目录

[Note: 12](#_Toc420738168)

[30个有关Python的小技巧 13](#_Toc420738169)

[1.1 拆箱a, b, c = 1, 2, 3 14](#_Toc420738170)

[帮助使用 27](#_Toc420738171)

[注释 27](#_Toc420738172)

[Sort 27](#_Toc420738173)

[Sorted() 30](#_Toc420738174)

[类型 30](#_Toc420738175)

[变量作用域 30](#_Toc420738176)

[位移 31](#_Toc420738177)

[右移 31](#_Toc420738178)

[左移 32](#_Toc420738179)

[计算公式 33](#_Toc420738180)

[字符串处理 33](#_Toc420738181)

[简介 33](#_Toc420738182)

[Len计算字符串大小 33](#_Toc420738183)

[转义符让文本更好处理 33](#_Toc420738184)

[字符串前面跟着的小尾巴到底是什么东西 34](#_Toc420738185)

[Count()计数 34](#_Toc420738186)

[字符串替换拼接 34](#_Toc420738187)

[字符串空格操作 36](#_Toc420738188)

[对字符串中大小写转换 37](#_Toc420738189)

[判断字符串大小写函数 38](#_Toc420738190)

[字符串大小写互换 38](#_Toc420738191)

[判断变量类型,长度 38](#_Toc420738192)

[isdigit()判断字符串是否为纯数字 39](#_Toc420738193)

[string的值修改 39](#_Toc420738194)

[split() 分隔器 40](#_Toc420738195)

[替换1 replace 41](#_Toc420738196)

[替换 2 翻译表 41](#_Toc420738197)

[find()方法 字符串寻找 42](#_Toc420738198)

[Index()方法 也是查找 44](#_Toc420738199)

[sys.stdout输出重定向 45](#_Toc420738200)

[raw字符串与多行字符串 46](#_Toc420738201)

[对字符串切片 47](#_Toc420738202)

[布尔类型 47](#_Toc420738203)

[短路计算 惰性求值 48](#_Toc420738204)

[文本 48](#_Toc420738205)

[List 索引 52](#_Toc420738206)

[内部都是指针指向空间 52](#_Toc420738207)

[添加 53](#_Toc420738208)

[Extend: 53](#_Toc420738209)

[append 55](#_Toc420738210)

[Insert 55](#_Toc420738211)

[删除 55](#_Toc420738212)

[Del直接删除 55](#_Toc420738213)

[pop( ) 55](#_Toc420738214)

[Remove 56](#_Toc420738215)

[替换 56](#_Toc420738216)

[是否在列表中 in not in 57](#_Toc420738217)

[排序sort（）和翻转reverse（） 57](#_Toc420738218)

[xrange的用法： 58](#_Toc420738219)

[切片：对list进行切片 59](#_Toc420738220)

[倒序切片 61](#_Toc420738221)

[list转变成dict 61](#_Toc420738222)

[List 变为 str 62](#_Toc420738223)

[排序使用技巧 62](#_Toc420738224)

[本身不变赋值的：sorted 62](#_Toc420738225)

[本身改变的函数sort（数据对象内部使用） 63](#_Toc420738226)

[Key 63](#_Toc420738227)

[多级排序 64](#_Toc420738228)

[Tuple 65](#_Toc420738229)

[索引 65](#_Toc420738230)

[“可变”的tuple 65](#_Toc420738231)

[字典 66](#_Toc420738232)

[字典dict 72](#_Toc420738233)

[dict的特点 72](#_Toc420738234)

[创建字典。{},dict() 73](#_Toc420738235)

[添加内容 a['xx'] = 'xx' 73](#_Toc420738236)

[修改内容 a['xx'] = 'xx' , 74](#_Toc420738237)

[删除 del,clear,pop 74](#_Toc420738238)

[访问dict 75](#_Toc420738239)

[遍历dict 76](#_Toc420738240)

[方法： 77](#_Toc420738241)

[Len() 77](#_Toc420738242)

[keys(): 返回的是列表，里面包含了字典的所有键 78](#_Toc420738243)

[values():返回的是列表，里面包含了字典的所有值 78](#_Toc420738244)

[items：生成一个字典的容器：[()]，每一个元素都是元组 78](#_Toc420738245)

[get：从字典中获得一个值 78](#_Toc420738246)

[info.get('age2','22') 79](#_Toc420738247)

[字典排序 79](#_Toc420738248)

[集合set 81](#_Toc420738249)

[什么是set 81](#_Toc420738250)

[访问set 81](#_Toc420738251)

[set的特点 82](#_Toc420738252)

[遍历set 82](#_Toc420738253)

[更新set 83](#_Toc420738254)

[add()方法添加 83](#_Toc420738255)

[update方法添加 83](#_Toc420738256)

[remove()方法删除 84](#_Toc420738257)

[成员关系in not in 84](#_Toc420738258)

[& | -交集，并集，差集 & | - 85](#_Toc420738259)

[set去重 85](#_Toc420738260)

[Frozenset（）不可变集合，类似tuple 86](#_Toc420738261)

[什么是迭代 86](#_Toc420738262)

[索引迭代 87](#_Toc420738263)

[dict迭代value 88](#_Toc420738264)

[dict迭代key和value 89](#_Toc420738265)

[生成列表 89](#_Toc420738266)

[100以内的奇数： 90](#_Toc420738267)

[1-100 数的平方 90](#_Toc420738268)

[字符串生成 90](#_Toc420738269)

[元组生成 91](#_Toc420738270)

[字典生成 91](#_Toc420738271)

[复杂表达式 91](#_Toc420738272)

[条件过滤 92](#_Toc420738273)

[多层表达式 92](#_Toc420738274)

[语句 93](#_Toc420738275)

[if语句 93](#_Toc420738276)

[if ... else ... 93](#_Toc420738277)

[if-elif-else 94](#_Toc420738278)

[For 94](#_Toc420738279)

[while循环 95](#_Toc420738280)

[break退出循环 95](#_Toc420738281)

[continue继续循环 96](#_Toc420738282)

[多重循环 96](#_Toc420738283)

[输入 97](#_Toc420738284)

[文件 97](#_Toc420738285)

[linecache模块缓存读取大文件指定行 97](#_Toc420738286)

[函数 99](#_Toc420738287)

[参数 99](#_Toc420738288)

[\*\*kr 是字典； 99](#_Toc420738289)

[\*kr是元组 100](#_Toc420738290)

[混合方式 100](#_Toc420738291)

[定义默认参数 def power(x, n=2) 101](#_Toc420738292)

[定义可变参数 \*args 102](#_Toc420738293)

[初等函数 103](#_Toc420738294)

[一、数学运算类 103](#_Toc420738295)

[二、集合类操作 104](#_Toc420738296)

[三、逻辑判断 106](#_Toc420738297)

[四、反射 106](#_Toc420738298)

[五、IO操作 111](#_Toc420738299)

[六、其他 112](#_Toc420738300)

[编写函数 112](#_Toc420738301)

[返回多值 112](#_Toc420738302)

[递归函数 113](#_Toc420738303)

[高阶函数 114](#_Toc420738304)

[把函数作为参数 114](#_Toc420738305)

[map()函数-分项操作 115](#_Toc420738306)

[reduce()函数-求和操作 115](#_Toc420738307)

[filter()函数-过滤 116](#_Toc420738308)

[自定义排序函数 117](#_Toc420738309)

[返回函数 118](#_Toc420738310)

[闭包 119](#_Toc420738311)

[匿名函数 120](#_Toc420738312)

[匿名函数-lambda 121](#_Toc420738313)

[装饰器（@log） 122](#_Toc420738314)

[编写无参数decorator 122](#_Toc420738315)

[编写带参数decorator-(l廖雪峰) 124](#_Toc420738316)

[完善decorator 127](#_Toc420738317)

[偏函数 128](#_Toc420738318)

[模块 130](#_Toc420738319)

[导入模块 130](#_Toc420738320)

[别名 131](#_Toc420738321)

[作用域 131](#_Toc420738322)

[动态导入模块 132](#_Toc420738323)

[使用\_\_future\_\_ 133](#_Toc420738324)

[安装第三方模块-（慕课有视频） 134](#_Toc420738325)

[面向对象编程 135](#_Toc420738326)

[定义类并创建实例 135](#_Toc420738327)

[创建实例属性 136](#_Toc420738328)

[初始化实例属性 136](#_Toc420738329)

[访问限制 137](#_Toc420738330)

[创建类属性 138](#_Toc420738331)

[类属性和实例属性名字冲突怎么办 139](#_Toc420738332)

[定义实例方法 140](#_Toc420738333)

[方法也是属性 140](#_Toc420738334)

[定义类方法（@classmethod） 142](#_Toc420738335)

[继承 143](#_Toc420738336)

[继承一个类 143](#_Toc420738337)

[判断类型 143](#_Toc420738338)

[多态 145](#_Toc420738339)

[多重继承 146](#_Toc420738340)

[获取对象信息 148](#_Toc420738341)

[魔术方法（特殊方法） 149](#_Toc420738342)

[Python的官方文档 149](#_Toc420738343)

[\_\_str\_\_和\_\_repr\_\_ 149](#_Toc420738344)

[\_\_cmp\_\_ 150](#_Toc420738345)

[\_\_len\_\_ 151](#_Toc420738346)

[数学运算-有理数（分数） 151](#_Toc420738347)

[类型转换 152](#_Toc420738348)

[@property 153](#_Toc420738349)

[\_\_slots\_\_ 155](#_Toc420738350)

[\_\_call\_\_ 155](#_Toc420738351)

[错误、调试和测试 156](#_Toc420738352)

[错误处理 156](#_Toc420738353)

[搭建环境 156](#_Toc420738354)

[安装python 156](#_Toc420738355)

[安装pip： 156](#_Toc420738356)

[开发编辑器或者IDE 157](#_Toc420738357)

[创建第一个Python项目“Hello World！” 158](#_Toc420738358)

[网页 160](#_Toc420738359)

[机器学习和深度学习学习资料 161](#_Toc420738360)

[深度学习阅读清单 167](#_Toc420738361)

[Free Online Books 167](#_Toc420738362)

[Courses 168](#_Toc420738363)

[Video and Lectures 168](#_Toc420738364)

[Papers 168](#_Toc420738365)

[Tutorials 169](#_Toc420738366)

[WebSites 169](#_Toc420738367)

[Datasets 169](#_Toc420738368)

[Frameworks 169](#_Toc420738369)

[Miscellaneous 170](#_Toc420738370)

[Python Scrapy笔记 171](#_Toc420738371)

[安装 171](#_Toc420738372)

[爬虫 176](#_Toc420738373)

[爬虫详解（入门教程）——精华 176](#_Toc420738374)

[Help-urllib2 176](#_Toc420738375)

[[Python]网络爬虫（一）：抓取网页的含义和URL基本构成 220](#_Toc420738376)

[[Python]网络爬虫（二）：利用urllib2通过指定的URL抓取网页内容 224](#_Toc420738377)

[[Python]网络爬虫（三）：异常的处理和HTTP状态码的分类 229](#_Toc420738378)

[[Python]网络爬虫（四）：Opener与Handler的介绍和实例应用 235](#_Toc420738379)

[[Python]网络爬虫（五）：urllib2的使用细节与抓站技巧 240](#_Toc420738380)

[[Python]网络爬虫（六）：一个简单的百度贴吧的小爬虫 248](#_Toc420738381)

[[Python]网络爬虫（七）：Python中的正则表达式教程 249](#_Toc420738382)

[[Python]网络爬虫（八）：糗事百科的网络爬虫（v0.3）源码及解析(简化更新) 268](#_Toc420738383)

[[Python]网络爬虫（九）：百度贴吧的网络爬虫（v0.4）源码及解析 272](#_Toc420738384)

[[Python]网络爬虫（十）：一个爬虫的诞生全过程（以山东大学绩点运算为例） 278](#_Toc420738385)

[[Python]网络爬虫（十一）：亮剑！爬虫框架小抓抓Scrapy闪亮登场！ 302](#_Toc420738386)

[[Python]网络爬虫（十二）：爬虫框架Scrapy的第一个爬虫示例入门教程 304](#_Toc420738387)

[[Python]项目打包：5步将py文件打包成exe文件 325](#_Toc420738388)

[实验楼：基于scrapy的天气数据采集 328](#_Toc420738389)

[一、实验说明 328](#_Toc420738390)

[1. 环境登录 328](#_Toc420738391)

[2. 环境介绍 328](#_Toc420738392)

[3. 环境使用 329](#_Toc420738393)

[二、项目介绍 329](#_Toc420738394)

[三、项目实战 330](#_Toc420738395)

[1. 安装Scrapy 330](#_Toc420738396)

[2. 创建项目 331](#_Toc420738397)

[3. 定义Item 332](#_Toc420738398)

[4. 编写获取天气数据的爬虫(Spider) 333](#_Toc420738399)

[5. 运行爬虫，对数据进行验证 336](#_Toc420738400)

[6. 保存爬取到的数据 337](#_Toc420738401)

[7. 把 ITEM\_PIPELINES 添加到设置中 339](#_Toc420738402)

[8. 运行爬虫 340](#_Toc420738403)

[四、答疑 342](#_Toc420738404)

[1、关于结果只出现城市的问题 342](#_Toc420738405)

[2、 关于只取到了9天的数据问题 345](#_Toc420738406)

# Note:

1，必须用熟：type，help，dir。

2，中文： #下面符合这个格式（正则）coding[:=]\s\*([\-w.]+)

coding=utf-8 or -\*-coding: utf-8 -\*-

3，常用基本数据类型.

int 整型

boolean 布尔

string 字符串

list 列表[]

tuple 元祖 ()

dict 字典

数据类型的可变和不可变

不可变类型：int，string,tuple

可变类型:list,dict

张 硕提示：大家可以用 dir(str) 这个方法，获得 str字符串的所有方法名单。

步进，倒序：

>>> c

'4234232sf23'

>>> c[::-1]

'32fs2324324'

5.变量之赋值一切皆为引用，动态性与动态类型之为何不再动态4，python里一切都是指针，所以不用再考虑指针这个问题了！

Python默认的编码格式是ASCII码，但是中文第Unicode码，一个中文占三个字符

Python 中引用开始时3开始的，不是一

## 30个有关Python的小技巧

### 1.1 拆箱a, b, c = 1, 2, 3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | >>> a, b, c = 1, 2, 3  >>> a, b, c  (1, 2, 3)  >>> a, b, c = [1, 2, 3]  >>> a, b, c  (1, 2, 3)  >>> a, b, c = (2 \* i + 1 for i in range(3))  >>> a, b, c  (1, 3, 5)  >>> a, (b, c), d = [1, (2, 3), 4]  >>> a  1  >>> b  2  >>> c  3  >>> d  4 |

1.2 拆箱变量交换

|  |  |
| --- | --- |
| 1  2  3  4 | >>> a, b = 1, 2  >>> a, b = b, a  >>> a, b  (2, 1) |

1.3 扩展拆箱（只兼容python3）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> a, \*b, c = [1, 2, 3, 4, 5]  >>> a  1  >>> b  [2, 3, 4]  >>> c  5 |

1.4 负数索引

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  >>> a[-1]  10  >>> a[-3]  8 |

1.5 切割列表

|  |  |
| --- | --- |
| 1  2  3 | >>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  >>> a[2:8]  [2, 3, 4, 5, 6, 7] |

1.6 负数索引切割列表

|  |  |
| --- | --- |
| 1  2  3 | >>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  >>> a[-4:-2]  [7, 8] |

1.7指定步长切割列表

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  >>> a[::2]  [0, 2, 4, 6, 8, 10]  >>> a[::3]  [0, 3, 6, 9]  >>> a[2:8:2]  [2, 4, 6] |

1.8 负数步长切割列表

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  >>> a[::-1]  [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]  >>> a[::-2]  [10, 8, 6, 4, 2, 0] |

1.9 列表切割赋值

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | >>> a = [1, 2, 3, 4, 5]  >>> a[2:3] = [0, 0]  >>> a  [1, 2, 0, 0, 4, 5]  >>> a[1:1] = [8, 9]  >>> a  [1, 8, 9, 2, 0, 0, 4, 5]  >>> a[1:-1] = []  >>> a  [1, 5] |

1.10 命名列表切割方式

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>> a = [0, 1, 2, 3, 4, 5]  >>> LASTTHREE = slice(-3, None)  >>> LASTTHREE  slice(-3, None, None)  >>> a[LASTTHREE]  [3, 4, 5] |

1.11 列表以及迭代器的压缩和解压缩

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> a = [1, 2, 3]  >>> b = ['a', 'b', 'c']  >>> z = zip(a, b)  >>> z  [(1, 'a'), (2, 'b'), (3, 'c')]  >>> zip(\*z)  [(1, 2, 3), ('a', 'b', 'c')] |

1.12 列表相邻元素压缩器

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | >>> a = [1, 2, 3, 4, 5, 6]  >>> zip(\*([iter(a)] \* 2))  [(1, 2), (3, 4), (5, 6)]    >>> group\_adjacent = lambda a, k: zip(\*([iter(a)] \* k))  >>> group\_adjacent(a, 3)  [(1, 2, 3), (4, 5, 6)]  >>> group\_adjacent(a, 2)  [(1, 2), (3, 4), (5, 6)]  >>> group\_adjacent(a, 1)  [(1,), (2,), (3,), (4,), (5,), (6,)]    >>> zip(a[::2], a[1::2])  [(1, 2), (3, 4), (5, 6)]    >>> zip(a[::3], a[1::3], a[2::3])  [(1, 2, 3), (4, 5, 6)]    >>> group\_adjacent = lambda a, k: zip(\*(a[i::k] for i in range(k)))  >>> group\_adjacent(a, 3)  [(1, 2, 3), (4, 5, 6)]  >>> group\_adjacent(a, 2)  [(1, 2), (3, 4), (5, 6)]  >>> group\_adjacent(a, 1)  [(1,), (2,), (3,), (4,), (5,), (6,)] |

1.13 在列表中用压缩器和迭代器滑动取值窗口

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | >>> def n\_grams(a, n):  ...     z = [iter(a[i:]) for i in range(n)]  ...     return zip(\*z)  ...  >>> a = [1, 2, 3, 4, 5, 6]  >>> n\_grams(a, 3)  [(1, 2, 3), (2, 3, 4), (3, 4, 5), (4, 5, 6)]  >>> n\_grams(a, 2)  [(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]  >>> n\_grams(a, 4)  [(1, 2, 3, 4), (2, 3, 4, 5), (3, 4, 5, 6)] |

1.14 用压缩器反转字典

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> m = {'a': 1, 'b': 2, 'c': 3, 'd': 4}  >>> m.items()  [('a', 1), ('c', 3), ('b', 2), ('d', 4)]  >>> zip(m.values(), m.keys())  [(1, 'a'), (3, 'c'), (2, 'b'), (4, 'd')]  >>> mi = dict(zip(m.values(), m.keys()))  >>> mi  {1: 'a', 2: 'b', 3: 'c', 4: 'd'} |

1.15 列表展开

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | >>> a = [[1, 2], [3, 4], [5, 6]]  >>> list(itertools.chain.from\_iterable(a))  [1, 2, 3, 4, 5, 6]    >>> sum(a, [])  [1, 2, 3, 4, 5, 6]    >>> [x for l in a for x in l]  [1, 2, 3, 4, 5, 6]    >>> a = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]  >>> [x for l1 in a for l2 in l1 for x in l2]  [1, 2, 3, 4, 5, 6, 7, 8]    >>> a = [1, 2, [3, 4], [[5, 6], [7, 8]]]  >>> flatten = lambda x: [y for l in x for y in flatten(l)] if type(x) is list else [x]  >>> flatten(a)  [1, 2, 3, 4, 5, 6, 7, 8] |

1.16 生成器表达式

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | >>> g = (x \*\* 2 for x in xrange(10))  >>> next(g)  0  >>> next(g)  1  >>> next(g)  4  >>> next(g)  9  >>> sum(x \*\* 3 for x in xrange(10))  2025  >>> sum(x \*\* 3 for x in xrange(10) if x % 3 == 1)  408 |

1.17 字典推导

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> m = {x: x \*\* 2 for x in range(5)}  >>> m  {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}    >>> m = {x: 'A' + str(x) for x in range(10)}  >>> m  {0: 'A0', 1: 'A1', 2: 'A2', 3: 'A3', 4: 'A4', 5: 'A5', 6: 'A6', 7: 'A7', 8: 'A8', 9: 'A9'} |

1.18 用字典推导反转字典

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> m = {'a': 1, 'b': 2, 'c': 3, 'd': 4}  >>> m  {'d': 4, 'a': 1, 'b': 2, 'c': 3}  >>> {v: k for k, v in m.items()}  {1: 'a', 2: 'b', 3: 'c', 4: 'd'} |

1.19 命名元组

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> Point = collections.namedtuple('Point', ['x', 'y'])  >>> p = Point(x=1.0, y=2.0)  >>> p  Point(x=1.0, y=2.0)  >>> p.x  1.0  >>> p.y  2.0 |

1.20 继承命名元组

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> class Point(collections.namedtuple('PointBase', ['x', 'y'])):  ...     \_\_slots\_\_ = ()  ...     def \_\_add\_\_(self, other):  ...             return Point(x=self.x + other.x, y=self.y + other.y)  ...  >>> p = Point(x=1.0, y=2.0)  >>> q = Point(x=2.0, y=3.0)  >>> p + q  Point(x=3.0, y=5.0) |

1.21 操作集合

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | >>> A = {1, 2, 3, 3}  >>> A  set([1, 2, 3])  >>> B = {3, 4, 5, 6, 7}  >>> B  set([3, 4, 5, 6, 7])  >>> A | B  set([1, 2, 3, 4, 5, 6, 7])  >>> A & B  set([3])  >>> A - B  set([1, 2])  >>> B - A  set([4, 5, 6, 7])  >>> A ^ B  set([1, 2, 4, 5, 6, 7])  >>> (A ^ B) == ((A - B) | (B - A))  True |

1.22 操作多重集合

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | >>> A = collections.Counter([1, 2, 2])  >>> B = collections.Counter([2, 2, 3])  >>> A  Counter({2: 2, 1: 1})  >>> B  Counter({2: 2, 3: 1})  >>> A | B  Counter({2: 2, 1: 1, 3: 1})  >>> A & B  Counter({2: 2})  >>> A + B  Counter({2: 4, 1: 1, 3: 1})  >>> A - B  Counter({1: 1})  >>> B - A  Counter({3: 1}) |

1.23 统计在可迭代器中最常出现的元素

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> A = collections.Counter([1, 1, 2, 2, 3, 3, 3, 3, 4, 5, 6, 7])  >>> A  Counter({3: 4, 1: 2, 2: 2, 4: 1, 5: 1, 6: 1, 7: 1})  >>> A.most\_common(1)  [(3, 4)]  >>> A.most\_common(3)  [(3, 4), (1, 2), (2, 2)] |

1.24 两端都可操作的队列

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | >>> Q = collections.deque()  >>> Q.append(1)  >>> Q.appendleft(2)  >>> Q.extend([3, 4])  >>> Q.extendleft([5, 6])  >>> Q  deque([6, 5, 2, 1, 3, 4])  >>> Q.pop()  4  >>> Q.popleft()  6  >>> Q  deque([5, 2, 1, 3])  >>> Q.rotate(3)  >>> Q  deque([2, 1, 3, 5])  >>> Q.rotate(-3)  >>> Q  deque([5, 2, 1, 3]) |

1.25 有最大长度的双端队列

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | >>> last\_three = collections.deque(maxlen=3)  >>> for i in xrange(10):  ...     last\_three.append(i)  ...     print ', '.join(str(x) for x in last\_three)  ...  0  0, 1  0, 1, 2  1, 2, 3  2, 3, 4  3, 4, 5  4, 5, 6  5, 6, 7  6, 7, 8  7, 8, 9 |

1.26 可排序词典

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | >>> m = dict((str(x), x) for x in range(10))  >>> print ', '.join(m.keys())  1, 0, 3, 2, 5, 4, 7, 6, 9, 8  >>> m = collections.OrderedDict((str(x), x) for x in range(10))  >>> print ', '.join(m.keys())  0, 1, 2, 3, 4, 5, 6, 7, 8, 9  >>> m = collections.OrderedDict((str(x), x) for x in range(10, 0, -1))  >>> print ', '.join(m.keys())  10, 9, 8, 7, 6, 5, 4, 3, 2, 1 |

1.27 默认词典

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | >>> m = dict()  >>> m['a']  Traceback (most recent call last):    File "<stdin>", line 1, in <module>  KeyError: 'a'  >>>  >>> m = collections.defaultdict(int)  >>> m['a']  0  >>> m['b']  0  >>> m = collections.defaultdict(str)  >>> m['a']  ''  >>> m['b'] += 'a'  >>> m['b']  'a'  >>> m = collections.defaultdict(lambda: '[default value]')  >>> m['a']  '[default value]'  >>> m['b']  '[default value]' |

1.28 默认字典的简单树状表达

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | >>> import json  >>> tree = lambda: collections.defaultdict(tree)  >>> root = tree()  >>> root['menu']['id'] = 'file'  >>> root['menu']['value'] = 'File'  >>> root['menu']['menuitems']['new']['value'] = 'New'  >>> root['menu']['menuitems']['new']['onclick'] = 'new();'  >>> root['menu']['menuitems']['open']['value'] = 'Open'  >>> root['menu']['menuitems']['open']['onclick'] = 'open();'  >>> root['menu']['menuitems']['close']['value'] = 'Close'  >>> root['menu']['menuitems']['close']['onclick'] = 'close();'  >>> print json.dumps(root, sort\_keys=True, indent=4, separators=(',', ': '))  {      "menu": {          "id": "file",          "menuitems": {              "close": {                  "onclick": "close();",                  "value": "Close"              },              "new": {                  "onclick": "new();",                  "value": "New"              },              "open": {                  "onclick": "open();",                  "value": "Open"              }          },          "value": "File"      }  } |

1.29 对象到唯一计数的映射

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | >>> import itertools, collections  >>> value\_to\_numeric\_map = collections.defaultdict(itertools.count().next)  >>> value\_to\_numeric\_map['a']  0  >>> value\_to\_numeric\_map['b']  1  >>> value\_to\_numeric\_map['c']  2  >>> value\_to\_numeric\_map['a']  0  >>> value\_to\_numeric\_map['b']  1 |

1.30 最大和最小的几个列表元素

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> a = [random.randint(0, 100) for \_\_ in xrange(100)]  >>> heapq.nsmallest(5, a)  [3, 3, 5, 6, 8]  >>> heapq.nlargest(5, a)  [100, 100, 99, 98, 98] |

1.31 两个列表的笛卡尔积

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | >>> for p in itertools.product([1, 2, 3], [4, 5]):  (1, 4)  (1, 5)  (2, 4)  (2, 5)  (3, 4)  (3, 5)  >>> for p in itertools.product([0, 1], repeat=4):  ...     print ''.join(str(x) for x in p)  ...  0000  0001  0010  0011  0100  0101  0110  0111  1000  1001  1010  1011  1100  1101  1110  1111 |

1.32 列表组合和列表元素替代组合

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | >>> for c in itertools.combinations([1, 2, 3, 4, 5], 3):  ...     print ''.join(str(x) for x in c)  ...  123  124  125  134  135  145  234  235  245  345  >>> for c in itertools.combinations\_with\_replacement([1, 2, 3], 2):  ...     print ''.join(str(x) for x in c)  ...  11  12  13  22  23  33 |

1.33 列表元素排列组合

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | >>> for p in itertools.permutations([1, 2, 3, 4]):  ...     print ''.join(str(x) for x in p)  ...  1234  1243  1324  1342  1423  1432  2134  2143  2314  2341  2413  2431  3124  3142  3214  3241  3412  3421  4123  4132  4213  4231  4312  4321 |

1.34 可链接迭代器

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | >>> a = [1, 2, 3, 4]  >>> for p in itertools.chain(itertools.combinations(a, 2), itertools.combinations(a, 3)):  ...     print p  ...  (1, 2)  (1, 3)  (1, 4)  (2, 3)  (2, 4)  (3, 4)  (1, 2, 3)  (1, 2, 4)  (1, 3, 4)  (2, 3, 4)  >>> for subset in itertools.chain.from\_iterable(itertools.combinations(a, n) for n in range(len(a) + 1))  ...     print subset  ...  ()  (1,)  (2,)  (3,)  (4,)  (1, 2)  (1, 3)  (1, 4)  (2, 3)  (2, 4)  (3, 4)  (1, 2, 3)  (1, 2, 4)  (1, 3, 4)  (2, 3, 4)  (1, 2, 3, 4) |

1.35 根据文件指定列类聚

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | >>> import itertools  >>> with open('contactlenses.csv', 'r') as infile:  ...     data = [line.strip().split(',') for line in infile]  ...  >>> data = data[1:]  >>> def print\_data(rows):  ...     print '\n'.join('\t'.join('{: <16}'.format(s) for s in row) for row in rows)  ...    >>> print\_data(data)  young               myope                   no                      reduced                 none  young               myope                   no                      normal                  soft  young               myope                   yes                     reduced                 none  young               myope                   yes                     normal                  hard  young               hypermetrope            no                      reduced                 none  young               hypermetrope            no                      normal                  soft  young               hypermetrope            yes                     reduced                 none  young               hypermetrope            yes                     normal                  hard  pre-presbyopic      myope                   no                      reduced                 none  pre-presbyopic      myope                   no                      normal                  soft  pre-presbyopic      myope                   yes                     reduced                 none  pre-presbyopic      myope                   yes                     normal                  hard  pre-presbyopic      hypermetrope            no                      reduced                 none  pre-presbyopic      hypermetrope            no                      normal                  soft  pre-presbyopic      hypermetrope            yes                     reduced                 none  pre-presbyopic      hypermetrope            yes                     normal                  none  presbyopic          myope                   no                      reduced                 none  presbyopic          myope                   no                      normal                  none  presbyopic          myope                   yes                     reduced                 none  presbyopic          myope                   yes                     normal                  hard  presbyopic          hypermetrope            no                      reduced                 none  presbyopic          hypermetrope            no                      normal                  soft  presbyopic          hypermetrope            yes                     reduced                 none  presbyopic          hypermetrope            yes                     normal                  none    >>> data.sort(key=lambda r: r[-1])  >>> for value, group in itertools.groupby(data, lambda r: r[-1]):  ...     print '-----------'  ...     print 'Group: ' + value  ...     print\_data(group)  ...  -----------  Group: hard  young               myope                   yes                     normal                  hard  young               hypermetrope            yes                     normal                  hard  pre-presbyopic      myope                   yes                     normal                  hard  presbyopic          myope                   yes                     normal                  hard  -----------  Group: none  young               myope                   no                      reduced                 none  young               myope                   yes                     reduced                 none  young               hypermetrope            no                      reduced                 none  young               hypermetrope            yes                     reduced                 none  pre-presbyopic      myope                   no                      reduced                 none  pre-presbyopic      myope                   yes                     reduced                 none  pre-presbyopic      hypermetrope            no                      reduced                 none  pre-presbyopic      hypermetrope            yes                     reduced                 none  pre-presbyopic      hypermetrope            yes                     normal                  none  presbyopic          myope                   no                      reduced                 none  presbyopic          myope                   no                      normal                  none  presbyopic          myope                   yes                     reduced                 none  presbyopic          hypermetrope            no                      reduced                 none  presbyopic          hypermetrope            yes                     reduced                 none  presbyopic          hypermetrope            yes                     normal                  none  -----------  Group: soft  young               myope                   no                      normal                  soft  young               hypermetrope            no                      normal                  soft  pre-presbyopic      myope                   no                      normal                  soft  pre-presbyopic      hypermetrope            no                      normal                  soft  presbyopic          hypermetrope            no                      normal                  soft |

## 帮助使用

Help（str.find()）

## 注释

Ulipad 中是：

Ctrl + ？是加注释

Ctrl+ | 是取消注释

## Sort

## Sorted

python列表排序   
  
简单记一下python中List的sort方法（或者sorted内建函数）的用法。   
  
关键字：   
python列表排序 python字典排序 sorted   
  
  
List的元素可以是各种东西，字符串，字典，自己定义的类等。   
  
sorted函数用法如下：

**Python代码**



1. sorted(data, cmp=None, key=None, reverse=False)

其中，data是待排序数据，可以使List或者iterator, cmp和key都是函数，这两个函数作用与data的元素上产生一个结果，sorted方法根据这个结果来排序。   
cmp(e1, e2) 是带两个参数的比较函数, 返回值: 负数: e1 < e2, 0: e1 == e2, 正数: e1 > e2. 默认为 None, 即用内建的比较函数.   
key 是带一个参数的函数, 用来为每个元素提取比较值. 默认为 None, 即直接比较每个元素.   
通常, key 和 reverse 比 cmp 快很多, 因为对每个元素它们只处理一次; 而 cmp 会处理多次.   
  
通过例子来说明sorted的用法：   
  
1. 对由tuple组成的List排序

**Python代码**



1. >>> students = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10),]

用key函数排序(lambda的用法见 注释1)

**Python代码**



1. >>> sorted(students, key=**lambda** student : student[2])   # sort by age
2. [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

用cmp函数排序

**Python代码**



1. >>> sorted(students, cmp=**lambda** x,y : cmp(x[2], y[2])) # sort by age
2. [('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

用 operator 函数来加快速度, 上面排序等价于:(itemgetter的用法见 注释2)

**Python代码**

1. >>> **from** operator **import** itemgetter, attrgetter
2. >>> sorted(students, key=itemgetter(2))

用 operator 函数进行多级排序

**Python代码**



1. >>> sorted(students, key=itemgetter(1,2))  # sort by grade then by age
2. [('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]

2. 对由字典排序

**Python代码**



1. >>> d = {'data1':3, 'data2':1, 'data3':2, 'data4':4}
2. >>> sorted(d.iteritems(), key=itemgetter(1), reverse=True)
3. [('data4', 4), ('data1', 3), ('data3', 2), ('data2', 1)]

注释1   
参考：http://jasonwu.me/2011/10/29/introduce-to-python-lambda.html   
  
注释2   
参考:http://ar.newsmth.net/thread-90745710c90cf1.html   
class itemgetter(\_\_builtin\_\_.object)   
|  itemgetter(item, ...) --> itemgetter object   
|   
|  Return a callable object that fetches the given item(s) from its operand.   
|  After, f=itemgetter(2), the call f(r) returns r[2].   
|  After, g=itemgetter(2,5,3), the call g(r) returns (r[2], r[5], r[3])   
  
相当于

**Python代码**



1. **def** itemgetter(i,\*a):
2. **def** func(obj):
3. r = obj[i]
4. **if** a:
5. r = (r,) + tuple(obj[i] **for** i **in** a)
6. **return** r
7. **return** func
9. >>> a = [1,2,3]
10. >>> b=operator.itemgetter(1)
11. >>> b(a)
12. 2
13. >>> b=operator.itemgetter(1,0)
14. >>> b(a)
15. (2, 1)
16. >>> b=itemgetter(1)
17. >>> b(a)
18. 2
19. >>> b=itemgetter(1,0)
20. >>> b(a)
21. (2, 1)

## Sorted()

返回一个字符串的排序为list格式。

>>> d

'3233fjdsDjfdsKJKLLfjdsjfjkj'

>>> l = sorted(d)

>>> l

['2', '3', '3', '3', 'D', 'J', 'K', 'K', 'L', 'L', 'd', 'd', 'd', 'f', 'f', 'f', 'f', 'j', 'j', 'j', 'j', 'j', 'j', 'k', 's', 's', 's']

>>> import string

>>> print sorted(l\_list, key = string.upper)

['D', 'd', 'd', 'd', 'f', 'f', 'f', 'f', 'J', 'j', 'j', 'j', 'j', 'j', 'j', 'K', 'K', 'k', 'L', 'L', 's', 's', 's']

key = string.upper 这个可以理解为这样在sorted对每个list单位做了key后面的的函数操作，然后再做了排序

## assert 断言

判断一个数据是否符合你的心里预期。。。

类型断言和内容断言是检测一个函数的基本方法

断言就是异常

类型断言：type(a) == list

判断a是否为list类型，正确过，错误，报错

Python中assert用来判断语句的真假，如果为假的话将触发AssertionError错误

1、assert语句用来声明某个条件是真的。

2、如果你非常确信某个你使用的列表中至少有一个元素，而你想要检验这一点，并且在它非真的时候引发一个错误，那么assert语句是应用在这种情形下的理想语句。

3、当assert语句失败的时候，会引发一AssertionError。

测试程序：

>>> mylist = ['item']

>>> assert len(mylist) >= 1

>>> mylist.pop()

'item'

>>> assert len(mylist) >= 1

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

AssertionError

## 变量None的判断

代码中经常会有变量是否为None的判断，有三种主要的写法：

 第一种是`if x is None`；

第二种是 `if not x：`；

第三种是`if not x is None`（这句这样理解更清晰`if not (x is None)`） 。

如果你觉得这样写没啥区别，那么你可就要小心了，这里面有一个坑。先来看一下代码：

**[python]** [view plain](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "view plain)[copy](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "copy)

1. >>> x = 1
2. >>> **not** x
3. False
4. >>> x = [1]
5. >>> **not** x
6. False
7. >>> x = 0
8. >>> **not** x
9. True
10. >>> x = [0]         # You don't want to fall in this one.
11. >>> **not** x
12. False

在python中 None,  False, 空字符串"", 0, 空列表[], 空字典{}, 空元组()都相当于False ，即：

**[python]** [view plain](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "view plain)[copy](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "copy)

1. <strong>**not** None == **not** False == **not** '' == **not** 0 == **not** [] == **not** {} == **not** ()</strong>

因此在使用列表的时候，如果你想区分x==[]和x==None两种情况的话, 此时`if not x:`将会出现问题：

**[python]** [view plain](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "view plain)[copy](http://blog.csdn.net/sasoritattoo/article/details/12451359" \o "copy)

1. >>> x = []
2. >>> y = None
3. >>>
4. >>> x **is** None
5. False
6. >>> y **is** None
7. True
8. >>>
9. >>>
10. >>> **not** x
11. True
12. >>> **not** y
13. True
14. >>>
15. >>>
16. >>> **not** x **is** None
17. >>> True
18. >>> **not** y **is** None
19. False
20. >>>

也许你是想判断x是否为None，但是却把`x==[]`的情况也判断进来了，此种情况下将无法区分。

**对于习惯于使用if not x这种写法的pythoner，必须清楚x等于None,  False, 空字符串"", 0, 空列表[], 空字典{}, 空元组()时对你的判断没有影响才行。**

而对于`if x is not None`和`if not x is None`写法，很明显前者更清晰，而后者有可能使读者误解为`if (not x) is None`，因此推荐前者，同时这也是谷歌推荐的风格

**结论：**

**`if x is not None`是最好的写法，清晰，不会出现错误，以后坚持使用这种写法。**

**使用if not x这种写法的前提是：必须清楚x等于None,  False, 空字符串"", 0, 空列表[], 空字典{}, 空元组()时对你的判断没有影响才行。**

================================================================

不过这并不适用于变量是函数的情况，以下转载自：[https://github.com/wklken/stackoverflow-py-top-qa/blob/master/contents/qa-control-flow.md](https://github.com/wklken/stackoverflow-py-top-qa/blob/master/contents/qa-control-flow.md" \t "_blank)

foo is None 和 foo == None的区别

问题 [链接](http://stackoverflow.com/questions/26595/is-there-any-difference-between-foo-is-none-and-foo-none" \t "_blank)

if foo is None: pass

if foo == None: pass

如果比较相同的对象实例，is总是返回True 而 == 最终取决于 "**eq**()"

>>> class foo(object):

def \_\_eq\_\_(self, other):

return True

>>> f = foo()

>>> f == None

True

>>> f is None

False

>>> list1 = [1, 2, 3]

>>> list2 = [1, 2, 3]

>>> list1==list2

True

>>> list1 is list2

False

另外

(ob1 is ob2) 等价于 (id(ob1) == id(ob2))

################################################################################

补充，2013.10.09

转自[http://zhidao.baidu.com/question/514056244.html](http://zhidao.baidu.com/question/514056244.html" \t "_blank)

python中的not具体表示是什么，举个例子说一下，衷心的感谢

在python中not是逻辑判断词，用于[布尔型](http://zhidao.baidu.com/search?word=%E5%B8%83%E5%B0%94%E5%9E%8B&fr=qb_search_exp&ie=utf8" \t "_blank)True和False，not True为False，not False为True，以下是几个常用的not的用法：

(1) not与逻辑[判断句](http://zhidao.baidu.com/search?word=%E5%88%A4%E6%96%AD%E5%8F%A5&fr=qb_search_exp&ie=utf8" \t "_blank)if连用，代表not后面的表达式为False的时候，执行[冒号](http://zhidao.baidu.com/search?word=%E5%86%92%E5%8F%B7&fr=qb_search_exp&ie=utf8" \t "_blank)后面的语句。比如：

a = False

if not a: (这里因为a是False，所以not a就是True)

print "hello"

这里就能够输出结果hello

(2) 判断元素是否在列表或者字典中，if a not in b，a是元素，b是列表或字典，这句话的意思是如果a不在列表b中，那么就执行[冒号](http://zhidao.baidu.com/search?word=%E5%86%92%E5%8F%B7&fr=qb_search_exp&ie=utf8" \t "_blank)后面的语句，比如：

a = 5

b = [1, 2, 3]

if a not in b:

print "hello"

这里也能够输出结果hello

not x 意思相当于 if x is false, then True, else False

## Pip

发现Python下也有同样的工具，包括easy\_install和Pip。不过，我没有细看easy\_install的方法，这就简单的介绍一下Pip的安装与使用：

准备：

$ curl -O http://python-distribute.org/distribute\_setup.py $ python distribute\_setup.py

安装：

$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py $ python get-pip.py

使用方法：

$ pip install SomePackage

$ pip search "query"

$ pip install --upgrade SomePackage

$ pip install --upgrade SomePackage==version

补充：

包安装后的py文件路径：/usr/local/lib/python2.7/dist-packages

## Linux虚拟机

Python 环境的虚拟机，可以模拟Python配置不改变本机的配置

以下操作都是在：/home/zhangshuo/virtualenv/ 目录下操作的。

安装虚拟机： sudo apt-get install python-virtualenv

安装环境：virtualenv el

进入环境：1） source ~/virtual/el/bin/activate

2) . ~/virtual/el/bin/activate

## 安装Django

Pip install django

# 类型

## 变量作用域

b = 3 # 全局变量

def a():

return b #本身没有局部变量，返回全局变量

def c():

global b

b = 8 这个是全局变量的定义，函数外也被定义为了b=8

return b 返回的是全局变量

def d():

return b 这个也是返回的全局变量

def e():

b = 3

return b 局部变量

print a()

print c()

print d()

print e()

print b

## 位移

这个移动是基于二进制移动的需注意bin()看十进制的二进制码

### 右移

1001

100

10

1

>>> bin(9)

'0b1001'

>>> 9>>1

4

>>> bin(4)

'0b100'

>>> 4>>1

2

>>> bin(2)

'0b10'

>>> 2>>1

1

>>> bin(1)

'0b1'

### 左移

11

110

1100

>>> bin(3)

'0b11'

>>> 3<<1

6

>>> bin(6)

'0b110'

>>> 6<<1

12

>>> bin(12)

'0b1100'

### 计算公式

x << y

答案就是x\*（2的y次方）

## 编码格式

#### 字符编码

我们已经讲过了，字符串也是一种数据类型，但是，字符串比较特殊的是还有一个编码问题。

因为计算机只能处理数字，如果要处理文本，就必须先把文本转换为数字才能处理。最早的计算机在设计时采用8个比特（bit）作为一个字节（byte），所以，一个字节能表示的最大的整数就是255（二进制11111111=十进制255），如果要表示更大的整数，就必须用更多的字节。比如两个字节可以表示的最大整数是65535，4个字节可以表示的最大整数是4294967295。

由于计算机是美国人发明的，因此，最早只有127个字母被编码到计算机里，也就是大小写英文字母、数字和一些符号，这个编码表被称为ASCII编码，比如大写字母A的编码是65，小写字母z的编码是122。

但是要处理中文显然一个字节是不够的，至少需要两个字节，而且还不能和ASCII编码冲突，所以，中国制定了GB2312编码，用来把中文编进去。

你可以想得到的是，全世界有上百种语言，日本把日文编到Shift\_JIS里，韩国把韩文编到Euc-kr里，各国有各国的标准，就会不可避免地出现冲突，结果就是，在多语言混合的文本中，显示出来会有乱码。



因此，Unicode应运而生。Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。现代操作系统和大多数编程语言都直接支持Unicode。

现在，捋一捋ASCII编码和Unicode编码的区别：ASCII编码是1个字节，而Unicode编码通常是2个字节。

字母A用ASCII编码是十进制的65，二进制的01000001；

字符0用ASCII编码是十进制的48，二进制的00110000，注意字符'0'和整数0是不同的；

汉字中已经超出了ASCII编码的范围，用Unicode编码是十进制的20013，二进制的01001110 00101101。

你可以猜测，如果把ASCII编码的A用Unicode编码，只需要在前面补0就可以，因此，A的Unicode编码是00000000 01000001。

新的问题又出现了：如果统一成Unicode编码，乱码问题从此消失了。但是，如果你写的文本基本上全部是英文的话，用Unicode编码比ASCII编码需要多一倍的存储空间，在存储和传输上就十分不划算。

所以，本着节约的精神，又出现了把Unicode编码转化为“可变长编码”的UTF-8编码。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，**常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节**。如果你要传输的文本包含大量英文字符，用UTF-8编码就能节省空间：

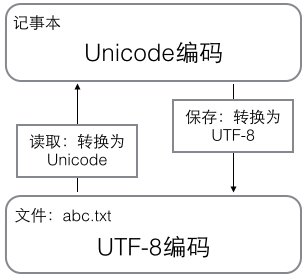
|  |  |  |  |
| --- | --- | --- | --- |
| **字符** | **ASCII** | **Unicode** | **UTF-8** |
| A | 01000001 | 00000000 01000001 | 01000001 |
| 中 | x | 01001110 00101101 | 11100100 10111000 10101101 |

从上面的表格还可以发现，UTF-8编码有一个额外的好处，就是ASCII编码实际上可以被看成是UTF-8编码的一部分，所以，大量只支持ASCII编码的历史遗留软件可以在UTF-8编码下继续工作。

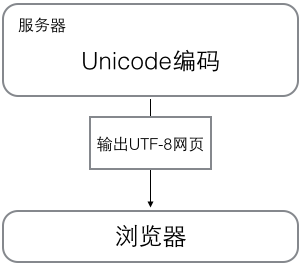
搞清楚了ASCII、Unicode和UTF-8的关系，我们就可以总结一下现在计算机系统通用的字符编码工作方式：

在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。

用记事本编辑的时候，从文件读取的UTF-8字符被转换为Unicode字符到内存里，编辑完成后，保存的时候再把Unicode转换为UTF-8保存到文件：



浏览网页的时候，服务器会把动态生成的Unicode内容转换为UTF-8再传输到浏览器：



所以你看到很多网页的源码上会有类似<meta charset="UTF-8" />的信息，表示该网页正是用的UTF-8编码。

#### Python的字符串

搞清楚了令人头疼的字符编码问题后，我们再来研究Python对Unicode的支持。

因为Python的诞生比Unicode标准发布的时间还要早，所以最早的Python只支持ASCII编码，普通的字符串'ABC'在Python内部都是ASCII编码的。Python提供了ord()和chr()函数，可以把字母和对应的数字相互转换：

>>> ord('A')

65

>>> chr(65)

'A'

Python在后来添加了对Unicode的支持，以Unicode表示的字符串用u'...'表示，比如：

>>> **print** u'中文'

中文

>>> u'中'

u'\u4e2d'

写u'中'和u'\u4e2d'是一样的，\u后面是十六进制的Unicode码。因此，u'A'和u'\u0041'也是一样的。

两种字符串如何相互转换？字符串'xxx'虽然是ASCII编码，但也可以看成是UTF-8编码，而u'xxx'则只能是Unicode编码。

把u'xxx'转换为UTF-8编码的'xxx'用encode('utf-8')方法：

>>> u'ABC'.encode('utf-8')

'ABC'

>>> u'中文'.encode('utf-8')

'\xe4\xb8\xad\xe6\x96\x87'

英文字符转换后表示的UTF-8的值和Unicode值相等（但占用的存储空间不同），而中文字符转换后1个Unicode字符将变为3个UTF-8字符，你看到的\xe4就是其中一个字节，因为它的值是228，没有对应的字母可以显示，所以以十六进制显示字节的数值。len()函数可以返回字符串的长度：

>>> len(u'ABC')

3

>>> len('ABC')

3

>>> len(u'中文')

2

>>> len('\xe4\xb8\xad\xe6\x96\x87')

6

反过来，把UTF-8编码表示的字符串'xxx'转换为Unicode字符串u'xxx'用decode('utf-8')方法：

>>> 'abc'.decode('utf-8')

u'abc'

>>> '\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')

u'\u4e2d\u6587'

>>> **print** '\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')

中文

由于Python源代码也是一个文本文件，所以，当你的源代码中包含中文的时候，在保存源代码时，就需要务必指定保存为UTF-8编码。当Python解释器读取源代码时，为了让它按UTF-8编码读取，我们通常在文件开头写上这两行：

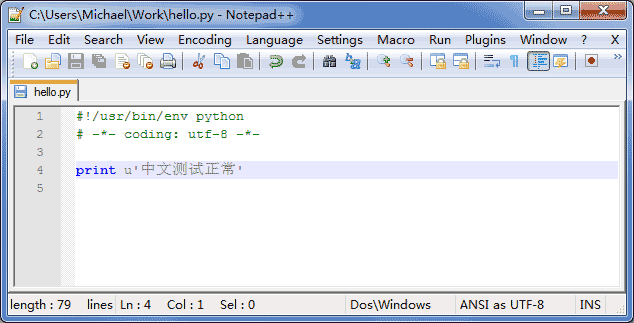
*#!/usr/bin/env python*

*# -\*- coding: utf-8 -\*-*

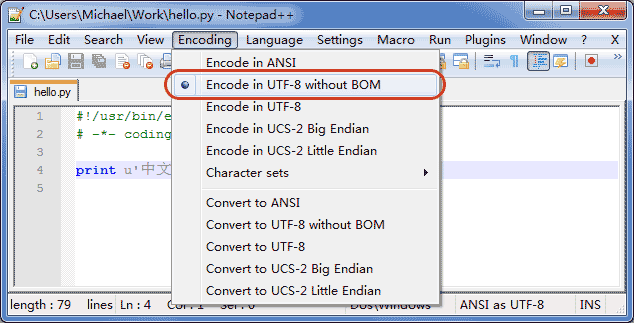
第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释；

第二行注释是为了告诉Python解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。

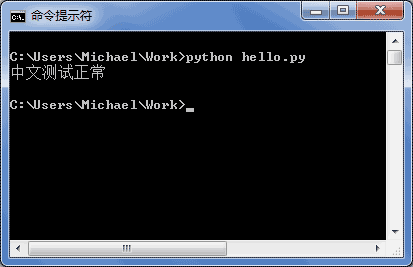
如果你使用Notepad++进行编辑，除了要加上# -\*- coding: utf-8 -\*-外，中文字符串必须是Unicode字符串：



申明了UTF-8编码并不意味着你的.py文件就是UTF-8编码的，必须并且要确保Notepad++正在使用UTF-8 without BOM编码：



如果.py文件本身使用UTF-8编码，并且也申明了# -\*- coding: utf-8 -\*-，打开命令提示符测试就可以正常显示中文：



小结

由于历史遗留问题，Python 2.x版本虽然支持Unicode，但在语法上需要'xxx'和u'xxx'两种字符串表示方式。

Python当然也支持其他编码方式，比如把Unicode编码成GB2312：

>>> u'中文'.encode('gb2312')

'\xd6\xd0\xce\xc4'

但这种方式纯属自找麻烦，如果没有特殊业务要求，请牢记仅使用Unicode和UTF-8这两种编码方式。

在Python 3.x版本中，把'xxx'和u'xxx'统一成Unicode编码，即写不写前缀u都是一样的，而以字节形式表示的字符串则必须加上b前缀：b'xxx'。

格式化字符串的时候，可以用Python的交互式命令行测试，方便快捷。

#### 详解 python 中文编码与处理

发表于3年前(2012-08-24 14:46)   阅读（5574） | 评论（[0](http://my.oschina.net/leejun2005/blog/74430" \l "comments)） **19**人收藏此文章, [我要收藏](javascript:add_to_favor(74430,3)" \o "添加到收藏夹)

赞0

**[大约十一点零八发，秒杀云主机赢P8手机](http://activity.hwclouds.com/ten/?utm_source=%E5%BC%80%E6%BA%90%E4%B8%AD%E5%9B%BD&utm_medium=MediaB&utm_term=%E6%96%87%E7%AB%A0%E9%A1%B5%E9%A1%B6%E9%83%A8%E6%96%87%E5%AD%97%E9%93%BE&utm_campaign=1008" \t "_blank)**

目录[-]

 [一、使用中文字符](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_1)

 [二、中文字符的存储](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_2)

 [三、中文字符的IO操作](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_3)

 [四、type ‘str’和 type ‘unicode’](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_4)

 [五、三行代码解决编码问题： Python 2.x字符串编码的小结](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_5)

 [六、跨平台处理技巧](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_6)

 [七、异常处理](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_7)

 [八、最后附一些关于 python 编码问题的总结：](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_8)

 [九、参考资料](http://my.oschina.net/leejun2005/blog/74430" \l "OSC_h3_9)

**注意：本文只是针对 python 2，在 python 3 中，编码方式与处理技巧有些许变化，具体请参考：**

**[Python 2 与 Python 3 的差异对比： http://my.oschina.net/leejun2005/blog/173553](http://my.oschina.net/leejun2005/blog/173553" \t "_blank)**

##### ****一、使用中文字符****

在python源码中如果使用了中文字符，运行时会有错误，解决的办法是在源码的开头部分加入字符编码的声明，下面是一个例子：   
#!/usr/bin/env python   
# -\*- coding: cp936 -\*-   
Python Tutorial中指出，python的源文件可以编码ASCII以外的字符集，最好的做法是在#!行后面用一个特殊的注释行来定义字符集：   
# -\*- coding: encoding -\*-   
根据这个声明，Python会尝试将文件中的字符编码转为encoding编码，并且，它尽可能的将指定地编码直接写成Unicode文本。   
注意，coding:encoding只是告诉Python文件使用了encoding格式的编码，但是编辑器可能会以自己的方式存储.py文件，因此最后文件保存的时候还需要编码中选指定的ecoding才行。 

##### ****二、中文字符的存储****

>>> str = u"中文"   
>>> str   
u'\xd6\xd0\xce\xc4'   
>>> str = "中文"   
>>> str   
'\xd6\xd0\xce\xc4'   
u"中文"只是声明unicode，实际的编码并没有变。这样子就发生变化了：   
>>> str = "中文"   
>>> str   
'\xd6\xd0\xce\xc4'   
>>> str = str.decode("gb2312")   
>>> str   
u'\u4e2d\u6587'   
更进一步：   
>>> s = '中文'   
>>> s.decode('gb2312')   
u'\u4e2d\u6587'   
>>> len(s)   
4   
>>> len(s.decode('gb2312'))   
2   
>>> s = u'中文'   
>>> len(s)   
4   
>>> s = '中文test'   
>>> len(s)   
8   
>>> len(s.decode('gb2312'))   
6   
>>> s = '中文test，'   
>>> len(s)   
10   
>>> len(s.decode('gb2312'))   
7   
可以看出，对于实际Non-ASCII编码存储的字符串，python可以正确的识别出其中的中文字符以及中文上下文中的标点符号。   
前缀“u”表示“后面这个字符串“是一个Unicode字符串”，这仅仅是一个声明，并不表示这个字符串就真的是Unicode了；就好比某正太声称自己已满18岁，但实际上他的真实年龄并不确定，现在体育界年龄造假可不稀罕幺!   
那么声明成u有什么作用呢？对于Python来说，只要你声明某字符串是Unicode，它就会用Unicode的一套机制对它进行处理。比方说，做字符串操作的时候会动用到内部的Unicode处理函数，保存的时候以Unicode字符（双字节）进行保存。等等。显而易见，对于一个实际上并不是Unicode的字符串，做Unicode动作的处理，是有可能会出问题的。 u前缀只适用于你的字符串常量真的是Unicode的情况 。 

##### ****三、中文字符的IO操作****

用python处理字符串很容易，但是在处理中文的时候需要注意一些问题。比如：   
a = "我们是python爱好者"   
print a[0]   
只能输出“我”字的前半部分，要想输出整个的“我”字还需要：   
b = a[0:2]   
print b   
才行，很不方便，并且当一段文本中同时有中英文如何处理？最好的办法就是转换为unicode。像这样：   
c = unicode(a, "gb2312")   
print c[0]   
这个时候c的下标对应的就是每一个字符，不再是字节，并且通过len(c)就可以获得字符数！还可以很方便的转换为其他编码，比如转换为utf-8：

d = c.encode("utf-8")

##### ****四、type ‘str’和 type ‘unicode’****

<type ‘str’>将字符串看作是字节的序列，而<type ‘unicode’>则将其看作是字符的序列，单个字符可能占用多个字节；字节相对于字符，其在存储层次中更低一些。

str转换为unicode要decode，可以这样想，因为要把字节序列解释成字符序列，字节序列是底层的存放方式，解码（decode）成更高层的字符以便使用；同理，unicode转换为str要encode，就象信息编码（encode）后才存储一样：   
s.decode(encoding) <type 'str'> to <type 'unicode'>   
u.encode(encoding) <type 'unicode'> to <type 'str'>   
例如：   
>>> s = 'str'   
>>> type(s)   
<type 'str'>   
>>> type(s.decode())   
<type 'unicode'>   
>>> s = u'str'   
>>> type(s)   
<type 'unicode'>   
>>> type(s.encode())   
<type 'str'>   
处理中文数据时最好采用如下方式：   
1. Decode early（尽早decode, 将文件中的内容转化成unicode再进行下一步处理)   
2. Unicode everywhere (程序内部处理都用unicode)   
3. Encode late (最后encode回所需的encoding, 例如把最终结果写进结果文件)   
下面是一个简单的演示，用re库查询一个中文字符串并打印：   
>>> p = re.compile(unicode("测试(.\*)", "gb2312"))   
>>> s = unicode("测试一二三", "gb2312")   
>>> for i in p.findall(s):   
print i.encode("gb2312")

一二三

##### ****五、三行代码解决编码问题： Python 2.x字符串编码的小结****

解决这一类的问题最好的方法就是在程序开头加上以下几行代码:

[?](http://my.oschina.net/leejun2005/blog/74430)

|  |  |
| --- | --- |
| 1  2  3  4 | #coding:utf-8  import sys  reload(sys)  sys.setdefaultencoding("utf-8") |

那么就可助你解决几乎95%的这种问题, 但是如果想刨根问底的话, 就需要去了解很多东西了.   
首先, 这个就是Python语言本身的问题. 因为在Python 2.x的语法中, 默认的str并不是真正意义上我们理解的字符串, 而是一个byte数组, 或者可以理解成一个纯ascii码字符组成的字符串, 与Python 3中的bytes类型的变量对应; 而真正意义上通用的字符串则是unicode类型的变量, 它则与Python 3中的str变量对应. 本来应该用作byte数组的类型, 却被用来做字符串用, 这种看似奇葩的设定是Python 2一直被人诟病的东西, 不过也没有办法, 为了与之前的程序保持兼容.   
在Python 2中作为两种字符串类型, str与unicode之间就需要各种转换的方式. 首先是一种显式转换的方式, 就是encode和decode两种方法. 在这里这两货的意思很容易被搞反, 科学的调用方式是:   
str --- decode方法 ---> unicode   
unicode --- encode方法 ---> str   
比如:

[?](http://my.oschina.net/leejun2005/blog/74430)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | >>> type('x')  <type 'str'>  >>> type('x'.decode('utf-8'))  <type 'unicode'>  >>> type(u'x'.encode('utf-8'))  <type 'str'> |

这个逻辑是这样的, 对于unicode字符串使用utf-8编码进行编码, 即调用encode('utf-8')方法生成byte数组类型的结果. 相反对于byte数组进行解码, 生成unicode字符串. 这个新手表示理解不能, 不过熟悉了就见怪不怪了.   
另外是隐式的转换, 和C语言中的int转double类似, 当一个unicode字符串和一个str字符串进行连接的时候会默认自动将str字符串转换成unicode类型然后再连接. 而这个时候使用的编码方式则是系统所默认的编码方式. 使用:   
import sys   
print sys.getdefaultencoding()   
可以得到当前默认的编码方式, 是不是'ascii'? 是的话就恭喜你中彩了~!! 在这个时候如果有以下一行代码就保证会出错:

[?](http://my.oschina.net/leejun2005/blog/74430)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | >>> x = u'喵'  >>> x  u'\u55b5'  >>> y = x.encode('utf-8')  >>> x + y  Traceback (most recent call last):  File "", line 1, in  UnicodeDecodeError: 'ascii' codec can't decode byte 0xe5 in position 0: ordinal not in range(128) |

x是unicode类型的变量, y是x经过encode后的结果是str类型的变量. x + y的时候, 首先要将y转换成unicode字符串, 那么使用什么编码格式转换呢, 用utf-8还是gb2312或者还是utf-16? 这个时候就要根据sys.getdefaultencoding()来确定, 而sys.getdefaultencoding()是'ascii'编码, 在ascii字符表中不存在0xe5这种大于128的字符存在, 所以当然报错啦! 通过加入

[?](http://my.oschina.net/leejun2005/blog/74430)

|  |  |
| --- | --- |
| 1  2  3  4 | #coding:utf-8  import sys  reload(sys)  sys.setdefaultencoding("utf-8") |

则可以将默认的编码转换格式变成utf-8, 且大多数情况下, 程序中的字符串是通过utf-8来编码的, 所以只要加上以上三行就可以了.   
但是有没有觉得, 加上这些会使得代码有些dirty? 咳, 至少对于我来说确实很dirty. 所以我觉得平时写程序的过程中要养成尽量使用显示的转换的习惯, 并且要明确某个函数返回的到底是str还是unicode, 凡是str的主动decode成unicode, 不要将两者混淆掉, 这样写出来的代码才比较干净. 此外还可以在代码最上方加入'from future import unicode\_literals'可以默认将用户自定义字符串变成unicode类型.   
最后我想大吼一声Python 2.x中str不是字符串, 而是B♂Y♂T♂E♂数♂组~!! 

