2、1，开始！）。还有，记得验证数字确实大于0，才开始倒数计时。

```
var count = prompt("Enter a number greater than 0:", "10");
if (count > 0) {
  for (var x = count; x > 0; x--)
    alert("Starting in..." + x);
  alert("Roll film!");
}
else
  alert("The number wasn't greater than 0. No movie for you!");
```

数字存储于count变量。

提醒用户输入数字。

倒数至1。

确认count大于0。

每跑一轮循环，计数器即递减一次。

以count值初始化循环计数器（x）。

呈现目前的count值。

不合格的数据。

倒数结束！

The number wasn't greater than 0. No movie for you!

OK

Roll film!

OK

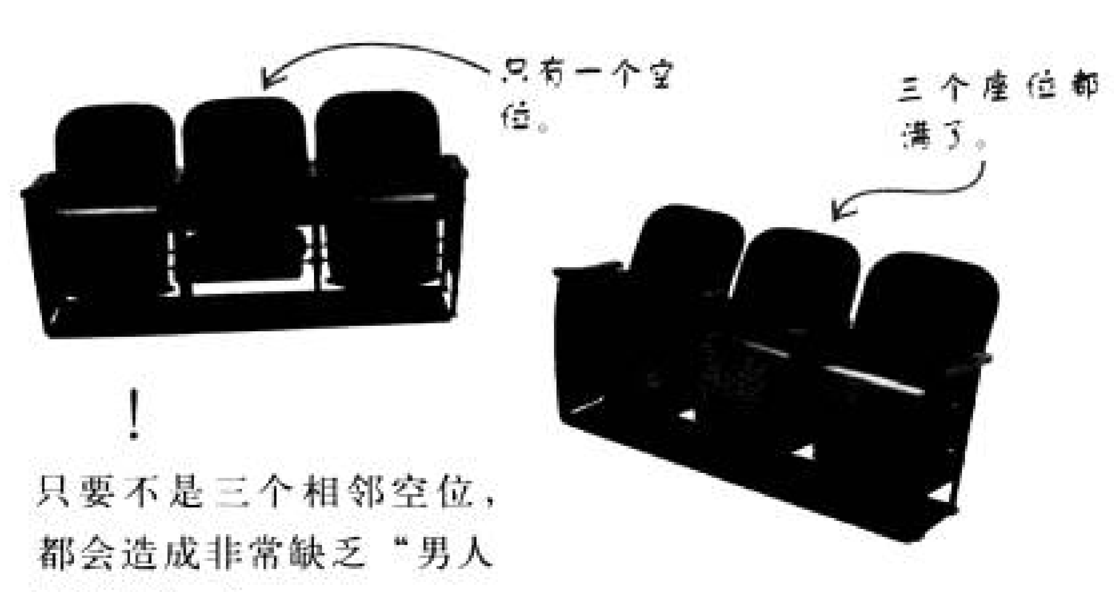
Mandango：壮汉专用电影院划位工具

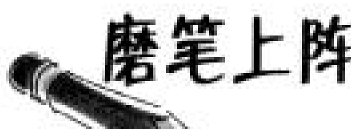
JavaScript 循环不只可用于电影片头的倒数计时。你可能也知道，大多数体格魁梧的彪型大汉一起去看电影时，都希望能与其他人相隔一个空位。因此，Seth 和 Jason 想制作 Mandango —— 专为壮汉设计的电影院划位工具。

他们想到可以一次划三个位置，买票的壮汉朋友间就能保有一个空位。问题是，Seth 和 Jason“还没”想出该怎么做……

先确认剩下的座位

他们面前的挑战是：逐一搜索每排座位，找出三个相邻的空位。





只要不是三个相邻空位，都会造成非常缺乏“男人的空间”。

三个都是空位。

酷！

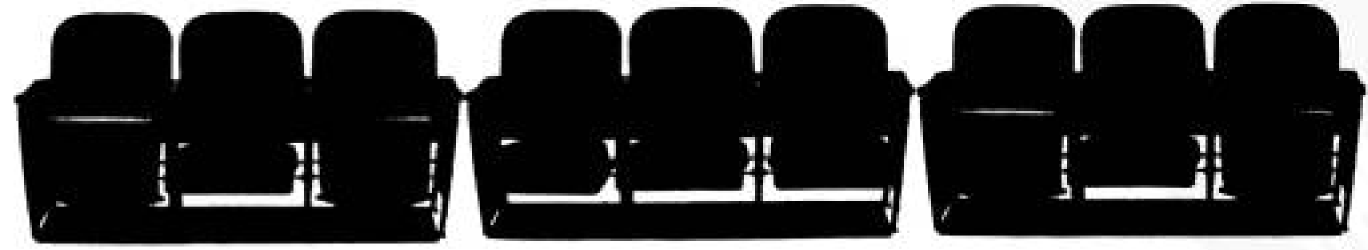
三个相邻的空位表示很充裕的电影欣赏空间。

磨笔上阵

使用下例的空位示意图，设计使用 for 循环寻找三个相邻空位的代码。请确定规划出循环处理每个空位的方式。

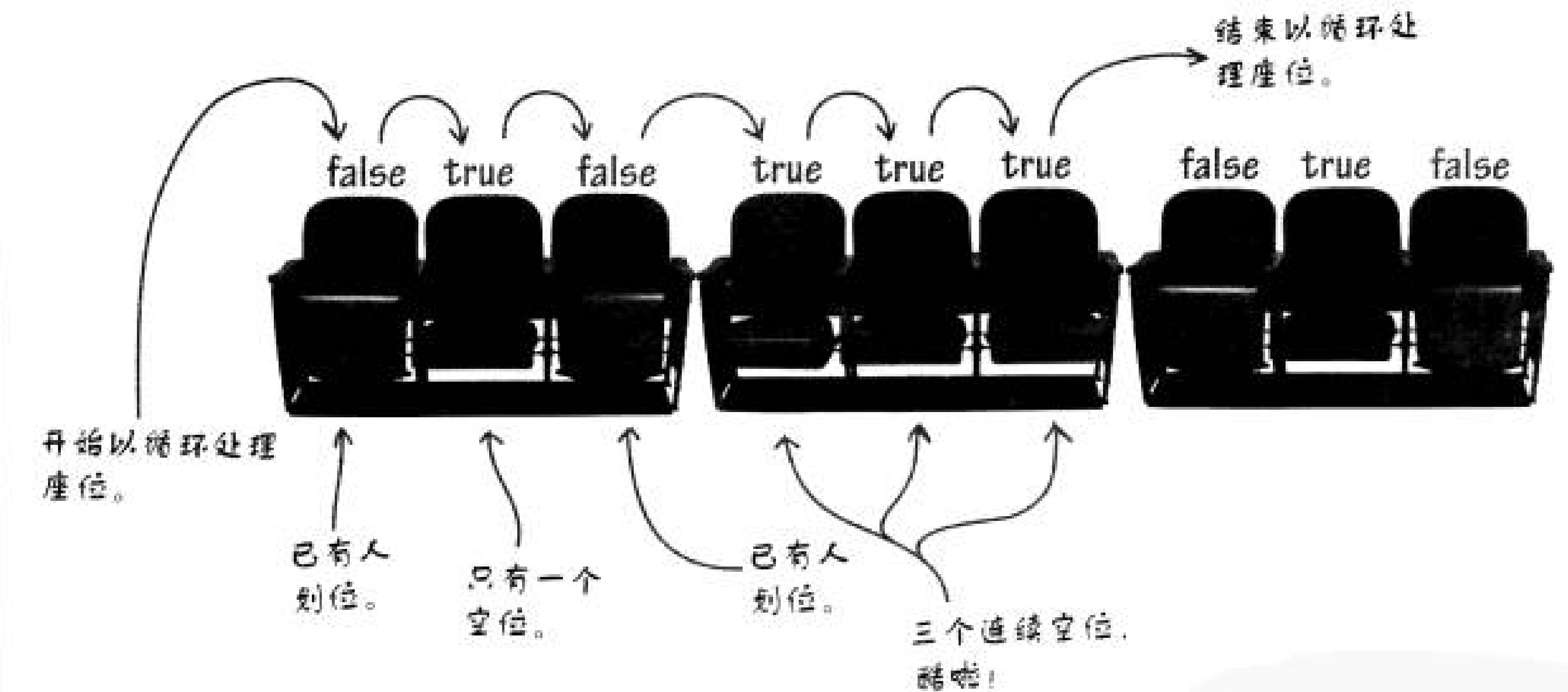
....

....



使用下例的空位示意图，设计使用 for 循环寻找三个相邻空位的代码。请确定规划出循环处理每个空位的方式。

如果空位以 boolean 类型的变量呈现，你可用循环逐一处理座位，找出同一排是否有三个相邻的空位（true）。



循环、HTML 与空位

Mandango 的设计理念是有点道理，但对于每个座位的状态如何转换成 HTML代码，就讲得不太清楚了。

每个座位都以图像形式呈现于 Mandango 网页上。

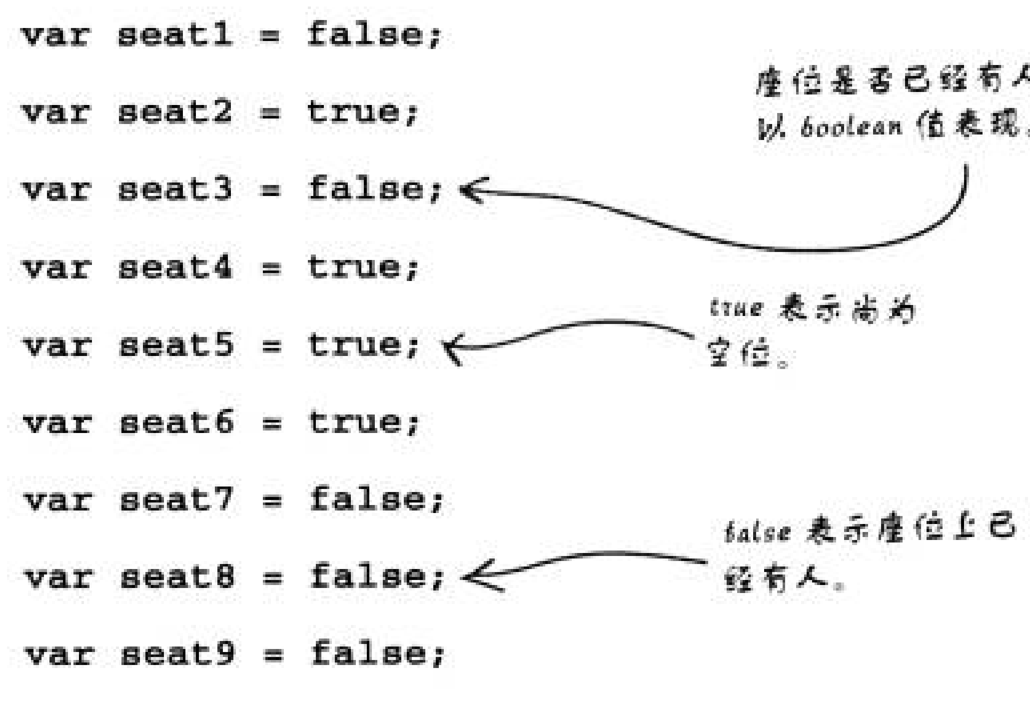
本例的完整 HTML 与图像可于 http://www.headfirstlabs.com/books/hfjs/ 取得。

```
<img id="seat1" src="seat_unavail.png" alt="Unavailable" />
<img id="seat2" src="seat_avail.png" alt="Available" />
<img id="seat3" src="seat_unavail.png" alt="Unavailable" />
<img id="seat4" src="seat_avail.png" alt="Available" />
<img id="seat5" src="seat_avail.png" alt="Available" />
<img id="seat6" src="seat_avail.png" alt="Available" />
<img id="seat7" src="seat_unavail.png" alt="Unavailable" />
<img id="seat8" src="seat_avail.png" alt="Available" />
<img id="seat9" src="seat_unavail.png" alt="Unavailable" />
```

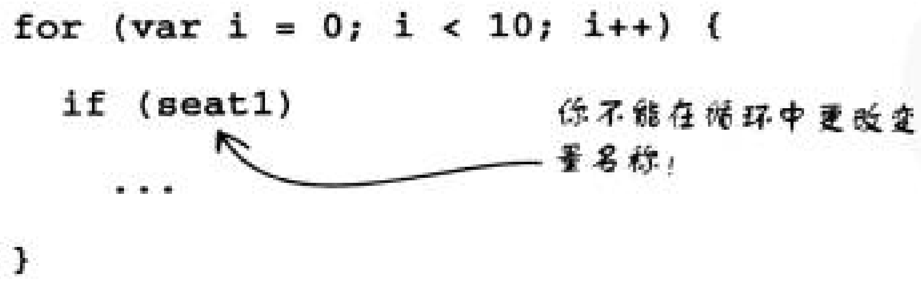
你不只需要以 JavaScript 代码的循环处理 HTML 的图像元素，也需要以 boolean 变量存储图像代表的座位是否为空位。

电影院座位变量

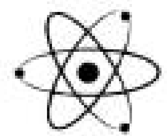
开始思考以循环处理座位、寻找空位的方式前，需以JavaScript代码呈现各个座位是否已被占据。一排9个座位的表现方式，可以是9个boolean变量。



现在你已准备好创建处理9个座位的 for 循环，用于检查是否能取得相邻的三个空位。



等一下，有个问题。for 循环的每一轮都需要检查不同座位变量的值。但是每个变量名称都不一样，没办法达成这项工作。



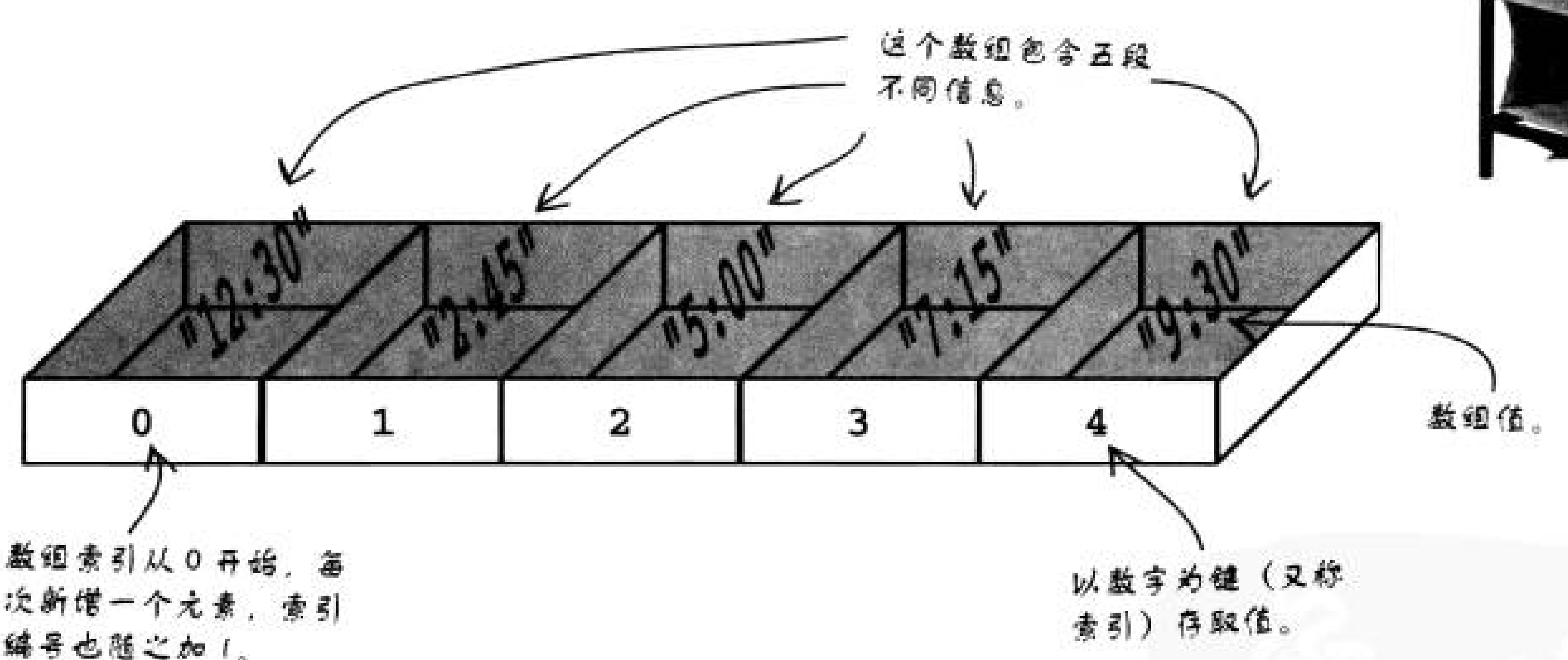
动动脑

如果“每个座位都各为一个变量”的设计不适用于循环，你该如何存储信息，以适应循环的处理方式？

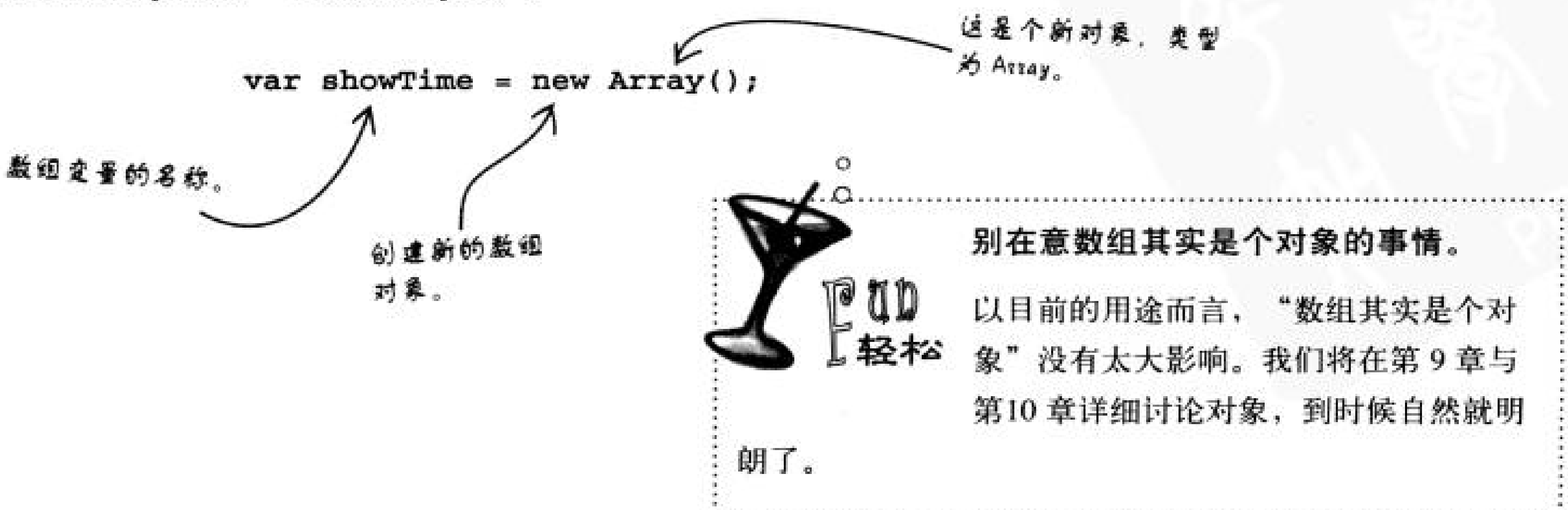
数组收集了多块数据

JavaScript 有一种特殊数据，**数组**（array），可用于存储多块数据于一个变量中。数组只有一个名称，这点与一般变量类似 —— 但数组有多个存储位置。它就像你家的储物柜，明明只是一件家具，但有很多格存储空间。

数组里的每个元素，都由两块信息组成：**值**（value）与具唯一性的**键**（key），需以**键**访问**值**。键通常只是从0开始计算、逐次加1的连续编号。数字编号键又称为**索引**（index），构成**索引数组**（index array）：



创建数组与创建一般变量相似，只不过你必须告诉JavaScript：现在需要数组，而不是仅需一格存储空间。事实上，你正在要求 JavaScript 创建一个**对象**（object）。

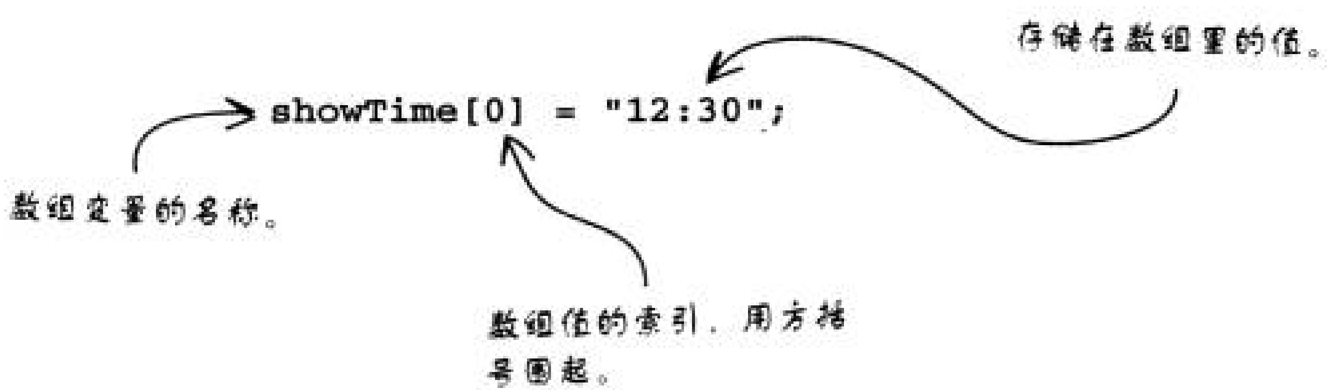


别在意数组其实是个对象的事情。

以目前的用途而言，“数组其实是个对象”没有太大影响。我们将在第 9 章与第10 章详细讨论对象，到时候自然就明朗了。

数组值与键一起存储

不论是否为对象，创建数组后，即可增加数据给数组，并从中访问数据。取得数组存储数据的关键，就是……**键**！独一无二、与特定数据相联的键，用于访问数据。若是索引数组，你只需使用想访问的数组元素的索引。



上段代码设定数组 showTime 的第一个值（本例为时间值）。如果你不希望每次只能手动设定一个数组值，也可以在创建时初始化整个数组：

```
var showTime = [ "12:30", "2:45", "5:00", "7:15", "9:30" ];
```

列出所有数组值，以逗号区隔。

数组创建时的第一部分，与创建一般变量相同。

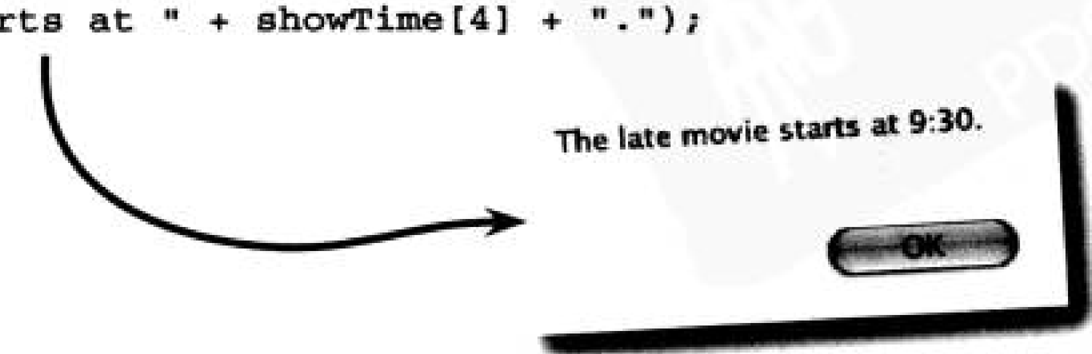
确定以方括号围起阵列值。

别忘了分号。

等一下，这段范例不像牵涉到对象啊？到底是怎么一回事？其实，上例借由直接以值创建数组（对象），而回避了正常创建空对象的方式。你只要列出数组应存储的元素并以方括号围起，即可存入数据。数组填入数据后，就可以使用了。

抓出数组的最后一个值。

```
alert("The late movie starts at " + showTime[4] + ".");
```



数据于单一场所存储多段数据。

数组真情指数

本周主题：
探索连续数据存储者的心灵深处

HeadFirst：Array，很高兴你来上节目！我听说你很擅长存储多段数据。

Array（数组）：确实如此。我就是这么地有容乃大。你需要存储50组字符串或300组数字吗？我就是你的最佳选择。

HeadFirst：听起来真让人跃跃欲试。不过，使用一般变量不就已能存储大量数据了吗？

Array：当然，就像你想赤脚走去上班，那也是你的自由。我跟你说，达成目标的方式一定不只一种。我比一般变量提供储存多段数据的更佳方式。

HeadFirst：好吧，我还是比较喜欢穿鞋出门。但你究竟好在哪里呢？

Array：这么说吧！假设你每天都会乖乖写一篇日记，过了几年后，你要如何保存每一页？

HeadFirst：不是都在日记本里吗？有什么问题吗？

Array：你预设了每一页日记都依某种关系而组织排列。如果我说的日记，只是随意丢在鞋盒里的小张备忘录呢？你的日记不就突然变得很难整理了？

HeadFirst：嗯……把数据存储在数组里，跟日记写在记事本里，又有什么关系？

Array：因为我以此方式组织数据，让它容易被访问。例如我问你去年7月6日的日记内容，你可能叫我翻到124页。数组数据亦同此理，只不过数组的页码称为键（key）。

HeadFirst：我听说过数组索引的大名，但没听过“键”。什么是“键”？

Array：啊，忘记先说明了。“键”是个通用的称呼方式，形容用于寻找另一段数据的信息。“索引”只是一种形式特殊的键，它是数值键。所以说，日记的页码不只是键，也是索引。如果你在讨论利用不重复的数字寻找数据，此时的键和索引其实都一样。

HeadFirst：原来如此。对了，这些东西怎么跟循环扯上关系？现在我只剩下这点不了解而已。

Array：嗯，不见得是任何东西都有关系。就算不扯上循环，我也一样很适用于存储数据。然而，我可以让循环循环处理一堆数据时更为方便。

HeadFirst：怎么办到的？

Array：还记得循环常使用倒数计时做控制吗？以计数器作为数组的索引，你看，这样不就得到循环处理数组数据的方式了？

HeadFirst：慢点、慢点，你说循环的计数器可当成数组的索引，用于寻找数据吗？

Array：正是如此。

HeadFirst：哇，这种能力真是太强大了！

Array：我知道。需要使用循环处理数据的脚本时，才会发现我如此不可或缺。只需几行代码，就能以循环处理整个数组的数据。很酷吧！

HeadFirst：确实可以想象。谢谢你今天过来澄清一些谜团，还有你跟循环的关系。

Array：我也很高兴。有空随时欢迎来找我哦！

磨笔上阵

为 Mandango 程序创建 seats 数组，然后以循环处理该数组，使用 alert 框提出空位信息。

....

....

....

....

....

....

....

没有蠢问题

问：for 循环的运转可能永无止境吗？

答： 呃，可能，那就是令人畏惧的无限循环。当然有可能创建不会停止的循环，注定在有限时空里一再循环……至少在你重新载入网页前。无限循环被视为不好，在于它们阻挡了脚本的其他活动——它是JavaScript版的锁定程序，很像Windows著名的蓝色屏幕，只是没那么明显。

无限循环，发生在循环的计数器不会正确更新或不会改变，因而无法造成条件测试为 false 值的状况。因此，你应该再三仔细地检查 for 循环的测试条件以及更新计数器的逻辑。对了，如果你发现脚本突然停下来没有动作，就表示你遇到无限循环啦！

问：for 循环的动作可以使用复合语句吗？

答： 当然可以！事实上，除了最简单的循环外，你将需要复合语句。因为最实用的循环最后都需要循环处理不只一段语句。

问：当循环的条件测试结果为 false 时，动作部分还会运行最后一次吗？

答： 不会。for循环的动作部分只在条件评估为true时执运行。一旦评估为false，循环立刻结束，不会运行其他代码。

问：索引数组一定从 0 开始编列索引值吗？

答： 对，也不对。预设中，所有索引数组均从 0 开始。你也可以改写这项设计，改设数值键为任何数值，虽然这样不符合常规。除非你有非常好的设计理由，不然最好不要打破从 0 开始的惯例，否则可能造成困扰。

问：存储于数组里的数据均为相同类型吗？

答： 不见得如此。以循环的用途而言，数组数据均为相同类型就很重要，因为循环的重点是处理一组相似的数据。假如你使用循环处理成绩数组并计算平均成绩，如果此时数组中有boolean值，有意义吗？所有数组元素都应该是数字才对。虽然数组可以包含不同类型的值，但最好还是存储相同类型的数据，尤其是存储一组相似数据时。

为 Mandango 程序创建 seats 数组，然后以循环处理该数组，使用 alert 框提出空位信息。

既然数组索引从 0 开始，循环计数器也从 0 开始。

创建 seats 数组并初始化为 boolean 值。

以逗号区隔数组值。

递增循环计数器，因为我们在计算数量。

虽然不能直接用 9，但使用 array 对象的 length 特性其实更好。因为就算 seats 数组存储的数据改变了，它还是可用。

```
var seats = [ false, true, false, true, true, true, false, true, false ];
for (var i = 0; i < seats.length; i++) {
  if (seats[i])
    alert("Seat " + i + " is available.");
  else
    alert("Seat " + i + " is not available.");
}
```

其实它把循环计数器当成数组的索引，以取得所有数组值。

根据空位（true）或非空位（false）而呈现不同的 alert 框。

Seat 0 is not available.

OK

Seat 1 is available.

OK

复习要点

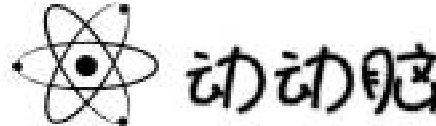
- for循环能把一段JavaScript代码重复**特定次数**。
- 递增（++）与递减（--）运算符，提供更新循环计数器的便利方式。
- 数组能把多段数据存储在**同一个地方**。
- 虽然数组存储了多段信息，但只有一个变量名称。
- 索引数组透过称为**索引**（index）的数值键访问。
- 索引数组与循环是好搭档，让我们能用循环计数器逐一处理数组里的数据。

从 JavaScript 到 HTML

Mandango 以一个 boolean 数组表示空位。所以，下一步就是把数组转换成反映座位划位状况的 HTML 图像（请至 *http://www.headfirstlabs.com/books/hfjs/* 下载），而后呈现在 Mandango 的网页上。

```
var seats = [ false, true, false, true, true, true, false, true, false ];
```

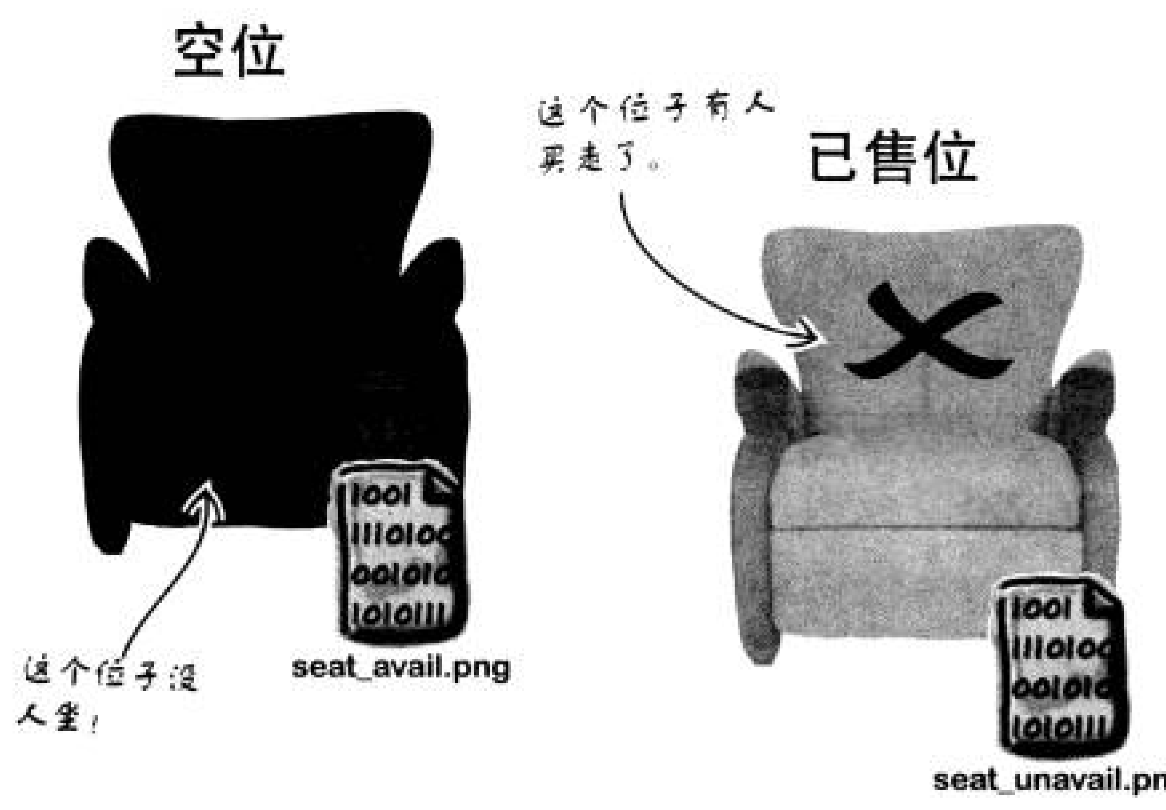
上图看起来还不错，实际上却没有任何代码用于对照boolean数组与网页上的座椅图像。这下子有问题了。



如何联结记录划位状况的 JavaScript 数组以及 Mandango 网页上的座椅图像?

Mandango的划位状况：视觉版

要联结JavaScript数组与HTML图像，首先需确认图像的编排采用可访问的方式，然后决定代表不同划位状况的图像。我们先解决第二项任务。

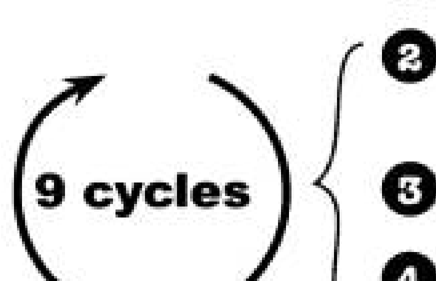


这些座位图像都指派给HTML的座椅图像的 src 属性，以设定出现在网页上的图像。

```
<img id="seat8" src="seat_unavail.png" alt="Unavailable" />
```

这个ID在映射数组与图像时极端重要——它必须从0开始，到8结束，与数组索引一致。

接下来的挑战，变成以循环处理boolean数组，为HTML的每个<img>标签设置座椅图像。这项任务的必需步骤与循环处理 seats 数组惊人相似。事实上，唯一差别只有动作那部分不同而已。



1. 计数器变量 i 初始化为0。
2. 检查 i 是否小于数组长度（9个元素）。倘若小于数组长度，则移往第3步并继续下一轮循环；如果不是，则离开循环。
3. 运行循环的动作部分代码，本例为**设置座椅图像**。
4. 增加 i并回到第2步，开启下一轮可能的循环。

放大 initSeats() 函数

Mandango 的座位状态于 initSeats() 函数里初始化，该函数使用把座位初始化的循环，负责把 JavaScript 数组映射至 HTML 的座位图像。

由 initSeats() 实践的初始化过程，包括设置网页上的座椅图像，这部分与 for 循环的初始化步骤不太一样。

1 循环计数器从 0 开始，因为索引数组从 0 开始。

2 测试条件检查所有座位是否都经过循环的处理。

4 递增循环计数器。

```
function initSeats() {
  // Initialize the appearance of all seats
  for (var i = 0; i < seats.length; i++) {
    if (seats[i]) {
      // Set the seat to available
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
    else {
      // Set the seat to unavailable
      document.getElementById("seat" + i).src = "seat_unavail.png";
      document.getElementById("seat" + i).alt = "Unavailable seat";
    }
  }
}
```

3 如果 seats 值为 true，则设定该座位为空位。

3 座椅图像的 ID 由每一轮循环的计数器创建。

3 如果 seats 值为 false，则设定该座位为不可划位。

```
<body onload="initSeats();">
  <div style="margin-top:75px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" /><br />
  </div>
</body>
</html>
```

每张座椅图像的 src 与 alt 属性均被动态调整。

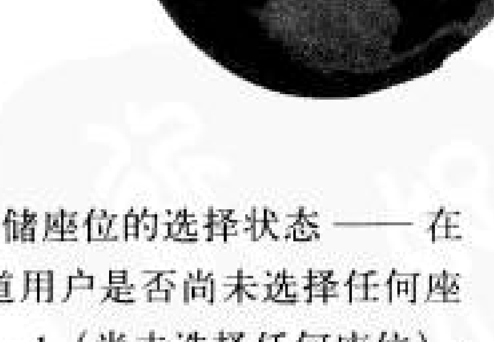
这张座椅图像的 ID 为 "seat6"。

非壮汉专用的划位程序

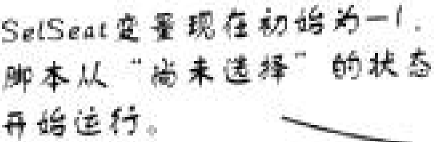
座位状态初始化后，终于可以前进到划位的部分，这才是Mandango的重点。Seth与Jason决定先尝试寻找单一座位的脚本，稍后再投入寻找三个连续座位的终极版本。寻找单一座位可简化目前的任务，他们可据此逐步建立应用程序。

既然Seth与Jason现在想搜索单一座位，脚本需要的第一件东西，就是追踪座位选择的变量。

这个变量存储划位信息，且脚本生存的期间都需要这个变量，所以它必为全局变量。所以，为用户处理座位搜索任务的 `findSeat()` 函数，将依赖变量 `selSeat` 以存储已选座位的数组。



Seth提出了很好的问题。变量 `selSeat` 存储座位的选择状态——在范0到8里已被选走的座位。但你还需要知道用户是否尚未选择任何座位。可用特殊值指示**尚未选择**的状态，例如−1（尚未选择任何座位）。所以变量 `selSeat` 一开始需要初始化为−1。



```
var selSeat = -1;
```

有了座位选择状态的变量，我们就准备好组合 `findSeat()` 函数了。这个函数将搜索座位数组、寻找空位，而后提醒用户接受或拒绝空位的选择。虽然大块头的壮汉们不太可能满意这个简易版的Mandango，不过它是成功的一小步！

JavaScript 冰箱磁铁

Mandango 的 findSeat() 函数负责为用户寻找空位，而后确认或取消划位。拿起下面的小磁铁，协助 Seth 与 Jason 补完消失的代码。

```
function findSeat() {
  // If seat is already selected, reinitialize all seats to clear them

  if (.................. >= 0) {

    .................. = -1;

    ...................... ();
  }

  // Search through all the seats for availability
  for (var i = 0; i < seats.length; i++) {
    // See if the current seat is available

    if (......................) {
      // Set the seat selection and update the appearance of the seat

      .................. = i;

      document.getElementById("seat" + i)............ = "seat_select.png";

      document.getElementById("seat" + i)............ = "Your seat";

      // Prompt the user to accept the seat

      var ............... = confirm("Seat " + (i + 1) + " is available. Accept?");

      if ( ................. ) {
        // The user rejected the seat, so clear the seat selection and keep looking

        ................... = -1;

        document.getElementById("seat" + i)............= "seat_avail.png";

        document.getElementById("seat" + i)............= "Available seat";
      }
    }
  }
}
```

initSeats　alt　accept　selSeat　src　seats[i]　!accept

JavaScript 冰箱磁铁解答

Mandango 的 findSeat() 函数负责为用户寻找空位，而后确认或取消划位。拿起下面的小磁铁，协助 Seth 与 Jason 补完消失的代码。

```
function findSeat() {
  // If seat is already selected, reinitialize all seats to clear them
  if (selSeat >= 0) {
    selSeat = -1;
    initSeats();
  }

  // Search through all the seats for availability
  for (var i = 0; i < seats.length; i++) {
    // See if the current seat is available
    if (seats[i]) {
      // Set the seat selection and update the appearance of the seat
      selSeat = i;
      document.getElementById("seat" + i).src = "seat_select.png";
      document.getElementById("seat" + i).alt = "Your seat";

      // Prompt the user to accept the seat
      var accept = confirm("Seat " + (i + 1) + " is available. Accept?");
      if (!accept) {
        // The user rejected the seat, so clear the seat selection and keep looking
        selSeat = -1;
        document.getElementById("seat" + i).src = "seat_avail.png";
        document.getElementById("seat" + i).alt = "Available seat";
      }
    }
  }
}
```

如果 selSeat 不是 -1，开始新一轮搜索并重设 seats。

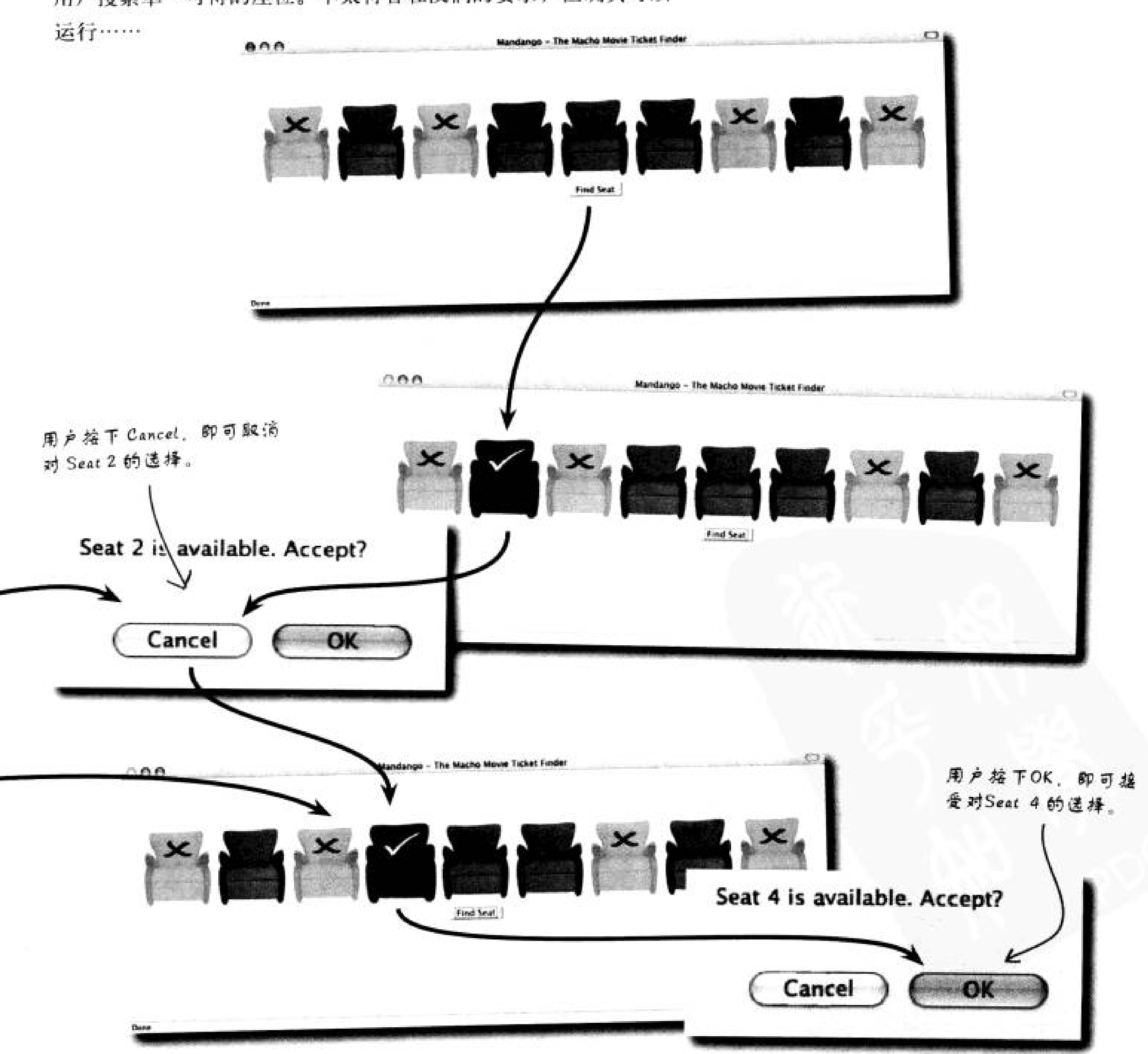
如果座位是空位，seats[i] 将为 true。

用户看到的号码比索引大一号，因为大多数用户都从 1 开始计数，而非 0。

确认用户是否接受这个空位。

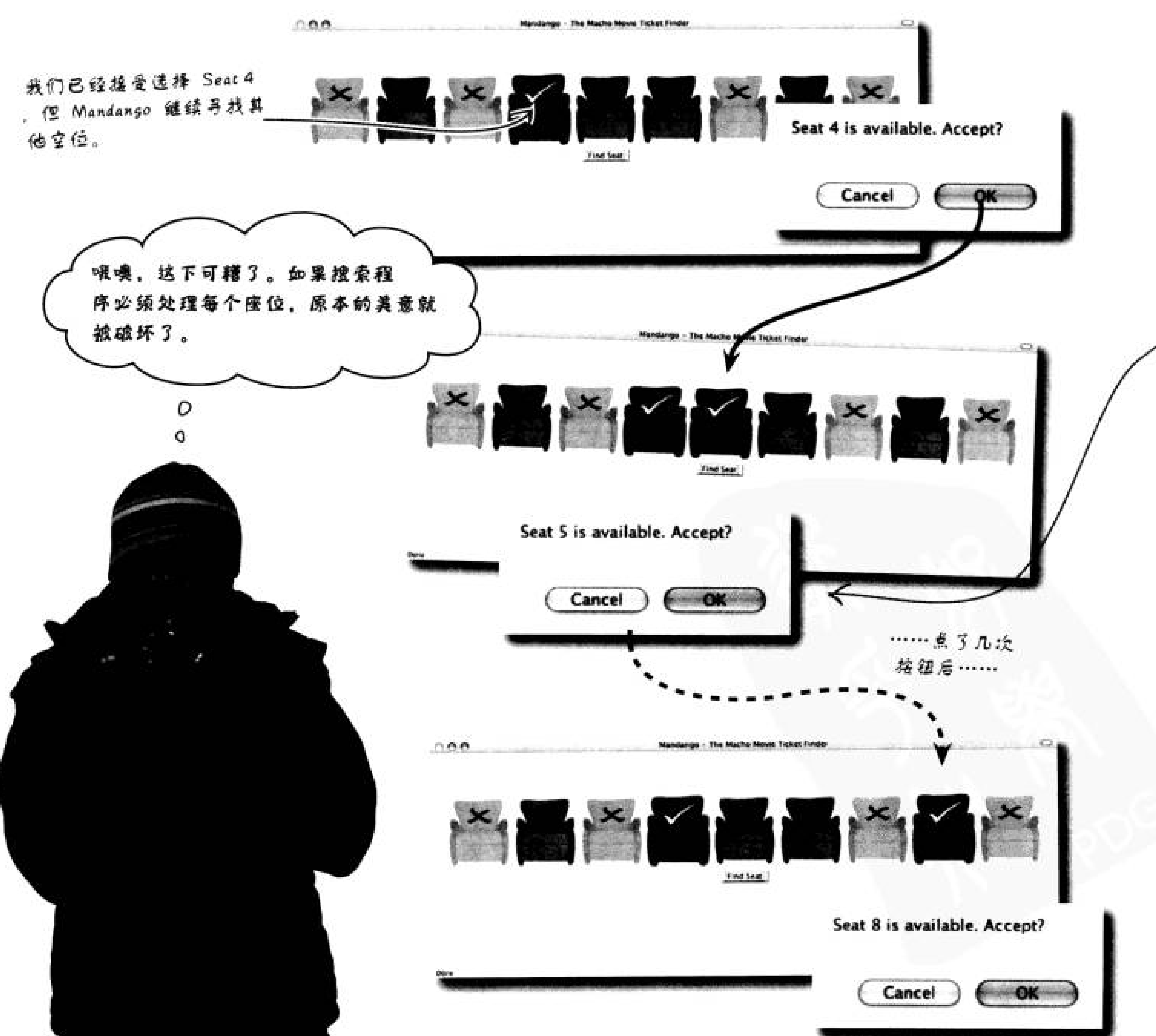
试行：寻找单一座位的程序

搜索单一座位的 Mandango 版本使用 for 循环及一个数组，让用户搜索单一可得的座位。不太符合壮汉们的要求，但确实可以运行……



太多好事也不好：结束不了的循环

技术上，Mandango 的单一座位版可以找到单一空位，却出现了程序不知何时结束的问题。就算用户选择了某个空位并按下“OK”，脚本仍会继续运行，继续跑过剩余空位。



循环总是需要一个（或两个）结束的信号！

我们的划位程序为何过度热心？似乎是因为循环不会在划位后结束。Jason 认为应该更仔细地查看 findSeat()函数的 for 循环了。

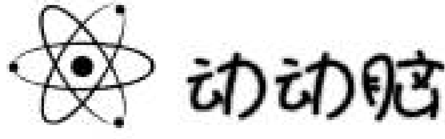
confirm() 函数提示用户回答“是”(true) 或“否”(false)，而后返回 true/false 值。

```
for (var i = 0; i < seats.length; i++) {
  // See if the current seat is available
  if (seats[i]) {
    // Set the seat selection and update the appearance of the seat
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Prompt the user to accept the seat
    var accept = confirm("Seat " + (i + 1) + " is available. Accept?");
    if (!accept) {
      // The user rejected the seat, so clear the seat selection and keep looking
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }        ?
}
```

如果用户接受某个空位，不会发生任何事情，循环继续运转。

这部分是用户“未”接受空位时运行的代码。

所以说，用户按下“Cancel”以不选择某个空位时，selSeat 变量设为 -1，循环继续运行。然而，代码完全没有提到选择空位后该怎么办。目前只能让selSeat变量记忆当前座位，但却没办法停止循环，不让循环继续寻找座位。



当用户按下 OK 按钮以接受当前的空位后，还需要有什么行动？

休息一下：break

Mandango 程序代码的问题，在于用户选择座位后，需要脱离循环。可行的修正方式之一，是尝试重设计数器值，使数目大于数组长度，以欺骗 for 循环。

```
i = seats.length + 1;
```

这么一来，条件测试将强制失败，而结束循环……但是，另有强制结束循环的更佳方式。

虽然上例是个非常聪明的技巧，但还有更佳的方式，而且不需在计数器上动手脚以欺瞒循环条件句。break 语句专为结束一段代码而设计，当然也能结束循环。

```
break;
```

立刻跳出循环，不传送任何指令。

当循环遇到 break 语句，它立刻结束，完全无视条件句。所以，break 语句提供了立刻跳出循环、没有多余疑问的便利方式。

continue 与 break 的关系很密切，它会摆脱当前这一轮循环，但不会完全离开循环。换句话说，你可以使用 continue，强迫循环跳入下一轮。

跳出当前这一轮循环，前往执行下一轮。

```
continue;
```

break 与 continue 都极度适合调整循环的控制，不过 Seth 和 Jason 在 Mandango 上遇到的问题，将由 break 提供解决方案。

我喜欢 break 语句。它能摆脱所有不必要的循环。

没有蠢问题

问：使用 break 语句时，当前这轮 for 循环的剩余动作代码仍会运行完吗？

答： 不会。break 语句强制“立刻”结束循环，完全缩短了正常的循环流程。

问：为什么利用循环计数器以跳出循环，是项不好的举动呢？

答： 因为循环计数器若未用在正途，有可能造成不寻常的缺陷风险。此例的计数器不是用于访问数组（预期用途），而是手动强迫计数器变成超出数组长度的值，用来结束循环。一般而言，我们希望循环的“更新”部分是唯一改变循环计数器的地方。总是会有允许投机取巧的特殊案例，但不是这个例子——break 语句已能妥善地处理循环的中断，又不会造成任何困扰。

磨笔上阵

Mandango 的 findSeat() 函数的 for 循环，需要在用户接受座位选择后中断循环。请帮忙填入用于中断循环的代码，务必加入说明代码工作方式的注释。

```
// Search through all the seats for availability
for (var i = 0; i < seats.length; i++) {
  // See if the current seat is available
  if (seats[i]) {
    // Set the seat selection and update the appearance of the seat
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Prompt the user to accept the seat
    var accept = confirm("Seat " + (i + 1) + " is available. Accept?");

    .............................................................

    .............................................................

    .............................................................

    .............................................................
    else {
      // The user rejected the seat, so clear the seat selection and keep looking
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }
}
```

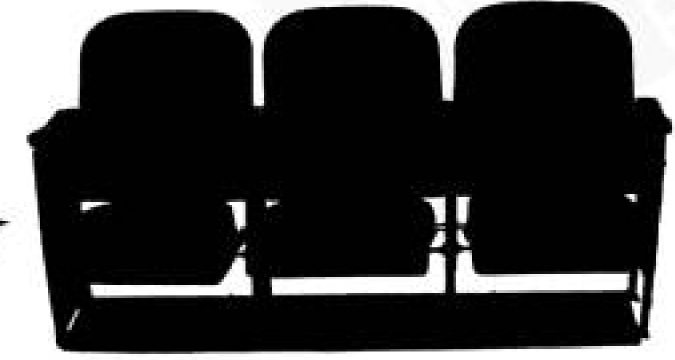
Mandango 的 findSeat() 函数的 for 循环，需要在用户接受座位选择后中断循环。请帮忙填入用于中断循环的代码，务必加入说明代码工作方式的注释。

```
// Search through all the seats for availability
for (var i = 0; i < seats.length; i++) {
  // See if the current seat is available
  if (seats[i]) {
    // Set the seat selection and update the appearance of the seat
    selSeat = i;
    document.getElementById("seat" + i).src = "seat_select.png";
    document.getElementById("seat" + i).alt = "Your seat";

    // Prompt the user to accept the seat
    var accept = confirm("Seat " + (i + 1) + " is available. Accept?");
    if (accept) {
      // The user accepted the seat, so we're done
      break;
    }
    else {
      // The user rejected the seat, so clear the seat selection and keep looking
      selSeat = -1;
      document.getElementById("seat" + i).src = "seat_avail.png";
      document.getElementById("seat" + i).alt = "Available seat";
    }
  }
}
```

真正壮汉版的 Mandango

设计 Mandango 的原始意图，就是允许用户搜索三个一组的电影院空位。单一空位搜索成功后，Seth 与 Jason 终于准备专心制作真正“有男人味”的电影院划位程序。他们需要检查三个一组的连续空位。

一排三个连续的座位……
很多空间！



放大代码

利用嵌套if语句寻找连续三个空位。

如果找到连续三个空位，设定为选择第一个位置。

把三个空位的状态都改为"已选择"的图像，让用户看到还有哪些空位。

```
for (var i = 0; i < seats.length; i++) {
  // See if the current seat plus the next two seats are available
  if (seats[i]) {
    if (seats[i + 1]) {
      if (seats[i + 2]) {
        // Set the seat selection and update the appearance of the seats
        selSeat = i;
        document.getElementById("seat" + i).src = "seat_select.png";
        document.getElementById("seat" + i).alt = "Your seat";
        document.getElementById("seat" + (i + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i + 1)).alt = "Your seat";
        document.getElementById("seat" + (i + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i + 2)).alt = "Your seat";

        // Prompt the user to accept the seats
        var accept = confirm("Seats " + (i + 1) + " through " + (i + 3) + " are available. Accept?");
        if (accept) {
          // The user accepted the seat, so we're done
          break;
        }
        else {
          // The user rejected the seats, so clear the seat selection and keep looking
          selSeat = -1;
          document.getElementById("seat" + i).src = "seat_avail.png";
          document.getElementById("seat" + i).alt = "Available seat";
          document.getElementById("seat" + (i + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i + 1)).alt = "Available seat";
          document.getElementById("seat" + (i + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i + 2)).alt = "Available seat";
        }
      }
    }
  }
}
```

如果用户不想选择座位，再把图像改回"空位"。

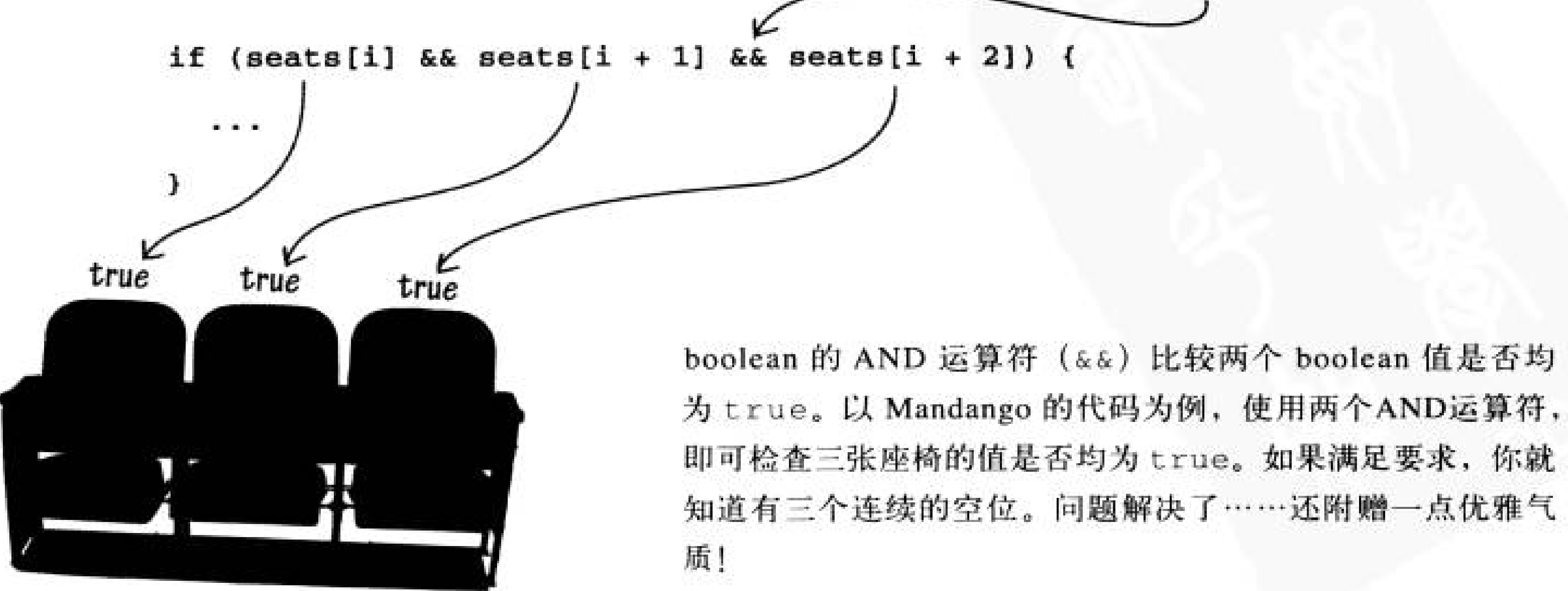
* 请记得：上例代码以及所有Mandango的代码及图像，都能从http://www.headfirstlabs.com/books/hfjs/取得。

如果能够结合这些嵌套if语句，形成更优雅的构造，该有多好？大概是我的白日梦吧？

合逻辑、优雅、设计良好的方案——&&

处理 Mandango 需要的连续三个空位检查时，还有更好的方式。嵌套 if 的版本的确可行，但还有改善空间，而且这个改变主要与程序的优雅有关。

虽然Seth打滚耍赖不愿接受“优雅”的改变，但有些情况的确值得让代码更为“优雅”——“干净、效率、易于理解和维护”的另一种说法。以嵌套 if 为例，把所有语句结合成一个 if 将优雅许多……但要怎么做？

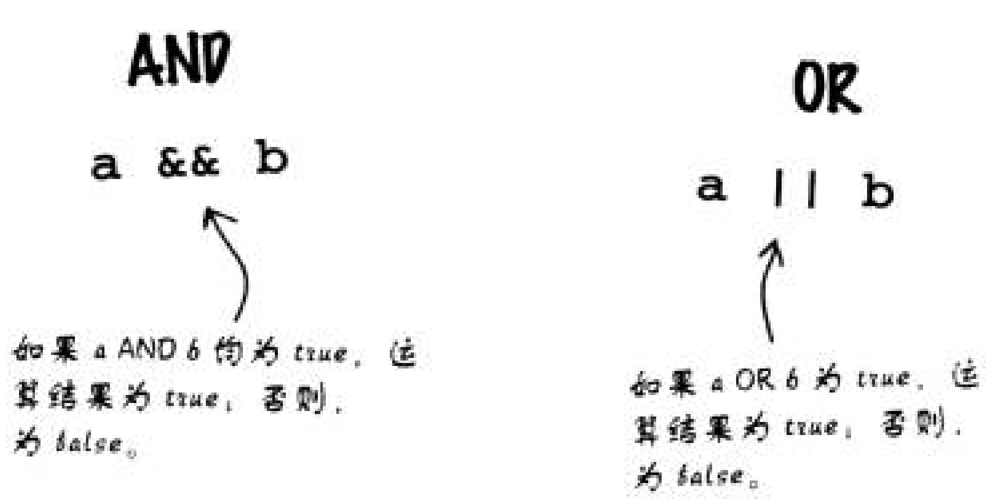



```
if (seats[i] && seats[i + 1] && seats[i + 2]) {
  ...
}
```

boolean 的 AND 运算符（&&）比较两个 boolean 值是否均为 true。以 Mandango 的代码为例，使用两个AND运算符，即可检查三张座椅的值是否均为 true。如果满足要求，你就知道有三个连续的空位。问题解决了……还附赠一点优雅气质！

boolean 运算符的逻辑揭密

各位已经看过几种比较运算符，例如 == 与 <。你看到的大多数比较运算符都先比较两个值，然后产生 boolean 值的结果。boolean 逻辑运算符的结果也是 boolean 值，但它们只负责比较 boolean 值 —— 执行 boolean 逻辑比较。



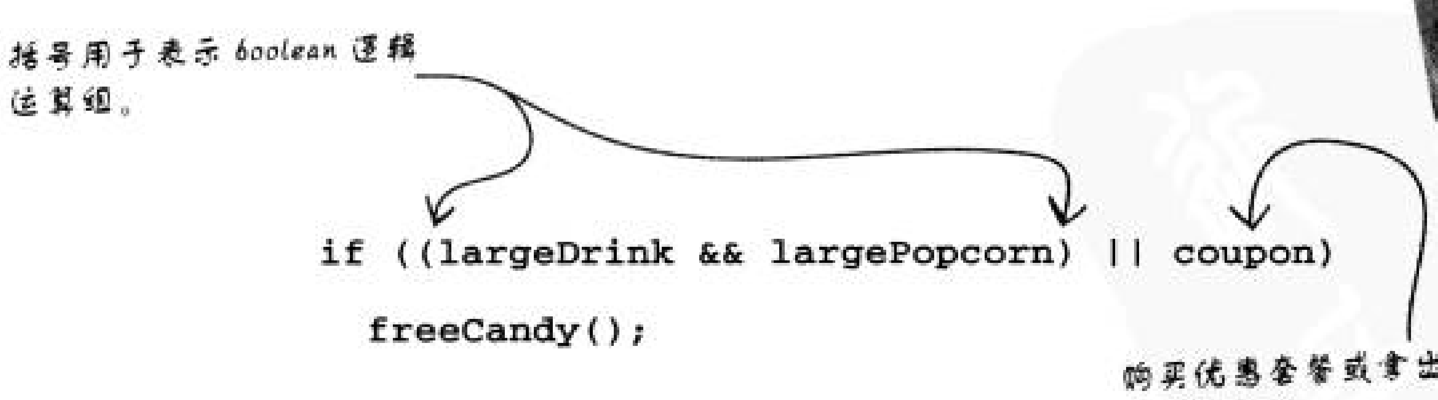
你看过它了！

NOT

!a

如果 a 为 true，运算结果为 false；如果 a 为 false，则结果为 true。

boolean 逻辑运算可互相结合，建立很有趣的逻辑比较；通常用于制作复杂的决策。



本例中，AND 运算符用于检查大杯饮料与大桶爆米花……这是优惠套餐！购买优惠套餐，附赠免费的糖果。或者（OR），还有另一个获赠糖果的途径 —— 优惠券。所以，你可以订购大杯饮料 AND 大桶爆米花，OR 拿出你的优惠券。如果没有 boolean 逻辑运算符的帮忙，将难以完成这类决策。

结合 boolean 逻辑运算符，可达成复杂的决策。

没有蠢问题

问：我还是看不出来一般 boolean 运算符与 boolean 逻辑运算符的不同。到底哪里不同？

答：首先，它们都是 boolean 运算符，所以运算完毕的结果都是 boolean 值。唯一的差别在于用在运算时的数据。一般boolean 运算符用于各种数据，它们完成一般比较，例如“等于”、“不等于”、“大于”等等。boolean 逻辑运算符只运用在 boolean 数据上，因此用于完成 AND、OR、NOT 等运算。所以，boolean 逻辑运算符只与 true/false 信息合用，一般 boolean 运算符则可运用于各种数据类型。

问：所以，NOT 运算符是个 boolean 逻辑运算符啰？

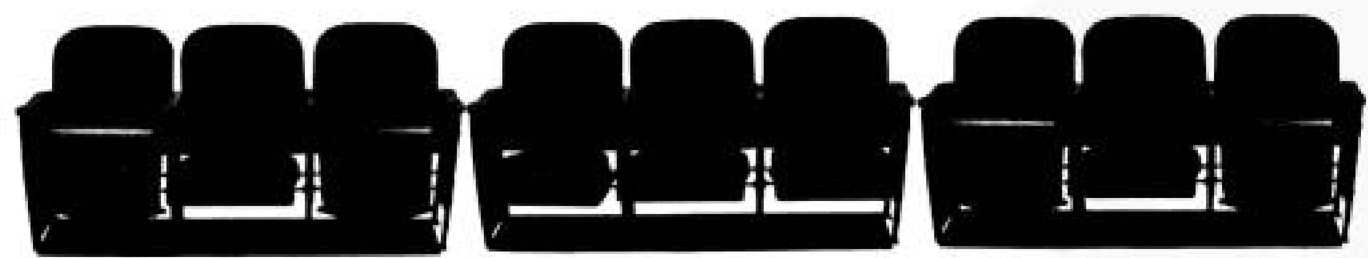
答：是的。它只对 boolean 值运算，符合 boolean 逻辑运算符的资格。它也是个一元运算符（unary operator），因为只需运算一段数据。

问：括号如何既尊重 boolean 运算符，又产生作用呢？

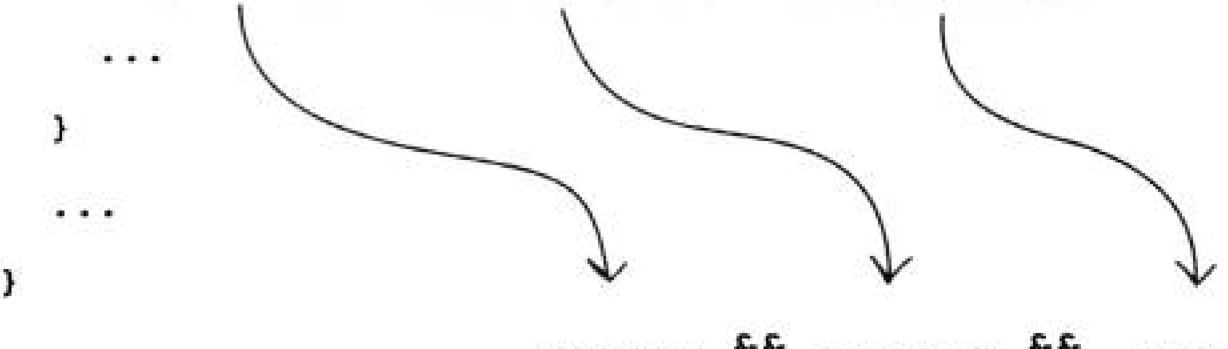
答：括号能让我们变换默认的运算符估算顺序，不只是可用于boolean运算符。把一项运算放在括号里，做成组，可强制该项运算优先执行。以免费糖果的代码为例，`largeDrink && largePopcorn`即被强制比||运算先执行，因为它被放在( )中。

磨笔上阵

Mandango 的 `for` 循环现在刚进入第六轮（i = 5），它需要你帮忙检查空位，判断是否出现三个连续的空位，并实践一些 boolean 逻辑。



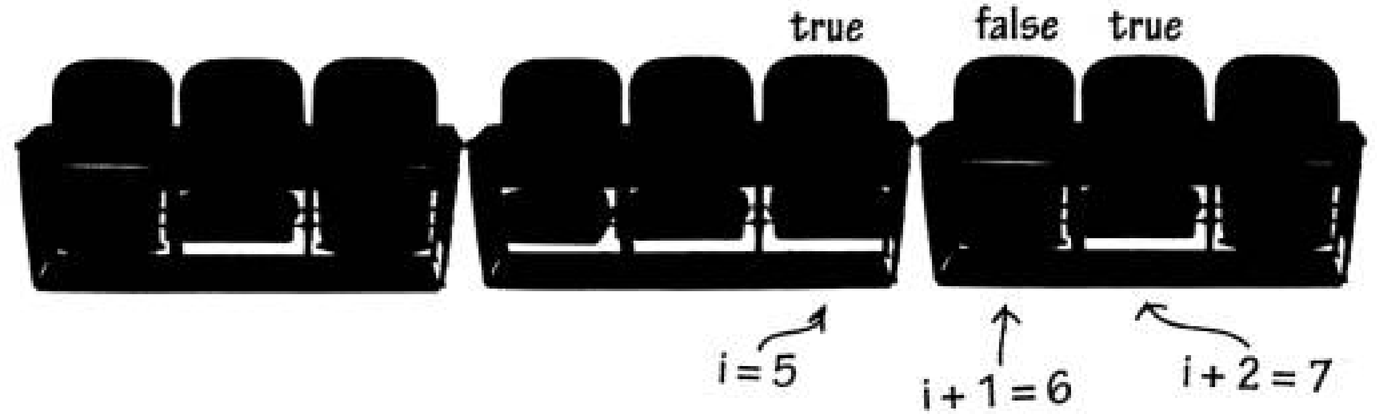
```
for (var i = 0; i < seats.length; i++) {
  // See if the current seat plus the next two seats are available
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    ...
  }
  ...
}
```



.... && .... && .... = ...

磨笔上阵解答

Mandango 的 for 循环现在刚进入第六轮（i = 5），它需要你帮忙检查空位，判断是否出现三个连续的空位，并实践一些 boolean 逻辑。



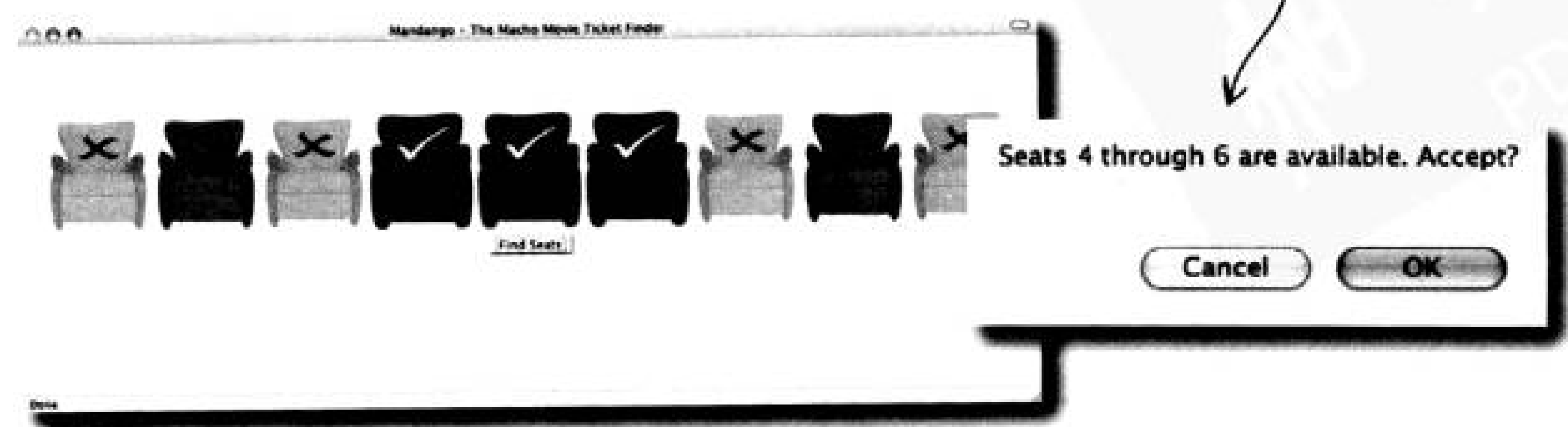

```
for (var i = 0; i < seats.length; i++) {
  // See if the current seat plus the next two seats are available
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    ...
  }
  ...
}
```

true && false && true = false

终于，充满男人味的划位程序

Mandango 划位程序现在能正确地搜索出三个连续的空位，就算是肌肉最发达的男士，也将对你的电影院划位程序感到满意。

提醒用户接受三个连续空位。



回到藏宝图

Mandango 上了轨道，我们可以回头寻找隐藏的宝藏。还记得藏宝图吗？

for 循环已能很好地走过藏宝图的第一部分。但第二部分还没解决，而且它的挑战看来也不适合 for 循环的独有技巧。如果不知道循环需要重复的次数，很难设定 for 循环计数器。

动动脑

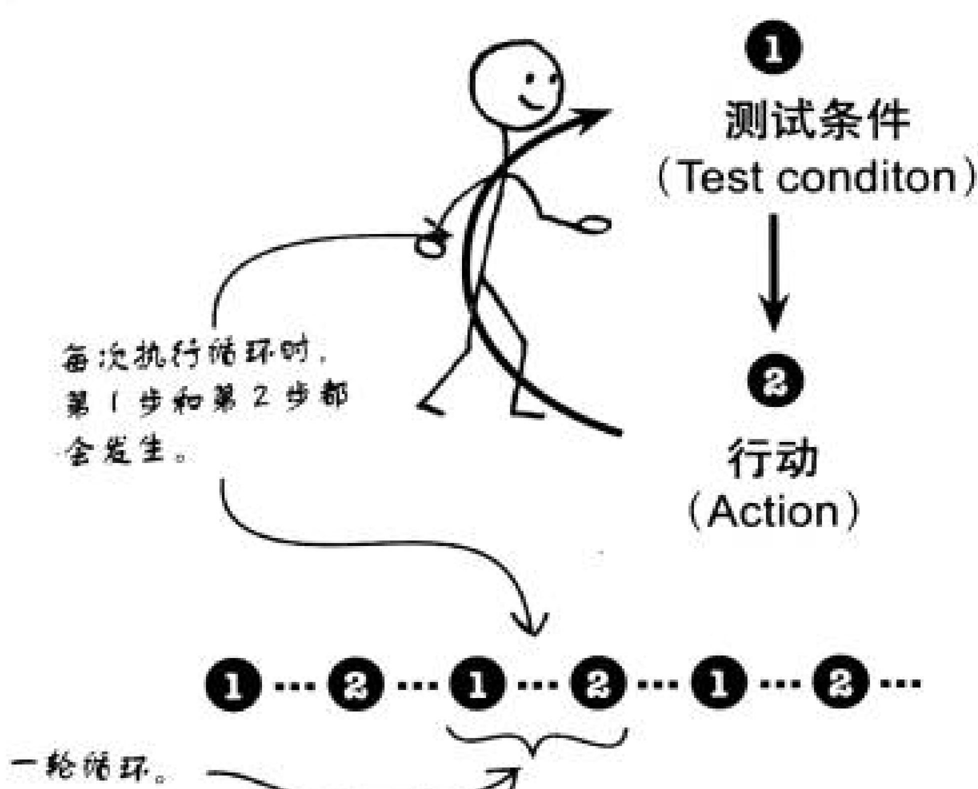
藏宝图上的两个步骤有什么不同？藏宝图的第二部分该如何转换成循环？

继续执行 while 循环，直到遇见条件句

虽然可以创建出执行藏宝图第 2 步骤的for 循环，但还有更好的选择。for 循环的构造以循环计数器的概念为中心，while 循环则设计为等到特定条件的达成。它的条件句不见得需与循环计数器有关。

while 循环由两个不同部分构成：

while 循环能重复代码，直到特定条件句为true。



测试条件
测试条件检查循环是否应该继续循环。

行动
循环的行动部分则是每一轮循环里实际重复的部分。

把 while 循环运用在藏宝图的第二部分，形成简单得不可思议的代码，至少比 for 循环简单：

测试条件 ❶
只在测试评估为 true 时执行另一轮循环，也就是看不到爆米花石头的时候。
`!rockVisible`

行动 ❷
调用 takeSetp() 函数，向前迈步。
`takeStep()`

```
while (!rockVisible)
  takeStep();
```

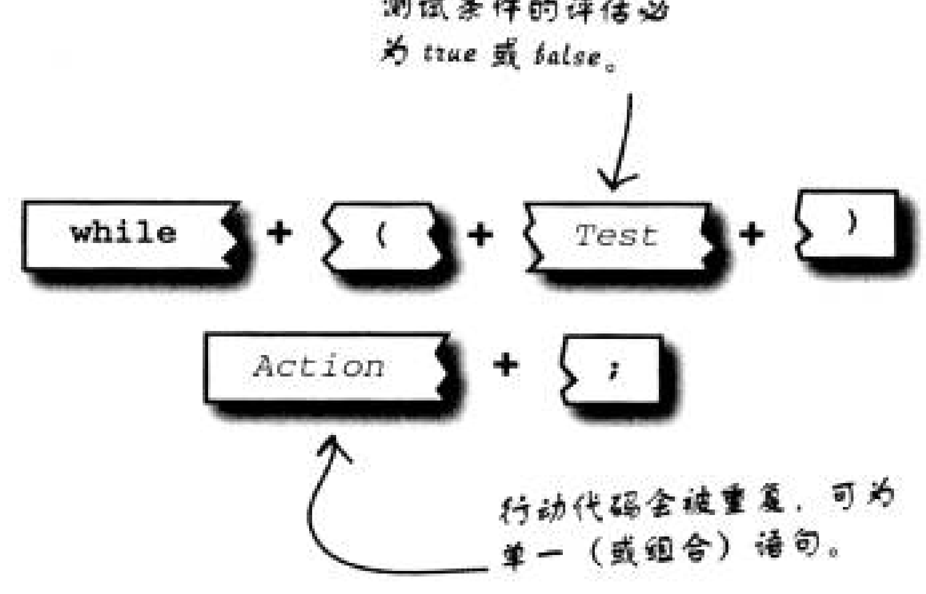
while 循环的各部分运作如下：

❶ 检查是否看不到石头。如果看不到，移向第 2 步，执行一轮循环；如果看到石头，则停止循环。

❷ 运行循环的行动代码，此例是指 takeStep() 函数。

停止 while 循环

结构比 for 循环简单，while 仍必须遵守格式：



小心处理 while 循环的测试条件。

因为 while 循环没有更新循环的内置代码，你必须确定循环里有影响测试条件的程序代码。否则，就有创建无限循环的风险。

重新设计电影片头习题的循环代码，它提醒用户输入大于 0 的数字，然后从该数字倒数计时，就像老电影的片头一样（4、3、2、1，开始！）。这一次用 while 循环改写 for 循环。

```
var count = prompt("Enter a number greater than 0:", "10");
if (count > 0) {

.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
}
else
  alert("The number wasn't greater than 0. No movie for you!");
```

重新设计电影片头习题的循环代码，它提醒用户输入大于 0 的数字，然后从该数字倒数计时，就像老电影的片头一样（4、3、2、1，开始！）。这一次用 while 循环改写 for 循环。

```
var count = prompt("Enter a number greater than 0:", "10");
if (count > 0) {
  var x = count;
  while (x > 0) {
    alert("Starting in..." + x);
    x--;
  }
  alert("Roll film!");
}
else
  alert("The number wasn't greater than 0. No movie for you!");
```

存储数字于 count 变量。

提醒用户输入数字。

确认 count 大于 0。

倒数至 0。

这里还有一个计数器，但创建于 while 循环外。

递减计数器，成为循环行动部分的一部分，与 x = x - 1 的效果相同。

循环的行动部分是个复合语句。

不合格的数据。

倒数结束。

The number wasn't greater than 0. No movie for you!

OK

Roll film!

OK

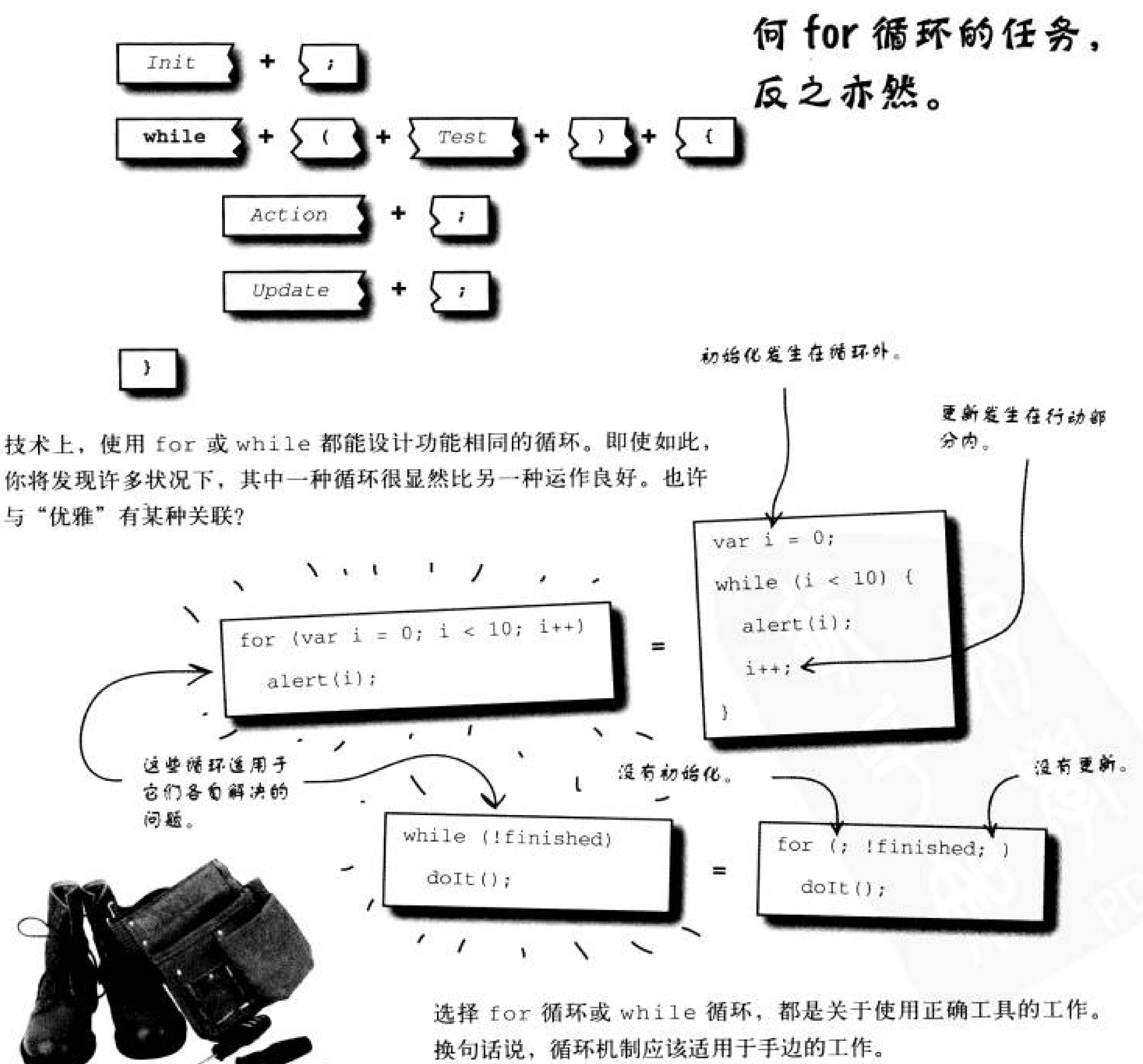
复习要点

- break 语句立即结束循环，跳过任何剩余循环代码。
- boolean 逻辑运算符允许你创建强大的 true/false 逻辑以利决策。
- while 循环一直运行一段代码，只要某个测试条件维持为 true。
- 确定测试条件能够被 while 循环代码影响，以避免无限循环。

选对循环

电影的倒数计时习题显示了 for 与 while 循环通常可解决相同问题。事实上，任何 for 循环都能使用下列形式重组为 while 循环：

while 循环能达成任何 for 循环的任务，反之亦然。



技术上，使用 for 或 while 都能设计功能相同的循环。即使如此，你将发现许多状况下，其中一种循环很显然比另一种运作良好。也许与“优雅”有某种关联？

选择 for 循环或 while 循环，都是关于使用正确工具的工作。换句话说，循环机制应该适用于手边的工作。

今晚主题：for 循环与 while 循环努力地自我重复

For 循环：

啊……又是日复一日重复的老友聚会。

While 循环：

是啊。不过我话说在前头，你那些工作所需的不同步骤，我可是一点兴趣都没有。看起来真是复杂。

For 循环：

我一点都不复杂，只是加了一点用于创建特定循环的结构。想以循环处理某些数字计数器时，因为我能让初始化与更新（控制我的）计数器变得简单，大家都会觉得安心。

While 循环：

没错，但循环不是只有计数。这世上有着各式各样甚至与数字无关的循环。有时你只是需要简单说出“继续做这件事”。这是我的循环风格。

For 循环：

听起来有点模糊，虽然我觉得你的方式也能运作。我倾向于较为精确的运作方式 —— 你也知道，就是密切监视让我走动的计数器。所以我特别注意在开始循环前把自己初始化。我也在每一轮循环结束时自我更新，以确保能如预期地运行。我就是忍不住想要像时钟一样，可能有点精准重复强迫症吧！

While 循环：

我虽然想对你的工作伦理喝彩，但你也知道，循环还是有可能运作得既可靠又可预测，同时又不需一切公式化的初始化与更新手续吧？此外，我重复代码的状况通常都不需要任何初始化，更新也不需要发生在行动代码里。我很满意不用理睬正式性、全心专注在循环的现状。

For 循环：

我知道还有其他构建循环的方式，我只是喜欢严格一点。

For 循环：

While 循环：

我觉得一切都要归究到风格，而每个循环都有自己偏好的风格。你喜欢严格保持对循环的所有控制，而我对于“控制”这回事则比较松散。

没错。好消息就是我们都能以自己的方式完成工作。我也知道，对于简单逻辑控制的循环而言，我的风格有点太严肃了。

你终于说了好话！看来我们还是能和平共处的。

好话不怕多讲。你刚刚那句话讲得真好。

我想我们还是能和平共处的……哎呀，一直重复的本性难移。真不好意思。

别在意，我完全了解。这次谈话真愉快。

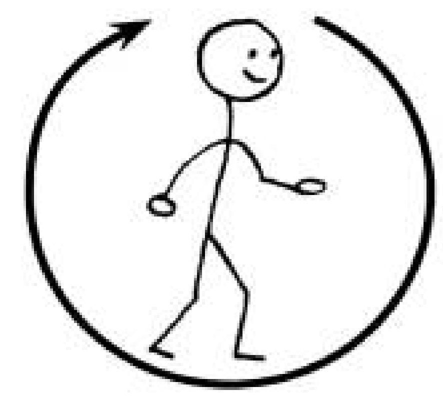
没有蠢问题

问：while 循环看来非常简单。我有错过什么重点吗？

答：没有。你只要知道简单不表示薄弱或有限。换句话说，你可能会对while循环的威力感到惊讶。没错，while循环只由一个测试条件与一段行动代码组成，但组成真正顺畅的循环，通常就只需要这些。再考虑到测试条件也能利用boolean逻辑运算符，事情就更有趣了。不只如此，while循环的动作部分如果利用复合语句，就能尽情包含你想要的程序代码。

问：如果我创建一个以 while (true) 开头的循环，它能运作吗？

答：可以……可能运作得太好了点。问题在于你刚刚创建了一个无限循环，因为它的测试条件永远是true。while循环一直跑到测试条件为false才停止，但本例停不下来。想想看，就在你读这段的时候，无限循环正在跑……一直跑……还在跑……不停地跑……啊！停下来休息一下啦（break）！



问：循环的行动代码（括号里的程序代码）可能完全没被调用吗？

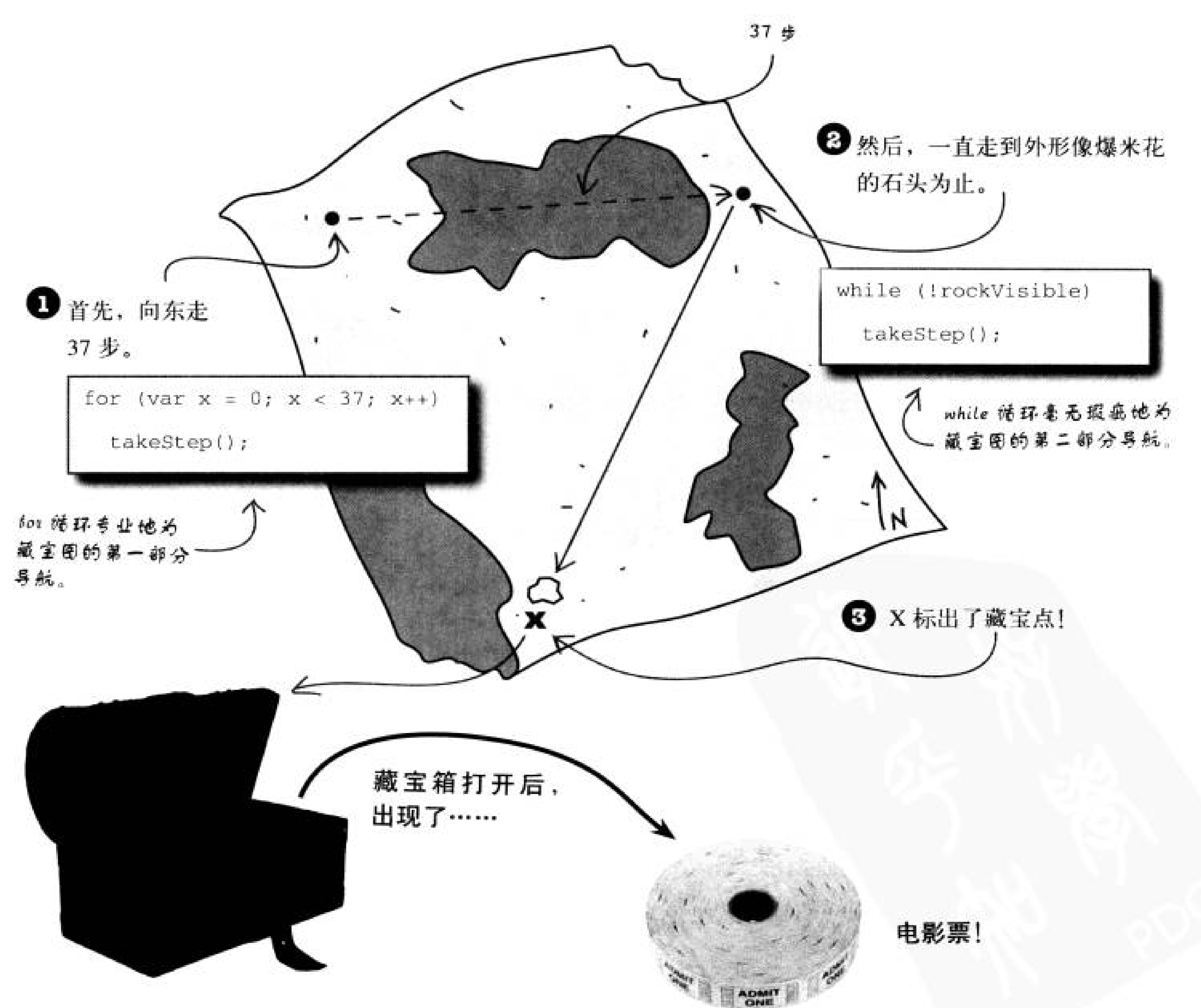
答：有可能。for循环和while循环的测试条件需为true，才能开始运行行动代码。所以，如果条件句最初因某种原因而失败，行动代码自然不会运行，循环则在开始前就已跳出。

问：循环可以放在另一个循环里吗？

答：当然可以！嵌套循环允许不只一层的重复。现在听来有点奇怪，但这个功能很棒哦！当 Mandango 扩大应用到整间电影院时，我们再来进一步讨论嵌套循环。

循环完成时的宝藏

使用 for 循环，再加上 while循环，终于能完整走过藏宝图，导向 X 点的藏宝箱。

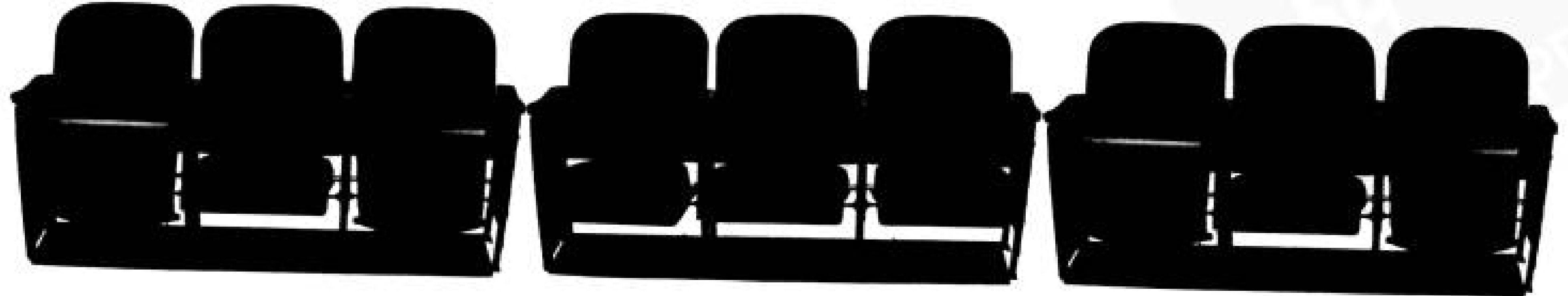


这是什么预兆吗？你的新帮手 while 处理了信息与电影票后，只导回一件事……Mandango！

磨笔上阵

重新设计 **Mandango** 的`findSeats()`函数，请改用`while`循环取代 `for`循环。并增加一个新的循环控制变量`finished`，作为透过测试条件跳出循环的方式，而不是使用 `break`。

```
..............................................................................

..............................................................................
  // See if the current seat plus the next two seats are available
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    // Set the seat selection and update the appearance of the seats
    ...

    // Prompt the user to accept the seats
    var accept = confirm("Seats " + (i + 1) + " through " + (i + 3) +
      " are available. Accept?");

    ..........................................................................

    ..........................................................................

    ..........................................................................

    ..........................................................................
    else {
      // The user rejected the seats, so clear the seat selection and keep looking
      ...
    }
  }

  // Increment the loop counter

  ............................................................................
}
```



磨笔上阵解答

重新设计 Mandango 的 findSeats() 函数，请改用 while 循环取代 for 循环。并增加一个新的循环控制变量 finished，作为透过测试条件跳出循环的方式，而不是使用 break。

循环计数器与变量 finished 的初始化。

只要循环器计数小于 seats 的数字，而且（AND）变量 finished 不为 true，循环继续。

这个循环是我们目前所学的集大成者，它同时依赖计数器与 boolean 逻辑表达式。使用 while 设计混合循环通常比较容易。

```
var i = 0, finished = false;
while ((i < seats.length) && !finished) {
  // See if the current seat plus the next two seats are available
  if (seats[i] && seats[i + 1] && seats[i + 2]) {
    // Set the seat selection and update the appearance of the seats
    ...

    // Prompt the user to accept the seats
    var accept = confirm("Seats " + (i + 1) + " through " + (i + 3) +
      " are available. Accept?");
    if (accept) {
      // The user accepted the seats, so we're done
      finished = true;
    }
    else {
      // The user rejected the seats, so clear the seat selection and keep looking
      ...
    }
  }

  // Increment the loop counter
  i++;
}
```

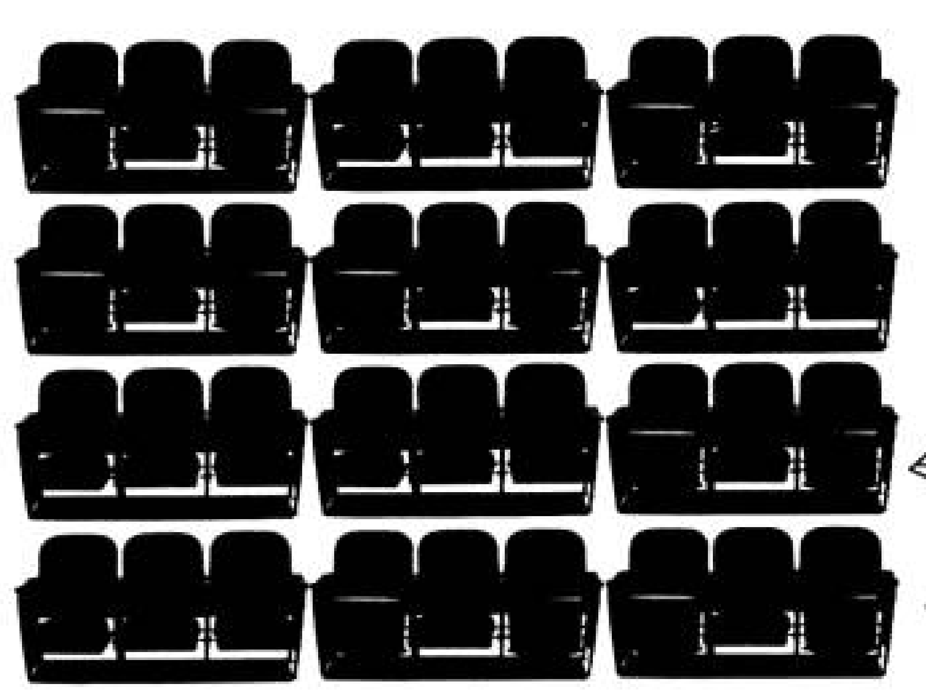
递增循环计数器。

设定变量 finished 为 true，离开循环。因为这个动作会影响测试条件，此处不需要 break。

Mandango加上循环后看起来真不错，不过电影院绝对不只一排座位。我们要想出如何处理更多排座位……

电影院座位数据建模

Jason讲得对。Mandango确实需要处理更多排座位的能力，才能真正派上用场。目前，一排座位就已足够，因为刚好映射到一个表示座位的boolean数组。想扩展成很多排座位，需要扩展数组 —— 需要另一个维度。没错，我们在讨论二维（2-D）数组。



这间电影院有4排座位，每排约为9个位子。嗯，很舒服！

我们需要一个9×4大小的数组，配合实际的座椅数量：4排，每排9个座位。

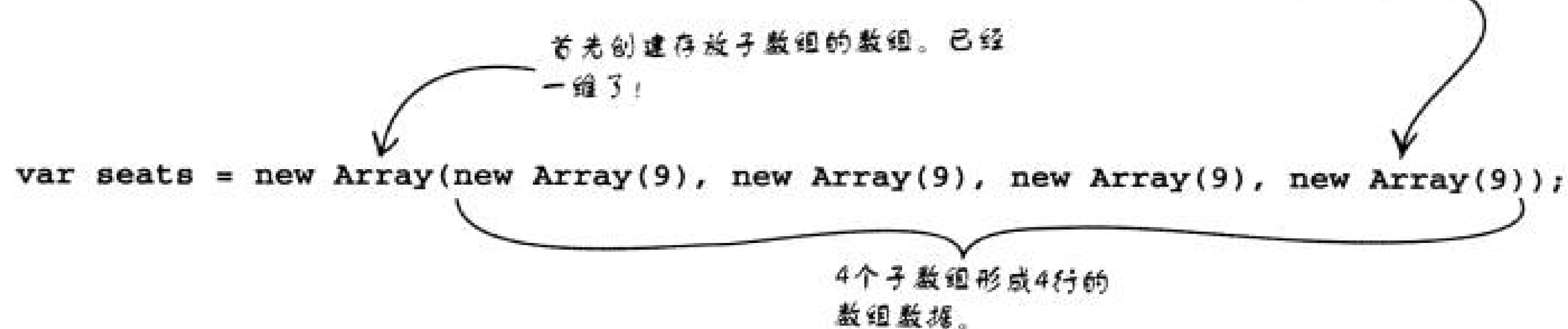
二维数组的每个元素还是boolean。

这里是数组索引的另一维。

数组里的数组：二维数组

创建二维数组，不用戴上特殊镜片或任何装备。事实上，创建二维数组与创建一般（一维）数组很类似，只是数组元素换成多个子数组。这类子数组形成第二维，制造出具有行与列的**数据表**（table）。

再创建作为外层数组元素的子数组。第二维！



以 Mandango 为例，我们已经知道数组元素的初始值，所以使用**不同方式**创建二维数组也算合理 —— 确实牵涉到数组字面量。我们要同时**创建与初始化**数组，这是个双赢的方式。

每个中括号指出了二维数组。

```
var seats = [[ false, true, false, true, true, true, false, true, false ],
  [ false, true, false, false, true, false, true, true, true ],
  [ true, true, true, true, true, true, false, true, false ],
  [ true, true, true, false, true, false, false, true, false ]];
```

每个子数组有自己的数组索引，本例由 0 到 3。

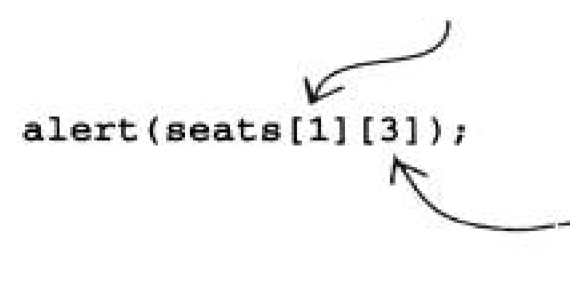
第一组 boolean 值是二维数组的第一行。

false —— 这个座位已经有人坐了。

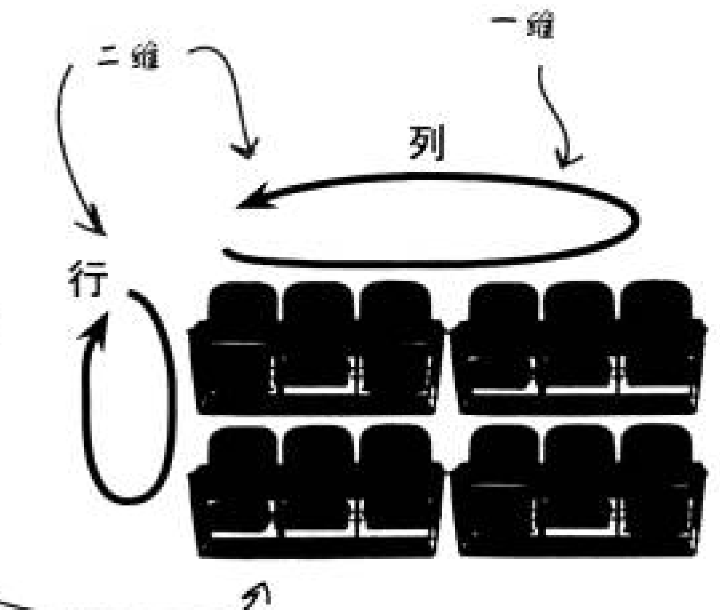
true —— 这个座位是空位。

访问二维数组数据的两个键

访问二维数组与访问一维数组数据的方式其实没什么不一样，只不过我们要多提供一些信息：额外数组的索引。说得更精确一点，你要指定行（row）与列（column）的索引，以指出数据在数组中的位置。例如想取得第2排第4张座椅的值，请使用下列代码：



二维数组让你有办法存储表格数据。



用循环处理一个多维数组，牵涉到为每一维制作一层嵌套循环。所以，二维数组需要两层循环，其中一个循环在另一个里面。外层循环处理数组数据的每一行，内层循环则处理每行中的每一列。

磨笔上阵

以循环处理二维数组seats 存储的座位值，每次遇到空位需提醒用户。

....

....

....

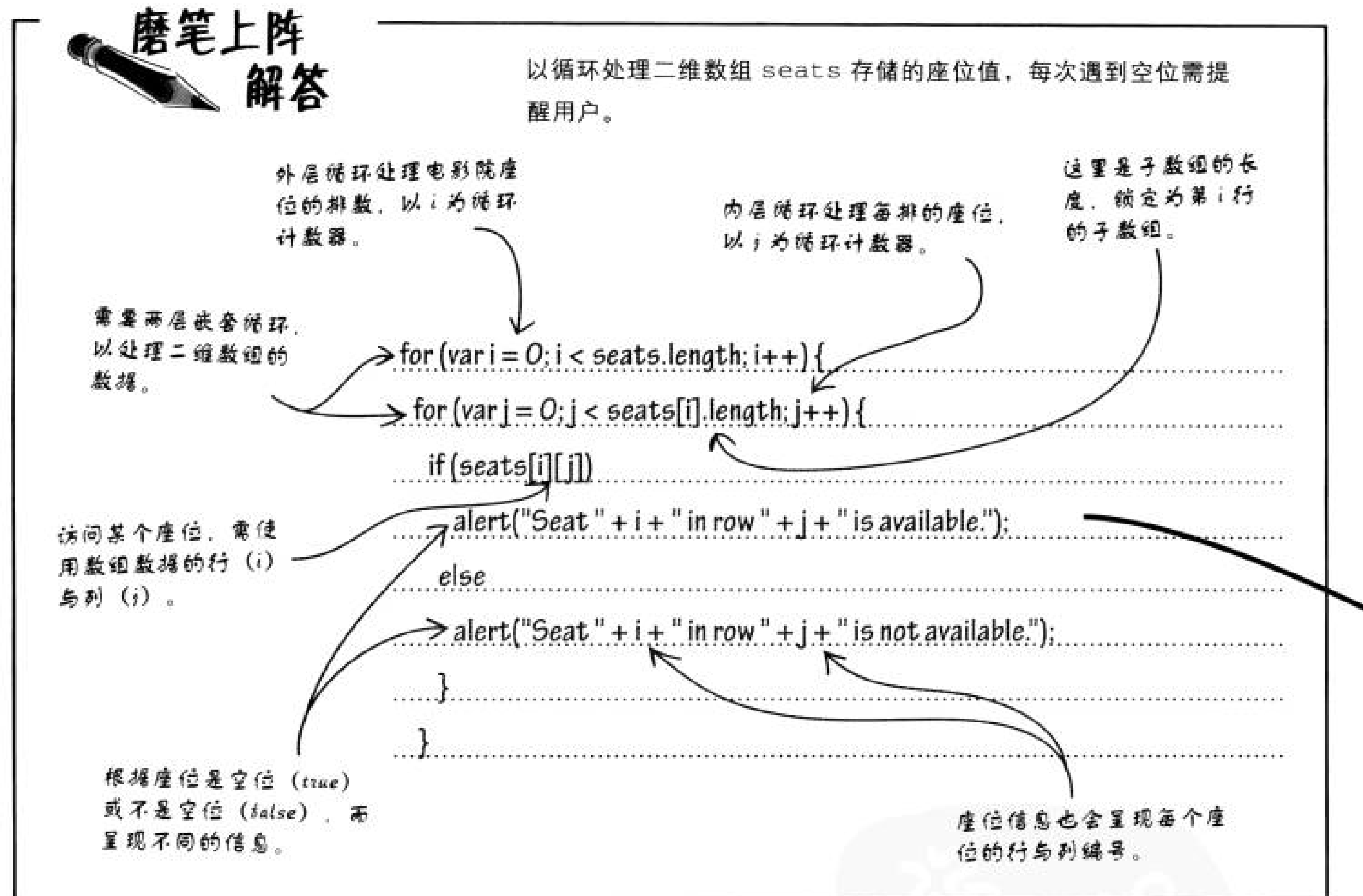
....

....

....

....

....



没有蠢问题

问：数组可以超过二维吗？

答：可以，虽然超过到某个程度后，数据就很难形像化了。三维数组对于建立现实数据模型（例如空间中的 x-y-z 三维座标）可能很方便。超过这个程度的额外维度，大概都可归类为非常独特的情况。增加新的维度时，可想成以子数组换掉原本的元素（个别数组）。

问：如果我在创建数组就已填入数据、做了初始化，事后还可以新增数组元素吗？

答：当然没问题。借由指定新数据至未使用的数组元素，你随时都可对数组增加更多数据。以Mandango为例，增加新的子数组（数组索引为4），就可新增第5排，只需指派子数组至seats[4]。你也可以调用Array对象的 push()方法，在数组尾端增加一个新元素。

问：二维数组必须包含相同行数吗？

答：没有严格的需要。但请记得，如果数组的每行元素数量不相同，你就等于自取灾难之道，因为嵌套循环通常都设计为处理一致的子数组长度。所以，是的，二维数组的每一行的长度确实可以都不一样，但是别冒这个险比较安全点。

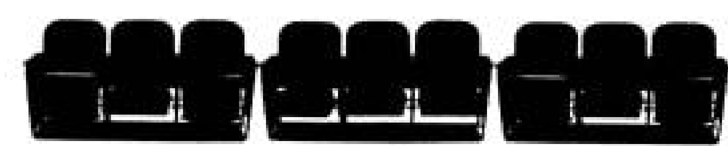
复习要点

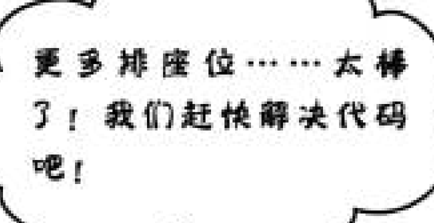
- 二维数组让我们以**表格**结构存储数据的行与列值。
- 访问二维数组里的一块数据时，你必须指定**行**与**列**的索引值。
- **嵌套循环**能用于处理二维数组的数据。
- 就像一般数组，二维数组能从数组对象字面量创建与初始化。

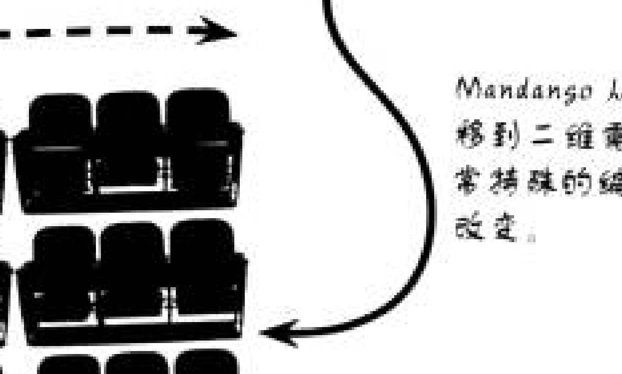
2-D 的 Mandango

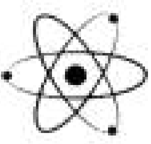
虽然你已经处理过代码片段，但 Mandango 要从一排座位成为整个电影院，需要好好地重新处理脚本才能适应二维的世界。

需要两个循环计数器，才能跑遍座位的二维数组。





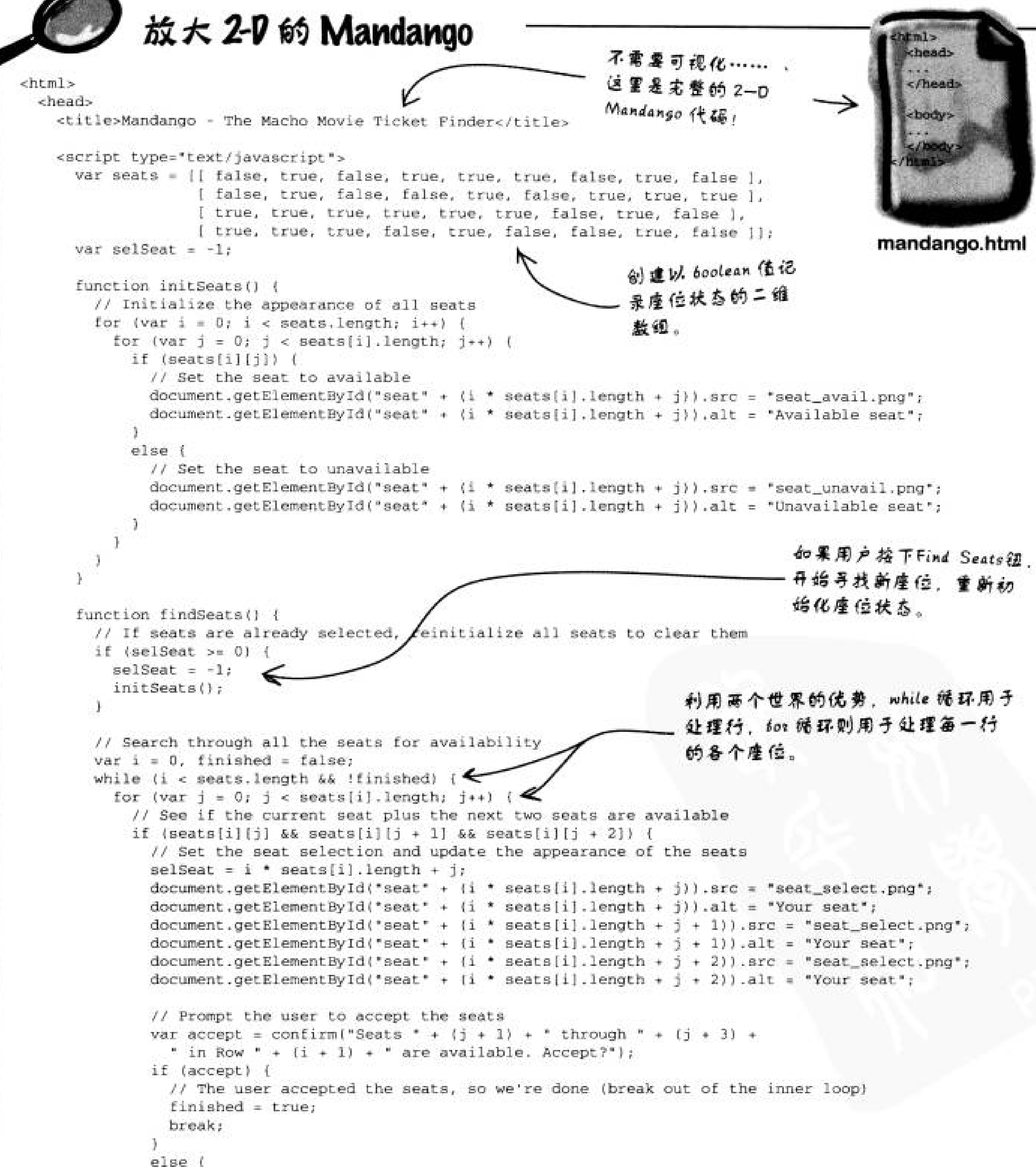




动动脑

当 Mandango 改成运用到整个电影院的座位数据时，二维数组将带来什么不同的影响？你又该如何把脚本可视化？

放大 2-D 的 Mandango



```
<html>
  <head>
    <title>Mandango - The Macho Movie Ticket Finder</title>

    <script type="text/javascript">
      var seats = [[ false, true, false, true, true, true, false, true, false ],
                   [ false, true, false, false, true, false, true, true, true ],
                   [ true, true, true, true, true, true, false, true, false ],
                   [ true, true, true, false, true, false, false, true, false ]];
      var selSeat = -1;

      function initSeats() {
        // Initialize the appearance of all seats
        for (var i = 0; i < seats.length; i++) {
          for (var j = 0; j < seats[i].length; j++) {
            if (seats[i][j]) {
              // Set the seat to available
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
            }
            else {
              // Set the seat to unavailable
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
            }
          }
        }
      }

      function findSeats() {
        // If seats are already selected, reinitialize all seats to clear them
        if (selSeat >= 0) {
          selSeat = -1;
          initSeats();
        }

        // Search through all the seats for availability
        var i = 0, finished = false;
        while (i < seats.length && !finished) {
          for (var j = 0; j < seats[i].length; j++) {
            // See if the current seat plus the next two seats are available
            if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
              // Set the seat selection and update the appearance of the seats
              selSeat = i * seats[i].length + j;
              document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
              document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

              // Prompt the user to accept the seats
              var accept = confirm("Seats " + (j + 1) + " through " + (j + 3) +
                " in Row " + (i + 1) + " are available. Accept?");
              if (accept) {
                // The user accepted the seats, so we're done (break out of the inner loop)
                finished = true;
                break;
              }
              else {
```

```
          // The user rejected the seats, so clear the seat selection and keep looking
          selSeat = -1;
          document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
      }
    }

    // Increment the outer loop counter
    i++;
  }
}
  </script>
</head>

<body onload="initSeats();">
  <<div style="margin-top:25px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" /><br />
    <img id="seat9" src="" alt="" />
    <img id="seat10" src="" alt="" />
    <img id="seat11" src="" alt="" />
    <img id="seat12" src="" alt="" />
    <img id="seat13" src="" alt="" />
    <img id="seat14" src="" alt="" />
    <img id="seat15" src="" alt="" />
    <img id="seat16" src="" alt="" />
    <img id="seat17" src="" alt="" /><br />
    <img id="seat18" src="" alt="" />
    <img id="seat19" src="" alt="" />
    <img id="seat20" src="" alt="" />
    <img id="seat21" src="" alt="" />
    <img id="seat22" src="" alt="" />
    <img id="seat23" src="" alt="" />
    <img id="seat24" src="" alt="" />
    <img id="seat25" src="" alt="" />
    <img id="seat26" src="" alt="" /><br />
    <img id="seat27" src="" alt="" />
    <img id="seat28" src="" alt="" />
    <img id="seat29" src="" alt="" />
    <img id="seat30" src="" alt="" />
    <img id="seat31" src="" alt="" />
    <img id="seat32" src="" alt="" />
    <img id="seat33" src="" alt="" />
    <img id="seat34" src="" alt="" />
    <img id="seat35" src="" alt="" /><br />
    <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
  </div>
</body>
</html>
```

改变座椅图像与说明文字时，需要行与列的循环计数。

首次载入网页时，调用 *initSeats()* 函数。

电影院里共有4排，每排9个座位，需要36张图像……唉……

当用户按下Find Seats按钮后，调用 *findSeats()* 函数。

Fun 轻松

别被长达两页的代码吓到了。

这里使用相同的二维数组技巧，只不过放入Mandango的情境里，并加上HTML代码与图像（均可从*http://www.headfirstlabs.com/books/hfjs/*下载）。

整间电影院的座位都很有男人味

有了二维数组的运作，Seth 与 Jason 能把 Mandango 带入更高层次，支持全电影院范围内的座位搜索……而且专为壮汉们设计！大家伙们现在兴致可高昂了！

JavaScript 填字游戏

讲了这么多电影院划位的东西，你该不会想去“实地勘察”一番吧？在你出门看电影前，我们一起做一下头脑体操，尝试一下填字游戏吧！

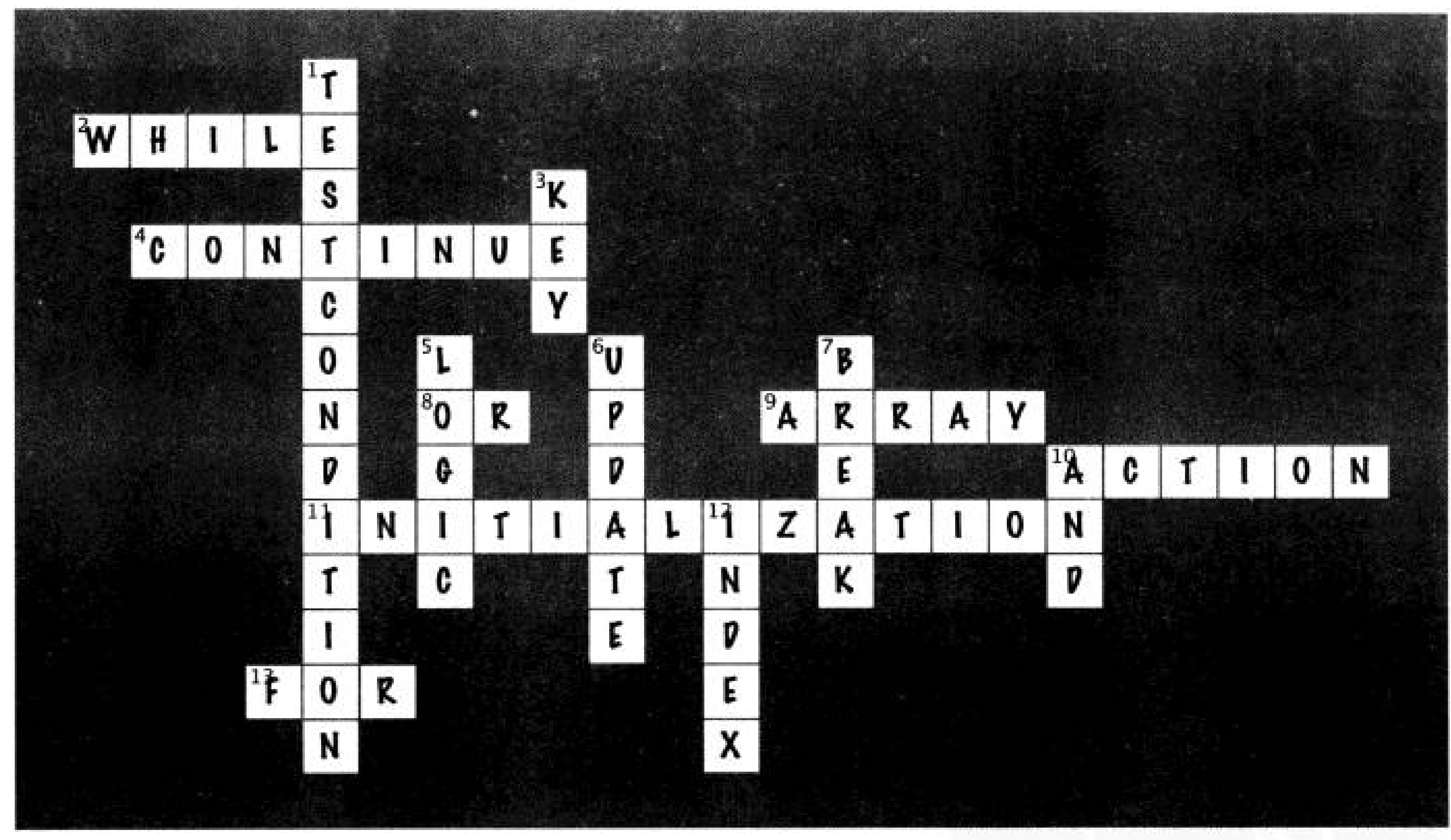
横向提示

2. 这种循环一直运行程序代码，直到满足测试条件（测试条件为 true）。
4. 使用这个语句可以跳出这一轮循环，但继续执行循环。
8. 如果“a为true”或“b为true”，则a ____ b为true；否则，a ____ b 为 false。
9. 这种数据类型能让你存储多段数据于单一变量中。
10. 循环中包含重复运行的代码的部分。
11. 循环中让循环准备开始的部分。
13. 这种循环特别适合计数。

纵向提示

1. 循环中必须得出 boolean 值的部分。
3. 用________访问数组值。
5. boolean ________ 运算符操作 boolean 值，也返回 boolean 值。
6. 循环中负责改变任何循环控制机制的部分。
7. 如果你想立刻结束循环，就使用________语句。
10. 如果“a 为 true”且“b 为 true”，则 a ______ b 为 true；否则，a ____ b 为 false。
12. 以数字访问数组值，需要________。

JavaScript 填字游戏

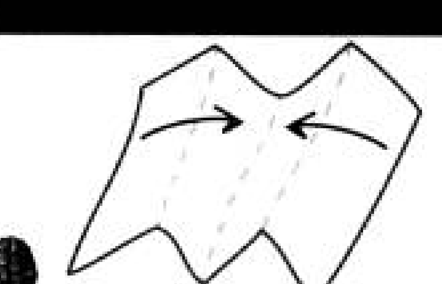


Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

循环和电影有哪些相同之处？

这是左右脑的秘密会谈！



有些电影的迂回情节非常出名，
观众跟不上电影行进的速度。
有些电影则使用
大量动作场面吸引观众。
无论如何，电影就只是电影。

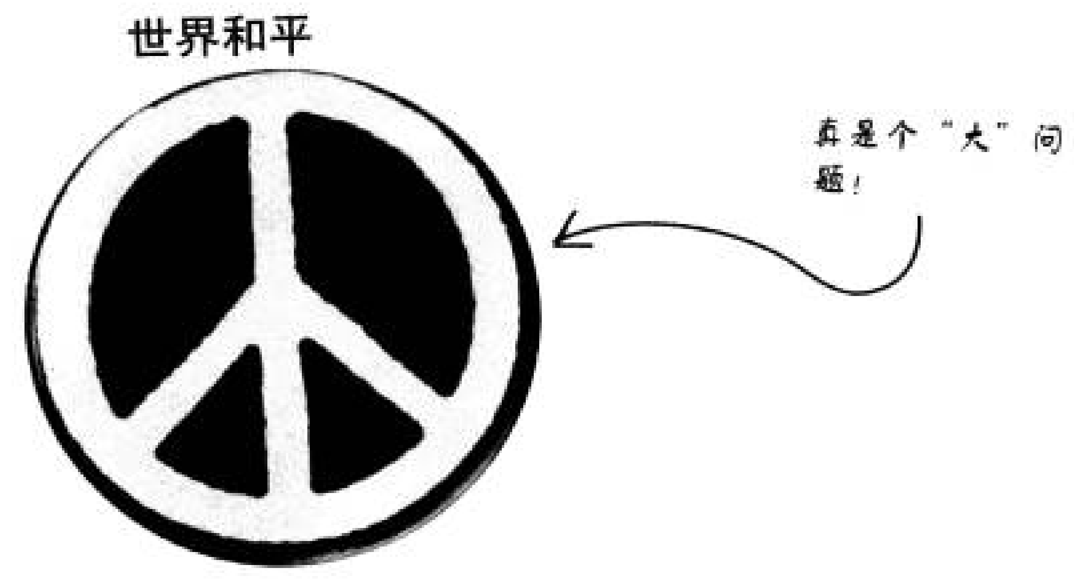
6 函数

简化、重复利用、回收再利用

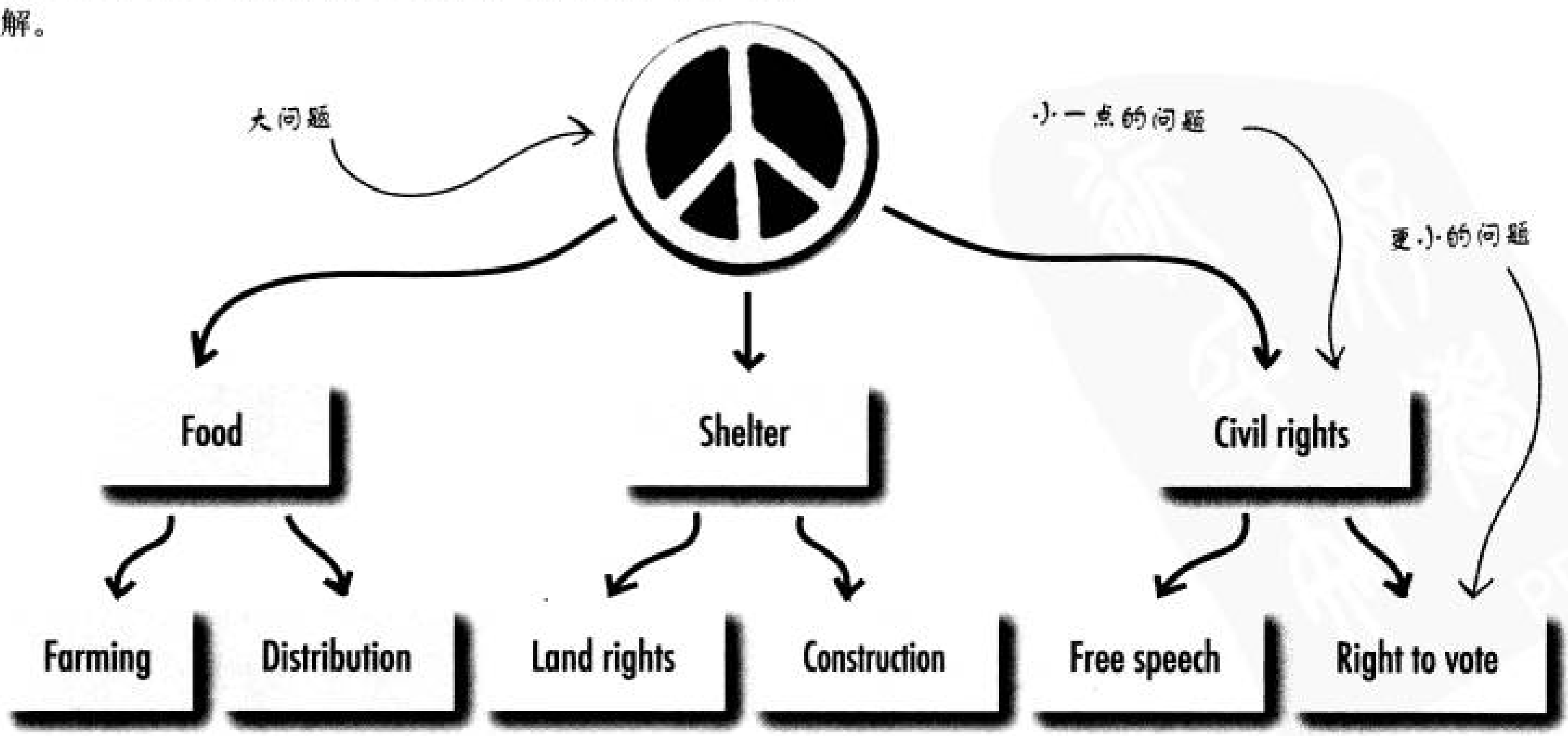
如果 JavaScript 里有整体环境移动的状况，将由函数领头。函数能让你把JavaScript代码变得更有效率，而且更能重复使用。函数是面向任务的，适用于组织代码，也是极佳的问题解决方案。听起来函数的履历表不错嘛！事实上，只有最简单的脚本才与函数重组代码的优势无关。虽然很难计算每个函数的二氧化碳排放量，但它们总能尽量帮助脚本更环保一点。

