Python 核心笔记

By 阿 King

cuijingjing@baidu.com

2011-5-5

目录

前言

【关于 Python】

对于 Python，想必很多朋友还未曾听说，其定义我已在本笔记的第一章给出，这里我想谈谈我对 Python 的个人理解。在我学习 Python 的过程中，有朋友问我，Python 到底是什么，它主要应用于什么方面，能做什么东西，这里就算是给你一个答复吧。

狂妄点，可以说：只有你想不到，没有 Python 做不到。

Python 是一种支持面向对象的解释性高级语言，Simple yet Powerful 是人们对它的一致评价。最初是在苹果计算机上被编译成功的，但现在他已经可以运行于世界上主流的操作平台之上了。跨平台性极强。它包含多种 Programming Paradigm，包括 Object-Oriented Programming，Procedure Programming 等。

Python 类似于 Javascript，可以写出很短小但是功能强大的小程序，而 Python 丰富的标准库（包括系统库，__builtin__，以及完全支持正则表达式的字符串操作）为实现这些实用功能提供了坚实的基础。

Python 类似于 C++和 Java，它是一个完全支持面向对象的语言（支持多继承），但是它的语法更灵活，支持 Dynamic Typing，变量从来不用进行声明便可使用，通常情况下 Python 的代码密度是 C++的 5 到 10 倍，是 Java 的 5 倍左右。

Python 采用严格的缩进方式来管理代码，提高了可读性，极大的降低了代码维护的代价。Python 自身包含 Gargage Collection 和内存管理机制，用户无需费心这些细节问题。

Python 支持 Interactive Programming。 Python 的运行时效率很高，却足以支持复杂的高层数据结构。Python 是一种理想的 Glue Language，也就是说我们可以用它来完成一般脚本编程所能完成的任务。

可以说，Python 是一个渗透了多种计算机语言特性的极佳的设计。

Python 的代码类型，基本可分为 3 种，分别是字节代码，二进制代码，优化代码。他们的后缀名分别是.py .pyc .pyo 这些代码都可以直接运行。无需做任何的编译或者连接。

Python 可以帮你出色地完成工作，而且一段时间以后，你还能看明白自己写的这段代码。你会对自己如此快速地学会和它强大的功能而感到十分的惊讶，更不用提你已经完成的工作了。

【关于《Python 核心编程》（第二版）】

这本书中拥有广泛的选题、丰富的例子和必要的深入解析，确实是一本不可多得的 Python 经典教材。该书分两大部分：第一部分，占据了大约三分之二的篇幅，来向你阐释这门语言的“核心”内容。第二部分则提供了各种高级主题来向你展示你可以使用 Python 的最新版本。

【关于本文档】

这是我学习《Python 核心编程》时整理出来的读书笔记，浓缩了其精华，以方便读者快速掌握要点。当然这本书笔记不能作为一本入门教材，如果你还没有对 Python 接触过而直接阅读这本笔记可能有点困难，但若你拥有足够的 C、C++、Java 语言基础，那么这本笔记绝对是你快速入门及进阶 Python 的强有力的助手。

本笔记中没有收入大量的例题及练习，而对于 Python 光看不练是远远不行的，因此希望读者能另找练习加以巩固。

本人也是刚走入 Python，能力有限，笔记中难免可能有所疏忽，望读者见谅。

【关于作者】

阿 King

1987 年 6 月出生

QQ：278601848

BaiduHi：cumthacker

博客：http://qasir.net

略懂一点 Python，属于入门级选手，期待与所有 python 爱好者一起学习探讨共同进步。

【致谢】

感谢我的老板 gaogregory，是你给了我工作后继续学习 python 的机会。

感谢我的女朋友，因为到现在还没找到你，所以才有时间写这份文档。

第一部分：Python 核心

第一章 欢迎来到 Python 世界

定义：

Python 是一门优雅而又健壮的编程语言，它继承了传统编译语言的强大性和通用性，同时也借鉴了简单脚本和解释语言的易用性。

特点：

- 高级
- 面向对象
- 可升级
- 可扩展
- 可移植性
- 易学
- 易读
- 易维护
- 健壮性
- 高效的快速原型开发工具
- 内存管理器
- 解释性和（字节）编译性

下载和安装 Python

运行 Python：

本笔记使用平台：python 2.5 IDLE(GUI)

第二章 快速入门

输出：print

输入：raw_input

列表元素： [] 类似于数组

列表的切片操作：[from:to] 截取 from 到 to 的列表元素（含 from，不含 to）

字典元素： { } 键值对

代码缩进：四个空格，尽量不用 Tab 键（不同 的 OS，长度定义不同）

打开文件：handle = open(file_name, access_mode = 'r')

对应 handle.close()

函数定义：

```
def function_name([arguments]):
    "optional documentation string"
    function_suite
```