##### ****六、跨平台处理技巧****

如果一个project必须在两个平台上开发，程序应该使用同样的encoding，比如要求所有的文件都使用UTF-8，如果实在不能统一（一般是为了满足许多所谓专家学者莫名其妙的要求），可以退而求其次，用当前系统编码决定文件内的编码：   
import locale   
import string   
import re   
#根据当前系统的encoding构造需要的编码取值   
lang = string.upper(locale.setlocale(locale.LC\_ALL, ""))   
textencoding = None 3   
#检查编码的值是不是满足我们需要的情况   
if re.match("UTF-8", lang) != None:   
# UTF-8编码   
textencoding = "utf-8"   
elif re.match(r"CHINESE|CP936", lang):   
# Windows下的GB编码   
textencoding = "gb18030"   
elif re.match(r"GB2312|GBK|GB18030", lang):   
# Linux下的GB编码   
textencoding = "gb18030"   
else:   
# 其他情况，抛个错误吧   
raise UnicodeError   
fd = file(filename, "r")   
fulltextlist = fd.readlines()   
# 把每一行转换成unicode   
for each in len(fulltextlist):   
fulltextlist[i] = unicode(each, textencoding)   
fd.close()   
# 如果要打印的话，可以用text.encode(encoding)来恢复成多字节编码 

##### ****七、异常处理****

编码encoding发生在Unicode字符串转换为字节序列时，而解码decoding发生在字节序列转换为Unicode字符串时（encoding always takes a Unicode string and returns a bytes sequence, and decoding always takes a bytes sequence and returns a Unicode string）。   
UnicodeDecodeError   
UnicodeDncodeError通常发生在将str字符串解码为特定Unicode字符串时。由于不同的编码只能映射部分str字符串到对应的Unicode字符，所以遇到一些字符时解码会失败。   
UnicodeEncodeError   
UnicodeEncodeError通常发生在将Unicode字符串编码为特定字节序列时。由于不同的编码只能映射部分Unicode字符到对应的str字符串，所以遇到一些字符时编码会失败。   
处理python编码转换时的UnicodeDecodeError异常   
python提供的unicode转换不像iconv或是mbstowcs之类的方便。如果转换一段unicode("1234中文",'ascii') 到utf8，会直接出现UnicodeDecodeError的错误。如果在你能预知字串符的编码的时候，比如你用unicode('1234中文', 'gbk') 就不会出现错误；不过很多时候，会出现CJK混合的情况，如果要做到将一段CJK文件转换成unicode可能就行不通了。好在python的codecs提供了register\_error这个功能：   
register\_error(name, error\_handler)   
原理很简单，不过要先看unicode是如何处理异常的。unicode这个函数是将一段string按输入的编码转换成目标的编码，如果出现了不与输入编码相符的，会出现一个UnicodeDecodeError的异常，通常有三种处理方法：strict、replace、ignore；默认是 strict,就是直接raise UnicodeDecodeError。通过register\_error，我们也可以有自己的处理方法，如果遇到与输入的编码不符的时候，我们就自己识别，比如GBK、BIG5、JP的字符。   
def cjk\_replace(exc):   
if not isinstance(exc, UnicodeDecodeError):   
raise TypeError("don't know how to handle %r" % exc)   
if exc.end + 1 > len(exc.object):   
raise TypeError('unknown codec ,the object too short!')   
ch1 = ord(exc.object[exc.start:exc.end])   
newpos = exc.end + 1   
ch2 = ord(exc.object[exc.start + 1:newpos])   
sk = exc.object[exc.start:newpos]   
if 0x81<=ch1<=0xFE and (0x40<=ch2<=0x7E or 0x7E<=ch2<=0xFE): # GBK   
return (unicode(sk,'cp936'), newpos)   
if 0x81<=ch1<=0xFE and (0x40<=ch2<=0x7E or 0xA1<=ch2<=0xFE): # BIG5   
return (unicode(sk,'big5'), newpos)   
raise TypeError('unknown codec !')   
codecs.register\_error("cjk\_replace", cjk\_replace)   
我们的cjk\_replace现在只能处理GBK与BIG5的，因为我对编码也不是特别了解，只是大概知道GBK与BIG5的，不太了解JP的。在 cjk\_replace这个函数里，我们对不认识的文字进行手工识别，如果认识的编码，就用正确的方法，并返回编码后的内容与新的pos，比如“1234中文”，在pos为4的时候，会调用我们的cjk\_replace，我们会返回一个从gbk转换成utf8的“中”字，并返回下个正确的位置“文”的起始位置。当然了，处理“文”的时候，还会再调用一次。下面看看是如何使用的：   
filedata = open('test.txt','r).read() #gbk and big5 file   
data = unicode(filedata,'ascii','cjk\_replace').encode('utf8')   
小结   
一个比较一般的Python中文处理的流程：   
\* 将欲处理的字符串用unicode函数以正确的编码转换为Unicode   
\* 在程序中统一用Unicode字符串进行操作   
\* 输出时，使用encode方法，将Unicode再转换为所需的编码   
有几点要说明一下：   
\* 所谓“正确的”编码，指得是指定编码和字符串本身的编码必须一致。这个其实并不那么容易判断，一般来说，我们直接输入的简体中文字符，有两种可能的编码：GB2312（GBK、GB18030）、以及UTF-8   
\* encode成本地编码的时候，必须要保证目标编码中存在欲转换字符的内码。encode这种操作一般是通过一个本地编码对应Unicode的编码转换表来进行的，事实上每个本地编码只能映射到Unicode的一部分。但是映射的区域是不同的，比如Big-5对应的Unicode的编码范围和 GBK对应的就不一样（实际上这两个编码有部分范围是重叠的）。所以，Unicode的一些字符（比如本身就是从GB2312转换来的那些），可以映射到 GBK，但未必可以映射到Big-5，如果你想转换到Big-5，很有可能就会出现编码找不到的异常。但UTF-8的码表范围实际上和Unicode是一样的（只是编码形式不同而已），所以，理论上来说，任何本地编码的字符，都可以被转换到UTF-8   
\* GB2312、GBK、GB18030本质上是同一种编码标准。只是在前者的基础上扩充了字符数量   
\* UTF-8和GB编码不兼容

\* 出现编解码异常时可能需要自己编写编解码解析函数，这需要了解一些字符编码的知识

##### ****八、最后附一些关于 python 编码问题的总结：****

1、写的代码模块需要指定编码  
   如果代码没有指定coding,python就默认所有的字符为ASCII码,  
   ASCII码只支持256个字符,ASCII码不支持中文,所以就报错。  
   所以要在代码前写上#coding:utf-8或#coding:gbk  
   建议你的代码统一写成这样：

[?](http://my.oschina.net/leejun2005/blog/74430)

|  |  |
| --- | --- |
| 1  2  3  4 | #coding:utf-8  import sys  reload(sys)  sys.setdefaultencoding('utf-8') |

2、python2内部所有编码统一为unicode  
  unicode可以处理世界上所有语言的字符。  
  utf-8为unicode的一种实现形式，所以需要在代码前写上#coding:utf-8  
  
3、编码转换  
   牢记python2内部编码为unicode.  
   其它的编码decode()为unicode,再编码encode()为你指定的编码,就不会出现乱码。  
  
4、网页采集时  
  代码指定#coding:utf-8  
  如果网页的编码为gbk  
  需要这样处理：  
  html = html.decode('gbk').encode('utf-8')  
  
5、代码前也可以写#coding:gbk,但也要保证你的代码文件的保存格式为gbk.这个在windos下会出现这样的问题。  
  
6、字典等key或值的汉字问题  
#coding:utf-8  
dict1 ={1:'python周末培训班',2:'咨询010-68165761 QQ：1465376564'}  
  
print dict1   
# 这样输出的没有显示汉字，是显示汉字的其它编码  
  
dict2 ={1:'python视频培训班',2:'咨询010-68165761 QQ：1465376564'}  
for key in dict2:  
    print dict2[key]  
  
7、unicode的汉字编码写到文本文件中  
  需要根据文本文件的编码进行转换  
  可以encode('utf-8')或encode('gbk')  
  
总结：凡是报错信息中出现的错误包含“ASCII”，就是没有指定汉字编码的问题。

##### 九、参考资料

1、  <http://bbs3.chinaunix.net/thread-1389703-1-2.html>2、 Python的中文处理及其它   
<http://www.go4pro.org/?p=38>3、 Python处理中文的时候的一些小技巧   
<http://cocre.com/?p=461>4、 Unicode In Python, Completely Demystified. Kumar McMillan   
<http://farmdev.com/talks/unicode>5、 python中文处理好方法   
<http://www.okpython.com/bbs/viewthread.php?tid=311>6、 Python的中文处理   
<http://hi.baidu.com/mrsz/blog/item/7812a5018c2cf2031d9583d2.html>7、 UnicodeDecodeError   
<http://wiki.python.org/moin/UnicodeDecodeError>8、 UnicodeEncodeError   
<http://wiki.python.org/moin/UnicodeEncodeError>9、 如何处理python编码转换时的UnicodeDecodeError异常   
<http://blog.chinaunix.net/u/8873/showart_1009737.html>10、codecs — Codec registry and base classes

<http://docs.python.org/library/codecs.html>

11、python 中文乱码 问题深入分析

[http://blog.csdn.net/kiki113/article/details/4062063](http://blog.csdn.net/kiki113/article/details/4062063" \t "_blank)

12、python新手必碰到的问题---encode与decode，中文乱码[转]

[http://www.51testing.com/?uid-524463-action-viewspace-itemid-817888](http://www.51testing.com/?uid-524463-action-viewspace-itemid-817888" \t "_blank)

13、Python 编码和 Unicode（深入理解 python 编码的文章，推荐阅读）

[http://blog.jobbole.com/50345/](http://blog.jobbole.com/50345/" \t "_blank)

14、关于python2中的一个编码问题（u'\xe5\xb0\x8f\xe4\xb8\x89' 与 '\xe5\xb0\x8f\xe4\xb8\x89' 的区别）

[http://segmentfault.com/q/1010000000370030](http://segmentfault.com/q/1010000000370030" \t "_blank)

15、Python的中文编码问题

[http://python.jobbole.com/80831/](http://python.jobbole.com/80831/" \t "_blank)

16、在Python中正确使用Unicode

[http://python.jobbole.com/80939/](http://python.jobbole.com/80939/" \t "_blank)

## 字符串处理

### 简介

Python默认编码是 ASCII码

所以中文用的时候要加utf-8

### Len计算字符串大小

输出变量长度： print(a)

### 转义符让文本更好处理

要转义的字符前面加个反斜杠 \

‘ fdsfds\’\’’

‘ \n ’

### 字符串前面跟着的小尾巴到底是什么东西

中文：print u’张硕’

不要转义 print r’\n’ ：直接输出 \n 不变为回车

### Count()计数

>>> c

'4234232sf23'

>>> c.count('4')

2

### 字符串替换拼接

#### %s格式化 占位符(%s字符串，%d 数字)

Print “wo shi hao %s haha”%”ren”

>>>wo shi hao ren haha

Print “wo %s hao %s haha”%(“shi”,”ren”)

>>>wo shi hao ren haha

a = "zhang %s shuo %s" % ("is" , "good")

print a

**常见的占位符有：**

|  |  |
| --- | --- |
| %d | 整数 |
| %f | 浮点数 |
| %s | 字符串 |
| %x | 十六进制整数 |

其中，格式化整数和浮点数还可以指定是否补0和整数与小数的位数：

>>> '%2d-%02d' % (3, 1)

' 3-01'

>>> '%.2f' % 3.1415926

'3.14'

如果你不太确定应该用什么，%s永远起作用，它会把任何数据类型转换为字符串：

>>> 'Age: %s. Gender: %s' % (25, True)

'Age: 25. Gender: True'

对于Unicode字符串，用法完全一样，但最好确保替换的字符串也是Unicode字符串：

>>> u'Hi, %s' % u'Michael'

u'Hi, Michael'

有些时候，字符串里面的%是一个普通字符怎么办？这个时候就需要转义，用%%来表示一个%：

>>> 'growth rate: %d %%' % 7

'growth rate: 7 %'

#### Format 好方法 重点推荐

b = "wo ai {name} love {fruit} ".format(fruit = "apple", name = "stevn")

print b

#### dict字典 替换

c = "zhang %(whose)s shuo %(fruit)s" % {'whose': 'my', 'fruit': 'orenge'}

print c

zhang is shuo good

wo ai stevn love apple

zhang my shuo orenge

#### join([])优秀的拼接方法

‘’.join([a,b])

>>> "\*\*\*".join([a,b])

'wo de ming shuo\*\*\*wo de wo de ming shuo hh'

### 字符串空格操作

我们先创建一个左右都有N个空格的字符串变量s，已知：

>>> s = “   张 硕    ”

去除字符串空格，在Python里面有它的内置方法，不需要我们自己去造轮子了。  
**lstrip：删除左边的空格**  
这个字符串方法，会删除字符串s开始位置前的空格。

>>> s.lstrip()  
'张 硕   '

**rstrip：删除右连的空格**  
这个内置方法可以删除字符串末尾的所有空格，看下面演示代码：

>>> s.rstrip()  
'    张 硕'

**strip：删除两端的空格**  
有的时候我们读取文件中的内容，每行2边都有空格，能不能一次性全部去掉呢，字符符有一个内置的strip()方法可以做到。

>>> s = “   张 硕    ”  
>>> s.strip()  
'张 硕'

### 对字符串中大小写转换

print 'just to test it'.upper() #所有字母都转换成大写

JUST TO TEST IT

print 'JUST TO TEST IT'.lower() #所有字母都转换成小写

just to test it

2、对字符串中的字符(仅对字母有效)部分大小写转换:

print 'JUST TO TEST IT'.capitalize() #字符串的首字母转换成大写， 其余转换成小写

Just to test it

print 'JUST TO TEST IT'.title() #字符串中所有单词的首字母转换成大写， 其余转换成小写

Just To Test It

### 判断字符串大小写函数

判断大小写函数str.lower() 和 str.upper()

print 'JUST TO TEST IT'.isupper()

True

print 'JUST TO TEST IT'.islower()

False

print 'JUST TO TEST IT'.istitle()

False

### 字符串大小写互换

>>> da

'aAsmr3idd4bgs7Dlsf9eAF'

>>> da.swapcase()

'AaSMR3IDD4BGS7dLSF9Eaf'

### 判断变量类型,长度

type(变量)

字符串长度len()

### isdigit()判断字符串是否为纯数字

>>> a

'4234232'

>>> a.isdigit()

True

>>> b

'sf23'

>>> b.isdigit()

False

### string的值修改

（导致“TypeError: 'str' object does not support item assignment”）   
string是一种不可变的数据类型，该错误发生在如下代码中：

spam = 'I have a pet cat.'   
spam[13] = 'r'   
print(spam)

而你实际想要这样做：

spam = 'I have a pet cat.'   
spam = spam[:13] + 'r' + spam[14:]   
print(spam)

### split() 分隔器

split()的用法是这样的，你只要在括号里面打上你需要以什么分割就可以了，比如你要以#分割，那就split("#")。例如a = "a#a#a", b = a.split("#") >>> print b 就输出[a,a,a]

>>> c

'my name is jay,I love python'

b = c.split();

>>> b

['my', 'name', 'is', 'jay,I', 'love', 'python']

>>> b = c.split(',');

>>> b

['my name is jay', 'I love python']

>>>

### 替换1 replace

描述 替换是有顺序的替换

Python replace() 方法把字符串中的 old（旧字符串） 替换成 new(新字符串)，如果指定第三个参数max，则替换不超过 max 次。

语法

replace()方法语法：

str.replace(old, new[, max])

参数

old -- 将被替换的子字符串。

new -- 新字符串，用于替换old子字符串。

max -- 可选字符串, 替换不超过 max 次

返回值

返回字符串中的 old（旧字符串） 替换成 new(新字符串)后生成的新字符串，如果指定第三个参数max，则替换不超过 max 次。

实例

以下实例展示了replace()函数的使用方法：

#!/usr/bin/python

str = "this is string example....wow!!! this is really string";

print str.replace("is", "was");

print str.replace("is", "was", 3);

以上实例输出结果如下：

thwas was string example....wow!!! thwas was really string

thwas was string example....wow!!! thwas is really string

### 替换 2 翻译表

首先引入string 然后定义翻译表，最后引用，它和replace的区别是可以无视顺序替换，上代码：

翻译表是逐字替换删除

>>> import string

>>> g = string.maketrans('abc','123') 定义翻译表

>>> k = 'fjdsklafjdsklajfajdjabcabcabc'

>>> k

'fjdsklafjdsklajfajdjabcabcabc'

>>> print k.translate(g) 翻译abc为123

fjdskl1fjdskl1jf1jdj123123123

>>> print k.translate(g,'b') 删除原字符串中的b并替换ac为12

fjdskl1fjdskl1jf1jdj131313

>>> g = string.maketrans('','') 定义空翻译表

>>> k

'fjdsklafjdsklajfajdjabcabcabc'

>>> k.translate(g,'ab') 直接删除字符串中的ab

'fjdsklfjdskljfjdjccc'

### find()方法 字符串寻找

描述

Python find() 方法检测字符串中是否包含子字符串 str ，如果指定 beg（开始） 和 end（结束） 范围，则检查是否包含在指定范围内，如果包含子字符串返回开始的索引值，否则返回-1。

语法

find()方法语法：

str.find(str, beg=0, end=len(string))

参数

str -- 指定检索的字符串

beg -- 开始索引，默认为0。

end -- 结束索引，默认为字符串的长度。

返回值

如果包含子字符串返回开始的索引值，否则返回-1。

实例

以下实例展示了find()方法的实例：

#!/usr/bin/python

str1 = "this is string example....wow!!!";

str2 = "exam";

print str1.find(str2);

print str1.find(str2, 10);

print str1.find(str2, 40);

以上实例输出结果如下：

15

15

-1

>>> c

'my name is jay,I love python'

>>> c.find("n")

3

>>> c.find("n",18)

27

### Index()方法 也是查找

区别就是在找不到的时候一个是 -1，另外一个是报错。

描述

Python index() 方法检测字符串中是否包含子字符串 str ，如果指定 beg（开始） 和 end（结束） 范围，则检查是否包含在指定范围内，该方法与 python find()方法一样，只不过如果str不在 string中会报一个异常。

语法

index()方法语法：

str.index(str, beg=0, end=len(string))

参数

str -- 指定检索的字符串

beg -- 开始索引，默认为0。

end -- 结束索引，默认为字符串的长度。

返回值

如果包含子字符串返回开始的索引值，否则抛出异常。

实例

以下实例展示了index()方法的实例：

#!/usr/bin/python

str1 = "this is string example....wow!!!";

str2 = "exam";

print str1.index(str2);

print str1.index(str2, 10);

print str1.index(str2, 40);

以上实例输出结果如下：

15

15

Traceback (most recent call last):

File "test.py", line 8, in

print str1.index(str2, 40);

ValueError: substring not found

shell returned 1

**注意：**在接下来的几个章节中，我们会详细介绍Python Exception的使用。

### sys.stdout输出重定向

sys.stdout 与 print

当我们在 Python 中打印对象调用 print obj 时候，事实上是调用了 sys.stdout.write(obj+'\n')

print 将你需要的内容打印到了控制台，然后追加了一个换行符

print 会调用 sys.stdout 的 write 方法

以下两行在事实上等价：

sys.stdout.write('hello'+'\n')

print 'hello'

sys.stdin 与 raw\_input

当我们用 raw\_input('Input promption: ') 时，事实上是先把提示信息输出，然后捕获输入

以下两组在事实上等价：

hi=raw\_input('hello? ')

print 'hello? ', #comma to stay in the same line

hi=sys.stdin.readline()[:-1] # -1 to discard the '\n' in input stream

从控制台重定向到文件

原始的 sys.stdout 指向控制台

如果把文件的对象的引用赋给 sys.stdout，那么 print 调用的就是文件对象的 write 方法

f\_handler=open('out.log', 'w')

sys.stdout=f\_handler

print 'hello'

# this hello can't be viewed on concole

# this hello is in file out.log

记住，如果你还想在控制台打印一些东西的话，最好先将原始的控制台对象引用保存下来，向文件中打印之后再恢复 sys.stdout



\_\_console\_\_=sys.stdout

# redirection start

# ...

# redirection end

sys.stdout=\_\_console\_\_

### raw字符串与多行字符串

字符转义 例如：

r'\(~\_~)/ \(~\_~)/'

如果要表示多行字符串，可以用'''...'''表示：

'''Line 1

Line 2

Line 3'''

r'''Python is created by "Guido".

It is free and easy to learn.

Let's start learn Python in imooc!'''

## 对字符串切片

字符串 'xxx'和 Unicode字符串 u'xxx'也可以看成是一种list，每个元素就是一个字符。因此，字符串也可以用切片操作，只是操作结果仍是字符串：

>>> 'ABCDEFG'[:3]

'ABC'

>>> 'ABCDEFG'[-3:]

'EFG'

>>> 'ABCDEFG'[::2]

'ACEG'

在很多编程语言中，针对字符串提供了很多各种截取函数，其实目的就是对字符串切片。Python没有针对字符串的截取函数，只需要切片一个操作就可以完成，非常简单。

## 布尔类型

计算的时候：从左到右，从先到后。

我们已经了解了Python支持布尔类型的数据，布尔类型只有True和False两种值，但是布尔类型有以下几种运算：

**与运算**：只有两个布尔值都为 True 时，计算结果才为 True。

True and True # ==> True

True and False # ==> False

False and True # ==> False

False and False # ==> False

**或运算**：只要有一个布尔值为 True，计算结果就是 True。

True or True # ==> True

True or False # ==> True

False or True # ==> True

False or False # ==> False

**非运算**：把True变为False，或者把False变为True：

not True # ==> False

not False # ==> True

布尔运算在计算机中用来做条件判断，根据计算结果为True或者False，计算机可以自动执行不同的后续代码。

在Python中，布尔类型还可以与其他数据类型做 and、or和not运算，请看下面的代码：

a = True

print a and 'a=T' or 'a=F'

计算结果不是布尔类型，而是字符串 'a=T'，这是为什么呢？

因为Python把0、空字符串''和None看成 False，其他数值和非空字符串都看成 True，所以：

True and 'a=T' 计算结果是 'a=T'

继续计算 'a=T' or 'a=F' 计算结果还是 'a=T'

### ****短路计算 惰性求值****

**要解释上述结果，又涉及到 and 和 or 运算的一条重要法则：****短路计算。**

1. 在计算 a and b 时，如果 a 是 False，则根据与运算法则，整个结果必定为 False，因此返回 a；如果 a 是 True，则整个计算结果必定取决与 b，因此返回 b。

2. 在计算 a or b 时，如果 a 是 True，则根据或运算法则，整个计算结果必定为 True，因此返回 a；如果 a 是 False，则整个计算结果必定取决于 b，因此返回 b。

所以Python解释器在做布尔运算时，只要能提前确定计算结果，它就不会往后算了，直接返回结果。

# 文本文件

## **[open/文件操作](http://www.cnblogs.com/dkblog/archive/2011/02/24/1980651.html)** 基本读写添加操作

f=open('/tmp/hello','w')  
  
#open(路径+文件名,读写模式)  
  
#读写模式:r只读,r+读写,w新建(会覆盖原有文件),a追加,b二进制文件.常用模式  
  
如:'rb','wb','r+b'等等

读写模式的类型有：

rU 或 Ua 以读方式打开, 同时提供通用换行符支持 (PEP 278)  
w     以写方式打开，  
a     以追加模式打开 (从 EOF 开始, 必要时创建新文件)  
r+     以读写模式打开  
w+     以读写模式打开 (参见 w )  
a+     以读写模式打开 (参见 a )  
rb     以二进制读模式打开  
wb     以二进制写模式打开 (参见 w )  
ab     以二进制追加模式打开 (参见 a )  
rb+    以二进制读写模式打开 (参见 r+ )  
wb+    以二进制读写模式打开 (参见 w+ )  
ab+    以二进制读写模式打开 (参见 a+ )

注意：

1、使用'W'，文件若存在，首先要清空，然后（重新）创建，

2、使用'a'模式 ，把所有要写入文件的数据都追加到文件的末尾，即使你使用了seek（）指向文件的其他地方，如果文件不存在，将自动被创建。

f.read([size]) size未指定则返回整个文件,如果文件大小>2倍内存则有问题.f.read()读到文件尾时返回""(空字串)  
  
file.readline() 返回一行  
  
file.readline([size]) 返回包含size行的列表,size 未指定则返回全部行  
  
for line in f: print line #通过迭代器访问  
  
f.write("hello\n") #如果要写入字符串以外的数据,先将他转换为字符串.  
  
f.tell() 返回一个整数,表示当前文件指针的位置(就是到文件头的比特数).  
  
f.seek(偏移量,[起始位置])  
  
用来移动文件指针  
  
偏移量:单位:比特,可正可负  
  
起始位置:0-文件头,默认值;1-当前位置;2-文件尾  
  
f.close() 关闭文件  
  
Code:

|  |
| --- |
| #!/usr/bin/env python # Filename: using\_file.py  poem='''\Programming is funWhen the work is doneif you wanna make your work also fun: use Python!''' f=file('poem.txt','w') # open for 'w'riting f.write(poem) # write text to file f.close() # close the file f=file('poem.txt')  # if no mode is specified, 'r'ead mode is assumed by default while True:  line=f.readline()  if len(line)==0: # Zero length indicates EOF  break  print line,  # Notice comma to avoid automatic newline added by Python f.close()  # close the file |

w write 写

r read 读取

a append 添加

写和读的过程

d = open(‘a.txt’,’w’)

d.write(‘my name is zhang’)

d.close()

d = open(‘a.txt’,’r’)

d.read(5)

>>> d = open('a.txt','w')

>>> d.write('hi./nsecond hi.')

>>> d.close()

>>> d = open('a.txt','r')

>>> d.readline()

'hi./nsecond hi.'

>>> d.readline()

''

>>> d.readline()

''

>>> print d.readline()

>>> d.seek(0)

>>> print d.read(100)

hi./nsecond hi.

>>> d.close()

### Linecatch

读取文件的第三行

print linecache.getline('a.txt',3)

注意：这个是把文件的每一行放到一个list里面组成一个list，方法是getline**s()**

>>> line = linecache.getlines("a.txt")

>>> print line

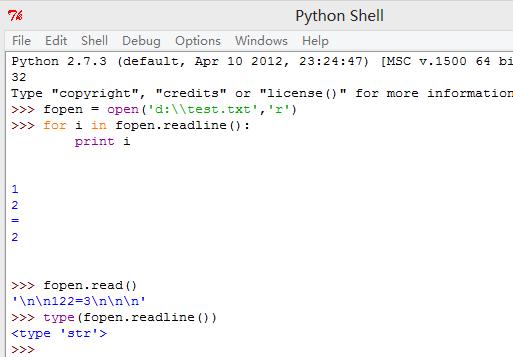
['wo\n', 'shi\n', 'hang\n', 'fdskf\n']

## [python中read() readline()以及readlines()区别 .](http://blog.csdn.net/zhongyhc/article/details/9026917)

**.read() 每次读取整个文件，它通常将读取到底文件内容放到一个字符串变量中，也就是说 .read() 生成文件内容是一个字符串类型，如下图；**

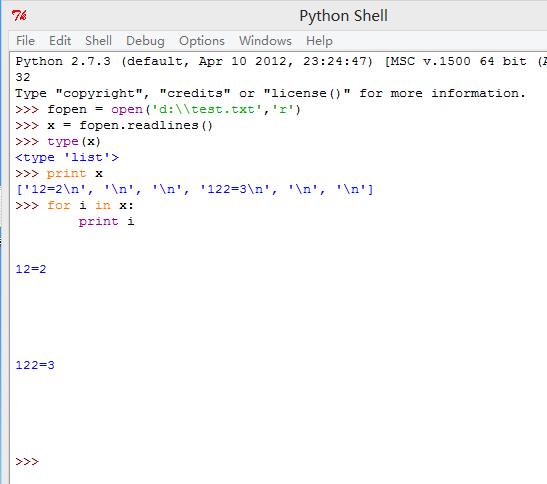


**.readline()每只读取文件的一行，通常也是读取到的一行内容放到一个字符串变量中，返回str类型，如下图；**



从for循环中我们可以看出.readline()只读取了文件中的一行内容。

**.readlines()每次按行读取整个文件内容，将读取到的内容放到一个列表中，返回list类型，如下图。**



## Decode：转码

decode() 方法转换中文乱码用的

str.decode(encoding='UTF-8',errors='strict')

**实例**

以下实例展示了decode()方法的实例：

#!/usr/bin/python

str = "this is string example....wow!!!";

str = str.encode('base64','strict');

print "Encoded String: " + str;

print "Decoded String: " + str.decode('base64','strict')

以上实例输出结果如下：

Encoded String: dGhpcyBpcyBzdHJpbmcgZXhhbXBsZS4uLi53b3chISE=

Decoded String: this is string example....wow!!!

## 文本中的print使用方法

两种输出的区别 “，”

test#coding=utf-8

print 2

print 4

print 2,

print 4,

print 6

输出

2

4

2 4 6

**运用print和重定向>>向文本中写入数据**

**f = open('f.txt','w')**

**print >> f, 'zhangshuo',**

**print >> f, 'wo ai ni'**

**f.close()**

## with打开文件添加

**原写法**

**f = open('a.txt', 'w')**

**f.write('zhagnshuo\nwoainimen')**

**f.close()**

**改变后写法，不用close（）了，省事，这个a是append追加的意思**

**with open('a.txt', 'a') as f:**

**f.write('\nhahahahh')**

## [linecache模块缓存读取大文件指定行](http://www.iplaypython.com/module/linecache.html)

**linecache模块**的作用是将[文件](http://www.iplaypython.com/text-file/" \t "_blank)内容读取到内存中，进行缓存，而不是每次都要从硬盘中读取，这样效率提高很多，又省去了对硬盘IO控制器的频繁操作。

**一、linecache模块简介**

我们看一下这个模块的名字叫做linecache，行-缓存，这对于读取内容非常多的文件，效果甚好，而且它还可以读取指定的行内容。

**二、linecache模块函数讲解**

1 )、 linecache.getline(filename, lineno[, module\_globals]) ，这个方法从filename也就是文件中读取内容，得到第 lineno行，注意没有去掉换行符，它将包含在行内。  
2 )、 linecache.clearcache() ，清除现有的文件缓存。  
3 )、 linecache.checkcache([filename]) ，参数是文件名，作用是检查缓存内容的有效性，可能硬盘内容发生了变化，更新了，如果不提供参数，将检查缓存中所有的项。

**三 、linecache模块源码演示**

# coding=utf-8

import [os](http://www.iplaypython.com/module/os.html" \t "_blank)  
import linecache

[def](http://www.iplaypython.com/jichu/function.html" \t "_blank) get\_content(path):  
    '''读取缓存中文件内容，以字符串形式返回'''  
    [if](http://www.iplaypython.com/jinjie/if.html" \t "_blank) os.path.exists(path):  
        content = ''  
        cache\_data = linecache.getlines(path)  
        [for](http://www.iplaypython.com/jinjie/for.html" \t "_blank) line in range(len(cache\_data)):  
            content += cache\_data[line]  
        return content  
    [else](http://www.iplaypython.com/jinjie/else-elif.html" \t "_blank):  
        [print](http://www.iplaypython.com/jichu/print.html" \t "_blank)('the path [{}] is not exist!'.format(path))

def main():  
    path = 'c:\\test.txt'  
    content = get\_content(path)  
    print(content)

if \_\_name\_\_ == '\_\_main\_\_':  
    main()

**四 、linecache模块注意事项**

linecache里面最常用到的就是getline方法，简单实用可以直接从内容中读到指定的行，日常编程中如果涉及读取大文件，一定要使用首选linecache模块，相比[open()](http://www.iplaypython.com/text-file/open.html" \t "_blank)那种方法要快N倍，它是你读取文件的效率之源。

**[python](http://www.jbxue.com/jb/python/" \t "_blank) linecache模块读取文件**

在python中，有个好用的模块linecache，该模块允许从任何文件里得到任何的行，并且使用缓存进行优化，常见的情况是从单个文件读取多行。

**linecache.getlines(filename)**从名为filename的文件中得到全部内容，输出为列表格式，以文件每行为列表中的一个元素,并以linenum-1为元素在列表中的位置存储

**linecache.getline(filename,lineno)**从名为filename的文件中得到第lineno行。这个函数从不会抛出一个异常–产生错误时它将返回”（换行符将包含在找到的行里）。  
如果文件没有找到，这个函数将会在sys.path搜索。

**linecache.clearcache()**清除缓存。如果你不再需要先前从getline()中得到的行

**linecache.checkcache(filename)**检查缓存的有效性。如果在缓存中的文件在硬盘上发生了变化，并且你需要更新版本，使用这个函数。如果省略filename，将检查缓存里的所有条目。

**linecache.updatecache(filename)**更新文件名为filename的缓存。如果filename文件更新了，使用这个函数可以更新linecache.getlines(filename)返回的列表。

代码示例:

# cat a.txt  
1a  
2b  
3c  
4d  
5e  
6f  
7g

1、获取a.txt文件的内容

代码示例:

>>> a=linecache.getlines('a.txt')  
>>> a  
['1a\n', '2b\n', '3c\n', '4d\n', '5e\n', '6f\n', '7g\n']

2、获取a.txt文件中第1-4行的内容

代码示例:

>>> a=linecache.getlines('a.txt')[0:4]  
>>> a  
['1a\n', '2b\n', '3c\n', '4d\n']

3、获取a.txt文件中第4行的内容

代码示例:

>>> a=linecache.getline('a.txt',4)  
>>> a  
'4d\n'

注意：使用linecache.getlines('a.txt')打开文件的内容之后，如果a.txt文件发生了改变，如果要再次用linecache.getlines获取的内容，不是文件的最新内容，还是之前的内容，此时有两种方法：  
1、使用linecache.checkcache(filename)来更新文件在硬盘上的缓存，然后在执行linecache.getlines('a.txt')就可以获取到a.txt的最新内容；  
2、直接使用linecache.updatecache('a.txt')，即可获取最新的a.txt的罪行内容

另外：  
1）、读取文件之后，不需要使用文件的缓存时，需要在最后清理一下缓存，使linecache.clearcache()清理缓存，释放缓存。  
2）、此模块使用内存来缓存文件内容，所以需要耗费内存，打开文件的大小和打开速度和你的内存大小有关系。

# List 索引

L = ['Adam', 'Lisa', 'Bart']

* >>> print L[0]
* Adam
* >>> print L[-1]
* Bart
* 使用索引时，**千万注意不要越界**。

## 内部都是指针指向空间

del a 删除列表对象的引用

del a[:] 清空列表对象里的元素

>>> a = ['wo','shi','zhangshuo']

>>> a

['wo', 'shi', 'zhangshuo']

>>> b

[9, 10, 11, 12]

>>> b=a

>>> a[2]='meiren'

>>> a

['wo', 'shi', 'meiren']

>>> b

['wo', 'shi', 'meiren']

>>> id(a)

139909013670152

>>> id(b)

139909013670152

>>> del a[:]

>>> a

[]

>>> b

## 添加

### Extend:

接收参数，并将参数的每个元素添加到原列表的末尾，不改变原参数的指针

**>>> a**

**[1, 2, 3, 4, 5, 6, 7, 8]**

**>>> b**

**[3, 4, 5, 6]**

**>>> c**

**[8, 6]**

**>>> d**

**[5, 7]**

**>>> a**

**[1, 2, 3, 4, 5, 6, 7, 8]**

**>>> b=[9,10,11,12]**

**>>> a**

**[1, 2, 3, 4, 5, 6, 7, 8]**

**>>> b**

**[9, 10, 11, 12]**

**>>> c=a+b**

**>>> c**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]**

**>>> id(a)**

**139909013670152**

**>>> id(b)**

**139909013669936**

**>>> a.extend(c)**

**>>> a**

**[1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]**

**>>> id(a)**

**139909013670152**

### append

 list 的 append() 方法，把新同学追加到 list 的末尾：

>>> L = ['Adam', 'Lisa', 'Bart']

>>> L.append('Paul')

>>> print L

['Adam', 'Lisa', 'Bart', 'Paul']

### Insert

 insert()方法，它接受两个参数，第一个参数是索引号，第二个参数是待添加的新元素：

>>> L = ['Adam', 'Lisa', 'Bart']

>>> L.insert(0, 'Paul')

>>> print L

['Paul', 'Adam', 'Lisa', 'Bart']

**L.insert(0, 'Paul')** 的意思是，'Paul'将被添加到索引为 0 的位置上（也就是第一个），而原来索引为 0 的Adam同学，以及后面的所有同学，都自动向后移动一位。

## 删除

### **Del直接删除**

**>>> a**

**[1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]**

**>>> del a[0]**

**>>> a**

**[2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]**

### **pop( )**

方法总是删掉list的最后一个元素，并且它还返回这个元素，所以我们执行 L.pop() 后，会打印出 'Paul'。

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> L.pop()

'Paul'

>>> print L

['Adam', 'Lisa', 'Bart']

Paul的索引是2，因此，用 pop(2)把Paul删掉：

>>> L.pop(2)

'Paul'

>>> print L

['Adam', 'Lisa', 'Bart']

### Remove

移除选中元素，没有的话报错

>>> a

[1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

>>> a.remove(11)

>>> a

[1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13]

>>> a.remove(101)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: list.remove(x): x not in list

>>>

## 替换

用Paul把Bart给替换掉：

>>> L[2] = 'Paul'

>>> print L

L = ['Adam', 'Lisa', 'Paul']

由于Bart还可以用 -1 做索引，因此，下面的代码也可以完成同样的替换工作：

>>> L[-1] = 'Paul'

## 是否在列表中 in not in

>>> a

[1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13]

>>> 5 in a

True

>>> 'a' not in a

True

>>> 101 in a

False

## 排序sort（）和翻转reverse（）

>>> f

[99, 97, 95, 93, 91, 89, 87, 85, 83, 81, 79, 77, 75, 73, 71, 69, 67, 65, 63, 61, 59, 57, 55, 53, 51, 49, 47, 45, 43, 41, 39, 37, 35, 33, 31, 29, 27, 25, 23, 21, 19, 17, 15, 13, 11, 9, 7, 5, 3, 1]

>>> f.sort()

>>> f

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]

>>> w = f.reverse()

>>> w

>>> f

[99, 97, 95, 93, 91, 89, 87, 85, 83, 81, 79, 77, 75, 73, 71, 69, 67, 65, 63, 61, 59, 57, 55, 53, 51, 49, 47, 45, 43, 41, 39, 37, 35, 33, 31, 29, 27, 25, 23, 21, 19, 17, 15, 13, 11, 9, 7, 5, 3, 1]

## xrange的用法：

xrange(开始，结束，步长)

xrange 它生成一个xrange对象。

比如我们

a = xrange(1,10)

print type(a)

print a[0]

比较

range: 直接生成一个列表对象。

xrange: 它是生成一个xrange对象.

xrange的用法：

1当我们需要操作一个非常大的数据，而且内存比较吃紧的时候，我们可以用xrange来操作省内存。

2xrange一般用在循环里面，比如我们只需要操作部分数据的话，而不是返回全部元素来完成操作，推荐用xrange,效率更高。

比如：

for m in range(1000)：

if m == 10:

print 'sss'

break

for m in xrange(1000):

if m == 10:

print 'sss'

break

## 切片：对list进行切片

取一个list的部分元素是非常常见的操作。比如，一个list如下：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

取前3个元素，应该怎么做？

笨办法：

>>> [L[0], L[1], L[2]]

['Adam', 'Lisa', 'Bart']

之所以是笨办法是因为扩展一下，取前N个元素就没辙了。

取前N个元素，也就是索引为0-(N-1)的元素，可以用循环：

>>> r = []

>>> n = 3

>>> for i in range(n):

... r.append(L[i])

...

>>> r

['Adam', 'Lisa', 'Bart']

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

对应上面的问题，取前3个元素，用一行代码就可以完成切片：

>>> L[0:3]

['Adam', 'Lisa', 'Bart']

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

>>> L[:3]

['Adam', 'Lisa', 'Bart']

也可以从索引1开始，取出2个元素出来：

>>> L[1:3]

['Adam', 'Lisa']

只用一个**:**，表示从头到尾：

>>> L[:]

['Adam', 'Lisa', 'Bart', 'Paul']

因此，L[:]实际上复制出了一个新list。

切片操作还可以指定第三个参数：

>>> L[::2]

['Adam', 'Bart']

第三个参数表示每N个取一个，上面的 L[::2] 会每两个元素取出一个来，也就是隔一个取一个。

把list换成tuple，切片操作完全相同，只是切片的结果也变成了tuple。

## 倒序切片

对于list，既然Python支持L[-1]取倒数第一个元素，那么它同样支持倒数切片，试试：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> L[-2:]

['Bart', 'Paul']

>>> L[:-2]

['Adam', 'Lisa']

>>> L[-3:-1]

['Lisa', 'Bart']

>>> L[-4:-1:2]

['Adam', 'Bart']

记住倒数第一个元素的索引是-1。倒序切片包含起始索引，不包含结束索引。

倒序切片，倒着来：

>>> b

[1, 2, 3, 4, 5, 6]

>>> b[-2:-4:-1]

[5, 4]

## list转变成dict

>>> a

[(0, 0), (4, 0), (0, 2), (12, 2)]

>>> print dict(a)

{0: 2, 12: 2, 4: 0}

## List 变为 str

>>> a

[1, 2, 5, 54, 7, 9, 435, 678]

>>> b = str(a)

>>> b

'[1, 2, 5, 54, 7, 9, 435, 678]'

>>> c = b[1:-1].replace(', ','')

>>> c

'1255479435678'

## index()从list中找出某个值第一个匹配项的索引位置。

语法

index()方法语法：

list.index(obj)

参数

obj -- 查找的对象。

返回值

该方法返回查找对象的索引位置，如果没有找到对象则抛出异常。

实例

以下实例展示了 index()函数的使用方法：

#!/usr/bin/python

aList = [123, 'xyz', 'zara', 'abc'];

print "Index for xyz : ", aList.index( 'xyz' ) ;

print "Index for zara : ", aList.index( 'zara' ) ;

以上实例输出结果如下：

Index for xyz : 1

Index for zara : 2

## 排序使用技巧

List内置方法本身修改，而外部的sorted呢，需要赋值，本身不变。

### 本身不变赋值的：sorted

>>> c

[43, 435, 53, 43, 54, 5456, 3455, 342.45, 345454, 5, 5434, 34, 33, 43]

>>> d = sorted(c)

>>> d

[5, 33, 34, 43, 43, 43, 53, 54, 342.45, 435, 3455, 5434, 5456, 345454]

>>> c

[43, 435, 53, 43, 54, 5456, 3455, 342.45, 345454, 5, 5434, 34, 33, 43]

### 本身改变的函数sort（数据对象内部使用）

>>> c

[43, 435, 53, 43, 54, 5456, 3455, 342.45, 345454, 5, 5434, 34, 33, 43]

>>> c.sort()

>>> c

[5, 33, 34, 43, 43, 43, 53, 54, 342.45, 435, 3455, 5434, 5456, 345454]

>>> c.sort(reverse=True)

>>> c

[345454, 5456, 5434, 3455, 435, 342.45, 54, 53, 43, 43, 43, 34, 33, 5]

### Key

#### 字符串 int 排序

>>> a

['32', '325454', '333', '34211']

>>> a.sort(key=int)

>>> a

['32', '333', '34211', '325454']

>>> a.sort(key=int,reverse=True)

>>> a

['325454', '34211', '333', '32']

#### 按分项排序【（32,2），（2323,1）】第二个游标排序

>>> d

[('a', 34), ('dfds', 4), ('fdsfds', 9), ('zhang', 5)]

>>> d.sort(key = lambda x:x[1],reverse = True)

>>> d

[('a', 34), ('fdsfds', 9), ('zhang', 5), ('dfds', 4)]

>>> d.sort(key = lambda x:x[1])

>>> d

[('dfds', 4), ('zhang', 5), ('fdsfds', 9), ('a', 34)]

### 多级排序

>>> e = [(1,3,4),(5,6,7),(0,1,3)]

>>> e

[(1, 3, 4), (5, 6, 7), (0, 1, 3)]

>>> import operator #引入包

>>> e.sort(key = operator.itemgetter(1,2)) #包函数，对二三项排序

>>> e

[(0, 1, 3), (1, 3, 4), (5, 6, 7)]

# Tuple

## 索引

用tuple表示如下：————**创建后内容不能改变**

>>> t = ('Adam', 'Lisa', 'Bart')

创建tuple和创建list唯一不同之处是用( )替代了[ ]

单元素 tuple 要多加一个逗号“,”，这样就避免了歧义：

>>> t = (1,)

>>> print t

(1,)

## “可变”的tuple

前面我们看到了tuple一旦创建就不能修改。现在，我们来看一个“可变”的tuple：

>>> t = ('a', 'b', ['A', 'B'])

**注意**到 t 有 3 个元素：**'a'，'b'**和一个list：**['A', 'B']**。list作为一个整体是tuple的第3个元素。list对象可以通过 t[2] 拿到：

>>> L = t[2]

然后，我们把list的两个元素改一改：

>>> L[0] = 'X'

>>> L[1] = 'Y'

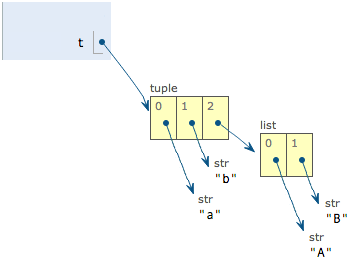
再看看tuple的内容：

>>> print t

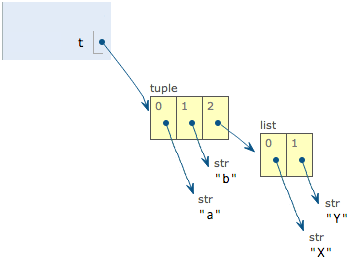
('a', 'b', ['X', 'Y'])

不是说tuple一旦定义后就不可变了吗？怎么现在又变了？

别急，我们先看看定义的时候tuple包含的3个元素：



当我们把list的元素**'A'和'B'**修改为**'X'和'Y'**后，tuple变为：



表面上看，tuple的元素确实变了，但其实变的不是 tuple 的元素，而是list的元素。

tuple一开始指向的list并没有改成别的list，所以，tuple所谓的**“不变”**是说，tuple的每个元素，指向永远不变。即**指向'a'，就不能改成指向'b'**，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的！

理解了**“指向不变”**后，要创建一个内容也不变的tuple怎么做？那就必须保证tuple的每一个元素本身也不能变。

# 字典dict

**一、什么是字典？**

字典是Python语言中唯一的映射类型。

映射类型对象里哈希值（键，key）和指向的对象（值，value）是一对多的的关系，通常被认为是可变的哈希表。

字典对象是可变的，它是一个容器类型，能存储任意个数的Python对象，其中也可包括其他容器类型。

字典类型与序列类型的区别：

1.存取和访问数据的方式不同。  
2.序列类型只用数字类型的键（从序列的开始按数值顺序索引）；  
3.映射类型可以用其他对象类型作键（如：数字、字符串、元祖，一般用字符串作键），和序列类型的键不同，映射类型的键直4.接或间接地和存储数据值相关联。  
5.映射类型中的数据是无序排列的。这和序列类型是不一样的，序列类型是以数值序排列的。  
6.映射类型用键直接“映射”到值。

字典是Python中最强大的数据类型之一。

**二、如何创建字典和给字典赋值**

简单地说字典就是用大括号包裹的键值对的集合。（键值对也被称作项）  
一般形式：

复制代码 代码如下:

adict = {}  
adict = {key1：value2， key2：value2， …}

或用dict()函数，如，adict = dict() 或 adict = dict（（['x',1],['y',2]））这样写对吗？adict = dict（['x',1],['y',2]）。关键字参数创建字典，如：adict= dict(name='allen',age='40′)  
或用fromkeys()方法，如，adict = {}.fromkeys((‘x','y'), -1) 这样创建的字典的value是一样的，若不给值，默认为None。

特点：  
1、键与值用冒号“：”分开；  
2、项与项用逗号“，”分开；  
3、字典中的键必须是唯一的，而值可以不唯一。

复制代码 代码如下:

adict = {‘name':'allen', ‘name':'lucy', ‘age':'40′} 与 bdict = {‘name':'allen', ‘name2′:'allen', ‘age':'40′}

注意：如果字典中的值为数字，最好使用字符串数字形式，如：'age':'040′ 而不用 ‘age':040

**三、字典的基本操作**

1、如何访问字典中的值？  
adict[key] 形式返回键key对应的值value，如果key不在字典中会引发一个KeyError。

2、如何检查key是否在字典中？

a、has\_key()方法 形如：adict.haskey(‘name') 有–>True，无–>False  
b、in 、not in   形如：'name' in adict      有–>True，无–>False

3、如何更新字典？

a、添加一个数据项（新元素）或键值对  
adict[new\_key] = value 形式添加一个项  
b、更新一个数据项（元素）或键值对  
adict[old\_key] = new\_value  
c、删除一个数据项（元素）或键值对  
del adict[key] 删除键key的项 / del adict 删除整个字典  
adict.pop(key) 删除键key的项并返回key对应的 value值

**四、映射类型操作符**

标准类型操作符（+，-，\*，<,>,<=,>=,==,!=,and,or, not)

a、字典不支持拼接和重复操作符（+，\*）  
b、字典的比较操作  
先比较字典的长度也就是字典的元素个数  
键比较  
值比较  
例子：

复制代码 代码如下:

adict = {}  
bdict = {‘name':'allen', ‘age':'40′}  
cmp(adict, bdict)  < –>-1 or > –>1 or ==  –>0

**五、映射相关的函数**

1、len() 返回字典的长度  
2、hash() 返回对象的哈希值，可以用来判断一个对象能否用来作为字典的键  
3、dict() 工厂函数，用来创建字典

**六、字典的方法**

1、adict.keys() 返回一个包含字典所有KEY的列表；  
2、adict.values() 返回一个包含字典所有value的列表；  
3、adict.items() 返回一个包含所有（键，值）元祖的列表；  
4、adict.clear() 删除字典中的所有项或元素；  
5、adict.copy() 返回一个字典浅拷贝的副本；  
6、adict.fromkeys(seq, val=None) 创建并返回一个新字典，以seq中的元素做该字典的键，val做该字典中所有键对应的初始值（默认为None）；  
7、adict.get(key, default = None) 返回字典中key对应的值，若key不存在字典中，则返回default的值（default默认为None）；  
8、adict.has\_key(key) 如果key在字典中，返回True，否则返回False。 现在用 in 、 not in；  
9、adict.iteritems()、adict.iterkeys()、adict.itervalues() 与它们对应的非迭代方法一样，不同的是它们返回一个迭代子，而不是一个列表；  
10、adict.pop(key[,default]) 和get方法相似。如果字典中存在key，删除并返回key对应的vuale；如果key不存在，且没有给出default的值，则引发keyerror异常；  
11、adict.setdefault(key, default=None) 和set()方法相似，但如果字典中不存在Key键，由 adict[key] = default 为它赋值；  
12、adict.update(bdict) 将字典bdict的键值对添加到字典adict中。

**七、字典的遍历**

1、遍历字典的key（键）

复制代码 代码如下:

for key in adict.keys():print key

2、遍历字典的value（值）

复制代码 代码如下:

for value in adict.values(): print value

3、遍历字典的项（元素）

复制代码 代码如下:

for item in adict.items():print item

4、遍历字典的key-value

复制代码 代码如下:

for item，value in adict.items(): print ‘key=%s, value=%s' %(item, value)  或   for item，value in adict.iteritems(): print ‘key=%s, value=%s' %(item, value)

注意：for item,value in adict.items(): print ‘key=%s', ‘value=%s', %(item, value) 这种写法是错误的

**八、使用字典的注意事项**

1、不能允许一键对应多个值；  
2、键必须是可哈希的。

字典是无序的，它不能通过偏移来存取，只能通过键来存取。

字典 = {'key':value} key：类似我们现实的钥匙，而value则是锁。一个钥匙开一个锁

特点：

内部没有顺序，通过键来读取内容，可嵌套，方便我们组织多种数据结构，并且可以原地修改里面的内容，

属于可变类型。

组成字典的键必须是不可变的数据类型，比如，数字，字符串，元组等，列表等可变对象不能作为键.

## dict的特点

**dict的第一个特点是查找速度快，无论dict有10个元素还是10万个元素，查找速度都一样**。而list的查找速度随着元素增加而逐渐下降。

不过dict的查找速度快不是没有代价的，**dict的缺点是占用内存大，还会浪费很多内容**，list正好相反，占用内存小，但是查找速度慢。

由于dict是按 key 查找，所以，在一个dict中，key不能重复。

**dict的第二个特点就是存储的key-value序对是没有顺序的！**这和list不一样：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

当我们试图打印这个dict时：

>>> print d

{'Lisa': 85, 'Adam': 95, 'Bart': 59}

打印的顺序不一定是我们创建时的顺序，而且，不同的机器打印的顺序都可能不同，这说明dict内部是**无序**的，不能用dict存储有序的集合。

**dict的第三个特点是作为 key 的元素必须不可变**，Python的基本类型如字符串、整数、浮点数都是不可变的，都可以作为 key。但是list是可变的，就不能作为 key。

可以试试用list作为key时会报什么样的错误。

不可变这个限制仅作用于key，value是否可变无所谓：

{

'123': [1, 2, 3], # key 是 str，value是list

123: '123', # key 是 int，value 是 str

('a', 'b'): True # key 是 tuple，并且tuple的每个元素都是不可变对象，value是 boolean

}

最常用的key还是字符串，因为用起来最方便。

## 创建字典。{},dict()

info = {'name':'lilei', 'age': 20}

info = dict(name='lilei',age=20)

## 添加内容 a['xx'] = 'xx'

比如 info['phone'] = 'iphone5'

dict是可变的，也就是说，我们可以随时往dict中添加新的 key-value。比如已有dict：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

要把新同学'Paul'的成绩 72 加进去，用赋值语句：

>>> d['Paul'] = 72

再看看dict的内容：

>>> print d

{'Lisa': 85, 'Paul': 72, 'Adam': 95, 'Bart': 59}

如果 key 已经存在，则赋值会用新的 value 替换掉原来的 value：

>>> d['Bart'] = 60

>>> print d

{'Lisa': 85, 'Paul': 72, 'Adam': 95, 'Bart': 60}

## 修改内容 a['xx'] = 'xx' ,

info['phone'] = 'htc'

update 参数是一个字典的类型，他会覆盖相同键的值

info.update({'city':'beijing','phone':'nokia'})

htc 变成了nokia了

## 删除 del,clear,pop

del student['age']删除某个元素

>>> student

{'age': 25, 'score': {'math': 75, 'chinese': 80, 'english': 85}, 'name': 'liming'}

>>> del student['age']

>>> student

{'score': {'python': 60, 'math': 75, 'chinese': 80, 'english': 85}, 'name': 'liming'}

student.clear()删除字典的全部元素

>>> student

{'score': {'python': 60, 'math': 75, 'chinese': 80, 'english': 85}, 'name': 'liming'}

>>> student.clear()

>>> student

{}

student2.pop('age') 存在key直接删除，或返回二参

>>> student2

{'age': 23, 'score': {'python': 80, 'math': 89, 'chinese': 75, 'english': 78}, 'name': 'zhangqiang'}

>>> student2.pop('age')

23

>>> student2

{'score': {'python': 80, 'math': 89, 'chinese': 75, 'english': 78}, 'name': 'zhangqiang'}

>>> student2.pop('age','bucunzai')

'bucunzai'

## 访问dict

我们已经能创建一个dict，用于表示名字和成绩的对应关系：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

可以简单地使用 d[key] 的形式来查找对应的 value，这和 list 很像，不同之处是，**list 必须使用索引返回对应的元素，而dict使用key：**

>>> print d['Adam']

95

>>> print d['Paul']

Traceback (most recent call last):

File "index.py", line 11, in <module>

print d['Paul']

KeyError: 'Paul'

**注意:** 通过 key 访问 dict 的value，只要 key 存在，dict就返回对应的value。如果key不存在，会直接报错：KeyError。

要避免 KeyError 发生，有两个办法：

**一是先判断一下 key 是否存在，用 in 操作符：**

if 'Paul' in d:

print d['Paul']

如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。

**二是使用dict本身提供的一个 get 方法，在Key不存在的时候，返回None：**

>>> print d.get('Bart')

59

>>> print d.get('Paul')

None

## 遍历dict

由于dict也是一个集合，所以，遍历dict和遍历list类似，都可以通过 for 循环实现。

直接使用for循环可以遍历 dict 的 key：

请用 for 循环遍历如下的dict，打印出 name: score 来。

通过d[key]获取对应的value。

d= {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

for key in d:

print key + ':', d[key]

## 方法：

### Len()

用 **dict**表示**“名字”-“成绩”**的查找表如下：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

我们把**名字称为key**，对应的**成绩称为value**，dict就是通过 **key** 来查找 **value**。

花括号 {} 表示这是一个dict，然后按照**key: value**, 写出来即可。最后一个 key: value 的逗号可以省略。

由于dict也是集合，len() 函数可以计算任意集合的大小：

>>> len(d)

3

**注意:** 一个 key-value 算一个，因此，dict大小为3。

### keys(): 返回的是列表，里面包含了字典的所有键

>>> a

{'shuo': 123, 'zhang': 11}

>>> a.keys()

['shuo', 'zhang']

### values():返回的是列表，里面包含了字典的所有值

>>> a

{'shuo': 123, 'zhang': 11}

>>> a.values()

[123, 11]

### items：生成一个字典的容器：[()]，每一个元素都是元组

>>> a

{'shuo': 123, 'zhang': 11}

>>> a.items()

[('shuo', 123), ('zhang', 11)]

### get：从字典中获得一个值

**info.get('name')**

>>> a.get('zhang')

11

### info.get('age2','22')

>>> a.get('shu1o','abc')#shu1o 这个是空的的没有的key

'abc'

>>> a

{'shuo': 123, 'zhang': 11}

### 字典排序

字典，形如 dic = {'a':1 , 'b':2 , 'c': 3},字典中的元素没有顺序，所以dic[0]是有语法错误的。并且不可以有重复的键值，所以 dic.add['c'] = 4后，字典变成 {'a':1 , 'b':2 , 'c': 4}.

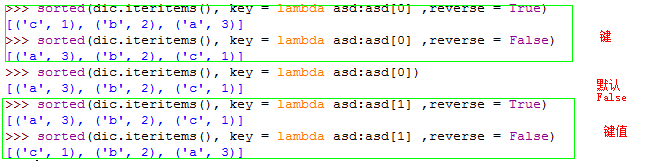
**函数原型**

sorted(dic,value,reverse)

* dic为比较函数，value 为排序的对象（这里指键或键值），
* reverse：注明升序还是降序，True--降序，False--升序（默认）

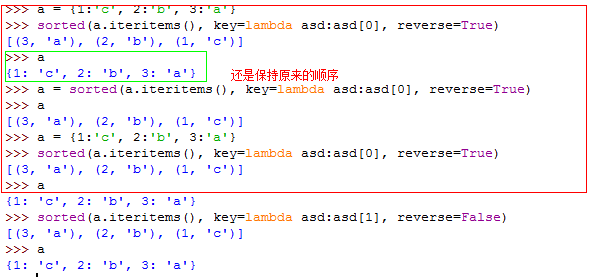
**案例**

**dic = {'a':3 , 'b':2 , 'c': 1}**



**注意**

排序之后原字典没有变,顺序依旧



**两个函数**

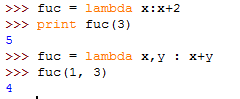
**1.lambda**

功能: 创建匿名函数

区别: 和def函数定义,以下两点区别

* lambda会创建一个函数对象，但不会把这个函数对象赋给一个标识符;而def则会把函数对象赋值给一个变量
* lambda它只是一个表达式，而def则是一个语句

案例：

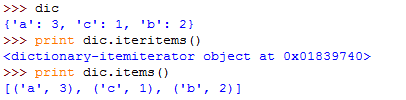


**2.iteritems()**

功能: iteritems()以迭代器对象返回字典键值对

区别: 和item相比:items以列表形式返回字典键值对

案例：



注释: 在函数sorted(dic.iteritems(), key = lambda asd:asd[1])中，第一个参数传给第二个参数“键-键值”，第二个参数取出其中的键([0])或键值(1])

 ###################################################

 下面的是按照value的值从大到小的顺序来排序。  
  
dic = {'a':31, 'bc':5, 'c':3, 'asd':4, 'aa':74, 'd':0}  
dict= sorted(dic.iteritems(), key=lambda d:d[1], reverse = True)  
print dict  
  
输出的结果：  
[('aa', 74), ('a', 31), ('bc', 5), ('asd', 4), ('c', 3), ('d', 0)]  
  
下面我们分解下代码  
print dic.iteritems() 得到[(键，值)]的列表。  
然后用sorted方法，通过key这个参数，指定排序是按照value，也就是第一个元素d[1的值来排序。reverse = True表示是需要翻转的，默认是从小到大，翻转的话，那就是从大到小。

### 字典最小最大值

max(d, key=d.get)

Demo:

>>> d= {'a':2,'b':5,'c':3}

>>> max(d, key=d.get)

'b'

# 集合set

## 什么是set

**dict的作用是建立一组 key 和一组 value 的映射关系，dict的key是不能重复的。**

有的时候，我们只想要 dict 的 key，不关心 key 对应的 value，目的就是保证这个集合的元素不会重复，这时，set就派上用场了。

**set 持有一系列元素，这一点和 list 很像，但是set的元素没有重复，而且是无序的，这点和 dict 的 key很像。**

创建 set 的方式是调用 set() 并传入一个 list，list的元素将作为set的元素：

>>> s = set(['A', 'B', 'C'])

可以查看 set 的内容：

>>> print s

set(['A', 'C', 'B'])

**请注意**，上述打印的形式类似 list， 但它不是 list，仔细看还可以发现，打印的顺序和原始 list 的顺序有可能是不同的，因为set内部存储的元素是**无序**的。

因为set不能包含重复的元素，所以，当我们传入包含重复元素的 list 会怎么样呢？

>>> s = set(['A', 'B', 'C', 'C'])

>>> print s

set(['A', 'C', 'B'])

>>> len(s)

3

结果显示，set会自动去掉重复的元素，原来的list有4个元素，但set只有3个元素。

## 访问set

由于**set存储的是无序集合**，所以我们没法通过索引来访问。

访问 set中的某个元素实际上就是判断一个元素是否在set中。

例如，存储了班里同学名字的set：

>>> s = set(['Adam', 'Lisa', 'Bart', 'Paul'])

我们可以用 in 操作符判断：

Bart是该班的同学吗？

>>> 'Bart' in s

True

bart是该班的同学吗？

>>> 'bart' in s

False

看来大小写很重要，'Bart' 和 'bart'被认为是两个不同的元素

## set的特点

**set的内部结构和dict很像，唯一区别是不存储value**，因此，判断一个元素是否在set中速度很快。

**set存储的元素和dict的key类似，必须是不变对象**，因此，任何可变对象是不能放入set中的。

最后，set存储的元素也是没有顺序的。

set的这些特点，可以应用在哪些地方呢？

星期一到星期日可以用字符串'MON', 'TUE', ... 'SUN'表示。

假设我们让用户输入星期一至星期日的某天，如何判断用户的输入是否是一个有效的星期呢？

weekdays = set(['MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'])

再判断输入是否有效，只需要判断该字符串是否在set中：

x = '???' # 用户输入的字符串

if x in weekdays:

print 'input error'

else:

print 'input ok'

## 遍历set

由于 set 也是一个集合，所以，遍历 set 和遍历 list 类似，都可以通过 for 循环实现。

直接使用 for 循环可以遍历 set 的元素：

>>> s = set(['Adam', 'Lisa', 'Bart'])

>>> for name in s:

... print name

...

Lisa

Adam

Bart

**注意:**观察 for 循环在遍历set时，元素的顺序和list的顺序很可能是不同的，而且不同的机器上运行的结果也可能不同。

## 更新set

由于**set存储的是一组不重复的无序元素**，因此，更新set主要做两件事：

**一是把新的元素添加到set中，二是把已有元素从set中删除。**

### add()方法****添加****

添加元素时，用set的add()方法：

>>> s = set([1, 2, 3])

>>> s.add(4)

>>> print s

set([1, 2, 3, 4])

如果添加的元素已经存在于set中，add()不会报错，但是不会加进去了：

>>> s = set([1, 2, 3])

>>> s.add(3)

>>> print s

set([1, 2, 3])

### update方法添加

update和add的不同就是add是把添加的对象作为一个加入而update是每一个元素作为一个插入

>>> a

set(['a', 'c', 'b'])

>>> a.update('python')

>>> a

set(['a', 'c', 'b', 'h', 'o', 'n', 'p', 't', 'y'])

>>> a.add('python')

>>> a

set(['a', 'c', 'b', 'python', 'h', 'o', 'n', 'p', 't', 'y'])

### remove()方法删除

删除set中的元素时，用set的remove()方法：

>>> s = set([1, 2, 3, 4])

>>> s.remove(4)

>>> print s

set([1, 2, 3])

如果删除的元素不存在set中，remove()会报错：

>>> s = set([1, 2, 3])

>>> s.remove(4)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 4

所以用add()可以直接添加，而remove()前需要判断。

### 成员关系in not in

>>> a

set(['a', 'c', 'b', 'h', 'o', 'n', 'p', 't', 'y'])

>>> 'a' in a

True

>>> 'python' in a

False

>>> 'python' not in a

### & | -交集，并集，差集 & | -

& 交集：两个集合相同的部分

| 并集：两个集合共同的部分

>>> a

set(['a', 'c', 'b', 'h', 'o', 'n', 'p', 't', 'y'])

>>> b

set(['a', 'python', 'h', 'o', 'n', 'p', 't'])

>>> a&b

set(['a', 'h', 'o', 'n', 'p', 't'])

>>> a|b

set(['a', 'c', 'b', 'python', 'h', 'o', 'n', 'p', 't', 'y'])

>>> a-b

set(['y', 'c', 'b'])

### set去重

>>> a = [1,2,3,4,5,6]

>>> a

[1, 2, 3, 4, 5, 6]

>>> a.append(1)

>>> a

[1, 2, 3, 4, 5, 6, 1]

>>> a.append(5)

>>> a.append(4)

>>> a

[1, 2, 3, 4, 5, 6, 1, 5, 4]

>>> a=set(a)

>>> a

set([1, 2, 3, 4, 5, 6])

>>> a=list(a)

>>> a

[1, 2, 3, 4, 5, 6]

## Frozenset（）不可变集合，类似tuple

>>> c = frozenset('abc')

>>> c

frozenset(['a', 'c', 'b'])

>>> c.add('w')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'frozenset' object has no attribute 'add'

# 什么是迭代

在Python中，如果给定一个**list**或**tuple**，我们可以通过for循环来遍历这个list或tuple，这种遍历我们成为迭代（Iteration）。

在Python中，迭代是通过 for ... in 来完成的，而很多语言比如C或者Java，迭代list是通过下标完成的，比如Java代码：

for (i=0; i<list.length; i++) {

n = list[i];

}

可以看出，Python的for循环抽象程度要高于Java的for循环。

**因为 Python 的 for循环不仅可以用在list或tuple上，还可以作用在其他任何可迭代对象上。**

因此，迭代操作就是对于一个集合，无论该集合是有序还是无序，我们用 for 循环总是可以依次取出集合的每一个元素。

**注意**: 集合是指包含一组元素的数据结构，我们已经介绍的包括：

1. **有序集合**：list，tuple，str和unicode；

2. **无序集合**：set

3. **无序集合并且具有 key-value 对**：dict

而迭代是一个动词，它指的是一种操作，在Python中，就是 for 循环。

迭代与按下标访问数组最大的不同是，后者是一种具体的迭代实现方式，而前者只关心迭代结果，根本不关心迭代内部是如何实现的。

## 索引迭代

Python中，**迭代永远是取出元素本身，而非元素的索引。**

对于有序集合，元素确实是有索引的。有的时候，我们确实想在 for 循环中拿到索引，怎么办？

方法是使用 **enumerate() 函数**：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> for index, name in enumerate(L):

... print index, '-', name

...

0 - Adam

1 - Lisa

2 - Bart

3 - Paul

使用 enumerate() 函数，我们可以在for循环中同时绑定索引index和元素name。但是，这不是 enumerate() 的特殊语法。实际上，enumerate() 函数把：

**['Adam', 'Lisa', 'Bart', 'Paul']**

变成了类似：

**[(0, 'Adam'), (1, 'Lisa'), (2, 'Bart'), (3, 'Paul')]**

因此，迭代的每一个元素实际上是一个tuple：

for t in enumerate(L):

index = t[0]

name = t[1]

print index, '-', name

如果我们知道每个tuple元素都包含两个元素，for循环又可以进一步简写为：

for index, name in enumerate(L):

print index, '-', name

这样不但代码更简单，而且还少了两条赋值语句。

可见，索引迭代也不是真的按索引访问，而是由 enumerate() 函数自动把每个元素变成 (index, element) 这样的tuple，再迭代，就同时获得了索引和元素本身。

dict迭代value

我们已经了解了**dict对象**本身就是可**迭代对象**，用 for 循环直接迭代 dict，可以每次拿到dict的一个key。

如果我们希望迭代 dict 对象的value，应该怎么做？

dict 对象有一个 **values() 方法**，这个方法把dict转换成一个包含所有value的list，这样，我们迭代的就是 dict的每一个 value：

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

print d.values()

# [85, 95, 59]

for v in d.values():

print v

# 85

# 95

# 59

如果仔细阅读Python的文档，还可以发现，dict除了**values()**方法外，还有一个**itervalues()**方法，用**itervalues()** 方法替代 **values()** 方法，迭代效果完全一样：

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

print d.itervalues()

# <dictionary-valueiterator object at 0x106adbb50>

for v in d.itervalues():

print v

# 85

# 95

# 59

**那这两个方法有何不同之处呢？**

1. **values()** 方法实际上把一个 dict 转换成了包含 value 的list。

2. 但是 **itervalues()** 方法不会转换，它会在迭代过程中依次从 dict 中取出 value，所以 itervalues() 方法比 values() 方法节省了生成 list 所需的内存。

3. 打印 itervalues() 发现它返回一个 <dictionary-valueiterator> 对象，这说明在Python中，**for 循环可作用的迭代对象远不止 list，tuple，str，unicode，dict等**，任何可迭代对象都可以作用于for循环，而内部如何迭代我们通常并不用关心。

**如果一个对象说自己可迭代，那我们就直接用 for 循环去迭代它，可见，迭代是一种抽象的数据操作，它不对迭代对象内部的数据有任何要求。**

## dict迭代key和value

我们了解了如何**迭代 dict** 的**key**和**value**，那么，在一个 for 循环中，能否同时迭代 key和value？答案是肯定的。

首先，我们看看 dict 对象的 **items()** 方法返回的值：

>>> d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

>>> print d.items()

[('Lisa', 85), ('Adam', 95), ('Bart', 59)]

可以看到，items() 方法把dict对象转换成了包含tuple的list，我们对这个list进行迭代，可以同时获得key和value：

>>> for key, value in d.items():

... print key, ':', value

...