问题之母

追根究底，网络脚本编程就是为了解决问题。不管问题有多大，只要思虑够周全、计划够详尽，总会找到解决方案。但如果**问题真的很大很大**呢？



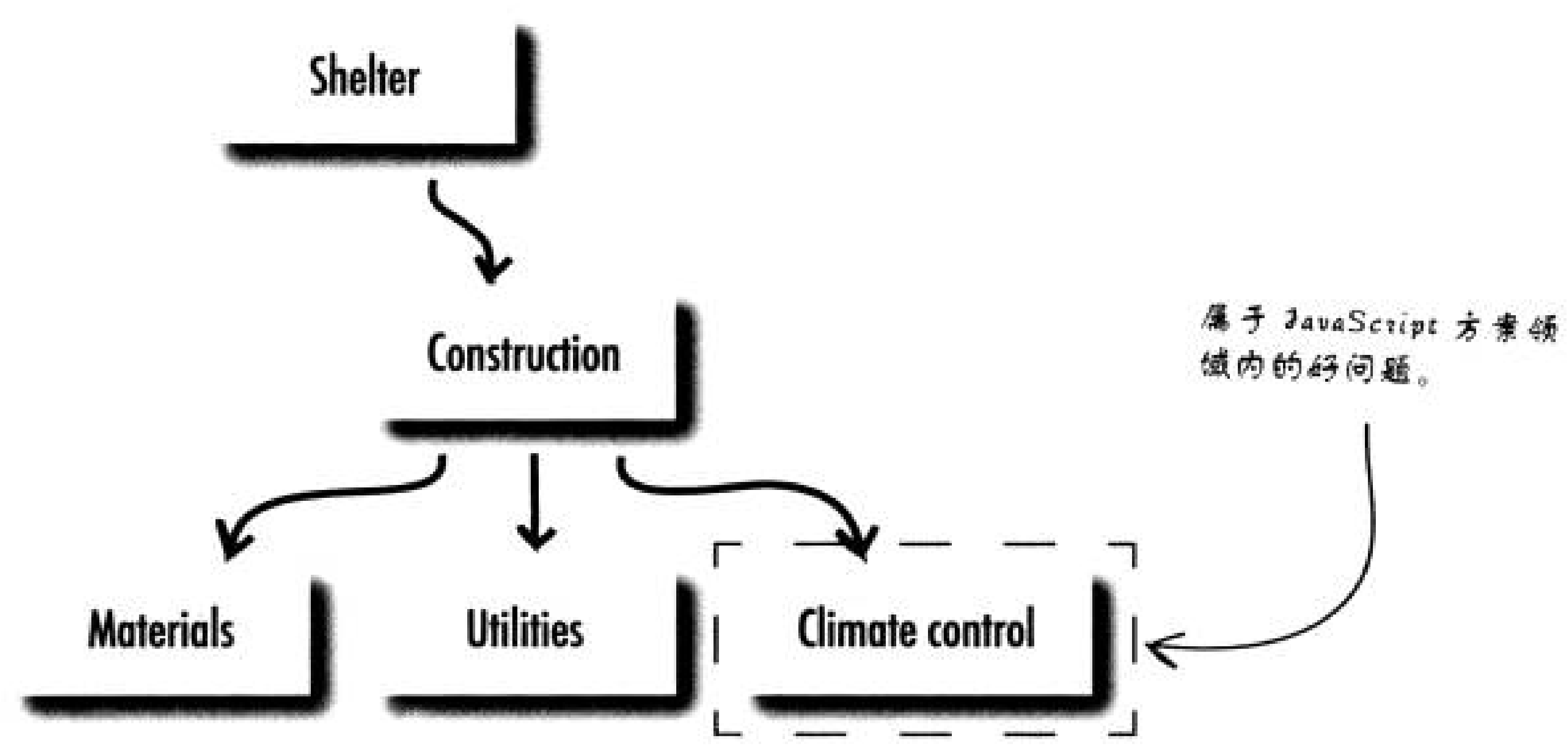
解决“大”问题的诀窍，就在于将其分解成较小、较易管理的问题。如果分解后的问题还是太庞大，再进一步分解。



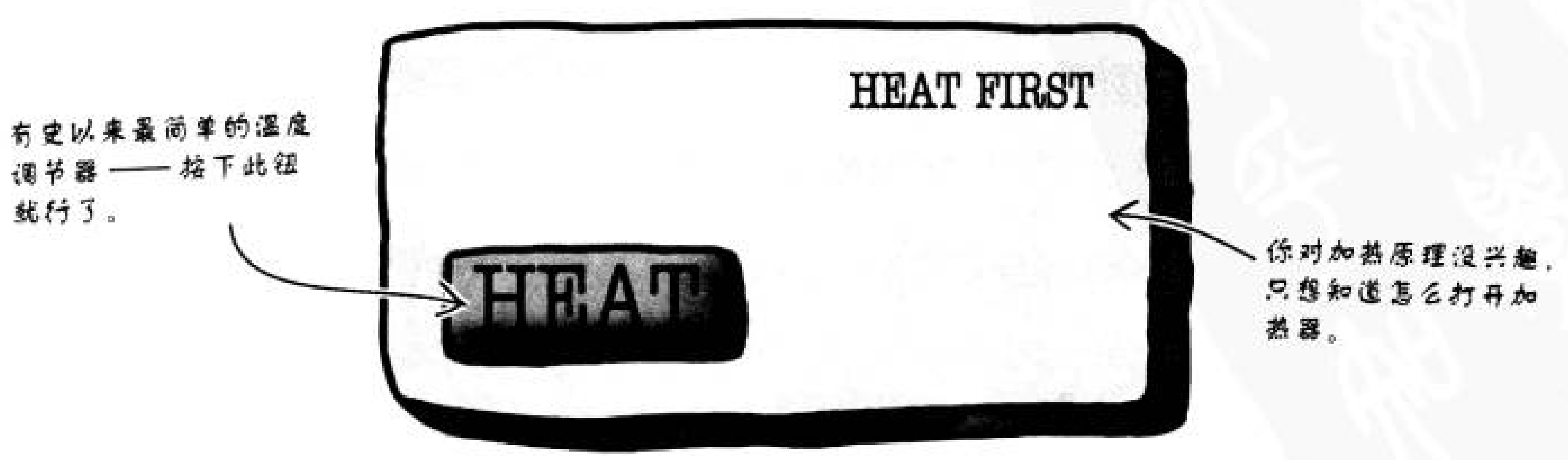
继续图上的流程……再继续……再继续……

见树而后知林

继续把世界和平的大问分解成较小的问题，最后终能小到 JavaScript 可以处理的程度。



设想一个等同于气候控制议题的 JavaScript，其中将包含自动调节温度的脚本设计（用于控制一个环境的温度）—— 最基本的温度调节器，只有个简单的加热钮（HEAT）。

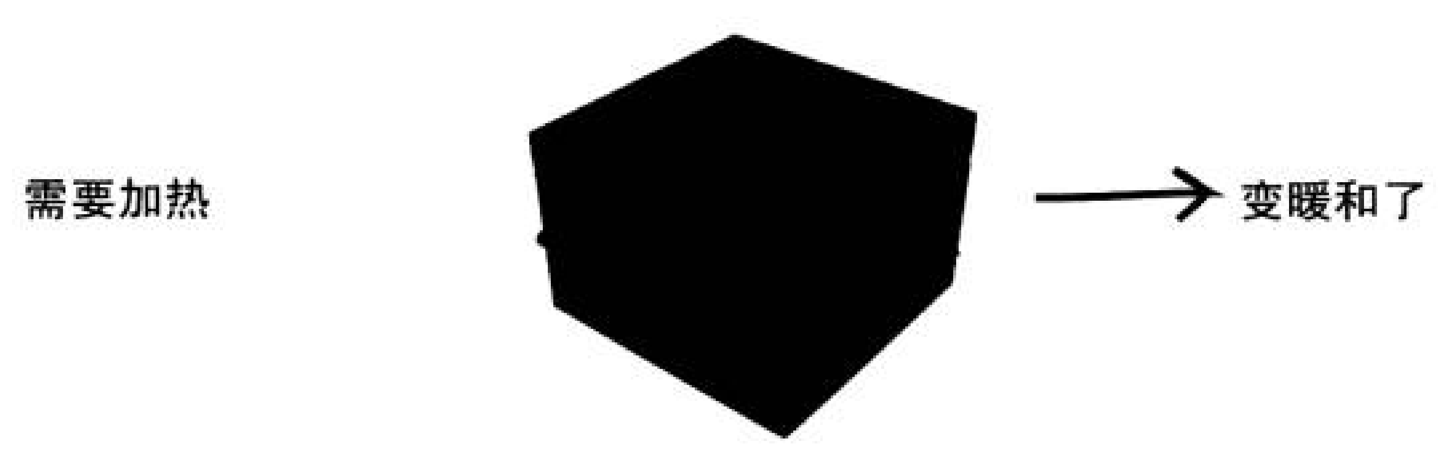


请注意，温度调节器完全没有揭露加热过程或原理。按下“HEAT”按钮，自然就有暖气了。气候控制问题解决了！

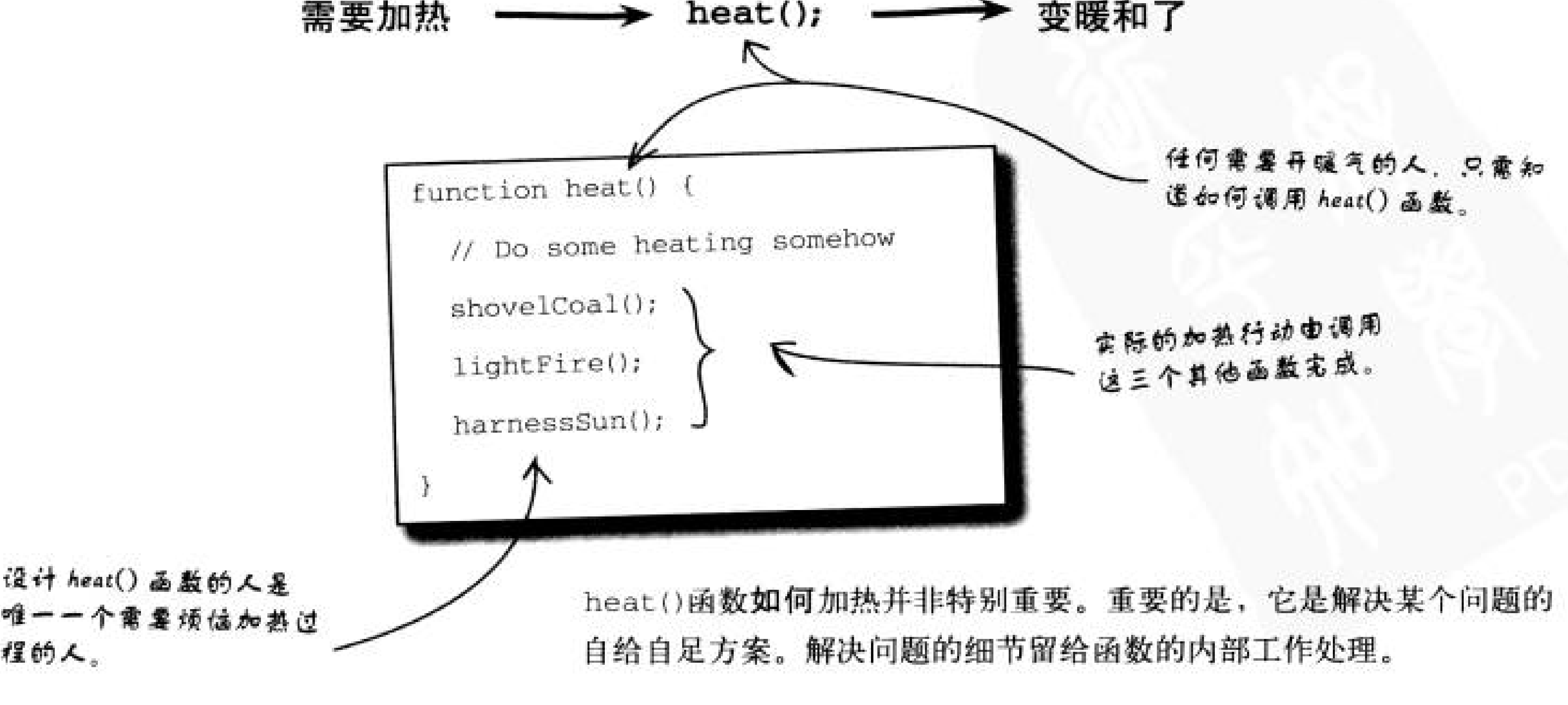
函数，问题解决者

温度调节器上的“HEAT”按钮，等同于JavaScript的函数（function）。原理其实与现实的温度调节器很相似——有人需要开暖气，函数就提供这项服务。加热的过程都在函数内部处理，调用函数的代码并不在意如何加热。你可以把函数想成“黑盒子”——信息可以流入流出，但盒子内的事情都由盒子负责，因此盒子外的代码完全没有相关。

函数转变大问题成为小问题。



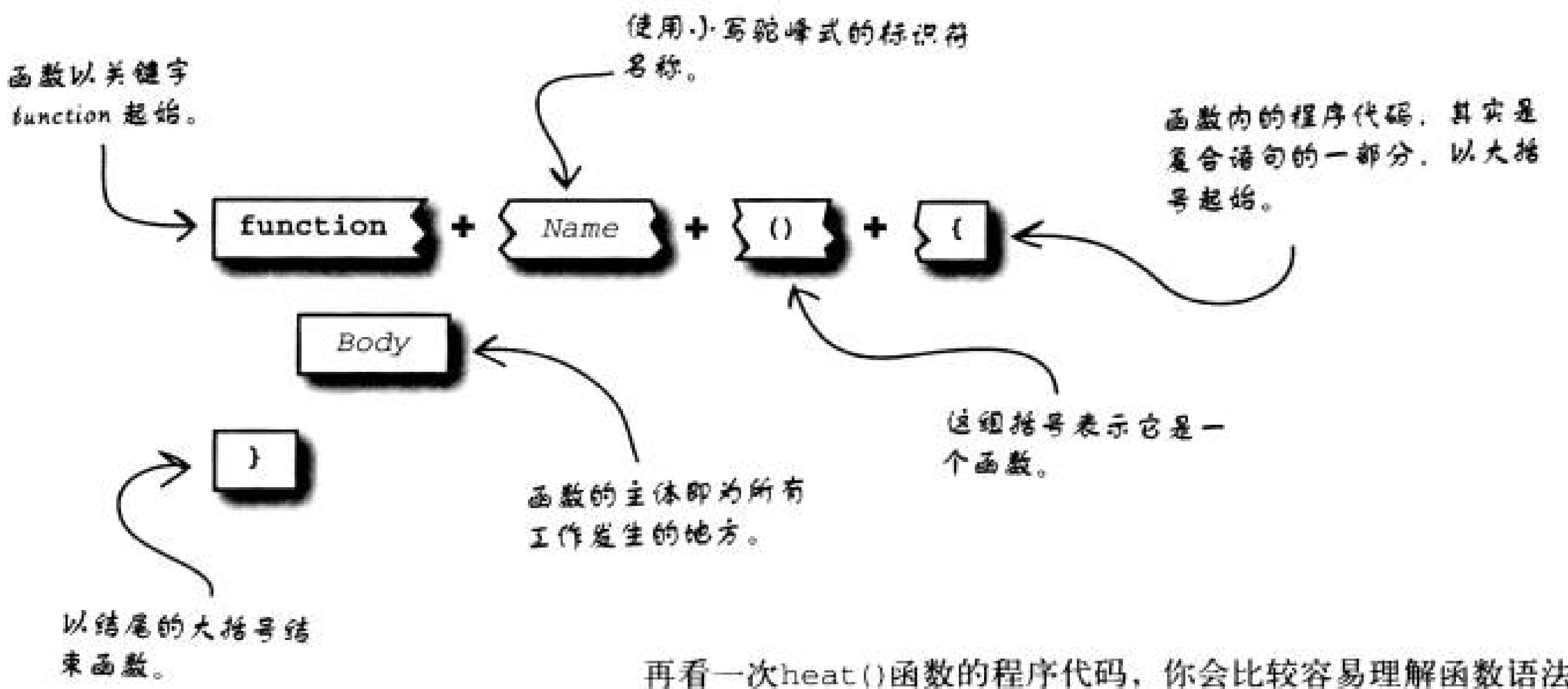
“HEAT按钮”转译为JavaScript代码后，牵涉到调用heat()函数……



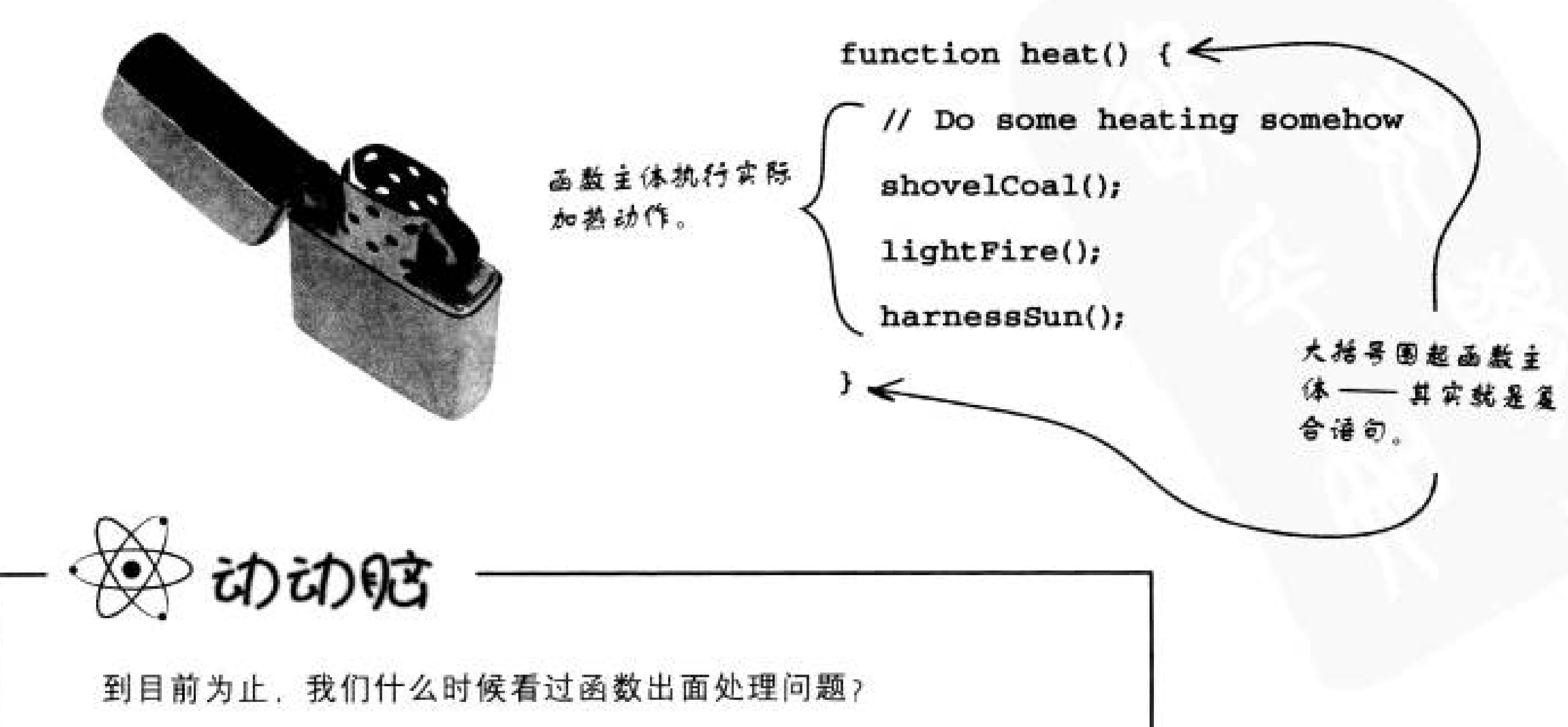
heat()函数**如何**加热并非特别重要。重要的是，它是解决某个问题的自给自足方案。解决问题的细节留给函数的内部工作处理。

函数的中枢

选择创建函数的一瞬间，你就成为一名问题解决者。创建函数需使用一致语法，能联结函数名称与实际运行的代码。下述即为最基本的 JavaScript 函数语法：



再看一次heat()函数的程序代码，你会比较容易理解函数语法：




```
function heat() {
    // Do some heating somehow
    shovelCoal();
    lightFire();
    harnessSun();
}
```

动动脑

到目前为止，我们什么时候看过函数出面处理问题？

你已经遇过的函数

Mandango，壮汉版座位搜索程序（*http://www.headfirstlabs.com/books/hfjs/*上有全部文件），翻开它的脚本，不用几行就能看到一个解决问题的函数的例子——本例为初始化电影座位的数据。以下即为Mandango问题的分解图：

Macho seat search ← 大问题

Initialize seats　Find seats ← 较小的问题

座位初始化的子问题，小到能以一个函数解决，即为`initSeats()`：

```
function initSeats() {
  // Initialize the appearance of all seats
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Set the seat to available
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
      }
      else {
        // Set the seat to unavailable
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
      }
    }
  }
}
```

`initSeats()`函数是Mandango网页的一部分。此函数需与`onload`事件处理器绑在一起，才会被调用。这也使得函数会在载入网页时运行。

initSeat()不是Mandango里唯一的问题解决者。

```
<body
onload="initSeats();">
    <div style="height:25px"></div>
    <div style="text-align:center">
      <img id="seat0" src="" alt="" />
      ...
      <img id="seat35" src="" alt="" /><br />
      <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
    </div>
  </body>
</html>
```

没有蠢问题

问：再提醒我一次，函数的命名惯例是什么啊？

答：小写驼峰式（lowerCamelCase）为JavaScript的标识符命名时的惯例——第一个词均小写，但后续字词则采取首字母大写。所以，评价电影的函数可能叫做`rateMovie()`，而“处理”坚持在看电影时讲电话的难缠的客人的函数可能叫做`removeInappropriateGuy()`。

问：函数总是把大问题转变成小问题吗？

答：不见得。有些情况下，函数只是纯粹被当成编码的劳力分工。换句话说，一个大问题可能由许多函数共同解决。此时，程序代码分解为函数的原因，在于协助区分工作，并为每个函数赋予单一目的。有点像我们每个人都有不同的职称，大家才能专注于特殊类型的任务。这类函数或许、也可能并未解决独特问题，但是函数的分工合作绝对改善了脚本的结构。

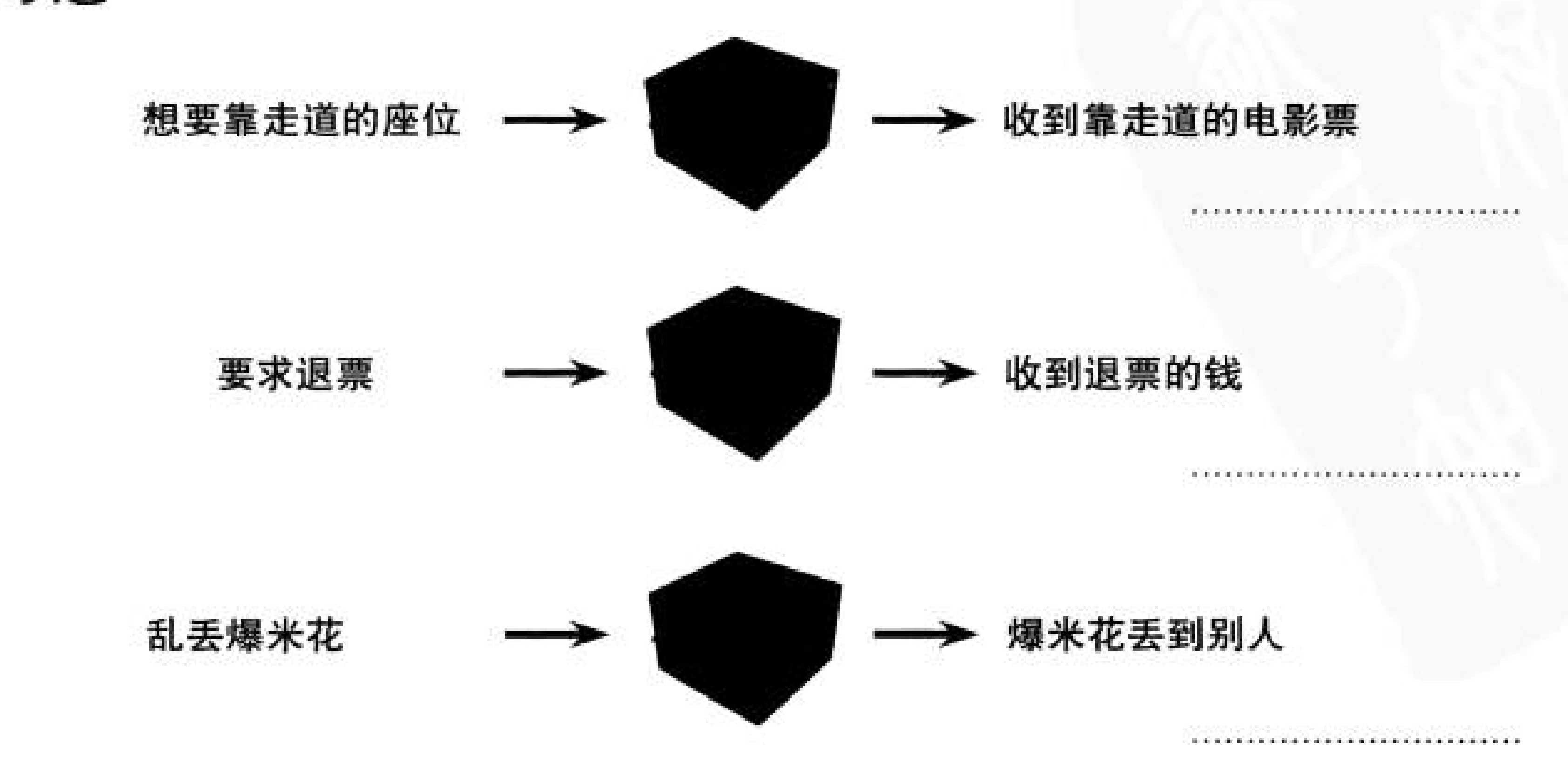
问：我该怎么知道哪些程序代码应该放到函数里？

答：很可惜，没有得知何时把一段程序代码放入函数最合理的魔法。但有些征兆可当成线索。一个是“复制某段程序代码”。复制代码很少是件好事，因为你必须在很多地方维护同一段代码。所以，复制代码是个做成函数的好征兆。另一个征兆则是某段代码逐渐变得笨重庞大，而且你可以把它分成数个逻辑片段。此时就是应用“分工合作”（division of labor）、考虑分解成数个函数的好时候。

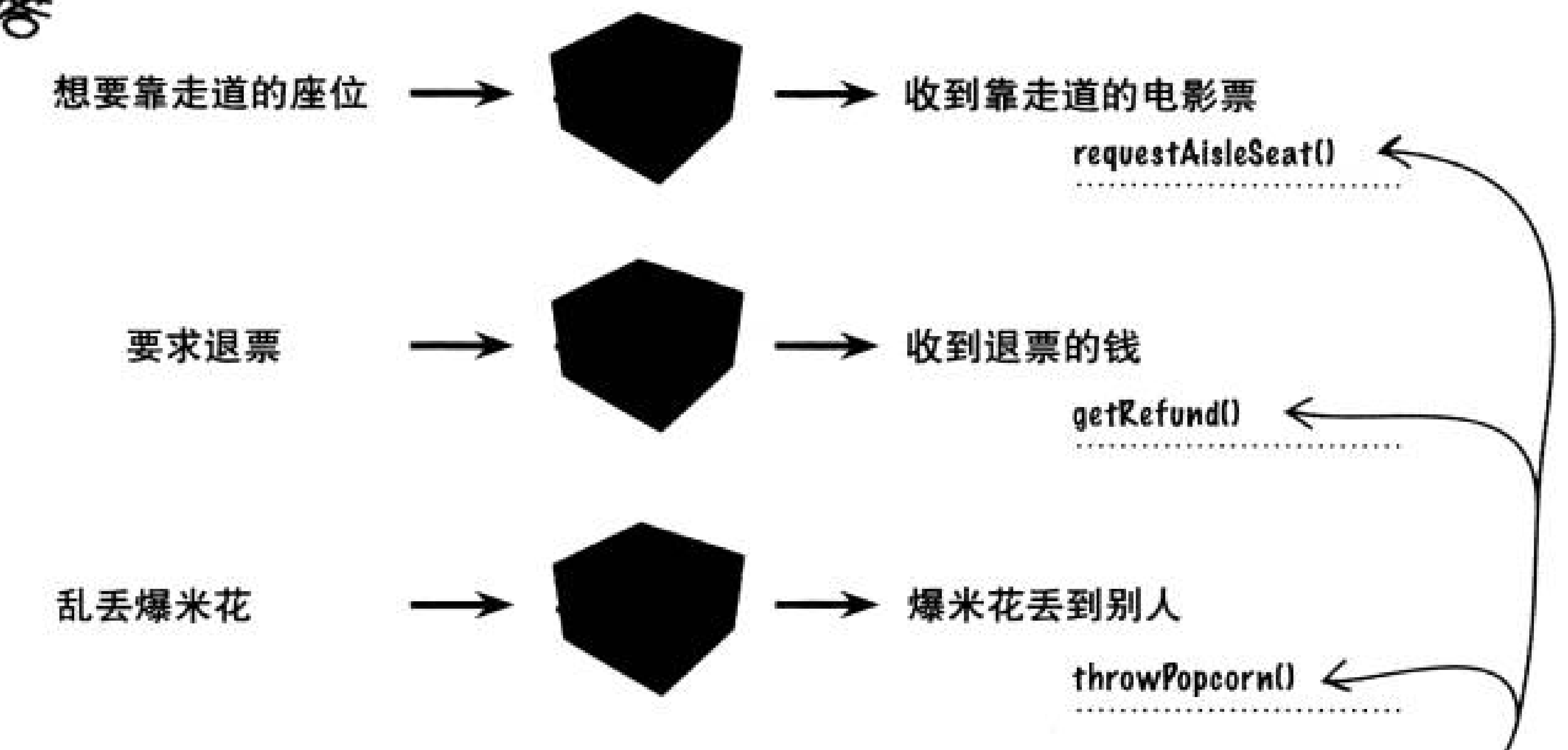
问：我记得看过接受自变量，然后返回数据的函数。难道我忘了什么吗？

答：没有没有，完全没有。确实存在着会接受数据、也会返回数据的函数。事实上，如果你坚持下去，各位将看到`heat()`变成这类函数。

在立刻表达函数的功能上，函数的名称非常重要。试着帮下列函数命名，请使用小写驼峰式。



在立刻表达函数的功能上，函数的名称非常重要。试着帮下列函数命名，请使用小写驼峰式。



还有很多函数名称适用于这个习题。这里示范了小写驼峰式的简洁名称。

我快热死了！谁快把暖气关掉吧！或者是全球变暖的威力所致？

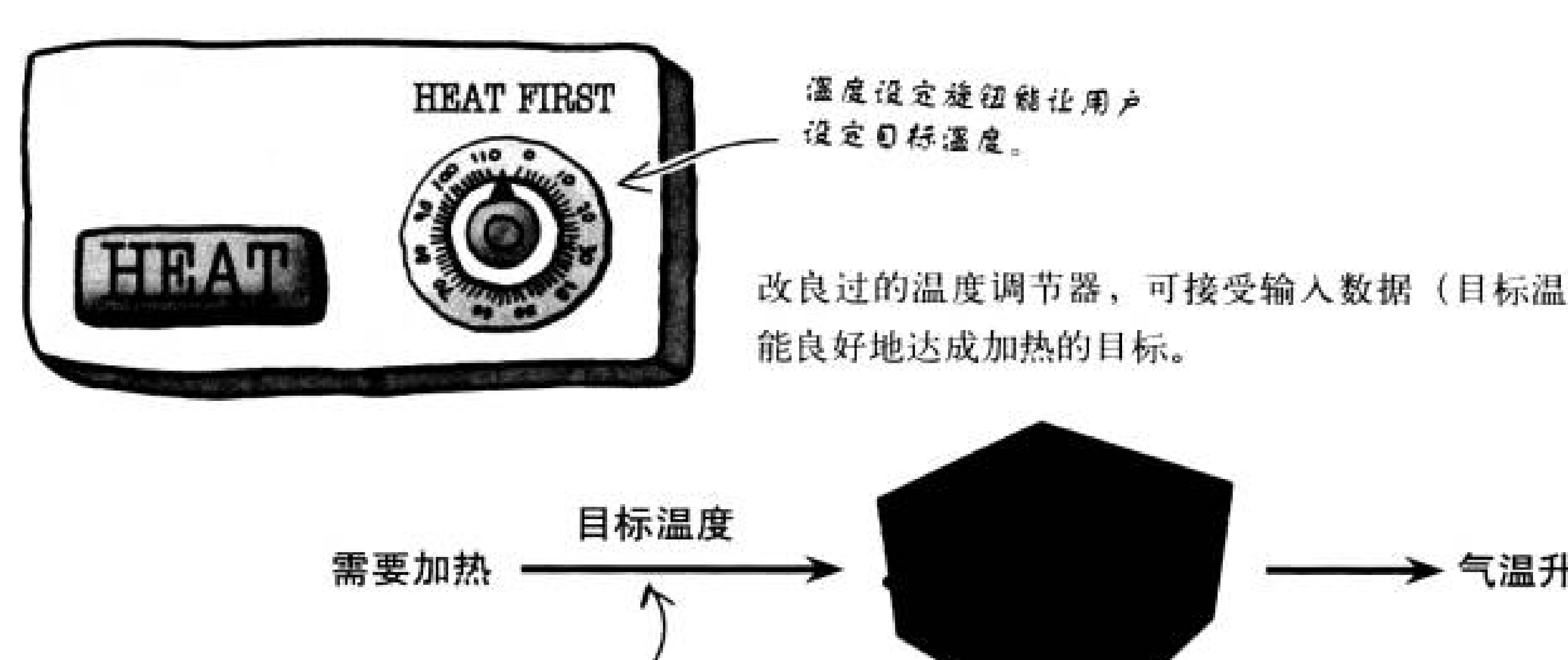
这温度真舒服。

太热了！受不了！

同时，透过控制气候而促进世界和平的努力遇到了一点障碍。看来“HEAT”按钮运作得太好了，问题也可能在于`heat()`函数需要更多数据。无论如何，都有等待修正的问题。

建立数据更多的温度调节器

回到温度调节器……它不知道何时该停止加热，因为我们没有设定目标温度。总之就是信息不足，无法有效地解决问题，也就是说有人按下了“HEAT”按钮后，温度调节器就一直开着……永远开着！



为改良过的heat() 函数设计程序代码。新版函数可接受目标温度，并只在温度低于目标温度设定时加热。提示：调用假设的 getTemp() 函数以取得目前的温度。

我们放了这一行，帮助你开始。

```
function heat(targetTemp){
```

为改良过的 heat() 函数设计程序代码。新版函数可接受目标温度，并只在温度低于目标温度设定时加热。提示：调用假设的getTemp() 函数以取得目前的温度。

```
function heat(targetTemp){
  while (getTemp() < targetTemp) {
    // Do some heating somehow
    shovelCoal();
    lightFire();
    harnessSun();
  }
}
```

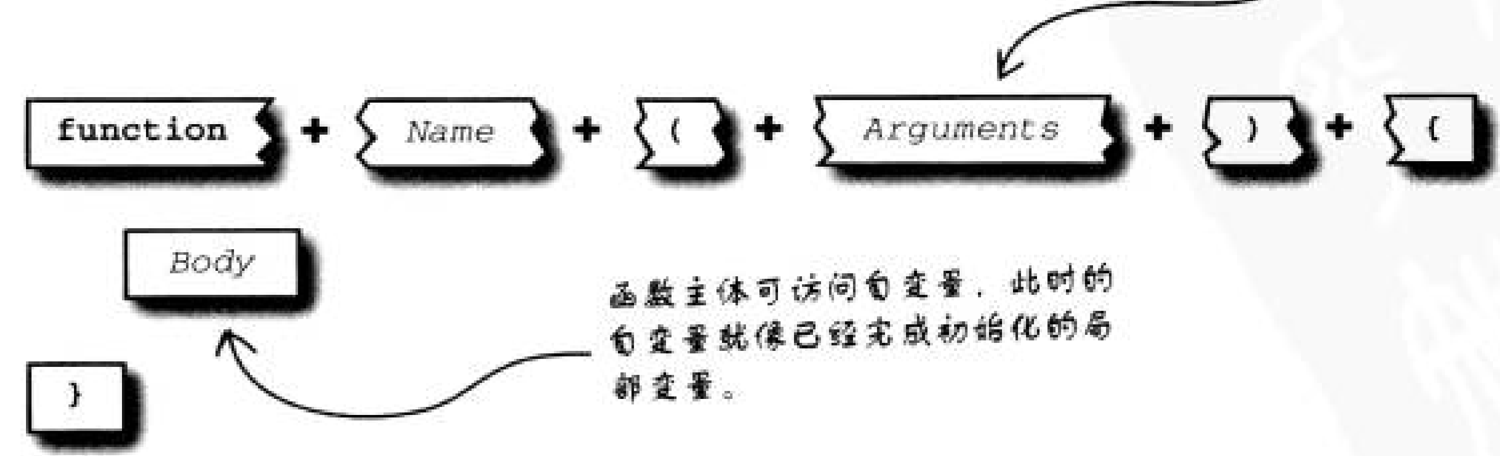
目标温度传给函数，作为函数的自变量。

目标温度是 while 循环测试条件的一部分。

现在，函数只在目前的温度低于目标温度时，才会加热。

传递信息给函数

数据透过函数的**自变量**（argument，很像输入数据）传入JavaScript函数。再度查看函数语法，你看到函数创建时，自变量放在括号中的方式了吗？



传给函数的自变量数量并无实际限制，不过试着把自变量数量维持在两、三个以下还是比较合理啦！你可用自变量传入任何数据给函数，例如：常量（Math、PI）、变量（temp）或字面量（72）。

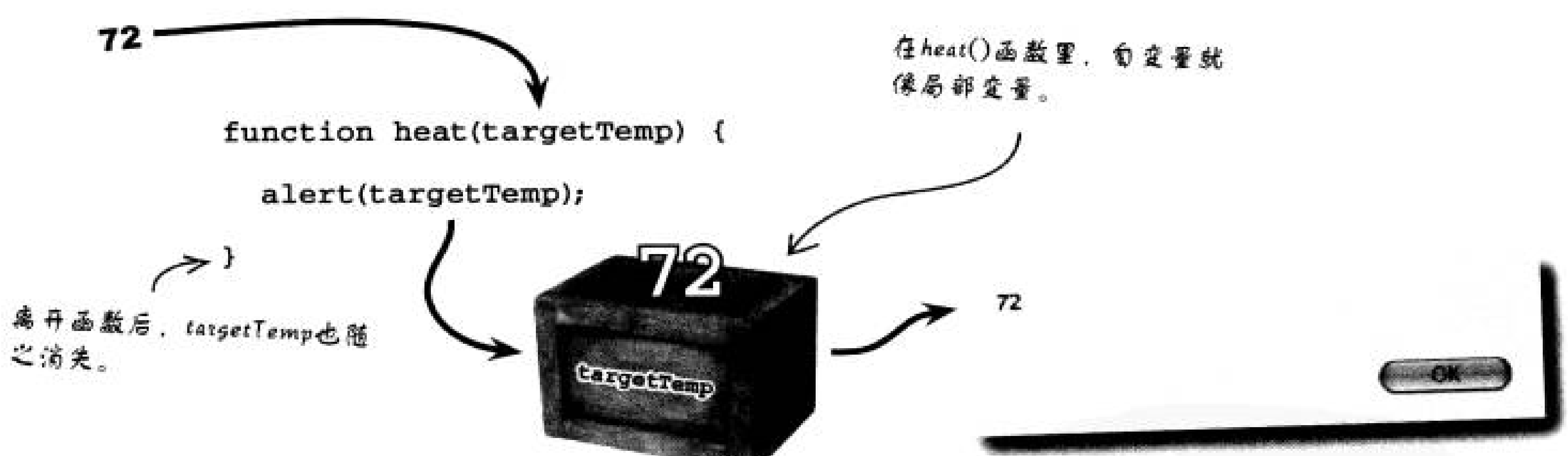
函数自变量作为数据

当数据作为自变量而被传入函数，它的行为就像函数内已初始化的局部自变量。下面的范例中，heat()函数获得的目标温度值，就是以自变量的形式传入函数：

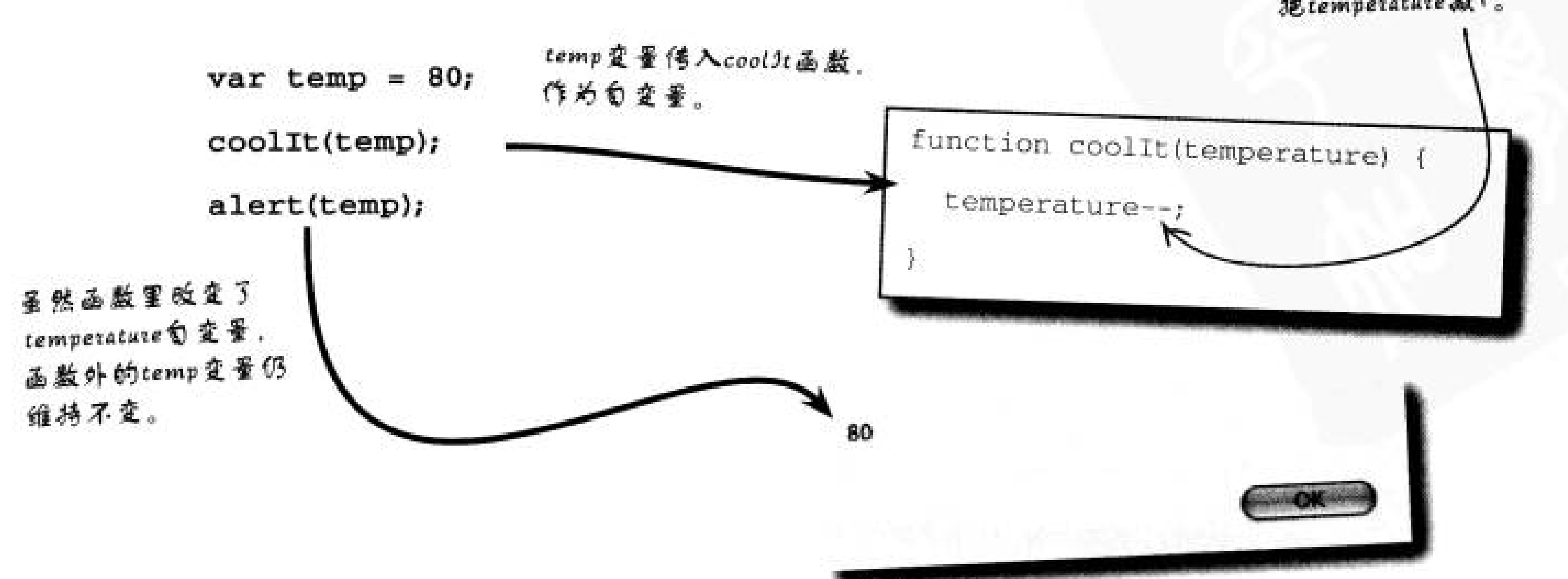
```
heat(72);
```

目标温度的数值一五一十地传给函数。

在heat()函数里，targetTemp自变量的访问方式，一如它是个初始化为72的局部变量。下面先以呈现自变量值的alert框取代平常的heat()函数代码。



虽然函数自变量的行为很像函数内的局部变量，但在函数**内**改变自变量，却不会影响函数**外**的任何事物。但这项规则不适用于作为自变量传入的对象——第 9 章和第 10 章将详细讨论对象。



函数去除重复代码

除了拆解问题让问题的解决更为容易，透过任务的统合，函数也是种极佳的削减重复代码的方式。统合任务（generalized task）可把出现在多个地方的相似程序代码予以消除。即使这类程序代码不见得一模一样，但多半总能统合为一致、可置于函数中的程序代码。然后，你就能以调用函数取代复制代码。

以下有三段不同的代码，都牵涉到相似任务，所以应该能统合为**可再利用**的单一任务：

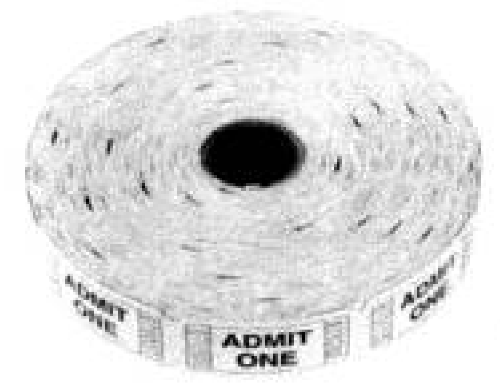
折扣的计算不需重复。

```
// Matinee ticket is 10% less
matineeTicket = adultTicket * (1 - 0.10);
```

```
// Senior ticket is 15% less
seniorTicket = adultTicket * (1 - 0.15);
```

```
// Child ticket is 20% less
childTicket = adultTicket * (1 - 0.20);
```

上述各项任务分别计算了三张电影票的折扣价。但这些任务能统合为一项任务——根据折扣比例计算电影票价。



函数也能返回数据。

```
function discountPrice(price, percentage) {
  return (price * (1 - (percentage / 100)));
}
```

有了通用性的票价折扣函数，其他三段代码就能改写得较有效率了：

```
// Matinee ticket is 10% less
matineeTicket = discountPrice(adultTicket, 10);
```

```
// Senior ticket is 15% less
seniorTicket = discountPrice(adultTicket, 15);
```

```
// Child ticket is 20% less
childTicket = discountPrice(adultTicket, 20);
```

与效率天人合一

下面是原始的"Mandango —— 壮汉版电影院划位程序"的findSeats()函数。使用你刚认识的效率知识，圈出相似的代码，这些代码可重新设计为通用、可重复利用的函数。

```
function findSeats() {
  // If seats are already selected, reinitialize all seats to clear them
  if (selSeat >= 0) {
    selSeat = -1;
    initSeats();
  }

  // Search through all the seats for availability
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // See if the current seat plus the next two seats are available
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Set the seat selection and update the appearance of the seats
        selSeat = i * seats[i].length + j;
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

        // Prompt the user to accept the seats
        var accept = confirm("Seats " + (j + 1) + " through " + (j + 3) +
          " in Row " + (i + 1) + " are available. Accept?");
        if (accept) {
          // The user accepted the seats, so we're done (break out of the inner loop)
          finished = true;
          break;
        }
        else {
          // The user rejected the seats, so clear the seat selection and keep looking
          selSeat = -1;
          document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
      }
    }

    // Increment the outer loop counter
    i++;
  }
}
```

与效率天人合一解答

下面是原始的"Mandango——壮汉版电影院划位程序"的findSeats()函数。使用你刚认识的效率知识，圈出相似的代码，这些代码可重新设计为通用、可重复利用的函数。

```
function findSeats() {
  // If seats are already selected, reinitialize all seats to clear them
  if (selSeat >= 0) {
    selSeat = -1;
    initSeats();
  }

  // Search through all the seats for availability
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // See if the current seat plus the next two seats are available
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Set the seat selection and update the appearance of the seats
        selSeat = i * seats[i].length + j;
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Your seat";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_select.png";
        document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Your seat";

        // Prompt the user to accept the seats
        var accept = confirm("Seats" + (j + 1) + "through" + (j + 3) +
          "in Row" + (i + 1) + "are available. Accept?");
        if (accept) {
          // The user accepted the seats, so we're done (break out of the inner loop)
          finished = true;
          break;
        }
        else {
          // The user rejected the seats, so clear the seat selection and keep looking
          selSeat = -1;
          document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 1)).alt = "Available seat";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).src = "seat_avail.png";
          document.getElementById("seat" + (i * seats[i].length + j + 2)).alt = "Available seat";
        }
      }
    }

    // Increment the outer loop counter
    i++;
  }
}
```

因为这6段代码执行相同的一般性任务，遂可转换成单一函数。

length特性仍然用于取得子数组的元素数量。

重复的代码。我们可以抽取几个属性……

创建设定座位的函数

Mandango 的壮汉们现在察觉了"效率"这回事，他们正忙着加入可让座位设定更有效率的函数（可至 *http://www.headfirstlabs.com/books/hfjs/*下载代码）。不过，为了设计`setSeat()`函数，他们真的需要找出所需自变量。借由查看各段重复代码中的信息，你可以分离出必要自变量。仔细观察 `findSeats()` 函数的重复部分，得出下列自变量：

座位编号（Seat Number）

为每个座位设定的编号。此非数组索引，只是从左到右、从上到下计算时的座位编号，从 0 开始。

由重复的代码中抽取这些属性，而后加入 findSeats() 函数。

状态（Status）

座位的状态为：空位、已售位、已选择。用于判定呈现的座椅图像

说明（Description）

座位状态的说明，例如"Available seat"（空位）、"Unavailableseat"（已售位）、"Your seat"（已选择）。用于设定座椅图像的 `alt` 属性的文本。

磨笔上阵

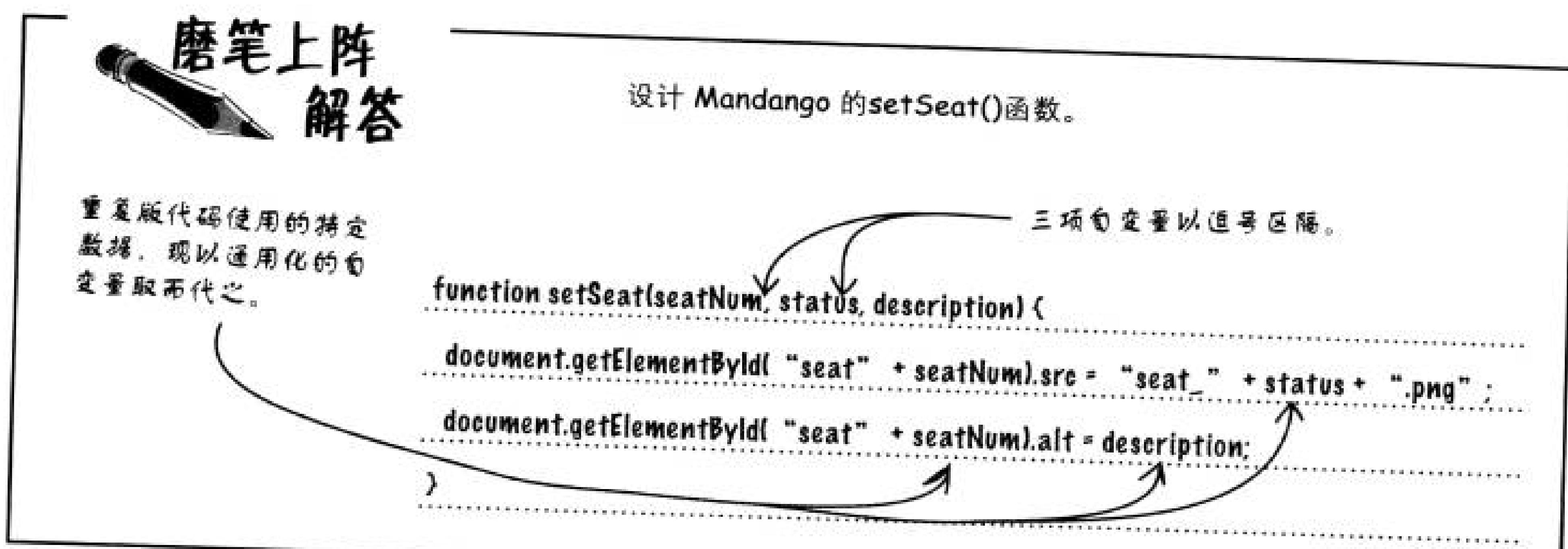
设计 Mandango 的setSeat()函数。

....

....

....

....



使用了函数后，更苗条、更清爽的 Mandango

相似、重复的代码拆解成`setSeat()`函数后，也大幅简化了`findSeats()`函数。共有6次对`setSeat()`函数的调用，以程序代码再利用的角度而言，实为长足进展。

座位编号、状态和说明文字，于各次调用函数时再被传给 *setSeat()*。

新的 *setSeat()* 函数被调用了6次。

```
function findSeats() {
  ...
  // Search through all the seats for availability
  var i = 0, finished = false;
  while (i < seats.length && !finished) {
    for (var j = 0; j < seats[i].length; j++) {
      // See if the current seat plus the next two seats are available
      if (seats[i][j] && seats[i][j + 1] && seats[i][j + 2]) {
        // Set the seat selection and update the appearance of the seats
        selSeat = i * seats[i].length + j;
        setSeat(i * seats[i].length + j, "select", "Your seat");
        setSeat(i * seats[i].length + j + 1, "select", "Your seat");
        setSeat(i * seats[i].length + j + 2, "select", "Your seat");

        // Prompt the user to accept the seats
        var accept = confirm("Seats" + (j + 1) + "through" + (j + 3) +
          "in Row" + (i + 1) + "are available. Accept?");
        if (accept) {
          // The user accepted the seats, so were done (break out of the inner loop)
          finished = true;
          break;
        }
        else {
          // The user rejected the seats, so clear the seat selection and keep looking
          selSeat = -1;
          setSeat(i * seats[i].length + j, "avail", "Available seat");
          setSeat(i * seats[i].length + j + 1, "avail", "Available seat");
          setSeat(i * seats[i].length + j + 2, "avail", "Available seat");
        }
      }
    }

    // Increment the outer loop counter
    i++;
  }
}
```

setSeat()函数让 Mandango 更好

但setSeat()函数不只对 findSeats()有利，它也协助 initSeats() 更有效率，因为这个函数也有相似的座位设定代码。

```
function initSeats() {
  // Initialize the appearance of all seats
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Set the seat to available
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_avail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Available seat";
      }
      else {
        // Set the seat to unavailable
        document.getElementById("seat" + (i * seats[i].length + j)).src = "seat_unavail.png";
        document.getElementById("seat" + (i * seats[i].length + j)).alt = "Unavailable seat";
      }
    }
  }
}
```

两行复杂的代码转变为相对简单的函数调用。

```
function initSeats() {
  // Initialize the appearance of all seats
  for (var i = 0; i < seats.length; i++) {
    for (var j = 0; j < seats[i].length; j++) {
      if (seats[i][j]) {
        // Set the seat to available
        setSeat(i * seats[i].length + j, "avail", "Available seat");
      }
      else {
        // Set the seat to unavailable
        setSeat(i * seats[i].length + j, "unavail", "Unavailable seat");
      }
    }
  }
}
```

借由统合设定座位的任务，setSeat() 函数就算在完全不同的环境中，也一样运作良好。

你看，一个还算简单、只有两行的函数，现在已于 Mandango 脚本中使用了8次。函数的使用不只**简化**了脚本，还让脚本更好维护，因为在你需要改变座位设定方式时，你只需改变一段setSeat()代码，而不需更动8块出现在不同地方的代码。只要并非必需情况，神智清醒的JavaScript设计者都不会想更改多处的代码。可维护性……是件好事。

复习要点

- 函数让我们转换**大问题至小问题**，小问题较容易解决。
- 函数提供了分离脚本任务的机制，而后再以**可再利用**的程序代码块予以统合。
- 函数是种**削减重复代码**的极佳方式，因为函数里的程序代码能依你的意愿尽情重复使用。
- **自变量**让你传递数据至函数里，作为既定任务的输入。

没有蠢问题

问：传给函数的自变量数量是否有限制呢？

答：没有，但也算有。我回答“没有”，因为并无硬性规定可以传入函数的自变量数量，除非你把计算机内存的能耐也算进来。如果你传入的自变量数量多到连内存都暂存不了，我建议你休息一下，重新想想你在做啥，因为要有**非常多**自变量，才会对内存造成问题。较实际的限制则与良好设计有关，也就是保持自变量数量为可管理的数量，函数才不会复杂得乱七八糟。通常，最好别使用超过你能控制的自变量数量。

问：我已学到函数能把大问题变成小问题、在脚本里分工合作、削减重复的代码。它究竟扮演哪一个角色？

答：以上皆是。函数不只擅长一项工作，很多时候，最好的函数能同时达到多项目标。解决子问题、任务分工合作、削减重复代码，创建同时有这些功能的函数并非不可能。事实上，这三个是创建函数时的好目标。但如果一定要挑出特别着重的目标，“分工合作”通常是单一函数目标的最佳选择。如果每个函数都擅长一项独特工作，你的脚本将受益良多。

问：让我再问一次，函数该放在网页的标头区（header）还是主体区（body）？

答：函数应该放在标头区的<script>标签里，或放在导入网页的外部JavaScript文件里。

问：如果真想用函数改变自变量值，我该怎么做？

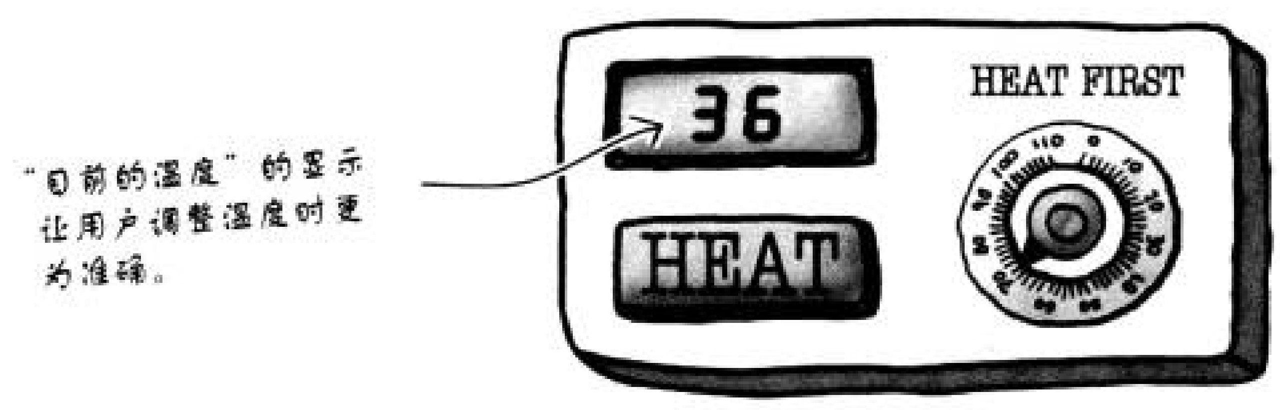
答：函数自变量不能直接改变，至少这类改变不会发生在函数外。所以，如果你想改变以自变量传入的数据，你需要返回函数改变过的值。继续往下看，你将理解返回值如何运作。

六月雪：函数的反馈

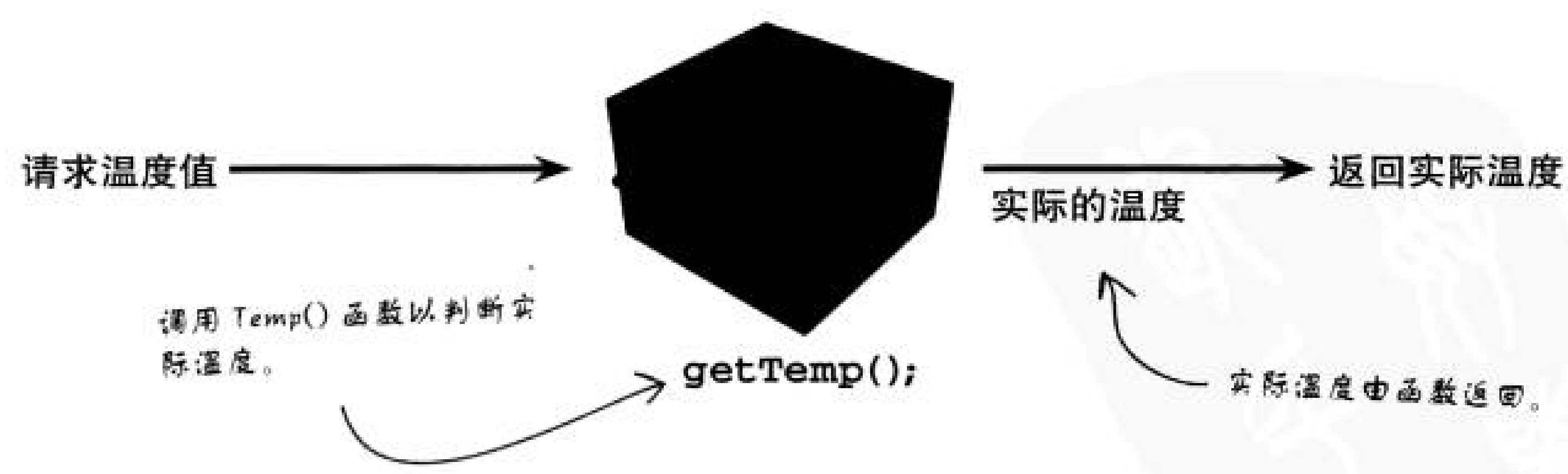
Mandango受惠于函数，向前迈进了一大步，但温度调节接口可没如此好运。显然JavaScript的温度调节器运作不顺，某些冻僵的用户开始怀念起以前的“HEAT”按钮，永不停止加热的“HEAT”按钮。

反馈的重要性

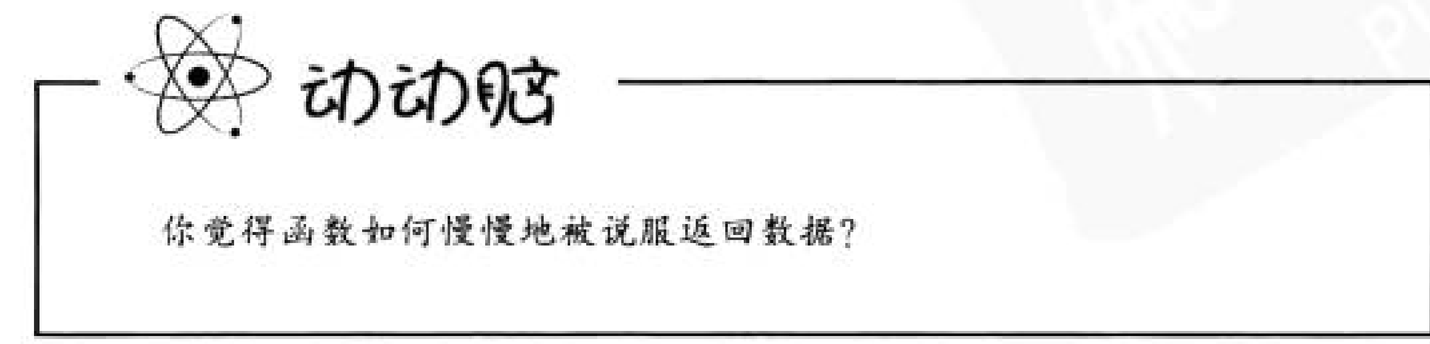
多亏了函数自变量，我们的调节器才能设定温度，但它不会汇报目前的温度。因为目前的温度是决定目标温度的基础，它也很重要。另外，不同温度调节器通常都汇报不同的温度，即使它们位于相同地方。追根究底，我们需要反馈机制……你需要知道目前的温度，才能设定有意义的目标温度。



温度调节器将定期显示目前的温度，作为协助判断理想温度的反馈。

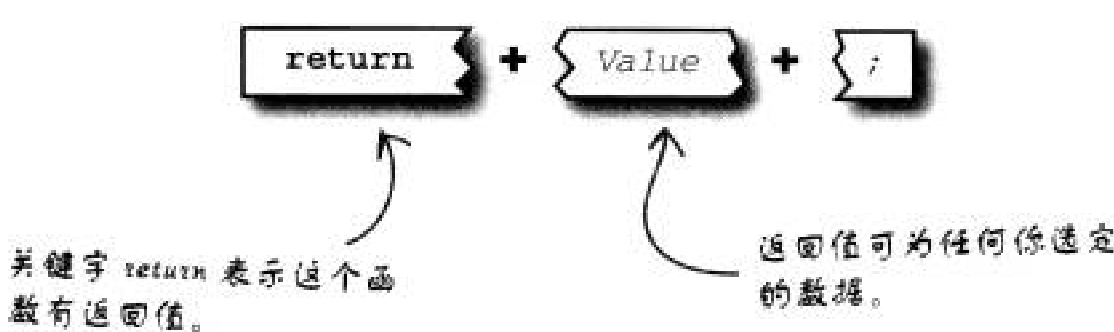


所以 JavaScript 函数真的需要返回信息给调用函数的代码。

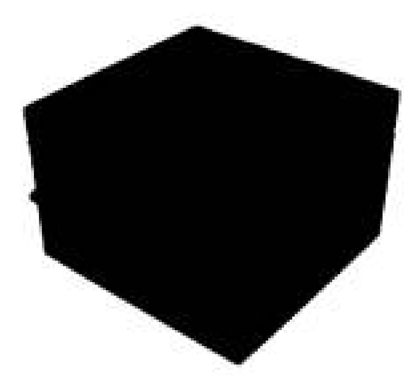


从函数返回数据

从函数返回信息牵涉到关键字return的使用，关键字后接欲返回的信息。然后这些数据即被返回给调用函数的代码。



返回值能让你从函数中返回一段数据。



return语句能放在函数的任何地方，只要记得，函数遇到return后将立刻结束。所以，return不只返回数据，同时也负责结束函数。例如，getTemp()函数即以返回实际温度而结束，实际温度为传感器（sensor）的读数（read）：

```
function getTemp() {
  // Read and convert the actual temperature
  var rawTemp = readSensor();
  var actualTemp = convertTemp(rawTemp);
  return actualTemp;
}
```

传感器数据的格式比较奇怪，需要转换为温度单位。

实际温度使用return语句，从函数返回。

如果你仔细回想，getTemp()函数已曾用于温度调节代码：

```
function heat(targetTemp) {
  while (getTemp() < targetTemp) {
    // Do some heating somehow
    ...
  }
}
```

getTemp()函数为heat()的while循环提供用于测试条件的值。

getTemp的返回值取代了getTemp()函数的调用，并成为while循环测试条件的一部分。

函数的返回值取代了函数的调用。

很多快乐的返回值

因为 `return` 语句立即结束函数，我们可用它控制函数流程，还能附带返回数据。不只如此，函数也经常用返回值表示运作成功。`heat()` 函数即示范了同时达成两种目标的机会：

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp)
    return false;
  while (getTemp() < targetTemp) {
    // Do some heating somehow
    ...
  }
  return true;
}
```

还记得变量 actualTemp 吗？它提供了 getTemp() 的返回值。

不需要任何加热，所以返回 false 并结束函数。

这段代码负责加热工作，它将影响温度，还有 getTemp() 的返回值。

结束加热，返回true，指出运作成功。

`heat()`函数的示范里可看到 boolean类型的返回值如何控制函数流程，以及如何指示函数的成功或失败。为了单纯的流程控制，你可以使用完全没有返回值的 `return` 语句，作为跳出函数的方式。下例的另一版 `heat()` 函数，即不需依赖指示函数成功或失败的返回值。

```
function heat(targetTemp) {
  if (getTemp() >= targetTemp)
    return;
  while (getTemp() < targetTemp) {
    // Do some heating somehow
    ...
  }
}
```

return 语句缩短了函数，因为不需要加热。

函数仍在该结束时结束，不需 return 语句的协助。

return 语句本身能用于结束函数。

Return真情指数

本周主题：函数脱逃大师的秘密

Head First：我听说，你是个非常滑溜的人，几乎能从任何情况下脱身。

Return：没错。把我放在任何函数中，我马上就能从函数中离开。我甚至还会带走一些数据。

Head First：当你离开函数后，你去了哪边？

Return：这个嘛，别忘记函数都是由其他程序代码所调用，从函数返回（returning）只表示回到调用它的代码。此时，返回数据的意思是数据回到调用函数的代码。

Head First：哦……怎么进行呢？

Return：如果你把“函数调用”想成“已有结果的表达式”，应该有点帮助。如果函数不返回任何数据，表达式的结果就是什么都没有。但如果函数真的返回了数据（很多函数的确返回），表达式的结果就是一段数据。

Head First：所以，函数如果只是段表达式，这表示我们可以指定函数返回值给一个变量啰？

Return：不行，但又可以。不行，在于函数本身并非表达式 —— 调用函数的部分才是表达式。可以，在于你可以且应该经常利用函数调用，使其结果被指派给变量。这里就是表达式登场的地方 —— 估算函数调用时，它被视为表达式，其结果则是函数的返回值。

Head First：原来如此。但什么都没返回时，表达式又会发生什么事呢？

Return：如果你用了我，但没有附上返回值，函数什么都不会返回，表达式即为空白。

Head First：这样不会有问题吗？

Return：没有，不会真的有问题。你要知道，大家只在函数可以返回数据时担心函数返回值。如果函数并未打算返回任何东西，你也不应担心任何跟返回值有关的行动。

Head First：了解！回来谈谈你的脱逃技术。阻止函数的自然执行过程，这好像不是好事喔？

Return：不见得，让我说明一下。因为一个函数有第一行和最后一行，不表示它的设计一定是从第一行运行到最后一行。事实上，认为函数具有开始与结束，本身就是危险的想法。函数的“自然”结束很可能在代码的中段，因为某些老奸巨滑的设计师把我放在正确的地方。

Head First：听不太懂耶……你是说，某些函数代码始终不曾被调用，也很正常吗？

Return：我可没说“永远”，但函数中通常有数条不同的路径，我常常协助建造各个路径。如果发生了某些事，指示函数不该继续运行，我可以等着给它提早的出口。在其他状况下，函数也可能一路运行到最后一行，甚至还没遇到我就离开了；或者也可能因我而结束，所以我才能返回某些数据。

Head First：喔，这样子啊。你能提供一些选择，包括返回数据与控制函数执行的流程。

Return：不错嘛！你抓到重点了！

Head First：嘿嘿嘿，我在这方面的反应还不错。谢谢你过来与我们聊天。

Return：我也很高兴。不过，我要离开了！

JavaScript 发现自己突然身陷气候控制的丑闻中。“反快速加热联盟”（又称 WARM）的成员自己写了脚本，宣传关于局部暖化的信息。但“生气又厌倦变冷的负责市民组织”（ARCTIC）则决定压制 WARM 的信息，动手破坏他们的脚本。你的工作就是分辨出正常与被破坏的代码，并弄清楚 WARM 的意图。

```
function showClimateMsg() {
  return;
  alert(constructMessage());
}

function constructClimateMsg() {
  var msg = "";

  msg += "Global" ; // "Local" ;

  if (getTemp() > 80)
    msg += "warming";
  else
    msg += "cooling";

  if (true)
    msg += "is not";
  else
    msg += "is";

  if (getTemp() <= 70)
    return msg + "a hoax!";
  else
    return msg + "real!";

  return "I don't believe it.";
}

function getTemp() {
  // Read the actual temperature
  var actualTemp = readSensor();
  return 64;
}
```

Stop local warming NOW!

WARM 的支持者

JavaScript 发现自己突然身陷气候控制的丑闻中。“反快速加热联盟”（又称WARM）的成员自己写了脚本，宣传关于局部暖化的信息。但“生气又厌倦变冷的负责市民组织”（ARCTIC）则决定压制 WARM 的信息，动手破坏他们的脚本。你的工作就是分辨出正常与被破坏的代码，并弄清楚 WARM 的意图。

```
function showClimateMsg() {
  return;
  alert(constructMessage());
}

function constructClimateMsg() {
  var msg = "";

  msg += "Global"; // "Local";

  if (getTemp() > 80)
    msg += "warming";
  else
    msg += "cooling";

  if (true)
    msg += "is not";
  else
    msg += "is";

  if (getTemp() <= 70)
    return msg + "a hoax!";
  else
    return msg + "real!";

  return "I don't believe it.";
}

function getTemp() {
  // Read the actual temperature
  var actualTemp = readSensor();
  return 64; actualTemp;
}
```

（代码中 `return;`、`"Global"; //`、`if (true)`、`msg += "is not";`、`else`（其后为 `msg += "is";`）、`return "I don't believe it.";` 以及 `64;` 均被划掉；`actualTemp;` 为手写补充。）

return阻止alert框的出现。

预期的文字被改成注释，阻止它出现在信息中。

这段代码没问题，因为实际温度才能控制信息。

一段永远为 true 的if语句，没有意义。

if-else 语句彻底阻止函数取用这段代码，所以没有理由保留。

返回传感器的实际温度读数。

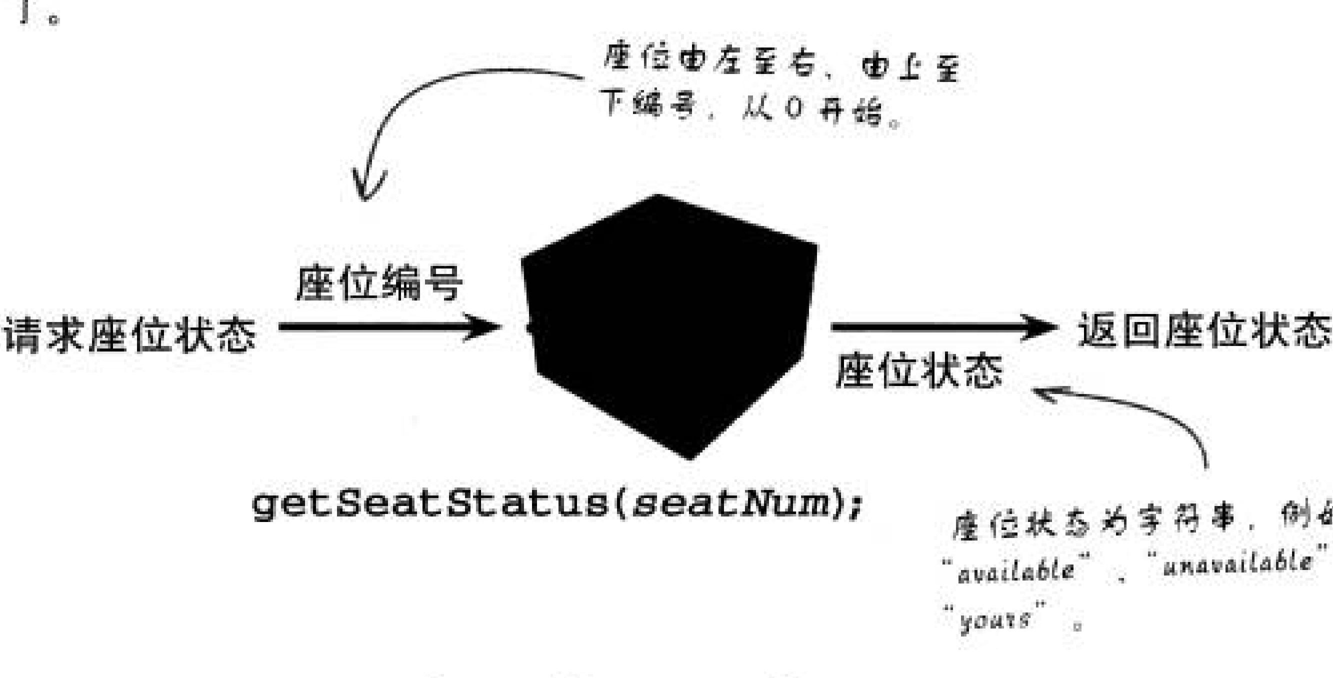
谢谢你，JavaScript！

这身打扮真的好热，救我！

局部暖化现象是真的！

取得座位状态

回到 Mandango，Seth 与 Jason 已经不想再听那些改变气候的鬼话，他们已经磨拳擦掌，准备进一步改善自己的脚本。有些用户说，他们老是搞不清楚座椅图像的颜色有什么意义，希望能有“点击座位即可查询是否为空位”的功能。听起来Mandango需要新的功能（函数）了。

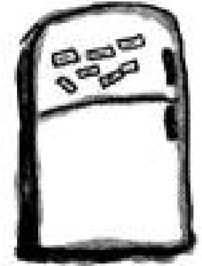


getSeatStagus()函数冰箱磁铁

Mandango 的getSeatStagus()函数缺少一些重要的代码，这样将无法找出特定座位的状态。函数首先检查座位是否为三个连续的可选择座位之一。如果不是，则从座位的数组里检查状态是否为“空位”（available）或“已售位”（unavailable）。请使用下面附上的代码磁铁完成这项练习。

```
function getSeatStatus(seatNum) {
  if (............ != -1 &&
    (............ == ............ || ............ == (............ + 1) || ............ == (............ + 2)))
    return "yours";
  else if (........[Math.floor(............ / ........[0].length)][ ............ % ........[0].length])
    return "available";
  else
    return "unavailable";
}
```

seats　　selSeat　　seatNum



getSeatStagus()函数冰箱磁铁解答

Mandango 的getSeatStagus()函数缺少一些重要的代码，这样将无法找出特定座位的状态。函数首先检查座位是否为三个连续的可选择座位之一。如果不是，则从座位的数组里检查状态是否为“空位”（available）或“已售位”（unavailable）。请使用下面附上的代码磁铁完成这项练习。

全局变量selSeat的值为−1时，表示用户尚未选择座位，所以优先检查。

我们要处理同一排的三个连续座位，所以必须检查某个座位和接下来的两个座位。

```
function getSeatStatus(seatNum) {
  if ( selSeat != -1 &&
   (seatNum == selSeat || seatNum == ( selSeat + 1) || seatNum == ( selSeat + 2)))
    return "yours";
  else if ( seats[Math.floor( seatNum / seats[0].length)][ seatNum % seats[0].length])
    return "available";
  else
    return "unavailable";
}
```

计算座位在数组里的行数（座位编号除以一行的座位数量，无条件舍去至整数）。

计算座位在数组里的列数（座位编号除以一行的座位数量，取余数）

你也可以写死数组长度（9），但如果改变了seats数组的长度就会失败。

显示划位状态

取得划位状态确实方便，但让用户查询任何座位状态，表示需在用户点击座位时呈现其状态。showSeatStatus()函数提供这个问题的简单解决方案，它把繁重的工作交给我们刚才写好的getSeatStatus()函数。

传递座位编号至getSeatStatus()，以取得座位状态。

```
function showSeatStatus(seatNum) {
  alert("This seat is" + getSeatStatus(seatNum) + ".");
}
```

字符串相连，建立座位状态的信息。

函数可与图像链接

把这个函数与 Mandango 网页的座椅图像接上线，用户即可透过点击图像而查询座位的状态。每个图像必须有个与 showSeatStatus() 函数联系的onclick事件，如下所示：

用户点击图像 "seat23" 时，将调用 showSeatStatus() 函数。

```
<img id="seat23" src=" " alt=" " onclick="showSeatStatus(23);" />
```

onclick!

Mandango - The Macho Movie Ticket Finder

This seat is yours.

OK

Find Seats

座位编号 23。

Done

想知道某个座位的状态时，只需点击一下鼠标，呈现状态的alert框便会弹出。对于不太会分辨座椅图像，希望只要点击某个座位就能取得状态的用户，新功能实在是一大利好。

复习要点

- return语句让函数返回数据给调用函数的代码。
- 函数返回一段数据时，数据将被放到调用函数的代码中。
- 函数只能返回一块数据。
- return 语句亦可不与任何数据同用，单纯用于提早结束函数。

重复的代码绝非好事

Mandango 脚本的运作不错，但 Seth 与 Jason 开始担心长期维护脚本的问题。尤其是 Jason，他做了一些研究，发现最新的网络应用程序多半受益于分离 HTML、JavaScript、CSS代码的设计。

```
<html>
  <head>
    <title>Mandango - The Macho Movie Ticket Finder</title>

    <script "type=text/javascript">
      ...

      function initSeats() {
        ...       }

      function getSeatStatus(seatNum) {
        ...
      }

      function showSeatStatus(seatNum) {
        alert("This seat is" + getSeatStatus(seatNum) + ".");
      }

      function setSeat(seatNum, status, description) {
        document.getElementById("seat" + seatNum).src = "seat_" + "status + .png";
        document.getElementById("seat" + seatNum).alt = description;
      }

      function findSeats() {
        ...
      }
    </script>
  </head>

  <body onload=initSeats();>
    <div style="margin-top:25px; text-align:center">
      <img id="seat0" src="" alt="" onclick="showSeatStatus(0);" />
      <img id="seat1" src="" alt="" onclick="showSeatStatus(1);" />
      <img id="seat2" src="" alt="" onclick="showSeatStatus(2);" />
      <img id="seat3" src="" alt="" onclick="showSeatStatus(3);" />
      <img id="seat4" src="" alt="" onclick="showSeatStatus(4);" />
      <img id="seat5" src="" alt="" onclick="showSeatStatus(5);" />
      <img id="seat6" src="" alt="" onclick="showSeatStatus(6);" />
      <img id="seat7" src="" alt="" onclick="showSeatStatus(7);" />
      <img id="seat8" src="" alt="" onclick="showSeatStatus(8);" /><br />
      <img id="seat9" src="" alt="" onclick="showSeatStatus(9);" />
      <img id="seat10" src="" alt="" onclick="showSeatStatus(10);" />
      <img id="seat11" src="" alt="" onclick="showSeatStatus(11);" />
      <img id="seat12" src="" alt="" onclick="showSeatStatus(12);" />
      <img id="seat13" src="" alt="" onclick="showSeatStatus(13);" />
      <img id="seat14" src="" alt="" onclick="showSeatStatus(14);" />
      <img id="seat15" src="" alt="" onclick="showSeatStatus(15);" />
      <img id="seat16" src="" alt="" onclick="showSeatStatus(16);" />
      <img id="seat17" src="" alt="" onclick="showSeatStatus(17);" /><br />
      <img id="seat18" src="" alt="" onclick="showSeatStatus(18);" />
      <img id="seat19" src="" alt="" onclick="showSeatStatus(19);" />
      <img id="seat20" src="" alt="" onclick="showSeatStatus(20);" />
      <img id="seat21" src="" alt="" onclick="showSeatStatus(21);" />
      <img id="seat22" src="" alt="" onclick="showSeatStatus(22);" />
      <img id="seat23" src="" alt="" onclick="showSeatStatus(23);" />
      <img id="seat24" src="" alt="" onclick="showSeatStatus(24);" />
      <img id="seat25" src="" alt="" onclick="showSeatStatus(25);" />
      <img id="seat26" src="" alt="" onclick="showSeatStatus(26);" /><br />
      <img id="seat27" src="" alt="" onclick="showSeatStatus(27);" />
      <img id="seat28" src="" alt="" onclick="showSeatStatus(28);" />
      <img id="seat29" src="" alt="" onclick="showSeatStatus(29);" />
      <img id="seat30" src="" alt="" onclick="showSeatStatus(30);" />
      <img id="seat31" src="" alt="" onclick="showSeatStatus(31);" />
      <img id="seat32" src="" alt="" onclick="showSeatStatus(32);" />
      <img id="seat33" src="" alt="" onclick="showSeatStatus(33);" />
      <img id="seat34" src="" alt="" onclick="showSeatStatus(34);" />
      <img id="seat35" src="" alt="" onclick="showSeatStatus(35);" /><br />
      <input type="button" id="findseats" value="Find Seats" onclick="findSeats();" />
    </div>
  </body>
</html>
```

这部分混杂了JavaScript与HTML代码，可以独立为事件处理器 HTML 属性。

Mandango网页上的JavaScript和HTML代码都混杂在一起。

功能与内容分离

混杂各种代码有什么大不了的吗？它不是运作得很正常吗？这里的问题主要在于你不该把 JavaScript 驱动的网页视为网页，应该视为**应用程序**（application）。正如任何良好的应用程序，JavaScript 应用程序也同样需要仔细规划与设计，小心驶得万年船嘛。更重要的一点，当内容、外观与功能分离时，良好的应用程序比较少有缺陷，而且比较容易维护。从 Mandango 的现状来看，它的内容、外观、功能全都乌漆抹黑地搅成了一团。

```
<html>
 ...
</html>
```

内容

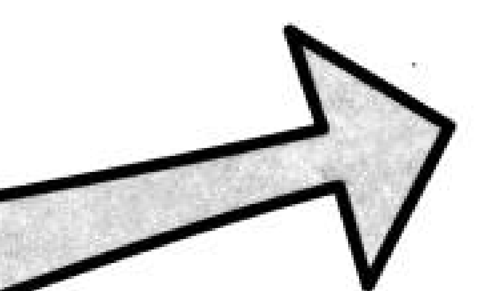
内容就是网页上的HTML代码，它提供网页实际上如何拼在一起的结构，同时也是网页数据的家。

内容、外观、功能的分离，可把大问题变成较小的问题。

```
<style>
 ...
</style>
```

外观

外观就是网页上的CSS代码，它妆点内容的外表，决定字体、色彩，甚至可决定排版结果。



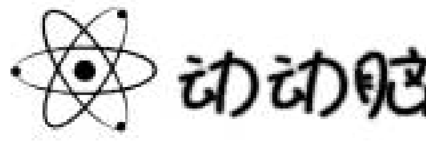
```
<script>
 ...
</script>
```

功能

功能就是驱动网页、带来交互性的JavaScript代码。你可以把这部分想象成负责工作的网页。

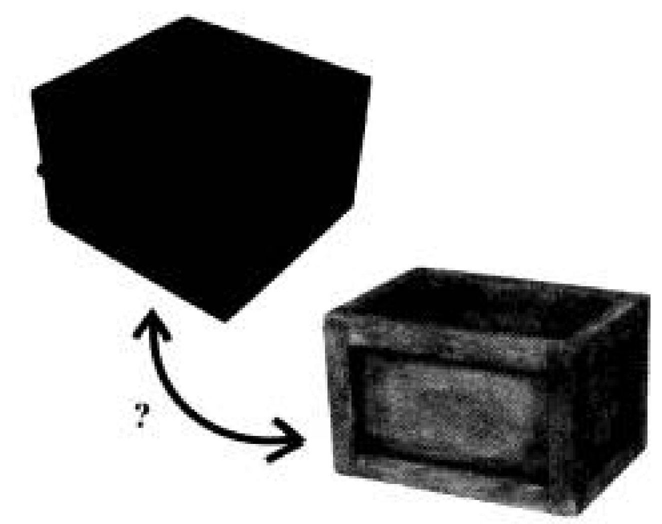
分离功能与内容，让网络应用程序更容易建立与维护。

请用这种方式看待代码分离的议题。假设 Seth 与 Jason 发现一段非常机灵的座位搜索代码，若想改用新脚本，他们将需要全面翻修 Mandango，但又得冒着弄坏网页结构的危险，因为JavaScript代码与 HTML代码太相亲相爱了，整天都黏在一起。如果 HTML代码能够独立，“JavaScript 转换成 HTML”则纯粹留给JavaScript处理，这个世界会更好。



你要如何利用函数从 Mandango 的内容中分离功能呢？

函数只是数据



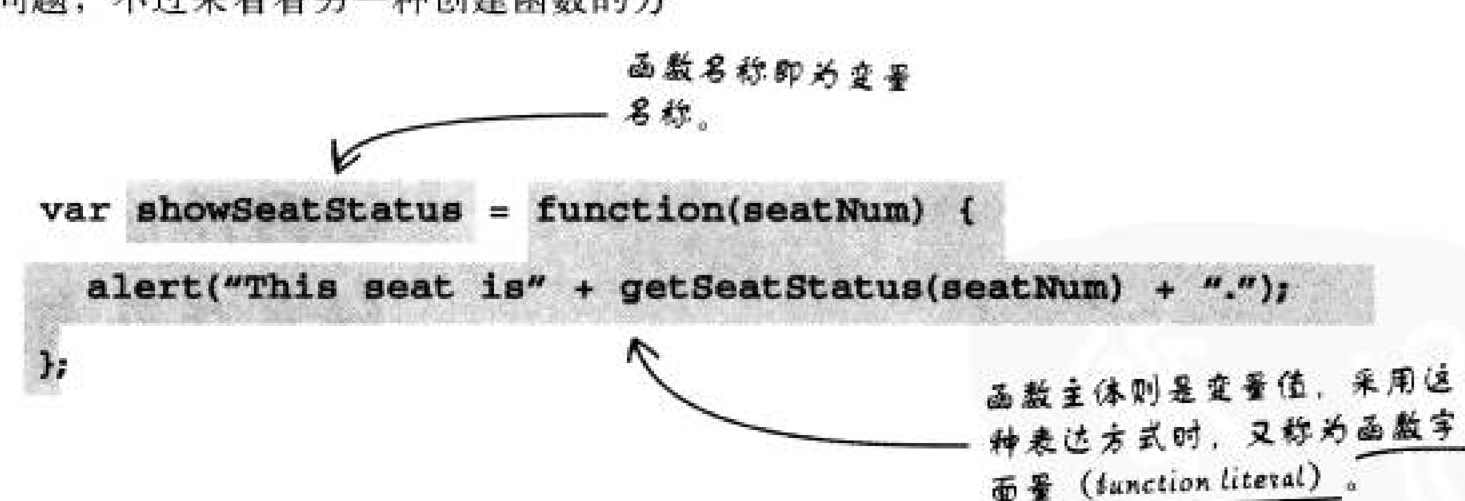
为了有效地分离代码，你将需要理解函数与事件联结的方式。目前为止，我们都是利用 HTML 的属性。但还有另一种方式，大多数人都觉得这种混合 JavaScript 与 HTML 的新方式比较高级。这种联结事件处理器的新方式，需要我们用不同的角度看函数。

函数只不过是变量，够惊讶了吧！我没骗你，真的。其中的曲折在于把函数主体看成值，函数名称则为变量名称。下例是各位习惯的看待函数的方式：

我们熟悉的函数创建方式。

```
function showSeatStatus(seatNum) {
  alert("This seat is" + getSeatStatus(seatNum) + ".");
}
```

上例代码没有问题，不过来看看另一种创建函数的方式：




```
var showSeatStatus = function(seatNum) {
  alert("This seat is" + getSeatStatus(seatNum) + ".");
};
```

这段代码显示了函数也能利用变量语法而建立，甚至构成元件也相同：独一无二的标识符（函数名称）、值（函数主体）。当函数主体单独出现而没有名称时，又被称为**函数字面量**（function literal）。

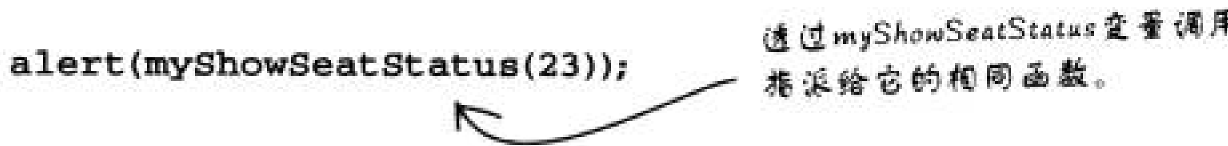
这个关于函数的秘密有什么好玩的吗？其实，它揭露了函数能像变量般被操纵的事实。例如说……你觉得下列代码有什么功用？

```
var myShowSeatStatus = showSeatStatus;
```

指派 showSeatStatus() 函数给变量 myShowSeatStatus。

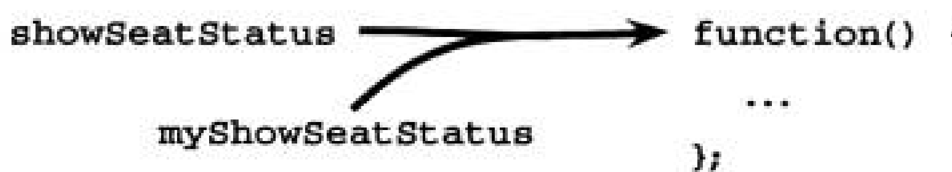
调用或引用你的函数

指派函数名称给另一个变量时，就是让变量访问函数主体。换句话说，你可以用下列方式设计代码，以便调用函数：



函数其实只是“值”引用到函数主体的变量。

调用myShowSeatStatus()函数的结果正与调用showSeatStatus()相同，因为两个函数最后都**引用**（reference）了相同代码。因此，函数名称也被称为**函数引用**（function reference）。



“**引用**函数”与“**调用**函数”的差别，与函数名称后是否附有括号 () 有关。函数引用只会单独出现，但函数调用则必定后随括号，很多时候还附有自变量。

```
var myShowSeatStatus = showSeatStatus;
myShowSeatStatus(23);
```

运行myShowSeatStatus()函数，与showSeatStatus()相同。

对myShowSeatStatus指派函数引用。

习题

试分析下列代码，并写出alert框中出现的数字。

```
function doThis(num) {
num++;
return num;
}

function doThat(num) {
num--;
return num;
}

var x = doThis(11);
var y = doThat;
var z = doThat(x);
x = y(z);
y = x;
alert(doThat(z - y));
```

试分析下列代码，并写出alert框中出现的数字。

```
function doThis(num) {
num++;
return num;
}

function doThat(num) {
num--;
return num;
}

var x = doThis(11);
var y = doThat;
var z = doThat(x);
x = y(z);
y = x;
alert (doThat(z-y));
```

x = 12
y = doThat
z = doThat(12) = 11
x = doThat(11) = 10
y = 10
alert(doThat(11 - 10))

0

OK

没有蠢问题

问：分离内容真的这么重要吗？

答：没错，但又不见得。对简单程序而言，HTML、CSS、JavaScript全都融合在一起不见得有问题。若在较复杂、用到大量代码的应用程序里，分离代码的好处才会突显。面对大型程序时，我们较难掌握整体状况；也就是说，修改大型程序时比较容易出问题，尤其在不同类型代码都混杂在一起的状况下。借由清楚地分离代码，修改功能时也可比较安心，不会误触网页结构或外观。同时，不同领域的专家也能同时处理同一个项目。例如，网站设计师处理结构与程序外观的同时，无需害怕造成JavaScript函数代码的错误（网站设计师可能不懂JavaScript代码）。

问：如果函数只是数据，我该如何分辨函数与一般变量？

答：函数与“一般”变量的差异，在于你如何使用数据。与函数相关的数据（或代码）可被执行。想执行函数时，你就在函数名称后加上括号()，如果函数需要自变量，也要记得附上。

问：函数引用有什么重要性？

答：函数不像一般变量，变量将其数据存储成存储器的某个区域里的值，函数则存储对代码的引用(reference)。所以，函数变量的值不是代码本身，而是指向存储代码的存储器位置的引用。有点像地址是你家的参考，但地址本身不等于你家。

函数使用引用而非实际的值，因为比起存储多份函数代码的副本，引用有效率多了。当你指派函数给事件处理器时（等下就会用到），其实只是指派对函数代码的引用，而不是代码本身。

嗯，函数引用听起来很简洁，不过，它们与内容和功能的分离有什么关系？

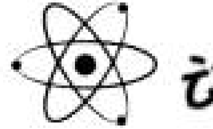
函数的两极相生（callback，回调功能）

函数引用与一种特殊的函数调用方式紧密相连，这种调用方式与内容和功能的分离大有关联。你对Mandango代码中的函数调用方式已经很熟了：

```
setSeat(i * seats[i].length + j, select, Your seat);
```

```
function setSeat(seatNum, status, description) {
  ...
}
```

不过，脚本调用函数的方式不只这一种。另一种方式称为**回调函数**（callback function），就算你没有直接操作，也能调用函数。



动动脑

你觉得 Mandango 能从回调函数获得什么好处呢？

今晚主题：一般函数与回调函数的针锋相对

一般函数： 想必你就是我最近常听人家说起的那个不肯接受本地调用的神秘人物。你为什么这么高傲呢？

回调函数： 跟态度无关，只是用途不同。我偏好接受外来的、遥远异地的调用。

一般函数： 你是指浏览器之类的调用吗？真是很另类。我还以为你不屑跟我们这些每天与脚本交流的小人物为伍咧。

回调函数： 我跟你说，一切无关谁好谁坏。我们都是脚本，只不过我让外来客有个访问脚本的管道。如果没有我，你永远不会知道脚本外发生的事。

一般函数： 哦……好大的损失啊……才怪！谁关心脚本外发生了什么事？

回调函数： 事实上，大家都关心。别忘记脚本编程就是为了给网络用户提供更佳的体验。如果脚本无法检测外部的事件，我们也很难改善用户的体验。

一般函数： 听来倒有几分道理。我的确想知道网页载入的时机，或是用户点击／键入某物的时机。你的意思是说，如果没有你，我就无法察觉这些事情吗？

回调函数： 没错。浏览器调用我，很多状况下我会再调用你，因为响应外界事件通常需要数个函数。

一般函数： 噢，原来我们没那么不一样啊！

回调函数： 对啦！这样你就不会避开我了吧！

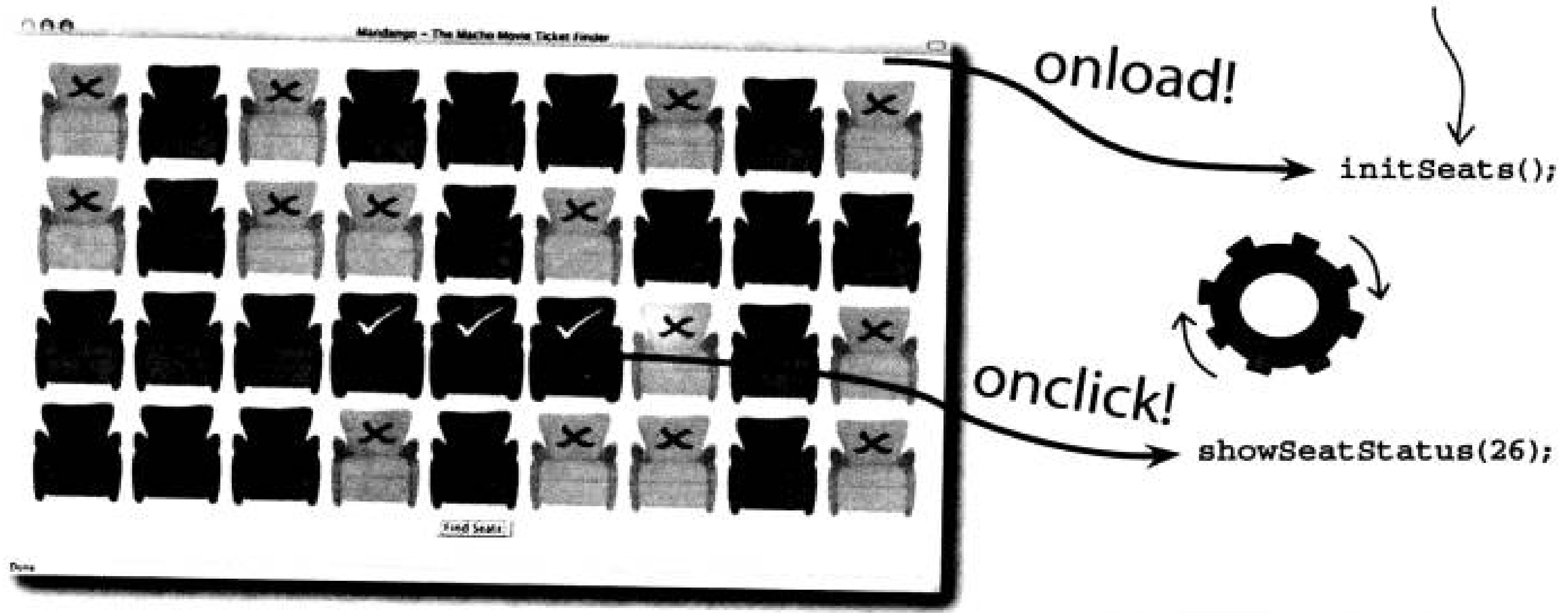
一般函数： 你不用主动调用我，我会调用你。

回调函数： 最好你办得到啦……

事件、回调与HTML属性

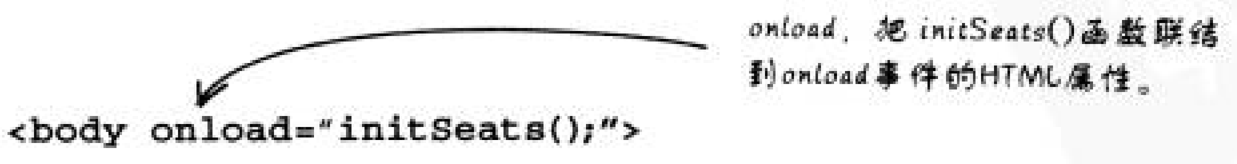
一直以来，我们都在利用回调函数，浏览器会调用它们（不是由你的程序代码使用）。回调函数最常用于处理事件。Mandango原已重度依赖回调函数。事实上，事件处理函数，就是HTML代码与JavaScript代码混杂问题的基础。

浏览器于载入页面时调用 initSeat()。



这些回调函数为 Mandango 网页连接了HTML代码的“事件”。

onload，把 initSeats()函数联结到onload事件的HTML属性。

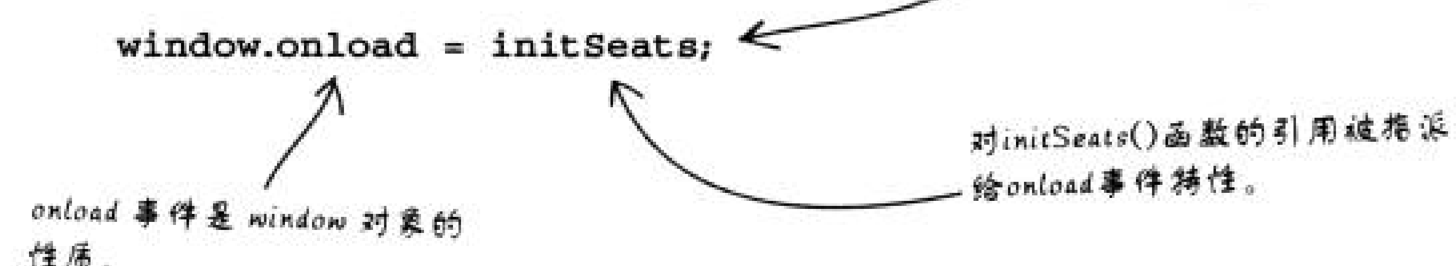


onclick，把showSeatStatus()函数联结到onclick事件的HTML属性。

透过HTML属性，联结“事件处理函数”与“其他函数”的技巧确实行得通，但却需要让 JavaScript 和 HTML代码龙蛇混杂，这是它的缺点。函数引用则可分离代码的大杂烩，让拆开 HTML 与 JavaScript 变得可行……

使用函数引用联结事件

你可以直接在JavaScript代码中指派函数引用作为事件处理器，以取代使用“HTML 属性”联结回调函数与事件的方式。换句话说，你完全不需冒着深入HTML代码的风险 —— 只要使用函数引用设定回调函数，一切都来自JavaScript代码内部。



所以，单纯于 JavaScript代码中设定事件处理器，包括指派函数引用给某个对象的事件特性。以`onload`事件处理代码为例，函数引用的指派，使得触发该事件即调用 `initSeats()` 函数。虽然“调用”于事件触发后自动发生，但效果其实是：

使用函数引用指派事件处理函数的好处，即在于创建与 HTML 清楚分离的JavaScript代码 —— 不需再把JavaScript代码指派给HTML的事件属性。

现在 `<body>` 标签可以单纯地做个`<body>`标签，函数处理已经都在JavaScript代码中完成联结。我们只要确保事件指派代码的启动尽可能简单，所以通常放在网页的标头区域（header）。

函数引用提供联结事件处理函数与事件的便利方式。

但是有个问题。如果我们需要把自变量传入事件处理器，才能协助它完成工作时怎么办？Mandango 使用的`onload`没有问题，但`onclick` 事件需在点击座位时传送座位编号。函数引用没有提供透过自变量传递数据的方式，我们需要其他选择……

函数字面量拔刀相助

Mandango 所用座椅图像的 onclick 事件，调用 showSeatStatus() 函数时必须带有自变量（座位编号）以指出被点击的座位。单纯指派函数引用而无法传递自变量，带来很大的问题。不过，事情总有其他解决方式 —— 使用函数字面量（fuction literal）作为函数引用，而后从函数字面量内调用showSeatStatus()函数。

onclick!

撷取座椅图像对象，它的 onclick 才能被访问。

```
document.getElementById("seat26").onclick = function(evt) {
  showSeatStatus(26);
};
```

函数字面量“包裹”对 showSeatStatus()的调用，允许自变量的传入。

函数字面量被指派给onclick事件特性，作为函数引用。

事件对象被自动传送给事件处理器，作为它的第一个自变量。

函数字面量单纯用于“包裹”（wrap）对showSeatStatus()的调用，但在允许我们传递适当的座位编号给函数一事上，它也扮演了关键角色。你可以把函数字面量想象成处理事件的无名函数（nameless function）。因此，函数字面量有时被称为**匿名函数**（anonymous function）。

范例代码呈现了JavaScript如何提供传入事件处理器的事件对象（event object），本例的对象透过自变量evt传入。事件对象包含各个事件特有的信息。本例中，我们不需知道任何关于事件的详细信息，所以用不到自变量evt其实也没关系。

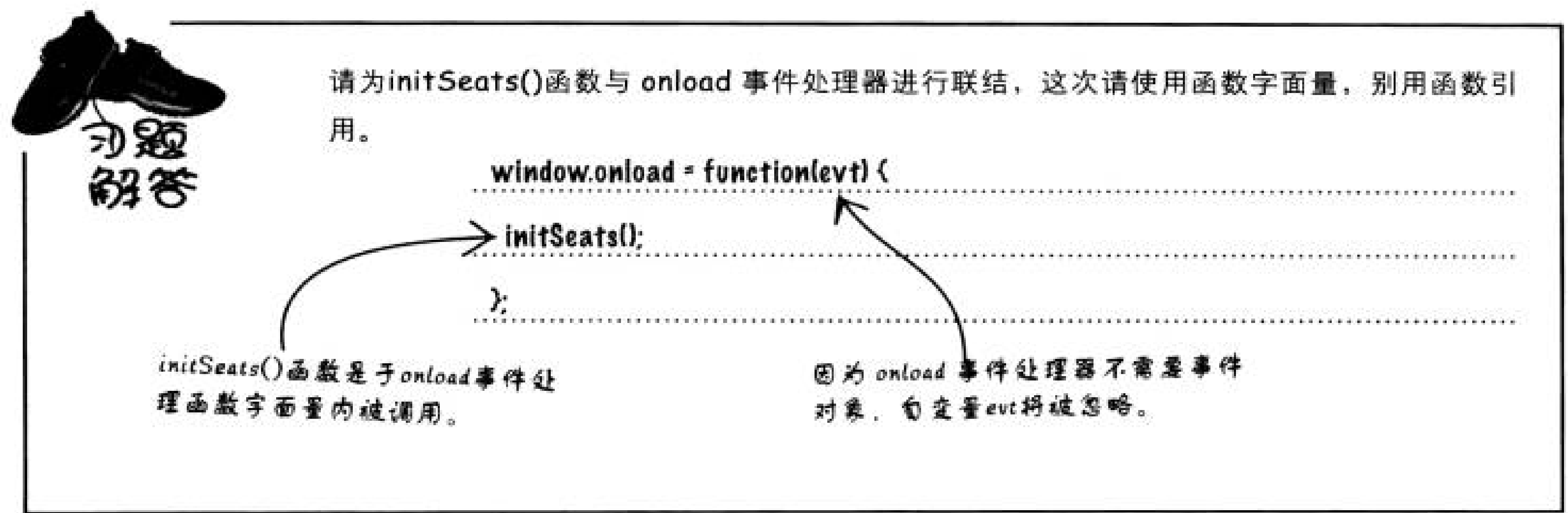
函数字面量让你创建匿名事件处理函数。

请为 initSeats() 函数与 onload 事件处理器进行联结，但这次请使用函数字面量，别用函数引用。

....

....

....

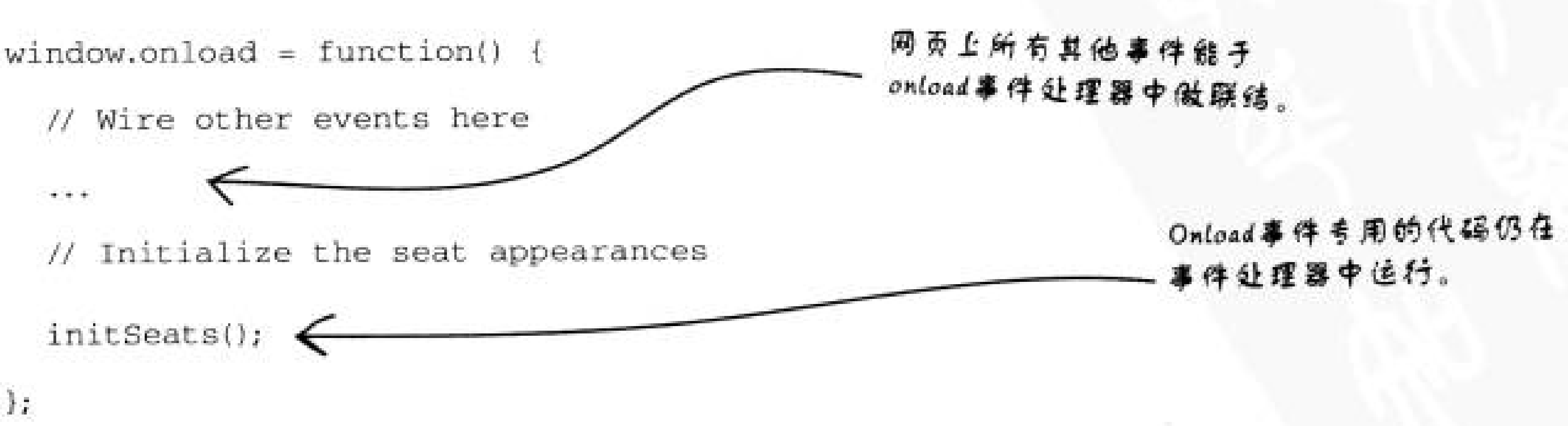


联结何在？

透过函数字面量，还是有个关于事件联结的问题没有解决。我们知道，onload 事件处理器能于网页标头的 <script> 标签里被指派，就像其他一般脚本。这个方式的运作一直不错，因为和 onload 绑在一起的代码要到网页载入后才会运行（当 onload 事件发生后）—— 就像指派 initSeats() 给 HTML 的 <body> 标签的 onload 属性这个老方法。但其他函数字面量事件处理器又该在哪里得到联结呢？

onload 事件处理器是个初始化所有事件的绝佳地方。

答案回到 onload 事件处理器回调函数，它是个联结网页上所有事件的好地方。



```
window.onload = function() {
  // Wire other events here
  ...
  // Initialize the seat appearances
  initSeats();
};
```


总结上例，onload 事件处理器变成了事件初始化函数，用于设置网页中其他事件。所以，onload 事件处理器不只执行一般的网页启始任务，例如座位状态初始化，它也把所有其他事件处理器回调函数联结上应用程序。

没有蠢问题

问：回调函数为什么重要？

答：回调函数的重要性，在于可让我们对代码外发生的事情有所反应。取代由你的代码中调用函数，改为创建回调函数，这是一直在等待“某事”发生好准备起身动作的函数。当我们说的“某事”发生了，通知回调函数运行则是浏览器的责任。只需要把回调函数联结上触发器（通常是个事件），设定活动舞台的工作就完成了。

问：除了事件处理器，还有其他回调函数吗？

答：有的。第12章将探讨另一种回调函数的常用法——处理使用Ajax请求数据时由服务器传来的数据。

问：我还是跟函数字面量不太熟，它们是什么东西？为什么这么重要？

答：函数字面量只是没有名称的函数主体，有点像是实际的数据，例如一个数字或字符串。函数字面量的重要性，则在于它们很适合需要快速偶发回调函数的情况。换句话说，函数只被调用一次，而且不是被你的程序代码调用。所以你创建一个函数字面量并直接指派给事件特性，而不是创建有名称的函数并指派到它的引用。其实一切比较关于编写程序时的效率，利用（某些情况）不需中规中矩地为函数命名的优势。也别忘了，除非不只需要函数引用（例如需要传送自变量给函数），函数字面量才真的必要。

磨笔上阵

请帮 Mandango 完成新的onload事件处理函数。

```
window.onload = function() {
  // Wire the Find Seat button event
  document.getElementById("findseats"). ...........= ............;

  // Wire the seat image events
  document.getElementById("seat0"). ........... = function(evt) { ........................... };
  document.getElementById("seat1"). ........... = function(evt) { ........................... };
  document.getElementById("seat2"). ........... = function(evt) { ........................... };
  ...

  // Initialize the seat appearances
  ................
};
```

磨笔上阵解答

请帮 Mandango 完成新的onload事件处理函数。

onload事件处理器其实是个函数字面量。

利用函数参考，findSeat() 函数与 onclick 事件相联系。

```
window.onload = function() {
  // Wire the Find Seat button event
  document.getElementById("findseats"). onclick = findSeats;

  // Wire the seat image events
  document.getElementById("seat0"). onclick = function(evt) { showSeatStatus(0); };
  document.getElementById("seat1"). onclick = function(evt) { showSeatStatus(1); };
  document.getElementById("seat2"). onclick = function(evt) { showSeatStatus(2); };
  ...

  // Initialize the seat appearances
  initSeats();
};
```

调用initSeat()函数以结束onload的任务。

访问每个座椅图像的onclick特性，以设定onclick事件处理器。

从函数字面量内调用showSeatStatus()，自变量才能传进去。

复习要点

- 回调函数被浏览器调用，以响应**发生在脚本外**的事情。
- 函数引用能像变量般指派函数。
- 函数引用让我们联结JavaScript代码设计的事件处理函数，而**不需改动HTML代码**。
- 函数字面量为**无名函数**，在不需要有名称的函数时非常方便。

问：Mandango为什么把onload事件处理器创建为函数字面量？

答：因为完全没有创建有名称的函数的理由。这个函数只会创建一次，创建目的就是响应onload事件。我们也可以创建一个函数并指派它的引用给window.onload，但函数直接使用函数字面量与事件绑在一起时，回调函数与事件的联结反而更为清楚。

问：其他回调函数也必须联结到onload事件处理器中吗？

答：是的。或许有人正想把回调函数直接与网页标头的<script>标签相联结。但请记得网页内容在载入标头时尚未载入。所以，所有调用getElementById()的行动都会失败，事件处理器也不会得到联结。onload处理器则保证网页一定已经载入。

以HTML网页为外壳

分离Mandango的JavaScript代码与HTML代码，揭露了网页的结构部分可变得多么轻巧迷你。HTML代码也更易维护，不用担心伤害JavaScript代码可能破坏了应用程序。

老兄，我要的远景就像它！

快看这些代码！真够可维护的啦！

```
<body>
  <div style="margin-top:75px; text-align:center">
    <img id="seat0" src="" alt="" />
    <img id="seat1" src="" alt="" />
    <img id="seat2" src="" alt="" />
    <img id="seat3" src="" alt="" />
    <img id="seat4" src="" alt="" />
    <img id="seat5" src="" alt="" />
    <img id="seat6" src="" alt="" />
    <img id="seat7" src="" alt="" />
    <img id="seat8" src="" alt="" />
    <img id="seat9" src="" alt="" /><br />
    <img id="seat10" src="" alt="" />
    <img id="seat11" src="" alt="" />
    <img id="seat12" src="" alt="" />
    <img id="seat13" src="" alt="" />
    <img id="seat14" src="" alt="" />
    <img id="seat15" src="" alt= />
    <img id="seat16" src="" alt="" />
    <img id="seat17" src="" alt="" />
    <img id="seat18" src="" alt="" />
    <img id="seat19" src="" alt="" /><br />
    <img id="seat20" src="" alt="" />
    <img id="seat21" src="" alt="" />
    <img id="seat22" src= alt="" />
    <img id="seat23" src="" alt="" />
    <img id="seat24" src="" alt="" />
    <img id="seat25" src="" alt="" />
    <img id="seat26" src="" alt="" />
    <img id="seat27" src="" alt="" />
    <img id="seat28" src="" alt="" />
    <img id="seat29" src="" alt="" /><br />
    <img id="seat30" src="" alt="" />
    <img id="seat31" src="" alt="" />
    <img id="seat32" src="" alt="" />
    <img id="seat33" src="" alt="" />
    <img id="seat34" src="" alt="" />
    <img id="seat35" src="" alt="" />
    <img id="seat36" src="" alt="" />
    <img id="seat37" src="" alt="" />
    <img id="seat38" src="" alt="" />
    <img id="seat39" src="" alt="" /><br />
    <input type="button"id="findseats" value="Find Seats" />
  </div>
</body>
```

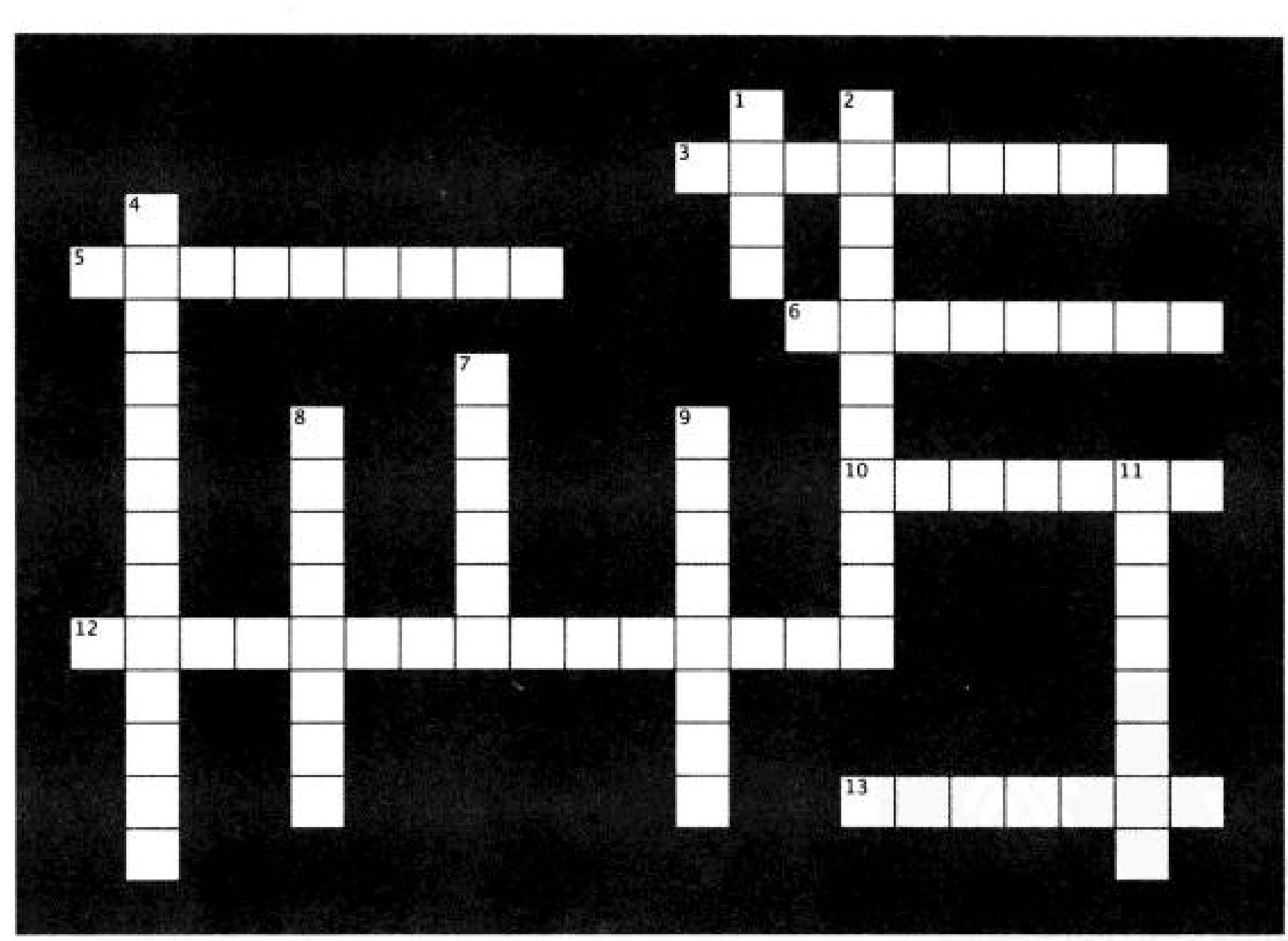
JavaScript 的一小步……

这一章虽然没办法解决世界和平的大问题，但在使用 JavaScript 处理气候控制的问题上，的确朝着正确方向迈进了一小步。把大问题转变成小问题、专注于目的独特性并致力于代码的再利用，都是函数改良脚本的方式。

当然，Seth 与 Jason 在工作上运用相同的解决技巧，创建结构更良好、更容易维护的 Mandango。如果没发生什么意外，世上的彪形大汉都能舒服地看电影了……

JavaScript 填字游戏

填字游戏的时间又到了。休息一下，利用填字游戏活络你的大脑皮层。

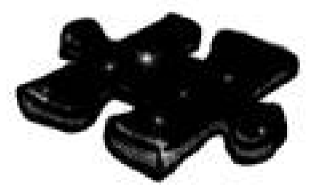


横向提示：

3. 指派函数给变量时，你在使用 function ________。
5. 函数有助于削减________代码。
6. 你本人不会调用这类函数。
10. 无名函数的主体。
12. 当函数相对容易调整，即视为具有良好的______。
13. HTML 呈现了网页的哪个部分？

纵向提示：

1. ________现在跟 Mandango 和平相处了。
2. 函数改善了程序的________，所以我们不必一直复制代码。
4. JavaScript 呈现了网页的哪个部分。
7. 想从函数中返回数据，请用关键字________。
8. 一段可以重复使用的JavaScript代码。
9. 函数很适合拆解________。
11. 想传入数据给函数，要靠________。



JavaScript 填字游戏解答

												1 S		2 R						
											3 R	E	F	E	R	E	N	C	E	
	4 F											T		U						
5 D	U	P	L	I	C	A	T	E				H		S						
	N												6 C	A	L	L	B	A	C	K
	C						7 R							B						
	T			8 F			E				9 P			I						
	I			U			T				R			10 L	I	T	E	R	11 A	L
	O			N			U				O			I					R	
	N			C			R				B			T					G	
12 M	A	I	N	T	A	I	N	A	B	I	L	I	T	Y					U	
	L			I							E								M	
	I			O							M								E	
	T			N							S			13 C	O	N	T	E	N	T
	Y																		T	

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

函数为你的JavaScript世界添加了什么？

这是左右脑的秘密会谈！

食物

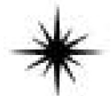
我好冷。

真热，一点都不舒服。

庇护所

人权

和平一直都是艰难的命题。就算利用JavaScript代码，却只有组织最佳的代码才能导向安稳与宁静。生命总是不易平稳无波，至少JavaScript的世界很容易动荡不安。



7 表单与验证

让用户全盘托出

有了 JavaScript，你不用风度翩翩或鬼鬼祟祟，一样能成功取得用户的信息。但你必须千万小心。人类很容易犯错，也就是说，在线表单提供的数据不见得全都**准确**或合格。JavaScript 能帮上一点忙。把输入的**表单数据传给 JavaScript 代码，可让网络应用程序更可靠，也能为服务器减少一些负担。珍贵的带宽应让给重要事物，**例如精彩的影片或可爱宠物的照片。

Bannerocity：飞翔的广告

Howard 是名热爱特技表演的飞行员，他最近的热忱转向空中横幅事业，Bannerocity。Howard 希望为“横幅广告”（banner ad）带来全新意义 —— 他要在网上接受空中横幅的订单。除了开启新事业，Howard 希望在线订单系统能帮他省下待在地上的时间，尽情于天际遨翔。

Howard 拥有一架二战年代的老式飞机。

Bannerocity...banner ads in the sky!

Howard 现有的书面订单格式，已经具备所有空中横幅的必需信息。

Message: Duncan's Donuts... only the best!

ZIP code: 74129

Fly date: December 14th, 2008

Name: Duncan Glutenberg

Phone #: 408-555-5309

Bannerocity 的在线订单系统必须捕捉所有和空中横幅相关的订购信息，这很重要。Howard 认为，在线订单应该包含现有纸质订单的每一栏，另外加上电子邮件栏，因为客户将在网络上填写订单。

Message（横幅信息）
客户想用空中横幅呈现的信息。

ZIP code（邮政编码）
呈现信息的地理区域。Howard 会于飞过指定邮政编码地域时亮出横幅。

Fly date（飞行日期）
载着横幅飞上天的日期。

Email（电子邮件）
客户的电子邮件地址。

Name（姓名）
客户姓名。

Phone number（电话号码）
客户的电话号码。

Bannerocity 的 HTML 表单

向HTML寻求了一点协助，Howard 踏上Bannerocity在线订购的第一步，看来还不错。

Bannerocity – Personalized Online Sky Banners

BANNEROCITY

Enter the banner message:
Enter ZIP code of the location:
Enter the date for the message to be shown:
Enter your name:
Enter your phone number:
Enter your email address:
Order Banner

Done

哇！这份在线订单很有模有样嘛！

全新的 Bannerocity 订购表单具有一切必要表单域，也在没有 JavaScript 的情况下准备好接受订单。有什么不对劲吗？

Howard目前为私人机构服务，不过他就是很想念以前那套制服。

请至http://www.headfirstlabs.com/books/hfjs/下载，准备上阵吧！

磨笔上阵

用 Howard 的 HTML 表单填写一份订单吧。别担心，我们不会跟你索取空中横幅广告的费用啦！

Enter the banner message:
Enter ZIP code of the location:
Enter the date for the message to be shown:
Enter your name:
Enter your phone number:
Enter your email address:

磨笔上阵解答

用 Howard 的 HTML 表单填写一份订单吧。别担心，我们不会跟你索取空中横幅广告的费用啦！

你不知道 Howard 的空中横幅只装得下 32 个字符，这一栏太长了。

邮政编码太长了——应该只有五位数。

Enter the banner message: Mandango... the movie seat picker for tough guys!

Enter ZIP code of the location: 100012

Enter the date for the message to be shown: March 11, 2009

Enter your name: Seth Tinselman

Enter your phone number: (212) 555-5339

Enter your email address: setht@mandango

还好，姓名没问题。

这是个无效的电子邮件地址——缺少 .biz 这类的域扩展名。

电话号码的格式应该是 ###-###-####，不采用括号。

日期并未采用表单预期的 MM/DD/YYYY 格式。

当只有 HTML 还不够的时候

Howard 发现在线表单只能提供数据输入的管道，他还需要 JavaScript 协助维持表单数据的可靠性。他也需要清楚告知用户，什么样的数据，才构成了“好数据”。举例来说，Bannerocity 的页面上需要一些提示，用户才会知道空中横幅需限定在 32 个字符以内，或者日期应该采用 MM/DD/YYYY 格式等等。

Enter the banner message: Mandango... the movie seat picker for tough guys!

借用一点 JavaScript 的协助，可阻挡不合格的数据。

HTML 表单通常不会讲话吧！

但有个小问题。如果 Howard 不知道如何让 JavaScript访问表单数据，JavaScript 数据操纵代码再聪明伶俐也无用武之地……

访问表单数据

为了访问输入表单的数据，首先需要区分表单中的每个域。这点可利用 HTML 代码的 id 或 name 属性（或两者并用）处理。

name 属性可独一无二地识别表单中的域。

id 属性可独一无二地识别网页元素。

```
<input id="zipcode" name="zipcode" type="text" size="5" />
```

以上两个属性构成为 input 域的标识符。

Enter ZIP code of the location: 100012

表单域具有两种识别方式的原因，均与访问表单域的途径有关。第一种途径使用 getElementById() —— 可访问网页上任何元素的函数。这个方式没问题，但还有更简单、更针对表单设计的途径。

每个表单域都有一个 form 对象，可被传给任何验证表单数据的函数。

```
<input id="zipcode" name="zipcode" type="text" size="5" onclick="showIt(this.form)"/>
```

form 对象厉害的地方，在于它也是个数组，负责存储表单中所有域。但它的数组元素并非利用数值索引存储，而是使用域独有、于 name 属性设定的标识符。假设 form 对象作为自变量 theFome 传给某个函数，则输入邮政编码（ZIP code）域的值将以下列方式被访问：

form 对象的自变量名称。

```
function showIt(theForm) {
  alert(theForm["zipcode"].value);
}
```

呈现出邮政编码域的值。

100012

表单域的独有名称，如 <input> 标签的 name 属性所设定。

我们希望存储域值，而不是域本身。

利用 name 属性的途径与使用 getElementById() 函数并无高下之分，只差在它形成的代码比较简短易读。既然 form 对象提供了捷径，我们就不要客气，好好利用吧。

问：为什么每个表单域都能访问 form 对象？

答：有时候不行，但请记得，表单域能用于调用需要访问其他表单域的验证函数。此时，每个表单域里都能访问 form 对象成为方便访问其他表单域的关键。这个对象通常被传入验证函数，让函数快速抓出需要的域。我们的 Bannerocity 范例将继续重用 form 对象，从订购表单中访问域。

问：value 是表单域的一个特性（property）吗？这表示每个表单域其实都是一个对象吗？

答：没错，正是如此。对 JavaScript 代码，每个表单域均表现为一个对象，form 对象则提供了快速而轻松地访问这些对象的方式（以取得表单中的任何域）。一旦你用 form["objectname"] 拿到表单域对象，再加上 value 特性就能访问域值。第 9 章和第 10 章还会进一步讨论对象。

检查表单数据的时机，取决于选择正确的用户输入事件去处理。

对于“何时”验证数据的答案，与事件有关。另外，还要知道让你察觉用户何时输入数据至域的事件。换句话说，挑战在于响应于数据输入后立刻被触发的事件。但问题还没有答案……究竟是哪个事件？

表单域带来一连串事件

当数据被输入表单时，产生了一连串事件。你可以根据域基础，使用这些事件作为验证域数据的进入点。我们先仔细观察一般输入数据时的顺序，并理解发生了哪些事件……与何时发生。

1. 选择输入域（onfocus）。
2. 在域里输入数据。
3. 离开该域，移向下个目标（onblur/onchange）。
4. 选择下个输入域（onfocus）。
5. 在域里输入数据……（下略）

输入数据至表单，点燃一系列有趣的JavaScript事件。

1 onfocus!

2

Enter the banner message: Mandango... the movie seat picker for tough guys!

Enter ZIP code of the location: 100012

4 onfocus!

3 onblur! onchange!

5

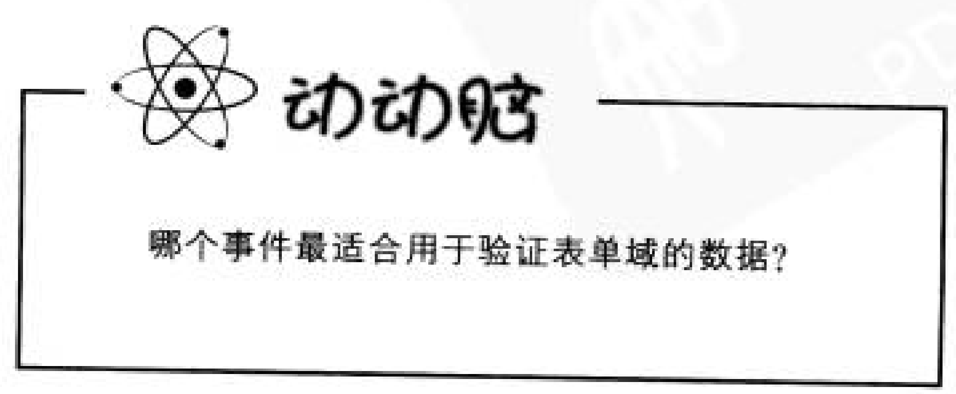
Enter the date for the message to be shown:

Enter your name:

Enter your phone number:

Enter your email address:

一开始选择输入的域时，触发了 onfocus 事件；当域不再被选择输入时，则触发了 onblur。onchange 事件与 onblur 有点相似，但它只在某个域不再被选择而且输入内容被改变时被触发。

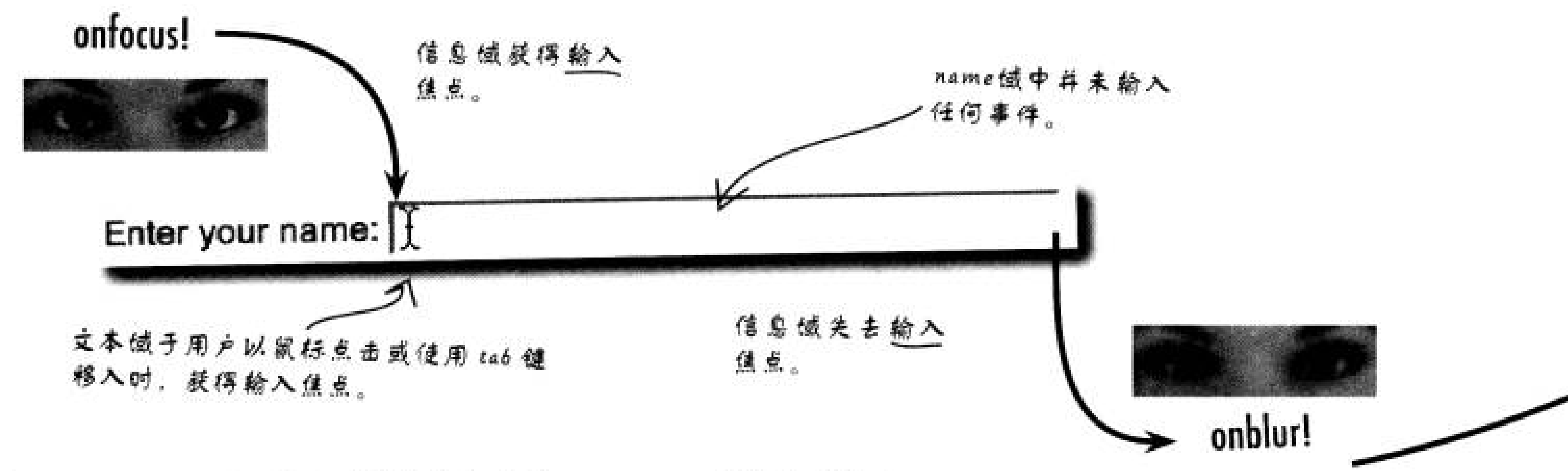


哪个事件最适合用于验证表单域的数据?

不再是焦点：onblur

虽然关于使用 onchange 事件于数据验证尚有争论，其中尤以“onchange 不可用于验证空域”特别值得注意。表单初次载入时完全没有任何域数据，就算用户浏览了每个空域，但因为表单数据并未改变，onchange 将无法被触发。onblue 事件能解决这个问题，每次输入选择（input selection，或称 focus，焦点）的离开，都会触发这个事件。

onblur事件极度适合触发数据验证。



不像 onchange，只要域不再是焦点即触发 onblur，就算并未触及数据。也就是说，onblur 非常强大，但也表示你必须小心处理数据验证问题的告知方式与时机。例如说……alert 框，即可能是个容易但有风险的验证告知备案。

问：在用户实际输入数据的时候，不会产生某些事件吗？

答：当然会。有许多响应键盘行为的事件，像 onkeypress、onkeyup、onkeydown等等。虽说有些状况适合响应这些事件，但通常不适合验证，因为在这些事件触发的时间点，用户多半还在输入信息。利用键盘相关行为于验证数据，将显得有点操心过头，像一下子提醒用户打错了字，一下子提醒信息输入不完整等等。最好等到用户离开域；离开域就表示数据的输入已经完毕，这项行为则由onblur事件负责响应。

问：onblur，这个事件名称好奇怪。它是什么意思？

答：onblur其实是onfocus的相反事件。如果onfocus在元素或表单域**取得**输入焦点时触发，onblur则于域**失去**焦点时被触发。虽然“focus”在此并非意指视线的聚焦，但“blur”还是代表失去焦点。这是 JavaScript 的文字游戏，后来变得有点让人搞不清楚。总之，请记住onblur在域失去焦点时被触发。

可使用 alert 框呈现验证信息

alert 框实在很适合对用户快速地呈现信息，也正好是最简单的让用户知道表单数据有问题的通知形式。如果在处理 onblur 事件时，检测到有问题的表单数据，调用 alert() 函数就对啦！

检查表单域是否空白。

调用验证函数，检查姓名数据。

```
function validateNonEmpty(inputField) {
  // See if the input value contains any text
  if (inputField.value.length == 0) {
    // The data is invalid, so notify the user
    alert("Please enter a value.");
    return false;
  }

  return true;
}
```

指示用户修正问题数据。

Please enter a value.

OK

既然 name 域空白，应呈现 alert 框。

磨笔上阵

依照下图的输入顺序，将产生多少个 onblur事件？多少个 onchange 事件？先别烦恼 onfocus 的问题。

Enter your name: Seth Tinselman

Enter your phone number:

Enter your email address: setht@mandango

onblur 事件的数量： ....

onchange 事件的数量： ....

依照下图的输入顺序，将产生多少个 onblur 事件？多少个 onchange 事件？先别烦恼 onfocus 的问题。

Enter your name: Seth Tinselman — onblur! onchange!

Enter your phone number: — onblur!

Enter your email address: setht@mandango — onblur! onchange!

onblur 事件的数量： 3

onchange 事件的数量： 2

麻辣夜话

今晚主题：onblur 与 onchange 讨论处理坏表单数据的时机

onblur:

最近，脚本好像总在担心用户的行为。我们两个大概就是被叫来解决问题的吧！

onchange:

没错。我们实在是很忠实的好朋友，总是随时待命，让有需要的人知道元素或表单域不再是焦点，或某些数据已被改变……还可以两件事都通知哦！

onblur:

我就是想跟你讨论这个问题。据说世面上有些空数据域四处流窜，很多人都把这件事的矛头指向你。

onchange:

这项指控真的有点吓到小人了。你也知道，凡是数据有改变的状况，我一定会通知脚本，就算天塌下来我也会尽忠职守。

onblur:

确实如此。没人怀疑你在数据改变时的可靠性。问题在于，如果表单以空白数据开始，而且后来一直没有改变呢？

onblur:

onchange:

你说用户居然可以送出包含空白域的表单？根本不合理嘛！

你说得对，一点都不合理，某些用户也觉得不合理，但他们就是把矛头指向你身上了。

好吧……现在的情况是有一个刚开始为空白域的表单，但使用者跳过某些数据不输入，而送出仍然有空白域的表单……天啦、天啦！噢，我的心脏！我的心脏！

冷静一点，放轻松，没事的。这不是你的错。听着，数据没被改变的问题不是你的责任。记住，你的名字是 **onchange**。

可是我们刚刚不是才讨论了一个很可怕的情况吗？我对空白数据的管理让脚本失望了，结果整个宇宙因为这个隙缝开始崩溃了……

我们先不要想得那么远。就像我刚讲的，事情的责任不在你身上。如果脚本验证数据时还要担心域是否空白，实在不该用你触发验证代码。

呼……听起来好多了，就算它表示我再也没有用处了。等一下，我好像又开始紧张了……

振作一点啊！反应不要那么激烈。虽然你可能不适合触发验证代码，但也不表示脚本对数据的改变与否完全没兴趣。想想看允许用户编辑数据并另外存储的表单？你在这种情况下，就很适合当个把关者，只存储真正有改变的数据。

嗯，对耶。所以我还是有用的啰？

当然啦！不要再一直打击你自己了。

谢谢。我现在安心多了。

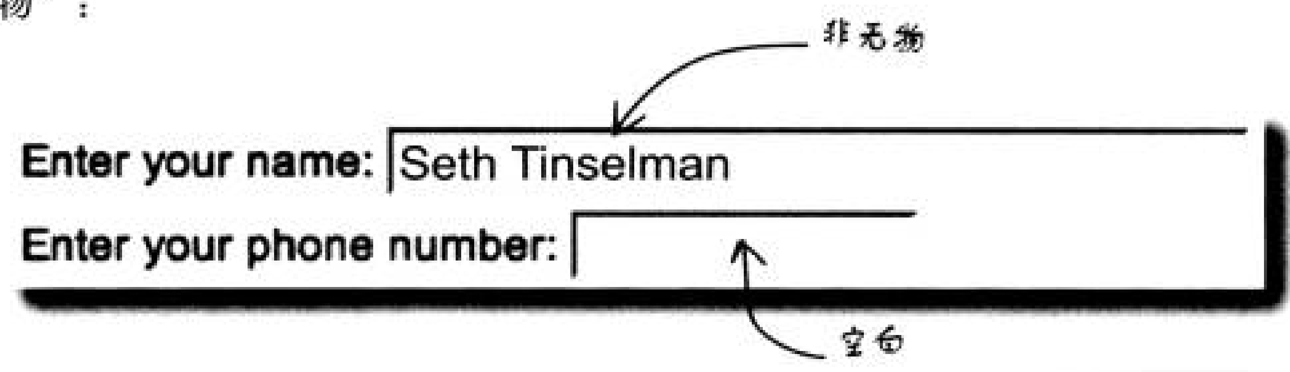
别客气。好了，我还想再多聊一点，不过我得去验证数据了……回头见！

检查……某些东西

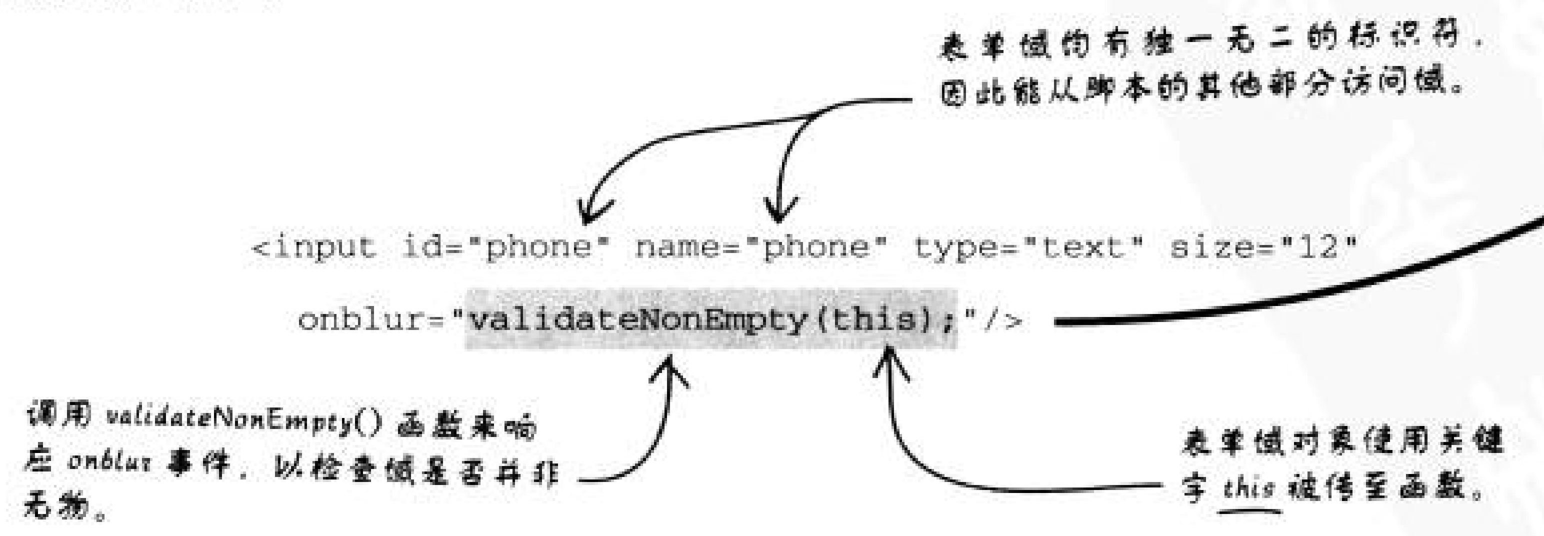
回到 Bannerocity 的生意上。Howard 知道，他的 Bannerocity 表单至少需要验证所有域都有数据。但从 JavaScript 的角度来看，这件事要从一个很奇怪的观点入手。说得更清楚一点，我们并非检查域是否**具有某物**（something），而是确认域是否**并非无物**。换句话说，“有某物”等于“非无物”。

有某物　=　非无物

如此反直觉思考的原因，就在于检查“空白”的表单域比检查“填满”的表单域更简单。所以，数据验证防御的第一路线，就是检查域是否“并非无物”：



Howard 的验证函数必须响应每个表单域的 `onblur` 事件，以执行非无物验证。如下所示：



上例代码使用了关键字 `this`，以引用表单域本身。通过把表单域当成对象传送给验证函数，提供函数访问表单域值，还有访问（存储所有表单域的）`fome` 对象的机会，有时候相当好用。

确认域"并非无物"的验证

每个表单域均有联结onblur 事件与 validateNonEmpty() 函数的相似程序代码。借由联结每个域的 onblur 事件与函数，表单的所有数据才可接受验证。

```
<input id="name" name="name" type="text" size="32"
  onblur="validateNonEmpty(this);"/>
```

表单域 name 调用验证函数，以确认是否为姓名。

表单域phone调用验证函数，以确认是否为电话号码。

```
function validateNonEmpty(inputField) {
  // See if the input value contains any text
  if (inputField.value.length == 0) {
    // The data is invalid, so notify the user
    alert("Please enter a value.");
    return false;
  }

  return true;
}
```

length 特性以字符串格式公布字符数量。

函数找不到电话号码，故呈现 alert 框并返回 false。

因为有姓名，所以函数返回 true……

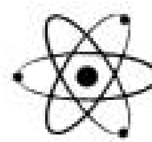
...

Please enter a value.

OK

就本例而言，validateNonEmpty() 函数的返回值并未使用。其目的在于回头与调用函数的程序代码沟通验证结果：如果数据没问题，返回 true；如果有问题，返回 false。稍后，我们将看到这些返回值如何用于确认表单数据没有问题，而后才送出表单，交给服务器处理。

非无物的验证函数确认表单域并非放着空白不管。



动动脑

你能想到任何使用 alert 方法提醒使用者"表单资料有误"的缺点吗？

不使用烦人 alert 框的验证

Howard 很快就发现，alert 框并非通知用户“数据不合格”的理想方式。他接到很多来自在线填写 Bannerocity 订单的用户的抱怨电话，大家纷纷抗议一直弹出来的 alert 框。当弹出的 alert 框干扰了在线体验时，我们多少都会因此觉得火大，数据验证也不例外，虽然此时的 alert 框是为了帮助用户而出现。

Howard 的解决方式，就是采用“被动辅助系统”，它不会使用 alert 框，因此不会干扰数据输入的流程。然而，被动的用户通知方式，需要新增几个新的 HTML 表单元素。

新的 HTML 元素，提供了呈现验证辅助信息的地方。

Enter your phone number: Please enter a value.

新加入的 HTML 辅助元素呈现优于 alert 框的大幅改善，它们不会阻挡用户，但仍能传递相同的信息。就结构而言，我们只需加入 HTML 的 <span> 标签，且其名称与代表的表单域相同。在网页的表单代码中，新加入的 <span> 标签就在输入域的下面。

传给 validateNonEmpty() 的第二个自变量，现在随着辅助元素一同传送。

```
<input id="phone" name="phone" type="text" size="12"
  onblur="validateNonEmpty(this, document.getElementById('phone_help'));" />
<span id="phone_help" class="help"></span>
```

<span> 标签原为空白，但它具有 ID，可与电话号码的表单域相关联。

样式类（class）用于设定说明文字为“红色斜体字”（不过从我们的印刷上看不出红色）。

这两个 ID 必须一致，辅助信息才能根据输入域而呈现。

有了 span 元素管理辅助说明，只剩下实际呈现辅助信息的代码还没出现。根据上例出现的 validateNonEmpty() 函数的第二个自变量，这个函数极可能负责确认用户收到了辅助信息。

更精密的非无物验证

Howard 制作的智慧被动辅助方案，出现在全新改良过的 `validateNonEmpty()` 函数里，新函数现在也为表单域设定和清除辅助信息。

helpText 对象作为传入函数的第二个自变量。

```
function validateNonEmpty(inputField, helpText) {
  // See if the input value contains any text
  if (inputField.value.length == 0) {
    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter a value.";
    return false;
  }
  else {
    // The data is OK, so clear the help message
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

首先确认 helpText 元素的存在 (helpText != null)，再把辅助信息指定给 innerHTML 特性。

在用户完成表单域输入后，清除辅助信息也是重要的一环。

好多了……没有 alert 框……比较不会打扰用户。

Bannerocity 的数据验证已有大幅改善，都要感谢新加入的被动式验证法，它依旧使用有益的 JavaScript 处方，但干净许多，至少在用户体验的流程上较为干净。

Enter ZIP code of the location: Please enter a value.
Enter the date for the message to be shown: Please enter a value.
Enter your name: Seth Tinselman
Enter your phone number: Please enter a value.
Enter your email address: Please enter a value.