类中的 __init__()方法：

当一个类被创建时， __init__()方法会自动被执行，类似构造器，它 仅仅是类对象创建后执行的第一个方法。目的是完成对象的初始化工作。

self 参数：

每个方法都有这个参数。它是类实例自身的引用。相当于其它面向对象编程语言中的 this。

第三章 Python 基础

特殊规则及特殊字符：

#号（#）：注释

换行（\n）：换行

反斜线（\）：继续上一行

分号（;）：两个语句连在一行

冒号（:）：将代码块的头和体分开

不同的缩进深度代表不同的代码块

Python 文件以模块的形式组织

多元赋值：x,y,z = 1,2,3　一一对应赋值（建议用小括号扩起来）

变量名必须由字母跟下划线组成，且区分大小写。对于一般的变量名建议不要用下划线开头。

__name__ 系统变量：

- 模块被导入：__name__的值为模块的名字
- 模块被直接执行：__name__的值为’__main__’

内存管理:

- 变量无须事先申明

- 变量无须指定类型
- 程序员不用担心内存管理
- 变量名会被“回收”
- del 语句能够直接释放资源

第四章 Python 对象

Python 对象的三个特性：身份，类型，值

内建函数 type()：获取对象的类型

None:Python 的 NULL 对象

切片对象：sequence[起始索引: 结束索引：步进值]

标准类型内建函数：

- cmp(obj1, obj2)　比较 obj1 和 obj2，返回整形 i（>0;=0;<0）
- repr(obj)或`obj`　返回 obj 的字符串表示
- str(obj)　　返回 obj 适合可读性好的字符串表示
- type(obj)　　返回 obj 的类型

str(),repr(),`` 三者的比较：

- str():生成一个对象的可读性比较好的字符串表示，对用户友好
- repr():对 Python 比较友好
- ``：效果跟 repr()一样，建议不再使用

type()和 isinstance()：

- type(obj)：返回 obj 的类型
- isinstance(obj,obj_type)：判断 obj 是否为 obj_type 类型，返回布尔值

可变类型：列表、字典

不可变类型：数字、字符串、元组

不支持的类型：

char 或 byte：无

指针：无

int VS short VS long : 无区别

float VS double ：无区别

第五章 数字

长整型：用大写的“L”标示（虽然也可以用小写的“l”，但不建议使用）

复数：

- 虚数不能单独存在。必须加上 0.0 的实数部分
- 由实数+虚数两部分构成
- 实数跟虚数部分都是浮点型
- 虚数部分后缀是 j 或 J

复数属性：

- num.real　　该复数的实数部分
- num.imag　　该复数的虚数部分
- num.conjugate()　　返回该复数的共轭复数

混合模式优先级(转化)： complex > float > long > int

幂运算(**)：

- 比左侧操作数的一元运算符优先级低，比右侧操作数的一元运算符优先级高。
- eg: -3 ** 2 = -9　　4.0 ** -1.0 = 0.25

位操作符：

只适用于整型

取反（~），安位与（&），或（|），异或（^），左移（<<），右移（>>）

负数当做正数的 2 进制补码处理

num << (或 >>) N = num * (或 /) 2 ** N

int(obj, base) ： 返回 obj 数字的 base 进制数

- pow() 和 **:虽然能完成相同的作用
- pow() ： 内建函数
- ** ： 操作符

数值运算内建函数：

- abs(num) num 的绝对值
- coerce(num1, num2) 转化为同一类型，返回一个元组
- divmod(num1, num2) 返回元组（num1/num2, num1%num2）
- pow(num1, num2, mod=1) 取 num1 的 num2 次方，若有 mod，则对其再取余
- round(flt, ndig=1) 对浮点型 flt 进行四舍五入，保留 ndig 位小数

仅适用于整型的内建函数：

- hex(num) 将 num 转化为十六进制，并以字符串返回
- otc(num) 将 num 转化为八进制，并以字符串返回
- chr(num) 返回 num 的 ASCII 值，范围：0 <= num <= 255

“True”和“False”严格区分大小写！且分别对应“1”和“0”。

第六章 序列：字符串、列表和元组

字符串类型同样也是不可变的，当你要改变一个字符串的时候就必须通过创建一个新的同名的字符串来取代它。

三引号（’’’或者”””）：

允许一个字符串跨多行，字符串中可以包含换行符，制表符及其他特殊字符。

ASCII 码：每个英文字符都是以 7 位二进制数的方式存放在计算机内，范围是 32~126。

Unicode 通过使用一个或多个字节来表示一个字符的方法突破了 ASCII 的限制，可以表示超过 90000 个字符。

ASCII 字符串 – StringType 类型

Unicode 字符串 – UnicodeType 类型

默认所有字面上的字符串都是 ASCII 编码，’u’前缀声明 Unicode 编码。

codec 是 Coder/DECoder 的首写字母组合。

Unicode 应用的规则：

- 程序中出现字符串时前面一定要加个’u’
- 不要用 str()函数，用 unicode()代替
- 不要用过时的 string 模块
- 不到必须时不要在你的程序里编解码 Unicode 字符。只在你写入文件或者数据库或者网络时，才调用 encode()编码，相应地，只在你需要把数据读回来的时候才调用 decode()函数解码

- Python 标准库中绝大部分模块都是兼容 Unicode 的，除了 pickle 模块只支持 ASCII。
- Python 字符串不是通过 NUL 或者'\0'来结束的，它除了你定义的东西，没有别的。

