

Python爬虫入门（4）：Urllib库的高级用法

本文作者：[伯乐在线 - 崔庆才](#)。未经作者许可，禁止转载！

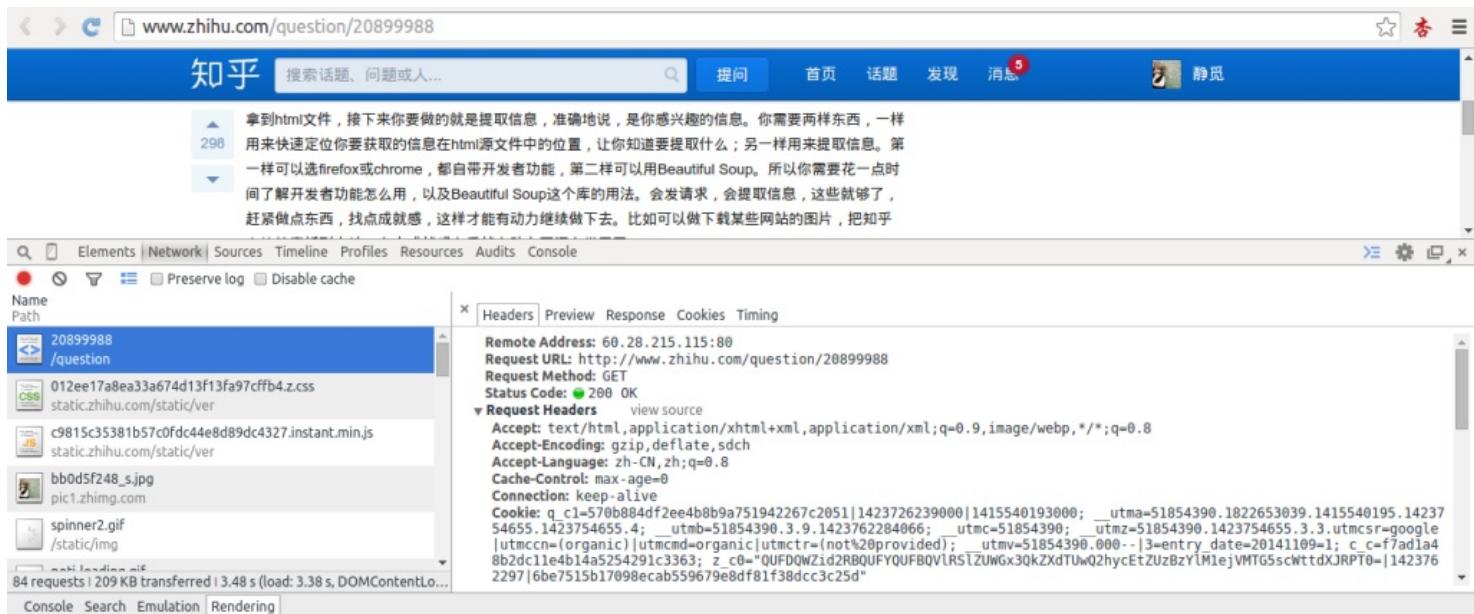
欢迎加入伯乐在线[专栏作者](#)。

- [Python爬虫入门（1）：综述](#)
- [Python爬虫入门（2）：爬虫基础了解](#)
- [Python爬虫入门（3）：Urllib库的基本使用](#)
- [Python爬虫入门（4）：Urllib库的高级用法](#)
- [Python爬虫入门（5）：URLError异常处理](#)
- [Python爬虫入门（6）：Cookie的使用](#)
- [Python爬虫入门（7）：正则表达式](#)
- [Python爬虫入门（8）：Beautiful Soup的用法](#)

1. 设置Headers

有些网站不会同意程序直接用上面的方式进行访问，如果识别有问题，那么站点根本不会响应，所以为了完全模拟浏览器的工作，我们需要设置一些Headers的属性。

首先，打开我们的浏览器，调试浏览器F12，我用的是Chrome，打开网络监听，示意如下，比如知乎，点登录之后，我们会发现登陆之后界面都变化了，出现一个新的界面，实质上这个页面包含了许许多多的内容，这些内容也不是一次性就加载完成的，实质上是执行了好多次请求，一般是首先请求HTML文件，然后加载JS，CSS等等，经过多次请求之后，网页的骨架和肌肉全了，整个网页的效果也就出来了。