Lisa : 85

Adam : 95

Bart : 59

和 values() 有一个 itervalues() 类似， **items()**也有一个对应的 **iteritems()**，iteritems() 不把dict转换成list，而是在迭代过程中不断给出 tuple，所以， iteritems() 不占用额外的内存。

## 生成列表

要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]，我们可以用range(1, 11)：

>>> range(1, 11)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

但如果要生成[1x1, 2x2, 3x3, ..., 10x10]怎么做？方法一是循环：

>>> L = []

>>> for x in range(1, 11):

... L.append(x \* x)

...

>>> L

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

但是循环太繁琐，而列表生成式则可以用一行语句代替循环生成上面的list：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

这种写法就是Python特有的列表生成式。利用列表生成式，可以以非常简洁的代码生成 list。

写列表生成式时，把要生成的元素 x \* x 放到前面，后面跟 for 循环，就可以把list创建出来，十分有用，多写几次，很快就可以熟悉这种语法。

### 100以内的奇数：

>>> [x for x in range(1,101) if x%2==1]

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99]

### 1-100 数的平方

>>> [x\*x for x in range(1,101)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481, 3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184, 5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225, 7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604, 9801, 10000]

>>>

### 字符串生成

>>> ["the %s" % d for d in range(1,101)]

['the 1', 'the 2', 'the 3', 'the 4', 'the 5', 'the 6', 'the 7', 'the 8', 'the 9', 'the 10', 'the 11', 'the 12', 'the 13', 'the 14', 'the 15', 'the 16', 'the 17', 'the 18', 'the 19', 'the 20', 'the 21', 'the 22', 'the 23', 'the 24', 'the 25', 'the 26', 'the 27', 'the 28', 'the 29', 'the 30', 'the 31', 'the 32', 'the 33', 'the 34', 'the 35', 'the 36', 'the 37', 'the 38', 'the 39', 'the 40', 'the 41', 'the 42', 'the 43', 'the 44', 'the 45', 'the 46', 'the 47', 'the 48', 'the 49', 'the 50', 'the 51', 'the 52', 'the 53', 'the 54', 'the 55', 'the 56', 'the 57', 'the 58', 'the 59', 'the 60', 'the 61', 'the 62', 'the 63', 'the 64', 'the 65', 'the 66', 'the 67', 'the 68', 'the 69', 'the 70', 'the 71', 'the 72', 'the 73', 'the 74', 'the 75', 'the 76', 'the 77', 'the 78', 'the 79', 'the 80', 'the 81', 'the 82', 'the 83', 'the 84', 'the 85', 'the 86', 'the 87', 'the 88', 'the 89', 'the 90', 'the 91', 'the 92', 'the 93', 'the 94', 'the 95', 'the 96', 'the 97', 'the 98', 'the 99', 'the 100']

### 元组生成

>>> [(x,y) for x in range(3) for y in range(3)]

[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]

### 字典生成

>>> dict([(x,y) for x in range(3) for y in range(3)])

{0: 2, 1: 2, 2: 2}

## 复杂表达式

使用**for循环**的迭代不仅可以迭代普通的list，还可以迭代dict。

假设有如下的dict：

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

完全可以通过一个复杂的列表生成式把它变成一个 HTML 表格：

tds = ['<tr><td>%s</td><td>%s</td></tr>' % (name, score) for name, score in d.iteritems()]

print '<table>'

print '<tr><th>Name</th><th>Score</th><tr>'

print '\n'.join(tds)

print '</table>'

**注：**字符串可以通过 % 进行格式化，用指定的参数替代%s。字符串的join()方法可以把一个 list 拼接成一个字符串。

把打印出来的结果保存为一个html文件，就可以在浏览器中看到效果了：

<table border="1">

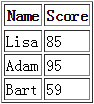
<tr><th>Name</th><th>Score</th><tr>

<tr><td>Lisa</td><td>85</td></tr>

<tr><td>Adam</td><td>95</td></tr>

<tr><td>Bart</td><td>59</td></tr>

</table>



## 条件过滤

列表生成式的 **for 循环后面还可以加上 if 判断**。例如：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

如果我们只想要偶数的平方，不改动 range()的情况下，可以加上 if 来筛选：

>>> [x \* x for x in range(1, 11) if x % 2 == 0]

[4, 16, 36, 64, 100]

有了 if 条件，只有 if 判断为 True 的时候，才把循环的当前元素添加到列表中。

## 多层表达式

对于字符串 'ABC' 和 '123'，可以使用两层循环，生成全排列：

>>> [m + n for m in 'ABC' for n in '123']

['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']

翻译成循环代码就像下面这样：

L = []

for m in 'ABC':

for n in '123':

L.append(m + n)

# 正则表达式

正则表达式，是字符串检索引擎，最早起源于unix。

**1.unix下的正则 awk grep egrep**

**2.正则的几个基本概念**

zhangshuo@zhangshuo-virtual-machine:~/文档$ ls | grep '1[1-9]\{1\}'

11

12

zhangshuo@zhangshuo-virtual-machine:~/文档$ ls

1 11 12 13 1a 2 3 a

（'1[1-9]\{1\}'表示正则表达式，1后[1-9]表示后面数字匹配1-9，{1\}表示匹配一位数字就是1-9配平一次 ）

**[0-9]配平0-9**

**\d 匹配全部数字 （我没有试出来，没有成功）**

**\w 单词类字符 a-z A-Z 0-9 \_**

**\W 非单词类字符**

**{2} {n} 前面的表达式匹配n次**

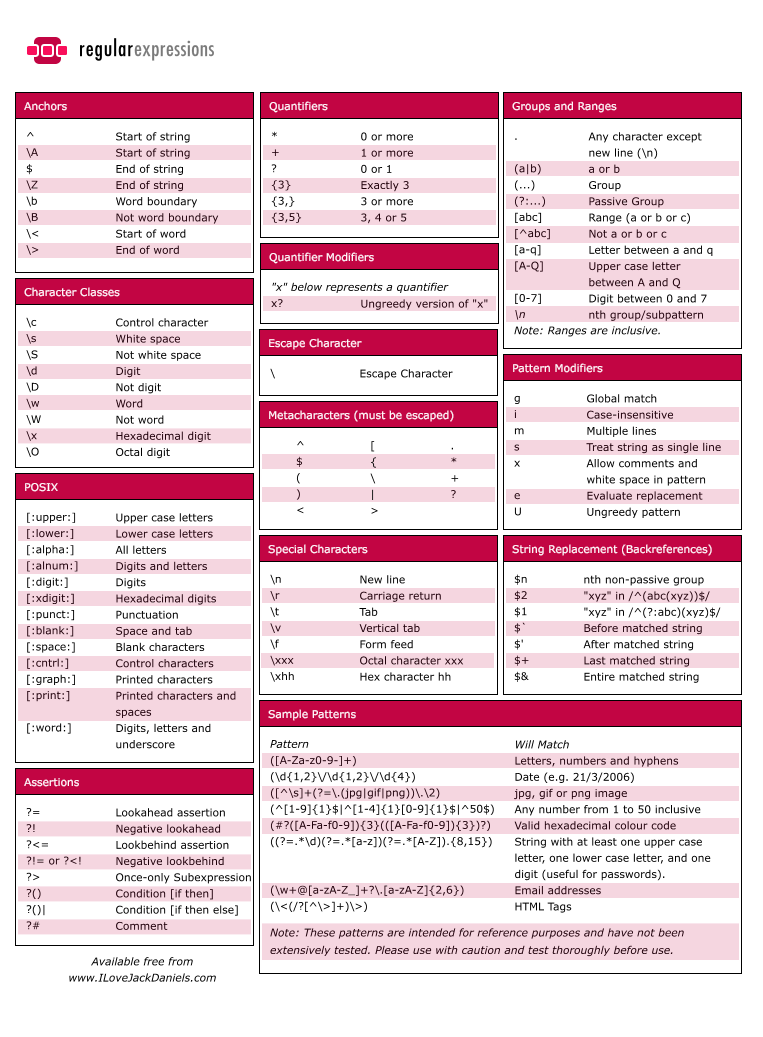
**{0,2} {m,n} 前面的表达式匹配m到n次**

**+ 前面的表达式，出现1到无限次 最少，出现1次**

**? 前面的表达式，出现0到1次 最多，出现1次**

**\* 前面的表达式，出现0到无限次 出现不出现，都没关系**

**$ 美元符号表示完全匹配，只有一模一样才会返回（ ‘/d$’,’44’）因为两个4没有匹配**



**3.python里的正则模块 re**

**4.一些基本操作**

**4.1 一次取配 match:"hello lilei" r'(\w+) (\w+)'**

**4.2 切割 split**

**4.3 查找全部 findall**

**4.4 finditer 迭代器什么的最有爱了**

#### 正则表达式

阅读: 3693

字符串是编程时涉及到的最多的一种数据结构，对字符串进行操作的需求几乎无处不在。比如判断一个字符串是否是合法的Email地址，虽然可以编程提取@前后的子串，再分别判断是否是单词和域名，但这样做不但麻烦，而且代码难以复用。

正则表达式是一种用来匹配字符串的强有力的武器。它的设计思想是用一种描述性的语言来给字符串定义一个规则，凡是符合规则的字符串，我们就认为它“匹配”了，否则，该字符串就是不合法的。

所以我们判断一个字符串是否是合法的Email的方法是：

创建一个匹配Email的正则表达式；

用该正则表达式去匹配用户的输入来判断是否合法。

因为正则表达式也是用字符串表示的，所以，我们要首先了解如何用字符来描述字符。

在正则表达式中，如果直接给出字符，就是精确匹配。用\d可以匹配一个数字，\w可以匹配一个字母或数字，所以：

'00\d'可以匹配'007'，但无法匹配'00A'；

'\d\d\d'可以匹配'010'；

'\w\w\d'可以匹配'py3'；

.可以匹配任意字符，所以：

* 'py.'可以匹配'pyc'、'pyo'、'py!'等等。

要匹配变长的字符，在正则表达式中，用\*表示任意个字符（包括0个），用+表示至少一个字符，用?表示0个或1个字符，用{n}表示n个字符，用{n,m}表示n-m个字符：

来看一个复杂的例子：\d{3}\s+\d{3,8}。

我们来从左到右解读一下：

\d{3}表示匹配3个数字，例如'010'；

\s可以匹配一个空格（也包括Tab等空白符），所以\s+表示至少有一个空格，例如匹配' '，' '等；

\d{3,8}表示3-8个数字，例如'1234567'。

综合起来，上面的正则表达式可以匹配以任意个空格隔开的带区号的电话号码。

如果要匹配'010-12345'这样的号码呢？由于'-'是特殊字符，在正则表达式中，要用'\'转义，所以，上面的正则是\d{3}\-\d{3,8}。

但是，仍然无法匹配'010 - 12345'，因为带有空格。所以我们需要更复杂的匹配方式。

### 进阶

要做更精确地匹配，可以用[]表示范围，比如：

[0-9a-zA-Z\\_]可以匹配一个数字、字母或者下划线；

[0-9a-zA-Z\\_]+可以匹配至少由一个数字、字母或者下划线组成的字符串，比如'a100'，'0\_Z'，'Py3000'等等；

[a-zA-Z\\_][0-9a-zA-Z\\_]\*可以匹配由字母或下划线开头，后接任意个由一个数字、字母或者下划线组成的字符串，也就是Python合法的变量；

[a-zA-Z\\_][0-9a-zA-Z\\_]{0, 19}更精确地限制了变量的长度是1-20个字符（前面1个字符+后面最多19个字符）。

A|B可以匹配A或B，所以[P|p]ython可以匹配'Python'或者'python'。

^表示行的开头，^\d表示必须以数字开头。

$表示行的结束，\d$表示必须以数字结束。

你可能注意到了，py也可以匹配'python'，但是加上^py$就变成了整行匹配，就只能匹配'py'了。

### re模块

有了准备知识，我们就可以在Python中使用正则表达式了。Python提供re模块，包含所有正则表达式的功能。由于Python的字符串本身也用\转义，所以要特别注意：

s = 'ABC\\-001' *# Python的字符串# 对应的正则表达式字符串变成：# 'ABC\-001'*

因此我们强烈建议使用Python的r前缀，就不用考虑转义的问题了：

s = r'ABC\-001' *# Python的字符串# 对应的正则表达式字符串不变：# 'ABC\-001'*

先看看如何判断正则表达式是否匹配：

>>> **import** re>>> re.match(r'^\d{3}\-\d{3,8}$', '010-12345') <\_sre.SRE\_Match object at 0x1026e18b8>>>> re.match(r'^\d{3}\-\d{3,8}$', '010 12345') >>>

match()方法判断是否匹配，如果匹配成功，返回一个Match对象，否则返回None。常见的判断方法就是：

test = '用户输入的字符串'**if** re.match(r'正则表达式', test): **print** 'ok'**else**: **print** 'failed'

### 切分字符串

用正则表达式切分字符串比用固定的字符更灵活，请看正常的切分代码：

>>> 'a b c'.split(' ') ['a', 'b', '', '', 'c']

嗯，无法识别连续的空格，用正则表达式试试：

>>> re.split(r'\s+', 'a b c') ['a', 'b', 'c']

无论多少个空格都可以正常分割。加入,试试：

>>> re.split(r'[\s\,]+', 'a,b, c d') ['a', 'b', 'c', 'd']

再加入;试试：

>>> re.split(r'[\s\,\;]+', 'a,b;; c d') ['a', 'b', 'c', 'd']

如果用户输入了一组标签，下次记得用正则表达式来把不规范的输入转化成正确的数组。

### 分组

除了简单地判断是否匹配之外，正则表达式还有提取子串的强大功能。用()表示的就是要提取的分组（Group）。比如：

^(\d{3})-(\d{3,8})$分别定义了两个组，可以直接从匹配的字符串中提取出区号和本地号码：

>>> m = re.match(r'^(\d{3})-(\d{3,8})$', '010-12345')>>> m <\_sre.SRE\_Match object at 0x1026fb3e8>>>> m.group(0)'010-12345'>>> m.group(1)'010'>>> m.group(2)'12345'

如果正则表达式中定义了组，就可以在Match对象上用group()方法提取出子串来。

注意到group(0)永远是原始字符串，group(1)、group(2)……表示第1、2、……个子串。

提取子串非常有用。来看一个更凶残的例子：

>>> t = '19:05:30'>>> m = re.match(r'^(0[0-9]|1[0-9]|2[0-3]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])\:(0[0-9]|1[0-9]|2[0-9]|3[0-9]|4[0-9]|5[0-9]|[0-9])$', t)>>> m.groups() ('19', '05', '30')

这个正则表达式可以直接识别合法的时间。但是有些时候，用正则表达式也无法做到完全验证，比如识别日期：

'^(0[1-9]|1[0-2]|[0-9])-(0[1-9]|1[0-9]|2[0-9]|3[0-1]|[0-9])$'

对于'2-30'，'4-31'这样的非法日期，用正则还是识别不了，或者说写出来非常困难，这时就需要程序配合识别了。

### 贪婪匹配

最后需要特别指出的是，正则匹配默认是贪婪匹配，也就是匹配尽可能多的字符。举例如下，匹配出数字后面的0：

>>> re.match(r'^(\d+)(0\*)$', '102300').groups() ('102300', '')

由于\d+采用贪婪匹配，直接把后面的0全部匹配了，结果0\*只能匹配空字符串了。

必须让\d+采用非贪婪匹配（也就是尽可能少匹配），才能把后面的0匹配出来，加个?就可以让\d+采用非贪婪匹配：

>>> re.match(r'^(\d+?)(0\*)$', '102300').groups() ('1023', '00')

### 编译

当我们在Python中使用正则表达式时，re模块内部会干两件事情：

编译正则表达式，如果正则表达式的字符串本身不合法，会报错；

用编译后的正则表达式去匹配字符串。

如果一个正则表达式要重复使用几千次，出于效率的考虑，我们可以预编译该正则表达式，接下来重复使用时就不需要编译这个步骤了，直接匹配：

>>> **import** re*# 编译:*>>> re\_telephone = re.compile(r'^(\d{3})-(\d{3,8})$')*# 使用：*>>> re\_telephone.match('010-12345').groups() ('010', '12345')>>> re\_telephone.match('010-8086').groups() ('010', '8086')

编译后生成Regular Expression对象，由于该对象自己包含了正则表达式，所以调用对应的方法时不用给出正则字符串。

### 小结

正则表达式非常强大，要在短短的一节里讲完是不可能的。要讲清楚正则的所有内容，可以写一本厚厚的书了。如果你经常遇到正则表达式的问题，你可能需要一本正则表达式的参考书。

请尝试写一个验证Email地址的正则表达式。版本一应该可以验证出类似的Email：

someone@gmail.com bill.gates@microsoft.com

版本二可以验证并提取出带名字的Email地址：

<Tom Paris> tom@voyager.org

引言

    正则表达式（regular expression）就是用一个“字符串”来描述一个特征，然后去验证另一个“字符串”是否符合这个特征。比如 表达式“ab+” 描述的特征是“一个 'a' 和 任意个 'b' ”，那么 'ab', 'abb', 'abbbbbbbbbb' 都符合这个特征。

    正则表达式可以用来：（1）验证字符串是否符合指定特征，比如验证是否是合法的邮件地址。（2）用来查找字符串，从一个长的文本中查找符合指定特征的字符串，比查找固定字符串更加灵活方便。（3）用来替换，比普通的替换更强大。  
  
   正则表达式学习起来其实是很简单的，不多的几个较为抽象的概念也很容易理解。之所以很多人感觉正则表达式比较复杂，一方面是因为大多数的文档没有做到由浅入深地讲解，概念上没有注意先后顺序，给读者的理解带来困难；另一方面，各种引擎自带的文档一般都要介绍它特有的功能，然而这部分特有的功能并不是我们首先要理解的。  
  
    文章中的每一个举例，都可以点击进入到测试页面进行测试。闲话少说，开始。

### 1. 正则表达式规则

##### 1.1 普通字符

    字母、数字、汉字、下划线、以及后边章节中没有特殊定义的标点符号，都是"普通字符"。表达式中的普通字符，在匹配一个字符串的时候，匹配与之相同的一个字符。  
  
    [举例1：表达式 "c"，在匹配字符串 "abcde" 时](http://www.regexlab.com/zh/workshop.htm?pat=c&txt=abcde)，匹配结果是：成功；匹配到的内容是："c"；匹配到的位置是：开始于2，结束于3。（注：下标从0开始还是从1开始，因当前编程语言的不同而可能不同）  
  
    [举例2：表达式 "bcd"，在匹配字符串 "abcde" 时](http://www.regexlab.com/zh/workshop.htm?pat=bcd&txt=abcde)，匹配结果是：成功；匹配到的内容是："bcd"；匹配到的位置是：开始于1，结束于4。



##### 1.2 简单的转义字符

    一些不便书写的字符，采用在前面加 "\" 的方法。这些字符其实我们都已经熟知了。

|  |  |
| --- | --- |
| 表达式 | 可匹配 |
| \r, \n | 代表回车和换行符 |
| \t | 制表符 |
| \\ | 代表 "\" 本身 |

    还有其他一些在后边章节中有特殊用处的标点符号，在前面加 "\" 后，就代表该符号本身。比如：^, $ 都有特殊意义，如果要想匹配字符串中 "^" 和 "$" 字符，则表达式就需要写成 "\^" 和 "\$"。

|  |  |
| --- | --- |
| 表达式 | 可匹配 |
| \^ | 匹配 ^ 符号本身 |
| \$ | 匹配 $ 符号本身 |
| \. | 匹配小数点（.）本身 |

    这些转义字符的匹配方法与 "普通字符" 是类似的。也是匹配与之相同的一个字符。  
  
    [举例1：表达式 "\$d"，在匹配字符串 "abc$de" 时](http://www.regexlab.com/zh/workshop.htm?pat=\\$d&txt=abc$de)，匹配结果是：成功；匹配到的内容是："$d"；匹配到的位置是：开始于3，结束于5。



##### 1.3 能够与 '多种字符' 匹配的表达式

    正则表达式中的一些表示方法，可以匹配 '多种字符' 其中的任意一个字符。比如，表达式 "\d" 可以匹配任意一个数字。虽然可以匹配其中任意字符，但是只能是一个，不是多个。这就好比玩扑克牌时候，大小王可以代替任意一张牌，但是只能代替一张牌。

|  |  |
| --- | --- |
| 表达式 | 可匹配 |
| \d | 任意一个数字，0~9 中的任意一个 |
| \w | 任意一个字母或数字或下划线，也就是 A~Z,a~z,0~9,\_ 中任意一个 |
| \s | 包括空格、制表符、换页符等空白字符的其中任意一个 |
| . | 小数点可以匹配除了换行符（\n）以外的任意一个字符 |

[举例1：表达式 "\d\d"，在匹配 "abc123" 时](http://www.regexlab.com/zh/workshop.htm?pat=\\d\\d&txt=abc123)，匹配的结果是：成功；匹配到的内容是："12"；匹配到的位置是：开始于3，结束于5。  
  
    [举例2：表达式 "a.\d"，在匹配 "aaa100" 时](http://www.regexlab.com/zh/workshop.htm?pat=a.\\d&txt=aaa100)，匹配的结果是：成功；匹配到的内容是："aa1"；匹配到的位置是：开始于1，结束于4。



##### 1.4 自定义能够匹配 '多种字符' 的表达式

    使用方括号 [ ] 包含一系列字符，能够匹配其中任意一个字符。用 [^ ] 包含一系列字符，则能够匹配其中字符之外的任意一个字符。同样的道理，虽然可以匹配其中任意一个，但是只能是一个，不是多个。

|  |  |
| --- | --- |
| 表达式 | 可匹配 |
| [ab5@] | 匹配 "a" 或 "b" 或 "5" 或 "@" |
| [^abc] | 匹配 "a","b","c" 之外的任意一个字符 |
| [f-k] | 匹配 "f"~"k" 之间的任意一个字母 |
| [^A-F0-3] | 匹配 "A"~"F","0"~"3" 之外的任意一个字符 |

[举例1：表达式 "[bcd][bcd]" 匹配 "abc123" 时](http://www.regexlab.com/zh/workshop.htm?pat=[bcd][bcd]&txt=abc123)，匹配的结果是：成功；匹配到的内容是："bc"；匹配到的位置是：开始于1，结束于3。  
  
    [举例2：表达式 "[^abc]" 匹配 "abc123" 时](http://www.regexlab.com/zh/workshop.htm?pat=[^abc]&txt=abc123)，匹配的结果是：成功；匹配到的内容是："1"；匹配到的位置是：开始于3，结束于4。



##### 1.5 修饰匹配次数的特殊符号

    前面章节中讲到的表达式，无论是只能匹配一种字符的表达式，还是可以匹配多种字符其中任意一个的表达式，都只能匹配一次。如果使用表达式再加上修饰匹配次数的特殊符号，那么不用重复书写表达式就可以重复匹配。  
  
    使用方法是："次数修饰"放在"被修饰的表达式"后边。比如："[bcd][bcd]" 可以写成 "[bcd]{2}"。

|  |  |
| --- | --- |
| 表达式 | 作用 |
| {n} | 表达式重复n次，比如：["\w{2}" 相当于 "\w\w"](http://www.regexlab.com/zh/workshop.htm?pat=\\w{2}&txt=ab+c6)；["a{5}" 相当于 "aaaaa"](http://www.regexlab.com/zh/workshop.htm?pat=a{5}&txt=bbaaaaaddee) |
| {m,n} | 表达式至少重复m次，最多重复n次，比如：["ba{1,3}"可以匹配 "ba"或"baa"或"baaa"](http://www.regexlab.com/zh/workshop.htm?pat=ba{1,3}&txt=a,baaa,baa,b,ba) |
| {m,} | 表达式至少重复m次，比如：["\w\d{2,}"可以匹配 "a12","\_456","M12344"...](http://www.regexlab.com/zh/workshop.htm?pat=\\w\\d{2,}&txt=b1,a12,_456,_4AA,M12344,12346546547446534543543) |
| ? | 匹配表达式0次或者1次，相当于 {0,1}，比如：["a[cd]?"可以匹配 "a","ac","ad"](http://www.regexlab.com/zh/workshop.htm?pat=a[cd]?&txt=a,c,d,ac,ad) |
| + | 表达式至少出现1次，相当于 {1,}，比如：["a+b"可以匹配 "ab","aab","aaab"...](http://www.regexlab.com/zh/workshop.htm?pat=a+b&txt=a,b,ab,aab,aaab) |
| \* | 表达式不出现或出现任意次，相当于 {0,}，比如：["\^\*b"可以匹配 "b","^^^b"...](http://www.regexlab.com/zh/workshop.htm?pat=\\^*b&txt=^,b,^^^b,^^^^^^^b) |

[举例1：表达式 "\d+\.?\d\*" 在匹配 "It costs $12.5" 时](http://www.regexlab.com/zh/workshop.htm?pat=\\d+\\.?\\d*&txt=It costs $12.5)，匹配的结果是：成功；匹配到的内容是："12.5"；匹配到的位置是：开始于10，结束于14。  
  
    [举例2：表达式 "go{2,8}gle" 在匹配 "Ads by goooooogle" 时](http://www.regexlab.com/zh/workshop.htm?pat=go{2,8}gle&txt=Ads by goooooogle, or gooogle)，匹配的结果是：成功；匹配到的内容是："goooooogle"；匹配到的位置是：开始于7，结束于17。



##### 1.6 其他一些代表抽象意义的特殊符号

    一些符号在表达式中代表抽象的特殊意义：

|  |  |
| --- | --- |
| 表达式 | 作用 |
| ^ | 与字符串开始的地方匹配，不匹配任何字符 |
| $ | 与字符串结束的地方匹配，不匹配任何字符 |
| \b | 匹配一个单词边界，也就是单词和空格之间的位置，不匹配任何字符 |

    进一步的文字说明仍然比较抽象，因此，举例帮助大家理解。  
  
    [举例1：表达式 "^aaa" 在匹配 "xxx aaa xxx" 时](http://www.regexlab.com/zh/workshop.htm?pat=^aaa&txt=xxx+aaa+xxx)，匹配结果是：失败。因为 "^" 要求与字符串开始的地方匹配，因此，只有当 "aaa" 位于字符串的开头的时候，"^aaa" 才能匹配，[比如："aaa xxx xxx"](http://www.regexlab.com/zh/workshop.htm?pat=^aaa&txt=aaa+xxx+xxx)。  
  
    [举例2：表达式 "aaa$" 在匹配 "xxx aaa xxx" 时](http://www.regexlab.com/zh/workshop.htm?pat=aaa$&txt=xxx+aaa+xxx)，匹配结果是：失败。因为 "$" 要求与字符串结束的地方匹配，因此，只有当 "aaa" 位于字符串的结尾的时候，"aaa$" 才能匹配，[比如："xxx xxx aaa"](http://www.regexlab.com/zh/workshop.htm?pat=aaa$&txt=xxx+xxx+aaa)。  
  
    [举例3：表达式 ".\b." 在匹配 "@@@abc" 时](http://www.regexlab.com/zh/workshop.htm?pat=.\\b.&txt=@@@abc)，匹配结果是：成功；匹配到的内容是："@a"；匹配到的位置是：开始于2，结束于4。  
    进一步说明："\b" 与 "^" 和 "$" 类似，本身不匹配任何字符，但是它要求它在匹配结果中所处位置的左右两边，其中一边是 "\w" 范围，另一边是 非"\w" 的范围。  
  
    [举例4：表达式 "\bend\b" 在匹配 "weekend,endfor,end" 时](http://www.regexlab.com/zh/workshop.htm?pat=\\bend\\b&txt=weekend,endfor,end)，匹配结果是：成功；匹配到的内容是："end"；匹配到的位置是：开始于15，结束于18。



    一些符号可以影响表达式内部的子表达式之间的关系：

|  |  |
| --- | --- |
| 表达式 | 作用 |
| | | 左右两边表达式之间 "或" 关系，匹配左边或者右边 |
| ( ) | (1). 在被修饰匹配次数的时候，括号中的表达式可以作为整体被修饰 (2). 取匹配结果的时候，括号中的表达式匹配到的内容可以被单独得到 |

[举例5：表达式 "Tom|Jack" 在匹配字符串 "I'm Tom, he is Jack" 时](http://www.regexlab.com/zh/workshop.htm?pat=Tom|Jack&txt=I'm+Tom,+he+is+Jack)，匹配结果是：成功；匹配到的内容是："Tom"；匹配到的位置是：开始于4，结束于7。匹配下一个时，匹配结果是：成功；匹配到的内容是："Jack"；匹配到的位置时：开始于15，结束于19。  
  
    [举例6：表达式 "(go\s\*)+" 在匹配 "Let's go go go!" 时](http://www.regexlab.com/zh/workshop.htm?pat=(go\\s*)+&txt=Let's go go go!)，匹配结果是：成功；匹配到内容是："go go go"；匹配到的位置是：开始于6，结束于14。  
  
    [举例7：表达式 "￥(\d+\.?\d\*)" 在匹配 "＄10.9,￥20.5" 时](http://www.regexlab.com/zh/workshop.htm?pat=%uFFE5(\\d+\\.?\\d*)&txt=%uFF0410.9,%uFFE520.5)，匹配的结果是：成功；匹配到的内容是："￥20.5"；匹配到的位置是：开始于6，结束于10。单独获取括号范围匹配到的内容是："20.5"。



#### 2. 正则表达式中的一些高级规则

##### 2.1 匹配次数中的贪婪与非贪婪

    在使用修饰匹配次数的特殊符号时，有几种表示方法可以使同一个表达式能够匹配不同的次数，比如："{m,n}", "{m,}", "?", "\*", "+"，具体匹配的次数随被匹配的字符串而定。这种重复匹配不定次数的表达式在匹配过程中，总是尽可能多的匹配。比如，针对文本 "dxxxdxxxd"，举例如下：

|  |  |
| --- | --- |
| 表达式 | 匹配结果 |
| [(d)(\w+)](http://www.regexlab.com/zh/workshop.htm?pat=(d)(\\w+)&txt=dxxxdxxxd) | "\w+" 将匹配第一个 "d" 之后的所有字符 "xxxdxxxd" |
| [(d)(\w+)(d)](http://www.regexlab.com/zh/workshop.htm?pat=(d)(\\w+)(d)&txt=dxxxdxxxd) | "\w+" 将匹配第一个 "d" 和最后一个 "d" 之间的所有字符 "xxxdxxx"。虽然 "\w+" 也能够匹配上最后一个 "d"，但是为了使整个表达式匹配成功，"\w+" 可以 "让出" 它本来能够匹配的最后一个 "d" |

    由此可见，"\w+" 在匹配的时候，总是尽可能多的匹配符合它规则的字符。虽然第二个举例中，它没有匹配最后一个 "d"，但那也是为了让整个表达式能够匹配成功。同理，带 "\*" 和 "{m,n}" 的表达式都是尽可能地多匹配，带 "?" 的表达式在可匹配可不匹配的时候，也是尽可能的 "要匹配"。这 种匹配原则就叫作 "贪婪" 模式 。

    非贪婪模式：  
  
    在修饰匹配次数的特殊符号后再加上一个 "?" 号，则可以使匹配次数不定的表达式尽可能少的匹配，使可匹配可不匹配的表达式，尽可能的 "不匹配"。这种匹配原则叫作 "非贪婪" 模式，也叫作 "勉强" 模式。如果少匹配就会导致整个表达式匹配失败的时候，与贪婪模式类似，非贪婪模式会最小限度的再匹配一些，以使整个表达式匹配成功。举例如下，针对文本 "dxxxdxxxd" 举例：

|  |  |
| --- | --- |
| 表达式 | 匹配结果 |
| [(d)(\w+?)](http://www.regexlab.com/zh/workshop.htm?pat=(d)(\\w+?)&txt=dxxxdxxxd) | "\w+?" 将尽可能少的匹配第一个 "d" 之后的字符，结果是："\w+?" 只匹配了一个 "x" |
| [(d)(\w+?)(d)](http://www.regexlab.com/zh/workshop.htm?pat=(d)(\\w+?)(d)&txt=dxxxdxxxd) | 为了让整个表达式匹配成功，"\w+?" 不得不匹配 "xxx" 才可以让后边的 "d" 匹配，从而使整个表达式匹配成功。因此，结果是："\w+?" 匹配 "xxx" |

    更多的情况，举例如下：  
  
    [举例1：表达式 "<td>(.\*)</td>" 与字符串 "<td><p>aa</p></td> <td><p>bb</p></td>" 匹配时](http://www.regexlab.com/zh/workshop.htm?pat=<td>(.*)</td>&txt=<td><p>aa</p></td><td><p>bb</p></td>)，匹配的结果是：成功；匹配到的内容是 "<td><p>aa</p></td> <td><p>bb</p></td>" 整个字符串， 表达式中的 "</td>" 将与字符串中最后一个 "</td>" 匹配。   
  
    [举例2：相比之下，表达式 "<td>(.\*?)</td>" 匹配举例1中同样的字符串时](http://www.regexlab.com/zh/workshop.htm?pat=<td>(.*?)</td>&txt=<td><p>aa</p></td><td><p>bb</p></td>)，将只得到 "<td><p>aa</p></td>"， 再次匹配下一个时，可以得到第二个 "<td><p>bb</p></td>"。



##### 2.2 反向引用 \1, \2...

    表达式在匹配时，表达式引擎会将小括号 "( )" 包含的表达式所匹配到的字符串记录下来。在获取匹配结果的时候，小括号包含的表达式所匹配到的字符串可以单独获取。这一点，在前面的举例中，已经多次展示了。在实际应用场合中，当用某种边界来查找，而所要获取的内容又不包含边界时，必须使用小括号来指定所要的范围。比如前面的 "<td>(.\*?)</td>"。  
  
    其实，"小括号包含的表达式所匹配到的字符串" 不仅是在匹配结束后才可以使用，在匹配过程中也可以使用。表达式后边的部分，可以引用前面 "括号内的子匹配已经匹配到的字符串"。引用方法是 "\" 加上一个数字。"\1" 引用第1对括号内匹配到的字符串，"\2" 引用第2对括号内匹配到的字符串……以此类推，如果一对括号内包含另一对括号，则外层的括号先排序号。换句话说，哪一对的左括号 "(" 在前，那这一对就先排序号。

    举例如下：  
  
    [举例1：表达式 "('|")(.\*?)(\1)" 在匹配 " 'Hello', "World" " 时](http://www.regexlab.com/zh/workshop.htm?pat=('|")(.*?)(\\1)&txt='Hello',+"World")，匹配结果是：成功；匹配到的内容是：" 'Hello' "。再次匹配下一个时，可以匹配到 " "World" "。  
  
    [举例2：表达式 "(\w)\1{4,}" 在匹配 "aa bbbb abcdefg ccccc 111121111 999999999" 时](http://www.regexlab.com/zh/workshop.htm?pat=(\\w)\\1{4,}&txt=aa bbbb abcdefg ccccc 111121111 999999999)，匹配结果是：成功；匹配到的内容是 "ccccc"。再次匹配下一个时，将得到 999999999。这个表达式要求 "\w" 范围的字符至少重复5次，[注意与 "\w{5,}" 之间的区别](http://www.regexlab.com/zh/workshop.htm?pat=\\w{5,}&txt=aa bbbb abcdefg ccccc 111121111 999999999)。  
  
    [举例3：表达式 "<(\w+)\s\*(\w+(=('|").\*?\4)?\s\*)\*>.\*?</\1>" 在匹配 "<td id='td1' style="bgcolor:white"></td>" 时](http://www.regexlab.com/zh/workshop.htm?pat=<(\\w+)\\s*(\\w+(=('|").*?\\4)?\\s*)*>.*?</\\1>&txt=<td+id='td1'+style="bgcolor:white"></td>%0D%0A<body+onload="doit()"></body>)，匹配结果是成功。如果 "<td>" 与 "</td>" 不配对，则会匹配失败；如果改成其他配对，也可以匹配成功。



##### 2.3 预搜索，不匹配；反向预搜索，不匹配

    前面的章节中，我讲到了几个代表抽象意义的特殊符号："^"，"$"，"\b"。它们都有一个共同点，那就是：它们本身不匹配任何字符，只是对 "字符串的两头" 或者 "字符之间的缝隙" 附加了一个条件。理解到这个概念以后，本节将继续介绍另外一种对 "两头" 或者 "缝隙" 附加条件的，更加灵活的表示方法。

    正向预搜索："(?=xxxxx)"，"(?!xxxxx)"  
  
    格式："(?=xxxxx)"，在被匹配的字符串中，它对所处的 "缝隙" 或者 "两头" 附加的条件是：所在缝隙的右侧，必须能够匹配上 xxxxx 这部分的表达式。因为它只是在此作为这个缝隙上附加的条件，所以它并不影响后边的表达式去真正匹配这个缝隙之后的字符。这就类似 "\b"，本身不匹配任何字符。"\b" 只是将所在缝隙之前、之后的字符取来进行了一下判断，不会影响后边的表达式来真正的匹配。  
  
    [举例1：表达式 "Windows (?=NT|XP)" 在匹配 "Windows 98, Windows NT, Windows 2000" 时](http://www.regexlab.com/zh/workshop.htm?pat=Windows+(?=NT|XP)&txt=Windows+98,+Windows+NT,+Windows+2000)，将只匹配 "Windows NT" 中的 "Windows "，其他的 "Windows " 字样则不被匹配。  
  
    [举例2：表达式 "(\w)((?=\1\1\1)(\1))+" 在匹配字符串 "aaa ffffff 999999999" 时](http://www.regexlab.com/zh/workshop.htm?pat=(\\w)((?=\\1\\1\\1)(\\1))+&txt=aaa+ffffff+999999999)，将可以匹配6个"f"的前4个，可以匹配9个"9"的前7个。这个表达式可以读解成：重复4次以上的字母数字，则匹配其剩下最后2位之前的部分。当然，这个表达式可以不这样写，在此的目的是作为演示之用。



    格式："(?!xxxxx)"，所在缝隙的右侧，必须不能匹配 xxxxx 这部分表达式。  
  
    [举例3：表达式 "((?!\bstop\b).)+" 在匹配 "fdjka ljfdl stop fjdsla fdj" 时](http://www.regexlab.com/zh/workshop.htm?pat=((?!\\bstop\\b).)+&txt=fdjka+ljfdl+stop+fjdsla+fdj)，将从头一直匹配到 "stop" 之前的位置，如果字符串中没有 "stop"，则匹配整个字符串。  
  
    [举例4：表达式 "do(?!\w)" 在匹配字符串 "done, do, dog" 时](http://www.regexlab.com/zh/workshop.htm?pat=do(?!\\w)&txt=done,+do,+dog)，只能匹配 "do"。在本条举例中，"do" 后边使用 "(?!\w)" 和使用 "\b" 效果是一样的。



    反向预搜索："(?<=xxxxx)"，"(?<!xxxxx)"  
  
    这两种格式的概念和正向预搜索是类似的，反向预搜索要求的条件是：所在缝隙的 "左侧"，两种格式分别要求必须能够匹配和必须不能够匹配指定表达式，而不是去判断右侧。与 "正向预搜索" 一样的是：它们都是对所在缝隙的一种附加条件，本身都不匹配任何字符。  
  
    举例5：表达式 "(?<=\d{4})\d+(?=\d{4})" 在匹配 "1234567890123456" 时，将匹配除了前4个数字和后4个数字之外的中间8个数字。由于 JScript.RegExp 不支持反向预搜索，因此，本条举例不能够进行演示。很多其他的引擎可以支持反向预搜索，比如：Java 1.4 以上的 java.util.regex 包，.NET 中System.Text.RegularExpressions 命名空间，以及本站推荐的[最简单易用的 DEELX 正则引擎](http://www.regexlab.com/zh/deelx/)。

#### 3. 其他通用规则

    还有一些在各个正则表达式引擎之间比较通用的规则，在前面的讲解过程中没有提到。

3.1 表达式中，可以使用 "\xXX" 和 "\uXXXX" 表示一个字符（"X" 表示一个十六进制数）

|  |  |
| --- | --- |
| 形式 | 字符范围 |
| \xXX | 编号在 0 ~ 255 范围的字符，比如：[空格可以使用 "\x20" 表示](http://www.regexlab.com/zh/workshop.htm?pat=\\x20&txt=It+is.) |
| \uXXXX | 任何字符可以使用 "\u" 再加上其编号的4位十六进制数表示，比如：["\u4E2D"](http://www.regexlab.com/zh/workshop.htm?pat=\\u4E2D&txt=%D6%D0%B9%FA) |

3.2 在表达式 "\s"，"\d"，"\w"，"\b" 表示特殊意义的同时，对应的大写字母表示相反的意义

|  |  |
| --- | --- |
| 表达式 | 可匹配 |
| \S | [匹配所有非空白字符（"\s" 可匹配各个空白字符）](http://www.regexlab.com/zh/workshop.htm?pat=\\S+&txt=abc+123+@%23$%) |
| \D | [匹配所有的非数字字符](http://www.regexlab.com/zh/workshop.htm?pat=\\D+&txt=abc+123+@%23$%) |
| \W | [匹配所有的字母、数字、下划线以外的字符](http://www.regexlab.com/zh/workshop.htm?pat=\\W+&txt=abc+123+@%23$%) |
| \B | [匹配非单词边界，即左右两边都是 "\w" 范围或者左右两边都不是 "\w" 范围时的字符缝隙](http://www.regexlab.com/zh/workshop.htm?pat=\\B.\\B&txt=abc+123+@%23$%) |

3.3 在表达式中有特殊意义，需要添加 "\" 才能匹配该字符本身的字符汇总

|  |  |
| --- | --- |
| 字符 | 说明 |
| ^ | 匹配输入字符串的开始位置。要匹配 "^" 字符本身，请使用 "\^" |
| $ | 匹配输入字符串的结尾位置。要匹配 "$" 字符本身，请使用 "\$" |
| ( ) | 标记一个子表达式的开始和结束位置。要匹配小括号，请使用 "\(" 和 "\)" |
| [ ] | 用来自定义能够匹配 '多种字符' 的表达式。要匹配中括号，请使用 "\[" 和 "\]" |
| { } | 修饰匹配次数的符号。要匹配大括号，请使用 "\{" 和 "\}" |
| . | 匹配除了换行符（\n）以外的任意一个字符。要匹配小数点本身，请使用 "\." |
| ? | 修饰匹配次数为 0 次或 1 次。要匹配 "?" 字符本身，请使用 "\?" |
| + | 修饰匹配次数为至少 1 次。要匹配 "+" 字符本身，请使用 "\+" |
| \* | 修饰匹配次数为 0 次或任意次。要匹配 "\*" 字符本身，请使用 "\\*" |
| | | 左右两边表达式之间 "或" 关系。匹配 "|" 本身，请使用 "\|" |

3.4 括号 "( )" 内的子表达式，如果希望匹配结果不进行记录供以后使用，可以使用 "(?:xxxxx)" 格式

[举例1：表达式 "(?:(\w)\1)+" 匹配 "a bbccdd efg" 时](http://www.regexlab.com/zh/workshop.htm?pat=(?:(\\w)\\1)+&txt=a bbccdd efg)，结果是 "bbccdd"。括号 "(?:)" 范围的匹配结果不进行记录，因此 "(\w)" 使用 "\1" 来引用。



3.5 常用的表达式属性设置简介：Ignorecase，Singleline，Multiline，Global

|  |  |
| --- | --- |
| 表达式属性 | 说明 |
| Ignorecase | 默认情况下，表达式中的字母是要区分大小写的。配置为 Ignorecase 可使匹配时不区分大小写。有的表达式引擎，把 "大小写" 概念延伸至 UNICODE 范围的大小写。 |
| Singleline | 默认情况下，小数点 "." 匹配除了换行符（\n）以外的字符。配置为 Singleline 可使小数点可匹配包括换行符在内的所有字符。 |
| Multiline | 默认情况下，表达式 "^" 和 "$" 只匹配字符串的开始 ① 和结尾 ④ 位置。如：  ①xxxxxxxxx②\n ③xxxxxxxxx④  配置为 Multiline 可以使 "^" 匹配 ① 外，还可以匹配换行符之后，下一行开始前 ③ 的位置，使 "$" 匹配 ④ 外，还可以匹配换行符之前，一行结束 ② 的位置。 |
| Global | 主要在将表达式用来替换时起作用，配置为 Global 表示替换所有的匹配。 |

#### 4. 其他提示

4.1 如果想要了解高级的正则引擎还支持那些复杂的正则语法，可参见[本站 DEELX 正则引擎的说明文档](http://www.regexlab.com/zh/deelx/syntax.htm)。

4.2 如果要要求表达式所匹配的内容是整个字符串，而不是从字符串中找一部分，那么可以在表达式的首尾使用 "^" 和 "$"，比如："^\d+$" 要求整个字符串只有数字。

4.3 如果要求匹配的内容是一个完整的单词，而不会是单词的一部分，那么在表达式首尾使用 "\b"，比如：[使用 "\b(if|while|else|void|int……)\b" 来匹配程序中的关键字](http://www.regexlab.com/zh/workshop.htm?pat=\\b(if|while|else|void|int)\\b&txt=if(ifdo)%0D%0A++++dosome();%0D%0Aelse%0D%0A++++doelse();)。



4.4 表达式不要匹配空字符串。否则会一直得到匹配成功，而结果什么都没有匹配到。比如：准备写一个匹配 "123"、"123."、"123.5"、".5" 这几种形式的表达式时，整数、小数点、小数数字都可以省略，但是不要将表达式写成："\d\*\.?\d\*"，因为如果什么都没有，这个表达式也可以匹配成功。[更好的写法是："\d+\.?\d\*|\.\d+"](http://www.regexlab.com/zh/workshop.htm?pat=\\d+\\.?\\d*|\\.\\d+&txt=123,+123.,+123.5,+.5,+.)。



4.5 能匹配空字符串的子匹配不要循环无限次。如果括号内的子表达式中的每一部分都可以匹配 0 次，而这个括号整体又可以匹配无限次，那么情况可能比上一条所说的更严重，匹配过程中可能死循环。虽然现在有些正则表达式引擎已经通过办法避免了这种情况出现死循环了，比如 .NET 的正则表达式，但是我们仍然应该尽量避免出现这种情况。如果我们在写表达式时遇到了死循环，也可以从这一点入手，查找一下是否是本条所说的原因。

4.6 合理选择贪婪模式与非贪婪模式，参见[话题讨论](http://www.regexlab.com/zh/regtopic.htm" \l "reluctant)。

4.7 或 "|" 的左右两边，对某个字符最好只有一边可以匹配，这样，不会因为 "|" 两边的表达式因为交换位置而有所不同。

#### 5. 进阶与实战

5.1 **重要提示：**本站 正则工具 Regex Match Tracer 2.1 **（免费）**已可以免费无限制使用：

    [[免费下载 Match Tracer](http://www.regexlab.com/zh/mtracer/download.htm)] - 396kb



5.2 下载正则表达式文档 chm 版本

    [[点击下载 chm 版本](http://www.regexlab.com/download/?/deelx/deelx_zh.zip)] - DEELX 正则语法，包含其他高级语法的 chm 版本。



5.3 更多深入话题及使用案例

    [[关于递归匹配的讨论](http://www.regexlab.com/zh/regtopic.htm)] - 讨论如何使用不支持递归的正则引擎匹配嵌套结构



    [[本页脚本](http://www.regexlab.com/zh/javascript:alert('%E7%BB%86%E5%BF%83%E7%9A%84%E8%AF%BB%E8%80%85%EF%BC%8C%E5%8F%AF%E9%87%87%E7%94%A8%E5%8F%A6%E5%AD%98%E4%B8%BA%E7%9A%84%E6%96%B9%E5%BC%8F%EF%BC%8C%E5%8F%82%E8%80%83%E6%9C%AC%E9%A1%B5%E4%BD%BF%E7%94%A8%E7%9A%84%E8%84%9A%E6%9C%AC%E3%80%82'))] - 本页的“[关闭高亮](http://www.regexlab.com/zh/javascript:colorize()" \o "采用同一种颜色显示表达式)”功能，采用 javascript 的正则表达式实现的。  
  
        比如表达式： **(a+b|[cd])$**



# 语句

## if语句

if的基本格式

if expression:

statement(s)

控制流的条件表达式(expression)结果，必须为True真

冒号永不忘。尽量使用4个空格，而不是制表符

做条件判断,可以用if语句实现：

age = 20

if age >= 18:

print 'your age is', age

print 'adult'

print 'END'

**注意:**Python代码的缩进规则。具有相同缩进的代码被视为代码块，上面的3，4行 print 语句就构成一个代码块（但不包括第5行的print）。如果 if 语句判断为 True，就会执行这个代码块。

缩进请严格按照Python的习惯写法：4个空格，不要使用Tab，更不要混合Tab和空格，否则很容易造成因为缩进引起的语法错误。

**注意**: if 语句后接表达式，然后用:表示代码块开始。

如果你在Python交互环境下敲代码，还要特别留意缩进，并且退出缩进需要多敲一行回车：

## if ... else ...

条件判断是“非此即彼”的，要么符合条件1，要么符合条件2，因此，完全可以用一个 if ... else ... 语句把它们统一起来：

if age >= 18:

print 'adult'

else:

print 'teenager'

利用 if ... else ... 语句，我们可以根据条件表达式的值为 True 或者 False ，分别执行 if 代码块或者 else 代码块。

**注意:** else 后面有个“:”。

## if-elif-else

有的时候，一个 if ... else ... 还不够用。比如，根据年龄的划分：

条件1：18岁或以上：adult

条件2：6岁或以上：teenager

条件3：6岁以下：kid

我们可以用一个 if age >= 18 判断是否符合条件1，如果不符合，再通过一个 if 判断 age >= 6 来判断是否符合条件2，否则，执行条件3：

if age >= 18:

print 'adult'

else:

if age >= 6:

print 'teenager'

else:

print 'kid'

**特别注意:**这一系列条件判断会从上到下依次判断，如果某个判断为 True，执行完对应的代码块，后面的条件判断就直接忽略，不再执行了。

## For

Python的 for 循环就可以依次把list或tuple的每个元素迭代出来：

L = ['Adam', 'Lisa', 'Bart']

for name in L:

print name

**注意:** name 这个变量是在 for 循环中定义的，意思是，依次取出list中的每一个元素，并把元素赋值给 name，然后执行for循环体（就是缩进的代码块）。

这样一来，遍历一个list或tuple就非常容易了

#### For else

for y in 'I love zhang shuo ,he is my sun.'.split(): #split()去掉空格，‘，’逗号连续输出

print y, #y在执行完后保存的现在是sun这个字符串

else:

print 'zhang'

for的最后一个迭代值将保留

## while循环

和 for 循环不同的另一种循环是 while 循环，while 循环不会迭代 list 或 tuple 的元素，而是根据表达式判断循环是否结束。

比如要从 0 开始打印不大于 N 的整数：

N = 10

x = 0

while x < N:

print x

x = x + 1

while循环每次先判断 x < N，如果为True，则执行循环体的代码块，否则，退出循环。

在循环体内，x = x + 1 会让 x 不断增加，最终因为 x < N 不成立而退出循环。

如果没有这一个语句，**while循环在判断 x < N 时总是为True**，就会无限循环下去，变成死循环，所以要特别留意while循环的退出条件。

## break退出循环

用 for 循环或者 while 循环时，如果要在循环体内**直接退出循环**，可以使用 break 语句。

比如计算1至100的整数和，我们用while来实现：

sum = 0

x = 1

while True:

sum = sum + x

x = x + 1

if x > 100:

break

print sum

咋一看， while True 就是一个死循环，但是在循环体内，我们还判断了 x > 100 条件成立时，用break语句退出循环，这样也可以实现循环的结束。

## continue继续循环

可以用break退出当前循环，还可以用continue跳过后续循环代码，继续下一次循环。

假设我们已经写好了利用for循环计算平均分的代码：

L = [75, 98, 59, 81, 66, 43, 69, 85]

sum = 0.0

n = 0

for x in L:

sum = sum + x

n = n + 1

print sum / n

现在老师只想统计及格分数的平均分，就要把 x < 60 的分数剔除掉，这时，利用 continue，可以做到当 x < 60的时候，不继续执行循环体的后续代码，直接进入下一次循环：

for x in L:

if x < 60:

continue

sum = sum + x

n = n + 1

## 多重循环

在循环内部，还可以嵌套循环，我们来看一个例子：

for x in ['A', 'B', 'C']:

for y in ['1', '2', '3']:

print x + y

x 每循环一次，y 就会循环 3 次，这样，我们可以打印出一个全排列：

A1  
A2  
A3  
B1  
B2  
B3  
C1  
C2  
C3

## 输入

现在，你已经可以用print输出你想要的结果了。但是，如果要让用户从电脑输入一些字符怎么办？Python提供了一个raw\_input，可以让用户输入字符串，并存放到一个变量里。比如输入用户的名字：

>>> name = raw\_input()

Michael

当你输入name = raw\_input()并按下回车后，Python交互式命令行就在等待你的输入了。这时，你可以输入任意字符，然后按回车后完成输入。

输入完成后，不会有任何提示，Python交互式命令行又回到>>>状态了。那我们刚才输入的内容到哪去了？答案是存放到name变量里了。可以直接输入name查看变量内容：

>>> name

'Michael'

# 函数

## 参数

### \*\*kwargs 是字典；

它是一个可变，可选的参数

def a(\*\*kr):

return kr

print a(a=2,b=3,c='e',d='w')

>>>{'a': 2, 'c': 'e', 'b': 3, 'd': 'w'}

### \*kargs是元组

各种数据格式都是元组

def b(\*kr): #\*kr 是一个元组tuple

return kr

print b(1,2,'d')

>>>(1, 2, 'd')

### 混合方式

参数从左向右是有顺序的，不能混淆，重名

\*和\*\*

def test(\*k,\*\*kr):

return k,kr

print test(1,2,3,4,'ds',a=1,b=2)

>>>((1, 2, 3, 4, 'ds'), {'a': 1, 'b': 2})

多个参数类型

def test(c,w,\*k,\*\*kr):

return k,kr

print test(1,2,3,4,'ds',a=1,b=2)

>>>((3, 4, 'ds'), {'a': 1, 'b': 2})

### 定义默认参数 def power(x, n=2)

定义函数的时候，还可以有默认参数。

例如Python自带的 **int()** 函数，其实就有两个参数，我们既可以传一个参数，又可以传两个参数：

>>> int('123')

123

>>> int('123', 8)

83

int()函数的第二个参数是转换进制，如果不传，默认是十进制 (base=10)，如果传了，就用传入的参数。

可见，**函数的默认参数的作用是简化调用**，你只需要把必须的参数传进去。但是在需要的时候，又可以传入额外的参数来覆盖默认参数值。

我们来定义一个计算 x 的N次方的函数:

def power(x, n):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

假设计算平方的次数最多，我们就可以把 n 的默认值设定为 2：

def power(x, n=2):

s = 1

while n > 0:

n = n - 1

s = s \* x

return s

这样一来，计算平方就不需要传入两个参数了：

>>> power(5)

25

由于函数的参数按从左到右的顺序匹配，所以**默认参数只能定义在必需参数的后面：**

# OK:

def fn1(a, b=1, c=2):

pass

# Error:

def fn2(a=1, b):

pass

### 定义可变参数 \*args

如果想让一个函数能接受任意个参数，我们就可以定义一个可变参数：

def fn(\*args):

print args

可变参数的名字前面有个 **\***号，我们可以传入0个、1个或多个参数给可变参数：

>>> fn()

()

>>> fn('a')

('a',)

>>> fn('a', 'b')

('a', 'b')

>>> fn('a', 'b', 'c')

('a', 'b', 'c')

可变参数也不是很神秘，Python解释器会把传入的一组参数组装成一个tuple传递给可变参数，因此，在函数内部，直接把变量 args 看成一个 tuple 就好了。

定义可变参数的目的也是为了简化调用。假设我们要计算任意个数的平均值，就可以定义一个可变参数：

def average(\*args):

...

这样，在调用的时候，可以这样写：

>>> average()

0

>>> average(1, 2)

1.5

>>> average(1, 2, 2, 3, 4)

2.4

函数写文档在def（）：下一行用‘’或“”其中写入说明

## 初等函数

最近一直在看python的document，打算在基础方面重点看一下python的keyword、Build-in Function、Build-in Constants、Build-in Types、Build-in Exception这四个方面，其实在看的时候发现整个《The Python Standard Library》章节都是很不错的，其中描述了很多不错的主题。先把Build-in Function罗列一下吧，初学者的了解，分类可能不准确，一起交流。

### 一、数学运算类

|  |  |
| --- | --- |
| abs(x) | 求绝对值 1、参数可以是整型，也可以是复数 2、若参数是复数，则返回复数的模 |
| complex([real[, imag]]) | 创建一个复数 |
| divmod(a, b) | 分别取商和余数 注意：整型、浮点型都可以 |
| float([x]) | 将一个字符串或数转换为浮点数。如果无参数将返回0.0 |
| int([x[, base]]) | 将一个字符转换为int类型，base表示进制 |
| long([x[, base]]) | 将一个字符转换为long类型 |
| pow(x, y[, z]) | 返回x的y次幂 |
| range([start], stop[, step]) | 产生一个序列，默认从0开始 |
| round(x[, n]) | 四舍五入 |
| sum(iterable[, start]) | 对集合求和 |
| oct(x) | 将一个数字转化为8进制 |
| hex(x) | 将整数x转换为16进制字符串 |
| chr(i) | 返回整数i对应的ASCII字符 |
| bin(x) | 将整数x转换为二进制字符串 |
| bool([x]) | 将x转换为Boolean类型 |

### 二、集合类操作

|  |  |
| --- | --- |
| basestring() | str和unicode的超类 不能直接调用，可以用作isinstance判断 |
| format(value [, format\_spec]) | 格式化输出字符串 格式化的参数顺序从0开始，如“I am {0},I like {1}” |
| unichr(i) | 返回给定int类型的unicode |
| enumerate(sequence [, start = 0]) | 返回一个可枚举的对象,该对象的next()方法将返回一个tuple |
| iter(o[, sentinel]) | 生成一个对象的迭代器，第二个参数表示分隔符 |
| max(iterable[, args...][key]) | 返回集合中的最大值 |
| min(iterable[, args...][key]) | 返回集合中的最小值 |
| dict([arg]) | 创建数据字典 |
| list([iterable]) | 将一个集合类转换为另外一个集合类 |
| set() | set对象实例化 |
| frozenset([iterable]) | 产生一个不可变的set |
| str([object]) | 转换为string类型 |
| sorted(iterable[, cmp[, key[, reverse]]]) | 队集合排序 |
| tuple([iterable]) | 生成一个tuple类型 |
| xrange([start], stop[, step]) | xrange()函数与range()类似，但xrnage()并不创建列表，而是返回一个xrange对象，它的行为与列表相似，但是只在需要时才计算列表值，当列表很大时，这个特性能为我们节省内存 |

### 三、逻辑判断

|  |  |
| --- | --- |
| all(iterable) | 1、集合中的元素都为真的时候为真 2、特别的，若为空串返回为True |
| any(iterable) | 1、集合中的元素有一个为真的时候为真 2、特别的，若为空串返回为False |
| cmp(x, y) | 如果x < y ,返回负数；x == y, 返回0；x > y,返回正数 |

### 四、反射

|  |  |
| --- | --- |
| callable(object) | 检查对象object是否可调用 1、类是可以被调用的 2、实例是不可以被调用的，除非类中声明了\_\_call\_\_方法 |
| classmethod() | 1、注解，用来说明这个方式是个类方法 2、类方法即可被类调用，也可以被实例调用 3、类方法类似于Java中的static方法 4、类方法中不需要有self参数 |
| compile(source, filename, mode[, flags[, dont\_inherit]]) | 将source编译为代码或者AST对象。代码对象能够通过exec语句来执行或者eval()进行求值。 1、参数source：字符串或者AST（Abstract Syntax Trees）对象。 2、参数 filename：代码文件名称，如果不是从文件读取代码则传递一些可辨认的值。 3、参数model：指定编译代码的种类。可以指定为 ‘exec’,’eval’,’single’。 4、参数flag和dont\_inherit：这两个参数暂不介绍 |
| dir([object]) | 1、不带参数时，返回当前范围内的变量、方法和定义的类型列表； 2、带参数时，返回参数的属性、方法列表。 3、如果参数包含方法\_\_dir\_\_()，该方法将被调用。当参数为实例时。 4、如果参数不包含\_\_dir\_\_()，该方法将最大限度地收集参数信息 |
| delattr(object, name) | 删除object对象名为name的属性 |
| eval(expression [, globals [, locals]]) | 计算表达式expression的值 |
| execfile(filename [, globals [, locals]]) | 用法类似exec()，不同的是execfile的参数filename为文件名，而exec的参数为字符串。 |
| filter(function, iterable) | 构造一个序列，等价于[ item for item in iterable if function(item)] 1、参数function：返回值为True或False的函数，可以为None 2、参数iterable：序列或可迭代对象 |
| getattr(object, name [, defalut]) | 获取一个类的属性 |
| globals() | 返回一个描述当前全局符号表的字典 |
| hasattr(object, name) | 判断对象object是否包含名为name的特性 |
| hash(object) | 如果对象object为哈希表类型，返回对象object的哈希值 |
| id(object) | 返回对象的唯一标识 |
| isinstance(object, classinfo) | 判断object是否是class的实例 |
| issubclass(class, classinfo) | 判断是否是子类 |
| len(s) | 返回集合长度 |
| locals() | 返回当前的变量列表 |
| map(function, iterable, ...) | 遍历每个元素，执行function操作 |
| memoryview(obj) | 返回一个内存镜像类型的对象 |
| next(iterator[, default]) | 类似于iterator.next() |
| object() | 基类 |
| property([fget[, fset[, fdel[, doc]]]]) | 属性访问的包装类，设置后可以通过c.x=value等来访问setter和getter |
| reduce(function, iterable[, initializer]) | 合并操作，从第一个开始是前两个参数，然后是前两个的结果与第三个合并进行处理，以此类推 |
| reload(module) | 重新加载模块 |
| setattr(object, name, value) | 设置属性值 |
| repr(object) | 将一个对象变幻为可打印的格式 |
| slice（） |  |
| staticmethod | 声明静态方法，是个注解 |
| super(type[, object-or-type]) | 引用父类 |
| type(object) | 返回该object的类型 |
| vars([object]) | 返回对象的变量，若无参数与dict()方法类似 |
| bytearray([source [, encoding [, errors]]]) | 返回一个byte数组 1、如果source为整数，则返回一个长度为source的初始化数组； 2、如果source为字符串，则按照指定的encoding将字符串转换为字节序列； 3、如果source为可迭代类型，则元素必须为[0 ,255]中的整数； 4、如果source为与buffer接口一致的对象，则此对象也可以被用于初始化bytearray. |
| zip([iterable, ...]) | 实在是没有看懂，只是看到了矩阵的变幻方面 |

### 五、IO操作

|  |  |
| --- | --- |
| file(filename [, mode [, bufsize]]) | file类型的构造函数，作用为打开一个文件，如果文件不存在且mode为写或追加时，文件将被创建。添加‘b’到mode参数中，将对文件以二进制形式操作。添加‘+’到mode参数中，将允许对文件同时进行读写操作 1、参数filename：文件名称。 2、参数mode：'r'（读）、'w'（写）、'a'（追加）。 3、参数bufsize：如果为0表示不进行缓冲，如果为1表示进行行缓冲，如果是一个大于1的数表示缓冲区的大小 。 |
| input([prompt]) | 获取用户输入 推荐使用raw\_input，因为该函数将不会捕获用户的错误输入 |
| open(name[, mode[, buffering]]) | 打开文件 与file有什么不同？推荐使用open |
| print | 打印函数 |
| raw\_input([prompt]) | 设置输入，输入都是作为字符串处理 |

### 六、其他

help()--帮助信息

\_\_import\_\_()--没太看明白了，看到了那句“Direct use of \_\_import\_\_() is rare”之后就没心看下去了

内置函数，一般都是因为使用频率比较频繁或是是元操作，所以通过内置函数的形式提供出来，通过对python的内置函数分类分析可以看出来：基本的数据操作基本都是一些数学运算（当然除了加减乘除）、逻辑操作、集合操作、基本IO操作，然后就是对于语言自身的反射操作，还有就是字符串操作，也是比较常用的，尤其需要注意的是反射操作。

## 函数学习

### isinstance

1. isinstance
3. isinstance(object, classinfo)
4. 判断实例是否是这个类或者object是变量
6. classinfo 是类型(tuple,dict,int,float)
7. 判断变量是否是这个类型
9. class objA:
10. pass
12. A = objA()
13. B = 'a','v'
14. C = 'a string'
16. print isinstance(A, objA)
17. print isinstance(B, tuple)
18. print isinstance(C, basestring)
19. 输出结果：
20. True
21. True
22. True

25. 不仅如此，还可以利用isinstance函数，来判断一个对象是否是一个已知的类型。
26. isinstance说明如下:
27. isinstance(object, class-or-type-or-tuple) -> bool
29. Return whether an object is an instance of a class or of a subclass thereof.
30. With a type as second argument, return whether that is the object's type.
31. The form using a tuple, isinstance(x, (A, B, ...)), is a shortcut for
32. isinstance(x, A) or isinstance(x, B) or ... (etc.).
34. 其第一个参数为对象，第二个为类型名或类型名的一个列表。其返回值为布尔型。若对象的类型与参数二的类型相同则返回True。若参数二为一个元组，则若对象类型与元组中类型名之一相同即返回True。
36. >>>isinstance(lst, list)
37. True
39. >>>isinstance(lst, (int, str, list) )
40. True
41. 另外:Python可以得到一个对象的类型 ，利用type函数：>>>lst = [1, 2, 3]>>>type(lst)<type 'list'>

### callable：判断对象是否可以调用

我们需要判断python中的对象是否可以调用，可以用内置方法python callable()

如果对象可以调用的，则返回true,如果对象不能调用，则返回false.