并非调用一个方法就会返回一个值。

那些可以改变对象值的可变对象的方法是没有返回值的。如：sort(),extend()等。

列表跟元组是两个非常相似的序列类型，之所以要保留二者是因为在某些情况下，其中一种类型要优于使用另一种类型。

list()跟 tuple()函数可以用一个列表来创建一个元组，反之亦然。

第七章 映射和集合类型

字典是 Python 中唯一的映射类型。

字典：

```
dict1 = {'name':'earth','port':80}

for key in dict1.keys():
   print 'key=%s, value=%s'% (key,dict1[key])
```

映射类型内建函数：

- dict.clear()　　删除字典中的所有元素
- dict.fromkeys(seq, val=None)　　返回一个新字典，seq 为键，val 为值
- dict.get(key, default=None)　　返回字典中 key 的值，若无此键则返 default
- dict.has_key(key)　　是否存在 key 键，返回布尔值
- dict.items()　　返回一个包括字典中键值对元组的列表
- dict.keys()　　返回一个字典中的键的列表
- dict.values()　　返回一个包含字典中所有值的列表
- dict.update(dict2)　　将字典 dict2 的键值对添加到字典 dict 中去

不允许一个键对应多个值

键必须是可哈希的对象。像列表和字典这样的可变类型，由于它们不是可哈希的，所以不能作为键

集合类型：

- 现已成为 Python 的基本数据类型

- 两种类型：可变集合（set）和不可变集合（frozenset）
- 可用 set()和 frozenset()来分别创建可变集合跟不可变集合

适用于所以集合类型的内建方法：

- s.issubset(t)　　判断 s 是否是 t 的子集，返回布尔值
- s.issuperset(t)　判断 s 是否是 t 的超集，返回布尔值
- s.union(t)　　返回新集合，s 跟 t 的并集
- s.intersetion(t)　返回新集合，s 跟 t 的交集
- s.difference(t)　返回新集合，其成员是 s 的成员，但不是 t 的成员

第八章 条件和循环

if 循环：

if expression1:

    expr1_true_suite

elif expression2:

    expr2_true_suite

elif expression3:

    expr3_true_suite

else:

    None_of_the_above_suite

Python 不支持 switch/case 语句，但完全可以用 if/else 的结构来代替

三元操作符：X if C else Y　（如果条件 C 成立，则结果为 X,否则为 Y）

while 语句：

while expression:

    suite_to_repeat

for 语句：

for iter_var **in** interable:

    suite_to_repeat

pass 语句：Python 中提供 pass 语句，表示不做任何事，NOP(No Operation)

def foo_func(): #空函数
pass

迭代器和 iter()函数：

- 迭代器就是有一个 next()方法的对象
- 对一个对象调用 iter()方法就可以得到它的迭代器

列表解析：

> 列表解析（list comps）来自函数式编程语言 Haskell。它是一个非常有用、简单而且灵活的工具，可以用来动态地创建列表。

Python 早就支持函数式编程特性，例如 **lambda**、**map()**和 **filter()**等

lambda 允许用户快速地创建只有一行的函数对象.例如：**map**(**lambda** x:x ** 2, range(6))

第九章 文件和输入输出

文件内建函数（open()和 file()）：

- open()语法：file_object = **open**(file_name, access_mode='r',buffering=-1)
- open()和 file()函数具有完全相同的功能，一般来说，建议使用 open()来读文件，在你想说明你是处理文件对象的时候使用 file(),例如 if isinstance(f, file)

文件内建方法：

- 输入：read()，readlines(),for **eachLine** in file
- 输出：write(),writelines() [注：没有 writeline()方法]
- 文件内移动：seek() 0,1,2 分别代表文件开头，当前位置，文件末尾

杂项操作：

保留行分隔符：

使用 read()或者 readlines()从文件中读取行时，Python 并不会删除行结束符，类似的，write()或 writelines()也不会自动的加入行结束符，这得由程序员自己完成。

行分隔符和其他文件系统的差异：

不同的操作系统对行结束符的规定是不同的，但 Python 的 os 模块已经替程序员解决了这个问题，只要导入了 os 模块，其属性会自动校准文件系统，设置为正确值。

命令行参数：

- sys.argv 是命令行参数的列表
- len(sys.argv)是命令行参数的个数（也就是 argc）

第十章 错误和异常

Python 中常见的异常：

- NameError： 尝试访问一个未申明的变量
- ZeroDivissionError : 除数为零
- SyntaxError： Python 解释器语法错误
- IndexError : 请求的索引超出序列范围
- KeyError： 请求一个不存在的字典关键字
- IOError： 输入\输出错误
- AttributeError： 尝试访问未知的对象属性
- KeyboardInterrupt： 中断异常
- BaseException： 所有异常的基类

检测和处理异常：try-except 和 try-finally

处理多个异常的时候可以用多个 except，也可以用一个 except，然后将异常放入一个元组里

不推荐使用空 except 语句

else 子句：try-except-else：在 try 范围内没有异常被检测到时，执行 else 子句

finally 子句：try-except(-else)-finally：无论异常是否发生，是否捕捉，都会执行的一段代码。

raise 语句：触发异常。一般语法：

raise [SomeException [,args [,traceback]]]