缺少数据时，Bannerocity 现在改为呈现被动式辅助信息。

有了姓名数据，所以它不会出现辅助信息。

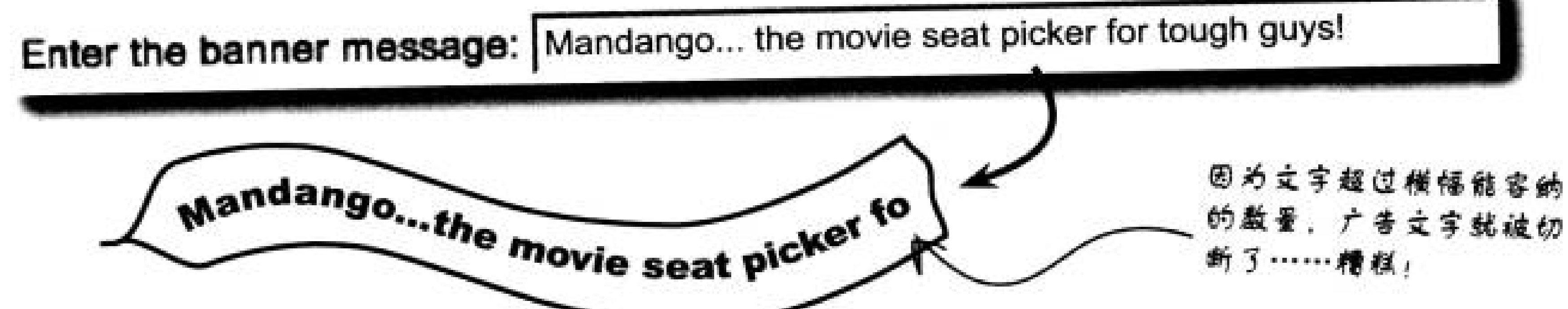
太多也是个问题

结果，非无物的验证运作确实良好，但太多数据与太少数据都可能带来问题。请看 Howard 最新收到的横幅广告订单，它突显出了 Bannerocity 订单的新问题。

尺寸很重要……

Bannerocity 遇到的困难，在于 Howard 的空中横幅只能容纳32个字符，但订单上的信息域却没有加上限制。当然，用户能够收到"尚未填入信息"的警告是件好事，但太长的信息仍然可以通过验证而不受阻挡，这对 Howard 却是天大的问题！

用户输入太多文字，但 Bannerocity 不会提示信息有问题。



因为文字超过横幅能容纳的数量，广告文字就被切断了……糟糕！

想在有限的空间呈现无限的信息，不可能成功，而且只会让客户不高兴……就像现在的 Seth 与 Jason。对信息域加上最大长度的限制，才是解决之道。最好也针对新的验证机制，特别制作一段辅助说明，确保用户知道信息长度的限制。

横幅文字需在 32 个字符以内。

加上尺寸限制后，文字刚好能填入空中横幅里。

磨笔上阵

请设计一段伪代码，呈现新的 Bannerocity 长度验证函数如何运作，确认最小和最大长度都获得验证。

....

....

....

....

请设计一段伪代码，呈现新的 Bannerocity 长度验证函数如何运作，确认最小和最大长度都获得验证。

Bannerocity横幅文字的函数自变量 minLength 与 maxLength 分别设定为 1 与 32。

If (fieldValue is shorter than minLength OR fieldValue is longer than maxLength)
  Show the help text
Else
  Clear the help text

验证数据长度

新打造的 validateLength() 函数用于检查并确认表单数据值是否遵守特定最大长度与最小长度。在 Bannerocity 的案例中，这个函数主要限制横幅文本域的长度，不过它也同时限制最小长度为 1 个字符。Howard 大概不会遇到只想看到孤零零的“L”在天上飞的客户吧！不过，重点在于确认该域的长度不会大于 32 个字符，也不会小于 1 个字符。

```
validateLength(minLength,
  maxLength, inputField,
  helpText);
```

除了以 validateLength()强制规定最小与最大长度，函数也需要验证另外两个自变量，即为输入域（inputField）与呈现辅助说明的辅助信息元素（helpText）。所以，这个函数共需4个自变量。

maxLength 输入域允许的最大长度。

Enter the banner message: Mandango...the movie seat picker for tough guys! *Please enter a value 1 to 32 characters in length.*

minLength 输入域允许的最小文本长度。

inputField 需要验证长度的输入域。

helpText 呈现辅助信息的元素。

```
<input id="message" name="message" type="text" size="32"
  onblur="validateLength(1, 32, this, document.getElementById('message_help'))" />
<span id="message_help" class="help"></span>
```

横幅信息输入域的对象。

validateLength() 函数接受 inputField 自变量的值，而后检查以确认该域的长度至少与 minLength 相同，但小于 maxLength。如果域长度值太短或太长，则以 helpText 元素呈现辅助信息。

复习要点

- 每个表单域均可作为 JavaScript 对象而被访问。
- 在表单域对象里有个 form 特性，使用数组表示了整份表单的域。
- onblur 事件于输入焦点离开某个域时发生，它是触发数据验证函数的绝佳方式。
- alert 框是种很烦人的验证问题通知方式。
- 被动式验证辅助比较直觉，也比较不会骚扰用户。
- 字符串 length 特性可显示字符串包含的字符数量。

磨笔上阵

请完成validateLength()函数代码，务必仔细注意传入函数的自变量。

```
function validateLength(minLength, maxLength, inputField, helpText) {
  // See if the input value contains at least minLength but no more than maxLength characters

  // The data is invalid, so set the help message

  // The data is OK, so clear the help message

}
```

请完成validateLength()函数代码，务必仔细注意传入函数的自变量。

```
function validateLength(minLength, maxLength, inputField, helpText) {
  // See if the input value contains at least minLength but no more than maxLength characters
  if (inputField.value.length < minLength || inputField.value.length > maxLength) {
    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter a value " + minLength + " to " + maxLength +
        " characters in length.";
    return false;
  }
  else {
    // The data is OK, so clear the help message
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

检查表单域值的最小与最大长度。

设定辅助信息，以反映域长度的问题。

如果域长度没问题，则清除辅助信息。

信息问题解决了

解决了横幅长度的问题，Howard 总算松了口气。既然买不到更长的布条，他也没有更好的解决方式，所以在 JavaScript 层次处理横幅长度问题，变成了很好的方式。至少用户下单前就知道 Bannerocity 的横幅长度有限了。

横幅信息若超过限制，现在会调用辅助信息。

Enter the banner message: Get your adventure on with Stick Figure Adventure! *Please enter a value 1 to 32 characters in length.*

没有蠢问题

问：利用 alert 框的验证到底错在哪里？大多数人不是都能分辨 alert 框与弹出式广告（pop-up ad）吗？

答： 或许大多数人都知道JavaScript的alert框不是弹出式广告，但无损于alert框被认为是高度干扰的事实。任何需要用户停下手边工作并按下其他窗口上某物的设计，都是分裂性的。虽然alert框在JavaScript程序设计中占有一席之地，但数据验证不是它出场的地方。

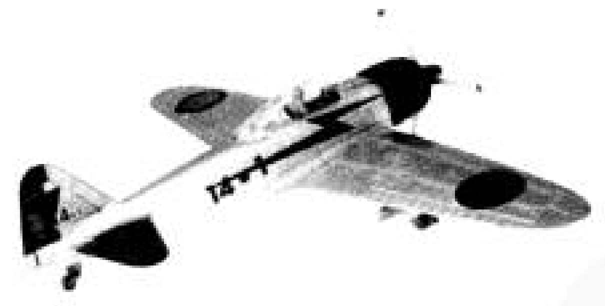
问：关于在 onblur 的部分使用 this 代表表单域，我还是觉得一头雾水。表单域就是个对象吗？或者表单本身才是对象？

答： 两个答案都对。在 HTML 元素的上下文中，关键字 this 指向代表该元素的对象。若在表单域对象的上下文中，有个 form 特性可把整份表单当成对象访问。所以，当你看到 this.form 出现在表单域的 onblur 代码中时，其实它用于引用表单（对象）本身。

Bannerocity里this.form的用途，在于取得（与某个表单相关的）辅助信息元素。还记得this.form是对form对象的引用吗？它也是个包含所有表单域的关联式数组（associative array）。所以，假设域名称my_field时，你可以把数组加入访问域的代码中，如this.form["my_field"]，即可快速访问。你也可使用 getElementById()，但利用表单的方式比较便捷。

问：当辅助信息与输入域相关时，它们拥有的 name 与 id 属性分别是什么呢？

答： 辅助信息元素的 id 属性，需根据相关输入域的 id/name 而定，但又不完全一样。更精确地说，辅助信息的 ID 会使用输入域的 ID，只是在后面加上 _help。这种命名惯例，是为了**在输入域与呈现该域的辅助信息的元素间**，建立清楚且一致的连接。事实上，你可以随意命名辅助元素的 ID，只要名称独一无二而且能传入验证函数就好了。



问：在验证函数中，为什么于数据验证无误后，清除说明文字是必要行为呢？

答： 请记得辅助信息的重点，是在有问题时协助用户。如果输入表单的数据通过检查，也就是没有问题，因此不需要呈现辅助信息。既然辅助信息可能已经因为前一次的验证而出现，最好每次通过域的验证时，都清除辅助信息。

问：如果“辅助信息”并未作为自变量传给验证函数，会发生什么事吗？

答： 脚本一而再、再而三地搜索失落的元素，网页过热超载，浏览器爆炸，剩下一团灰烬……开玩笑的啦！根据设计，如果辅助信息自变量并未用于验证函数，Bannerocity的被动式辅助系统就会静静退场。所以输入域的辅助信息只是不会出现而已。这个方式的好处，就是能随我们的意愿尽量出现或尽量不出现。即使就各个域的角度而言，你也不需强制为表单中的每个域加上辅助信息。

检查htmlText自变量是否non-null的验证代码，即允许了辅助信息元素的可选择性。如果辅助信息元素不为null，表示元素存在，能呈现辅助信息；否则表示没有元素，什么事都不做。

问：HTML 的表单域不是有个 size 属性，难道它没有限制域长度吗？

答： HTML的size属性只限制表单域在网页上的实际大小——与输入的数据量的限制没有关系。举例来说，Bannerocity的邮政编码域的size属性设为5，表示这个域在网页上的尺寸大小刚好装得下 5 个字符。使用HTML属性maxlength的确可以限制文字的实际长度，但它没有相对的minlength属性。验证函数对于控制输入域的字符长度提供了最大灵活性，虽然邮政编码应该不只确认数据为 5 个字符，还应确认数据是 5 个数字。也许 Howard 应该考虑为 Bannerocity 再加上一些功能……

对的横幅，错的地方

虽然 Howard 尽量做到最好的验证，但他的在线表单还是继续带来问题。这一次是邮政编码输入有误的问题，结果 Howard 在错误的地方盘旋了好几个小时。或许还有比 Howard 浪费时间更糟的状况，他的客户 Duncan 错失了不少销售甜甜圈的机会。

Enter ZIP code of the location: 7412I

Duncan's Donuts...only the best!

74121

74122

74123

74124

74125

74126

74127

74128

74129

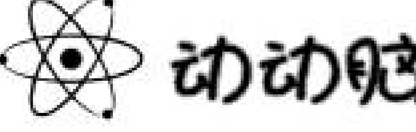
74130

这里输入了大写字母I，而不是数字9，但它被误看成1了。

哈哈哈！我自己的计划都没这么完美…………Duncan 的横幅出现在错误的区域。

还记得 Frankie 吗？他是卖热狗的人，跟 Duncan 争夺早餐市场。

在这里的状况中，人为错误加到数据错误上，造成一团混乱。顾客想按 9，不小心按成下面的 I。Howard 的肉眼则把 I 误判为1，结果就飞到奇怪的地方做横幅广告了。



你会如何验证邮政编码?

验证邮政编码

Howard 遇到的问题，与邮政编码的输入不正确有关。美国邮政编码的最简形式由 5 个数字构成。所以，验证邮政编码可以只是确认用户输入了 5 个数字……不多也不少。

磨笔上阵

请完成下列验证邮政编码的 validateZIPCode() 函数，确保输入的邮政编码都是 5 个字符长，而且都是数字。

```
function validateZIPCode(inputField, helpText) {
  // First see if the input value length is anything other than 5

  if (..............................) {

    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter exactly five digits.";

    ..........................
  }
  // Then see if the input value is a number

  else if (..............................) {

    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter a number.";

    ..........................
  }
  else {
    // The data is OK, so clear the help message
    if (helpText != null)
      helpText.innerHTML = "";

    ..........................
  }
}
```

请完成下列验证邮政编码的validateZIPCode()函数，确保输入的邮政编码都是5个字符长，而且都是数字。

确认邮政编码字符串的长度是否不等于5（个字符）。

```
function validateZIPCode(inputField, helpText) {
  // First see if the input value length is anything other than 5
  if ( inputField.value.length != 5 ) {
    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter exactly five digits.";
    return false;
  }
  // Then see if the input value is a number
  else if ( isNaN(inputField.value) ) {
    // The data is invalid, so set the help message
    if (helpText != null)
      helpText.innerHTML = "Please enter a number.";
    return false;
  }
  else {
    // The data is OK, so clear the help message
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

既然邮政编码的长度不是5，返回false。

isNan()函数可检查值是否"不为数字"。

既然邮政编码不是数字，返回false。

返回true，指出邮政编码的验证没问题了。

注意！

假设邮政编码均为数字并非绝对安全。

如果网络表单能够接受美国以外地址的邮政编码，纯数字的验证就不是什么好主意了。因为很多国家的邮政编码或许包含字母数字的混合。另外，完整的美国邮政编码应该包含9位数，格式为#####-####，此时，连字符（-）将使邮政编码数据不为数字。

坏数据不该抵达服务器。

哎呀！尽管有一卡车好意的提醒，但如果用户可以按下按钮、避开验证而送出表单，做了这么多数据验证代码也是枉然啊！Bannerocity 的致命缺点，就是没在送出表单时要求验证，所以坏数据目前还是能被送进服务器。

如果用户还是可以选择送出包含坏数据的表单，这些数据验证就只是白费工夫。

Bannerocity – Personalized Online Sky Banners

BANNEROCITY

Enter the banner message: Please enter a value 1 to 32 characters in length.
Enter ZIP code of the location: Please enter exactly five digits.
Enter the date for the message to be shown: Please enter a value.
Enter your name: Please enter a value.
Enter your phone number: Please enter a value.
Enter your email address: Please enter a value.
Order Banner
Done

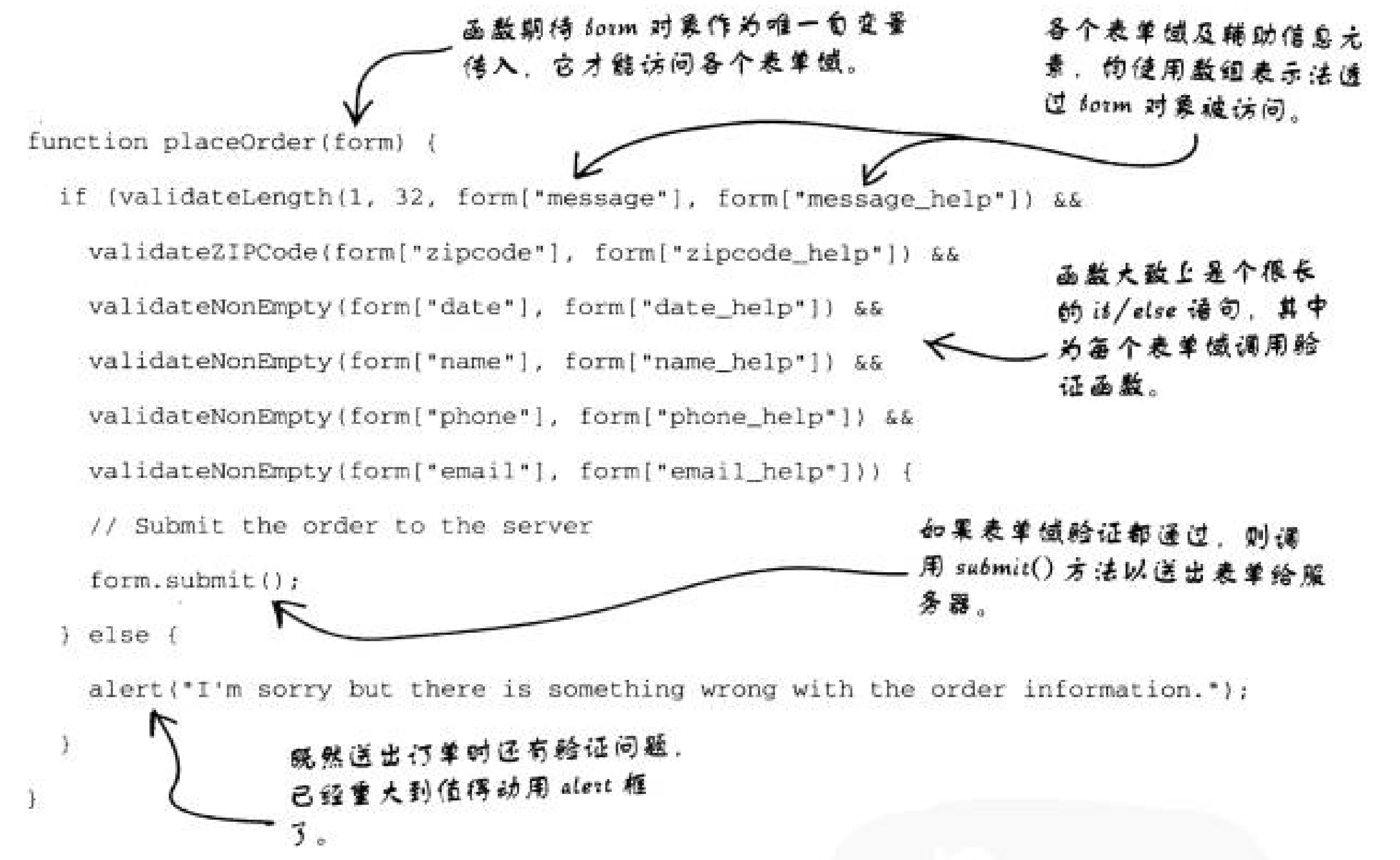
“Order Banner”按钮需要一些验证，才能把表单数据送往服务器。

一个真正可靠的应用程序也会验证服务器上的数据，以策安全。

Bannerocity 需要其他函数，它的工作就是验证所有表单域，才能送出表单给服务器处理。我们自定义的`placeOrder()`函数与“Order Banner”按钮相联系，它将于完成订单前受到调用并最后一次验证表单。

```
<input type="button" value="Order Banner" onclick="placeOrder(this.form);" />
```

placeOrder()函数靠过来



```
function placeOrder(form) {
  if (validateLength(1, 32, form["message"], form["message_help"]) &&
    validateZIPCode(form["zipcode"], form["zipcode_help"]) &&
    validateNonEmpty(form["date"], form["date_help"]) &&
    validateNonEmpty(form["name"], form["name_help"]) &&
    validateNonEmpty(form["phone"], form["phone_help"]) &&
    validateNonEmpty(form["email"], form["email_help"])) {
    // Submit the order to the server
    form.submit();
  } else {
    alert("I'm sorry but there is something wrong with the order information.");
  }
}
```


没有蠢问题

问：placeOrder()函数如何控制表单是否送往服务器？

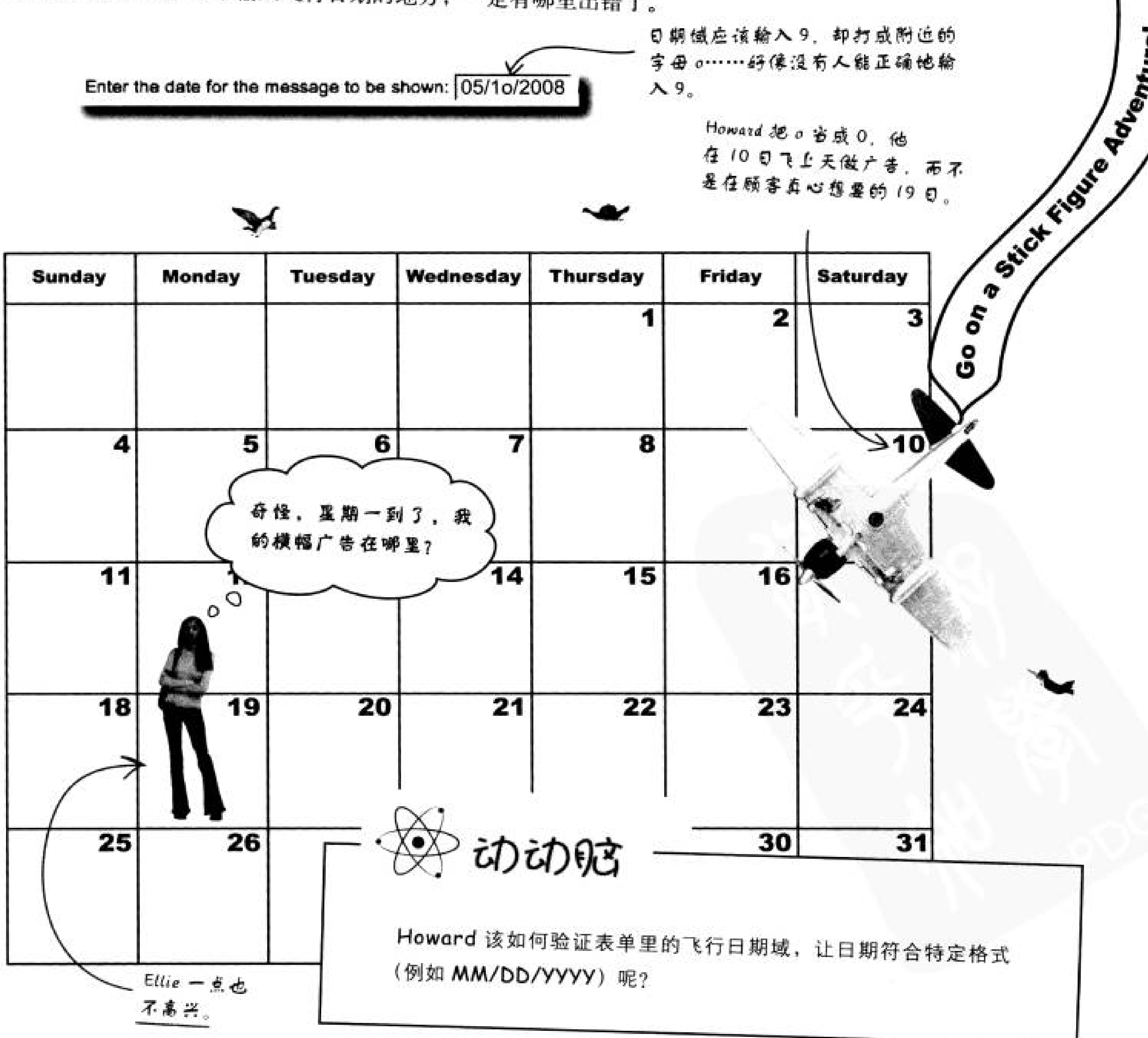
答：首先，函数中的if/else语句被架构为验证表单中的每个域，也就是说，如果任何表单数据不合格，则执行else子句（clause）。else子句只负责调用alert()函数，所以函数如果走到else子句，不会再发生其他事情。另外一方面，如果数据验证没问题，则调用form对象的submit()方法，用于送出表单给服务器。所以送出表单的动作由是否调用表单的submit()方法所控制，这个方法是JavaScript版的“submit”按钮。

问：我还以为alert框对验证有害，它为什么又出现了？

答：很多时候是这样没错，但这里真正的问题，则是何时可以干扰页面流程，呈现弹出式信息（alert），要求用户阅读信息并按下“OK”。因为只在用户想要送出订单时才会按下“Order Banner”按钮，值得确认用户知道数据有问题，而本例的问题严重到值得用户alert框。别忘记了，被动式辅助信息仍会呈现在网页上，引导用户修正问题。

数据验证……时机就是一切

很可惜，Howard 对邮政编码与表单送出验证的修改，只能带来一时的轻松，因为现在又出现全新的问题。由于有了邮政编码的验证，他不再飞过错误的地方，但他有时候却在错误的日期广播横幅广告，这点好像更糟糕。关于输入飞行日期的地方，一定有哪里出错了。

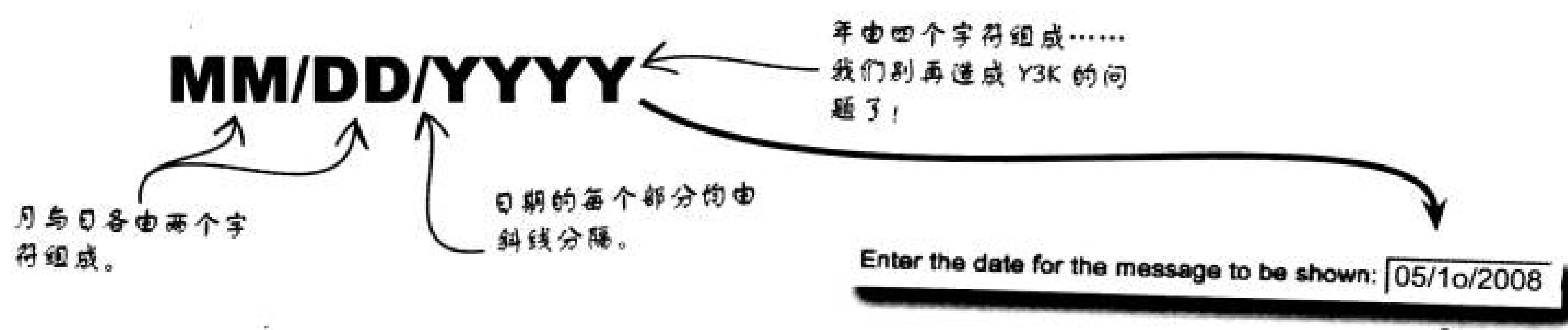


动动脑

Howard 该如何验证表单里的飞行日期域，让日期符合特定格式（例如 MM/DD/YYYY）呢？

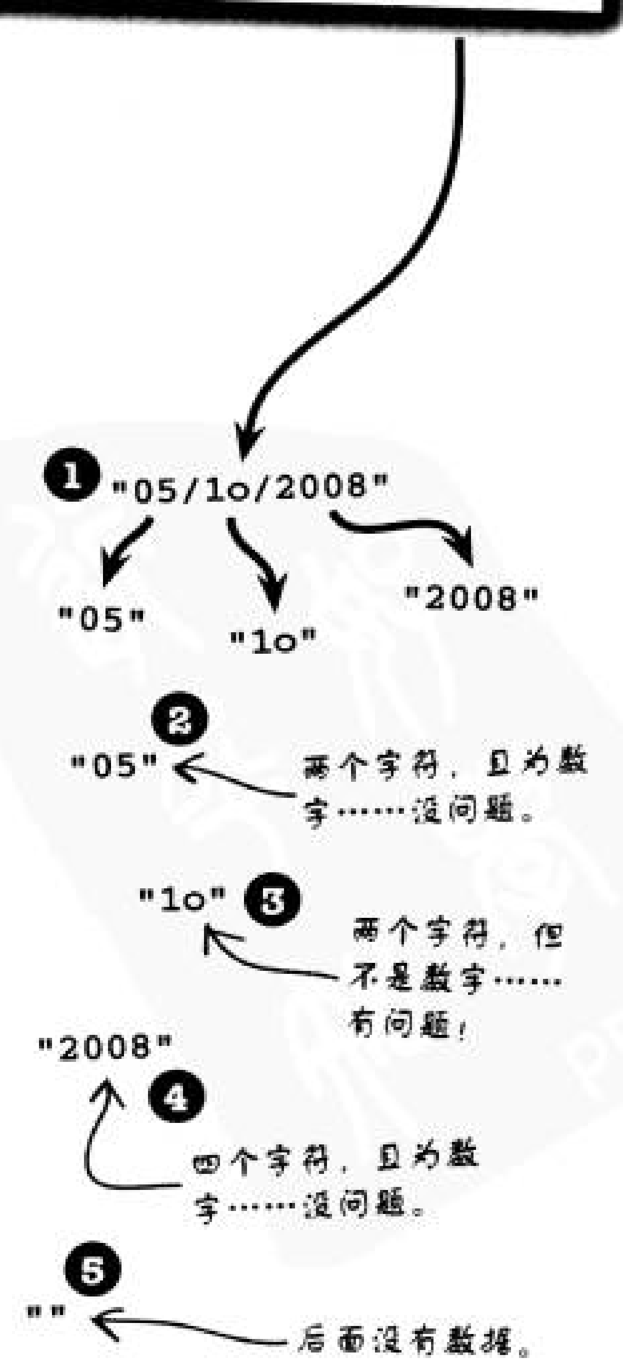
验证日期

很明显，Howard 无法只依靠用户是否输入日期数据的检查，他还需要实际确认输入的日期是否合格。验证日期的关键，取决于指定日期格式，然后强制实行。有种常见的日期格式：先以两位数表示日，再以两位数表示月，最后以四位数表示年，其间以斜线区隔。



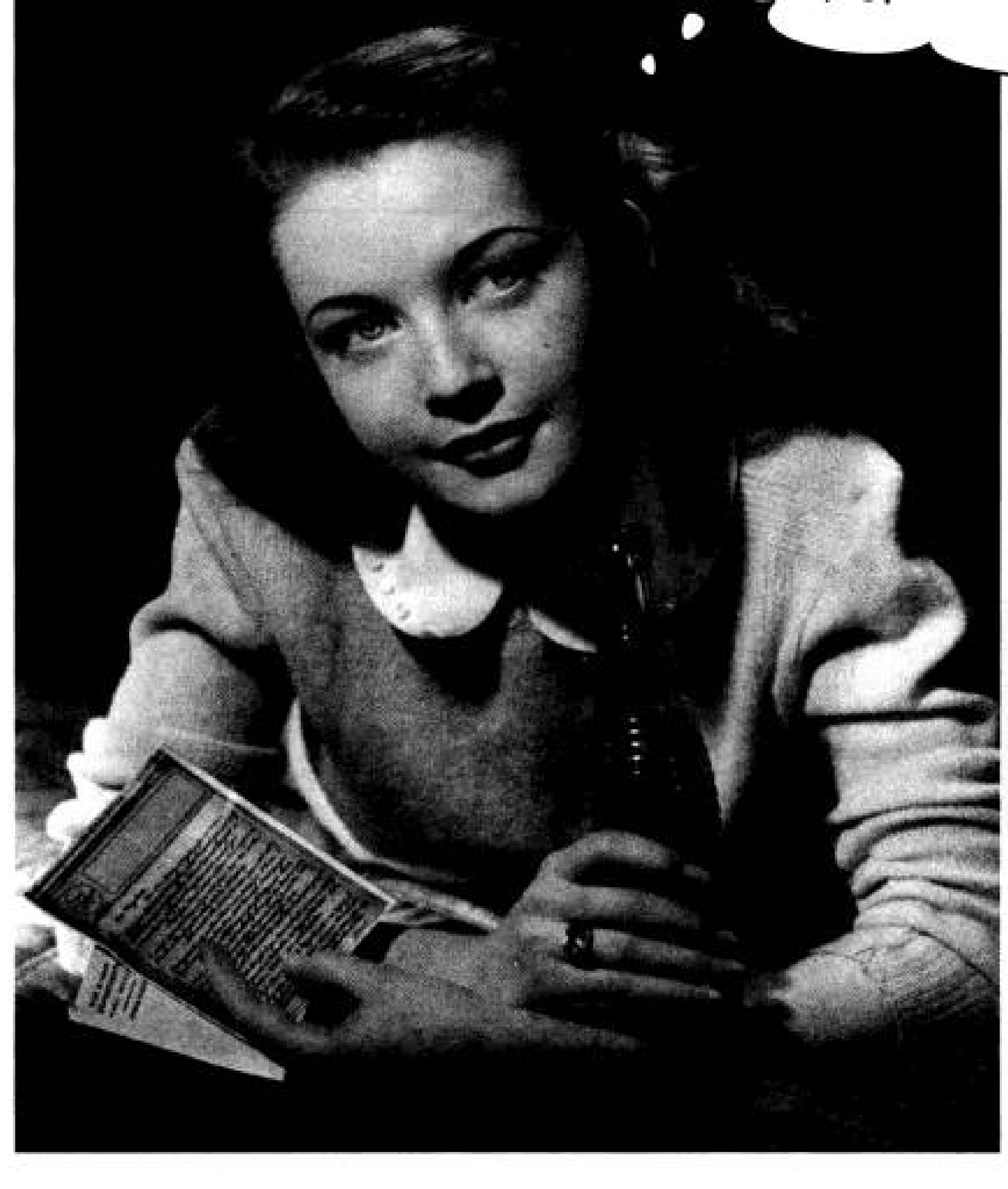
分解日期格式是最简单的部分……设计出验证日期是否符合格式的代码，才是困难的开始。有些很强大的字符串函数能够根据特定字符分解字符串，例如根据斜线（/）分解。但是，分解字符串，再确认各个部分均为数字且为指定长度，这项努力实在太复杂了。就像情况最极端的邮政编码验证挑战。

我们逐步分析日期验证函数的运作方式：



1. 分解表单域值成为许多组子字符串（substring），以斜线作为分解字符串的基础。
2. 分析“月”子字符串，确认刚好只有两个字符，而且是数字。
3. 分析“日”子字符串，确认刚好只有两个字符，而且是数字。
4. 分析“年”子字符串，确认刚好只有四个字符，而且是数字。
5. 忽略其他跟随在第二个斜线后的数据。

虽然上述步骤还不致于成为编程噩梦，但对于验证一段简单的格式而言，好像太复杂了点。


如果有个比分解字符串更好的日期验证方法，该有多好……就让闺中少女稍微幻想一下吧！

正则表达式一点都不“正规”

JavaScript 刚好有种非常强大的内置工具 —— **正则表达式**（regular expression），专门设计用于匹配（match）文本模式（pattern）。正则表达式可用于创建模式，而后应用于文本字符串，搜索匹配的部分……就像指认出嫌犯一样！不过我们的字符合作多了。

正则表达式用于匹配文本模式。

上例的模式叙述了某个人的外表属性，然后匹配出实际人物。正则表达式也有相同功用，它能匹配字符串的模式。

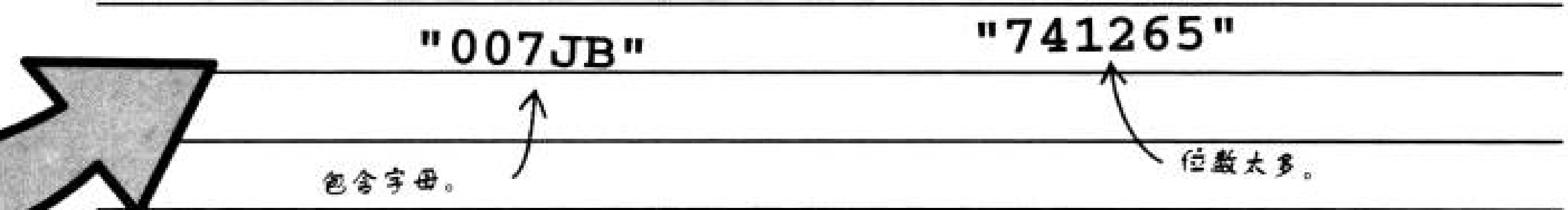
正则表达式定义用于匹配的模式

就像警方指认嫌犯时可能用到身高、发型或其他外表特征，文本模式也有特定的字符顺序，例如一行 5 个数字。等一下，听起来好耳熟……邮政编码……吗？

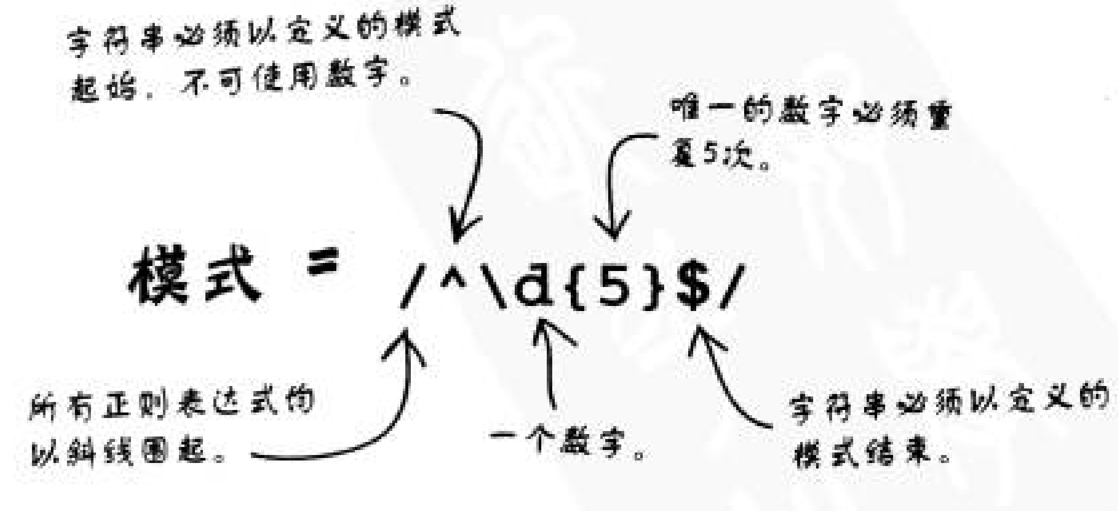
模式 = #####

模式包含一个只有 5 个数字的序列。

匹配到了！



很可惜，把 5 位数的邮政编码转换为正则表达式不太符合直觉。因为正则表达式在描述文本模式时，会仰赖一种非常简洁但又有点难解的语法。你看右侧的正则表达式范例，很难立刻看出来它用于验证邮政编码吧！



如果正则表达式让你看得头昏脑胀，别惊慌。

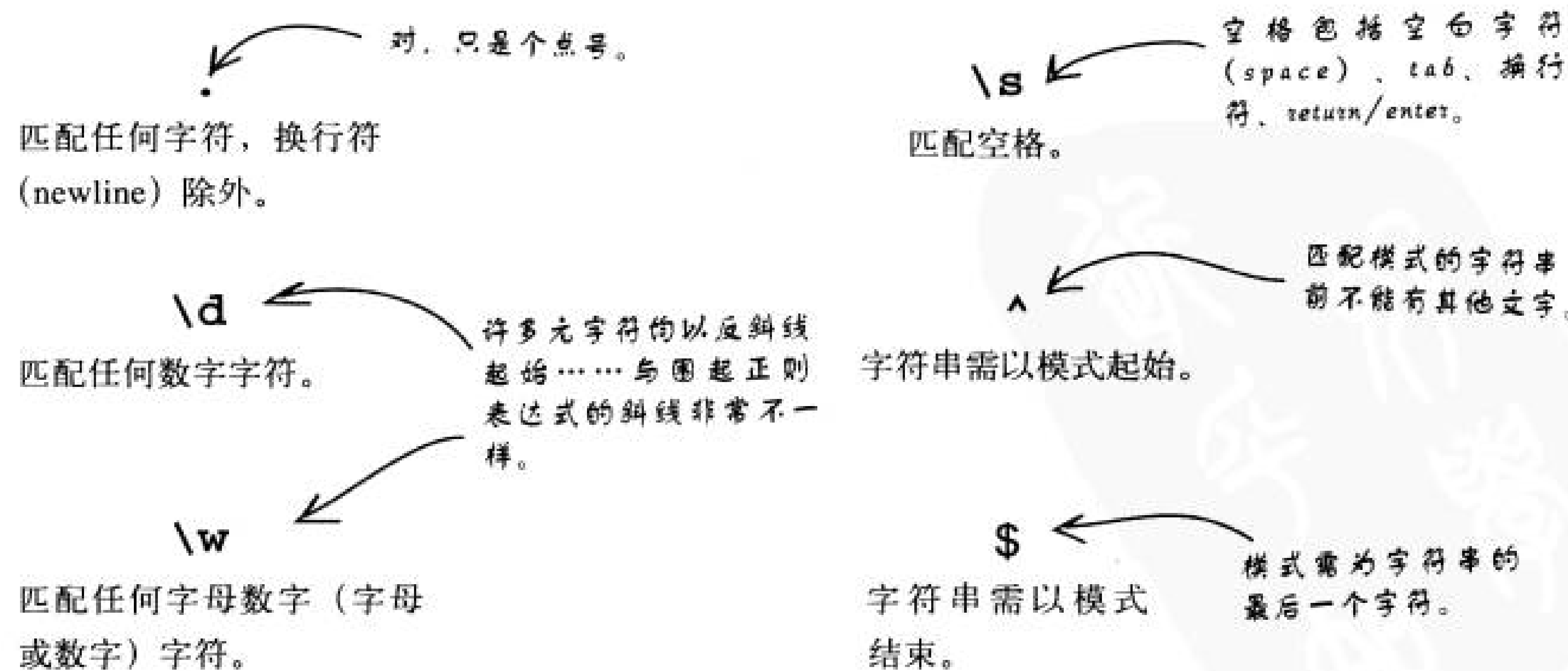
接下来我们还会讨论更多验证实例，你会慢慢了解正则表达式。

揭开正则表达式的面纱

创建正则表达式，有点像创建字符串字面量（string literal），只不过正则表达式出现在一对斜线（//）里，而不是出现在一对引号里。

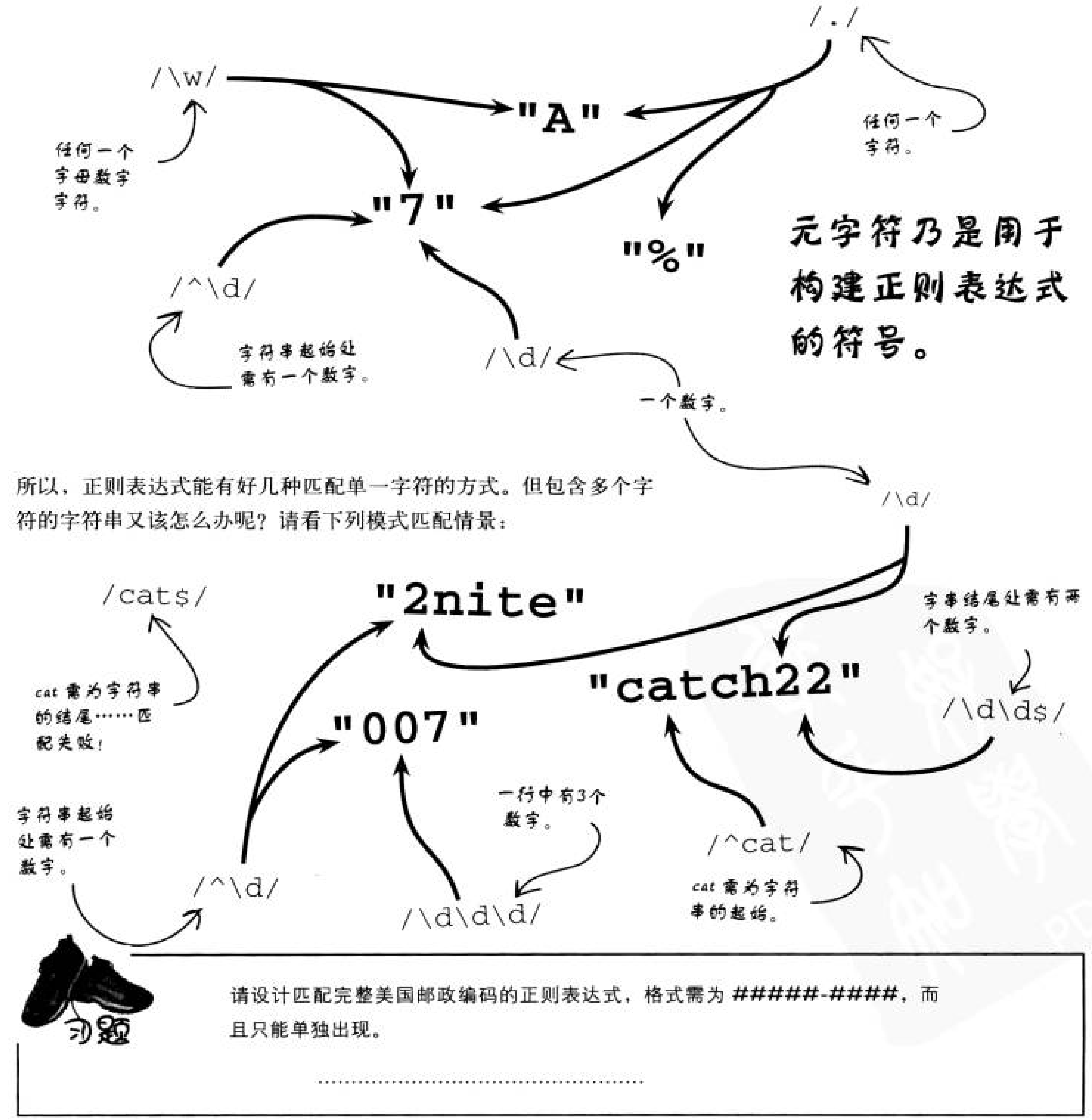
正则表达式总是以斜线起始和结束。

讲到正则表达式本身，有一组称为**元字符**（metacharacter）的特殊符号，用于连接字母与数字，创建高度描述性的文本模式。好消息是：创建实用的正则表达式时，不需要了解正则表达式“语言”的每个细微差异。以下是一些常用正则表达式元字符：



虽然上述元字符的描述都很准确，但元字符放在模式上下文中，其实更容易理解……

元字符不只表示一个字面量字符

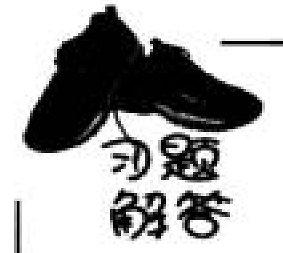


所以，正则表达式能有好几种匹配单一字符的方式。但包含多个字符的字符串又该怎么办呢？请看下列模式匹配情景：

习题

请设计匹配完整美国邮政编码的正则表达式，格式需为 #####-####，而且只能单独出现。

....



请设计匹配完整美国邮政编码的正则表达式，格式需为 #####-####，而且只能单独出现。

字符串必须以指定模式开始。

以另一个斜线结束正则表达式。

字符串必须以指定模式结束。

```
/^\d\d\d\d\d-\d\d\d\d$/
```

以斜线起始一段正则表达式。

比对出恰好一行 5 个数字。

连字符没有特殊意义……只是区分邮政编码而已。

匹配出恰好一行 4 个数字。

深入正则表达式：限定符

不是元字符的任何文字将于正则表达式里“照样匹配”。也就是说，/howard/ 可找出任何包含“howard”的字符串。另外，还有另一种正则表达式结构，称为**限定符**（quantifier），可进一步把模式调整得更好。限定符前为子模式（sub-pattern），限定符即应用在子模式，并控制子模式出现在模式里的次数。

`*`

限定符前的子模式必须出现 0 或多次。

`{n}` ← 控制子模式可以出现的次数。

限定符前的子模式必须出现恰好 *n* 次。

`+` ← 子模式必须出现，可出现任意次数。

限定符前的子模式必须出现 1 或多次。

`?` ← 子模式为可选的，若出现也只能出现一次。

限定符前的子模式必须出现 0 或 1 次。

虽然括号在技术上不算限定符，但常用于归类子模式，就像平常区分数学表达式一样。

`()`

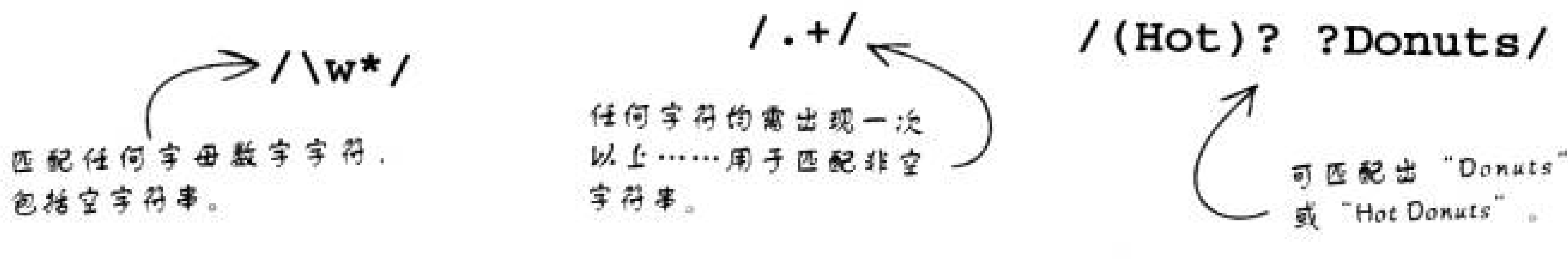
集合字符或/和元字符，成为子模式。

模式限定

限定符使得正则表达式设计比起只有元字符时更为精确。不再直接重复子模式，限定符能指定子模式出现的精确次数。请看使用限定符后的邮政编码范例：

限定符控制子模式出现于正则表达式里的次数。

利用元字符与限定符，可以创建相当强大、几乎可匹配任何字符串内容的正则表达式。



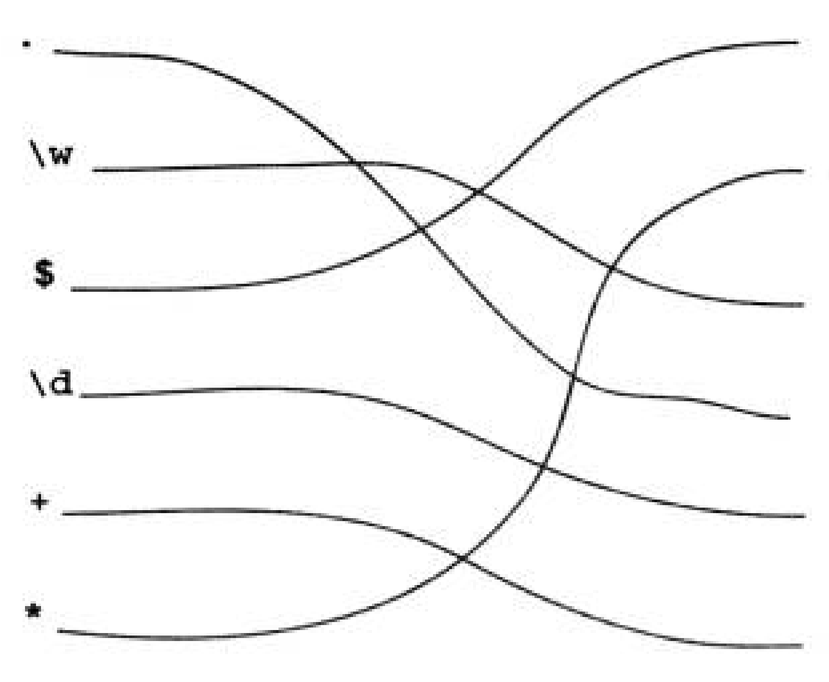
猜猜我是谁

请把每个正则表达式元字符或限定符，与它在模式中的功用连起来。

.	字符串必须以指定模式结束。
\w	子模式为**可选的**，且可出现任意次数。
$	匹配任何字母数字字符。
\d	匹配任何字符（newline 除外）。
+	匹配任何数字。
*	子模式**必须出现**，且可出现任意次数。

猜猜我是谁

请把每个正则表达式元字符或限定符，与它在模式中的功用连起来。



字符串必须以指定模式结束。

子模式为**可选的**，且可出现任意次数。

匹配任何字母数字字符。

匹配任何字符（newline 除外）。

匹配任何数字。

子模式**必须出现**，且可出现任意次数。

没有蠢问题

问：正则表达式是个字符串吗？

答：不是。你可以把正则表达式当成对字符串的**描述**，至少当成对部分字符串的描述。正则表达式与字符串紧密相关，且用于匹配出现在字符串中的文本模式，但正则表达式本身不是字符串。

问：正则表达式可以应用在其他类型的数据上吗？

答：不行，正则表达式单纯设计用于匹配文本字符串内的字符模式，所以只能应用在字符串上。但这项限制并未减低正则表达式处理"只用字符串太困难"的复杂文本匹配任务时的极度好用程度。

问：如果想匹配元字符，例如 $，会发生什么事？

答：与 JavaScript 字符串相似，正则表达式中具有特殊意义的字符能以反斜线转义。所以想在正则表达式里匹配 $ 时，你必须把原本的 $ 特别改写成 \$。这项规则适用其他在正则表达式里具有特殊意义的字符，例如 ^、*、+等等。任何没有特殊意义的字符则可直接放入正则表达式，不需特殊格式。

问：正则表达式与数据验证有什么关系吗？我们本来不是在验证 Bannerocity 的日期域吗？

答：耐心一点，年轻的绝地武士。很快就会用到正则表达式了。是的，这趟小小的正则表达式式岔题之旅，乃是为了寻找验证复杂格式数据的方式，例如日期或电子邮件地址。Bannerocity 的数据格式前台还需要很多协助，所以有很多应用正则表达式的机会。请各位耐心地再等候一下。

问：正则表达式如何用于 JavaScript里？

答：我们马上就要谈到了……不骗你！正则表达式以对象表现于 JavaScript 中，该对象支持数个方法，以利使用正则表达式匹配字符串内的模式。

模式大师真情指教

本周主题：

神秘难解但魔力强大的正则表达式。

Head First：您……就是轰动万教、惊动武林、金光闪闪、瑞气千条，据说可以在字符串内做搜索，也能辨识模式的神秘客吗？

正则表达式：没错，我算是某种代码中介客，能够仔细搜索文本字符串，并立刻找出模式。中情局实在需要我这种人才……不过他们没有回我的电话。

Head First：唉……你对间谍这一行有兴趣哦？

正则表达式：不是啦！我只是喜欢找出文本间的模式。事实上，我**就是**模式、任何模式都是我。只需给我一些参数，说明你在寻找的模式，我就能找出来，至少能让你知道字符串中是否包含你要的模式。

Head First：听起来很不错，不过，`String`对象的`indexOf()`方法不是已经能处理字符串搜索了吗？

正则表达式：拜托，你刚才真的提到“那个东西”了吗……那个业余的根本不知道何为模式。听好，如果你需要非常简单、简单到不行，只是在字符串中寻找“滥竽充数”一词的搜索功能，你可以用`indexOf()`。否则，你很快就会发现`indexOf()`完全无法严格地分析字符串。

Head First：字符串搜索不也是模式匹配的一种形式吗？

正则表达式：是啊……打开冰箱门也是一种运动，因为有人还没看过奥运的竞技……我的重点是，简易字符串搜索其实是极度简化到最简化的模式匹配——此时的模式只是个静态词汇或短语。你想想看日期或网站 URL 的结构，这些才是真正的模式，因为虽然它们遵守严格格式，但搜索的详细规范却非静态。

Head First：我开始懂你的意思了。“模式”是一组对文本的叙述，这段文本可能出现在字符串中，但“模式”不见得必为文本本身。

正则表达式：就是这样啦！就像我请你看到“头发短短的、身材高高的、没戴着眼镜的人”时通知我一样。这是一段对外表的叙述，但并非真的是个人。如果有个叫 Alan 的人符合叙述，而且正好走进来，就可以说他匹配模式。但可能也有很多人符合这段叙述。如果没有模式，我们就无法根据模式找人，而必须寻找确定的人物。所以，使用 `indexOf()` 搜索特定文本与使用我匹配模式间的差异，就如同直接寻找 Alan 与寻找一名身材高挑、短发、没戴眼镜的人之间的一样。

Head First：这样听来就清楚多了。但是，模式匹配如何应用在数据验证上呢？

正则表达式：这个嘛，验证数据主要在于确认数据符合“某种预先构想的格式”，也称为模式。所以我的工作就是拿到一组模式，并检查字符串是否相符。如果符合，则数据被视为合格的（valid），否则即为有问题的数据。

Head First：匹配不同数据时，也会用到不同的正则表达式吗？

正则表达式：当然啊。在使用我们正则表达式验证数据时，找出成功塑造数据格式的正则表达式，其实才是最费功夫的工作。

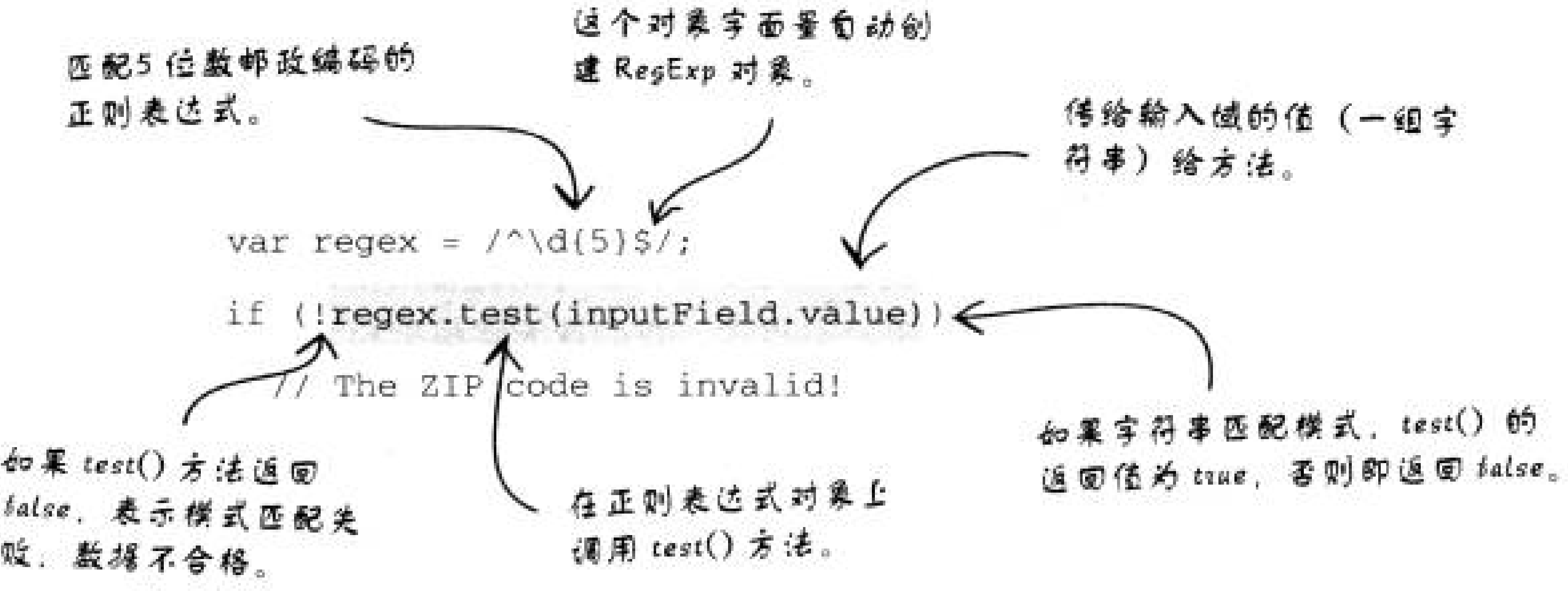
Head First：真是太有趣了。非常感谢你今天为我们说明你在数据验证中扮演的角色。

正则表达式：小事一桩。我常常需要自我解释……这大概是我的行为模式吧！

利用正则表达式验证数据

虽然，单纯为了找出模式而创建正则表达式好像非常有趣，但我们对正则表达式另有迫切的需要，我们需要它协助验证 Bannerocity 的日期域，并把 Howard 带回天空。JavaScript 里的正则表达式由RegExp对象表示，其中包含使用正则表达式验证数据的关键 —— test()方法，它检查字符串里是否存在指定的模式。

RegExp 对象的 test() 方法用于检查字符串是否包含正则表达式模式。



虽然也能在每个不同的验证函数内调用 test() 方法，但有个机会可创建通用的正则表达式验证函数，可供更专门化的函数调用以执行通用性的验证工作。下列步骤将由通用的validateRegEx()函数负责：

1. 根据当作自变量传入的正则表达式，对同样传入的字符串执行检查。
2. 如果没有找到模式，把辅助信息指定给（作为自变量）传入的 helpText，然后返回 false。
3. 如果匹配模式，清除说明信息并返回 true。

万事俱备，只欠函数代码。事实上，这个函数代码已有大部分出现在其他验证函数里，所以 validateRegEx() 其实同时牵涉到代码再利用，以及创建全功能的正则表达式验证工具。

```
function validateRegEx(regex, inputStr, helpText, helpMessage) {
  // See if the inputStr data validates OK
  if (!regex.test(inputStr)) {
    // The data is invalid, so set the help message and return false
    if (helpText != null)
      helpText.innerHTML = helpMessage;
    return false;
  }
  else {
    // The data is OK, so clear the help message and return true
    if (helpText != null)
      helpText.innerHTML = "";
    return true;
  }
}
```

正则表达式、输入的字符串、辅助信息文字、辅助信息元素，均以自变量形式传入。

根据正则表达式检验输入的字符串。

如果检验失败，表示数据不合格，所以要呈现辅助信息。

如果数据检查没问题，则清除辅助信息。

磨笔上阵

请撰写validateDate()的函数代码，其中调用validateNonEmpty()与validateRegEx()以验证Bannerocity表单的日期域。提示：这个函数接受两个自变量，日期域值以及相关的辅助信息元素。

磨笔上阵解答

请撰写validateDate()的函数代码，其中调用validate-NonEmpty()与validateRegEx()以验证Bannerocity表单的日期域。提示：这个函数接受两个自变量，日期域值以及相关的辅助信息元素。

先调用 validateNonEmpty() 函数，确认域并非空白。

```
function validateDate(inputField, helpText) {
  // First see if the input value contains data
  if (!validateNonEmpty(inputField, helpText))
    return false;

  // Then see if the input value is a date
  return validateRegEx(/^\d{2}\/\d{2}\/\d{4}$/, inputField.value, helpText,
    "Please enter a date (for example, 01/14/1975).");
}
```

传递日期正则表达式给validateRegEx()函数。

因为斜线在正则表达式里具有特殊意义，需使用反斜线转义。

日期正则表达式使用元字符与限定符维持MM/DD/YYYY的格式。

能让用户只输入2位数的年份，或许也不错。

到了2100年，很难说你的脚本是否还在使用。

Y2100还早……

既然我们距离下个世纪还很远，或许用户只输入2位数的年份也没什么问题。现实而言，现在写好的任何JavaScript代码不太可能存活90年，造成世纪交替时的呈现问题。Howard短暂地考虑了前瞻未来的严格验证版Bannerocity（维持输入4位数年份），然后决定等到下个世纪再来承担后果。

问：为什么 `validateDate()` 需要调用 `validateNonEmpty()` 函数？正则表达式不是也考虑到了空数据吗？

答：是的，正则表达式确实天生就会验证是否为空数据，移除非空验证的部分，日期验证一样能正常运行。不过，先检查日期是否输入，网页对用户会显得比较直观，因为页面上提供了各个验证问题的辅助说明。如果没有输入数据，它显示的信息将不会是输入不合格数据时的信息。感觉上，我们的被动式辅助系统似乎能引导用户完成输入表单的过程。这种微妙的可用性强化，或许值得稍微多用一点代码。

问：如果我真的想考虑脚本的未来呢？会是个问题吗？

答：不会，完全不会。企图预先考虑未来的需求，并设计能够适应这些需求的程序代码，很少成为问题。以Bannerocity为例，要求填入4位数年份的日期域，绝对比2位数的版本更能高枕无忧。也请记得，如果你真的想要点手段，也可以让用户只输入2位数，但在背后加上代表世纪的前2位数。所以，表单呈现效果是 2 位数的年份，但实际存储的日期却是4位数的年份。

注意！

日期不只MM/DD/YYYY 一种格式。

假设所有用户均以 MM/DD/YYYY 的格式输入日期，不见得安全。世界上还有很多地方输入日期时的格式与之相反，他们采用的格式是 DD/MM/YYYY。

匹配最少次数与最多次数

限定符 {} 可接受数字，决定子模式出现在字符串里的次数。这个限定符还有另一个版本可接受两个数字，分别指定子模式出现在字符串里的最少次数（min）与最多次数（max）。这个版本为子模式的出现提供良好微调的便利方式。

```
/^\w{5,8}$/
```

有些密码可选择 5～8 个字符，此时最适合限定符{}。

重新设计`validateDate()`函数使用的正则表达式，让它接受 2 位数和 4 位数的年份。

....

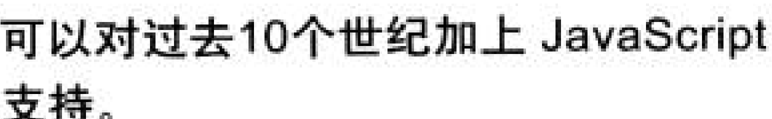
重新设计 `validateDate()` 函数使用的正则表达式，让它接受 2 位数和 4 位数的年份。

```
/^\d{2}\/\d{2}\/\d{2,4}$/
```

加入 min/max 的 {} 限定可设定日期里允许出现的年份位数。

等一下。新的日期验证代码好像也能让 3 位数的年份通过耶？没道理啊……

日期的正则表达式也让 3 位数的年份通过匹配……不太妙！



可以对过去10个世纪加上 JavaScript 支持。

既然不支持过去，自然没有理由允许用户输入年份仅到百位数的日期。事实上，有可能的话，根本不需要让用户在过去的时间下订购空中横幅广告。所以，从验证代码中减去 3 位数的年份是项重要的修补，也将协助 Howard 避免被新浮现的 Bannerocity 数据输入问题攻击。

复习要点

- 正则表达式以**模式**匹配字符串里的文本；正则表达式需以斜线围起。
- 除了一般文本，正则表达式亦由元字符与限定符建立，可对文本模式的匹配方式提供精细的控制。
- JavaScript透过内置RegExp对象而支持正则表达式，但正则表达式通常建立为literal，所以很少见到这个对象。
- RegExp对象的test()方法用于对字符串应用正则表达式模式的检测。

删除3位数的年份，该使用这个……或那个

另一个在正则表达式工具箱中非常好用的元字符，则是选替（alternation），它跟 JavaScript 的 OR 逻辑运算符非常像。但与 JavaScript 的 OR 逻辑运算符不一样，选替只用到一个 |，而且它的功用是让模式指定一连串可供选择的子模式。换句话说，这些可供选择的子模式中，只要有一个匹配成功，就算模式匹配成功。这一点很像逻辑 OR 运算符，因为它的意思基本上也是"这个，或这个，或这个……"

`this|that`

选替（|）提供指定可供选择的匹配模式的便利方式。

如果匹配子模式 **this** 或 **that**，模式匹配就成功了。

`/small|medium|large/`

使用多个选替元字符，即可指定多种匹配可能性。

`/(red|blue) pill/`

简易的二选一，"red pill"或"blue pill"都符合这个模式的匹配结果。

磨笔上阵

再重新设计 `validateDate()` 函数使用的正则表达式，这次确认年份只可是 2 位数或 4 位数，没有其他可能。

....

磨笔上阵解答

再重新设计 `validateDate()` 函数使用的正则表达式，这次确认年份只可是 2 位数或 4 位数，没有其他可能。

```
/^\d{2}\/\d{2}\/(\d{2}|\d{4})$/
```

选替中介字符（|）让样式只接受 2 位数或 4 位数年份。

不留下任何需要的改变

Howard 真的很喜欢这个使用正则表达式（以精确匹配模式）、崭新而牢靠的日期验证器。事实上，他太喜欢这种验证方式了，所以他想进一步使用正则表达式来验证 Bannerocity 剩下的两个域：电话号码与电子邮件地址。

Bannerocity – Personalized Online Sky Banners

BANNEROCITY

Enter the banner message: Mandango...macho movie tickets!
Enter ZIP code of the location: 10012
Enter the date for the message to be shown: 03/11/200 Please enter a date (for example, 01/14/1975).
Enter your name:
Enter your phone number:
Enter your email address:
Order Banner
Done

表单里的日期域现在使用正则表达式来验证，正则表达式对强制日期格式非常精确。

Howard 这项验证 Bannerocity 订单上电话号码与电子邮件地址的想法非常好，但也表示我们将要准备一些新的正则表达式，才能成功地支配这些数据格式。

你听到了吗？电话号码的验证

单从验证角度而言，电话号码不算太难检查，它们有固定的格式。当然，没有正则表达式的话，还是需要相当程度的字符串分解，但正则表达式使得电话号码验证只是小事一桩。美国的电话号码遵守下列模式：

模式 = ###-###-####

既然 Howard 不打算飞离老家太远，预设采用美国电话号码格式，应该没有问题。

把电话号码模式里的连字符（-）换成斜线，并稍微调整数字部分的数量，则上述电话号码模式显然与日期模式非常相似。

日期模式使用元字符 \d 与限定符{}，让日期遵守 MM/DD/YYYY 或 MM/DD/YYYY 格式。

```
/^\d{2}\/\d{2}\/\d{2,4}$/
```

电话号码模式与日期模式相似，只不过它使用连字符区分不同部分的号码。

```
/^\d{3}-\d{3}-\d{4}$/
```

validatePhone() 函数变得非常可以预测，都要感谢电话号码的正则表达式与 validateRegEx() 函数。

```
function validatePhone(inputField, helpText) {
  // First see if the input value contains data
  if (!validateNonEmpty(inputField, helpText))
    return false;

  // Then see if the input value is a phone number
  return validateRegEx(/^\d{3}-\d{3}-\d{4}$/,
    inputField.value, helpText,
    "Please enter a phone number (for example, 123-456-7890).");
}
```

你有一封新邮件：验证电子邮件地址

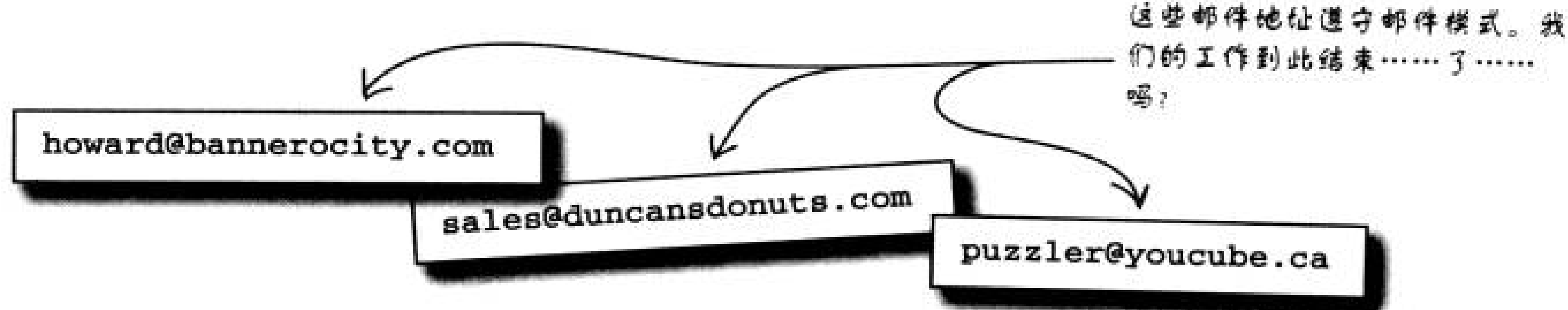
解决了电话号码验证的任务，Howard 最后只剩下验证电子邮件地址域的挑战。与其他数据一样，验证邮件地址的关键在于分解格式，直到成为一以贯之的模式，能用正则表达式建立模型。

2 或 3 个字符的字母数字。

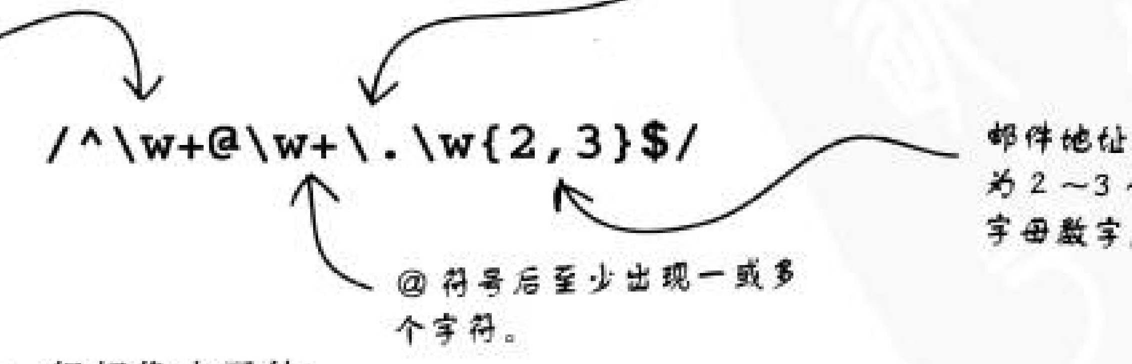
模式 = *LocalName@DomainPrefix.DomainSuffix*

字母数字字符。

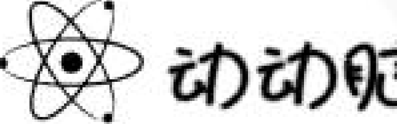
看起来不会很难 —— 电子邮件地址只是三段字母数字字符，其中加入 @ 和点号（.）字符。



创建匹配上述邮件模式的正则表达式非常简单，毕竟各项组成都很容易预测。



虽然这个模式达成了任务，但好像少了什么。所有邮件地址都会遵守这种可预测的格式吗？

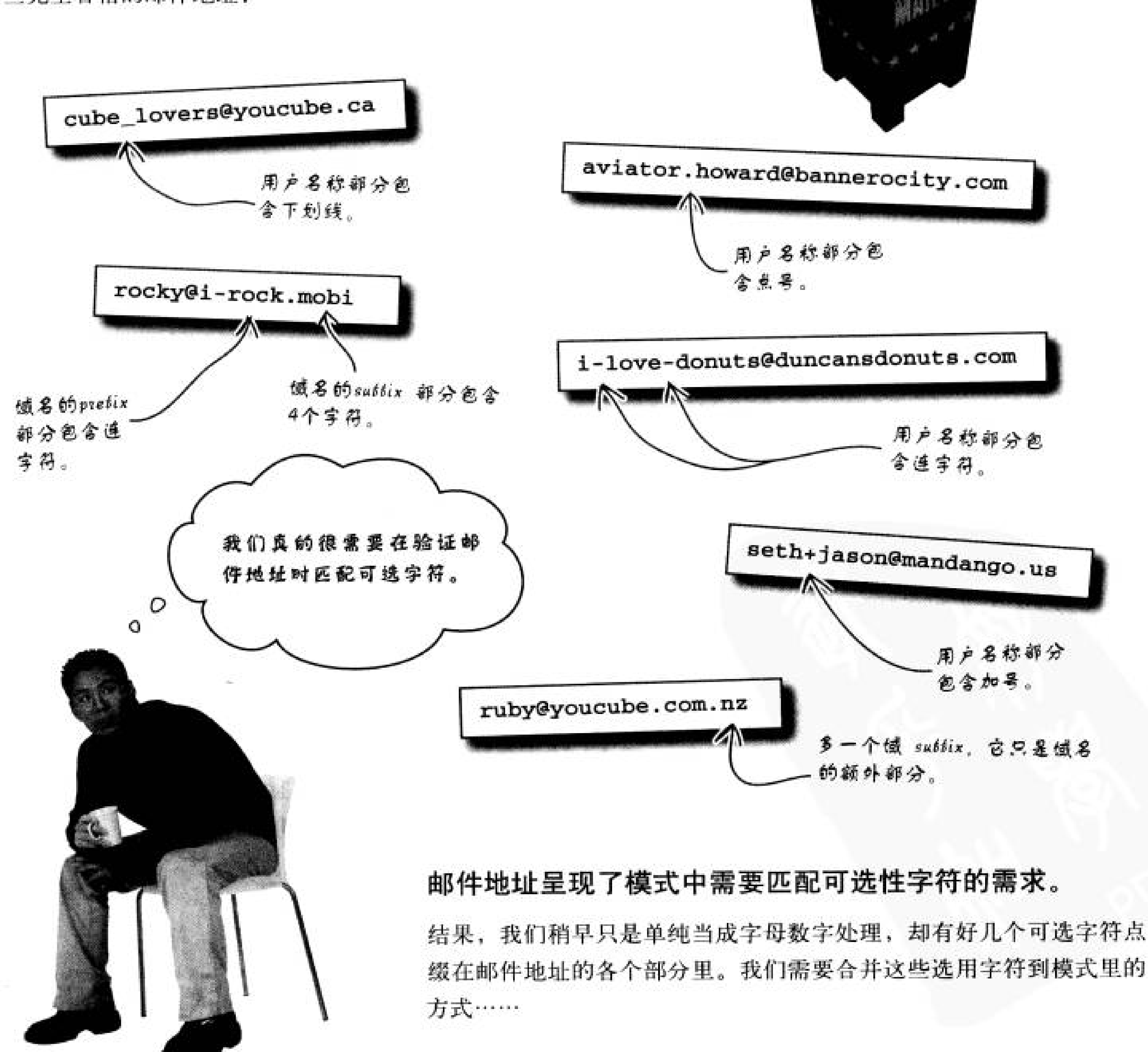


动动脑

还有其他可能的邮件模式吗？请想想你目前见过的各式各样的邮件地址。

例外也是规则

邮件地址其实比它给人的第一眼印象更为复杂。为数据验证构筑可靠的邮件模式时，其实还要考虑几种基础邮件格式的变化版。以下举出一些完全合格的邮件地址：



邮件地址呈现了模式中需要匹配可选性字符的需求。

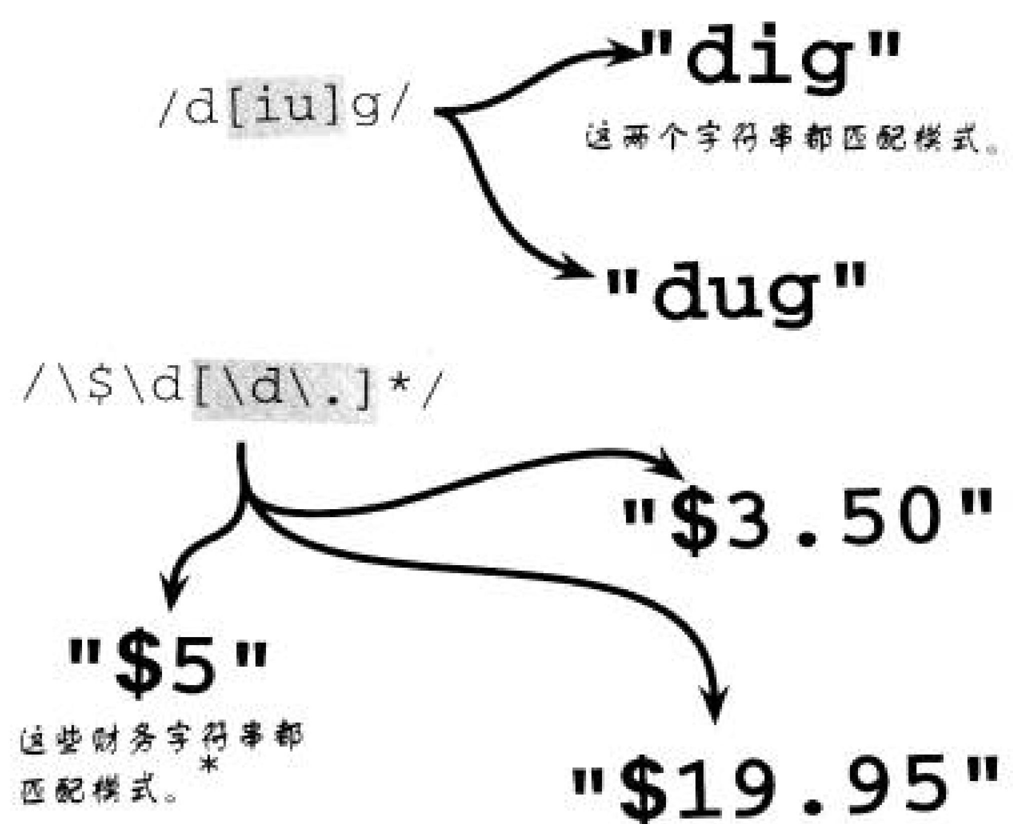
结果，我们稍早只是单纯当成字母数字处理，却有好几个可选字符点缀在邮件地址的各个部分里。我们需要合并这些选用字符到模式里的方式……

从集合中匹配可选字符

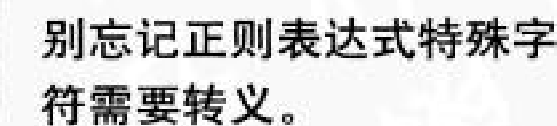
另一个非常便利、常用，又直接影响邮件模式的功能，则是**字符类**（character class），让我们能在模式里创建受到紧密控制的的子模式。说得更精确一点，字符类擅长把非常着重于可选字符的规则建立到子模式中。你可以把字符类想成一组匹配单一字符的规则。

在字符类内，每个列出的字符都是合格的字符匹配目标，有点像元字符间的选替，能建立可替换的子模式列表。不过，字符类的结果都是对单一字符的匹配，除非字符类后面加上限定符。实际看看范例，比较容易了解字符类的概念。

在正则表达式的模式中，字符类提供控制可选字符的有效率方式。



注意！



别忘记正则表达式特殊字符需要转义。

在正则表达式里具有特殊意义的字符需要经过转义，才能把实际字符放在正则表达式里。下列字符前均需加上反斜线（\）做转义：[、\、^、$、.、|、?、*、+、()。

字符类正是塑造邮件地址所需的工具，如此一来，也能把邮件验证加入 Bannerocity……

* 这个模式并不严格，像"$5..."这类字符串也符合它的模式要求。

构建邮件地址验证工具

加入所有可能出现在用户名称与域名中的可选字符后，终于可以建立更为牢靠的邮件地址模式。

还有一点要记得，成功建立模式的方法有很多种，包括邮件地址模式。想建立一个成功表现某种数据格式的每个微妙差异的模式，可能比想象中困难许多。我们已经体验过，通用模式设计如果行得通，把它转换成实际的正则表达式其实相当简单。

磨笔上阵

请补上validateEmail()失落的代码，这个函数用于验证Bannerocity的电子邮件地址。

```
function validateEmail(inputField, helpText) {
  // First see if the input value contains data
  if (!................................(inputField, helpText))
    return false;

  // Then see if the input value is an email address
  return validateRegEx(..................................................,
    inputField.value, helpText,
    ..................................................................);
}
```

请补上validateEmail()失落的代码，这个函数用于验证Bannerocity的电子邮件地址。

```
function validateEmail(inputField, helpText) {
  // First see if the input value contains data
  if (!validateNonEmpty(inputField, helpText))
    return false;

  // Then see if the input value is an email address
  return validateRegEx(/^[\w\.-_\+]+@[\w-]+(\.\w{2,4})+$/,
    inputField.value, helpText,
    "Please enter an email address (for example, johndoe@acme.com).");
}
```

validateNonEmpty()函数仍然先被调用，以检查是否缺少数据。

验证邮件地址的正则表达式时，用到我们所学过的大部分正则表达式技巧。

域名suffix可使用2~4个字母数字字符作为字符串结尾。

用户名称可为字母数字字符，以及"."、"-"、"_"、"+"等字符，且需位于字符串起始处。

如果邮件地址验证失败，即呈现辅助信息，示范正确的输入格式。

有了万全防护的Bannerocity表单

Bannerocity对空中横幅订单的数据收集已经非常完美，一切都要归功于一些绷紧神经的验证工作。Howard高兴于自己下了一张订单，飞上空中……

Data validation is a good thing!

Howard太激动了！终于又能回到他的最爱……飞行啦！

Enter the banner message: Mandango...macho movi...ets!
Enter ZIP code of the location: 10012
Enter the date for the message to be shown: 03/11/2009
Enter your name: Seth Tinselman
Enter your phone number: (212) 555-5339 *Please enter a phone number (for example, 123-456-7890).*
Enter your email address: setht@mandango *Please enter an email address (for example, johndoe@acme.com).*
Order Banner

电话号码和邮件地址域，现已根据非常严格的数据格式接受验证。

JavaScript填字游戏

这个模式大家应该感到很熟悉……填字游戏又来了！现在不用再验证了，只要填上一些答案。

横向提示：

1. 支持正则表达式的JavaScript对象。
2. 当表单域的数据改变时，触发________事件。
4. 指定正则表达式中可选字符的便利方式。
7. 这个对象包含表单中各个域。
9. 正则表达式的特殊字符称为________。
10. 对数据格式的描述。
12. 确认表单域具有数据的验证是________。
13. 控制子模式出现在正则表达式中次数的方式。

纵向提示：

1. 用于匹配文本模式。
3. ________表单数据，确定数据合法。
5. 使用正则表达式匹配字符串的方法（method）。
6. 当用户离开表单域时，触发________事件。
8. HTML属性，能独一无二地办识别表单内的域。
11. 很多时候都很方便的功能，但通常不是对用户告知数据不合格的最佳方式。

JavaScript 填字游戏解答

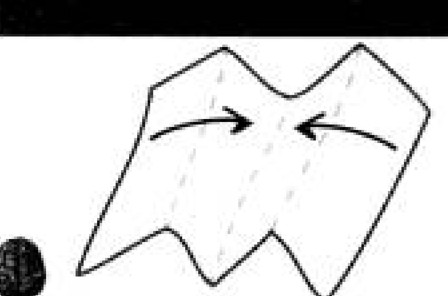
<table>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>1 R</td><td>E</td><td>G</td><td>E</td><td>X</td><td>P</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>2 O</td><td>N</td><td>C</td><td>H</td><td>A</td><td>N</td><td>G</td><td>E</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>G</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>3 V</td><td></td><td></td><td></td><td></td><td></td><td>U</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td>4 C</td><td>H</td><td>A</td><td>R</td><td>A</td><td>C</td><td>5 T</td><td>E</td><td>R</td><td>C</td><td>L</td><td>A</td><td>S</td><td>S</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>L</td><td></td><td>E</td><td></td><td></td><td></td><td>A</td><td></td><td></td><td></td><td></td><td>6 O</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>I</td><td></td><td>S</td><td></td><td>7 F</td><td>O</td><td>R</td><td>M</td><td></td><td></td><td></td><td>N</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>D</td><td></td><td>T</td><td></td><td></td><td></td><td>E</td><td></td><td></td><td></td><td></td><td>B</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>A</td><td></td><td></td><td></td><td></td><td></td><td>X</td><td></td><td></td><td></td><td></td><td>L</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>T</td><td></td><td>8 N</td><td></td><td></td><td></td><td>P</td><td></td><td></td><td></td><td></td><td>U</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>9 M</td><td>E</td><td>T</td><td>A</td><td>C</td><td>H</td><td>A</td><td>R</td><td>A</td><td>C</td><td>T</td><td>E</td><td>R</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>M</td><td></td><td></td><td></td><td>E</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td>10 P</td><td>11 A</td><td>T</td><td>T</td><td>E</td><td>R</td><td>N</td><td></td><td>S</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>L</td><td></td><td></td><td></td><td></td><td></td><td></td><td>S</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>E</td><td></td><td></td><td></td><td></td><td></td><td></td><td>I</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td>R</td><td></td><td></td><td></td><td></td><td></td><td>12 N</td><td>O</td><td>N</td><td>E</td><td>M</td><td>P</td><td>T</td><td>Y</td></tr>
<tr><td>13 Q</td><td>U</td><td>A</td><td>N</td><td>T</td><td>I</td><td>F</td><td>I</td><td>E</td><td>R</td><td></td><td>N</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

JavaScript为网络表单带来什么贡献？

这是左右脑的秘密会谈。



"Mandango...the movie seat picker for tough guys!" "...macho movie seats!"

105012 100012

03/11/200 March 11, 2009

212-555-5339 (212) 555-5339

setht@mandango seth%t@mandango.us

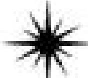
那些数据看起来乱七八糟的！！

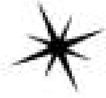
我觉得还好！感觉上就像冲浪……

/^val(ley|ue|krie)/ /name|id$/

JavaScript对表单有不少贡献，所以很难帮任何一项事物制造合格的自变量。答案几乎或多或少都牵涉到数据，但究竟如何牵涉？

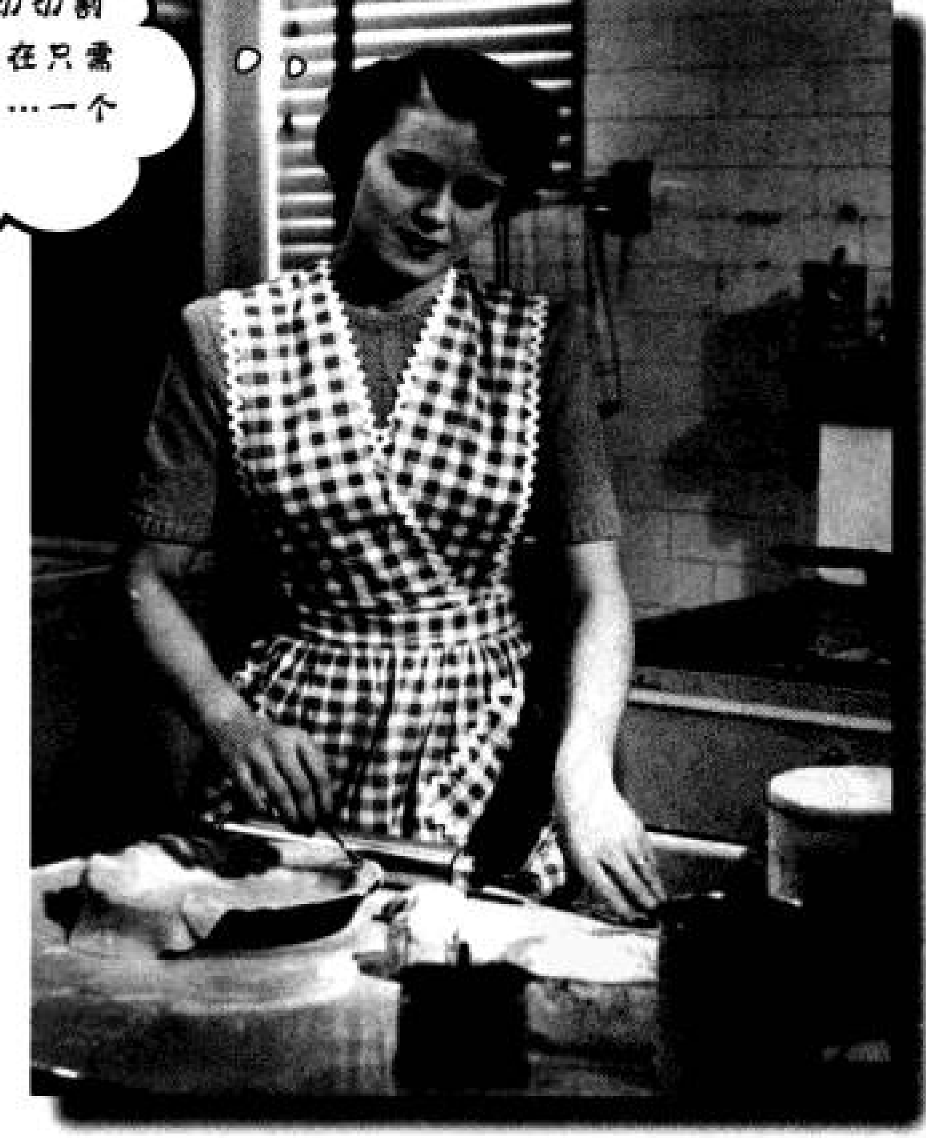
8 驾驭网页





利用 DOM 分割 HTML

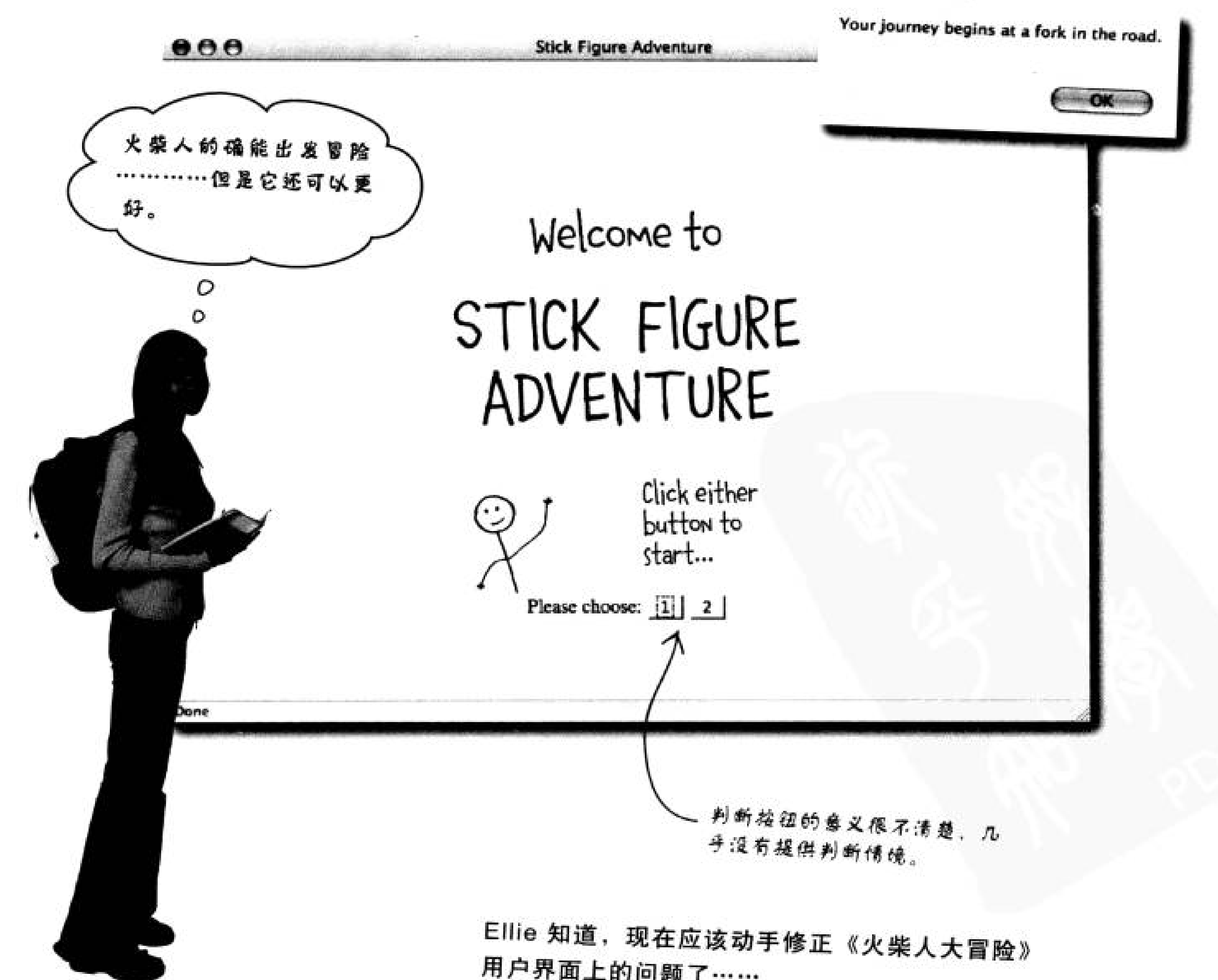
给我正确的成分、轻巧地切切割割，我能造出任何事物。现在只需要尽量贴近我想做的东西……一个派！



利用 JavaScript 控制网页内容其实很像烹饪。只是不用收拾残羹剩肴，但也没办法享受美味的成果。不过，你将能**完整访问**网页内的HTML构成要素；更重要的是，你将拥有**替换**网页成分的能力。**JavaScript让你可以随心所欲地操控网页内的HTML代码**，从而打开各种有趣的机会之门，一切都是因为**标准对象**：DOM（Document Object Model）的存在。

能用但是粗糙简陋……界面很重要

第 4 章的《火柴人大冒险》（Stick Figure Adventure），是个利用 JavaScript 设计人机交互的绝佳范例，但它的用户界面实在有点简陋，尤其是从当代网站标准的眼光来看。使用alert框导览冒险的进行很容易让人厌烦，而隐秘的选项按钮也不够直观，它们只是单纯标示为 1、2 而已。

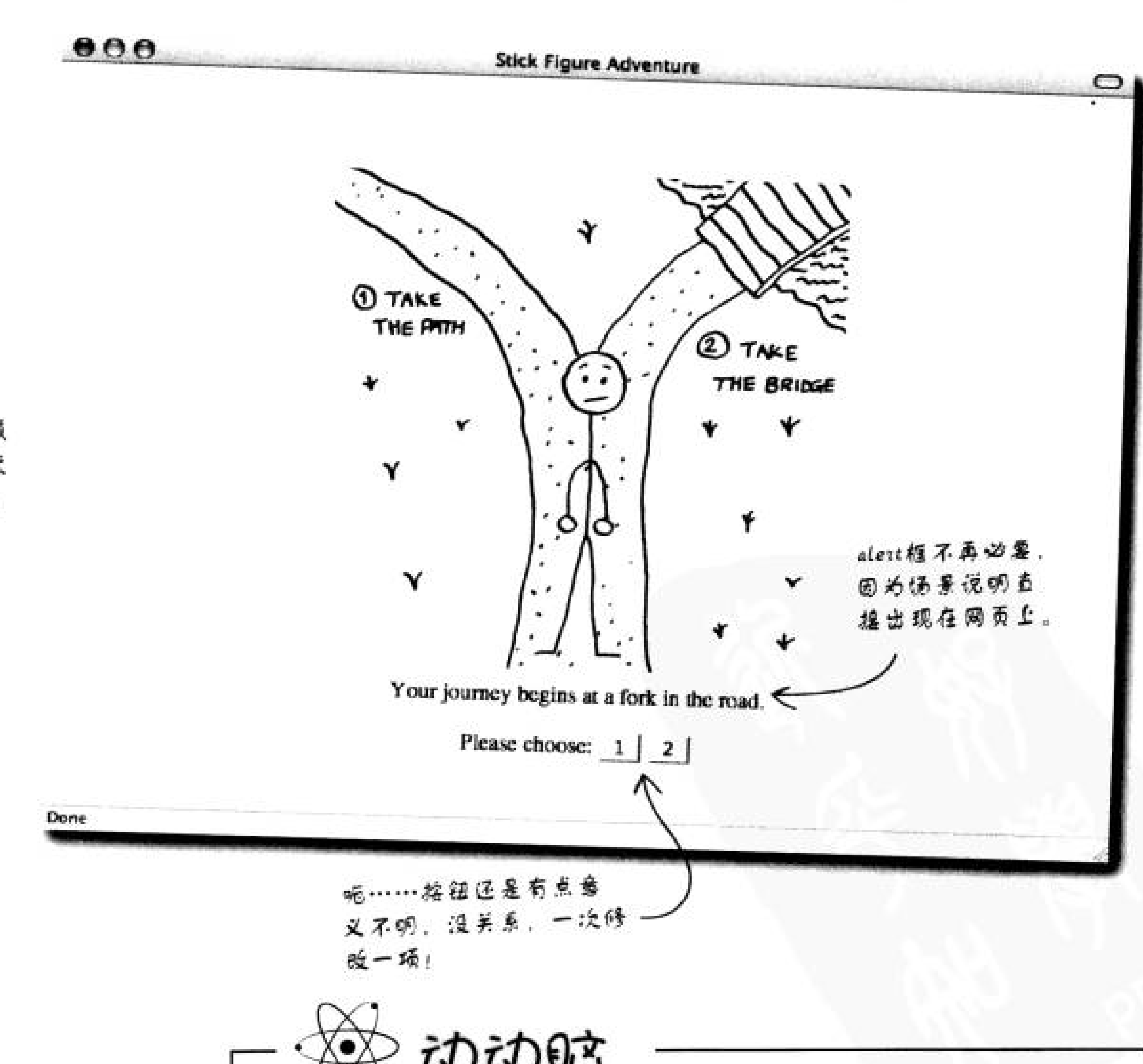


Ellie 知道，现在应该动手修正《火柴人大冒险》用户界面上的问题了……

不使用alert框描述情节

采用alert框呈现场景说明的问题在于：用户按下“OK”后，场景说明文字也随之消失。如果场景的说明能直接呈现在网页上，摆脱恼人的alert框，并把故事带回网页主体会更好。下图是Ellie希望《火柴人大冒险》所呈现的外貌：

最新的《火柴人大冒险》相关文件，已经放在http://www.headfirstlabs.com/books/hfjs/等你下载了。

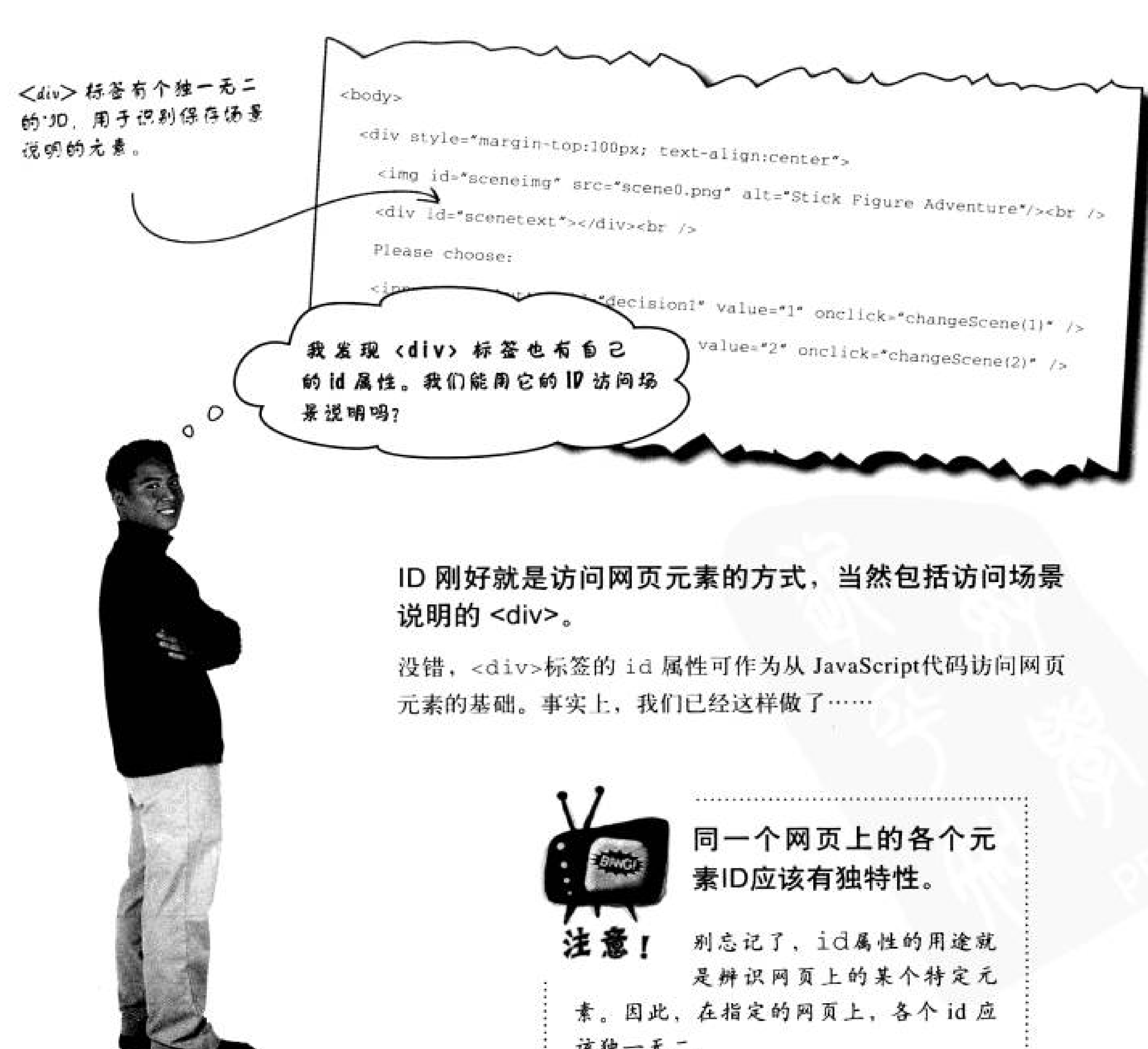


动动脑

你觉得，使用JavaScript支持新场景说明功能的方式是什么？

使用 div 在网页上创建空间

为了在网页上呈现场景说明，我们首先需以 HTML 元素在网页上定义一块实际区域，才能真正进入使用 JavaScript代码的主题。既然场景说明自成一段文本，<div>标签应该很适合保存这段文本。



ID 刚好就是访问网页元素的方式，当然包括访问场景说明的 <div>。

没错，<div>标签的 id 属性可作为从 JavaScript代码访问网页元素的基础。事实上，我们已经这样做了……

注意！

同一个网页上的各个元素ID应该有独特性。

别忘记了，id属性的用途就是辨识网页上的某个特定元素。因此，在指定的网页上，各个 id 应该独一无二。

访问HTML元素

我们已经很熟悉标准document对象的 getElementById() 方法，它让我们接触网页内部并访问 HTML 元素……只要元素具有独一无二的 ID。

```
var sceneDesc = document.getElementById("scenetext");
```

这部分必须符合 HTML元素的id属性，本例的元素为div。

div 元素可透过它的 id 属性而被访问。

```
<img id="sceneimg" src="scene0.png"...
<div id="scenetext"></div><br />
Please choose:
```

有了场景说明元素，我们又向操纵元素内容的目标更接近了一步。但有件事值得我们停下来探究一番 —— getElementsByTagName() 方法，它能抓出网页上所有特定类型的元素，例如div或img。这个方法返回的结果数组里，包含出现在网页上的每一个指定元素，并按照出现在 HTML 的顺序排列。

标签本身的名称，但去除 <>。

```
var divs = document.getElementsByTagName("div");
```

请参考下列 HTML主体代码，设计可以访问橘色图像的 JavaScript代码，先使用getElementById()，再改用getElementsByTagName()。

```
<body>
  <p>Before starting, please choose an adventure stress level:</p>
  <img id="green" src="green.png" alt="Relaxing" /><br />
  <img id="blue" src="blue.png" alt="Irritating" /><br />
  <img id="yellow" src="yellow.png" alt="Frazzled" /><br />
  <img id="orange" src="orange.png" alt="Panicked" /><br />
  <img id="red" src="red.png" alt="Maddening" />
</body>
```

使用getElementById()：..

使用getElementsByTagName()：..

习题解答

请参考下列HTML主体代码，设计可以访问橘色图像的JavaScript代码，先使用getElementById()，再改用getElementsByTagName()。

```
<body>
  <p>Before starting, please choose an adventure stress level:</p>
  <img id="green" src="green.png" alt="Relaxing" /><br />
  <img id="blue" src="blue.png" alt="Irritating" /><br />
  <img id="yellow" src="yellow.png" alt="Frazzled" /><br />
  <img id="orange" src="orange.png" alt="Panicked" /><br />
  <img id="red" src="red.png" alt="Maddening" />
</body>
```

橘色图像是数组中的第4个img元素，索引编号为3。

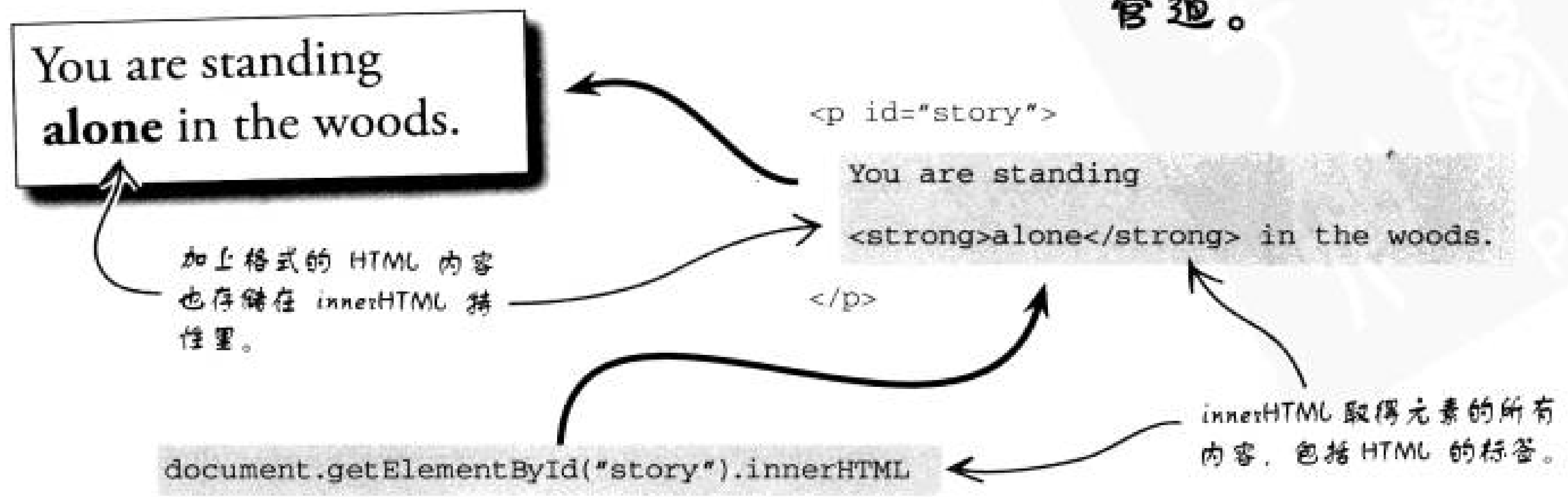
使用 getElementById()：document.getElementById("orange ")

使用 getElementsByTagName()：document.getElementsByTagName("img ")[3]

与内在的HTML建立接触

讲到访问HTML元素的事务，真正的重点都在于元素里存储的内容。我们可以透过innerHTML特性，访问能够保存文本内容的元素，例如div、p。

innerHTML特性对所有存储在元素里的内容提供了访问管道。

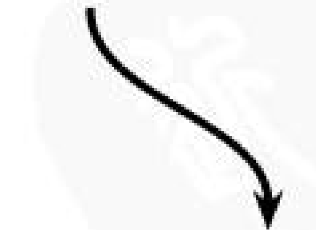


innerHTML 也能用于设置网页内容。

事实上，以innerHTML特性**设置**（set）HTML 内容的使用率，与利用它**取得**（get）内容的使用率差不多。只要把文本字符串指派给元素的 innerHTML 特性，元素内容即可设为 HTML 文本字符串。新内容将取代任何原本属于元素的内容。

```
document.getElementById("story").innerHTML =
  "You are <strong>not</strong> alone!";
```

元素内容经由指派字符串给 innerHTML 而设置，亦可如本例“取代”原始内容。



You are **not** alone!

磨笔上阵

假设已根据用户的判断而正确地设置了场景说明，请利用 innerHTML特性，对《火柴人大冒险》的场景说明元素设置信息文本。

....

假设已根据用户的判断而正确地设置了场景说明，请利用innerHTML特性，对《火柴人大冒险》的场景说明元素设置信息文本。

场景说明元素<div>的ID是"sceneText"。

```
document.getElementById( "scenetext" ).innerHTML = message;
```

较少干扰的冒险旅程

动态变更的场景说明为《火柴人大冒险》省去麻烦的警告信息，带来更顺畅、更享受的用户体验。

哇噢，虽然只是小小的改变，但是我喜欢！

Stick Figure Adventure

① WALK ACROSS BRIDGE

② GAZE INTO STREAM

You are standing on the bridge overlooking a peaceful stream.