拆分这些请求，我们只看第一个请求，你可以看到，有个Request URL，还有headers，下面便是response，图片显示得不全，小伙伴们可以亲身实验一下。那么这个头中包含了许许多多信息，有文件编码啦，压缩方式啦，请求的agent啦等等。

其中，agent就是请求的身份，如果没有写入请求身份，那么服务器不一定会响应，所以可以在headers中设置agent，例如下面的例子，这个例子只是说明了怎样设置的headers，小伙伴们看一下设置格式就好。

Python

```
import urllib
1 import urllib
2 import urllib2
3
4 url = 'http://www.server.com/login'
5 user_agent = 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)'
6 values = {'username' : 'cqq', 'password' : 'XXXX'}
7 headers = { 'User-Agent' : user_agent }
8 data = urllib.urlencode(values)
9 request = urllib2.Request(url, data, headers)
10 response = urllib2.urlopen(request)
11 page = response.read()
```

这样，我们设置了一个headers，在构建request时传入，在请求时，就加入了headers传送，服务器若识别了是浏览器发来的请求，就会得到响应。

另外，我们还有对付”反盗链”的方式，对付防盗链，服务器会识别headers中的referer是不是它自己，如果不是，有的服务器不会响应，所以我们还可以在headers中加入referer

例如我们可以构建下面的headers

Python

```
headers = { 'User-Agent':  
           ': Mozilla/4.0'  
           }  
1 headers = { 'User-Agent' : 'Mozilla/4.0 (compatible; MSIE 5.5; Windows NT)' ,  
2             'Referer': 'http://www.zhihu.com/articles' }
```

同上面的方法，在传送请求时把headers传入Request参数里，这样就能应付防盗链了。

另外headers的一些属性，下面的需要特别注意一下：

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。

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

其他的有必要的可以审查浏览器的headers内容，在构建时写入同样的数据即可。

2. Proxy（代理）的设置

urllib2默认会使用环境变量http_proxy来设置HTTP Proxy。假如一个网站它会检测某一段时间某个IP的访问次数，如果访问次数过多，它会禁止你的访问。所以你可以设置一些代理服务器来帮助你做工作，每隔一段时间换一个代理，网站君都不知道是谁在捣鬼了，这酸爽！

下面一段代码说明了代理的设置用法

Python

```
import urllib2  
#  
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)
```

3. Timeout设置

上一节已经说过urlopen方法了，第三个参数就是timeout的设置，可以设置等待多久超时，为了解决一些网站实在响应过慢而造成的影响。

例如下面的代码，如果第二个参数data为空那么要特别指定是timeout是多少，写明形参，如果data已经传入，则不必声明。

Python

```
import urllib2  
response =  
#  
1 import urllib2  
2 response = urllib2.urlopen('http://www.baidu.com', timeout=10)
```

Python

```
import urllib2  
response =  
#  
1 import urllib2  
2 response = urllib2.urlopen('http://www.baidu.com', data, 10)
```

4. 使用HTTP的PUT和DELETE方法

http协议有六种请求方法，get,head,put,delete,post,options，我们有时候需要用到PUT方式或者DELETE方式请求。

PUT：这个方法比较少见。HTML表单也不支持这个。本质上讲，PUT和POST极为相似，都是向服务器发送数据，但它们之间有一个重要区别，PUT通常指定了资源的存放位置，而POST则没有，POST的数据存放位置由服务器自己决定。

DELETE: 删除某一个资源。基本上这个也很少见，不过还是有一些地方比如amazon的S3云服务里面就用的这个方法来删除资源。

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，不过用的次数的确是少，在这里提一下。

Python

```
import urllib2  
request =  
1 import urllib2  
2 request = urllib2.Request(uri, data=data)  
3 request.get_method = lambda: 'PUT' # or 'DELETE'  
4 response = urllib2.urlopen(request)
```

5. 使用DebugLog

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，这个也不太常用，仅提一下

Python

```
import urllib2  
httpHandler =  
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.baidu.com')
```

以上便是一部分高级特性，前三个是重要内容，在后面，还有cookies的设置还有异常的处理，小伙伴们加油！

[打赏支持我写出更多好文章，谢谢！](#)

[打赏作者](#)

[打赏支持我写出更多好文章，谢谢！](#)

任选一种支付方式



1 赞 8 收藏 [2 评论](#)