第一个参数：SomeException：触发异常的名字

第二个参数：args：可选，作为一个对象或者对象的元组传给异常

第三个参数：traceback：可选，很少用

断言：断言是一句必须等价于布尔真的判定，否则将产生 AssertionError（断言错误）的异常，同时也说明是假。

语法：**assert** expression[, arguments]

建议跟 try-except 连用，将断言语句放在 try 中

sys 模块中的 exc_info()函数，通过其提供的一个 3 元组信息同样可以捕捉异常信息：

sys.exc_info()得到的元组：

exc_type:异常类

exc_value:异常类的实例

exc_traceback:跟踪记录对象

不过，在未来的 python 中，这三个对象将被逐步停用，并最终移除。

第十一章 函数和函数式编程

创建函数：

def function_name(arguments):

“function_documentation_string”

function_body_suite

前向引用：Python 也不允许在函数未声明之前，对其进行引用或调用。

Python 支持在外部函数的定义体内创建内嵌函数

函数中使用默认参数会使程序的健壮性上升到极高的级别。

Python 不是也不大可能会成为一种函数式编程语言，但是它支持许多有价值的函数式编程语言构建。

函数式编程的内建函数：

Filter(func, seq)：

调用一个布尔函数 func 来迭代遍历每个 seq 中的元素；返回一个使 func 返回值为 true 的元素的序列

Map(func, seq1[,sql2…])：

将函数 func 作用于给定的序列 seq 中的每个元素，并用一个列表来提供返回值

Reduce(func, seq)：

将一个二元函数作用于 seq 序列的元素，每次携带一对（先前的结果以及下一个序列元素），连续地将现有的结果和下一个值作用，最后减少我们的序列为一个单一的返回值

变量作用域：

全局变量除非被删除，否则它们存活到脚本运行结束，且对所有的函数都是可访问的。

为了明确地引用一个已命名的全局变量，必须使用 global 语句。

第十二章 模块

模块支持从逻辑上组织 Python 代码。

与其他可以导入类（class）的语言不同，Python 带入的是模块或模块属性。

语法：　　**import** module1[,module2[,…]]

from module **import** name1[,name2[,…]]

from module **import** name **as** shortname

from module **import ***

import 语句的模块顺序：

- Python 标准库模块
- Python 第三方模块
- 应用程序自定义模块

导入（import）和加载（load）：

一个模块之恩那个被加载一次，无论它被导入多少次。

模块内建函数：

__import__()函数：__import__(module_name[,globals[,locals[,fromlist]]])

globals()和 locals()分别返回调用者全局和局部名称空间的字典。

reload()重新导入一个已经导入的模块：　　reload(module) 使用 reload()时候必须是全部导入，而不是使用 from-import。

禁止模块的某个属性导入，可以在该属性名称前加一个下划线：**import** foo._bar

第十三章 面向对象编程

在 Python 中，面向对象编程主要有两个主题，就是类和类实例。

利用 class 关键字创建一个类：

```
class MyNewObjectType(bases):                    #bases 参数用于继承的父类
        'define MyNewObjectType class'
        Class_suite
```

object 是“所有类之母”，若未指明父类，则 object 将作为默认的父类。

创建一个实例的语法：（注意：没有使用 **new**，Python 根本就没有 new 这个关键字）

```
myFirstObject = MyNewObjectType()
```

添加类方法：

```
class MyDataWithMethod(object):                  #定义类
        def printFoo(self):                      #定义方法
                print 'You invoked printFoo()'
```

你可能注意到 self 参数，它在所有的方法声明中都存在。并且必须是第一个参数。这个参数代表实例对象本身，当你调用方法的时候由解释器悄悄地传递给方法，而不需要你自己传递 self。

__init__()方法：类似于一个构造器，但不能说是一个构造器（因为 Python 没有使用 new），它在创建一个新的对象时被调用，完成一个初始化工作。

类、属性和方法的命名方式：

类：通常大写字母打头。这是标准惯例。有助于识别类。

属性：小写字母打头 + 驼峰，使用名词作为名字。

方法：小写字母打头 + 驼峰，使用谓词作为名字。

在 Python 中，声明与定义类是没有什么区别的，因为它们是同时进行的，定义（类体）紧跟在声明（含 class 关键字的头行）和可选（但总是推荐）的文档字符串后面。

Python 不支持纯虚函数（如 C++）或抽象方法（如 Java）。

Python 严格要求，没有实例，方法是不能被调用的。方法必须绑定（到一个实例）才能被直接调用。

dir(class_name):　　返回类对象的属性的一个名字列表

class_name**.__dict__**　:　　返回的是一个字典，key 是属性名，value 是数据值

__init__()方法不应该返回任何对象，即，它应该返回 None，否则将产生 TypeError。

实例属性 VS 类属性：

类属性跟实例无关。这些值像静态成员那样被引用，即使在多次实例化中调用类，类属性的值都不会改变。

子类可以通过继承覆盖父类的方法。类似，如果在子类中覆盖了了__init__()方法，那么基类的__init__()就不会被自动调用了。

同 C++一样，Python 支持多继承。但要处理好两个方面：

- 要找到合适的属性
- 重写方法时，如何调用对应父类以“发挥他们的作用”，同时，在子类中处理好自己的义务

vars()内建函数与 dir()相似，知识给定的对象都必须有一个**__dict__**属性。**vars()**返回一个字典，包含了对象存储于其**__dict__**中的属性（键）和值。