Please choose: 1 2

场景说明终于能配合网页流程。

除了为场景说明区域（message）增加<div>元素，以及设置innerHTML特性的代码，《火柴人大冒险》还需要增加message变量，并设定各幕场景所需的说明信息……

放大《火柴人大冒险》

```
<html>
  <head>
    <title>Stick Figure Adventure</title>

    <script type="text/javascript">
      // Initialize the current scene to Scene 0 (Intro)
      var curScene = 0;

      function changeScene(decision) {
        // Clear the scene message
        var message = "";

        switch (curScene) {
        case 0:
          curScene = 1;
          message = "Your journey begins at a fork in the road.";
          break;
        case 1:
          if (decision == 1) {
            curScene = 2
            message = "You have arrived at a cute little house in the woods.";
          }
          else {
            curScene = 3;
            message = "You are standing on the bridge overlooking a peaceful stream.";
          }
          break;
        ...
        }

        // Update the scene image
        document.getElementById("sceneimg").src = "scene" + curScene + ".png";

        // Update the scene description text
        document.getElementById("scenetext").innerHTML = message;
      }
    </script>
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="sceneimg" src="scene0.png" alt="Stick Figure Adventure" /><br />
      <div id="scenetext"></div><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```

创建局部变量message，为新场景存储说明信息。

为每幕场景设置独一无二的场景说明。

使用innerHTML特性，把场景说明设置为变量message的内容。

没有蠢问题

问：**我可以用**getElementById()**访问网页上的任何元素吗？**

答：可以，但需要先为每个元素加上独一无二的id属性值。在使用getElementById()时，id属性极度重要。

问：innerHTML**特性能用于设定任何HTML元素的内容吗？**

答：不行。想设定元素的“内在HTML”（inner HTML），该元素必须能够包含HTML的内容。所以innerHTML特性实际上用于设置内容类元素，例如div、span、p，以及其他装载内容的元素。

问：**使用**innerHTML**特性设定元素内容时，元素内容会发生什么变化啊？**

答：设置元素内容时，innerHTML特性一定完全改写任何原有内容，所以它没有**附加**（appending）的概念。不过，附加的效果可借由串联新内容至旧内容上，然后指定串联结果给innerHTML而达成，如下所示：

```
elem.innerHTML += " This
sentence gets appended.
```

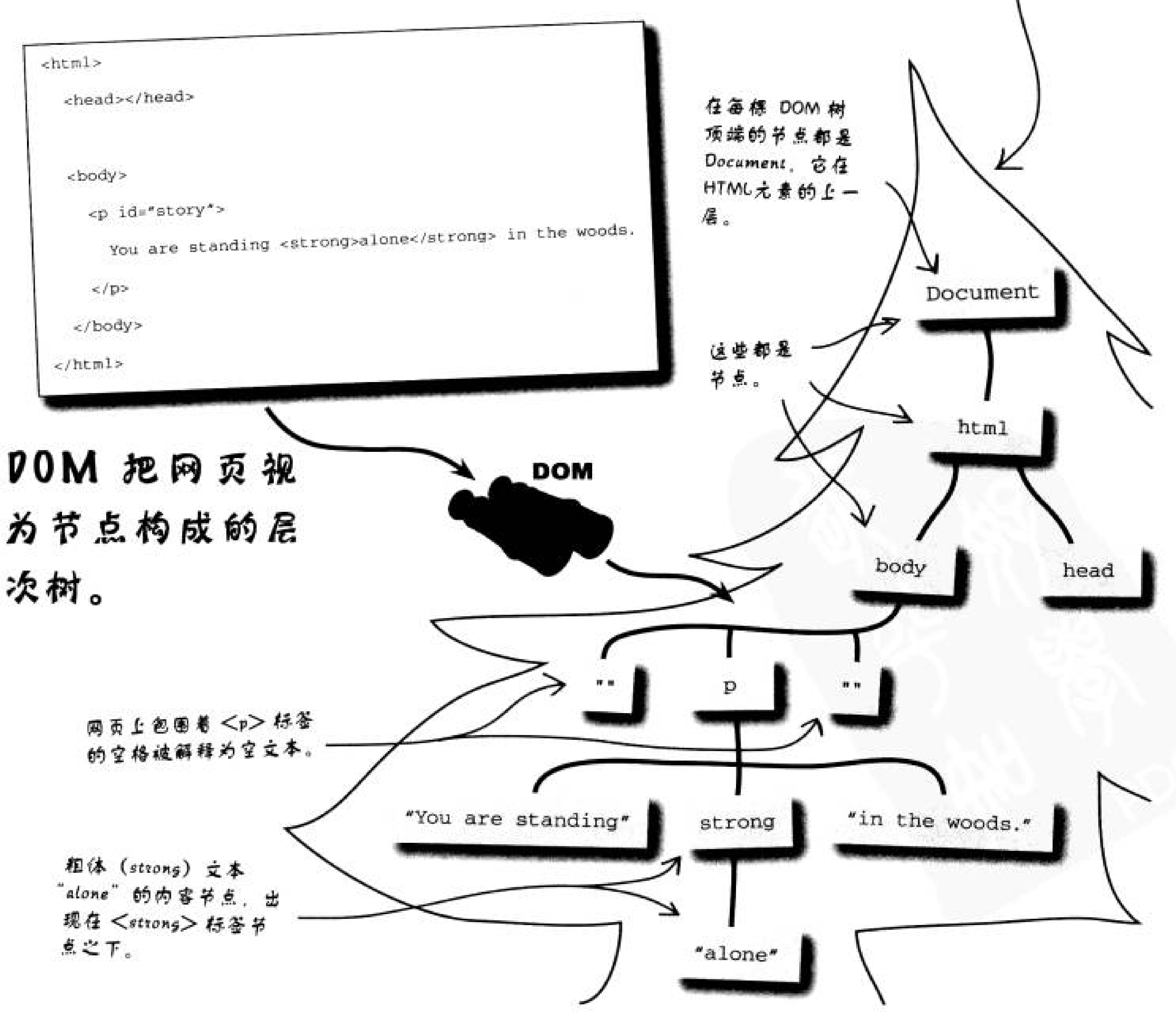
呃，没错，可是我们真的需要担心万维网的标准吗？

innerHTML最初由Microsoft创建，作为IE浏览器的专利功能。从那时候开始，其他浏览器也开始采用innerHTML，它渐渐变成**非正式**的标准，用于快速轻松地改变网页元素内容。

不过，innerHTML并非标准的事实仍不会改变。或许看起来不怎么重要，但关于“标准”的概念重点，其实是为了让网页和网络应用程序能尽量在各种浏览器与平台上运行。除此之外，我们也有既可以达成相同任务，又遵守标准的方式，而且还比较有灵活性并强大，只是不算很简单。遵守标准的方式牵涉到DOM，Document Object Model（文档对象模型），一组为JavaScript提供完整、全面控制网页结构与内容的对象。

见树也见林：Document Object Model（DOM）

DOM 提供对脚本友善（script-friendly）的网页结构与内容的视图，如果你想用 JavaScript 动态改变网页，这点非常重要。透过加装 DOM 的透镜，网页看来就像形成树状的层层元素。树上的每片叶子是个**节点**（node），直接关联到网页上的各个元素。当树上的某节点出现在另一个节点下时，则被称为**子节点**（child）。



DOM 把网页视为节点构成的层次树。

你的网页是DOM节点的集合

DOM树的每个节点均根据类型（type）分类。主要的节点类型对应至网页的结构，主要以元素节点（element node）和文本节点（text node）构成。

DOM的节点根据节点类型分类。

DOCUMENT

位于DOM树最顶端的节点，代表文档本身，且出现在 `html` 元素的上一层。

ELEMENT

对应到HTML标签的HTML元素。

TEXT

元素的文本内容，一定存储在元素下的子节点里。

ATTRIBUTE

元素的属性，可透过元素节点访问，但不会直接出现在DOM树里。

把节点类型应用到网页的DOM树上，有助于厘清DOM理解网页各部分的方式。我们的兴趣特别着重于：`TEXT` 节点如何总是立即出现在`ELEMENT`元素下，作为该节点内容的一部分（或全部内容）。

```
<html>
  <head></head>
  <body>
    <p id="story">
    You are standing <strong>alone</strong> in the
woods.
    </p>
  </body>
</html>
```

虽可透过DOM访问属性，属性也具有自己的节点类型，它们却不会出现在网页的节点树中。

DOCUMENT
Document
ELEMENT
html
ELEMENT
body
ELEMENT
head
TEXT
""
ELEMENT
p
TEXT
""
TEXT
"You are standing"
strong
TEXT
"in the woods."
ELEMENT
TEXT
"alone"

虽然这里的head元素是空的，但多数网页的head节点下都有子节点。

请完成代表《火柴人大冒险》的HTML代码的 DOM 树，填入每个节点的名称。也请记得注明每个节点的类型。

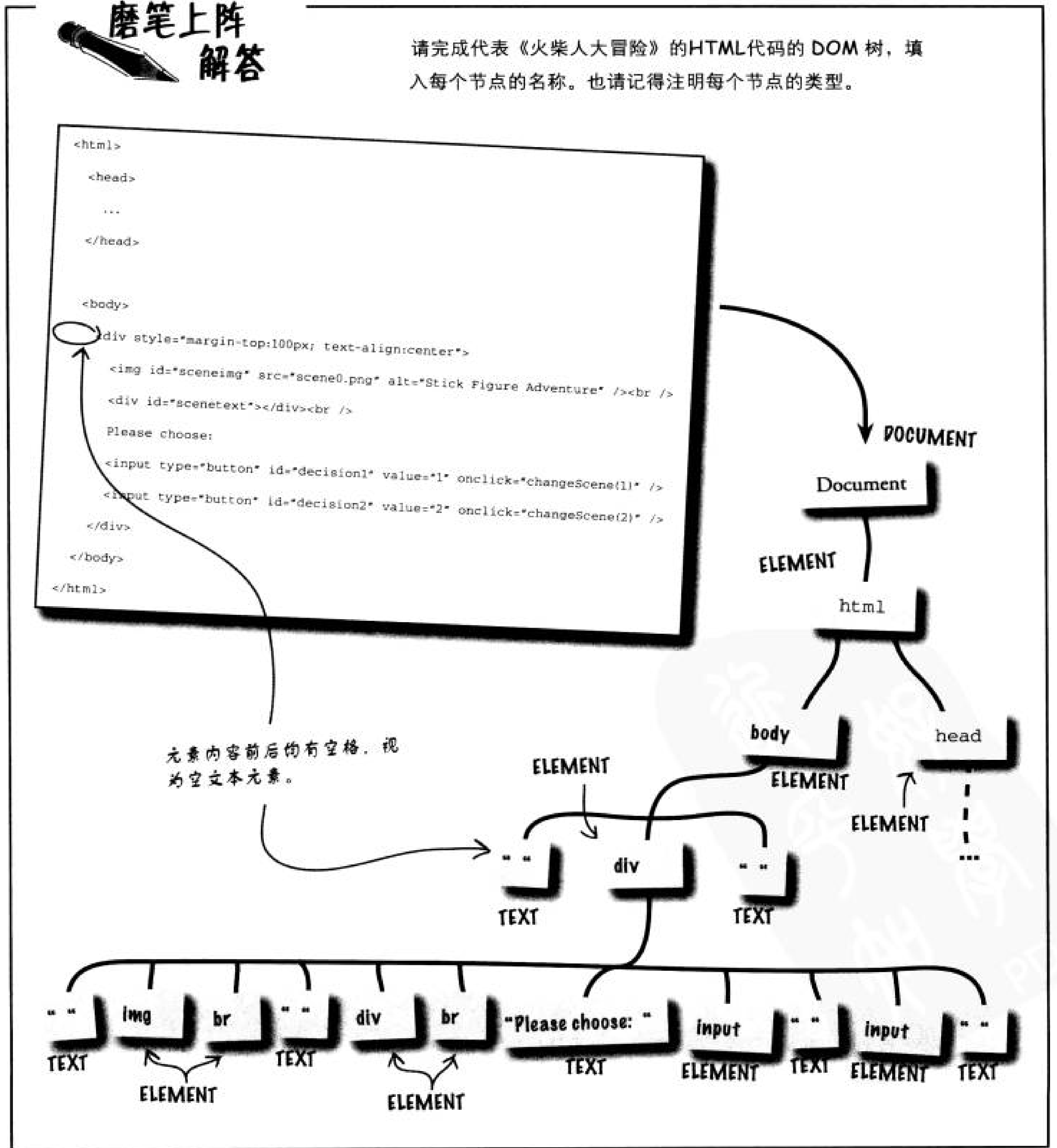
```
<html>
  <head>
    ...
  </head>

  <body>
    <div style="margin-top:100px; text-align:center">
      <img id="sceneimg" src="scene0.png" alt="Stick Figure Adventure" /><br />
      <div id="scenetext"></div><br />
      Please choose:
      <input type="button" id="decision1" value="1" onclick="changeScene(1)" />
      <input type="button" id="decision2" value="2" onclick="changeScene(2)" />
    </div>
  </body>
</html>
```

Document

html

head

...



磨笔上阵
解答
请完成代表《火柴人大冒险》的HTML代码的 DOM 树，填入每个节点的名称。也请记得注明每个节点的类型。
<html>
<head>
...
</head>
<body>
<div style="margin-top:100px; text-align:center">

<div id="scenetext"></div>
Please choose:
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
</div>
</body>
</html>
DOCUMENT
Document
ELEMENT
html
body
head
ELEMENT
ELEMENT
元素内容前后均有空格，视为空文本元素。
ELEMENT
" "
div
" "
TEXT
TEXT
...
" "
img
br
" "
div
br
"Please choose: "
input
" "
input
" "
TEXT
ELEMENT
TEXT
ELEMENT
TEXT
ELEMENT
TEXT
ELEMENT
TEXT

利用特性攀爬 DOM 树

大部分与 DOM 的交互均从document对象开始，它是文档节点树的最上层节点。文档对象提供各种好用方法（method），例如 getElementById()、getElementsByTagName()，以及许多特性。document 对象的许多特性可由树上的**每个**节点取得，有些特性甚至能帮我们导向其他节点。也就是说，节点特性能用于导览节点树。

在 DOM 树的节点间往返移动时，节点特性是个便利的好工具。

nodeValue
存储于节点的值，只限文本与属性节点使用（不含元素）。

nodeType
节点类型，例如它是DOCUMENT或TEXT等等，但以代号表示。

childNodes
包含节点下所有子节点的数组，以出现在 HTML代码中的顺序而排列。

firstChild
节点下的第一个子节点。

lastChild
节点下的最后一个子节点。

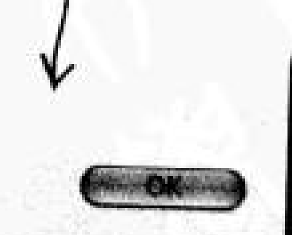
这些特性就是巧妙操纵文档树以访问特定节点数据的关键。例如使用节点特性并加上 getElementById()（访问节点的）方法，即可快速分离出某个节点。

nodeValue 特性总是包含纯文本，没有额外格式。

```
alert(document.getElementById("scenetext").nodeValue);
```

nodeValue 特性访问存储在节点里的文本内容。

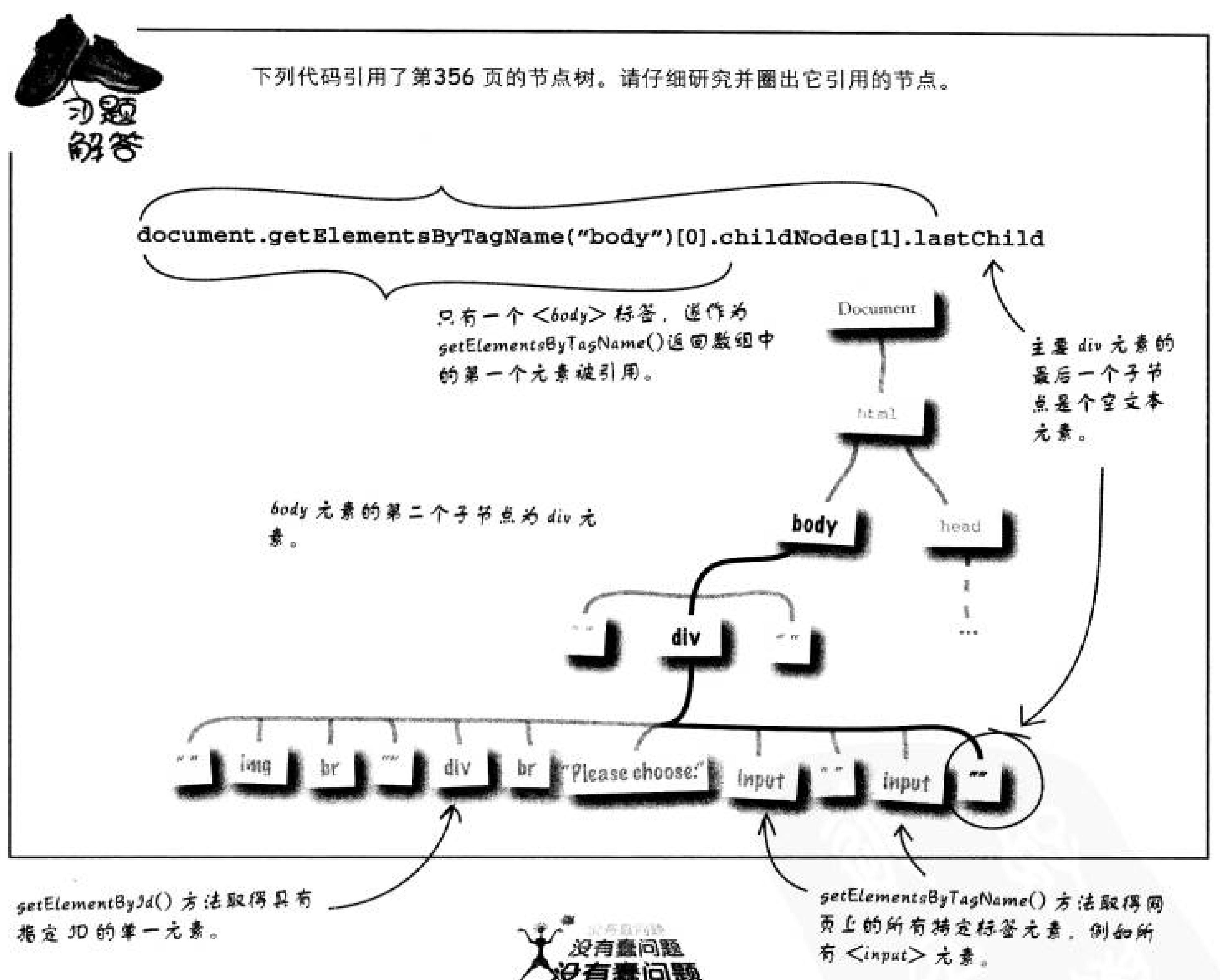
《火柴人大冒险》的场景说明，刚开始只是一片空白。



好吧，刚才这个《火柴人大冒险》的例子不算很好，场景说明文本div原本就是一片空白。但场景最终应该随着故事的进行，设置为非常令人信服的文本，想到这点，上例代码似乎稍微聪明了一点。

下列代码引用了第356页的节点树。请仔细研究并圈出它引用的节点。

```
document.getElementsByTagName("body")[0].childNodes[1].lastChild
```

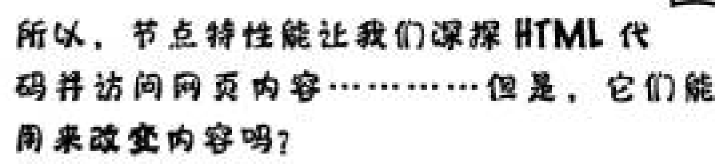


没有蠢问题

问：对 DOM 树，使用`getElementById()`与使用`getElementsByTagName()`有什么不一样？我该如何选择使用的时机？

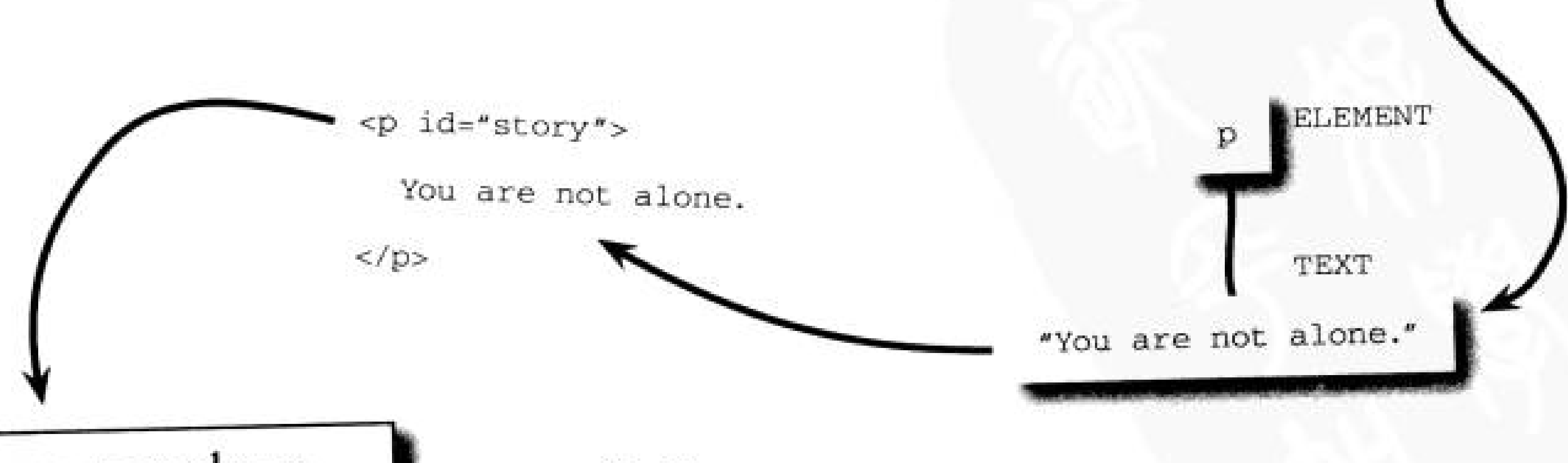
答：这两个方法提供不同途径，基本上与隔离单一元素或一群相似元素有关。需要隔离单一元素时，没人能打败`getElementById()`——只要亮出元素 ID，你就过关了。

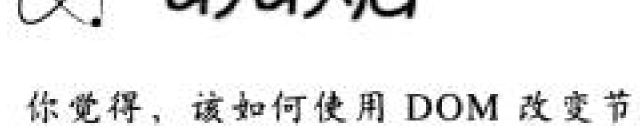
若以一群节点为目标，`getElementsByTagName()`则是较佳选择。假设你打算利用JavaScript隐藏网页上所有图像元素，首先必须调用`getElementsByTagName()`函数并传入自变量"img"，以取得网页上所有图像节点。然后改变 CSS 样式表中的`visibility`特性，以隐藏所有图像元素……哎呀，我讲过头了，下一章将回到 DOM 与 CSS。现在，各位只要了解`getElements-ByTagName()`虽然没有`getElementById()`那么受欢迎，仍在特殊情况中占有一席之地。



DOM 特性让我们能改变网页内容并维持与网络标准兼容。

既然 DOM 视网页文档里的一切为节点，改变网页即牵涉到改变其中的节点。以文本内容为例，像div、span、p这类元素的文字，通常呈现为该元素的子节点，立即出现在元素（节点）下。如果文本内容是个独立的文本节点，没有其他HTML 元素，则这个文本节点将被放在第一个子节点的位置。如下所示：



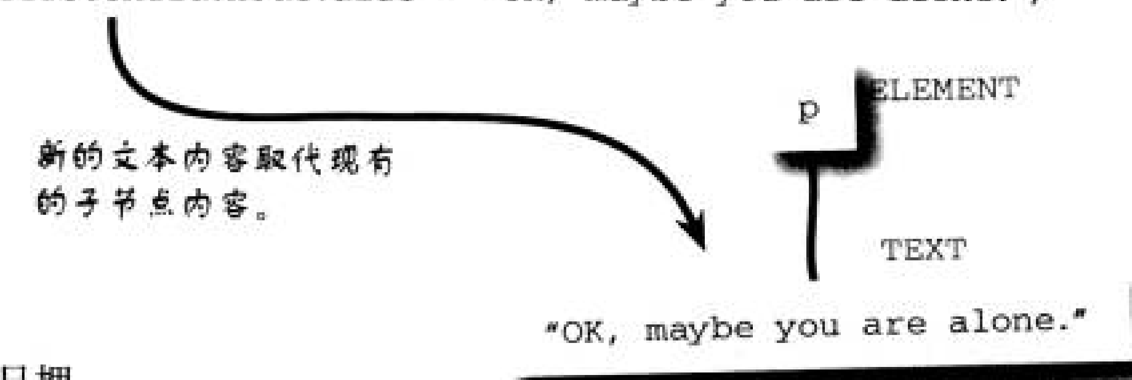


你觉得，该如何使用 DOM 改变节点的文本？

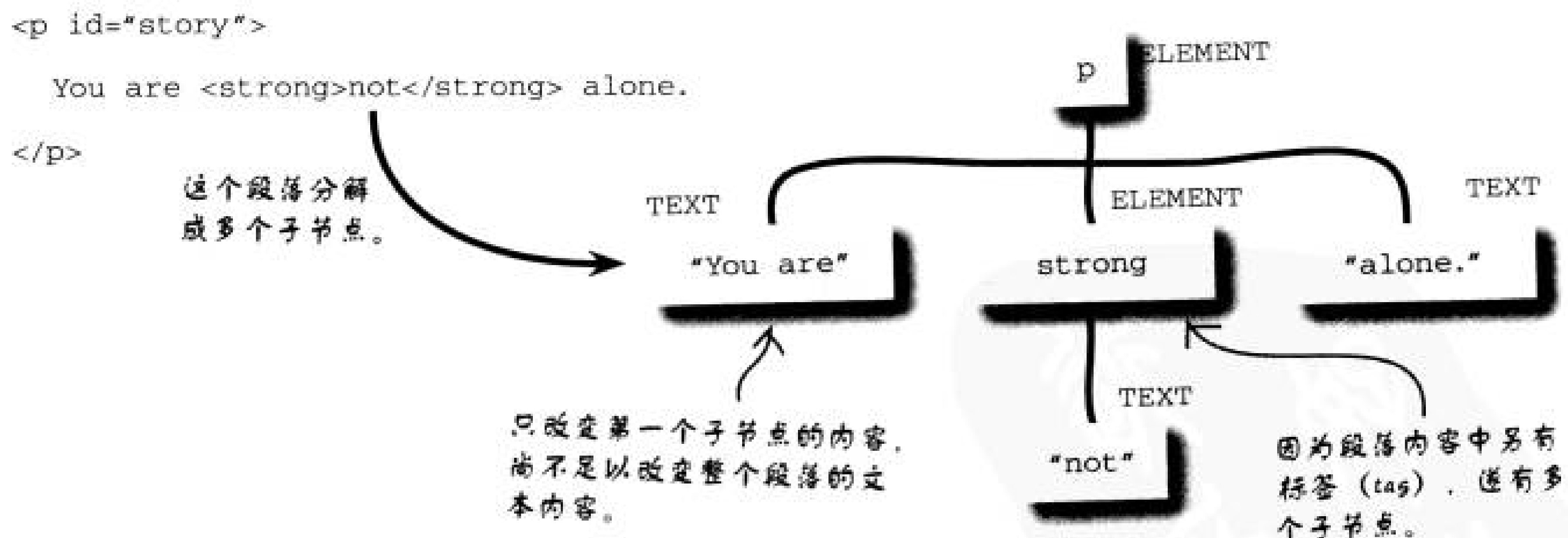
使用DOM改变节点文本

如果每个节点只有一个保存文本内容的子节点，就有可能利用nodeValue特性，轻易地指派新文本内容给子节点。这个方法可以运作，但除非元素只拥有一个节点。

```
document.getElementById("story").firstChild.nodeValue = "OK, maybe you are alone.";
```



但事情不见得一直这么简单。如果节点下不只拥有一个子节点呢？如下例：



如果只取代第一个子节点，剩余子节点仍在原处，我们会制造出很奇怪的效果：

```
document.getElementById("story").firstChild.nodeValue = "OK, maybe you are alone.";
```

OK, maybe you are alone.not alone.

只取代了第一个子节点，剩余子节点都没改变……这个修改结果真是自相矛盾。

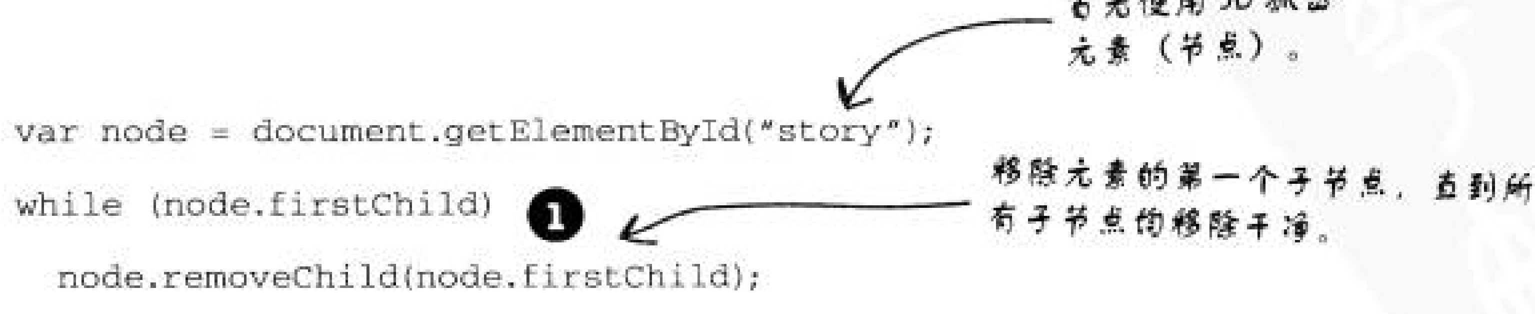
改变节点文本的（安全）三步骤

只透过第一个子节点改变节点内容的问题，在于并未将其他子节点纳入考量因素。讲到改变节点内容，我们其实应该清除所有子节点，然后新增具有新内容的新子节点。

1. 移除所有子节点。
2. 根据新内容创建新的文本节点。
3. 把新创建的文本子节点附加在节点下。

上述步骤可透过三个 DOM 方法完成：

想改变范例“you are not alone”的文本内容，我们必须走过这三个步骤：首先确认移除所有子节点，再来创建新的文本节点，最后对段落节点附加文本子节点。




```
var node = document.getElementById("story");
while (node.firstChild)
  node.removeChild(node.firstChild);
node.appendChild(document.createTextNode("OK, maybe you are alone."));
```

OK, maybe you are alone.

DOM 结构宝座

本周主题：

节点讨论 DOM 树的智慧

HeadFirst：我听说你是 DOM 树存储单位的最小部分，有点像 HTML 内容的原子。真的吗？

节点（Node）：我不太确定自己有多么像原子，多么具原子性，但我的确代表 DOM 树中分离的信息。你可以把 DOM 想成把网页分解成非常小、比特大小的信息……我就是比特大小的那个部分。

HeadFirst：这有什么重要的吗？讲来有点失礼，但能够把网页分解成小段数据，真的有那么重要吗？

Node：如果你重视访问或改变网页内容的话。许多脚本就很重视内容的改变，此时 DOM 就非常重要了。但 DOM 重要的真正原因，其实是它能够把网页分解成细小的片段与部分。

HeadFirst：当你在分解网页时，是否也冒着失去部分网页的风险？很多人自己动手拆开东西，结果都是剩下一堆破铜烂铁，接下来你就发现大家把东西弄坏了。

Node：不会，DOM 没有这种问题，因为把网页读成节点树时，并不需要实际把一切都拆开来。DOM 只是提供树状视图，无论你是否打算雕塑或修剪网页上的数据。

HeadFirst：太好了，让人松了一口气。但如果我真的想修剪网页，是不是就轮到你登场了呢？

Node：是的。只不过我的用途不限于修剪 —— 你也可以在节点上加入网络数据。

HeadFirst：听起来很棒耶！要怎么做呢？

Node：你看，网页上的每块信息，都被塑造成树上的一个节点。所以你可以透过我 —— 节点，访问网页上的任何事物。或者，你也能利用我创建全新的数据，然后再把新数据加到节点树上。DOM 真的非常有灵活性。

HeadFirst：听起来很聪明。不过，我还是有个疑问 —— 你和元素究竟如何产生关联呢？你们其实是同一个人吗？

Node：没错，事实就是如此。但我比较进步一点。还记得元素只是另一种看待标签（例如 `<div>`、`<span>`）的角度吗？网页上的每个元素在文档树中都表示为一个节点 —— 从这个角度来看，我和元素并无二致。不过，我也能代表存储于元素中的内容，所以存储于 `<div>` 的文本本身也是个节点，存储在文档树中的 `div` 节点之下。

HeadFirst：听起来还是有点混乱耶！你要怎么区分元素和它的内容有何不同呢？

Node：这个嘛……首先，存储在元素（或节点）中的内容，在 DOM 树中一定是该元素的子节点。其次，所有节点都可依类型区分：元素节点的类型是 `ELEMENT`，文本节点的类型则是`TEXT`。

HeadFirst：所以，如果我想访问某个元素的文本内容，是否只需寻找 `TEXT` 类型的节点呢？

Node：确实可以，但请记得，`nodeType` 特性其实返回的是每种节点类型的代号，例如 `TEXT`类型的代号是 3，`ELEMENT` 是 1。但其实不用这么麻烦，因为你只需要找寻元素节点的子节点，就能取得它的内容。

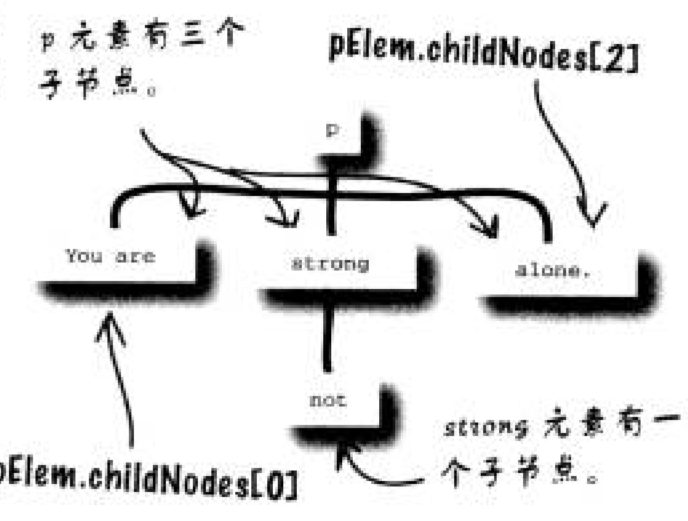
HeadFirst：我懂了。感谢你拨冗接受访谈，还说明了 DOM 树的各种好处。

Node：别客气。如果你对“树手术”的话题有兴趣，别忘记再找我上节目哦！

没有蠢问题

问：我还是不太清楚子节点和它们的组织方式。例如，childNodes特性究竟如何运作呢？

答：当节点内包含数据时，它就是个父节点（parent node），其中的内容则被DOM解释为子节点。如果数据不只由纯文本构成，则分解成许多个子节点。某个父节点下的所有子节点将以数组存储，出现在父节点的childNodes特性里；子节点在数组里的存储顺序，则与它们出现在HTML代码时的顺序相同。childNodes数组的第一个子节点即可使用childNodes[0]访问。也能以循环处理这个数组，以访问每一个子节点。



问：在移除某节点下所有子节点的代码中，while循环的测试条件如何运作？

答：你讲的while循环是：

```
while(nodefirstChild)
```

这项测试会检查节点是否含有第一个子节点。如果第一个子节点还存在，while循环上下文中的结果值将为true，因而循环继续下一轮运作。如果没有第一个子节点，则表示已经没有其他节点，此时，node.firstChild的结果即为null，将造成while循环上下文中的结果值为false。所以，while循环其实在检查第一个子节点是否已为null，由此再推导出有无子节点隐身暗处的结论。

JavaScript冰箱磁铁

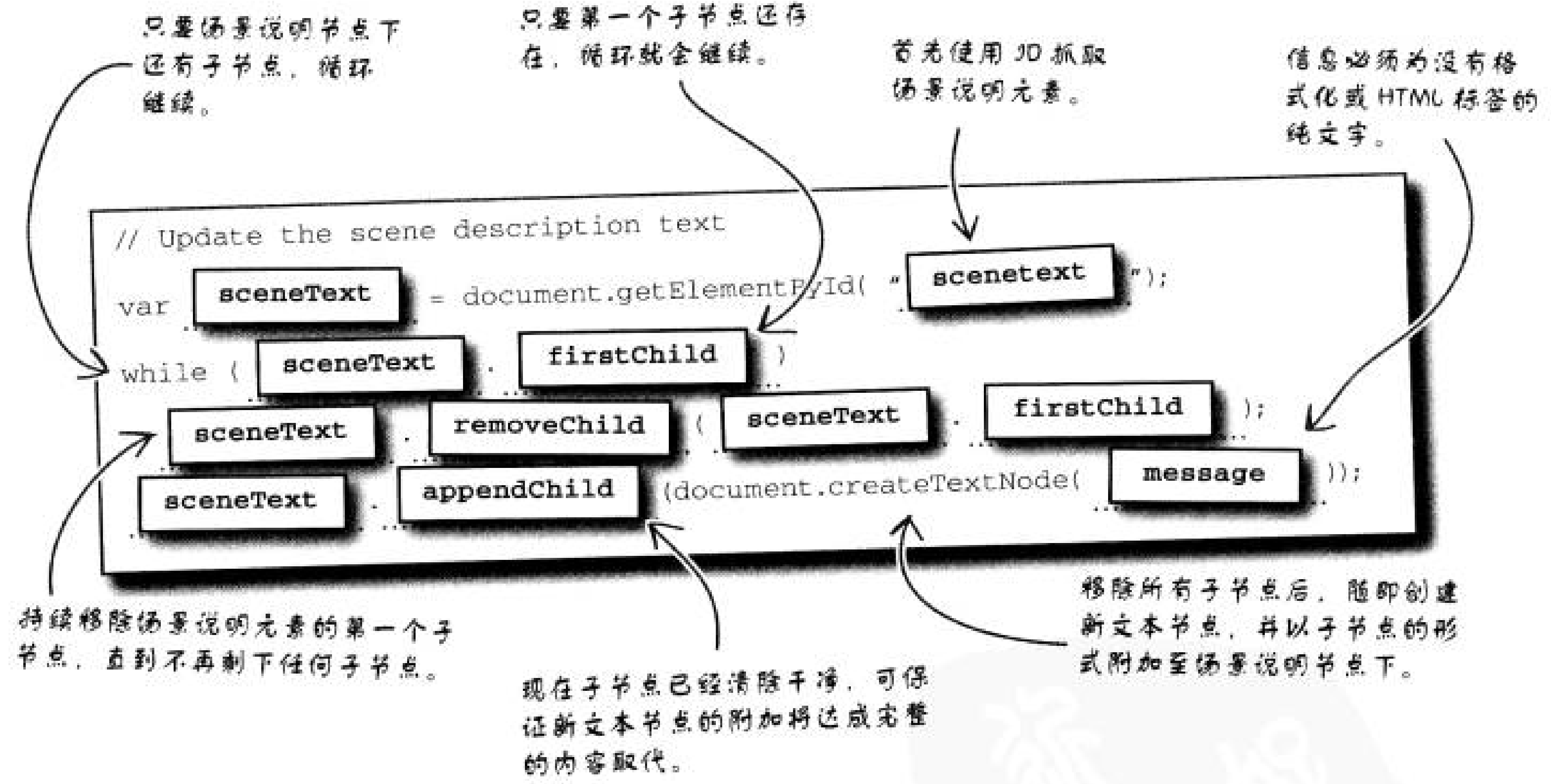
与DOM兼容的《火柴人大冒险》代码少了好几块重要代码。请使用我们附上的磁铁，完成改变场景说明元素（节点）的文本。磁铁可重复使用。

```
// Update the scene description text

var ........ = document.getElementById(" ........ ");
while ( ........ . ........ )
  ........ . ........ ( ........ . ........ );
........ . ........ (document.createTextNode( ........ ));
```

firstChild　appendChild　scenetext　removeChild　sceneText　message

JavaScript 冰箱磁铁解答

与DOM兼容的《火柴人大冒险》代码少了好几块重要代码。请使用我们附上的磁铁，完成改变场景说明元素（节点）的文字。磁铁可重复使用。

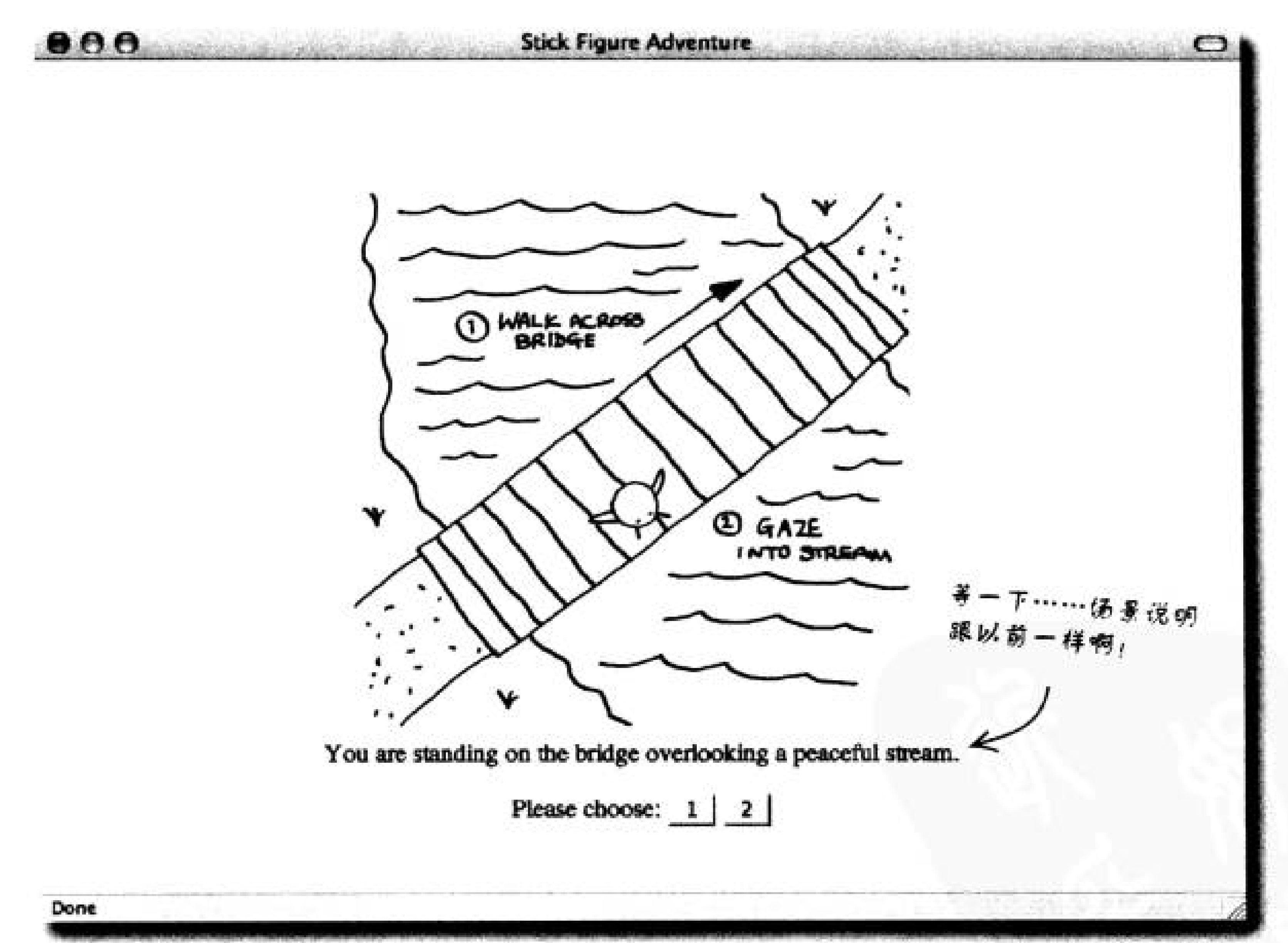


复习要点

- innerHTML虽非万维网的标准，但这个特性能访问元素内存储的所有内容。
- Document Object Model（文档对象模型），简称DOM，提供访问和修改网页数据的**标准化**机制。
- DOM视网页为关联节点的层次**树**。
- 使用DOM（而非innerHTML）改变网页内容的方案，需移除元素下所有子节点，然后创建并附加上包含新内容的新子节点。

与标准兼容的大冒险

听起来真好玩！任何有趣冒险的旅途标志都做了标准兼容……也可能没有。但在当代网络应用程序的状况下，与标准兼容可能还不错。更重要的是，你看 DOM 对修改《火柴人大冒险》场景说明文本的方式，带来了多么巨大的改变……



咦……好吧，网页看起来没什么不一样，但它在台面下使用了 DOM，从而遵守了最新的万维网标准。JavaScript代码也有不为肉眼所察觉的一面，像我们看到 DOM 强化版《火柴人大冒险》时的满足感，就来自它的内在。

DOM是种符合万维网标准的HTML操纵方式，它能比innerHTML特性达成更多操控功能。

寻找更佳选择

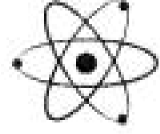
动态的场景说明已经做了两次进厂大维修，但神秘隐晦的选项按钮仍然待在《火柴人大冒险》的页面上。我们一定能让故事导览过程比起选 1 或选 2 更有参与性，更为直观简单。

Please choose: 1 2

只有数字的选项实在很不好——它们根本没有说明用户面对的抉择。

选项按钮要有像样的改善，想必需要变化为实际反映可用选项的外观。按钮上的文字可包含目前场景中的两个实际选项，如下所示：

再仔细想想，没人规定非得利用表单按钮来做选项 —— 只要是可以包含文本的 HTML 元素都能妥善运作。CSS 样式则可妆点元素，让元素看来像输入控制类的元素。



动动脑

在《火柴人大冒险》中该如何实践由数据驱动的选项，才能让选项依据各幕场景，呈现专用的选项说明文字呢？

设计更好、更干净的选项

既然《火柴人大冒险》的全新改良版决策选项，将是能包含文本的HTML元素，DOM 即可进场，用于动态改变每幕场景的说明文本。因此每幕场景将依据场景说明而设定决策选项。而且，这也表示changeScene()函数需要两个新变量，以存储决策说明文本decision1与decision 2。

下面示范如何在changeScene()函数中把Scene 1转向Scene 3后，设定Scene 1的决策说明：

```
curScene = 3;
message = "You are standing on the bridge overlooking a peaceful stream.";
decision1 = "Walk across Bridge";
decision2 = "Gaze into Stream";
```

磨笔上阵

想为《火柴人大冒险》提供动态选项，我们需要新的HTML代码表达方式。请为新的文本元素设计代码，用于取代现有的<input>按钮。

提示：赋予新元素的 CSS 样式类（class）称为“decision”，第一个元素内容的初始设定则为“Start Game”。

Please choose: 1 2

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

为了制作动态选项，重新设计代码吧！

Start Game

....

....

磨笔上阵解答

想为《火柴人大冒险》提供动态选项，我们需要新的HTML代码表达方式。请为新的文本元素设计代码，用于取代现有的 <input>按钮。

提示：赋予新元素的CSS样式类（class）称为“decision”，第一个元素内容的初始设定则为“Start Game”。

Please choose: 1 2

```
<input type="button" id="decision1" value="1" onclick="changeScene(1)" />
<input type="button" id="decision2" value="2" onclick="changeScene(2)" />
```

为了制作动态选项，重新设计代码……原本的两个按钮变成 span 元素。

Start Game

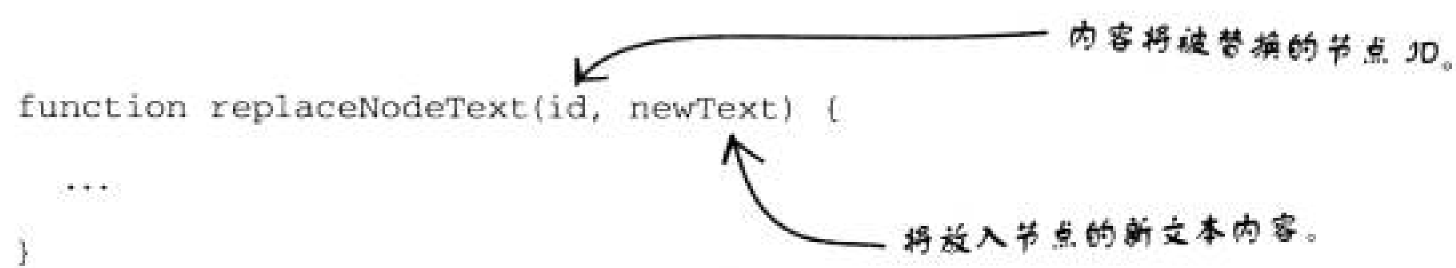
```
<span id= "decision1" class= "decision" onclick= "changeScene(1)" >Start Game</span>
<span id= "decision2" class= "decision" onclick= "changeScene(2)" ></span>
```

重新思考节点文本替换功能

《火柴人大冒险》的全新动态决策说明机制，现在只差实际为 span元素设定文本的代码。这段代码最终将与稍早的动态改变场景说明文本的DOM代码执行相同工作。事实上，我们因此看到了问题 —— 现在需要对三个不同的元素运用完全相同的文本替换任务，包括：场景说明元素，以及两个场景决策元素……

改用函数取代节点文本

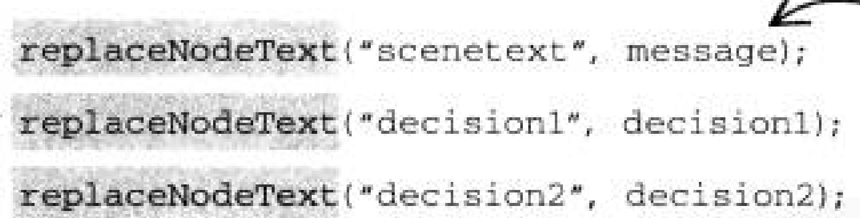
有个万用的节点文本替换函数将非常方便，而且不只《火柴人大冒险》有此需要。这类函数就像我们稍早研究的场景说明取代代码，不过这次的代码要处理函数自变量。




```
function replaceNodeText(id, newText) {
  ...
}
```

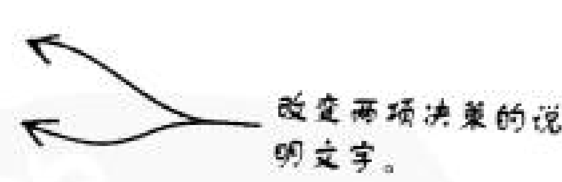
自定义的replaceNodeText()函数接受两个自变量：内容将被替换的节点ID，以及准备放入节点的新文本内容。使用这个函数，可改变网页上任何文本存储元素的内容。以《火柴人大冒险》为例，这个函数现在能让我们动态改变场景说明，以及同时改变两个场景决策选项的说明……当然，你得先把它写出来。

与其三度复制相同代码，我们改让函数被调用三次。



```
replaceNodeText("scenetext", message);
replaceNodeText("decision1", decision1);
replaceNodeText("decision2", decision2);
```

以新信息替代场景说明。



设计replaceNodeText()函数的代码，它是替换节点内文本的万用函数，而节点透过ID被引用。

别忘了函数可接受两个自变量，id与newText。

....

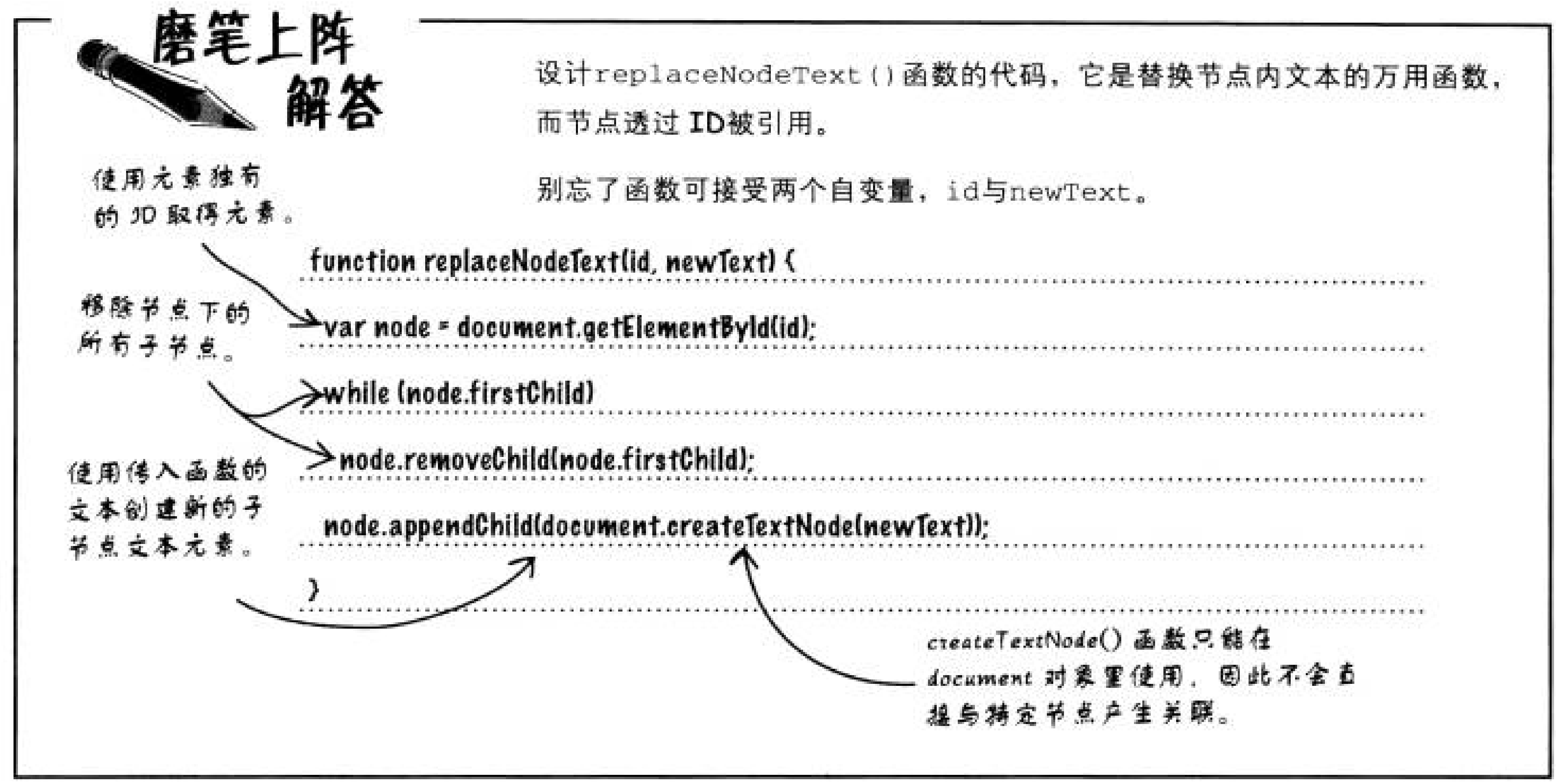
....

....

....

....

....



磨笔上阵解答

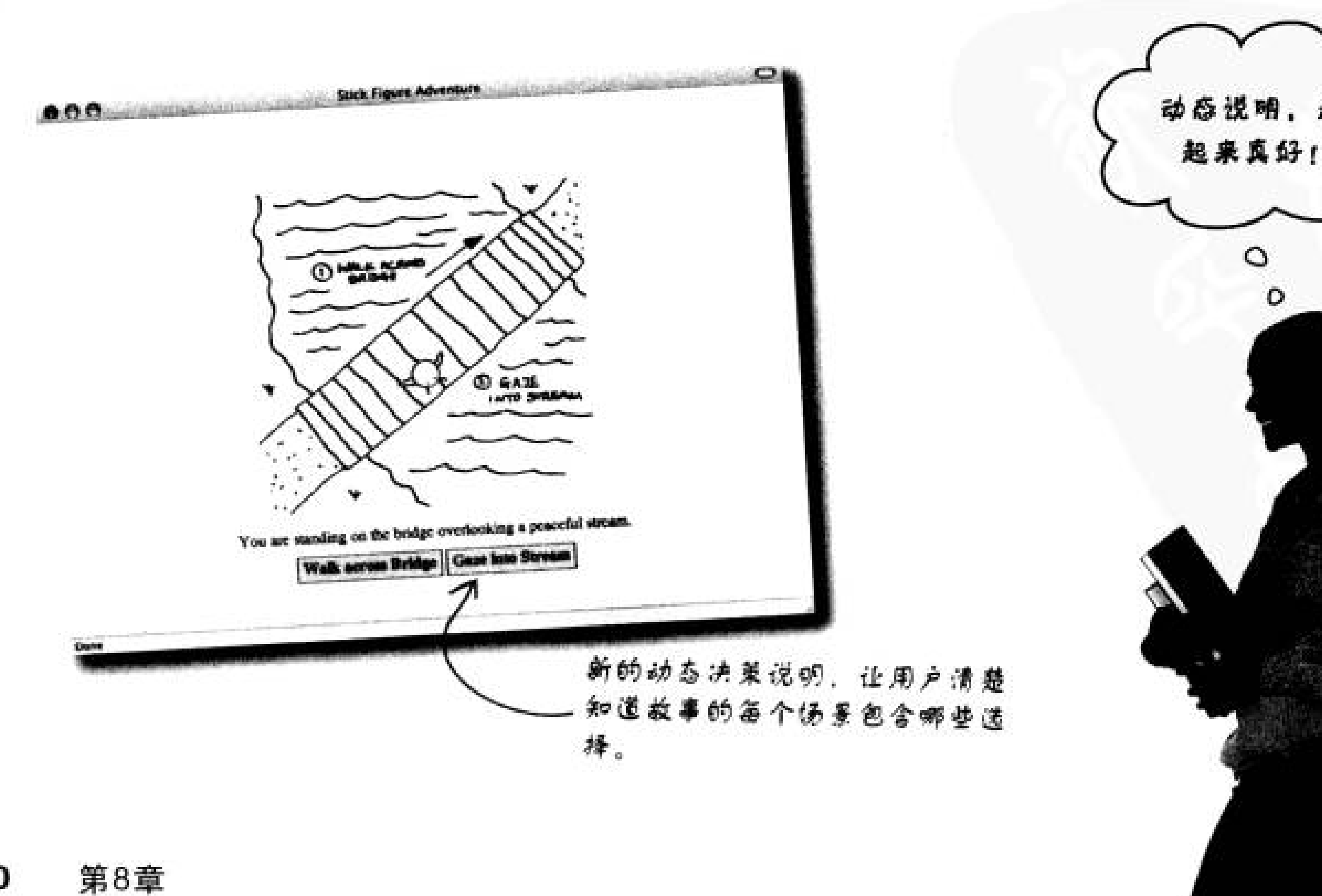
设计replaceNodeText()函数的代码，它是替换节点内文本的万用函数，而节点透过 ID被引用。

别忘了函数可接受两个自变量，id与newText。

```
function replaceNodeText(id, newText) {
  var node = document.getElementById(id);
  while (node.firstChild)
    node.removeChild(node.firstChild);
  node.appendChild(document.createTextNode(newText));
}
```


动态选项真正好

新加入《火柴人大冒险》的动态决策文本，比起旧版的按钮直观多了。



问：为什么采用span作为《火柴人大冒险》的判断元素，而不是div呢？

答：因为决策元素需要并排呈现，它们不能是需要换行的块元素（block element）。div是个块元素，但span则是行内元素（inline element）。决策需要行内元素，所以采用span。

问：当我使用createTextNode()创建新节点时，节点会跑到哪里？

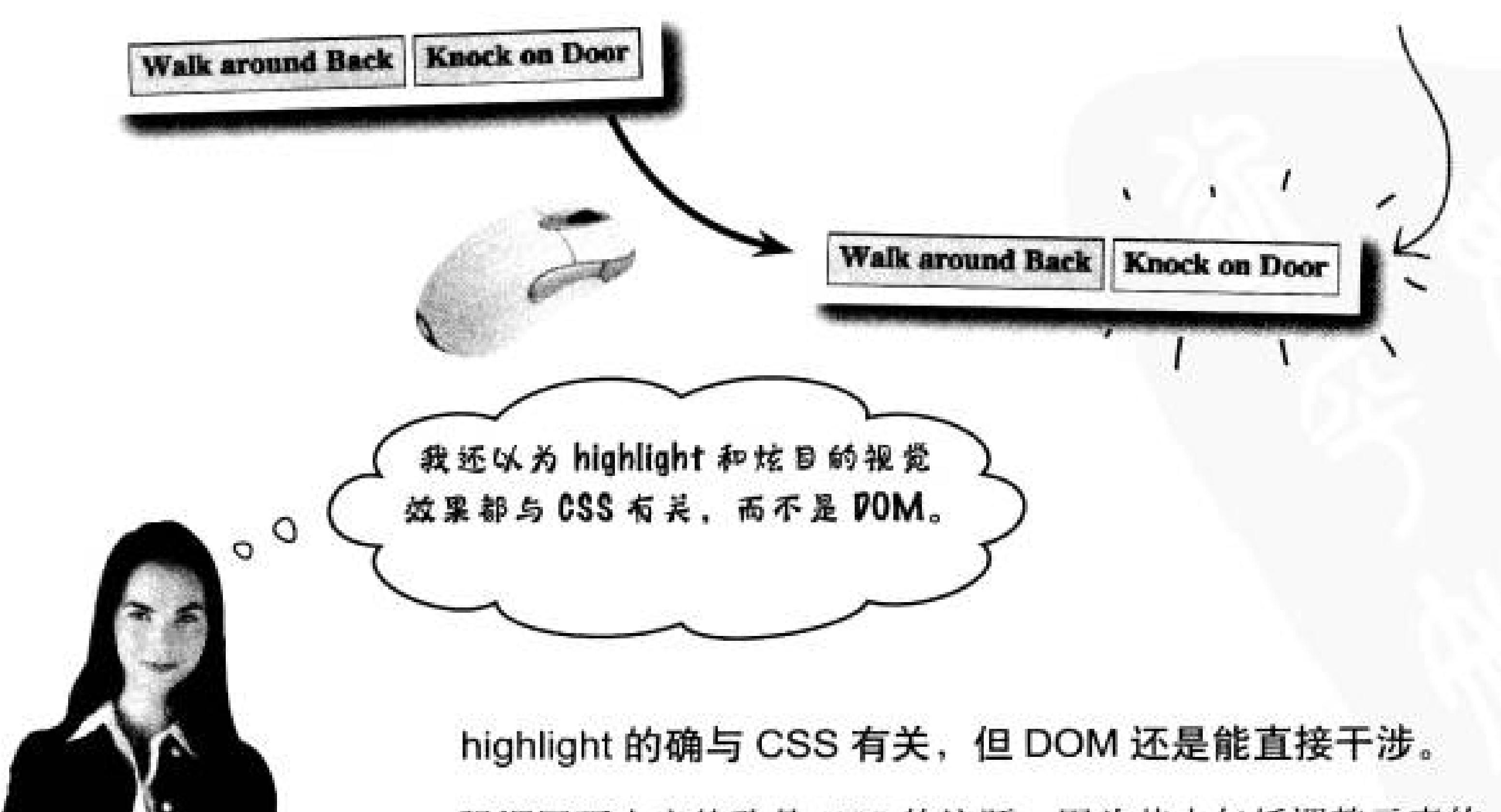
答：哪里都不会去。从某个网页的DOM树的角度来看，刚开始创建新的文本节点时，它飘浮在空中。直到你附加它为另一个节点的子节点，它才真正加入节点树，然后才加入网页里。

问：以createTextNode()创建的文本节点内容只能是文本吗？

答：是的。DOM的作用方式不像innerHTML，后者可以指派混合了标签的文本。当DOM提到"文本节点"，它的意思真的是单纯、没有其他标签或附加格式的文本。

更加改良的交互选项

相对于晦涩不清的按钮前辈，《火柴人大冒险》的动态决策文本已是改良的，但百尺竿头总能更进一步。举例来说，还可以在鼠标指针移过动态文本元素时自我强调（highlight），显示出可以按下去的样子。



highlight的确与CSS有关，但DOM还是能直接干涉。

强调网页内容的确是CSS的议题，因为其中包括调整元素的背景颜色等等。但DOM也是强调等式的一部分，因为它提供了对元素的CSS样式的访问……

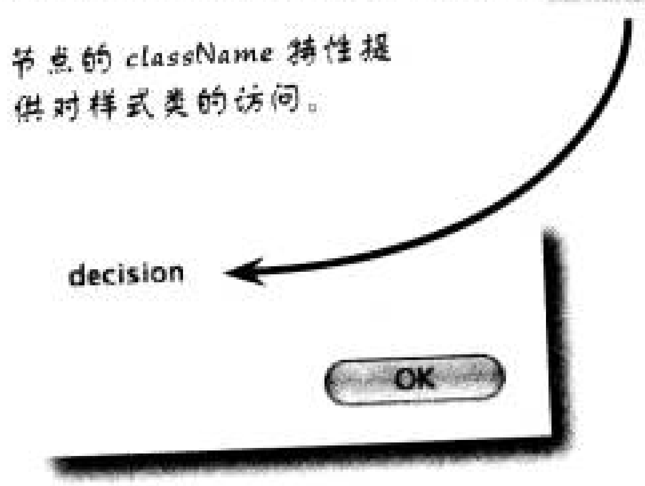
样式的问题：CSS 与 DOM

CSS 样式与 HTML 元素紧密相联，DOM 则提供了透过元素（节点）访问样式的途径。借由使用 DOM 调整 CSS 样式，即有了动态操纵内容外观的可能性。透过 DOM 揭露 CSS 样式的方式之一，就是利用元素的 style类 —— 应用一组样式（一个类）至元素的地方。

```
<span id="decision1" class="decision" onclick="changeScene(1)">Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"></span>
```

DOM 透过节点对象的 className特性，提供对元素样式类的访问。

```
alert(document.getElementById("decision1").className);
```



className 特性提供对元素样式类的访问。

注意！

尽量别把CSS的样式类与JavaScript的类搞混了。

CSS样式类和JavaScript的类，两者的差异非常大。CSS样式类是一组样式，可应用至网页中的元素；而 JavaScript 的类是个创建JavaScript对象的模板。我们将于第10章详细说明JavaScript的类和对象。

交换样式类

想使用完全不一样的样式类改变元素外观，只需改为另一个 CSS 样式类的名称。

```
document.getElementById("decision1").className = "decisioninverse";
```

decisioninverse 样式类反转决策文字的配色。

相同的决策元素，不同的配色！

使用 className，应用新的样式类在决策元素上。

Start Game

```
<style type="text/css">
  span.decisioninverse {
    font-weight:bold;
    color:#FFFFFF;
    border:thin solid #DDDDDD;
    padding:5px;
    background-color:#000000;
  }
</style>
```

使用className特性改变元素样式类，立即改变元素外观为新样式。这项技巧可用于制造非常戏剧性的网页元素外观变化，而又只需要相对较少的编程工夫。

磨笔上阵

请使用两个与鼠标有关的事件 onmouseover 与 onmouseout，对《火柴人大冒险》的 <span>决策元素添加代码，让元素于鼠标指针移过时，出现样式改变的强调效果。

提示：鼠标移过（hover）效果的样式效果名称为decisionhover。

```
<span id="decision1" class="decision" onclick="changeScene(1)"
  ......................................................................
  ......................................................>Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"
  ......................................................................
  ......................................................></span>
```

磨笔上阵 解答

请使用两个与鼠标有关的事件onmouseover与onmouseout，对《火柴人大冒险》的 <span>决策元素添加代码，让元素于鼠标指针移过时，出现样式改变的强调效果。

提示：鼠标移过（hover）效果的样式效果名称为decisionhover。

```
<span id="decision1" class="decision" onclick="changeScene(1)"
onmouseover=" this.className = 'decisionhover' "
onmouseout=" this.className = 'decision' "  >Start Game</span>
<span id="decision2" class="decision" onclick="changeScene(2)"
onmouseover=" this.className = 'decisionhover' "
onmouseout=" this.className = 'decision' "  ></span>
```

decisionhover 样式效果设定为响应 onmouseover 事件。

当鼠标指针移过 span 元素时，触发这个事件。

去除强调效果后的决策样式，重新存储到 onmouseover 事件。

鼠标指针移开 span 元素时，触发这个事件。

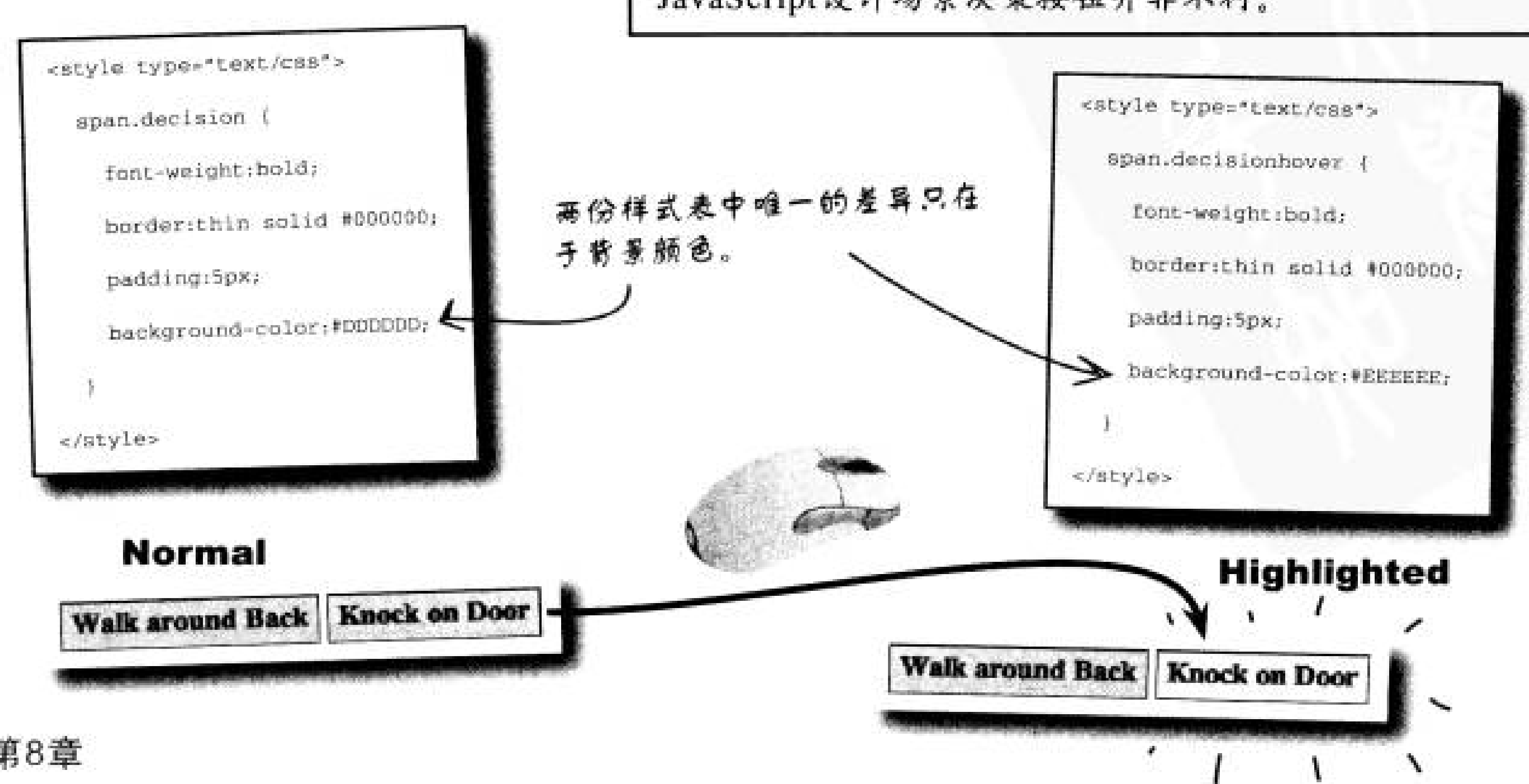
没有蠢问题

问：要创建于鼠标指针移过时出现强调效果的按钮，我不能只使用 CSS 吗？

答：可以。而且在很多状况下，只使用CSS创建强调效果的按钮才是比较好的方式，因为各类浏览器对CSS的支持程度，比对JavaScript的支持更广泛，例如在某些移动设备上。然而，《火柴人大冒险》是个 JavaScript 应用程序，可做到所有无法单独以CSS达成的效果。所以，在这个例子里，使用JavaScript设计场景决策按钮并非不利。

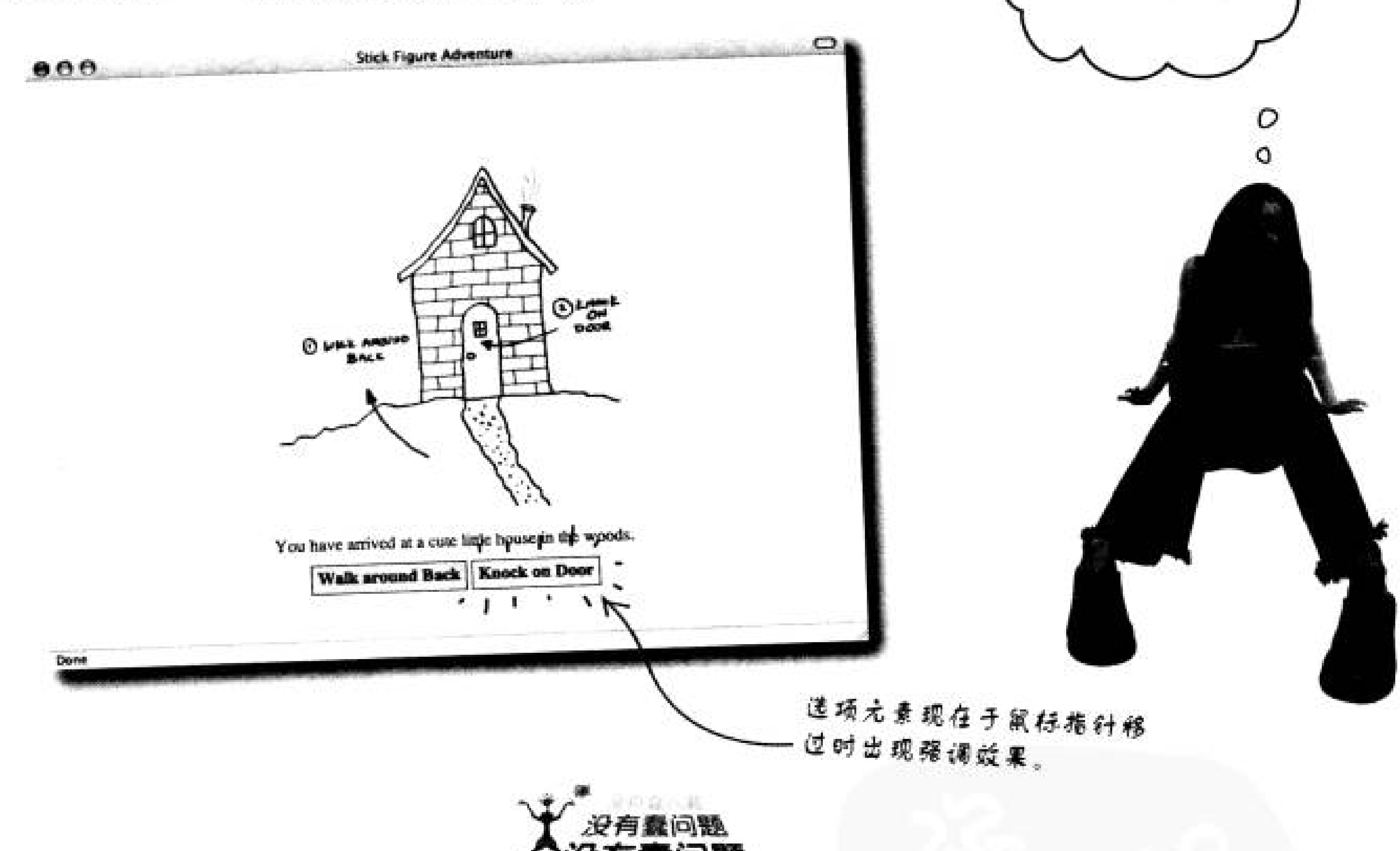
有格调的选项

应用样式类至《火柴人大冒险》的代码，为决策元素产生两种完全不同的外观：正常版与强调版。



测试加上样式的选项

幸好 DOM 能够应观众要求改变元素的样式类，《火柴人大冒险》的用户界面做了改善。Ellie 现在觉得她的脚本更好了。



没有蠢问题

问：我不记得听过 onmouseover 与onmouseout事件，它们也是标准事件吗？

答：是的。事实上，我们还有很多标准JavaScript事件没有讨论。但是，事件的重点在于我们如何对事件反应，就算不见得清楚知道事件的一切。以两个鼠标相关事件为例，只要看到它们的名称，你就大概知道其中之一于鼠标指针移过（over）元素时发生，另一个则于鼠标指针移出（out）元素时发生。

问：为什么在设定决策元素的样式类时，不需要使用 getElementById() 呢？

答：JavaScript的每个元素都是一个对象，在HTML代码中我们能透过关键字this访问相对于某个元素的对象。所以在《火柴人大冒险》代码中，关键字this即可为span元素引用节点对象，这个对象则具有访问该元素样式类的className特性。所以，改变样式类只需设定this.className。

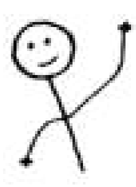
问：样式类是不错，但我只想改变一个样式特性而已。有可能办到吗？

答：哇，你的直觉真准！Ellie 其实也很想解决《火柴人大冒险》这个问题。解决方式则刚好与使用 JavaScript 和 DOM 单独操纵样式特性有关……

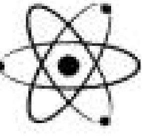
选项有问题：空按钮

《火柴人大冒险》里一直都有这个问题，Ellie 到目前为止都在忍耐。但现在应该起身处理那些怪异的空选项了。在某些场景里，其实只有一个选项，但两个决策元素仍然都会出现，如下图所示。对用户而言，看到没有信息的决策元素实在有点浑身不舒服。

有些场景出现了空选项的事情已经困扰我一阵子了。空选项没有意义，而且只会造成困惑。



空选项元素既奇怪又让人糊涂。



动动脑

还有哪些场景具有空选项的问题？有什么修正的方式呢？

调整单项样式

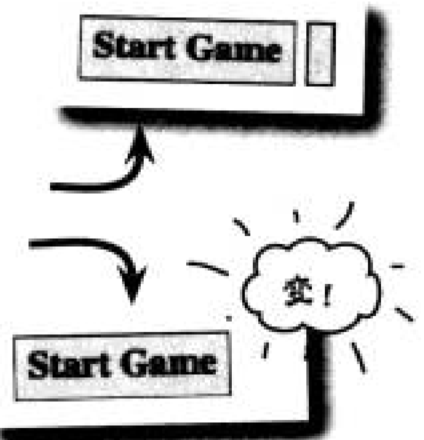
有时候，改变元素的整个样式类实在太过火了。需要细微调整的时候，就轮到style对象为我们服务了。CSS 样式里的 visibility 特性可设定元素的显示或隐藏。在《火柴人大冒险》的 HTML 中，可将第二个决策元素预设为隐藏，如下所示：

节点的 style 特性提供对单一样式特性的访问。

```
<span id="decision2" class="decision" onclick="changeScene(2)"
  onmouseover="this.className = 'decisionhover'"
  onmouseout="this.className = 'decision'"
  style="visibility:hidden"></span>
```

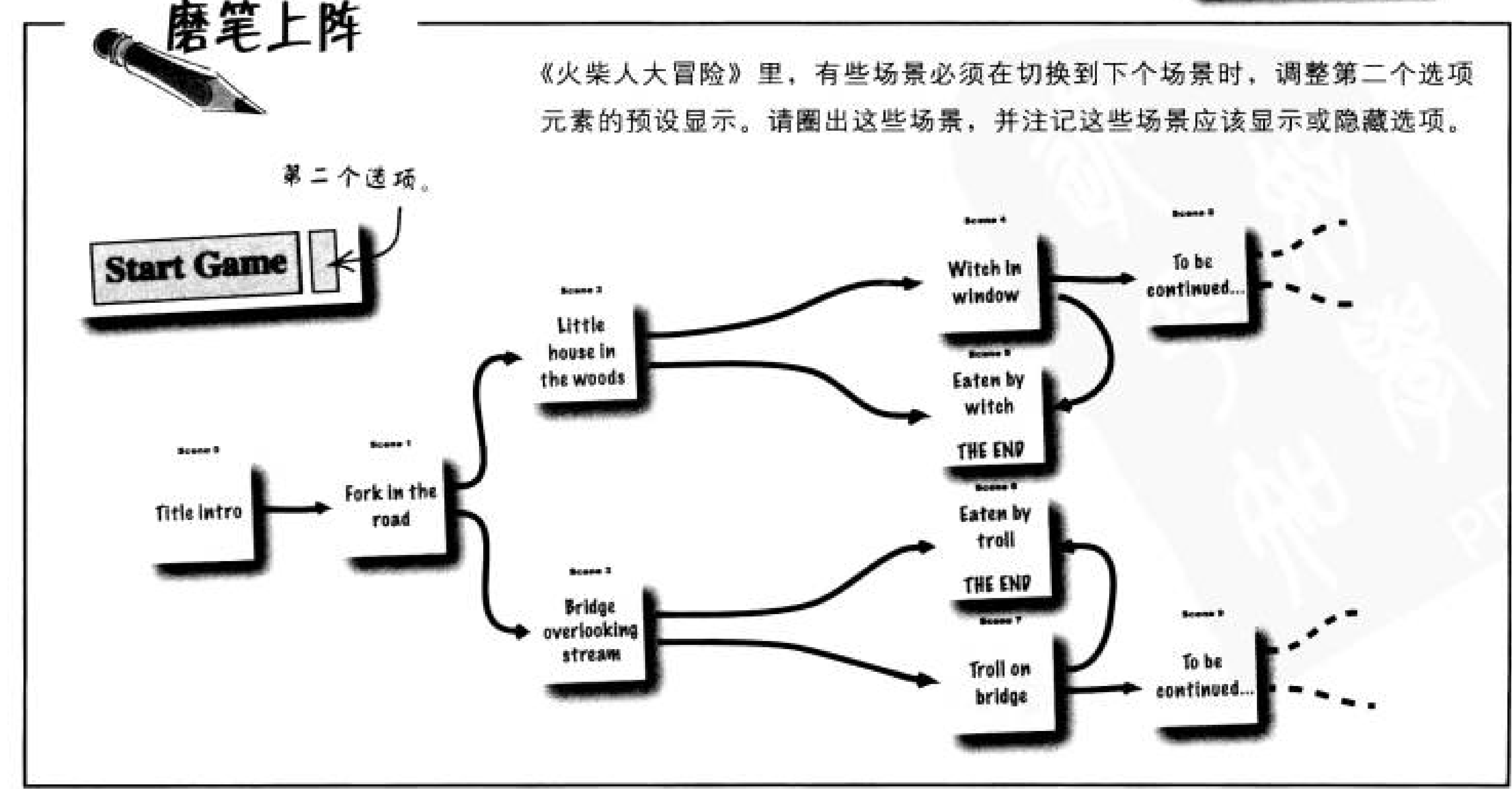
从此之后，显示与隐藏元素只是设定 visibility 样式特性为 visible 或 hidden 而已。

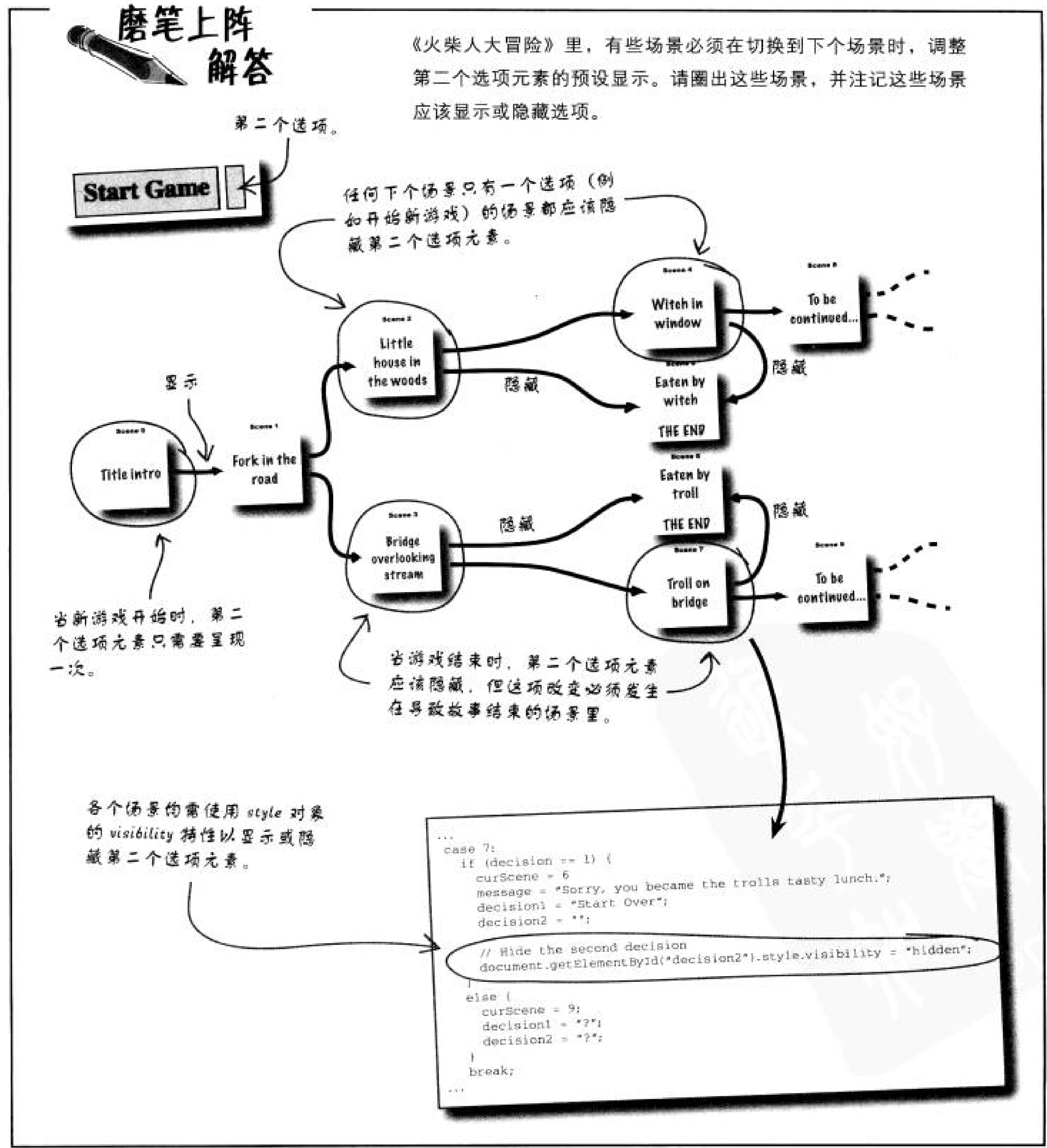
```
document.getElementById("decision2").style.visibility = "visible";
document.getElementById("decision2").style.visibility = "hidden";
```



磨笔上阵

《火柴人大冒险》里，有些场景必须在切换到下个场景时，调整第二个选项元素的预设显示。请圈出这些场景，并注记这些场景应该显示或隐藏选项。






```
...
case 7:
  if (decision == 1) {
    curScene = 6
    message = "Sorry, you became the trolls tasty lunch.";
    decision1 = "Start Over";
    decision2 = "";

    // Hide the second decision
    document.getElementById("decision2").style.visibility = "hidden";
  }
  else {
    curScene = 9;
    decision1 = "?";
    decision2 = "?";
  }
  break;
...
```

复习要点

- 借由改变节点的**整份样式类**，className节点特性达成戏剧性的样式变化。
- 借由访问节点的**单一样式特性**，style节点特性达成少量样式变化。
- CSS 样式类与JavaScript类完全无关 —— 它们是完全不同的东西。
- 网页元素可利用元素对象的visibiliey样式特性做动态的显示或隐藏。

样式特性display亦可达成相似的显隐效果，设定方式为：display:none（隐藏）或display:block（显示）。

不再出现假选项

使用 DOM 操纵单一样式，即可选择第二个选项元素的显示或隐藏。结果则让用户界面更合理，因为空选项元素已经消失了。

Stick Figure Adventure

Welcome to

STICK FIGURE ADVENTURE

Click button to start...

Start Game

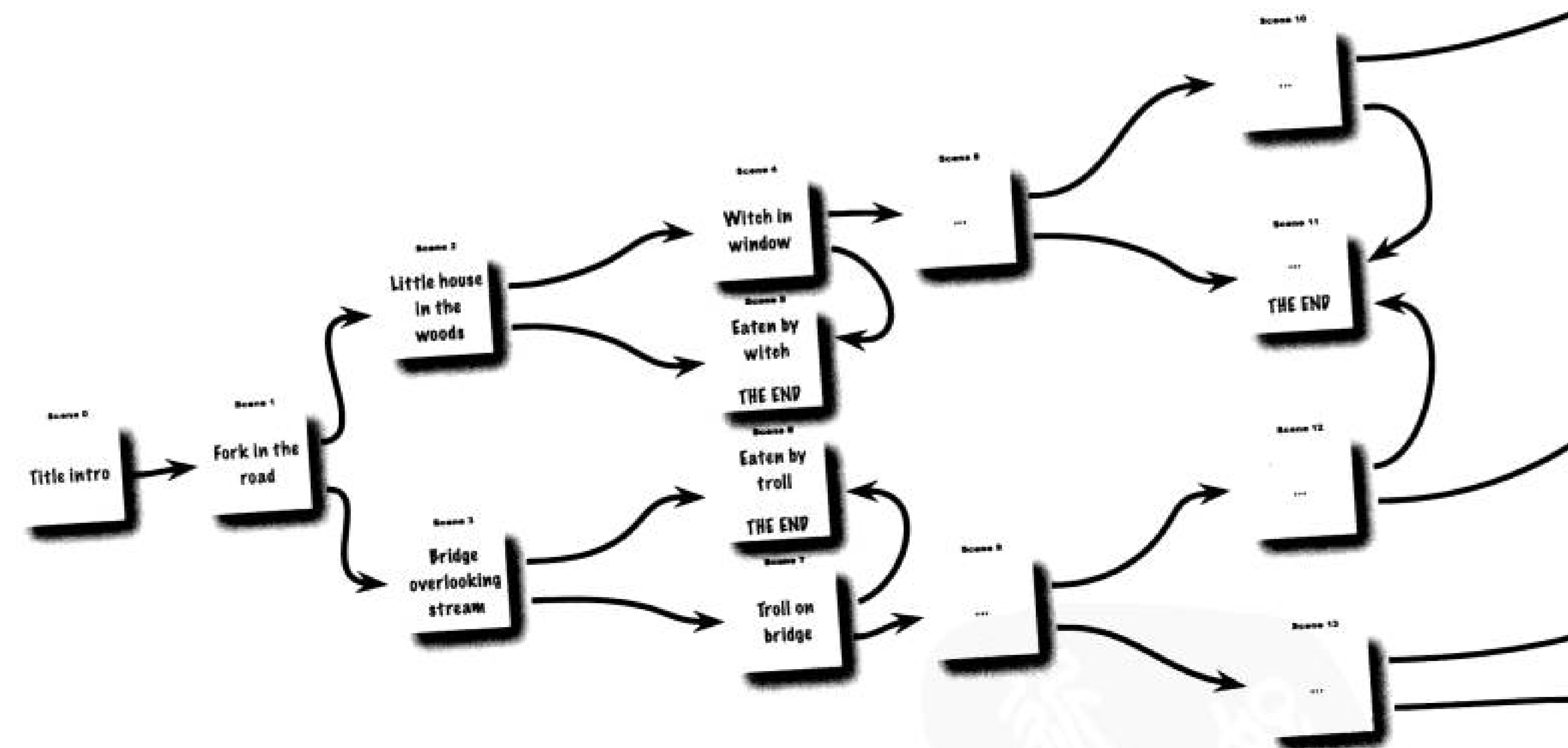
Done

好多了…………这些空选项实在太烦人了！

不需要第二个选项元素时，现在可以把它隐藏起来，就像片头场景的示范一样。

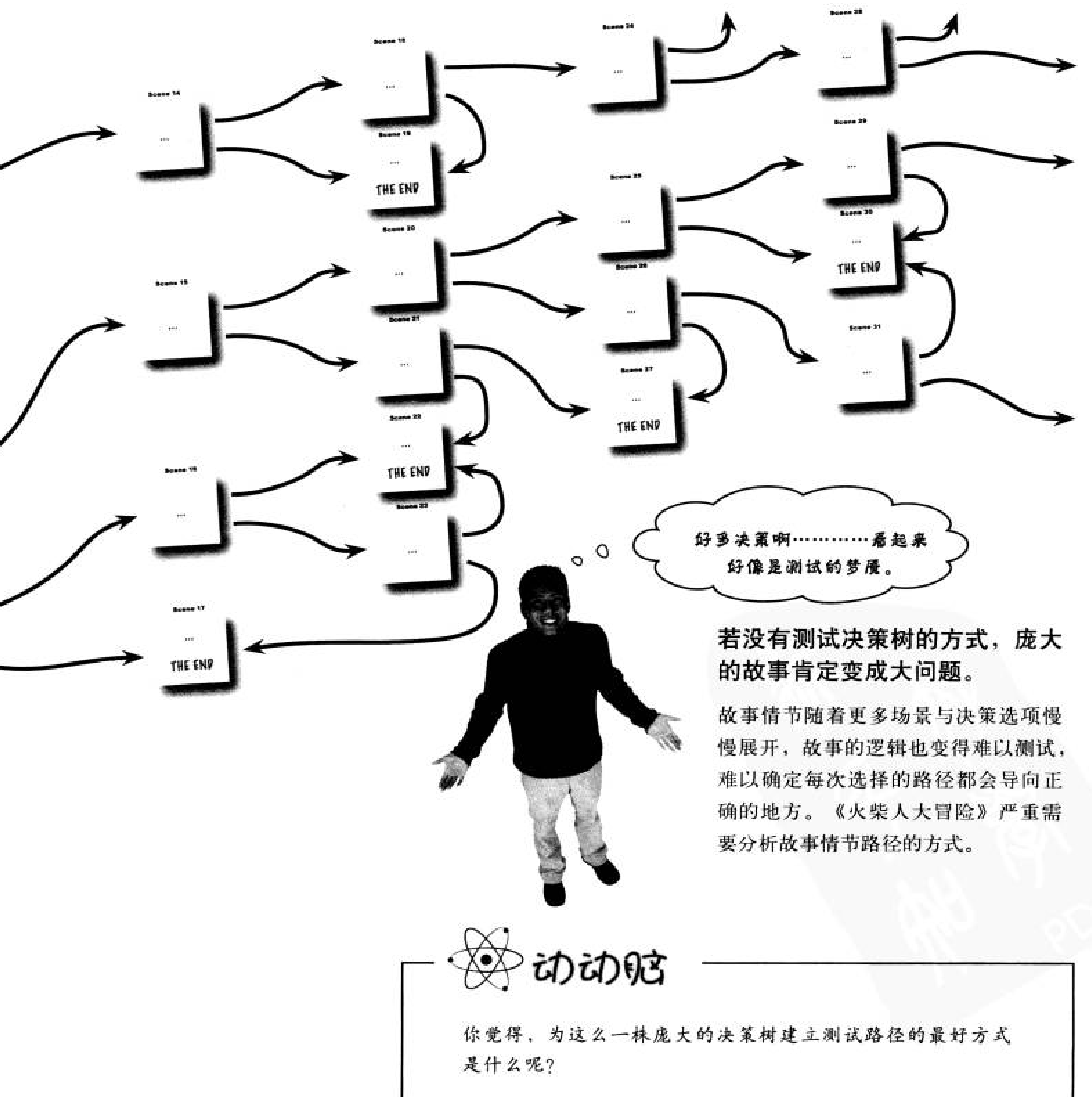
更多选项，更为复杂

Ellie想象着《火柴人大冒险》的故事情节飞跃性地成长，呈现各种有趣的新场景与决策选项。DOM 有很多协助处理《火柴人大冒险》的更深层复杂性的方式。



更深入的冒险 = 更大的决策树！

*最新版《火柴人大冒险》已经上线，等待你的编码协助。请至 http://www.headfirstlabs.com/books/hfjs/ 下载。



若没有测试决策树的方式，庞大的故事肯定变成大问题。

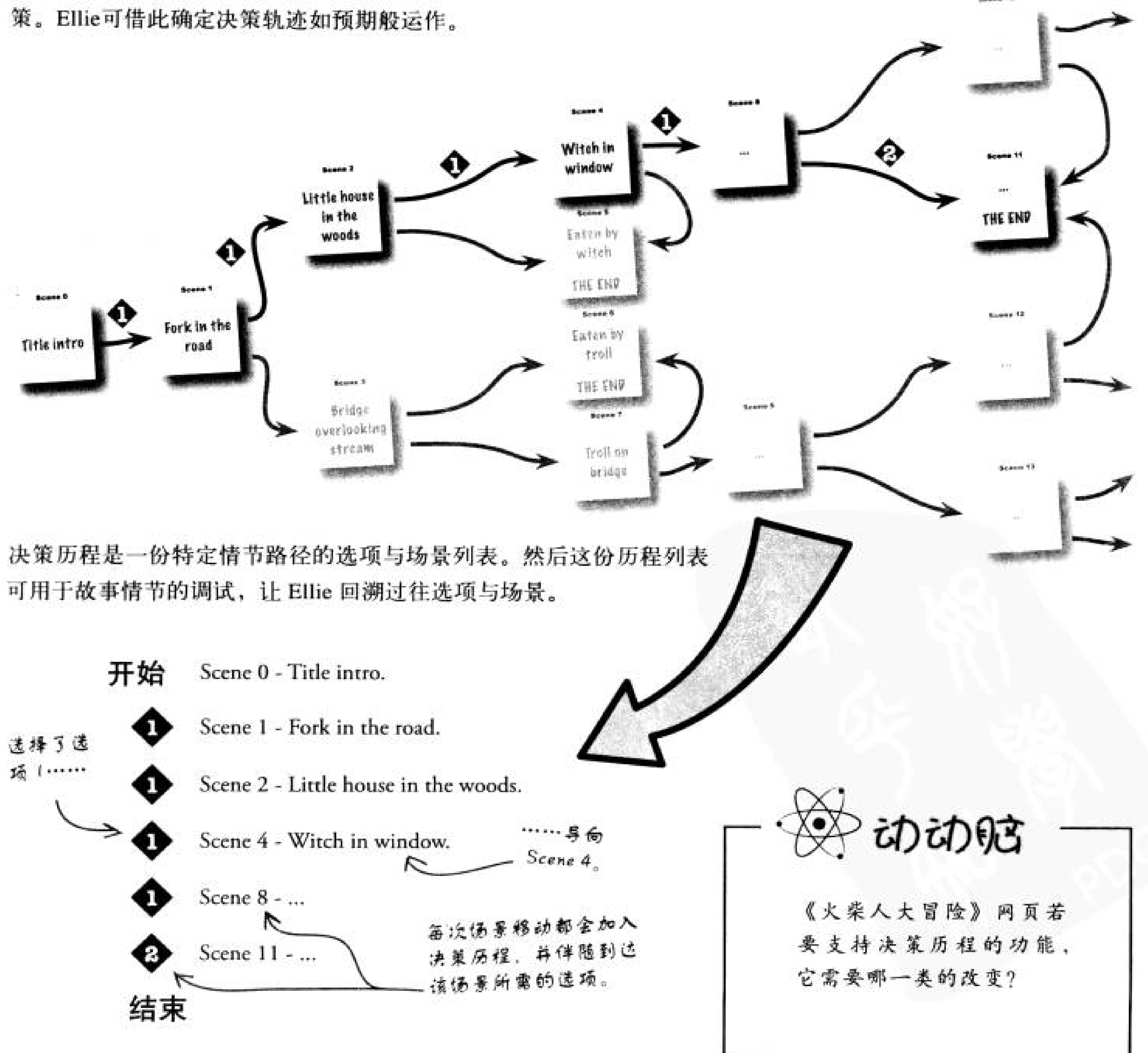
故事情节随着更多场景与决策选项慢慢展开，故事的逻辑也变得难以测试，难以确定每次选择的路径都会导向正确的地方。《火柴人大冒险》严重需要分析故事情节路径的方式。

动动脑

你觉得，为这么一株庞大的决策树建立测试路径的最好方式是什么呢？

追踪决策树的轨迹

网络浏览器有个历程记录功能，可追踪我们浏览过的网页，《火柴人大冒险》的决策树历程记录功能则可测试故事情节并调试。我们打算揭示一系列导向特定结局的决策。Ellie可借此确定决策轨迹如预期般运作。



决策历程是一份特定情节路径的选项与场景列表。然后这份历程列表可用于故事情节的调试，让 Ellie 回溯过往选项与场景。

开始 Scene 0 - Title intro.

1 Scene 1 - Fork in the road.

1 Scene 2 - Little house in the woods.

1 Scene 4 - Witch in window.

1 Scene 8 - ...

2 Scene 11 - ...

结束

动动脑

《火柴人大冒险》网页若要支持决策历程的功能，它需要哪一类的改变？

把决策历程改为 HTML

从 HTML 的角度来看，决策历程的代码还不算太复杂：div元素加上记录选项的**文本段落**（p），只需要这样就够了。

```
<div id="history">
  <p>Decision 1 -> Scene 1 : Fork in the road.</p>
  <p>Decision 1 -> Scene 2 : Little house in the woods.</p>
  <p>Decision 1 -> Scene 4 : Witch in window.</p>
  ...
</div>
```

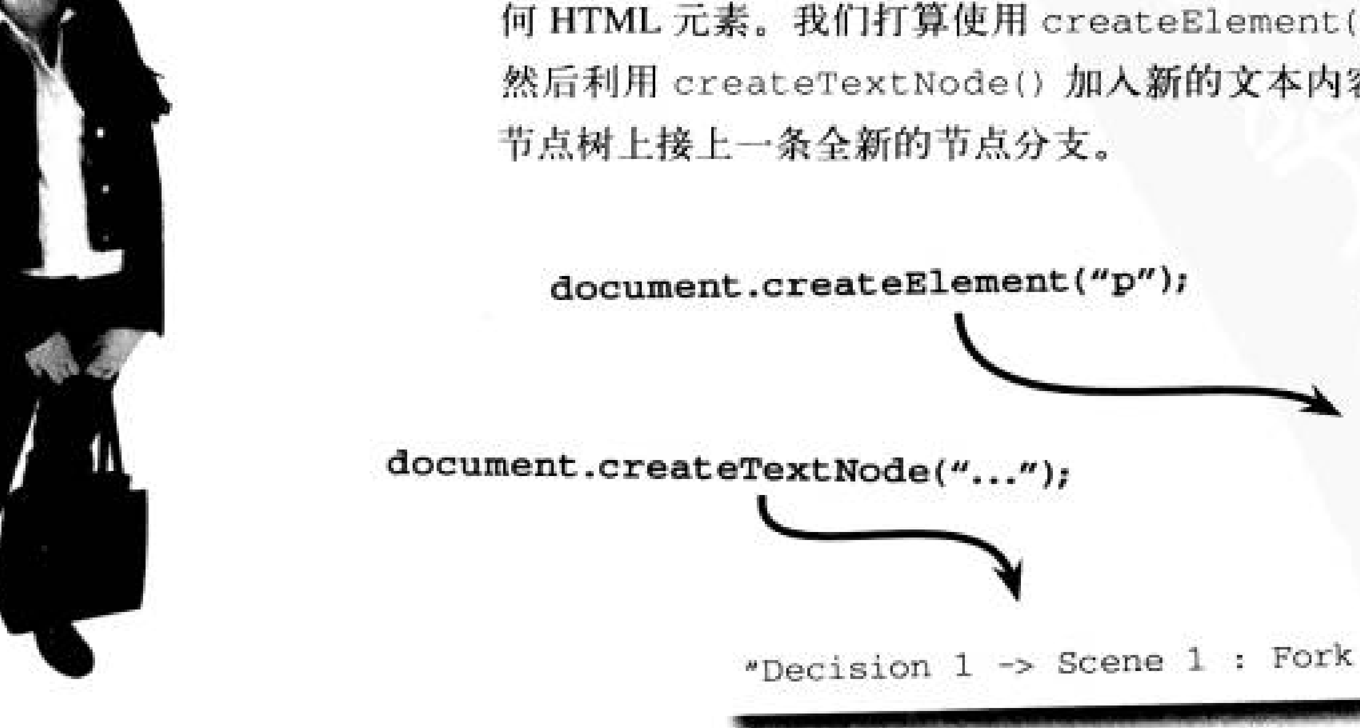
每个p元素均包含决策历程中的一次选择。

剩下来的工作，就是让 JavaScript 利用 DOM 产生决策历程（形成一组节点）。

《火柴人大冒险》的决策历程，能成为非常好用的故事情节调试工具。

DOM 能随意创建任何HTML元素，当然包括文本段落。

事实上，就是可以随意创建。现在牵涉到另一个document对象的方法——createElement()，顾名思义，这个方法可创建任何 HTML 元素。我们打算使用 createElement()创建新的容器元素，然后利用 createTextNode() 加入新的文本内容。最后将在网页的节点树上接上一条全新的节点分支。



操纵HTML代码

使用createTextNode()方法创建新元素只需要标签名称。所以，创建段落文本（p），只需要调用这个方法并附上自变量“p”，请确定你会紧紧抓住创建的元素。

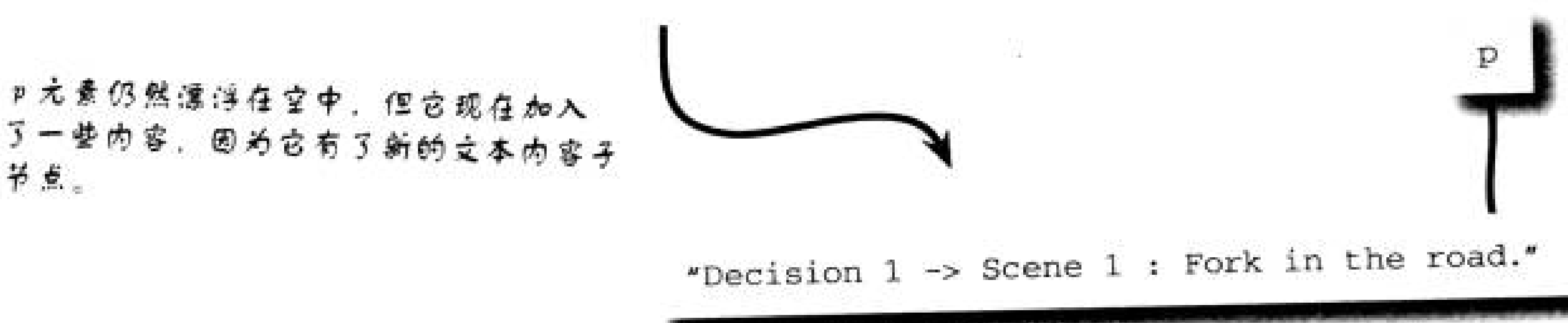
我们从漂浮在空中的全新p元素开始。

```
var decisionElem = document.createElement("p");
```

p

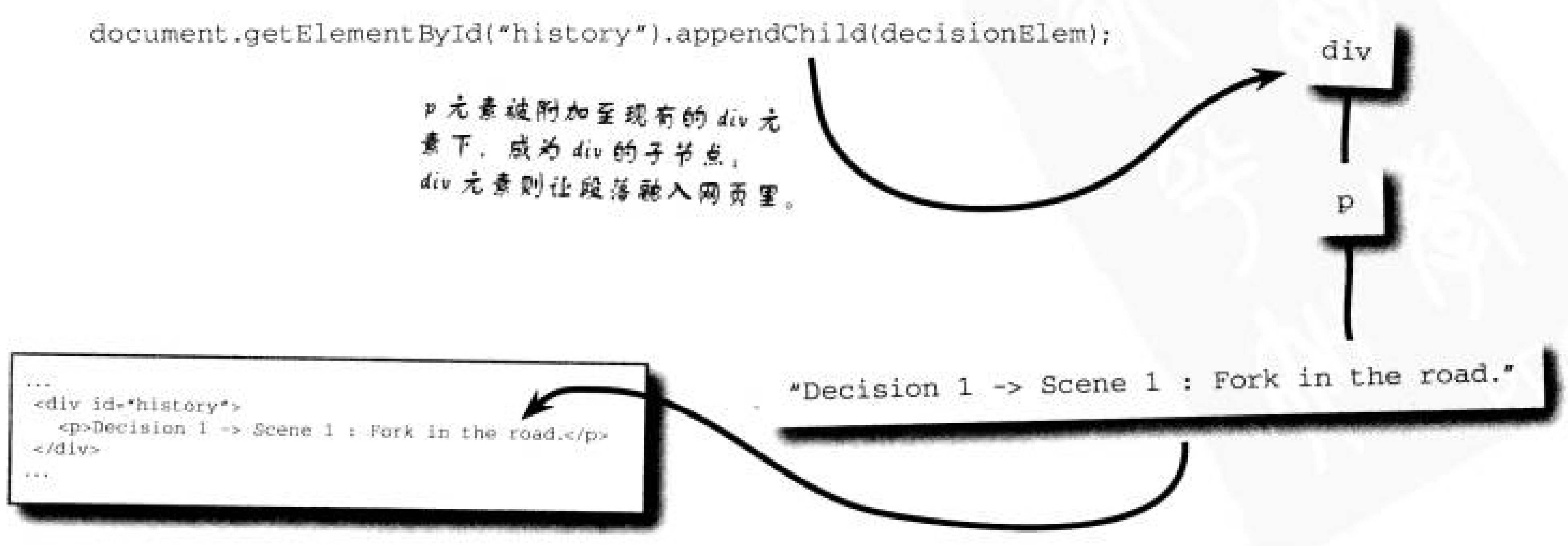
现在，有个新的、没有内容的段落元素，也还不是任何网页的一部分。若想对元素加入文本内容，则应创建文本节点，而后将其附加至新p节点下作为子节点。

```
decisionElem.appendChild(document.createTextNode("Decision 1 -> Scene 1 : Fork in the road."));
```



最后一步，则是在网页上新增段落元素，让它成为div历程记录元素的子节点。

```
document.getElementById("history").appendChild(decisionElem);
```



每次《火柴人大冒险》发生场景转换时重复上述步骤，即动态地创建了决策历程记录。

复习要点

- 利用document对象的createElement()方法，能够创建任何 HTML 元素。
- 若需新增元素的文本内容，必须创建一个文本内容子元素，并附加至元素下。
- 借由小心地新增与移除 DOM 树上的节点，网页可以随意拆解与重组。

磨笔上阵

请为《火柴人大冒险》的changeScene()函数新增程序代码，让它支持决策历程功能。提示：你需要于目前的场景不为 Scene 0 时，在决策历程元素下，新增附有文本子元素的段落元素，然后于 Scene（场景）为 0 时清除决策历程。

```
function changeScene(decision) {
  ...
  // Update the decision history
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
  .............................................
}
```

请为《火柴人大冒险》的changeScene()函数新增程序代码，让它支持决策历程功能。提示：你需要于目前的场景不为 Scene 0 时，在决策历程元素下，新增附有文本子元素的段落元素，然后于 Scene（场景）为 0 时清除决策历程。

```
function changeScene(decision) {
  ...
  // Update the decision history
  var history = document.getElementById( "history" );
  if (curScene != 0) {
    // Add the latest decision to the history
    var decisionElem = document.createElement( "p" );
    decisionElem.appendChild(document.createTextNode( "decision  " + decision +
      " -> Scene  " + curScene + "  :  " + message));
    history.appendChild(decisionElem);
  }
  else {
    // Clear the decision history
    while (history.firstChild)
      history.removeChild(history.firstChild);
  }
}
```

附加新文本节点至新段落元素里。

changeScene() 函数已经有个局部变量 decision，所以这个新变量必须取为别的名称。

使用 ID 抓住 div 历程元素。

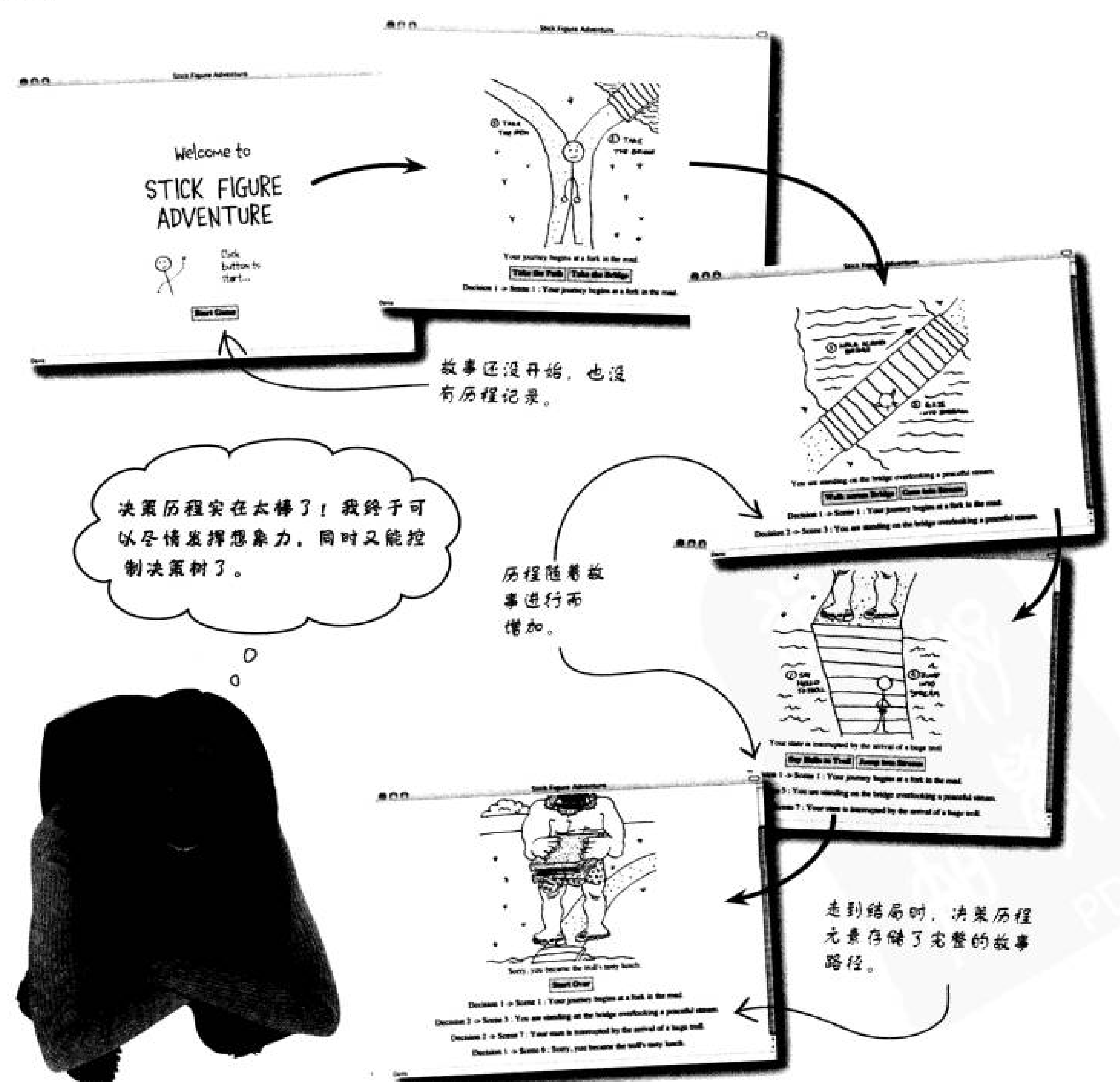
附加段落元素至 div 下，才能成为网页的一部分。

以决策历程信息创建新文本节点。

清除 div 历程元素——移除它的所有子节点。

追溯冒险历程

有了《火柴人大冒险》的决策历程，我们可以小心地沿途追踪故事情节了。



漫长而怪异的旅程……

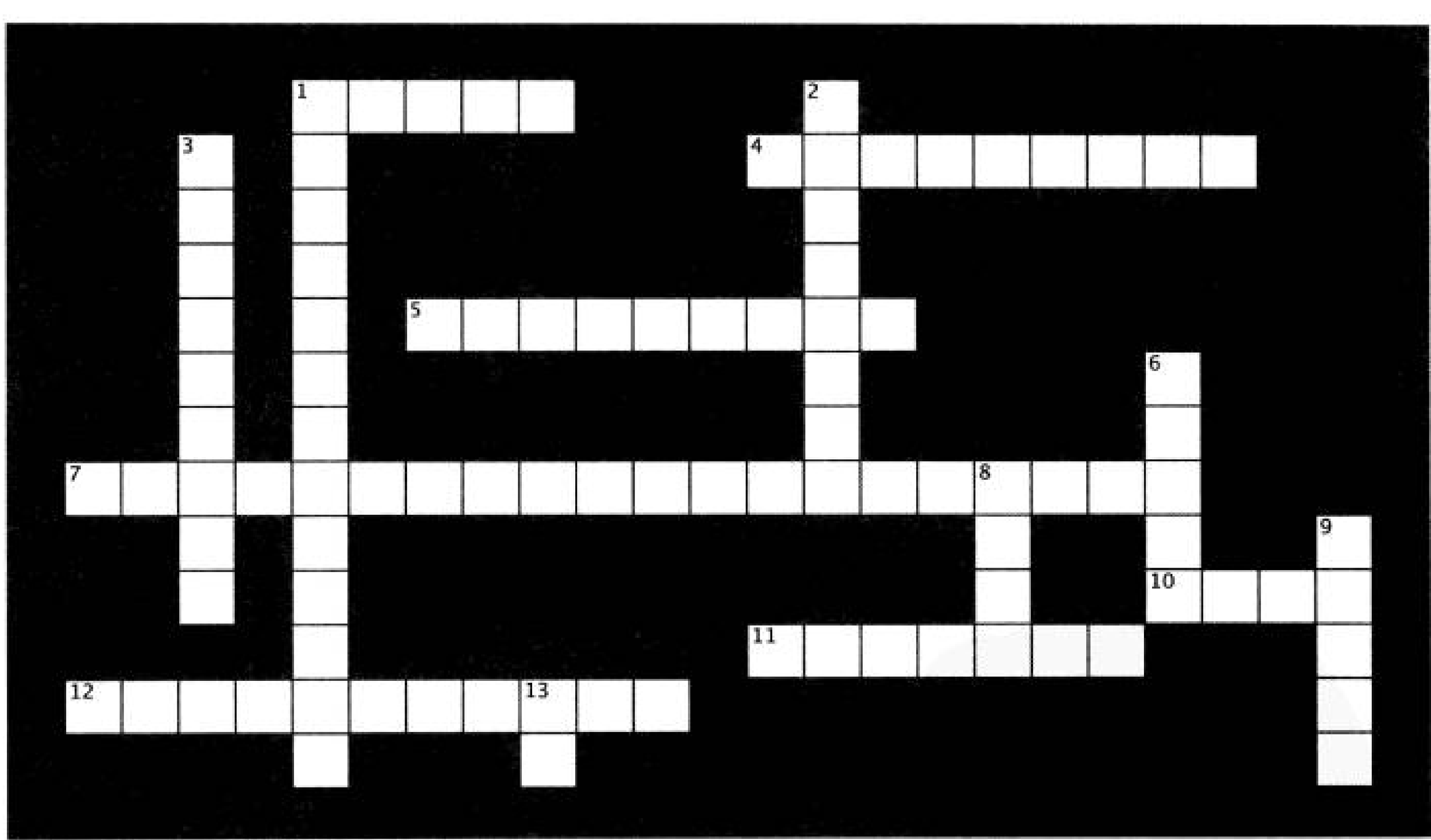
现在换你释放想象力，扩展《火柴人大冒险》的故事，让它的规模更适合决策历程调试。火柴人正等着新的冒险呢……

写出你自己的《火柴人大冒险》吧！增加一些能与现有《火柴人大冒险》程序相符的代码，成为可以和人分享的在线互动冒险。

这一题没有标准解答……开心地分享你的冒险幻想吧！

JavaScript 填字游戏

在你进一步投入火柴人的冒险前，轻松一下，体验我们的填字游戏吧！



横向提示：

1. 在 DOM 树里出现在另一个节点下的节点，称为________。
4. 用于取得节点对象值的特性。
5. 用于设定元素的样式类。
7. 调用________方法，以取得某一类对象。
10. 存储文本内容的节点类型。
11. 等同于HTML标签的DOM节点类型。
12. 在节点下新增子节点的方法（method）。

纵向提示：

1. 调用________方法可创建HTML元素。
2. DOM 树里最顶端的节点。
3. 用于改变 HTML 元素内容，但非标准的方式。
6. 讲述在线故事的笨拙方式。
8. 网页内容的 DOM 树上的一片叶子，称为________。
9. 用于访问元素的某个样式的特性。
13. 设定HTML标签的________属性，使标签可为JavaScript 访问。

JavaScript 填字游戏解答

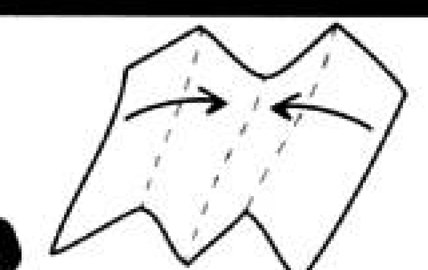
1 CHILD
1 CREATEELEMENT
2 DOCUMENT
3 INNERHTML
4 NODEVALUE
5 CLASSNAME
6 ALERT
7 GETELEMENTSBYTAGNAME
8 NODE
9 STYLE
10 TEXT
11 ELEMENT
12 APPENDCHILD
13 ID

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

DOM究竟是什么？

这是左右脑的秘密会谈



DOM

Document

html

body

head

""

div

""

The DOM

span

the browser,

span

is

strong

confused with

em

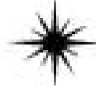
in control.

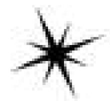
often

which is

使用JavaScript的设计师，都需小心别让DOM牵着走。DOM确实是个便于访问HTML标签的方式。但小心别变得太爱操纵一切，否则你可能让节点工作过度。

9 为数据带来生命



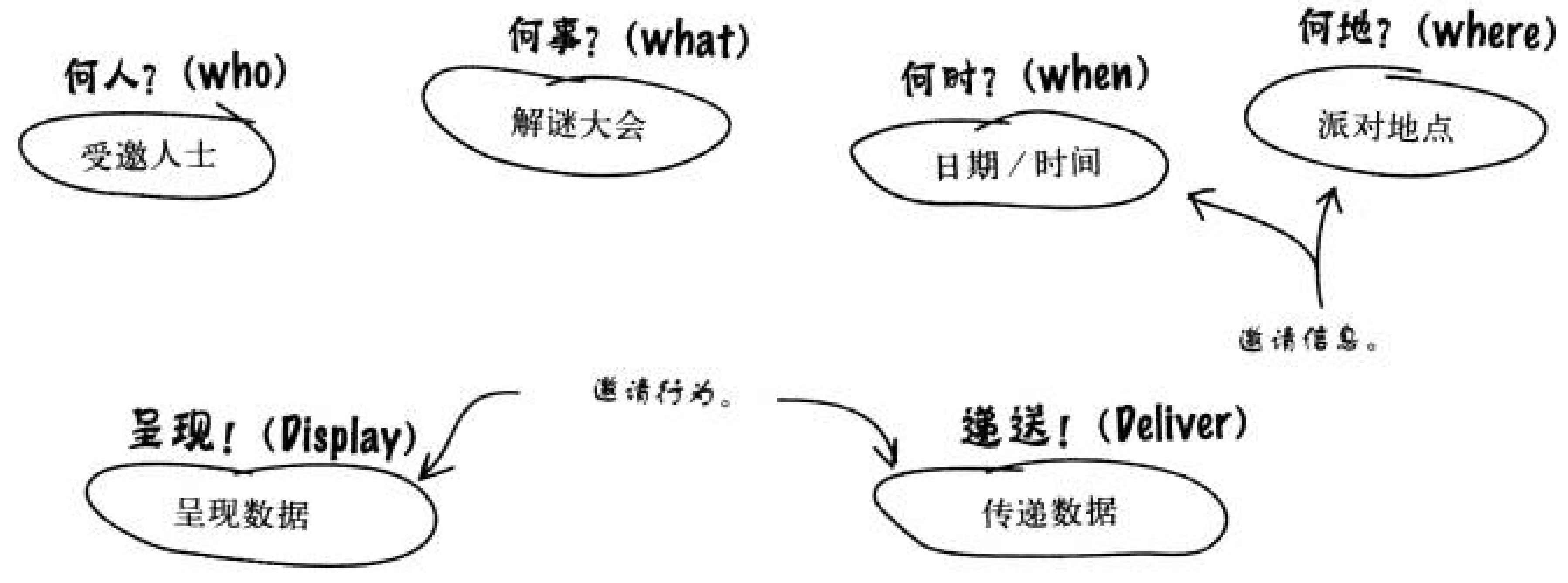


科学对象怪人

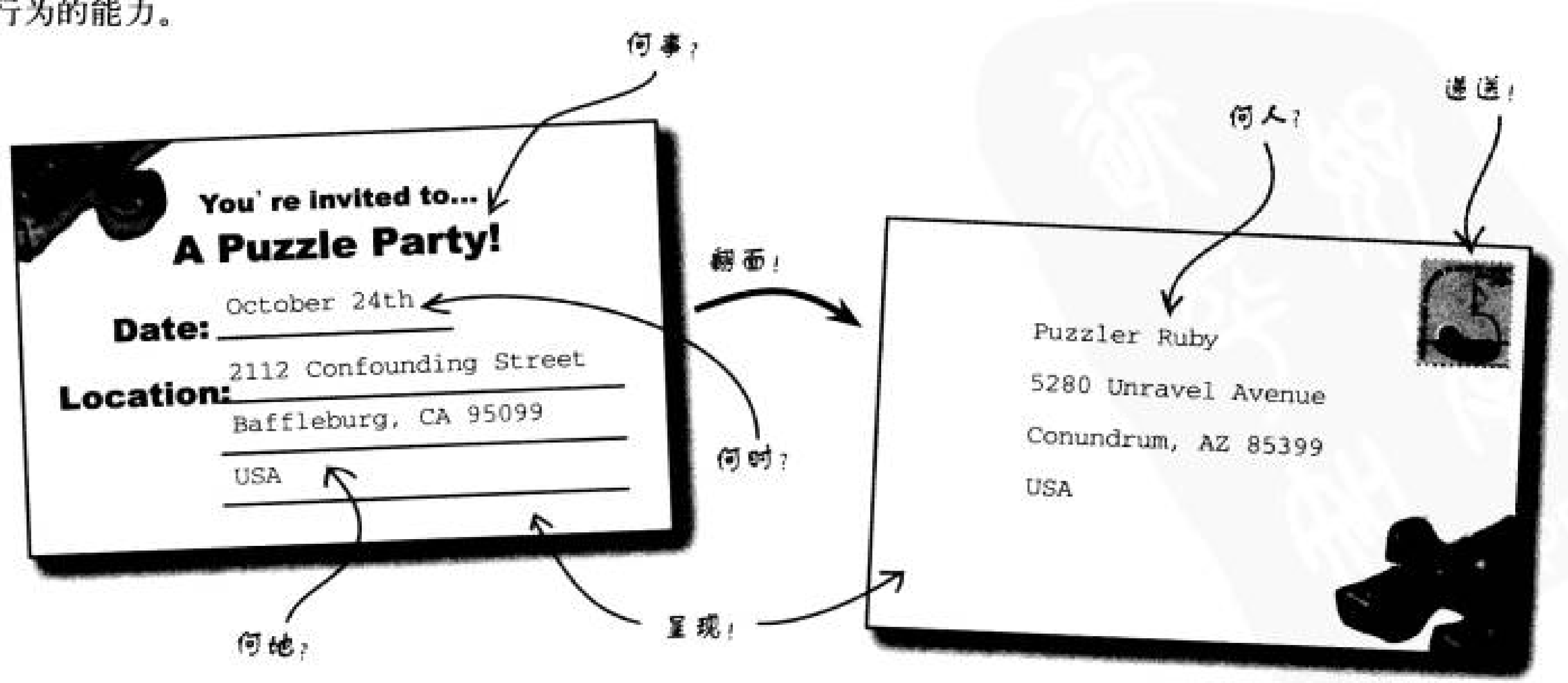
JavaScript对象不见得总像医生说的一样阴森可怕。但它们拼凑起JavaScript语言的林林总总，所以它们团结在一起的力量比较大。**对象结合数据与行为**，以创建一种新的**数据类型**，比起我们到目前为止见过的其他数据更“活灵活现”。你将拿到**可以自行排序的数据、可以自我搜索的字符串**，还有会长毛、会在满月时咆哮的脚本！最后一项可能是我瞎说的，不过你了解我的意思了……

JavaScript 开派对

即将有场派对，你要负责发出邀请函。现在的首要问题就是：完美的派对邀请函上，该有哪些信息？



一份给 JavaScript 的派对邀请函上，数据将被塑造成变量，行为则被塑造成函数。问题在于，现实生活中并不存在分离数据与行为的能力。

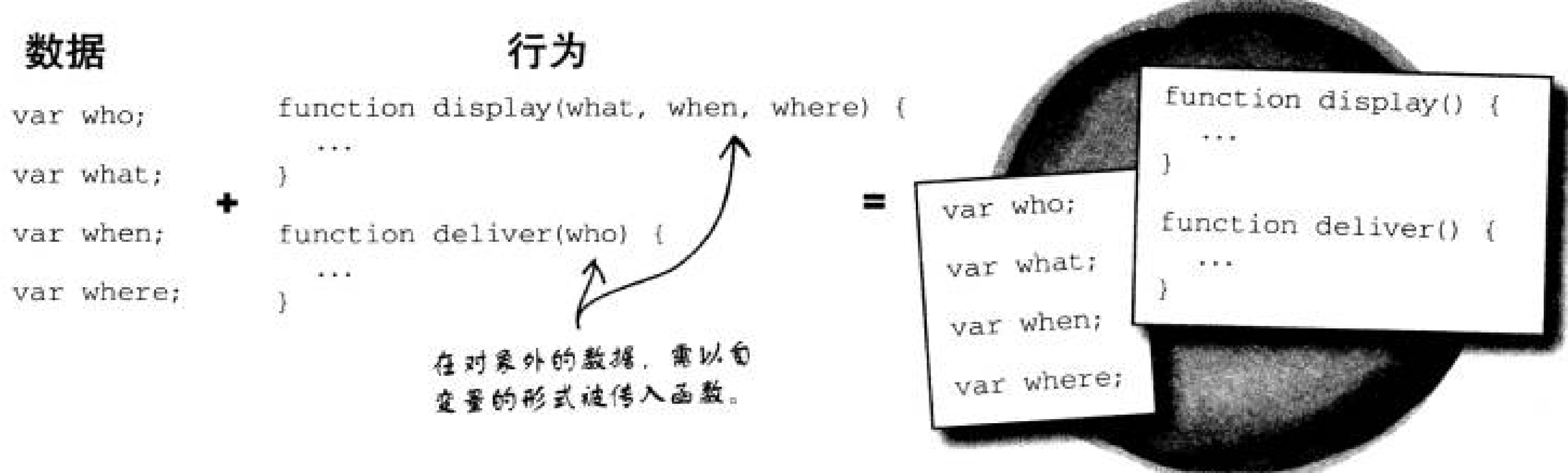


在现实生活中，邀请函结合了数据与行为为一个单一实体——也就是对象。

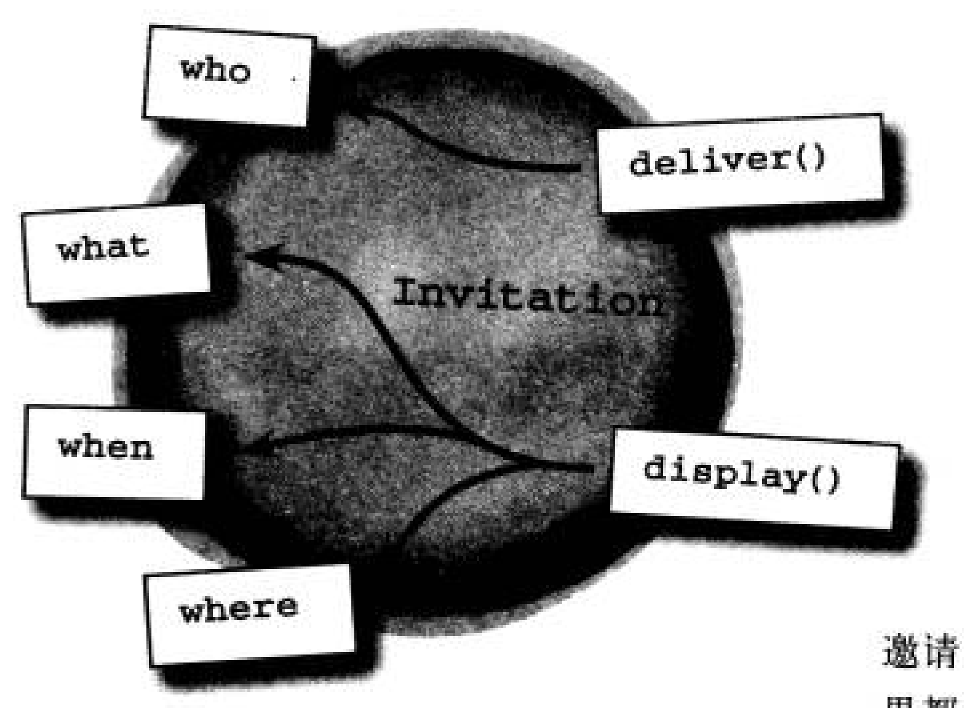
数据+行为=对象

在 JavaScript 里，你不会只遇到分开工作的数据（data）与行为（action）。事实上，JavaScript的**对象**（object）结合了数据与行为，成为一种全然不同的数据结构，其中可以**存储**（store）数据，也可以根据数据而**行动**（act）。这项功能使得 JavaScript 能把现实生活的思考方式运用到脚本上。所以我们能依“事项”思考，而不需受限于数据与行为。

当我们从对象的观点查看邀请函时，你将看到：



在邀请函对象里，数据与函数一起存在，且比起对象外的联系更为紧密。说得更仔细点，放在对象内的函数可以直接访问对象里的变量，而不需另外用自变量把变量传入函数。



对象在一个存储容器内链接变量与函数。

邀请函对象里的数据可被函数访问，但对象以外的世界都看不到这种数据。所以对象是个容器，其中存储数据并链接数据与依据数据行动的代码。

会员独享权益

对象拥有自己的数据

当变量与函数被放在对象里，它们都被当成**对象成员**（object member）。更详细地说，变量称为对象的**特性**（propoerty），函数则称为对象的**方法**（method）。它们还是能存储数据、依据数据而行动，只是这些事情都只发生在特定对象的上下文中。

变量

```
var who;
var what;
var when;
var where;
```

函数

```
function display(what, when, where) {
  ...
}
function deliver(who) {
  ...
}
```

函数变成对象方法。

变量成为对象特性。

对象

特性

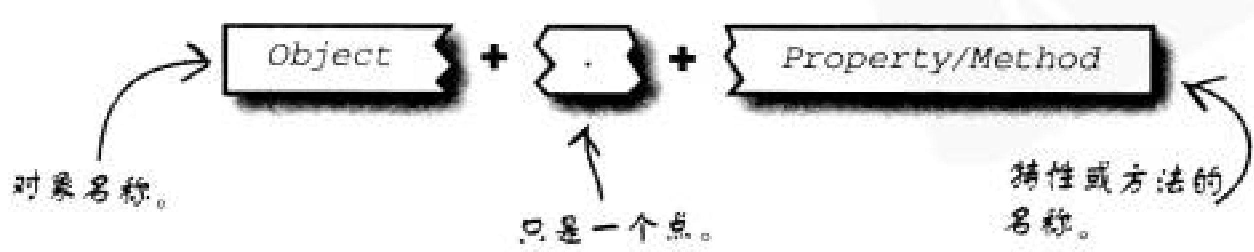
```
var who;
var what;
var when;
var where;
```

方法

```
function display() {
  ...
}
function deliver() {
  ...
}
```

对象中的特性与方法等于对象外的变量与函数。

特性与方法乃被对象“拥有”（own），意思是说，它们存储在对象里，就像数据存储在数组里一样。但访问的方式则不像数组，我们通常使用一种特殊运算符访问对象的方法与特性——**点号运算符**（., dot operator）。



以点号引用对象成员

点号运算符在对象和属于对象的特性或方法间建立起引用。有点像每个人的名字代表**自我个体**，但姓氏则代表他们所属的**家族**。对象也能类比 —— 特性名称告诉我们特性本身，同时**对象名称**告诉我们特性所属的对象。点号运算符则用于结合两者。

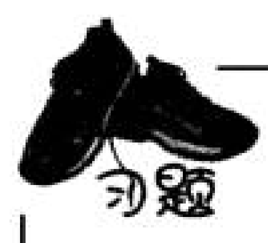
点号运算符引用来自对象的特性或方法。

现在即可使用特性与点号运算符汇整 JavaScript 邀请函对象里的数据了。