*#encoding=utf-8*  
  
**def** info():  
  
   a = 1  
  
   **return** a  
  
  
flag = callable(info)  
  
**print** flag  
  
sflag = callable(str.lower)  
  
**print** sflag  
  
  
tflag = callable(2)  
  
**print** tflag

 返回的结果：true,true,false

我们就可以看出哪些是可以调用的，哪些不可以调用。

## 编写函数

在Python中，定义一个函数要使用**def**语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用 return语句返回。

我们以自定义一个求绝对值的 my\_abs 函数为例：

def my\_abs(x):

if x >= 0:

return x

else:

return -x

**请注意**，函数体内部的语句在执行时，一旦执行到return时，函数就执行完毕，并将结果返回。因此，函数内部通过条件判断和循环可以实现非常复杂的逻辑。

如果没有return语句，函数执行完毕后也会返回结果，只是结果为 None。

return None可以简写为return。

## 返回多值

函数可以返回多个值吗？答案是肯定的。

比如在游戏中经常需要从一个点移动到另一个点，给出坐标、位移和角度，就可以计算出新的坐标：

**# math**包提供了**sin()**和 **cos()**函数，我们先用import引用它：

import math

def move(x, y, step, angle):

nx = x + step \* math.cos(angle)

ny = y - step \* math.sin(angle)

return nx, ny

这样我们就可以同时获得返回值：

>>> x, y = move(100, 100, 60, math.pi / 6)

>>> print x, y

151.961524227 70.0

但其实这只是一种假象，Python函数返回的仍然是单一值：

>>> r = move(100, 100, 60, math.pi / 6)

>>> print r

(151.96152422706632, 70.0)

用print打印返回结果，原来返回值是一个**tuple**！

但是，在语法上，返回一个tuple可以省略括号，而多个变量可以同时接收一个tuple，按位置赋给对应的值，所以，**Python的函数**返回多值其实就是**返回一个tuple**，但写起来更方便。

## 递归函数

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。

举个例子，我们来计算阶乘 **n! = 1 \* 2 \* 3 \* ... \* n**，用函数 **fact(n)**表示，可以看出：

fact(n) = n! = 1 \* 2 \* 3 \* ... \* (n-1) \* n = (n-1)! \* n = fact(n-1) \* n

所以，**fact(n)**可以表示为 **n \* fact(n-1)**，只有n=1时需要特殊处理。

于是，fact(n)用递归的方式写出来就是：

def fact(n):

if n==1:

return 1

return n \* fact(n - 1)

如果我们计算fact(5)，可以根据函数定义看到计算过程如下：

===> fact(5)

===> 5 \* fact(4)

===> 5 \* (4 \* fact(3))

===> 5 \* (4 \* (3 \* fact(2)))

===> 5 \* (4 \* (3 \* (2 \* fact(1))))

===> 5 \* (4 \* (3 \* (2 \* 1)))

===> 5 \* (4 \* (3 \* 2))

===> 5 \* (4 \* 6)

===> 5 \* 24

===> 120

递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。可以试试计算 fact(10000)。

# 高阶函数

以函数为参数的函数

## 把函数作为参数

在2.1小节中，我们讲了高阶函数的概念，并编写了一个简单的高阶函数：

def add(x, y, f):

return f(x) + f(y)

如果传入abs作为参数f的值：

add(-5, 9, abs)

根据函数的定义，函数执行的代码实际上是：

abs(-5) + abs(9)

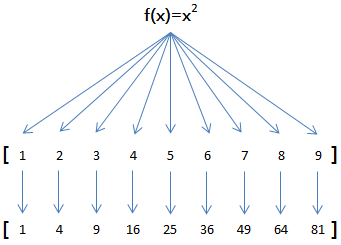
由于参数 x, y 和 f 都可以任意传入，如果 f 传入其他函数，就可以得到不同的返回值。

## map()函数-分项操作

**map()**是 Python 内置的高阶函数，它接收一个**函数 f** 和一个**list**，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回。

例如，对于list [1, 2, 3, 4, 5, 6, 7, 8, 9]

如果希望把list的每个元素都作平方，就可以用map()函数：



因此，我们只需要传入函数f(x)=x\*x，就可以利用map()函数完成这个计算：

def f(x):

return x\*x

print map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])

**输出结果：**

[1, 4, 9, 10, 25, 36, 49, 64, 81]

**注意：**map()函数不改变原有的 list，而是返回一个新的 list。

利用map()函数，可以把一个 list 转换为另一个 list，只需要传入转换函数。

由于list包含的元素可以是任何类型，因此，map() 不仅仅可以处理只包含数值的 list，事实上它可以处理包含任意类型的 list，只要传入的函数f可以处理这种数据类型。

## reduce()函数-求和操作

**reduce()**函数也是Python内置的一个高阶函数。reduce()函数接收的参数和 map()类似，**一个函数 f，一个list**，但行为和 map()不同，reduce()传入的函数 f 必须接收两个参数，reduce()对list的每个元素反复调用函数f，并返回最终结果值。

例如，编写一个f函数，接收x和y，返回x和y的和：

def f(x, y):

return x + y

调用 **reduce(f, [1, 3, 5, 7, 9])**时，reduce函数将做如下计算：

先计算头两个元素：f(1, 3)，结果为4；

再把结果和第3个元素计算：f(4, 5)，结果为9；

再把结果和第4个元素计算：f(9, 7)，结果为16；

再把结果和第5个元素计算：f(16, 9)，结果为25；

由于没有更多的元素了，计算结束，返回结果25。

上述计算实际上是对 list 的所有元素求和。虽然Python内置了求和函数sum()，但是，利用reduce()求和也很简单。

**reduce()还可以接收第3个可选参数，作为计算的初始值。**如果把初始值设为100，计算：

reduce(f, [1, 3, 5, 7, 9], 100)

结果将变为125，因为第一轮计算是：

计算初始值和第一个元素：**f(100, 1)**，结果为**101**。

## filter()函数-过滤

**filter()**函数是 Python 内置的另一个有用的高阶函数，filter()函数接收一个**函数 f**和一个**list**，这个函数 f 的作用是对每个元素进行判断，返回 True或 False，**filter()根据判断结果自动过滤掉不符合条件的元素，返回由符合条件元素组成的新list。**

例如，要从一个list [1, 4, 6, 7, 9, 12, 17]中删除偶数，保留奇数，首先，要编写一个判断奇数的函数：

def is\_odd(x):

return x % 2 == 1

然后，利用filter()过滤掉偶数：

filter(is\_odd, [1, 4, 6, 7, 9, 12, 17])

**结果：**[1, 7, 9, 17]

利用filter()，可以完成很多有用的功能，例如，删除 None 或者空字符串：

def is\_not\_empty(s):

return s and len(s.strip()) > 0

filter(is\_not\_empty, ['test', None, '', 'str', ' ', 'END'])

**结果：**['test', 'str', 'END']

**注意:** s.strip(rm) 删除 s 字符串中开头、结尾处的 rm 序列的字符。

当rm为空时，默认删除空白符（包括'\n', '\r', '\t', ' ')，如下：

a = ' 123'

a.strip()

**结果：** '123'

a='\t\t123\r\n'

a.strip()

**结果：**'123'

#### startswith和endswith文本开始结束匹配

 (做文本处理的时候经常要判断一个文本有没有以一个子串开始，或者结束。Python为此提供了两个函数：  
S.startswith(prefix[, start[, end]]) -> bool  
如果字符串S以prefix开始，返回True，否则返回False。start和end是两个可以缺省的参数。分别是开始比较的位置和结束比较的位置。这个函数也可以写成S[start:end].startswith(prefix)。  
S.endswith(suffix[, start[, end]]) -> bool  
如果字符串S以suffix结束，返回True，否者返回False。与startswith类似，这个函数也可以写成S[start:end].endswith(suffix)。start和end仍然是从左数起。  
做个实例：  
>>> “fish”.startswith(”fi”)  
True  
>>> “fish”.startswith(”fi”,1)  
False  
>>> “fish”.endswith(”sh”)  
True  
>>> “fish”.endswith(”sh”,3)  
False  
  
Python的这两个函数有个特别的地方——它的prefix和suffix参数不仅可以是字符串，还都可以是一个元组。只要其中一个成立，就返回True，也就是一种“或”的关系。比如：  
if filename.endswith((’.gif’, ‘.jpg’, ‘.tiff’)):  
print “%s是一个图片文件”%filename  
上面两行代码根据文件扩展名是否是“gif”、“jpg”或“tiff”之一来决定文件是不是图片文件。这个代码也可以写成：  
if filename.endswith(”.gif”) or filename.endswith(”.jpg”) or filename.endswith(”.tiff”):  
print “%s是一个图片文件”%filename  
不过这样比较麻烦。值得注意的是，不熟悉Python的朋友可能会忘了元组周围的括号。

## 自定义排序函数

Python内置的 **sorted()**函数可对list进行排序：

>>>sorted([36, 5, 12, 9, 21])  
[5, 9, 12, 21, 36]

但 **sorted()**也是一个高阶函数，它可以接收一个比较函数来实现自定义排序，比较函数的定义是，传入两个待比较的元素 x, y，**如果 x 应该排在 y 的前面，返回 -1，如果 x 应该排在 y 的后面，返回 1。如果 x 和 y 相等，返回 0。**

因此，如果我们要实现倒序排序，只需要编写一个reversed\_cmp函数：

def reversed\_cmp(x, y):

if x > y:

return -1

if x < y:

return 1

return 0

这样，调用 sorted() 并传入 reversed\_cmp 就可以实现倒序排序：

>>> sorted([36, 5, 12, 9, 21], reversed\_cmp)

[36, 21, 12, 9, 5]

sorted()也可以对字符串进行排序，字符串默认按照ASCII大小来比较：

>>> sorted(['bob', 'about', 'Zoo', 'Credit'])

['Credit', 'Zoo', 'about', 'bob']

'Zoo'排在'about'之前是因为'Z'的ASCII码比'a'小

## 返回函数

Python的函数不但可以返回int、str、list、dict等数据类型，还可以返回函数！

例如，定义一个函数 f()，我们让它返回一个函数 g，可以这样写：

def f():

print 'call f()...'

# 定义函数g:

def g():

print 'call g()...'

# 返回函数g:

return g

仔细观察上面的函数定义，我们在函数 f 内部又定义了一个函数 g。由于函数 g 也是一个对象，函数名 g 就是指向函数 g 的变量，所以，最外层函数 f 可以返回变量 g，也就是函数 g 本身。

调用函数 f，我们会得到 f 返回的一个函数：

>>> x = f() # 调用f()

call f()...

>>> x # 变量x是f()返回的函数：

<function g at 0x1037bf320>

>>> x() # x指向函数，因此可以调用

call g()... # 调用x()就是执行g()函数定义的代码

请注意区分返回函数和返回值：

def myabs():

return abs # 返回函数

def myabs2(x):

return abs(x) # 返回函数调用的结果，返回值是一个数值

返回函数可以把一些计算延迟执行。例如，如果定义一个普通的求和函数：

def calc\_sum(lst):

return sum(lst)

调用calc\_sum()函数时，将立刻计算并得到结果：

>>> calc\_sum([1, 2, 3, 4])

10

但是，如果返回一个函数，就可以“延迟计算”：

def calc\_sum(lst):

def lazy\_sum():

return sum(lst)

return lazy\_sum

# 调用calc\_sum()并没有计算出结果，而是返回函数:

>>> f = calc\_sum([1, 2, 3, 4])

>>> f

<function lazy\_sum at 0x1037bfaa0>

# 对返回的函数进行调用时，才计算出结果:

>>> f()

10

由于可以返回函数，我们在后续代码里就可以决定到底要不要调用该函数。

## 闭包

在函数内部定义的函数和外部定义的函数是一样的，只是他们无法被外部访问：

def g():

print 'g()...'

def f():

print 'f()...'

return g

将**g** 的定义移入函数 **f** 内部，防止其他代码调用 **g**：

def f():

print 'f()...'

def g():

print 'g()...'

return g

但是，考察上一小节定义的 **calc\_sum**函数：

def calc\_sum(lst):

def lazy\_sum():

return sum(lst)

return lazy\_sum

**注意:**发现没法把 **lazy\_sum** 移到 **calc\_sum** 的外部，因为它引用了 **calc\_sum** 的参数 **lst**。

像这种内层函数引用了外层函数的变量（参数也算变量），然后返回内层函数的情况，称为**闭包（Closure）**。

**闭包的特点**是返回的函数还引用了外层函数的局部变量，所以，要正确使用闭包，就要确保引用的局部变量在函数返回后不能变。举例如下：

# 希望一次返回3个函数，分别计算1x1,2x2,3x3:

def count():

fs = []

for i in range(1, 4):

def f():

return i\*i

fs.append(f)

return fs

f1, f2, f3 = count()

你可能认为调用f1()，f2()和f3()结果应该是1，4，9，但实际结果全部都是 9（请自己动手验证）。

原因就是当count()函数返回了3个函数时，这3个函数所引用的变量 i 的值已经变成了3。由于f1、f2、f3并没有被调用，所以，此时他们并未计算 i\*i，当 f1 被调用时：

>>> f1()

9 # 因为f1现在才计算i\*i，但现在i的值已经变为3

因此，返回函数不要引用任何循环变量，或者后续会发生变化的变量。

## lambda匿名函数

高阶函数可以接收函数做参数，有些时候，我们不需要显式地定义函数，直接传入匿名函数更方便。

在Python中，对匿名函数提供了有限支持。还是以map()函数为例，计算 f(x)=x2时，除了定义一个f(x)的函数外，还可以直接传入匿名函数：

>>> map(lambda x: x \* x, [1, 2, 3, 4, 5, 6, 7, 8, 9])

[1, 4, 9, 16, 25, 36, 49, 64, 81]

通过对比可以看出，匿名函数 lambda x: x \* x 实际上就是：

def f(x):

return x \* x

关键字lambda 表示匿名函数，冒号前面的 x 表示函数参数。

匿名函数有个限制，就是**只能有一个表达式**，**不写return**，返回值就是该表达式的结果。

使用匿名函数，可以不必定义函数名，直接创建一个函数对象，很多时候可以简化代码：

>>> sorted([1, 3, 9, 5, 0], lambda x,y: -cmp(x,y))

[9, 5, 3, 1, 0]

返回函数的时候，也可以返回匿名函数：

>>> myabs = lambda x: -x if x < 0 else x

>>> myabs(-1)

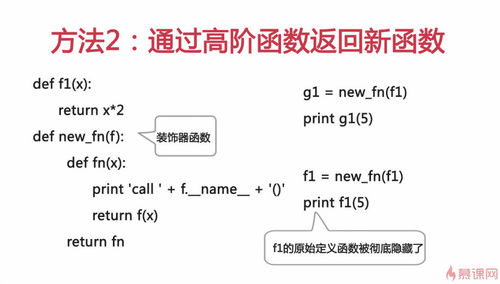
1

>>> myabs(1)

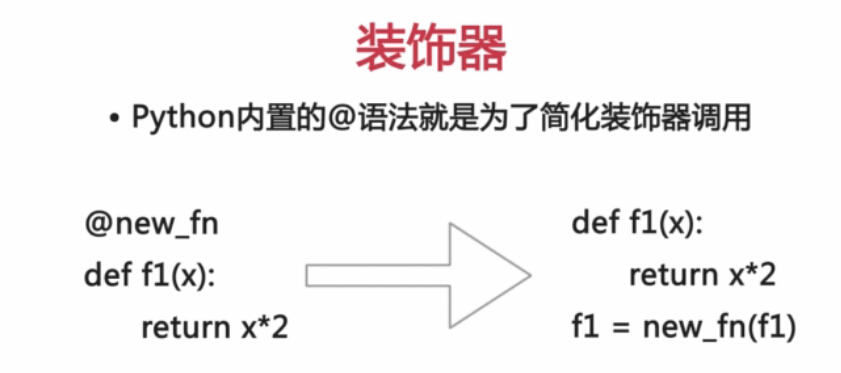
1

## 装饰器（@log）

装饰器所起到的作用，是把 f1 通过 new\_fn 进行了变换了，得到了新的函数



不改变原函数



## 编写无参数decorator

Python的 decorator 本质上就是一个高阶函数，它接收一个函数作为参数，然后，返回一个新函数。

使用 decorator 用Python提供的 @ 语法，这样可以避免手动编写 f = decorate(f) 这样的代码。

**考察一个****@log的定义：**

def log(f):

def fn(x):

print 'call ' + f.\_\_name\_\_ + '()...'

return f(x)

return fn

对于阶乘函数，@log工作得很好：

@log

def factorial(n):

return reduce(lambda x,y: x\*y, range(1, n+1))

print factorial(10)

**结果：**

call factorial()...

3628800

但是，对于参数不是一个的函数，调用将报错：

@log

def add(x, y):

return x + y

print add(1, 2)

**结果：**

Traceback (most recent call last):

File "test.py", line 15, in <module>

print add(1,2)

TypeError: fn() takes exactly 1 argument (2 given)

因为 add() 函数需要传入两个参数，但是 @log 写死了只含一个参数的返回函数。

要让 @log 自适应任何参数定义的函数，可以利用Python的 \*args 和 \*\*kw，保证任意个数的参数总是能正常调用：

def log(f):

def fn(\*args, \*\*kw):

print 'call ' + f.\_\_name\_\_ + '()...'

return f(\*args, \*\*kw)

return fn

现在，对于任意函数，@log 都能正常工作。

## 编写带参数decorator-(l廖雪峰)

由于函数也是一个对象，而且函数对象可以被赋值给变量，所以，通过变量也能调用该函数。

>>> def now():

... print '2013-12-25'

...

>>> f = now

>>> f()

2013-12-25

函数对象有一个\_\_name\_\_属性，可以拿到函数的名字：

>>> now.\_\_name\_\_

'now'

>>> f.\_\_name\_\_

'now'

现在，假设我们要增强now()函数的功能，比如，在函数调用前后自动打印日志，但又不希望修改now()函数的定义，这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。

本质上，decorator就是一个返回函数的高阶函数。所以，我们要定义一个能打印日志的decorator，可以定义如下：

def log(func):

def wrapper(\*args, \*\*kw):

print 'call %s():' % func.\_\_name\_\_

return func(\*args, \*\*kw)

return wrapper

观察上面的log，因为它是一个decorator，所以接受一个函数作为参数，并返回一个函数。我们要借助Python的@语法，把decorator置于函数的定义处：

@log

def now():

print '2013-12-25'

调用now()函数，不仅会运行now()函数本身，还会在运行now()函数前打印一行日志：

>>> now()

call now():

2013-12-25

把@log放到now()函数的定义处，相当于执行了语句：

now = log(now)

由于log()是一个decorator，返回一个函数，所以，原来的now()函数仍然存在，只是现在同名的now变量指向了新的函数，于是调用now()将执行新函数，即在log()函数中返回的wrapper()函数。

wrapper()函数的参数定义是(\*args, \*\*kw)，因此，wrapper()函数可以接受任意参数的调用。在wrapper()函数内，首先打印日志，再紧接着调用原始函数。

如果decorator本身需要传入参数，那就需要编写一个返回decorator的高阶函数，写出来会更复杂。比如，要自定义log的文本：

def log(text):

def decorator(func):

def wrapper(\*args, \*\*kw):

print '%s %s():' % (text, func.\_\_name\_\_)

return func(\*args, \*\*kw)

return wrapper

return decorator

这个3层嵌套的decorator用法如下：

@log('execute')

def now():

print '2013-12-25'

执行结果如下：

>>> now()

execute now():

2013-12-25

和两层嵌套的decorator相比，3层嵌套的效果是这样的：

>>> now = log('execute')(now)

我们来剖析上面的语句，首先执行log('execute')，返回的是decorator函数，再调用返回的函数，参数是now函数，返回值最终是wrapper函数。

以上两种decorator的定义都没有问题，但还差最后一步。因为我们讲了函数也是对象，它有\_\_name\_\_等属性，但你去看经过decorator装饰之后的函数，它们的\_\_name\_\_已经从原来的'now'变成了'wrapper'：

>>> now.\_\_name\_\_

'wrapper'

因为返回的那个wrapper()函数名字就是'wrapper'，所以，需要把原始函数的\_\_name\_\_等属性复制到wrapper()函数中，否则，有些依赖函数签名的代码执行就会出错。

不需要编写wrapper.\_\_name\_\_ = func.\_\_name\_\_这样的代码，Python内置的functools.wraps就是干这个事的，所以，一个完整的decorator的写法如下：

import functools

def log(func):

@functools.wraps(func)

def wrapper(\*args, \*\*kw):

print 'call %s():' % func.\_\_name\_\_

return func(\*args, \*\*kw)

return wrapper

或者针对带参数的decorator：

import functools

def log(text):

def decorator(func):

@functools.wraps(func)

def wrapper(\*args, \*\*kw):

print '%s %s():' % (text, func.\_\_name\_\_)

return func(\*args, \*\*kw)

return wrapper

return decorator

import functools是导入functools模块。模块的概念稍候讲解。现在，只需记住在定义wrapper()的前面加上@functools.wraps(func)即可。

**小结**

对于这种3层嵌套的decorator定义，你可以先把它拆开：

# 标准decorator:

def log\_decorator(f):

def wrapper(\*args, \*\*kw):

print '[%s] %s()...' % (prefix, f.\_\_name\_\_)

return f(\*args, \*\*kw)

return wrapper

return log\_decorator

# 返回decorator:

def log(prefix):

return log\_decorator(f)

拆开以后会发现，调用会失败，因为在3层嵌套的decorator定义中，最内层的wrapper引用了最外层的参数prefix，所以，把一个闭包拆成普通的函数调用会比较困难。不支持闭包的编程语言要实现同样的功能就需要更多的代码

在面向对象（OOP）的设计模式中，decorator被称为装饰模式。OOP的装饰模式需要通过继承和组合来实现，而Python除了能支持OOP的decorator外，直接从语法层次支持decorator。Python的decorator可以用函数实现，也可以用类实现。

decorator可以增强函数的功能，定义起来虽然有点复杂，但使用起来非常灵活和方便。

## 完善decorator

@decorator可以动态实现函数功能的增加，但是，经过@decorator“改造”后的函数，和原函数相比，除了功能多一点外，有没有其它不同的地方？

在没有decorator的情况下，打印函数名：

def f1(x):

pass

print f1.\_\_name\_\_

**输出：**f1

有decorator的情况下，再打印函数名：

def log(f):

def wrapper(\*args, \*\*kw):

print 'call...'

return f(\*args, \*\*kw)

return wrapper

@log

def f2(x):

pass

print f2.\_\_name\_\_

**输出：** wrapper

可见，由于decorator返回的新函数函数名已经不是'f2'，而是@log内部定义的'wrapper'。这对于那些依赖函数名的代码就会失效。decorator还改变了函数的\_\_doc\_\_等其它属性。如果要让调用者看不出一个函数经过了@decorator的“改造”，就需要把原函数的一些属性复制到新函数中：

def log(f):

def wrapper(\*args, \*\*kw):

print 'call...'

return f(\*args, \*\*kw)

wrapper.\_\_name\_\_ = f.\_\_name\_\_

wrapper.\_\_doc\_\_ = f.\_\_doc\_\_

return wrapper

这样写decorator很不方便，因为我们也很难把原函数的所有必要属性都一个一个复制到新函数上，所以Python内置的functools可以用来自动化完成这个“复制”的任务：

import functools

def log(f):

@functools.wraps(f)

def wrapper(\*args, \*\*kw):

print 'call...'

return f(\*args, \*\*kw)

return wrapper

最后需要指出，由于我们把原函数签名改成了(\*args, \*\*kw)，因此，无法获得原函数的原始参数信息。即便我们采用固定参数来装饰只有一个参数的函数：

def log(f):

@functools.wraps(f)

def wrapper(x):

print 'call...'

return f(x)

return wrapper

也可能改变原函数的参数名，因为新函数的参数名始终是 'x'，原函数定义的参数名不一定叫 'x'。

## 偏函数

当一个函数有很多参数时，调用者就需要提供多个参数。如果减少参数个数，就可以简化调用者的负担。

比如，int()函数可以把字符串转换为整数，当仅传入字符串时，int()函数默认按十进制转换：

>>> int('12345')

12345

但int()函数还提供额外的base参数，默认值为10。如果传入base参数，就可以做 N 进制的转换：

>>> int('12345', base=8)

5349

>>> int('12345', 16)

74565

假设要转换大量的二进制字符串，每次都传入int(x, base=2)非常麻烦，于是，我们想到，可以定义一个int2()的函数，默认把base=2传进去：

def int2(x, base=2):

return int(x, base)

这样，我们转换二进制就非常方便了：

>>> int2('1000000')

64

>>> int2('1010101')

85

functools.partial就是帮助我们创建一个偏函数的，不需要我们自己定义int2()，可以直接使用下面的代码创建一个新的函数int2：

>>> import functools

>>> int2 = functools.partial(int, base=2)

>>> int2('1000000')

64

>>> int2('1010101')

85

所以，functools.partial可以把一个参数多的函数变成一个参数少的新函数，少的参数需要在创建时指定默认值，这样，新函数调用的难度就降低了。

# 模块

Import xx相当于赋值

使用是时候是xx.\*

Import xx from \*\*:导入注意魔术变量\_\_all\_\_这个是指定可以外部使用的几个函数

使用的时候直接用\*

## 常用模块

1 urllib urllib2 网络

2 datetime time 时间

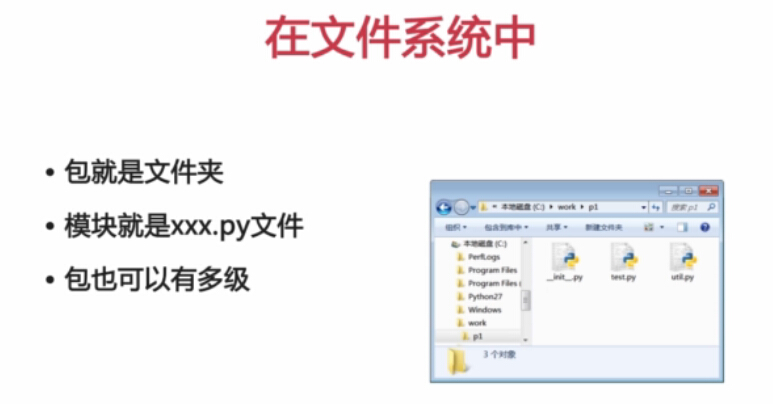
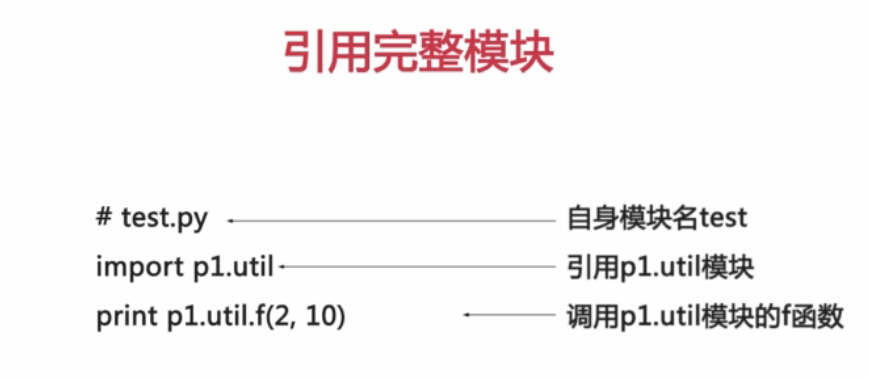
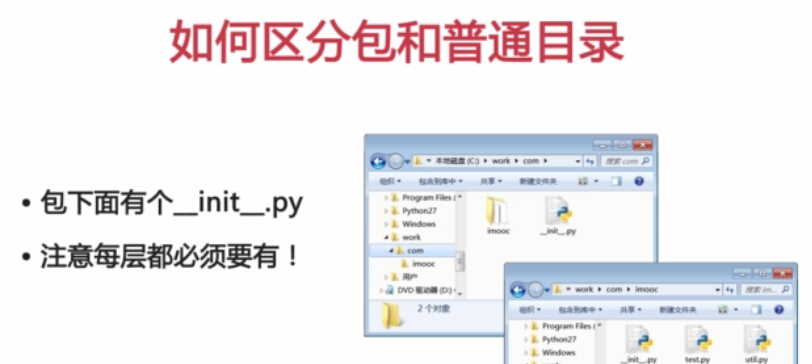
3 os 系统

4 pickle 对象序列化

常用数据交换格式 json xml

5 bsddb key=>value

6 logging 日志



## 导入模块

要使用一个模块，我们必须首先导入该模块。Python使用import语句导入一个模块。例如，导入系统自带的模块 math：

import math

你可以认为math就是一个指向已导入模块的变量，通过该变量，我们可以访问math模块中所定义的所有公开的函数、变量和类：

>>> math.pow(2, 0.5) # pow是函数

1.4142135623730951

>>> math.pi # pi是变量

3.141592653589793

如果我们只希望导入用到的math模块的某几个函数，而不是所有函数，可以用下面的语句：

from math import pow, sin, log

这样，可以直接引用 pow, sin, log 这3个函数，但math的其他函数没有导入进来：

>>> pow(2, 10)

1024.0

>>> sin(3.14)

0.0015926529164868282

## 别名

如果遇到名字冲突怎么办？比如math模块有一个log函数，logging模块也有一个log函数，如果同时使用，如何解决名字冲突？

如果使用import导入模块名，由于必须通过模块名引用函数名，因此不存在冲突：

import math, logging

print math.log(10) # 调用的是math的log函数

logging.log(10, 'something') # 调用的是logging的log函数

如果使用 from...import 导入 log 函数，势必引起冲突。这时，可以给函数起个“别名”来避免冲突：

from math import log

from logging import log as logger # logging的log现在变成了logger

print log(10) # 调用的是math的log

logger(10, 'import from logging') # 调用的是logging的log

## 作用域

在一个模块中，我们可能会定义很多函数和变量，但有的函数和变量我们希望给别人使用，有的函数和变量我们希望仅仅在模块内部使用。在Python中，是通过\_前缀来实现的。

正常的函数和变量名是公开的（public），可以被直接引用，比如：abc，x123，PI等；

类似\_\_xxx\_\_这样的变量是特殊变量，可以被直接引用，但是有特殊用途，比如上面的\_\_author\_\_，\_\_name\_\_就是特殊变量，hello模块定义的文档注释也可以用特殊变量\_\_doc\_\_访问，我们自己的变量一般不要用这种变量名；

类似\_xxx和\_\_xxx这样的函数或变量就是非公开的（private），不应该被直接引用，比如\_abc，\_\_abc等；

之所以我们说，private函数和变量“不应该”被直接引用，而不是“不能”被直接引用，是因为Python并没有一种方法可

以完全限制访问private函数或变量，但是，从编程习惯上不应该引用private函数或变量。

private函数或变量不应该被别人引用，那它们有什么用呢？请看例子：

def \_private\_1(name):

return 'Hello, %s' % name

def \_private\_2(name):

return 'Hi, %s' % name

def greeting(name):

if len(name) > 3:

return \_private\_1(name)

else:

return \_private\_2(name)

我们在模块里公开greeting()函数，而把内部逻辑用private函数隐藏起来了，这样，调用greeting()函数不用关心内部的private函数细节，这也是一种非常有用的代码封装和抽象的方法，即：

外部不需要引用的函数全部定义成private，只有外部需要引用的函数才定义为public。

## 动态导入模块

如果导入的模块不存在，Python解释器会报 ImportError 错误：

>>> import something

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ImportError: No module named something

有的时候，两个不同的模块提供了相同的功能，比如 **StringIO**和 **cStringIO**都提供了StringIO这个功能。

这是因为Python是动态语言，解释执行，因此Python代码运行速度慢。

如果要提高Python代码的运行速度，最简单的方法是把某些关键函数用 C 语言重写，这样就能大大提高执行速度。

同样的功能，StringIO 是纯Python代码编写的，而 cStringIO 部分函数是 C 写的，因此 cStringIO 运行速度更快。

利用ImportError错误，我们经常在Python中动态导入模块：

try:

from cStringIO import StringIO

except ImportError:

from StringIO import StringIO

上述代码先尝试从cStringIO导入，如果失败了（比如cStringIO没有被安装），再尝试从StringIO导入。这样，如果cStringIO模块存在，则我们将获得更快的运行速度，如果cStringIO不存在，则顶多代码运行速度会变慢，但不会影响代码的正常执行。

**try**的作用是捕获错误，并在捕获到指定错误时执行 **except**语句。

## 使用\_\_future\_\_

Python的新版本会引入新的功能，但是，实际上这些功能在上一个老版本中就已经存在了。要“试用”某一新的特性，就可以通过导入\_\_future\_\_模块的某些功能来实现。

例如，Python 2.7的整数除法运算结果仍是整数：

>>> 10 / 3

3

但是，Python 3.x已经改进了整数的除法运算，“**/**”除将得到浮点数，“**//**”除才仍是整数：

>>> 10 / 3

3.3333333333333335

>>> 10 // 3

3

要在Python 2.7中引入3.x的除法规则，导入**\_\_future\_\_**的**division**：

>>> from \_\_future\_\_ import division

>>> print 10 / 3

3.3333333333333335

当新版本的一个特性与旧版本不兼容时，该特性将会在旧版本中添加到**\_\_future\_\_**中，以便旧的代码能在旧版本中测试新特性。

## 安装第三方模块-（慕课有视频）

在Python中，安装第三方模块，是通过setuptools这个工具完成的。Python有两个封装了setuptools的包管理工具：easy\_install和pip。目前官方推荐使用pip。

如果你正在使用Mac或Linux，安装pip本身这个步骤就可以跳过了。

如果你正在使用Windows，请参考[安装Python](http://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/001374738150500472fd5785c194ebea336061163a8a974000)一节的内容，确保安装时勾选了pip和Add python.exe to Path。

在命令提示符窗口下尝试运行pip，如果Windows提示未找到命令，可以重新运行安装程序添加pip。

现在，让我们来安装一个第三方库——Python Imaging Library，这是Python下非常强大的处理图像的工具库。一般来说，第三方库都会在Python官方的[pypi.python.org](https://pypi.python.org/" \t "_blank)网站注册，要安装一个第三方库，必须先知道该库的名称，可以在官网或者pypi上搜索，比如Python Imaging Library的名称叫PIL，因此，安装Python Imaging Library的命令就是：

pip install PIL

耐心等待下载并安装后，就可以使用PIL了。

有了PIL，处理图片易如反掌。随便找个图片生成缩略图：

>>> import Image

>>> im = Image.open('test.png')

>>> print im.format, im.size, im.mode

PNG (400, 300) RGB

>>> im.thumbnail((200, 100))

>>> im.save('thumb.jpg', 'JPEG')

其他常用的第三方库还有MySQL的驱动：MySQL-python，用于科学计算的NumPy库：numpy，用于生成文本的模板工具Jinja2，等等。

**模块搜索路径**

当我们试图加载一个模块时，Python会在指定的路径下搜索对应的.py文件，如果找不到，就会报错：

>>> import mymodule

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ImportError: No module named mymodule

默认情况下，Python解释器会搜索当前目录、所有已安装的内置模块和第三方模块，搜索路径存放在sys模块的path变量中：

>>> import sys

>>> sys.path

['', '/Library/Python/2.7/site-packages/pycrypto-2.6.1-py2.7-macosx-10.9-intel.egg', '/Library/Python/2.7/site-packages/PIL-1.1.7-py2.7-macosx-10.9-intel.egg', ...]

如果我们要添加自己的搜索目录，有两种方法：

一是直接修改sys.path，添加要搜索的目录：

>>> import sys

>>> sys.path.append('/Users/michael/my\_py\_scripts')

这种方法是在运行时修改，运行结束后失效。

第二种方法是设置环境变量PYTHONPATH，该环境变量的内容会被自动添加到模块搜索路径中。设置方式与设置Path环境变量类似。注意只需要添加你自己的搜索路径，Python自己本身的搜索路径不受影响。

# 模块学习

## Urllib

  urllib模块提供的上层接口，使我们可以像读取本地文件一样读取www和ftp上的数据。每当使用这个模块的时候，老是会想起公司产品的客户端，同事用C++下载Web上的图片，那种“痛苦”的表情。我以前翻译过libcurl教程，这是在C/C++环境下比较方便实用的网络操作库，相比起libcurl，Python的urllib模块的使用门槛则低多了。可能有些人又会用效率来批评Python，其实在操作网络，或者在集群交互的时候， 语言的执行效率绝不是瓶颈。这种情况下，一个比较好的方法是，将python嵌入到C/C++中，让Python来完成一些不是核心的逻辑处理。又扯远了，废话少说，开始urllib之旅吧~~ （前几天我用这个模块写了个蜘蛛，感兴趣的同学可以在以前的博客中找到代码）  
    先看一个例子，这个例子把Google首页的html抓取下来并显示在控制台上：

**[python]** [view plain](http://blog.csdn.net/jgood/article/details/5493824" \o "view plain)[copy](http://blog.csdn.net/jgood/article/details/5493824" \o "copy)

1. # 别惊讶，整个程序确实只用了两行代码
2. **import** urllib
3. **print** urllib.urlopen('http://www.google.com').read()

### urllib.urlopen(url[, data[, proxies]]) :

    创建一个表示远程url的类文件对象，然后像本地文件一样操作这个类文件对象来获取远程数据。参数url表示远程数据的路径，一般是网址；参数data表示以post方式提交到url的数据(玩过web的人应该知道提交数据的两种方式：post与get。如果你不清楚，也不必太在意，一般情况下很少用到这个参数)；参数proxies用于设置代理（这里不详细讲怎么使用代理，感兴趣的看客可以去翻阅Python手册urllib模块）。urlopen返回 一个类文件对象，他提供了如下方法：

* read() , readline() , readlines() , fileno() , close() ：这些方法的使用方式与文件对象完全一样;
* info()：返回一个httplib.HTTPMessage 对象，表示远程服务器返回的头信息；
* getcode()：返回Http状态码。如果是http请求，200表示请求成功完成;404表示网址未找到；
* geturl()：返回请求的url；

    下面来扩充一下上面的例子，看官可以运行一下这个例子，加深对urllib的印象：

**[python]** [view plain](http://blog.csdn.net/jgood/article/details/5493824" \o "view plain)[copy](http://blog.csdn.net/jgood/article/details/5493824" \o "copy)

1. google = urllib.urlopen('http://www.google.com')
2. **print** 'http header:/n', google.info()
3. **print** 'http status:', google.getcode()
4. **print** 'url:', google.geturl()
5. **for** line **in** google: # 就像在操作本地文件
6. **print** line,
7. google.close()

### urllib.urlretrieve(url[, filename[, reporthook[, data]]])：

    urlretrieve方法直接将远程数据下载到本地。参数filename指定了保存到本地的路径（如果未指定该参数，urllib会生成一个临时文件来保存数据）；参数reporthook是一个回调函数，当连接上服务器、以及相应的数据块传输完毕的时候会触发该回调。我们可以利用这个回调函 数来显示当前的下载进度，下面的例子会展示。参数data指post到服务器的数据。该方法返回一个包含两个元素的元组(filename, headers)，filename表示保存到本地的路径，header表示服务器的响应头。下面通过例子来演示一下这个方法的使用，这个例子将新浪首页的html抓取到本地，保存在D:/sina.html文件中，同时显示下载的进度。

**[python]** [view plain](http://blog.csdn.net/jgood/article/details/5493824" \o "view plain)[copy](http://blog.csdn.net/jgood/article/details/5493824" \o "copy)

1. **def** cbk(a, b, c):
2. '''''回调函数
3. @a: 已经下载的数据块
4. @b: 数据块的大小
5. @c: 远程文件的大小
6. '''
7. per = 100.0 \* a \* b / c
8. **if** per > 100:
9. per = 100
10. **print** '%.2f%%' % per
12. url = 'http://www.sina.com.cn'
13. local = 'd://sina.html'
14. urllib.urlretrieve(url, local, cbk)

    上面介绍的两个方法是urllib中最常用的方法，这些方法在获取远程数据的时候，内部会使用URLopener或者 FancyURLOpener类。作为urllib的使用者，我们很少会用到这两个类，这里我不想多讲。如果对urllib的实现感兴趣， 或者希望urllib支持更多的协议，可以研究这两个类。在Python手册中，urllib的作者还列出了这个模块的缺陷和不足，感兴趣的同学可以打开 Python手册了解一下。

    urllib中还提供了一些辅助方法，用于对url进行编码、解码。url中是不能出现一些特殊的符号的，有些符号有特殊的用途。我们知道以get方式提交数据的时候，会在url中添加key=value这样的字符串，所以在value中是不允许有'='，因此要对其进行编码；与此同时服务器接收到这些参数的时候，要进行解码，还原成原始的数据。这个时候，这些辅助方法会很有用：

* urllib.quote(string[, safe])：对字符串进行编码。参数safe指定了不需要编码的字符;
* urllib.unquote(string) ：对字符串进行解码；
* urllib.quote\_plus(string [ , safe ] ) ：与urllib.quote类似，但这个方法用'+'来替换' '，而quote用'%20'来代替' '
* urllib.unquote\_plus(string ) ：对字符串进行解码；
* urllib.urlencode(query[, doseq])：将dict或者包含两个元素的元组列表转换成url参数。例如 字典{'name': 'dark-bull', 'age': 200}将被转换为"name=dark-bull&age=200"
* urllib.pathname2url(path)：将本地路径转换成url路径；
* urllib.url2pathname(path)：将url路径转换成本地路径；

    用一个例子来体验一下这些方法吧~~：

**[python]** [view plain](http://blog.csdn.net/jgood/article/details/5493824" \o "view plain)[copy](http://blog.csdn.net/jgood/article/details/5493824" \o "copy)

1. data = 'name = ~a+3'
3. data1 = urllib.quote(data)
4. **print** data1 # result: name%20%3D%20%7Ea%2B3
5. **print** urllib.unquote(data1) # result: name = ~a+3
7. data2 = urllib.quote\_plus(data)
8. **print** data2 # result: name+%3D+%7Ea%2B3
9. **print** urllib.unquote\_plus(data2)    # result: name = ~a+3
11. data3 = urllib.urlencode({ 'name': 'dark-bull', 'age': 200 })
12. **print** data3 # result: age=200&name=dark-bull
14. data4 = urllib.pathname2url(r'd:/a/b/c/23.php')
15. **print** data4 # result: ///D|/a/b/c/23.php
16. **print** urllib.url2pathname(data4)    # result: D:/a/b/c/23.php

    urllib模块的基本使用，就这么简单。oh~~yeah~~又一个模块写完了，想想，我已经写了将近30个模块了，有时间我要好好整理一下@@@@

## urlparse学习

**通过Python所带的urlparse模块，我们能够轻松地把URL分解成元件，之后，还能将这些元件重新组装成一个URL。当我们处理HTML 文档的时候，这项功能是非常方便的。**

urlparse模块主要是把url拆分为6部分，并返回元组。并且可以把拆分后的部分再组成一个url。主要有函数有urljoin、urlsplit、urlunsplit、urlparse等。

urlparse.urlparse(urlstring[, scheme[, allow\_fragments]])

    将urlstring解析成6个部分，它从urlstring中取得URL，并返回元组 (scheme, netloc, path, parameters, query, fragment)，但是实际上是基于namedtuple，是tuple的子类。它支持通过名字属性或者索引访问的部分URL，每个组件是一串字符，也有可能是空的。组件不能被解析为更小的部分，%后面的也不会被解析，分割符号并不是解析结果的一部分，除非用斜线转义，注意，返回的这个元组非常有用，例如可以用来确定网络协议(HTTP、FTP等等 )、服务器地址、文件路径，等等。

[?](http://my.oschina.net/guol/blog/95699)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> import urlparse  >>> url=urlparse.urlparse('http://www.baidu.com/index.php?username=guol')  >>> print url  ParseResult(scheme='http', netloc='www.baidu.com', path='/index.php', params='', query='username=guol', fragment='')  >>> print url.netloc  www.baidu.com  >>> |

urlparse.urlunparse(parts)

    从一个元组构建一个url，元组类似urlparse返回的，它接收元组(scheme, netloc, path, parameters, query, fragment)后，会重新组成一个具有正确格式的URL，以便供Python的其他HTML解析模块使用。

[?](http://my.oschina.net/guol/blog/95699)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> import urlparse  >>> url=urlparse.urlparse('http://www.baidu.com/index.php?username=guol')  >>> print url  ParseResult(scheme='http', netloc='www.baidu.com', path='/index.php', params='', query='username=guol', fragment='')  >>> u=urlparse.urlunparse(url)  >>> print u  http://www.baidu.com/index.php?username=guol |

urlparse.urlsplit(urlstring[, scheme[, allow\_fragments]])

    主要是分析urlstring，返回一个包含5个字符串项目的元组：协议、位置、路径、查询、片段。allow\_fragments为False时，该元组的组后一个项目总是空，不管urlstring有没有片段，省略项目的也是空。urlsplit()和urlparse()差不多。不过它不切分URL的参数。适用于遵循RFC2396的URL，每个路径段都支持参数。这样返回的元组就只有5个元素。

[?](http://my.oschina.net/guol/blog/95699)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | >>> import urlparse  >>> url=urlparse.urlparse('http://www.baidu.com/index.php?username=guol')  >>> print url  ParseResult(scheme='http', netloc='www.baidu.com', path='/index.php', params='', query='username=guol', fragment='')  >>> url=urlparse.urlsplit('http://www.baidu.com/index.php?username=guol')  >>> print url  SplitResult(scheme='http', netloc='www.baidu.com', path='/index.php', query='username=guol', fragment='') |

urlparse.urlunsplit(parts)

    urlunsplit使用urlsplit()返回的值组合成一个url

urlparse.urljoin(base, url[, allow\_fragments])

    urljoin主要是拼接URL，它以base作为其基地址，然后与url中的相对地址相结合组成一个绝对URL地址。函数urljoin在通过为URL基地址附加新的文件名的方式来处理同一位置处的若干文件的时候格外有用。需要注意的是，如果基地址并非以字符/结尾的话，那么URL基地址最右边部分就会被这个相对路径所替换。如果希望在该路径中保留末端目录，应确保URL基地址以字符/结尾。

[?](http://my.oschina.net/guol/blog/95699)

|  |  |
| --- | --- |
| 1  2  3  4  5 | >>> import urlparse  >>> urlparse.urljoin('http://www.oschina.com/tieba','index.php')  'http://www.oschina.com/index.php'  >>> urlparse.urljoin('http://www.oschina.com/tieba/','index.php')  'http://www.oschina.com/tieba/index.php' |

## Logging

**简单将日志打印到屏幕：**

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. **import** logging
2. logging.debug('debug message')
3. logging.info('info message')
4. logging.warning('warning message')
5. logging.error('error message')
6. logging.critical('critical message')

输出：

WARNING:root:warning message  
ERROR:root:error message  
CRITICAL:root:critical message

可见，默认情况下python的logging模块将日志打印到了标准输出中，且只显示了大于等于WARNING级别的日志，这说明默认的日志级别设置为WARNING（日志级别等级CRITICAL > ERROR > WARNING > INFO > DEBUG > NOTSET），默认的日志格式为日志级别：Logger名称：用户输出消息。

**灵活配置日志级别，日志格式，输出位置**

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. **import** logging
2. logging.basicConfig(level=logging.DEBUG,
3. format='%(asctime)s %(filename)s[line:%(lineno)d] %(levelname)s %(message)s',
4. datefmt='%a, %d %b %Y %H:%M:%S',
5. filename='/tmp/test.log',
6. filemode='w')
8. logging.debug('debug message')
9. logging.info('info message')
10. logging.warning('warning message')
11. logging.error('error message')
12. logging.critical('critical message')

查看输出：  
cat /tmp/test.log   
Mon, 05 May 2014 16:29:53 test\_logging.py[line:9] DEBUG debug message  
Mon, 05 May 2014 16:29:53 test\_logging.py[line:10] INFO info message  
Mon, 05 May 2014 16:29:53 test\_logging.py[line:11] WARNING warning message  
Mon, 05 May 2014 16:29:53 test\_logging.py[line:12] ERROR error message  
Mon, 05 May 2014 16:29:53 test\_logging.py[line:13] CRITICAL critical message  
  
可见在logging.basicConfig()函数中可通过具体参数来更改logging模块默认行为，可用参数有  
filename：用指定的文件名创建FiledHandler（后边会具体讲解handler的概念），这样日志会被存储在指定的文件中。  
filemode：文件打开方式，在指定了filename时使用这个参数，默认值为“a”还可指定为“w”。  
format：指定handler使用的日志显示格式。   
datefmt：指定日期时间格式。   
level：设置rootlogger（后边会讲解具体概念）的日志级别   
stream：用指定的stream创建StreamHandler。可以指定输出到sys.stderr,sys.stdout或者文件，默认为sys.stderr。若同时列出了filename和stream两个参数，则stream参数会被忽略。  
  
format参数中可能用到的格式化串：  
%(name)s Logger的名字  
%(levelno)s 数字形式的日志级别  
%(levelname)s 文本形式的日志级别  
%(pathname)s 调用日志输出函数的模块的完整路径名，可能没有  
%(filename)s 调用日志输出函数的模块的文件名  
%(module)s 调用日志输出函数的模块名  
%(funcName)s 调用日志输出函数的函数名  
%(lineno)d 调用日志输出函数的语句所在的代码行  
%(created)f 当前时间，用UNIX标准的表示时间的浮 点数表示  
%(relativeCreated)d 输出日志信息时的，自Logger创建以 来的毫秒数  
%(asctime)s 字符串形式的当前时间。默认格式是 “2003-07-08 16:49:45,896”。逗号后面的是毫秒  
%(thread)d 线程ID。可能没有  
%(threadName)s 线程名。可能没有  
%(process)d 进程ID。可能没有  
%(message)s用户输出的消息  
  
  
**若要对logging进行更多灵活的控制有必要了解一下Logger，Handler，Formatter，Filter的概念**  
上述几个例子中我们了解到了logging.debug()、logging.info()、logging.warning()、logging.error()、logging.critical()（分别用以记录不同级别的日志信息），logging.basicConfig()（用默认日志格式（Formatter）为日志系统建立一个默认的流处理器（StreamHandler），设置基础配置（如日志级别等）并加到root logger（根Logger）中）这几个logging模块级别的函数，另外还有一个模块级别的函数是logging.getLogger([name])（返回一个logger对象，如果没有指定名字将返回root logger）  
  
先看一个具体的例子

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #coding:utf-8
2. **import** logging
4. # 创建一个logger
5. logger = logging.getLogger()
7. logger1 = logging.getLogger('mylogger')
8. logger1.setLevel(logging.DEBUG)
10. logger2 = logging.getLogger('mylogger')
11. logger2.setLevel(logging.INFO)
13. logger3 = logging.getLogger('mylogger.child1')
14. logger3.setLevel(logging.WARNING)
16. logger4 = logging.getLogger('mylogger.child1.child2')
17. logger4.setLevel(logging.DEBUG)
19. logger5 = logging.getLogger('mylogger.child1.child2.child3')
20. logger5.setLevel(logging.DEBUG)
22. # 创建一个handler，用于写入日志文件
23. fh = logging.FileHandler('/tmp/test.log')
25. # 再创建一个handler，用于输出到控制台
26. ch = logging.StreamHandler()
28. # 定义handler的输出格式formatter
29. formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
30. fh.setFormatter(formatter)
31. ch.setFormatter(formatter)
33. #定义一个filter
34. #filter = logging.Filter('mylogger.child1.child2')
35. #fh.addFilter(filter)
37. # 给logger添加handler
38. #logger.addFilter(filter)
39. logger.addHandler(fh)
40. logger.addHandler(ch)
42. #logger1.addFilter(filter)
43. logger1.addHandler(fh)
44. logger1.addHandler(ch)
46. logger2.addHandler(fh)
47. logger2.addHandler(ch)
49. #logger3.addFilter(filter)
50. logger3.addHandler(fh)
51. logger3.addHandler(ch)
53. #logger4.addFilter(filter)
54. logger4.addHandler(fh)
55. logger4.addHandler(ch)
57. logger5.addHandler(fh)
58. logger5.addHandler(ch)
60. # 记录一条日志
61. logger.debug('logger debug message')
62. logger.info('logger info message')
63. logger.warning('logger warning message')
64. logger.error('logger error message')
65. logger.critical('logger critical message')
67. logger1.debug('logger1 debug message')
68. logger1.info('logger1 info message')
69. logger1.warning('logger1 warning message')
70. logger1.error('logger1 error message')
71. logger1.critical('logger1 critical message')
73. logger2.debug('logger2 debug message')
74. logger2.info('logger2 info message')
75. logger2.warning('logger2 warning message')
76. logger2.error('logger2 error message')
77. logger2.critical('logger2 critical message')
79. logger3.debug('logger3 debug message')
80. logger3.info('logger3 info message')
81. logger3.warning('logger3 warning message')
82. logger3.error('logger3 error message')
83. logger3.critical('logger3 critical message')
85. logger4.debug('logger4 debug message')
86. logger4.info('logger4 info message')
87. logger4.warning('logger4 warning message')
88. logger4.error('logger4 error message')
89. logger4.critical('logger4 critical message')
91. logger5.debug('logger5 debug message')
92. logger5.info('logger5 info message')
93. logger5.warning('logger5 warning message')
94. logger5.error('logger5 error message')
95. logger5.critical('logger5 critical message')

输出：  
2014-05-06 12:54:43,222 - root - WARNING - logger warning message  
2014-05-06 12:54:43,223 - root - ERROR - logger error message  
2014-05-06 12:54:43,224 - root - CRITICAL - logger critical message  
2014-05-06 12:54:43,224 - mylogger - INFO - logger1 info message  
2014-05-06 12:54:43,224 - mylogger - INFO - logger1 info message  
2014-05-06 12:54:43,225 - mylogger - WARNING - logger1 warning message  
2014-05-06 12:54:43,225 - mylogger - WARNING - logger1 warning message  
2014-05-06 12:54:43,226 - mylogger - ERROR - logger1 error message  
2014-05-06 12:54:43,226 - mylogger - ERROR - logger1 error message  
2014-05-06 12:54:43,227 - mylogger - CRITICAL - logger1 critical message  
2014-05-06 12:54:43,227 - mylogger - CRITICAL - logger1 critical message  
2014-05-06 12:54:43,228 - mylogger - INFO - logger2 info message  
2014-05-06 12:54:43,228 - mylogger - INFO - logger2 info message  
2014-05-06 12:54:43,229 - mylogger - WARNING - logger2 warning message  
2014-05-06 12:54:43,229 - mylogger - WARNING - logger2 warning message  
2014-05-06 12:54:43,230 - mylogger - ERROR - logger2 error message  
2014-05-06 12:54:43,230 - mylogger - ERROR - logger2 error message  
2014-05-06 12:54:43,231 - mylogger - CRITICAL - logger2 critical message  
2014-05-06 12:54:43,231 - mylogger - CRITICAL - logger2 critical message  
2014-05-06 12:54:43,232 - mylogger.child1 - WARNING - logger3 warning message  
2014-05-06 12:54:43,232 - mylogger.child1 - WARNING - logger3 warning message  
2014-05-06 12:54:43,232 - mylogger.child1 - WARNING - logger3 warning message  
2014-05-06 12:54:43,234 - mylogger.child1 - ERROR - logger3 error message  
2014-05-06 12:54:43,234 - mylogger.child1 - ERROR - logger3 error message  
2014-05-06 12:54:43,234 - mylogger.child1 - ERROR - logger3 error message  
2014-05-06 12:54:43,235 - mylogger.child1 - CRITICAL - logger3 critical message  
2014-05-06 12:54:43,235 - mylogger.child1 - CRITICAL - logger3 critical message  
2014-05-06 12:54:43,235 - mylogger.child1 - CRITICAL - logger3 critical message  
2014-05-06 12:54:43,237 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 12:54:43,237 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 12:54:43,237 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 12:54:43,237 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 12:54:43,239 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 12:54:43,239 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 12:54:43,239 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 12:54:43,239 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 12:54:43,240 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 12:54:43,240 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 12:54:43,240 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 12:54:43,240 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 12:54:43,242 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 12:54:43,242 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 12:54:43,242 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 12:54:43,242 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 12:54:43,243 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 12:54:43,243 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 12:54:43,243 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 12:54:43,243 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 12:54:43,244 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 12:54:43,244 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 12:54:43,244 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 12:54:43,244 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 12:54:43,244 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 12:54:43,246 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 12:54:43,246 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 12:54:43,246 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 12:54:43,246 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 12:54:43,246 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 12:54:43,247 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 12:54:43,247 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 12:54:43,247 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 12:54:43,247 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 12:54:43,247 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 12:54:43,249 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 12:54:43,249 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 12:54:43,249 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 12:54:43,249 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 12:54:43,249 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 12:54:43,250 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 12:54:43,250 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 12:54:43,250 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 12:54:43,250 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 12:54:43,250 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
  
先简单介绍一下，logging库提供了多个组件：Logger、Handler、Filter、Formatter。Logger对象提供应用程序可直接使用的接口，Handler发送日志到适当的目的地，Filter提供了过滤日志信息的方法，Formatter指定日志显示格式。  
  
**Logger**  
Logger是一个树形层级结构，输出信息之前都要获得一个Logger（如果没有显示的获取则自动创建并使用root Logger，如第一个例子所示）。  
logger = logging.getLogger()返回一个默认的Logger也即root Logger，并应用默认的日志级别、Handler和Formatter设置。  
当然也可以通过Logger.setLevel(lel)指定最低的日志级别，可用的日志级别有logging.DEBUG、logging.INFO、logging.WARNING、logging.ERROR、logging.CRITICAL。  
Logger.debug()、Logger.info()、Logger.warning()、Logger.error()、Logger.critical()输出不同级别的日志，只有日志等级大于或等于设置的日志级别的日志才会被输出。  
  
我们看到程序中

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. logger.debug('logger debug message')
2. logger.info('logger info message')
3. logger.warning('logger warning message')
4. logger.error('logger error message')
5. logger.critical('logger critical message')

只输出了  
2014-05-06 12:54:43,222 - root - WARNING - logger warning message  
2014-05-06 12:54:43,223 - root - ERROR - logger error message  
2014-05-06 12:54:43,224 - root - CRITICAL - logger critical message  
从这个输出可以看出logger = logging.getLogger()返回的Logger名为root。这里没有用logger.setLevel()显示的为logger设置日志级别，所以使用默认的日志级别WARNIING，故结果只输出了大于等于WARNIING级别的信息。  
  
另外，我们明明通过logger1.setLevel(logging.DEBUG)将logger1的日志级别设置为了DEBUG，为何显示的时候没有显示出DEBUG级别的日志信息，而是从INFO级别的日志开始显示呢？原来logger1和logger2对应的是同一个Logger实例，只要logging.getLogger（name）中名称参数name相同则返回的Logger实例就是同一个，且仅有一个，也即name与Logger实例一一对应。在logger2实例中通过logger2.setLevel(logging.INFO)设置mylogger的日志级别为logging.INFO，所以最后logger1的输出遵从了后来设置的日志级别。

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. logger1 = logging.getLogger('mylogger')
2. logger1.setLevel(logging.DEBUG)
3. logger2 = logging.getLogger('mylogger')
4. logger2.setLevel(logging.INFO)

为什么logger1、logger2对应的每个输出分别显示两次，logger3对应的输出显示3次，logger4对应的输出显示4次......呢？  
这是因为我们通过logger = logging.getLogger()显示的创建了root Logger，而logger1 = logging.getLogger('mylogger')创建了root Logger的孩子(root.)mylogger,logger2同样。  
logger3 = logging.getLogger('mylogger.child1')创建了(root.)mylogger.child1  
logger4 = logging.getLogger('mylogger.child1.child2')创建了(root.)mylogger.child1.child2  
logger5 = logging.getLogger('mylogger.child1.child2.child3')创建了(root.)mylogger.child1.child2.child3  
而孩子,孙子，重孙……既会将消息分发给他的handler进行处理也会传递给所有的祖先Logger处理。  
  
试着注释掉如下一行程序，观察程序输出

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #logger.addHandler(fh)

发现标准输出中每条记录对应两行（因为root Logger默认使用StreamHandler）  
2014-05-06 15:10:10,980 - mylogger - INFO - logger1 info message  
2014-05-06 15:10:10,980 - mylogger - INFO - logger1 info message  
2014-05-06 15:10:10,981 - mylogger - WARNING - logger1 warning message  
2014-05-06 15:10:10,981 - mylogger - WARNING - logger1 warning message  
2014-05-06 15:10:10,982 - mylogger - ERROR - logger1 error message  
2014-05-06 15:10:10,982 - mylogger - ERROR - logger1 error message  
2014-05-06 15:10:10,984 - mylogger - CRITICAL - logger1 critical message  
2014-05-06 15:10:10,984 - mylogger - CRITICAL - logger1 critical message  
……  
而在文件输出中每条记录对应一行（因为我们注释掉了logger.addHandler(fh)，没有对root Logger启用FileHandler）  
2014-05-06 15:10:10,980 - mylogger - INFO - logger1 info message  
2014-05-06 15:10:10,981 - mylogger - WARNING - logger1 warning message  
2014-05-06 15:10:10,982 - mylogger - ERROR - logger1 error message  
2014-05-06 15:10:10,984 - mylogger - CRITICAL - logger1 critical message  
  
孩子,孙子，重孙……可逐层继承来自祖先的日志级别、Handler、Filter设置，也可以通过Logger.setLevel(lel)、Logger.addHandler(hdlr)、Logger.removeHandler(hdlr)、Logger.addFilter(filt)、Logger.removeFilter(filt)。设置自己特别的日志级别、Handler、Filter。若不设置则使用继承来的值。  
  
**Handler**  
上述例子的输出在标准输出和指定的日志文件中均可以看到，这是因为我们定义并使用了两种Handler。

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. fh = logging.FileHandler('/tmp/test.log')
2. ch = logging.StreamHandler()

Handler对象负责发送相关的信息到指定目的地，有几个常用的Handler方法：  
Handler.setLevel(lel):指定日志级别，低于lel级别的日志将被忽略  
Handler.setFormatter()：给这个handler选择一个Formatter  
Handler.addFilter(filt)、Handler.removeFilter(filt)：新增或删除一个filter对象  
  
可以通过addHandler()方法为Logger添加多个Handler：

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. logger.addHandler(fh)
2. logger.addHandler(ch)

有多中可用的Handler：

logging.StreamHandler 可以向类似与sys.stdout或者sys.stderr的任何文件对象(file object)输出信息  
logging.FileHandler 用于向一个文件输出日志信息  
logging.handlers.RotatingFileHandler 类似于上面的FileHandler，但是它可以管理文件大小。当文件达到一定大小之后，它会自动将当前日志文件改名，然后创建一个新的同名日志文件继续输出  
logging.handlers.TimedRotatingFileHandler 和RotatingFileHandler类似，不过，它没有通过判断文件大小来决定何时重新创建日志文件，而是间隔一定时间就自动创建新的日志文件  
logging.handlers.SocketHandler 使用TCP协议，将日志信息发送到网络。  
logging.handlers.DatagramHandler 使用UDP协议，将日志信息发送到网络。  
logging.handlers.SysLogHandler 日志输出到syslog  
logging.handlers.NTEventLogHandler 远程输出日志到Windows NT/2000/XP的事件日志   
logging.handlers.SMTPHandler 远程输出日志到邮件地址  
logging.handlers.MemoryHandler 日志输出到内存中的制定buffer  
logging.handlers.HTTPHandler 通过"GET"或"POST"远程输出到HTTP服务器  
各个Handler的具体用法可查看参考书册：  
https://docs.python.org/2/library/logging.handlers.html#module-logging.handlers  
  
  
**Formatter**  
Formatter对象设置日志信息最后的规则、结构和内容，默认的时间格式为%Y-%m-%d %H:%M:%S。

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #定义Formatter
2. formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
3. #为Handler添加Formatter
4. fh.setFormatter(formatter)
5. ch.setFormatter(formatter)

Formatter参数中可能用到的格式化串参见上文（logging.basicConfig()函数format参数中可能用到的格式化串：）  
  
**Filter**  
限制只有满足过滤规则的日志才会输出。  
比如我们定义了filter = logging.Filter('a.b.c'),并将这个Filter添加到了一个Handler上，则使用该Handler的Logger中只有名字带a.b.c前缀的Logger才能输出其日志。  
  
取消下列两行程序的注释

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #filter = logging.Filter('mylogger.child1.child2')
2. #fh.addFilter(filter)

标准输出中输出结果并没有发生变化，但日志文件输出中只显示了如下内容：  
2014-05-06 15:27:36,227 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:27:36,227 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:27:36,227 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:27:36,227 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:27:36,228 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:27:36,228 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:27:36,228 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:27:36,228 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:27:36,230 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:27:36,230 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:27:36,230 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:27:36,230 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:27:36,232 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:27:36,232 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:27:36,232 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:27:36,232 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:27:36,233 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:27:36,233 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:27:36,233 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:27:36,233 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:27:36,235 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:27:36,235 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:27:36,235 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:27:36,235 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:27:36,235 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:27:36,236 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:27:36,236 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:27:36,236 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:27:36,236 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:27:36,236 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:27:36,238 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:27:36,238 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:27:36,238 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:27:36,238 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:27:36,238 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:27:36,240 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:27:36,240 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:27:36,240 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:27:36,240 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:27:36,240 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:27:36,242 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:27:36,242 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:27:36,242 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:27:36,242 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:27:36,242 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
  
当然也可以直接给Logger加Filter。若为Handler加Filter则所有使用了该Handler的Logger都会受到影响。而为Logger添加Filter只会影响到自身。  
注释掉

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #fh.addFilter(filter)

并取消如下几行的注释

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. #logger.addFilter(filter)
2. #logger1.addFilter(filter)
3. #logger3.addFilter(filter)
4. #logger4.addFilter(filter)

输出结果  
2014-05-06 15:32:10,746 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:32:10,746 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:32:10,746 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:32:10,746 - mylogger.child1.child2 - DEBUG - logger4 debug message  
2014-05-06 15:32:10,748 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:32:10,748 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:32:10,748 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:32:10,748 - mylogger.child1.child2 - INFO - logger4 info message  
2014-05-06 15:32:10,751 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:32:10,751 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:32:10,751 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:32:10,751 - mylogger.child1.child2 - WARNING - logger4 warning message  
2014-05-06 15:32:10,753 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:32:10,753 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:32:10,753 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:32:10,753 - mylogger.child1.child2 - ERROR - logger4 error message  
2014-05-06 15:32:10,754 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:32:10,754 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:32:10,754 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:32:10,754 - mylogger.child1.child2 - CRITICAL - logger4 critical message  
2014-05-06 15:32:10,755 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:32:10,755 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:32:10,755 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:32:10,755 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:32:10,755 - mylogger.child1.child2.child3 - DEBUG - logger5 debug message  
2014-05-06 15:32:10,757 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:32:10,757 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:32:10,757 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:32:10,757 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:32:10,757 - mylogger.child1.child2.child3 - INFO - logger5 info message  
2014-05-06 15:32:10,759 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:32:10,759 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:32:10,759 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:32:10,759 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:32:10,759 - mylogger.child1.child2.child3 - WARNING - logger5 warning message  
2014-05-06 15:32:10,761 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:32:10,761 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:32:10,761 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:32:10,761 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:32:10,761 - mylogger.child1.child2.child3 - ERROR - logger5 error message  
2014-05-06 15:32:10,762 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:32:10,762 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:32:10,762 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:32:10,762 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
2014-05-06 15:32:10,762 - mylogger.child1.child2.child3 - CRITICAL - logger5 critical message  
  
发现root、mylogger、mylogger.child1的输出全部被过滤掉了。

**除了直接在程序中设置Logger，Handler,Filter,Formatter外还可以将这些信息写进配置文件中。**

例如典型的logging.conf

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. [loggers]
2. keys=root,simpleExample
4. [handlers]
5. keys=consoleHandler
7. [formatters]
8. keys=simpleFormatter
10. [logger\_root]
11. level=DEBUG
12. handlers=consoleHandler
14. [logger\_simpleExample]
15. level=DEBUG
16. handlers=consoleHandler
17. qualname=simpleExample
18. propagate=0
20. [handler\_consoleHandler]
21. **class**=StreamHandler
22. level=DEBUG
23. formatter=simpleFormatter
24. args=(sys.stdout,)
26. [formatter\_simpleFormatter]
27. format=%(asctime)s - %(name)s - %(levelname)s - %(message)s
28. datefmt=

程序可以这么写

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. **import** logging
2. **import** logging.config
4. logging.config.fileConfig("logging.conf")    # 采用配置文件
6. # create logger
7. logger = logging.getLogger("simpleExample")
9. # "application" code
10. logger.debug("debug message")
11. logger.info("info message")
12. logger.warn("warn message")
13. logger.error("error message")
14. logger.critical("critical message")

**多模块使用logging**  
logging模块保证在同一个python解释器内，多次调用logging.getLogger('log\_name')都会返回同一个logger实例，即使是在多个模块的情况下。所以典型的多模块场景下使用logging的方式是在main模块中配置logging，这个配置会作用于多个的子模块，然后在其他模块中直接通过getLogger获取Logger对象即可。  
  
main.py：

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. **import** logging
2. **import** logging.config
4. logging.config.fileConfig('logging.conf')
5. root\_logger = logging.getLogger('root')
6. root\_logger.debug('test root logger...')
8. logger = logging.getLogger('main')
9. logger.info('test main logger')
10. logger.info('start import module \'mod\'...')
11. **import** mod
13. logger.debug('let\'s test mod.testLogger()')
14. mod.testLogger()
16. root\_logger.info('finish test...')