第十四章 执行环境

Python 有三种不同类型的函数对象。分别是：

- 内建函数（BIF）：C/C++写的，编译过后放到 Python 解释器当中的。
- 用户定义的函数（UDF）
- lambda 表达式

compile():

- 它允许程序员在运行时迅速生成代码对象，然后就可以用 exec 语句或者内建函数 eval()来执行这些代码或者对它们进行求值。
- compile(string, file, type) 这三个参数都是必须的。
- 第一个参数：string：要编译的 Python 代码
- 第二个参数：file：虽然是必须的，但通常被置为空串
- 第三个参数：type：字符串，用来表明代码的类型。有三个可能值：

‘eval’：可求值的表达式（和 eval()连用）

‘single’：单一可执行语句（和 exec 连用）

‘exec’：可执行语句组（和 exec 连用）

Eg1:

```
>>>eval_code = compile('100+200', '', 'eval')
>>>eval(eval_code)
300
```

Eg2:

```
>>>single_code = compile('print "hello"', '', 'single')
>>>exec single_code
Hello
```

用 compile()预编译重复代码有助于改善性能，因为在调用时不必经过字节编译处理。

exec obj ：

接受对象（obj）可以是原始的字符串，也可以是有效的 Python 文件。

一旦执行完毕，继续对 exec 的调用就会失败，因为 obj 已经到了 EOF 了，若想继续调用 exec，则必须对 obj 调用 seek（0）到文件开头。

tell()方法：告知当前在文件的何处

os.path.getsize()：告知对象文件有多大

结束执行：

- **sys.exit()** and **SystemExit**
- **sys.exitfunc()**
- **os._exit(**status**)**：跟以上两种不同，它不执行任何清理就直接退出 Python 解释器。且 status 参数是必须的。

第 2 部分 高级主题

第十五章 正则表达式

Python 通过标准库的 re 模块来支持正则表达式（RE）。

搜索 and **匹配**：“模式匹配（patten-mactching）”

- **搜索**（searching）：在字符串任意部分中搜索匹配的模式
- **匹配**（matching）：判断一个字符串能否从起始处全部或部分的匹配某个模式

正则表达式中常见的符号和字符

记号	说明	举例
re1|re2	匹配 re1 或者 re2	foo|bar
.	匹配任意字符串（换行符除外）	b.b
^	匹配字符串的开始	^Dear
$	匹配字符串的结尾	/bin/*sh$
*	匹配前面出现的正则表达式零次或多次	[A-Za-z0-9]*
+	匹配前面出现的正则表达式一次或多次	[a-z]+\.com
?	匹配前面出现的正则表达式零次或一次	goo?
{N}	匹配前面出现的正则表达式 N 次	[0-9]{3}
{M,N}	匹配重复出现 M 到 N 次的正则表达式	[0-9]{5,9}
[…]	匹配字符组中出现的任意字符	[aeiou]
[x-y]	匹配从字符 x 到 y 的任意一个字符	[0-9],[a-z]
[^…]	不匹配此字符集中的任意字符	[^aeiou]

\d	匹配任何数字，和[0-9]一样，\D 是\d 的反义	data\d.txt
\w	匹配任何数字字母字符，和[A-Za-z0-9]一样	[A-Za-z]\w+
\s	匹配任何空白字符，和[\n\t\r\v\f]一样	Of\sthe
\b	匹配单词边界	\bThe\b
\A(\Z)	匹配字符串的起始（结束）	\ADear

反斜杠（\）表示对特殊字符进行转译

re 模块：核心函数和方法

函数/方法	描述
模块的函数	
compile(pattern)	对正则表达式模式 pattern 进行编译
re 模块的函数和 regex 对象的方法	
match(pattern,string)	用 pattern 去匹配 string ， 成功返回对象，否则返回 None
search(pattern,string)	在 string 中去搜索 pattern 第一次出现，结果同上
findall(pattern,string)	在 string 中搜素 pattern 非重复出现，返回列表
finditer(pattern,string)	功能同 findall，但返回的是一个迭代器
split(pattern,string)	用 pattern 中的分隔符把 string 割成一个列表
sub(pattern,repl,string)	把 string 中所有 pattern 的地方替换为 repl
匹配对象的方法	
group(num=0)	返回全部匹配对象（或指定编号是 num 的子组）
groups()	返回一个包含全部匹配的子组的元组

r’sring’：用于忽略 string 中的所有特殊字符

正则表达式本身默认是贪心的，解决办法就是用“非贪婪”操作符“?”。这个操作符可以用在“*”、“+”或者“？”的后面，它的作用是要求正则表达式引擎匹配的字符越少越好。

第十六章 网络编程

客户端/服务器架构

客户端/服务器网络编程：

在完成服务之前，服务器必需要先完成一些设置。先要创建一个通讯端点，让服务器能够“监听”请求。当服务器准备好之后，要通知客户端，否则客户端不会提出请求。客户端比较简单，只要创建一个通信端点，建立到服务器的连接，然后提出请求。一旦请求处理完成，客户端收到了结果，通信就完成了。