```
invitation.who = "Puzzler Ruby";
invitation.what = "A puzzle party!";
invitation.when = "October 24th";
invitation.where = "2112 Confounding Street";
```

请大家记得，既然数据和行为都是相同对象的一部分，想让方法使用数据，你不需要传入任何东西。这点使得邀请函对象可以轻松采取行动。

```
invitation.deliver();
```



派对邀请函里还少了一个RSVP特性，它让受邀人士能够答覆是否参加派对。请为Puzzler Ruby女士增加`rsvp`特性至邀请函（她准备参加派对），然后调用`sendRSVP()`方法以送出她的响应。

....

....

派对邀请函里还少了一个RSVP特性，它让受邀人士能够答覆是否参加派对。请为Puzzler Ruby女士增加rsvp特性至邀请函（她准备参加派对），然后调用sendRSVP()方法以送出她的响应。

```
invitation.rsvp = "attending";
invitation.sendRSVP();
```

点号运算符引用邀请函对象的特性与方法。

这个值也可以是个boolean类型的特性，以true表示受邀人愿意参加派对，false则是不能参加。

没有蠢问题

问：对象究竟是什么？它有数据类型（data type）吗？

答：是的，对象有数据类型。对象是一组有名称的特性与方法的集合。或者这么说吧，对象**就是**一种数据类型。你遇过的其他类型包括：number、text、boolean……统称为**基本数据类型**（primitive data type），因为它们只表示一块信息。对象则被称为complex数据类型，因为其中包含多块数据。你可以把“对象”当成第四种数据类型，写在你已经学过的基本数据类型（number、text、boolean）的后面。所以说，你创建的任何对象，或被你使用的任何JavaScript内置对象，它们的数据类型都是对象。

问：我不能只用全局变量加上函数，不去管对象特性与方法吗？函数也能访问全局变量，不是吗？

答：你讲的是没错啦……问题是，全局变量也没办法阻挡自己被其他代码访问。这点的问题在于，我们都试着限制数据曝露的程度，只让真正有需要的代码接触数据，如此有助于避免数据不小心被其他代码改变。

很可惜地，JavaScript目前无法真正避免对象特性被外来代码访问，而且有些状况下，你会特别希望对象特性能被直接访问。不过，大观念就是把数据放入对象后，数据在逻辑上即与对象相关联。与对象紧密联系的数据，比起在脚本中飘零的一般数据（全局变量），具有更多背景与意义。

问：我已经看过好几次这种“对象加上点号运算符”的注记方式。我真的一直都在使用对象吗？

答：没错，而且你还会发现只用JavaScript却不用对象，是件非常困难的事，因为JavaScript本身就是个大型的对象集合。举例来说，alert()函数技术上其实是window对象的方法，它能以window.alert()被调用。window对象代表浏览器窗口，但不需刻意指出它是个对象，所以你才能只用alert()。

问：好了，你又把我弄糊涂了。这是说函数就是方法吗？

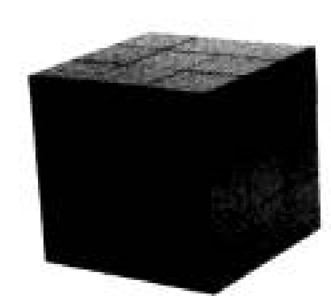
答：对，不过如此看待函数的确是很困扰。各位已经知道函数是一段有名称的代码，其他代码能用函数的名称调用函数。方法其实只是放在对象里的函数。使人困惑之处在于每个函数其实都属于某个对象。

所以说，alert()既是函数也是方法，因此它能被当成函数或方法而被调用——大多数方法被调用时必须当成方法，需加上对象注记（点号）。但在很多情况下，alert()隶属于浏览器的window对象。既然在调用方法时window对象是没有指定对象时的默认对象（如调用alert()时），遂可以把这类方法视为函数。window对象只是偶然拥有这些方法，方法与对象间并无逻辑上的联系。

复习要点

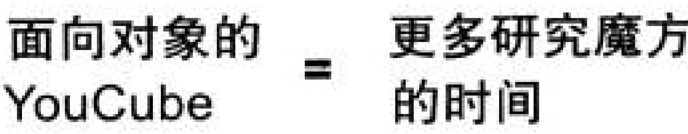
- 对象是种特殊的**数据结构**，结合了数据与**依据数据行动的程序代码**。
- 实际上，对象只是结合成**单一结构体**的变量和函数。
- 当变量被放在对象里时，它称为**特性**；当函数被放在对象里时，它称为**方法**。
- 引用特性和方法时需提供**对象**的名称，后随**点号**运算符，再接上特性或方法的**名称**。

魔方的博客



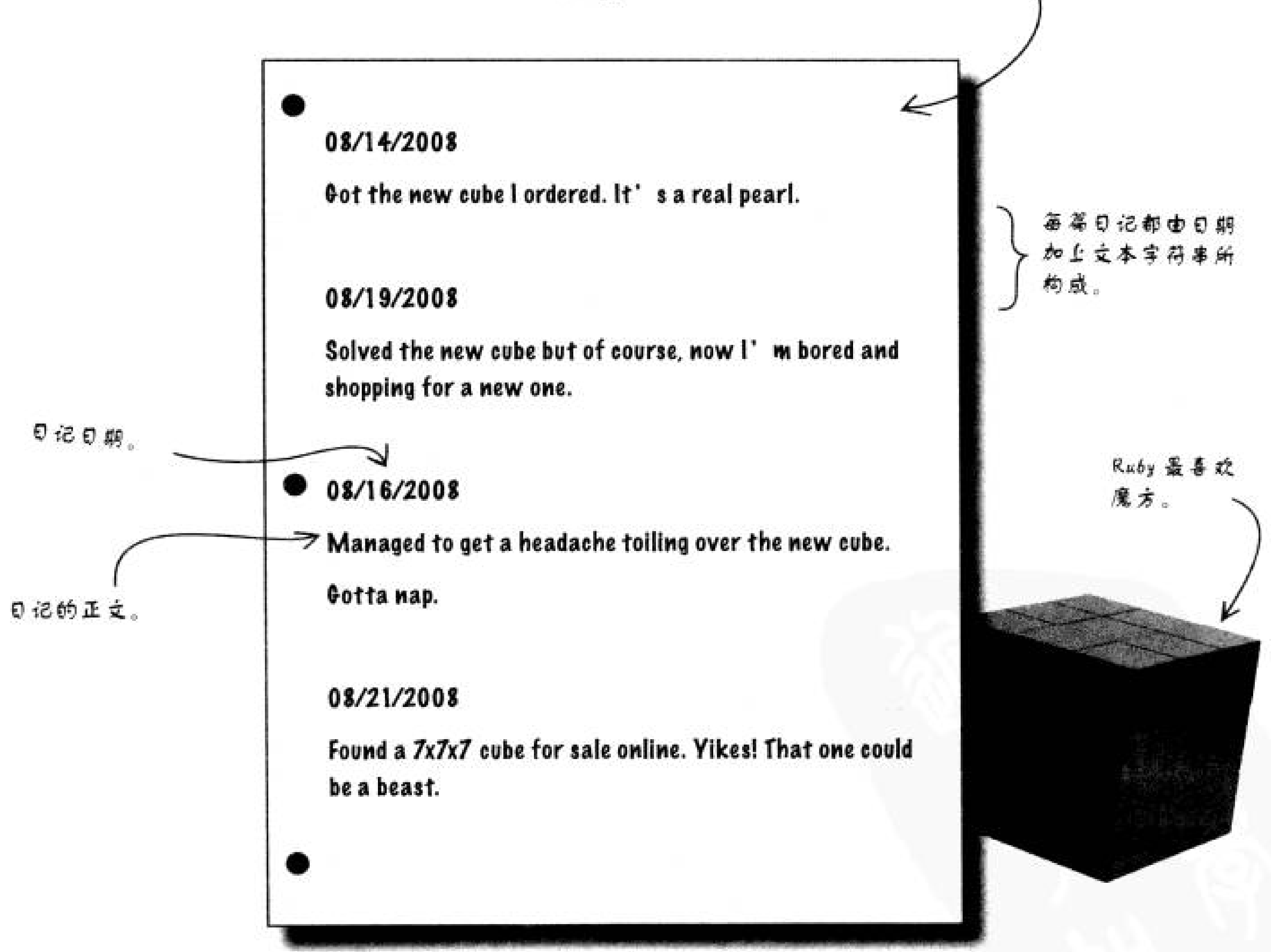
Ruby 收到解谜大会的邀请函，她最爱玩魔方，等不及想和其他热爱解谜的朋友一起聚会。但 Ruby 现在还有比参加派对更重视的事情 —— 她想建立一个博客，与外界共享对魔方的热爱。她已经准备在 YouCube 上分享三维世界的智慧。

Ruby 听说 JavaScript 支持自定义对象，用于创建更可靠、最终更可维护的代码。她也听过很多博主渐渐疏于更新，所以他们的博客经营也转趋停滞。因此，Ruby 想把博客建筑在正确的基础上 —— 她要采用**面向对象**（object-oriented）的脚本，使用自定义对象迎向充满谜团的未来。



解构 YouCube

Ruby 有份手写的日记，她也浏览过不少博客，知道还需要加上日期与文字，但她不知道如何使用 JavaScript 存储这些数据。总而言之，Ruby 不想再把魔方的日记（很快将变成博客）写在纸上了！



Ruby 迫切需要直接存储与访问多组“日期加文字”数据的方式。听起来真像 JavaScript 对象的功能……结合多段信息，组成单一实体。

博客日期 + 博客正文 = 博客对象

一个自定义对象，可让两块博客数据结合为单一实体。

自定义对象扩展了JavaScript

JavaScript 语言包含许多便利的标准对象，本章稍后将讨论数个此类对象。虽然标准对象便利好用，但有些时候，“标准”还是不够。YouCube 博客即可看出这类限制，因为它牵涉到的数据存储问题，无法使用 JavaScript 内置数据类型解决……此时适用自定义对象。

真的，字符串真的也是对象！

String

"Got the new cube I ordered. It's a real pearl."

标准 JavaScript 对象。

Date

October 24th, 2008 7:00pm

Array

数组也是对象！

	date	body
0	08/14/2008	Got the new cube I ordered...
1	08/19/2008	Solved the new cube but of...
2	08/16/2008	Managed to get a headache...
3	08/21/2008	Found a 7x7x7 cube for sale...

博客日期

Blog

"08/14/2008"

"Got the new cube I ordered. It's a real pearl."

专为 YouCube 设计的自定义对象！

没有对象的帮忙，把博客日志存储在 2D 数组里好像也是个选择。

博客正文。

Blog 对象是种复合数据类型——结合两块数据为一段数据。

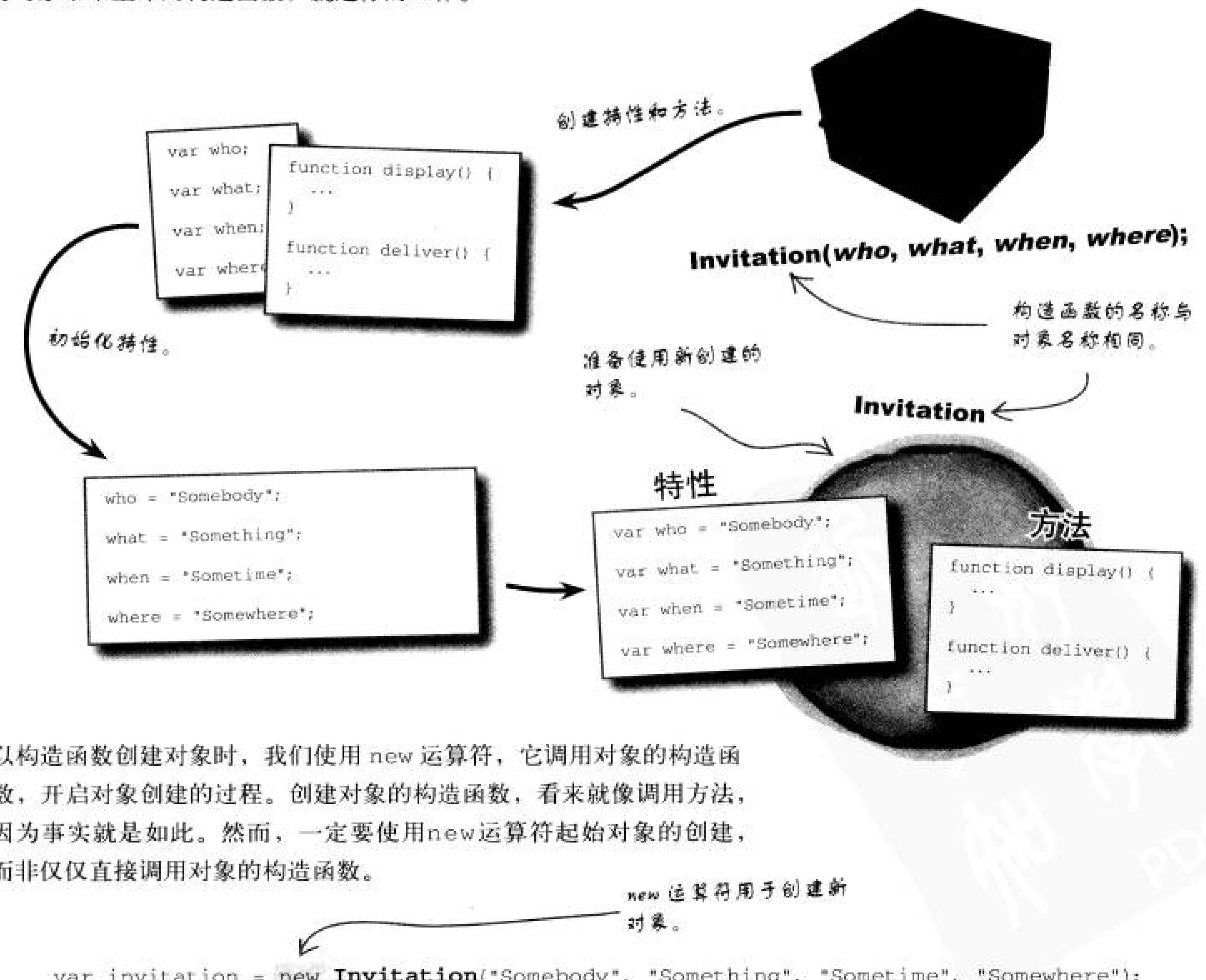
自定义对象增加了JavaScript的功能，以适应我们的特殊需求。以Ruby的例子而言，自定义对象可形成博客日志，使用**特性**（property）以表达博客日期与日志正文。另外，**方法**（method）则可增加博客日志的行为，让博客的建立与管理更直观。

为了带来自定义对象的生命，我们必须先知道如何自定义对象……

构造你的自定义对象

既然对象具有相关的数据，数据便需在对象创建时初始化，此时需要特殊的方法——**构造函数**（constructor），来为对象准备运行。每个自定义对象都需要自己的构造函数，其名称与对象相同。创建对象时，需调用构造函数以初始化对象。在创建自定义对象时，设计合适的、为对象带来生命的构造函数，就是你的工作。

构造函数负责创建对象。



以构造函数创建对象时，我们使用 new 运算符，它调用对象的构造函数，开启对象创建的过程。创建对象的构造函数，看来就像调用方法，因为事实就是如此。然而，一定要使用new运算符起始对象的创建，而非仅仅直接调用对象的构造函数。

new 运算符用于创建新对象。

```
var invitation = new Invitation("Somebody", "Something", "Sometime", "Somewhere");
```

新对象存储在变量里。

对构造函数的调用就像调用方法。

通过传入自变量给构造函数以设置特性。

构造函数里有什么？

构造函数的大部分工作，就是创建对象的特性，还有对象的初始值。在构造函数里创建特性时，你需要使用 JavaScript 的关键字`this`。`this`指派对象特性的所有权，同时设置特性的初始值。这个关键字的功用与它的字面意义相同 —— 创建属于“这个”（this）对象的特性，而不只是构造函数里的局部变量。

```
function Invitation(who, what, when, where) {
  this.who = who;
  this.what = what;
  this.when = when;
  this.where = where;
}
```

对象特性的创建与初始化需使用对象注记（点号运算符）和关键字`this`。没有`this`，构造函数不会知道你正在创建对象特性。上例构造函数的结果创建了4个特性，每个特性分别被指派了作为自变量传入构造函数的4个值。

关键字 this 是于构造函数里创建对象特性的重点。

磨笔上阵

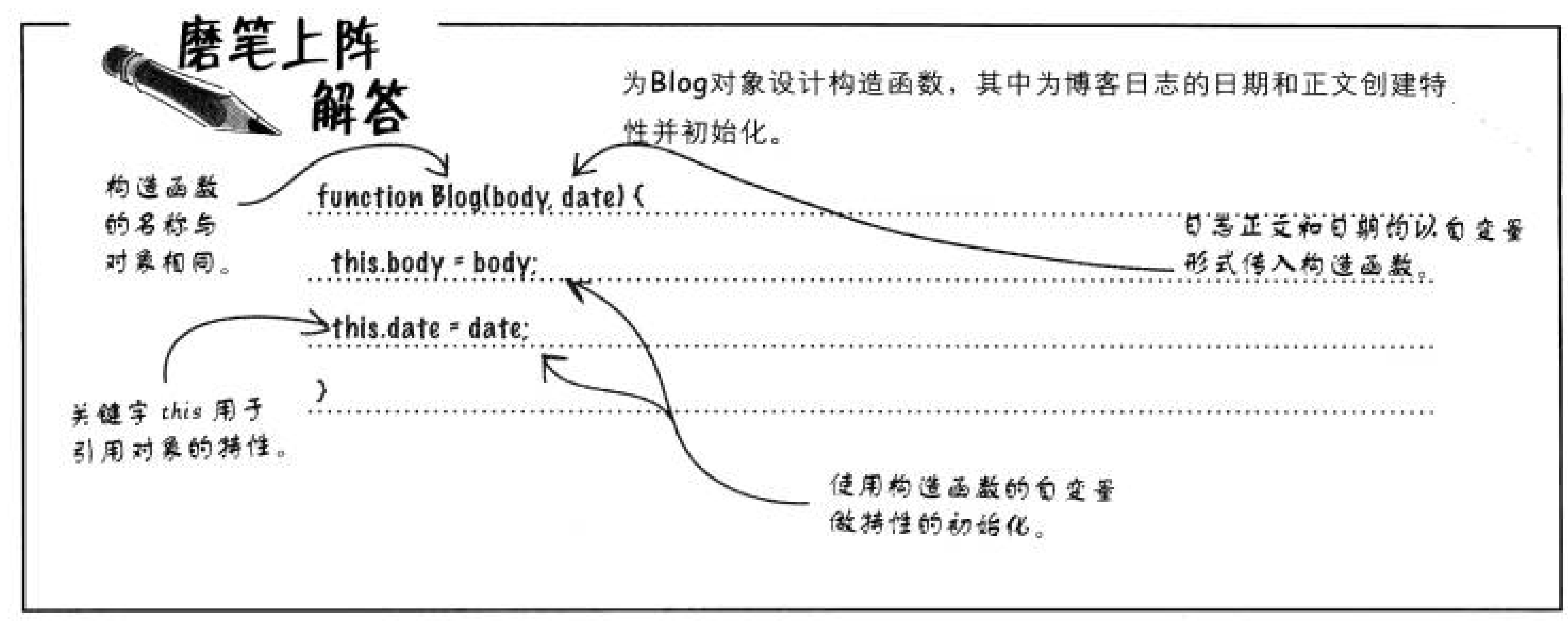
为Blog对象设计构造函数，其中为博客日志的日期和正文创建特性并初始化。

....

....

....

....



为博客对象带来生命

Blog对象渐渐有模有样了，但它尚未真正地建立。理论看起来再美好，也只是等待证明的假设。请记住，构造函数虽然确立了对象的设计，但实际上并未创建任何事物；直到使用 new 运算符调用构造函数，才创建了对象。所以我们继续下一步，创建活跃的 Blog 对象。

关于本章的范例，可以至http://www.headfirstlabs.com/books/hfjs/下载。

手写的博客日志。

08/14/2008
Got the new cube I ordered. It' s a real pearl.

JavaScript 制作的 Blog 对象。

```
var blogEntry = new Blog("Got the new cube I ordered...", "08/14/2008");
```

Blog
"Got the new cube I ordered. It's a real pearl."
"08/14/2008"

调用Blog()构造函数以创建对象。

```
function Blog(body, date) {
  ...
}
```

对象的创建。

没有蠢问题

问：对象的创建搞得我一个头两个大。究竟是用 new运算符创建对象，还是用构造函数创建对象？

答：都是！new运算符负责设定对象的创建，它的工作主要在于确定调用了构造函数。仅像调用函数般调用构造函数，而未使用new运算符，并不会创建对象；只使用new运算符，而没有构造函数，也不会有任何意义。

问：每个自定义对象都需要构造函数吗？

答：是的。因为构造函数负责创建对象的特性，所以没有构造函数，就没有任何特性。没有任何特性，也就不是个有意义的对象。

但构造函数的规则有项例外——创建单纯的 organizational object 时（这类对象由一组方法集合组成，但这些方法不会依据对象的特性行动）。此时，技术上可以省略构造函数。但也请你记得，这类对象并非面向对象编程实践的绝佳范例，它只是许多相关函数的集合而已。尽管如此，JavaScript 仍然提供了这类对象，用于收集数学的相关任务，本章稍后将会提到。

问：this究竟是什么？

答：this是个 JavaScript 的关键字，用于引用对象。说得更详细点，this 从对象内部引用同一个对象。我知道，听起来很诡异，而且有点精神分裂。不过只要好好地换个角度想清楚，一切就合理了。以现实生活举例，如果你丢了手表，但有人在挤满了人的屋子的某个地方找到了你的手表，当他高举手表时，你很可能会大叫：“那是我的手表！”对吧！你用了代名词“我的”，以自我引用。更重要的是，“我的”还用于澄清你是手表的主人。this的运作方式也相同——它指出对象的**所有权**。所以，this.date表示date特性属于代码出现处的对象。

磨笔上阵

创建属于Blog对象的数组，名称为blog，以YouCube博客日志数据做初始化。在每则记录的正文部分，只填入最前面的几个字意思一下就可以了。

```
var blog =
  [ ........................................................

  ..........................................................

  ..........................................................

  ...................................................... ];
```

08/14/2008
Got the new cube I ordered. It's a real pearl.

08/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

08/16/2008
Managed to get a headache toiling over the new cube.
Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes!
That one could be a beast.

磨笔上阵解答

创建属于Blog对象的数组，名称为blog，以YouCube博客日志数据做初始化。在每则记录的正文部分，只填入最前面的几个字意思一下就可以了。

08/14/2008
Got the new cube I ordered. It's a real pearl.

08/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

08/16/2008
Managed to get a headache toiling over the new cube.
Gotta nap.

08/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.

```
var blog = [ new Blog("Got the new cube I ordered...", "08/14/2008"),
             new Blog("Solved the new cube but of course...", "08/19/2008"),
             new Blog("Managed to get a headache toiling...", "08/16/2008"),
             new Blog("Found a 7x7x7 cube for sale...", "08/21/2008") ];
```

每一则博客日志都被创建为Blog对象，具有自己的日志正文与日期。

YouCube 1.0

结合Blog对象的数组以及一些呈现博客数据的JavaScript代码，初版 YouCube 就此诞生。Ruby 知道革命尚未完成，但博客已经上线运作，她很高兴初步展现的成果。

存储在每个Blog对象的日期，均整齐地呈现在YouCube的网页上。

YouCube - The Blog for Cube Puzzlers

8/14/2008
Got the new cube I ordered. It's a real pearl.

8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

8/16/2008
Managed to get a headache toiling over the new cube. Gotta nap.

8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.

Show All Blog Entries

Done

我喜欢Blog对象结合日期与YouCube日志正文的方式。

让我们瞧一下为Blog对象带来生命，让 YouCube 1.0 美梦成真的代码……

放大 YouCube

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript">
      // Blog object constructor
      function Blog(body, date) {
        // Assign the properties
        this.body = body;
        this.date = date;
      }

      // Global array of blog entries
      var blog = [ new Blog("Got the new cube I ordered..", "08/14/2008"),
                   new Blog("Solved the new cube but of course...", "08/19/2008"),
                   new Blog("Managed to get a headache toiling...", "08/16/2008"),
                   new Blog("Found a 7x7x7 cube for sale online...", "08/21/2008") ];

      // Show the list of blog entries
      function showBlog(numEntries) {
        // Adjust the number of entries to show the full blog, if necessary
        if (!numEntries)
          numEntries = blog.length;

        // Show the blog entries
        var i = 0, blogText = "";
        while (i < blog.length && i < numEntries) {
          // Use a gray background for every other blog entry
          if (i % 2 == 0)
            blogText += "<p style='background-color:#EEEEEE'>";
          else
            blogText += "<p>";

          // Generate the formatted blog HTML code
          blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";

          i++;
        }

        // Set the blog HTML code on the page
        document.getElementById("blog").innerHTML = blogText;
      }
    </script>
  </head>

  <body onload="showBlog(5);">
    <h3>YouCube - The Blog for Cube Puzzlers</h3>
    <img src="cube.png" alt="YouCube" />
    <div id="blog"></div>
    <input type="button" id="showall" value="Show All Blog Entries" onclick="showBlog();" />
  </body>
</html>
```

Blog() 构造函数创建两个特性。

Blog 对象的数组。

showBlog() 函数在网页的 div 元素 “blog” 里，绘制博客日志数据。

如果并未以自变量传入预备呈现的博客日志编号，则显示所有日志。

更换博客日志的背景颜色，让阅读体验更舒服。

为div元素 “blog” 设定日志正文的格式。

div 元素 “blog” 最初为空白，但最后将以具有格式的文本填满。

按下按钮时，列出所有博客日志。

没有蠢问题

问：为什么 YouCube 需要一个"Show All Blog Entries"按钮？

答： 以博客目前的状态而言，的确不需要这个按钮，毕竟，总日志数只有四篇已。但随着博客的成长，限制 YouCube 主页上出现的预设日志数量，将变得越来越重要，以免磨掉用户等待的耐心或信息接收量。所以博客代码里默认为只列出5篇日志。"Show All Blog Entries"按钮可改写默认值，列出所有日志。

问：为什么使用innerHTML呈现博客日志，而不用 DOM 方法？

答： 虽然DOM方法绝对是考虑万维网标准时的较佳选择，但讲到动态产生格式丰富的HTML代码，它实在有点笨重。因为每个容器标签，例如<p>与<strong>，必须被创建为父节点，以包含作为内容的子节点。在这种状况下，innerHTML实在方便太多了，而且也能简化YouCube的程序代码。

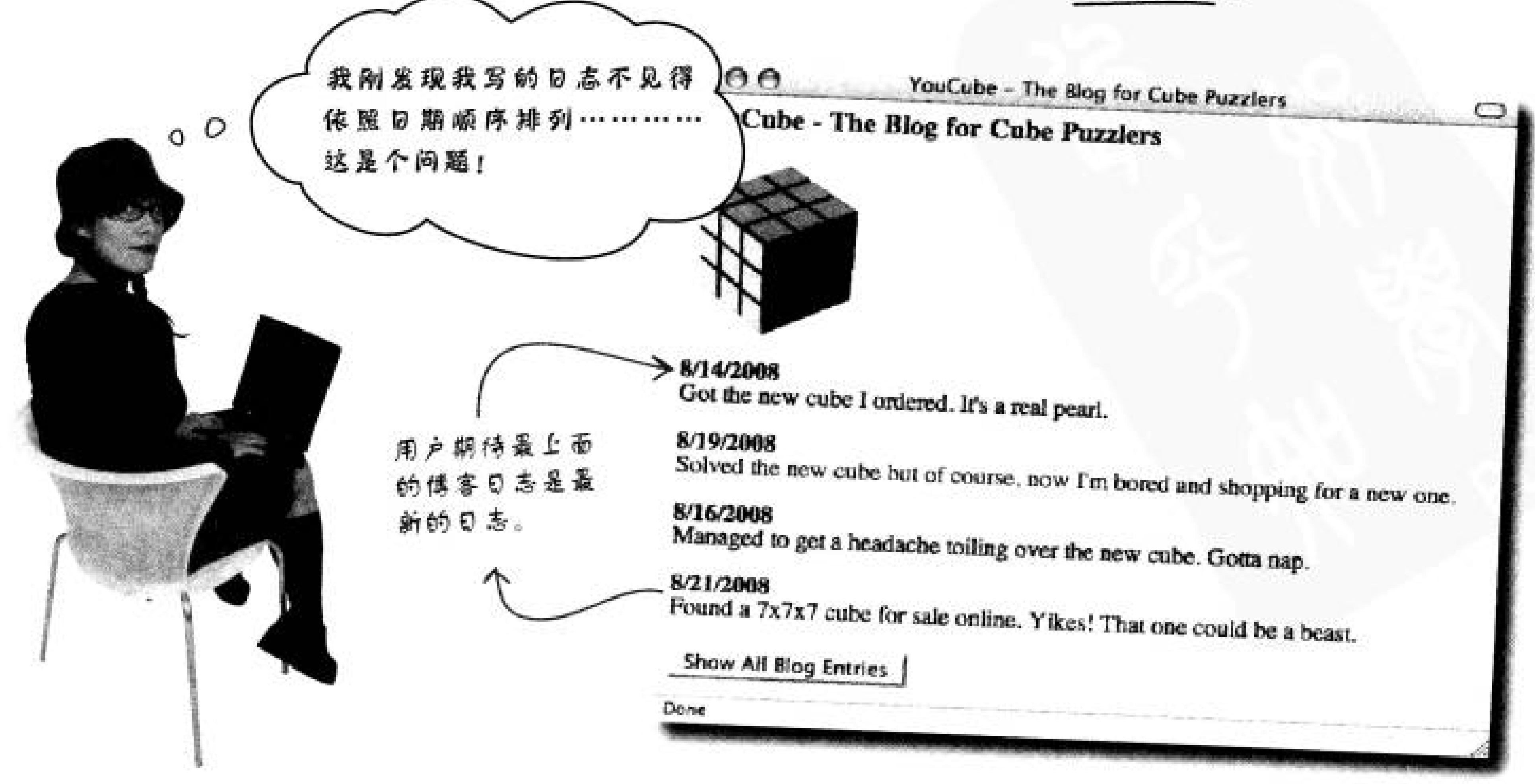
问：为什么 Blog 对象没有任何方法？

答： 你很野心勃勃啊……不错！事实上，YouCube还有很多地方需要处理，然后才轮到Blog对象的方法。但是，别惊慌。对象方法肯定是YouCube远景规划的一部分。"方法"是优良对象设计的重要部分，Blog也不会是个例外。

失序的博客

YouCube 1.0 看起来不错，不过还是有些缺点。Ruby 注意到博客日志的顺序不对 —— 应该把最新的日期排在最前面才对。目前的呈现顺序就是日志存储的顺序，而存储顺序不是个值得依赖其编年的顺序。

博客日志应该把最新的日志排在最前面。



排序的需求

Ruby 想到，可依日期排列博客的数组顺序，以解决排序的问题。既然 JavaScript 支持循环与比较运算，应该可用循环处理博客数组，比较每则日志的日期，然后依反向的编年顺序排列（先列出最新的日志）。

这两则日志顺序对调，因为第二篇的日期比较新。

Blog
"Got the new cube I ordered. It's a real pearl."
"08/14/2008"

Blog
"Solved the new cube but of course, now..."
"08/19/2008"

Blog
"Managed to get a headache toiling over the new..."
"08/16/2008"

Blog
"Found a 7x7x7 cube for sale online. Yikes..."
"08/21/2008"

这篇博客日志应该最先出现，因为它的日期最新。

1. 以循环处理博客日志。
2. 比较每个Blog 对象与下个对象的日期。
3. 如果下个对象比当前对象的日期更新，则交换其位置。

博客排序的解决方法听起来很聪明，而且好像可行……假如我们已经知道如何比较日期顺序的话。

等一下！如果日期当成字符串而排序，该如何比较日期，从而判断哪组日期比较新呢？

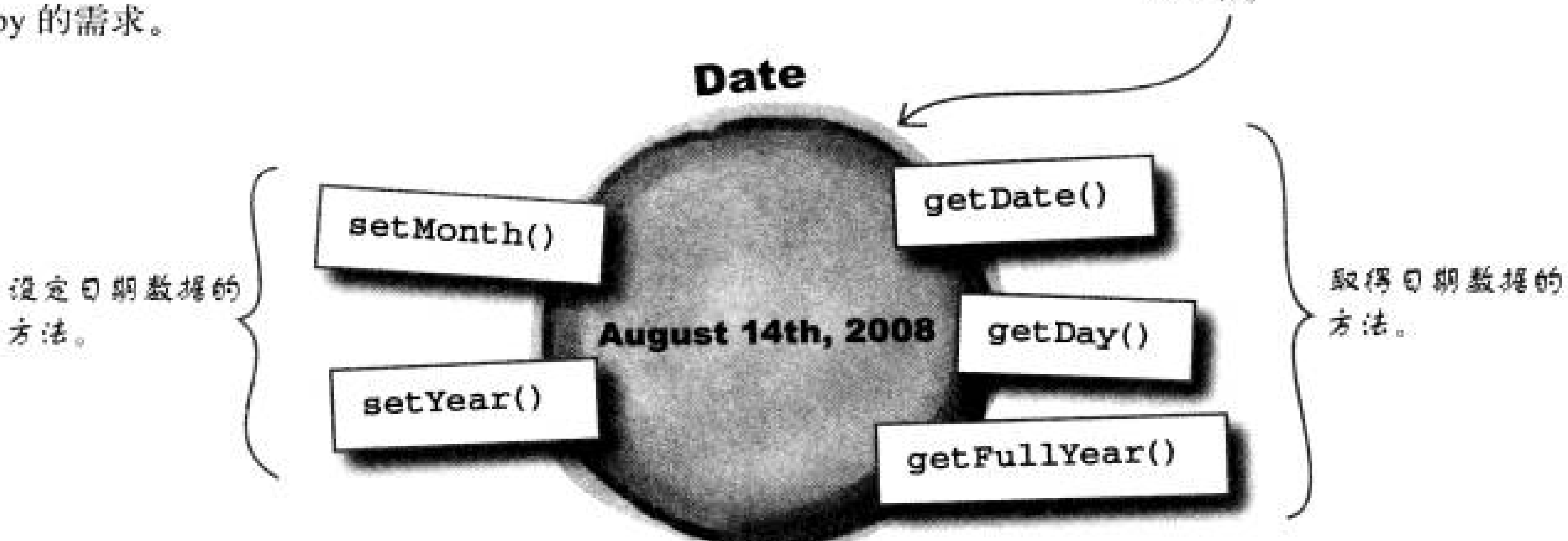
存储为字符串的日期并非真正的日期。

Ruby 的博客排序策略遇到了非常严重的意外障碍，因为存储为字符串的日期并不具有时间的观念。换句话说，只是比较字符串"08/14/2008"与"08/19/2008"，我们无法得知何者的日期比较新，因为它们只是字符串。虽然字符串可作比较，但字符串比较运算并不了解日期的独特格式，自然也无法比较日期中的月、日、年等部分。

所以，在我们认真考虑以日期排列博客日志的顺序前，首先需要重新思考博客中存储日期的方式。

日期专用的 JavaScript 对象

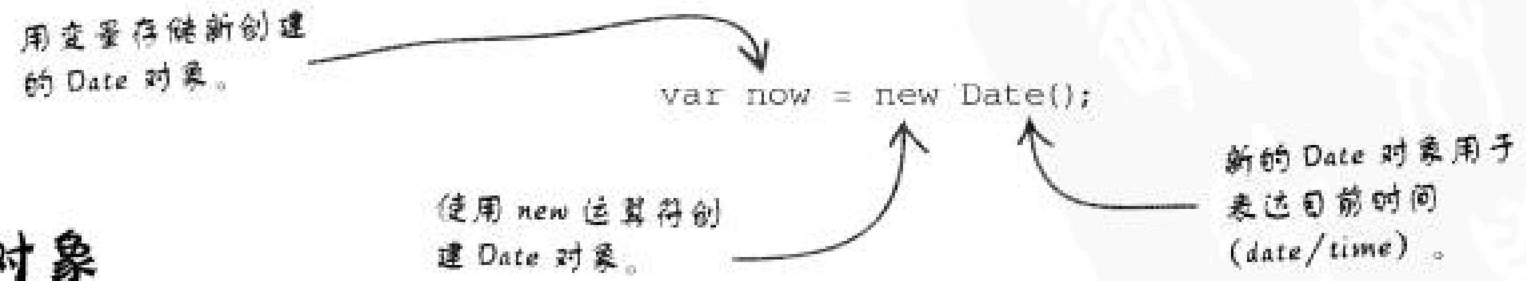
Ruby 需要一种能够比较日期的存储方式。换句话说，日期需要了解自己是日期，行为也要依据日期而行动。等一下，听起来怎么像个对象！事实上，JavaScript 确实提供内置的Date对象，非常适合 Ruby 的需求。



内置Date对象可表达时间里的瞬间。

Date对象可表达时间中特定的一刻，最小单位可达毫秒，而且它是 JavaScript 的标准配备。不过 Date 的内部其实用到了特性，只是对象的用户看不到特性。我们纯粹透过 Date 的方法而运用这个对象。

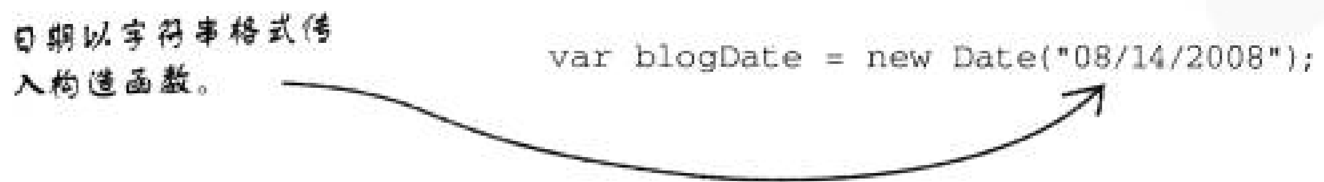
与 Blog 对象相似，Date 对象需以 new 运算符创建。下例创建一个表示目前时间的 Date 对象：




```
var now = new Date();
```

在 Date 对象里，时间以毫秒数表达。

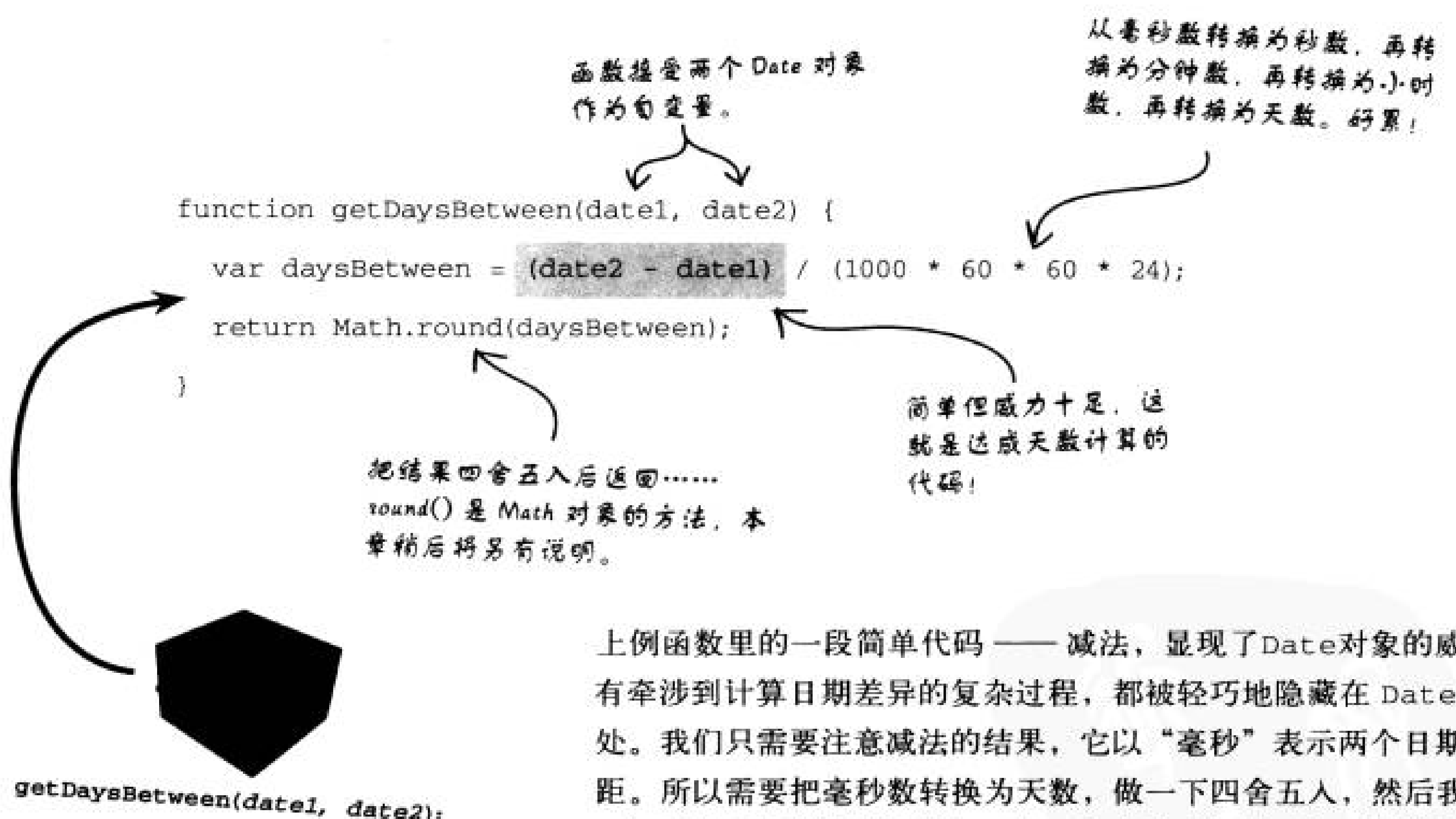
上例创建Date对象并初始化为目前的时间。请注意创建Date对象的语法很接近对函数或方法的调用 —— 因为你其实正在调用Date对象的构造函数。如果丢一段字符串自变量给 Date() 的构造函数，即可指定日期。如下例的 Date 对象即代表第一篇 YouCube 博客日志的日期：




```
var blogDate = new Date("08/14/2008");
```

计算时间

对象最强大的一项功能，就是它们天生知道自我操纵的方式。以计算两个日期间差距的天数为例，用我们的人脑计算天数实在不太容易，你必须参考某个时间点，把某个日期转换为天数，再确定考虑了闰年的因素。或者，把这项工作交给Date对象代劳吧……请看利用几个 Date对象完成这项繁重工作的函数范例：



上例函数里的一段简单代码 —— 减法，显现了Date对象的威力。所有牵涉到计算日期差异的复杂过程，都被轻巧地隐藏在 Date 对象深处。我们只需要注意减法的结果，它以“毫秒”表示两个日期间的差距。所以需要把毫秒数转换为天数，做一下四舍五入，然后我们就有一个可以重复使用的便利小工具，每次需要知道两个日期的差距时即可拿出来利用。

请为前两则 YouCube 博客日志创建 Date 对象。然后调用 getDaysBetween() 函数，传入刚创建的两个 Date 对象，并以 alert框呈现运算结果。

习题解答

请为前两则YouCube博客日志创建Date对象。然后调用 getDaysBetween()函数，传入刚创建的两个 Date对象，并以 alert框呈现运算结果。

为两则博客日志的日期创建Date 对象。

```
var blogDate1 = new Date("08/14/2008");
var blogDate2 = new Date("08/19/2008");
alert("The dates are separated by " + getDaysBetween(blogDate1, blogDate2) + " days.");
```

函数返回日期差距。

把两个Date对象当成自变量传给函数。

The dates are separated by 5 days.

OK

重新思考博客的日期

有了 JavaScript 提供的 Date 对象，终于可以聪明地操纵日期，但 YouCube 的 Blog 对象目前仍然把日期存储为字符串，而不是 Date 对象。为了利用 Date 对象的各种功能，我们需要改变博客的日期，让它们成为 Date对象。

问题是，Blog 对象的 date 特性可以存储成Date对象吗？

对象里的对象

Blog对象是个“对象常常**包含**（contain）其他对象”的好例子。Blog对象的两个特性实际上本身已经是对象 —— 都是String对象。String对象看起来不太像对象，它们只简单地括起一段文本字符串，创建为**对象字面量**（object literal）。但Date对象就没这么灵活了，必须使用new运算符创建Date对象。

为了把博客的date特性创建成Date对象，我们在创建Blog对象时，必须使用new运算符创建新的Date对象。听起来很可怕吗？或许直接看范例代码有助于抒缓恐惧。

字符串字面量自动创建String对象。

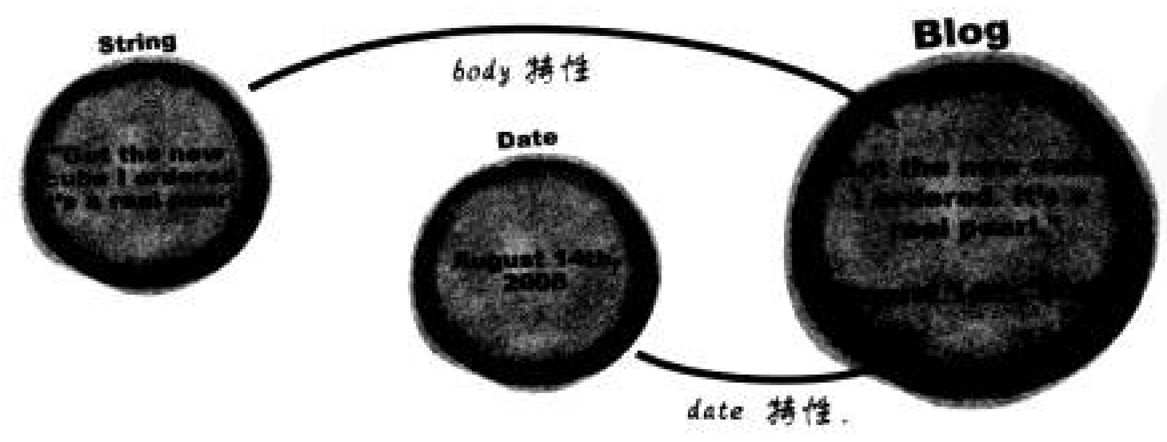
使用new运算符创建Blog对象。

```
var blogEntry = new Blog("Nothing going on but the weather.",
  new Date("10/31/2008"));
```

创建了Date对象并传入Blog()构造函数，同样使用new运算符。

现在的YouCube博客日志，是个包含两种其他对象（一个String与一个Date对象）的对象。当然，我们仍然需要创建Blog对象的数组，才能成功地表示所有YouCube博客的日志。

new运算符在构造函数的协助下创建对象。



磨笔上阵

重新设计创建YouCube的Blog对象的数组的代码，把日期改为Date对象。不把日志正文全都抄上来也没关系。

....

....

....

....

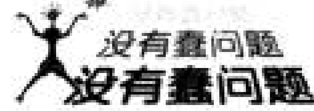
重新设计创建YouCube 的Blog对象的数组的代码，把日期改为 Date 对象。不把日志正文全都抄上来也没关系。

每一则博客日志仍创建为 Blog 对象。

```
var blog = [ new Blog("Got the new cube I ordered...", new Date("08/14/2008")),
  new Blog("Solved the new cube but of course...", new Date("08/19/2008")),
  new Blog("Managed to get a headache toiling...", new Date("08/16/2008")),
  new Blog("Found a 7x7x7 cube for sale...", new Date("08/21/2008")) ];
```

字符串字面量已很适合作为博客日志的正文。

每个 Blog 对象的日期均创建为 Date 对象。



问：为什么 Date 对象的日期单位为毫秒？

答：首先，请了解 Date 对象在于表达“瞬间”。如果宇宙有个暂停按钮，我们就可以得到时间长河中冻结的瞬间。但还需要一个参考点，才能表达这个瞬间。所以有人擅自决定 1970 年 1 月 1 日就是参考点，接下来需要测量从参考点到冻结瞬间的偏移量。或许是 30 年又 8 个月又 14 天又 3 小时又 29 分又 11 秒……这种表示时间偏移的方式也太累赘了吧！不如只采用单一测量单位，以能够表示最小的时间片段为原则。那就使用毫秒吧？所以我们不会看到那么多时间单位，而是得出 1,218,702,551,000 毫秒。没错，使用毫秒的决定就是这么任性，反正 JavaScript 并不在意处理大数字。

问：使用 Date 对象时，我需要担心毫秒转换的问题吗？

答：看情况。Date 对象有几个提取部分日期单位的方法，以免直接处理毫秒数。不过，如果你需要处理两个日期的差距，大概就躲不开毫秒的使用了。

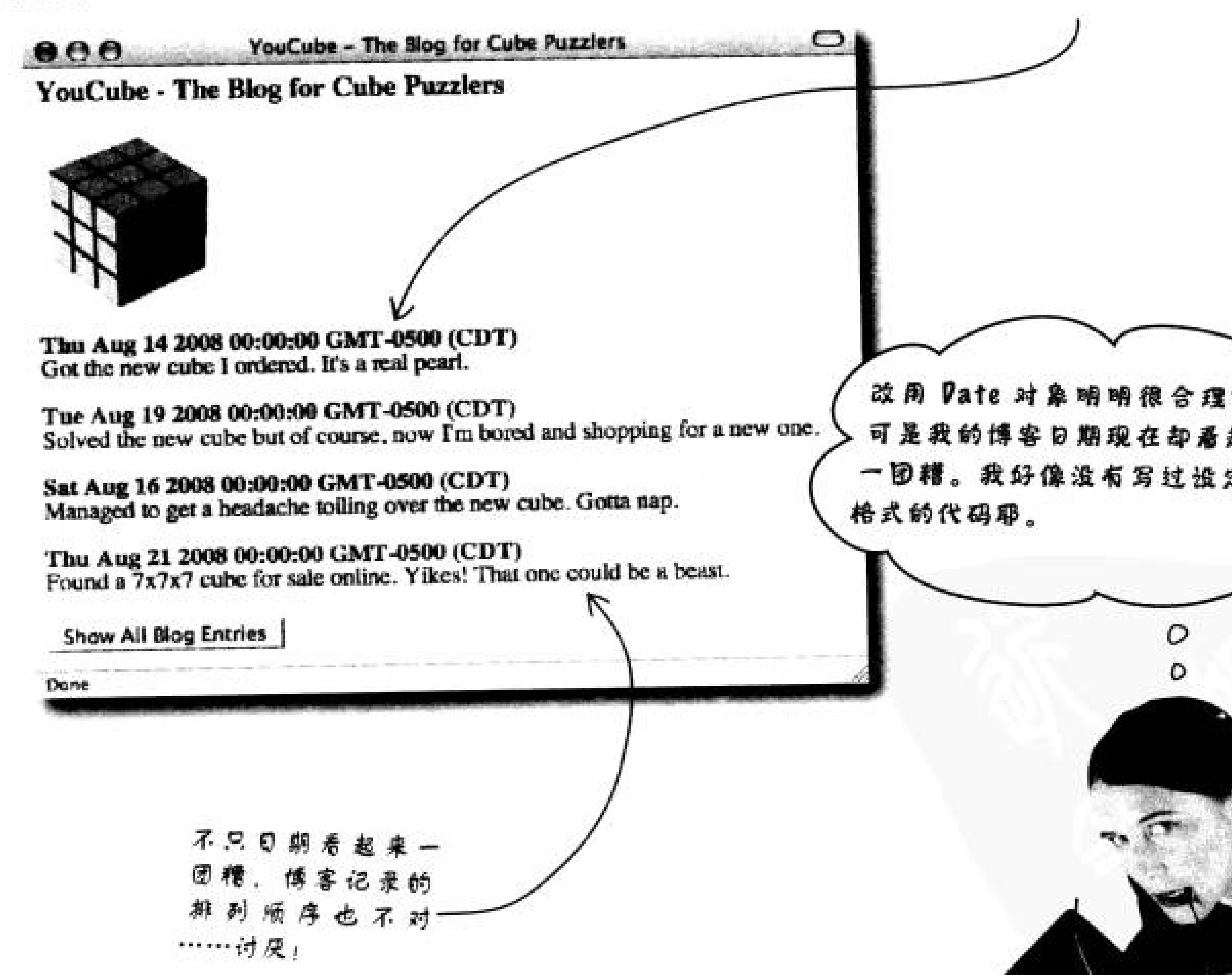
复习要点

- 标准的 JavaScript Date 对象，以毫秒表达瞬间。
- Date对象具有一些访问各部分时间值的方法。
- Date 对象聪明到能够对日期做四则运算，还有比较日期。
- 与大部分对象相同（String对象除外），Date对象需以 new 运算符创建。

日期没有用……对人类而言

把Blog 对象的 date 特性转换为 Date 对象后，Ruby 准备把注意力转回博客日志的排序。好吧，几乎准备好了。她现在好像为博客的日期带来新的问题，新的日期格式很难懂。Ruby 在想，用户应该不会关心每则日志在哪个时区发表吧……这类信息只会干扰 YouCube 的使用体验。显然，引入 Date对象至 YouCube 后，还有一些需要仔细检验的问题。

博客日期真乱……信息过载！



Ruby 对于 YouCube 谜样的日期信息感到大惑不解，她根本不记得写过任何呈现日期的代码。她明明只把日期字符串转换成 Date对象而已。难道她的日期受到什么邪恶的 JavaScript 妖力诅咒吗？

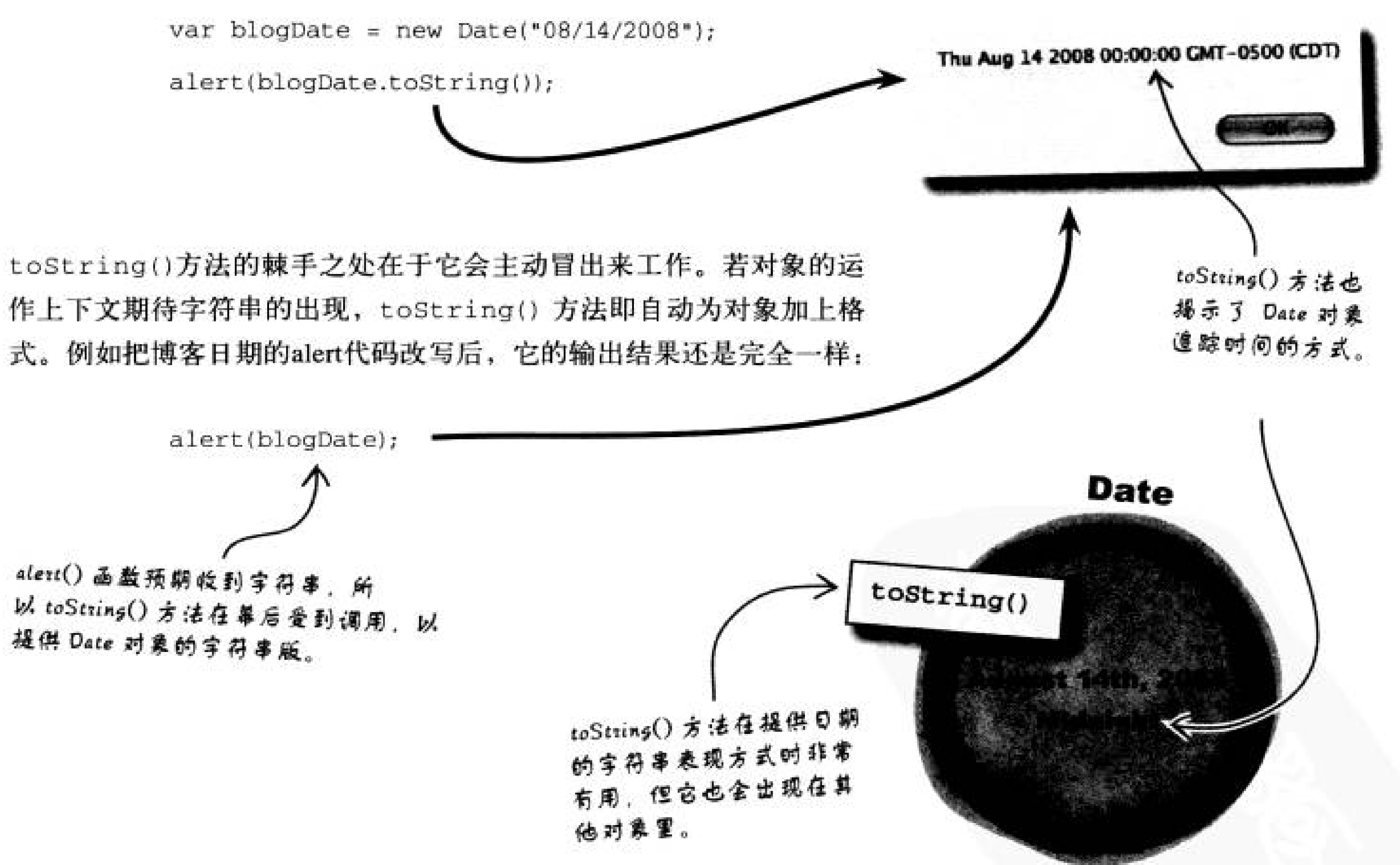
对象转换成文本

幸好，并非什么莫名的邪恶力量造成难看的 YouCube 日期。事实上，一切来自 JavaScript 日期对象的本性 —— 日期自己为自己加上格式！运作方式如下：每个 JavaScript 对象都具有toString()方法，它试图为对象提供文本字符串版的表达方式。谜般的日期就是 Date 对象的默认toString()方法的输出结果。

```
var blogDate = new Date("08/14/2008");
alert(blogDate.toString());
```

toString()方法的棘手之处在于它会主动冒出来工作。若对象的运作上下文期待字符串的出现，toString() 方法即自动为对象加上格式。例如把博客日期的alert代码改写后，它的输出结果还是完全一样：

```
alert(blogDate);
```



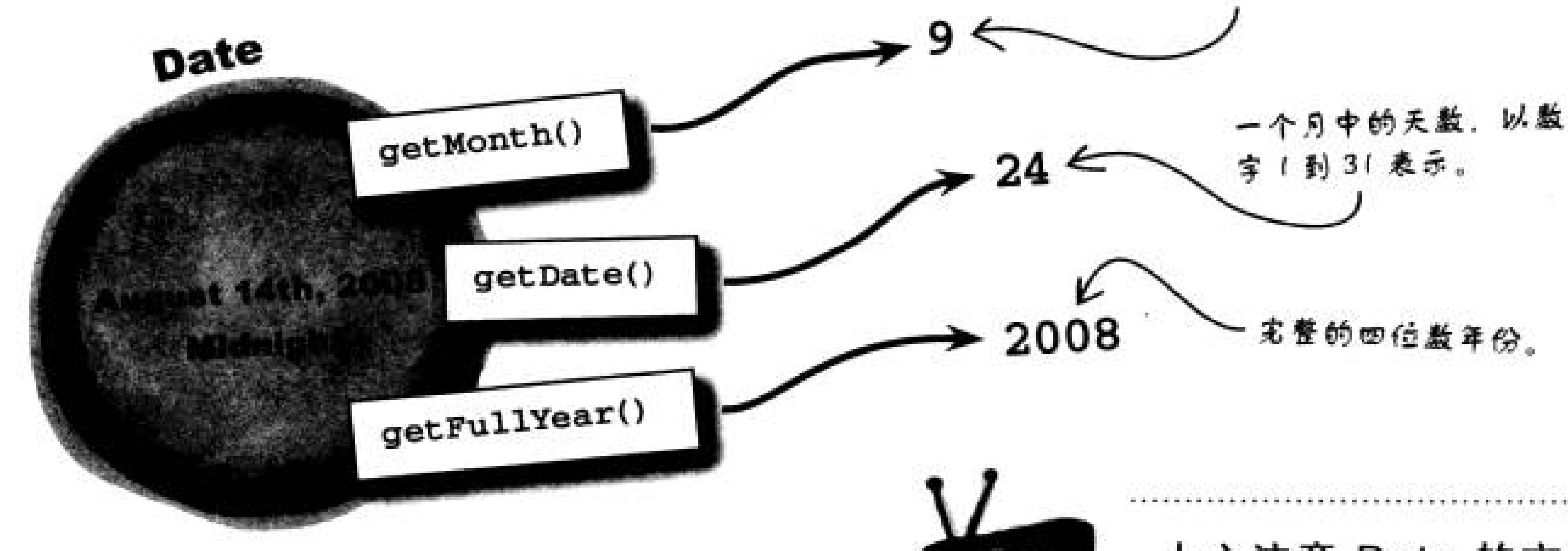
toString()方法提供对象的字符串表现方式。

因为alert()函数预期收到字符串，而Date对象也很聪明，知道它必须提供自身的字符串表现方式，所以它调用toString()方法以处理这项任务。

toString()的业务平常不是问题，除非遇到日期需呈现得简单易读的状况，就像 YouCube 的需求（如采取MM/DD/YYYY格式）。总而言之，YouCube 无法受惠于 Date 对象默认的字符串表现方式（由它的 toString 负责）。

访问日期格式的片段

Ruby 需要自定义日期格式的方式。自定义`Date` 对象格式的关键，在于访问日期的各个片段，例如月、日、年等，然后再把日期片段重组为理想格式。幸好，`Date`对象提供了访问这些片段的方法（method）。



`Date`对象事实上不只支持这三个方法，它提供许许多多访问日期与时刻的方式。不过，上述三个方法已可塑造 YouCube 理想中的日期格式。

注意！

小心注意 Date 的方法返回的值。

getMonth()方法返回的月份数，从0（一月）开始，到11（十二月）为止，但getDate()方法返回的天数却是从 1 到 31。

磨笔上阵

请修正看不懂 YouCube 博客日期的问题，我们需要重写自定义博客日志格式的代码，并把它存储在`blogText`变量里。请确认博客日期的格式为 MM/DD/YYYY。下面列出原始版本：

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";
```

....

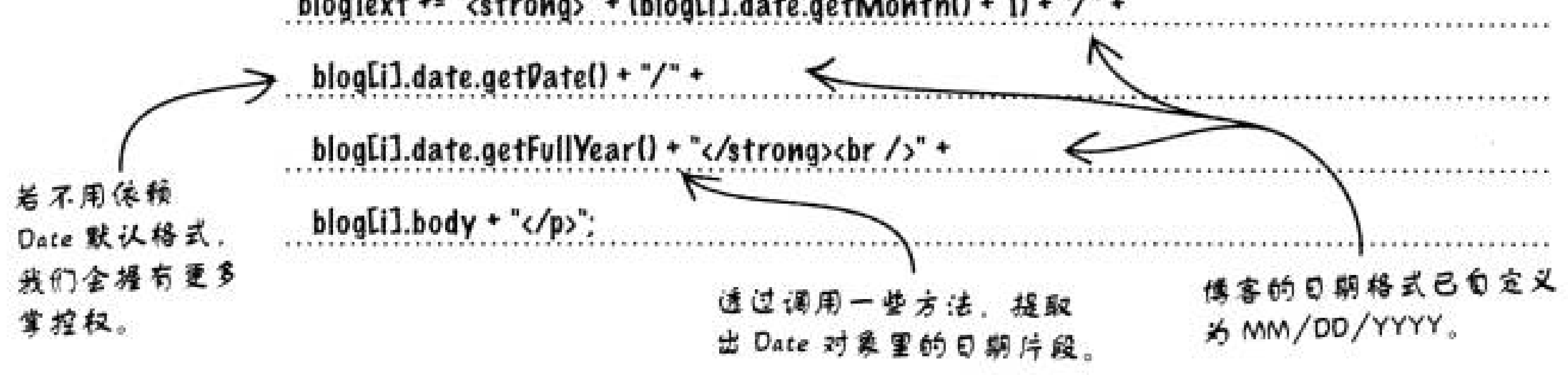
....

....

....

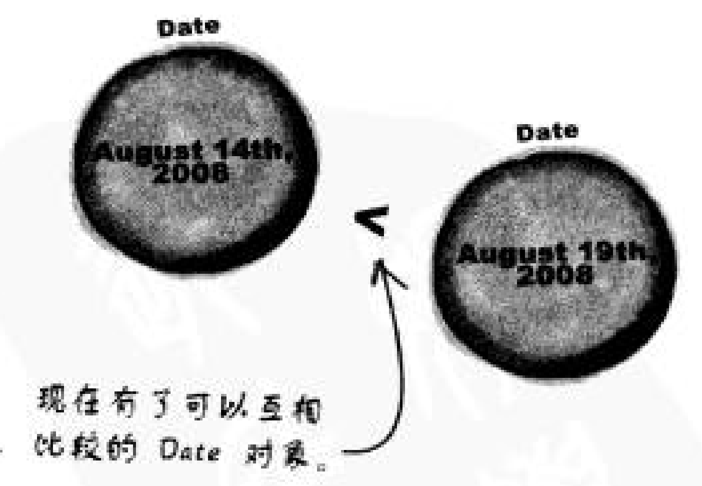
请修正看不懂 YouCube 博客日期的问题，我们需要重写自定义博客日志格式的代码，并把它存储在 blogText 变量里。请确认博客日期的格式为MM/DD/YYYY。下面列出原始版本：

```
blogText += "<strong>" + blog[i].date + "</strong><br />" + blog[i].body + "</p>";
```



Date把排序变简单

博客日期既已成功转换为Date 对象（比字符串更适合的存储方式），也该回头查看排序的问题了。目前，博客日志以存储至 blog 数组时的顺序排列，但存储顺序不见得是日期顺序。另外，大多数博客采用先列出最新文章的（逆向编年）顺序。既然如此，最好逆转原本的博客排序策略：



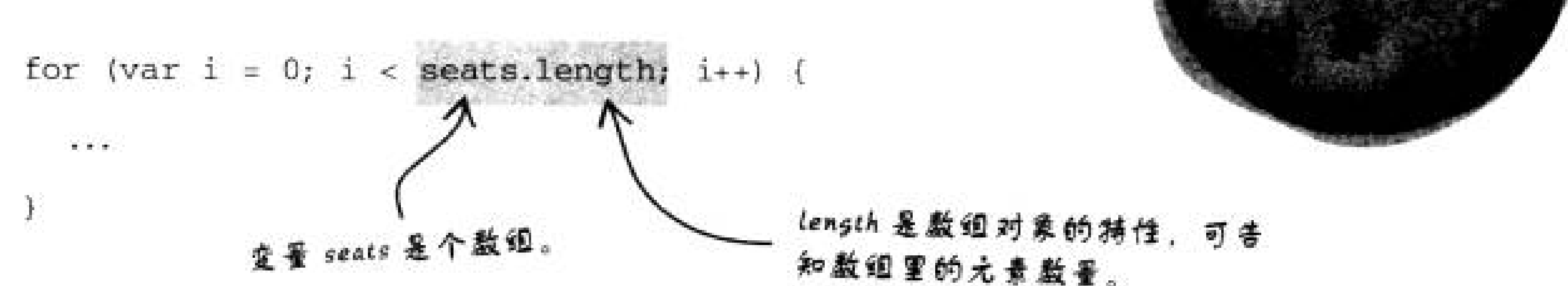
1. 以循环处理博客日志。
2. 比较每个 Blog对象与它的下个对象里的Date对象。
3. 如果下个对象的日期比现在的日志新，则交换其顺序。

虽然加入Date对象的协助后，日期的比较似乎没那么吓人了，但其他策略计划还是需要相当大量的自定义编码。排列日期顺序，听起来是个相当常见的程序设计问题，一定有很多人解决过这个问题。我不想重蹈前人的覆辙……

如果 JavaScript 具有某种内置的排序功能，能为我分担沉闷无聊的日期排序工作就好了…………我会不会妄想太多了呢？

视数组为对象

数组能否自我排序？如果日期知道如何把自己变成字符串，希望“数组也能自我排序”或许就没那么异想天开了。不过，想要美梦成真，数组必须是个对象，才能在方法（method）中排序。事情正是如此。还记得 Mandango 的这段代码吗？



好了，秘密揭晓了，数组是对象，可是这样就表示数组可以自我排序吗？数组不只拥有 length 特性，还有其他根据数组数据而行动、为数据带来生命的方法（method）。而且，有个方法就叫sort()，可为数组里的数据排列顺序。我们一起看看它的运作方式：

nums

0 1 2 3 4 5 6 7 8

一个数字数组。

```
var nums = [ 51, 11, 34, 29, 17, 46, 22, 58, 16 ];
nums.sort();
```

依升幂顺序排列数组。

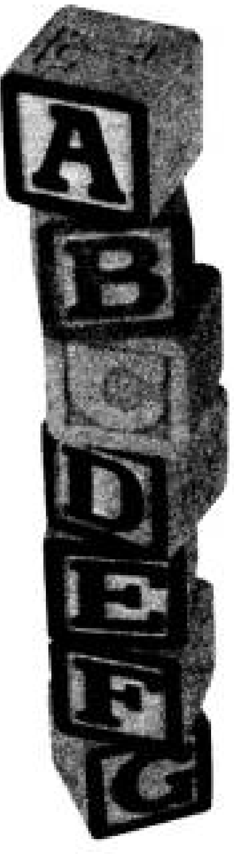
sort()

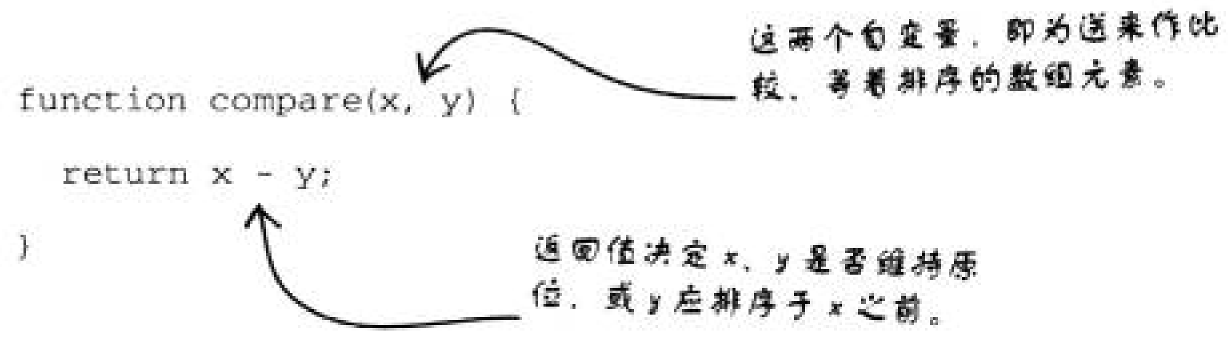
sort()方法改变了数组里的元素顺序。默认采取由小排到大的升幂顺序，所以 nums 数组变成下图所示：

0 1 2 3 4 5 6 7 8

自定义数组排序方式

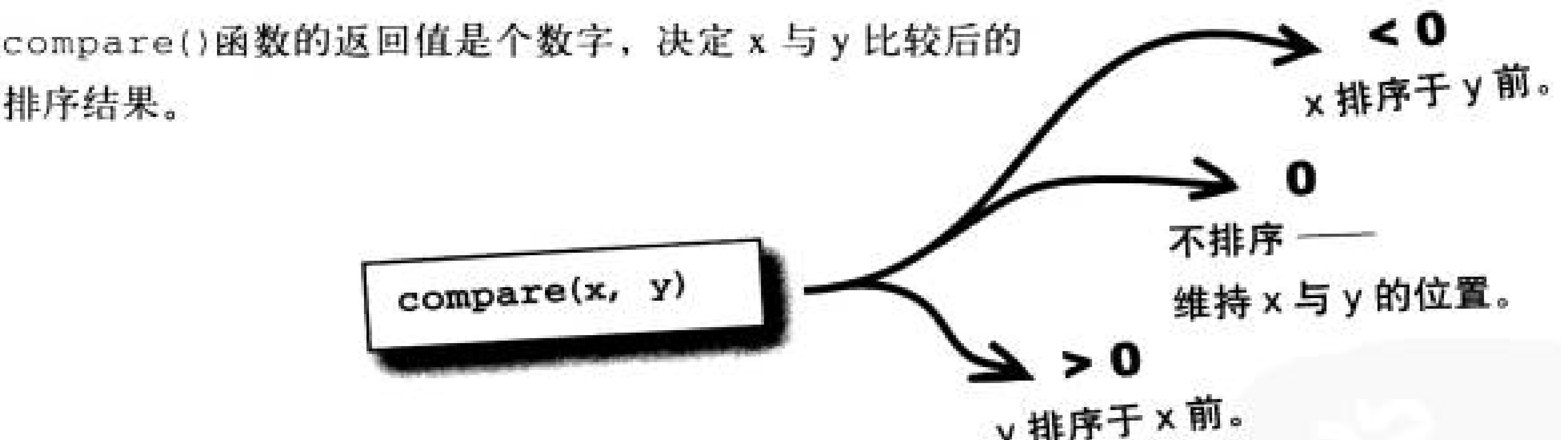
只靠Array对象的sort()方法的默认行为通常不够。还好，排序行为可由**比较函数**（comparison function）决定。sort()调用比较函数，用于比较数组中各个等着排序的元素。自己提供比较函数，即可调整排序方式。下例是比较函数的常见形态：



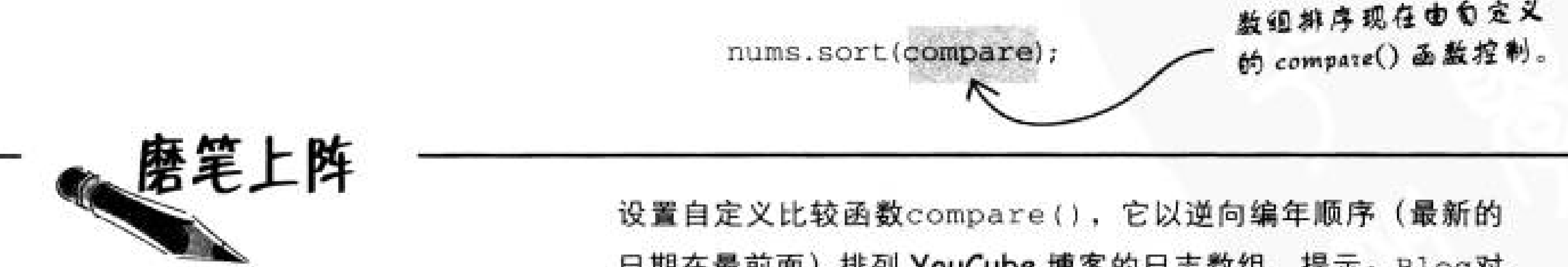


```
function compare(x, y) {
  return x - y;
}
```


compare()函数的返回值是个数字，决定 x 与 y 比较后的排序结果。



调用sort()方法时，把你自定义的 compare() 函数注射到数组排序等式里 —— 只需把 compare() 函数的引用传给方法。



```
nums.sort(compare);
```

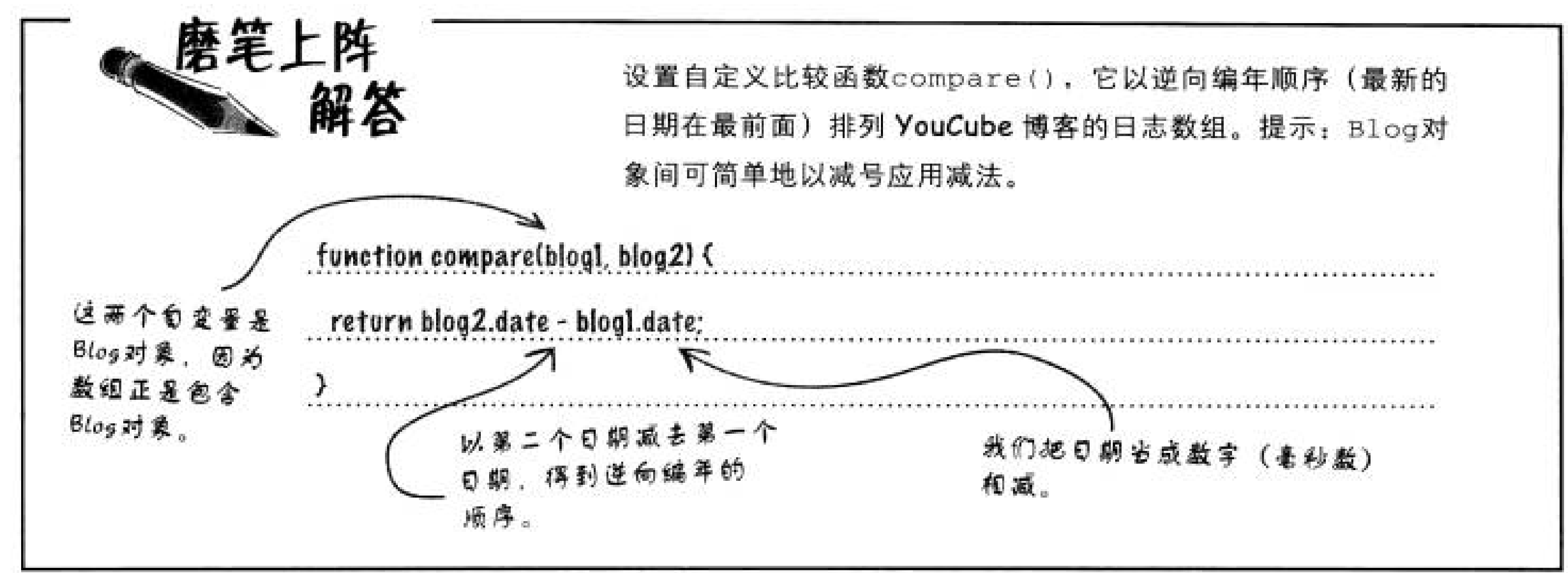

磨笔上阵

设置自定义比较函数compare()，它以逆向编年顺序（最新的日期在最前面）排列 YouCube 博客的日志数组。提示：Blog对象间可简单地以减号应用减法。

....

....

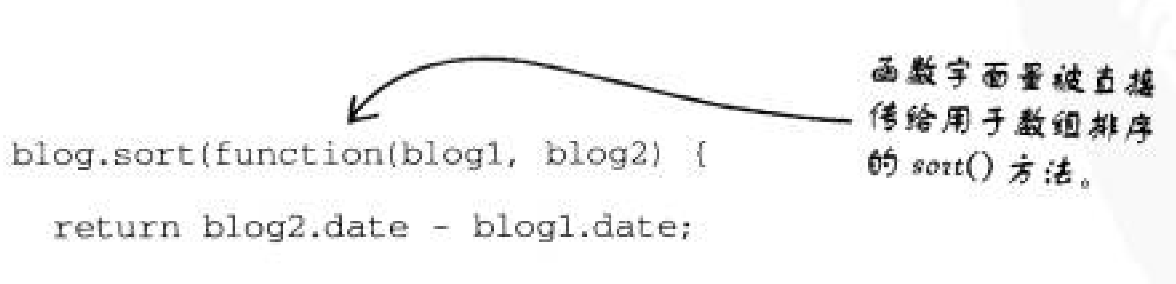
....



利用函数字面量，排序变得简单

讲到数组排序用的比较函数，它只是供sort()方法使用，完全不会出现在别的地方。既然比较函数不会被 YouCube 代码的其他部分调用，它其实不需是个有名称的函数。

还记得第 6 章提过函数字面量吗？compare()的用途使它成为函数字面量的理想候补。事实上，若把 compare() 转换为函数字面量，直接传入 sort() 方法，可进一步简化 YouCube 博客的排序功能。



身为虔诚的解谜者，Ruby 非常重视效率。此时，效率等于删除不必要的函数，而这个不必要的函数只是sort()方法的助手。Ruby真的很重视效率，所以她不懂比较函数为什么需要用到三行代码。虽然 JavaScript代码的排版方式并不会影响程序代码的运作，但本例的函数字面量因为够简单，把代码压缩成一行也自有其道理。

函数字面量挤入一行代码中。

```
blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });
```

没有蠢问题

问：每个对象都有toString方法吗？

答：是的。就算你创建了自定义对象，且并未给它toString方法，但在期待字符串的上下文中，JavaScript至少也会汇报遇到了对象。当然，此时的字符串不算非常有意义，但是否为自定义对象提供toString则是你的自由，如果你希望它能传达关于对象的意义的话。

问：在Date对象间，排序的比较如何运作？

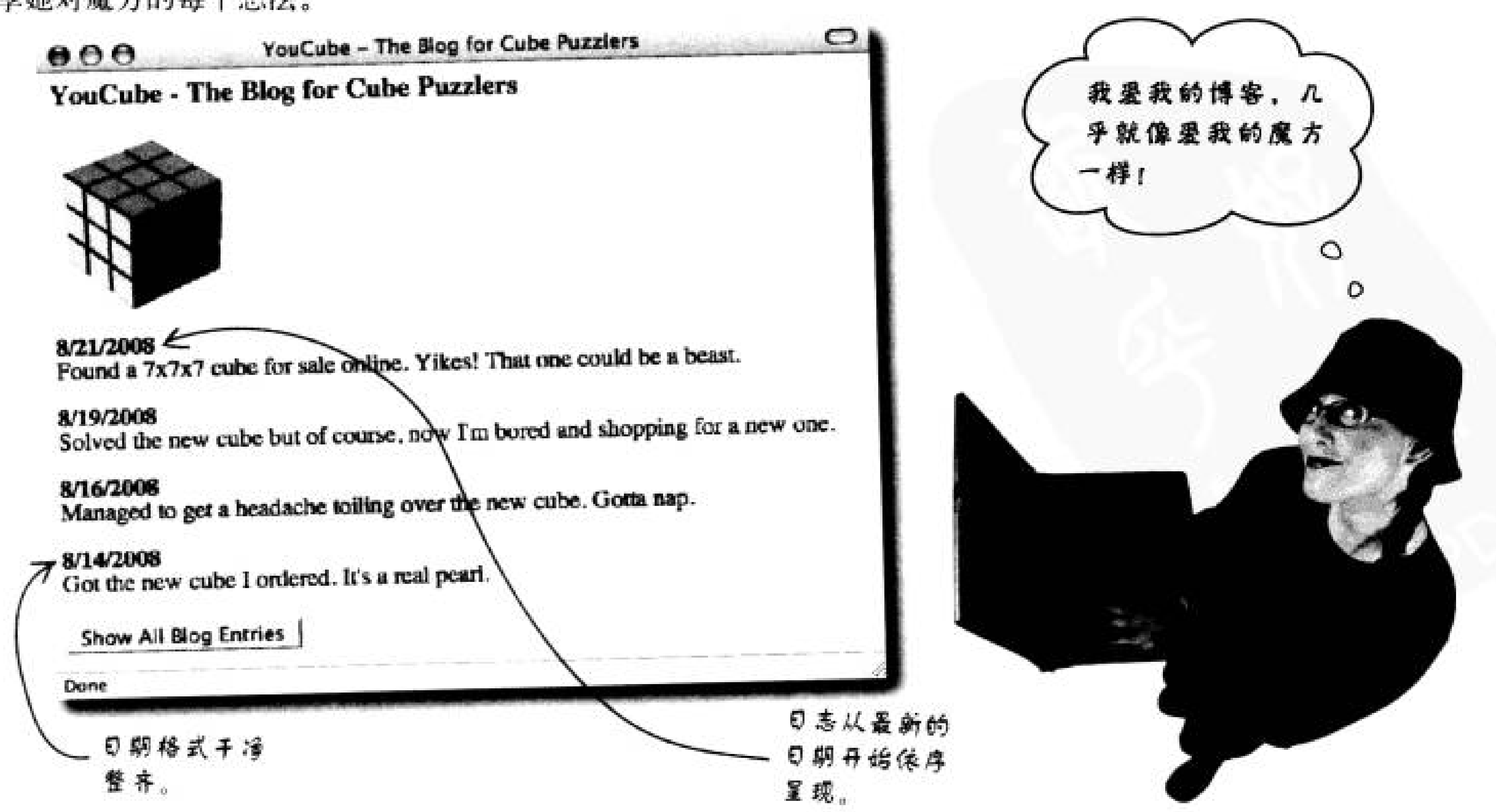
答：（排序用）比较函数的目标是返回数字，这些数字控制两个自变量的排序结果。我们的范例用于比较日期，而且希望较新的日期被排序在前。较新的日期也是较“大”的日期，所以，从第二个日期里减去第一个日期，可达成把较新日期排在较旧日期前的目标。也就是说，只在第二个日期大于第一个日期时（结果大于0），才排在第一个日期前。

问：Array.sort()方法该如何知道使用自定义比较函数或默认的比较方式？

答：基本上根据是否有自变量传给sort()而决定。如果没有自变量，则采用默认方式；如果提供了自变量，则解释为函数引用，并作为排序时的比较基础。所以比较函数是个可选（optional）自变量。

Ruby和魔方都很快乐

YouCube博客现在很接近Ruby想象中的魔方专门志，能与全世界分享她对魔方的每个想法。



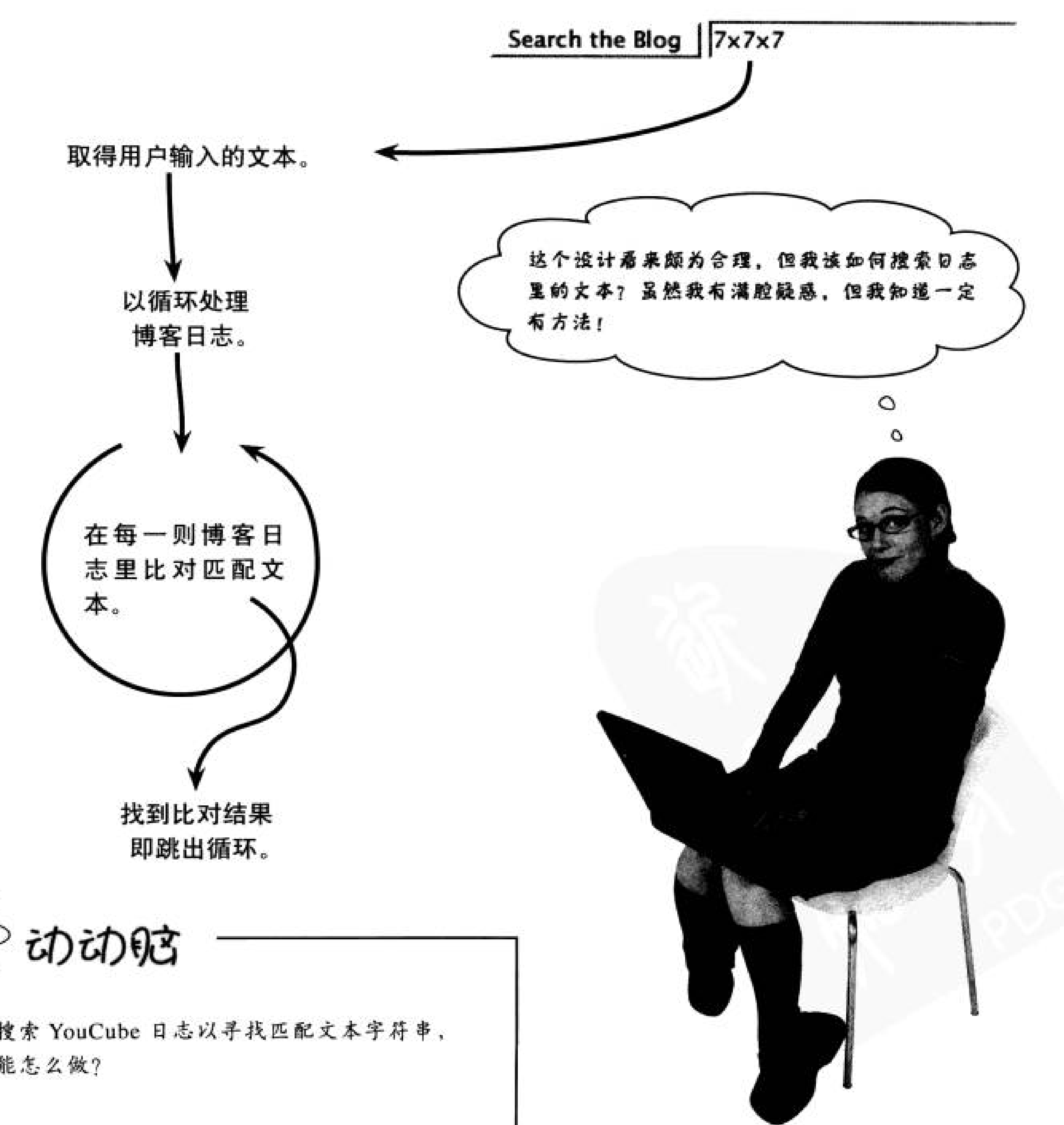
能够搜索会更好

YouCube 的运作十分平顺，不过有几位用户希望能有“搜索博客日志”的功能。既然 Ruby 的计划中将有非常多日志，她也觉得搜索会是非常方便的功能，尤其就长远考量而言。

Ruby 需要规划 YouCube 的搜索程序代码……有可能用到对象吗？

搜索 blog 数组

YouCube 的搜索功能牵涉到以循环处理 `blog`博客数组中的每则日志，在每篇文章中寻找匹配的文本。



动动脑

关于搜索 YouCube 日志以寻找匹配文本字符串，你可能怎么做？

字符串是个可以搜索的对象。

有人或许已经发现JavaScript里到处都是对象。字符串是对象，而且包含很多与字符串数据（text）交互的便利方法。没错，其中有个方法可以搜索字符串里的文本片段。字符串里的字符串有时被称为**子字符串**（substring）。

搜索字符串内部：indexOf()

indexOf()方法，可在String对象里搜索一段文本，或称子字符串。以子字符串作为自变量，传给indexOf() 方法 —— 因为你在String 对象里调用这个方法，故无需传入其他数据。indexOf()方法返回子字符串位置的索引值，或于找不到匹配子字符串时返回 -1。

```
var str = "Got the new cube I ordered. It's a real pearl.";
alert(str.indexOf("new"));
```

8

OK

想了解上例的结果 —— 8，你必须把字符串当成由字符组成的数组。

0 10 20 30 40

"Got the new cube I ordered. It's a real pearl."

搜索的目标字符串"new"出现在字符串中索引 8 的位置。

字符串里的每个字符均标有独一无二的索引位置，索引从字符串起始处由 0 开始计算。

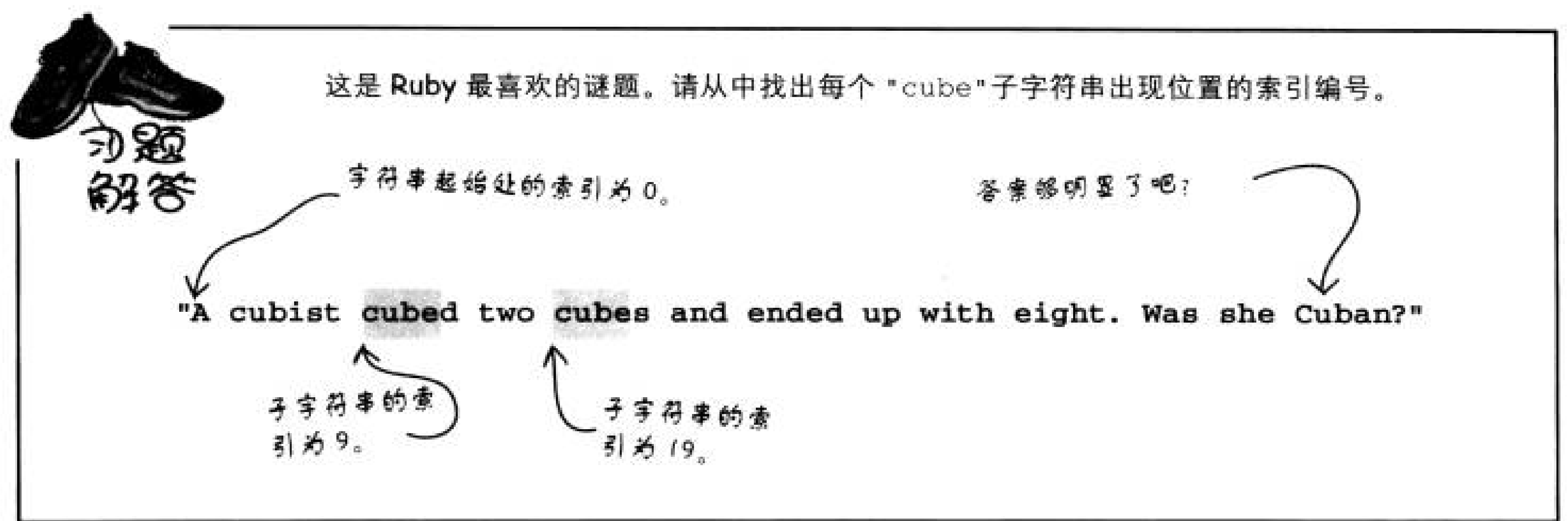
indexOf() 若用于寻找不存在的字符串，结果将为 −1。

```
var searchIndex = str.indexOf("used");
```

结果为 −1，因为目标字符串并未出现在 String 对象里。

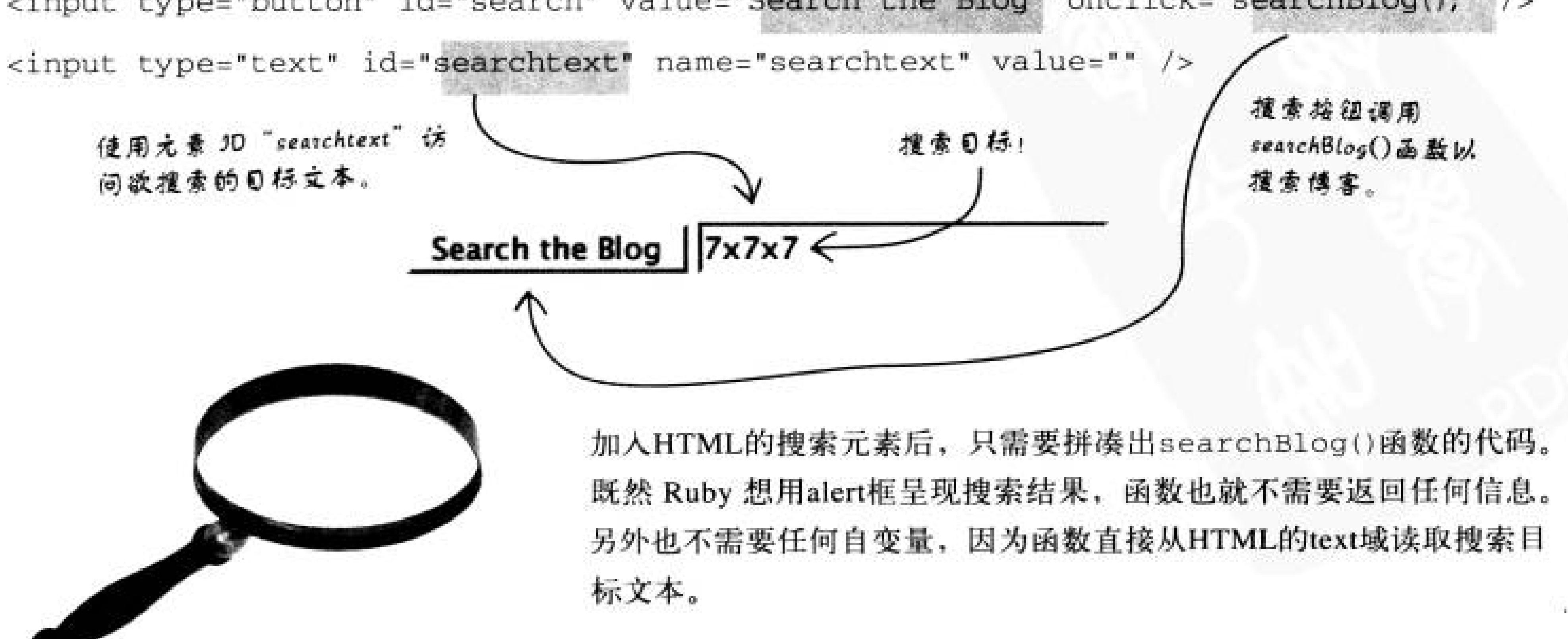
这是 Ruby 最喜欢的谜题。请从中找出每个"cube"子字符串出现位置的索引编号。

"A cubist cubed two cubes and ended up with eight. Was she Cuban?"



搜索 blog 数组

有了String对象的indexOf()方法协助后，字符串搜索已经不算太难，但 Ruby 仍然需要搜索整个博客。她计划以循环处理博客数组，然后使用 indexOf() 方法，逐一在每则日志的正文内容里搜索子字符串。如果找到相符字符串，她希望以 alert框表示。



加入HTML的搜索元素后，只需要拼凑出searchBlog()函数的代码。既然 Ruby 想用alert框呈现搜索结果，函数也就不需要返回任何信息。另外也不需要任何自变量，因为函数直接从HTML的text域读取搜索目标文本。

JavaScript 冰箱磁铁

YouCube 的 searchBlog() 函数，负责以循环处理博客日志的数组，并于日志正文中搜索匹配文本。请帮助 Ruby 完成这个函数，利用下面的磁铁填入失落的程序代码。提示：搜索目标的匹配结果里，应该把日志的日期（采用 MM/DD/YYYY 格式）放在一对中括号（[]）里，后随日志的正文内容。

```
function searchBlog() {

  var ............ = document.getElementById("............").value;

  for (var i = 0; i < ............ ; i++) {

    // See if the blog entry contains the search text

    if (blog[i]. ........ .toLowerCase().indexOf(searchText.toLowerCase()) != -1) {

      alert("[" + (blog[i]. ........ . ............ + ..... ) + "/" +

        blog[i].date.getDate() + "/" + blog[i]. ........ .getFullYear() + "] " +

        blog[i]. ........ );

      break;
    }
  }

  // If the search text wasn't found, display a message

  if (i == ............ )

    alert("Sorry, there are no blog entries containing the search text.");
}
```

searchText

blog.length

date

1

searchtext

body

getMonth()

JavaScript 冰箱磁铁解答

YouCube 的 searchBlog() 函数，负责以循环处理博客日志的数组，并于日志正文中搜索匹配文本。请帮助 Ruby 完成这个函数，利用下面的磁铁填入失落的程序代码。提示：搜索目标的匹配结果里，应该把日志的日期（采用 MM/DD/YYYY 格式）放在一对中括号（[]）里，后随日志的正文内容。

首先，从 HTML 的 text 域取得搜索目标文本。

```
function searchBlog() {
  var searchText = document.getElementById("searchtext").value;
  for (var i = 0; i < blog.length; i++) {
    // See if the blog entry contains the search text
    if (blog[i].body.toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
      alert("[" + (blog[i].date.getMonth() + 1) + "/" +
        blog[i].date.getDate() + "/" + blog[i].date.getFullYear() + "] " +
        blog[i].body);
      break;
    }
  }

  // If the search text wasn't found, display a message
  if (i == blog.length)
    alert("Sorry, there are no blog entries containing the search text.");
}
```

如果 i 等于 blog 的长度，表示 for 循环已经跑过整份日志，但没有搜索到匹配结果。

日志中若有匹配文本，则使用中括号围起 MM/DD/YYYY，后随正文内容。

搜索功能也上线了！

YouCube 2.0的搜索功能现在已经完工，重度仰赖内置于String对象的搜索功能。它是个对象为数据带来生命的绝佳范例，本例中，文本字符串（纯数据）转变为一个具有行为（可以自我搜索）的实体。或许更为重要的是：Ruby 因此不需再发明一次搜索例程，让她可以专心把时间放在博客的经营上。

Ruby 对新的博客功能兴奋不已，但她还想追求更大的成就，她的心中已经在计划 YouCube 3.0。

没有蠢问题

问：我还是不太了解每个字符串为什么其实是个对象耶。真的是这样吗？

答：真的。JavaScript 下的每个字符串都是个对象。如果你在JavaScript 中把姓名放在引号里，如"Ruby"，其实你已经创建了一个对象。虽然看起来有点夸张，但 JavaScript 把所有字符串视为对象的好处，在于每个字符串都能完成一些方便好用的工作，如知道自己的长度、搜索子字符串等等。

问：我知道字符串是个对象了。不过，它也很像具有字符索引的数组。字符串也是个数组吗？

答：不是。字符串绝对不是个数组。不过，许多 String 的方法操作字符串数据的方式，的确很像面对由字符组成的数组。举例而言，在字符串内的字符索引从 0 开始，随着在字符串里移动，索引值也递增。但你无法以中括号（[]）访问字符串内的某个字符，数组才能这样做。虽然说，把**字符串内的字符**类比于**数组的元素**有所帮助，但实际处理 String 对象时，仍然与处理 Array 对象不同。

问：searchBlog()函数能够不采用 indexOf()，改以charAt()搜索博客吗？

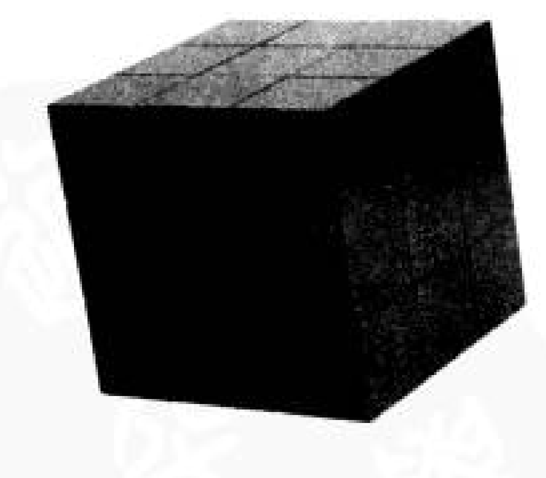
答：不行。charAt() 方法只搜索单一字符，在寻找短语或文本时并无大用。indexOf()方法则搜索字符串，而不只搜索单一字符，它是搜索博客的最佳工具。

问：有可能在一个字符串里，寻找某个子字符串出现的每个地方吗？

答：可以。indexOf()方法默认为搜索子字符串第一次出现的位置。但你也可以传入选用的第二个自变量，告诉indexOf()从何处开始搜索。假设“cube”是搜索目标，而你已经在索引 11 的位置发现匹配文段（a match）。你可以再次调用 indexOf()，并加上第二个自变量 11，强制要求搜索从索引 12 的位置开始。所以，通用方案是把上一次搜索到的索引传入indexOf()方法，以继续搜索字符串。

问：searchBlog()函数里为什么要调用两次 toLowerCase() 呢？

答：好问题。答案与搜索博客时遇到的文本大小写问题有关。如果有人搜索“cube”，他大概希望找出所有匹配这个词汇的结果，包括 cube、Cube、CUBE及其他大小写变化。避开这个问题的简单方式，就是一并转换**搜索目标**与**博客正文**的大小写，再继续搜索任务。虽然searchBlog()函数采用toLowerCase()函数，但 toUpperCase()也有相同功用。重点只是彻底消除搜索中的大小写影响因素。



复习要点

- toString方法用于转换任何对象为文本表达形式。
- 数组和字符串事实上都是对象，依靠 JavaScript的标准对象Array和 String 提供方法与数据的存储。
- Array对象的sort()方法能依任何顺序排列数组。
- String对象的indexOf()方法在字符串内搜索另一个字符串，返回搜索目标的索引位置。

随机的 YouCube

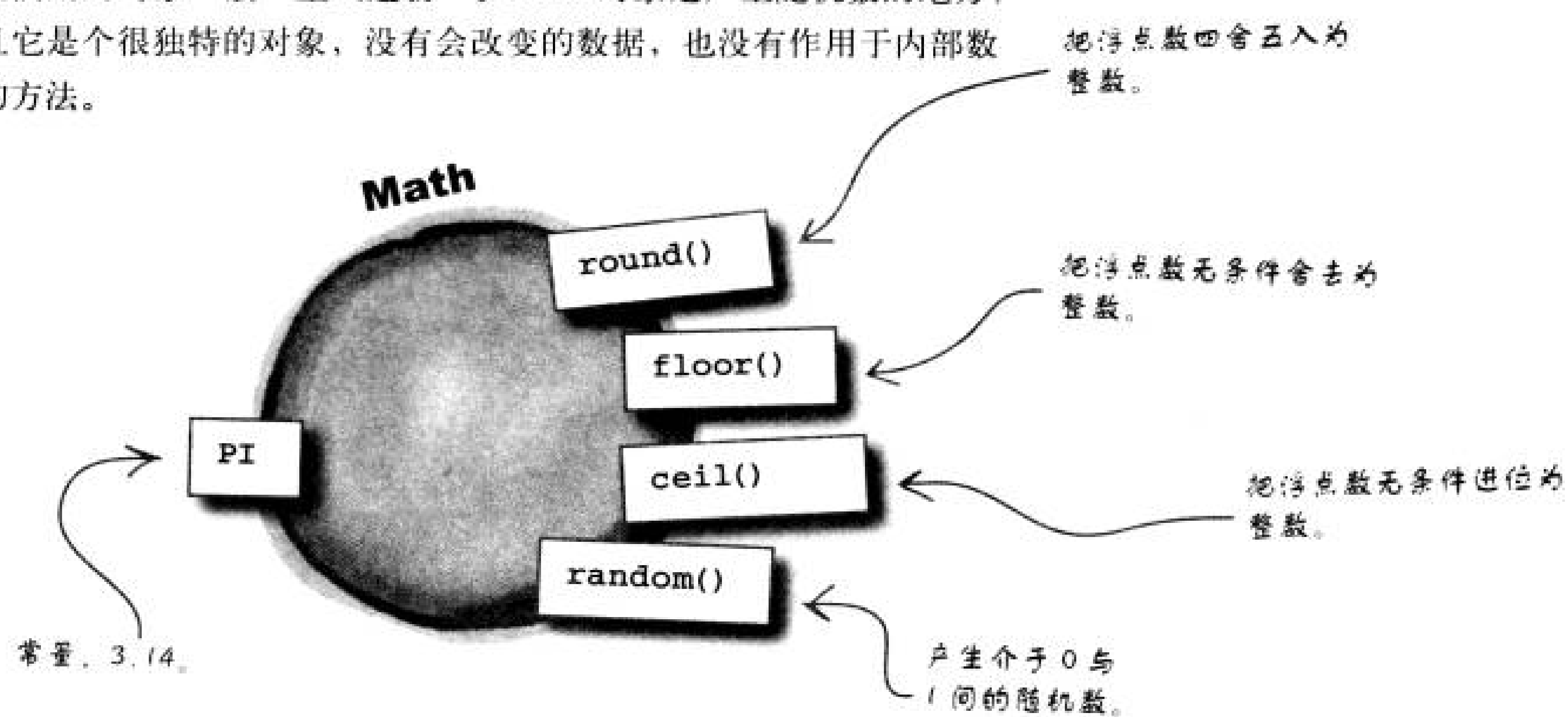
维持读者的兴趣是项永无止境的任务，Ruby 想为 YouCube 加上另一项魔方同好可能有兴趣的功能。她想加上一个“Random”（随机）按钮，让用户随机读取博客日志。

动动脑

你该如何随机选取博客日志?

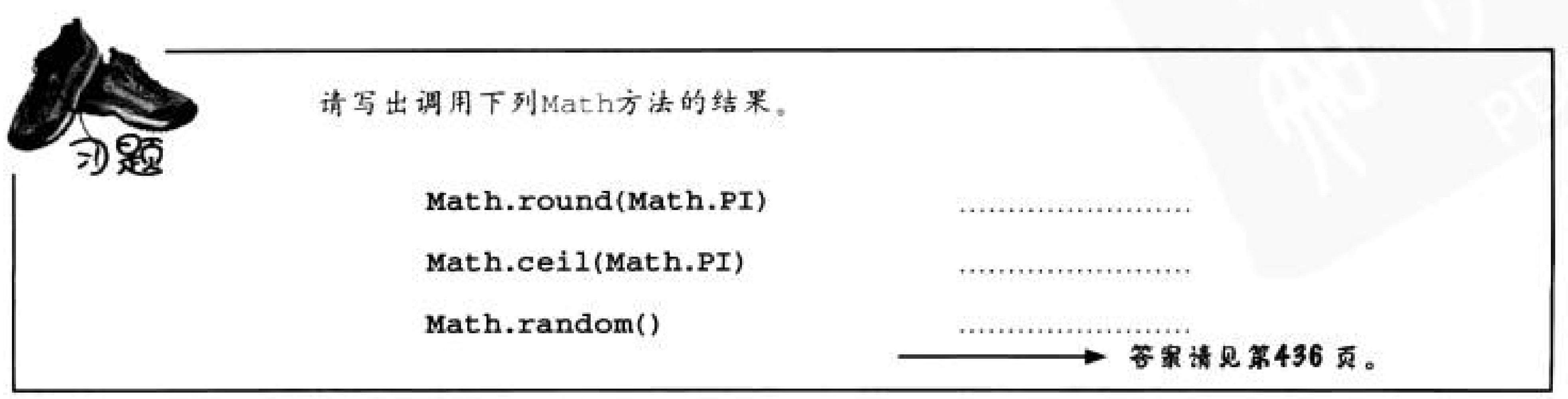
Math 对象是个 organizational object

为了帮助 Ruby 加上 YouCube 的随机功能，我们非常需要产生随机数的方式。这项工作关系到使用某个 JavaScript 的内置对象，它不像其他我们用的对象一般“生气蓬勃”。Math 对象是产生随机数的地方，而且它是个很独特的对象，没有会改变的数据，也没有作用于内部数据的方法。



Math对象是个organizational object（组织对象），它只是数学相关的公用方法与常量的集合。Math对象中没有变量，意即这个对象没有状态——不能用于存储任何事物。它唯一包含的数据只是几个常量，例如 PI(3.14)。不过，Math 对象的方法很好用。例如random()方法就能产生介于 0 与 1 间的随机数。

Math 对象是个收藏数学方法与常量的 organizational object。



请写出调用下列Math方法的结果。

```
Math.round(Math.PI)     ........................
Math.ceil(Math.PI)      ........................
Math.random()           ........................
```

答案请见第436页。

Math 对象真情指数

本周主题：当数学函数抵触时

Head First： 咳，我真的很困惑。你是个对象，但我又听说你不负责其他事，只负责保存一些数学运算方法和几个常量。我一直以为，对象的职责就是让数据有生命力。你知道的，就是包起一些数据，然后让方法根据数据，形成一些很炫的结果。

Math： 因袭常见的 JavaScript 智慧让大家都这么想，但并非所有对象都用于为数据带来生命。对象也可以扮演组织人员的角色，就像我。

Head First： 可是，这些数学方法不能创建为标准函数吗？

Math： 是的，也可以，但你忘记了JavaScript语言原本就以对象建立的。所以就事实而言，其实没有"标准"函数这回事。

Head First： 但我可以在对象外创建函数，而且它们也能运作良好啊？

Math： 没错，但事实上，所有函数都是方法，它们属于某个对象，即使你看不到藏起来的对象。这样应该有助于解释没有"标准函数"的问题。

Head First： 嗯，原来如此。连你包含数学方法的原因，我也开始有点概念了。

Math： 也别忘记，虽然我没有内部数据可供方法操纵，但不表示我这个对象失去了重要性。

Head First： 这话怎么说咧？

Math： 假设有群人都对魔方很感兴趣，他们常常集合在一起，为了有兴趣的主题互相交流。数学方法虽然不像人类一样有社交性，但的确因为我提供的组织性而受益。

Head First： 你是指因为它们都具有共同的兴趣？

Math： 是的！而这份兴趣达成了数学任务，例如四舍五入数字、计算三角函数、产生随机数等等。

Head First： 你刚刚提到产生随机数。我听说你产生的数字并非真正随机，这是事实还是谣言呢？

Math： 我必须承认，不是真正的随机。大部分计算机产生的随机数也非真正的随机数字。我的随机数是"伪随机数"，大部分情况下已经足够了。

Head First： 伪随机数，就像伪科学……或是伪代码？

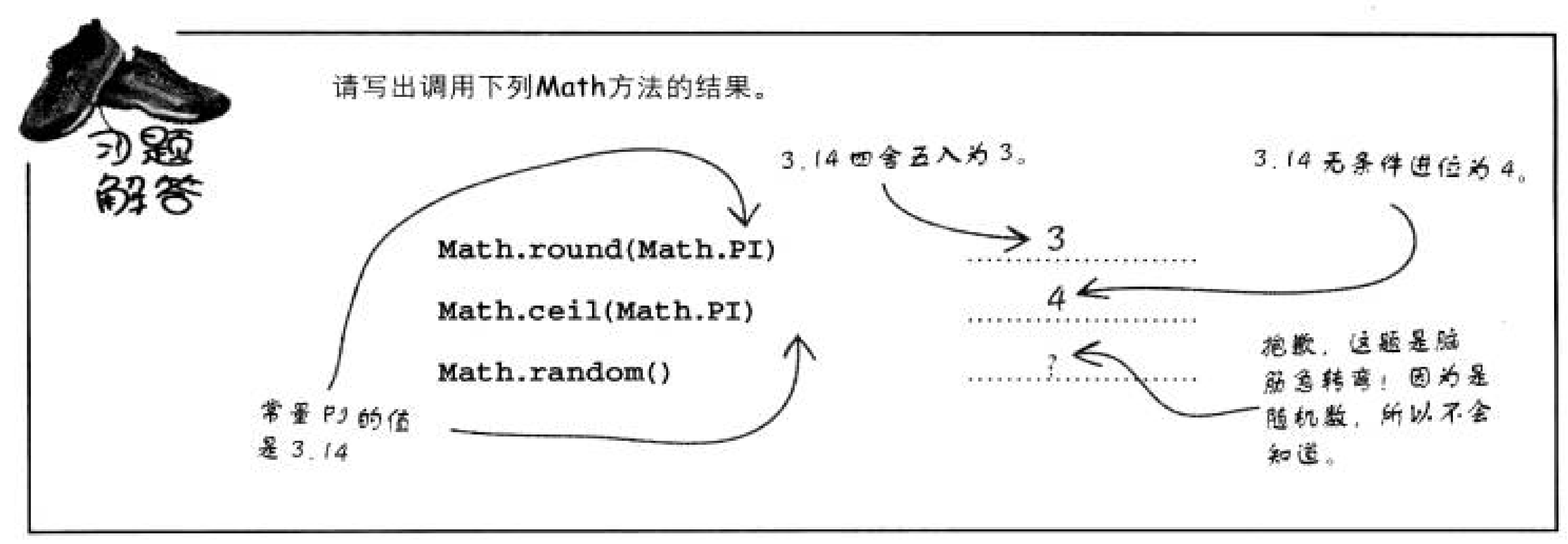
Math： 呃，前者不对，后者是对的。我跟伪科学一点关系都没有。还有，是的，有点像是伪代码，因为伪代码试图表达代码背后的意义，但又并非真正的程序代码。伪随机数的状况则是近似随机数，但不是真正的随机数。

Head First： 这样子啊……"伪随机数"对大多数 JavaScript 应用程序而言，够随机了吗？

Math： 可以。而且你说的"够随机"也是很好的形容方式。如果是国家安全这种大阵仗，大概就不能信任伪随机数；但如果是在日常使用的脚本里注入伪随机数，它们的工作情况就还不错。

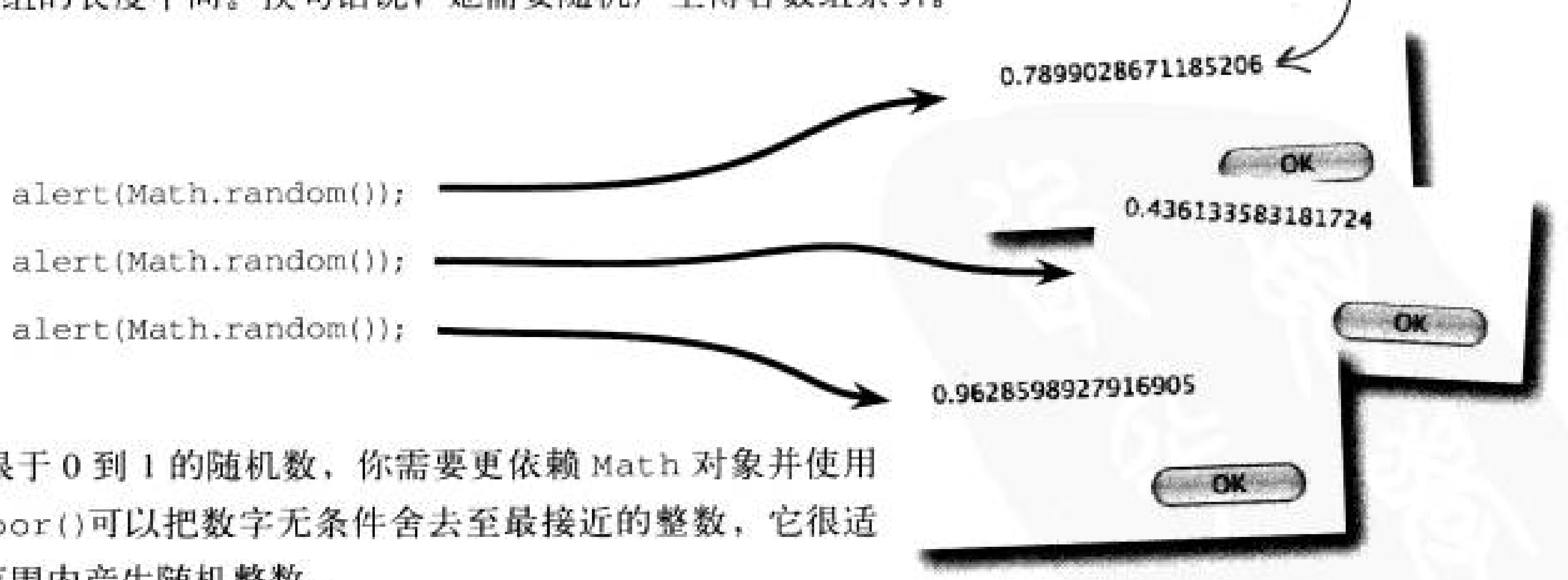
Head First： 了解。谢谢你接受今天的访问……尤其是关于随机数的诚实说明。

Math： 我也很高兴……你知道的，我没办法说谎。

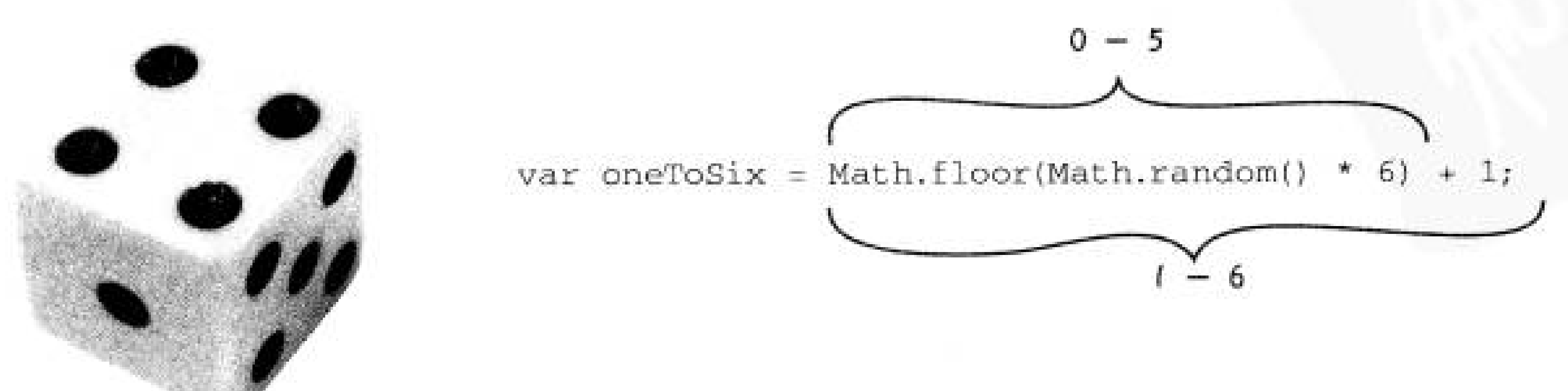


使用 Math.random 产生随机数

无论是否为伪随机数，由Math对象的random()方法产生的随机数，对YouCube这类需要从一组数据中随机选择的应用程序非常有用。问题在于，random()返回的数字介于0与1中间；但Ruby需要的随机数则是介于0与blog数组的长度中间。换句话说，她需要随机产生博客数组索引。



想产生范围不限于0到1的随机数，你需要更依赖Math对象并使用其他方法。floor()可以把数字无条件舍去至最接近的整数，它很适合在既定整数范围内产生随机整数。



没有蠢问题

问：为什么在使用Math对象前，不需先创建这个对象呢？

答：嗯，这是个关于概念的问题，而且碰触到一个关于对象的极重要概念。既然Math对象并未实际包含可以使用的数据（也称为 instance data），所以不需要创建对象。Math 对象只是静态（static）方法和常量的集合，进入 Math 对象里的所有事物均已存在 —— 没有创建任何事物的需要。到了第 10 章，讨论过对象实例（object instance）和类后，你会比较容易了解这个主题。

问：Math对象的round()和floor()方法，有什么不同？

答：round()方法根据小数点后的位数决定进位或舍去。例如 Math.round(11.375) 的结果为 11，Math.round(11.625) 的结果则为 12。另一方面，floor()方法则直接舍去，不管小数点后的位数。你可以把 floor()视为直接砍掉小数点后的尾数。

问：Math 对象还有哪些能耐呢？

答：很多哦！像是两个我们目前用不到的便利方法 min() 与 max()，用于分析两个数字，分别回传较小或较大的数字。abs() 则是另一个非常好用的 Math 方法 —— 它负责回传正数，无论你给它什么数字。

技客新知

如果你发现自己遇到严重需要正牌随机数的 JavaScript 程序，请移驾到 http://random.org，了解如何超越伪随机数的国度。

磨笔上阵

请设计randomBlog()的函数代码，它能随机挑选博客的日志，并以alert框呈现挑选结果。提示：以alert框呈现的日志，可采用与searchBlog()相同的格式。

....

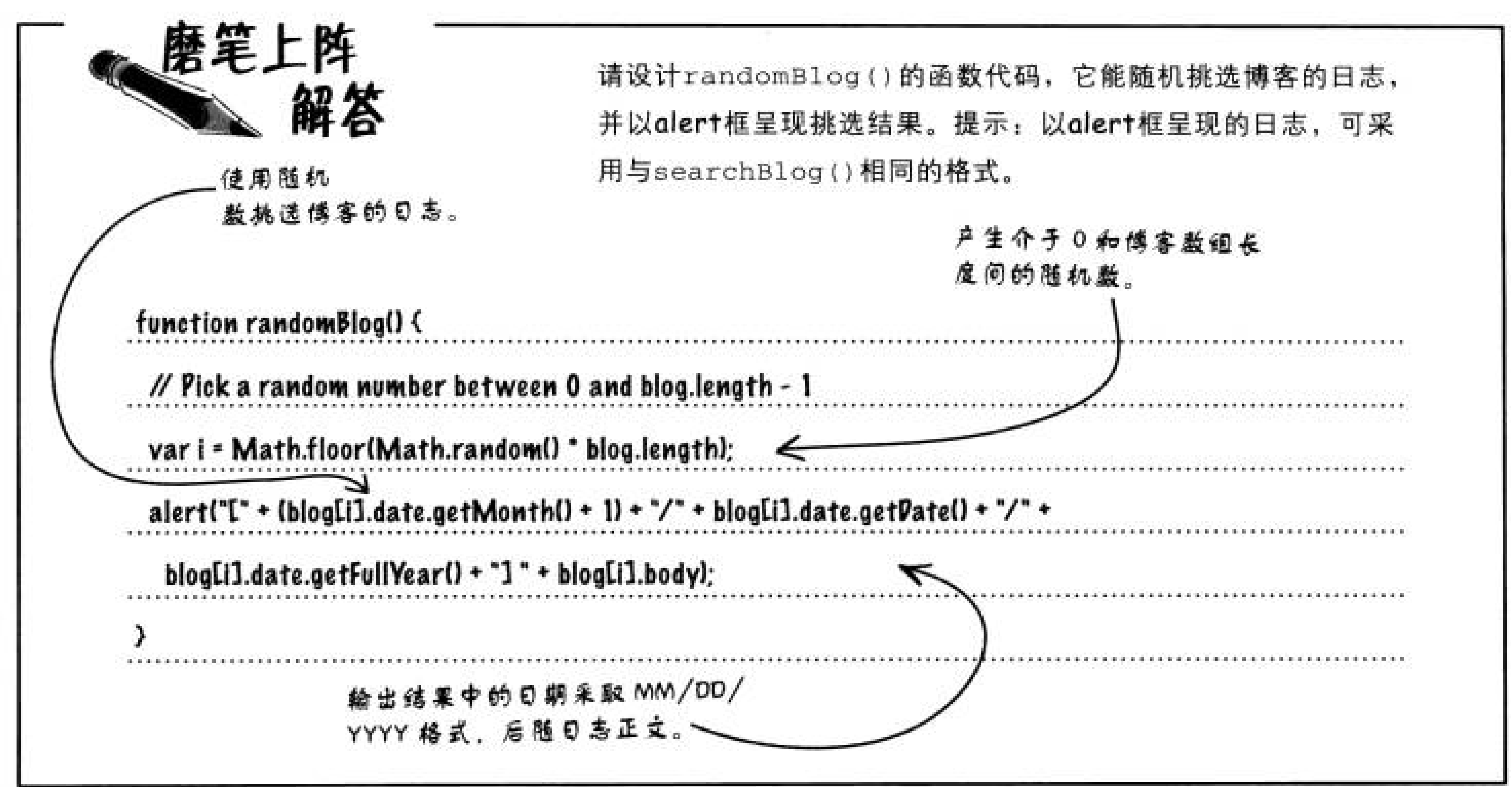
....

....

....

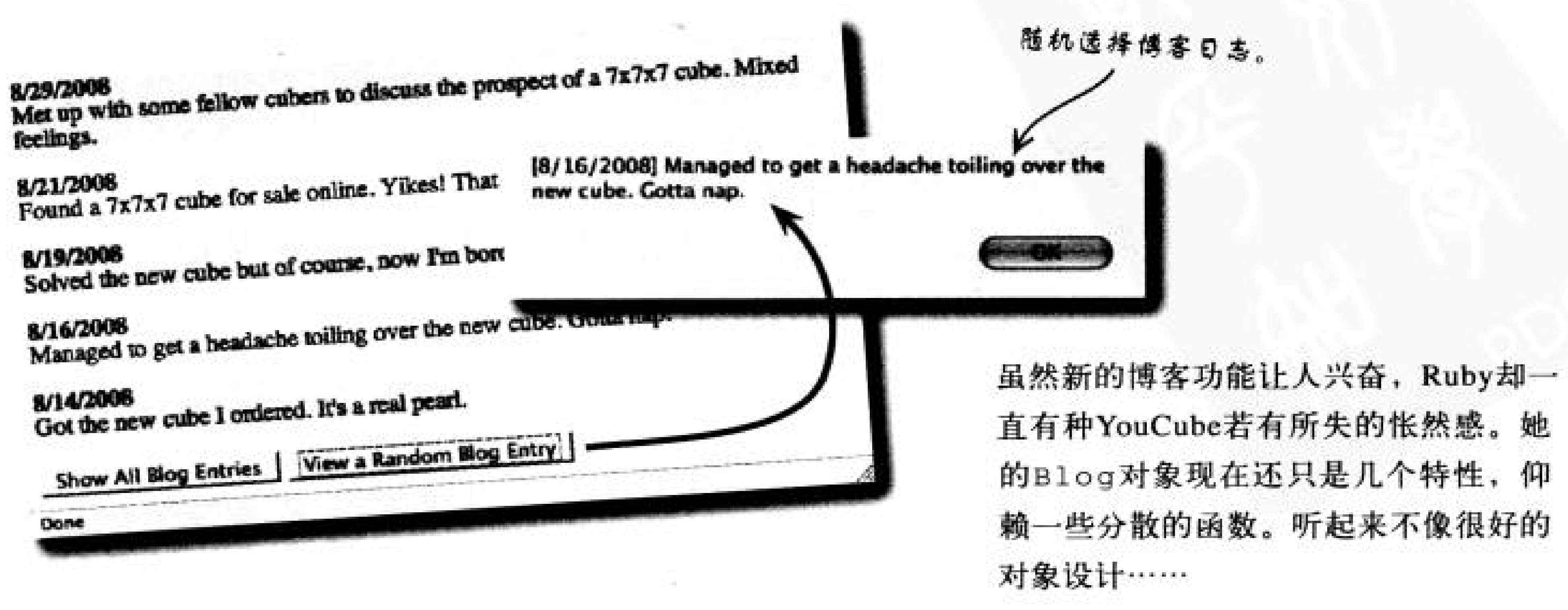
....

....



随机但尚有不足

Ruby 的博客已能支持随机搜索功能，她很高兴。现在，浏览 YouCube 博客的用户，都相当地好奇，因为他们无法预测出现的日志。



虽然新的博客功能让人兴奋，Ruby却一直有种YouCube若有所失的怅然感。她的Blog对象现在还只是几个特性，仰赖一些分散的函数。听起来不像很好的对象设计……

对象寻求行动

Ruby 关于 YouCube 对象的直觉完全正确。目前极度缺乏对象的行为部分，适合认真地重新架构，让对象以方法管理博客专用的任务。Ruby 需要为 Blog对象增加**行动**的方法！

请研究 YouCube 的代码，并圈出任何你觉得可以制成 Blog方法的部分。请记得为每个方法命名。

```
function showBlog(numEntries) {
  // First sort the blog in reverse chronological order (most recent first)
  blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

  // Adjust the number of entries to show the full blog, if necessary
  if (!numEntries)
    numEntries = blog.length;

  // Show the blog entries
  var i = 0, blogText = "";
  while (i < blog.length && i < numEntries) {
    // Use a gray background for every other blog entry
    if (i % 2 == 0)
      blogText += "<p style='background-color:#EEEEEE'>";
    else
      blogText += "<p>";

    // Generate the formatted blog HTML code
    blogText += "<strong>" + (blog[i].date.getMonth() + 1) + "/" +
      blog[i].date.getDate() + "/" +
      blog[i].date.getFullYear() + "</strong><br />" +
      blog[i].body + "</p>";

    i++;
  }

  // Set the blog HTML code on the page
  document.getElementById("blog").innerHTML = blogText;
}

function searchBlog() {
  var searchText = document.getElementById("searchtext").value;
  for (var i = 0; i < blog.length; i++) {
    // See if the blog entry contains the search text
    if (blog[i].body.toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
      alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
        blog[i].date.getFullYear() + "] " + blog[i].body);
      break;
    }
  }

  // If the search text wasn't found, display a message
  if (i == blog.length)
    alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog() {
  // Pick a random number between 0 and blog.length - 1
  var i = Math.floor(Math.random() * blog.length);
  alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
    blog[i].date.getFullYear() + "] " + blog[i].body);
}
```

请研究 YouCube 的代码，并圈出任何你觉得可以制成 Blog方法的部分。请记得为每个方法命名。

```
function showBlog(numEntries) {
  // First sort the blog in reverse chronological order (most recent first)
  blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

  // Adjust the number of entries to show the full blog, if necessary
  if (!numEntries)
    numEntries = blog.length;

  // Show the blog entries
  var i = 0, blogText = "";
  while (i < blog.length && i < numEntries) {
    // Use a gray background for every other blog entry
    if (i % 2 == 0)
      blogText += "<p style='background-color:#EEEEEE'>";
    else
      blogText += "<p>";

    // Generate the formatted blog HTML code
    blogText += "<strong>" + (blog[i].date.getMonth() + 1) + "/" +
      blog[i].date.getDate() + "/" +
      blog[i].date.getFullYear() + "</strong><br />" +
      blog[i].body + "</p>";

    i++;
  }

  // Set the blog HTML code on the page
  document.getElementById("blog").innerHTML = blogText;
}

function searchBlog() {
  var searchText = document.getElementById("searchtext").value;
  for (var i = 0; i < blog.length; i++) {
    // See if the blog entry contains the search text
    if (blog[i].body.toLowerCase().indexOf(searchText.toLowerCase()) != -1) {
      alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
        blog[i].date.getFullYear() + "] " + blog[i].body);
      break;
    }
  }