子模块mod.py：

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



1. **import** logging
2. **import** submod
4. logger = logging.getLogger('main.mod')
5. logger.info('logger of mod say something...')
7. **def** testLogger():
8. logger.debug('this is mod.testLogger...')
9. submod.tst()

子子模块submod.py：

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)

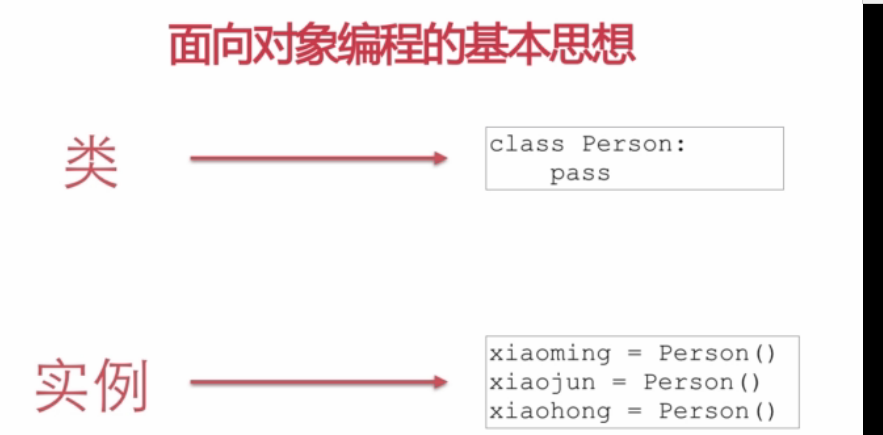


1. **import** logging
3. logger = logging.getLogger('main.mod.submod')
4. logger.info('logger of submod say something...')
6. **def** tst():
7. logger.info('this is submod.tst()...')

**[python]** [view plain](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "view plain)[copy](http://blog.csdn.net/zyz511919766/article/details/25136485" \o "copy)



# 面向对象编程



## 定义类并创建实例

在Python中，类通过 **class**关键字定义。以 **Person** 为例，定义一个**Person类**如下：

class Person(object):

pass

按照 **Python** 的编程习惯，类名以大写字母开头，紧接着是(object)，表示该类是从哪个类继承下来的。类的继承将在后面的章节讲解，现在我们只需要简单地从**object**类继承。

有了Person类的定义，就可以创建出具体的**xiaoming、xiaohong**等实例。创建实例使用 **类名+()**，类似函数调用的形式创建：

xiaoming = Person()

xiaohong = Person()

1.类通过 class 关键字定义。以 Person 为例，定义一个Person类  
2.类名以大写字母开头，紧接着是(object)，表示该类是从哪个类继承下来的  
>>>class Person(object):  
3.创建实例使用 类名+()，类似函数调用的形式创建  
>>>xiaoming = Person()

## 创建实例属性

虽然可以通过**Person**类创建出**xiaoming、xiaohong**等实例，但是这些实例看上除了地址不同外，没有什么其他不同。在现实世界中，区分**xiaoming、xiaohong**要依靠他们各自的名字、性别、生日等属性。

如何让每个实例拥有各自不同的属性？由于Python是动态语言，对每一个实例，都可以直接给他们的属性赋值，例如，给**xiaoming**这个实例加上**name、gender**和**birth**属性：

xiaoming = Person()

xiaoming.name = 'Xiao Ming'

xiaoming.gender = 'Male'

xiaoming.birth = '1990-1-1'

给**xiaohong**加上的属性不一定要和**xiaoming**相同：

xiaohong = Person()

xiaohong.name = 'Xiao Hong'

xiaohong.school = 'No. 1 High School'

xiaohong.grade = 2

实例的属性可以像普通变量一样进行操作：

xiaohong.grade = xiaohong.grade + 1

## 初始化实例属性

虽然我们可以自由地给一个实例绑定各种属性，但是，现实世界中，一种类型的实例应该拥有相同名字的属性。例如，**Person类**应该在创建的时候就拥有 **name、gender**和 **birth**属性，怎么办？

在定义 Person 类时，可以为Person类添加一个特殊的**\_\_init\_\_()**方法，当创建实例时，**\_\_init\_\_()**方法被自动调用，我们就能在此为每个实例都统一加上以下属性：

class Person(object):

def \_\_init\_\_(self, name, gender, birth):

self.name = name

self.gender = gender

self.birth = birth

**\_\_init\_\_()**方法的第一个参数必须是 **self**（也可以用别的名字，但建议使用习惯用法），后续参数则可以自由指定，和定义函数没有任何区别。

相应地，创建实例时，就必须要提供除 **self**以外的参数：

xiaoming = Person('Xiao Ming', 'Male', '1991-1-1')

xiaohong = Person('Xiao Hong', 'Female', '1992-2-2')

有了**\_\_init\_\_()**方法，每个Person实例在创建时，都会有 **name、gender**和 **birth**这3个属性，并且，被赋予不同的属性值，访问属性使用.操作符：

print xiaoming.name

# 输出 'Xiao Ming'

print xiaohong.birth

# 输出 '1992-2-2'

要特别注意的是，初学者定义**\_\_init\_\_()**方法常常忘记了 self 参数：

>>> class Person(object):

... def \_\_init\_\_(name, gender, birth):

... pass

...

>>> xiaoming = Person('Xiao Ming', 'Male', '1990-1-1')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: \_\_init\_\_() takes exactly 3 arguments (4 given)

这会导致创建失败或运行不正常，因为第一个参数name被Python解释器传入了实例的引用，从而导致整个方法的调用参数位置全部没有对上。

## 访问限制

我们可以给一个实例绑定很多属性，如果有些属性不希望被外部访问到怎么办？

Python对属性权限的控制是通过属性名来实现的，如果一个属性由双下划线开头(\_\_)，该属性就无法被外部访问。看例子：

class Person(object):

def \_\_init\_\_(self, name):

self.name = name

self.\_title = 'Mr'

self.\_\_job = 'Student'

p = Person('Bob')

print p.name

# => Bob

print p.\_title

# => Mr

print p.\_\_job

# => Error

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'Person' object has no attribute '\_\_job'

可见，只有以双下划线开头的"**\_\_job**"不能直接被外部访问。

但是，如果一个属性以"\_\_xxx\_\_"的形式定义，那它又可以被外部访问了，以"\_\_xxx\_\_"定义的属性在Python的类中被称为特殊属性，有很多预定义的特殊属性可以使用，通常我们不要把普通属性用"**\_\_xxx\_\_**"定义。

以单下划线开头的属性"\_xxx"虽然也可以被外部访问，但是，按照习惯，他们不应该被外部访问。

## 创建类属性

**类是模板，而实例则是根据类创建的对象。**

绑定在一个实例上的属性不会影响其他实例，但是，类本身也是一个对象，如果在类上绑定一个属性，则所有实例都可以访问类的属性，并且，所有实例访问的类属性都是同一个！也就是说，实例属性每个实例各自拥有，互相独立，而类属性有且只有一份。

定义类属性可以直接在 **class**中定义：

class Person(object):

address = 'Earth'

def \_\_init\_\_(self, name):

self.name = name

因为类属性是直接绑定在类上的，所以，访问类属性不需要创建实例，就可以直接访问：

print Person.address

# => Earth

对一个实例调用类的属性也是可以访问的，所有实例都可以访问到它所属的类的属性：

p1 = Person('Bob')

p2 = Person('Alice')

print p1.address

# => Earth

print p2.address

# => Earth

由于Python是动态语言，类属性也是可以动态添加和修改的：

Person.address = 'China'

print p1.address

# => 'China'

print p2.address

# => 'China'

因为类属性只有一份，所以，当**Person**类的**address**改变时，所有实例访问到的类属性都改变了。

## 类属性和实例属性名字冲突怎么办

修改类属性会导致所有实例访问到的类属性全部都受影响，但是，如果在实例变量上修改类属性会发生什么问题呢？

class Person(object):

address = 'Earth'

def \_\_init\_\_(self, name):

self.name = name

p1 = Person('Bob')

p2 = Person('Alice')

print 'Person.address = ' + Person.address

p1.address = 'China'

print 'p1.address = ' + p1.address

print 'Person.address = ' + Person.address

print 'p2.address = ' + p2.address

结果如下：

Person.address = Earth

p1.address = China

Person.address = Earth

p2.address = Earth

我们发现，在设置了 **p1.address = 'China'**后，p1访问 address 确实变成了 'China'，但是，Person.address和p2.address仍然是'Earch'，怎么回事？

原因是 **p1.address = 'China'并没有改变 Person 的 address**，而是给 **p1这个实例绑定了实例属性address** ，对p1来说，它有一个实例属性address（值是'China'），而它所属的类Person也有一个类属性address，所以:

**访问 p1.address 时**，优先查找实例属性，返回'China'。

**访问 p2.address 时**，p2没有实例属性address，但是有类属性address，因此返回'Earth'。

可见，当实例属性和类属性重名时，实例属性优先级高，它将屏蔽掉对类属性的访问。

当我们把 p1 的 address 实例属性删除后，访问 p1.address 就又返回类属性的值 'Earth'了：

del p1.address

print p1.address

# => Earth

可见，千万不要在实例上修改类属性，它实际上并没有修改类属性，而是给实例绑定了一个实例属性。

## 定义实例方法

一个实例的私有属性就是以\_\_开头的属性，无法被外部访问，那这些属性定义有什么用？

虽然私有属性无法从外部访问，但是，从类的内部是可以访问的。除了可以定义实例的属性外，还可以定义实例的方法。

**实例的方法**就是在类中定义的函数，它的第一个参数永远是 self，指向调用该方法的实例本身，其他参数和一个普通函数是完全一样的：

class Person(object):

def \_\_init\_\_(self, name):

self.\_\_name = name

def get\_name(self):

return self.\_\_name

**get\_name(self)**就是一个实例方法，它的第一个参数是self。\_**\_init\_\_(self, name)**其实也可看做是一个特殊的实例方法。

调用实例方法必须在实例上调用：

p1 = Person('Bob')

print p1.get\_name() # self不需要显式传入

# => Bob

在实例方法内部，可以访问所有实例属性，这样，如果外部需要访问私有属性，可以通过方法调用获得，这种数据封装的形式除了能保护内部数据一致性外，还可以简化外部调用的难度。

## 方法也是属性

我们在 **class** 中定义的实例方法其实也是属性，它实际上是一个函数对象：

class Person(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

def get\_grade(self):

return 'A'

p1 = Person('Bob', 90)

print p1.get\_grade

# => <bound method Person.get\_grade of <\_\_main\_\_.Person object at 0x109e58510>>

print p1.get\_grade()

# => A

也就是说，**p1.get\_grade**返回的是一个函数对象，但这个函数是一个绑定到实例的函数，**p1.get\_grade()**才是方法调用。

因为方法也是一个属性，所以，它也可以动态地添加到实例上，只是需要用 types.MethodType() 把一个函数变为一个方法：

import types

def fn\_get\_grade(self):

if self.score >= 80:

return 'A'

if self.score >= 60:

return 'B'

return 'C'

class Person(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

p1 = Person('Bob', 90)

p1.get\_grade = types.MethodType(fn\_get\_grade, p1, Person)

print p1.get\_grade()

# => A

p2 = Person('Alice', 65)

print p2.get\_grade()

# ERROR: AttributeError: 'Person' object has no attribute 'get\_grade'

# 因为p2实例并没有绑定get\_grade

给一个实例动态添加方法并不常见，直接在class中定义要更直观。

## 定义类方法（@classmethod）

和属性类似，方法也分实例方法和类方法。

在**class**中定义的全部是实例方法，实例方法第一个参数 **self**是实例本身。

要在class中定义类方法，需要这么写：

class Person(object):

count = 0

@classmethod

def how\_many(cls):

return cls.count

def \_\_init\_\_(self, name):

self.name = name

Person.count = Person.count + 1

print Person.how\_many()

p1 = Person('Bob')

print Person.how\_many()

通过标记一个 @classmethod，该方法将绑定到**Person**类上，而非类的实例。类方法的第一个参数将传入类本身，通常将参数名命名为**cls**，上面的 **cls.count**实际上相当于 **Person.count**。

因为是在类上调用，而非实例上调用，因此类方法无法获得任何实例变量，只能获得类的引用。

## Note

### \_\_init\_\_

实例化传参数的时候构造函数会将参数变为类内的全局变量

class test(object):

"""

get被称之为test对象的方法

"""

def \_\_init\_\_(self,var1):

self.var1 = var1

def get(self,a=None):

return self.var1

pass

a = test(“zhagnshuo”)

print a.get()

输出的是“zhagnshuo”

# 继承



## 继承一个类

如果已经定义了**Person**类，需要定义新的**Student**和**Teacher**类时，可以直接从Person类继承：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

定义**Student**类时，只需要把额外的属性加上，例如score：

class Student(Person):

def \_\_init\_\_(self, name, gender, score):

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

一定要用 super(Student, self).\_\_init\_\_(name, gender) 去初始化父类，否则，继承自 **Person** 的 **Student** 将没有**name** 和 **gender**。

函数**super(Student, self)**将返回当前类继承的父类，即**Person**，然后调用**\_\_init\_\_()**方法，注意self参数已在super()中传入，在\_\_init\_\_()中将隐式传递，不需要写出（也不能写）。

## 判断类型

函数**isinstance()**可以判断一个变量的类型，既可以用在Python内置的数据类型如**str、list、dict**，也可以用在我们自定义的类，它们本质上都是数据类型。

假设有如下的 **Person、Student** 和 **Teacher**的定义及继承关系如下：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

class Student(Person):

def \_\_init\_\_(self, name, gender, score):

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

class Teacher(Person):

def \_\_init\_\_(self, name, gender, course):

super(Teacher, self).\_\_init\_\_(name, gender)

self.course = course

p = Person('Tim', 'Male')

s = Student('Bob', 'Male', 88)

t = Teacher('Alice', 'Female', 'English')

当我们拿到变量 **p、s、t**时，可以使用 **isinstance** 判断类型：

>>> isinstance(p, Person)

True # p是Person类型

>>> isinstance(p, Student)

False # p不是Student类型

>>> isinstance(p, Teacher)

False # p不是Teacher类型

这说明在继承链上，一个父类的实例不能是子类类型，因为子类比父类多了一些属性和方法。

**我们再考察 s ：**

>>> isinstance(s, Person)

True # s是Person类型

>>> isinstance(s, Student)

True # s是Student类型

>>> isinstance(s, Teacher)

False # s不是Teacher类型

**s** 是Student类型，不是Teacher类型，这很容易理解。但是，**s** 也是Person类型，因为Student继承自Person，虽然它比Person多了一些属性和方法，但是，把**s** 看成Person的实例也是可以的。

这说明在一条继承链上，一个实例可以看成它本身的类型，也可以看成它父类的类型。

## 多态

类具有继承关系，并且子类类型可以向上转型看做父类类型，如果我们从 **Person**派生出 **Student**和**Teacher**，并都写了一个 **whoAmI()**方法：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def whoAmI(self):

return 'I am a Person, my name is %s' % self.name

class Student(Person):

def \_\_init\_\_(self, name, gender, score):

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

def whoAmI(self):

return 'I am a Student, my name is %s' % self.name

class Teacher(Person):

def \_\_init\_\_(self, name, gender, course):

super(Teacher, self).\_\_init\_\_(name, gender)

self.course = course

def whoAmI(self):

return 'I am a Teacher, my name is %s' % self.name

在一个函数中，如果我们接收一个变量**x**，则无论该 **x**是**Person、Student**还是 **Teacher**，都可以正确打印出结果：

def who\_am\_i(x):

print x.whoAmI()

p = Person('Tim', 'Male')

s = Student('Bob', 'Male', 88)

t = Teacher('Alice', 'Female', 'English')

who\_am\_i(p)

who\_am\_i(s)

who\_am\_i(t)

运行结果：

I am a Person, my name is Tim

I am a Student, my name is Bob

I am a Teacher, my name is Alice

这种行为称为多态。也就是说，方法调用将作用在 **x** 的实际类型上。**s** 是**Student**类型，它实际上拥有自己的**whoAmI()**方法以及从 Person继承的 whoAmI方法，但调用 **s.whoAmI()**总是先查找它自身的定义，如果没有定义，则顺着继承链向上查找，直到在某个父类中找到为止。

由于Python是动态语言，所以，传递给函数 **who\_am\_i(x)**的参数 **x** 不一定是 Person 或 Person 的子类型。任何数据类型的实例都可以，只要它**有一个whoAmI()**的方法即可：

class Book(object):

def whoAmI(self):

return 'I am a book'

这是动态语言和静态语言（例如Java）最大的差别之一。动态语言调用实例方法，不检查类型，只要方法存在，参数正确，就可以调用。

## 多重继承

除了从一个父类继承外，Python允许从多个父类继承，称为多重继承。

多重继承的继承链就不是一棵树了，它像这样：

class A(object):

def \_\_init\_\_(self, a):

print 'init A...'

self.a = a

class B(A):

def \_\_init\_\_(self, a):

super(B, self).\_\_init\_\_(a)

print 'init B...'

class C(A):

def \_\_init\_\_(self, a):

super(C, self).\_\_init\_\_(a)

print 'init C...'

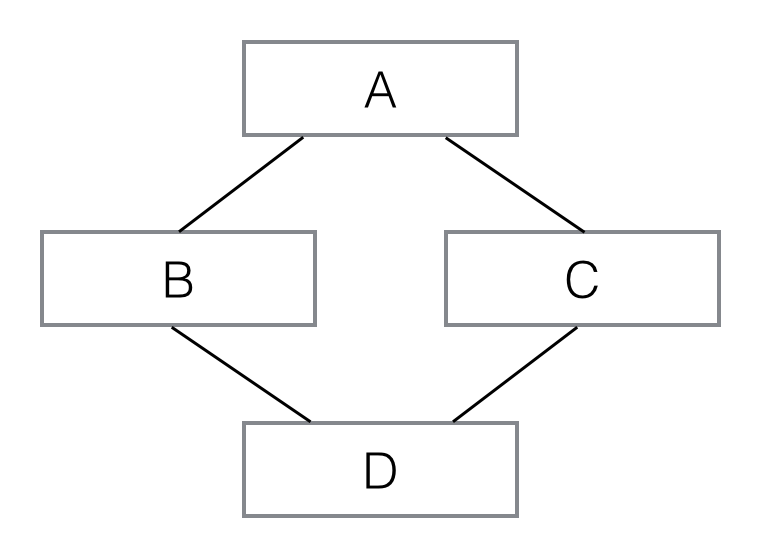
class D(B, C):

def \_\_init\_\_(self, a):

super(D, self).\_\_init\_\_(a)

print 'init D...'

看下图:



像这样，**D**同时继承自 **B** 和 **C**，也就是 **D** 拥有了 **A、B、C** 的全部功能。多重继承通过**super()**调用**\_\_init\_\_()**方法时，**A**虽然被继承了两次，但**\_\_init\_\_()**只调用一次：

>>> d = D('d')

init A...

init C...

init B...

init D...

**多重继承的目的**是从两种继承树中分别选择并继承出子类，以便组合功能使用。

举个例子，Python的网络服务器有**TCPServer、UDPServer、UnixStreamServer、UnixDatagramServer**，而服务器运行模式有 **多进程ForkingMixin**和 **多线程ThreadingMixin**两种。

要创建多进程模式的**TCPServer**：

class MyTCPServer(TCPServer, ForkingMixin)

pass

要创建多线程模式的**UDPServer**：

class MyUDPServer(UDPServer, ThreadingMixin):

pass

如果没有多重继承，要实现上述所有可能的组合需要 4x2=8 个子类。

## 获取对象信息

拿到一个变量，除了用 **isinstance()**判断它是否是某种类型的实例外，还有没有别的方法获取到更多的信息呢？

例如，已有定义：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

class Student(Person):

def \_\_init\_\_(self, name, gender, score):

super(Student, self).\_\_init\_\_(name, gender)

self.score = score

def whoAmI(self):

return 'I am a Student, my name is %s' % self.name

首先可以用 **type()**函数获取变量的类型，它返回一个 **Type**对象：

>>> type(123)

<type 'int'>

>>> s = Student('Bob', 'Male', 88)

>>> type(s)

<class '\_\_main\_\_.Student'>

其次，可以用 **dir()**函数获取变量的所有属性：

>>> dir(123) # 整数也有很多属性...

['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_class\_\_', '\_\_cmp\_\_', ...]

>>> dir(s)

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_getattribute\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'gender', 'name', 'score', 'whoAmI']

对于实例变量，**dir()**返回所有实例属性，包括`**\_\_class\_\_**`这类有特殊意义的属性。注意到方法`**whoAmI**`也是 **s**的一个属性。

如何去掉**`\_\_xxx\_\_`**这类的特殊属性，只保留我们自己定义的属性？回顾一下**filter()**函数的用法。

**dir()**返回的属性是字符串列表，如果已知一个属性名称，要获取或者设置对象的属性，就需要用 **getattr()**和**setattr( )**函数了：

>>> getattr(s, 'name') # 获取name属性

'Bob'

>>> setattr(s, 'name', 'Adam') # 设置新的name属性

>>> s.name

'Adam'

>>> getattr(s, 'age') # 获取age属性，但是属性不存在，报错：

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'Student' object has no attribute 'age'

>>> getattr(s, 'age', 20) # 获取age属性，如果属性不存在，就返回默认值20：

20

# 魔术方法（特殊方法）

## [Python的官方文档](http://docs.python.org/2/reference/datamodel.html" \l "special-method-names)



## \_\_str\_\_和\_\_repr\_\_

如果要把一个类的实例变成**str**，就需要实现特殊方法\_\_str\_\_()：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def \_\_str\_\_(self):

return '(Person: %s, %s)' % (self.name, self.gender)

现在，在交互式命令行下用 **print**试试：

>>> p = Person('Bob', 'male')

>>> print p

(Person: Bob, male)

但是，如果直接敲变量 **p**：

>>> p

<main.Person object at 0x10c941890>

似乎\_\_str\_\_() 不会被调用。

因为 Python 定义了**\_\_str\_\_()**和**\_\_repr\_\_()**两种方法，\_\_str\_\_()用于显示给用户，而**\_\_repr\_\_()**用于显示给开发人员。

有一个偷懒的定义\_\_repr\_\_的方法：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def \_\_str\_\_(self):

return '(Person: %s, %s)' % (self.name, self.gender)

\_\_repr\_\_ = \_\_str\_\_

## \_\_cmp\_\_

对 **int**、**str**等内置数据类型排序时，Python的 **sorted()**按照默认的比较函数 **cmp**排序，但是，如果对一组 **Student**类的实例排序时，就必须提供我们自己的特殊方法 \_\_cmp\_\_()：

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

def \_\_str\_\_(self):

return '(%s: %s)' % (self.name, self.score)

\_\_repr\_\_ = \_\_str\_\_

def \_\_cmp\_\_(self, s):

if self.name < s.name:

return -1

elif self.name > s.name:

return 1

else:

return 0

上述 Student 类实现了\_\_cmp\_\_()方法，**\_\_cmp\_\_**用实例自身**self**和传入的实例 **s**进行比较，如果**self** 应该排在前面，就返回 -1，如果 **s** 应该排在前面，就返回1，如果两者相当，返回 0。

Student类实现了按name进行排序：

>>> L = [Student('Tim', 99), Student('Bob', 88), Student('Alice', 77)]

>>> print sorted(L)

[(Alice: 77), (Bob: 88), (Tim: 99)]

**注意**: 如果list不仅仅包含 Student 类，则 \_\_cmp\_\_ 可能会报错： 请思考如何解决。

L = [Student('Tim', 99), Student('Bob', 88), 100, 'Hello']

print sorted(L)

## \_\_len\_\_

如果一个类表现得像一个list，要获取有多少个元素，就得用 **len()**函数。

要让 **len()**函数工作正常，类必须提供一个特殊方法\_\_len\_\_()，它返回元素的个数。

例如，我们写一个 Students 类，把名字传进去：

class Students(object):

def \_\_init\_\_(self, \*args):

self.names = args

def \_\_len\_\_(self):

return len(self.names)

只要正确实现了**\_\_len\_\_()**方法，就可以用**len()**函数返回**Students**实例的“长度”：

>>> ss = Students('Bob', 'Alice', 'Tim')

>>> print len(ss)

3

## 数学运算-有理数（分数）

Python 提供的基本数据类型 **int、float**可以做整数和浮点的四则运算以及乘方等运算。

但是，四则运算不局限于int和float，还可以是有理数、矩阵等。

要表示有理数，可以用一个**Rational类**来表示：

class Rational(object):

def \_\_init\_\_(self, p, q):

self.p = p

self.q = q

p、q 都是整数，表示有理数 p/q。

如果要让**Rational进行+运算**，需要正确实现\_\_add\_\_：

class Rational(object):

def \_\_init\_\_(self, p, q):

self.p = p

self.q = q

def \_\_add\_\_(self, r):

return Rational(self.p \* r.q + self.q \* r.p, self.q \* r.q)

def \_\_str\_\_(self):

return '%s/%s' % (self.p, self.q)

\_\_repr\_\_ = \_\_str\_\_

现在可以试试有理数加法：

>>> r1 = Rational(1, 3)

>>> r2 = Rational(1, 2)

>>> print r1 + r2

5/6

## 类型转换

**Rational**类实现了有理数运算，但是，如果要把结果转为 **int**或**float** 怎么办？

考察整数和浮点数的转换：

>>> int(12.34)

12

>>> float(12)

12.0

如果要把**Rational**转为**int**，应该使用：

r = Rational(12, 5)

n = int(r)

要让**int()**函数正常工作，只需要实现特殊方法\_\_int\_\_():

class Rational(object):

def \_\_init\_\_(self, p, q):

self.p = p

self.q = q

def \_\_int\_\_(self):

return self.p // self.q

**结果如下：**

>>> print int(Rational(7, 2))

3

>>> print int(Rational(1, 3))

0

同理，要让**float()**函数正常工作，只需要实现特殊方法**\_\_float\_\_()**。

## @property

考察 **Student**类：

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.score = score

当我们想要修改一个**Student** 的 **scroe** 属性时，可以这么写：

s = Student('Bob', 59)

s.score = 60

但是也可以这么写：

s.score = 1000

显然，直接给属性赋值无法检查分数的有效性。

如果利用两个方法：

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.\_\_score = score

def get\_score(self):

return self.\_\_score

def set\_score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

这样一来，**s.set\_score(1000)** 就会报错。

这种使用 **get/set**方法来封装对一个属性的访问在许多面向对象编程的语言中都很常见。

但是写**s.get\_score()** 和 **s.set\_score()**没有直接写 **s.score** 来得直接。

有没有两全其美的方法？----有。

因为Python支持高阶函数，在函数式编程中我们介绍了装饰器函数，可以用装饰器函数把 **get/set**方法“装饰”成属性调用：

class Student(object):

def \_\_init\_\_(self, name, score):

self.name = name

self.\_\_score = score

@property

def score(self):

return self.\_\_score

@score.setter

def score(self, score):

if score < 0 or score > 100:

raise ValueError('invalid score')

self.\_\_score = score

**注意:** 第一个score(self)是get方法，用@property装饰，第二个score(self, score)是set方法，用@score.setter装饰，@score.setter是前一个@property装饰后的副产品。

现在，就可以像使用属性一样设置score了：

>>> s = Student('Bob', 59)

>>> s.score = 60

>>> print s.score

60

>>> s.score = 1000

Traceback (most recent call last):

...

ValueError: invalid score

说明对**score**赋值实际调用的是 **set方法**。

## \_\_slots\_\_

由于Python是动态语言，任何实例在运行期都可以动态地添加属性。

如果要限制添加的属性，例如，**Student**类只允许添加 **name、gender**和**score**这3个属性，就可以利用Python的一个特殊的**\_\_slots\_\_**来实现。

顾名思义，**\_\_slots\_\_**是指一个类允许的属性列表：

class Student(object):

\_\_slots\_\_ = ('name', 'gender', 'score')

def \_\_init\_\_(self, name, gender, score):

self.name = name

self.gender = gender

self.score = score

现在，对实例进行操作：

>>> s = Student('Bob', 'male', 59)

>>> s.name = 'Tim' # OK

>>> s.score = 99 # OK

>>> s.grade = 'A'

Traceback (most recent call last):

...

AttributeError: 'Student' object has no attribute 'grade'

**\_\_slots\_\_**的目的是限制当前类所能拥有的属性，如果不需要添加任意动态的属性，使用**\_\_slots\_\_**也能节省内存。

## \_\_call\_\_

在Python中，函数其实是一个对象：

>>> f = abs

>>> f.\_\_name\_\_

'abs'

>>> f(-123)

123

由于 **f** 可以被调用，所以，**f** 被称为可调用对象。

所有的函数都是可调用对象。

一个类实例也可以变成一个可调用对象，只需要实现一个特殊方法**\_\_call\_\_()**。

我们把**Person**类变成一个可调用对象：

class Person(object):

def \_\_init\_\_(self, name, gender):

self.name = name

self.gender = gender

def \_\_call\_\_(self, friend):

print 'My name is %s...' % self.name

print 'My friend is %s...' % friend

现在可以对**Person**实例直接调用：

>>> p = Person('Bob', 'male')

>>> p('Tim')

My name is Bob...

My friend is Tim...

单看**p('Tim')**你无法确定 **p** 是一个函数还是一个类实例，所以，在Python中，函数也是对象，对象和函数的区别并不显著。

# 错误、调试和测试

## [错误处理](http://www.liaoxuefeng.com/wiki/001374738125095c955c1e6d8bb493182103fac9270762a000/001386832284796780f5db7b5744bf9989f8d845ef77712000)

# 搭建环境

## 安装python

## 安装pip：

**使用pip工具安装（推荐）**

    先安装pip工具，下载[get-pip](http://pygo.qiniudn.com/get-pip.py" \t "_blank) ，命令行进入到get-pip.py存放的路径

    运行命令

python get-pip.py

    pip 安装完成后，windows下需要配置环境变量path，增加pip工具路径（[你的Python安装路径]\Scripts，例如：C:\Python27\Scripts）到path中。

## 开发编辑器或者IDE

一个好的代码编辑器或IDE可以让编写程序更加舒适、高效。

Python是脚本语言，编写Python代码，使用最基本的文本编辑器既可完成，但如果没有语法高亮、自动缩进、语法提示、资源管理等功能，就不能满足高效开发的要求。

    1.IDLE

        这是Python自带的编辑器，Python安装后就有，功能简洁，语法高亮，带有编辑器、调试器、交互命令行。初学者可以先从它开始。

    2.Eclipse

        一个夸平台的老牌集成开发环境，如果你曾经是Java开发者一定不会陌生。Eclipse安装PyDev插件后就可以成为一个非常优秀的Python开发环境，语法高亮、代码重构、代码补全、拼写检查、调试器等功能都具备。

[Eclipse官网](http://www.eclipse.org/" \t "_blank)

[PyDev下载](http://pydev.sourceforge.net/" \t "_blank)

    3.Microsoft Visual Studio

        微软的集成开发环境，主要用于.net项目的开发，微软官方提供一个插件Python Tools for Visual Studio，安装此插件后，Visual Studio就可以进行Python开发了，除了基础高亮、代码补全、重构等的功能外，还可以断点调试

[Visual Studio官网](http://www.visualstudio.com/" \t "_blank)

[Python Tools for Visual Studio官网](http://pytools.codeplex.com/" \t "_blank)

    4.Sublime Text

        Sublime Text 是一个很性感的代码编辑器，外观优雅、简洁，非常Cool的特征功能，还可以方便的根据自己的习惯安装各种插件。跨平台，半免费（可以购买授权版，也可以完全免费的使用，只是偶尔在保存文件时会弹一个是否购买的提示窗口，不影响任何功能使用）

[Sublime Text官网](http://www.sublimetext.com/" \t "_blank)

    5.PyCharm

        PyCharm由JetBrains出品的针对Python开发的IDE产品，功能丰富而强大，同样可以通过安装插件扩展，是目前最理想的Python IDE，由于PyCharm的智能和强大，并不推荐新手立即就使用PyCharm，否则可能会太过依赖它而遗漏掉学习Python时需要掌握的知识点。

[PyCharm官网](http://www.jetbrains.com/pycharm/" \t "_blank)

## 创建第一个Python项目“Hello World！”

配置好Python的开发环境并且选择好开发工具后，我们可以开始第一个Python项目了。

惯例，做一个Hello World的项目吧

1.新建一个文件 helloworld.py

2.写入以下代码

print "Hello world!"

完成

是不是太简单了。Python作为脚本语言，就是这么简单，不过我们可以多使用一些Python的特征来做一个稍微高级点的Hello World！

#!/usr/bin/env python

# -\*- coding:utf-8 -\*-

#hello 函数，需要youname 参数，返回字符串

def hello(youname):

return "Hello, %s"%(youname,)

#如果该脚本独立运行

if \_\_name\_\_ == "\_\_main\_\_":

print "What's you name?"    #输出一个字符串，询问名称

youname = raw\_input("my name is: ")    #捕捉用户输入，并保存到youname变量中

so\_i\_say = hello(youname)    #将用户输入作为参数，调用hello函数，并将返回值保存到so\_i\_say变量中

print so\_i\_say    #打印 so\_i\_say

结果

> What's you name?

> my name is: Zeta

> Hello, Zeta

看，使用Python很简单的完成了一次对话，不是吗？

下面来看看它是如何工作的。

第一行是告诉操作系统用哪个Python解释器执行代码，本例中的写法是告诉系统使用系统环境配置里的python解释器，你也可以写入一个完整python解释器路径，例如 #!/usr/bin/python

第二行 # **-\*- coding:utf-8 -\*-**是告诉解释器，这个脚本里的文本编码是utf-8，如果没有这行代码，Python解释器会默认使用ASCII作为脚本的编码，当脚本中出现中文、西欧字符、日文、韩文的时候就会产生异常。

def hello(youname):

return "Hello, %s"%(youname,)

def 定义一个函数，函数名是hello，该函数需要一个参数 youname

函数体返回一个字符串。%s是一个字符串占位符，它会被youname替换掉。

if \_\_name\_\_ == "\_\_main\_\_":

print "What's you name?"

youname = raw\_input("my name is: ")

so\_i\_say = hello(youname)

print so\_i\_say

if \_\_name\_\_=="\_\_main\_\_" 用于判断这个脚本是独立运行，还是被作为模块导入到别的脚本的

如果该脚本是独立运行的， \_\_name\_\_=="\_\_main\_\_" 返回True

raw\_input 是Python标准库里自带的函数，用于等待并捕获用户输入。

print 也是Python标准库里自带的函数，用于打印到控制台

    注:本里以Python2.x的代码作为范例，Python3.x稍有区别

# 网页

[Python爬虫框架Scrapy实战之定向批量获取职位招聘信息](http://blog.csdn.net/hantangsongming/article/details/24454453)

[17个新手常见Python运行时错误](http://www.oschina.net/question/89964_62779)

[python 机器学习的开发环境搭建（numpy，scipy,matplotlib）](http://blog.chinaunix.net/uid-26642637-id-4543576.html)

[Python的机器学习库汇总与梳理](http://blog.chinaunix.net/uid-22414998-id-4143709.html)

[UFLDL教程](http://deeplearning.stanford.edu/wiki/index.php/UFLDL教程)

**[BVLC](https://github.com/BVLC)/**[caffe](https://github.com/BVLC/caffe)机器学习的一个框架（github）

[离不开大数据的社会计算](http://blog.chinaunix.net/uid/22414998.html)

[机器学习](https://www.coursera.org/course/ml?from_restricted_preview=1&course_id=16&r=https%3A%2F%2Fclass.coursera.org%2Fml%2Fclass)



**[机器学习课程（吴恩达）](http://v.163.com/special/opencourse/machinelearning.html)**

[【原】斯坦福大学Andrew Ng教授主讲的《机器学习》公开课观后感](http://liliphd.iteye.com/blog/1929358)

# [机器学习和深度学习学习资料](http://suanfazu.com/t/ji-qi-xue-xi-he-shen-du-xue-xi-xue-xi-zi-liao/126)

比较全面的收集了机器学习的介绍文章，从感知机、神经网络、决策树、SVM、Adaboost到随机森林、Deep Learning。

* [《机器学习经典论文/survey合集》](http://suanfazu.com/t/ji-qi-xue-xi-jing-dian-lun-wen-slash-surveyhe-ji/14)  
  介绍：看题目你已经知道了是什么内容,没错。里面有很多经典的机器学习论文值得仔细与反复的阅读。
* [《Brief History of Machine Learning》85](http://www.erogol.com/brief-history-machine-learning/)  
  介绍:这是一篇介绍机器学习历史的文章，介绍很全面，从感知机、神经网络、决策树、SVM、Adaboost到随机森林、Deep Learning.
* [《Deep Learning in Neural Networks: An Overview》43](http://www.idsia.ch/~juergen/DeepLearning15May2014.pdf)  
  介绍:这是瑞士人工智能实验室Jurgen Schmidhuber写的最新版本《神经网络与深度学习综述》本综述的特点是以时间排序，从1940年开始讲起，到60-80年代，80-90年代，一直讲到2000年后及最近几年的进展。涵盖了deep learning里各种tricks，引用非常全面.
* [《A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library》36](http://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/)  
  介绍:这是一份python机器学习库,如果您是一位python工程师而且想深入的学习机器学习.那么这篇文章或许能够帮助到你.
* [《How to Layout and Manage Your Machine Learning Project》21](http://machinelearningmastery.com/how-to-layout-and-manage-your-machine-learning-project/)  
  介绍:这一篇介绍如果设计和管理属于你自己的机器学习项目的文章，里面提供了管理模版、数据管理与实践方法.
* [《Machine Learning is Fun!》33](https://medium.com/code-poet/80ea3ec3c471)  
  介绍:如果你还不知道什么是机器学习，或则是刚刚学习感觉到很枯燥乏味。那么推荐一读。这篇文章已经被翻译成中文,如果有兴趣可以移步[《有趣的机器学习：最简明入门指南》](http://suanfazu.com/shuoit/20140512/339858.html)
* [《R语言参考卡片》18](http://cran.r-project.org/doc/contrib/Liu-R-refcard.pdf)  
  介绍:R语言是机器学习的主要语言,有很多的朋友想学习R语言，但是总是忘记一些函数与关键字的含义。那么这篇文章或许能够帮助到你
* [《Choosing a Machine Learning Classifier》17](http://blog.echen.me/2011/04/27/choosing-a-machine-learning-classifier/)  
  介绍:我该如何选择机器学习算法，这篇文章比较直观的比较了Naive Bayes，Logistic Regression，SVM，决策树等方法的优劣，另外讨论了样本大小、Feature与Model权衡等问题。此外还有已经翻译了的版本:[18http://www.52ml.net/15063.html18](http://www.52ml.net/15063.html)
* [《An Introduction to Deep Learning: From Perceptrons to Deep Networks》11](http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks)  
  介绍：深度学习概述：从感知机到深度网络，作者对于例子的选择、理论的介绍都很到位，由浅入深。翻译版本：[21http://www.cnblogs.com/xiaowanyer/p/3701944.html21](http://www.cnblogs.com/xiaowanyer/p/3701944.html)
* [《The LION Way: Machine Learning plus Intelligent Optimization》19](http://www.lionsolver.com/LIONbook/)  
  介绍:<机器学习与优化>这是一本机器学习的小册子, 短短300多页道尽机器学习的方方面面. 图文并茂, 生动易懂, 没有一坨坨公式的烦恼. 适合新手入门打基础, 也适合老手温故而知新. 比起MLAPP/PRML等大部头, 也许这本你更需要!具体内容推荐阅读:[8http://intelligent-optimization.org/LIONbook/8](http://intelligent-optimization.org/LIONbook/)
* [《深度学习与统计学习理论》47](http://1.guzili.sinaapp.com/?p=174)  
  介绍:作者是来自百度，不过他本人已经在2014年4月份申请离职了。但是这篇文章很不错如果你不知道深度学习与支持向量机/统计学习理论有什么联系？那么应该立即看看这篇文章.
* [《计算机科学中的数学》37](http://courses.csail.mit.edu/6.042/fall13/mcs.pdf)  
  介绍:这本书是由谷歌公司和MIT共同出品的计算机科学中的数学：Mathematics for Computer Science，Eric Lehman et al 2013 。分为5大部分：1）证明，归纳。2）结构，数论，图。3）计数，求和，生成函数。4）概率，随机行走。5）递归。等等
* [《数据科学入门》20](http://zh.scribd.com/doc/223731055/Data-Science-Book-v-2)  
  介绍:这是一本由雪城大学新编的第二版《数据科学入门》教材：偏实用型，浅显易懂，适合想学习R语言的同学选读。
* [《Twenty Questions for Donald Knuth》6](http://www.informit.com/articles/article.aspx?p=2213858)  
  介绍:这并不是一篇文档或书籍。这是篇向图灵奖得主Donald Knuth提问记录稿： 近日， Charles Leiserson, Al Aho, Jon Bentley等大神向Knuth提出了20个问题，内容包括TAOCP，P/NP问题，图灵机，逻辑，以及为什么大神不用电邮等等。
* [《Automatic Construction and Natural-Language Description of Nonparametric Regression Models》2](http://arxiv.org/pdf/1402.4304v2.pdf)  
  介绍：不会统计怎么办？不知道如何选择合适的统计模型怎么办？那这篇文章你的好好读一读了麻省理工Joshua B. Tenenbaum和剑桥Zoubin Ghahramani合作，写了一篇关于automatic statistician的文章。可以自动选择回归模型类别，还能自动写报告...
* [《ICLR 2014论文集》2](http://openreview.net/venue/iclr2014)  
  介绍:对深度学习和representation learning最新进展有兴趣的同学可以了解一下
* [《Introduction to Information Retrieval》1](http://www-nlp.stanford.edu/IR-book/)  
  介绍：这是一本信息检索相关的书籍，是由斯坦福Manning与谷歌副总裁Raghavan等合著的Introduction to Information Retrieval一直是北美最受欢迎的信息检索教材之一。最近作者增加了该课程的幻灯片和作业。IR相关资源：[1http://www-nlp.stanford.edu/IR-book/information-retrieval.html1](http://www-nlp.stanford.edu/IR-book/information-retrieval.html)
* [《Machine learning in 10 pictures》17](http://www.denizyuret.com/2014/02/machine-learning-in-5-pictures.html)  
  介绍:Deniz Yuret用10张漂亮的图来解释机器学习重要概念：1. Bias/Variance Tradeoff 2. Overfitting 3. Bayesian / Occam's razor 4. Feature combination 5. Irrelevant feature 6. Basis function 7. Discriminative / Generative 8. Loss function 9. Least squares 10. Sparsity.很清晰
* [《雅虎研究院的数据集汇总》12](http://webscope.sandbox.yahoo.com/catalog.php?datatype=l)  
  介绍：雅虎研究院的数据集汇总： 包括语言类数据，图与社交类数据，评分与分类数据，计算广告学数据，图像数据，竞赛数据，以及系统类的数据。
* [《An Introduction to Statistical Learning with Applications in R》1](http://www-bcf.usc.edu/~gareth/ISL/)  
  介绍：这是一本斯坦福统计学著名教授Trevor Hastie和Robert Tibshirani的新书，并且在2014年一月已经开课：<https://class.stanford.edu/courses/HumanitiesScience/StatLearning/Winter2014/about>
* [Best Machine Learning Resources for Getting Started5](http://machinelearningmastery.com/best-machine-learning-resources-for-getting-started/)  
  介绍：机器学习最佳入门学习资料汇总是专为机器学习初学者推荐的优质学习资源，帮助初学者快速入门。而且这篇文章的介绍已经被翻译成[中文版](http://suanfazu.com/shuoit/20140625/340777.html)。如果你不怎么熟悉，那么我建议你先看一看中文的介绍。
* [My deep learning reading list6](http://blog.sina.com.cn/s/blog_bda0d2f10101fpp4.html)  
  介绍:主要是顺着Bengio的PAMI review的文章找出来的。包括几本综述文章，将近100篇论文，各位山头们的Presentation。全部都可以在google上找到。
* [Cross-Language Information Retrieval](http://www.morganclaypool.com/doi/abs/10.2200/S00266ED1V01Y201005HLT008?journalCode=hlt)  
  介绍：这是一本书籍，主要介绍的是跨语言信息检索方面的知识。理论很多
* [探索推荐引擎内部的秘密，第 1 部分: 推荐引擎初探7](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy1/index.html?ca=drs-)  
  介绍:本文共有三个系列，作者是来自IBM的工程师。它主要介绍了推荐引擎相关算法，并帮助读者高效的实现这些算法。[探索推荐引擎内部的秘密，第 2 部分: 深度推荐引擎相关算法5](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy2/index.html?ca=drs-)
* [协同过滤5](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy2/index.html?ca=drs-)

,[探索推荐引擎内部的秘密，第 3 部分: 深度推荐引擎相关算法](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy3/index.html?ca=drs-)

[聚类](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy3/index.html?ca=drs-)

[8](http://www.ibm.com/developerworks/cn/web/1103_zhaoct_recommstudy3/index.html?ca=drs-)

* [《Advice for students of machine learning》4](http://mimno.infosci.cornell.edu/b/articles/ml-learn/)  
  介绍：康奈尔大学信息科学系助理教授David Mimno写的《对机器学习初学者的一点建议》， 写的挺实际，强调实践与理论结合，最后还引用了冯 ? 诺依曼的名言: "Young man, in mathematics you don't understand things. You just get used to them."
* [分布式并行处理的数据1](http://web.stanford.edu/group/pdplab/pdphandbook/)  
  介绍：这是一本关于分布式并行处理的数据《Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises》,作者是斯坦福的James L. McClelland。着重介绍了各种神级网络算法的分布式实现,做Distributed Deep Learning 的童鞋可以参考下
* [《“机器学习”是什么？》](http://suanfazu.com/book/324935.html)  
  介绍:【“机器学习”是什么？】John Platt是微软研究院杰出科学家，17年来他一直在机器学习领域耕耘。近年来机器学习变得炙手可热，Platt和同事们遂决定开设[博客](http://suanfazu.com/book/324935.html)，向公众介绍机器学习的研究进展。机器学习是什么，被应用在哪里？来看Platt的这篇[博文](http://suanfazu.com/book/324935.html)
* [《2014年国际机器学习大会ICML 2014 论文》8](http://icml.cc/2014/index/article/15.htm)  
  介绍：2014年国际机器学习大会（ICML）已经于6月21-26日在国家会议中心隆重举办。本次大会由微软亚洲研究院和清华大学联手主办，是这个有着30多年历史并享誉世界的机器学习领域的盛会首次来到中国，已成功吸引海内外1200多位学者的报名参与。干货很多，值得深入学习下
* [《Machine Learning for Industry: A Case Study》](http://suanfazu.com/book/324935.html)  
  介绍：这篇文章主要是以Learning to Rank为例说明企业界机器学习的具体应用，RankNet对NDCG之类不敏感，加入NDCG因素后变成了LambdaRank，同样的思想从神经网络改为应用到Boosted Tree模型就成就了LambdaMART。[Chirs Burges](http://research.microsoft.com/en-us/people/cburges/?WT.mc_id=Blog_MachLearn_General_DI)，微软的机器学习大神，Yahoo 2010 Learning to Rank Challenge第一名得主，排序模型方面有RankNet，LambdaRank，LambdaMART，尤其以LambdaMART最为突出，代表论文为：[From RankNet to LambdaRank to LambdaMART: An Overview](http://research.microsoft.com/en-us/um/people/cburges/tech_reports/msr-tr-2010-82.pdf)此外，Burges还有很多有名的代表作，比如：[A Tutorial on Support Vector Machines for Pattern Recognition](http://research.microsoft.com/pubs/67119/svmtutorial.pdf)  
    
  [Some Notes on Applied Mathematics for Machine Learning1](http://research.microsoft.com/en-us/um/people/cburges/tech_reports/tr-2004-56.pdf)
* [100 Best GitHub: Deep Learning3](http://meta-guide.com/software-meta-guide/100-best-github-deep-learning/)  
  介绍:100 Best GitHub: Deep Learning
* [《UFLDL-斯坦福大学Andrew Ng教授“Deep Learning”教程》6](http://www.52ml.net/12019.html)  
  介绍:本教程将阐述无监督特征学习和深度学习的主要观点。通过学习，你也将实现多个功能学习/深度学习算法，能看到它们为你工作，并学习如何应用/适应这些想法到新问题上。本教程假定机器学习的基本知识（特别是熟悉的监督学习，逻辑回归，梯度下降的想法），如果你不熟悉这些想法，我们建议你去这里[机器学习课程1](http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning)，并先完成第II，III，IV章（到逻辑回归）。此外这关于这套教程的源代码在github上面已经有python版本了[UFLDL Tutorial Code](https://github.com/jatinshah/ufldl_tutorial)  
  \*[《Deep Learning for Natural Language Processing and Related Applications》3](http://research.microsoft.com/pubs/217165/ICASSP_DeepTextLearning_v07.pdf)  
  介绍:这份文档来自微软研究院,精髓很多。如果需要完全理解，需要一定的机器学习基础。不过有些地方会让人眼前一亮,毛塞顿开。
* [Understanding Convolutions6](https://colah.github.io/posts/2014-07-Understanding-Convolutions/)  
  介绍:这是一篇介绍图像卷积运算的文章，讲的已经算比较详细的了
* [《Machine Learning Summer School》4](http://mlss2014.com/)  
  介绍：每天请一个大牛来讲座，主要涉及机器学习，大数据分析，并行计算以及人脑研究。[1https://www.youtube.com/user/smolix1](https://www.youtube.com/user/smolix)（需翻墙）
* [《Awesome Machine Learning》3](https://github.com/josephmisiti/awesome-machine-learning)  
  介绍：一个超级完整的机器学习开源库总结，如果你认为这个碉堡了，那后面这个列表会更让你惊讶：【Awesome Awesomeness】,国内已经有热心的朋友进行了翻译[中文介绍](http://suanfazu.com/shuoit/20140723/341355.html)
* [斯坦福《自然语言处理》课程视频2](http://see.stanford.edu/see/lecturelist.aspx?coll=63480b48-8819-4efd-8412-263f1a472f5a)  
  介绍:ACL候任主席、斯坦福大学计算机系Chris Manning教授的《自然语言处理》课程所有视频已经可以在斯坦福公开课网站上观看了（如Chrome不行，可用IE观看） 作业与测验也可以下载。
* [《Deep Learning and Shallow Learning》3](http://freemind.pluskid.org/machine-learning/deep-learning-and-shallow-learning/)  
  介绍:对比 Deep Learning 和 Shallow Learning 的好文，来着浙大毕业、MIT 读博的 Chiyuan Zhang 的博客。
* [《Recommending music on Spotify with deep learning》2](http://benanne.github.io/2014/08/05/spotify-cnns.html)  
  介绍:利用卷积神经网络做音乐推荐。
* [《Neural Networks and Deep Learning》6](http://neuralnetworksanddeeplearning.com/index.html)  
  介绍：神经网络的免费在线书，已经写了三章了，还有对应的开源代码：[2https://github.com/mnielsen/neural-networks-and-deep-learning爱好者的福音。2](https://github.com/mnielsen/neural-networks-and-deep-learningç±å¥½èçç¦é³ã)
* [《Java Machine Learning》4](http://machinelearningmastery.com/java-machine-learning/)  
  介绍：Java机器学习相关平台和开源的机器学习库，按照大数据、NLP、计算机视觉和Deep Learning分类进行了整理。看起来挺全的，Java爱好者值得收藏。
* [《Machine Learning Theory: An Introductory Primer》5](http://www.oschina.net/translate/6-tips-for-writing-better-code)  
  介绍：机器学习最基本的入门文章，适合零基础者
* [《机器学习常见算法分类汇总》](http://suanfazu.com/shuoit/20140925/342326.html)  
  介绍：机器学习的算法很多。很多时候困惑人们都是，很多算法是一类算法，而有些算法又是从其他算法中延伸出来的。这里，我们从两个方面来给大家介绍，第一个方面是学习的方式，第二个方面是算法的类似性。
* [《机器学习视频库》3](http://work.caltech.edu/library/)  
  介绍：视频由加州理工学院（Caltech）出品。需要英语底子。
* [《机器学习经典书籍》](http://suanfazu.com/shuoit/20140708/341114.html)  
  介绍：总结了机器学习的经典书籍，包括数学基础和算法理论的书籍，可做为入门参考书单。
* [《16 Free eBooks On Machine Learning》7](http://efytimes.com/e1/fullnews.asp?edid=121516)  
  介绍:16本机器学习的电子书，可以下载下来在pad，手机上面任意时刻去阅读。不多我建议你看完一本再下载一本。
* [《A Large set of Machine Learning Resources for Beginners to Mavens》1](http://www.erogol.com/large-set-machine-learning-resources-beginners-mavens/)  
  介绍:标题很大，从新手到专家。不过看完上面所有资料。肯定是专家了
* [《机器学习最佳入门学习资料汇总》](http://suanfazu.com/shuoit/20140625/340777.html)  
  介绍：入门的书真的很多，而且我已经帮你找齐了。
* [《Sibyl》2](http://users.soe.ucsc.edu/~niejiazhong/slides/chandra.pdf)  
  介绍：Sibyl 是一个监督式机器学习系统，用来解决预测方面的问题，比如 YouTube 的视频推荐。
* [《Deep Learning》1](http://www.iro.umontreal.ca/~bengioy/dlbook/)  
  介绍：Yoshua Bengio, Ian Goodfellow, Aaron Courville著
* [《Neural Network & Text Mining》1](http://www.slideshare.net/ssuser9cc1bd/piji-li-dltm)  
  介绍:关于(Deep) Neural Networks在 NLP 和 Text Mining 方面一些paper的总结
* [《前景目标检测1（总结）》3](http://www.cnblogs.com/lxy2017/p/3927226.html)  
  介绍:计算机视觉入门之前景目标检测1（总结）
* [《行人检测》3](http://www.52ml.net/17004.html)  
  介绍:计算机视觉入门之行人检测
* [《Deep Learning – important resources for learning and understanding》](http://www.kdnuggets.com/2014/08/deep-learning-important-resources-learning-understanding.html)  
  介绍:Important resources for learning and understanding . Is awesome
* [《Machine Learning Theory: An Introductory Primer》1](http://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer)  
  介绍:这又是一篇机器学习初学者的入门文章。值得一读
* [《Neural Networks and Deep Learning》2](http://neuralnetworksanddeeplearning.com/)  
  介绍:在线Neural Networks and Deep Learning电子书
* [《Python 网页爬虫 & 文本处理 & 科学计算 & 机器学习 & 数据挖掘兵器谱》6](http://www.52nlp.cn/python-ç½é¡µç¬è«-ææ¬å¤ç-ç§å­¦è®¡ç®-æºå¨å­¦ä¹ -æ°æ®ææ)  
  介绍:python的17个关于机器学习的工具
* [《神奇的伽玛函数(上)》2](http://www.flickering.cn/æ¦çç»è®¡/2014/06/ç¥å¥çä¼½çå½æ°ä¸/)  
  介绍:下集在这里[神奇的伽玛函数(下)2](http://www.flickering.cn/æ¦çç»è®¡/2014/06/ç¥å¥çä¼½çå½æ°ä¸/)
* [《分布式机器学习的故事》](http://suanfazu.com/shuoit/20140123/337763.html)  
  介绍:作者王益目前是腾讯广告算法总监，王益博士毕业后在google任研究。这篇文章王益博士7年来从谷歌到腾讯对于分布机器学习的所见所闻。值得细读
* [《机器学习提升之道（Level-Up Your Machine Learning）》4](http://metacademy.org/roadmaps/cjrd/level-up-your-ml)  
  介绍:把机器学习提升的级别分为0~4级，每级需要学习的教材和掌握的知识。这样，给机器学习者提供一个上进的路线图，以免走弯路。另外，整个网站都是关于机器学习的，资源很丰富。
* [《Machine Learning Surveys》2](http://www.mlsurveys.com/)  
  介绍:机器学习各个方向综述的网站
* [《Deep Learning Reading list》](http://deeplearning.net/reading-list/)  
  介绍:深度学习阅资源列表
* [《Deep Learning: Methods and Applications》1](http://research.microsoft.com/pubs/219984/DeepLearningBook_RefsByLastFirstNames.pdf)  
  介绍：这是一本来自微的研究员 li Peng和Dong Yu所著的关于深度学习的方法和应用的电子书
* [《Machine Learning Summer School 2014》7](http://pan.baidu.com/s/1pJ0ok7T)  
  介绍:2014年七月CMU举办的机器学习夏季课刚刚结束 有近50小时的视频、十多个PDF版幻灯片，覆盖 深度学习，贝叶斯，分布式机器学习，伸缩性 等热点话题。所有13名讲师都是牛人：包括大牛Tom Mitchell （他的［机器学习］是名校的常用教材），还有CMU李沐 .（1080P高清哟）
* [《Sibyl: 来自Google的大规模机器学习系统》2](http://users.soe.ucsc.edu/~niejiazhong/slides/chandra.pdf)  
  介绍:在今年的IEEE/IFIP可靠系统和网络（DSN）国际会议上，Google软件工程师Tushar Chandra做了一个关于Sibyl系统的主题演讲。 Sibyl是一个监督式机器学习系统，用来解决预测方面的问题，比如YouTube的视频推荐。详情请阅读[google sibyl](http://suanfazu.com/shuoit/20140715/341222.html)
* [《Building a deeper understanding of images》2](http://googleresearch.blogspot.com/2014/09/building-deeper-understanding-of-images.html)  
  介绍:谷歌研究院的Christian Szegedy在谷歌研究院的博客上简要地介绍了他们今年参加ImageNet取得好成绩的GoogLeNet系统.是关于图像处理的。
* [《Bayesian network 与python概率编程实战入门》1](https://github.com/memect/hao/blob/master/awesome/bayesian-network-python.md)  
  介绍:贝叶斯学习。如果不是很清可看看[概率编程语言与贝叶斯方法实践2](http://www.infoq.com/cn/news/2014/07/programming-language-bayes)
* [《AMA: Michael I Jordan》2](http://www.reddit.com/r/MachineLearning/comments/2fxi6v/ama_michael_i_jordan/)  
  介绍:网友问伯克利机器学习大牛、美国双料院士Michael I. Jordan："如果你有10亿美金，你怎么花？Jordan: "我会用这10亿美金建造一个NASA级别的自然语言处理研究项目。"
* [《机器学习&数据挖掘笔记\_16（常见面试之机器学习算法思想简单梳理）》5](http://www.cnblogs.com/tornadomeet/p/3395593.html)  
  介绍:常见面试之机器学习算法思想简单梳理
* [《文本与数据挖掘视频汇总》3](http://www.kdnuggets.com/2014/09/most-viewed-web-mining-lectures-videolectures.html)  
  介绍：Videolectures上最受欢迎的25个文本与数据挖掘视频汇总
* [《怎么选择深度学习的GPUs》2](http://timdettmers.wordpress.com/2014/08/14/which-gpu-for-deep-learning/)  
  介绍:在Kaggle上经常取得不错成绩的Tim Dettmers介绍了他自己是怎么选择深度学习的GPUs, 以及个人如何构建深度学习的GPU集群:[2http://t.cn/RhpuD1G2](http://t.cn/RhpuD1G)
* [《对话机器学习大神Michael Jordan：深度模型》4](http://www.infoq.com/cn/news/2014/09/depth-model)  
  介绍:对话机器学习大神Michael Jordan
* [《Deep Learning 和 Knowledge Graph 引爆大数据革命》4](http://blog.sina.com.cn/s/blog_46d0a3930101fswl.html)  
  介绍:还有２，３部分。[1http://blog.sina.com.cn/s/blog\_46d0a3930101gs5h.html1](http://blog.sina.com.cn/s/blog_46d0a3930101gs5h.html)
* [《Deep Learning 教程翻译》23](http://blog.sina.com.cn/s/blog_46d0a3930101h6nf.html)  
  介绍:是Stanford 教授 Andrew Ng 的 Deep Learning 教程，国内的机器学习爱好者很热心的把这个教程翻译成了中文。如果你英语不好，可以看看这个

# [深度学习阅读清单](http://suanfazu.com/t/shen-du-xue-xi-yue-du-qing-dan/245)

Deep learning Reading List

Following is a growing list of some of the materials i found on the web for Deep Learning beginners.