套接字：通信端点

- 套接字是一种具有“通信端点”概念的计算机网络数据结构。
- 网络化的应用程序在开始任何通讯前都必需创建套接字。没有它将没办法通信。
- 套接字有两种，分别是**基于文件型**的和**基于网络型**的。
- AF_UNIX：“地址家族：UNIX”，基于文件
- AF_INET：“地址家族：Internet”，基于网络
- Python 只支持 AF_UNIX，AF_INET 和 AF_NETLINK 家族

面向连接/面向无连接：

- 面向连接：“虚电路”或者“流套接字”，TCP，套接字类型为 SOCK_STREAM
- 面向无连接：UDP，套接字类型为 SOCK_DGRAM （datagram 数据报）

创建套接字语法：

from **socket** import *

TCP 套接字：tcpSock = **socket**(AF_INET, SOCK_STREAM)

UDP 套接字：udpSock = **socket**(AF_INET, SOCK_DGRAM)

套接字对象的常用函数

函数	描述
服务器端套接字函数	
s.bind(addr)	绑定地址 addr（主机号，端口号）到套接字
s.listen(num)	开始 TCP 监听，最多允许 num 个连接进来
s.accept()	被动接受 TCP 客户端连接，（阻塞式）等待连接的到来
客户端套接字函数	
s.connect()	主动初始化 TCP 服务器连接
s.connect_ex()	Connect()函数的扩展版本，出错时返回出错代码，而不是异常
公共用途的套接字函数	
s.recv()	接收 TCP 数据
s.send()	发送 TCP 数据
s.sendall()	完整发送 TCP 数据
s.recvfrom()	接收 UDP 数据
s.sendto()	发送 UDP 数据
s.close()	关闭套接字

第十七章 网络客户端编程

把因特网比作是一个数据交换中心，数据交换的参与者是一个服务提供者和一个服务的使用者，有人把它比作“生产者 – 消费者”，一般是一个生产者对多个消费者。

文件传输网际协议：

- FTP：文件传输协议（File Transfer Protocol）
- UUCP：Unix-to-Unix 复制协议（Unix-to-Unix Copy Protocol）
- HTTP：超文本传输协议（Hypertext Transfer Protocol）
- rcp/scp/rsync：Unix 下的远程文件复制指令

一般编程步骤：

- 连接到服务器
- 登陆（如果需要的话）
- 发出服务请求（有可能有返回信息）
- 退出

Python 和 FTP：**from** ftplib **import** FTP

Python 和 NNTP：**from** nntplib **import** NNTP

Python 和 SMTP：**from** smtplib **import** SMTP

对于 Python 中的 FTP、NNTP、SMTP 客户端编程的“API”，此核心笔记不予累赘，请请阅相关文档。

第十八章 多线程编程

Python 代码的执行由 Python 虚拟机（也叫解释器主循环）来控制。

虽然 Python 解释器中可以“运行”多个程序，但在任意时刻，只有一个线程在解释器中运行。与单个 CPU 的多线程原理是一样的。

对 Python 虚拟机的访问由全局解释器锁（GIL）来控制的，也正是这个锁保证了同一时刻只有一个线程在运行。

不建议使用 thread 模块，而推荐使用 threading 模块，原因：

- Thread 模块不支持守护进程。在主线程退出的时候，所有他其他线程没有被清除就退出了
- Threading 模块支持守护进程。能保证所有“重要的”子线程都退出后，进程才会结束
- Threading 模块更为先进，对线程的支持更为完善
- Thread 模块中的属性可能与 threading 出现冲突
- 低级别的 Thread 模块的同步原语只有一个，而 Threading 则有很多

time.**sleep**(secs):睡眠多长时间，secs 单位秒（不是毫秒）

start_new_thread(function, args keargs=None) 产生一个新的线程，在新线程中用指定的参数和可选的 kwargs 来调用这个函数

Threading 的 Thread 类是我们主要的运行对象。它有很多 Thread 模块中没有的函数。

函数	描述
start()	开始线程的执行
run()	定义线程的功能的函数（一般会被子类重写）

join(timeout=None)	程序挂起直到线程结束；给了 timeout，则最多阻塞 timeout 秒
getName()	返回线程的名字
setName(name)	设置线程的名字
isAlive()	布尔标志，标示这个线程是否还在运行中
isDaemon()	返回线程的 daemon 标志
setDaemon(daemonic)	把线程的 daemon 标志设为 daemonic（一定要在 start()前调用）

注：thread.start()前调用 thread.setDaemon(True)表示这个线程“不重要”

第十九章 图形用户界面编程

GUI：graphical user interface

Python 的默认 GUI 工具集是 TK，我们可以通过 Python 接口 Tkinter 来使用 Tk。

import Tkinter 先测试一下系统有没有开启 Tkinter。

创建 GUI 程序的五个基本步骤：

1. Import Tkinter
2. 创建顶层窗口对象容纳你的 GUI： top = Tkinter.Tk()
3. 在 top 中创建所有的 GUI 模块
4. 将 3 中的 GUI 模块与底层代码相连接
5. 进入主事件循环

A Tkinter example: “Hello world!”：

```
from Tkinter import *                    #导入 Tkinter
top = Tk()                 #顶层窗口
label = Label(top, text='Hello world!') #创建模块
label.pack()               #装载连接
mainloop()                 #主事件循环
```