  // If the search text wasn't found, display a message
  if (i == blog.length)
    alert("Sorry, there are no blog entries containing the search text.");
}

function randomBlog() {
  // Pick a random number between 0 and blog.length - 1
  var i = Math.floor(Math.random() * blog.length);
  alert("[" + (blog[i].date.getMonth() + 1) + "/" + blog[i].date.getDate() + "/" +
    blog[i].date.getFullYear() + "] " + blog[i].body);
}
```

转换博客日志为具有格式的 HTML，命名为 Blog.toHTML()。

Blog.toHTML()

转换博客日志为具有格式的HTML代码，对其他只想干净地呈现（附HTML格式的）日志的程序代码而言，它能大幅减轻负担。

Blog.containsText()

代码不多，但值得独立为一个方法，因为博客日志应该能够搜索自己的内容主体。

Blog.toString()

转换博客日志为字符串。当日期需要放在方括号里，后随博客日志正文时，就需要它上场处理。

没有蠢问题

问：该如何知道代码应放入方法呢？

答：这个嘛……首先自我回想一下，何为方法的理想目的？答案就是根据对象的状态（数据）而采取某些行动。到某种程度，理解对象里的方法，也包括理解对象的实际功用，或是需求。然后专注于强化对象对自己施展行动的能力。以Blog对象为例，它拥有自我转换为字符串或附格式HTML代码的能力就很合理，因为这两种行为需要访问对象内部的数据。相似地，在博客日志里搜索文本，也应该是Blog对象的内部行为，因此很适合制成方法。

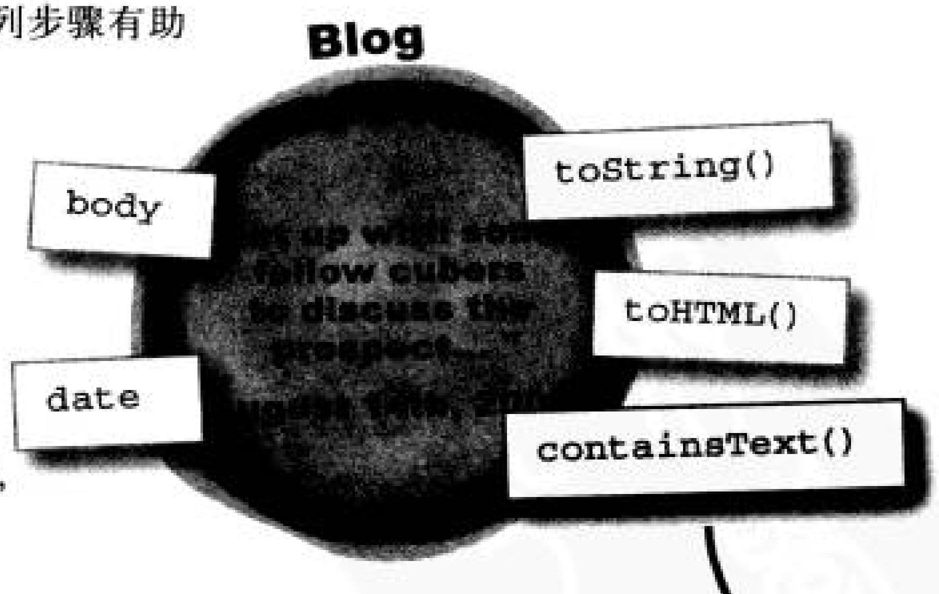
问：那有 Blog 对象不该采取的行动吗？

答：只要超过Blog对象的作用域都算，或许像呈现或搜索博客日志列表，因为Blog对象只负责呈现单一日志。这也是blog数组为何包含多个Blog对象，而每个Blog对象不需把自己视为Blog对象大集合的一部分。各个Blog对象只应该照顾自己的事务，包括根据自己的日期和正文而行动。

把函数转变为方法

既然已经隔离出 YouCube 里某些适合作为 Blog 对象的方法的代码，我们就来仔细观察把它们转换为 Blog 对象方法的过程。示范方法是 containsText()，它负责在博客日志正文里搜索子字符串。把搜索用的程序代码移入方法，主要关系到直接操作Blog对象的body特性，可与仍为searchBlog()函数时操作局部变量相对照。下列步骤有助于厘清过程：

1. 声明方法。如果必要，则完成自变量列表，例如containsText()所需的搜索目标文本。
2. 把现有的程序代码移入新方法里。
3. 修改必要的程序代码，改为使用对象特性，例如containsText()的this.body。



磨笔上阵

请设计Blog对象的containsText()方法，借由指派函数字面量给this.containText，于Blog的构造函数里创建这个方法。

....

....

....

请设计Blog对象的containsText()方法，借由指派函数实字给this.containText，于Blog的构造函数里创建这个方法。

借由指派函数字面量给方法引用而创建方法。

```
this.containsText = function(text) {
  return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

然后使用关键字this创建方法，与创建特性的用法类似。

在方法中的代码直接使用关键字this访问对象特性。

崭新的blog对象公开亮相

另外两个新的方法也加入containsText()的行列，组成新版Blog对象，它现在拥有特性，也拥有行为了。

嘿！我的雕塑计划很成功耶！

```
function Blog(body, date) {
  // Assign the properties
  this.body = body;
  this.date = date;

  // Return a string representation of the blog entry
  this.toString = function() {
    return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
      this.date.getFullYear() + "] " + this.body;
  };

  // Return a formatted HTML representation of the blog entry
  this.toHTML = function(highlight) {
    // Use a gray background as a highlight, if specified
    var blogHTML = "";
    blogHTML += highlight ? "<p style='background-color:#EEEEEE'>" : "<p>";

    // Generate the formatted blog HTML code
    blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" +
      this.date.getDate() + "/" + this.date.getFullYear() + "</strong><br />" +
      this.body + "</p>";
    return blogHTML;
  };

  // See if the blog body contains a string of text
  this.containsText = function(text) {
    return ((this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
  };
}
```

创建并初始化特性。

toString()方法返回字符串格式的博客日志。

toHTML()方法返回精美的HTML格式的博客日志。

ContainsText()方法在日志正文包含搜索目标字符串时，返回true，未找到则返回false。

对象为 YouCube 做了什么？

直到Blog对象的新版本（可至*http://www.headfirstlabs.com/books/hfjs/*下载）插入 YouCube 脚本，面向对象程序设计的好处才显现出来。现在有好几项博客专用的任务都委派给 Blog 的方法，脚本因而简单许多。

崭新的Blog对象简化了YouCube脚本。

```
// Show the list of blog entries
function showBlog(numEntries) {
  // First sort the blog in reverse chronological order (most recent first)
  blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });

  // Adjust the number of entries to show the full blog, if necessary
  if (!numEntries)
    numEntries = blog.length;

  // Show the blog entries
  var i = 0, blogListHTML = "";
  while (i < blog.length && i < numEntries) {
    blogListHTML += blog[i].toHTML(i % 2 == 0);
    i++;
  }

  // Set the blog HTML code on the page
  document.getElementById("blog").innerHTML = blogListHTML;
}

// Search the list of blog entries for a piece of text
function searchBlog() {
  var searchText = document.getElementById("searchtext").value;
  for (var i = 0; i < blog.length; i++) {
    // See if the blog entry contains the search text
    if (blog[i].containsText(searchText)) {
      alert(blog[i]);
      break;
    }
  }

  // If the search text wasn't found, display a message
  if (i == blog.length)
    alert("Sorry, there are no blog entries containing the search text.");
}

// Display a randomly chosen blog entry
function randomBlog() {
  // Pick a random number between 0 and blog.length - 1
  var i = Math.floor(Math.random() * blog.length);
  alert(blog[i]);
}
```

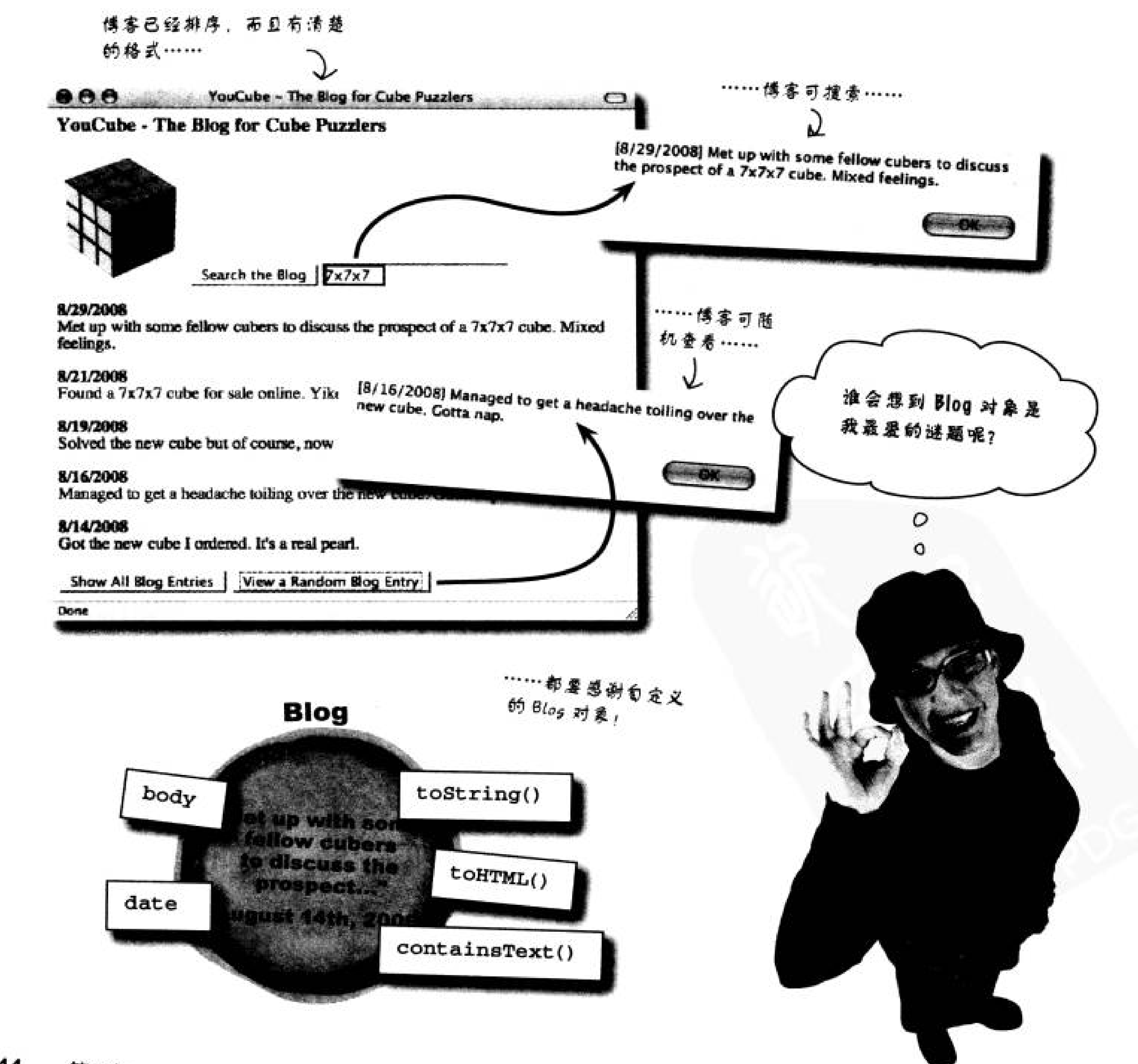
toHTML() 方法全面负责 HTML 格式加到博客日志上的事宜。

containsText() 方法于博客日志中搜索子字符串。

toString() 方法比较精密，它于期待博客日志是字符串格式时自动受到调用。

YouCube 3.0

为了这个项目，Ruby真是投入不少精力，她终于认为YouCube 3.0已经够好，她可以暂停开发，回到心爱的魔方上。她也期待着解谜大会，准备好好精心打扮一番……



JavaScript 填字游戏

Ruby 已经为这一刻等了一整章……谜题现身！不过，这可不是魔方，而是填字游戏。这个嘛……鱼与熊掌不能兼得啊！

横向提示：

1. 使用 String 的________方法搜索文本字符串。
4. 当你把函数放入对象，它变成________。
6. JavaScript 数组和字符串其实是________。
8. 使用________访问对象里的成员。
11. 近乎随机。
13. 这个方法转换对象为文本字符串。
14. Ruby 的家乡。

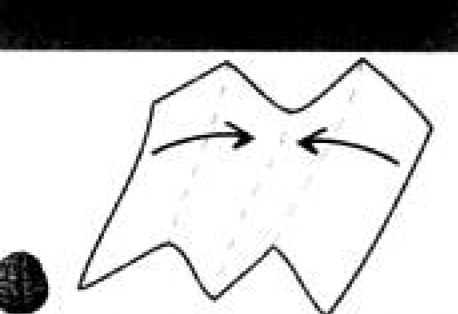
纵向提示：

2. 使用这个对象处理时间。
3. 一个无条件舍去尾数的 Math 方法。
5. 在对象中，特性存储________。
7. 对象特性在此创建。
9. 对象里的一块数据。
10. 方法让对象能接受________。
12. 调用这个方法，以改变数组元素的排序。

JavaScript 填字游戏解答

1 INDEXOF
2 DATE
3 FLOOR
4 METHOD
5 DATA
6 OBJECTS
7 CONSTRUCTOR
8 DOTOPERATOR
9 PROPERTY
10 ACTIONS
11 PSEUDORANDOM
12 SORT
13 TOSTRING
14 CONUNDRUM

Page Bender



请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

JavaScript希望对象对数据做什么？

这是左右脑的秘密会谈。

You're invited to...
A Puzzle Party!
Date: October 24th
Location: 2112 Confoun
Baffleburg, CA
USA
String
Date
Math
Array
Puzzler Ruby
5280 Unravel Avenue
Conundrum, AZ 85399

欢迎各位尽情搜索，但在搜索和分析数据上，大概很难找到比JavaScript对象更好的工具。它们甚至连随机排列数字也不费吹灰之力。

10 创建自定义对象

自定义对象让你为所欲为

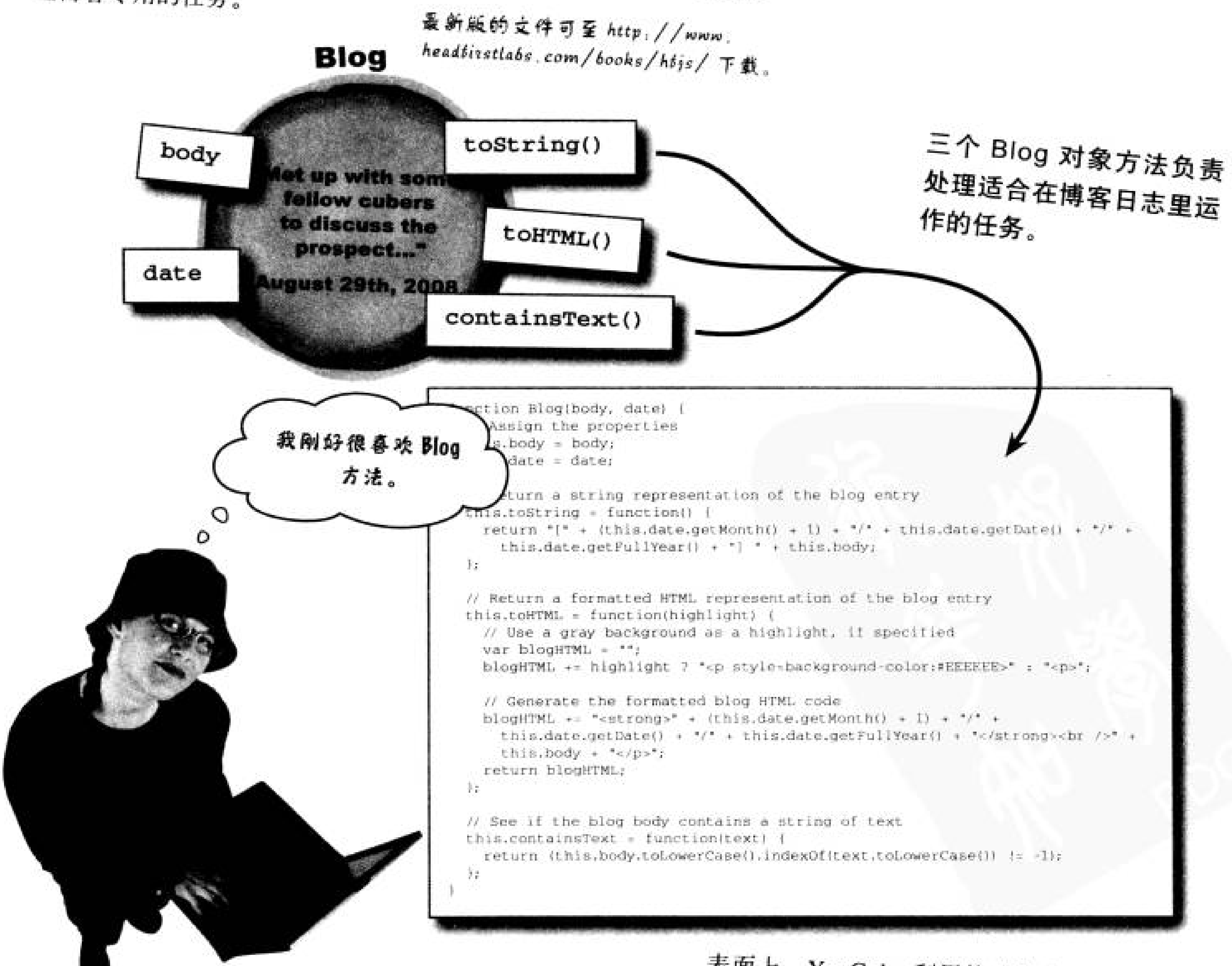
如果有这么简单，当然谁都想做。JavaScript 没有退货保证，但你绝对可以让它照你的意思行动。自定义对象，就像 JavaScript 版的极热去奶泡不加鲜奶油要三份低咖啡因浓缩咖啡的大杯摩卡玛奇朵……好一杯特调咖啡啊！有了自定义 JavaScript 对象，你也能煮出任你操纵又利用了特性和方法优势的代码。最后甚至能制造出可重复使用的面向对象程序代码，既扩展了 JavaScript 语言的效率……而且又只为你服务！

重新查看YouCube的Blog方法

上次告别 Ruby 时，她正为了博客的建立而相当兴奋，这是个由对象驱动、记录她对魔方热爱的博客。虽然 Ruby创建的`Blog`对象还不赖（用于驱动 YouCube 的对象），但她却浑然不知自己错过了运用面向对象规则的关键机会。更重要的是：她尚未全面探索 `Blog` 对象的制作方式，它可以更有效率、更有组织，未来也因此而更容易维护。

Ruby 对 `Blog` 对象的最后一次调整，包括创建了三个方法，以处理一些博客专用的任务。

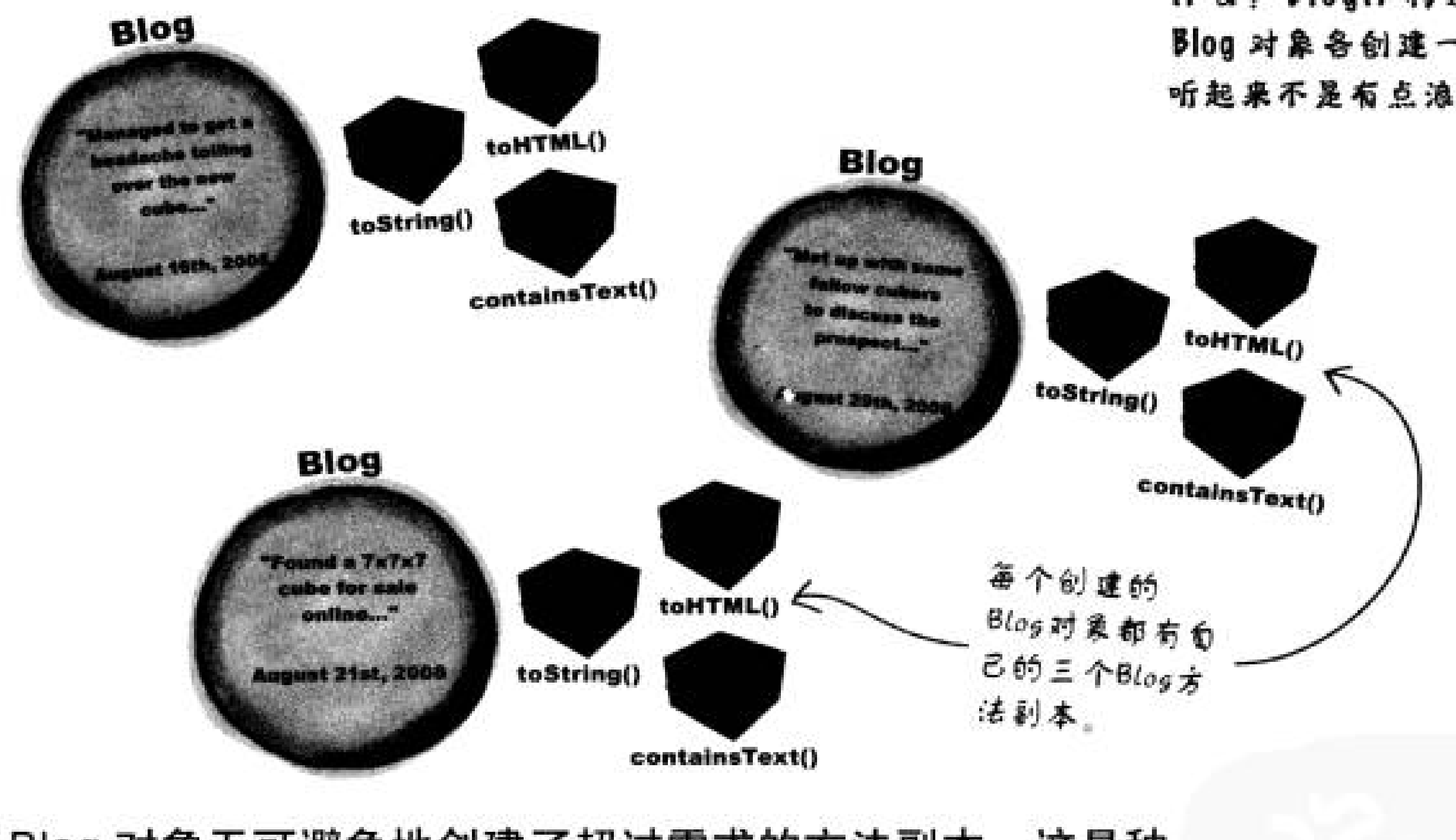
YouCube 博客可以运作，但还不是面向对象程序设计的标竿。



表面上，YouCube 利用的对象方法看来还不错，但有个小问题……

方法过载

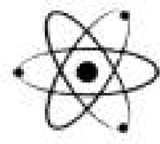
与博客的对象特性相似，Blog对象的方法也是在构造函数里利用关键字 this 而创建。这种方式确实可行，但据此创建的每个 Blog 对象，均各有一份对象方法的副本。如果博客包含6个实体，Blog 的三个对象方法将各有6个副本。



什么？Blog() 构造函数为每个 Blog 对象各创建一份方法副本？听起来不是有点浪费吗？

Blog 对象无可避免地创建了超过需求的方法副本，这是种非常浪费又没效率的行为。

没错，Blog 对象每次都新创建三个方法，每个 Blog 对象都有自己的三个方法副本。不像特性需要为每个对象存储独特数据，方法应该为对象所**共享**。如果所有 Blog 对象共享一份方法副本，将是更好的设计。如此一来，逐渐加入更多博客日志（对象）后，可防止脚本因不必要的方法而膨胀。

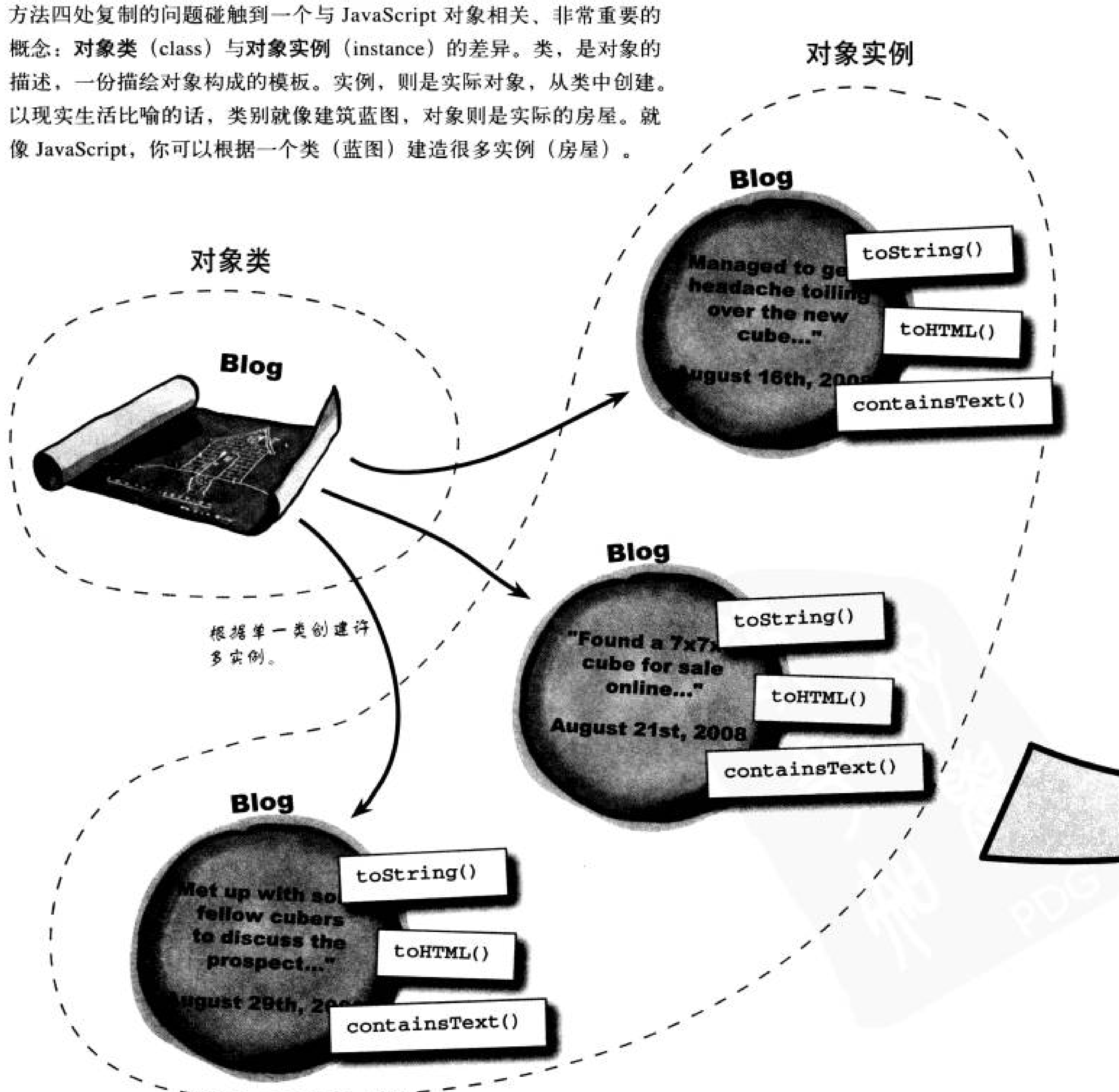


动动脑

如何重新设计 Blog 对象，让方法代码不会一再复制到新对象里呢？

类与实例

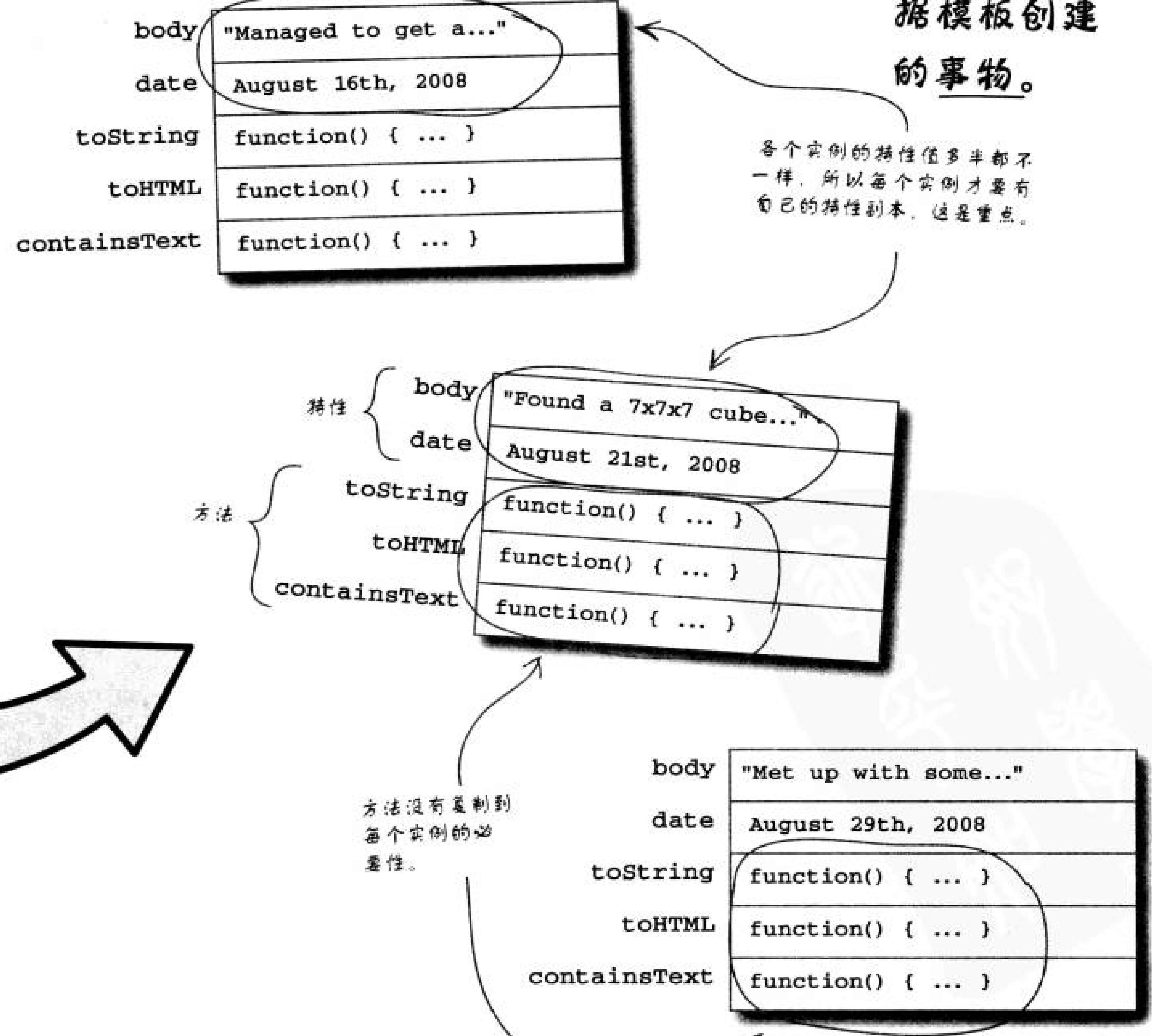
方法四处复制的问题碰触到一个与 JavaScript 对象相关、非常重要的概念：**对象类**（class）与**对象实例**（instance）的差异。类，是对象的描述，一份描绘对象构成的模板。实例，则是实际对象，从类中创建。以现实生活比喻的话，类别就像建筑蓝图，对象则是实际的房屋。就像 JavaScript，你可以根据一个类（蓝图）建造很多实例（房屋）。



根据类创建实例

类描述对象的特性与方法，实例则把真正的数据放入特性，并为数据注入生命。每个实体都有自己的特性副本，因此实例才能与其他实例有所不同。

对象类是种<u>模板</u>，对象实例则是根据模板创建的<u>事物</u>。



以 this 访问实例的特性

关键字 this 用于设置实例拥有的特性和方法。

到目前为止，我们处理过的所有特性都是**实例特性**（instance property），意指它们为实例所拥有；更重要的是，每个实例都有自己的特性副本。你可以轻易分辨出实例特性，因为它于构造函数中使用关键字this设定。

```
function Blog(body, date) {
  this.body = body;
  this.date = date ;
  ...
}
```

这些都是实例特性，因为它们都使用关键字 this 被引用。

另外还有实例方法，但它们比较狡猾一点，因为它们可被实例**或**类所拥有。我们只创建过实例方法 —— 使用 this 关键字设置 —— 这表示它们**被各个实例所拥有**。如此解释了方法会被复制到每个实例的原因。

```
function Blog(body, date) {
  ...
  this.toString = function() {
    ...
  }
  this.toHTML = function() {
    ...
  }
  this.containsText = function() {
    ...
  }
}
```

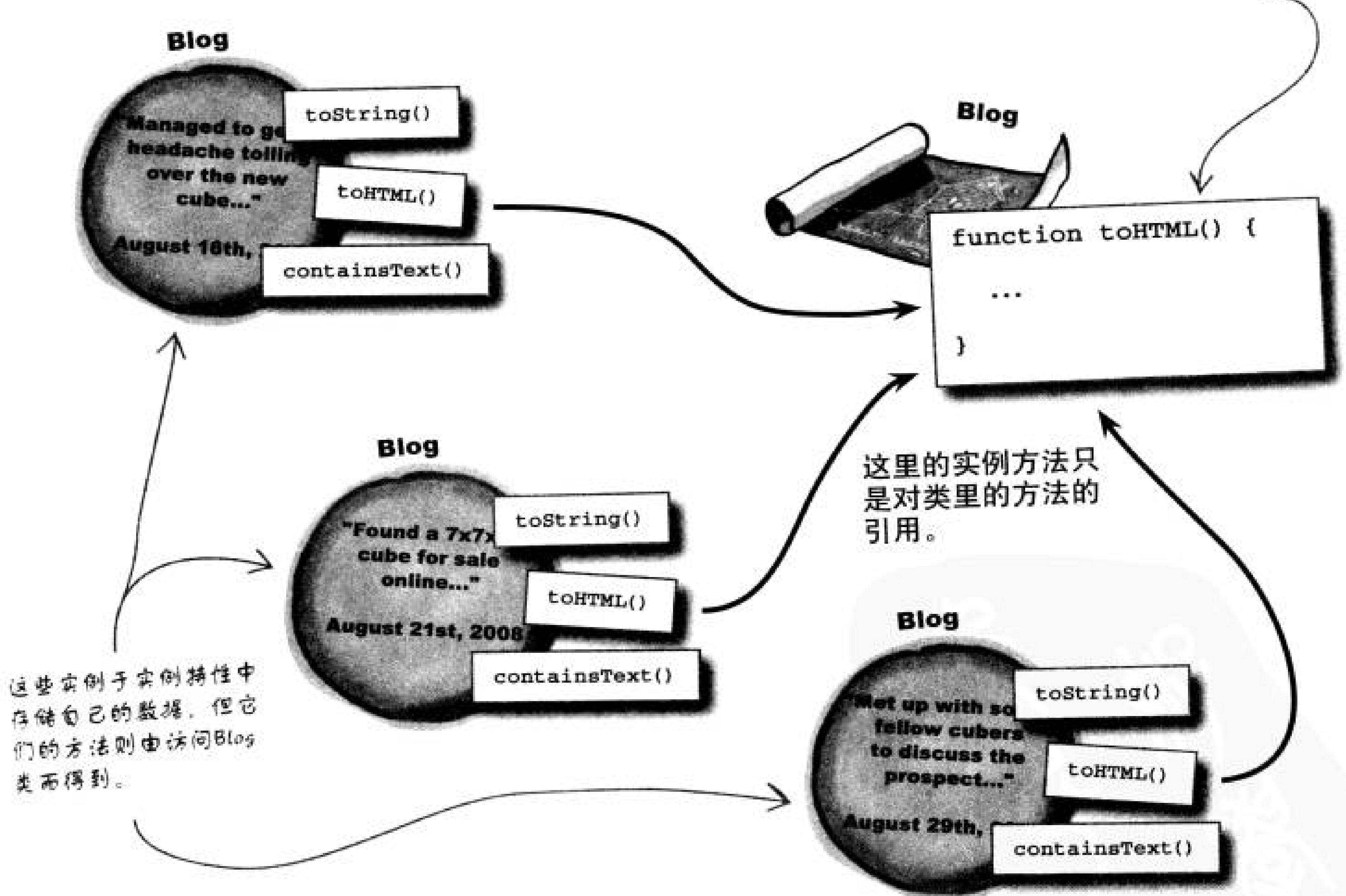
每个 Blog 的实例都有自己的方法副本。

这些是实例方法，因为在构造函数中使用了关键字 this 设置方法。

幸好，自定义对象并非都是把方法复制到每个新实例的浪费设计。解决方案在于另一种创建方法的方式，可让实例共享同一份方法代码。

拥有一次，运行多次：类拥有的方法

有种实例方法直接为类所有，这表示该方法只有一份，由**所有**实例共享。**类拥有的**实例方法（class-owned instance method），比起在每个实例里各存储一份方法，实在有效率多了。



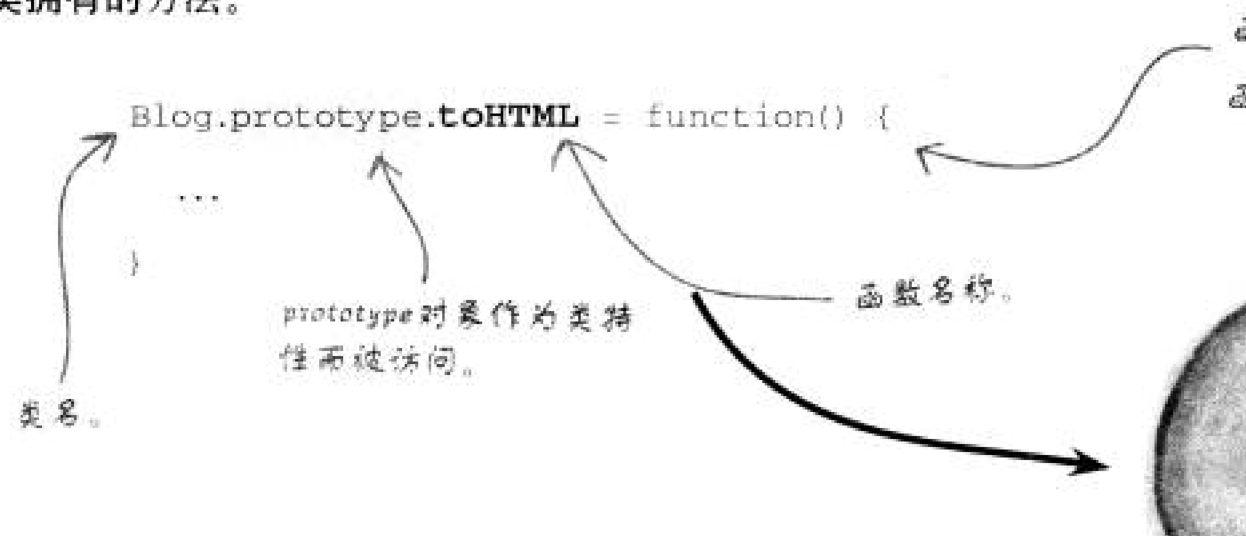
当方法为类所拥有时，所有该类的实例都可访问方法，因此不需要另外复制一份。这样比较有效率，尤其在程序创建许多对象实例时，你想想方法的众多副本将吃掉多少空间？以 YouCube 为例，三个方法（`toString()`、`toHTML()`、`containsText()`）将被不必要地复制到创建的每个博客日志中。

当然，我们需要把方法的拥有权指派给类，而不是指派给单一实例……

方法存储在类里，让所有实例共享一份方法代码。

在类层，使用 prototype

多亏了每个对象都有个隐藏对象prototype（它以特性的形式存在），JavaScript中的类才能成真。prototype对象用于设定隶属于**类层**（class level）的特性与方法，而非属于实例的。以方法为例的，使用 prototype对象，即可创建**类拥有的**方法。



Blog

prototype

toHTML()

以上例而言，toHTML()方法直接附加至 Blog 类本身，而非隶属于某个类实例。无论 Blog 类创建多少实例，都只会有一份 toHTML() 方法。

Blog

toHTML()

既然toHTML()方法成为 Blog 类的一部分，当它接受调用时，将在类内运行。然而，就技术上而言，这个方法仍是实例的方法，因为它能透过实例对象被调用，也可访问实例里的特性。

```
var blogEntry1 = new Blog("Not much going on.", ...);
blogEntry1.toHTML();
```

调用 toHTML() 方法，导致代码在类里运行。

倘若创建了其他 Blog 类实例，实例也调用了 toHTML() 方法，仍会运行类内的同一个方法。这就是把方法存储在类里的美感——**存储一次，运行多次！**

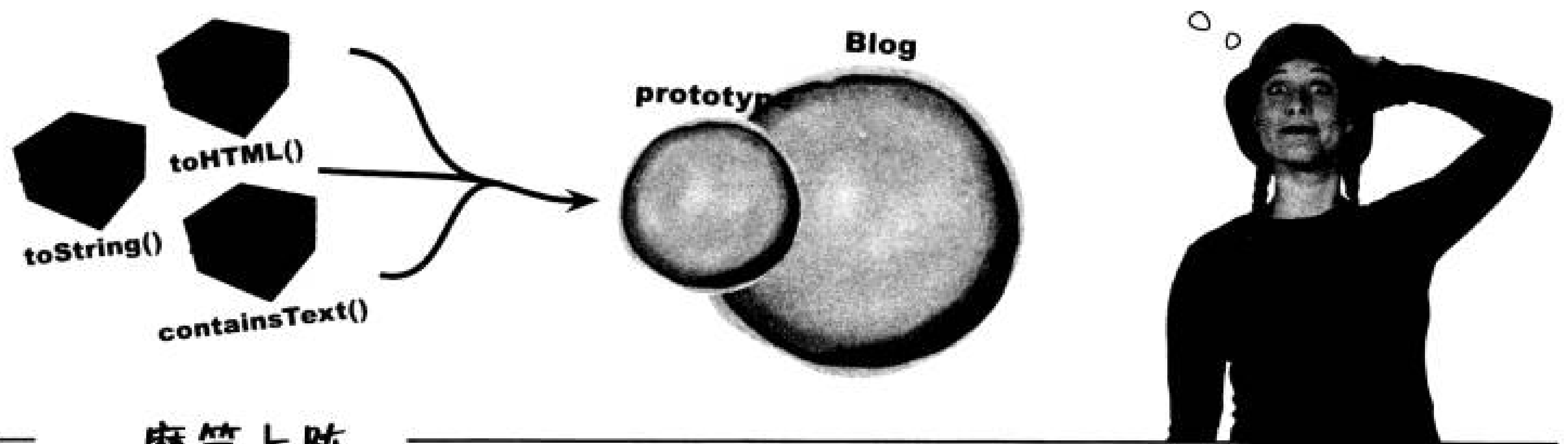
```
var blogEntry2 = new Blog("Still just hanging around.", ...);
blogEntry2.toHTML();
```

其他实例仍享有存储在类里的同一个方法。

类、prototype与YouCube

Ruby 暂时无法消化这番关于类与原型（prototype）的讨论，但她清楚知道，若重新思考 Blog对象加上 prototype对象的存储方式，YouCube 将因此受益。

哇，使用 prototype 对象存储方法，可让 YouCube代码更有效率！



磨笔上阵

Blog代码现在改用prototype对象存储方法，让方法为类所拥有。请圈出相关代码，并解释各段代码的运作。

```
function Blog(body, date) {
  // Assign the properties
  this.body = body;
  this.date = date;
}

// Return a string representation of the blog entry
Blog.prototype.toString = function() {
  return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "] " + this.body;
};

// Return a formatted HTML representation of the blog entry
Blog.prototype.toHTML = function(highlight) {
  // Use a gray background as a highlight, if specified
  var blogHTML = "";
  blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";

  // Generate the formatted blog HTML code
  blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "</strong><br />" + this.body + "</p>";
  return blogHTML;
};

// See if the blog body contains a string of text
Blog.prototype.containsText = function(text) {
  return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

磨笔上阵 解答

Blog代码现在改用prototype对象存储方法，让方法为类所拥有。请圈出相关代码，并解释各段代码的运作。

```
function Blog(body, date) {
  // Assign the properties
  this.body = body;
  this.date = date;
}

// Return a string representation of the blog entry
Blog.prototype.toString = function() {
  return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "] " + this.body;
};

// Return a formatted HTML representation of the blog entry
Blog.prototype.toHTML = function(highlight) {
  // Use a gray background as a highlight, if specified
  var blogHTML = "";
  blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";

  // Generate the formatted blog HTML code
  blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "</strong><br />" + this.body + "</p>";
  return blogHTML;
};

// See if the blog body contains a string of text
Blog.prototype.containsText = function(text) {
  return (this.body.toLowerCase().indexOf(text.toLowerCase()) != -1);
};
```

构造函数现在只需专注于特性的创建和初始化。

因为方法并未被指派给特定Blog实例，指派行动即发生在构造函数外。

每个方法均于prototype对象里设置，而非于Blog()构造函数里使用关键字this。

更有效率的YouCube

YouCube博客已使用**类拥有的方法**（class-owned method）来削减多余的代码，都要感谢对象原型（object prototyping）的协助。无论创建了多少个Blog对象，方法都只有一份，因为它们改隶于类。最酷的是，从YouCube脚本的角度使用Blog对象时，一切都未改变。

复习要点

- 类是对象的**描述**，实例则是**实际的**对象，根据对象描述所创建。
- 类勾画了对象的特性与方法，实例则把真正的数据放在特性里以供方法利用。
- 关键字this用于从实例**本身的程序代码**内访问实例。
- prototype对象让方法可存储于**类**内，以免实例不必要地一直复制程序代码。

问：我还是不太了解类与实例的大观念耶？到底哪里重要？

答：关于类的概念，是让“对象的创建和重复利用”更为容易。你可以整天都拿来创建只使用单次的对象字面量，一切都没有问题，只不过你真的不必浪费这么多精力，因为你只是在复制自己的成果，浪费不必要的时间。有点像建筑师每次建造同一型房屋，都坚持重新绘制一模一样的蓝图。

何不创建能用于尽情创建实例的模板，以节省更多精力呢？这就是类的用途——创建一个类，而后于需要时使用它，尽情创建实例。

问：所以说，类让创建相似的对象更简单。但this和prototype又怎么会搅和进来呢？

答：关键字this从实例内的方法访问实例，它的主要用途在于访问实例特性。假如你想访问某个方法里的x特性，即用this.x。如果你只讲x，代码无法得知你其实想访问实例里的特性，代码可能以为x是个变量。所以在构造函数里创建和初始化特性时，需要使用this。

prototype的故事则完全不一样。它提供创建类的机制。但不像C++或Java等其他程序语言，JavaScript并未真正支持类作为语言的具体构成要素。JavaScript使用prototype以模拟类。结果很相似，但JavaScript还需要我们操纵prototype对象（出现为每个JavaScript对象里的隐藏对象）以创建“类”。透过把特性或方法存储在prototype对象里，即可有效地将其当成类的一部分而予以访问，不再只是实例的一部分。

问：构造函数在类方程里的地位是什么呢？

答：构造函数是创建JavaScript类时非常重要的一部分，因为它们负责创建对象实例。你可以把构造函数和原型分别当成JavaScript类拼图的两块重要拼图。构造函数负责设定实例的一切事项，原型则处理类层的一切事项。两者同心协力赋予某种资格，让我们有能力达成一些很酷的事，因为把某些成员放在实例层，有些则放在类层，有其强迫性的理由。本章后续仍会继续探讨这个主题。

问：我对对象的命名惯例还是有点疑惑。有时候对象会用首字母大写式，有时候又采用小写驼峰式。我错过了什么规则吗？

答：对象命名的唯一原则，只有首字母大写，但实例的命名则采用小写驼峰式。因为实例其实是个变量，变量的命名惯例是小写驼峰式。你会觉得不连贯的原因，多半肇因于我们使用“对象”时的定义比较松散。为了准确起见，像Blog类应该首字母大写，至于blogEntry或blog[0]这类实例应该采用小写驼峰式。

如果回想过去使用的其他标准对象，这项命名惯例也就合理了。你可以存储目前的日期/时刻于名为now的变量（实例）里，now则由Date对象（类）所创建。

为博客加上签名

Ruby 正在为 YouCube 发掘面向对象程序设计（OOP）带来的效率与组织改良，但她的兴趣远不只在背后改善程序代码而已……她想加入新功能。

我真的很想加上签名，让签名出现在每一则博客日志的下方……就在这里！

YouCube - The Blog for Cube Puzzlers

Cube - The Blog for Cube Puzzlers

Search the Blog

8/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.

8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.

8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.

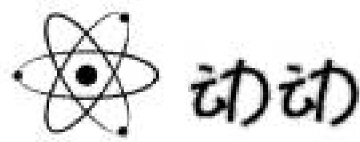
8/16/2008
Managed to get a headache toiling over the new cube. Gotta nap.

8/14/2008
Got the new cube I ordered. It's a real pearl.

Show All Blog Entries | View a Random Blog Entry

Done

Ruby 想到，为 `Blog`类加上 `signature` 特性或可奏效。然后，她只要在构造函数里设置特性，再于每则日志中列出签名……问题解决完毕……



动动脑

Ruby 应该把签名创建为实例特性吗？你可以想到任何反对这种方式的好理由吗？

没有蠢问题

问：我一直看到“面向对象”这个词，它是什么意思？

答：程序设计圈里非常常用“面向对象”一词（甚至有点滥用），它对不同的人可能代表不同意思。一般而言，面向对象程序设计（object-oriented programming，OOP）涉及以对象建立软件，就像你在 Youcube 的 date 特性里用到 Date 对象。

大多数程序设计师把“在程序中广泛使用对象”和 OOP 联想在一起。至少在理论上，真正的面向对象程序可分解成许多组彼此交互的对象集合。有些面向对象纯化主义者会说，JavaScript 并非够格的 OOP 语言。但你可以把精力放在别的地方，不参与这类口舌之争。争论的双方各执有根有据的论点，最后也不会出现真正的赢家。

或许一个签名就够了。

在知道每个博客日志实例的签名都一样后，自然不需要让每个实例都有自己的签名特性。Ruby 此时需要**类特性**（class property），这种特性只在类里存储一次，而不是复制许多副本到各个实例里。

类特性也能共享

类特性（class property）与**类拥有的实例方法**很相似，它们都隶属于类，只有**单一一份**可供所有实例访问。就某些方面而言，类特性对数据更为明显，因为这表示特性只有一个值，且为所有实例共享。Ruby 的signature特性就是要寻找这种功能，因为整个 YouCube 博客其实只有一个签名。

每个实例存储在自己的实例特性里。

Blog

"Managed to get a headache toiling over the new cube..."

August 16th, 2008

如果改变了 Blog.signature 特性，它会改变所有实例里的这个特性。

Blog

"Found a 7x7x7 cube for sale online..."

August 21st, 2008

Blog

signature = "Puzzler Ruby"

Blog

"Met up with some fellow cubers to discuss the prospect..."

August 29th, 2008

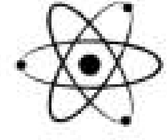
如果你改变这则日志的 body 特性，它不会影响其他日志。

类特性 signature 只存储于类里。

如果你改变这则日志的 date 特性，改变只会影响这篇日志。

类特性在类里存储一次，但能被所有实例访问。

虽然signature特性存储在Blog类里，它却准备好被任何实例访问，只要实例想访问博客作者的签名。

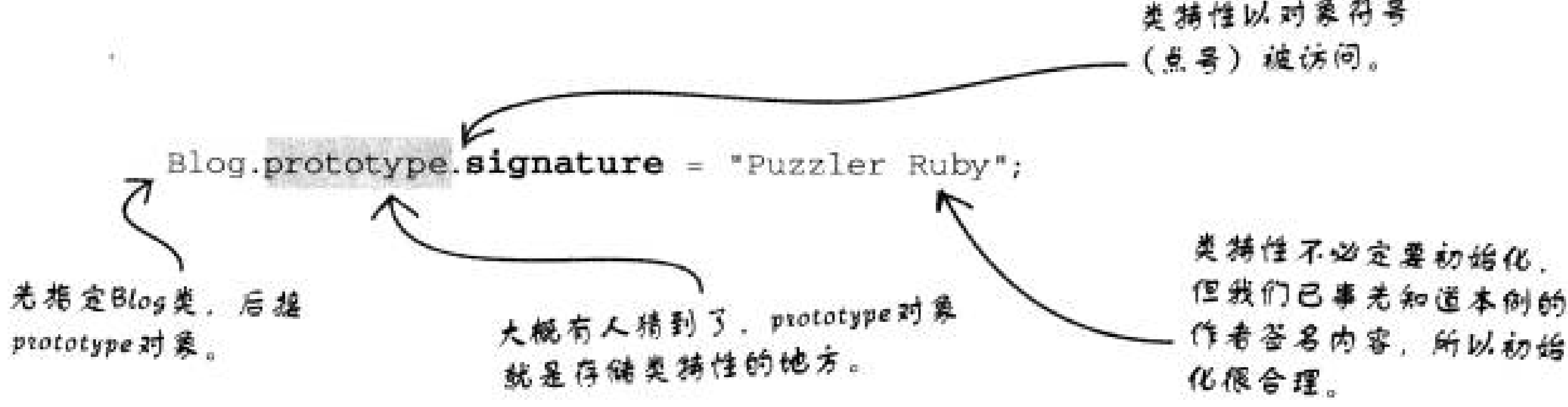


动动脑

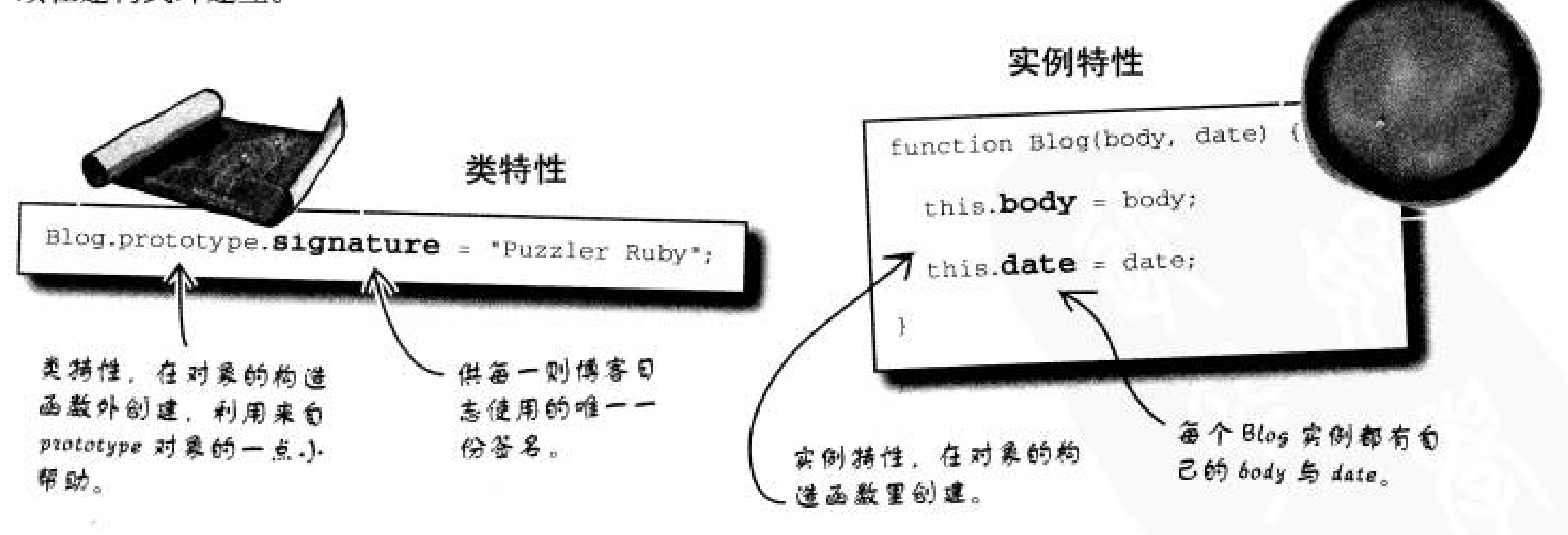
你认为创建类特性有什么用途？

使用prototype创建类对象

讲了这么多类特性的存储地点以及知道后带来的广泛冲击，创建影响这么大的特性却意外地世俗。事实上，只要一行代码就够了：



关于上述范例代码最有趣的一点，在于你无法一眼了解它的意义——代码并未出现在构造函数内，不像创建实例对象的状况。构造函数用于为实例赋予生命，因而无法建立类别特性。不过，类别特性必须在建构式外建立。



磨笔上阵

请写出用alert框呈现 signature 特性的代码。提示：假设代码位于某个Blog方法内。

....

没有蠢问题

问：为什么会需要把 YouCube 的签名存储在特性里？它不能当作日志的部分正文吗？

答：当然有可能把签名放在日志的正文内容里，但会浪费不必要的时间与精神。假设只有一个人在维护博客，当外面有这么干净的签名处理方式，Ruby 终究会对孜孜矻矻地为每则日志签名感到厌烦。而且，谁说她不会失手打错字，变成"Puzzled Ruby"？

另一个较可行的选择方案，则是在为日志加上 HTML 格式时，使用字符串字面量制作签名。这项方式的运作也不错，不过它把一段重要的数据——签名藏在博客的格式化代码中，不容易寻找也不容易维护。借由把签名放在类特性里，签名较容易访问，因而也较容易辨识及修改。

问：如果签名做成实例特性，对创建一则博客日志将带来什么改变？

答：还记得吗？对象的每个实例都拥有自己一组在构造函数中做了初始化的实例特性？如果签名改为实例特性，Blog()构造函数需在每个实例里设置这个特性。因为构造函数能设定特性为签名字符串，这样不至于造成太严重的编码麻烦。但是，在幕后将有很多个签名副本，与实例的数量一样多，真是浪费。不仅如此，签名实例特性还可能自己改变，完全影响不到其他签名特性。

问：所以说，如果我想修正 YouCube，让它支持多名博主时，我就可以把签名改为实例特性啰？

答：是的，而且也是个很好的想法，因为在多名博主共笔的情境中，signature特性有可能需要每个实例各存储不同的值。最佳处理方式大概是增加一个引数至 Blog()构造函数，允许签名字符串的传入，然后使用字符串做 signature 实例特性的初始化。换句话说，就是把 signature 特性跟其他 Blog 实例特性一起处理。

问：类特性似乎有点像全局变量。它们如何分辨呢？

答：类特性确实很像全局变量，两者均可于几乎任何地方被访问。类特性的创建也与全局变量很像，至少创建的位置很像——位于其他程序代码外的脚本层。类特性与全局变量的差异，在于它们与类的关联，因而也推及与实例对象的关联。也就是说，类特性的访问一定与实例有关。

问：暂停一下。类特性"必须"透过实例访问吗？

答：虽然类特性使用prototype对象创建，并借此存储在类中，但它们还是必须透过实例被访问。所以，类特性的访问方式就像实例特性，也要使用关键字this 和对象符号（点号）。差异其实在于，特性的存储地点在类中（类特性）或在某个实例里（实例特性）。

签名且传递完毕

在类特性signature的创建、初始化并准备派上用场后，Ruby 正跃跃欲试，想让它活动一下。我们回头看看为博客日志加上格式，以便在浏览器上呈现的代码，toHTML()方法就是签名加入每则日志外观呈现的地方。

toHTML() 方法已把签名作为博客日志的一部分。

```
Blog.prototype.toHTML = function(highlight) {
  // Use a gray background as a highlight, if specified
  var blogHTML = "";
  blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";

  // Generate the formatted blog HTML code
  blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
    this.date.getFullYear() + "</strong><br />" + this.body + "<br /><em>" + this.signature +
    "</em></p>";
  return blogHTML;
};
```

类特性 signature 的引用方式与一般实例特性一样。

Ruby 的签名出现在每则博客日志中。

关于谁是每篇日志的作者，以后再也没有疑虑了。

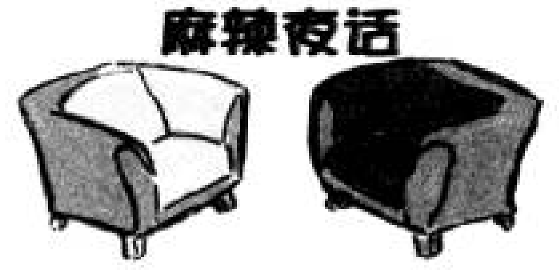
9/1/2008
Went ahead and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.
by Puzzler Ruby

8/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mi
feelings.
by Puzzler Ruby

8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast
by Puzzler Ruby

8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.
by Puzzler Ruby

Ruby 使用 OOP 的技巧——在Blog类里加入类特性signature，进一步扩展了 JavaScript 语言。对她更重要的是，她为 YouCube 博客加入更多人性化的气息。



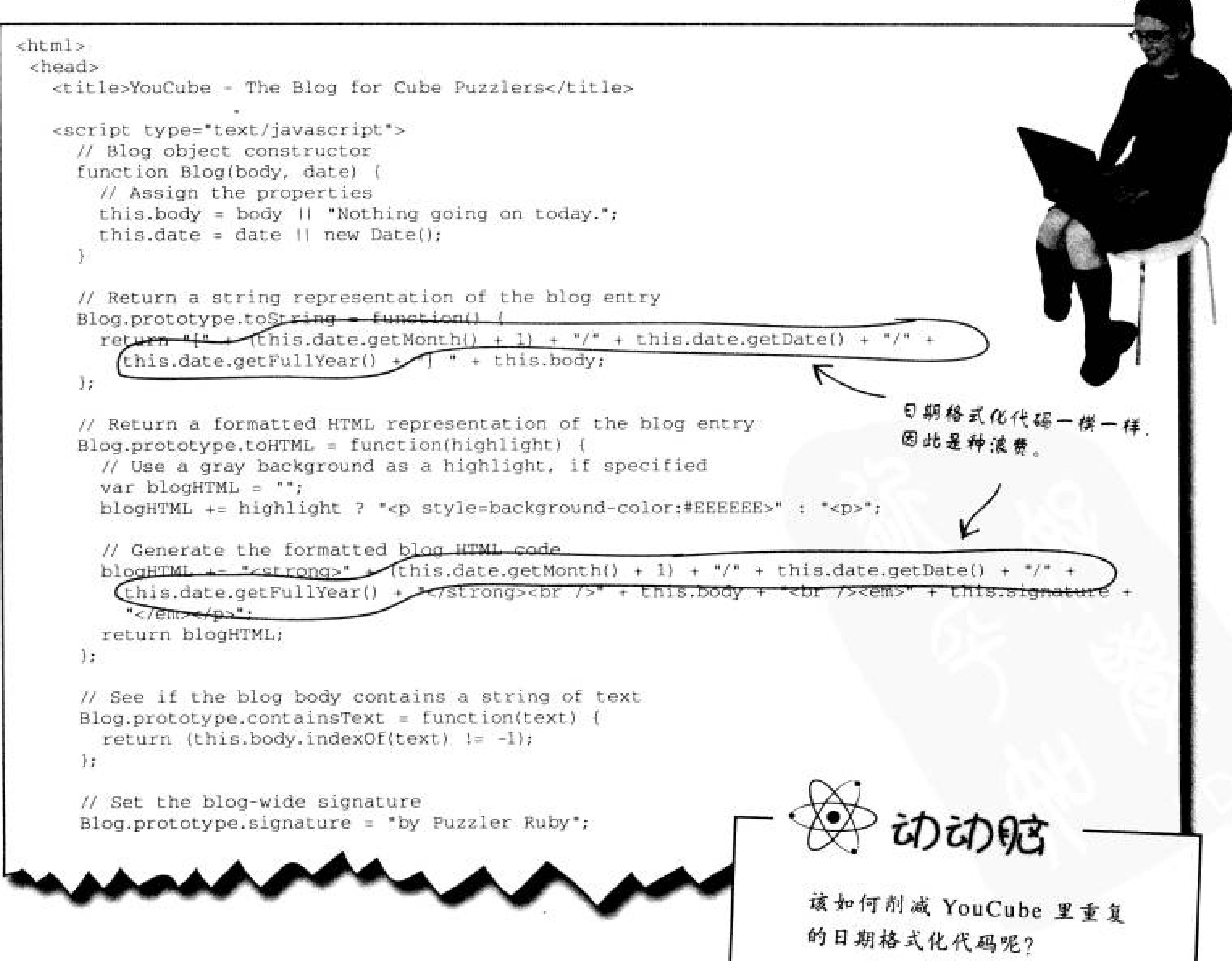
今晚主题：实例特性与类特性讨论数据的拥有权与秘密社团

实例特性：	类特性：
原来你就是我最近一直听说的那个人。老实说，我不知道你为什么要出现在这里。我已经非常擅长让对象实例独一无二，并保持追踪它们各自的特性值了。	
	我完全相信，而且你的敬业精神也很值得钦佩。不过，你知道实例有时想省下保存自己数据的麻烦吗？
我不相信……你继续讲……	
	有时候，某块数据对所有实例而言都一样，有点像是秘密社团的秘密握手信号。每个社团成员都知道秘密握手，但只有社团内才知道这件事。如果有谁发明了自己的握手方式，将使一切信号混乱。其他成员也必须自行发明新的握手信号，以凌驾原本的方式，很快地，没有人记得原本的秘密握手，社团里出现太多秘密了。
你的意思是，我不是个存储秘密握手的最佳方式吗？	
	没错。但我可没有冒犯的意思喔，只是在这个例子里，虽然每个秘密社团成员都需要知道握手方式，但他们全体只需要一种方式。
我懂了。那密码呢？我可以存储密码吗？	
	或许可以。如果每个人都有自己的、私人的密码，答案就是“可以”，此时你将非常适合存储密码。
太棒了！我们赶快开始分工合作吧！我马上开办一个秘密社团，然后你和我各设定一组密码。	
	可是你又不知道共通的秘密握手方式……这部分交给我吧！
太好了！到底是什么秘密握手方式？我好想知道……	

复制代码绝对是坏预兆

Ruby 又开始行动了，从不满足于现有的成就，她决定把眼光放得更远，再改良 YouCube代码的效率。她已经发现一些重复的日期格式化代码，而且认为这些部分可以巧妙地利用 OOP 原则而削减。

这些代码看来是不必要的重复。我该如何削除重复呢？



```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript">
      // Blog object constructor
      function Blog(body, date) {
        // Assign the properties
        this.body = body || "Nothing going on today.";
        this.date = date || new Date();
      }

      // Return a string representation of the blog entry
      Blog.prototype.toString = function() {
        return "[" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
          this.date.getFullYear() + "] " + this.body;
      };

      // Return a formatted HTML representation of the blog entry
      Blog.prototype.toHTML = function(highlight) {
        // Use a gray background as a highlight, if specified
        var blogHTML = "";
        blogHTML += highlight ? "<p style=background-color:#EEEEEE>" : "<p>";

        // Generate the formatted blog HTML code
        blogHTML += "<strong>" + (this.date.getMonth() + 1) + "/" + this.date.getDate() + "/" +
          this.date.getFullYear() + "</strong><br />" + this.body + "<br /><em>" + this.signature +
          "</em></p>";
        return blogHTML;
      };

      // See if the blog body contains a string of text
      Blog.prototype.containsText = function(text) {
        return (this.body.indexOf(text) != -1);
      };

      // Set the blog-wide signature
      Blog.prototype.signature = "by Puzzler Ruby";
```


动动脑

该如何削减 YouCube 里重复的日期格式化代码呢？

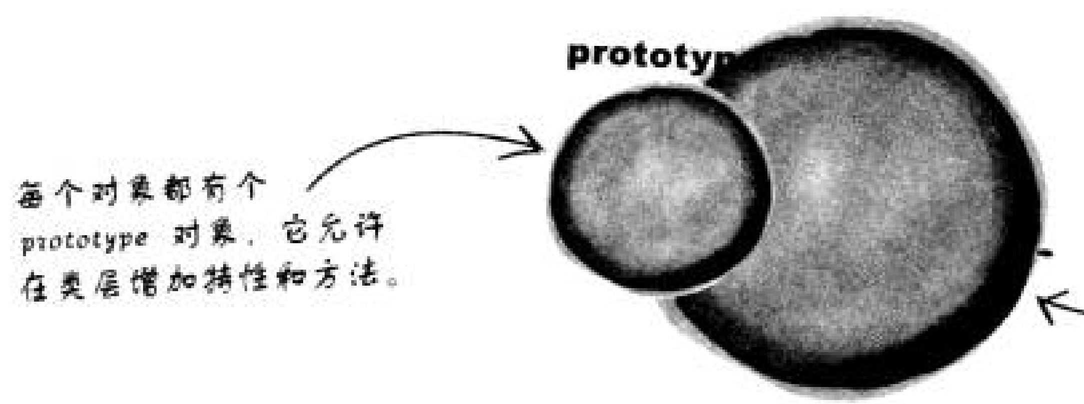
日期格式化方法

Ruby 认为，解决重复日期格式化代码的恰当方案，就是在 Blog对象里增加一个日期格式化方法。为了代码的重复利用，日期格式化的设计必须放入函数或方法。她或许会选择方法，因为Blog对象负责为日期加上格式，并制成博客日志的一部分。她应该这样做吗？

如果“设计日期格式”其实是 Date 对象的一项行为，把它制作成附属于 Date 对象的方法，不是更合理吗？有办法把方法加到标准 JavaScript 对象中吗？

回到 prototype 对象

利用原本存在的对象并让它们更加完美，你能想到更为强大的力量吗？确实有办法调整标准对象，而且还是扩展JavaScript语言的终极选项。扩展标准对象或任何JavaScript对象的关键，都是prototype对象。我们已经使用 prototype 对象扩展过隶属于 Blog 类的特性和方法，也能采用相同方式扩展内置的JavaScript类，没人能阻止我们。

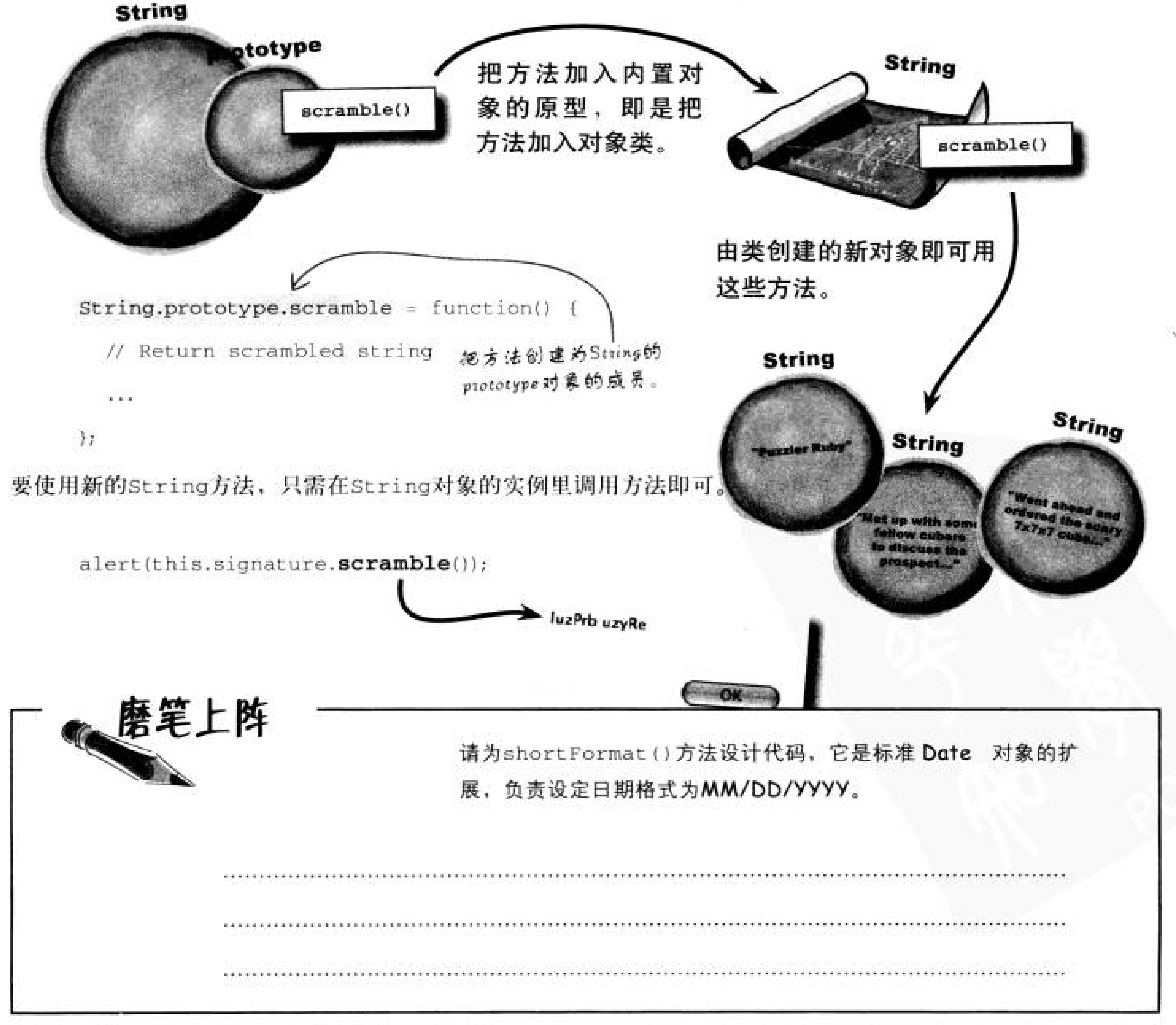


任何对象，甚至包括 JavaScript 标准对象。

扩展标准对象

扩展对象的关键就是prototype对象，每个 JavaScript 对象里都有它。所以，扩展任何对象，只需要把特性和方法加入它的prototype对象里，借以建立**类特性**和**类拥有的方法**。在内置JavaScript 对象的情况中，把特性和方法新增到 prototype 对象里，即表示任何由内置对象制造的新实例，均可访问我们自定义的特性和方法。

prototype对象能扩展内置JavaScript对象。




```
String.prototype.scramble = function() {
  // Return scrambled string
  ...
};
```

要使用新的String方法，只需在String对象的实例里调用方法即可。

```
alert(this.signature.scramble());
```

磨笔上阵

请为shortFormat()方法设计代码，它是标准 Date 对象的扩展，负责设定日期格式为MM/DD/YYYY。

....

....

....

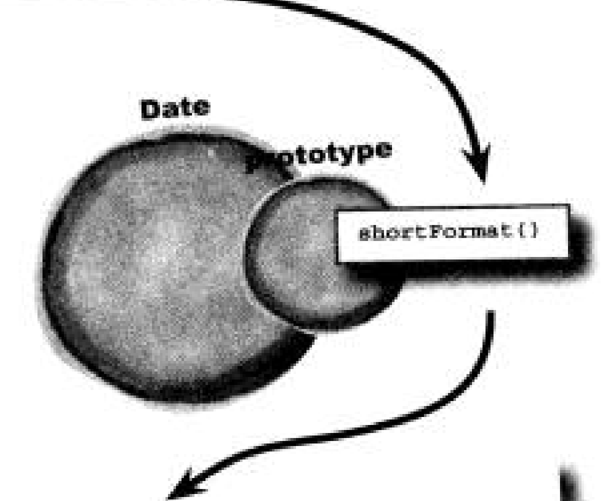
请为shortFormat()方法设计代码，它是标准Date对象的扩展，负责设定日期格式为MM/DD/YYYY。

这个方法被加入Data对象的prototype里。

```
Date.prototype.shortFormat = function() {
  return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();
};
```

自定义的日期对象 = 改良的 YouCube

自定义的Date对象让 YouCube 更有效率，而且扩展了内置对象的功能。YouCube 也更容易维护，因为日期格式已可只于单一位置修改，但同时影响整个博客的日期外观。当然，OOP 对脚本的改良虽不见得总能立刻看到改良的魔雾蒸腾而起，但通常能在代码层发生作用，而且长期下来运作得更好。



9/3/2008
Attended a rally outside of a local toy store that stopped carrying cube puzzles. Power to the puzzlers!
by Puzzler Ruby

9/1/2008
Went ahead and ordered the scary 7x7x7 cube. Starting a mental exercise regimen to prepare.
by Puzzler Ruby

8/29/2008
Met up with some fellow cubers to discuss the prospect of a 7x7x7 cube. Mixed feelings.
by Puzzler Ruby

8/21/2008
Found a 7x7x7 cube for sale online. Yikes! That one could be a beast.
by Puzzler Ruby

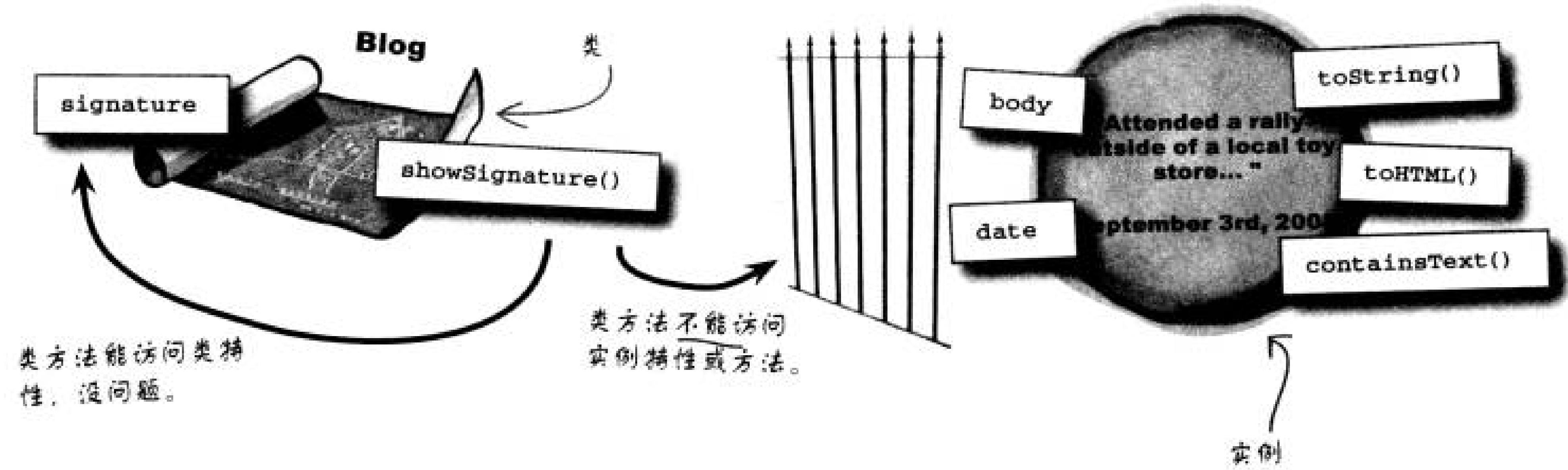
8/19/2008
Solved the new cube but of course, now I'm bored and shopping for a new one.
by Puzzler Ruby

现在使用Date对象的自定义方法，设定了博客日期的函数。

类能有自己的方法

类方法为类所拥有，但只能访问类特性。

自定义的 shortFormat() 方法（隶属于 Date 对象），是个**类拥有的实例方法**（class-owned instance method），意思是说，虽然它属于类，但能访问实例特性。因此，方法才能为存储在实例里的日期数据加上格式。另外，也可以创建**类方法**（class method）—— 为类所有但**无法**访问实例特性的方法。不过，类方法能访问类特性，例如访问Blog类的signature特性。



类方法的创建，是为类设定方法，但**不**使用 prototype 对象 —— 仅使用类名和对象符号（点号）把方法指派给类。

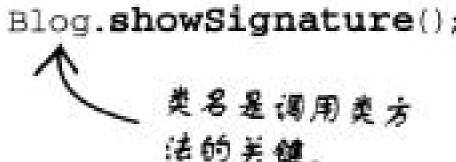
为了从类方法里访问类特性，你必须下探至prototype特性。

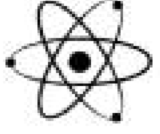
```
Blog.showSignature = function() {
  alert("This blog created by " + Blog.prototype.signature + ".");
};
```

因为signature是个类特性，类别方法可以访问它。

既然类方法与实例没有关联，调用时只需引用类名。但是实例并非不能调用类方法，只是需要透过类名。

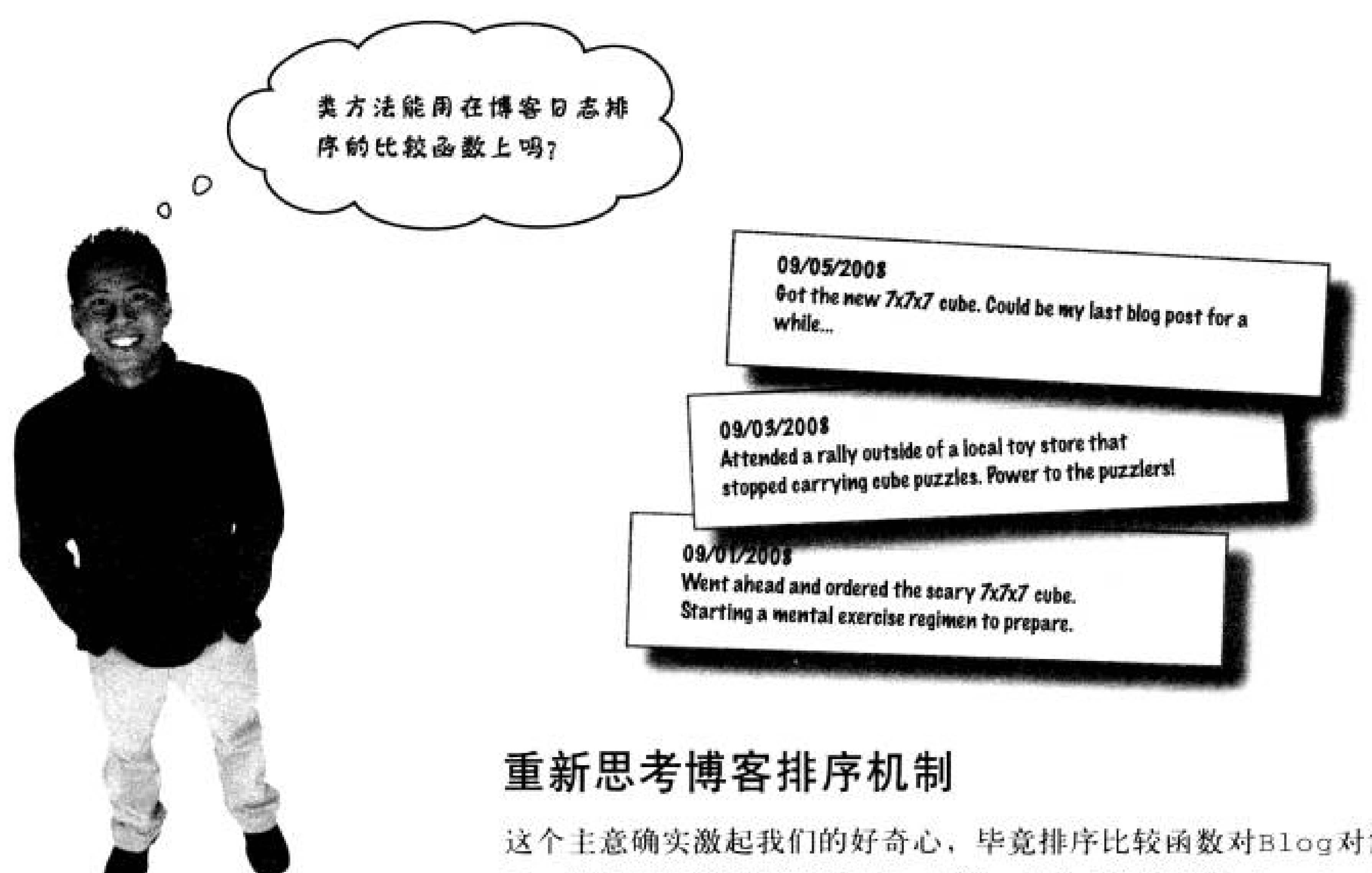
```
Blog.showSignature();
```





动动脑

在 YouCube 里，你能想出任何适合作为 Blog 类方法的代码吗？



重新思考博客排序机制

这个主意确实激起我们的好奇心，毕竟排序比较函数对Blog对象而言，绝对占有非常特殊的地位。目前，这个函数创建为showBlog()函数里的一个函数字面量，showBlog()也是需要排序机制出现的地方。

```
function showBlog(numEntries) {
  // First sort the blog in reverse chronological order (most recent first)
  blog.sort(function(blog1, blog2) { return blog2.date - blog1.date; });
  ...
}
```

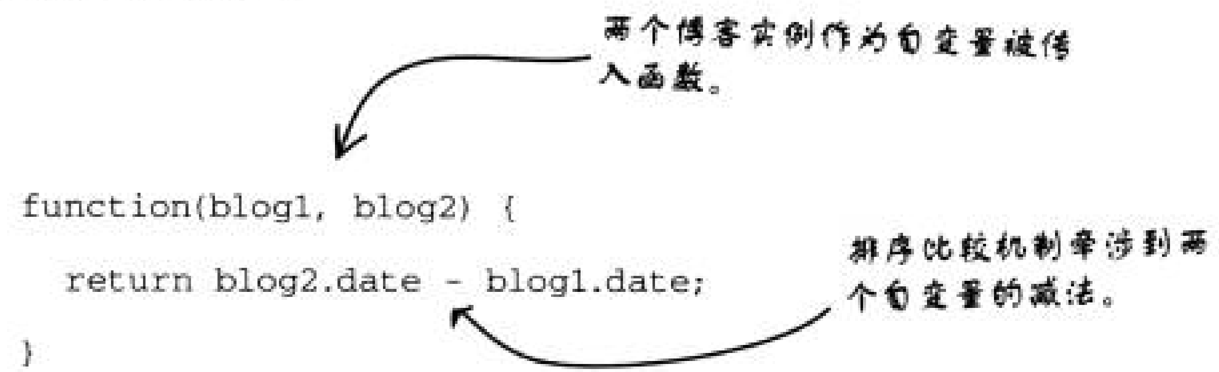
博客的排序在slowBlog()函数内处理，它不是Blog对象的一部分。

或许能把排序比较代码移到类方法里。

OOP的基础概念之一：把对象的功能**封装**（encapsulate）至对象里，表示外部代码不应该负责对象本身可负责的工作。就本例而言，比较博客日志以利排序可在对象里处理，而不用另外做一个showBlog()函数。但是，排序比较代码能放在 Blog 类的类方法里吗？为了回答这个问题，我们必须知道方法是否需要访问实例数据或方法。因为类方法无法访问实例的任何资源，这是个大问题。

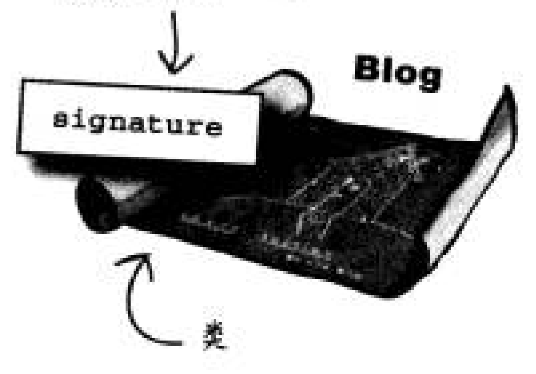
检验排序比较函数

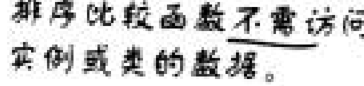
想知道类方法是否可行的唯一方式，就是拆解函数并理解它的运作。
下面列出排序比较函数字面量：

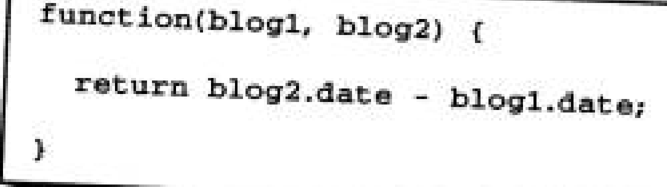


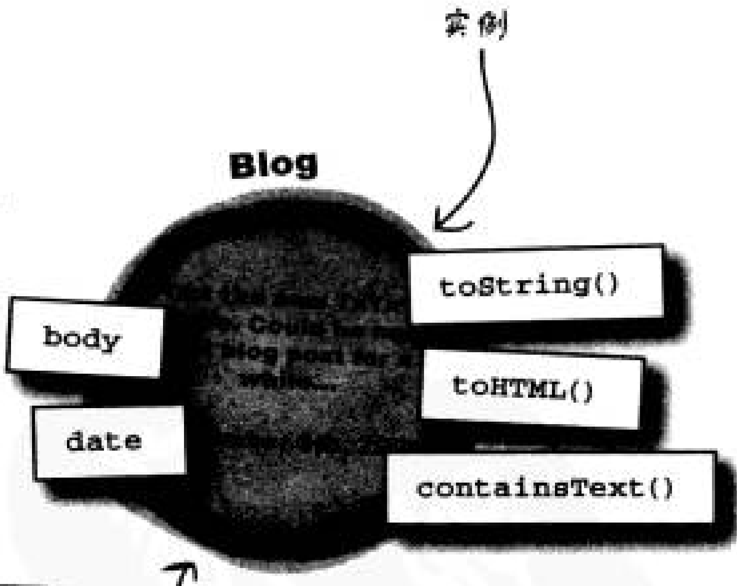
虽然函数直接处理博客实例，但它们是以自变量的形式被传入函数。"使用自变量"与试图"利用 this 而访问实例**内部**的特性和方法"有所不同，类方法无法达成后者。所以，排序比较函数不需访问实例中的任何事物，它是类函数的最佳候选者。

事实上，排序比较函数甚至不需要类特性，虽然它在必需时可以访问，因为类方法确实有权访问类特性。









重新设计YouCube的博客排序比较函数，把它改为Blog对象的类方法并命名为blogSorter()。

....

....

....

磨笔上阵解答

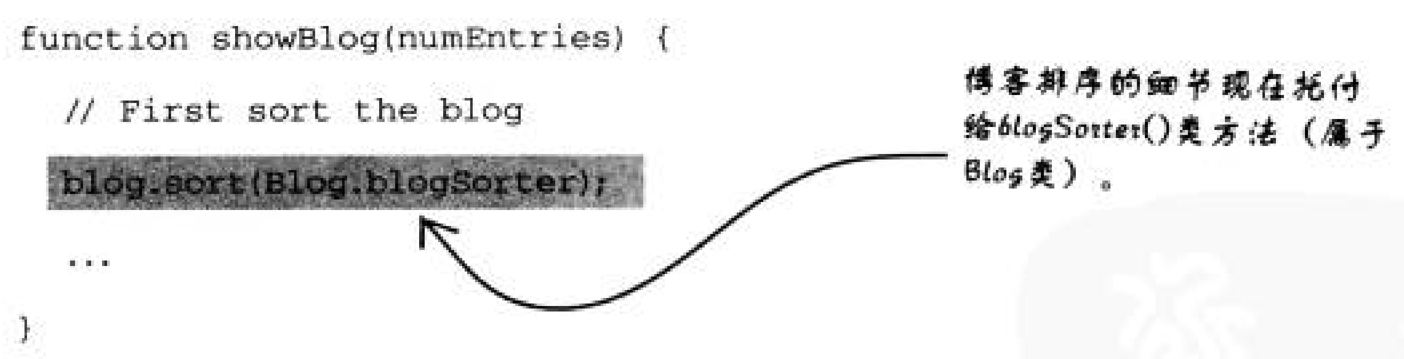
重新设计 YouCube 的博客排序比较函数，把它改为Blog对象的类方法并命名为 blogSorter()。

```
Blog.blogSorter = function(blog1, blog2) {
  return blog2.date - blog1.date;
};
```

排序比较方法现在是 Blog 对象的类并改名为 blogSorter()。

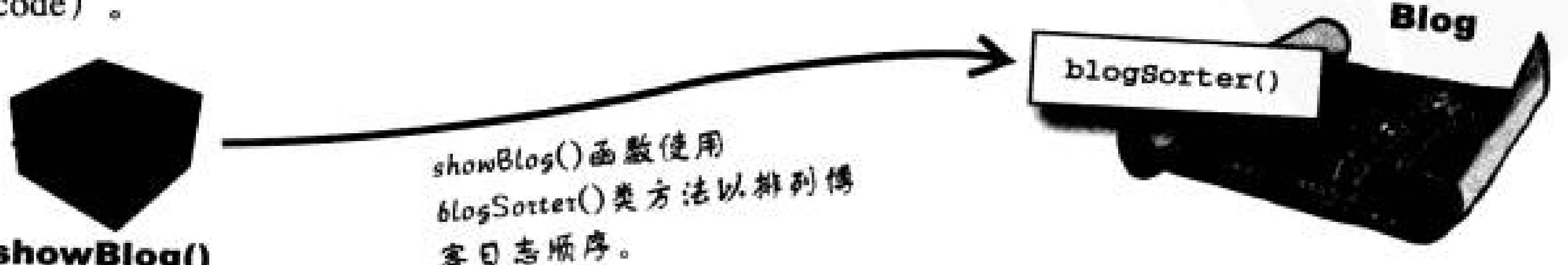
调用类方法

把博客的排序比较函数移作 Blog 方法后，变得较容易看出程序代码何时调用方法。



上述程序代码的优美之处非常细致，但非常重要。showBlog() 函数不再需要担心博客日志排序的问题。关于博客如何排序的细节，都交由（逻辑所属的）Blog 对象内部处理。

它的整洁灵巧之处，在于排序任务一样在 Blog 类外由showBlog() 函数起始。这很合理，因为排序会影响所有博客日志实例，但关于排序如何针对每则日志执行的细节，则是Blog 类可以处理的范畴。优秀的 OOP 设计通常关系到小心地架构对象与它周遭的环境代码（surrounding code）。



一张图胜过博客千言万语

Ruby 持续为了 OOP 带来的改良感到兴奋，但她也知道用户不见得能分享她的喜悦，因为到目前为止的 OOP 改良对于用户体验并没有多么戏剧化的影响。因此，她决定要为博客加上一些看得到的改变！

Ruby 想让每则日志支持图像的选用，与日期和日志主体一起呈现。因为并非每则日志都需要图像，所以图像的选用是此处的重点。如此亦可避免目前写好的博客日志格式崩溃。

动动脑

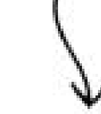
该如何调整 YouCube 的 Blog 对象以支持图像？

把图像整合至YouCube

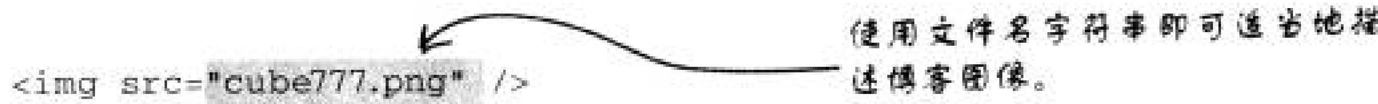
为 YouCube 博客增加对图像的支持，牵涉到如何整合图像至 `Blog` 对象，而且不会干涉对象现有的运作方式。由此导出两个驱动这项设计的重要问题：

1 什么是在 Blog 对象里存储博客图像的最佳方式？

2 博客图像如何增加到 YouCube 里，但又维持完全选用？



无论博客的图像如何存储，我们知道它终将放入 HTML的 `<img>` 标签，才能呈现在 YouCube 的网页上。



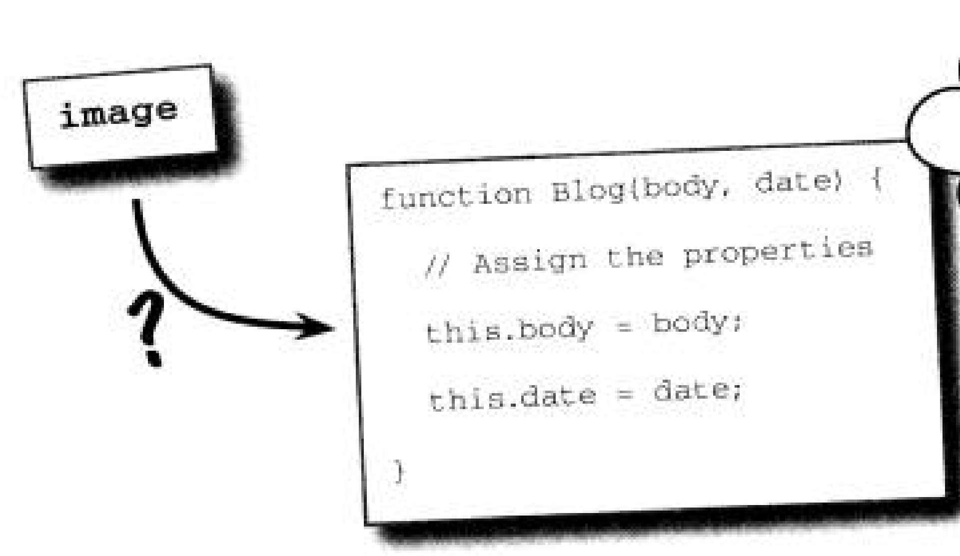
从上例代码看来，图像之于博客只是一个字符串。当然，这个字符串最后引用到图像在网站服务器上的存储位置，但就 `Blog` 对象而言，它就只是个字符串。

就 Blog 对象而言，图像只是个字符串。

所以，图像可以单纯当作一个加入 `Blog` 对象的特性，它存储的数据是字符串，就像 `body` 特性。

选用的博客图像

❷ 所以说，Blog 对象把图像当成字符串特性 image 而存储。但另一个问题还是存在：特性该如何制作成博客的选用功能。我们的问题最后必定导回构造函数 —— 创建与初始化对象的地方。想必有些特殊代码能放在构造函数里，让特性可为选用功能。



我还不太确定…………如果不传入自变量给构造函数呢？特性会被设定为null吗？

失落的函数自变量变成 null。

当某个自变量未被传给函数、方法或构造函数时，在任何试图使用自变量值的代码里，它的值都是 null。尤其在构造函数的例子中，这一点表示：与缺少的自变量相对应的特性将被设为 null，但不见得是坏事。诀窍在于，确认构造函数的可选自变量列在列表的最后面，所以不填入可选自变量也不会造成其他自变量的困扰。这项技巧其实可应用在任何函数或方法上，但它特别适用于Blog()构造函数里的image自变量。

磨笔上阵

请重新设计YouCube的Blog构造函数，以支持存储博客图像的新特性image。

....

....

....

....

....

....

磨笔上阵解答

请重新设计YouCube的Blog构造函数，以支持存储博客图像的新特性 image。

```
function Blog(body, date, image) {
  // Assign the properties
  this.body = body;
  this.date = date;
  this.image = image;
}
```

自变量image加到构造函数的自变量列表的最后面。

image 特性根据自变量image而创建与初始化。

没有蠢问题

问：image自变量出现在 Blog()构造函数的自变量列表的最后面，这件事很重要吗？

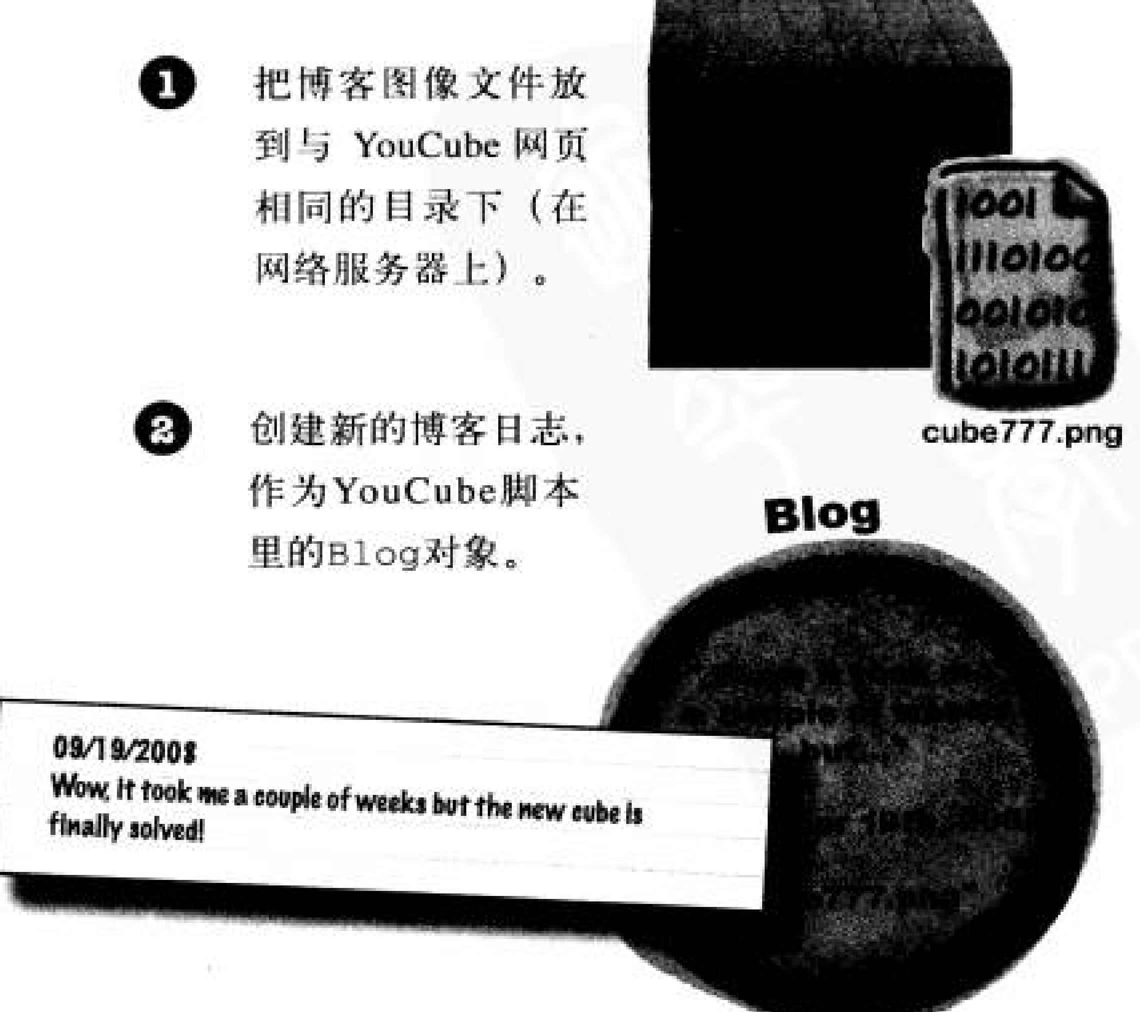
答：是的，原因在于图像被视为博客日志的选用功能。真正的重点在于自变量传给函数的方式，尤其在关于可选自变量的状况下。如果函数有两个自变量，你的选择将有：两个自变量都传入、只传入第一个自变量或两个自变量都不传入。**没办法**只传入第二个自变量。

在遇到可选自变量时，试着把它们当成放在自变量列表的尾端。也要试着考量自变量的重要性，较重要的自变量先出现。较不重要的自变量比较可能会被考虑划为选用功能，应该出现在自变量列表的较后面。既然image对 Blog() 是个可选自变量，它必须出现在自变量列表的最后面，才能轻易舍弃不写。

为 YouCube 添加图像

如果没有表演的舞台，支持图像的崭新 Blog()构造函数也无用武之地。为了创建支持图像的博客日志，需要办到两件事：

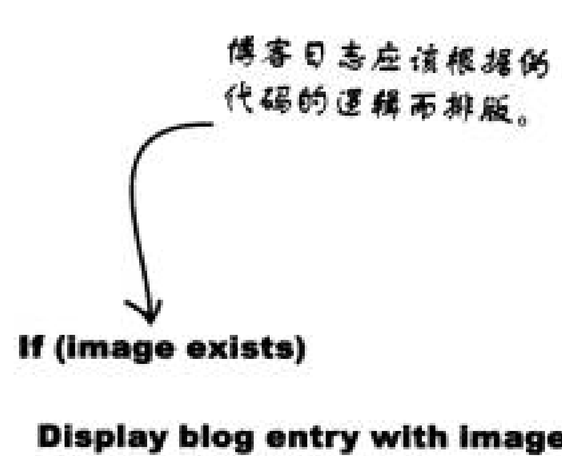
1. 把博客图像文件放到与 YouCube 网页相同的目录下（在网络服务器上）。
2. 创建新的博客日志，作为YouCube脚本里的Blog对象。



完成这些事项将形成如下代码，可成功地创建新博客日志，且该日志接受图像字符串，作为传入 Blog() 的最后一个自变量：

呈现博客图像

博客日志的创建加上图像后，YouCube 图像加强版还剩下一项工作。我们做了这么多构造函数与可选自变量的调整，但还有呈现博客日志的代码。若未考虑到新加入的 image 特性，我们等于做了半天白工。



相关代码位于 toHTML() 方法。我们已经知道这个方法负责把博客的格式转换为 HTML代码，它现在只需多考虑一件事：image 特性是否为有意义的值。实际上将有两种呈现博客日志的方式，一种附有图像，另一种没有，而图像的存在与否则决定日志采用的呈现方式。

If (image exists)

Display blog entry with image

Else

Display blog entry without image

Blog对象的toHTML()方法缺少一些代码，都是关于呈现选用图像的部分。请补完缺少的部分并说明其用途。

```
if ( ............................ ) {
  blogHTML += "<strong>" + this.date.shortFormat() +
    "</strong><br /><table><tr><td><img src='" + this.image +
    "'/></td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +
    this.signature + "</em></p>";
}
else {
  blogHTML += "<strong>" + this.date.shortFormat() + "</strong><br />" + this.body +
    "<br /><em>" + this.signature + "</em></p>";
}
```

磨笔上阵解答

Blog 对象的 toHTML() 方法缺少一些代码，都是关于呈现选用图像的部分。请补完缺少的部分并说明其用途。

```
if ( this.image ) {
  blogHTML += "<strong>" + this.date.shortFormat() +
    "</strong><br /><table><tr><td><img src='" + this.image +
    "'/></td><td style='vertical-align:top'>" + this.body + "</td></tr></table><em>" +
    this.signature + "</em></p>";
}
else {
  blogHTML += "<strong>" + this.date.shortFormat() + "</strong><br />" + this.body +
    "<br /><em>" + this.signature + "</em></p>";
}
```

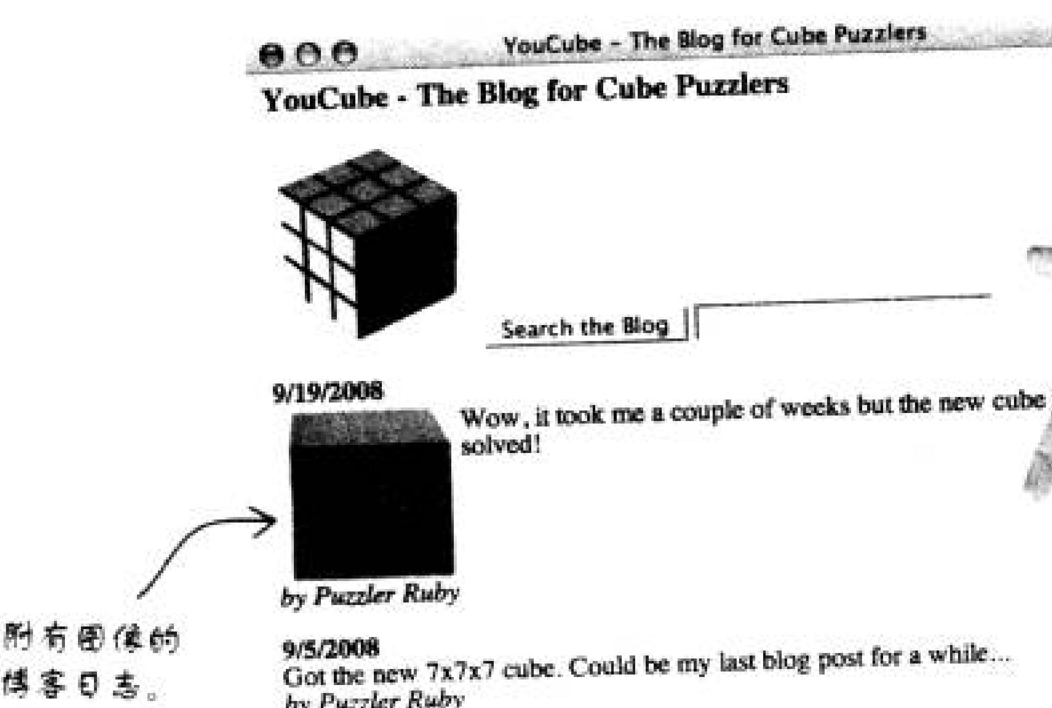
如果 image 特性设定为实际图像，if 条件句即为 true，因而呈现图像。

否则，博客日志就以普通、没有图像的格式呈现。

我知道……
它很美吧！

由对象驱动的 YouCube

Ruby 此时雀跃不已。她的博客因为对象而有跳跃性的改良，而且现在又加上了用户想必喜欢的图像功能。



附有图像的博客日志。

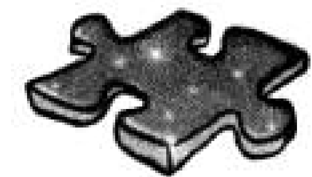
JavaScript 填字游戏

横向提示：

2. 自己拥有数据、有生命的对象。
4. 创建对象实例的运算符。
6. YouCube 的签名是种________。
9. 使用对象设计的软件，称为________。
11. 从对象自身的程序代码引用对象的关键字。
12. 从其他对象继承特性和方法的对象。

纵向提示：

1. 对象里等于函数的事物。
3. YouCube 的博客排序比较代码是种________。
5. 当一块数据放入对象里，它被称为________。
7. 每个对象都有个________对象隐藏在它心里。
8. 用于创建对象实例的模板。
10. 用于访问特性和方法的对象符号。



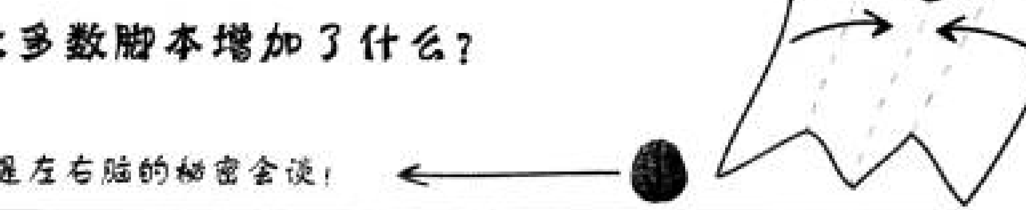
JavaScript 填字游戏解答

													1 M					
			2 I	N	S	T	A	N	3 C	E		4 N	E	W				
									L				T					
			5 P						A				H					
			R			6 C	L	A	S	S	P	R	O	P	E	R	T	Y
7 P			O						S				D					
R			P						M							8 C		
9 O	B	J	E	C	T	O	R	I	E	N	T	E	10 D			L		
T			R						T				O			A		
O			T						H				11 T	H	I	S		
T			Y						O							S		
Y					12 C	H	I	L	D									
P																		
E																		

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

对象为大多数脚本增加了什么？



who

what

signature

display()

where

deliver()

when

对象为脚本增加太多酷炫的功能，
很难单独挑出一项。
没错，有些对象比其他
对象的层次高，让我们的挑选更为困难，
不过最终答案应该很明显。

11 除错务尽

好脚本也会出错

即使最佳规划的 JavaScript 蓝图也可能出错。当这种情形发生，事实上一定会发生，你的工作就是 —— 别惊慌。最好的 JavaScript 开发人员，不是从未造成程序出错的人 —— 这种人叫骗子。最好的 JavaScript 开发人员，要能够**成功猎捕并消灭自己**制造的**缺陷**（bug）。还有，一流的 JavaScript 缺陷终结者也会**发展出良好的编程习惯**，使最狡诈、最棘手的缺陷的出现率降到最低。**小心驶得万年船。**但缺陷如果登舰了，你将需要握紧自动机枪，准备战斗……

现实生活中的除虫大队

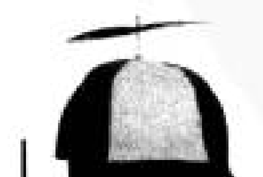
点心世界面临了大冲击……一条巧克力最高可以容纳 60 块昆虫碎片！先不管这则花絮听起来多恐怖，我们在 JavaScript 世界中不用害怕昆虫的出现。JavaScript 代码的控制比巧克力处理设备严谨多了，甚至还有单独移除 JavaScript 臭虫（缺陷）的力量。

Bug Scene Investigators（罪虫现场调查小组），在同业间简称BSI。Owen最近刚加入BSI，专司JavaScript调查，他正急切地想证明自己的能力，并协助万维网摆脱JavaScript缺陷。

Owen，BSI 的 JavaScript 调查员，原本也是巧克力爱好者。

爬满缺陷的巧克力棒……好恶心！

Owen 想爬上成功的宝座，他需要赶快处理几个紧迫逼人的案子，熟悉 JavaScript 的除虫黑魔法，才能提升自己的职位、成为干练的 JavaScript 探长。



技客新知

根据美国食品及药物管理局的规定，每条巧克力棒里最高可能有 60 块昆虫“碎片”。与“现实生活”的状况相反，BSI 小组成员对于 JavaScript 缺陷采取毫不容忍的政策，你也应该这样做。

案件：爬满缺陷的IQ计算器

Owen 桌上的第一个案子是 IQ 计算器，它是某个网页的一部分，负责根据 IQ数组计算平均智商。所以，这个脚本会收到一个数字数组，而后计算数组平均值并表示平均智商值。

Owen 面对的案件代码可至 http://www.headfirstlabs.com/books/hfjs/ 下载。

同事们已经跟 Owen 讲过，这个脚本的错误多如牛毛。很不幸地，除了“没办法用”，没人提出更多信息。

你的除错人生不见得总会拿到写得很好的程序代码。

改用不同浏览器

Owen 想到，多在几种浏览器上运行问题脚本，或许有助于发现问题的蛛丝马迹。他先用 Internet Explorer……

Internet Explorer

双击 IE 窗口左下角的黄色图示，打开错误信息窗口。

Internet Explorer 在网页首度载入时汇报了错误信息，但Owen不确定是否可以信任汇报的错误。看一下代码，变量iqs明明存在，但 IE 浏览器说它不存在。Owen 知道，浏览器在汇报错误时不见得准确，他决定换 Safari 试试……

代码确实定义了变量 iqs，所以 IE 的错误汇报不见得有意义。

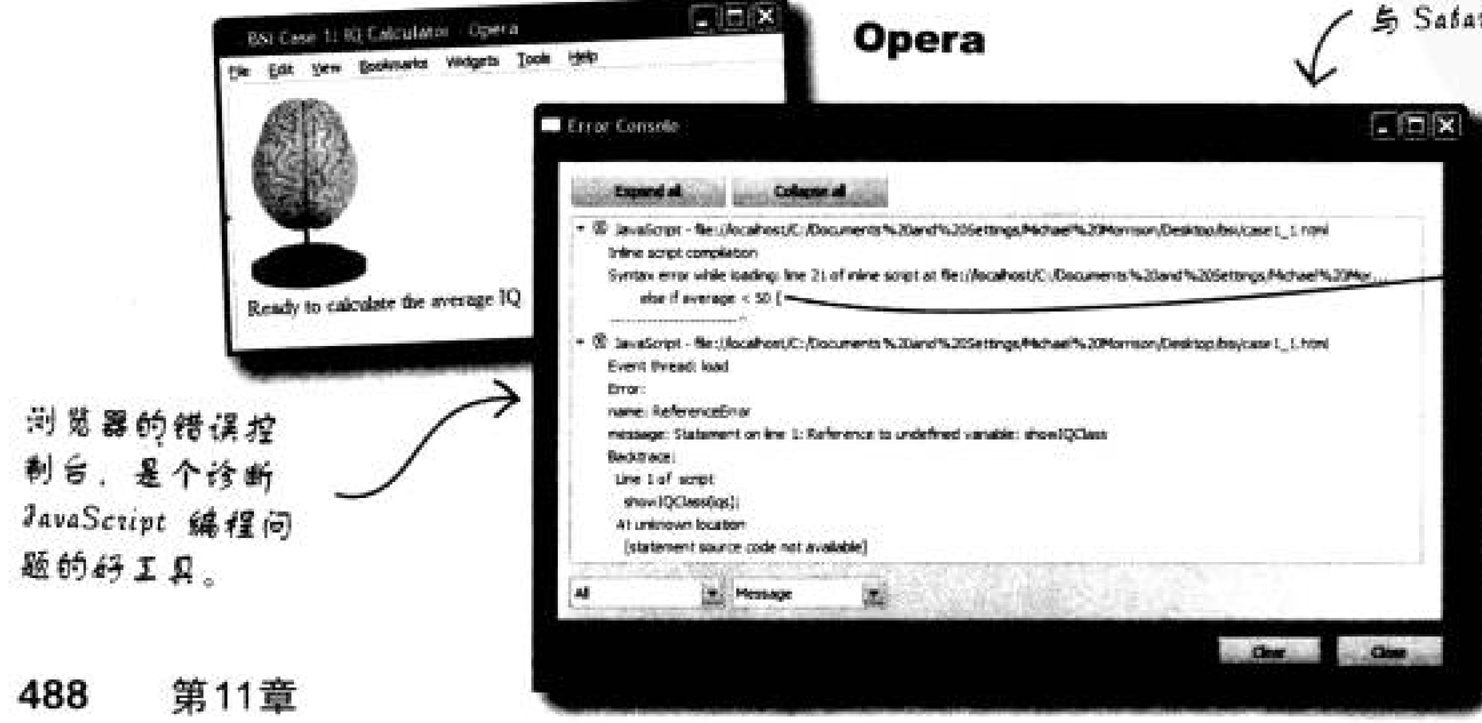
Safari

如果你从 1 开始计算代码行数，Safari 指出的错误代码看起来好像没问题。

Safari 指出的错误与 IE的完全不在不同一行，而 Owen 现在也还看不出 Safari 指出的这行有什么问题。所以他决定改用 Opera 尝试找出错误问题的位置……

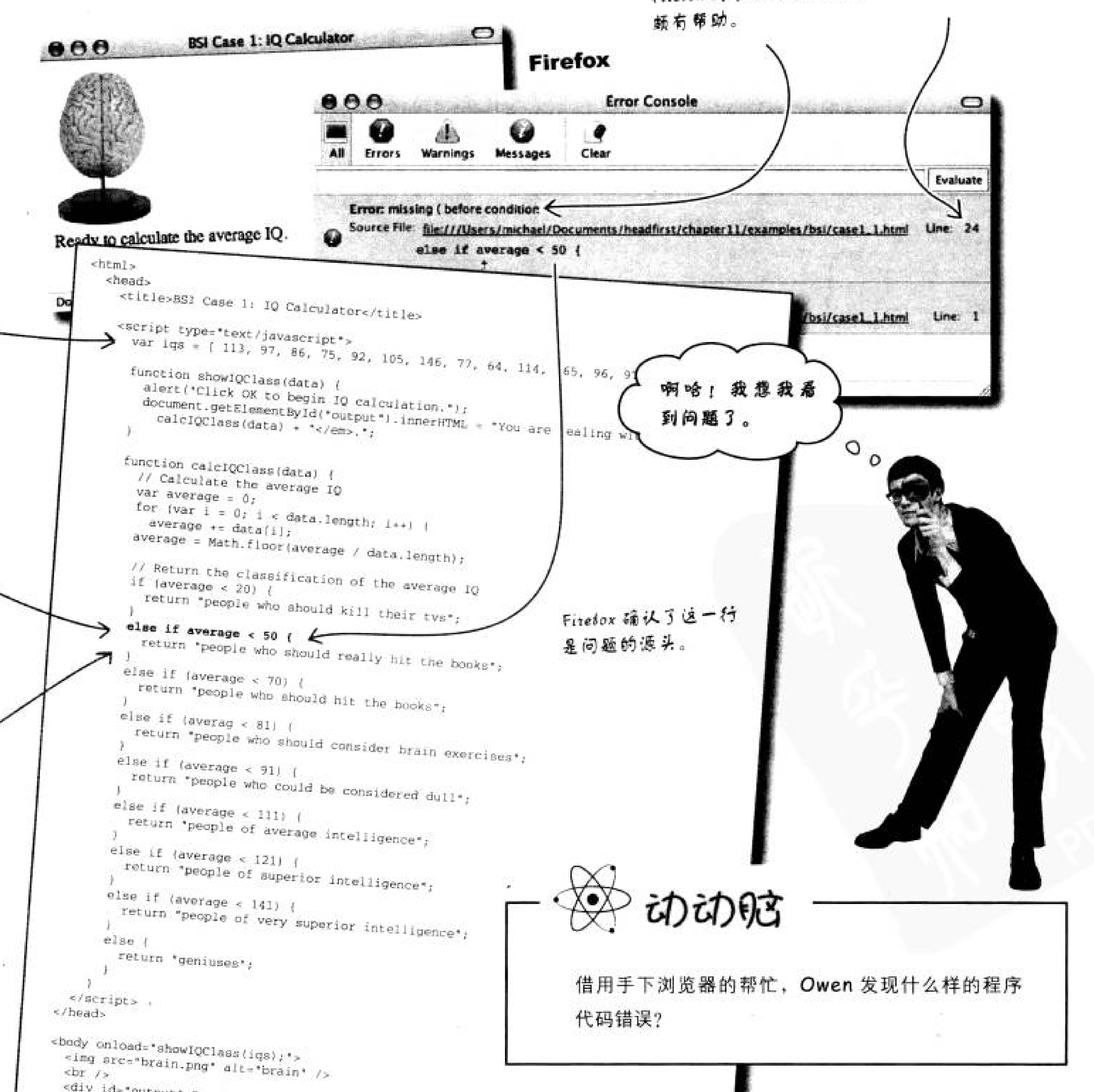
行数又不一样了，但 Opera 错误控制台呈现的代码问题与 Safari 的相同。

Opera



浏览器的错误控制台，是个诊断 JavaScript 编程问题的好工具。

有点奇怪。Opera 提出的行数不同，但显然与 Safari 指出的问题相同，这对 Owen 倒是个好消息。不过，他还是看不出那一行代码的问题。所以他决定再试另一种选择，Firefox……




```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 65, 96, 9

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are  eeling wi
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calculate the average IQ
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Return the classification of the average IQ
        if (average < 20) {
          return "people who should kill their tvs";
        }
        else if average < 50 {
          return "people who should really hit the books";
        }
        else if (average < 70) {
          return "people who should hit the books";
        }
        else if (averag < 81) {
          return "people who should consider brain exercises";
        }
        else if (average < 91) {
          return "people who could be considered dull";
        }
        else if (average < 111) {
          return "people of average intelligence";
        }
        else if (average < 121) {
          return "people of superior intelligence";
        }
        else if (average < 141) {
          return "people of very superior intelligence";
        }
        else {
          return "geniuses";
        }
      }
    </script>
  </head>

  <body onload="showIQClass(iqs);">
    <img src="brain.png" alt="brain" />
    <br />
    <div id="output">Ready to calculate the average IQ.</div>
  </body>
</html>
```

动动脑

借用手下浏览器的帮忙，Owen 发现什么样的程序代码错误？

它是只鸟，它是架飞机……它是火狐狸

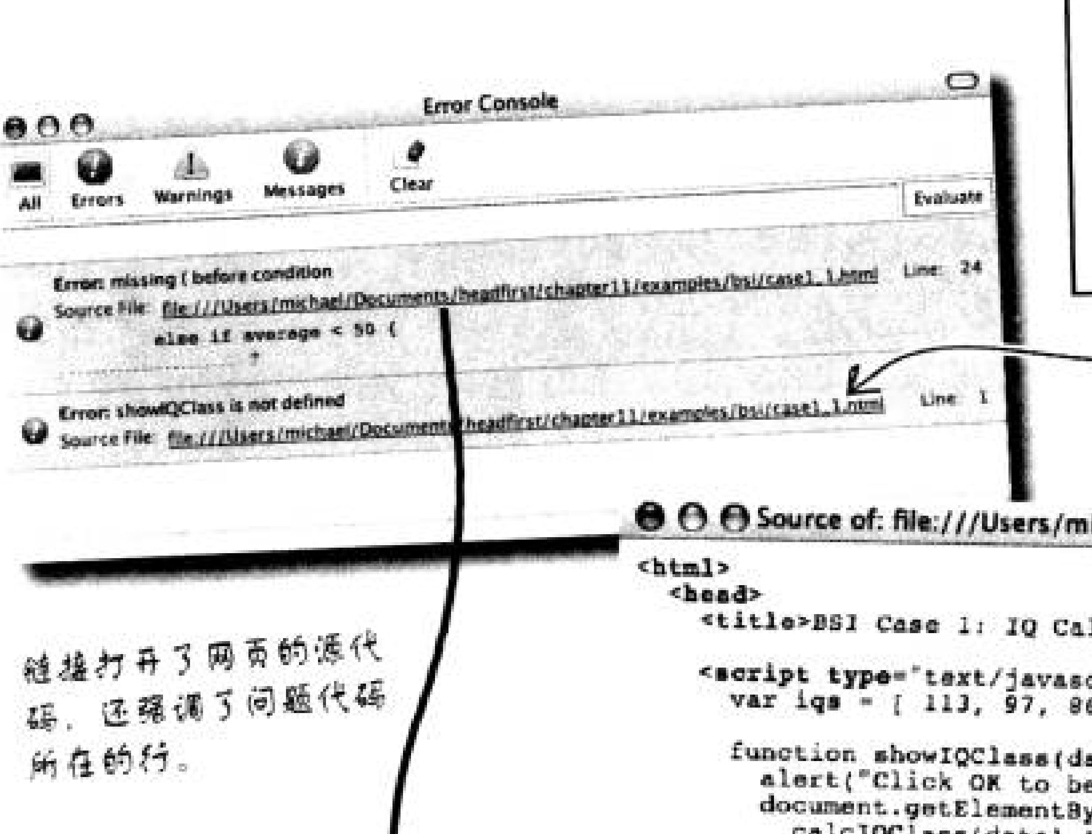
Firefox 伸出援手

有了 Firefox 详细的缺陷说明，Owen 决定使用 Firefox 深入研究。所以他按下 Firefox 错误控制台里的链接，链接把他直接带到问题代码所在的那行。

Firefox是普遍认为最好的调试浏览器，至少目前还是。

技客新知

Firefox不只具有极佳的内置缺陷检测能力，它也有调试器（debugger）插件Firebug，把调试工作带到完全不同的层次。Firefox的Firebug调试器可于 *http://www.getfirebug.com* 免费下载。



先别烦恼第二项错误，我们一项一项各个击破。

链接打开了网页的源代码，还强调了问题代码所在的行。

这就是造成问题的代码。

Source of: file:///Users/michael/Documents/headfirst/examples/chapter11/bsi/ca...