关于作者：崔庆才



静觅 静静寻觅生活的美好个人站点 cuiqingcai.com [个人主页](#) · [我的文章](#) · 13 ·

Python爬虫入门（3）：Urllib库的基本使用

本文作者：[伯乐在线 - 崔庆才](#)。未经作者许可，禁止转载！
欢迎加入伯乐在线[专栏作者](#)。

- [Python爬虫入门（1）：综述](#)
- [Python爬虫入门（2）：爬虫基础了解](#)
- [Python爬虫入门（3）：Urllib库的基本使用](#)
- [Python爬虫入门（4）：Urllib库的高级用法](#)
- [Python爬虫入门（5）：URLError异常处理](#)
- [Python爬虫入门（6）：Cookie的使用](#)
- [Python爬虫入门（7）：正则表达式](#)
- [Python爬虫入门（8）：Beautiful Soup的用法](#)

那么接下来，小伙伴们就一起和我真正迈向我们的爬虫之路吧。

1. 分分钟扒一个网页下来

怎样扒网页呢？其实就是根据URL来获取它的网页信息，虽然我们在浏览器中看到的是一幅幅优美的画面，但是其实是由浏览器解释才呈现出来的，实质它是一段HTML代码，加JS、CSS，如果把网页比作一个人，那么HTML便是他的骨架，JS便是他的肌肉，CSS便是它的衣服。所以最重要的部分是存在于HTML中的，下面我们就写个例子来扒一个网页下来。

Python

```
import urllib2  
  
1 import urllib2  
2  
3 response = urllib2.urlopen("http://www.baidu.com")  
4 print response.read()
```

是的你没看错，真正的程序就两行，把它保存成 demo.py，进入该文件的目录，执行如下命令查看运行结果，感受一下。

Python

```
python demo.py  
  
1 python demo.py
```

```
cqc@cqc-Lenovo-IdeaPad-Y480: ~/workspace/Urllib/src
e/i))||c.match(/(linux.*firefox)/i)||c.match(/Chrome\|29/i)||c.match(/mac os x.*firefox/i)||b.match(/\bISSW=1/)||UPS.get("isSwitch")==0;if(bds&&bds.comm){bds.comm.supportis!=a;bds.comm.isui=true}window._restart_confirm_timeout=true;window._confirm_timeout=8000;window._disable_is_guide=true;window._disable_swap_to_empty=true;window._switch_add_mask=true;if(window._async_merge){window._async_his=a;document.write("<script src='http://s1.bdstatic.com/r/www/cache/static/global/js/all_async_search_c0ab2264.js'></script>")}else{if(a){document.write("<script src='http://s1.bdstatic.com/r/www/cache/static/global/js/all_async_popstate1_c9b5f5f0.js'></script>")}else{document.write("<script src='http://s1.bdstatic.com/r/www/cache/static/global/js/all_instant_search1_508f54dd.js'></script>")}}if(bds.comm.newindex){$(window).on("index_off",function(){$('div c-tips-container').insertAfter("#wrapper");if(window._sample_dynamic_tab){$("#s_tab").remove()}})}$(function(){setTimeout(function(){$.ajax({url:"http://s1.bdstatic.com/r/www/cache/static/baiduia/baidu_a_63043d0c.js",cache:true,dataType:"script"}),0)});if(bds.comm&&bds.comm.ishome&&Cookie.get("H_PS_PSSID")){bds.comm.indexSid=Cookie.get("H_PS_PSSID")}})();<script><script>if(bds.comm.supportis){window._restart_confirm_timeout=true;window._confirm_timeout=8000;window._disable_is_guide=true;window._disable_swap_to_empty=true;}initPreload({'isui':true,'index_form':'#form','index_kw':'#kw','result_form':'#form','result_kw':'#kw'});</script><script>if(navigator.cookieEnabled){document.cookie="NOJS=;expires=Sat, 01 Jan 2000 00:00:00 GMT";}</script></body></html>
```

```
cqc@cqc-Lenovo-IdeaPad-Y480:~/workspace/Urllib/src$
```

看，这个网页的[源码](#)已经被我们扒下来了，是不是很酸爽？

2.分析扒网页的方法

那么我们来分析这两行代码，第一行

Python

```
response = 
urllib2.urlopen("http://w
1 response = urllib2.urlopen("http://www.baidu.com")
```