运行截图：

Python 拥有大量的图形工具集，其中 4 种比较流行的工具集：

- **Tix**（Tk Interface eXtensions）
- **Pmw**（Python MegaWidgets 的 Tkinter 扩展）
- **wxPython**（wxWidgets 的 Python 绑定）
- **PyGTK**（GTK+的 Python 绑定）

其中 Tix 模块包含在 Python 标准库中，其它工具集是第三方的，必须自己下载。

Learn more about Python GUI:

http://wiki.python.org/moin/GuiProgramming

第二十章 Web 编程

urlparse 模块：

Urlparse 功能	描述
Urlparse(urlstr,defProtSch=None,allowFrag=None	将 urlstr 解析成各个部件 defProtSch 为 URL 协议，allowFrag 为决定是否允许 URL 零部件
Urlunparse(urltup)	将 URL 数据的一个元组反解析成一个 URL 字符串
Urljoin(baseurl,newurl,allowFrag=None)	将 URL 基部件 baseurl 和 newurl 拼合成一个完整的 URL

Urllib 模块：

Urllib 模块提供了所有你需要的功能，它提供了一个高级的 Web 交流库。其特殊功能在于利用各种协议（HTTP、FTP 等）从网络上下载数据。

urllib.open()

语法：urlopen(urlstr, postQueryData=None)

作用：打开 urlstr 所指向的 URL

结果：成功则返回一个文件类型对象

urllib.urlretrieve()

语法：urlretrieve(urlstr, localfile=None, downloadSta-tusHook=None)

作用：将 urlstr 定位到的整个 HTML 文件下载到本地硬盘

结果：返回一个二元组（filename,mine_hdrs）本地文件名、MIME 文件头

（以上是两个 urllib 模块核心函数中特别常用的两个函数，故为大家列了出来，望学习时多加关注，另外对于更为复杂的 URL 打开问题可用 urllib2 模块进行处理）

Python 的基本 Web 客户端一般用于在 Web 上查询或下载文件。而其**高级 Web 客户端**可

以在 internet 上完成基于不同目的的所有和下载页面，包括：

- 为 Google 和 Yahoo 的搜索引擎建索引
- 脱机浏览—将文档下载到本地，重新设定超链接，为本地浏览器创建镜像
- 下载并保持历史记录或框架
- Web 页的缓存，节省再次访问 Web 站点的下载时间

高级 Web 客户端的一个例子就是“网络爬虫”（也称蜘蛛或机器人），与正则表达式的完美结合使用，使你在 internet 上想“爬”什么就“爬”什么。

CGI：帮助 Web 服务器处理客户端数据

Web 服务器接收到表单反馈，与外部应用程序交互，收到并返回新的生成的 HTML 页面都发生在 Web 服务器的 CGI（标准网关接口，Common Gateway Interface）接口上。