```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calculate the average IQ
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Return the classification of the average IQ
        if (average < 20) {
          return "people who should kill their TVs";
        }
        else if average < 50 {
          return "people who should really hit the books";
        }
        else if (average < 70) {
          return "people who should hit the books";
        }
        else if (averag < 81) {
          return "people who should consider brain exercises";
        }
        else if (average < 91) {
```

Line 25, Col 1

分析了 Firefox 的错误信息后，Owen 发现 Safari 对于行数的汇报（25）其实没错；Firefox 虽然汇报第 24 行，但它强调的地方却是第 25 行。更重要的一点，Firefox 解释了程序代码究竟有什么问题，本例的问题虽然简单，却很容易众里寻它千百度，还是漏掉了。

```
else if average < 50 {
```

if 语句的条件测试周围缺少括号。

问：我不知道要怎么打开浏览器的错误控制台耶。该怎么打开它呢？

答：很可惜，每家浏览器的作风都不一样，有些甚至很难找到 JavaScript 错误控制台。例如说 Mac 版 Safari，它只允许从 Debug 菜单访问错误控制台，但这个菜单默认为关闭的。你必须在 Terminal 应用程序里执行下列命令（命令的输入需为一行，不加回车符〔carriage return〕），以开启 Debug 菜单：

```
defaults write com.apple.Safari
IncludeDebugMenu 1
```

请参考你所使用的浏览器说明文档，了解该如何开启错误控制台以及如何查看脚本错误。Firefox 的错误控制台，可以"工具"（Tool）→"错误控制台"（Error Console）打开。

问：为什么 Firefox 如此特别？

答：Firefox 的众多开发者为它的错误汇报能力做了相当好的贡献。讲到评估脚本错误，它就是比其他浏览器表现得更好，而且能指引正确的错误寻找方向。不知道未来是否将出现超越 Firefox 的浏览器调试功能，但 Firefox 已证明自己非常适合去除网页脚本里所包含的缺陷。

问：那 Internet Explorer 讲的错误又是什么呢？

答：其实没办法百分之百地确定。因为 Internet Explorer 汇报的错误必定关于脚本未适当地载入，这是 JavaScript 解释器遇到错误的结果。为什么知道脚本未适当地载入？因为它汇报 iqs 未定义（undefined），虽然代码明明就先创建了变量 iqs。所以它成为问题的唯一成因，就是有什么力量阻止脚本完整载入。

由此也引出另一个问题……脚本里还有其他错误吗？"未定义"的真正意义又是什么？

在轻松街道上除虫

这么快就找出 IQ 计算器脚本的缺陷，Owen 为此手舞足蹈、乐不可支。而且这个案子的代码修正工作又很简单，他想，要在工作上达到最高效能，晋升 BSI 探长已经指日可待了。

```
else if (average < 50) {
```

把条件测试放入括号中，解决了 IQ 计算器的缺陷一只。

加上缺少的括号，消灭缺陷！

Owen 会不会太过自信呢？他在上班溜号前，最好检查一下刚修理过的脚本……

汇报的缺陷不见得都是错误之源

很不幸，Owen 的 IQ 计算器案件尚未结束，因为 Firefox 还在抱怨，但这次抱怨另一个完全不同的问题。他虽然想维持前面的策略，直接信任 Firefox 的评估，但 Owen 这次开始质疑缺陷汇报的合格度了。

Firefox 提到的函数大括号 ({}) 已经成对出现了，所以并未缺少大括号。

一次处理一只缺陷，我们还是先忽略这个问题。

Error Console

All Errors Warnings Messages Clear Evaluate

Error: missing } after function body
Source File: file:///Users/michael/Documents/headfirst/chapter11/examples/bsi/case1_2.html Line: 53

Error: showIQClass is not defined
Source File: file:///Users/michael/Documents/headfirst/chapter11/examples/bsi/case1_2.html Line: 1

这就是 Firefox 提到的大括号。

我看你的神奇调试器刚刚坏掉了。显然，这一对围起函数的大括号没有问题。

你不能无条件地相信浏览器。

没错，函数的大括号没有问题。虽然在某些状况下的表现还不错，但 Firefox 面对这只缺陷时搞错对象了。不过，消失的大括号仍然是条线索，建议你仔细研究代码中的每个大括号。

与 JavaScript 解释器天人合一

你将扮演 JavaScript 解释器，并研究代码中的大括号，以找出问题所在。

```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calculate the average IQ
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Return the classification of the average IQ
        if (average < 20) {
          return "people who should kill their tvs";
        }
        else if (average < 50) {
          return "people who should really hit the books";
        }
        else if (average < 70) {
          return "people who should hit the books";
        }
        else if (averag < 81) {
          return "people who should consider brain exercises";
        }
        else if (average < 91) {
          return "people who could be considered dull";
        }
        else if (average < 111) {
          return "people of average intelligence";
        }
        else if (average < 121) {
          return "people of superior intelligence";
        }
        else if (average < 141) {
          return "people of very superior intelligence";
        }
        else {
          return "geniuses";
        }
      }
    </script>
  </head>

  <body onload="showIQClass(iqs);">
    <img src="brain.png" alt="brain" />
    <br />
    <div id="output">Ready to calculate the average IQ.</div>
  </body>
</html>
```

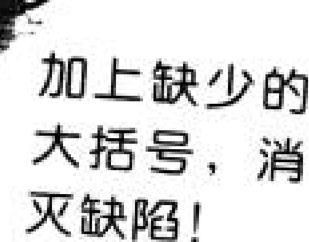
与 JavaScript 解释器天人合一解答

你将扮演 JavaScript 解释器，并研究代码中的大括号，以找出问题所在。

```
<html>
  <head>
    <title>BSI Case 1: IQ Calculator</title>

    <script type="text/javascript">
      var iqs = [ 113, 97, 86, 75, 92, 105, 146, 77, 64, 114, 165, 96, 97, 88, 108 ];

      function showIQClass(data) {
        alert("Click OK to begin IQ calculation.");
        document.getElementById("output").innerHTML = "You are dealing with <em>" +
          calcIQClass(data) + "</em>.";
      }

      function calcIQClass(data) {
        // Calculate the average IQ
        var average = 0;
        for (var i = 0; i < data.length; i++) {
          average += data[i];
        average = Math.floor(average / data.length);

        // Return the classification of the average IQ
        if (average < 20) {
          return "people who should kill their tvs";
        }
        else if (average < 50) {
          return "people who should really hit the books";
        }
        else if (average < 70) {
          return "people who should hit the books";
        }
        else if (averag < 81) {
          return "people who should consider brain exercises";
        }
        else if (average < 91) {
          return "people who could be considered dull";
        }
        else if (average < 111) {
          return "people of average intelligence";
        }
        else if (average < 121) {
          return "people of superior intelligence";
        }
        else if (average < 141) {
          return "people of very superior intelligence";
        }
        else {
          return "geniuses";
        }
      }
    </script>
  </head>

  <body onload="showIQClass(iqs);">
    <img src="brain.png" alt="brain" />
    <br />
    <div id="output">Ready to calculate the average IQ.</div>
  </body>
</html>
```

消失的结尾大括号应该加入这里，结束变量average的加法运算。

这个起始大括号缺少成对的结尾大括号！



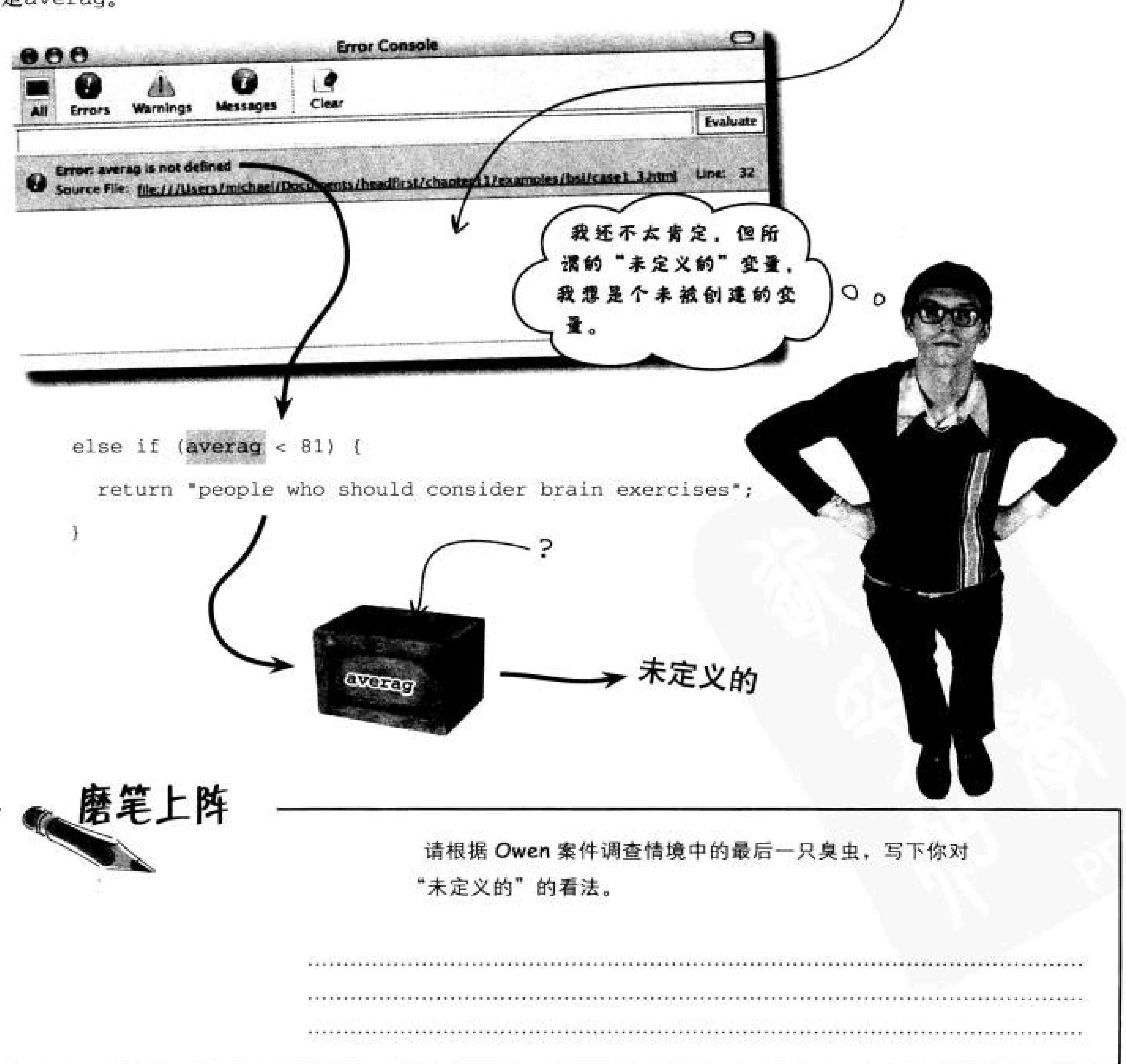
加上缺少的大括号，消灭缺陷！

因为这个循环只运行一行代码，你也可以删除for循环的起始大括号，虽然大括号有助于厘清循环里运行的代码内容。

不对称或缺少的大括号是种常见的JavaScript缺陷，但只要小心提防就能避免。

变量~~发狂~~未定义

臭虫一只接一只出现，Owen 连停下来喘口气的时间都没有。现在，Firefox汇报有个“未定义的”（not defined）变量，听起来很像Internet Explorer稍早汇报的错误。但是，这次的未定义变量不是iqs，而是averag。




```
else if (averag < 81) {
  return "people who should consider brain exercises";
}
```

磨笔上阵

请根据 Owen 案件调查情境中的最后一只臭虫，写下你对“未定义的”的看法。

....

....

....

请根据 Owen 案件调查情境中的最后一只臭虫，写下你对“未定义的”的看法。

“未定义的”是指未（使用 var）创建的变量，或虽然创建了，却未指派值的变量。无论是哪种意义，问题都在于引用了没有值的变量。

出错的地方有时候很简单

在这个案件里，“未定义的”（undefined）表示使用了未经创建的变量，虽然它是个意外。变量未定义的唯一理由，其实是打错字，造成 JavaScript 解释器认为遇到了全新的变量。

像打错字这种简单问题，通常能造成脚本的大浩劫。

检测出打错字才是困难的部分……修复这个问题很简单。

修复打错的变量名称，解决变量未定义的问题。

```
else if (average < 81) {
  return "people who should consider
    brain exercises";
}
```

averag != average

没有蠢问题

问：“并未定义的”与“未定义的”（undefined）有什么不同吗？

答：没有。它们表示相同意义，只是某些浏览器爱用某个词，有些则爱用另一个词而已。请把这两个词视为可完全互相交换。

问：好的，那么“未定义的”与null有什么不同吗？

答：这个问题比较值得注意。是的，在非常技术的层面，“未定义的”与null确实有些差异，但还不到需要担心的程度。“未定义的”不像null，你甚至不用考虑把它指派给变量。有个undefined数据类型，凡是未被指派值的变量，即自动假设为这个类型。另一方面，变量从来不会被自动设为null。然而，有时若在初始化步骤时设定对象变量为null，可用于确保对象于尚未创建前保持干净。

先不管核心的技术细节，关于“未定义的”与null的差别：它们在boolean上下文中都被转换为false，例如if语句的测试条件里。所以像if (someObject)这类程序代码，常先用于检查对象是否创建，再试图访问对象的成员。

问：我还是不太了解“打错字”（typo）为什么会把average变成未定义的变量。究竟是怎么一回事？

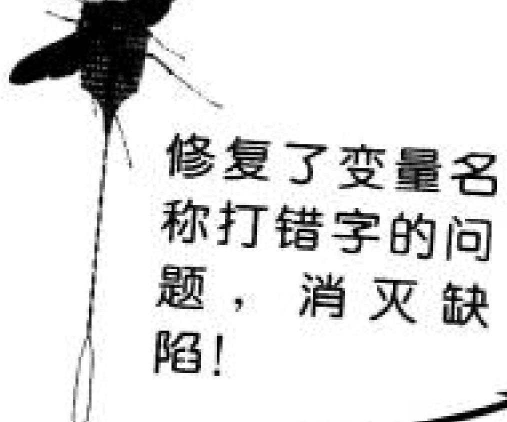
答：虽然创建并初始化的变量的名称是average，JavaScript并无法因为名称**几乎一样**，就把averag和average联想在一起。对JavaScript而言，变量averag也可能是shazbot或lederhosen。我的意思是说，JavaScript解释器把它视为全新的变量。既然这个新变量未被指派值，想在if语句里比较它的值，当然就是个大问题了。就像在决定看哪部电影前，就先下笔写电影评论一样。

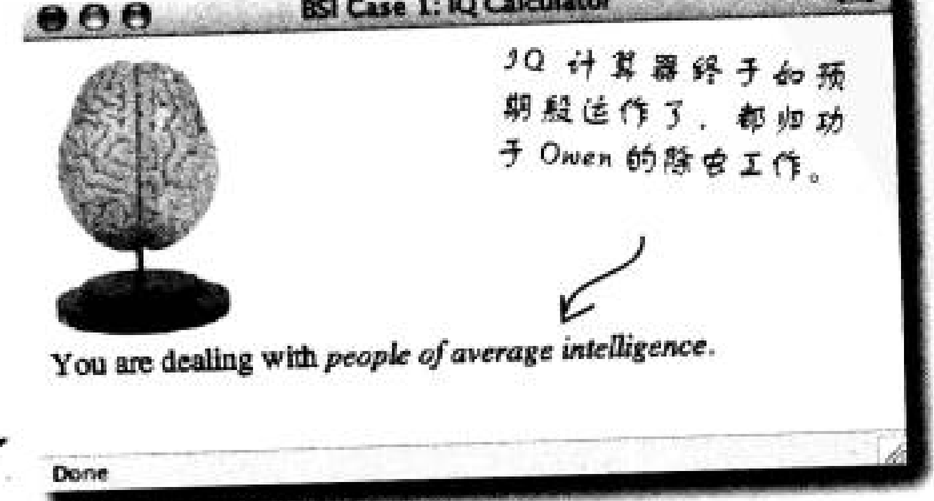
问：你不是说真的吧？我每次用文字处理软件都不知道打了多少个错字，但它都不会有事。为什么JavaScript这么敏感？

答：请先深呼吸，然后把这一段重要而且经典的名言刻入你的脑海里：“习惯就好。”我们并非撰写给其他人类看的脚本，而无论你使用何种语言编写脚本，机器完全不懂得何为“谅解”。即使只有一个字符放在错误的地方，都可能把整份脚本逼入绝境。对于JavaScript代码前后空白（例如空格〔space〕和换行〔newline〕）的相关规定还算有点弹性，但代码本身则必须非常精确。

计算智商

控制了打错字的缺陷后，IQ计算脚本现在正常运作了，它能根数组列计算平均IQ，而后表示智力等级的文字信息。Owen结束了一件案子，沐浴在完成工作的喜悦中……但喜悦能维持多久？





复习要点

- 虽然大多数浏览器都提供了错误控制台（记录 JavaScript 的错误），但并非完全准确，你不能全盘相信它们。
- 虽然浏览器通常产生不完全的错误信息，但通常都会提供寻找问题的**线索**。
- 围起代码块的大括号，是个常见的缺陷起源 —— 请小心，务必确认起始与结尾括号**成对出现**。
- 大家都常犯简单的打字错误，但不见得很容易找出来 —— 请务必检查标识符的名称。

案件：电台直播来电缺陷

Owen 还在为了结束第一个案件而庆祝，下个案件马上就出现在他的桌上。他的新案件是一个处理电台直播来电竞赛的脚本，根据来电顺序决定赢家。这份脚本应该一直计算来电人数，并在来电到达一定数量时（本例为第7通电话）宣布赢家。

抱歉，你是2号来电人。

抱歉，你是5号来电人。

Case 2

BSI Case 2: Winning Caller

Caller name: Next caller　Call

Done

……6通来电后……

BSI Case 2: Winning Caller

Caller name: Alan　Call

Done

Alan, caller number 7...today's winner!

OK

开始调查

用浏览器打开电台直播来电网页前，Owen 觉得可以先看一下代码（可至 *http://www.headfirstlabs.com/books/hfjs/* 下载），对代码结构稍微有点概念，或许会看到一些很明显的错误，至少也可以了解代码**应该如何**运作。

```
<html>
  <head>
    <title>BSI Case 2: Winning Caller</title>

    <script type="text/javascript">
      // Total number of calls
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Increment the call number
        var callNum;
        ++callNum;

        // Check for a winner
        if (callNum = winningNum) {
          alert(caller + ", caller number  + callNum + "... today's winner!");
          form.submit();
        }
        else {
          // Reset the caller field for the next caller
          var callerField = document.getElementById('caller');
          callerField.value = 'Next caller";
          callerField.focus();
          callerField.select();
        }
      }
    </script>
  </head>

  <body>
    <form name="callform" action="radiocall.php" method="POST">
      <img src="radio.png" alt="radio" />
      Caller name: <input id="caller" name="caller" type="text" />
      <input type="button" value="Call"
        onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
    </form>
  </body>
</html>
```

来电人的姓名、赢家序号（winningNum）还有 form 对象都被传入 checkWinner() 函数。

来电计数器初始化为 0。

增加来电计数器。

如果数字等于赢家序号，即提醒用户并送出表单。

如果不是赢家序号，为下个来电人重设来电人姓名域。

focus() 方法设定网页元素的输入焦点。

select() 方法选择存储在文本元素里的值。

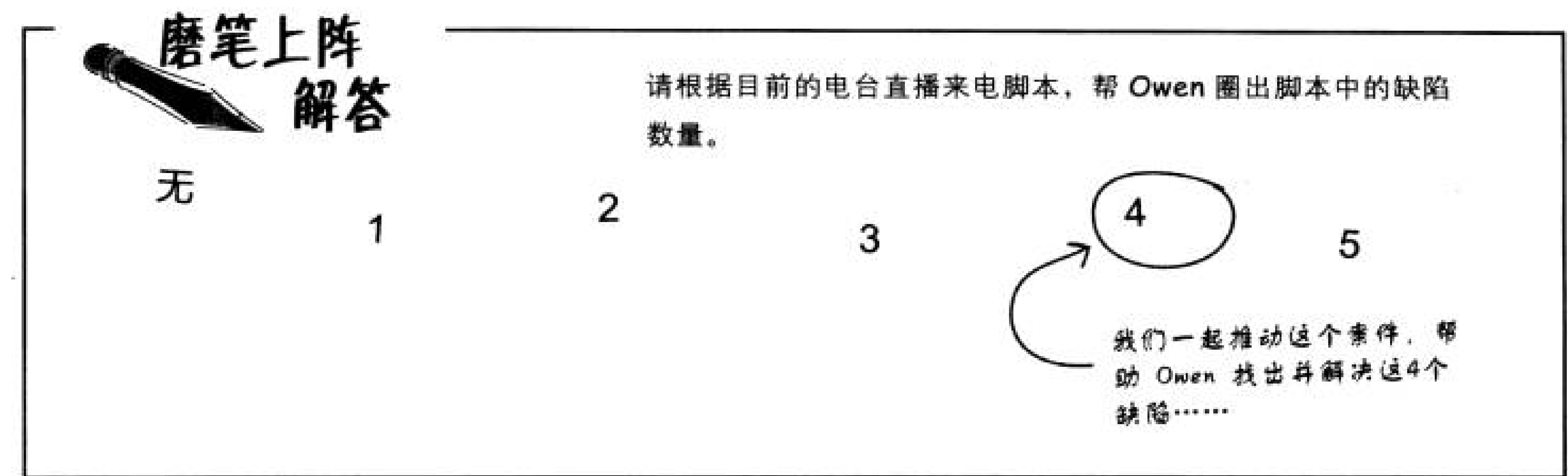
另外存储赢家的服务器脚本。

按下 Call 按钮时，调用 checkWinner() 函数。

磨笔上阵

请根据目前的电台直播来电脚本，帮 Owen 圈出脚本中的缺陷数量。

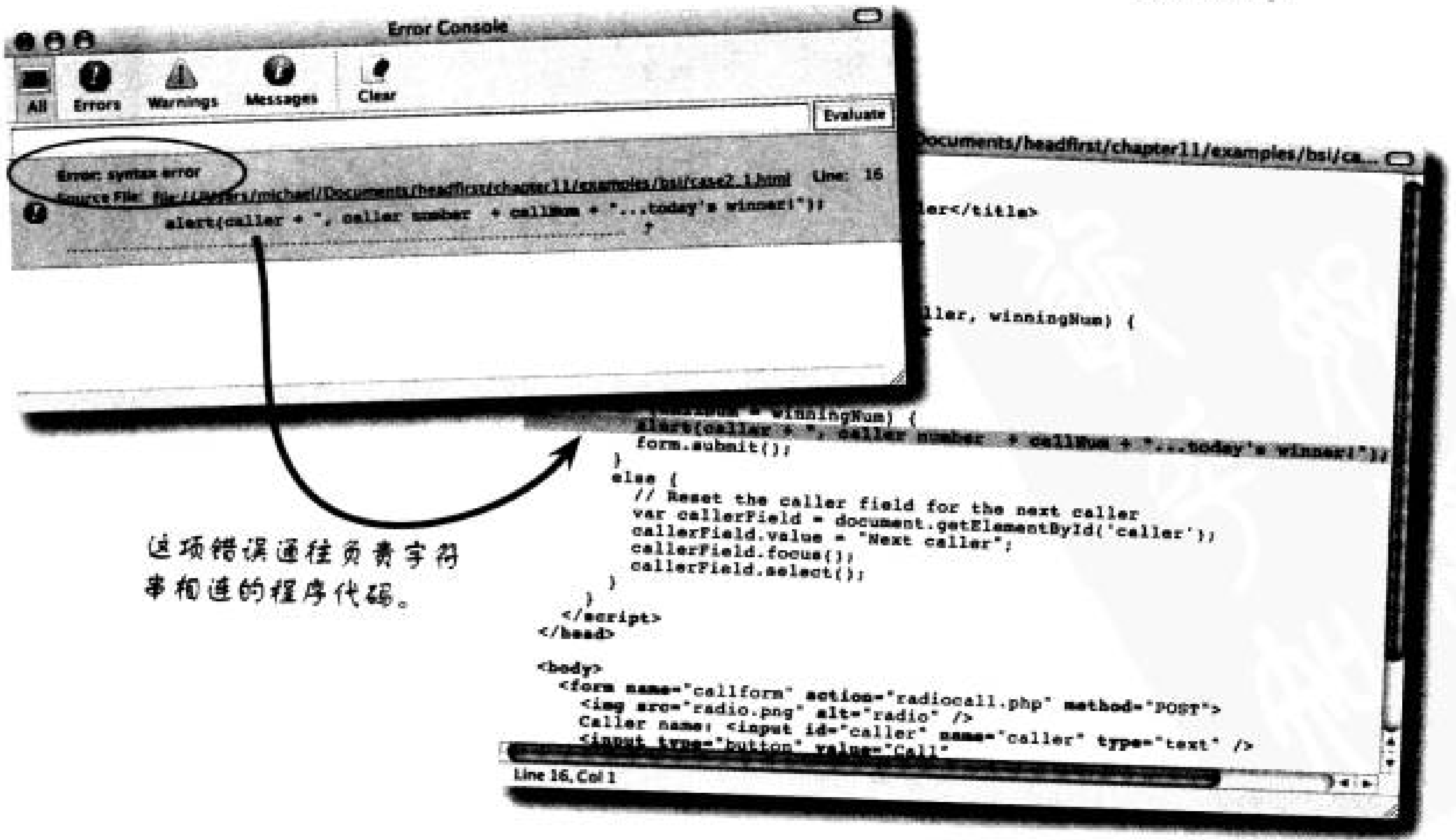
无　　1　　2　　3　　4　　5



语法验证问题（Bug #1）

对程序代码应有的运作方式有概念后，现在该打开 Firefox，看看电台直播来电脚本的实际运作状况了。与我们看过的其他错误相似，Firefox 立刻汇报了**语法错误**（syntax error），这是种违反了 JavaScript 语言规则的编程错误。

语法错误总是由浏览器秒汇报，假设开启了汇报功能。

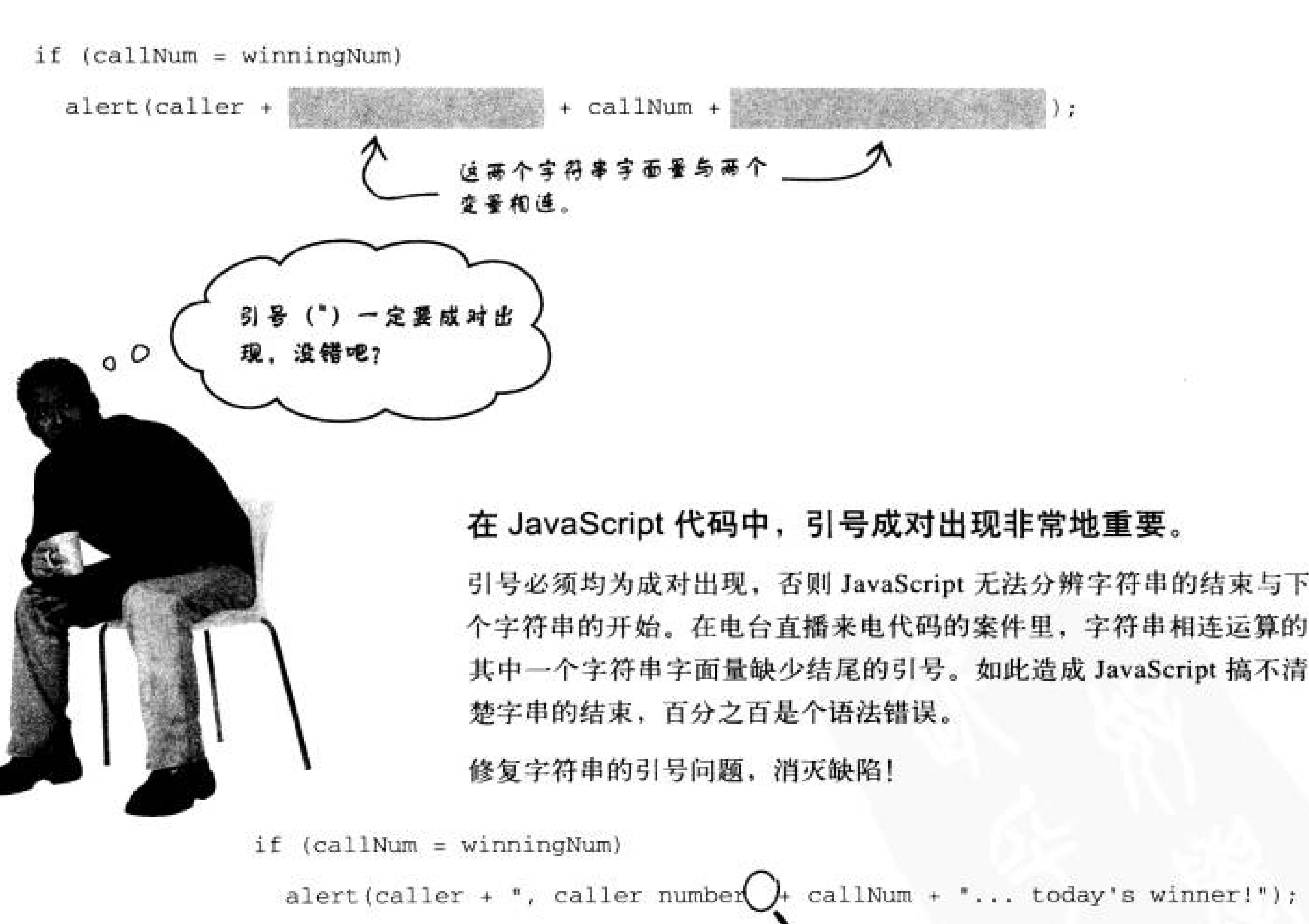


语法错误多半引发浏览器的某种通知，假如用户开启了浏览器的汇报功能。对于追踪错误，这点是个非常重要的强心剂。

小心处理字符串

Firefox 指向一行具有字符串相连的代码，这个线索告诉我们小心检查这一行。代码中调用 `alert()` 函数，其中有好几个字符串，并与 `caller`、`callNum` 变量相连。

```
if (callNum = winningNum)
  alert(caller + ░░░░░░░░ + callNum + ░░░░░░░░░░);
```



在 JavaScript 代码中，引号成对出现非常地重要。

引号必须均为成对出现，否则 JavaScript 无法分辨字符串的结束与下个字符串的开始。在电台直播来电代码的案件里，字符串相连运算的其中一个字符串字面量缺少结尾的引号。如此造成 JavaScript 搞不清楚字串的结束，百分之百是个语法错误。

修复字符串的引号问题，消灭缺陷！

```
if (callNum = winningNum)
  alert(caller + ", caller number + callNum + "... today's winner!");
```

```
if (callNum = winningNum)
  alert(caller + ", caller number " + callNum + "... today's winner!");
```

修复字符串的引号问题，消灭缺陷！

引号、撇号，还有一致性

缺少的引号只是字符串中的引号相关错误的一半。因为JavaScript与HTML均支持引号（"）和撇号（'），但分别用于包围字符串（JavaScript）和属性（HTML），所以在混用两者时，务必注意一致性。

引号用于围起所有HTML属性。

```
<input type="button" value="Call"
  onclick="checkwinner(this.form, document.getElementById('caller').value, 7)" />
```

撇号用于围起属性里的JavaScript字串。

HTML属性使用引号，属性里的JavaScript字符串则使用撇号，这个方法运作得很完美，而且是个好主意。不过，也可以在HTML中反转两个符号的用途，如下所示：

现在的XHTML网页标准不允许使用撇号围起属性。

```
<input type='button' value='Call'
  onclick='checkwinner(this.form, document.getElementById("caller").value, 7)' />
```

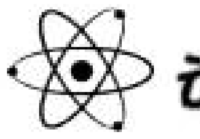
这里的重点在于确定引号都用于甲语言，撇号都用于乙语言。既然现在的HTML、XHTML都要求以引号围起属性，所以属性用引号，属性里的JavaScript字符串则用撇号，也很合理。

但是，在你突然特别需要引号与撇号，偏偏又已把其中一项用于围起字符串时（又称字符串**边界**〔boundary〕），问题就出现了。请看下例：

```
alert('It's so exciting!');
```

这段代码能运作吗？

在HTML属性中用到JavaScript字符串时，引号与撇号应该交替运用。



动动脑

当你的字符串特别需要引号或撇号字符，但已经用相同字符围起字符串时，会发生什么事？

当引号不是引号，使用转义符

在字符串里使用引号与撇号字符，而不是把它们解释为字符串边界，这是个常见错误。所以我们刚才看到的 alert 框代码具有语法错误，JavaScript 解释器无法分辨出哪个撇号是字符串边界，哪个又该当成撇号字符本身。幸好，有个声明字符是个**真正字符**的便利方式 —— **转义符**（escape character），加上反斜线（\）从而把它后面的字符实际视为字符原本的意义。

```
alert('It\'s so exciting!');
```

转义符用在字符串中表示实际字符。

转换撇号为字面量后，JavaScript 即可毫无疑虑地知道我们想把撇号放在字符串里，而不是用它区隔字符的边界。当然，我们可以改变这里的字符串边界为引号，以免用到转义符。

```
alert("It's so exciting!");
```

不需要转义了。

一样可行，但如果遇到下例的情况呢？

```
alert("They said, "you've won!"");
```

字符串包含引号与撇号字面量，所以只能利用转义符。在这种情境中，转换字符的意义通常比较安全，虽然此例的撇号不需转义。

```
alert("They said, \"you\'ve won!\"");
```

不需转义，但仍是个好主意。

修正下列代码片段中的引号与撇号，于需要时使用转义符。

```
var message = 'Hey, she's the winner!';
```

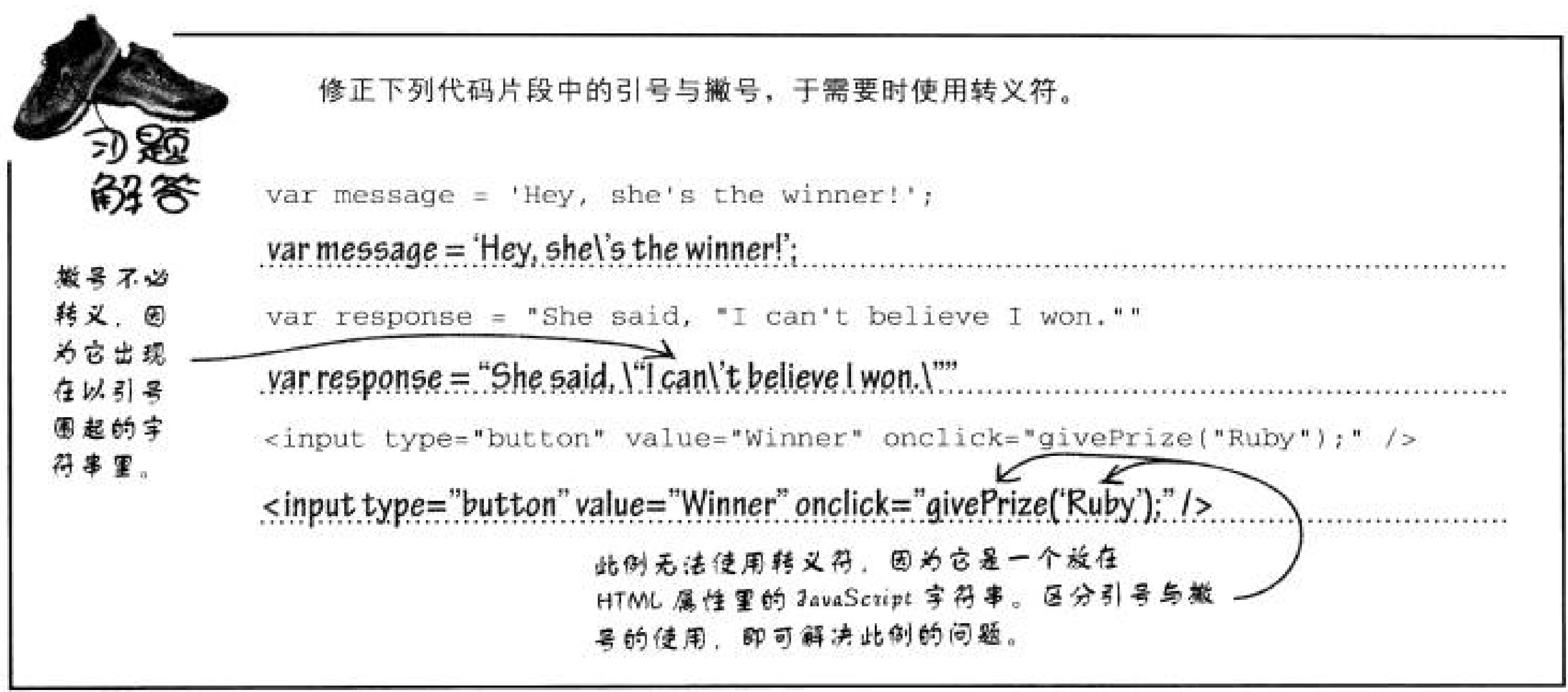
....

```
var response = "She said, "I can't believe I won.""
```

....

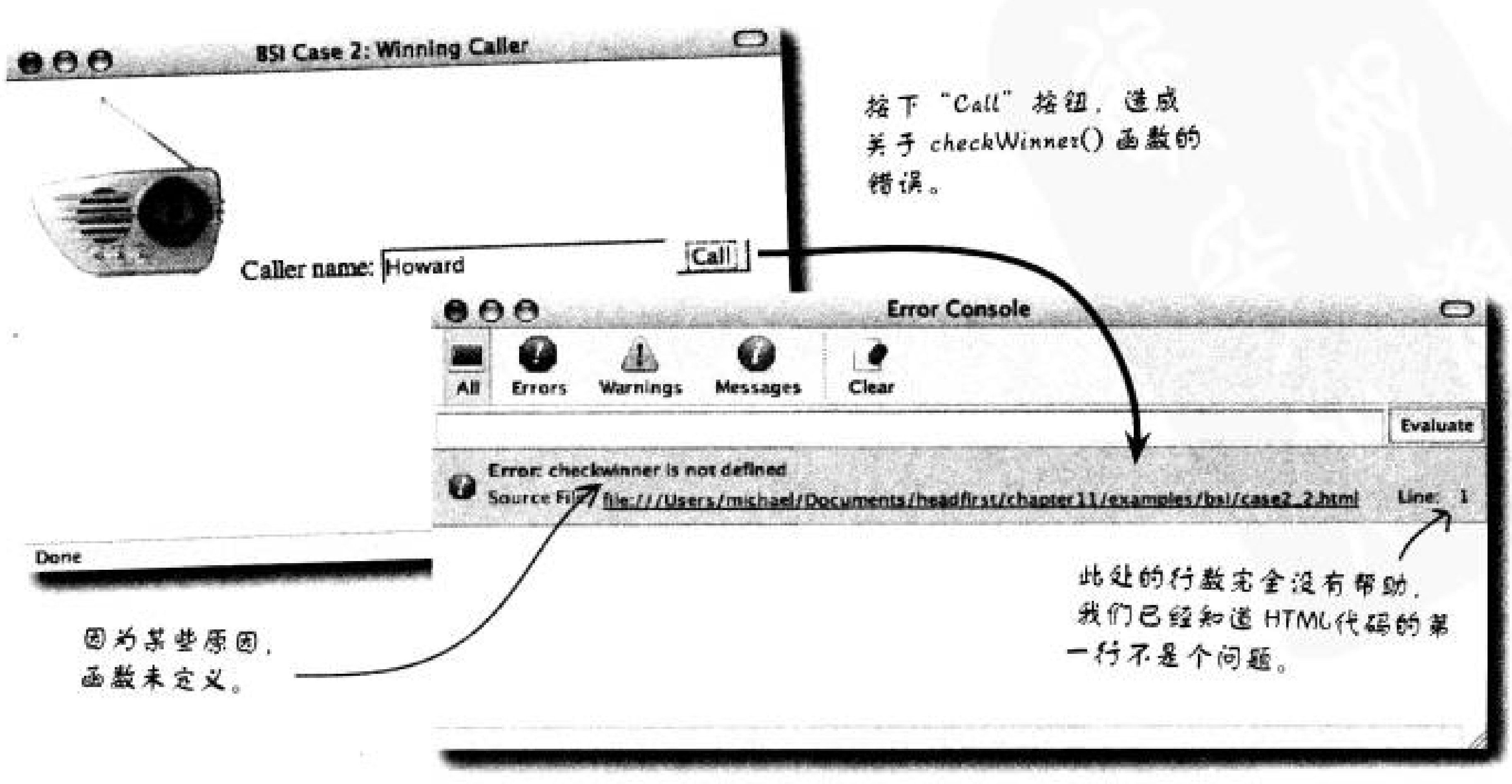
```
<input type="button" value="Winner" onclick="givePrize("Ruby");" />
```

....



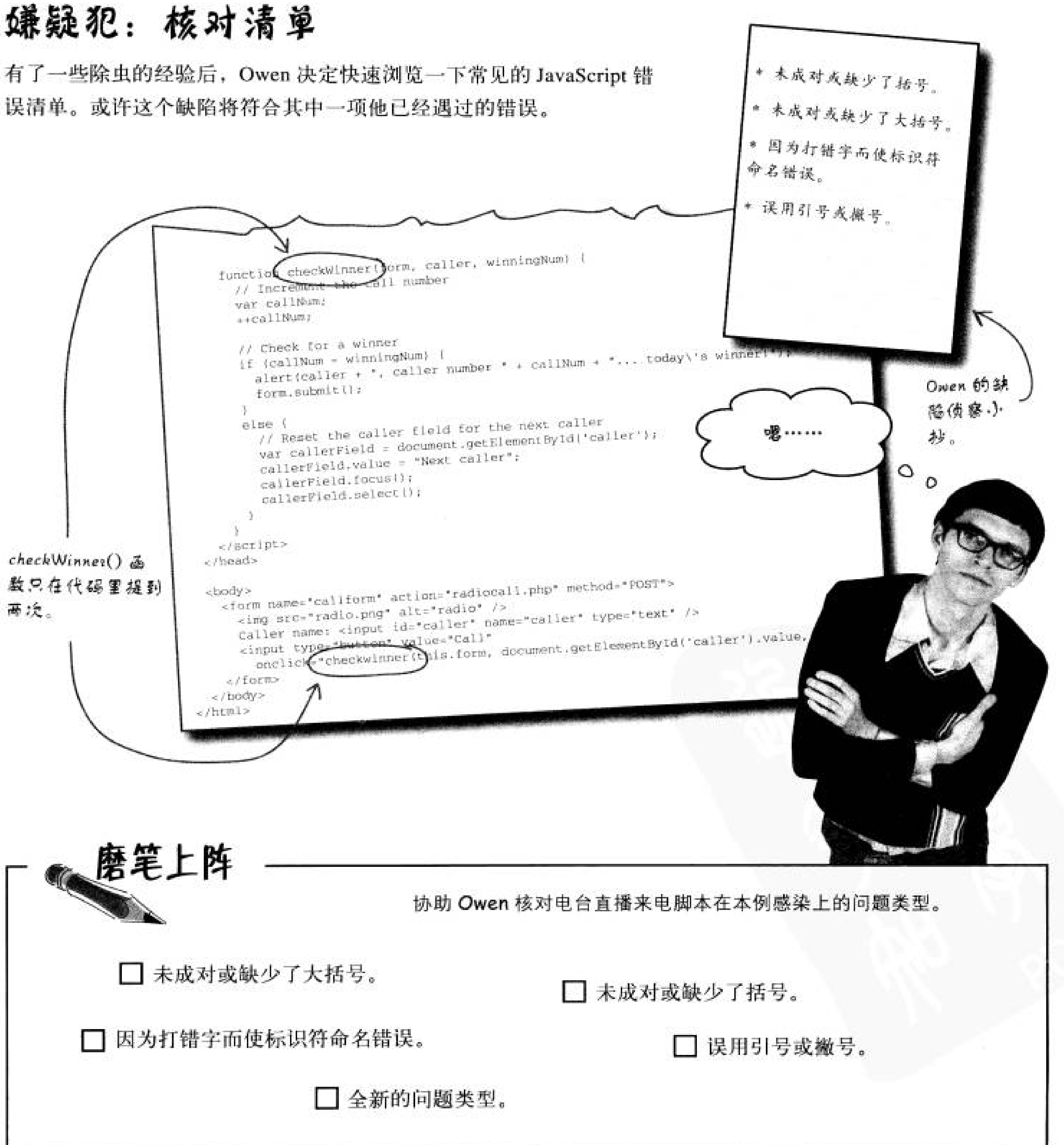
未定义不只是变量的专利（Bug #2）

解决了一只缺陷，但 Owen 的工作尚未结束。电台直播来电脚本刚开始没有错误传出，但只要输入来电人姓名、按下“Call”按钮后，立刻出现另一个问题。这个问题似乎与 checkWinner() 函数有关。



嫌疑犯：核对清单

有了一些除虫的经验后，Owen 决定快速浏览一下常见的 JavaScript 错误清单。或许这个缺陷将符合其中一项他已经遇过的错误。



磨笔上阵

协助 Owen 核对电台直播来电脚本在本例感染上的问题类型。

☐ 未成对或缺少了大括号。

☐ 未成对或缺少了括号。

☐ 因为打错字而使标识符命名错误。

☐ 误用引号或撇号。

☐ 全新的问题类型。

磨笔上阵解答

协助 Owen 核对电台直播来电脚本在本例感染上的问题类型。

- ☐ 未成对或缺少了大括号。
- ☐ 未成对或缺少了括号。
- ☑ 因为打错字而使标识符命名错误。
- ☐ 误用引号或撇号。
- ☐ 全新的问题类型。

checkWinner() 函数的调用不小心使用了小写 w，变成 checkwinner()。打字错误造成 JavaScript 以为 checkwinner() 是个完全不同的函数，而且尚未定义。

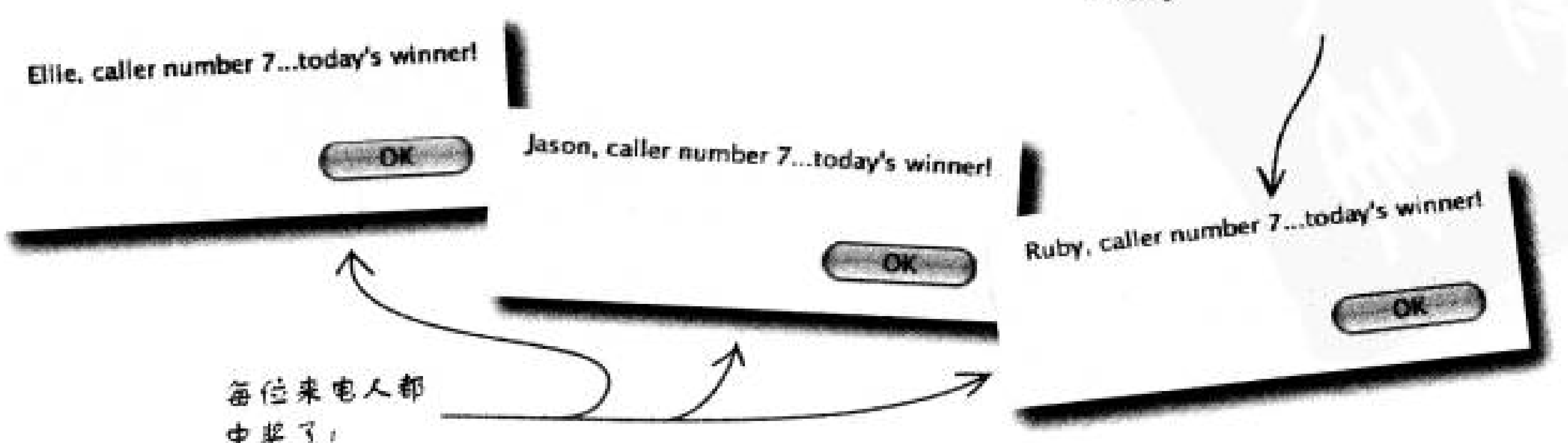
把函数的调用修改为大写的 W，checkWinner()，即可修正这个问题。

修正函数名称的打字错误，消灭缺陷。

```
<input type="button" value="Call"
  onclick="checkWinner(this.form, document.getElementById('caller').value, 7)" />
```

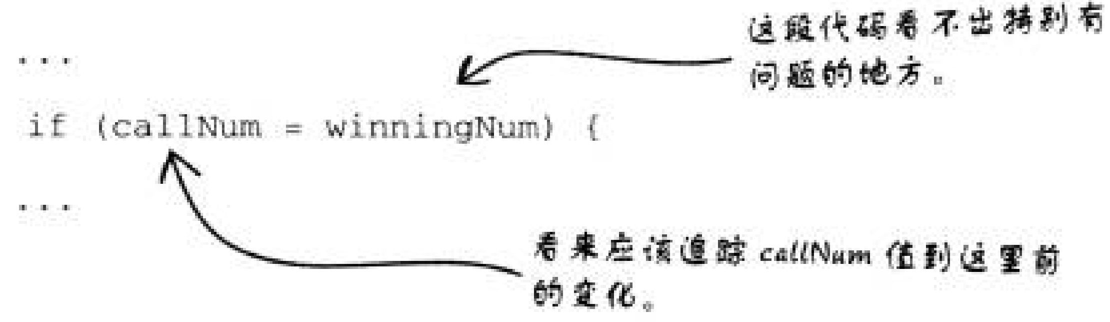
每个人都是赢家（Bug #3）

处理了麻烦的“未定义的”打字错误缺陷，电台直播来电脚本仍然在触发错误。好消息是浏览器不再汇报任何问题，但每个来电人现在都是赢家 —— 脚本甚至对来电序号不对的人宣告正确的序号。如果 Owen 不想出解决办法，电台就得发出很多奖项！



用 alert 框调试

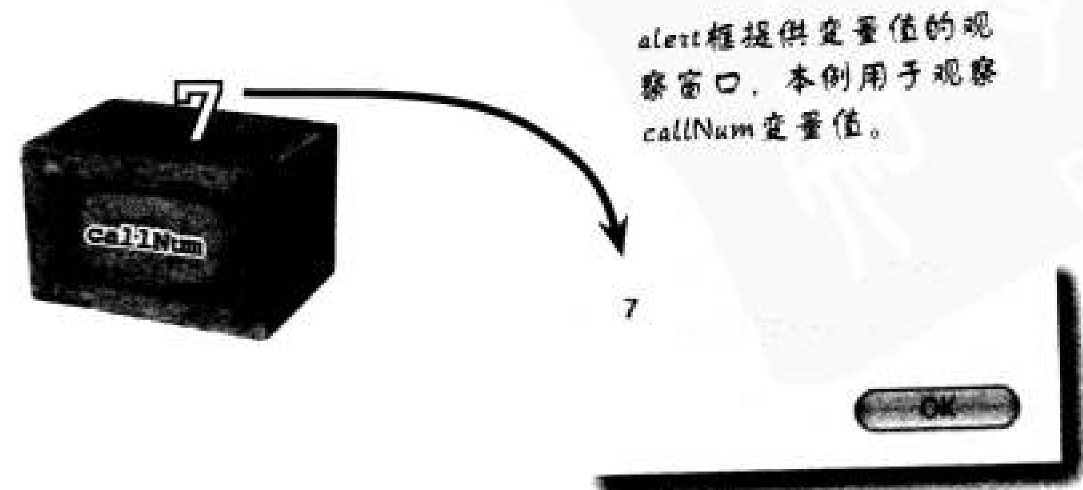
我们知道获奖序号的处理，是以变量callNum和checkWinner()函数的自变量winningNum作比较。但这段程序代码显然没问题……我们需要更仔细检查callNum发生什么变化的方式。



alert框可作为快速查看变量值的便利工具。

alert 框可作为调试观察窗。

结果，alert 框不只可对终端用户（end user）呈现弹出信息，它们对 JavaScript的开发端也很有用，可作为变量的暂时观察窗口。不只如此，alert 框也可确认某段代码的调用是否与预期相符。在本例中，使用 alert 框单纯为了检查callNum的变量值。



以 alert 观察变量

watch（观察）是调试时的术语，表示在运行程序时持续观察一个变量。alert 框提供了虽非持续、但仍非常有用的初级观察管道。alert 能用于观察JavaScript 代码的任何一处变量有问题的地方。

```
alert(callNum);
if (callNum = winningNum) {
  alert(caller + ", caller number " + callNum + "... today\'s winner!");
  form.submit();
}
```

这里有问题……callNum 应该设定为目前来电的序号。

NaN

OK

Owen发现，电台直播来电脚本不知何故，把`callNum`与`winningNum`视为相等，虽然在`if`语句前的`callNum`显然是 `NaN`。虽然 `callNum` 为何是 `NaN` 已经够伤脑筋了，Owen 仍决定再把 alert 移入 `if` 语句里，检查是否有什么变化。

```
if (callNum = winningNum) {
  alert(callNum);
  alert(caller + ", caller number " + callNum + "... today\'s winner!");
  form.submit();
}
```

7

OK

经过 if 测试条件后，callNum显示为7。

嘿！我想我找到问题了！

动动脑

从 alert 观察到 `callNum` 神奇地将自己设定为 7，而且就发生在一行代码内，你认为造成这个缺陷的原因是什么？Owen 又发现了什么问题呢？

Alert 的调试真情指南

本周主题：

Alert 浅谈它对缺陷的鄙视

Heart First：我必须承认，你的评价很复杂。有人说你很烦人，但我也听说你是调试专家的好朋友。你愿意为我们说明哪个角色才是真正的你吗？

Alert：说我烦的那些人真是疯了。我是好人。我也很单纯 —— 你给我要呈现的信息，“变！”我弹上台为你呈现信息。就是这样。哪有什么问题呢？

Heart First：我想“弹”出来这部分有点争议。因为那些奇怪的广告窗口总是一直弹出来烦人，弹出式窗口（pop-up）最近的名声实在有点差。

Alert：噢！了解……嗯，我了解你说的烦人之处了。可是你不能因为愚蠢的木匠不懂得善用铁槌而怪罪铁槌啊！知道我的意思吗？

Heart First：所以说，我所听到的一切恶劣传言，都是因为你被误用了？

Alert：就是这样。刚刚也解释了，我只是照章行事。你告诉我弹出一堆广告窗口，我就照做。并不是说我喜欢这样做哦！只是决定权真的不在我身上。咦，我还以为今天要讨论我对调试界的贡献。

Heart First：对耶，真不好意思。我听说了你的不少事迹，协助 JavaScript 开发人员追踪代码的缺陷。你是怎么办到的呢？

Alert：非常简单，真的。假设有个发狂的变量被设置为完全没有意义的值。程序设计师也发狂了、咖啡因过量……相信你也听过这种状况，他们急需观察变量在程序代码不同处的状态，了解变量如何改变。所以，他们要求我跳出来显示变量。

Heart First：但你如何在程序代码不同处显示变量改变的值呢？听起来很难。

Alert：一点都不难。你只要多调用我几次，每次调用都放在程序代码不同处即可。

Heart First：原来如此。你身为调试协助工具的时候，遇过任何问题吗？

Alert：确实遇过，如果有段程序代码运行了很多次，例如在循环里，我就不太适合弹出它的调试信息了。

Heart First：为什么呢？

Alert：我是个弹出窗口，所以必须按下“OK”，我才会关掉。如果我会弹出很多次，表示鼠标要按很多次。

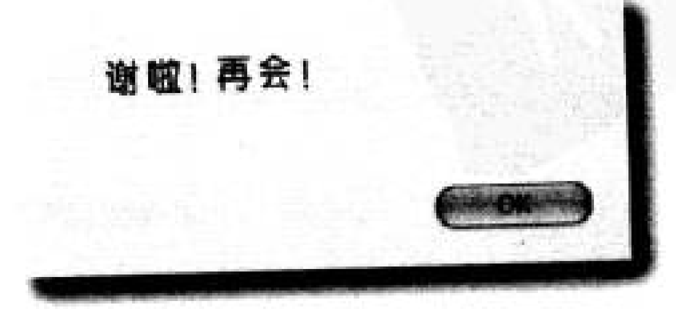
Heart First：有道理。我还听说，就算不是要注意数据的状况，你也很方便好用。

Alert：对啊。有些时候不太确定是否调用或何时调用了程序代码，在一段代码中叫我出来，你将可知道程序代码是否真的被调用了。

Heart First：在所有这些调试状况中，你是在说你只是暂时存在吗？

Alert：当然啰！而且我一点都不介意。毕竟我另有其他终生职责 —— 调试只是一点小小的社会服务而已。

Heart First：真感谢你花时间讲解你在调试上扮演的角色。希望下次还能再看见你。



逻辑差虽然合法却有缺陷

Owen把焦点锁定在**逻辑错误**（logic error），也就是合乎JavaScript语法规则，但完全不符合程序意图的错误。本例中，误以 = 取代 ==，表示winningNum的值被**指派**（assign）给 callNum，而不是比较两者的值。你觉得吹毛求疵吗？是的。但它就是非常有问题。

之前"乍看"之下没问题的代码，结果有只很微小、难以察觉的缺陷。

这项错误的真正问题，在于它不像语法错误般会让浏览器找到问题。JavaScript 解释器不会抱怨，因为指派（assignment）的行为"返回"了被指派的值，在winningNum的例子中，即自动于if测试条件中解释为true（不为零）。换句话说，这段范例代码完全合格，虽然它的作用跟想象中完全不一样。

这么说，逻辑错误甚至不会出现在浏览器的错误控制台吗？

逻辑错误就像灭蚊灯抓不到的苍蝇。

逻辑错误如此难以处理的原因，就是在于它们很少能像语法错误一样显示出来。虽然看到浏览器列出的脚本错误总让人丧失自信，但这些还是不幸中的大幸，它们是已经检测好、等你解决的错误。逻辑错误并未违反任何JavaScript的语法规则，所以通常较难检测。

另一方面，逻辑错误有时会在脚本运行时造成脚本语法错误。例如说逻辑错误在于变量并未初始化，"未定义"错误将于脚本试图引用变量时出现。所以，有些逻辑错误还是能省下追捕缺陷的心力。

复习要点

- 语法错误关系到程序代码违反了 JavaScript 的语言规则，通常能被 JavaScript 解释器捕捉。
- 字符串必须小心地以成对引号或撇号围起。
- 在 HTML 事件处理器的属性里包含 JavaScript 代码时，应小心混用引号与撇号。
- 对于“在脚本中观察变量”，alert框提供了初级但好用的选项。
- 常见的错误：不小心把测试条件里的 == 写成 =。

没有蠢问题

问：转义符只用于转换引号与撇号的意义吗？

答：不止如此，JavaScript 还支持数个转义符，例如使用\t 可在字符串里插入 tab 字符；相同的道理，转义符 \n 可表示newline 字符。字面量版的反斜线符号则以 \\ 转义。

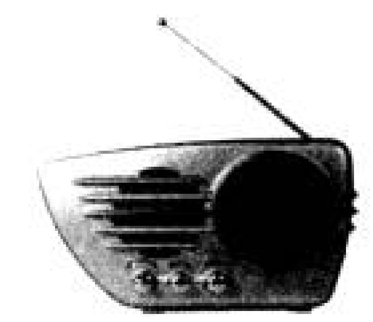
为 alert 框里的文字加上格式时，则是转义符能被有效运用的一个场合。你可以使用\t与\n，tab负责文字的对齐，newline则控制文本、段落的换行。

问：转义符用于 HTML 属性时，为什么会有限制呢？

答：限制是关于“HTML属性并未理会JavaScript规则”的事实，至少不太尊重用于绑定属性值的字符。虽然可以把HTML属性中的JavaScript字符串里的字符转义，但它不能是围起属性值的相同字符。

听起来很乱七八糟吗？这样说吧，HTML 单纯地把属性视为“必定出现于引号或撇号中的值”。就这么简单。无论你用哪一个符号当作边界字符，HTML 均假设它遇到的下个相同字符，就是结束属性值的成对结尾字符。因为 HTML 处理属性值时不会在意 JavaScript 的转义符。

但转义符在 HTML 属性中还是行得通，只要用于围起属性的字符并未与之冲突。例如在事件处理器的状况中，属性值最后仍将被解释为 JavaScript 代码。



问：有没有在调试过程中提供详细控制的精美调试器？

答：有一些，而且稍微研究、考虑试用现成的调试器，也是不错的主意。不过，好的编码习惯加上本章学到的调试技巧，将在未来的长路上与你相伴，直到协助各位创建无错的脚本为止。

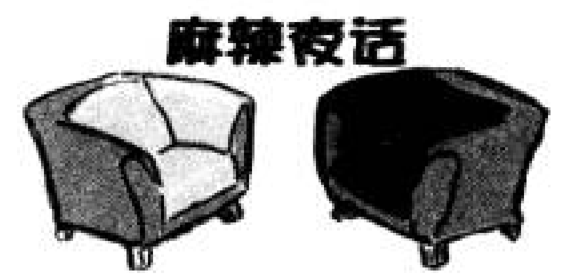
问：当 JavaScript 试图引用未定义的变量或函数时，究竟发生何事？

答：未定义变量，是个未被创建或虽已创建却未被指派任何事物的变量。无论哪种状况，变量中存储的值都是“未知的”，更精确地说，都是“未定义的”。凡是企图读取该值并行使有意义的行为都变得完全无意义，JavaScript因而产生错误。

相似的情况发生在函数受到调用，但JavaScript解释器无法根据名称找到函数的状况。函数未定义，表示调用它也没意义 —— 无物可调用。于是，JavaScript认为这是个错误，因为无法有意义地运行程序代码。

问：`callNum`变量在被送入测试条件前变成了`NaN`，这又是怎么一回事？

答：现在还不知道。不过这说明了一件事，脚本中尚有错误。所以继续追踪缺陷仍是项重要的任务……



今晚主题：语法错误与逻辑错误分享他们对破烂脚本编程技巧的热爱

语法错误

嘿，我听过你耶。据说你很狡猾。不过我在想……你是不是真的和我一样享受写得非常糟的脚本？

逻辑错误

那当然。表面好像正常，但离开视线就出现各种奇怪问题的脚本是我的最爱啊！

语法错误

我不这么认为。我喜欢那种一眼就能看到麻烦的脚本，那是我的专长。在脚本里面放几个我，保证浏览器尖叫个不停。

逻辑错误

可是那样有什么好玩呢？大家都知道偷袭更有效率。先引诱他们觉得一切都没问题，然后慢慢地开始在各处放出一些小问题。如果安排得够巧妙，大家甚至会以为浏览器可能运作不正常了。

语法错误

我不太喜欢你那种扭曲的观点，但重点是你仍然允许脚本运行。我不爱这一套，我偏好半路挡下它的运行。

逻辑错误

你讲到重点了。真可惜我一直找不出像你一样停止脚本、彰显自己的方式。或者来点更好的……让整个浏览器失事爆炸。这样很酷吧！

语法错误

是啊，可惜我们都受限于损害能造成的范围。没错，能够搅乱网页，阻止它的正常运作是很好玩，但我讨厌不能访问其他事物的限制。唉……如果我能玩弄充满重要数据的硬盘，该有多好啊？

逻辑错误

噢，一定会好玩到不行的！你确定我们连一个切入的角度都没有吗？

语法错误

没有……JavaScript解释器的安全锁非常紧。

逻辑错误

好吧，反正还是有不少好玩的事。我讲过关于 = 与 == 的小魔术吗？

没说过，魔术怎么变的？

很好玩喔。程序设计师本来想输入 ==，比较两个值，但他们错手输入成 =，结果变成值的指派。真好笑，他们花了几个小时，却看不穿这个小把戏。而且 JavaScript 解释器也不灵光了，因为这样的代码在技术上仍然合法。

那你要怎么脱身呢？

我有很多这类把戏。游走在合法边缘却又造成麻烦，是项非常精致的艺术啊。

我也有个不错的类似经历，就是大家忘记为每段 JavaScript 语句加上分号的情况。解释器会让只有单行的语句通过考验，但总有一天，某个年轻气盛的设计师会试着“最佳化”代码，把语句合并成一行，此时就是我现身的时机。这招永远不会失败 —— 我每次都笑得满地打滚！

讲到这个，我又想到一个不错的故事。我喜欢有人在撰写函数后才决定修改函数自变量。这招也不曾失败 —— 他们忘记修改所有对函数的调用，这些调用应该都要更新为使用新的自变量。如果一切顺利，解释器不会注意这件事，他们则因为坏掉的自变量而得到意外结果。

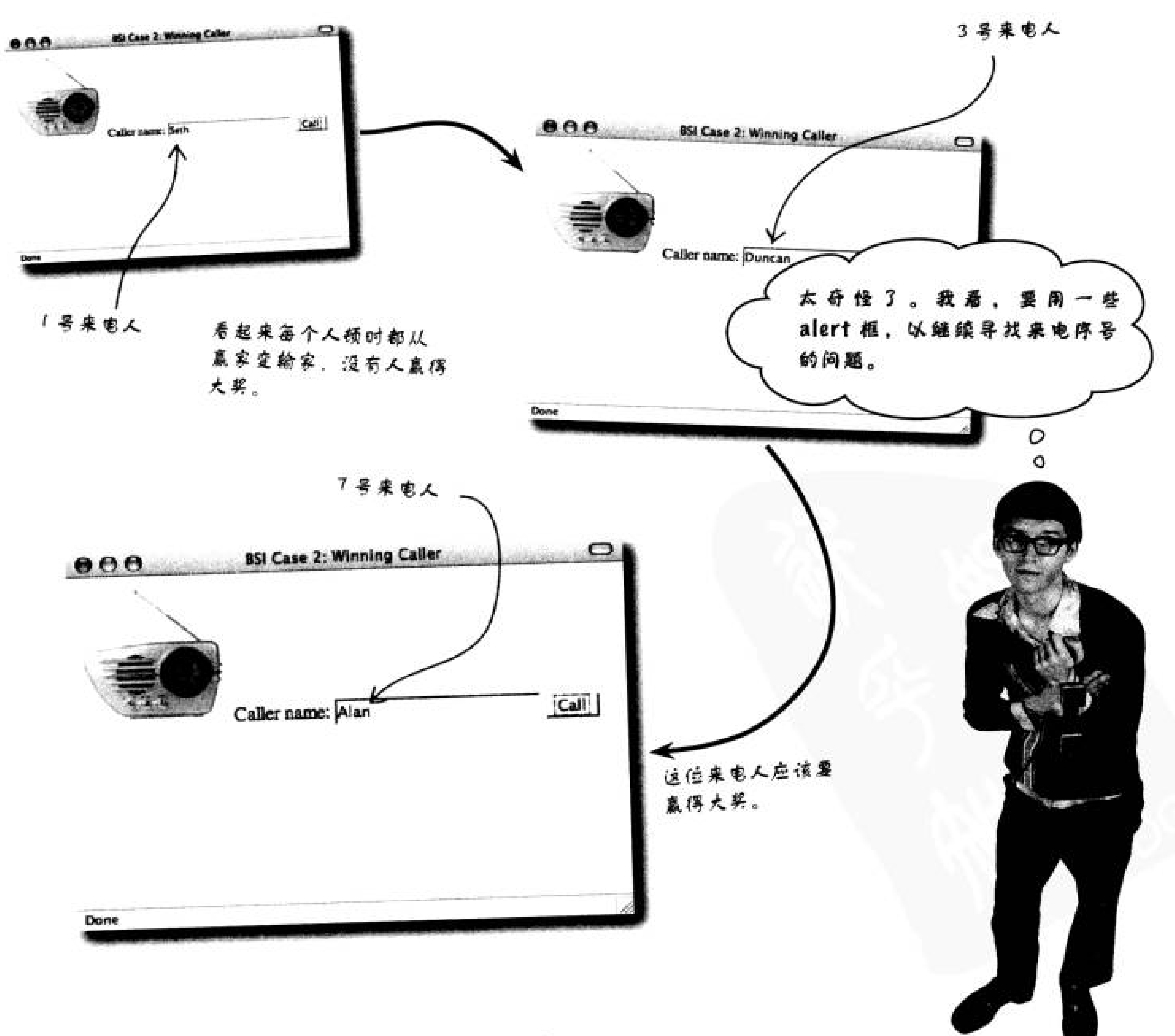
这个我喜欢，如果解释器注意不到，表示我能突然现身吓他们一跳。我看，我们不如联手出击吧？应该可以造成更多损害哦！

这就动身吧！

赞成。马上动手吧！

每个人都是输家！（Bug #4）

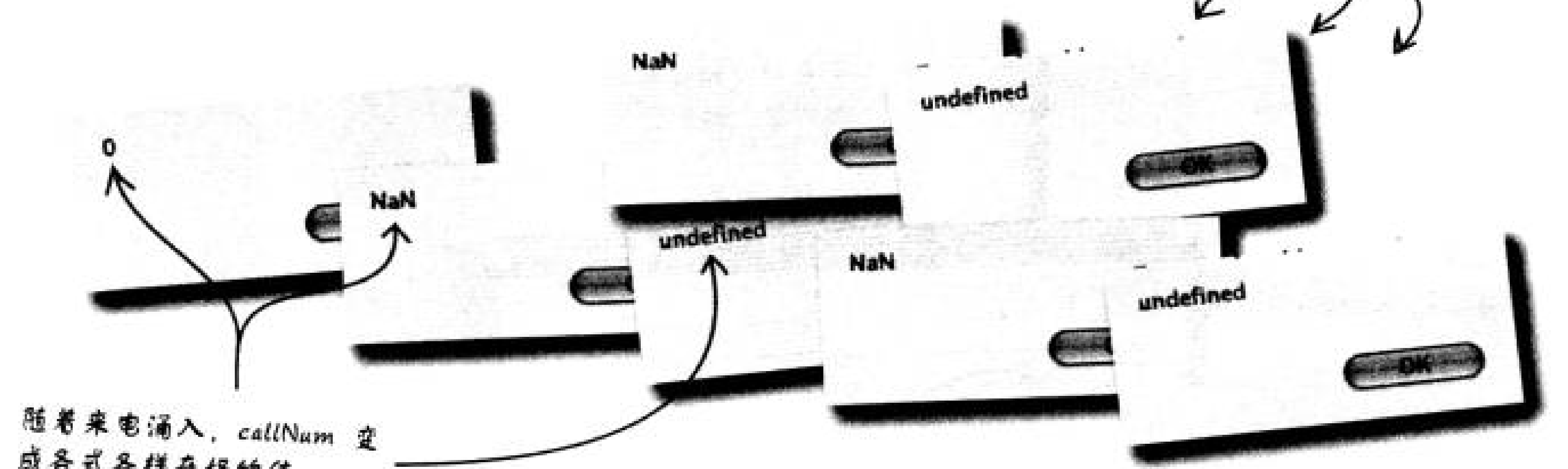
Owen 渐渐了解，除虫的工作不如他原本所想得那么简单。修好 if 语句的逻辑错误后，脚本变成不曾宣布赢家。我们从每个人都是赢家的天堂，掉到每个人都是输家的地狱。如果 Owen 不赶快把这只缺陷解决掉，有些人的自尊心想必会受影响。



被烦人的alert淹没

Owen 尝试使用 alert 观察变量 `callNum`，并试着找出问题根源。不过遇到一个问题，在通向第7名来电人的路上，alert 框一直跳出来挡道，真是烦人。他在程序代码的不同部分安装了几个 alert 框，却被过多无用的弹跳窗口搞到精疲力竭，根本不知从何着手。

以 alert 观察的缺点，就是它们若用在重复性质的代码时，可能变得很烦人。



加果能够观察变量，又不用面对这些弹出窗口，不就像是美梦成真一般吗…………

浏览器的调试控制台能帮忙

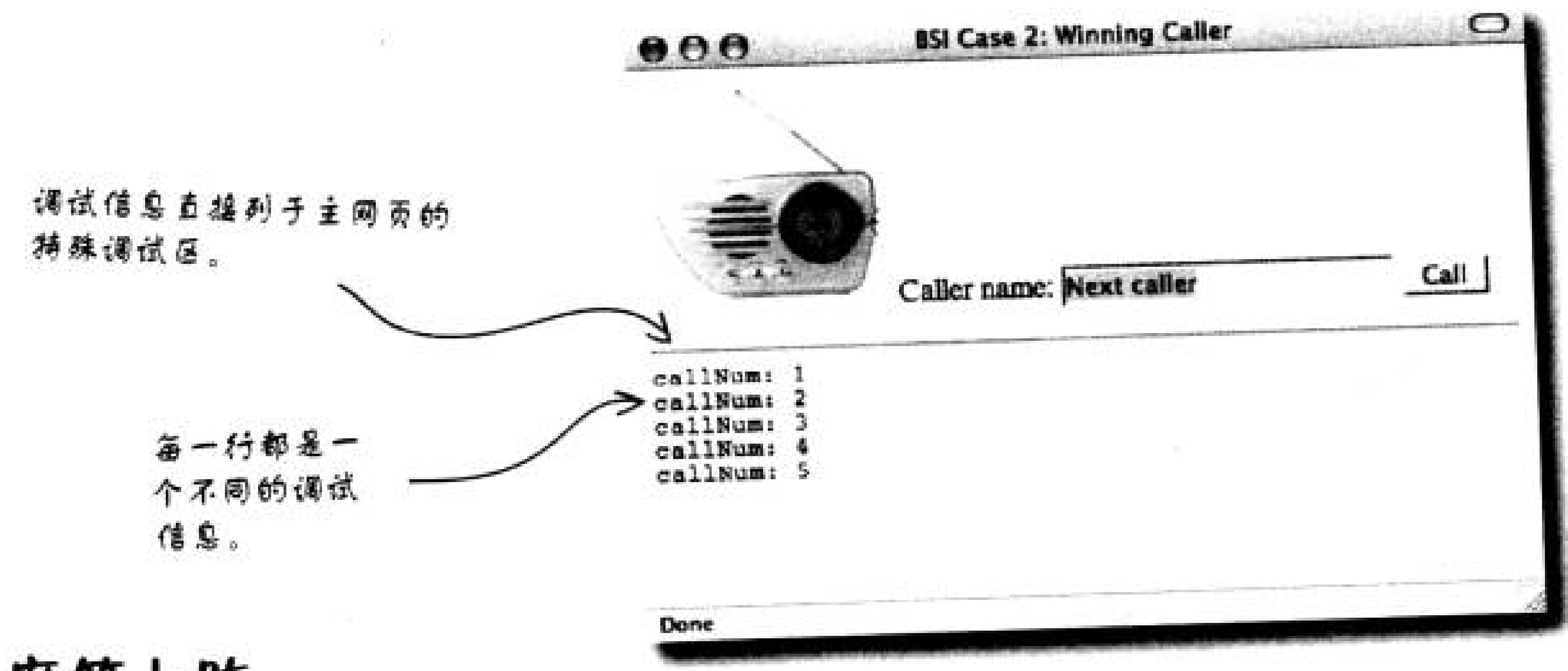
大多数浏览器都有调试用的控制台，真正的**错误控制台**，负责列出脚本里出现的错误。错误控制台非常有助于找出脚本里的问题，而且多半都能诊断出发生的问题。Firefox 拥有特别出色的错误控制台。

观察变量时，错误控制台并非真能帮忙。

或许现成的错误控制台真的很便利，但它们对观察变量没有实质帮助。不过有个好消息 —— 创建你自己的错误控制台作为观察窗口，仍是在我们的能力范围之内。

自定义一个调试控制台

自定义一个调试控制台的想法刚开始听起来或许很吓人，但它其实只需依照要求呈现文字。关键在于控制台必须直接在网页上列出调试信息，而非使用 alert 框。一个分离的弹出窗口仍然可用，只要用户无需一直按“OK”按钮。但直接在网页上呈现调试信息比较简单，而且一样有效率。

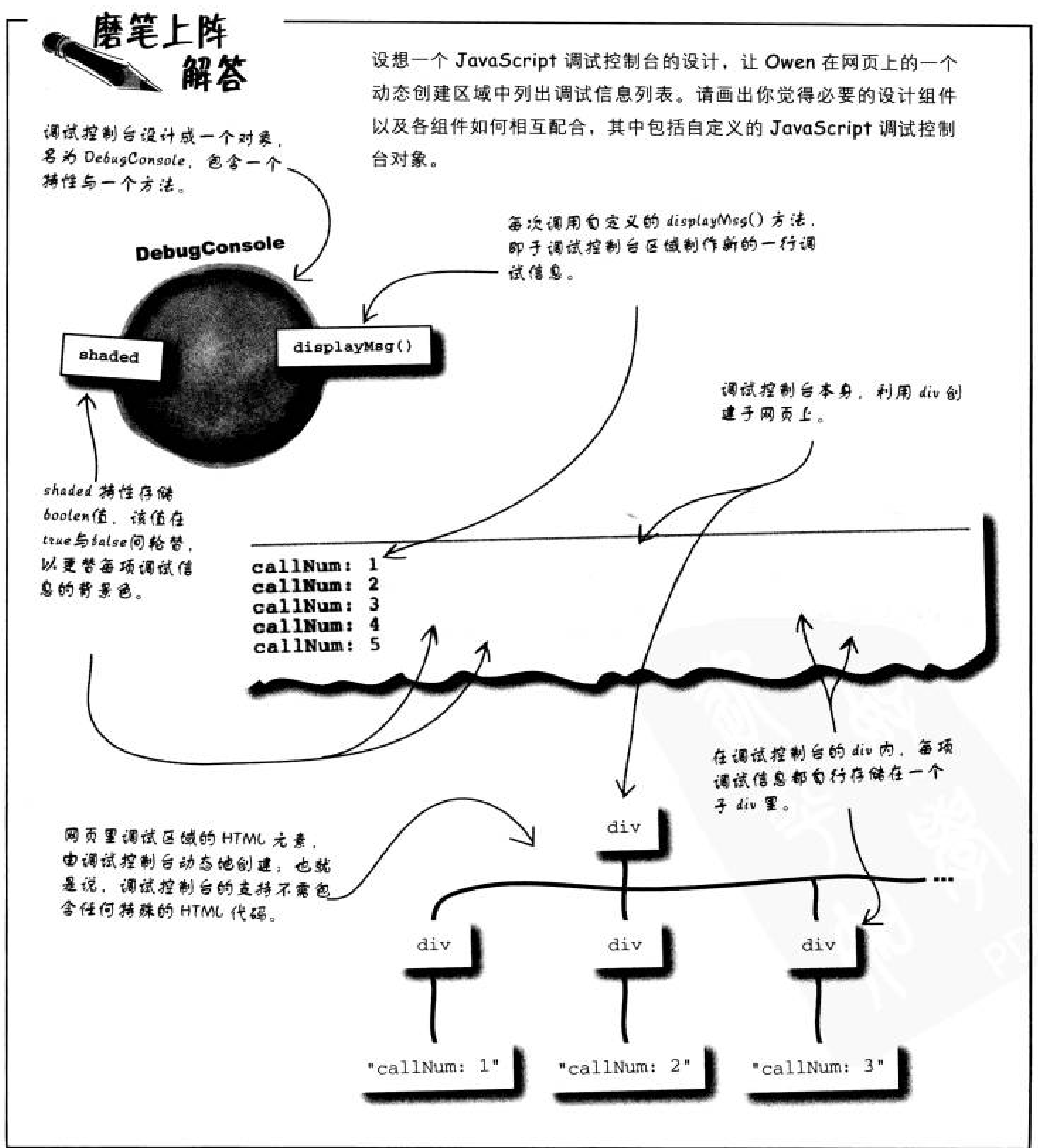


磨笔上阵

设想一个 JavaScript 调试控制台的设计，让 Owen 在网页上的一个动态创建区域中列出调试信息列表。请画出你觉得必要的设计组件以及各组件如何相互配合，其中包括自定义的 JavaScript 调试控制台对象。

磨笔上阵解答

设想一个 JavaScript 调试控制台的设计，让 Owen 在网页上的一个动态创建区域中列出调试信息列表。请画出你觉得必要的设计组件以及各组件如何相互配合，其中包括自定义的 JavaScript 调试控制台对象。



JavaScript 冰箱磁铁

调试控制台的程序代码少了一些片段。请使用下面的磁铁填入空白的部分，完成 DebugConsole 对象的创建。

```
function DebugConsole() {
  // Create the debug console area

  var consoleElem = document.________________(________);

  consoleElem.id = "debug";
  consoleElem.style.fontFamily = "monospace";
  consoleElem.style.color = "#333333";

  document.body.______________(consoleElem);

  consoleElem.______________(document.________________("hr"));


  // Create the alternating background color property

  this.________ = false;

}

DebugConsole.prototype.displayMsg = function(msg) {
  // Create the message
  var msgElement = document.createElement("div");

  msgElement.appendChild(document.________________(msg));

  msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";

  var consoleElem = document.getElementById(__________);

  consoleElem.appendChild(______________);

  // Toggle the alternating background color property

  this.shaded = ____this.shaded;

}
```

`"div"` `msgElement` `"debug"` `!` `shaded` `appendChild` `createElement` `createTextNode`

JavaScript 冰箱磁铁解答

调试控制台的程序代码少了一些片段。请使用下面的磁铁填入空白的部分，完成 DebugConsole 对象的创建。

```
function DebugConsole() {
  // Create the debug console area
  var consoleElem = document.createElement("div");

  consoleElem.id = "debug";
  consoleElem.style.fontFamily = "monospace";
  consoleElem.style.color = "#333333";

  document.body.appendChild(consoleElem);

  consoleElem.appendChild(document.createElement("hr"));

  // Create the alternating background color property
  this.shaded = false;
}

DebugConsole.prototype.displayMsg = function(msg) {
  // Create the message
  var msgElement = document.createElement("div");

  msgElement.appendChild(document.createTextNode(msg));

  msgElement.style.backgroundColor = this.shaded ? "#EEEEEE" : "#FFFFFF";

  var consoleElem = document.getElementById("debug");

  consoleElem.appendChild(msgElement);

  // Toggle the alternating background color property

  this.shaded = !this.shaded;
}
```

调试控制台的div被"附加"到文档主体下，因此它会出现在网页的末端。

调试控制台的第一个子元素是个水平线，用于区隔调试信息与网页的其他部分。

背景颜色刚开始设为false，形成预设的白背景。

信息构成子div元素，加到调试控制台中。

每则信息的背景颜色会更替，增加易读性。

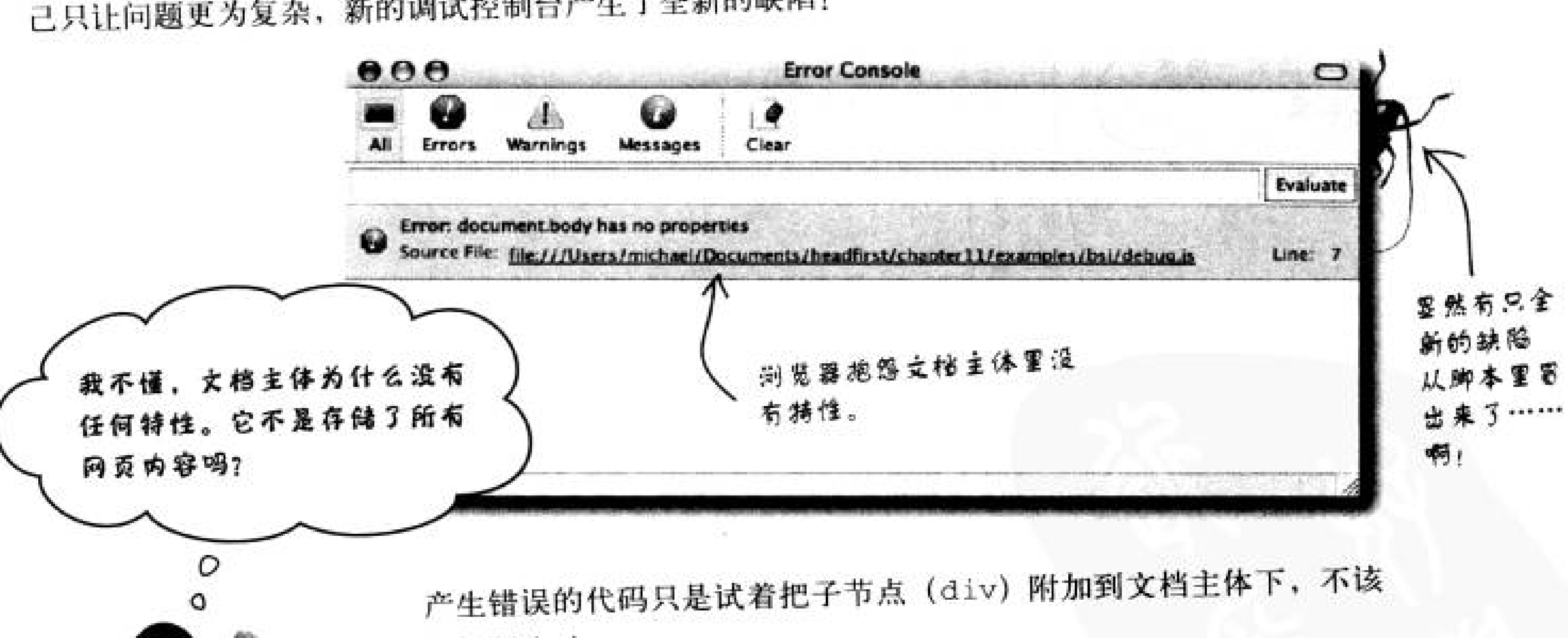
为你的调试控制台调试

Owen 已经等不及要启动新的调试控制台，找出电台直播来电脚本的剩余错误了。所以他在网页里导入了 **debug.js** 文件，并于页面标头创建了 DebugConsole 对象。

```
<script type="text/javascript">
  // Debug console global variable
  var console = new DebugConsole();
  ...
```

在页面标头创建一个全局变量的 DebugConsole 对象。

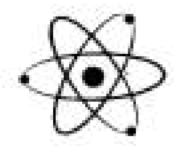
可惜，天不从人愿。Owen 第一次企图使用调试控制台时，他发现自己只让问题更为复杂，新的调试控制台产生了全新的缺陷！



产生错误的代码只是试着把子节点（div）附加到文档主体下，不该有问题才对。

```
document.body.appendChild(console);
```

一定出了什么差错，看来绝对与新的 DebugConsole 对象有关。



动动脑

这块程序代码里发生了什么错误，才会造成文档主体完全空白呢？

等待网页

调试控制台的问题，出在网页载入方式的时机以及脚本访问网页主体的时间点。

网页标头（head）比主体（body）更早载入，此时无法取得任何主体内容。

实际构成网页的HTML元素，要到主体载入后才会载入……晚于标头。

```
<html>
  <head>
    <title>BSI Case 2: Winning Caller</title>

    <script type="text/javascript" src="debug.js"></script>

    <script type="text/javascript">
      // Debug console global variable
      var console = new DebugConsole();

      // Total number of calls
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Increment the call number
        var callNum;
        ++callNum;

        // Check for a winner
        if (callNum == winningNum) {
          alert(caller + ", caller number " + callNum + "... today\'s winner!");
          form.submit();
        }
        else {
          // Reset the caller field for the next caller
          var callerField = document.getElementById('caller');
          callerField.value = "Next caller";
          callerField.focus();
          callerField.select();
        }
      }
    </script>
  </head>

  <body>
    <form name="callform" action="radiocall.php" method="POST">
      <img src="radio.png" alt="radio" />
      Caller name: <input id="caller" name="caller" type="text" />
      <input type="button" value="Call"
        onclick="checkWinner(this.form, document.getElementById('caller').value, 7)" />
    </form>
  </body>
</html>
```

在网页标头中运行的 JavaScript 代码无法访问网页内容。

既然网页标头比主体更早载入，任何直接在网页标头中运行的脚本必须小心地不访问任何网页主体里的 HTML 元素。这项限制或许看来很奇怪，但只要考虑到并非所有代码都在网页标头里运行，就没那么奇怪了。

并非所有在网页标头里的代码都在网页标头里运行。

把函数代码放在网页标头，与函数代码在网页标头里**运行**是两回事——函数直到被调用，才运行其中的代码。但在函数外的代码则于标头载入时直接运行。这些就是可能造成问题的代码。

以 DebugConsole 对象为例，它不能直接创建在网页标头里，因为它的构造函数与网页主体内容非常有关。

磨笔上阵

请写出你觉得 DebugConsole 对象应有的创建时机与地点，以确保它能安全地访问网页元素。

....

....

....

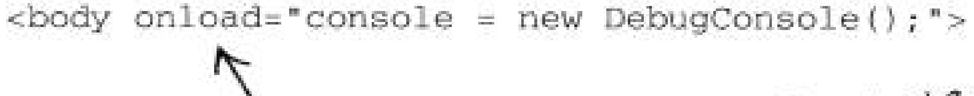
....

....

请写出你觉得 DebugConsole 对象应有的创建时机与地点，以确保它能安全地访问网页元素。

浏览器发出的 onload 事件能让我们知道网页完全载入的时机。所以，DebugConsole 对象应该创建为响应 onload 事件。不过，存储这个对象的 console 变量仍应该在网页标头里声明，让这个对象为全局变量——我们只是暂时不调用构造函数以实际创建对象，而是等到 onload 事件发生。

移动DebugConsole对象的创建位置，削减调试控制台的缺陷！



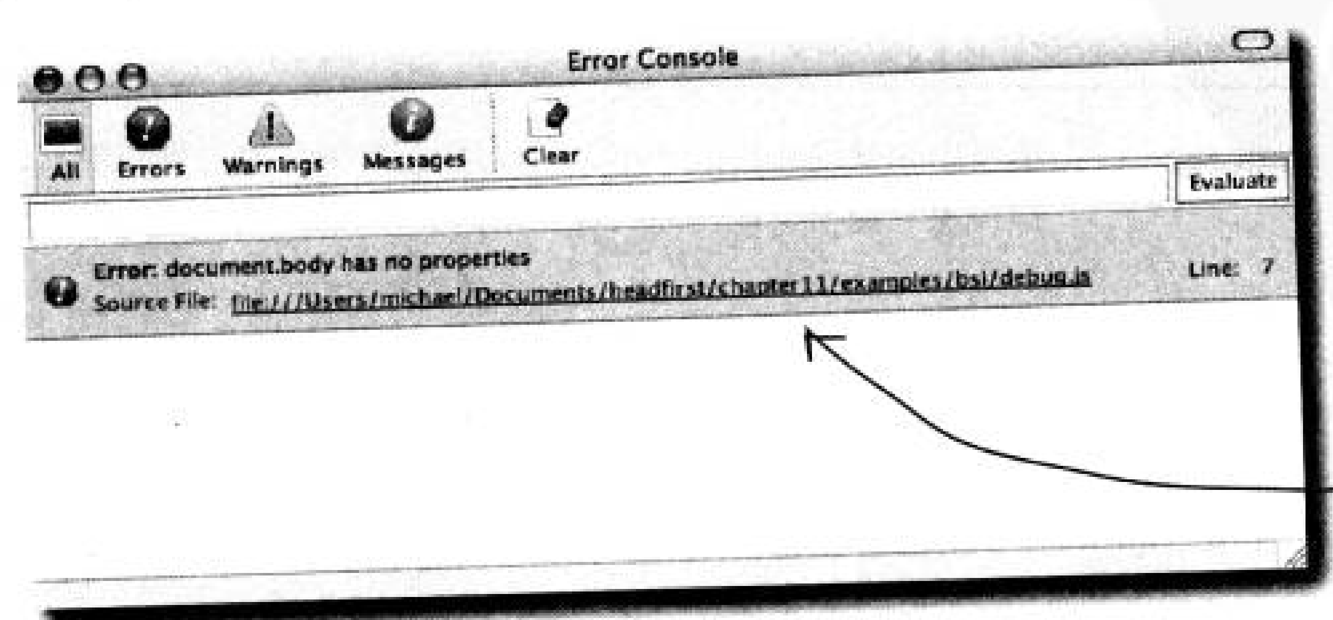
```
<body onload="console = new DebugConsole();">
```

DebugConsole 对象现在创建为响应 onload 事件。

最麻烦的错误：运行时

不会载入文档主体的问题是个运行时错误（runtime error）的例子——只在脚本实际运行时才在某种状况下露脸的错误。有时候，运行时错误只在非常特殊的状况下浮现，例如遇到某种用户输入的数据或发生特定次数的循环迭代时。运行时错误通常是最难缠的错误，它们非常难以预测。有时候，光是重制别人遇到的运行时错误，就是项挑战了。

运行时错误只在脚本运行时发生特定情况时出现。



调试控制台的缺陷，是个运行时错误，由于在数据载入前尝试访问它而造成。这是个只在脚本运行时出现的错误。

JavaScript 三虫客

运行时错误，加上我们见过的另外两种 JavaScript 缺陷，构成“三虫客”：语法错误、逻辑错误、运行时错误。这些错误能出现在任何脚本中，通常还会同时出现！了解这三者的不同，是成功寻找并消灭它们的重要课题。

语法错误
(Syntax error)

由于违反 JavaScript 语言规则而造成的错误，表示程序代码不适合于JavaScript解释器中运行。

运行时错误
(Runtime error)

只因运行时的条件而出现的错误，例如用户在表单里输入特定类型、脚本无法处理的数据，或试图在对象创建或初始化前访问对象。

逻辑错误
(Logic error)

因错误的逻辑而造成的错误，通常是意图做某件事，却意外地写成另一件事的代码。有些具有逻辑错误的代码仍会如它的原始意图般运行，此时是程序设计师一开始就误解了任务。

```
<html>
  <head>
    <title>BSI Case 2: Winning Caller</title>

    <script type="text/javascript" src="debug.js"></script>

    <script type="text/javascript">
      // Debug console global variable
      var console = new DebugConsole();

      // Total number of calls
      var callNum = 0;

      function checkWinner(form, caller, winningNum) {
        // Increment the call number
        var callNum;
        ++callNum;

        console.displayMsg("callNum: " + callNum);

        // Check for a winner
        if (callNum = winningNum) {
          alert(caller + ", caller number " callNum + "... today's winner!");
          form.submit();
        }
        else {
```

习题

请写出下列错误描述的类型。

在 if 语句的测试条件外缺少括号。 ....

忘记初始化计数器变量为 0。 ....

创建一个循环次数超过数组最后一个元素的循环。 ....

忘记以结尾大括号结束函数。 ....

请写出下列错误描述的类型。

在 if 语句的测试条件外缺少括号。 Syntax

忘记初始化计数器变量为 0。 Logic

创建一个循环次数超过数组最后一个元素的循环。 Runtime

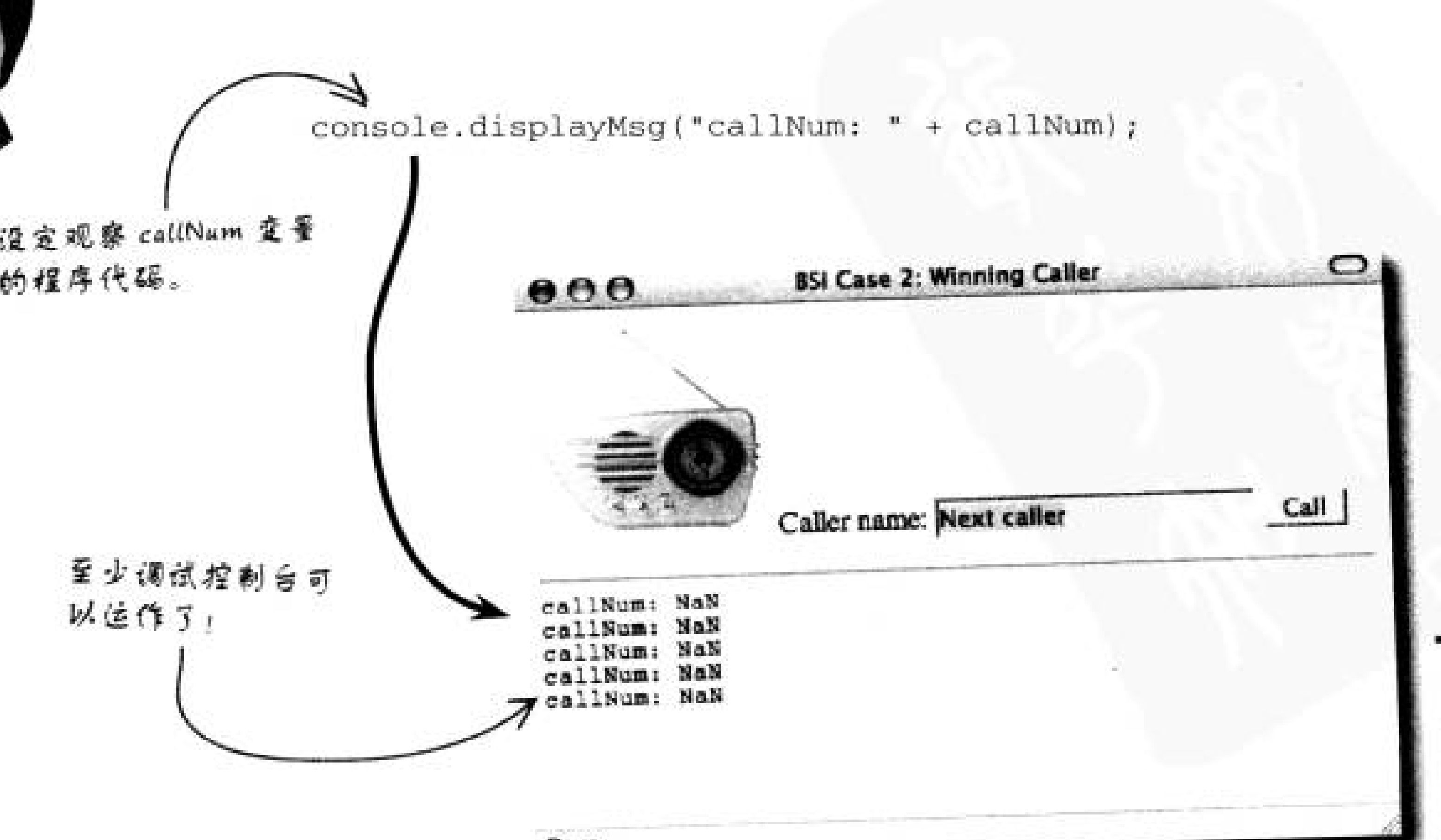
忘记以结尾大括号结束函数。 Syntax

来电序号显示为“并非数字”（NaN）。非常奇怪……

不是数字不是数字

调试控制台终于上线运行后，现在可以观察电话打进来时的 callNum 变量，又不用挣扎着关掉一个个 alert 框。结果，我们终于看到 Owen 一直忽略的错误。callNum 变量显现为 NaN，意思是说它不是数字。但是，为什么会这样？

```
console.displayMsg("callNum: " + callNum);
```



当观察还不够时

有时候，观察变量制造了更多的问题，而不是提供答案。为什么 `callNum` 不是个数字？这样的调试控制台有什么用，如果它只是证明你已经知道的事情……程序代码有问题！我们究竟该如何找出是什么样特定的问题呢？

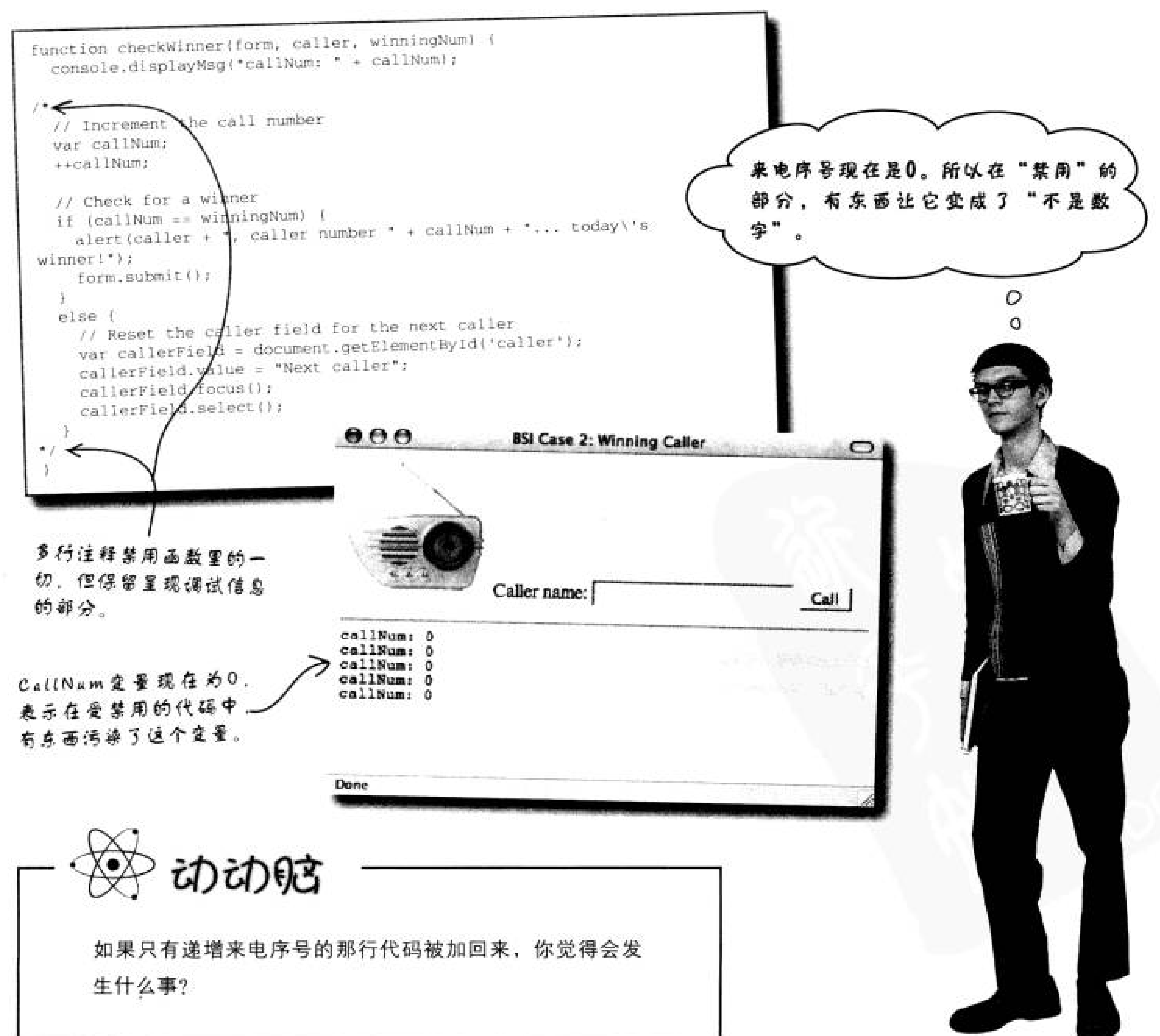
在猎捕缺陷时，移除程序代码是个绝佳的脚本简化方式。

在 JavaScript 调试的工作中，有时候简单反而表示了千言万语。在本例中，移除代码并观察变量值的改变，就是个非常棒的主意。但删除程序代码听起来实在让人不舒服，毕竟，完成调试后仍需留下大部分代码。我们需要**禁用**（diable）程序代码的方式，而非真的把它们移除。

利用注释暂时禁用代码

把可执行代码藏在注释里，是在调试过程中禁用代码的绝佳便利方式。代码可以选择性地被抽离脚本的运行，但不需真正地删除。把这种用注释隐藏的做法，当成在需要隔离缺陷时削减一行或一段代码的手段。

注释在暂时禁用代码时非常好用。

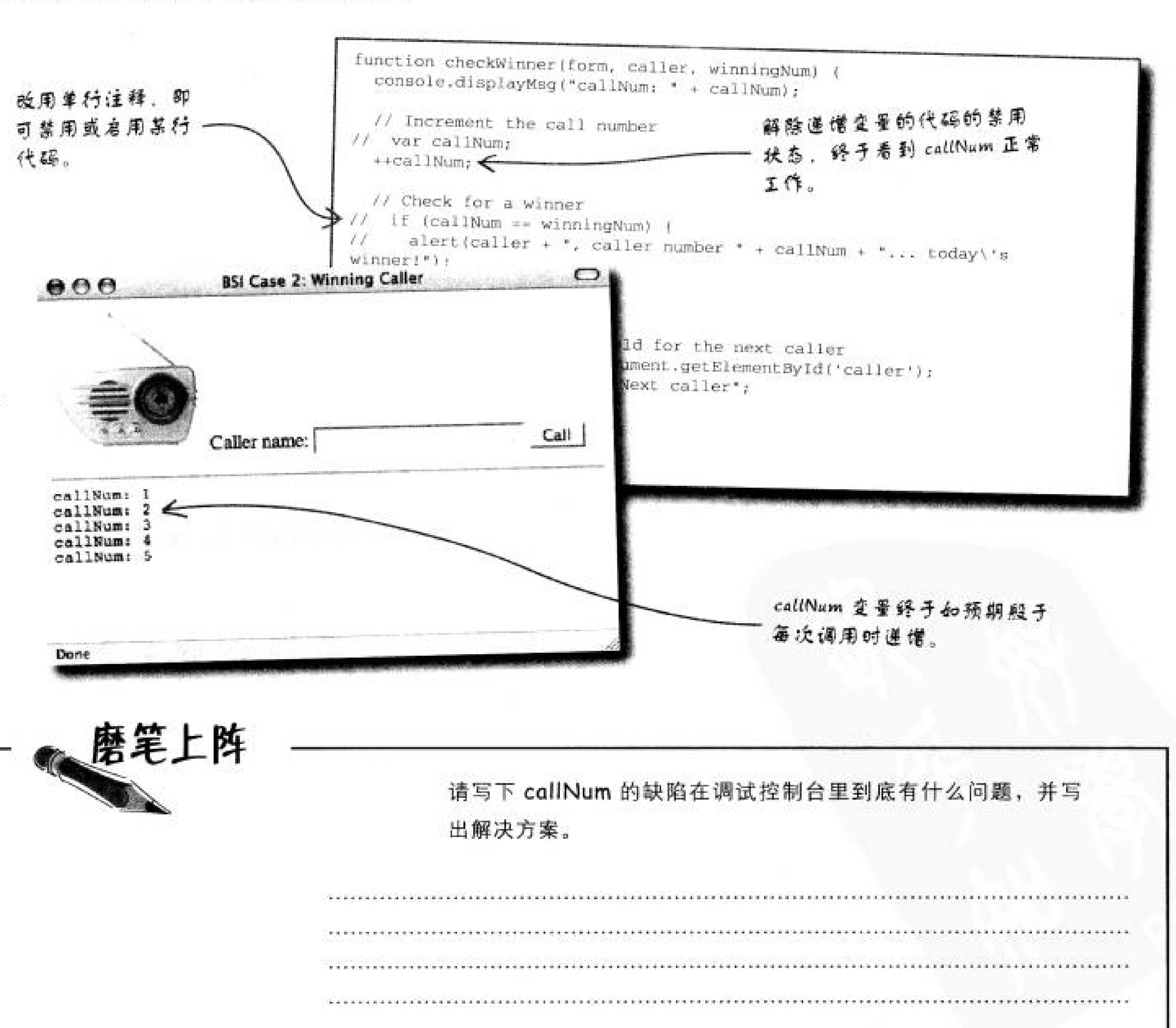


动动脑

如果只有递增来电序号的那行代码被加回来，你觉得会发生什么事？

问题解决……算是解决了

改用单行注释后，在禁用代码上比较有选择性。如果加入递增 callNum 变量的程序代码，它的运作方式终于像该有的样子了。所以剩余的禁用代码中潜藏着问题的成因。



磨笔上阵

请写下 callNum 的缺陷在调试控制台里到底有什么问题，并写出解决方案。

....

....

....

....

....

....

请写下 callNum 的缺陷在调试控制台里到底有什么问题，并写出解决方案。

在 checkWinner() 函数里，意外地以 var 创建了另一个名称同样是 callNum 的局部变量。所以，局部变量"掩蔽"了全局变量，造成一个很难检测的问题。因为局部变量并未初始化，所以递增与稍后的赢家序号比较都导致"不是数字"的问题。修正方式只需移除函数里的一行代码——以 var 创建局部变量 callNum 的那一行。

删除创建局部变量的代码，消灭缺陷！

```
// Increment the call number
var callNum;
++callNum;
```

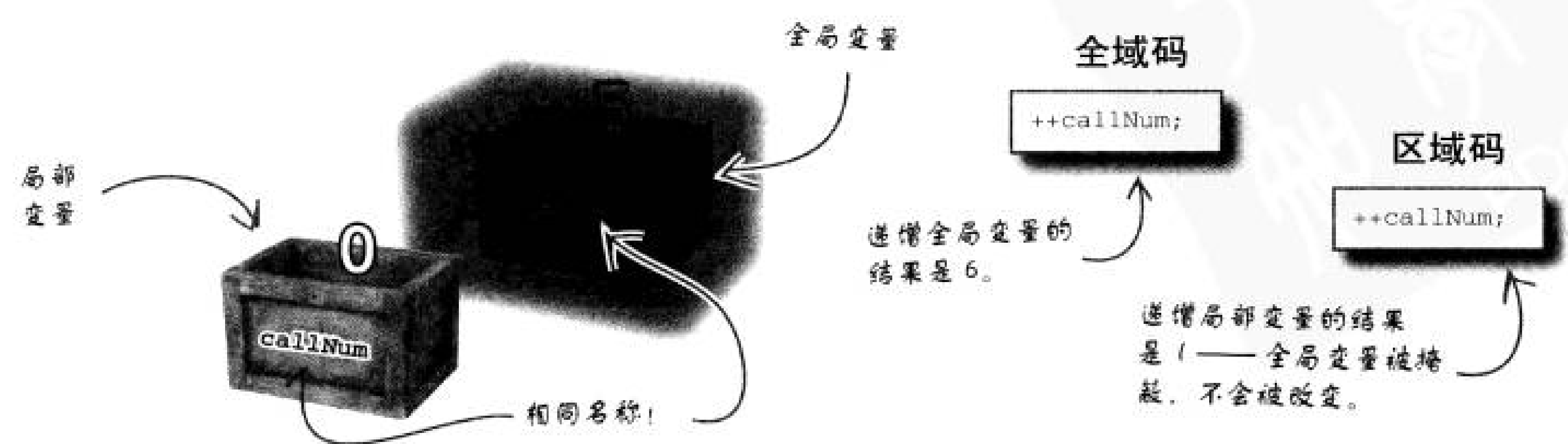
（var callNum; 一行被划掉）

借由移除意外创建局部变量callNum的代码，函数即可改用全局变量callNum，这才是最初的意图。

shadow variable的危险

电台直播来电脚本里的 callNum 的缺陷，它是 **shadow variable**（阴影变量）的范例，这是种意外以相同名称遮盖了其他变量的变量。问题于局部变量的名称与全局变量相同时浮现。JavaScript 创建局部变量，并在局部范围内给它较高优先权。所以，任何在局部变量内的改变不会带到全局变量里——局部变量掩蔽了全局变量，暂时让脚本看不到全局变量。

shadow 变量发生在局部变量与全局变量采用相同名称创建时……大事不妙。



问：**使用注释隐藏代码以便追踪缺陷时，我要如何判断哪些代码该禁用呢？**

答：这是经验问题，随着你的JavaScript调试经验的增加，你的判断能力也会越来越好。然而，如果不是禁用接近问题区域的全部代码，过度地禁用大部分代码也绝不会出错。如果真的出现棘手问题，立刻禁用网页上所有脚本，不要犹豫。也别忘记暂时移除任何导入标签，它们会把外部代码拉入网页。

如果你已经隔离出缺陷，要确认它在脚本的某个区域，还有另一个方法。请一行行地禁用代码，直到缺陷消失。此时不是禁用所有东西，再慢慢地启用代码，直到缺陷**出现**，而是逐行禁用代码，直到缺陷**消失**。前者在不知道缺陷大概位置时的效果较好，后者则在某种程度上隔离出缺陷位置时比较适用。

问：**我可以故意创建shadow variable吗？有关系吗？**

答：这个问题好比"我可以故意打断我的腿吗？"你说有没有关系啊？答案当然是不行。故意造成痛苦，害你自己受苦，也不会让这件事被大家接受。另外，为意图完美运作的程序代码调试原本已是东跌西撞的工作，你实在不需刻意增加风险因子。所以，问题的真正答案是：shadow variable为JavaScript带来许多困惑与鬼鬼祟祟的行为，在任何情况下都应避免。

案件终结

加上适量的耐心，还有新的除虫技巧的协助，Owen结束了这个案件，并晋升为BSI的JavaScript探长。

案子结束了！

BSI Case 2: Winning Caller

Caller name: Alan [Call]

callNum: 1
callNum: 2
callNum: 3
callNum: 4
callNum: 5
callNum: 6
callNum: 7

Done

Alan, caller number 7...today's winner!

OK

Owen探长，干练的JavaScript除虫侦察人员。

没有缺陷的电台直播来电完美执行，并附上调试控制台。

Owen 的除虫清单

确定括号都成对出现。

确定围起代码块的大括号都成对出现 —— 小心地缩排代码有助于维持成对的大括号。

努力避免打错标识符名称 —— 变量与函数的名称如果不一致，很容易造成大问题。

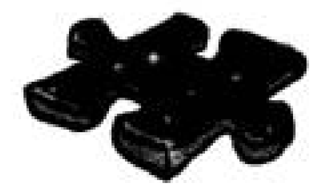
在使用引号与撇号时，维持使用方式的一致；如果需要，在 HTML 属性中小心地混用这两个符号。

使用转义符，为字符串中具有特殊意义的字符编码，例如引号（\"）或撇号（\'）。

千万绝对切莫别在想用 == 时误用了 =。JavaScript 或许不会把这点视为错误，但你的程序代码运作得将跟你想的不一样。

确定对象在被访问前已经创建 —— 主要发生在网页元素上，网页元素要到 onload 事件触发的前一刻才完成创建。

别为局部变量和全局变量取相同的名称，因为局部变量将掩蔽全局变量，造成某些无法预测的行为。



JavaScript 填字游戏

在你发现自己对缺陷的敬爱如淘淘江水绵延不绝，因而跑去买了一座蚂蚁农场前，先试试我们的填字游戏。

横向提示：

2. 使用________快速地观察变量。
4. 把 JavaScript 字符串放入 HTML 属性时，请混合使用撇号与________。
7. 用于呈现错误的特殊观察窗口。
10. 违反 JavaScript 语言规则的错误类型。
12. 虽然在 JavaScript 中完全合法，却会得出错误结果的错误类别。
13. 未被指派值的变量就是________。
14. Owen 自定义的缺陷对抗对象。

纵向提示：

1. 使用________暂时禁用代码。
3. 如果________出现在变量名称里，是个问题。
5. 目前最适合 JavaScript 调试的网络浏览器品牌。
6. 围起一段程序代码时若少了________，你将遇到麻烦。
8. 一条巧克力里允许的昆虫碎片数量。
9. 只在脚本运行时出现的错误。
11. 使用________，为字符串中的特殊意义字符编码。

JavaScript 填字游戏解答

		1 C									2 A	L	E	R	3 T							
4 Q	U	O	T	E	S		5 F			6 C					Y							
		M					I			U					P							
		M				7 E	R	R	O	R	C	O	N	S	O	L	E					
		E			8 S		E			L												
		N			I		F			Y												
		T			X		O			B		9 R										
		10 S	Y	N	T	A	X			R		U										11 E
					Y					A		N										S
										C		T						12 L	O	G	I	C
							13 U	N	D	E	F	I	N	E	D							A
										S		M										P
											14 D	E	B	U	G	C	O	N	S	O	L	E

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

JavaScript缺陷该有什么下场？

这是左右脑的秘密会谈！

结束！

Case 1

Case 2

太酷了…………我不是缺陷哦！真的。

转过头装作没看到缺陷，
是个错误的方式，或许终将超支
你对缺陷的容忍度。所有
讨人厌的缺陷将使你的
代码衰弱，这是个问题。

12 动态数据

贴心易感的网络应用程序

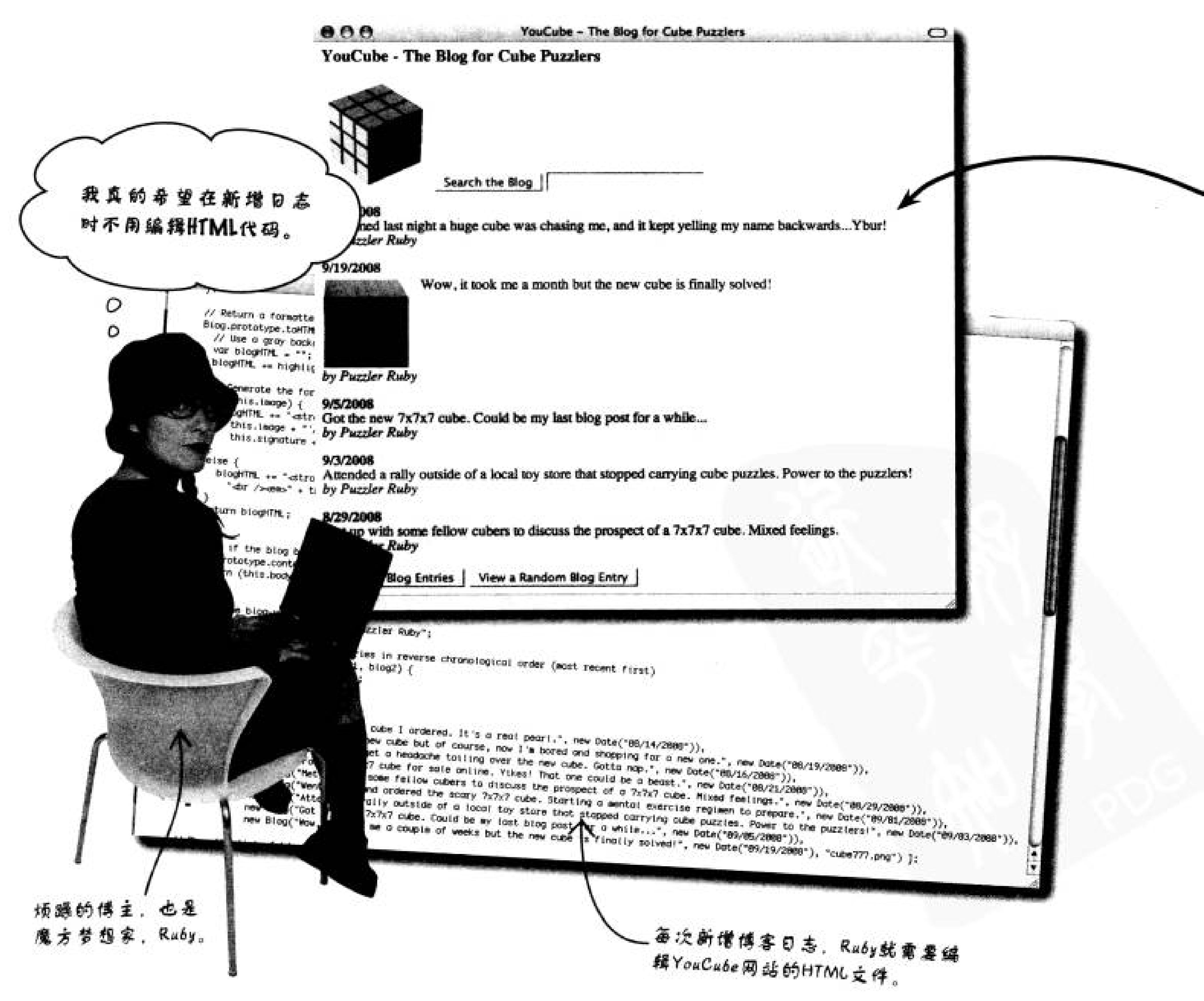
别被我的眼神骗了，在这副冷硬的表情下，藏着赤裸裸、等不及要猛虎出闸的情感。事实上，动态的个性才是我最大的资产。

当代万维网是个非常有反应的世界，网页被期待着响应读者的每声呻吟，至少许多网站用户和开发者都有这个梦想。透过一个称为 Ajax —— 提供戏剧性地改变网页“感觉”机制的编程技术，JavaScript 在这个梦想中扮演了重要角色。有了 Ajax，网页的行为比较像完全成熟的应用程序，因为它们就能够**动态地快速下载并存储数据**以及**实时响应用户**，却不用重新载入网页或利用浏览器的某些小手段。

渴望着动态数据

还记得Ruby吗？我们的魔方狂热解谜家，也是博主。Ruby很喜欢由JavaScript驱动的YouCube博客，但每次为了新日志的增加，都要编辑整页HTML文件，实在让她感到烦躁不堪。她想从描述博客的HTML代码中划开博客日志的部分，把注意力集中到博客内容本身。

新增一则日志到YouCube不该需要编辑网页。



由数据驱动的YouCube

Ruby的数据驱动网页文件可至http://www.headfirstlabs.com/books/hfjs/下载。

Ruby又在策划行动了。从网页结构中划分博客内容的版本，牵涉到动态数据（dynamic data）：在网页被浏览器处理时，动态供给网页的数据。以动态数据建造的网页称为**数据驱动**网页（data-driven page），因为这类网页只定义了等着数据填入的页面架构而已。换句话说，数据负责网页的内容。

博客数据的实体另外存储成一份文件，可在不碰触网页的状况下编辑这份文件。

博客数据

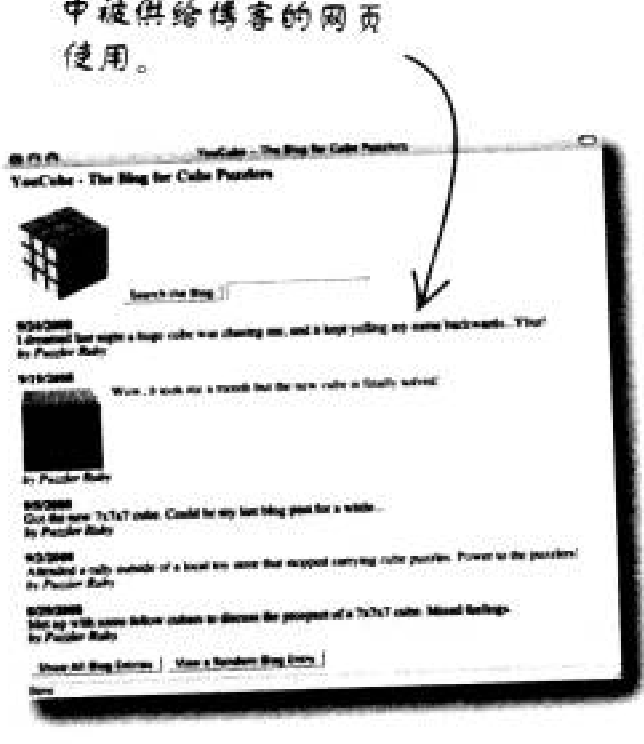
网页

网页包含网页结构所用的HTML代码，还有整合动态博客数据到网页上的JavaScript代码。

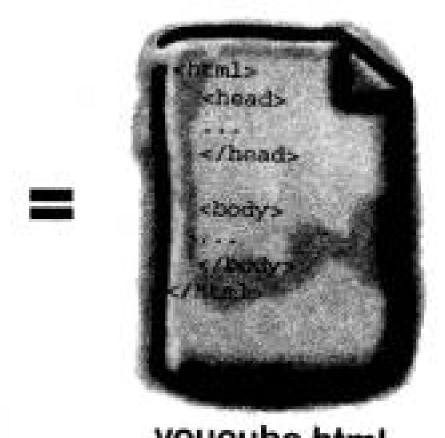
JavaScript负责处理博客数据，并融入最后完成的HTML网页里。

利用JavaScript的协助，生鲜博客数据被动态融入HTML代码，产生最终版、看起来与原始YouCube完全一样的网页。但这个数据驱动的网页是由不同成分组成，包括：结构页面和博客数据。因为博客数据分家到自己的文件中，Ruby可以自由操纵博客的内容，而内容现在已经与网页上的HTML、CSS和JavaScript代码分离了。

博客日志从别份文件中被供给博客的网页使用。

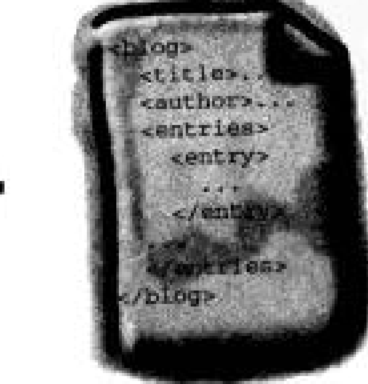


=



youcube.html

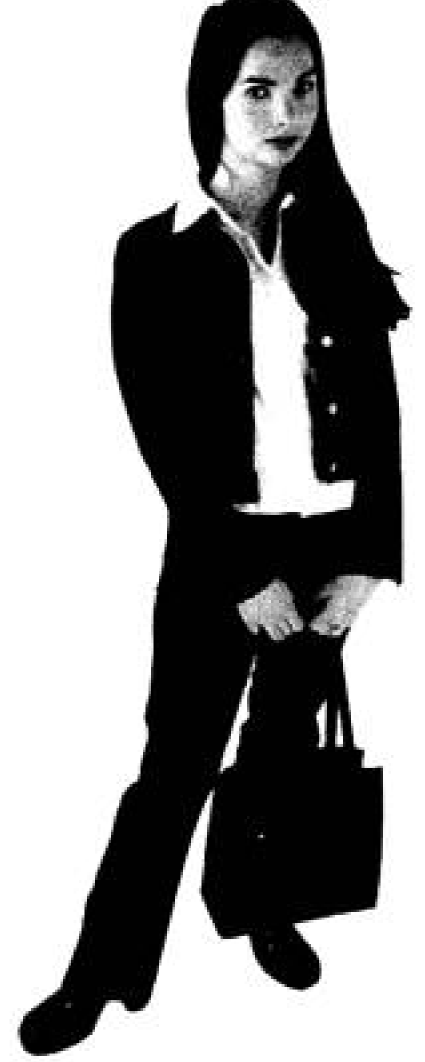
+



blog.xml

Ruby只需编辑这个文件，就可更新这个数据驱动的新博客。

动态数据听起来很复杂。我看它一定需要像座小山似的 JavaScript 代码，对不对？



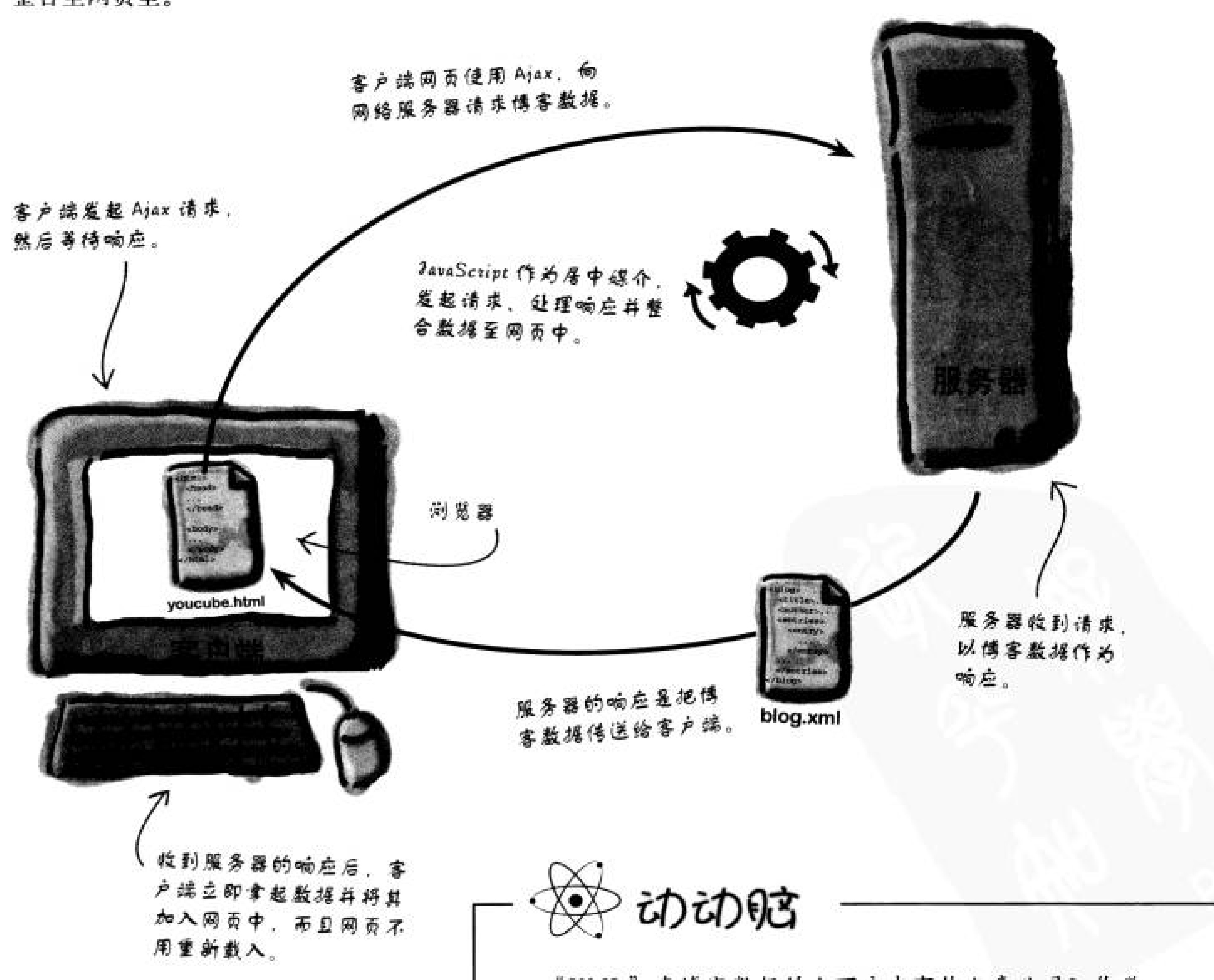
动态数据需要一点事前编程工作，但日后带来庞大的回报。

虽然，由动态数据驱动的网页确实需要一些额外规划与事前努力，但长期而言，它带来快速简单的网页更新的回报远比付出大多了。此外，JavaScript 也内置动态数据的支持，都要感谢一种名为 Ajax 的程序设计技术。

Ajax 献身沟通

Ajax 让客户端的浏览器能与网络服务器有些“小对话”(conversation)，动态数据因而成真。更详细点说，脚本可向服务器请求部分数据，例如一组博客日志，服务器则用 Ajax 传递数据。而后脚本收下博客数据并动态地整合至网页里。

Ajax 让网页能动态接收网络服务器的数据。



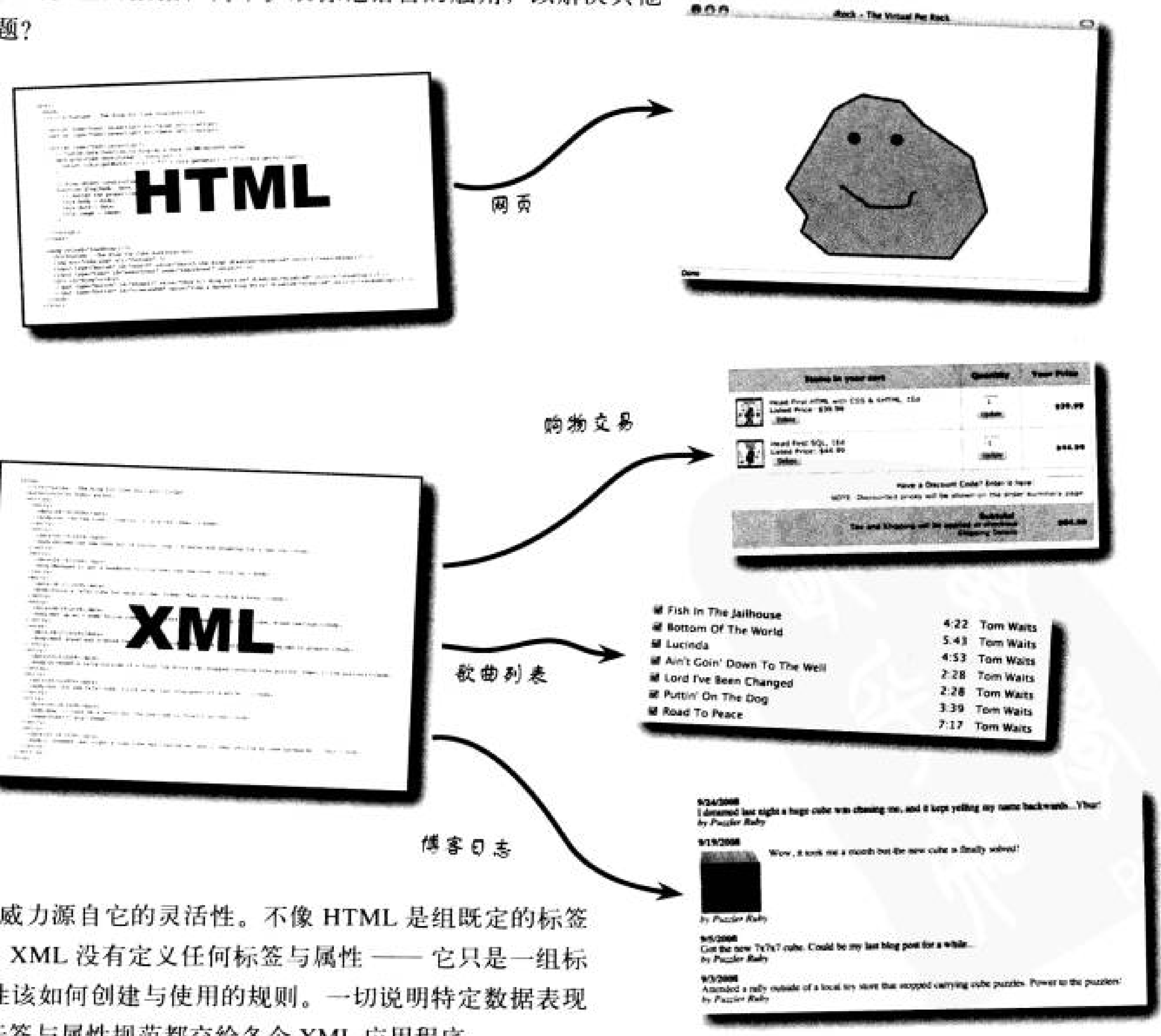
动动脑

“XML”在博客数据的上下文中有什么意义吗？你觉得它对动态数据有什么帮助？

万用 HTML：XML

在HTML里的“ML”，是指markup language（标记语言），也指出了HTML使用标签（tag）和属性（attribute）创建超文本（HT）的事实。正如 HTML 这种标记语言用于创建超文本网页，XML标记语言用于创建……咳……几乎一切你想创建的东西。这就是“X”的意思 —— 任何事件！既然有着各种能因存储成标签与属性而获益的数据，何不扩展标记语言的触角，以解决其他数据问题？

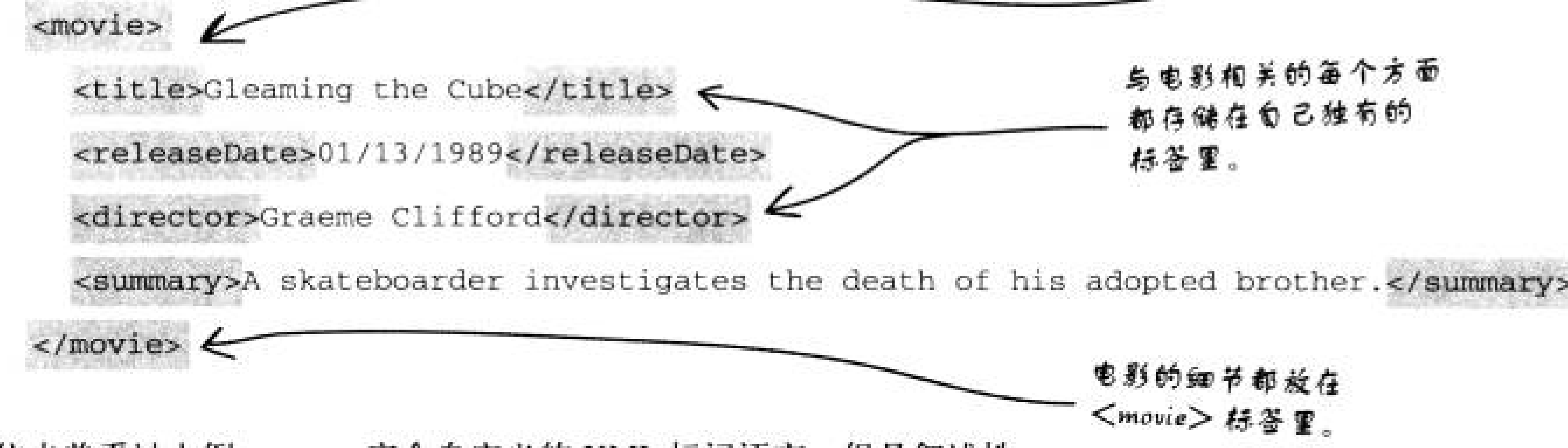
XML 是种为任何类型的数据设计格式的标记语言。



XML 的威力源自它的灵活性。不像 HTML 是组既定的标签与属性，XML 没有定义任何标签与属性 —— 它只是一组标签与属性该如何创建与使用的规则。一切说明特定数据表现方式的标签与属性规范都交给各个 XML 应用程序。

XML 让你用自己的标签标示数据

XML 真正让人心花怒放的地方，在于它能把随便一个路人变成标签自定义专家，只需要一点点标签与属性的炼金术，就可依任何目的熬出完全自定义的标记语言。现在当然已经有很多创建完毕的 XML 语言，用于解决各式各样的问题，而且在适合你的需要时借用这些 XML 语言，这也是不错的主意。但想到创建自己专属的标记语言……真是难以抵抗的诱惑。

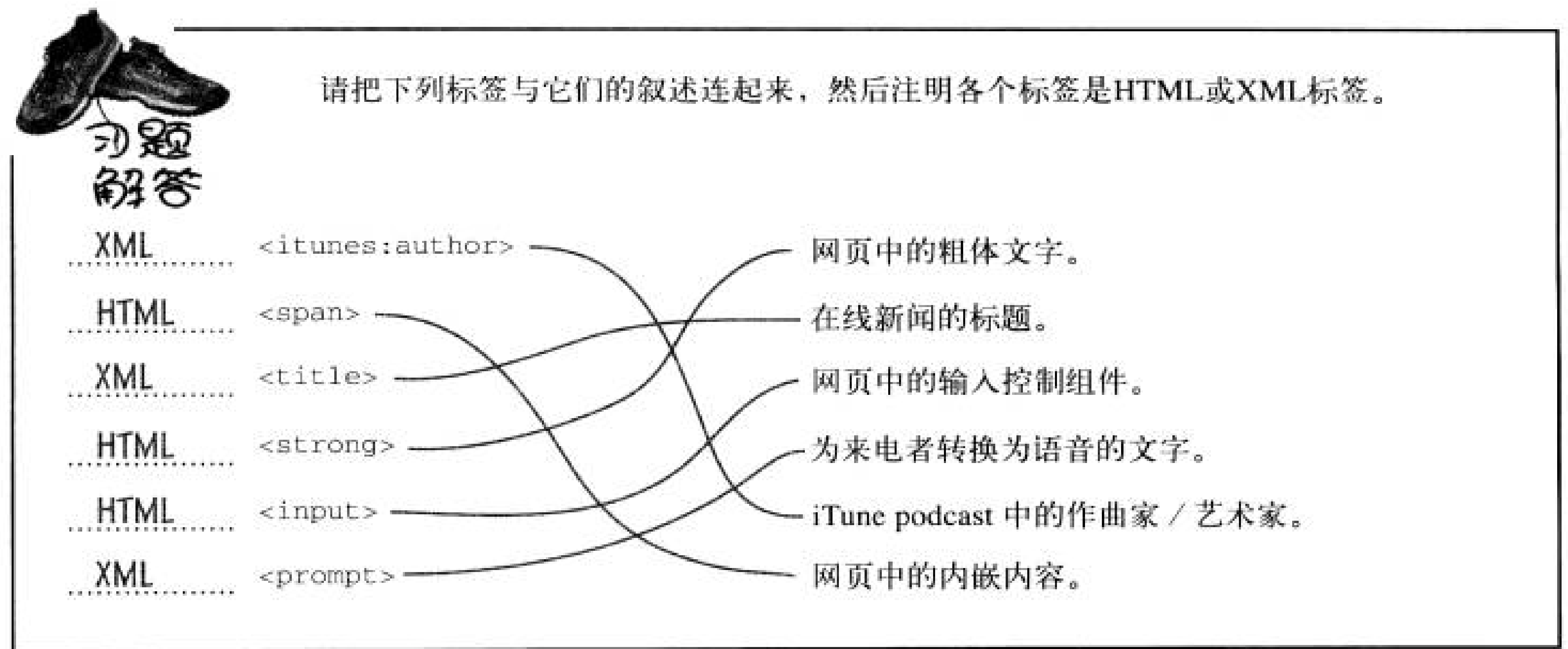


各位未曾看过上例 —— 完全自定义的 XML 标记语言，但具叙述性的标签有助于解释它的数据。更重要的是，标签非常针对它存储的数据 —— 使用 <director> 标签存储导演数据，实在太合理了！

习题

请把下列标签与它们的叙述连起来，然后注明各个标签是HTML或XML标签。

	标签	叙述
....	<itunes:author>	网页中的粗体文字。
....	<span>	在线新闻的标题。
....	<title>	网页中的输入控制组件。
....	<strong>	为来电者转换为语音的文字。
....	<input>	iTune podcast 中的作曲家/艺术家。
....	<prompt>	网页中的内嵌内容。



XML只是文本

与HTML相似，XML只是**文本**，也就是说，它存储在一般纯文本文件里。然而，XML文件的扩展名为.xml，而不像HTML使用的.html或.htm。

XML + HTML = XHTML

扩展名或许不同，但XML与HTML有着非常重要的联系，就是XHTML。XHTML是现代版的HTML，遵循XML的严格规则。例如说，XHTML网页中的每个起始标签必须有相应的结尾标签。HTML的语法比较快速但松散，有时候就算没为<p>加上</p>，还是可以过关。XHTML就没这么容忍，它要求这类标签必须成对出现。

XHTML是遵守XML的较严格语法规则的新版HTML。

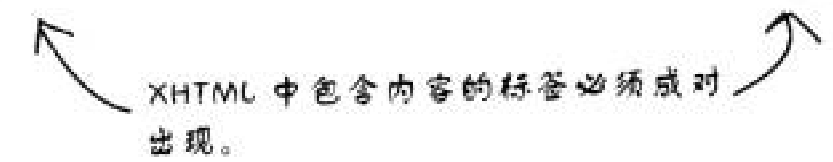
HTML

```
This is a paragraph of text in HTML.<p>
```

<p>标签通常单独出现在HTML代码中，指示段落的起始或结束。

XHTML

```
<p>This is a paragraph of text in XHTML.</p>
```



HTML与XHTML的另一个重要差异则关于空标签，例如
，它们的格式里必须加上一个空格与反斜线以表示没有结尾标签。

HTML

```
This is just a sentence.<br>
```

无内容的换行标签在HTML中通常不会加上反斜线。

XHTML

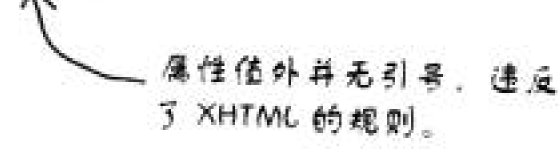
```
This is just a sentence.<br />
```

XHTML中所有空标签都需要加上空格与反斜线。

再说一个 HTML 与 XHTML 的重要差异，XHTML 要求所有属性值均以引号围起。

HTML

```
<a href=home.html>Go home</a>
```



XHTML

```
<a href="home.html">Go home</a>
```

所有XHTML属性值必须出现在引号里。

虽然 XHTML 并未直接影响 Ruby 的立即需求（把博客数据塑造成 XML），但它勾勒出一些最重要的 XML 语法规则，这些规则适用所有以 XML 为基础的语言，包含 Ruby 的自定义博客数据语言。

今晚主题：HTML 和 XML 的网络数据密谈

最新版的 HTML 使用 XML 二度成形，称为 XHTML。

XHTML: 你为我的生活带来多大困扰，你知道吗？我一直是万维网的骨干，现在大家都因为你的关系，一头雾水地看着我。

XML: 你的如豆目光只看得到网页，这又不是我的错。我扩展了自己的心胸，因而可以表示任何类型的数据。

XHTML: 可是没有我的话，你也没什么好用处，浏览器只会呈现 HTML 代码而已。它们甚至不知道你由什么东西组成。

XML: 我是神秘，神秘是我。事实上，我是个没有脸的人 —— 只有内在，没有外观。在需要自我表现的时机时，我才需要你。

XHTML: 怎么可能？谁会在意看不到的数据呢？

XML: 唉，你还是不懂吗？外面的世界大部分时间都在看不见的数据上运作，如银行交易、政党投票、天气预报……你想得到的都是。

XHTML: 但都要有我，它们才能被看到啊 —— 一切都在万维网上。

XML: 是没错啦……不过，你以为数据被送到网络浏览器前，究竟怎么存储？我可以告诉你，数据可不是直接存储成你的段落和表格哦。数据通常利用我存储，因为我提供许多结构与上下文 —— 我让数据容易处理。

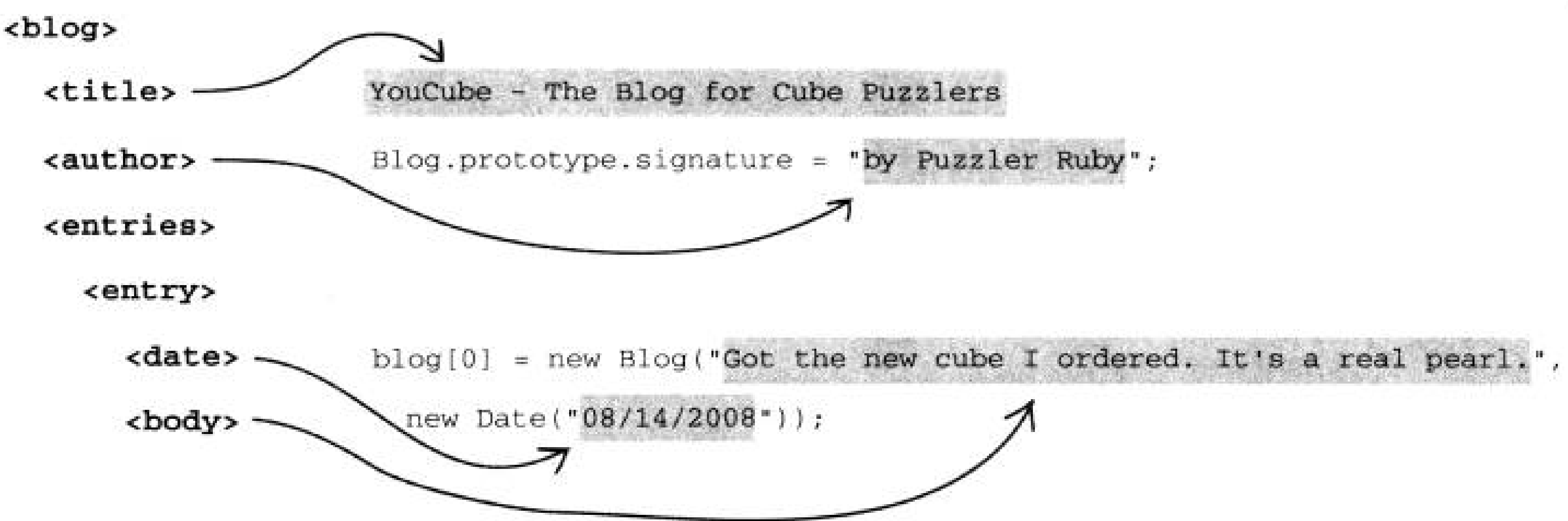
XHTML: 这样啊……所以说，我们其实在一起工作吗？

XML: 正是如此！我对数据的外观没有概念，相对地，我专心于数据的**意义**。只要人类继续使用网络浏览器，我就会一直需要你帮忙呈现我代表的数据。

XHTML: 真是让人松了一口气。

XML 与 YouCube 博客数据

XHTML 是个很棒的 XML 应用程序，快速地改善了网页结构和稳固度。不过，从 YouCube 博客的角度来看，Ruby 需要自定义一个模仿她的特殊博客的 XML 语言。我们需要访问博客所需的不同数据，还要考虑数据如何融入有层次的 XML 标签上下文。



磨笔上阵

请各位发明自己的博客存储用 XML 语言，并使用你的语言编写博客日志代码。请把标题、日期、作者、日志正文等因素纳入考量。

```
<blog>

........................................................................