首先我们调用的是urllib2库里面的urlopen方法，传入一个URL，这个网址是百度首页，协议是HTTP协议，当然你也可以把HTTP换做FTP,FILE,HTTPS 等等，只是代表了一种[访问控制](#)协议，urlopen一般接受三个参数，它的参数如下：

Python

```
urlopen(url, data, timeout)
1 urlopen(url, data, timeout)
```

第一个参数url即为URL，第二个参数data是访问URL时要传送的数据，第三个timeout是设置超时时间。

第二三个参数是可以不传送的，data默认为空None，timeout默认为socket._GLOBAL_DEFAULT_TIMEOUT

第一个参数URL是必须要传送的，在这个例子里面我们传送了百度的URL，执行urlopen方法之后，返回一个response对象，返回信息便保存在这里面。

Python

```
print response.read()
```

```
1 print response.read()
```

response对象有一个read方法，可以返回获取到的网页内容。

如果不加read直接打印会是什么？答案如下：

Python

```
<addinfourl at  
139728495260376
```

```
1 <addinfourl at 139728495260376 whose fp = <socket._fileobject object at 0x7f1513fb3ad0>>
```

直接打印出了该对象的描述，所以记得一定要加read方法，否则它不出来内容可就不怪我咯！

3.构造Request

其实上面的urlopen参数可以传入一个request请求，它其实就是一个Request类的实例，构造时需要传入Url,Data等等的内容。比如上面的两行代码，我们可以这么改写

Python

```
import urllib2
```

```
1 import urllib2  
2  
3 request = urllib2.Request("http://www.baidu.com")  
4 response = urllib2.urlopen(request)  
5 print response.read()
```

运行结果是完全一样的，只不过中间多了一个request对象，推荐大家这么写，因为在构建请求时还需要加入好多内容，通过构建一个request，[服务器响应请求得到应答](#)，这样显得逻辑上清晰明确。

4.POST和GET数据传送

上面的程序演示了最基本的网页抓取，不过，现在大多数网站都是动态网页，需要你动态地传递参数给它，它做出对应的响应。所以，在访问时，我们需要传递数据给它。最常见的情况是什么？对了，就是登录注册的时候呀。

把数据用户名和密码传送到一个URL，然后你得到服务器处理之后的响应，这个该怎么办？下面让我来为小伙伴们揭晓吧！

数据传送分为POST和GET两种方式，两种方式有什么区别呢？

最重要的区别是GET方式是直接以链接形式访问，链接中包含了所有的参数，当然如果包含了密码的话是一种不安全的选择，不过你可以直观地看到自己提交了什么内容。POST则不会在网址上显示所有的参数，不过如果你想直接查看提交了什么就不太方便了，大家可以酌情选择。

POST方式：

上面我们说了data参数是干嘛的？对了，它就是用在这里的，我们传送的数据就是这个参数data，下面演示一下POST方式。

Python

```
import urllib
import urllib2
1 import urllib
2 import urllib2
3
4 values = {"username":"1016903103@qq.com", "password":"XXXX"}
5 data = urllib.urlencode(values)
6 url = "https://passport.csdn.net/account/login?from=http://my.csdn.net/my/myscsdn"
7 request = urllib2.Request(url,data)
8 response = urllib2.urlopen(request)
9 print response.read()
```

我们引入了urllib库，现在我们模拟登陆CSDN，当然上述代码可能登陆不进去，因为还要做一些设置头部header的工作，或者还有一些参数没有设置全，还没有提及到在此就不写上去了，在此只是说明登录的原理。我们需要定义一个字典，名字为values，参数我设置了username和password，下面利用urllib的urlencode方法将字典编码，命名为data，构建request时传入两个参数，url和data，运行程序，即可实现登陆，返回的便是登陆后呈现的页面内容。当然你可以自己搭建一个[服务器](#)来测试一下。

注意上面字典的定义方式还有一种，下面的写法是等价的

Python

```
import urllib
import urllib2
1 import urllib
2 import urllib2
3
4 values = {}
5 values['username'] = "1016903103@qq.com"
6 values['password'] = "XXXX"
7 data = urllib.urlencode(values)
8 url = "http://passport.csdn.net/account/login?from=http://my.csdn.net/my/myscsdn"
9 request = urllib2.Request(url,data)
10 response = urllib2.urlopen(request)
11 print response.read()
```