CGI 其实只是一个适用于小型 Web 网站开发的工具。

FieldStorge 类：Python CGI 脚本开始时被实例化，包含一个类似字典的对象，键—表单栏目的名字，值—栏目相应的数据。

建立 Python 自带的 Web 服务器：

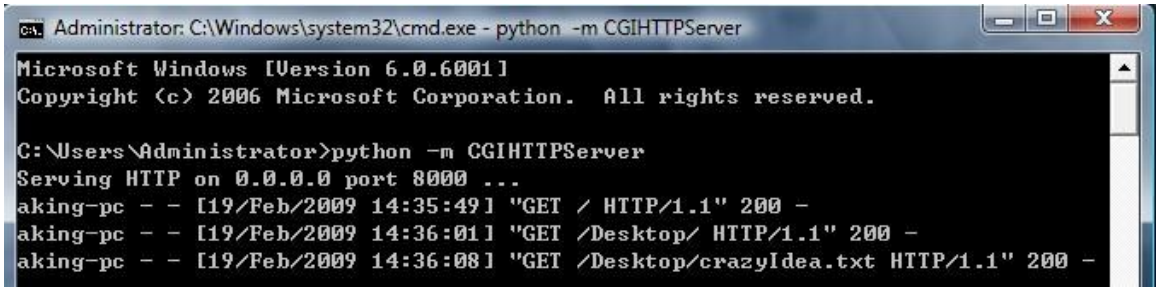
命令：$ Python –m CGIHTTPServer

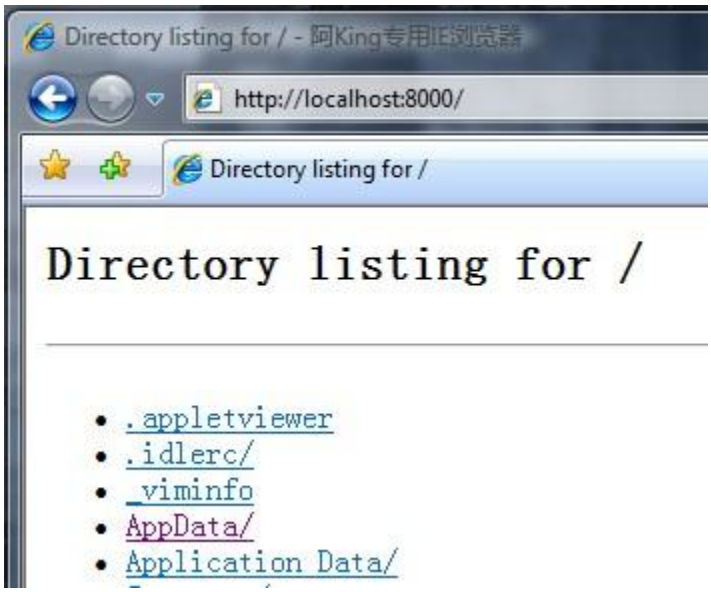
端口：8000

目录：在目录下手工建立 Cgi-bin 存放.py CGI 脚本

访问：http://localhost:8000/ex.html

http://localhost:8000/cgi-bin/ex.py





用 Python 建立 Web 服务器：

要建立一个 Web 服务，一个基本的服务器和一个“处理器”是必备的。

基本的服务器：在客户端和服务器端完成必要的 HTTP 交互。

处理器：处理 Web 服务的简单软件。处理客户端请求，并返回适当的文件。

Web 服务器模块和类

模块	描述
BaseHTTPServer	提供基本的 Web 服务和处理器类，分别是 HTTPServer 和 BaseHTTPRequestHandler
SimpleHTTPServer	包含执行 GET 和 HEAD 请求的 SimpleHTTPRequestHandle 类

CGIHTTPServer	包含处理 POST 请求和执行 CGIHTTPRequestHandle 类

第二十一章 数据库编程

本章的主题是如何通过 Python 访问关系型数据库（RDBMS）

阅读本章读者须掌握基本的数据库操作和 SQL 语言，这部分相关内容请查阅相关的数据库方面的书籍

Python 数据库 API：Python 能够直接通过数据库接口，也可以通过 ORM(需要自己书写 SQL)来访问关系数据库

Python 应用程序（嵌入 SQL）-- Python DB 接口程序 — RDBMS 客户端库 — 关系数据库

DB-API：这是一个规范。它定义了一系列必需的对象和数据库存取方式，以便在各种各样的底层数据库系统和多种多样的数据库接口程序提供一致的访问接口。

DB-API 模块属性

属性名	描述
apilevel	模块兼容的 DB-API 版本号
threadsafety	线程安全级别
paramstyle	该模块支持的 SQL 语句参考风格
connect()	连接函数

连接对象：　　要与数据库通信，必须先和数据库建立连接。连接对象用于处理将命令送往服务器，以及从服务器接受数据等基本功能。

游标对象：　　允许用于执行数据库命令和得到查询结果。

对于不支持游标的数据库来说，connect 对象的 cursor()方法仍然会返回一个尽量模仿游标对象的对象。

游标对象最重要的属性是 execute*()和 fetch*()，所有对数据库服务器的请求均由它们来完成。

Python 到底支持哪些平台下的数据库？答案是几乎所有！

以 MySQL 举例：

```
>>> import MySQLdb
>>> cxn = MySQLdb.connect(user=' rooot' )
>>> cxn.query( 'CREATE DATABASE test' )
>>> cxn.commit()
>>> cxn.close()
-----------------
>>> cxn = MySQLdb.connect(db=' test' )
>>> cur = cxn.cursor()
>>> cur.execute( 'CREATE TABLE users(login VARCHAR(8), uid INT)' )

0L

>>> cur.execute( 'INSERT INTO users VALUES( 'john' , 7000)' )

1L

>>> for data in cur.fetcharall():
      print  '%s\t%s'  % data
john   7000
>>> cur.close()
>>> cxn.commit()
>>> cxn.close()
```

ORM：对象-关系管理器

考虑对象，而不是 SQL

Python 和 ORM：最著名的 Python ORM 模块是 SQLAlchemy 和 SQLObject

第二十二章 扩展 Python

一般来说，所有能被整合或者导入到其他 Python 脚本的代码，都可以称为扩展。

扩展的理由：

- 添加额外的（非 Python）功能
- 性能瓶颈的效率提升
- 保持专有源代码私密

为 Python 创建扩展需要 3 个主要的步骤：

1. 创建应用程序代码：
    - 我们要建立的是一个“库”，一个将要在 Python 内运行的模块
    - 在 C 代码中放一个 main()用于测试代码的正确性
2. 利用样板来包装代码
    1) 包含 Python 头文件 #include “Python.h”
    2) 为每个模块的每一个函数增加一个形如 PyObject* Module_func()的包装函数
    3) 为每个模块增加一个形如 PyMethodDef ModuleMethods[]的数组
    4) 增加模块初始化函数 void initModule()
3. 编译与测试：distutils 包被用来编译、安装和分发这些模块、扩展和包
    1) 创建 setup.py
    2) 通过运行 setup.py 来编译和连接你的代码
    3) 从 Python 中导入你的模块 ： $ python setup.py install
    4) 测试功能

在编译的时候，我们需要将代码跟 Python 库放在一起进行编译。

Other topics for more learning:

SWIG

Pyrex

Psyco

嵌入

第二十三章 其他话题

Web 服务

用 Win32 的 COM 来操作 Microsoft Office

用 Jpython 写 Python 和 Java 的程序