........................................................................

........................................................................

........................................................................

........................................................................

........................................................................

........................................................................

........................................................................

</blog>
```

请各位发明自己的博客存储用 XML 语言，并使用你的语言编写博客日志代码。请把标题、日期、作者、日志正文等因素纳入考量。

```
<blog>
  <title>YouCube - The Blog for Cube Puzzlers</title>
  <author>Puzzler Ruby</author>
  <entries>
   <entry>
    <date>11/14/2007</date>
    <body>Got the new cube I ordered. It's a real pearl.</body>
   </entry>
  </entries>
</blog>
```

整体博客包含在 <blog> 标签里。

<title> 标签存储博客标题。

你觉得哪个标签负责存储博客的作者呢？

博客日志的集合存储在 <entries> 标签里。

一则博客日志表示为一对 <entry> 标签。

日期与每则博客日志主体各有自己的标签。

没有蠢问题

问：为何不把博客日志存储成一般没有格式的文本呢？

答：是可以这么做，但将造成脚本极大的负担，脚本需筛选数据，试着分出不同笔日志，还不能漏了日期与正文。XML为数据加上可预测的结构，所以你可以轻易地分辨不同域的数据，例如分辨出某则博客日志的日期与正文，更别提还可以加上标题与作者了。

问：<entries> 标签真的是 XML 版博客数据的必需条件吗？

答：它并非严格必要，但的确让数据格式更有结构，而且更容易理解。你看"磨笔上阵"的例子，若没有<entries>标签，将无法得知博客是否支持多个<entry>搭配只出现一次的<title>与<author>标签。<entries>标签则指出有许多则博客日志，数据因此更有组织，关于数据的使用方式也较为明显。

问：XML 与 Ajax 有什么关联呢？

答：Ajax 曾经是"Asynchronous JavaScript And XML"的缩写，所以 XML 一度与 Ajax 直接绑在一起。这个缩写最近被认为过时了，Ajax 的角色已经扩展到不见得都把 XML 当成方程的一部分。事实上，XML 仍然形成大多数 Ajax 应用程序的基础，因为它提供了数据建模的绝佳机制。

本章稍后将提到，Ajax 与 XML 产生关系的方式才使 JavaScript 支持 Ajax。JavaScript 并未限制大家在实行 Ajax 请求与响应时只能使用 XML 作为数据格式，但这样做的确较容易应付一切但也是最琐碎的数据。虽然 Ajax 纯粹主义者会说 XML 与 Ajax 两不相关，实际上，它们常常手牵手并肩而行。就算社会开始嫌那个缩写的原意过时，大多数时候它仍然表示了事实。我们稍后将讨论"异步"（asynchronous）的部分。

我还是如坠五里雾中。以特殊格式存储数据，怎么会让数据变成动态的？

只有XML当然还不是动态的，但它刚好与Ajax和DOM都能紧密调合。

XML 是最常与 Ajax 合用的数据格式，因此也是表示 YouCube 博客数据的合理候选者。数据驱动版 YouCube 的数据将在服务器端与客户端间频繁往来传送。XML 的高度结构化本质，让它成为快速运送数据的理想选择。

而且XML与HTML（XHTML）相似，遂可透过DOM，把XML视为节点树而访问数据。也就是说，你可以设计JavaScript代码，在XML节点树中往复移动，小心地分离出想要的数据，然后动态地融合数据与网页。这么多优点，使XML是个创建动态、数据驱动网页时的良好数据存储方案。

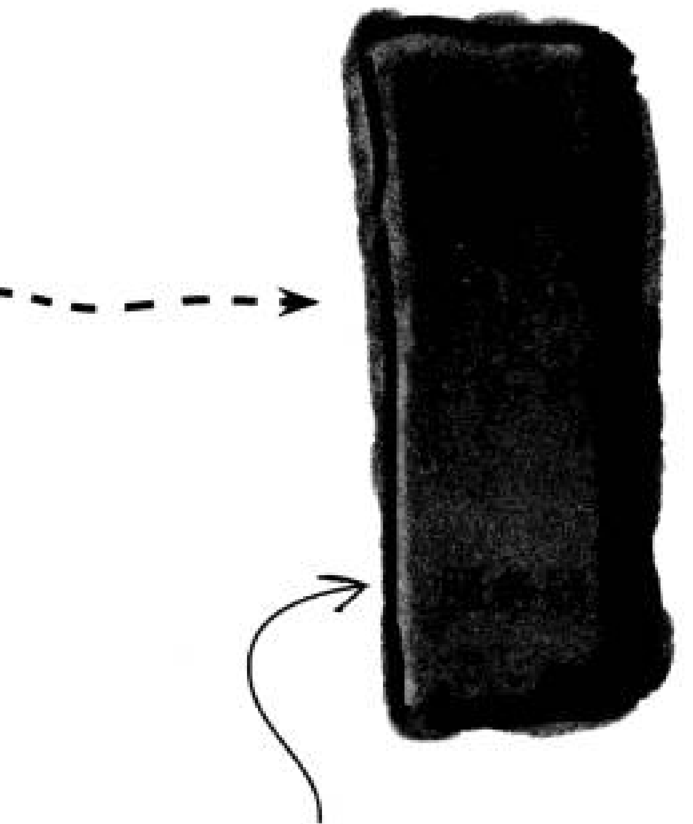
为 YouCube 注入 Ajax

手上有了新鲜的 XML 博客文件，Ruby 准备开始在 Ajax 的协助下动态地载入文件至 YouCube 网页了。

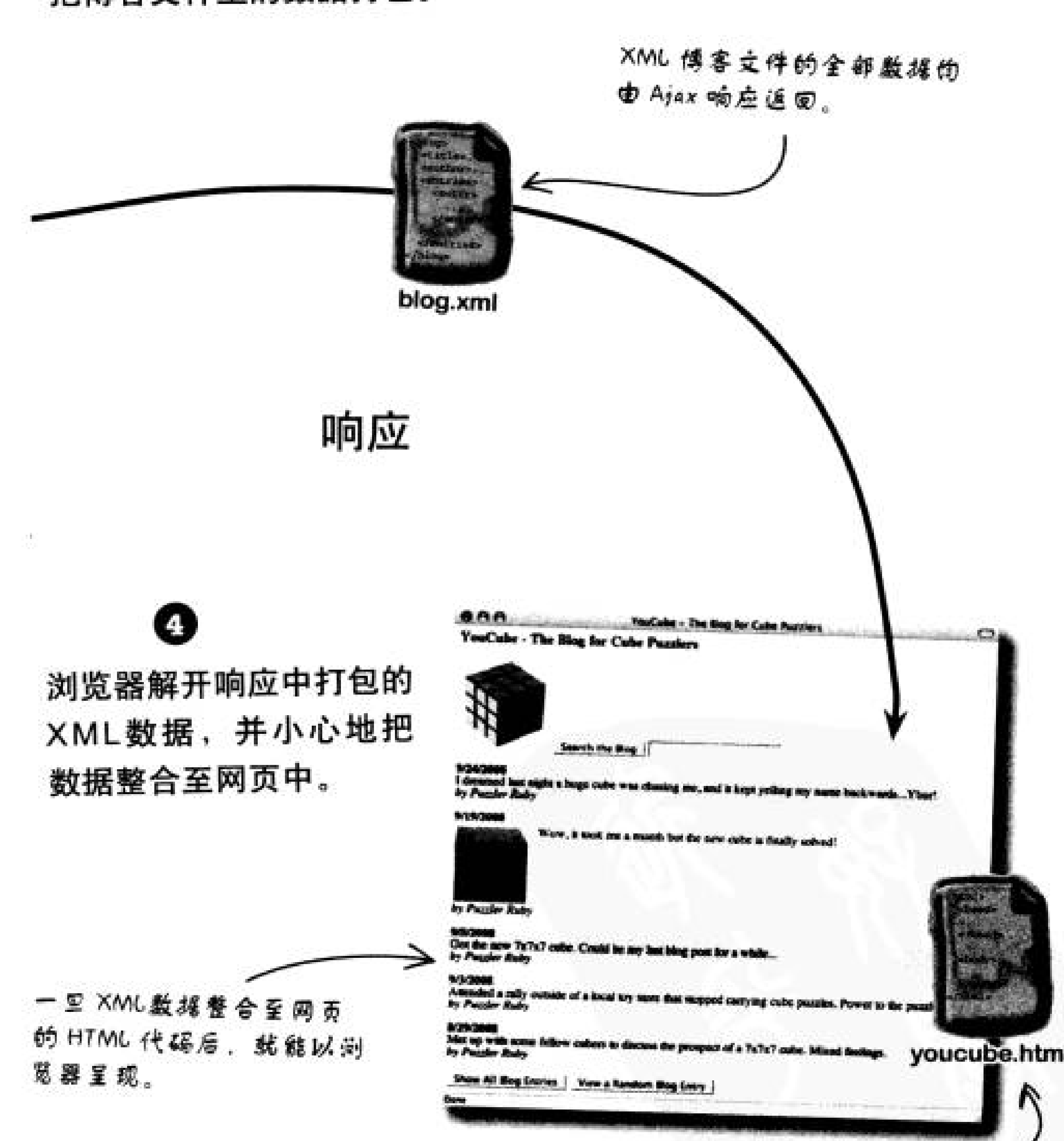
在实行客户端浏览器与服务端的数据通信时，Ajax以**请求**（request）与**响应**（response）为中心概念。

3

服务器创建给浏览器的响应，
把博客文件里的数据打包。



XML 博客文件的全部数据均由 Ajax 响应返回。



响应

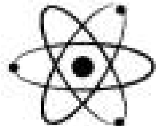
服务器端脚本（不是 JavaScript）有时需要处理 Ajax 请求并预备响应的数据。

4

浏览器解开响应中打包的 XML 数据，并小心地把数据整合至网页中。

一旦 XML 数据整合至网页的 HTML 代码后，就能以浏览器呈现。

负责创建 Ajax 请求并处理响应的 JavaScript 代码就在网页内运作。



动动脑

你认为，负责处理 Ajax 请求与响应的 JavaScript 代码是什么样的程序代码呢？

援助Ajax的JavaScript：XMLHttpRequest

JavaScript内置一个称为`XMLHttpRequest`的对象，用于发起Ajax请求并处理Ajax响应。这个对象非常复杂，包含许多通力合作以支持Ajax的方法与特性。

还有其他 XMLHttpRequest 的方法与特性，但我们列出最重要的几个。

`readyState`

请求的状态代码：0（未初始）、1（开启）、2（已传送）、3（接收中）、4（已载入）。

`status`

HTTP 的请求状态代码，例如 404（找不到文件）或 200（OK）。

合用上述两个特性，可判断 Ajax 请求是否以合格的响应结束。

`abort()`

取消请求。

这个方法只用在需要取消 Ajax 请求的时候。

`open()`

准备请求，指定请求的类型、URL 及其他细节。

`send()`

传送请求，交给服务器处理。

上述两个方法合作准备 Ajax 请求，而后送给服务器。

`onreadystatechange`

请求状态改变时会被调用的函数引用。

这个特性的独特处，在于它存储一个引用，引用于 Ajax 请求的状态改变时被调用的自定义事件处理器 —— 这个事件处理器就是处理响应的地方。

`responseText`

由服务器返回的响应数据，格式为纯文本字符串。

`responseXML`

由服务器返回的响应数据，格式为XML节点树构成的对象。

上述两个特性存储服务器返回的 Ajax 响应数据。

XMLHttpRequest 相当复杂

XMLHttpRequest 的威力强大得不可思议，同时又有惊人的灵活性。伴随着威力与灵活性而来的就是**复杂度**（complexity），即使是最基础的 Ajax 请求，也需要相当数量的 JavaScript 代码。部分原因在于浏览器的不一致，但各种微调对象行为的选项，又何尝未曾推波助澜？我们明明只需要快速动态地移动数据的方式。

XMLHttpRequest 对象很强大，但使用上有点麻烦。

以下面的范例而言，XMLHttpRequest 对象要在数种浏览器上运作，需要这些代码：

```
var request = null;
if (window.XMLHttpRequest) {
  try {
    request = new XMLHttpRequest();
  } catch(e) {
    request = null;
  }
// Now try the ActiveX (IE) version
} else if (window.ActiveXObject) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  // Try the older ActiveX object for older versions of IE
  } catch(e) {
    try {
      request = new ActiveXObject("Microsoft.XMLHTTP");
    } catch(e) {
      request = null;
    }
  }
}
```

范例代码必须尝试几种不同的方式，以创建 XMLHttpRequest 对象，因为有些浏览器（IE）支持它的方式不一样。

try-catch 语句是 JavaScript 的高级错误处理机制，能让脚本优雅地处理运行时错误（runtime error）。

创建XMLHttpRequest对象的问题，在于浏览器必须提供自己的对象实现。好消息是所有浏览器使用的方法和特性都一致——对象的创建方式才是浏览器间需要考量的差异。

创建 XMLHttpRequest 对象后，换成设定处理请求的函数，然后开启请求。

```
request.onreadystatechange = handler;
request.open(type, url, true); // always asynchronous (true)
```

服务器响应我们的请求时会被调用的自定义函数。

开启请求，让请求准备被传送，同时决定请求的类型（GET 或 POST）。

开启请求时，你必须指定请求的类型（GET 或 POST），还有服务器的 URL，以及请求是否“异步”。**非同步请求**（asynchronous request）在后台运作，不用让脚本等待，所以几乎所有 Ajax 请求都是异步请求。

关于 GET 与 POST

Ajax 请求的**类型**（type）非常重要，不只反映传送到服务器的事物，也反映请求的**意图**。请求类型，或称请求方法（method）之一，是 GET，主要用于从服务器取得数据，而不影响服务器上的其他东西。另一种请求类型，POST，通常与传送数据到服务器有关，传送后的服务器状态通常有所改变，以响应传入的数据。

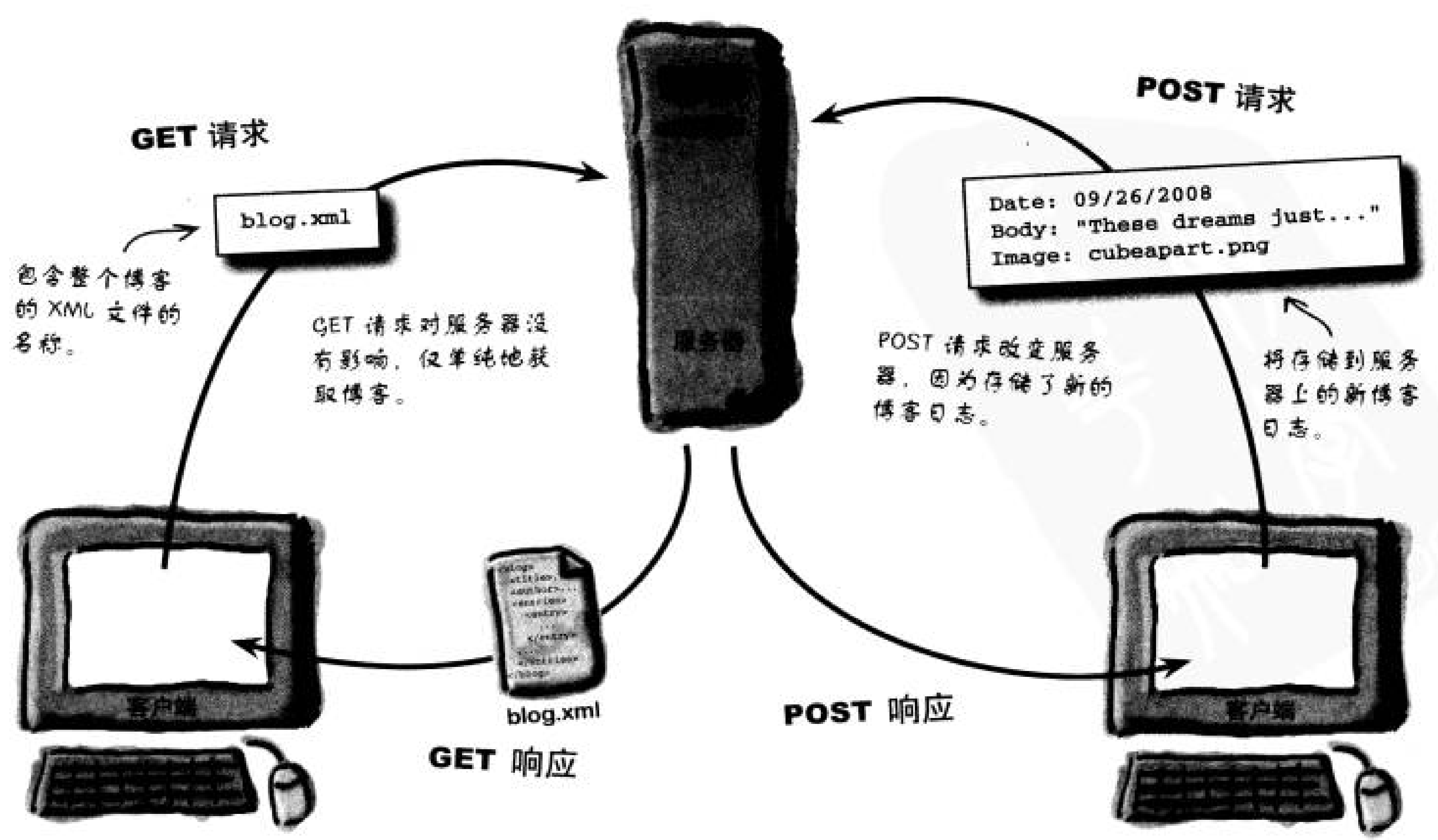
Ajax使用的两类请求为GET与POST，与送出HTML表单时相同。

GET

不会改变服务器上任何事物的数据获取方式。如果需要，还是能透过URL传入小量数据给服务器。GET非常适合从服务器上的XML文件获取博客数据。

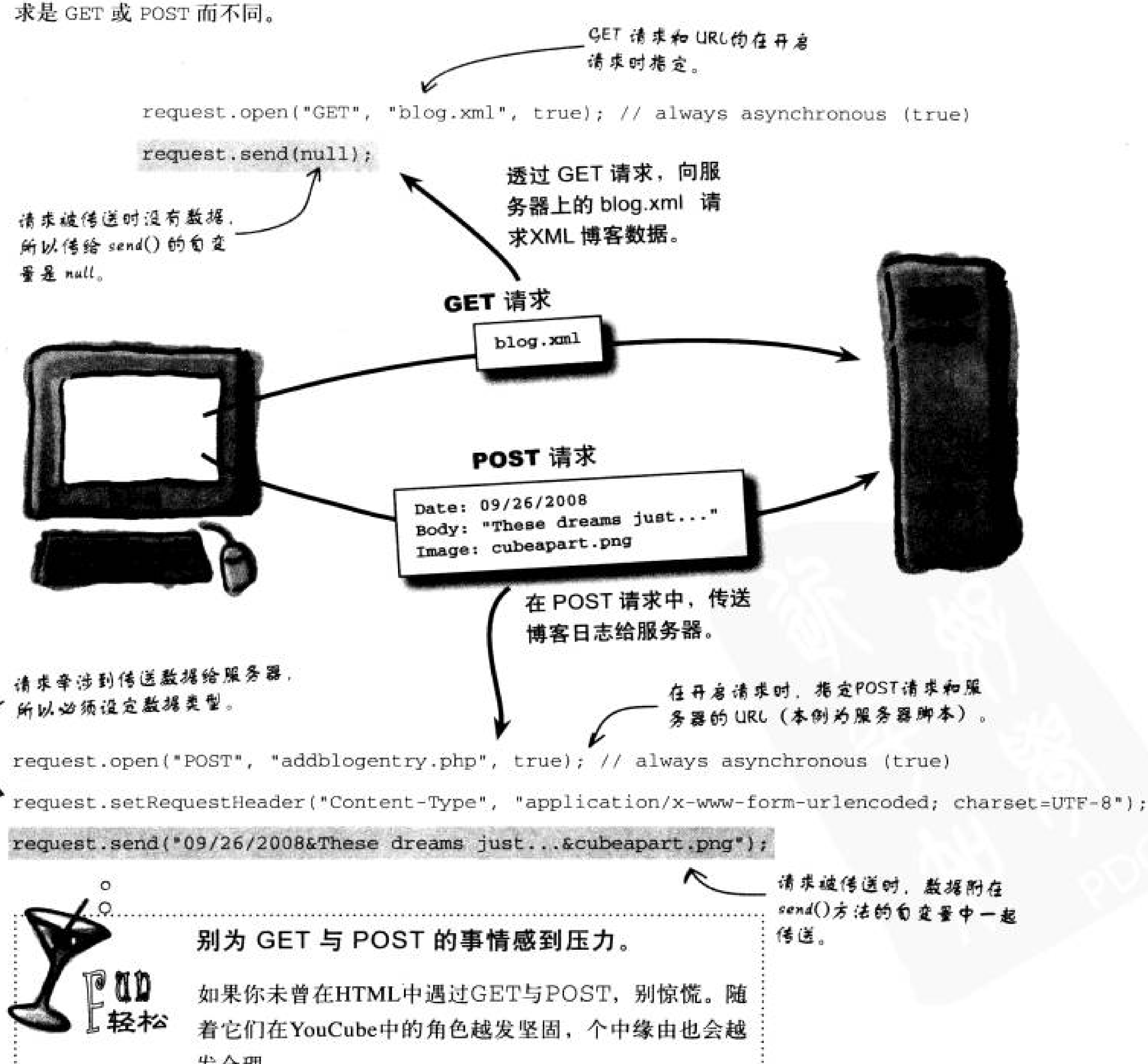
POST

此处传送给服务器的数据，因为某种方式会改变服务器状态，例如存储数据到数据库中。数据还是能在响应中返回。POST是使用网页表单动态新增博客日志这类任务的理想帮手。



GET 或 POST？使用 XMLHttpRequest 的请求

决定了请求类型并于开启请求时指定类型后，终于到了传送请求给服务器处理的时间了。实际送出请求的代码因请求是 GET 或 POST 而不同。




```
request.open("GET", "blog.xml", true); // always asynchronous (true)
request.send(null);
```

```
request.open("POST", "addblogentry.php", true); // always asynchronous (true)
request.setRequestHeader("Content-Type", "application/x-www-form-urlencoded; charset=UTF-8");
request.send("09/26/2008&These dreams just...&cubeapart.png");
```

Fun 轻松

别为 GET 与 POST 的事情感到压力。

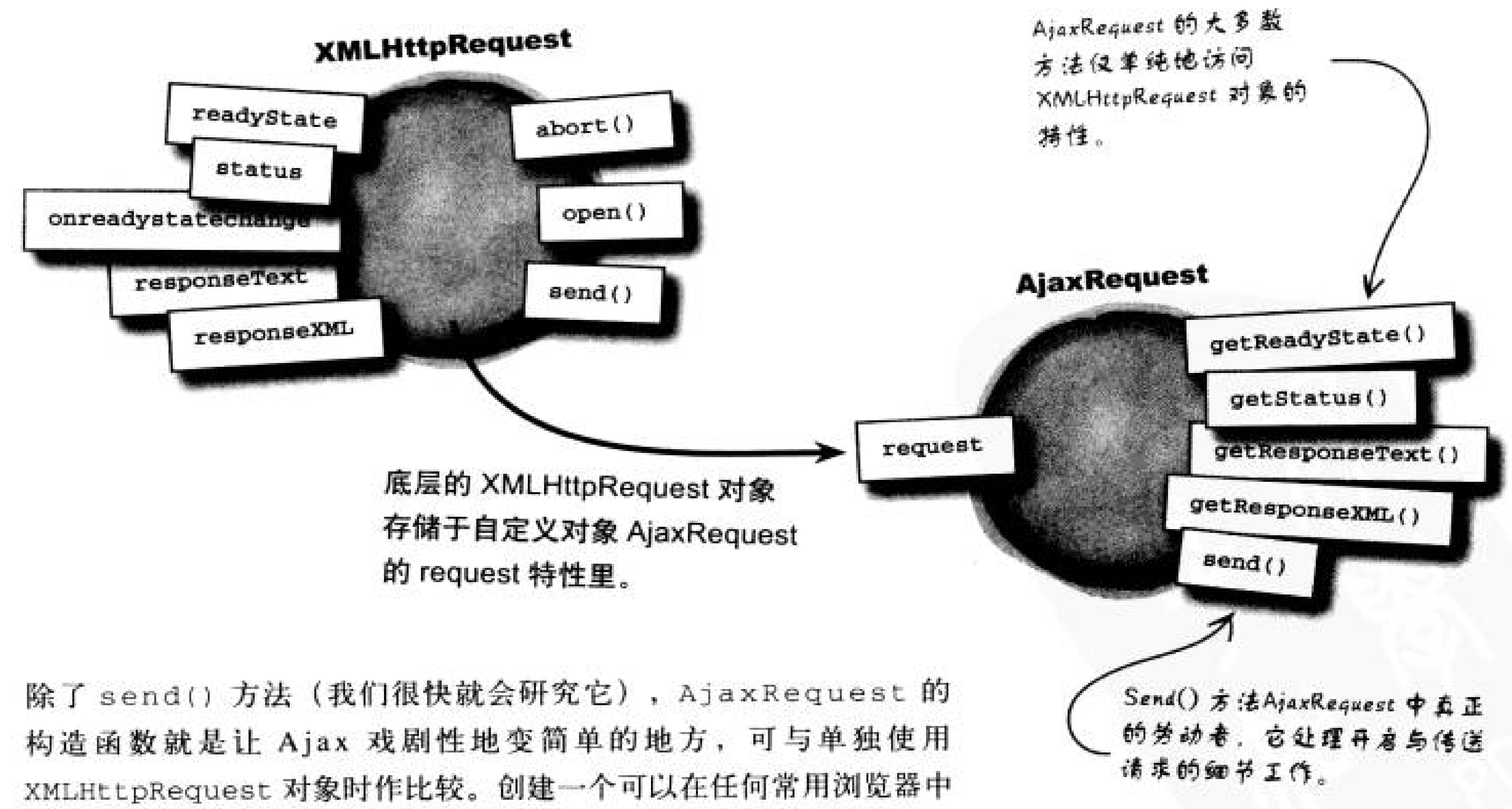
如果你未曾在HTML中遇过GET与POST，别惊慌。随着它们在YouCube中的角色越发坚固，个中缘由也会越发合理。

减轻XMLHttpRequest带来的痛苦

虽然 XMLHttpRequest 对象极为强大，却有着极为陡峭的学习曲线，想必各位都有同感。不仅如此，这个对象还需要把一定数量的“样板”代码塞到使用它的 Ajax 应用程序里。故而出现许多第三方函数库，让 XMLHttpRequest 对象的使用简单一点。这些函数库很多都扩展了 JavaScript 的功能，固然很好，但也需要更多学习。

自定义的AjaxRequest对象减轻了制作 Ajax 请求的痛苦过程。

因此，对于YouCube有帮助的策略，应是创建最小量的自定义对象作为XMLHttpRequest的助手，让我们单纯地专注于“使用Ajax工作”，而不是花费时间与XMLHttpRequest缠斗或全神贯注地操纵某个第三方函数。我们的自定义对象AjaxRequest采用最少量极简的方式，让 XMLHttpRequest对象更好运用。



除了 send() 方法（我们很快就会研究它），AjaxRequest 的构造函数就是让 Ajax 戏剧性地变简单的地方，可与单独使用 XMLHttpRequest 对象时作比较。创建一个可以在任何常用浏览器中发起 Ajax 请求的 AjaxRequest 对象，只需要：

```
var ajaxReq = new AjaxRequest();
```

AjaxRequest 的构造函数自动计算创建底层 XMLHttpRequest 对象时的种种错综复杂之处。

JavaScript 冰箱磁铁

自定义对象 AjaxRequest 把标准对象 XMLHttpRequest 包裹起来，为 Ajax 请求的传送及响应提供简单许多的处理接口。问题在于，这个对象中的 send()方法漏掉了一些关键代码。请使用下面的磁铁完成这个方法。

```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
  if (this.request != null) {
    // Kill the previous request
    this.request.abort();

    // Tack on a dummy parameter to override browser caching
    url += "?dummy=" + new Date().getTime();

    try {

      this.request.onreadystatechange = ......................;

      this.request.open(................,.............., true); // always asynchronous (true)

      if (type.toLowerCase() == "get") {
        // Send a GET request; no data involved

        this.request.send(..............);

      } else {
        // Send a POST request; the last argument is data

        this.request.setRequestHeader("Content-Type", ..............................);

        this.request.send(........................);

      }
    } catch(e) {
      alert("Ajax error communicating with the server.\n" + "Details: " + e);
    }
  }
}
```

handler

url

null

type

postDataType

postData

JavaScript 冰箱磁铁解答

自定义对象 AjaxRequest 把标准对象 XMLHttpRequest 包裹起来，为 Ajax 请求的传送及响应，提供简单许多的处理接口。问题在于，这个对象中的 send() 方法漏掉了一些关键代码。请使用下面的磁铁完成这个方法。

传送 Ajax 请求的 send() 方法，它的细节就在自变量里。

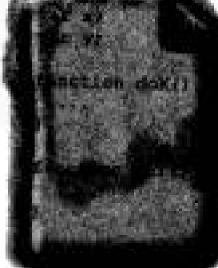
```
AjaxRequest.prototype.send = function(type, url, handler, postDataType, postData) {
  if (this.request != null) {
    // Kill the previous request
    this.request.abort();

    // Tack on a dummy parameter to override browser caching
    url += "?dummy=" + new Date().getTime();

    try {
      this.request.onreadystatechange = handler ;

      this.request.open( type , url , true); // always asynchronous (true)

      if (type.toLowerCase() == "get") {
        // Send a GET request; no data involved
        this.request.send( null );
      } else {
        // Send a POST request; the last argument is data
        this.request.setRequestHeader("Content-Type", postDataType );
        this.request.send( postData );
      }
    } catch(e) {
      alert("Ajax error communicating with the server.\n" + "Details: " + e);
    }
  }
}
```

自定义的处理器函数将被调用来处理服务器对请求的响应。

send() 的 type 自变量决定是 GET 或 POST 请求。

只在 POST 请求时，才传送数据至服务器。

这个方法的代码，与AjaxRequest的构造函数及其他方法，存储在外部JavaScript文件ajax.js里。



ajax.js

理解 Ajax 请求

自定义的 AjaxRequest 对象由构造函数和数个方法组成，其中有一个方法特别重要。send() 方法全权负责准备 Ajax 请求和对服务器发出请求。所有使用 send() 发起的 Ajax 请求必为 GET 或 POST 请求与 HTML 表单送出的请求一致。差别在于 Ajax 请求不需全面重新载入网页。

AjaxRequest

request

getReadyState()

getStatus()

getResponseText()

getResponseXML()

send()

send(type, url, handler, postDataType, postData)

type

请求的类型，GET 或 POST。

url

服务器的 URL（YouCube 一例中为 blog.xml）。如有需要，数据也可以包装在 URL 里。

postDataType

被传送的数据类型（只用于POST，GET不需要）。

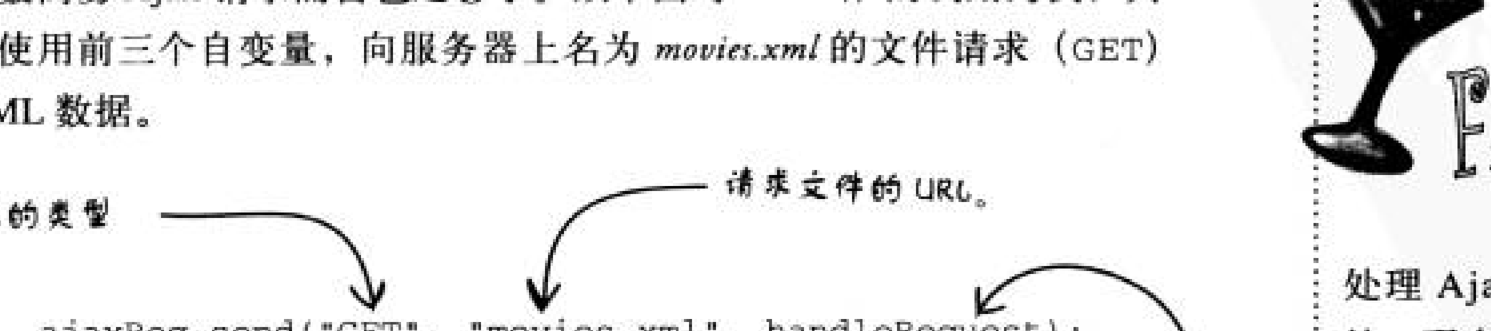
handler

用于处理响应的回调函数。

postData

被传送的数据（只用于POST，GET不需要）。POST数据能以数种格式送出。

所有 Ajax 请求都关系到相似信息片段，虽然 GET 请求省略了最后两项可选自变量。所以，前三个自变量对 send() 而言尤其重要，对大多数简易 Ajax 请求而言也足够了。以下面对 send() 的调用为例，其中使用前三个自变量，向服务器上名为 *movies.xml* 的文件请求（GET）XML 数据。



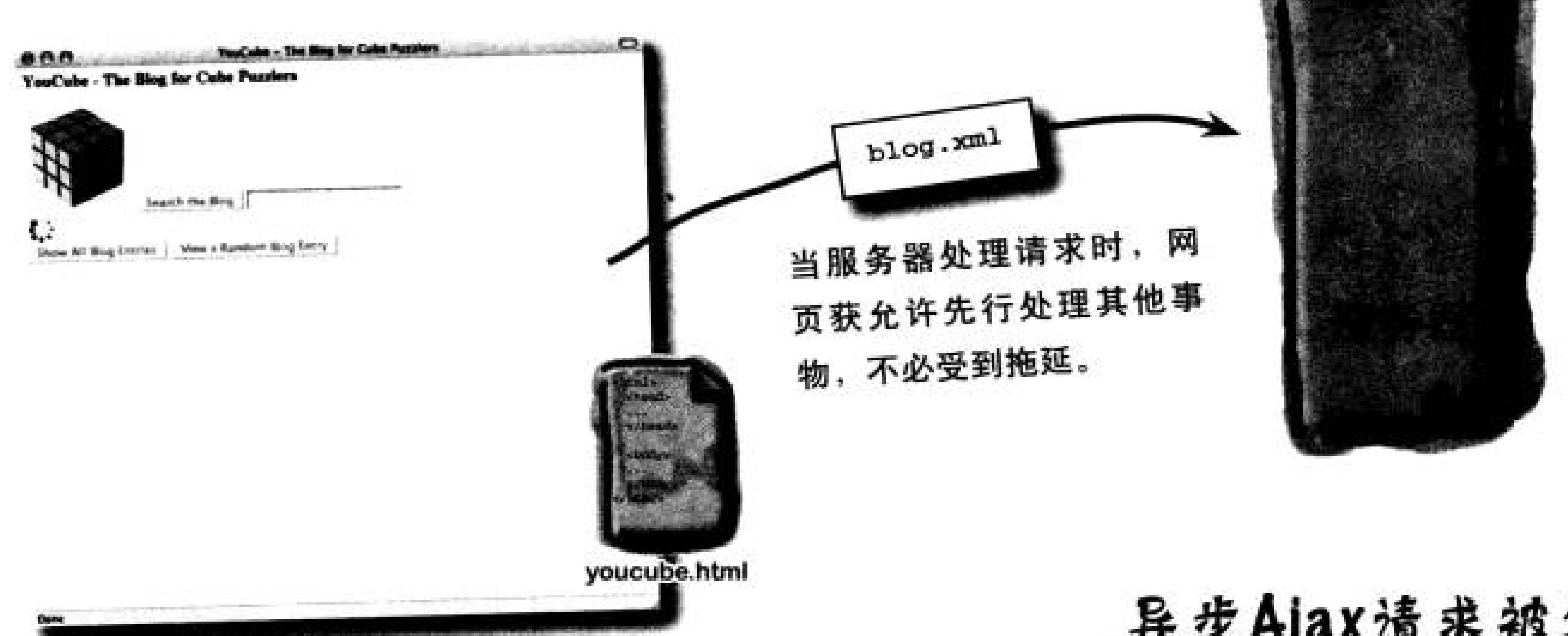
Fun 轻松

别害怕请求的处理。

我们很快就要进入如何处理 Ajax 请求的曲折奥妙之处。现在，你只要了解**自定义的请求处理器函数**必须为请求而设置，并于请求完毕时被调用，这样就够了。

请求跑上服务器运动场

调用 `AjaxRequest` 对象的 `send()` 方法后，有个 Ajax 请求被传送到服务器，网页则先行处理网页事务，同时服务器也在处理请求。这就是 Ajax 的“异步”发光发热的地方。如果请求需要同步，网页只好先行冻结、无法动作，直到服务器返回响应。但因为这里的请求**异步**处理，网页遂无需暂停，用户体验也不会为此滞塞。



异步Ajax请求被处理时，网页不需冻结行动以等待服务器处理请求。

只因为网页并未在处理请求时冻结，不代表用户真的能做什么具生产力的行为。一切取决于网页。在 YouCube 的范例中，若要成功地查看博客，完全仰赖 Ajax 响应从服务器带来的博客数据。此时，用户体验还是与 Ajax 响应紧紧相系。

复习要点

- `XMLHttpRequest` 对象是处理 Ajax 请求的**标准对象**，但实在有点不好使用。
- **自定义**`AjaxRequest`对象提供了使用Ajax，但不需直接面对`XMLHttpRequest`对象的便利方式。
- Ajax请求必为两种类型之一，`GET`或`POST`，类型由传送给服务器的数据，也是根据数据如何影响服务器而决定。

问：实践 Ajax 请求时，一定需要 AjaxRequest 对象吗？

答：不一定。XMLHttpRequest 对象直接用于产生 Ajax 请求并处理它们的响应，也不会有什么问题。但是明明有比较简单的 AjaxRequest 对象可用，为什么会有人想开倒车呢？这个对象并非什么惊天动地的大改变——它只是个便利对象，代替我们负责繁忙的 Ajax 请求构成事务，以协助简化运用 Ajax 的任务。

问：Ajax的“请求/响应”与 HTTP 的“请求/响应”，有什么不同？

答：HTTP的请求与响应为网络浏览器所用，从网络服务器上获取HTML网页。Ajax的请求与响应与HTTP的非常相似，但有几项关键差异：Ajax版随时可能发生，不见得牵涉到 HTML 数据的递送。事实上，Ajax最大的好处之一，就是它能用于请求任何类型的数据。

问：所以Ajax让我们能动态地拼出网页啰？

答：没错！这就是Ajax背后的主要概念。但不只是拼凑网页而已，也关系到组合网页的**时机**。Ajax的请求与响应为实时发生，通常不会干涉网页的使用性。换句话说，当实际上只有网页上的一小块需要更新时，用户不用停下来等着整个网页重新载入。那块区域可于“背景”载入，用户则可继续浏览网页其他部分并与之交互。

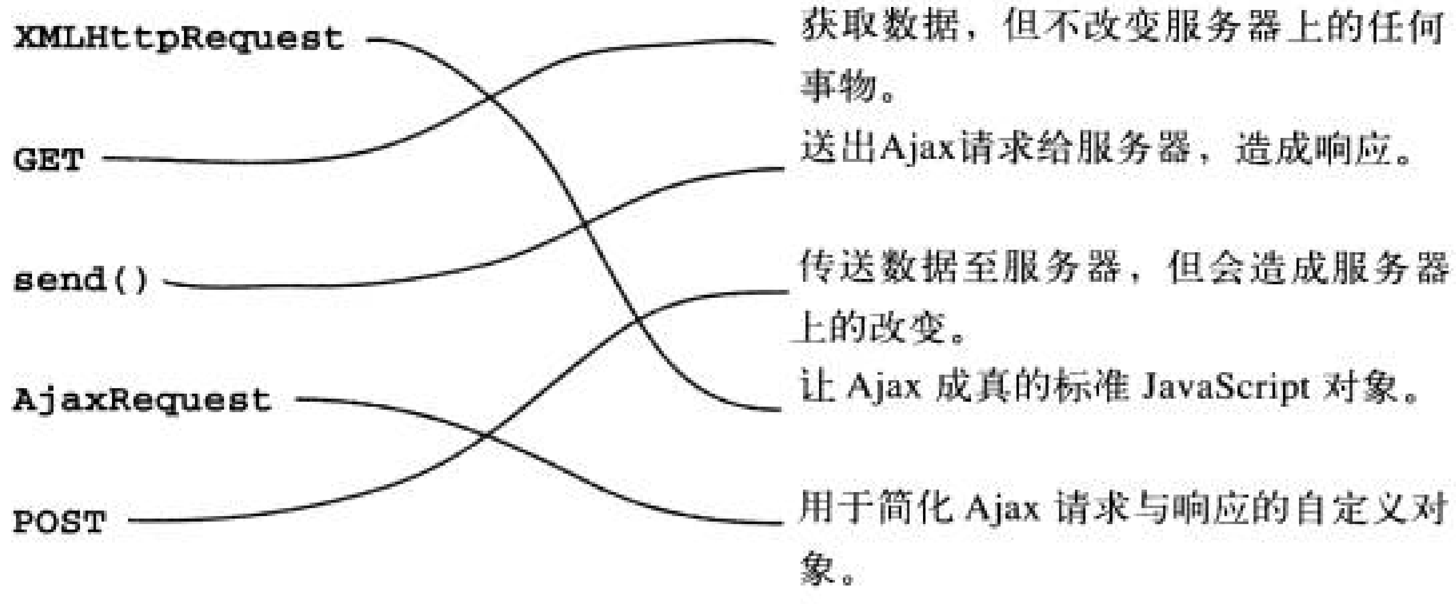
问：GET与POST，它们和这一切有什么关系呢？

答：GET与POST决定了服务器处理Ajax请求的特定方式。不过，在随时动态地请求任何类型的数据上，它们没有任何差别。关于GET与POST的主要区别，在于服务器是否经历（缘自数据的）状态改变，例如存储数据至数据库。倘若如此，则是POST；否则，即为GET。

请把每个与 Ajax 相关的程序代码与它的用途连起来。

XMLHttpRequest	获取数据，但不改变服务器上的任何事物。
GET	送出 Ajax 请求给服务器，造成响应。
send()	传送数据至服务器，但会造成服务器上的改变。
AjaxRequest	让 Ajax 成真的标准 JavaScript 对象。
POST	用于简化 Ajax 请求与响应的自定义对象。

请把每个与 Ajax 相关的程序代码与它的用途连起来。



AjaxRequest

XMLHttpRequest

自定义对象AjaxRequest的角色是标准对象XMLHttpRequest的“外包装”，用于简化 Ajax 的使用。

GET

blog.xml

send() 方法发出 Ajax 请求（必为 GET 或 POET）。

```
Date: 09/26/2008
Body: "These dreams just..."
Image: cubeapart.png
```

send()

POST

交互式网页就从请求对象开始

无论Ajax如何使用，或者试图用Ajax取用何种数据，任何数据的Ajax通信都从“请求”开始。所以，Ruby把 YouCube转换为数据驱动程序的第一项任务，就是对包含博客数据的XML文件发出一个Ajax请求。

磨笔上阵

请设计创建`AjaxRequest`对象的代码，然后用它送出对XML博客数据的请求。

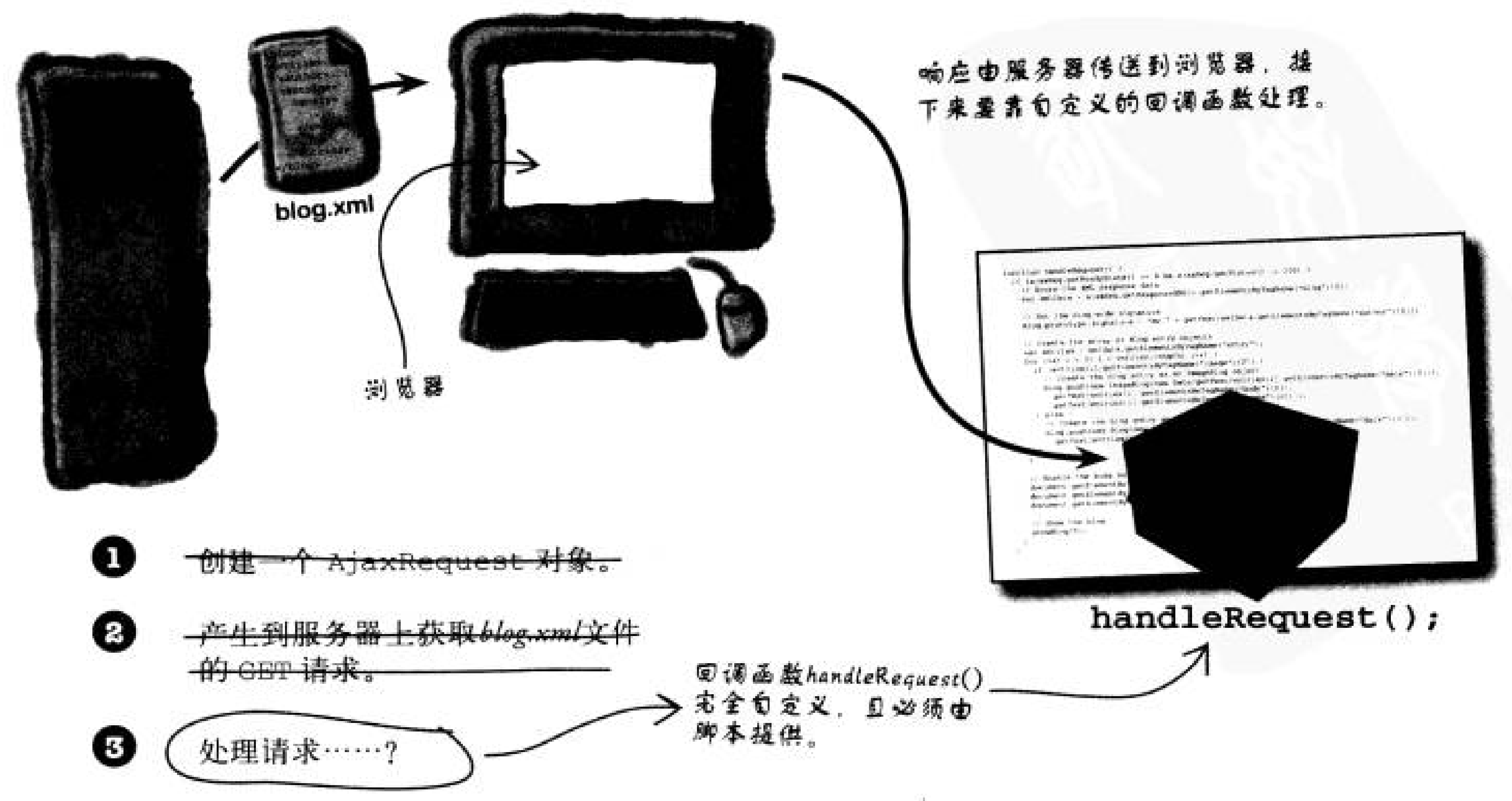
....

....

完工后再呼唤我

送出 Ajax 请求后，浏览器的角色改变了 —— 它不再等待来自服务器的响应。但因为 Ajax 请求通常为**异步**发生，用户可以继续与网页交互，同时浏览器也在幕后等待响应传来。换句话说，Ajax 请求不会在服务器处理请求时暂停网页。一旦请求在服务器上处理完毕，它的响应则在 JavaScript 代码中使用回调的请求处理器函数处理。

客户端脚本使用自定义的回调函数处理Ajax请求的响应。



处理响应……天衣无缝

自定义的请求处理器回调函数，本例中为 `handleRequest()`，在Ajax请求结束后立刻被调用。除了表示请求已完全成功，这个函数的工作还包括根据服务器返回的响应数据而行动。

我知道请求处理器被调用来处理 Ajax 响应，但它如何访问响应数据呢？

AjaxRequest 对象的方法具有对 Ajax 响应数据的访问权。

请求处理器函数提供对Ajax响应所返回数据的访问，透过两个`AjaxRequest`对象的方法：`getResponseText()`与`getResponseXML()`。

AjaxRequest

request
getReadyState()
getStatus()
getResponseText()
getResponseXML()
send()

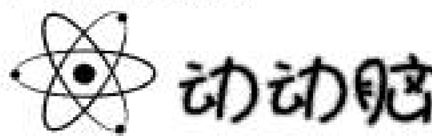
getResponseText()

以纯文本的格式取得 Ajax 响应里的访问。

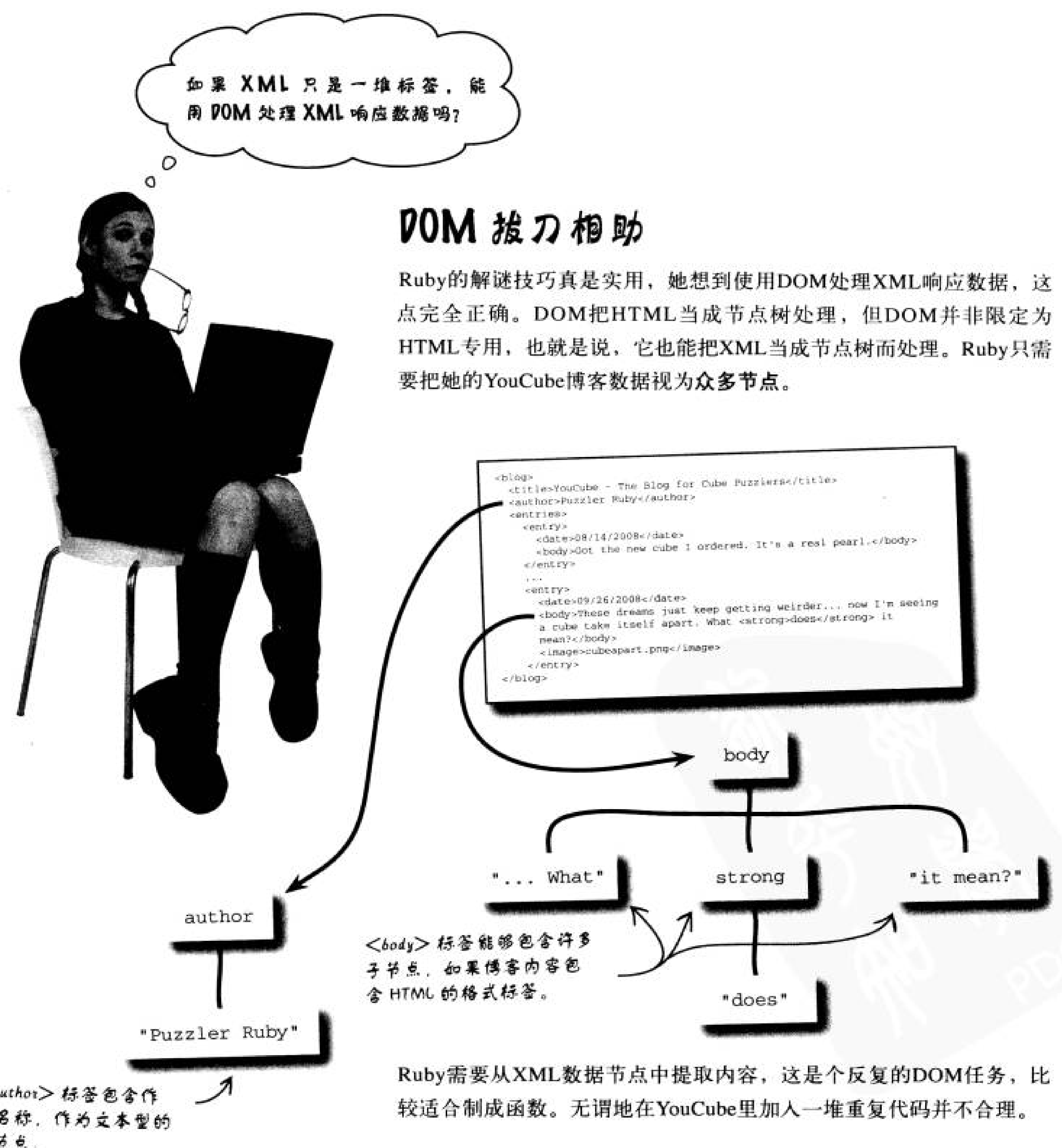
getResponseXML()

以结构XML代码的格式取得Ajax响应里的数据。

只有我们提出的其中一个方法，能访问任何给定响应的可行数据，也就是由数据格式决定应该使用的方法。所以，`getResponseXML()`在响应数据为XML时使用，此时的`getResponseText()`无法返回有意义的数据。如果数据是纯文本，而不是 XML，则情况相反。



我们知道 XML 代码的结构很像 HTML 代码，你可以如何在请求处理器里访问 XML 的博客数据呢？



DOM 拔刀相助

Ruby的解谜技巧真是实用，她想到使用DOM处理XML响应数据，这点完全正确。DOM把HTML当成节点树处理，但DOM并非限定为HTML专用，也就是说，它也能把XML当成节点树而处理。Ruby只需要把她的YouCube博客数据视为**众多节点**。

Ruby需要从XML数据节点中提取内容，这是个反复的DOM任务，比较适合制成函数。无谓地在YouCube里加入一堆重复代码并不合理。

放大 getText() 函数

自定义的 getText() 函数处理深探 DOM 树中元素的单调工作，并拉出所有元素内容。

```
function getText(elem) {
  var text = "";
  if (elem) {
    if (elem.childNodes) {
      for (var i = 0; i < elem.childNodes.length; i++) {
        var child = elem.childNodes[i];
        if (child.nodeValue)
          text += child.nodeValue;
        else {
          if (child.childNodes)
            if (child.childNodes[0].nodeValue)
              text += child.childNodes[0].nodeValue;
        }
      }
    }
  }
  return text;
}
```

elem（元素）自变量是欲提取其内容的元素名称。

以循环处理所有元素的子节点。

把子节点内容加到 text 变量里。

如果子节点内还有子节点，只抓出第一个子节点的内容，然后继续处理。

返回 text 变量，其中包含了元素所有子节点的内容。

磨笔上阵

假设 XML 响应数据已经全都存储在变量 xmlData 中，请写出设置 YouCube 签名至 XML 标签 <author> 的程序代码。

....

磨笔上阵解答

假设 XML 响应数据已经全部存储在变量 xmlData 中，请写出设置 YouCube 签名至 XML 标签 <author> 的程序代码。

```
Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);
```

signature 是个类特性，必须透过 Blog 的 prototype 对象设定。

使用自定义的 getText() 协助函数，以提取 <author> 标签的内容。

在XML数据里应该只有一个 <author>标签，所以直接提取用第一个元素。

Ajax处理函数真情指数

本周主题：

Ajax 的请求处理器 handleRequest() 大告白

HeadFirst：我们听说你很擅长响应 Ajax 请求。请问，需要哪些技能呢？

handleRequest()：当 Ajax 请求处理完毕，我就被找出来处理响应，响应通常包含服务器传来的数据。我的工作首先要确定请求在服务器上的处理没问题，这项检查通过了，我才进一步挖掘数据，并在需要时处理与网页整合的事宜。

HeadFirst：所以你其实在请求处理完毕后才被调用啰？

handleRequest()：对啊。事实上，我在请求处理过程中会被调用好几次，但多数时候，大家只对我在最后做的事情感兴趣。

HeadFirst：原来如此。你怎么知道请求什么时候完毕？

handleRequest()：这个嘛……AjaxRequest 对象有几个能让我调用来检查请求状态的方法，以确保请求成功处理，没有遇到麻烦。

HeadFirst：完成后，你又怎么知道该做什么事？

handleRequest()：这点就看我要怎么做了。记得吗？我是个自定义函数，所以每个程序中的我都不一样。

HeadFirst：为什么会这样？

handleRequest()：因为不同的应用程序使用响应数据的方式也不一样 —— 完全根据程序量身打造。我也一样。

HeadFirst：等一下，你是说有人必须为了不同程序而重新设计你吗？

handleRequest()：没错，但是非常合理。像购物车程序对 Ajax 响应的处理，想必与博客的处理非常不同。Ajax 确保服务器处理完请求后我会被调用，接下来的一切都完全量身打造了。

HeadFirst：这么说来，创建一个 Ajax 驱动的网页时，部分工作包括创建自定义的请求处理器啰？

handleRequest()：正是如此。这也是 Ajax 应用程序最主要工作的地方。

HeadFirst：真是“听君一席话，胜读十年书”。小人的疑惑顿时豁然开朗啊！

与 JavaScript 的注解者天人合一

你的任务是扮演 JavaScript 的注解者，为下列 handleRequest() 函数代码加上解释作用方式的注解。7是个魔术数字—— 是否共有7项任务能导向成功的请求呢？

```
function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Store the XML response data
    var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];

    // Set the blog-wide signature
    Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);

    // Create the array of Blog entry objects
    var entries = xmlData.getElementsByTagName("entry");
    for (var i = 0; i < entries.length; i++) {
      // Create the blog entry
      blog.push(new Blog(getText(entries[i].getElementsByTagName("body")[0]),
        new Date(getText(entries[i].getElementsByTagName("date")[0])),
        getText(entries[i].getElementsByTagName("image")[0])));
    }

    // Show the blog
    showBlog(5);
  }
}
```

与 JavaScript 的注解者天人合一解答

你的任务是扮演 JavaScript 的注解者，为下列 handleRequest() 函数代码加上解释作用方式的注解。7是个魔术数字 —— 是否共有7项任务能导向成功的请求呢？

设定博客签名为 <author> 标签的内容。

透过检查请求的状态，确定 Ajax 请求完全成功。

XML 数据里只有一个 <body> 标签，故抓取由 getElementByTagName() 返回的数组的第一个元素。

```
function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Store the XML response data
    var xmlData = ajaxReq.getResponseXML().getElementsByTagName("blog")[0];

    // Set the blog-wide signature
    Blog.prototype.signature = "by " + getText(xmlData.getElementsByTagName("author")[0]);

    // Create the array of Blog entry objects
    var entries = xmlData.getElementsByTagName("entry");
    for (var i = 0; i < entries.length; i++) {
      // Create the blog entry
      blog.push(new Blog(getText(entries[i].getElementsByTagName("body")[0]),
        new Date(getText(entries[i].getElementsByTagName("date")[0])),
        getText(entries[i].getElementsByTagName("image")[0])));
    }

    // Show the blog
    showBlog(5);
  }
}
```

取得所有entry元素，其中存储了博客日志。可以直接提触每个entry元素时，不见得必须利用entries元素。

调用 showBlog() 函数，于网页上呈现最新的5则博客日志。

为新日志创建新的 blog 对象并附加到数组末端，使用 Array 对象的 push() 方法。

1. ~~创建一个 AjaxRequest 对象。~~
2. ~~产生到服务器上获取 *blog.xml* 文件的 GET 请求。~~
3. ~~处理请求……?~~ 解决了！

YouCube改由数据驱动了

最新版的YouCube文件可至http://www.headfirstlabs.com/books/hfjs/取得。

Ruby对经过Ajax整理的YouCube真是感动到鸡皮疙瘩都站起来了（让她节省非常多时间），但她还是烦恼网页的可用性 —— 当博客数据正在载入的时候。

把数据分离到XML文件中后，博客的运作确实很不错，但有什么让用户知道博客正在忙着载入的管道吗？只要让他们知道网页正在运转就行了。

博客现在由XML数据驱动……

……但网页载入时的空白页面让有些用户大感不解。

磨笔上阵

请为 loadBlog() 函数补上遗失的代码，这个函数负责于载入博客数据时，呈现“等待”图像 *wait.gif*。

提示：使用博客的主要 div，它的 ID 是 "blog"。

```
function loadBlog() {

  .....................................................................

  ajaxReq.send("GET", "blog.xml", handleRequest);
}
```

磨笔上阵 解答

请为 loadBlog() 函数补上遗失的代码，这个函数负责于载入博客数据时，呈现“等待”图像 *wait.gif*。

提示：使用博客的主要 div，它的 ID 是 "blog"。

```
function loadBlog() {
  document.getElementById("blog").innerHTML = "<img src='wait.gif' alt='Loading...' />";
  ajaxReq.send("GET", "blog.xml", handleRequest);
}
```

innerHTML 比 DOM 简单，我们此时只需要加上一个 img 标签与几个属性而已。

载入博客数据时，等待图像完全取代博客内容。

动画文件wait.gif用于代替博客日志，让用户知道博客数据正在载入。

wait.gif

YouCube - The Blog for Cube Puzzlers

Search the Blog

Show All Blog Entries

View a Random Blog Entry

Done

问：最后一则 YouCube 博客日志的内容含有 HTML 的 <strong> 标签，真的有可能改用 XML 表现吗？

答：还记得 XML 代码能用于表现任何类型的数据吧？在这里，博客日志主体将安插至网页中，技术上确实可以包括影响主体外观的 HTML 标签。换句话说，某篇日志的主体内容可以包含 HTML 标签，标签则被视为特殊格式节点，随着 XML 代码一并传送。不过，这是个非常古怪的观点，因为当 XML 安插至网页时，我们必须在网页的 HTML 代码中重新架构 HTML 格式节点。与其走上那条路，YouCube 选择抽出所有 HTML 标签中的内容，放着格式不管。Ruby 还是能自由地为日志内容加上 HTML 格式标签，或许用于未来的新版 YouCube，但这些标签目前因为格式的原因而被忽略。不过文本内容仍会保留，这是好事。

问：Ajax 响应的准备state与status如何运作？

答：这两个特性最终都来自 XMLHttpRequest，但它们的工作是追踪请求的 **state**，例如 (0) uninitialized（未初始化）或 (4) loaded（已载入），也追踪请求的**status**，例如 404（not found，未找到）或 200（OK，没问题）。当然还能更进一步追踪这些特性，但没有必要。你只需要知道 Ajax 请求在 state 是 4(loaded) 且 status 是 200(OK) 时，表示完全成功。所以 handleRequest() 函数只在同时达成这两个状态时起身工作。

不正常的按钮

虽然 Ajax 对 YouCube 的全面检修原本发生在幕后，不应被 YouCube 的用户看见，但显然出现用户也看得到的界面问题。更精确地说，显然网页上的按钮的运作未如预期。

坏掉的按钮 = 不高兴的用户

动动脑

为什么博客的按钮不能运作？你觉得网页载入的过程中，何时发生了问题？

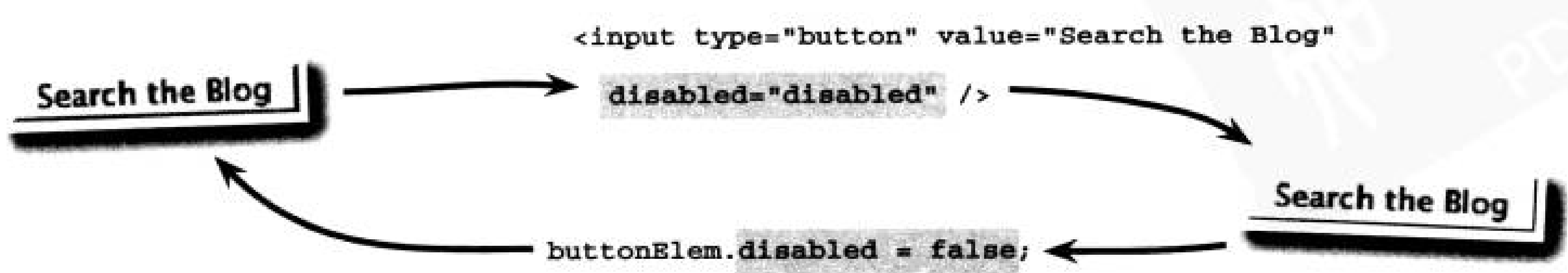
按钮需要数据

YouCube按钮的问题，在于它们只于可获得博客数据时可应用。既然博客数据现在从外部XML文件载入，会有一小段时间，非常短暂的时间，网页上没有数据。这段期间，按钮完全没有意义，只会让用户困惑。

暂时禁用按钮是个绝佳的方案。

在博客数据载入时暂时禁用按钮，是个简单又优雅的按钮问题解决方式。既然在网页首度载入时Ajax才发出载入博客数据的请求，按钮可以先被禁用，而后于`handleRequest()`函数中启用，这个函数知道Ajax请求已经处理完毕。

要实际禁用按钮，我们需要使用`<input>`标签的`disabled`属性，于 HTML代码中设定这个属性为`"disabled"`，以禁用按钮。相反地，JavaScript代码中需设定这个属性为`false`以启用按钮。



JavaScript 冰箱磁铁

使用下面的磁铁完成 YouCube 的网页代码，让按钮先被禁用，直到博客数据完成载入后再启用。有些磁铁不只使用一次。

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>

    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // Enable the blog buttons

          document.getElementById(.......................).........................=.................;

          document.getElementById(.........................).........................=.................;

          document.getElementById(.............................).........................=.................;

          ...
        }
      }
      ...
    </script>
  </head>

  <body onload="loadBlog();">
    <h3>YouCube - The Blog for Cube Puzzlers</h3>
    <img src="cube.png" alt="YouCube" />
    <input type="button" id="search" value="Search the Blog"

      .........................=.........................  onclick="searchBlog();" />

    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value="Show All Blog Entries"

      .........................=.........................  onclick="showBlog();" />

    <input type="button" id="viewrandom" value="View a Random Blog Entry"

      .........................=.........................  onclick="randomBlog();" />

  </body>
</html>
```

true

"search"

"viewrandom"

disabled

false

"showall"

"disabled"

JavaScript 冰箱磁铁解答

使用下面的磁铁完成 YouCube 的网页代码，让按钮先被禁用，直到博客数据完成载入后再启用。有些磁铁不只使用一次。

```
<html>
  <head>
    <title>YouCube - The Blog for Cube Puzzlers</title>

    <script type="text/javascript" src="ajax.js"> </script>
    <script type="text/javascript" src="date.js"> </script>

    <script type="text/javascript">
      ...
      function handleRequest() {
        if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
          ...
          // Enable the blog buttons

          document.getElementById( "search" ). disabled = false ;

          document.getElementById( "showall" ). disabled = false ;

          document.getElementById( "viewrandom" ). disabled = false ;

          ...
        }
      }
      ...
    </script>
  </head>

  <body onload="loadBlog();">
    <h3>YouCube - The Blog for Cube Puzzlers</h3>
    <img src="cube.png" alt="YouCube" />
    <input type="button" id="search" value="Search the Blog"
      disabled = "disabled" onclick="searchBlog();" />
    <input type="text" id="searchtext" name="searchtext" value="" />
    <div id="blog"></div>
    <input type="button" id="showall" value="Show All Blog Entries"
      disabled = "disabled" onclick="showBlog();" />
    <input type="button" id="viewrandom" value="View a Random Blog Entry"
      disabled = "disabled" onclick="randomBlog();" />
  </body>
</html>
```

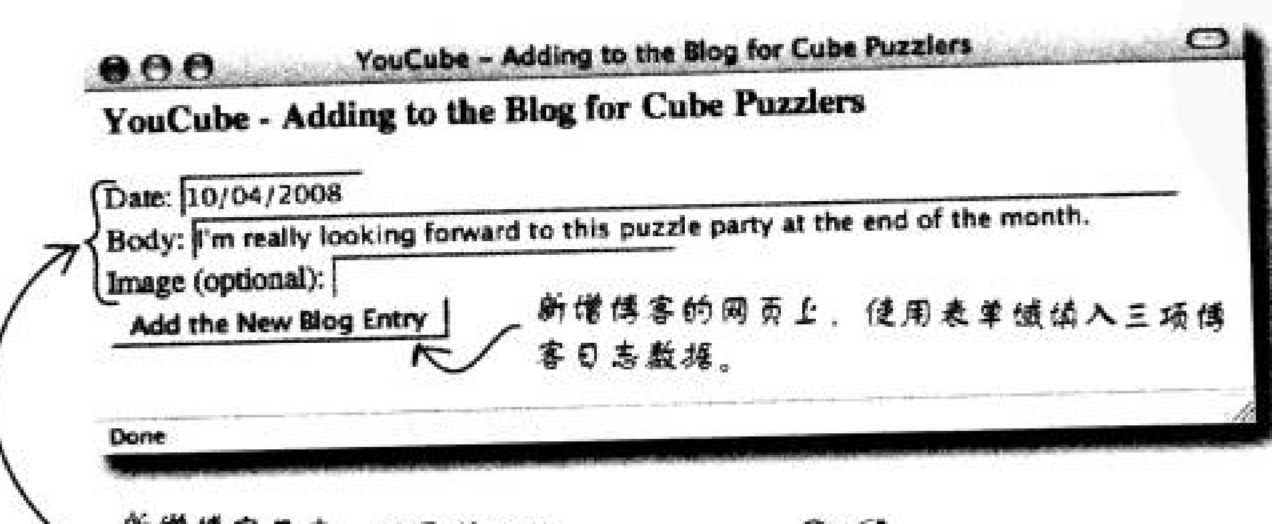
省时的网络版新增日志方案

YouCube 现在已经由动态数据所驱动，但 Ruby 还无法全盘满意。她还没享受到 YouCube 使用动态数据的真正好处，除非她能使用**基于网络的界面**（web-based interface）新增博客日志。与其编辑 XML 再加到博客上，她想直接在网页上新增日志，而且直接存储到服务器里。

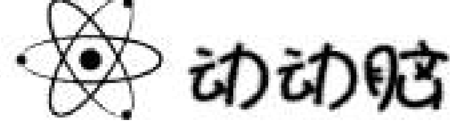
我不想再为了更新博客而一直上传（FTP）编辑好的更新文件到服务器。我想升级 YouCube，让一切都在浏览器里完成。

编辑代码 ＋ 上传文件 ＝ 逊！

Ruby想有个她专属的、用于贴上新的博客日志的网页，她只要在这个网页里填写表单。每次更新博客对她来说都是弹指小事，而且她只需要浏览器。不用准备文本编辑程序，不用寻找FTP客户端，只要带着她的满腔魔方热情就够了。



新增博客日志，只需简单地填写后并按下按钮！



透过网页用户界面，Ajax 如何用于新增博客日志呢？

写出博客数据

从 Ajax 的角度思考增加博客数据时，大概就是想象对服务器传送一个 Ajax `POST` 请求，其中包含新的博客日志数据；然后，服务器把数据写入 `blog.xml`，作为新的博客日志。此时的 Ajax 响应没有什么作用，因为没有返回的东西。

等一下，新的博客日志如何写入服务器上的 blog.xml？我还以为 JavaScript 不能“写入”文件。还有，JavaScript 不是个客户端的技术吗？

JavaScript 并非写入文件至服务器的工具。

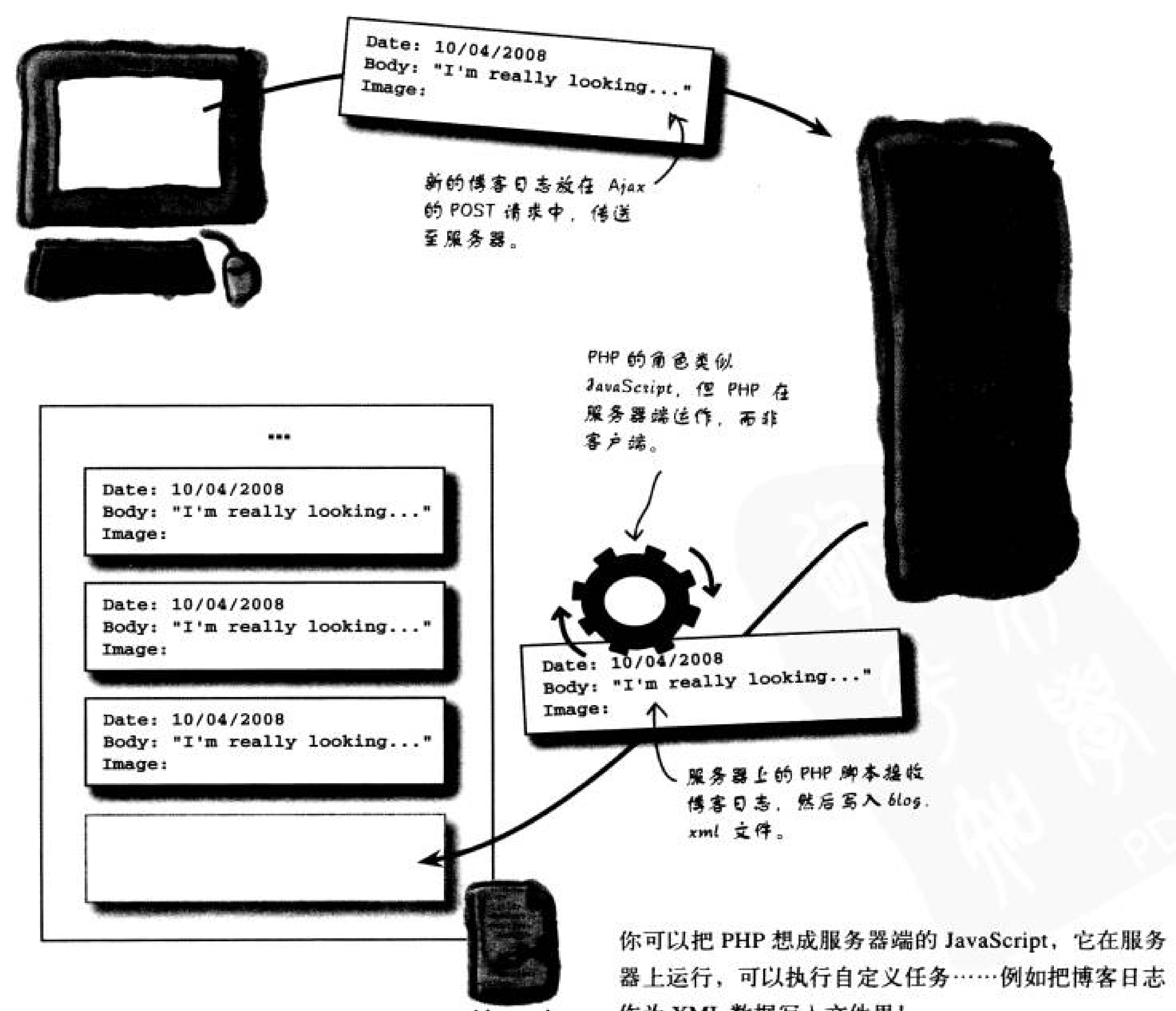
JavaScript 并非在服务器上写入 *blog.xml* 的选项。事实上，你甚至无法在服务器上运行 JavaScript。因为 JavaScript 是种客户端技术，它设计为只在浏览器上运作。这里的特殊情况中，JavaScript 无法协助我们，因为我们需要写入服务器上的某个文件。这个问题并不少见，所以服务器端技术常与 JavaScript 合用。

我们需要一种与 JavaScript 相似，但单纯在服务器上运作的技术。市面上有不少选择，但我想到一个不太复杂，而且又与 XML 数据很合得来的技术……

PHP伸出援手…

有种称为PHP的脚本语言提供了我们需要的一切，让我们写入博客数据至服务器上XML文件。实际任务包含了读取XML文件、新增日志到现有内容中，然后把整份博客日志写回原始文件。但一切又回到从客户端浏览器发出Ajax请求，以在服务器上获取新的博客日志数据。

PHP是种可在服务器上执行任务的脚本语言。



你可以把PHP想成服务器端的JavaScript，它在服务器上运行，可以执行自定义任务……例如把博客日志作为XML数据写入文件里！

PHP 上菜

在 YouCube 的服务器端，由 PHP 脚本处理新增博客日志的细节，新的博客日志将加入存储 XML 格式数据的 *blog.xml* 文件中。

```
<?php
$filename = "blog.xml";

if (file_exists($filename)) {
  // Load the blog entries from the XML file
  $rawBlog = file_get_contents($filename);
}
else {
  // Create an empty XML document
  $rawBlog = "<?xml version=\"1.0\" encoding=\"utf-8\" ?>";
  $rawBlog .= "<blog><title>YouCube - The Blog for Cube Puzzlers</
title>";
  $rawBlog .= "<author>Puzzler Ruby</author><entries></entries></blog>";
}
$xml = new SimpleXmlElement($rawBlog);

// Add the new blog entry as a child node
$entry = $xml->entries->addChild("entry");
$entry->addChild("date", $_REQUEST["date"]);
$entry->addChild("body", stripslashes($_REQUEST["body"]));
if ($_REQUEST["image"] != "")
  $entry->addChild("image", $_REQUEST["image"]);

// Write the entire blog to the file
$file = fopen($filename, 'w');
fwrite($file, $xml->asXML());
fclose($file);
?>
```

载入纯XML数据至变量$rawblog。

检查博客文件是否存在。

如果博客文件尚未存在，创建一份空白的XML博客文件。

转换原始博客数据为XML数据结构，这部分很像JavaScript的DOM树。

博客日志新增为XML数据结构的子节点。

以新的博客数据改写博客文件。

addblogentry.php

PHP脚本存储在addblogentry.php文件中。

没有蠢问题

问：我必须使用 PHP 写入文件至服务器上吗？

答： 不用，没这回事。外面有很多设计服务器端脚本的技术。像是Perl（CGI）和 Java servlet，它们也能达成 PHP 的任务。如果你已经习惯某种技术，完全可以使用它创建你的 Ajax 程序的服务器端组件。

问：我可以使用 Ajax，但完全不使用服务器端的程序吗？

答： 某些状况下，可以；但大多数时候，没办法。请记得，除了最简单的请求，所有Ajax请求都牵涉到服务器端接收来自客户端的数据，然后根据数据而行动，例如寻找数据库里的某个东西，或写入至某个文件或数据库。YouCube的主页面就是最简易Ajax请求的好例子，它简单到不需要任何服务器端脚本。大多数Ajax程序没这么好运，所以你多半需要某种程度的服务器端编程。真正的问题在于无论如何，服务器可以返回整份文件（如*blog.xml*），或于服务器处理数据（如写入数据）。幸好，许多Ajax应用程序需要的服务器端脚本都相当简单，就算你不是服务器端脚本编程技术大师，通常也看得懂。

PHP也有需要

不像 JavaScript天生就有新式浏览器的支持，服务器不见得非得支持 PHP。在你把 PHP 文件传上网络服务器前，最好确认一下你的系统管理人员或虚拟主机服务是否支持 PHP。如果不行，你需要尽可能加上这项支持，或考虑换个网络服务器。除非服务器支持 PHP，否则 YouCube 的 PHP 脚本无法运作。

运行PHP或许需要调校一下你的网络服务器。

支持 PHP 网络服务器，是你的第一个障碍。第二个障碍则是找出服务器上放置 PHP 文件的位置。许多时候，把 PHP 文件与 HTML 网页、外部 JavaScript 文件放在一起都没问题。然而，有些 PHP 的安装比较挑剔，要求 PHP 脚本存储在特殊目录下。同样地，这个问题还是要请教系统管理人员。

许多时候，你可以把PHP脚本与网页存储在相同目录里。

addblogentry.php

youcube.html blog.xml ajax.js

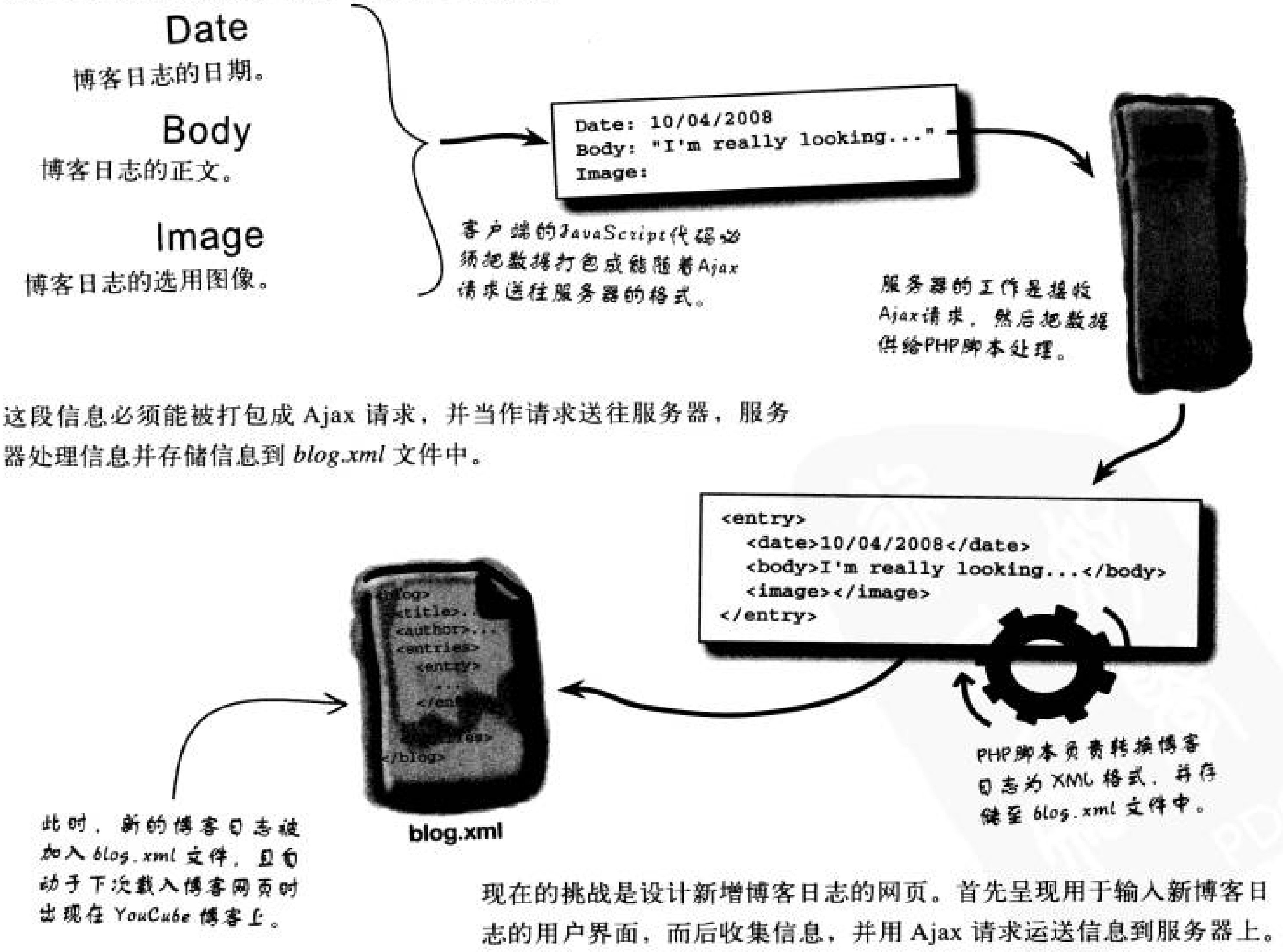
一旦你找到 PHP 文件应该存放在网络服务器上的位置时，就准备好把文件复制上去，并可继续创建新增 YouCube 博客日志的网页了。

供应数据给 PHP 脚本

有了在服务器端运作的 PHP，还有放对位置的 PHP 脚本文件，我们可以更仔细地检验 PHP 脚本的需求，才能在服务器端写入数据至 XML 文件。如此将协助我们达成设计目标——以 Ajax 请求对服务器提供完成任务所需的事项。

数据透过Ajax请求被供给PHP脚本。

PHP脚本的预期中，Ajax传来的数据将是新的博客日志。我们知道这些数据至少由两种信息组成，且有可能包含三种信息。



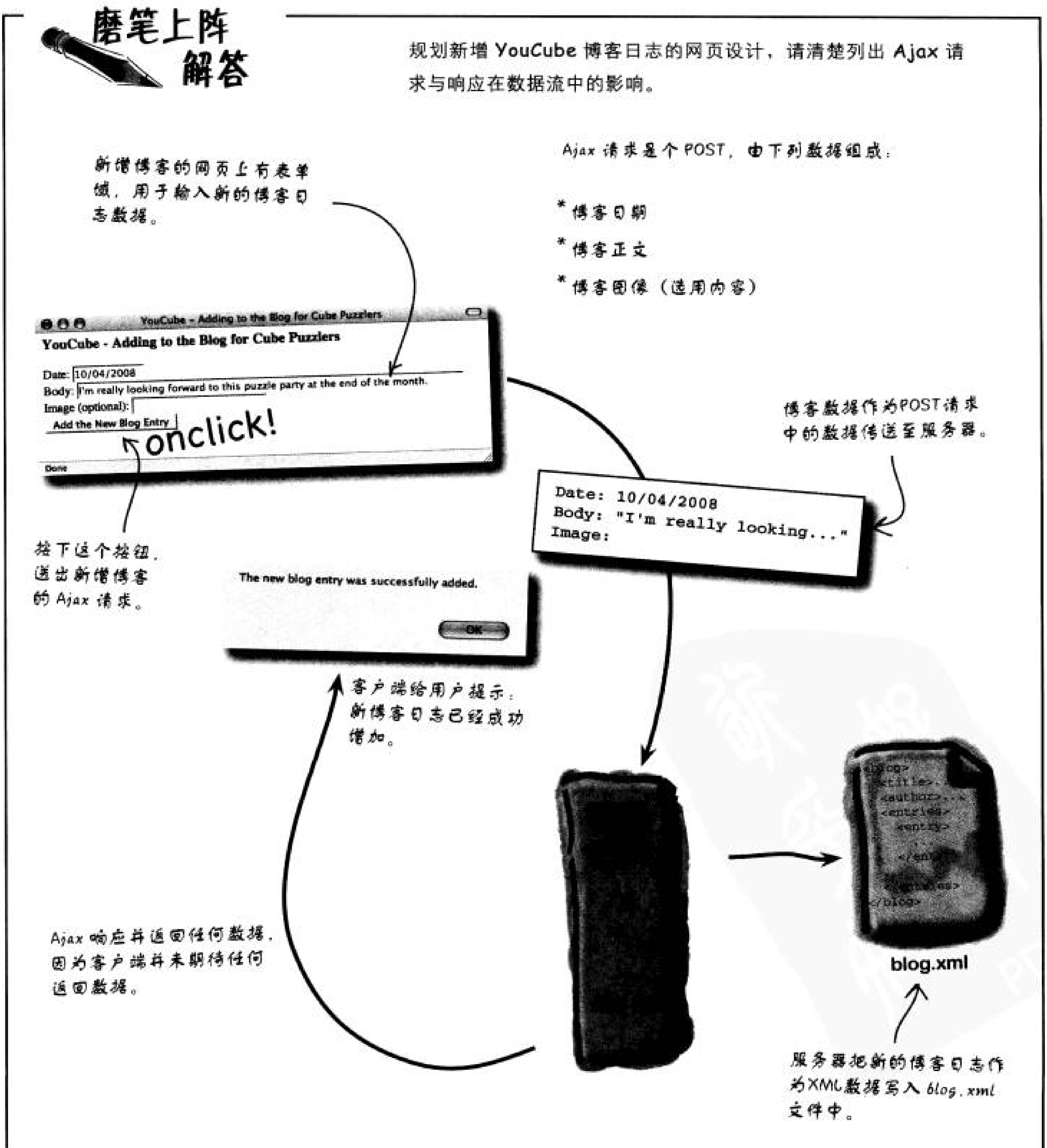
这段信息必须能被打包成 Ajax 请求，并当作请求送往服务器，服务器处理信息并存储信息到 *blog.xml* 文件中。

现在的挑战是设计新增博客日志的网页。首先呈现用于输入新博客日志的用户界面，而后收集信息，并用 Ajax 请求运送信息到服务器上。好消息是我们不需要响应，或许只要确认新博客日志成功存储即可。

规划新增 YouCube 博客日志的网页设计，请清楚列出 Ajax 请求与响应在数据流中的影响。

磨笔上阵解答

规划新增 YouCube 博客日志的网页设计，请清楚列出 Ajax 请求与响应在数据流中的影响。



组合代码：发表文章到博客服务器上

Ajax 的 POST 请求比起 GET 多了一些要求，因为它需要传送数据到服务器上。虽然 POST 请求支持不同的数据打包方式，受信赖的**数据域URL encoding** 技术已足可胜任。这项技术就是浏览器透过网页 URL传送域的数据给服务器的相同技术，特色就是区分每段数据用的 & 符号。

```
Date: 10/04/2008
Body: "I'm really looking... "
Image:
```

```
"date=10/04/2008&body=I'm really looking forward... &image="
```

一段数据由名称/值对构成。

每段数据均以 & 和其他数据区隔。

在这种数据打包方式中，每段数据需有自己的名称与值，中间插入等号（=），每对名称/值中间则以 & 区隔。这种格式称为 **URL-encoded**，其数据类型设置为 Ajax 的 POST 请求的数据类型。

它是 URL-encoded 数据的正式类型，必须在 POST 请求中指定。

```
"application/x-www-form-urlencoded; charset=UTF-8"
```

把博客日志的格式设为 URL-encoded，并设定 POST 请求的数据类型后，我们终于准备把请求的代码拼凑起来，并传送数据至服务器，让数据存储至 *blog.xml* 文件里。

把下列数据片段打包成适合 POST 请求的 URL-encoded 格式。

```
releaseDate: 01/13/1989
title: Gleaming the Cube
director: Graeme Clifford
```

....

把下列数据片段打包成适合 POST 请求的 URL-encoded 格式。

```
releaseDate: 01/13/1989
title: Gleaming the Cube
director: Graeme Clifford
```

"title=Gleaming the Cube&releaseDate=01/13/1989&director=Graeme Clifford"

没有蠢问题

问：如果 YouCube 的新增博客脚本在 Ajax 请求中不需要任何来自服务器的数据，为何还要费神处理这个请求呢？

答：因为知道请求已经完成仍然很重要。即使我们不需要服务器返回任何数据以响应请求，我们仍然非常需要知道请求是否完全成功以及何时成功。脚本才能因此决定弹出 alert 框，确认博客日志的新增成功。

问：GET 请求也能用于新增博客脚本吗？

答：技术上可以。技术上，GET 请求仍然可能传送数据至服务器，但你必须直接在请求的 URL 中指定数据。这点不是问题 —— 问题在于 GET 并非为了改变服务器状态的情况而设计。此时，服务器状态绝对因为写入博客日志到 *blog.xml* 而改变。所以 POST 请求才是正确方式，就算不为别的原因，也是因为它明确指出了与服务器通信的企图。

问：既然服务器在处理 Ajax 请求以及存储博客日志时都需要时间，如果新增的按钮在请求结束前被按下去，会发生什么问题吗？

答：是的，会有问题。每次按下新增按钮（Add the New Blog Entry），都会取消目前的 Ajax 请求并开启新请求。虽然按钮两次的人或许就是要这种效果，但用户界面如果能在请求处理期间移除按钮的选择，界面将显得更清楚。所以，新增博客日志的代码应该在处理 Ajax 请求时禁用新增按钮，等请求处理完毕才再度启用。像这样对用户界面的小改变，长久下来可能让 JavaScript 程序更直觉、更容易使用，也让用户更高兴。

问：博客数据中的空格为什么在构成 URL-encoded 字符串后就不见了？看来空格有时造成 URL 的问题。

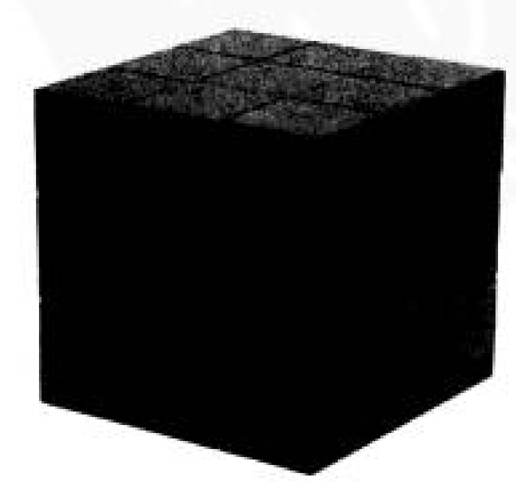
答：空格在这里不是问题，因为 Ajax 自动负责数据的处理，并确认数据以恰当格式抵达服务器。

问：既然图像是博客的选用内容，新增博客日志时，一定要传送它的数据到服务器上吗？

答：不用，不需如此。但请记得，在 URL-encoded 字符串中，传送空数据，在等号后不加任何值，其实没有关系，如下所示：

```
"date=...&body=...&image="
```

上例中，图像数据域仍然被送往服务器，虽然它并未包含任何数据。这就是服务器端的 PHP 脚本闪闪发亮的地方，因为它很聪明地理解了图像域为空白，所以新的博客日志没有图像。



磨笔上阵

请为YouCube的新增博客脚本填入addBlogEntry()与handleRequest()函数中失落的代码。

```
function addBlogEntry() {
  // Disable the Add button and set the status to busy
  ................................................................
  ................................................................

  // Send the new blog entry data as an Ajax request
  ajaxReq.send("POST", "addblogentry.php", handleRequest,
    "application/x-www-form-urlencoded; charset=UTF-8",
    ................................................................
    ................................................................
    ................................................................);
}

function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Enable the Add button and clear the status
    ................................................................
    ................................................................

    // Confirm the addition of the blog entry
    alert("The new blog entry was successfully added.");
  }
}
```

磨笔上阵

请为YouCube的新增博客脚本填入 addBlogEntry()与handleRequest() 函数中失落的代码。

新增按钮在服务器存储博客日志时先行禁用。

网页上的状态区域显示“忙碌”信息，让用户知道正在等待某事。

```
function addBlogEntry() {
  // Disable the Add button and set the status to busy
  document.getElementById("add").disabled = true;
  document.getElementById("status").innerHTML = "Adding...";

  // Send the new blog entry data as an Ajax request
  ajaxReq.send("POST", "addblogentry.php", handleRequest,
    "application/x-www-form-urlencoded; charset=UTF-8",
    "date=" + document.getElementById("date").value +
    "&body=" + document.getElementById("body").value +
    "&image=" + document.getElementById("image").value);
}
```

这是个POST请求。

服务器端的PHP脚本用于处理博客日志，并存储到服务器上的博客文件。

组合POST请求的数据，包括日期、正文、图像等表单域。

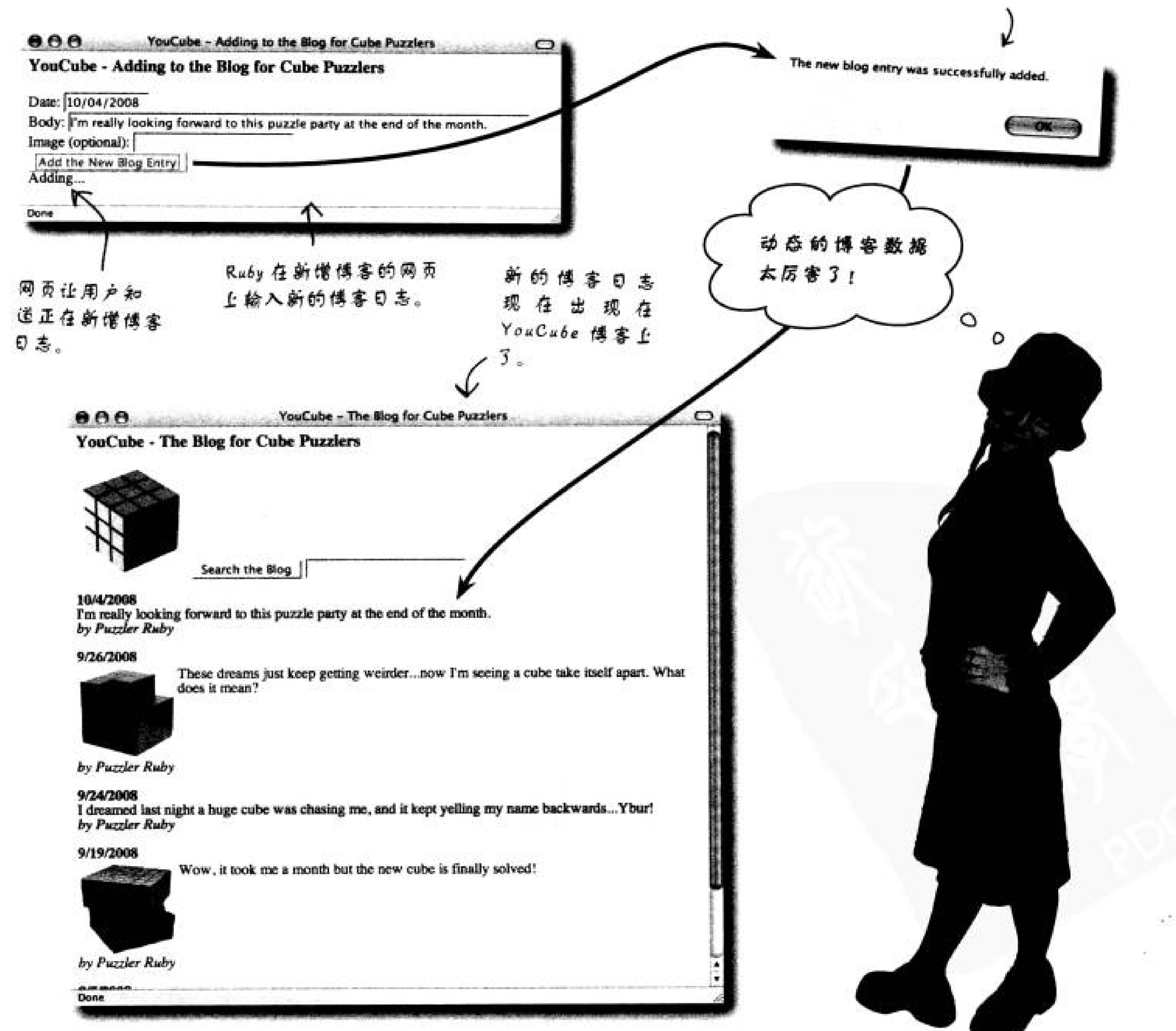
```
function handleRequest() {
  if (ajaxReq.getReadyState() == 4 && ajaxReq.getStatus() == 200) {
    // Enable the Add button and clear the status
    document.getElementById("add").disabled = false;
    document.getElementById("status").innerHTML = "";

    // Confirm the addition of the blog entry
    alert("The new blog entry was successfully added.");
  }
}
```

确认存储博客的请求成功结束。

启用新增按钮并清除状态区域，因为博客日志已经结束存储。

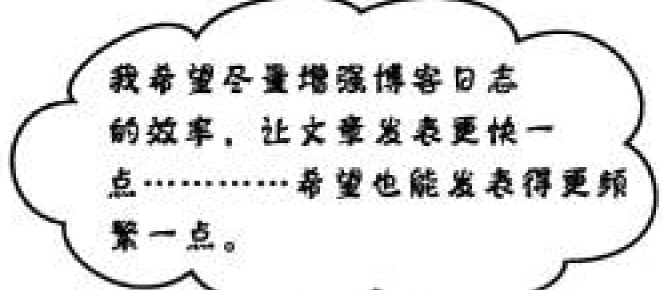
玩博客变简单了

Ruby 真不敢相信眼前的一切！原来不需打开文件、编辑代码、上传文件至服务器，就能让博客的更新改变这么多。她的博客现在不只真的由数据驱动，她也更有动力发表新的博客文件了！



让 YouCube 更……好用

能拿到三度空间魔方的黑带检定，势必相当注意细节。所以Ruby 进一步要求让新增博客网页绝对完美，似乎也不太让人意外了。Ruby 知道，讲到对可用性细节的注重，Ajax 应用程序相当出名。所以她希望改善新网页的可用性，要与当代的新潮网页比美。



YouCube博客日志数据最大化

既然大部分博客日志都在“现在”制作，Ruby想到，在博客表单的日期域（Date）里自动填上今天的日期，可节省不少宝贵的打字时间。而且，既然大多数博客日志会使用现在的日期，她希望直接把输入焦点移到表单里的正文域（Body），如此一来，她即可在网页开启后直接输入。当然，这些改变对于博客的运作算不上极端重要，而且也未直接与Ajax相关，但它们戏剧化地改良了网页给人的“感受”，这差不多就是Ajax的精神。另外，也能确保Ruby更勤快地更新博客。

自动输入目前的日期。

YouCube - Adding to the Blog for Cube Puzzlers

YouCube - Adding to the Blog for Cube Puzzlers

Date: 10/11/2008
Body:
Image (optional):
Add the New Blog Entry

Done

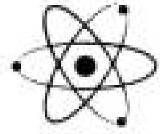
把输入焦点设定在这里，让 Ruby 能立刻开始输入博客正文。

为用户自动填写域

如果你还有印象，博客日期格式为 MM/DD/YYYY，我们需要确保自动填写至域中的日期也采用相同的格式。所以，我们需要把目前的日期格式设为 MM/DD/YYYY 的代码。

共享通用代码一向都是避免重复的好想法。

我们绝对不希望有任何重复的代码在 YouCube 里游荡，以免需要维护两次，所以网页间共享日期格式代码是很好的主意。一定有方式存储代码在某个地方，让任何需要的网页都能取用。



动动脑

如何让两份YouCube网页共享`shortFormat()`的程序代码？

重复的任务？不如来个函数吧？

在多个网页间共享 JavaScript 代码，包括把代码另外拆解成一个文件或模块（module），然后导入文件至每个网页。AjaxRequest 对象已经做过这件事，它存储在 *ajax.js* 文件里，然后以下列程序代码导入：

JavaScript 文件的名称设定给 <script> 标签的 src 属性。

```
<script type="text/javascript" src="ajax.js"> </script>
```

熟悉的<script>标签用于导入存储在外部文件中的JavaScript代码。

Date 对象的 shortFormat() 方法如果放在 *date.js* 文件里，然后被导入 YouCube 的两个网页，它也能达成相同效果。

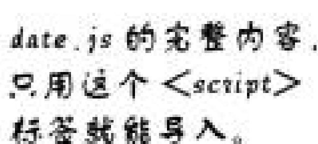
```
Date.prototype.shortFormat = function() {
  return (this.getMonth() + 1) + "/" + this.getDate() + "/" + this.getFullYear();
}
```

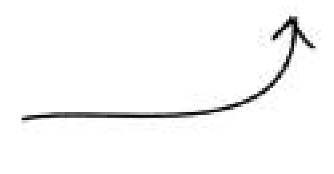
date.js

两个不同地方共享相同一份 JavaScript 代码，这都是把代码存储在外部文件的功劳。

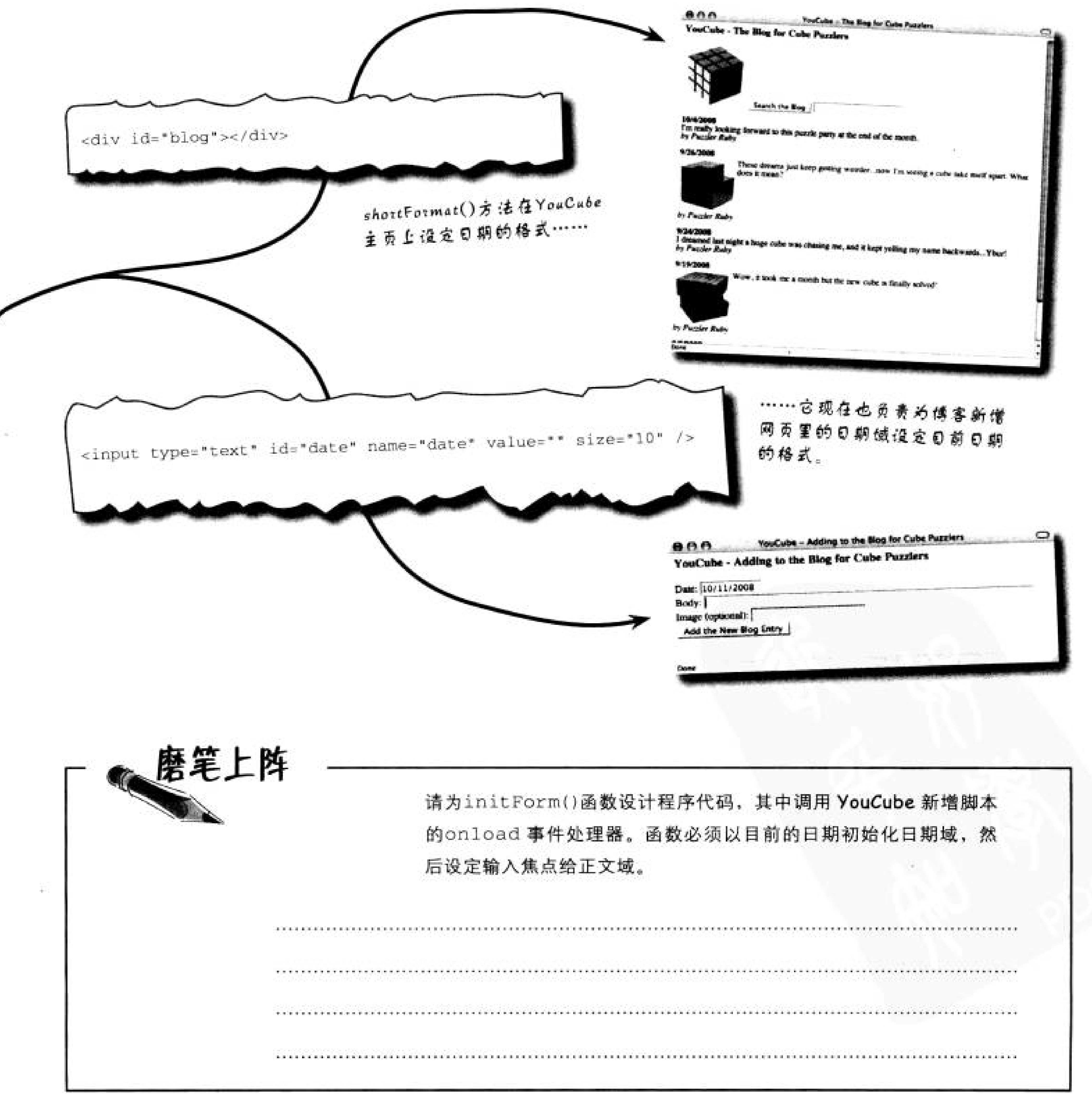
类似于 Ajax 代码使用的 <script> 标签，现在用于 YouCube 的两个网页中，以导入存储在 *date.js* 的脚本。

```
<script type="text/javascript" src="date.js"> </script>
```





把可再利用的JavaScript代码另外分成它自己的文件以供分享，几乎不会是个坏主意。



磨笔上阵

请为initForm()函数设计程序代码，其中调用 YouCube 新增脚本的onload 事件处理器。函数必须以目前的日期初始化日期域，然后设定输入焦点给正文域。

....

....

....

....

磨笔上阵解答

请为initForm()函数设计程序代码，其中调用 YouCube 新增脚本的onload事件处理器。函数必须以目前的日期初始化日期域，然后设定输入焦点给正文域。

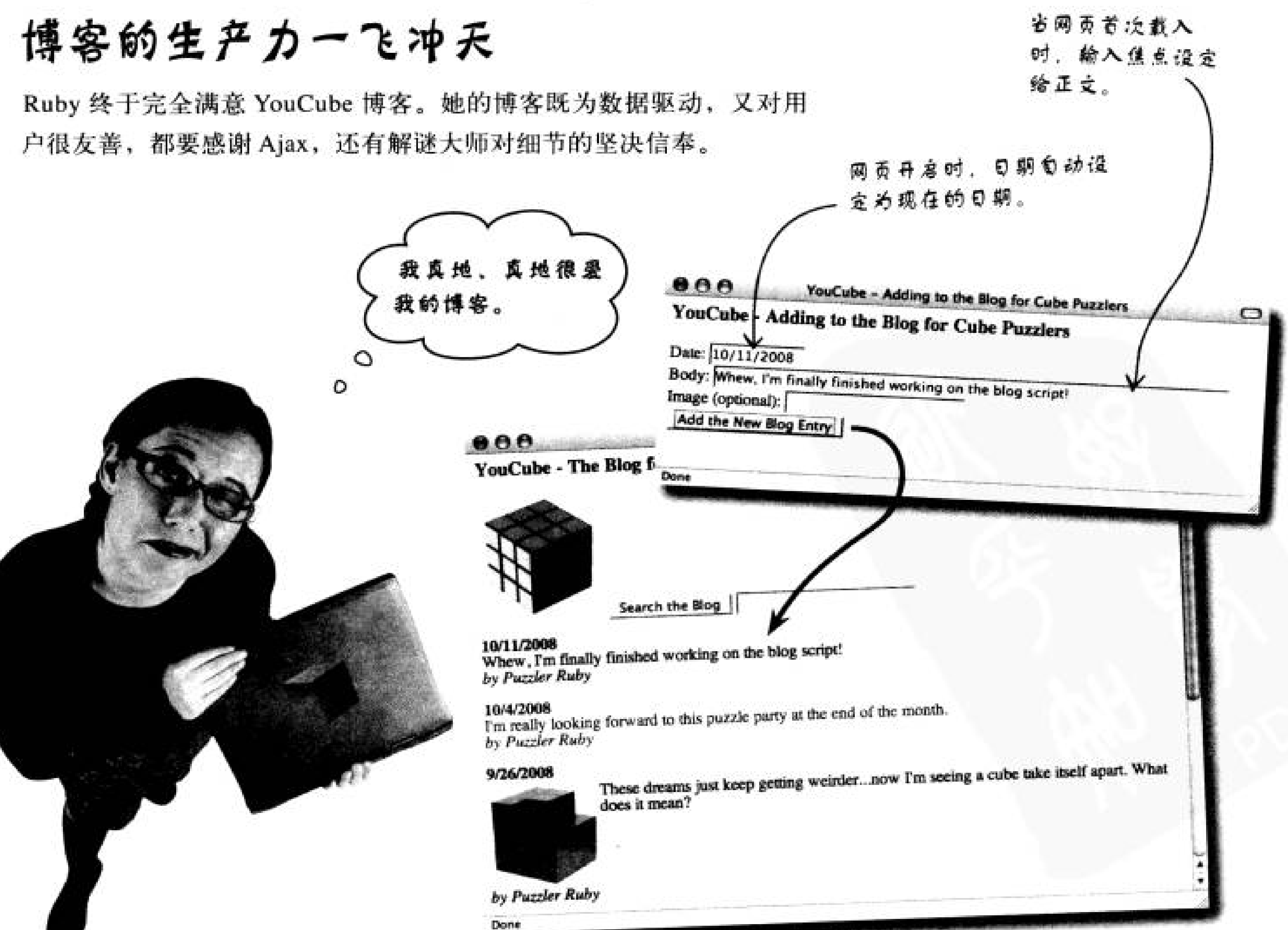
日期域设定为现在的日期。

设定输入焦点于正文域。

```
function initForm() {
  document.getElementById("date").value = (new Date()).shortFormat();
  document.getElementById("body").focus();
}
```

博客的生产力一飞冲天

Ruby 终于完全满意 YouCube 博客。她的博客既为数据驱动，又对用户很友善，都要感谢 Ajax，还有解谜大师对细节的坚决信奉。



JavaScript 填字游戏

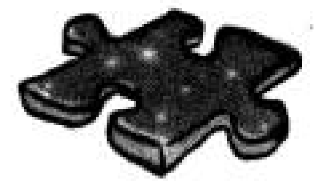
你感觉到动态了吗？来点搭配活泼态度的数据如何？填字游戏，就是它了，上吧！

横向提示：

2. 服务器端的脚本编程技术，与 Ajax 应用程序里的 JavaScript 互补。
3. 服务器对 Ajax 请求的答案。
5. Ajax 请求结束时被调用的函数。
8. 这项技术用于让网页更有反应。
9. 这种数据让网页更有趣。
10. 支持 Ajax 功能的标准 JavaScript 对象。
12. 通常只向服务器要求数据的请求类型。

纵向提示：

1. HTML 中的“ML”代表 ______ Language。
2. 通常还会改变服务器状态的请求类型。
4. AjaxRequest 对象里的一个方法，用于发出请求。
6. 为了简化 Ajax 的使用而自定义的对象。
7. XML 的“X”代表________。
10. 使 <blog>、<author>、<entry> 可被利用的事物。
11. Ajax 向服务器要求数据，称为________。



JavaScript 填字游戏解答

[1] M						[2] P	H	P								
A						O										
[3] R	E	[4] S	P	O	N	S	E		[5] C	[6] A	L	L	B	A	C	K
K		E				T				J						
U		N		[7] A						[8] A	J	A	X			
P		[9] D	Y	N	A	M	I	C		X						
				Y						R						
				T						E						
	[10] X	M	L	H	T	T	P	[11] R	E	Q	U	E	S	T		
	M			I				E		U						
	L			N				Q		E						
				[12] G	E	T		U		S						
								E		T						
								S								
								T								

Page Bender

请垂直对折本页，模拟左右脑的区分，并解决我们提出的谜团。

Ajax为Ruby带来什么？

这是左右脑的秘密会谈！

AjaxRequest

我正在分解，一次一块！

让我们把你拼回去。

Ajax带了很多东西给Ruby，实在很难逐一列举。动态数据是其中一项，让魔方博客的经营更容易。Ruby现在能轻松地讨论她的魔方了。

你的后续旅程

你终于走完这一趟“深入浅出JavaScript”之旅，也准备继续带着 JavaScript踏上创建交互式用户体验的道路……但是，下一站应该是哪一站？在你为万维网大荒野着手建立应用程序时，我们为你提供一些可能有兴趣的景点。

深入浅出 JavaScript

被表达式烦到火冒三丈吗？被运算符撞得眼冒金星吗？或者你想与Head First社区分享最新的JavaScript发明呢？欢迎光临Head First Labs（*http://www.headfirstlabs.com*）的Head First JavaScript讨论区并加入现有的讨论……更欢迎你自己开启新的讨论！

补充书籍

你已经掌握了要点，可以准备深入进阶 JavaScript 的里里外外。

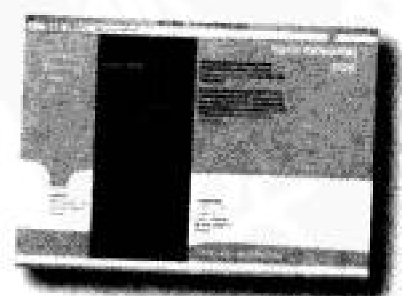
《JavaScript the Definitive Guide》

《JavaScript & DHTML Cookbook》

向其他网站学

Quirksmode *http://www.quirksmode.org*

很可惜，不同浏览器有时自有一套解释JavaScript的方式。Quirksmode提供了JavaScript在浏览器上不一致行为的内幕消息。



Mozilla JavaScript Reference

http://developer.mozilla.org/en/docs/JavaScript

很快，你将暂时放下手边的一团乱麻，试图寻找更多JavaScript内置对象的信息。欢迎利用Mozilla的在线参考文档，探索JavaScript每个不起眼的角落与密室。

Prototype JavaScript Framework

http://prototypejs.org

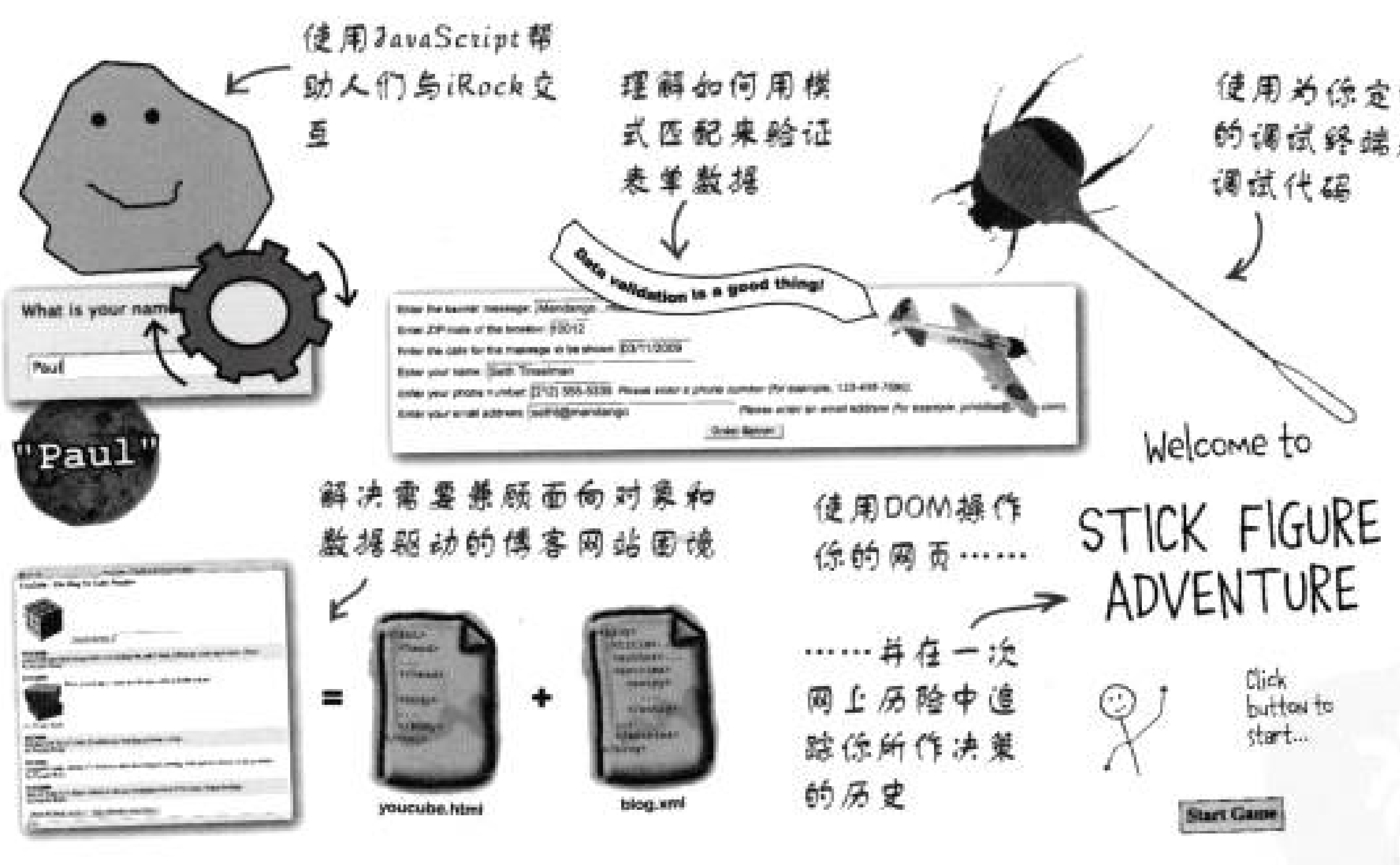
忍不住想尝试第三方程序库的口味，把你的 JavaScript 代码带入全新层次吗？这里是最好的程序库之一，而且完全免费！

深入浅出JavaScript（中文版）

JavaScript/Web Programming

你将从本书学到什么？

这么说你准备从写HTML和CSS的静态网页跃进到编写动态网络应用程序脚本了？这里就是起点。《深入浅出JavaScript》带你游历令人兴奋的交互式网页创建过程。为了启发你的思考，本书覆盖了所有的JavaScript基本知识，从基本网络编程技巧，如变量、函数和循环语句，到高级一些的专题，如表单验证、DOM操作、客户端对象、脚本程序调试——甚至是Ajax！赶快做好准备……快速响应的网站离你只有几页纸那么远。



为何本书看上去如此不同？

我们认为你的时间如此宝贵以至于不应该花费在为新概念伤脑筋上面。《深入浅出JavaScript》用最新的认知科学和学习理论打造多感官的学习体验，它运用丰富的视觉样式激发你的大脑工作，而不是密密麻麻的文字让你看了昏昏欲睡。

"非常实际而且有用，同时也解释得非常清楚。本书在向一个完全新手介绍JavaScript时做得非常出色，并且本书是'Head First（深入浅出）'系列教学风格的又一见证。与其他关于JavaScript的书相比，用《深入浅出JavaScript》来学习是很棒的，想想其他的参考书都有电话簿那么厚。"

——*Alex Lee*，学生
休斯敦大学（*University of Houston*）

"一个初级JavaScript开发人员的完美选择。"

——*Fletcher Moore*，
网络开发人员和设计者，
乔治亚理工学院（*Georgia Institute of Technology*）

"经典'深入浅出'系列里又一本伟大的书籍。"

——*TW Scannell*

责任编辑：张烨
封面设计：Louise Barr, Steve Fehler, 张健

O'Reilly Media, Inc.授权东南大学出版社出版

O'REILLY®
www.oreilly.com
www.headfirstlabs.com

ISBN 978-7-5641-2416-8
9 787564 124168 >
定价：98.00元

此简体中文版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)