以上方法便实现了POST方式的传送

GET方式：

至于GET方式我们可以直接把参数写到网址上面，直接构建一个带参数的URL出来即可。

Python

```
import urllib  
import urllib2  
  
1 import urllib  
2 import urllib2  
3  
4 values={}  
5 values['username'] = "1016903103@qq.com"  
6 values['password']="XXXX"  
7 data = urllib.urlencode(values)  
8 url = "http://passport.csdn.net/account/login"  
9 geturl = url + "?" + data  
10 request = urllib2.Request(geturl)  
11 response = urllib2.urlopen(request)  
12 print response.read()
```

你可以print geturl，打印输出一下url，发现其实就是原来的url加? 然后加编码后的参数

Python

```
http://passport.csdn.net/  
account/login?  
1 http://passport.csdn.net/account/login?username=1016903103%40qq.com&password=XXXX
```

和我们平常GET访问方式一模一样，这样就实现了数据的GET方式传送。

本节讲解了一些基本使用，可以抓取到一些基本的网页信息，小伙伴们加油！

打赏支持我写出更多好文章，谢谢！

[打赏作者](#)

打赏支持我写出更多好文章，谢谢！

任选一种支付方式

微信扫一扫转账



向崔庆才转账

赞赏伯乐在线的文章

¥2.00

支付宝扫一扫，向我付款



赞赏你发在伯乐在线的文章

¥2.00

4 赞 13 收藏 [4 评论](#)

关于作者：[崔庆才](#)



静觅 静静寻觅生活的美好个人站点 cuiqingcai.com [个人主页](#) · [我的文章](#) · 13 ·

Python爬虫入门（2）：爬虫基础了解

本文作者：[伯乐在线 - 崔庆才](#)。未经作者许可，禁止转载！

欢迎加入伯乐在线[专栏作者](#)。

- [Python爬虫入门（1）：综述](#)
- [Python爬虫入门（2）：爬虫基础了解](#)
- [Python爬虫入门（3）：Urllib库的基本使用](#)
- [Python爬虫入门（4）：Urllib库的高级用法](#)
- [Python爬虫入门（5）：URLError异常处理](#)
- [Python爬虫入门（6）：Cookie的使用](#)
- [Python爬虫入门（7）：正则表达式](#)
- [Python爬虫入门（8）：Beautiful Soup的用法](#)

1.什么是爬虫

爬虫，即网络爬虫，大家可以理解为在网络上爬行的一只蜘蛛，互联网就比作一张大网，而爬虫便是在这张网上爬来爬去的蜘蛛咯，如果它遇到资源，那么它就会抓取下来。想抓取什么？这个由你来控制它咯。

比如它在抓取一个网页，在这个网中他发现了一条道路，其实这就是指向网页的超链接，那么它就可以爬到另一张网上来获取数据。这样，整个连在一起的大网对这之蜘蛛来说触手可及，分分钟爬下来不是事儿。

2.浏览网页的过程

在用户浏览网页的过程中，我们可能会看到许多好看的图片，比如<http://image.baidu.com/>，我们会看到几张的图片以及百度搜索框，这个过程其实就是用户输入网址之后，经过DNS服务器，找到服务器主机，向服务器发出一个请求，服务器经过解析之后，发送给用户的浏览器HTML、JS、CSS等文件，浏览器解析出来，用户便可以看到形形色色的图片了。

因此，用户看到的网页实质是由HTML代码构成的，爬虫爬来的便是这些内容，通过分析和过滤这些HTML代码，实现对图片、文字等资源的获取。

3.URL的含义

URL，即统一资源定位符，也就是我们说的网址，统一资源定位符是对可以从互联网上得到的资源的位置和访问方法的一种简洁的表示，是互联网上标准资源的地址。互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。

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。

爬虫爬取数据时必须要有一个目标的URL才可以获取数据，因此，它是爬虫获取数据的基本依据，准确理解它的含义对爬虫学习有很大帮助。

4. 环境的配置

学习Python，当然少不了环境的配置，最初我用的是Notepad++，不过发现它的提示功能实在是太弱了，于是，在Windows下我用了PyCharm，在Linux下我用了Eclipse for Python，另外还有几款比较优秀的IDE，大家可以参考这篇文章 [学习Python推荐的IDE](#)。好的[开发工具](#)是前进的[推进器](#)，希望大家可以找到适合自己的IDE