### Free Online Books

1. [Deep Learning](http://www.iro.umontreal.ca/~bengioy/dlbook/) by Yoshua Bengio, Ian Goodfellow and Aaron Courville
2. [Neural Networks and Deep Learning](http://neuralnetworksanddeeplearning.com/) by Michael Nielsen
3. [Deep Learning](http://research.microsoft.com/pubs/209355/DeepLearning-NowPublishing-Vol7-SIG-039.pdf) by Microsoft Research
4. [Deep Learning Tutorial](http://deeplearning.net/tutorial/deeplearning.pdf) by LISA lab, University of Montreal

### Courses

1. [Machine Learning](https://class.coursera.org/ml-005) by Andrew Ng in Coursera
2. [Neural Networks for Machine Learning](https://class.coursera.org/neuralnets-2012-001) by Geoffrey Hinton in Coursera
3. [Neural networks class](https://www.youtube.com/playlist?list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH) by Hugo Larochelle from Université de Sherbrooke
4. [Deep Learning Course](http://cilvr.cs.nyu.edu/doku.php?id=deeplearning:slides:start) by CILVR lab @ NYU
5. [CS231n: Convolutional Neural Networks for Visual Recognition](http://vision.stanford.edu/teaching/cs231n/syllabus.html) On-Going
6. [CS224d: Deep Learning for Natural Language Processing](http://cs224d.stanford.edu/) Going to start

### Video and Lectures

1. [How To Create A Mind](https://www.youtube.com/watch?v=RIkxVci-R4k) By Ray Kurzweil - Is a inspiring talk
2. [Deep Learning, Self-Taught Learning and Unsupervised Feature Learning](https://www.youtube.com/watch?v=n1ViNeWhC24) By Andrew Ng
3. [Recent Developments in Deep Learning](https://www.youtube.com/watch?v=vShMxxqtDDs&index=3&list=PL78U8qQHXgrhP9aZraxTT5-X1RccTcUYT) By Geoff Hinton
4. [The Unreasonable Effectiveness of Deep Learning](https://www.youtube.com/watch?v=sc-KbuZqGkI) by Yann LeCun
5. [Deep Learning of Representations](https://www.youtube.com/watch?v=4xsVFLnHC_0) by Yoshua bengio
6. [Principles of Hierarchical Temporal Memory](https://www.youtube.com/watch?v=6ufPpZDmPKA) by Jeff Hawkins
7. [Machine Learning Discussion Group - Deep Learning w/ Stanford AI Lab](https://www.youtube.com/watch?v=2QJi0ArLq7s&list=PL78U8qQHXgrhP9aZraxTT5-X1RccTcUYT) by Adam Coates
8. [Making Sense of the World with Deep Learning](http://vimeo.com/80821560) By Adam Coates
9. [Demystifying Unsupervised Feature Learning](https://www.youtube.com/watch?v=wZfVBwOO0-k)By Adam Coates
10. [Visual Perception with Deep Learning](https://www.youtube.com/watch?v=3boKlkPBckA) By Yann LeCun

### Papers

1. [ImageNet Classification with Deep Convolutional Neural Networks](http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)
2. [Using Very Deep Autoencoders for Content Based Image Retrieval](http://www.cs.toronto.edu/~hinton/absps/esann-deep-final.pdf)
3. [Learning Deep Architectures for AI](http://www.iro.umontreal.ca/~lisa/pointeurs/TR1312.pdf)
4. [CMU’s list of papers](http://deeplearning.cs.cmu.edu/)

### Tutorials

1. [UFLDL Tutorial 1](http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial)
2. [UFLDL Tutorial 2](http://ufldl.stanford.edu/tutorial/supervised/LinearRegression/)
3. [Deep Learning for NLP (without Magic)](http://www.socher.org/index.php/DeepLearningTutorial/DeepLearningTutorial)
4. [A Deep Learning Tutorial: From Perceptrons to Deep Networks](http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks)

### WebSites

1. [deeplearning.net](http://deeplearning.net/)
2. [deeplearning.stanford.edu](http://deeplearning.stanford.edu/)

### Datasets

1. [MNIST](http://yann.lecun.com/exdb/mnist/) Handwritten digits
2. [Google House Numbers](http://ufldl.stanford.edu/housenumbers/) from street view
3. [CIFAR-10 and CIFAR-100](http://www.cs.toronto.edu/~kriz/cifar.html)
4. [IMAGENET](http://www.image-net.org/)
5. [Tiny Images](http://groups.csail.mit.edu/vision/TinyImages/) 80 Million tiny images
6. [Flickr Data](http://yahoolabs.tumblr.com/post/89783581601/one-hundred-million-creative-commons-flickr-images) 100 Million Yahoo dataset
7. [Berkeley Segmentation Dataset 500](http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/)

### Frameworks

1. [Caffe](http://caffe.berkeleyvision.org/)
2. [Torch7](http://torch.ch/)
3. [Theano](http://deeplearning.net/software/theano/)
4. [cuda-convnet](https://code.google.com/p/cuda-convnet2/)
5. [Ccv](http://libccv.org/doc/doc-convnet/)
6. [NuPIC](http://numenta.org/nupic.html)
7. [DeepLearning4J](http://deeplearning4j.org/)

### Miscellaneous

1. [Google Plus - Deep Learning Community](https://plus.google.com/communities/112866381580457264725)
2. [Caffe Webinar](http://on-demand-gtc.gputechconf.com/gtcnew/on-demand-gtc.php?searchByKeyword=shelhamer&searchItems=&sessionTopic=&sessionEvent=4&sessionYear=2014&sessionFormat=&submit=&select=+)
3. [100 Best Github Resources in Github for DL](http://meta-guide.com/software-meta-guide/100-best-github-deep-learning/)
4. [Word2Vec](https://code.google.com/p/word2vec/)
5. [Caffe DockerFile](https://registry.hub.docker.com/u/tleyden5iwx/caffe/)
6. [TorontoDeepLEarning convnet](https://github.com/TorontoDeepLearning/convnet)
7. [Vision data sets](http://www.cs.cmu.edu/~cil/v-images.html)
8. [Fantastic Torch Tutorial](http://code.cogbits.com/wiki/doku.php) My personal favourite. Also check out [gfx.js](https://github.com/clementfarabet/gfx.js)
9. [Torch7 Cheat sheet](https://github.com/torch/torch7/wiki/Cheatsheet)

# Python Scrapy笔记

## 安装

1 首先安装python注意版本，我安装的是Windows下的2.7这一版，傻瓜式安装在python官网下载2.7安装包就行了。

官方主页：<http://www.python.org/>

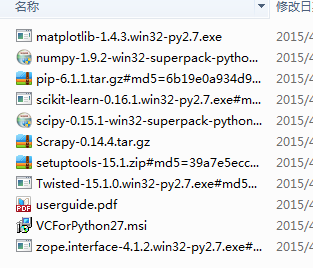
下载地址：<http://www.python.org/ftp/python/2.7.3/python-2.7.3.msi>

2 修改环境变量，在我的电脑右键属性，环境变量，path然后下面是我的修改。

D:\Program Files (x86)\Microsoft Visual Studio\Common\Tools\WinNT;D:\Program Files (x86)\Microsoft Visual Studio\Common\MSDev98\Bin;D:\Program Files (x86)\Microsoft Visual Studio\Common\Tools;D:\Program Files (x86)\Microsoft Visual Studio\VC98\bin

验证：在cmd下输入python有如下结果显示安装完成

3 安装pip和easy\_install包管理安装如下：



一、 Scrapy简

Scrapy is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages. It can be used for a wide range of purposes, from data mining to monitoring and automated testing.

官方主页： <http://www.scrapy.org/>

三、 安装Twisted

Twisted is an event-driven networking engine written in Python and licensed under the open source

**1)**安装setuptools

Download, build, install, upgrade, and uninstall Python packages -- easily!

官方主页：<http://pypi.python.org/pypi/setuptools>

下载地址：<http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe>

**2)**安装Zope.Interface

官方主页：<http://pypi.python.org/pypi/zope.interface/>

下载地址：<http://pypi.python.org/packages/2.7/z/zope.interface/zope.interface-4.0.1-py2.7-win32.egg>

安装过程：



T:\>d:

D:\>cd D:\Python27\Scripts

D:\Python27\Scripts>easy\_install.exe zope.interface-4.0.1-py2.7-win32.egg

Processing zope.interface-4.0.1-py2.7-win32.egg

creating d:\python27\lib\site-packages\zope.interface-4.0.1-py2.7-win32.egg

Extracting zope.interface-4.0.1-py2.7-win32.egg to d:\python27\lib\site-packages

Adding zope.interface 4.0.1 to easy-install.pth file

Installed d:\python27\lib\site-packages\zope.interface-4.0.1-py2.7-win32.egg

Processing dependencies for zope.interface==4.0.1

Finished processing dependencies for zope.interface==4.0.1

D:\Python27\Scripts>



验证安装：



D:\Python27\Scripts>python

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import zope.interface

>>>



**3)**安装Twisted

官方主页：<http://twistedmatrix.com/trac/wiki/TwistedProject>

下载地址：<http://pypi.python.org/packages/2.7/T/Twisted/Twisted-12.1.0.win32-py2.7.msi>

四、 安装w3lib

官方主页：<http://pypi.python.org/pypi/w3lib>

下载地址： <http://pypi.python.org/packages/source/w/w3lib/w3lib-1.2.tar.gz>

安装过程：



T:\w3lib-1.2>python setup.py install

running install

running build

running build\_py

creating build

creating build\lib

creating build\lib\w3lib

copying w3lib\encoding.py -> build\lib\w3lib

copying w3lib\form.py -> build\lib\w3lib

copying w3lib\html.py -> build\lib\w3lib

copying w3lib\http.py -> build\lib\w3lib

copying w3lib\url.py -> build\lib\w3lib

copying w3lib\util.py -> build\lib\w3lib

copying w3lib\\_\_init\_\_.py -> build\lib\w3lib

running install\_lib

creating D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\encoding.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\form.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\html.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\http.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\url.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\util.py -> D:\Python27\Lib\site-packages\w3lib

copying build\lib\w3lib\\_\_init\_\_.py -> D:\Python27\Lib\site-packages\w3lib

byte-compiling D:\Python27\Lib\site-packages\w3lib\encoding.py to encoding.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\form.py to form.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\html.py to html.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\http.py to http.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\url.py to url.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\util.py to util.pyc

byte-compiling D:\Python27\Lib\site-packages\w3lib\\_\_init\_\_.py to \_\_init\_\_.pyc

running install\_egg\_info

Writing D:\Python27\Lib\site-packages\w3lib-1.2-py2.7.egg-info

T:\w3lib-1.2>



验证安装：



T:\>python

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import w3lib

>>>



五、 安装libxml2

官方主页：[http://users.skynet.be/sbi/libxml-python/http://pypi.python.org/pypi/pyOpenSSL](http://users.skynet.be/sbi/libxml-python/http:/pypi.python.org/pypi/pyOpenSSL)

下载地址：<http://users.skynet.be/sbi/libxml-python/binaries/libxml2-python-2.7.7.win32-py2.7.exe>

验证安装：



T:\>python

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import libxml2

>>>



六、 安装pyOpenSSL

官方主页：<http://pypi.python.org/pypi/pyOpenSSL>

下载地址：<http://pypi.python.org/packages/2.7/p/pyOpenSSL/pyOpenSSL-0.13.winxp32-py2.7.msi>

验证安装：



T:\>python

Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> import OpenSSL

>>>



七、 安装Scrapy

官方主页：<http://scrapy.org/>

下载地址：<http://pypi.python.org/packages/source/S/Scrapy/Scrapy-0.14.4.tar.gz>

安装过程：



T:\Scrapy-0.14.4>python setup.py install

……

Installing easy\_install-2.7-script.py script to D:\Python27\Scripts

Installing easy\_install-2.7.exe script to D:\Python27\Scripts

Installing easy\_install-2.7.exe.manifest script to D:\Python27\Scripts

Using d:\python27\lib\site-packages

Finished processing dependencies for Scrapy==0.14.4

T:\Scrapy-0.14.4>



验证安装：



T:\>scrapy

Scrapy 0.14.4 - no active project

Usage:

scrapy <command> [options] [args]

Available commands:

fetch Fetch a URL using the Scrapy downloader

runspider Run a self-contained spider (without creating a project)

settings Get settings values

shell Interactive scraping console

startproject Create new project

version Print Scrapy version

view Open URL in browser, as seen by Scrapy

Use "scrapy <command> -h" to see more info about a command

T:\>



# 爬虫

## [爬虫详解（入门教程）——精华](http://blog.csdn.net/column/details/why-bug.html)

### Help-urllib2

**Help on module urllib2:**

**NAME**

**urllib2 - An extensible library for opening URLs using a variety of protocols**

**FILE**

**d:\python27\lib\urllib2.py**

**DESCRIPTION**

**The simplest way to use this module is to call the urlopen function,**

**which accepts a string containing a URL or a Request object (described**

**below). It opens the URL and returns the results as file-like**

**object; the returned object has some extra methods described below.**

**The OpenerDirector manages a collection of Handler objects that do**

**all the actual work. Each Handler implements a particular protocol or**

**option. The OpenerDirector is a composite object that invokes the**

**Handlers needed to open the requested URL. For example, the**

**HTTPHandler performs HTTP GET and POST requests and deals with**

**non-error returns. The HTTPRedirectHandler automatically deals with**

**HTTP 301, 302, 303 and 307 redirect errors, and the HTTPDigestAuthHandler**

**deals with digest authentication.**

**urlopen(url, data=None) -- Basic usage is the same as original**

**urllib. pass the url and optionally data to post to an HTTP URL, and**

**get a file-like object back. One difference is that you can also pass**

**a Request instance instead of URL. Raises a URLError (subclass of**

**IOError); for HTTP errors, raises an HTTPError, which can also be**

**treated as a valid response.**

**build\_opener -- Function that creates a new OpenerDirector instance.**

**Will install the default handlers. Accepts one or more Handlers as**

**arguments, either instances or Handler classes that it will**

**instantiate. If one of the argument is a subclass of the default**

**handler, the argument will be installed instead of the default.**

**install\_opener -- Installs a new opener as the default opener.**

**objects of interest:**

**OpenerDirector -- Sets up the User Agent as the Python-urllib client and manages**

**the Handler classes, while dealing with requests and responses.**

**Request -- An object that encapsulates the state of a request. The**

**state can be as simple as the URL. It can also include extra HTTP**

**headers, e.g. a User-Agent.**

**BaseHandler --**

**exceptions:**

**URLError -- A subclass of IOError, individual protocols have their own**

**specific subclass.**

**HTTPError -- Also a valid HTTP response, so you can treat an HTTP error**

**as an exceptional event or valid response.**

**internals:**

**BaseHandler and parent**

**\_call\_chain conventions**

**Example usage:**

**import urllib2**

**# set up authentication info**

**authinfo = urllib2.HTTPBasicAuthHandler()**

**authinfo.add\_password(realm='PDQ Application',**

**uri='https://mahler:8092/site-updates.py',**

**user='klem',**

**passwd='geheim$parole')**

**proxy\_support = urllib2.ProxyHandler({"http" : "http://ahad-haam:3128"})**

**# build a new opener that adds authentication and caching FTP handlers**

**opener = urllib2.build\_opener(proxy\_support, authinfo, urllib2.CacheFTPHandler)**

**# install it**

**urllib2.install\_opener(opener)**

**f = urllib2.urlopen('http://www.python.org/')**

**CLASSES**

**exceptions.IOError(exceptions.EnvironmentError)**

**URLError**

**HTTPError(URLError, urllib.addinfourl)**

**AbstractBasicAuthHandler**

**HTTPBasicAuthHandler(AbstractBasicAuthHandler, BaseHandler)**

**ProxyBasicAuthHandler(AbstractBasicAuthHandler, BaseHandler)**

**AbstractDigestAuthHandler**

**BaseHandler**

**AbstractHTTPHandler**

**HTTPHandler**

**HTTPSHandler**

**FTPHandler**

**CacheFTPHandler**

**FileHandler**

**HTTPCookieProcessor**

**HTTPDefaultErrorHandler**

**HTTPDigestAuthHandler(BaseHandler, AbstractDigestAuthHandler)**

**HTTPErrorProcessor**

**HTTPRedirectHandler**

**ProxyDigestAuthHandler(BaseHandler, AbstractDigestAuthHandler)**

**ProxyHandler**

**UnknownHandler**

**HTTPPasswordMgr**

**HTTPPasswordMgrWithDefaultRealm**

**OpenerDirector**

**Request**

**class AbstractBasicAuthHandler**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, password\_mgr=None)**

**|**

**| http\_error\_auth\_reqed(self, authreq, host, req, headers)**

**|**

**| retry\_http\_basic\_auth(self, host, req, realm)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| rx = <\_sre.SRE\_Pattern object>**

**class AbstractDigestAuthHandler**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, passwd=None)**

**|**

**| get\_algorithm\_impls(self, algorithm)**

**|**

**| get\_authorization(self, req, chal)**

**|**

**| get\_cnonce(self, nonce)**

**|**

**| get\_entity\_digest(self, data, chal)**

**|**

**| http\_error\_auth\_reqed(self, auth\_header, host, req, headers)**

**|**

**| reset\_retry\_count(self)**

**|**

**| retry\_http\_digest\_auth(self, req, auth)**

**class AbstractHTTPHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, debuglevel=0)**

**|**

**| do\_open(self, http\_class, req, \*\*http\_conn\_args)**

**| Return an addinfourl object for the request, using http\_class.**

**|**

**| http\_class must implement the HTTPConnection API from httplib.**

**| The addinfourl return value is a file-like object. It also**

**| has methods and attributes including:**

**| - info(): return a mimetools.Message object for the headers**

**| - geturl(): return the original request URL**

**| - code: HTTP status code**

**|**

**| do\_request\_(self, request)**

**|**

**| set\_http\_debuglevel(self, level)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class BaseHandler**

**| Methods defined here:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| handler\_order = 500**

**class CacheFTPHandler(FTPHandler)**

**| Method resolution order:**

**| CacheFTPHandler**

**| FTPHandler**

**| BaseHandler**

**|**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self)**

**| # XXX would be nice to have pluggable cache strategies**

**| # XXX this stuff is definitely not thread safe**

**|**

**| check\_cache(self)**

**|**

**| clear\_cache(self)**

**|**

**| connect\_ftp(self, user, passwd, host, port, dirs, timeout)**

**|**

**| setMaxConns(self, m)**

**|**

**| setTimeout(self, t)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from FTPHandler:**

**|**

**| ftp\_open(self, req)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class FTPHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| connect\_ftp(self, user, passwd, host, port, dirs, timeout)**

**|**

**| ftp\_open(self, req)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class FileHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| file\_open(self, req)**

**| # Use local file or FTP depending on form of URL**

**|**

**| get\_names(self)**

**|**

**| open\_local\_file(self, req)**

**| # not entirely sure what the rules are here**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| names = None**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPBasicAuthHandler(AbstractBasicAuthHandler, BaseHandler)**

**| Method resolution order:**

**| HTTPBasicAuthHandler**

**| AbstractBasicAuthHandler**

**| BaseHandler**

**|**

**| Methods defined here:**

**|**

**| http\_error\_401(self, req, fp, code, msg, headers)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| auth\_header = 'Authorization'**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractBasicAuthHandler:**

**|**

**| \_\_init\_\_(self, password\_mgr=None)**

**|**

**| http\_error\_auth\_reqed(self, authreq, host, req, headers)**

**|**

**| retry\_http\_basic\_auth(self, host, req, realm)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from AbstractBasicAuthHandler:**

**|**

**| rx = <\_sre.SRE\_Pattern object>**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPCookieProcessor(BaseHandler)**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, cookiejar=None)**

**|**

**| http\_request(self, request)**

**|**

**| http\_response(self, request, response)**

**|**

**| https\_request = http\_request(self, request)**

**|**

**| https\_response = http\_response(self, request, response)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPDefaultErrorHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| http\_error\_default(self, req, fp, code, msg, hdrs)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPDigestAuthHandler(BaseHandler, AbstractDigestAuthHandler)**

**| An authentication protocol defined by RFC 2069**

**|**

**| Digest authentication improves on basic authentication because it**

**| does not transmit passwords in the clear.**

**|**

**| Method resolution order:**

**| HTTPDigestAuthHandler**

**| BaseHandler**

**| AbstractDigestAuthHandler**

**|**

**| Methods defined here:**

**|**

**| http\_error\_401(self, req, fp, code, msg, headers)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| auth\_header = 'Authorization'**

**|**

**| handler\_order = 490**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractDigestAuthHandler:**

**|**

**| \_\_init\_\_(self, passwd=None)**

**|**

**| get\_algorithm\_impls(self, algorithm)**

**|**

**| get\_authorization(self, req, chal)**

**|**

**| get\_cnonce(self, nonce)**

**|**

**| get\_entity\_digest(self, data, chal)**

**|**

**| http\_error\_auth\_reqed(self, auth\_header, host, req, headers)**

**|**

**| reset\_retry\_count(self)**

**|**

**| retry\_http\_digest\_auth(self, req, auth)**

**class HTTPError(URLError, urllib.addinfourl)**

**| Raised when HTTP error occurs, but also acts like non-error return**

**|**

**| Method resolution order:**

**| HTTPError**

**| URLError**

**| exceptions.IOError**

**| exceptions.EnvironmentError**

**| exceptions.StandardError**

**| exceptions.Exception**

**| exceptions.BaseException**

**| \_\_builtin\_\_.object**

**| urllib.addinfourl**

**| urllib.addbase**

**|**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, url, code, msg, hdrs, fp)**

**|**

**| \_\_str\_\_(self)**

**|**

**| info(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors defined here:**

**|**

**| reason**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors inherited from URLError:**

**|**

**| \_\_weakref\_\_**

**| list of weak references to the object (if defined)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from exceptions.IOError:**

**|**

**| \_\_new\_\_ = <built-in method \_\_new\_\_ of type object>**

**| T.\_\_new\_\_(S, ...) -> a new object with type S, a subtype of T**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from exceptions.EnvironmentError:**

**|**

**| \_\_reduce\_\_(...)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors inherited from exceptions.EnvironmentError:**

**|**

**| errno**

**| exception errno**

**|**

**| filename**

**| exception filename**

**|**

**| strerror**

**| exception strerror**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from exceptions.BaseException:**

**|**

**| \_\_delattr\_\_(...)**

**| x.\_\_delattr\_\_('name') <==> del x.name**

**|**

**| \_\_getattribute\_\_(...)**

**| x.\_\_getattribute\_\_('name') <==> x.name**

**|**

**| \_\_getitem\_\_(...)**

**| x.\_\_getitem\_\_(y) <==> x[y]**

**|**

**| \_\_getslice\_\_(...)**

**| x.\_\_getslice\_\_(i, j) <==> x[i:j]**

**|**

**| Use of negative indices is not supported.**

**|**

**| \_\_repr\_\_(...)**

**| x.\_\_repr\_\_() <==> repr(x)**

**|**

**| \_\_setattr\_\_(...)**

**| x.\_\_setattr\_\_('name', value) <==> x.name = value**

**|**

**| \_\_setstate\_\_(...)**

**|**

**| \_\_unicode\_\_(...)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors inherited from exceptions.BaseException:**

**|**

**| \_\_dict\_\_**

**|**

**| args**

**|**

**| message**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from urllib.addinfourl:**

**|**

**| getcode(self)**

**|**

**| geturl(self)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from urllib.addbase:**

**|**

**| close(self)**

**class HTTPErrorProcessor(BaseHandler)**

**| Process HTTP error responses.**

**|**

**| Methods defined here:**

**|**

**| http\_response(self, request, response)**

**|**

**| https\_response = http\_response(self, request, response)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| handler\_order = 1000**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**class HTTPHandler(AbstractHTTPHandler)**

**| Method resolution order:**

**| HTTPHandler**

**| AbstractHTTPHandler**

**| BaseHandler**

**|**

**| Methods defined here:**

**|**

**| http\_open(self, req)**

**|**

**| http\_request = do\_request\_(self, request)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractHTTPHandler:**

**|**

**| \_\_init\_\_(self, debuglevel=0)**

**|**

**| do\_open(self, http\_class, req, \*\*http\_conn\_args)**

**| Return an addinfourl object for the request, using http\_class.**

**|**

**| http\_class must implement the HTTPConnection API from httplib.**

**| The addinfourl return value is a file-like object. It also**

**| has methods and attributes including:**

**| - info(): return a mimetools.Message object for the headers**

**| - geturl(): return the original request URL**

**| - code: HTTP status code**

**|**

**| do\_request\_(self, request)**

**|**

**| set\_http\_debuglevel(self, level)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPPasswordMgr**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self)**

**|**

**| add\_password(self, realm, uri, user, passwd)**

**|**

**| find\_user\_password(self, realm, authuri)**

**|**

**| is\_suburi(self, base, test)**

**| Check if test is below base in a URI tree**

**|**

**| Both args must be URIs in reduced form.**

**|**

**| reduce\_uri(self, uri, default\_port=True)**

**| Accept authority or URI and extract only the authority and path.**

**class HTTPPasswordMgrWithDefaultRealm(HTTPPasswordMgr)**

**| Methods defined here:**

**|**

**| find\_user\_password(self, realm, authuri)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from HTTPPasswordMgr:**

**|**

**| \_\_init\_\_(self)**

**|**

**| add\_password(self, realm, uri, user, passwd)**

**|**

**| is\_suburi(self, base, test)**

**| Check if test is below base in a URI tree**

**|**

**| Both args must be URIs in reduced form.**

**|**

**| reduce\_uri(self, uri, default\_port=True)**

**| Accept authority or URI and extract only the authority and path.**

**class HTTPRedirectHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| http\_error\_301 = http\_error\_302(self, req, fp, code, msg, headers)**

**|**

**| http\_error\_302(self, req, fp, code, msg, headers)**

**| # Implementation note: To avoid the server sending us into an**

**| # infinite loop, the request object needs to track what URLs we**

**| # have already seen. Do this by adding a handler-specific**

**| # attribute to the Request object.**

**|**

**| http\_error\_303 = http\_error\_302(self, req, fp, code, msg, headers)**

**|**

**| http\_error\_307 = http\_error\_302(self, req, fp, code, msg, headers)**

**|**

**| redirect\_request(self, req, fp, code, msg, headers, newurl)**

**| Return a Request or None in response to a redirect.**

**|**

**| This is called by the http\_error\_30x methods when a**

**| redirection response is received. If a redirection should**

**| take place, return a new Request to allow http\_error\_30x to**

**| perform the redirect. Otherwise, raise HTTPError if no-one**

**| else should try to handle this url. Return None if you can't**

**| but another Handler might.**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| inf\_msg = 'The HTTP server returned a redirect error that w...n infini...**

**|**

**| max\_redirections = 10**

**|**

**| max\_repeats = 4**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class HTTPSHandler(AbstractHTTPHandler)**

**| Method resolution order:**

**| HTTPSHandler**

**| AbstractHTTPHandler**

**| BaseHandler**

**|**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, debuglevel=0, context=None)**

**|**

**| https\_open(self, req)**

**|**

**| https\_request = do\_request\_(self, request)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractHTTPHandler:**

**|**

**| do\_open(self, http\_class, req, \*\*http\_conn\_args)**

**| Return an addinfourl object for the request, using http\_class.**

**|**

**| http\_class must implement the HTTPConnection API from httplib.**

**| The addinfourl return value is a file-like object. It also**

**| has methods and attributes including:**

**| - info(): return a mimetools.Message object for the headers**

**| - geturl(): return the original request URL**

**| - code: HTTP status code**

**|**

**| do\_request\_(self, request)**

**|**

**| set\_http\_debuglevel(self, level)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class OpenerDirector**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self)**

**|**

**| add\_handler(self, handler)**

**|**

**| close(self)**

**|**

**| error(self, proto, \*args)**

**|**

**| open(self, fullurl, data=None, timeout=<object object>)**

**class ProxyBasicAuthHandler(AbstractBasicAuthHandler, BaseHandler)**

**| Method resolution order:**

**| ProxyBasicAuthHandler**

**| AbstractBasicAuthHandler**

**| BaseHandler**

**|**

**| Methods defined here:**

**|**

**| http\_error\_407(self, req, fp, code, msg, headers)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| auth\_header = 'Proxy-authorization'**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractBasicAuthHandler:**

**|**

**| \_\_init\_\_(self, password\_mgr=None)**

**|**

**| http\_error\_auth\_reqed(self, authreq, host, req, headers)**

**|**

**| retry\_http\_basic\_auth(self, host, req, realm)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from AbstractBasicAuthHandler:**

**|**

**| rx = <\_sre.SRE\_Pattern object>**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**class ProxyDigestAuthHandler(BaseHandler, AbstractDigestAuthHandler)**

**| Method resolution order:**

**| ProxyDigestAuthHandler**

**| BaseHandler**

**| AbstractDigestAuthHandler**

**|**

**| Methods defined here:**

**|**

**| http\_error\_407(self, req, fp, code, msg, headers)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| auth\_header = 'Proxy-Authorization'**

**|**

**| handler\_order = 490**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from AbstractDigestAuthHandler:**

**|**

**| \_\_init\_\_(self, passwd=None)**

**|**

**| get\_algorithm\_impls(self, algorithm)**

**|**

**| get\_authorization(self, req, chal)**

**|**

**| get\_cnonce(self, nonce)**

**|**

**| get\_entity\_digest(self, data, chal)**

**|**

**| http\_error\_auth\_reqed(self, auth\_header, host, req, headers)**

**|**

**| reset\_retry\_count(self)**

**|**

**| retry\_http\_digest\_auth(self, req, auth)**

**class ProxyHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, proxies=None)**

**|**

**| proxy\_open(self, req, proxy, type)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes defined here:**

**|**

**| handler\_order = 100**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**class Request**

**| Methods defined here:**

**|**

**| \_\_getattr\_\_(self, attr)**

**|**

**| \_\_init\_\_(self, url, data=None, headers={}, origin\_req\_host=None, unverifiable=False)**

**|**

**| add\_data(self, data)**

**|**

**| add\_header(self, key, val)**

**|**

**| add\_unredirected\_header(self, key, val)**

**|**

**| get\_data(self)**

**|**

**| get\_full\_url(self)**

**|**

**| get\_header(self, header\_name, default=None)**

**|**

**| get\_host(self)**

**|**

**| get\_method(self)**

**|**

**| get\_origin\_req\_host(self)**

**|**

**| get\_selector(self)**

**|**

**| get\_type(self)**

**|**

**| has\_data(self)**

**|**

**| has\_header(self, header\_name)**

**|**

**| has\_proxy(self)**

**|**

**| header\_items(self)**

**|**

**| is\_unverifiable(self)**

**|**

**| set\_proxy(self, host, type)**

**class URLError(exceptions.IOError)**

**| Method resolution order:**

**| URLError**

**| exceptions.IOError**

**| exceptions.EnvironmentError**

**| exceptions.StandardError**

**| exceptions.Exception**

**| exceptions.BaseException**

**| \_\_builtin\_\_.object**

**|**

**| Methods defined here:**

**|**

**| \_\_init\_\_(self, reason)**

**| # URLError is a sub-type of IOError, but it doesn't share any of**

**| # the implementation. need to override \_\_init\_\_ and \_\_str\_\_.**

**| # It sets self.args for compatibility with other EnvironmentError**

**| # subclasses, but args doesn't have the typical format with errno in**

**| # slot 0 and strerror in slot 1. This may be better than nothing.**

**|**

**| \_\_str\_\_(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors defined here:**

**|**

**| \_\_weakref\_\_**

**| list of weak references to the object (if defined)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from exceptions.IOError:**

**|**

**| \_\_new\_\_ = <built-in method \_\_new\_\_ of type object>**

**| T.\_\_new\_\_(S, ...) -> a new object with type S, a subtype of T**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from exceptions.EnvironmentError:**

**|**

**| \_\_reduce\_\_(...)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors inherited from exceptions.EnvironmentError:**

**|**

**| errno**

**| exception errno**

**|**

**| filename**

**| exception filename**

**|**

**| strerror**

**| exception strerror**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from exceptions.BaseException:**

**|**

**| \_\_delattr\_\_(...)**

**| x.\_\_delattr\_\_('name') <==> del x.name**

**|**

**| \_\_getattribute\_\_(...)**

**| x.\_\_getattribute\_\_('name') <==> x.name**

**|**

**| \_\_getitem\_\_(...)**

**| x.\_\_getitem\_\_(y) <==> x[y]**

**|**

**| \_\_getslice\_\_(...)**

**| x.\_\_getslice\_\_(i, j) <==> x[i:j]**

**|**

**| Use of negative indices is not supported.**

**|**

**| \_\_repr\_\_(...)**

**| x.\_\_repr\_\_() <==> repr(x)**

**|**

**| \_\_setattr\_\_(...)**

**| x.\_\_setattr\_\_('name', value) <==> x.name = value**

**|**

**| \_\_setstate\_\_(...)**

**|**

**| \_\_unicode\_\_(...)**

**|**

**| ----------------------------------------------------------------------**

**| Data descriptors inherited from exceptions.BaseException:**

**|**

**| \_\_dict\_\_**

**|**

**| args**

**|**

**| message**

**class UnknownHandler(BaseHandler)**

**| Methods defined here:**

**|**

**| unknown\_open(self, req)**

**|**

**| ----------------------------------------------------------------------**

**| Methods inherited from BaseHandler:**

**|**

**| \_\_lt\_\_(self, other)**

**|**

**| add\_parent(self, parent)**

**|**

**| close(self)**

**|**

**| ----------------------------------------------------------------------**

**| Data and other attributes inherited from BaseHandler:**

**|**

**| handler\_order = 500**

**FUNCTIONS**

**StringIO(...)**

**StringIO([s]) -- Return a StringIO-like stream for reading or writing**

**build\_opener(\*handlers)**

**Create an opener object from a list of handlers.**

**The opener will use several default handlers, including support**

**for HTTP, FTP and when applicable, HTTPS.**

**If any of the handlers passed as arguments are subclasses of the**

**default handlers, the default handlers will not be used.**

**install\_opener(opener)**

**parse\_http\_list(s)**

**Parse lists as described by RFC 2068 Section 2.**

**In particular, parse comma-separated lists where the elements of**

**the list may include quoted-strings. A quoted-string could**

**contain a comma. A non-quoted string could have quotes in the**

**middle. Neither commas nor quotes count if they are escaped.**

**Only double-quotes count, not single-quotes.**

**parse\_keqv\_list(l)**

**Parse list of key=value strings where keys are not duplicated.**

**randombytes(n)**

**Return n random bytes.**

**request\_host(request)**

**Return request-host, as defined by RFC 2965.**

**Variation from RFC: returned value is lowercased, for convenient**

**comparison.**

**urlopen(url, data=None, timeout=<object object>, cafile=None, capath=None, cadefault=False, context=None)**

**DATA**

**\_\_version\_\_ = '2.7'**

**VERSION**

**2.7**

### [[Python]网络爬虫（一）：抓取网页的含义和URL基本构成](http://blog.csdn.net/pleasecallmewhy/article/details/8922826)

**一、网络爬虫的定义**

网络爬虫，即Web Spider，是一个很形象的名字。

把互联网比喻成一个蜘蛛网，那么Spider就是在网上爬来爬去的蜘蛛。  
网络蜘蛛是通过网页的链接地址来寻找网页的。

从网站某一个页面（通常是首页）开始，读取网页的内容，找到在网页中的其它链接地址，

然后通过这些链接地址寻找下一个网页，这样一直循环下去，直到把这个网站所有的网页都抓取完为止。

如果把整个互联网当成一个网站，那么网络蜘蛛就可以用这个原理把互联网上所有的网页都抓取下来。

这样看来，网络爬虫就是一个爬行程序，一个抓取网页的程序。

网络爬虫的基本操作是抓取网页。

那么如何才能随心所欲地获得自己想要的页面？

我们先从URL开始。

**二、浏览网页的过程**

抓取网页的过程其实和读者平时使用IE浏览器浏览网页的道理是一样的。

比如说你在浏览器的地址栏中输入    www.baidu.com    这个地址。

打开网页的过程其实就是浏览器作为一个浏览的“客户端”，向服务器端发送了 一次请求，把服务器端的文件“抓”到本地，再进行解释、展现。

HTML是一种标记语言，用标签标记内容并加以解析和区分。

浏览器的功能是将获取到的HTML代码进行解析，然后将原始的代码转变成我们直接看到的网站页面。

**三、URI和URL的概念和举例**

简单的来讲，URL就是在浏览器端输入的    http://www.baidu.com    这个字符串。

在理解URL之前，首先要理解URI的概念。

什么是URI？

Web上每种可用的资源，如 HTML文档、图像、视频片段、程序等都由一个通用资源标志符(Universal Resource Identifier， URI)进行定位。

URI通常由三部分组成：

①访问资源的命名机制；

②存放资源的主机名；

③资源自身 的名称，由路径表示。

如下面的URI：  
http://www.why.com.cn/myhtml/html1223/

我们可以这样解释它：

①这是一个可以通过HTTP协议访问的资源，

②位于主机 www.webmonkey.com.cn上，

③通过路径“/html/html40”访问。

**四、URL的理解和举例**

URL是URI的一个子集。它是Uniform Resource Locator的缩写，译为“统一资源定位 符”。

通俗地说，URL是Internet上描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上。

采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。

URL的一般格式为(带方括号[]的为可选项)：

***protocol :// hostname[:port] / path / [;parameters][?query]#fragment***

URL的格式由三部分组成：

①第一部分是协议(或称为服务方式)。

②第二部分是存有该资源的主机IP地址(有时也包括端口号)。

③第三部分是主机资源的具体地址，如目录和文件名等。

第一部分和第二部分用“://”符号隔开，

第二部分和第三部分用“/”符号隔开。

第一部分和第二部分是不可缺少的，第三部分有时可以省略。

五、URL和URI简单比较

URI属于URL更低层次的抽象，一种字符串文本标准。

换句话说，URI属于父类，而URL属于URI的子类。URL是URI的一个子集。

URI的定义是：统一资源标识符；

URL的定义是：统一资源定位符。

二者的区别在于，URI表示请求服务器的路径，定义这么一个资源。

而URL同时说明要如何访问这个资源（http://）。

下面来看看两个URL的小例子。

1.HTTP协议的URL示例：  
使用超级文本传输协议HTTP，提供超级文本信息服务的资源。

例：http://www.peopledaily.com.cn/channel/welcome.htm

其计算机域名为www.peopledaily.com.cn。

超级文本文件(文件类型为.html)是在目录 /channel下的welcome.htm。

这是中国人民日报的一台计算机。

例：http://www.rol.cn.net/talk/talk1.htm

其计算机域名为www.rol.cn.net。

超级文本文件(文件类型为.html)是在目录/talk下的talk1.htm。

这是瑞得聊天室的地址，可由此进入瑞得聊天室的第1室。

2．文件的URL  
用URL表示文件时，服务器方式用file表示，后面要有主机IP地址、文件的存取路 径(即目录)和文件名等信息。

有时可以省略目录和文件名，但“/”符号不能省略。

例：file://ftp.yoyodyne.com/pub/files/foobar.txt

上面这个URL代表存放在主机ftp.yoyodyne.com上的pub/files/目录下的一个文件，文件名是foobar.txt。

例：file://ftp.yoyodyne.com/pub

代表主机ftp.yoyodyne.com上的目录/pub。

例：file://ftp.yoyodyne.com/

代表主机ftp.yoyodyne.com的根目录。

爬虫最主要的处理对象就是URL，它根据URL地址取得所需要的文件内容，然后对它 进行进一步的处理。

因此，准确地理解URL对理解网络爬虫至关重要。

### [[Python]网络爬虫（二）：利用urllib2通过指定的URL抓取网页内容](http://blog.csdn.net/pleasecallmewhy/article/details/8923067)

版本号：Python2.7.5，Python3改动较大，各位另寻教程。

所谓网页抓取，就是把URL地址中指定的网络资源从网络流中读取出来，保存到本地。   
类似于使用程序模拟IE浏览器的功能，把URL作为HTTP请求的内容发送到服务器端， 然后读取服务器端的响应资源。

在Python中，我们使用urllib2这个组件来抓取网页。  
urllib2是Python的一个获取URLs(Uniform Resource Locators)的组件。

它以urlopen函数的形式提供了一个非常简单的接口。

最简单的urllib2的应用代码只需要四行。

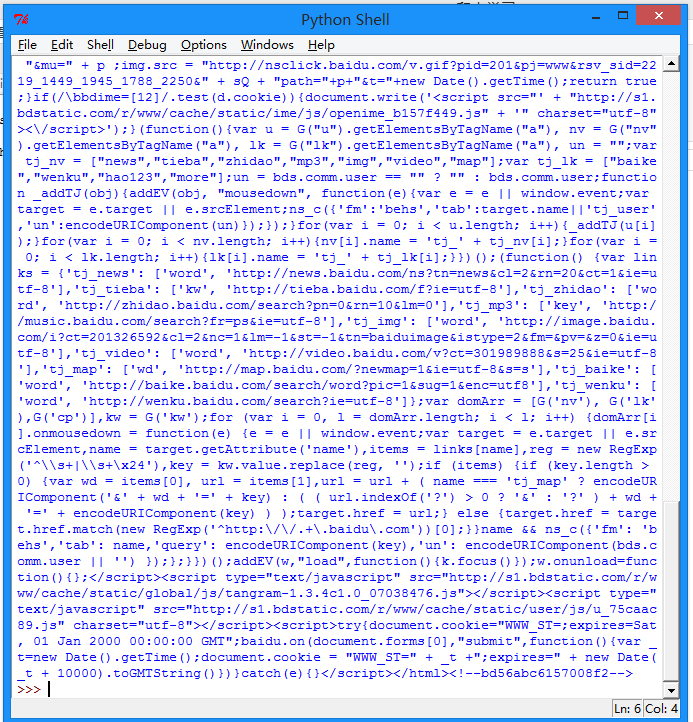
我们新建一个文件urllib2\_test01.py来感受一下urllib2的作用：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. **import** urllib2
2. response = urllib2.urlopen('http://www.baidu.com/')
3. html = response.read()
4. **print** html

按下F5可以看到运行的结果：



我们可以打开百度主页，右击，选择查看源代码（火狐OR谷歌浏览器均可），会发现也是完全一样的内容。

也就是说，上面这四行代码将我们访问百度时浏览器收到的代码们全部打印了出来。

这就是一个最简单的urllib2的例子。

除了"http:"，URL同样可以使用"ftp:"，"file:"等等来替代。

HTTP是基于请求和应答机制的：

客户端提出请求，服务端提供应答。

urllib2用一个Request对象来映射你提出的HTTP请求。

在它最简单的使用形式中你将用你要请求的地址创建一个Request对象，

通过调用urlopen并传入Request对象，将返回一个相关请求response对象，

这个应答对象如同一个文件对象，所以你可以在Response中调用.read()。

我们新建一个文件urllib2\_test02.py来感受一下：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. **import** urllib2
2. req = urllib2.Request('http://www.baidu.com')
3. response = urllib2.urlopen(req)
4. the\_page = response.read()
5. **print** the\_page

可以看到输出的内容和test01是一样的。

urllib2使用相同的接口处理所有的URL头。例如你可以像下面那样创建一个ftp请求。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. req = urllib2.Request('ftp://example.com/')

在HTTP请求时，允许你做额外的两件事。

**1.发送data表单数据**

这个内容相信做过Web端的都不会陌生，

有时候你希望发送一些数据到URL(通常URL与CGI[通用网关接口]脚本，或其他WEB应用程序挂接)。

在HTTP中,这个经常使用熟知的POST请求发送。

这个通常在你提交一个HTML表单时由你的浏览器来做。

并不是所有的POSTs都来源于表单，你能够使用POST提交任意的数据到你自己的程序。

一般的HTML表单，data需要编码成标准形式。然后做为data参数传到Request对象。

编码工作使用urllib的函数而非urllib2。

我们新建一个文件urllib2\_test03.py来感受一下：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. **import** urllib
2. **import** urllib2
4. url = 'http://www.someserver.com/register.cgi'
6. values = {'name' : 'WHY',
7. 'location' : 'SDU',
8. 'language' : 'Python' }
10. data = urllib.urlencode(values) # 编码工作
11. req = urllib2.Request(url, data)  # 发送请求同时传data表单
12. response = urllib2.urlopen(req)  #接受反馈的信息
13. the\_page = response.read()  #读取反馈的内容

如果没有传送data参数，urllib2使用GET方式的请求。

GET和POST请求的不同之处是POST请求通常有"副作用"，

它们会由于某种途径改变系统状态(例如提交成堆垃圾到你的门口)。

Data同样可以通过在Get请求的URL本身上面编码来传送。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. **import** urllib2
2. **import** urllib
4. data = {}
6. data['name'] = 'WHY'
7. data['location'] = 'SDU'
8. data['language'] = 'Python'
10. url\_values = urllib.urlencode(data)
11. **print** url\_values
13. name=Somebody+Here&language=Python&location=Northampton
14. url = 'http://www.example.com/example.cgi'
15. full\_url = url + '?' + url\_values
17. data = urllib2.open(full\_url)

这样就实现了Data数据的Get传送。

**2.设置Headers到http请求**

有一些站点不喜欢被程序（非人为访问）访问，或者发送不同版本的内容到不同的浏览器。

默认的urllib2把自己作为“Python-urllib/x.y”(x和y是Python主版本和次版本号,例如Python-urllib/2.7)，  
  
这个身份可能会让站点迷惑，或者干脆不工作。

浏览器确认自己身份是通过User-Agent头，当你创建了一个请求对象，你可以给他一个包含头数据的字典。

下面的例子发送跟上面一样的内容，但把自身模拟成Internet Explorer。

（多谢大家的提醒，现在这个Demo已经不可用了，不过原理还是那样的）。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923067" \o "copy)



1. **import** urllib
2. **import** urllib2
4. url = 'http://www.someserver.com/cgi-bin/register.cgi'
6. user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
7. values = {'name' : 'WHY',
8. 'location' : 'SDU',
9. 'language' : 'Python' }
11. headers = { 'User-Agent' : user\_agent }
12. data = urllib.urlencode(values)
13. req = urllib2.Request(url, data, headers)
14. response = urllib2.urlopen(req)
15. the\_page = response.read()

### [[Python]网络爬虫（三）：异常的处理和HTTP状态码的分类](http://blog.csdn.net/pleasecallmewhy/article/details/8923725)

先来说一说HTTP的异常处理问题。  
当urlopen不能够处理一个response时，产生urlError。  
不过通常的Python APIs异常如ValueError,TypeError等也会同时产生。  
HTTPError是urlError的子类，通常在特定HTTP URLs中产生。  
   
**1.URLError**  
通常，URLError在没有网络连接(没有路由到特定服务器)，或者服务器不存在的情况下产生。

这种情况下，异常同样会带有"reason"属性，它是一个tuple（可以理解为不可变的数组），

包含了一个错误号和一个错误信息。

我们建一个urllib2\_test06.py来感受一下异常的处理：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "copy)

1. **import** urllib2
3. req = urllib2.Request('http://www.baibai.com')
5. **try**: urllib2.urlopen(req)
7. **except** urllib2.URLError, e:
8. **print** e.reason

按下F5，可以看到打印出来的内容是：

[Errno 11001] getaddrinfo failed

也就是说，错误号是11001，内容是getaddrinfo failed

**2.HTTPError**服务器上每一个HTTP 应答对象response包含一个数字"状态码"。

有时状态码指出服务器无法完成请求。默认的处理器会为你处理一部分这种应答。

例如:假如response是一个"重定向"，需要客户端从别的地址获取文档，urllib2将为你处理。

其他不能处理的，urlopen会产生一个HTTPError。

典型的错误包含"404"(页面无法找到)，"403"(请求禁止)，和"401"(带验证请求)。

HTTP状态码表示HTTP协议所返回的响应的状态。

比如客户端向服务器发送请求，如果成功地获得请求的资源，则返回的状态码为200，表示响应成功。

如果请求的资源不存在， 则通常返回404错误。

HTTP状态码通常分为5种类型，分别以1～5五个数字开头，由3位整数组成：

------------------------------------------------------------------------------------------------

200：请求成功      处理方式：获得响应的内容，进行处理

201：请求完成，结果是创建了新资源。新创建资源的URI可在响应的实体中得到    处理方式：爬虫中不会遇到

202：请求被接受，但处理尚未完成    处理方式：阻塞等待

204：服务器端已经实现了请求，但是没有返回新的信 息。如果客户是用户代理，则无须为此更新自身的文档视图。    处理方式：丢弃

300：该状态码不被HTTP/1.0的应用程序直接使用， 只是作为3XX类型回应的默认解释。存在多个可用的被请求资源。    处理方式：若程序中能够处理，则进行进一步处理，如果程序中不能处理，则丢弃  
301：请求到的资源都会分配一个永久的URL，这样就可以在将来通过该URL来访问此资源    处理方式：重定向到分配的URL  
302：请求到的资源在一个不同的URL处临时保存     处理方式：重定向到临时的URL

304 请求的资源未更新     处理方式：丢弃

400 非法请求     处理方式：丢弃

401 未授权     处理方式：丢弃

403 禁止     处理方式：丢弃

404 没有找到     处理方式：丢弃

5XX 回应代码以“5”开头的状态码表示服务器端发现自己出现错误，不能继续执行请求    处理方式：丢弃

------------------------------------------------------------------------------------------------

HTTPError实例产生后会有一个整型'code'属性，是服务器发送的相关错误号。

Error Codes错误码  
因为默认的处理器处理了重定向(300以外号码)，并且100-299范围的号码指示成功，所以你只能看到400-599的错误号码。  
BaseHTTPServer.BaseHTTPRequestHandler.response是一个很有用的应答号码字典，显示了HTTP协议使用的所有的应答号。

当一个错误号产生后，服务器返回一个HTTP错误号，和一个错误页面。

你可以使用HTTPError实例作为页面返回的应答对象response。

这表示和错误属性一样，它同样包含了read,geturl,和info方法。

我们建一个urllib2\_test07.py来感受一下：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "copy)

1. **import** urllib2
2. req = urllib2.Request('http://bbs.csdn.net/callmewhy')
4. **try**:
5. urllib2.urlopen(req)
7. **except** urllib2.URLError, e:
9. **print** e.code
10. #print e.read()

按下F5可以看见输出了404的错误码，也就说没有找到这个页面。

**3.Wrapping**

所以如果你想为HTTPError或URLError做准备，将有两个基本的办法。推荐使用第二种。

我们建一个urllib2\_test08.py来示范一下第一种异常处理的方案：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "copy)

1. **from** urllib2 **import** Request, urlopen, URLError, HTTPError
3. req = Request('http://bbs.csdn.net/callmewhy')
5. **try**:
7. response = urlopen(req)
9. **except** HTTPError, e:
11. **print** 'The server couldn\'t fulfill the request.'
13. **print** 'Error code: ', e.code
15. **except** URLError, e:
17. **print** 'We failed to reach a server.'
19. **print** 'Reason: ', e.reason
21. **else**:
22. **print** 'No exception was raised.'
23. # everything is fine

和其他语言相似，try之后捕获异常并且将其内容打印出来。

这里要注意的一点，except HTTPError 必须在第一个，否则except URLError将同样接受到HTTPError 。  
因为HTTPError是URLError的子类，如果URLError在前面它会捕捉到所有的URLError（包括HTTPError ）。

我们建一个urllib2\_test09.py来示范一下第二种异常处理的方案：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8923725" \o "copy)

1. **from** urllib2 **import** Request, urlopen, URLError, HTTPError
3. req = Request('http://bbs.csdn.net/callmewhy')
5. **try**:
7. response = urlopen(req)
9. **except** URLError, e:
11. **if** hasattr(e, 'code'):
12. #hasattr用于确定一个对象是否具有某个属性。
13. 语法：
14. hasattr(object, name) -> bool
15. 判断object中是否有name属性，返回一个布尔值。
16. **print** 'The server couldn\'t fulfill the request.'
18. **print** 'Error code: ', e.code
20. **elif** hasattr(e, 'reason'):
21. **print** 'We failed to reach a server.'
23. **print** 'Reason: ', e.reason
25. **else**:
26. **print** 'No exception was raised.'
27. # everything is fine

### [[Python]网络爬虫（四）：Opener与Handler的介绍和实例应用](http://blog.csdn.net/pleasecallmewhy/article/details/8924889)

更好的学习网址：[http://www.voidspace.org.uk/python/articles/urllib2.shtml#openers-and-handlers](http://www.voidspace.org.uk/python/articles/urllib2.shtml" \l "openers-and-handlers" \t "_blank)

以下为个人学习笔记。

在开始后面的内容之前，先来解释一下urllib2中的两个个方法：info and geturl

urlopen返回的应答对象response(或者HTTPError实例)有两个很有用的方法info()和geturl()

**1.geturl()：**

这个返回获取的真实的URL，这个很有用，因为urlopen(或者opener对象使用的)或许会有重定向。获取的URL或许跟请求URL不同。

以人人中的一个超级链接为例,

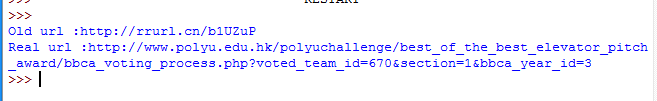
我们建一个urllib2\_test10.py来比较一下原始URL和重定向的链接：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "copy)

1. **from** urllib2 **import** Request, urlopen, URLError, HTTPError

4. old\_url = 'http://rrurl.cn/b1UZuP'
5. req = Request(old\_url)
6. response = urlopen(req)
7. **print** 'Old url :' + old\_url
8. **print** 'Real url :' + response.geturl()

运行之后可以看到真正的链接指向的网址：



**2.info()：**

这个返回对象的字典对象，该字典描述了获取的页面情况。通常是服务器发送的特定头headers。目前是httplib.HTTPMessage 实例。

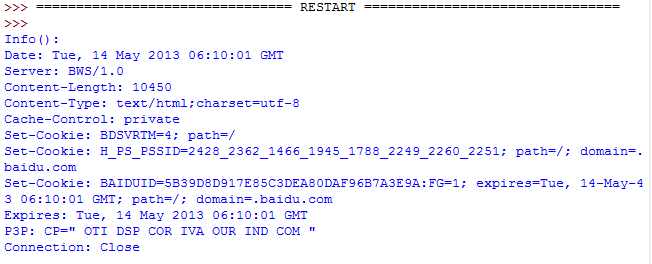
经典的headers包含"Content-length"，"Content-type"，和其他内容。

我们建一个urllib2\_test11.py来测试一下info的应用：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "copy)

1. **from** urllib2 **import** Request, urlopen, URLError, HTTPError
3. old\_url = 'http://www.baidu.com'
4. req = Request(old\_url)
5. response = urlopen(req)
6. **print** 'Info():'
7. **print** response.info()

运行的结果如下，可以看到页面的相关信息：



下面来说一说urllib2中的两个重要概念：Openers和Handlers。

**1.Openers：**

当你获取一个URL你使用一个opener(一个urllib2.OpenerDirector的实例)。

正常情况下，我们使用默认opener：通过urlopen。

但你能够创建个性的openers。

**2.Handles：**

Openers使用处理器handlers，所有的“繁重”工作由handlers处理。

每个handlers知道如何通过特定协议打开URLs，或者如何处理URL打开时的各个方面。

例如HTTP重定向或者HTTP cookies。

如果你希望用特定处理器获取URLs你会想创建一个openers，例如获取一个能处理cookie的opener，或者获取一个不重定向的opener。

要创建一个 opener，可以实例化一个OpenerDirector，

然后调用.add\_handler(some\_handler\_instance)。

同样，可以使用build\_opener，这是一个更加方便的函数，用来创建opener对象，他只需要一次函数调用。  
build\_opener默认添加几个处理器，但提供快捷的方法来添加或更新默认处理器。

其他的处理器handlers你或许会希望处理代理，验证，和其他常用但有点特殊的情况。

install\_opener 用来创建（全局）默认opener。这个表示调用urlopen将使用你安装的opener。

Opener对象有一个open方法。

该方法可以像urlopen函数那样直接用来获取urls：通常不必调用install\_opener，除了为了方便。

说完了上面两个内容，下面我们来看一下基本认证的内容，这里会用到上面提及的Opener和Handler。

Basic Authentication 基本验证

为了展示创建和安装一个handler，我们将使用HTTPBasicAuthHandler。

当需要基础验证时，服务器发送一个header(401错误码) 请求验证。这个指定了scheme 和一个‘realm’，看起来像这样：Www-authenticate: SCHEME realm="REALM".

例如  
Www-authenticate: Basic realm="cPanel Users"

客户端必须使用新的请求，并在请求头里包含正确的姓名和密码。

这是“基础验证”，为了简化这个过程，我们可以创建一个HTTPBasicAuthHandler的实例，并让opener使用这个handler就可以啦。

HTTPBasicAuthHandler使用一个密码管理的对象来处理URLs和realms来映射用户名和密码。

如果你知道realm(从服务器发送来的头里)是什么，你就能使用HTTPPasswordMgr。

通常人们不关心realm是什么。那样的话，就能用方便的HTTPPasswordMgrWithDefaultRealm。

这个将在你为URL指定一个默认的用户名和密码。

这将在你为特定realm提供一个其他组合时得到提供。

我们通过给realm参数指定None提供给add\_password来指示这种情况。

最高层次的URL是第一个要求验证的URL。你传给.add\_password()更深层次的URLs将同样合适。

说了这么多废话，下面来用一个例子演示一下上面说到的内容。

我们建一个urllib2\_test12.py来测试一下info的应用：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8924889" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. **import** urllib2
4. # 创建一个密码管理者
5. password\_mgr = urllib2.HTTPPasswordMgrWithDefaultRealm()
7. # 添加用户名和密码
9. top\_level\_url = "http://example.com/foo/"
11. # 如果知道 realm, 我们可以使用他代替 ``None``.
12. # password\_mgr.add\_password(None, top\_level\_url, username, password)
13. password\_mgr.add\_password(None, top\_level\_url,'why', '1223')
15. # 创建了一个新的handler
16. handler = urllib2.HTTPBasicAuthHandler(password\_mgr)
18. # 创建 "opener" (OpenerDirector 实例)
19. opener = urllib2.build\_opener(handler)
21. a\_url = 'http://www.baidu.com/'
23. # 使用 opener 获取一个URL
24. opener.open(a\_url)
26. # 安装 opener.
27. # 现在所有调用 urllib2.urlopen 将用我们的 opener.
28. urllib2.install\_opener(opener)

注意：以上的例子我们仅仅提供我们的HHTPBasicAuthHandler给build\_opener。

默认的openers有正常状况的handlers：ProxyHandler，UnknownHandler，HTTPHandler，HTTPDefaultErrorHandler， HTTPRedirectHandler，FTPHandler， FileHandler， HTTPErrorProcessor。

代码中的top\_level\_url 实际上可以是完整URL(包含"http:"，以及主机名及可选的端口号)。

例如：http://example.com/。

也可以是一个“authority”(即主机名和可选的包含端口号)。

例如：“example.com” or “example.com:8080”。

后者包含了端口号。

### [[Python]网络爬虫（五）：urllib2的使用细节与抓站技巧](http://blog.csdn.net/pleasecallmewhy/article/details/8925978)

前面说到了urllib2的简单入门，下面整理了一部分urllib2的使用细节。

**1.Proxy 的设置**

urllib2 默认会使用环境变量 http\_proxy 来设置 HTTP Proxy。

如果想在程序中明确控制 Proxy 而不受环境变量的影响，可以使用代理。

新建test14来实现一个简单的代理Demo：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. enable\_proxy = True
3. proxy\_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
4. null\_proxy\_handler = urllib2.ProxyHandler({})
5. **if** enable\_proxy:
6. opener = urllib2.build\_opener(proxy\_handler)
7. **else**:
8. opener = urllib2.build\_opener(null\_proxy\_handler)
9. urllib2.install\_opener(opener)

这里要注意的一个细节，使用 urllib2.install\_opener() 会设置 urllib2 的全局 opener 。

这样后面的使用会很方便，但不能做更细致的控制，比如想在程序中使用两个不同的 Proxy 设置等。

比较好的做法是不使用 install\_opener 去更改全局的设置，而只是直接调用 opener 的 open 方法代替全局的 urlopen 方法。

**2.Timeout 设置**在老版 Python 中（Python2.6前），urllib2 的 API 并没有暴露 Timeout 的设置，要设置 Timeout 值，只能更改 Socket 的全局 Timeout 值。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. **import** socket
3. socket.setdefaulttimeout(10) # 10 秒钟后超时
4. urllib2.socket.setdefaulttimeout(10) # 另一种方式

在 Python 2.6 以后，超时可以通过 urllib2.urlopen() 的 timeout 参数直接设置。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. response = urllib2.urlopen('http://www.google.com', timeout=10)

**3.在 HTTP Request 中加入特定的 Header**

要加入 header，需要使用 Request 对象：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. request = urllib2.Request('http://www.baidu.com/')
3. request.add\_header('User-Agent', 'fake-client')
4. response = urllib2.urlopen(request)
5. **print** response.read()

对有些 header 要特别留意，服务器会针对这些 header 做检查  
User-Agent : 有些服务器或 Proxy 会通过该值来判断是否是浏览器发出的请求  
Content-Type : 在使用 REST 接口时，服务器会检查该值，用来确定 HTTP Body 中的内容该怎样解析。常见的取值有：  
application/xml ： 在 XML RPC，如 RESTful/SOAP 调用时使用  
application/json ： 在 JSON RPC 调用时使用  
application/x-www-form-urlencoded ： 浏览器提交 Web 表单时使用  
在使用服务器提供的 RESTful 或 SOAP 服务时， Content-Type 设置错误会导致服务器拒绝服务

**4.Redirect**urllib2 默认情况下会针对 HTTP 3XX 返回码自动进行 redirect 动作，无需人工配置。要检测是否发生了 redirect 动作，只要检查一下 Response 的 URL 和 Request 的 URL 是否一致就可以了。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. my\_url = 'http://www.google.cn'
3. response = urllib2.urlopen(my\_url)
4. redirected = response.geturl() == my\_url
5. **print** redirected
7. my\_url = 'http://rrurl.cn/b1UZuP'
8. response = urllib2.urlopen(my\_url)
9. redirected = response.geturl() == my\_url
10. **print** redirected

如果不想自动 redirect，除了使用更低层次的 httplib 库之外，还可以自定义HTTPRedirectHandler 类。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. **class** RedirectHandler(urllib2.HTTPRedirectHandler):
3. **def** http\_error\_301(self, req, fp, code, msg, headers):
4. **print** "301"
5. **pass**
6. **def** http\_error\_302(self, req, fp, code, msg, headers):
7. **print** "303"
8. **pass**
10. opener = urllib2.build\_opener(RedirectHandler)
11. opener.open('http://rrurl.cn/b1UZuP')

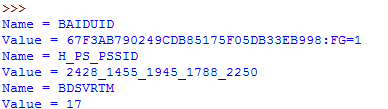
**5.Cookie**

urllib2 对 Cookie 的处理也是自动的。如果需要得到某个 Cookie 项的值，可以这么做：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. **import** cookielib
3. cookie = cookielib.CookieJar()
4. opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))
5. response = opener.open('http://www.baidu.com')
6. **for** item **in** cookie:
7. **print** 'Name = '+item.name
8. **print** 'Value = '+item.value

运行之后就会输出访问百度的Cookie值：



**6.使用 HTTP 的 PUT 和 DELETE 方法**

urllib2 只支持 HTTP 的 GET 和 POST 方法，如果要使用 HTTP PUT 和 DELETE ，只能使用比较低层的 httplib 库。虽然如此，我们还是能通过下面的方式，使 urllib2 能够发出 PUT 或DELETE 的请求：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. request = urllib2.Request(uri, data=data)
3. request.get\_method = **lambda**: 'PUT' # or 'DELETE'
4. response = urllib2.urlopen(request)

**7.得到 HTTP 的返回码**

对于 200 OK 来说，只要使用 urlopen 返回的 response 对象的 getcode() 方法就可以得到 HTTP 的返回码。但对其它返回码来说，urlopen 会抛出异常。这时候，就要检查异常对象的 code 属性了：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. **try**:
3. response = urllib2.urlopen('http://bbs.csdn.net/why')
4. **except** urllib2.HTTPError, e:
5. **print** e.code

**8.Debug Log**

使用 urllib2 时，可以通过下面的方法把 debug Log 打开，这样收发包的内容就会在屏幕上打印出来，方便调试，有时可以省去抓包的工作

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. **import** urllib2
2. httpHandler = urllib2.HTTPHandler(debuglevel=1)
3. httpsHandler = urllib2.HTTPSHandler(debuglevel=1)
4. opener = urllib2.build\_opener(httpHandler, httpsHandler)
5. urllib2.install\_opener(opener)
6. response = urllib2.urlopen('http://www.google.com')

这样就可以看到传输的数据包内容了：



**9.表单的处理**

登录必要填表，表单怎么填？

首先利用工具截取所要填表的内容。  
比如我一般用firefox+httpfox插件来看看自己到底发送了些什么包。  
以verycd为例，先找到自己发的POST请求，以及POST表单项。  
可以看到verycd的话需要填username,password,continueURI,fk,login\_submit这几项，其中fk是随机生成的（其实不太随机，看上去像是把epoch时间经过简单的编码生成的），需要从网页获取，也就是说得先访问一次网页，用正则表达式等工具截取返回数据中的fk项。continueURI顾名思义可以随便写，login\_submit是固定的，这从源码可以看出。还有username，password那就很显然了：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. **import** urllib
3. **import** urllib2
4. postdata=urllib.urlencode({
5. 'username':'汪小光',
6. 'password':'why888',
7. 'continueURI':'http://www.verycd.com/',
8. 'fk':'',
9. 'login\_submit':'登录'
10. })
11. req = urllib2.Request(
12. url = 'http://secure.verycd.com/signin',
13. data = postdata
14. )
15. result = urllib2.urlopen(req)
16. **print** result.read()

**10.伪装成浏览器访问**  
某些网站反感爬虫的到访，于是对爬虫一律拒绝请求  
这时候我们需要伪装成浏览器，这可以通过修改http包中的header来实现

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8925978" \o "copy)

1. #…
3. headers = {
4. 'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6'
5. }
6. req = urllib2.Request(
7. url = 'http://secure.verycd.com/signin/\*/http://www.verycd.com/',
8. data = postdata,
9. headers = headers
10. )
11. #...

**11.对付"反盗链"**  
某些站点有所谓的反盗链设置，其实说穿了很简单，

就是检查你发送请求的header里面，referer站点是不是他自己，

所以我们只需要像把headers的referer改成该网站即可，以cnbeta为例：

#...

headers = {

'Referer':'http://www.cnbeta.com/articles'

}

#...

headers是一个dict数据结构，你可以放入任何想要的header，来做一些伪装。

例如，有些网站喜欢读取header中的X-Forwarded-For来看看人家的真实IP，可以直接把X-Forwarde-For改了。

### [[Python]网络爬虫（六）：一个简单的百度贴吧的小爬虫](http://blog.csdn.net/pleasecallmewhy/article/details/8927832)

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8927832" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8927832" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：百度贴吧爬虫
4. #   版本：0.1
5. #   作者：why
6. #   日期：2013-05-14
7. #   语言：Python 2.7
8. #   操作：输入带分页的地址，去掉最后面的数字，设置一下起始页数和终点页数。
9. #   功能：下载对应页码内的所有页面并存储为html文件。
10. #---------------------------------------
12. **import** string, urllib2
14. #定义百度函数
15. **def** baidu\_tieba(url,begin\_page,end\_page):
16. **for** i **in** range(begin\_page, end\_page+1):
17. sName = string.zfill(i,5) + '.html'#自动填充成六位的文件名
18. **print** '正在下载第' + str(i) + '个网页，并将其存储为' + sName + '......'
19. f = open(sName,'w+')
20. m = urllib2.urlopen(url + str(i)).read()
21. f.write(m)
22. f.close()

25. #-------- 在这里输入参数 ------------------
27. # 这个是山东大学的百度贴吧中某一个帖子的地址
28. #bdurl = 'http://tieba.baidu.com/p/2296017831?pn='
29. #iPostBegin = 1
30. #iPostEnd = 10
32. bdurl = str(raw\_input(u'请输入贴吧的地址，去掉pn=后面的数字：\n'))
33. begin\_page = int(raw\_input(u'请输入开始的页数：\n'))
34. end\_page = int(raw\_input(u'请输入终点的页数：\n'))
35. #-------- 在这里输入参数 ------------------

38. #调用
39. baidu\_tieba(bdurl,begin\_page,end\_page)

### [[Python]网络爬虫（七）：Python中的正则表达式教程](http://blog.csdn.net/pleasecallmewhy/article/details/8929576)

1. [2 数量词的贪婪模式与非贪婪模式](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \l "t0)
2. [3 反斜杠的问题](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \l "t1)
3. [二 介绍re模块](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \l "t2)
4. [1  Compile](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \l "t3)
   1. [3 Pattern](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \l "t4)

接下来准备用糗百做一个爬虫的小例子。

但是在这之前，先详细的整理一下Python中的正则表达式的相关内容。

正则表达式在Python爬虫中的作用就像是老师点名时用的花名册一样，是必不可少的神兵利器。

以下内容转自CNBLOG：[http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html](http://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html" \t "_blank)

整理时没有注意，实在抱歉。

**一、 正则表达式基础**

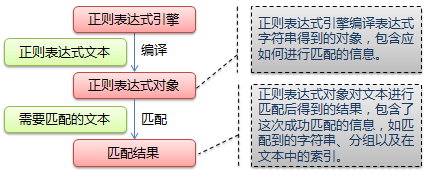
**1.1.概念介绍**

正则表达式是用于处理字符串的强大工具，它并不是Python的一部分。

其他编程语言中也有正则表达式的概念，区别只在于不同的编程语言实现支持的语法数量不同。

它拥有自己独特的语法以及一个独立的处理引擎，在提供了正则表达式的语言里，正则表达式的语法都是一样的。

下图展示了使用正则表达式进行匹配的流程：



正则表达式的大致匹配过程是：

1.依次拿出表达式和文本中的字符比较，

2.如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。

3.如果表达式中有量词或边界，这个过程会稍微有一些不同。  
  
下图列出了Python支持的正则表达式元字符和语法：



**1.2. 数量词的贪婪模式与非贪婪模式**

正则表达式通常用于在文本中查找匹配的字符串。

贪婪模式，总是尝试匹配尽可能多的字符；

非贪婪模式则相反，总是尝试匹配尽可能少的字符。

Python里数量词默认是贪婪的。

例如：正则表达式"ab\*"如果用于查找"abbbc"，将找到"abbb"。

而如果使用非贪婪的数量词"ab\*?"，将找到"a"。

**1.3. 反斜杠的问题**

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。

假如你需要匹配文本中的字符"\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\\\"：

第一个和第三个用于在编程语言里将第二个和第四个转义成反斜杠，

转换成两个反斜杠\\后再在正则表达式里转义成一个反斜杠用来匹配反斜杠\。

这样显然是非常麻烦的。

Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r"\\"表示。

同样，匹配一个数字的"\\d"可以写成r"\d"。

有了原生字符串，妈妈再也不用担心我的反斜杠问题~

**二、 介绍re模块**

**2.1.  Compile**

Python通过re模块提供对正则表达式的支持。

使用re的一般步骤是：

Step1：先将正则表达式的字符串形式编译为Pattern实例。

Step2：然后使用Pattern实例处理文本并获得匹配结果（一个Match实例）。

Step3：最后使用Match实例获得信息，进行其他的操作。

我们新建一个re01.py来试验一下re的应用：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)

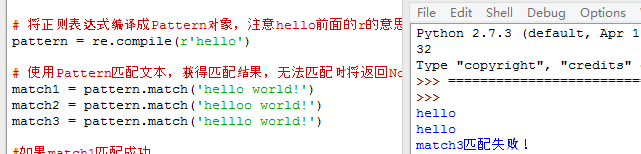


1. # -\*- coding: utf-8 -\*-
2. #一个简单的re实例，匹配字符串中的hello字符串
4. #导入re模块
5. **import** re
7. # 将正则表达式编译成Pattern对象，注意hello前面的r的意思是“原生字符串”
8. pattern = re.compile(r'hello')
10. # 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
11. match1 = pattern.match('hello world!')
12. match2 = pattern.match('helloo world!')
13. match3 = pattern.match('helllo world!')
15. #如果match1匹配成功
16. **if** match1:
17. # 使用Match获得分组信息
18. **print** match1.group()
19. **else**:
20. **print** 'match1匹配失败！'

23. #如果match2匹配成功
24. **if** match2:
25. # 使用Match获得分组信息
26. **print** match2.group()
27. **else**:
28. **print** 'match2匹配失败！'

31. #如果match3匹配成功
32. **if** match3:
33. # 使用Match获得分组信息
34. **print** match3.group()
35. **else**:
36. **print** 'match3匹配失败！'

可以看到控制台输出了匹配的三个结果：



下面来具体看看代码中的关键方法。

★ re.compile(strPattern[, flag]):

这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。

第二个参数flag是匹配模式，取值可以使用按位或运算符'|'表示同时生效，比如re.I | re.M。

另外，你也可以在regex字符串中指定模式，

比如re.compile('pattern', re.I | re.M)与re.compile('(?im)pattern')是等价的。

可选值有：

* re.I(全拼：IGNORECASE): 忽略大小写（括号内是完整写法，下同）
* re.M(全拼：MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
* re.S(全拼：DOTALL): 点任意匹配模式，改变'.'的行为
* re.L(全拼：LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
* re.U(全拼：UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于unicode定义的字符属性
* re.X(全拼：VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。

以下两个正则表达式是等价的：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #两个等价的re匹配,匹配一个小数
3. **import** re
5. a = re.compile(r"""\d +  # the integral part
6. \.    # the decimal point
7. \d \*  # some fractional digits""", re.X)
9. b = re.compile(r"\d+\.\d\*")
11. match11 = a.match('3.1415')
12. match12 = a.match('33')
13. match21 = b.match('3.1415')
14. match22 = b.match('33')
16. **if** match11:
17. # 使用Match获得分组信息
18. **print** match11.group()
19. **else**:
20. **print** u'match11不是小数'
22. **if** match12:
23. # 使用Match获得分组信息
24. **print** match12.group()
25. **else**:
26. **print** u'match12不是小数'
28. **if** match21:
29. # 使用Match获得分组信息
30. **print** match21.group()
31. **else**:
32. **print** u'match21不是小数'
34. **if** match22:
35. # 使用Match获得分组信息
36. **print** match22.group()
37. **else**:
38. **print** u'match22不是小数'

re提供了众多模块方法用于完成正则表达式的功能。

这些方法可以使用Pattern实例的相应方法替代，唯一的好处是少写一行re.compile()代码，

但同时也无法复用编译后的Pattern对象。

这些方法将在Pattern类的实例方法部分一起介绍。

如一开始的hello实例可以简写为：

**[html]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #一个简单的re实例，匹配字符串中的hello字符串
3. import re
5. m = re.match(r'hello', 'hello world!')
6. print m.group()

re模块还提供了一个方法escape(string)，用于将string中的正则表达式元字符如\*/+/?等之前加上转义符再返回

**2.2. Match**

Match对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用Match提供的可读属性或方法来获取这些信息。

属性：

1. string: 匹配时使用的文本。
2. re: 匹配时使用的Pattern对象。
3. pos: 文本中正则表达式开始搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
4. endpos: 文本中正则表达式结束搜索的索引。值与Pattern.match()和Pattern.seach()方法的同名参数相同。
5. lastindex: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为None。
6. lastgroup: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为None。

方法：

1. group([group1, …])：  
   获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。group1可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回group(0)；没有截获字符串的组返回None；截获了多次的组返回最后一次截获的子串。
2. groups([default])：   
   以元组形式返回全部分组截获的字符串。相当于调用group(1,2,…last)。default表示没有截获字符串的组以这个值替代，默认为None。
3. groupdict([default])：  
   返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。default含义同上。
4. start([group])：   
   返回指定的组截获的子串在string中的起始索引（子串第一个字符的索引）。group默认值为0。
5. end([group])：  
   返回指定的组截获的子串在string中的结束索引（子串最后一个字符的索引+1）。group默认值为0。
6. span([group])：  
   返回(start(group), end(group))。
7. expand(template)：   
   将匹配到的分组代入template中然后返回。template中可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。\id与\g<id>是等价的；但\10将被认为是第10个分组，如果你想表达\1之后是字符'0'，只能使用\g<1>0。

下面来用一个py实例输出所有的内容加深理解：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #一个简单的match实例
4. **import** re
5. # 匹配如下内容：单词+空格+单词+任意字符
6. m = re.match(r'(\w+) (\w+)(?P<sign>.\*)', 'hello world!')
8. **print** "m.string:", m.string
9. **print** "m.re:", m.re
10. **print** "m.pos:", m.pos
11. **print** "m.endpos:", m.endpos
12. **print** "m.lastindex:", m.lastindex
13. **print** "m.lastgroup:", m.lastgroup
15. **print** "m.group():", m.group()
16. **print** "m.group(1,2):", m.group(1, 2)
17. **print** "m.groups():", m.groups()
18. **print** "m.groupdict():", m.groupdict()
19. **print** "m.start(2):", m.start(2)
20. **print** "m.end(2):", m.end(2)
21. **print** "m.span(2):", m.span(2)
22. **print** r"m.expand(r'\g<2> \g<1>\g<3>'):", m.expand(r'\2 \1\3')
24. ### output ###
25. # m.string: hello world!
26. # m.re: <\_sre.SRE\_Pattern object at 0x016E1A38>
27. # m.pos: 0
28. # m.endpos: 12
29. # m.lastindex: 3
30. # m.lastgroup: sign
31. # m.group(1,2): ('hello', 'world')
32. # m.groups(): ('hello', 'world', '!')
33. # m.groupdict(): {'sign': '!'}
34. # m.start(2): 6
35. # m.end(2): 11
36. # m.span(2): (6, 11)
37. # m.expand(r'\2 \1\3'): world hello!

2.3. Pattern

Pattern对象是一个编译好的正则表达式，通过Pattern提供的一系列方法可以对文本进行匹配查找。

Pattern不能直接实例化，必须使用re.compile()进行构造，也就是re.compile()返回的对象。

Pattern提供了几个可读属性用于获取表达式的相关信息：

1. pattern: 编译时用的表达式字符串。
2. flags: 编译时用的匹配模式。数字形式。
3. groups: 表达式中分组的数量。
4. groupindex: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。

可以用下面这个例子查看pattern的属性：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #一个简单的pattern实例
4. **import** re
5. p = re.compile(r'(\w+) (\w+)(?P<sign>.\*)', re.DOTALL)
7. **print** "p.pattern:", p.pattern
8. **print** "p.flags:", p.flags
9. **print** "p.groups:", p.groups
10. **print** "p.groupindex:", p.groupindex
12. ### output ###
13. # p.pattern: (\w+) (\w+)(?P<sign>.\*)
14. # p.flags: 16
15. # p.groups: 3
16. # p.groupindex: {'sign': 3}

下面重点介绍一下pattern的实例方法及其使用。

**1.match**

match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])：

这个方法将从string的pos下标处起尝试匹配pattern；

如果pattern结束时仍可匹配，则返回一个Match对象；

如果匹配过程中pattern无法匹配，或者匹配未结束就已到达endpos，则返回None。

pos和endpos的默认值分别为0和len(string)；

re.match()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

注意：这个方法并不是完全匹配。

当pattern结束时若string还有剩余字符，仍然视为成功。

想要完全匹配，可以在表达式末尾加上边界匹配符'$'。

下面来看一个Match的简单案例：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # encoding: UTF-8
2. **import** re
4. # 将正则表达式编译成Pattern对象
5. pattern = re.compile(r'hello')
7. # 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
8. match = pattern.match('hello world!')
10. **if** match:
11. # 使用Match获得分组信息
12. **print** match.group()
14. ### 输出 ###
15. # hello

**2.search**  
search(string[, pos[, endpos]]) | re.search(pattern, string[, flags]): 这个方法用于查找字符串中可以匹配成功的子串。

从string的pos下标处起尝试匹配pattern，

如果pattern结束时仍可匹配，则返回一个Match对象；

若无法匹配，则将pos加1后重新尝试匹配；

直到pos=endpos时仍无法匹配则返回None。

pos和endpos的默认值分别为0和len(string))；

re.search()无法指定这两个参数，参数flags用于编译pattern时指定匹配模式。

那么它和match有什么区别呢？

match()函数只检测re是不是在string的开始位置匹配，

search()会扫描整个string查找匹配，

match（）只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，match()就返回none  
例如：  
print(re.match(‘super’, ‘superstition’).span())

会返回(0, 5)

print(re.match(‘super’, ‘insuperable’))

则返回None  
  
search()会扫描整个字符串并返回第一个成功的匹配  
例如：

print(re.search(‘super’, ‘superstition’).span())

返回(0, 5)  
print(re.search(‘super’, ‘insuperable’).span())

返回(2, 7)

看一个search的实例：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #一个简单的search实例
4. **import** re
6. # 将正则表达式编译成Pattern对象
7. pattern = re.compile(r'world')
9. # 使用search()查找匹配的子串，不存在能匹配的子串时将返回None
10. # 这个例子中使用match()无法成功匹配
11. match = pattern.search('hello world!')
13. **if** match:
14. # 使用Match获得分组信息
15. **print** match.group()
17. ### 输出 ###
18. # world

**3.split**

split(string[, maxsplit]) | re.split(pattern, string[, maxsplit]):按照能够匹配的子串将string分割后返回列表。

maxsplit用于指定最大分割次数，不指定将全部分割。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. **import** re
3. p = re.compile(r'\d+')
4. **print** p.split('one1two2three3four4')
6. ### output ###
7. # ['one', 'two', 'three', 'four', '']

**4.findall**

findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags]):搜索string，以列表形式返回全部能匹配的子串。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. **import** re
3. p = re.compile(r'\d+')
4. **print** p.findall('one1two2three3four4')
6. ### output ###
7. # ['1', '2', '3', '4']

**5.finditer**

finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags]):搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

**[html]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. import re
3. p = re.compile(r'\d+')
4. for m in p.finditer('one1two2three3four4'):
5. print m.group(),
7. ### output ###
8. # 1 2 3 4

**6.sub**

sub(repl, string[, count]) | re.sub(pattern, repl, string[, count]):使用repl替换string中每一个匹配的子串后返回替换后的字符串。   
当repl是一个字符串时，可以使用\id或\g<id>、\g<name>引用分组，但不能使用编号0。   
当repl是一个方法时，这个方法应当只接受一个参数（Match对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。   
count用于指定最多替换次数，不指定时全部替换。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. **import** re
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
6. **print** p.sub(r'\2 \1', s)
8. **def** func(m):
9. **return** m.group(1).title() + ' ' + m.group(2).title()
11. **print** p.sub(func, s)
13. ### output ###
14. # say i, world hello!
15. # I Say, Hello World!

**7.subn**

subn(repl, string[, count]) |re.sub(pattern, repl, string[, count]):返回 (sub(repl, string[, count]), 替换次数)。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8929576" \o "copy)



1. **import** re
3. p = re.compile(r'(\w+) (\w+)')
4. s = 'i say, hello world!'
6. **print** p.subn(r'\2 \1', s)
8. **def** func(m):
9. **return** m.group(1).title() + ' ' + m.group(2).title()
11. **print** p.subn(func, s)
13. ### output ###
14. # ('say i, world hello!', 2)
15. # ('I Say, Hello World!', 2)

至此，Python的正则表达式基本介绍就算是完成了^\_^

### [[Python]网络爬虫（八）：糗事百科的网络爬虫（v0.3）源码及解析(简化更新)](http://blog.csdn.net/pleasecallmewhy/article/details/8932310)

**Q&A：**

**1.为什么有段时间显示糗事百科不可用？**

**答：**前段时间因为糗事百科添加了Header的检验，导致无法爬取，需要在代码中模拟Header。现在代码已经作了修改，可以正常使用。

2.为什么需要单独新建个线程？

答：基本流程是这样的：爬虫在后台新起一个线程，一直爬取两页的糗事百科，如果剩余不足两页，则再爬一页。用户按下回车只是从库存中获取最新的内容，而不是上网获取，所以浏览更顺畅。也可以把加载放在主线程，不过这样会导致爬取过程中等待时间过长的问题。

**项目内容：**

用Python写的糗事百科的网络爬虫。

**使用方法：**

新建一个Bug.py文件，然后将代码复制到里面后，双击运行。

**程序功能：**

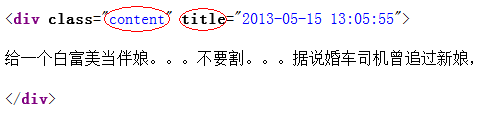
在命令提示行中浏览糗事百科。

**原理解释：**

首先，先浏览一下糗事百科的主页：[http://www.qiushibaike.com/hot/page/1](http://www.qiushibaike.com/hot/page/1" \t "_blank)

可以看出来，链接中page/后面的数字就是对应的页码，记住这一点为以后的编写做准备。

然后，右击查看页面源码：



观察发现，每一个段子都用div标记，其中class必为content，title是发帖时间，我们只需要用正则表达式将其“扣”出来就可以了。

明白了原理之后，剩下的就是正则表达式的内容了，可以参照这篇博文：

[http://blog.csdn.net/wxg694175346/article/details/8929576](http://blog.csdn.net/wxg694175346/article/details/8929576" \t "_blank)

**运行效果：**



**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8932310" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8932310" \o "copy)

1. # -\*- coding: utf-8 -\*-
3. **import** urllib2
4. **import** urllib
5. **import** re
6. **import** thread
7. **import** time

10. #----------- 加载处理糗事百科 -----------
11. **class** Spider\_Model:
13. **def** \_\_init\_\_(self):
14. self.page = 1
15. self.pages = []
16. self.enable = False
18. # 将所有的段子都扣出来，添加到列表中并且返回列表
19. **def** GetPage(self,page):
20. myUrl = "http://m.qiushibaike.com/hot/page/" + page
21. user\_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
22. headers = { 'User-Agent' : user\_agent }
23. req = urllib2.Request(myUrl, headers = headers)
24. myResponse = urllib2.urlopen(req)
25. myPage = myResponse.read()
26. #encode的作用是将unicode编码转换成其他编码的字符串
27. #decode的作用是将其他编码的字符串转换成unicode编码
28. unicodePage = myPage.decode("utf-8")
30. # 找出所有class="content"的div标记
31. #re.S是任意匹配模式，也就是.可以匹配换行符
32. myItems = re.findall('<div.\*?class="content".\*?title="(.\*?)">(.\*?)</div>',unicodePage,re.S)
33. items = []
34. **for** item **in** myItems:
35. # item 中第一个是div的标题，也就是时间
36. # item 中第二个是div的内容，也就是内容
37. items.append([item[0].replace("\n",""),item[1].replace("\n","")])
38. **return** items
40. # 用于加载新的段子
41. **def** LoadPage(self):
42. # 如果用户未输入quit则一直运行
43. **while** self.enable:
44. # 如果pages数组中的内容小于2个
45. **if** len(self.pages) < 2:
46. **try**:
47. # 获取新的页面中的段子们
48. myPage = self.GetPage(str(self.page))
49. self.page += 1
50. self.pages.append(myPage)
51. **except**:
52. **print** '无法链接糗事百科！'
53. **else**:
54. time.sleep(1)
56. **def** ShowPage(self,nowPage,page):
57. **for** items **in** nowPage:
58. **print** u'第%d页' % page , items[0]  , items[1]
59. myInput = raw\_input()
60. **if** myInput == "quit":
61. self.enable = False
62. **break**
64. **def** Start(self):
65. self.enable = True
66. page = self.page
68. **print** u'正在加载中请稍候......'
70. # 新建一个线程在后台加载段子并存储
71. thread.start\_new\_thread(self.LoadPage,())
73. #----------- 加载处理糗事百科 -----------
74. **while** self.enable:
75. # 如果self的page数组中存有元素
76. **if** self.pages:
77. nowPage = self.pages[0]
78. **del** self.pages[0]
79. self.ShowPage(nowPage,page)
80. page += 1

83. #----------- 程序的入口处 -----------
84. **print** u"""
85. ---------------------------------------
86. 程序：糗百爬虫
87. 版本：0.3
88. 作者：why
89. 日期：2014-06-03
90. 语言：Python 2.7
91. 操作：输入quit退出阅读糗事百科
92. 功能：按下回车依次浏览今日的糗百热点
93. ---------------------------------------
94. """

97. **print** u'请按下回车浏览今日的糗百内容：'
98. raw\_input(' ')
99. myModel = Spider\_Model()
100. myModel.Start()

### [[Python]网络爬虫（九）：百度贴吧的网络爬虫（v0.4）源码及解析](http://blog.csdn.net/pleasecallmewhy/article/details/8934726)

分类： [爬虫](http://blog.csdn.net/wxg694175346/article/category/1418998) [Python](http://blog.csdn.net/wxg694175346/article/category/1344887)2013-05-16 13:48 32373人阅读 [评论](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \l "comments)(53) [收藏](javascript:void(0);" \o "收藏) [举报](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \l "report" \o "举报)

更新：感谢评论中朋友的提醒，百度贴吧现在已经改成utf-8编码了吧，需要把代码中的decode('gbk')改成decode('utf-8')。

百度贴吧的爬虫制作和糗百的爬虫制作原理基本相同，都是通过查看源码扣出关键数据，然后将其存储到本地txt文件。

**源码下载：**

[http://download.csdn.net/detail/wxg694175346/6925583](http://download.csdn.net/detail/wxg694175346/6925583" \t "_blank)

**项目内容：**

用Python写的百度贴吧的网络爬虫。

**使用方法：**

新建一个BugBaidu.py文件，然后将代码复制到里面后，双击运行。

**程序功能：**

将贴吧中楼主发布的内容打包txt存储到本地。

**原理解释：**

首先，先浏览一下某一条贴吧，点击只看楼主并点击第二页之后url发生了一点变化，变成了：

http://tieba.baidu.com/p/2296712428?see\_lz=1&pn=1

可以看出来，see\_lz=1是只看楼主，pn=1是对应的页码，记住这一点为以后的编写做准备。

这就是我们需要利用的url。

接下来就是查看页面源码。

首先把题目抠出来存储文件的时候会用到。

可以看到百度使用gbk编码，标题使用h1标记：

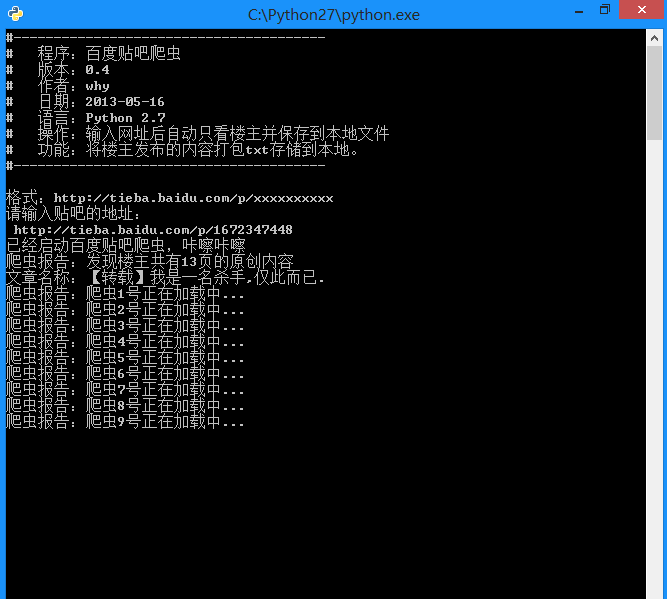
**[html]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \o "copy)



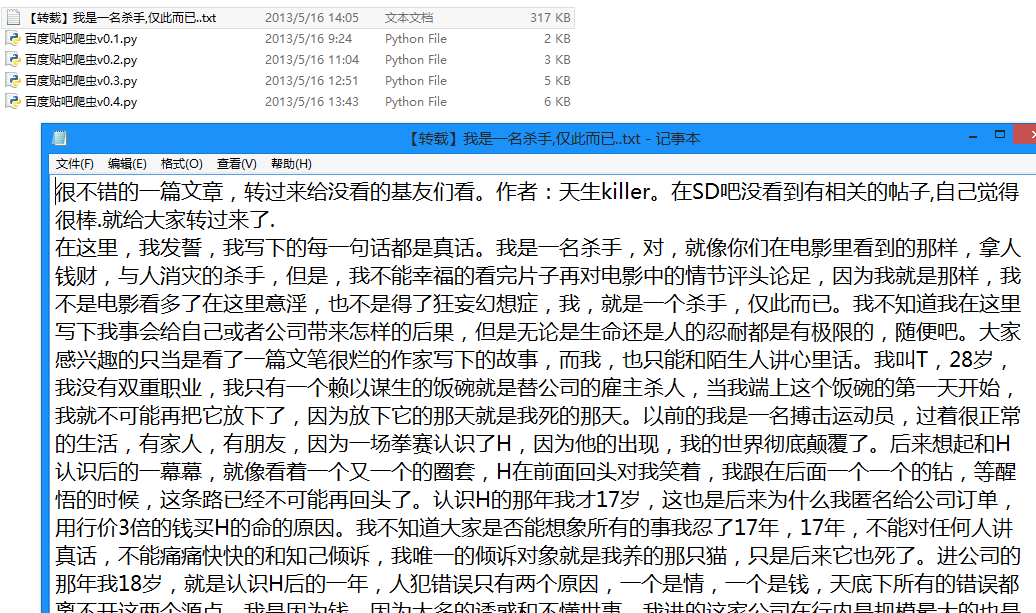
1. **<h1** class="core\_title\_txt" title="【原创】时尚首席（关于时尚，名利，事业，爱情，励志）"**>**【原创】时尚首席（关于时尚，名利，事业，爱情，励志）**</h1>**

同样，正文部分用div和class综合标记，接下来要做的只是用正则表达式来匹配即可。

运行截图：



生成的txt文件：



**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/8934726" \o "copy)



1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：百度贴吧爬虫
4. #   版本：0.5
5. #   作者：why
6. #   日期：2013-05-16
7. #   语言：Python 2.7
8. #   操作：输入网址后自动只看楼主并保存到本地文件
9. #   功能：将楼主发布的内容打包txt存储到本地。
10. #---------------------------------------
12. **import** string
13. **import** urllib2
14. **import** re
16. #----------- 处理页面上的各种标签 -----------
17. **class** HTML\_Tool:
18. # 用非 贪婪模式 匹配 \t 或者 \n 或者 空格 或者 超链接 或者 图片
19. BgnCharToNoneRex = re.compile("(\t|\n| |<a.\*?>|<img.\*?>)")
21. # 用非 贪婪模式 匹配 任意<>标签
22. EndCharToNoneRex = re.compile("<.\*?>")
24. # 用非 贪婪模式 匹配 任意<p>标签
25. BgnPartRex = re.compile("<p.\*?>")
26. CharToNewLineRex = re.compile("(<br/>|</p>|<tr>|<div>|</div>)")
27. CharToNextTabRex = re.compile("<td>")
29. # 将一些html的符号实体转变为原始符号
30. replaceTab = [("<","<"),(">",">"),("&","&"),("&","\""),(" "," ")]
32. **def** Replace\_Char(self,x):
33. x = self.BgnCharToNoneRex.sub("",x)
34. x = self.BgnPartRex.sub("\n    ",x)
35. x = self.CharToNewLineRex.sub("\n",x)
36. x = self.CharToNextTabRex.sub("\t",x)
37. x = self.EndCharToNoneRex.sub("",x)
39. **for** t **in** self.replaceTab:
40. x = x.replace(t[0],t[1])
41. **return** x
43. **class** Baidu\_Spider:
44. # 申明相关的属性
45. **def** \_\_init\_\_(self,url):
46. self.myUrl = url + '?see\_lz=1'
47. self.datas = []
48. self.myTool = HTML\_Tool()
49. **print** u'已经启动百度贴吧爬虫，咔嚓咔嚓'
51. # 初始化加载页面并将其转码储存
52. **def** baidu\_tieba(self):
53. # 读取页面的原始信息并将其从gbk转码
54. myPage = urllib2.urlopen(self.myUrl).read().decode("gbk")
55. # 计算楼主发布内容一共有多少页
56. endPage = self.page\_counter(myPage)
57. # 获取该帖的标题
58. title = self.find\_title(myPage)
59. **print** u'文章名称：' + title
60. # 获取最终的数据
61. self.save\_data(self.myUrl,title,endPage)
63. #用来计算一共有多少页
64. **def** page\_counter(self,myPage):
65. # 匹配 "共有<span class="red">12</span>页" 来获取一共有多少页
66. myMatch = re.search(r'class="red">(\d+?)</span>', myPage, re.S)
67. **if** myMatch:
68. endPage = int(myMatch.group(1))
69. **print** u'爬虫报告：发现楼主共有%d页的原创内容' % endPage
70. **else**:
71. endPage = 0
72. **print** u'爬虫报告：无法计算楼主发布内容有多少页！'
73. **return** endPage
75. # 用来寻找该帖的标题
76. **def** find\_title(self,myPage):
77. # 匹配 <h1 class="core\_title\_txt" title="">xxxxxxxxxx</h1> 找出标题
78. myMatch = re.search(r'<h1.\*?>(.\*?)</h1>', myPage, re.S)
79. title = u'暂无标题'
80. **if** myMatch:
81. title  = myMatch.group(1)
82. **else**:
83. **print** u'爬虫报告：无法加载文章标题！'
84. # 文件名不能包含以下字符： \ / ： \* ? " < > |
85. title = title.replace('\\','').replace('/','').replace(':','').replace('\*','').replace('?','').replace('"','').replace('>','').replace('<','').replace('|','')
86. **return** title

89. # 用来存储楼主发布的内容
90. **def** save\_data(self,url,title,endPage):
91. # 加载页面数据到数组中
92. self.get\_data(url,endPage)
93. # 打开本地文件
94. f = open(title+'.txt','w+')
95. f.writelines(self.datas)
96. f.close()
97. **print** u'爬虫报告：文件已下载到本地并打包成txt文件'
98. **print** u'请按任意键退出...'
99. raw\_input();
101. # 获取页面源码并将其存储到数组中
102. **def** get\_data(self,url,endPage):
103. url = url + '&pn='
104. **for** i **in** range(1,endPage+1):
105. **print** u'爬虫报告：爬虫%d号正在加载中...' % i
106. myPage = urllib2.urlopen(url + str(i)).read()
107. # 将myPage中的html代码处理并存储到datas里面
108. self.deal\_data(myPage.decode('gbk'))

111. # 将内容从页面代码中抠出来
112. **def** deal\_data(self,myPage):
113. myItems = re.findall('id="post\_content.\*?>(.\*?)</div>',myPage,re.S)
114. **for** item **in** myItems:
115. data = self.myTool.Replace\_Char(item.replace("\n","").encode('gbk'))
116. self.datas.append(data+'\n')


120. #-------- 程序入口处 ------------------
121. **print** u"""#---------------------------------------
122. #   程序：百度贴吧爬虫
123. #   版本：0.5
124. #   作者：why
125. #   日期：2013-05-16
126. #   语言：Python 2.7
127. #   操作：输入网址后自动只看楼主并保存到本地文件
128. #   功能：将楼主发布的内容打包txt存储到本地。
129. #---------------------------------------
130. """
132. # 以某小说贴吧为例子
133. # bdurl = 'http://tieba.baidu.com/p/2296712428?see\_lz=1&pn=1'
135. **print** u'请输入贴吧的地址最后的数字串：'
136. bdurl = 'http://tieba.baidu.com/p/' + str(raw\_input(u'http://tieba.baidu.com/p/'))
138. #调用
139. mySpider = Baidu\_Spider(bdurl)
140. mySpider.baidu\_tieba()

### [[Python]网络爬虫（十）：一个爬虫的诞生全过程（以山东大学绩点运算为例）](http://blog.csdn.net/pleasecallmewhy/article/details/9305229)

分类： [爬虫](http://blog.csdn.net/wxg694175346/article/category/1418998) [Python](http://blog.csdn.net/wxg694175346/article/category/1344887)2013-07-12 13:46 33341人阅读 [评论](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \l "comments)(82) [收藏](javascript:void(0);" \o "收藏) [举报](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \l "report" \o "举报)

先来说一下我们学校的网站：

[http://jwxt.sdu.edu.cn:7777/zhxt\_bks/zhxt\_bks.html](http://jwxt.sdu.edu.cn:7777/zhxt_bks/zhxt_bks.html" \t "_blank)

查询成绩需要登录，然后显示各学科成绩，但是只显示成绩而没有绩点，也就是加权平均分。



显然这样手动计算绩点是一件非常麻烦的事情。所以我们可以用python做一个爬虫来解决这个问题。

**1.决战前夜**

先来准备一下工具：HttpFox插件。

这是一款http协议分析插件，分析页面请求和响应的时间、内容、以及浏览器用到的COOKIE等。

以我为例，安装在火狐上即可，效果如图：



可以非常直观的查看相应的信息。

点击start是开始检测，点击stop暂停检测，点击clear清除内容。

一般在使用之前，点击stop暂停，然后点击clear清屏，确保看到的是访问当前页面获得的数据。

**2.深入敌后**

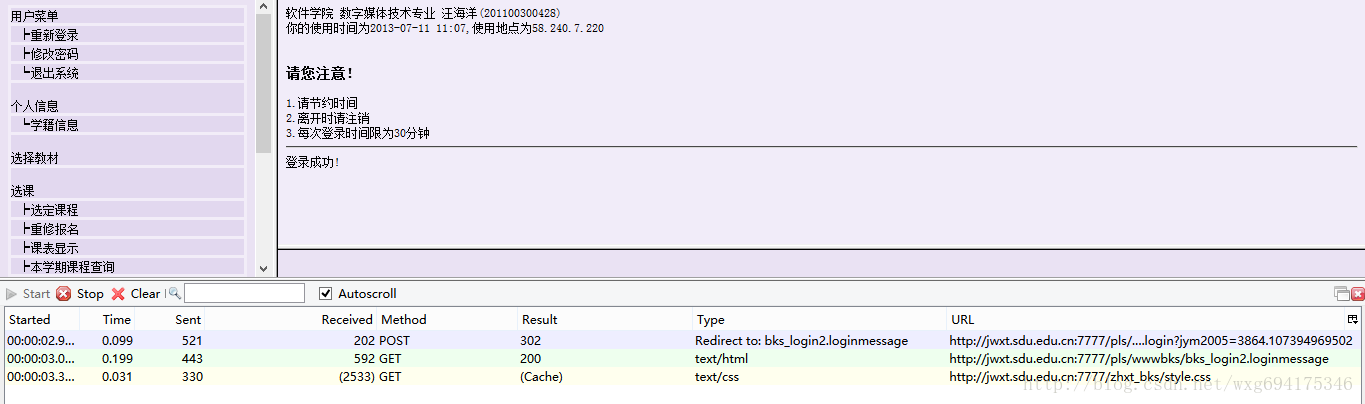
下面就去山东大学的成绩查询网站，看一看在登录的时候，到底发送了那些信息。

先来到登录页面，把httpfox打开，clear之后，点击start开启检测：



输入完了个人信息，确保httpfox处于开启状态，然后点击确定提交信息，实现登录。

这个时候可以看到，httpfox检测到了三条信息：



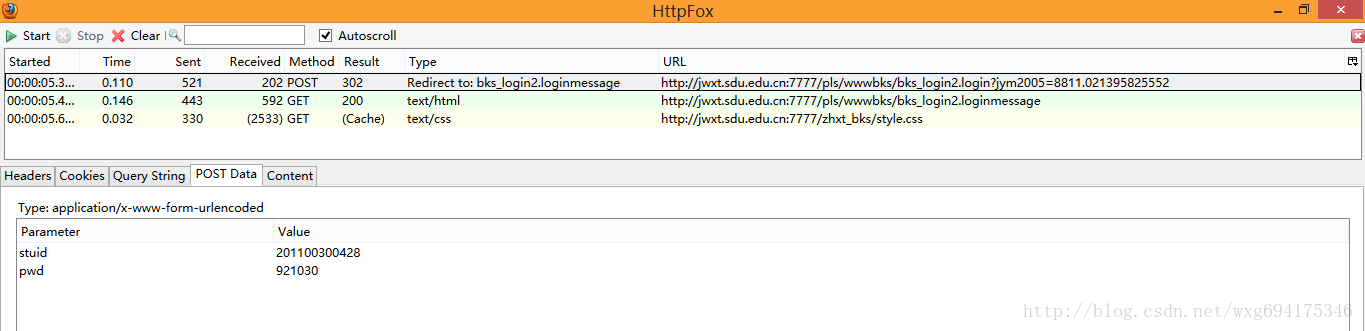
这时点击stop键，确保捕获到的是访问该页面之后反馈的数据，以便我们做爬虫的时候模拟登陆使用。

**3.庖丁解牛**

乍一看我们拿到了三个数据，两个是GET的一个是POST的，但是它们到底是什么，应该怎么用，我们还一无所知。

所以，我们需要挨个查看一下捕获到的内容。

先看POST的信息：



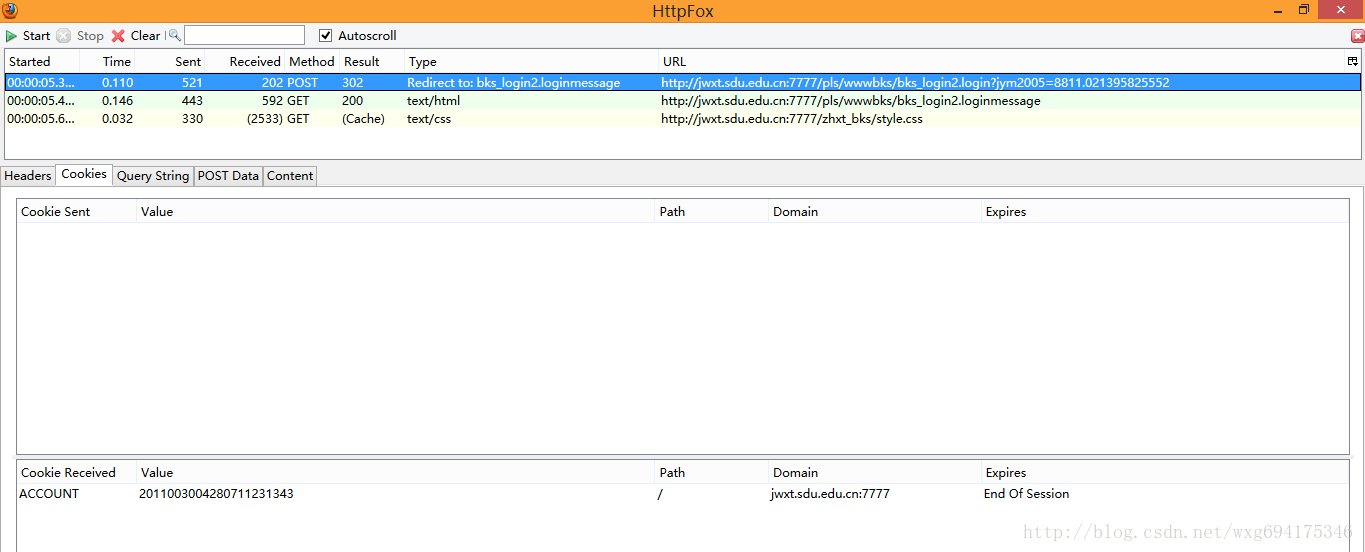
既然是POST的信息，我们就直接看PostData即可。

可以看到一共POST两个数据，stuid和pwd。

并且从Type的Redirect to可以看出，POST完毕之后跳转到了bks\_login2.loginmessage页面。

由此看出，这个数据是点击确定之后提交的表单数据。

点击cookie标签，看看cookie信息：

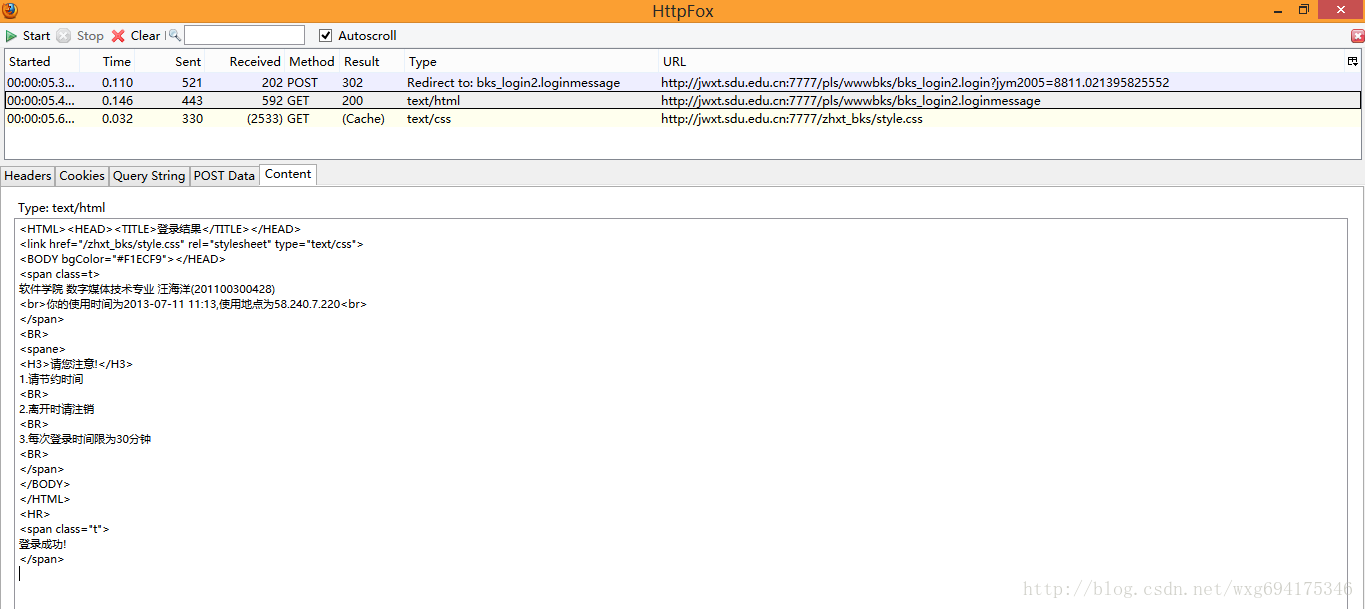


没错，收到了一个ACCOUNT的cookie，并且在session结束之后自动销毁。

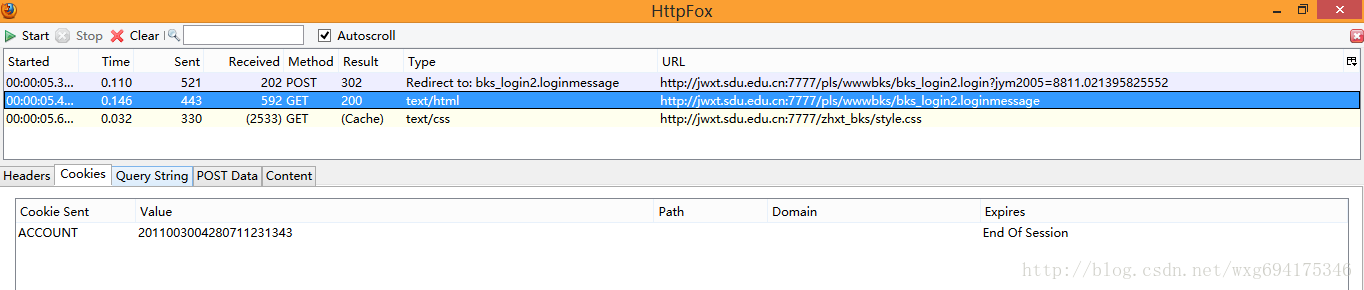
那么提交之后收到了哪些信息呢？

我们来看看后面的两个GET数据。

先看第一个，我们点击content标签可以查看收到的内容，是不是有一种生吞活剥的快感-。-HTML源码暴露无疑了：

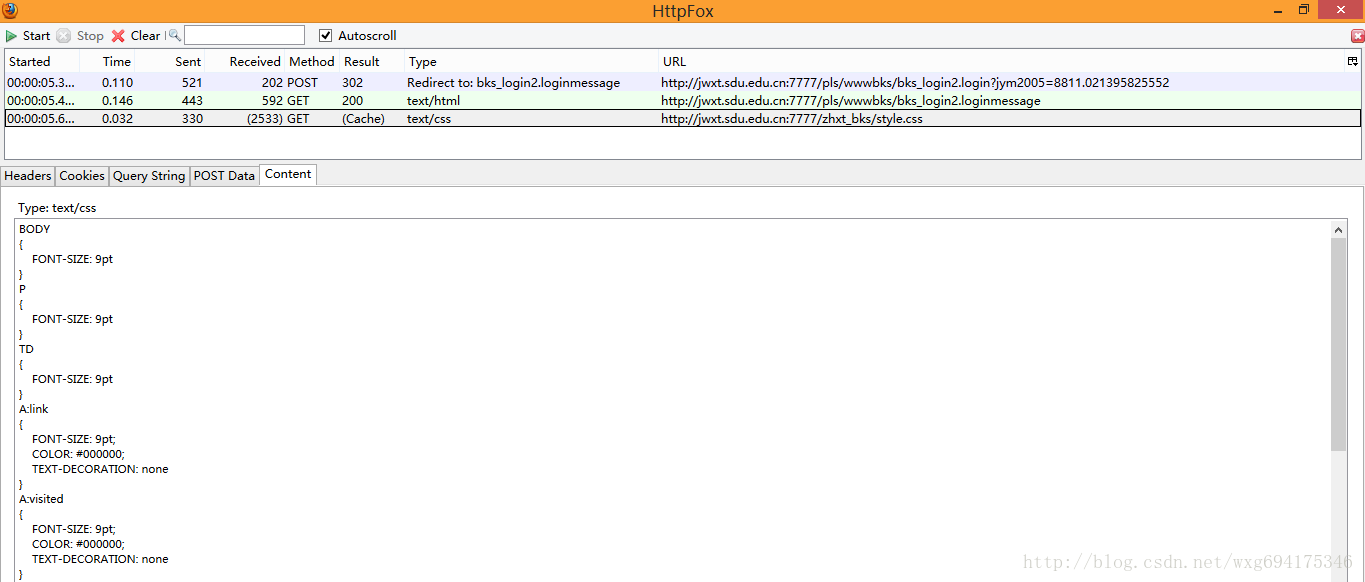


看来这个只是显示页面的html源码而已，点击cookie，查看cookie的相关信息：



啊哈，原来html页面的内容是发送了cookie信息之后才接受到的。

再来看看最后一个接收到的信息：



大致看了一下应该只是一个叫做style.css的css文件，对我们没有太大的作用。

**4.冷静应战**

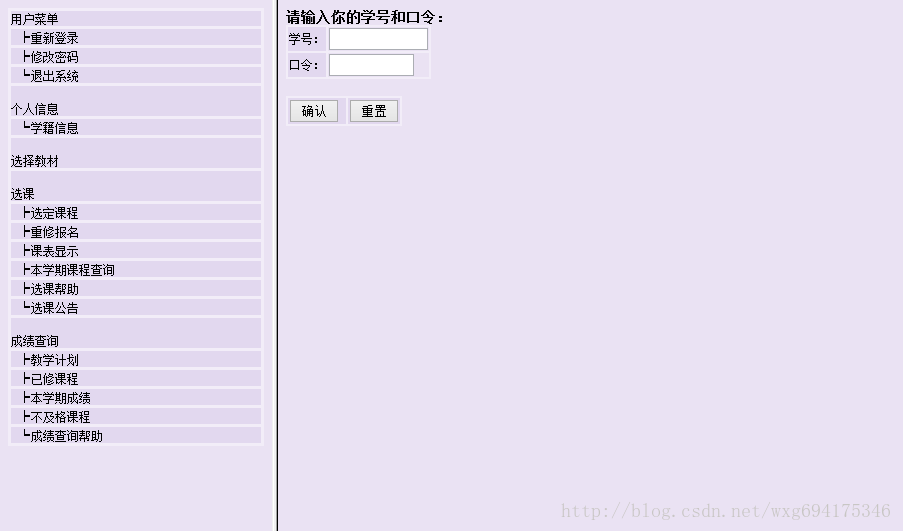
既然已经知道了我们向服务器发送了什么数据，也知道了我们接收到了什么数据，基本的流程如下：

* 首先，我们POST学号和密码--->然后返回cookie的值
* 然后发送cookie给服务器--->返回页面信息。
* 获取到成绩页面的数据，用正则表达式将成绩和学分单独取出并计算加权平均数。

OK，看上去好像很简单的样纸。那下面我们就来试试看吧。

但是在实验之前，还有一个问题没有解决，就是POST的数据到底发送到了哪里？

再来看一下当初的页面：



很明显是用一个html框架来实现的，也就是说，我们在地址栏看到的地址并不是右边提交表单的地址。

那么怎样才能获得真正的地址-。-右击查看页面源代码：

嗯没错，那个name="w\_right"的就是我们要的登录页面。

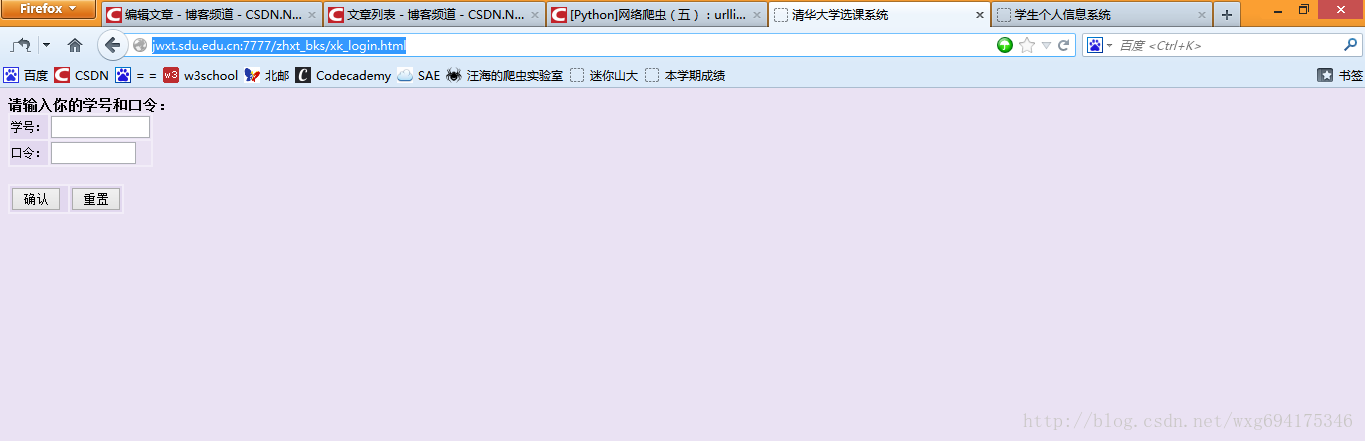
网站的原来的地址是：

http://jwxt.sdu.edu.cn:7777/zhxt\_bks/zhxt\_bks.html

所以，真正的表单提交的地址应该是：

http://jwxt.sdu.edu.cn:7777/zhxt\_bks/xk\_login.html

输入一看，果不其然：



靠居然是清华大学的选课系统。。。目测是我校懒得做页面了就直接借了。。结果连标题都不改一下。。。

但是这个页面依旧不是我们需要的页面，因为我们的POST数据提交到的页面，应该是表单form的ACTION中提交到的页面。

也就是说，我们需要查看源码，来知道POST数据到底发送到了哪里：



嗯，目测这个才是提交POST数据的地址。

整理到地址栏中，完整的地址应该如下：

[http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks\_login2.login](http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks_login2.login" \t "_blank)

（获取的方式很简单，在火狐浏览器中直接点击那个链接就能看到这个链接的地址了）

**5.小试牛刀**

接下来的任务就是：用python模拟发送一个POST的数据并取到返回的cookie值。

关于cookie的操作可以看看这篇博文：

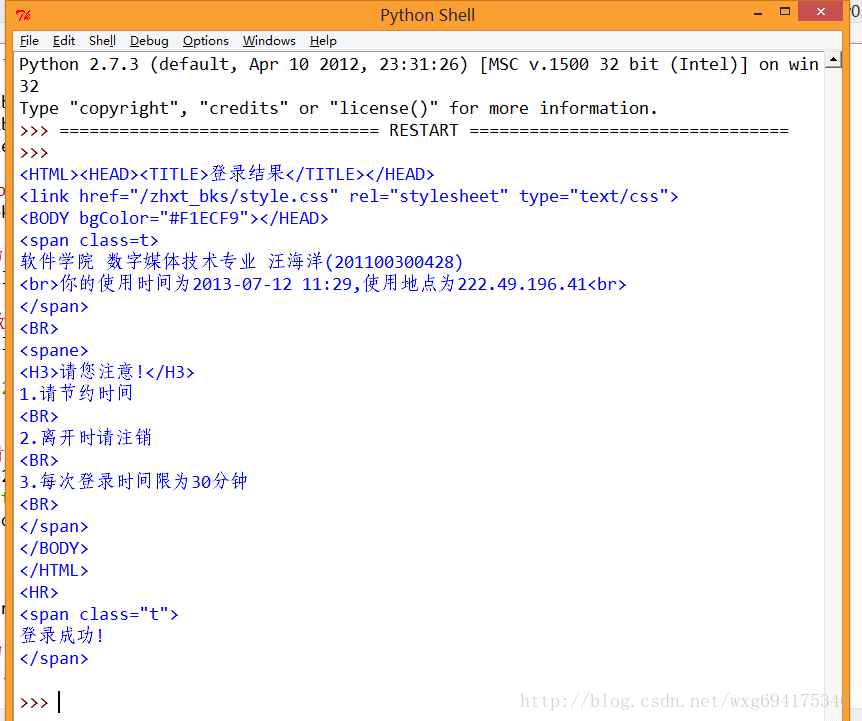
[http://blog.csdn.net/wxg694175346/article/details/8925978](http://blog.csdn.net/wxg694175346/article/details/8925978" \t "_blank)

我们先准备一个POST的数据，再准备一个cookie的接收，然后写出源码如下：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：山东大学爬虫
4. #   版本：0.1
5. #   作者：why
6. #   日期：2013-07-12
7. #   语言：Python 2.7
8. #   操作：输入学号和密码
9. #   功能：输出成绩的加权平均值也就是绩点
10. #---------------------------------------
12. **import** urllib
13. **import** urllib2
14. **import** cookielib
16. cookie = cookielib.CookieJar()
17. opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))
19. #需要POST的数据#
20. postdata=urllib.urlencode({
21. 'stuid':'201100300428',
22. 'pwd':'921030'
23. })
25. #自定义一个请求#
26. req = urllib2.Request(
27. url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks\_login2.login',
28. data = postdata
29. )
31. #访问该链接#
32. result = opener.open(req)
34. #打印返回的内容#
35. **print** result.read()

如此这般之后，再看看运行的效果：



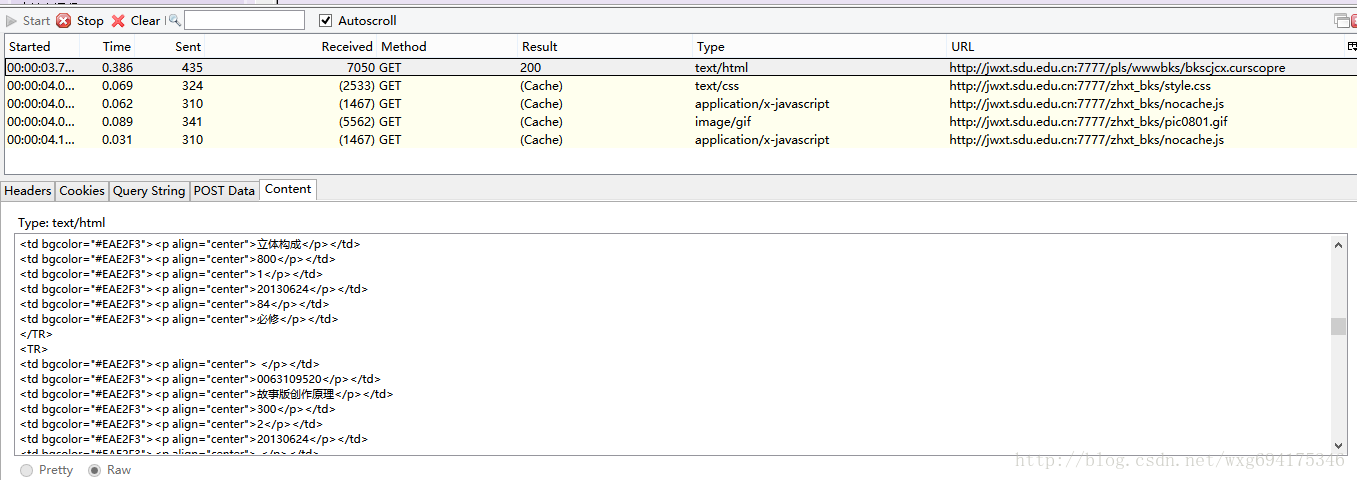
ok，如此这般，我们就算模拟登陆成功了。

**6.偷天换日**

接下来的任务就是用爬虫获取到学生的成绩。

再来看看源网站。

开启HTTPFOX之后，点击查看成绩，发现捕获到了如下的数据：



点击第一个GET的数据，查看内容可以发现Content就是获取到的成绩的内容。

而获取到的页面链接，从页面源代码中右击查看元素，可以看到点击链接之后跳转的页面（火狐浏览器只需要右击，“查看此框架”，即可）：



从而可以得到查看成绩的链接如下：

[http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre](http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre" \t "_blank)

**7.万事俱备**

现在万事俱备啦，所以只需要把链接应用到爬虫里面，看看能否查看到成绩的页面。

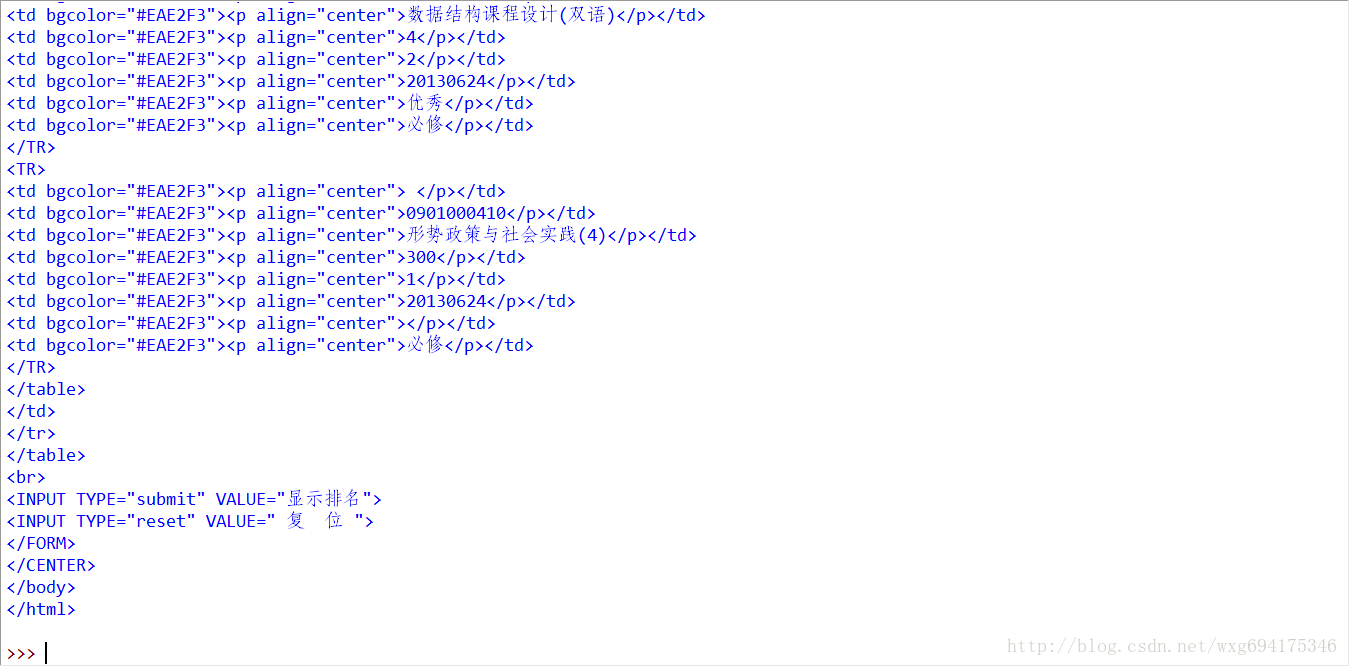
从httpfox可以看到，我们发送了一个cookie才能返回成绩的信息，所以我们就用python模拟一个cookie的发送，以此来请求成绩的信息：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：山东大学爬虫
4. #   版本：0.1
5. #   作者：why
6. #   日期：2013-07-12
7. #   语言：Python 2.7
8. #   操作：输入学号和密码
9. #   功能：输出成绩的加权平均值也就是绩点
10. #---------------------------------------
12. **import** urllib
13. **import** urllib2
14. **import** cookielib
16. #初始化一个CookieJar来处理Cookie的信息#
17. cookie = cookielib.CookieJar()
19. #创建一个新的opener来使用我们的CookieJar#
20. opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))
22. #需要POST的数据#
23. postdata=urllib.urlencode({
24. 'stuid':'201100300428',
25. 'pwd':'921030'
26. })
28. #自定义一个请求#
29. req = urllib2.Request(
30. url = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks\_login2.login',
31. data = postdata
32. )
34. #访问该链接#
35. result = opener.open(req)
37. #打印返回的内容#
38. **print** result.read()
40. #打印cookie的值
41. **for** item **in** cookie:
42. **print** 'Cookie：Name = '+item.name
43. **print** 'Cookie：Value = '+item.value

46. #访问该链接#
47. result = opener.open('http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre')
49. #打印返回的内容#
50. **print** result.read()

按下F5运行即可，看看捕获到的数据吧：



既然这样就没有什么问题了吧，用正则表达式将数据稍稍处理一下，取出学分和相应的分数就可以了。

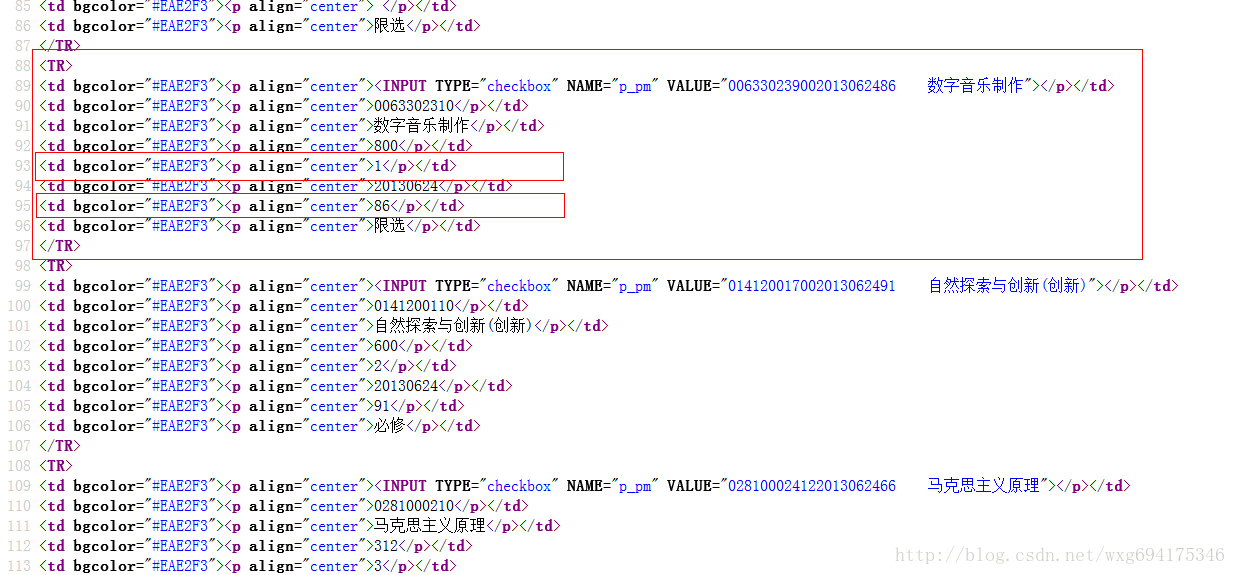
**8.手到擒来**

这么一大堆html源码显然是不利于我们处理的，下面要用正则表达式来抠出必须的数据。

关于正则表达式的教程可以看看这个博文：

[http://blog.csdn.net/wxg694175346/article/details/8929576](http://blog.csdn.net/wxg694175346/article/details/8929576" \t "_blank)

我们来看看成绩的源码：



既然如此，用正则表达式就易如反掌了。

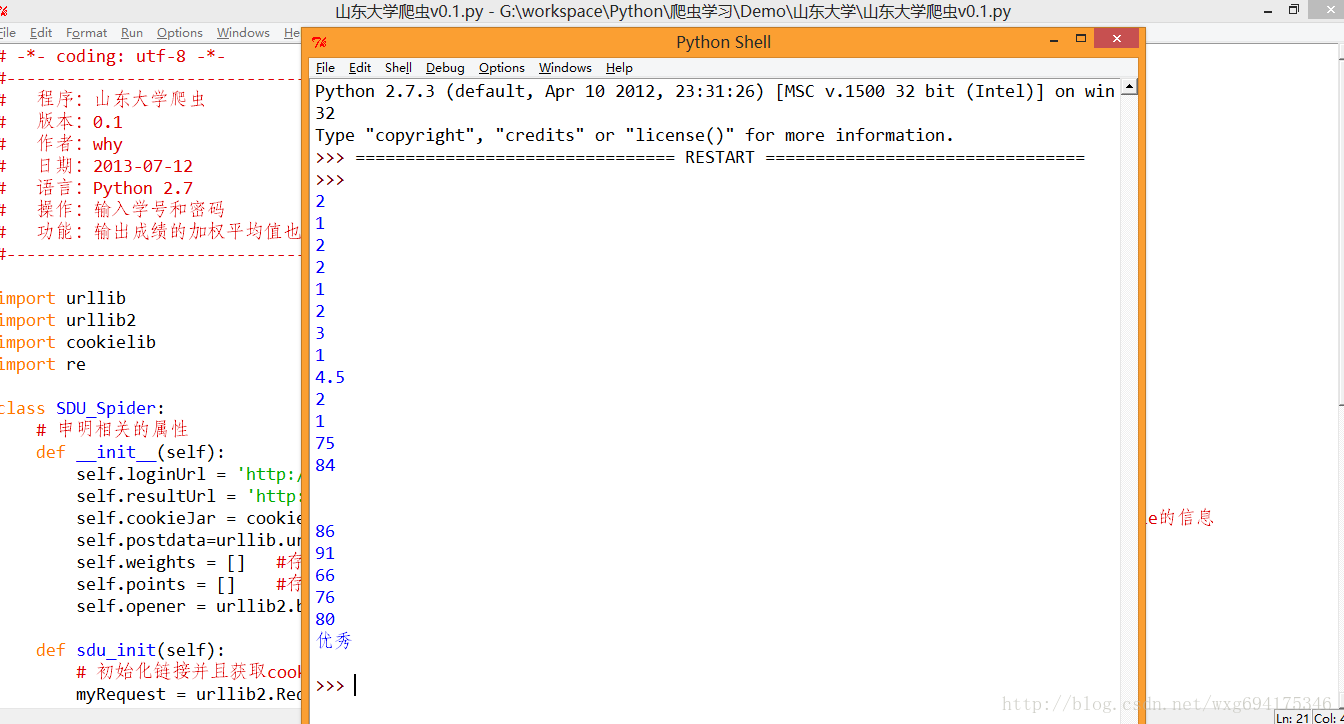
我们将代码稍稍整理一下，然后用正则来取出数据：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：山东大学爬虫
4. #   版本：0.1
5. #   作者：why
6. #   日期：2013-07-12
7. #   语言：Python 2.7
8. #   操作：输入学号和密码
9. #   功能：输出成绩的加权平均值也就是绩点
10. #---------------------------------------
12. **import** urllib
13. **import** urllib2
14. **import** cookielib
15. **import** re
17. **class** SDU\_Spider:
18. # 申明相关的属性
19. **def** \_\_init\_\_(self):
20. self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks\_login2.login'   # 登录的url
21. self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 显示成绩的url
22. self.cookieJar = cookielib.CookieJar()                                      # 初始化一个CookieJar来处理Cookie的信息
23. self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'})     # POST的数据
24. self.weights = []   #存储权重，也就是学分
25. self.points = []    #存储分数，也就是成绩
26. self.opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(self.cookieJar))
28. **def** sdu\_init(self):
29. # 初始化链接并且获取cookie
30. myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata)   # 自定义一个请求
31. result = self.opener.open(myRequest)            # 访问登录页面，获取到必须的cookie的值
32. result = self.opener.open(self.resultUrl)       # 访问成绩页面，获得成绩的数据
33. # 打印返回的内容
34. # print result.read()
35. self.deal\_data(result.read().decode('gbk'))
36. self.print\_data(self.weights);
37. self.print\_data(self.points);
39. # 将内容从页面代码中抠出来
40. **def** deal\_data(self,myPage):
41. myItems = re.findall('<TR>.\*?<p.\*?<p.\*?<p.\*?<p.\*?<p.\*?>(.\*?)</p>.\*?<p.\*?<p.\*?>(.\*?)</p>.\*?</TR>',myPage,re.S)     #获取到学分
42. **for** item **in** myItems:
43. self.weights.append(item[0].encode('gbk'))
44. self.points.append(item[1].encode('gbk'))

47. # 将内容从页面代码中抠出来
48. **def** print\_data(self,items):
49. **for** item **in** items:
50. **print** item
52. #调用
53. mySpider = SDU\_Spider()
54. mySpider.sdu\_init()

水平有限，，正则是有点丑，。运行的效果如图：



ok，接下来的只是数据的处理问题了。。

**9.凯旋而归**

完整的代码如下，至此一个完整的爬虫项目便完工了。

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/9305229" \o "copy)

1. # -\*- coding: utf-8 -\*-
2. #---------------------------------------
3. #   程序：山东大学爬虫
4. #   版本：0.1
5. #   作者：why
6. #   日期：2013-07-12
7. #   语言：Python 2.7
8. #   操作：输入学号和密码
9. #   功能：输出成绩的加权平均值也就是绩点
10. #---------------------------------------
12. **import** urllib
13. **import** urllib2
14. **import** cookielib
15. **import** re
16. **import** string

19. **class** SDU\_Spider:
20. # 申明相关的属性
21. **def** \_\_init\_\_(self):
22. self.loginUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bks\_login2.login'   # 登录的url
23. self.resultUrl = 'http://jwxt.sdu.edu.cn:7777/pls/wwwbks/bkscjcx.curscopre' # 显示成绩的url
24. self.cookieJar = cookielib.CookieJar()                                      # 初始化一个CookieJar来处理Cookie的信息
25. self.postdata=urllib.urlencode({'stuid':'201100300428','pwd':'921030'})     # POST的数据
26. self.weights = []   #存储权重，也就是学分
27. self.points = []    #存储分数，也就是成绩
28. self.opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(self.cookieJar))
30. **def** sdu\_init(self):
31. # 初始化链接并且获取cookie
32. myRequest = urllib2.Request(url = self.loginUrl,data = self.postdata)   # 自定义一个请求
33. result = self.opener.open(myRequest)            # 访问登录页面，获取到必须的cookie的值
34. result = self.opener.open(self.resultUrl)       # 访问成绩页面，获得成绩的数据
35. # 打印返回的内容
36. # print result.read()
37. self.deal\_data(result.read().decode('gbk'))
38. self.calculate\_date();
40. # 将内容从页面代码中抠出来
41. **def** deal\_data(self,myPage):
42. myItems = re.findall('<TR>.\*?<p.\*?<p.\*?<p.\*?<p.\*?<p.\*?>(.\*?)</p>.\*?<p.\*?<p.\*?>(.\*?)</p>.\*?</TR>',myPage,re.S)     #获取到学分
43. **for** item **in** myItems:
44. self.weights.append(item[0].encode('gbk'))
45. self.points.append(item[1].encode('gbk'))
47. #计算绩点，如果成绩还没出来，或者成绩是优秀良好，就不运算该成绩
48. **def** calculate\_date(self):
49. point = 0.0
50. weight = 0.0
51. **for** i **in** range(len(self.points)):
52. **if**(self.points[i].isdigit()):
53. point += string.atof(self.points[i])\*string.atof(self.weights[i])
54. weight += string.atof(self.weights[i])
55. **print** point/weight

58. #调用
59. mySpider = SDU\_Spider()
60. mySpider.sdu\_init()

### [[Python]网络爬虫（十一）：亮剑！爬虫框架小抓抓Scrapy闪亮登场！](http://blog.csdn.net/pleasecallmewhy/article/details/19354723)

前面十章爬虫笔记陆陆续续记录了一些简单的Python爬虫知识，

用来解决简单的贴吧下载，绩点运算自然不在话下。

不过要想批量下载大量的内容，比如知乎的所有的问答，那便显得游刃不有余了点。

于是乎，爬虫框架Scrapy就这样出场了！

Scrapy = Scrach+Python，Scrach这个单词是抓取的意思，

暂且可以叫它：小抓抓吧。

小抓抓的官网地址：[点我点我](http://doc.scrapy.org/en/latest/" \t "_blank)。

那么下面来简单的演示一下小抓抓Scrapy的安装流程。

具体流程参照：[官网教程](http://doc.scrapy.org/en/latest/intro/install.html" \l "intro-install-platform-notes" \t "_blank)

友情提醒：一定要按照Python的版本下载，要不然安装的时候会提醒找不到Python。建议大家安装32位是因为有些版本的必备软件64位不好找。

1.安装Python（建议32位）

建议安装Python2.7.x，3.x貌似还不支持。

安装完了记得配置环境，将python目录和python目录下的Scripts目录添加到系统环境变量的Path里。

在cmd中输入python如果出现版本信息说明配置完毕。

2.安装lxml

lxml是一种使用 Python 编写的库，可以迅速、灵活地处理 XML。点击[这里](https://pypi.python.org/pypi/lxml/3.3.1" \t "_blank)选择对应的Python版本安装。

3.安装setuptools

用来安装egg文件，点击[这里](https://pypi.python.org/packages/2.7/s/setuptools/" \t "_blank)下载python2.7的对应版本的setuptools。

4.安装zope.interface

可以使用第三步下载的setuptools来安装egg文件，现在也有exe版本，点击[这里](https://pypi.python.org/pypi/zope.interface/4.1.0" \l "downloads" \t "_blank)下载。

5.安装Twisted

Twisted是用Python实现的基于事件驱动的网络引擎框架，点击[这里](http://twistedmatrix.com/trac/wiki/Downloads" \t "_blank)下载。

6.安装pyOpenSSL

pyOpenSSL是Python的OpenSSL接口，点击[这里](https://launchpad.net/pyopenssl" \t "_blank)下载。

7.安装win32py

提供win32api，点击[这里](http://sourceforge.net/projects/pywin32/files/" \t "_blank)下载

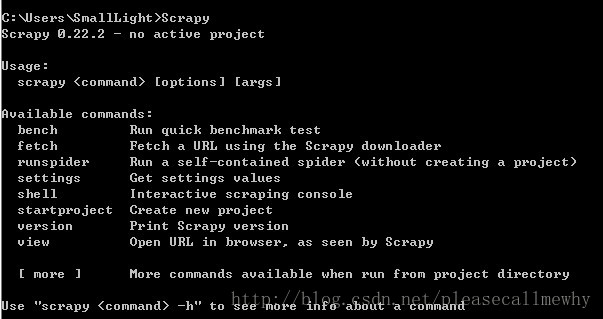
8.安装Scrapy

终于到了激动人心的时候了！安装了那么多小部件之后终于轮到主角登场。

直接在cmd中输入easy\_install scrapy回车即可。

9.检查安装

打开一个cmd窗口，在任意位置执行scrapy命令，得到下列页面，表示环境配置成功。



### [[Python]网络爬虫（十二）：爬虫框架Scrapy的第一个爬虫示例入门教程](http://blog.csdn.net/pleasecallmewhy/article/details/19642329)

（建议大家多看看官网教程：[教程地址](http://doc.scrapy.org/en/latest/intro/tutorial.html" \t "_blank)）

我们使用[dmoz.org](http://www.dmoz.org/" \t "_blank)这个网站来作为小抓抓一展身手的对象。

首先先要回答一个问题。

问：把网站装进爬虫里，总共分几步？

答案很简单，四步：

* 新建项目 (Project)：新建一个新的爬虫项目
* 明确目标（Items）：明确你想要抓取的目标
* 制作爬虫（Spider）：制作爬虫开始爬取网页
* 存储内容（Pipeline）：设计管道存储爬取内容

好的，基本流程既然确定了，那接下来就一步一步的完成就可以了。

**1.新建项目（Project）**

在空目录下按住Shift键右击，选择“在此处打开命令窗口”，输入一下命令：

**[plain]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. scrapy startproject tutorial

其中，tutorial为项目名称。

可以看到将会创建一个tutorial文件夹，目录结构如下：

**[plain]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. tutorial/
2. scrapy.cfg
3. tutorial/
4. \_\_init\_\_.py
5. items.py
6. pipelines.py
7. settings.py
8. spiders/
9. \_\_init\_\_.py
10. ...

下面来简单介绍一下各个文件的作用：

* scrapy.cfg：项目的配置文件
* tutorial/：项目的Python模块，将会从这里引用代码
* tutorial/items.py：项目的items文件
* tutorial/pipelines.py：项目的pipelines文件
* tutorial/settings.py：项目的设置文件
* tutorial/spiders/：存储爬虫的目录

**2.明确目标（Item）**

在Scrapy中，items是用来加载抓取内容的容器，有点像Python中的Dic，也就是字典，但是提供了一些额外的保护减少错误。

一般来说，item可以用scrapy.item.Item类来创建，并且用scrapy.item.Field对象来定义属性（可以理解成类似于ORM的映射关系）。

接下来，我们开始来构建item模型（model）。

首先，我们想要的内容有：

* 名称（name）
* 链接（url）
* 描述（description）

修改tutorial目录下的items.py文件，在原本的class后面添加我们自己的class。

因为要抓dmoz.org网站的内容，所以我们可以将其命名为DmozItem：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. # Define here the models for your scraped items
2. #
3. # See documentation in:
4. # http://doc.scrapy.org/en/latest/topics/items.html
6. **from** scrapy.item **import** Item, Field
8. **class** TutorialItem(Item):
9. # define the fields for your item here like:
10. # name = Field()
11. **pass**
13. **class** DmozItem(Item):
14. title = Field()
15. link = Field()
16. desc = Field()

刚开始看起来可能会有些看不懂，但是定义这些item能让你用其他组件的时候知道你的 items到底是什么。

可以把Item简单的理解成封装好的类对象。

**3.制作爬虫（Spider）**

制作爬虫，总体分两步：先爬再取。

也就是说，首先你要获取整个网页的所有内容，然后再取出其中对你有用的部分。

**3.1爬**

Spider是用户自己编写的类，用来从一个域（或域组）中抓取信息。

他们定义了用于下载的URL列表、跟踪链接的方案、解析网页内容的方式，以此来提取items。

要建立一个Spider，你必须用scrapy.spider.BaseSpider创建一个子类，并确定三个强制的属性：

* name：爬虫的识别名称，必须是唯一的，在不同的爬虫中你必须定义不同的名字。
* start\_urls：爬取的URL列表。爬虫从这里开始抓取数据，所以，第一次下载的数据将会从这些urls开始。其他子URL将会从这些起始URL中继承性生成。
* parse()：解析的方法，调用的时候传入从每一个URL传回的Response对象作为唯一参数，负责解析并匹配抓取的数据(解析为item)，跟踪更多的URL。

这里可以参考宽度爬虫教程中提及的思想来帮助理解，教程传送：[[Java] 知乎下巴第5集：使用HttpClient工具包和宽度爬虫](http://blog.csdn.net/pleasecallmewhy/article/details/18010015" \t "_blank)。

也就是把Url存储下来并依此为起点逐步扩散开去，抓取所有符合条件的网页Url存储起来继续爬取。

下面我们来写第一只爬虫，命名为dmoz\_spider.py，保存在tutorial\spiders目录下。

dmoz\_spider.py代码如下：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. **from** scrapy.spider **import** Spider
3. **class** DmozSpider(Spider):
4. name = "dmoz"
5. allowed\_domains = ["dmoz.org"]
6. start\_urls = [
7. "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
8. "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
9. ]
11. **def** parse(self, response):
12. filename = response.url.split("/")[-2]
13. open(filename, 'wb').write(response.body)

allow\_domains是搜索的域名范围，也就是爬虫的约束区域，规定爬虫只爬取这个域名下的网页。

从parse函数可以看出，将链接的最后两个地址取出作为文件名进行存储。

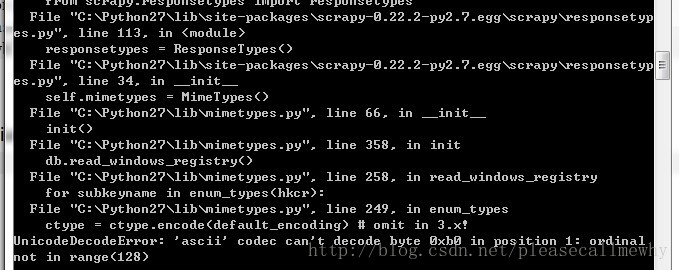
然后运行一下看看，在tutorial目录下按住shift右击，在此处打开命令窗口，输入：

**[plain]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. scrapy crawl dmoz

运行结果如图：



报错了：

UnicodeDecodeError: 'ascii' codec can't decode byte 0xb0 in position 1: ordinal not in range(128)

运行第一个Scrapy项目就报错，真是命运多舛。

应该是出了编码问题，谷歌了一下找到了解决方案：

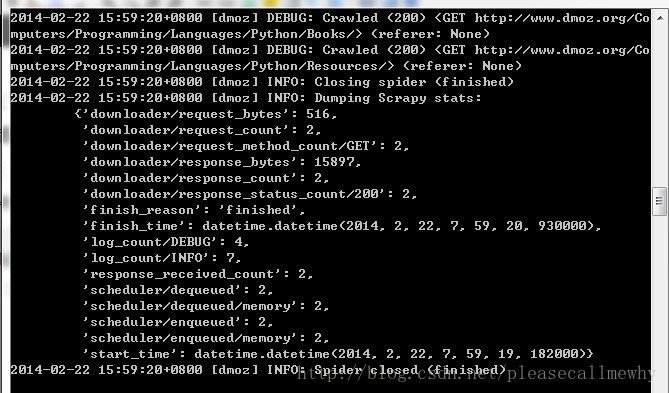
在python的Lib\site-packages文件夹下新建一个sitecustomize.py：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. **import** sys
2. sys.setdefaultencoding('gb2312')

再次运行，OK，问题解决了，看一下结果：



最后一句INFO: Closing spider (finished)表明爬虫已经成功运行并且自行关闭了。

包含 [dmoz]的行 ，那对应着我们的爬虫运行的结果。

可以看到start\_urls中定义的每个URL都有日志行。

还记得我们的start\_urls吗？

http://www.dmoz.org/Computers/Programming/Languages/Python/Books  
http://www.dmoz.org/Computers/Programming/Languages/Python/Resources

因为这些URL是起始页面，所以他们没有引用(referrers)，所以在它们的每行末尾你会看到 (referer: <None>)。

在parse 方法的作用下，两个文件被创建：分别是 Books 和 Resources，这两个文件中有URL的页面内容。

那么在刚刚的电闪雷鸣之中到底发生了什么呢？

首先，Scrapy为爬虫的 start\_urls属性中的每个URL创建了一个 scrapy.http.Request 对象 ，并将爬虫的parse 方法指定为回调函数。

然后，这些 Request被调度并执行，之后通过parse()方法返回scrapy.http.Response对象，并反馈给爬虫。

**3.2取**

爬取整个网页完毕，接下来的就是的取过程了。

光存储一整个网页还是不够用的。

在基础的爬虫里，这一步可以用正则表达式来抓。

在Scrapy里，使用一种叫做 XPath selectors的机制，它基于 XPath表达式。

如果你想了解更多selectors和其他机制你可以查阅资料：[点我点我](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "topics-selectors" \t "_blank)

这是一些XPath表达式的例子和他们的含义

* /html/head/title: 选择HTML文档<head>元素下面的<title> 标签。
* /html/head/title/text(): 选择前面提到的<title> 元素下面的文本内容
* //td: 选择所有 <td> 元素
* //div[@class="mine"]: 选择所有包含 class="mine" 属性的div 标签元素

以上只是几个使用XPath的简单例子，但是实际上XPath非常强大。

可以参照W3C教程：[点我点我](http://www.w3school.com.cn/xpath/" \t "_blank)。

为了方便使用XPaths，Scrapy提供XPathSelector 类，有两种可以选择，HtmlXPathSelector(HTML数据解析)和XmlXPathSelector(XML数据解析)。

必须通过一个 Response 对象对他们进行实例化操作。

你会发现Selector对象展示了文档的节点结构。因此，第一个实例化的selector必与根节点或者是整个目录有关 。

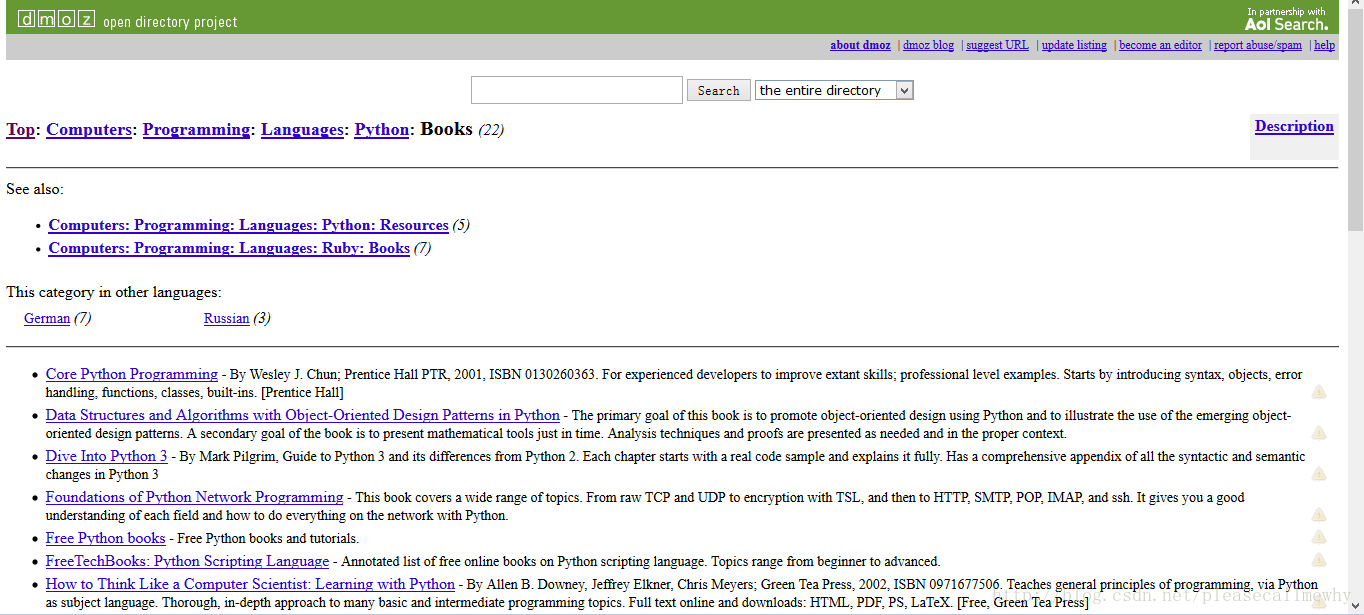
在Scrapy里面，Selectors 有四种基础的方法（点击查看API文档）：

* [xpath()](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "scrapy.selector.Selector.xpath" \t "_blank" \o "scrapy.selector.Selector.xpath)：返回一系列的selectors，每一个select表示一个xpath参数表达式选择的节点
* [css()](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "scrapy.selector.Selector.css" \t "_blank" \o "scrapy.selector.Selector.css)：返回一系列的selectors，每一个select表示一个css参数表达式选择的节点
* [extract()](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "scrapy.selector.Selector.extract" \t "_blank" \o "scrapy.selector.Selector.extract)：返回一个unicode字符串，为选中的数据
* [re()](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "scrapy.selector.Selector.re" \t "_blank" \o "scrapy.selector.Selector.re)：返回一串一个unicode字符串，为使用正则表达式抓取出来的内容

**3.3xpath实验**

下面我们在Shell里面尝试一下Selector的用法。

实验的网址：[http://www.dmoz.org/Computers/Programming/Languages/Python/Books/](http://www.dmoz.org/Computers/Programming/Languages/Python/Books/" \t "_blank)



熟悉完了实验的小白鼠，接下来就是用Shell爬取网页了。

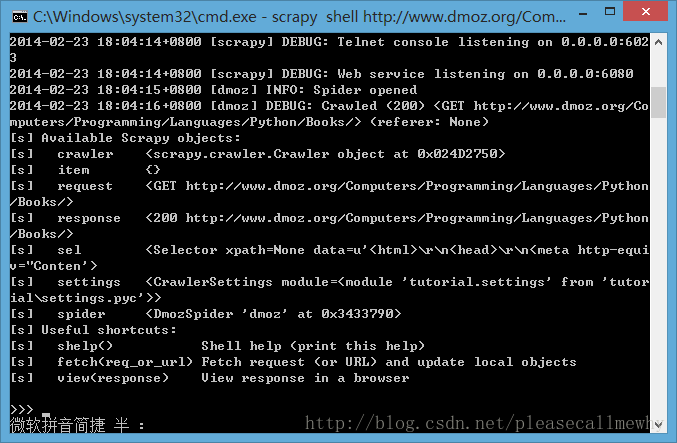
进入到项目的顶层目录，也就是第一层tutorial文件夹下，在cmd中输入：

**[plain]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. scrapy shell http://www.dmoz.org/Computers/Programming/Languages/Python/Books/

回车后可以看到如下的内容：

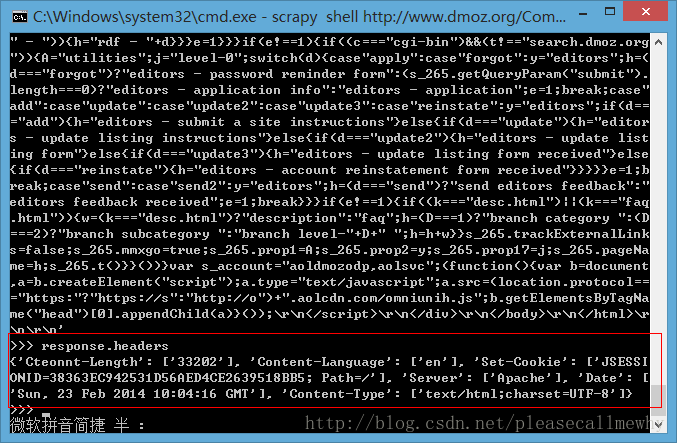


在Shell载入后，你将获得response回应，存储在本地变量 response中。

所以如果你输入response.body，你将会看到response的body部分，也就是抓取到的页面内容：



或者输入response.headers 来查看它的 header部分：



现在就像是一大堆沙子握在手里，里面藏着我们想要的金子，所以下一步，就是用筛子摇两下，把杂质出去，选出关键的内容。

selector就是这样一个筛子。

在旧的版本中，Shell实例化两种selectors，一个是解析HTML的 hxs 变量，一个是解析XML 的 xxs 变量。

而现在的Shell为我们准备好的selector对象，sel，可以根据返回的数据类型自动选择最佳的解析方案(XML or HTML)。

然后我们来捣弄一下！~

要彻底搞清楚这个问题，首先先要知道，抓到的页面到底是个什么样子。

比如，我们要抓取网页的标题，也就是<title>这个标签：



可以输入：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. sel.xpath('//title')

结果就是：



这样就能把这个标签取出来了，用extract()和text()还可以进一步做处理。

备注：简单的罗列一下有用的xpath路径表达式：

|  |  |
| --- | --- |
| **表达式** | **描述** |
| nodename | 选取此节点的所有子节点。 |
| / | 从根节点选取。 |
| // | 从匹配选择的当前节点选择文档中的节点，而不考虑它们的位置。 |
| . | 选取当前节点。 |
| .. | 选取当前节点的父节点。 |
| @ | 选取属性。 |

全部的实验结果如下，In[i]表示第i次实验的输入，Out[i]表示第i次结果的输出（建议大家参照：[W3C教程](http://www.w3school.com.cn/xpath/xpath_nodes.asp" \t "_blank)）：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. In [1]: sel.xpath('//title')
2. Out[1]: [<Selector xpath='//title' data=u'<title>Open Directory - Computers: Progr'>]
4. In [2]: sel.xpath('//title').extract()
5. Out[2]: [u'<title>Open Directory - Computers: Programming: Languages: Python: Books</title>']
7. In [3]: sel.xpath('//title/text()')
8. Out[3]: [<Selector xpath='//title/text()' data=u'Open Directory - Computers: Programming:'>]
10. In [4]: sel.xpath('//title/text()').extract()
11. Out[4]: [u'Open Directory - Computers: Programming: Languages: Python: Books']
13. In [5]: sel.xpath('//title/text()').re('(\w+):')
14. Out[5]: [u'Computers', u'Programming', u'Languages', u'Python']

当然title这个标签对我们来说没有太多的价值，下面我们就来真正抓取一些有意义的东西。

使用火狐的审查元素我们可以清楚地看到，我们需要的东西如下：



我们可以用如下代码来抓取这个<li>标签：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. sel.xpath('//ul/li')

从<li>标签中，可以这样获取网站的描述：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. sel.xpath('//ul/li/text()').extract()

可以这样获取网站的标题：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. sel.xpath('//ul/li/a/text()').extract()

可以这样获取网站的超链接：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. sel.xpath('//ul/li/a/@href').extract()

当然，前面的这些例子是直接获取属性的方法。

我们注意到xpath返回了一个对象列表，

那么我们也可以直接调用这个列表中对象的属性挖掘更深的节点

（参考：[Nesting selectors](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "topics-selectors-nesting-selectors" \t "_blank) and[Working with relative XPaths](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "topics-selectors-relative-xpaths" \t "_blank) in the [Selectors](http://doc.scrapy.org/en/latest/topics/selectors.html" \l "topics-selectors" \t "_blank)）：

sites = sel.xpath('//ul/li')

for site in sites:

title = site.xpath('a/text()').extract()

link = site.xpath('a/@href').extract()

desc = site.xpath('text()').extract()

print title, link, desc

**3.4xpath实战**

我们用shell做了这么久的实战，最后我们可以把前面学习到的内容应用到dmoz\_spider这个爬虫中。

在原爬虫的parse函数中做如下修改：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



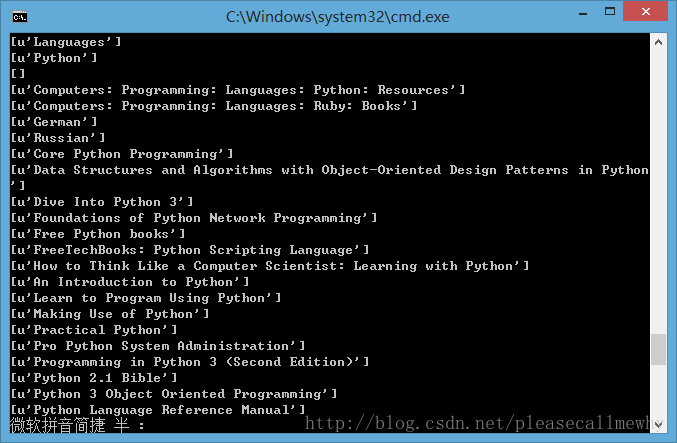
1. **from** scrapy.spider **import** Spider
2. **from** scrapy.selector **import** Selector
4. **class** DmozSpider(Spider):
5. name = "dmoz"
6. allowed\_domains = ["dmoz.org"]
7. start\_urls = [
8. "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
9. "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
10. ]
12. **def** parse(self, response):
13. sel = Selector(response)
14. sites = sel.xpath('//ul/li')
15. **for** site **in** sites:
16. title = site.xpath('a/text()').extract()
17. link = site.xpath('a/@href').extract()
18. desc = site.xpath('text()').extract()
19. **print** title

注意，我们从scrapy.selector中导入了Selector类，并且实例化了一个新的Selector对象。这样我们就可以像Shell中一样操作xpath了。

我们来试着输入一下命令运行爬虫（在tutorial根目录里面）：

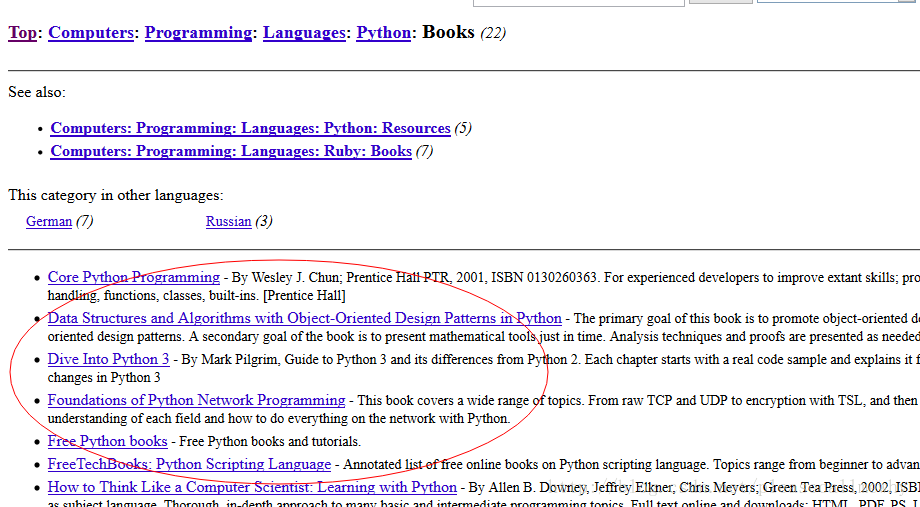
scrapy crawl dmoz

运行结果如下：



果然，成功的抓到了所有的标题。但是好像不太对啊，怎么Top，Python这种导航栏也抓取出来了呢？

我们只需要红圈中的内容：



看来是我们的xpath语句有点问题，没有仅仅把我们需要的项目名称抓取出来，也抓了一些无辜的但是xpath语法相同的元素。

审查元素我们发现我们需要的<ul>具有class='directory-url'的属性，

那么只要把xpath语句改成sel.xpath('//ul[@class="directory-url"]/li')即可

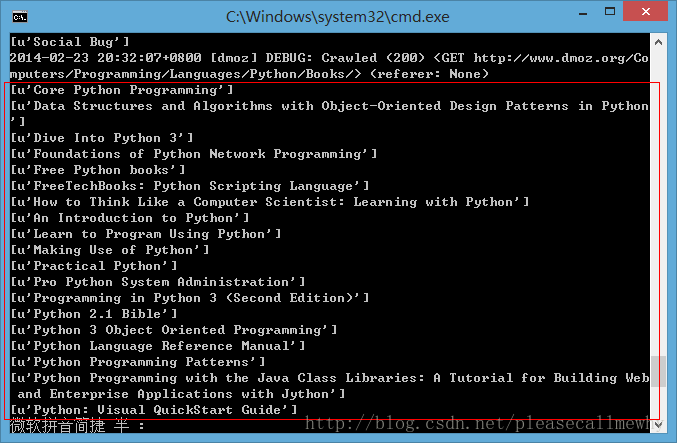
将xpath语句做如下调整：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. **from** scrapy.spider **import** Spider
2. **from** scrapy.selector **import** Selector
4. **class** DmozSpider(Spider):
5. name = "dmoz"
6. allowed\_domains = ["dmoz.org"]
7. start\_urls = [
8. "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
9. "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
10. ]
12. **def** parse(self, response):
13. sel = Selector(response)
14. sites = sel.xpath('//ul[@class="directory-url"]/li')
15. **for** site **in** sites:
16. title = site.xpath('a/text()').extract()
17. link = site.xpath('a/@href').extract()
18. desc = site.xpath('text()').extract()
19. **print** title

成功抓出了所有的标题，绝对没有滥杀无辜：



**3.5使用Item**

接下来我们来看一看如何使用Item。

前面我们说过，Item 对象是自定义的python字典，可以使用标准字典语法获取某个属性的值：

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. >>> item = DmozItem()
2. >>> item['title'] = 'Example title'
3. >>> item['title']
4. 'Example title'

作为一只爬虫，Spiders希望能将其抓取的数据存放到Item对象中。为了返回我们抓取数据，spider的最终代码应当是这样:

**[python]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. **from** scrapy.spider **import** Spider
2. **from** scrapy.selector **import** Selector
4. **from** tutorial.items **import** DmozItem
6. **class** DmozSpider(Spider):
7. name = "dmoz"
8. allowed\_domains = ["dmoz.org"]
9. start\_urls = [
10. "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",
11. "http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"
12. ]
14. **def** parse(self, response):
15. sel = Selector(response)
16. sites = sel.xpath('//ul[@class="directory-url"]/li')
17. items = []
18. **for** site **in** sites:
19. item = DmozItem()
20. item['title'] = site.xpath('a/text()').extract()
21. item['link'] = site.xpath('a/@href').extract()
22. item['desc'] = site.xpath('text()').extract()
23. items.append(item)
24. **return** items

**4.存储内容（Pipeline）**

保存信息的最简单的方法是通过[Feed exports](http://doc.scrapy.org/en/0.14/topics/feed-exports.html" \l "topics-feed-exports" \t "_blank)，主要有四种：JSON，JSON lines，CSV，XML。

我们将结果用最常用的JSON导出，命令如下：

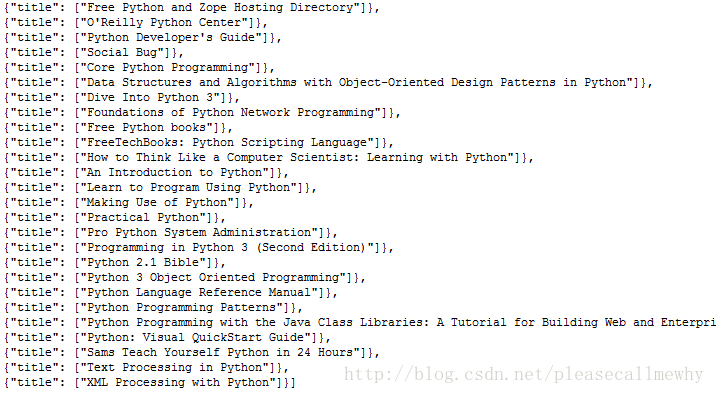
**[plain]** [view plain](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "view plain)[copy](http://blog.csdn.net/pleasecallmewhy/article/details/19642329" \o "copy)



1. scrapy crawl dmoz -o items.json -t json

-o 后面是导出文件名，-t 后面是导出类型。

然后来看一下导出的结果，用文本编辑器打开json文件即可（为了方便显示，在item中删去了除了title之外的属性）：



因为这个只是一个小型的例子，所以这样简单的处理就可以了。

如果你想用抓取的items做更复杂的事情，你可以写一个 Item Pipeline(条目管道)。

这个我们以后再慢慢玩^\_^

### [[Python]项目打包：5步将py文件打包成exe文件](http://blog.csdn.net/pleasecallmewhy/article/details/8935135)

分类： [Python](http://blog.csdn.net/wxg694175346/article/category/1344887)2013-05-16 14:54 14570人阅读 [评论](http://blog.csdn.net/pleasecallmewhy/article/details/8935135" \l "comments)(26) [收藏](javascript:void(0);" \o "收藏) [举报](http://blog.csdn.net/pleasecallmewhy/article/details/8935135" \l "report" \o "举报)

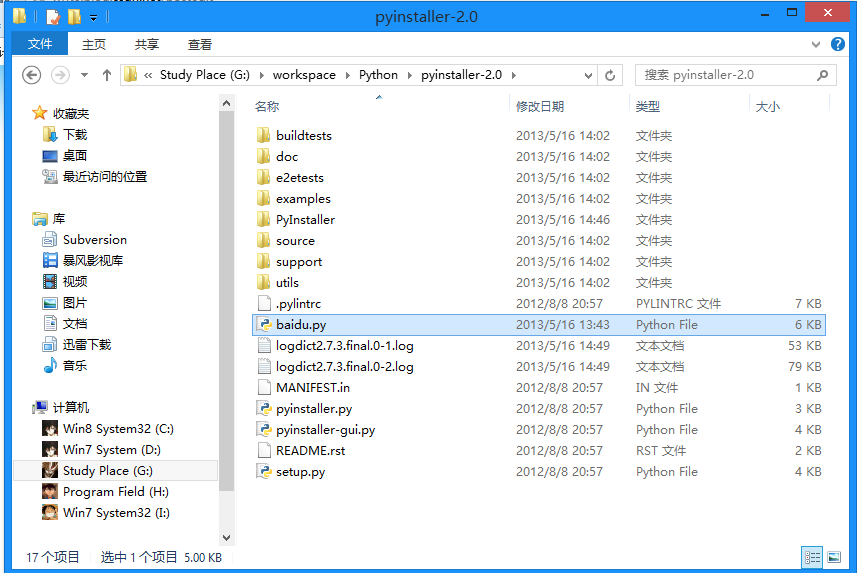
1.下载pyinstaller并解压（可以去官网下载最新版）：

[http://nchc.dl.sourceforge.net/project/pyinstaller/2.0/pyinstaller-2.0.zip](http://nchc.dl.sourceforge.net/project/pyinstaller/2.0/pyinstaller-2.0.zip" \t "_blank)

2.下载pywin32并安装（注意版本，我的是python2.7）：

[http://download.csdn.net/download/lanlandechong/4367925](http://download.csdn.net/download/lanlandechong/4367925" \t "_blank)

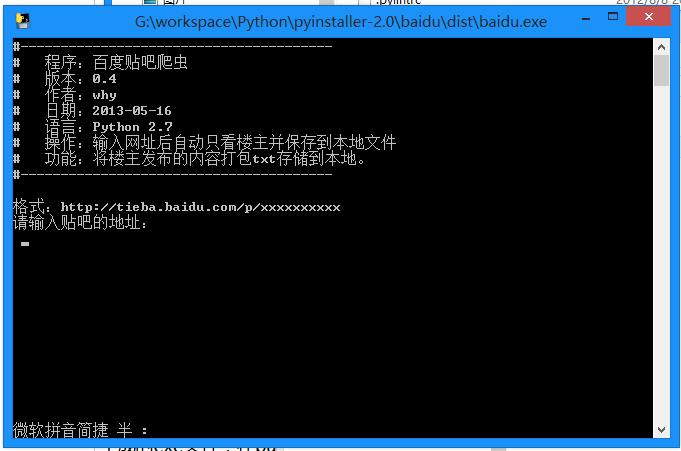
3.将项目文件放到pyinstaller文件夹下面（我的是baidu.py）：



4.按住shift键右击，在当前路径打开命令提示行，输入以下内容（最后的是文件名）：

python pyinstaller.py -F baidu.py

5.生成的exe文件，在baidu文件夹下的dist文件夹中，双击即可运行：



## 实验楼：基于scrapy的天气数据采集

## 一、实验说明

### 1. 环境登录

无需密码自动登录，系统用户名shiyanlou，密码shiyanlou

### 2. 环境介绍

本实验环境采用带桌面的Ubuntu Linux环境，实验中会用到桌面上的程序：

1. LX终端（LXTerminal）: Linux命令行终端，打开后会进入Bash环境，可以使用Linux命令
2. Scrapy：Scrapy是一个为了爬取网站数据，提取结构性数据而用Python编写的应用框架。 可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。
3. GVim：非常好用的编辑器，最简单的用法可以参考课程[Vim编辑器](http://www.shiyanlou.com/courses/2" \t "_blank)

### 3. 环境使用

使用GVim编辑器输入实验所需的代码及文件，使用LX终端（LXTerminal）运行所需命令进行操作。

完成实验后可以点击桌面上方的“实验截图”保存并分享实验结果到微博，向好友展示自己的学习进度。实验楼提供后台系统截图，可以真实有效证明您已经完成了实验。

实验记录页面可以在“我的主页”中查看，其中含有每次实验的截图及笔记，以及每次实验的有效学习时间（指的是在实验桌面内操作的时间，如果没有操作，系统会记录为发呆时间）。这些都是您学习的真实性证明。

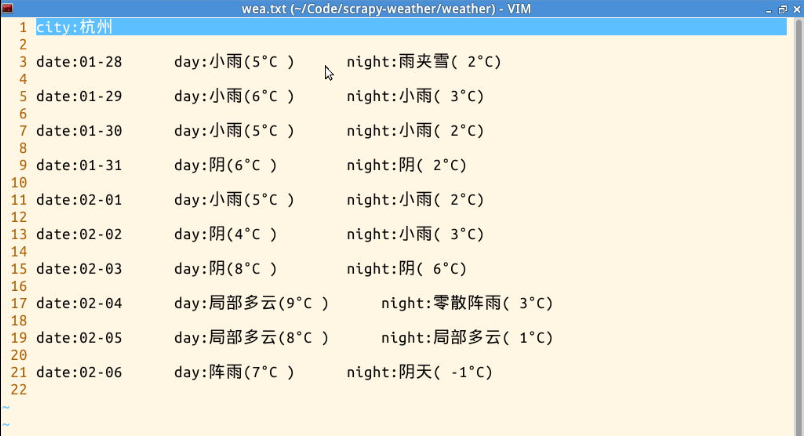
本课程中的所有源码可以通过以下方式下载:

http://git.shiyanlou.com/shiyanlou/scrapy-weather

## 二、项目介绍

很多网站都提供了浏览者本地的天气信息，这些信息是如何获取到的呢，方法有很多种，大多是利用某些网站提供的天气api获取的，也有利用爬虫采集的。本项目就介绍如何用Scrapy来采集天气信息（从新浪天气频道采集: [http://weather.sina.com.cn/](http://weather.sina.com.cn/" \t "_blank) ），学完本项目后大家能对Scrapy有个初步的认识，能进行简单的采集需求开发。

先上一张最终的效果截图：



大家看了截图，是不是马上就想动手完成它，下面我们就来开始。

## 三、项目实战

### 1. 安装Scrapy

首先把Scrapy签名的GPG密钥添加到APT的钥匙环中：

$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 627220E7

然后执行如下命令，创建 /etc/apt/sources.list.d/scrapy.list 文件：

$ echo 'deb http://archive.scrapy.org/ubuntu scrapy main' | sudo tee /etc/apt/sources.list.d/scrapy.list

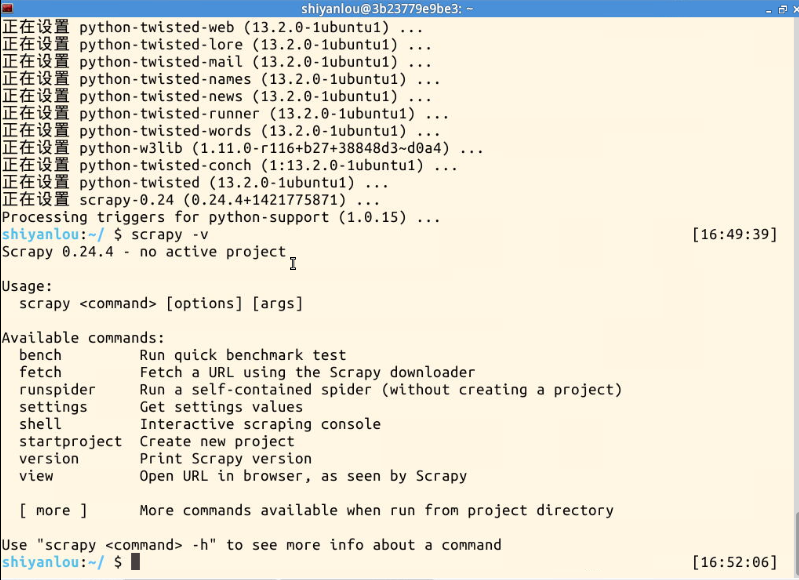
最后更新包列表并安装 scrapy-0.24：

$ sudo apt-get update && sudo apt-get install scrapy-0.24

完成这步后，可以用下面的命令测试一下安装是否正确：

$ scrapy -v

如果正常，效果如图所示：

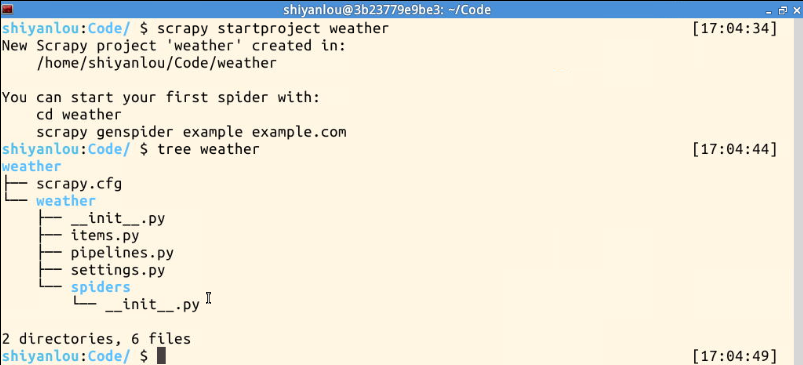


### 2. 创建项目

在开始爬取之前，必须创建一个新的Scrapy项目。进入您打算存储代码的目录中，运行下列命令：

$ scrapy startproject weather

如果正常，效果如图所示：



这些文件分别是:

* scrapy.cfg: 项目的配置文件
* weather/: 该项目的python模块。之后将在此加入代码。
* weather/items.py: 项目中的item文件.
* weather/pipelines.py: 项目中的pipelines文件.
* weather/settings.py: 项目的设置文件.
* weather/spiders/: 放置spider代码的目录.

### 3. 定义Item

Item 是保存爬取到的数据的容器；其使用方法和python字典类似，并且提供了额外保护机制来避免拼写错误导致的未定义字段错误。

首先根据需要从weather.sina.com.cn获取到的数据对item进行建模。 我们需要从weather.sina.com.cn中获取当前城市名，后续9天的日期，天气描述和温度等信息。对此，在item中定义相应的字段。编辑 weather 目录中的 items.py 文件:

# -\*- coding: utf-8 -\*-

# Define here the models for your scraped items

#

# See documentation in:

# http://doc.scrapy.org/en/latest/topics/items.html

import scrapy

class WeatherItem(scrapy.Item):

# define the fields for your item here like:

# name = scrapy.Field()

# demo 1

city = scrapy.Field()

date = scrapy.Field()

dayDesc = scrapy.Field()

dayTemp = scrapy.Field()

pass

### 4. 编写获取天气数据的爬虫(Spider)

Spider是用户编写用于从单个网站(或者一些网站)爬取数据的类。

其包含了一个用于下载的初始URL，如何跟进网页中的链接以及如何分析页面中的内容， 提取生成 item 的方法。

为了创建一个Spider，必须继承 scrapy.Spider 类， 且定义以下三个属性:

* name: 用于区别Spider。该名字必须是唯一的，您不可以为不同的Spider设定相同的名字。
* start\_urls: 包含了Spider在启动时进行爬取的url列表。因此，第一个被获取到的页面将是其中之一。后续的URL则从初始的URL获取到的数据中提取。
* parse() 是spider的一个方法。 被调用时，每个初始URL完成下载后生成的 Response 对象将会作为唯一的参数传递给该函数。 该方法负责解析返回的数据(response data)，提取数据(生成item)以及生成需要进一步处理的URL的 Request 对象。

我们通过浏览器的查看源码工具先来分析一下需要获取的数据网源代码：

<h4 class="slider\_ct\_name" id="slider\_ct\_name">武汉</h4>

...

<div class="blk\_fc\_c0\_scroll" id="blk\_fc\_c0\_scroll" style="width: 1700px;">

<div class="blk\_fc\_c0\_i">

<p class="wt\_fc\_c0\_i\_date">01-28</p>

<p class="wt\_fc\_c0\_i\_day wt\_fc\_c0\_i\_today">今天</p>

<p class="wt\_fc\_c0\_i\_icons clearfix">

<img class="icons0\_wt png24" src="http://www.sinaimg.cn/dy/weather/main/index14/007/icons\_42\_yl/w\_04\_27\_00.png" alt="雨夹雪" title="雨夹雪">

<img class="icons0\_wt png24" src="http://www.sinaimg.cn/dy/weather/main/index14/007/icons\_42\_yl/w\_04\_29\_01.png" alt="中雪" title="中雪">

</p>

<p class="wt\_fc\_c0\_i\_times">

<span class="wt\_fc\_c0\_i\_time">白天</span>

<span class="wt\_fc\_c0\_i\_time">夜间</span>

</p>

<p class="wt\_fc\_c0\_i\_temp">1°C / -2°C</p>

<p class="wt\_fc\_c0\_i\_tip">北风 3～4级</p>

<p class="wt\_fc\_c0\_i\_tip">无持续风向 小于3级</p>

</div>

<div class="blk\_fc\_c0\_i">

<p class="wt\_fc\_c0\_i\_date">01-29</p>

<p class="wt\_fc\_c0\_i\_day ">星期四</p>

<p class="wt\_fc\_c0\_i\_icons clearfix">

<img class="icons0\_wt png24" src="http://www.sinaimg.cn/dy/weather/main/index14/007/icons\_42\_yl/w\_04\_29\_00.png" alt="中雪" title="中雪">

<img class="icons0\_wt png24" src="http://www.sinaimg.cn/dy/weather/main/index14/007/icons\_42\_yl/w\_07\_25\_01.png" alt="阴" title="阴">

</p>

<p class="wt\_fc\_c0\_i\_times">

<span class="wt\_fc\_c0\_i\_time">白天</span>

<span class="wt\_fc\_c0\_i\_time">夜间</span>

</p>

<p class="wt\_fc\_c0\_i\_temp">1°C / -2°C</p>

<p class="wt\_fc\_c0\_i\_tip">无持续风向 小于3级</p>

</div>

...

</div>

我们可以看到：

* 城市名可以通过获取id为slider\_ct\_name的h4元素获取
* 日期可以通过获取id为blk\_fc\_c0\_scroll下的class为wt\_fc\_c0\_i\_date的p元素获取
* 天气描述可以通过获取id为blk\_fc\_c0\_scroll下的class为icons0\_wt的img元素获取
* 温度可以通过获取id为blk\_fc\_c0\_scroll下的class为wt\_fc\_c0\_i\_temp的p元素获取

因此，我们的Spider代码如下，保存在 weather/spiders 目录下的 localweather.py 文件中:

# -\*- coding: utf-8 -\*-

import scrapy

from weather.items import WeatherItem

class WeatherSpider(scrapy.Spider):

name = "myweather"

allowed\_domains = ["sina.com.cn"]

start\_urls = ['http://weather.sina.com.cn']

def parse(self, response):

item = WeatherItem()

item['city'] = response.xpath('//\*[@id="slider\_ct\_name"]/text()').extract()

tenDay = response.xpath('//\*[@id="blk\_fc\_c0\_scroll"]');

item['date'] = tenDay.css('p.wt\_fc\_c0\_i\_date::text').extract()

item['dayDesc'] = tenDay.css('img.icons0\_wt::attr(title)').extract()

item['dayTemp'] = tenDay.css('p.wt\_fc\_c0\_i\_temp::text').extract()

return item

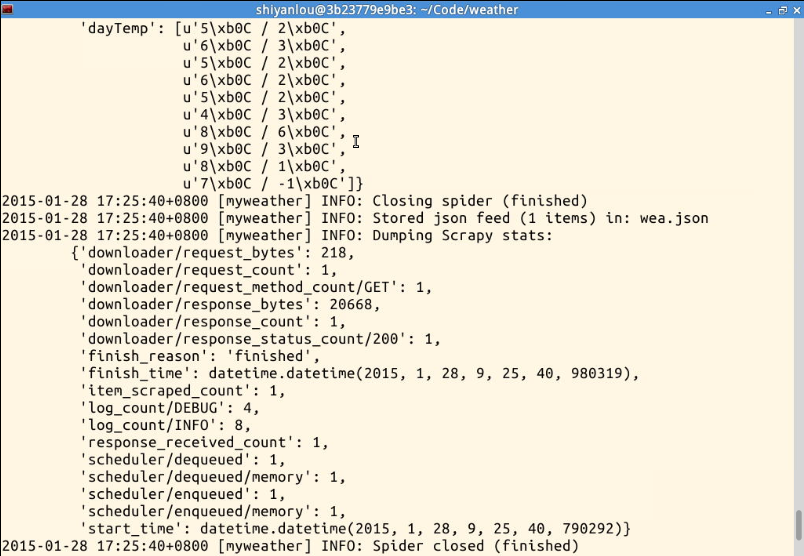
代码中的xpath和css后面括号的内容为选择器，关于xpath和css选择器的内容可参考官方教程：[http://doc.scrapy.org/en/0.24/topics/selectors.html](http://doc.scrapy.org/en/0.24/topics/selectors.html" \t "_blank)

### 5. 运行爬虫，对数据进行验证

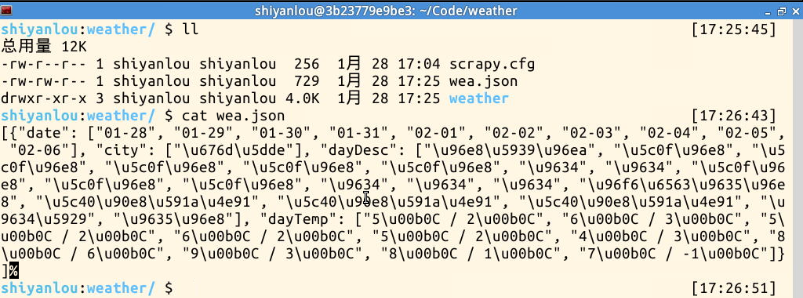
到这里为止，我们需要验证一下爬虫是否能正常工作（即能否取到我们想要的数据），验证的方法就是在命令行（**重要：在项目的scrapy.cfg文件同级目录运行命令，下同**）中运行下面的代码：

$ scrapy crawl myweather -o wea.json

这行命令的意思是，运行名字为 myweather 的爬虫（我们在上一步中定义的），然后把结果以json格式保存在wea.json文件中。命令运行结果如下：



然后，我们查看当前目录下的wea.json文件，正常情况下效果如下：



我们看到，wea.json中已经有数据了，只是数据是以unicode方式编码的。

### 6. 保存爬取到的数据

上面只是把数据保存在json文件中了，如果我们想自己保存在文件或数据库中，如何操作呢？

这里就要用到 Item Pipeline 了，那么 Item Pipeline 是什么呢？

当Item在Spider中被收集之后，它将会被传递到Item Pipeline中，一些组件会按照一定的顺序执行对Item的处理。

每个item pipeline组件(有时称之为“Item Pipeline”)是实现了简单方法的Python类。他们接收到Item并通过它执行一些行为，同时也决定此Item是否继续通过pipeline，或是被丢弃而不再进行处理。

item pipeline的典型应用有：

* 清理HTML数据
* 验证爬取的数据(检查item包含某些字段)
* 查重(并丢弃)
* 将爬取结果保存到文件或数据库中

每个item pipeline组件都需要调用 process\_item 方法，这个方法必须返回一个 Item (或任何继承类)对象， 或是抛出 DropItem异常，被丢弃的item将不会被之后的pipeline组件所处理。

我们这里把数据转码后保存在 wea.txt 文本中。

pipelines.py文件在创建项目时已经自动被创建好了，我们在其中加上保存到文件的代码：

# -\*- coding: utf-8 -\*-

# Define your item pipelines here

#

# Don't forget to add your pipeline to the ITEM\_PIPELINES setting

# See: http://doc.scrapy.org/en/latest/topics/item-pipeline.html

class WeatherPipeline(object):

def \_\_init\_\_(self):

self.file = open('wea.txt', 'w+')

def process\_item(self, item, spider):

city = item['city'][0].encode('utf-8')

self.file.write('city:' + str(city) + '\n\n')

date = item['date']

desc = item['dayDesc']

dayDesc = desc[1::2]

nightDesc = desc[0::2]

dayTemp = item['dayTemp']

weaitem = zip(date, dayDesc, nightDesc, dayTemp)

for i in range(len(weaitem)):

item = weaitem[i]

d = item[0]

dd = item[1]

nd = item[2]

ta = item[3].split('/')

dt = ta[0]

nt = ta[1]

txt = 'date:{0}\t\tday:{1}({2})\t\tnight:{3}({4})\n\n'.format(

d,

dd.encode('utf-8'),

dt.encode('utf-8'),

nd.encode('utf-8'),

nt.encode('utf-8')

)

self.file.write(txt)

return item

代码比较简单，都是python比较基础的语法，如果您感觉比较吃力，建议先去学一下python基础课。

### 7. 把 ITEM\_PIPELINES 添加到设置中

写好ITEM\_PIPELINES后，还有很重要的一步，就是把 ITEM\_PIPELINES 添加到设置文件 settings.py 中。

ITEM\_PIPELINES = {

'weather.pipelines.WeatherPipeline': 1

}

另外，有些网站对网络爬虫进行了阻止（注：本项目仅从技术角度处理此问题，个人强烈不建议您用爬虫爬取有版权信息的数据），我们可以在设置中修改一下爬虫的 USER\_AGENT 和 Referer 信息，增加爬虫请求的时间间隔。

整个 settings.py 文件内容如下：

# -\*- coding: utf-8 -\*-

# Scrapy settings for weather project

#

# For simplicity, this file contains only the most important settings by

# default. All the other settings are documented here:

#

# http://doc.scrapy.org/en/latest/topics/settings.html

#

BOT\_NAME = 'Googlebot'

SPIDER\_MODULES = ['weather.spiders']

NEWSPIDER\_MODULE = 'weather.spiders'

# Crawl responsibly by identifying yourself (and your website) on the user-agent

#USER\_AGENT = 'weather (+http://www.yourdomain.com)'

USER\_AGENT = 'User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.95 Safari/537.36'

DEFAULT\_REQUEST\_HEADERS = {

'Referer': 'http://www.weibo.com'

}

ITEM\_PIPELINES = {

'weather.pipelines.WeatherPipeline': 1

}

DOWNLOAD\_DELAY = 0.5

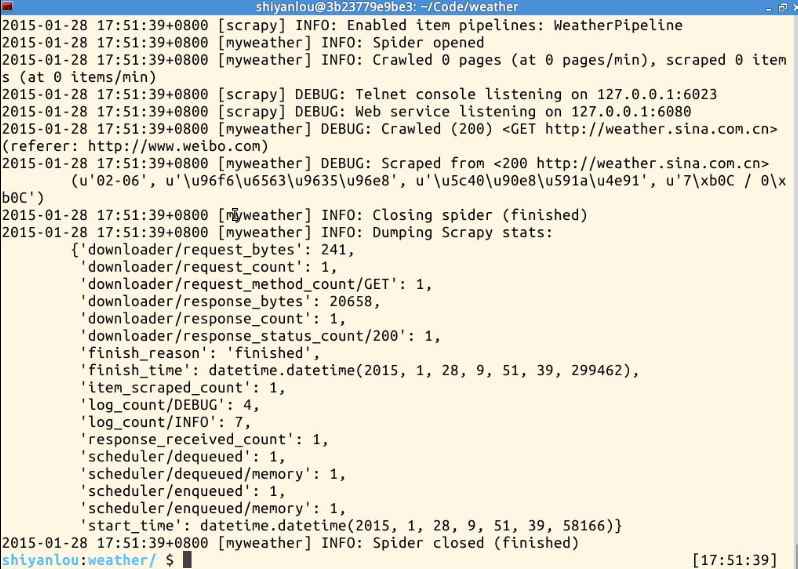
到现在为止，代码主要部分已经写完了。

### 8. 运行爬虫

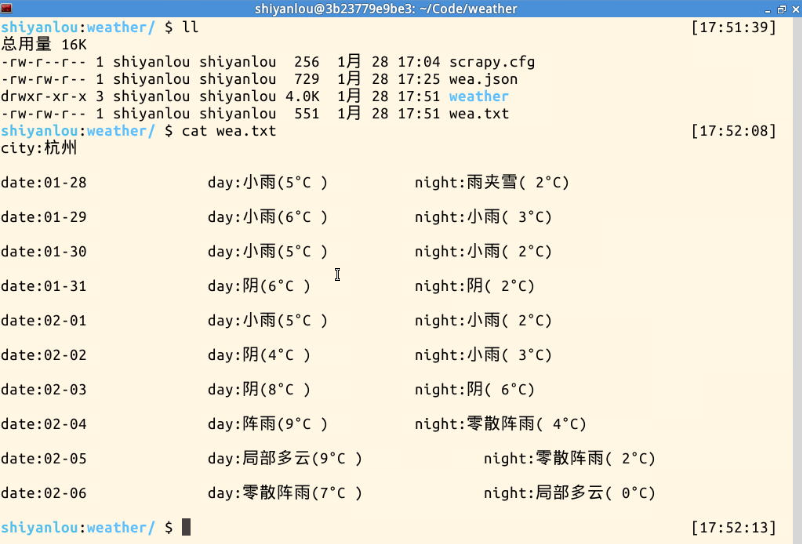
在**项目的scrapy.cfg同级目录下**用下面的命令运行爬虫：

$ scrapy crawl myweather

正常情况下，效果如下：



然后，在当前目录下会多一个 wea.txt 文件，内容如下：



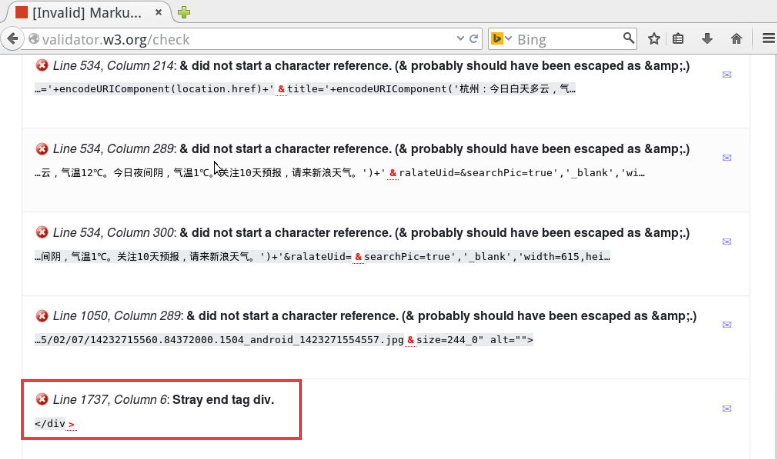
到此我们基于scrapy的天气数据采集就完成了。

## 四、答疑

### 1、关于结果只出现城市的问题

最近看到有同学反馈代码按课程运行后，最后的数据中只有城市数据，没有天气数据，我检查了一下代码，找到了问题存在的原因。

scrapy内置的html解析是基于lxml库的，这个库对html的解析的容错性不是很好，通过检查虚拟机中获取到的网页源码，发现有部分标签是不匹配的（地区和浏览器不同取到的源码可能不同），检查结果如图：



所以导致在spider中取到的日期数据(item['date'])为空，然后在pilepine代码中做zip操作后，整个 weaitem 为空，所以最终只有城市数据了。

既然找到了原因，我们换个html代码解析器就可以了，这里建议用 BeautifulSoup （官网：[http://www.crummy.com/software/BeautifulSoup/bs4/doc/index.html](http://www.crummy.com/software/BeautifulSoup/bs4/doc/index.html" \t "_blank) ），这个解析器有比较好的容错能力，具体用法可以参考上面的文档。

BeautifulSoup安装：

#下载BeautifulSoup

$ wget http://labfile.oss.aliyuncs.com/beautifulsoup4-4.3.2.tar.gz

#解压

$ tar -zxvf beautifulsoup4-4.3.2.tar.gz

#安装

$ cd beautifulsoup4-4.3.2

$ sudo python setup.py install

安装成功后，优化WeatherSpider代码，改进后的代码如下：

# -\*- coding: utf-8 -\*-

import scrapy

from bs4 import BeautifulSoup

from weather.items import WeatherItem

class WeatherSpider(scrapy.Spider):

name = "myweather"

allowed\_domains = ["sina.com.cn"]

start\_urls = ['http://weather.sina.com.cn']

def parse(self, response):

html\_doc = response.body

#html\_doc = html\_doc.decode('utf-8')

soup = BeautifulSoup(html\_doc)

itemTemp = {}

itemTemp['city'] = soup.find(id='slider\_ct\_name')

tenDay = soup.find(id='blk\_fc\_c0\_scroll')

itemTemp['date'] = tenDay.findAll("p", {"class": 'wt\_fc\_c0\_i\_date'})

itemTemp['dayDesc'] = tenDay.findAll("img", {"class": 'icons0\_wt'})

itemTemp['dayTemp'] = tenDay.findAll('p', {"class": 'wt\_fc\_c0\_i\_temp'})

item = WeatherItem()

for att in itemTemp:

item[att] = []

if att == 'city':

item[att] = itemTemp.get(att).text

continue

for obj in itemTemp.get(att):

if att == 'dayDesc':

item[att].append(obj['title'])

else:

item[att].append(obj.text)

return item

然后再次运行爬虫:

$ scrapy crawl myweather

然后查看 wea.txt，数据如下：



### 2、 关于只取到了9天的数据问题

如果是晚上运行爬虫，当天的白天天气是没有的（已经过去了），针对这部分建议自己优化。