下一节，我们就正式步入 Python 爬虫学习的殿堂了，小伙伴准备好了嘛？

打赏支持我写出更多好文章，谢谢！

[打赏作者](#)

打赏支持我写出更多好文章，谢谢！

任选一种支付方式



1 赞 6 收藏 [2 评论](#)

关于作者：崔庆才



静觅 静静寻觅生活的美好个人站点 cuiqingcai.com [个人主页](#) · [我的文章](#) · 13 ·

Python爬虫入门（1）：综述

本文作者：[伯乐在线 - 崔庆才](#)。未经作者许可，禁止转载！

欢迎加入伯乐在线[专栏作者](#)。

- [Python爬虫入门（1）：综述](#)
- [Python爬虫入门（2）：爬虫基础了解](#)
- [Python爬虫入门（3）：Urllib库的基本使用](#)
- [Python爬虫入门（4）：Urllib库的高级用法](#)
- [Python爬虫入门（5）：URLError异常处理](#)
- [Python爬虫入门（6）：Cookie的使用](#)
- [Python爬虫入门（7）：正则表达式](#)
- [Python爬虫入门（8）：Beautiful Soup的用法](#)

大家好哈，最近博主在学习Python，学习期间也遇到一些问题，获得了一些经验，在此将自己的学习系统地整理下来，如果大家有兴趣学习爬虫的话，可以将这些文章作为参考，也欢迎大家一共分享学习经验。

Python版本:2.7，Python 3请另寻其他博文。

首先爬虫是什么？

网络爬虫（又被称为网页蜘蛛，网络机器人，在FOAF社区中间，更经常的称为网页追逐者），是一种按照一定的规则，自动的抓取[万维网](#)信息的程序或者脚本。

根据我的经验，要学习Python爬虫，我们要学习的共有以下几点：

- Python基础知识
- Python中urllib和urllib2库的用法
- Python正则表达式
- Python爬虫框架Scrapy
- Python爬虫更高级的功能

1. Python基础学习

首先，我们要用Python写爬虫，肯定要了解Python的基础吧，万丈高楼平地起，不能忘啦那地基，哈哈，那么我就分享一下自己曾经看过的一些Python教程，小伙伴们可以作为参考。

1) 慕课网Python教程

曾经有一些基础的语法是在慕课网上看的，上面附有一些练习，学习完之后可以作为练习，感觉效果还是蛮不错的，不过稍微遗憾的是内容基本上都是最基础的，入门开始的话，就这个吧

学习网址：[慕课网Python教程](#)

2) 廖雪峰Python教程

后来，我发现了廖老师的Python教程，讲的那是非常通俗易懂哪，感觉也是非常不错，大家如果想进一步了解Python就看一下这个吧。

学习网址：[廖雪峰Python教程](#)

3) 简明Python教程

还有一个我看过的，简明Python教程，感觉讲的也不错

学习网址：[简明Python教程](#)

2.Python urllib和urllib2 库的用法

urllib和urllib2库是学习Python爬虫最基本的库，利用这个库我们可以得到网页的内容，并对内容用正则表达式提取分析，得到我们想要的结果。这个在学习过程中我会和大家分享的。

3.Python 正则表达式

Python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。这个在后面的博文会分享的。

4.爬虫框架Scrapy

如果你是一个Python高手，基本的爬虫知识都已经掌握了，那么就寻觅一下Python框架吧，我选择的框架是Scrapy框架。这个框架有什么强大的功能呢？下面是它的官方介绍：

HTML, XML源数据 选择及提取 的内置支持

提供了一系列在spider之间共享的可复用的过滤器(即 Item Loaders)，对智能处理爬取数据提供了内置支持。

通过 feed导出 提供了多格式(JSON、 CSV、 XML)，多[存储](#)后端(FTP、 S3、 本地[文件系统](#))的内置支持

提供了media pipeline，可以 自动下载 爬取到的数据中的图片(或者其他资源)。

高扩展性。您可以通过使用 signals ，设计好的API(中间件, extensions, pipelines)来定制实现您的功能。

内置的[中间件](#)及扩展为下列功能提供了支持:

cookies and session 处理

HTTP 压缩

HTTP 认证

HTTP 缓存

user-agent模拟

robots.txt

爬取深度限制

针对非英语语系中不标准或者错误的编码声明，提供了自动检测以及健壮的编码支持。

支持根据模板生成爬虫。在加速爬虫创建的同时，保持在大型项目中的代码更为一致。详

细内容请参阅 genspider 命令。

针对多爬虫下性能评估、失败[检测](#)，提供了可扩展的 状态收集工具。

提供[交互式](#)shell终端，为您测试XPath表达式，编写和调试爬虫提供了极大的方便

提供 System service, 简化在生产环境的部署及运行

内置 Web service, 使您可以监视及控制您的机器

内置 Telnet终端，通过在Scrapy进程中钩入Python终端，使您可以查看并且调试爬虫

Logging 为您提供在爬取过程中捕捉错误提供了方便

支持 Sitemaps 爬取

具有缓存的DNS解析器

官方文档：<http://doc.scrapy.org/en/latest/>

等我们掌握了基础的知识，再用这个 Scrapy 框架吧！

扯了这么多，好像没多少有用的东西额，那就不扯啦！

下面开始我们正式进入爬虫之旅吧！

打赏支持我写出更多好文章，谢谢！

[打赏作者](#)

打赏支持我写出更多好文章，谢谢！

任选一种支付方式

微信扫一扫转账



向崔庆才转账

赞赏伯乐在线的文章

¥ 2.00

支付宝扫一扫，向我付款



¥2.00

赞赏你发在伯乐在线的文章

1 赞 46 收藏 [评论](#)

关于作者：[崔庆才](#)



静觅 静静寻觅生活的美好个人站点 cuiqingcai.com [个人主页](#) · [我的文章](#) · 13 ·

Python下用Scrapy和MongoDB构建爬虫系统(2)

本文由[伯乐在线 - PyPer](#) 翻译, [笑虎](#) 校稿。未经许可, 禁止转载!

英文出处: [realpython](#)。欢迎加入[翻译组](#)。

[在上一篇中, 我们实现了一个基本网络爬虫](#), 它可以从StackOverflow上下载最新的问题, 并将它们存储在MongoDB数据库中。在本文中, 我们将对其扩展, 使它能够爬取每个网页底部的分页链接, 并从每一页中下载问题(包含问题标题和URL)。

在你开始任何爬取工作之前, 检查目标网站的使用条款并遵守robots.txt文件。同时, 做爬取练习时遵守道德, 不要在短时间内向某个网站发起大量请求。像对待自己的网站一样对待任何你将爬取的网站。

开始

有两种可能的方法来接着从上次我们停下的地方继续进行。

第一个方法是, 扩展我们现有的网络爬虫, 通过利用一个xpath表达式从"parse_item"方法里的响应中提取每个下一页链接, 并通过回调同一个parse_item方法产生一个请求对象。利用这种方法, 爬虫会自动生成针对我们指定的链接的新请求, 你可以在[Scrapy文档](#)这里找到更多有关该方法的信息。

另一个更简单的方法是, 使用一个不同类型的爬虫—CrawlSpider ([链接](#))。这是基本Spider的一个扩展版本, 它刚好满足我们的要求。

CrawlSpider

我们将使用与上一篇教程中相同的爬虫项目, 所以如果你需要的话可以从repo上获取这些代码。

创建样板

在"stack"目录中, 首先由crawl模板[生成](#)爬虫样板。

Python

```
$ scrapy genspider  
stack_crawler
```

```
1 $ scrapy genspider stack_crawler stackoverflow.com -t crawl  
2 Created spider 'stack_crawler' using template 'crawl' in module:  
3 stack.spiders.stack_crawler
```

Scrapy项目现在看起来应该像这样:

Python



```
1 └── scrapy.cfg
2 └── stack
3     ├── scrapy.cfg
4     └── stack
5         ├── __init__.py
6         ├── items.py
7         ├── pipelines.py
8         ├── settings.py
9         └── spiders
10            ├── __init__.py
11            ├── stack_crawler.py
12            └── stack_spider.py
```

stack_crawler.py文件内容如下：

Python

```
# -*- coding: utf-8 -*-
import scrapy
# -*- coding: utf-8 -*-
1 import scrapy
2 from scrapy.contrib.linkextractors import LinkExtractor
3 from scrapy.contrib.spiders import CrawlSpider, Rule
4
5 from stack.items import StackItem
6
7 class StackCrawlerSpider(CrawlSpider):
8     name = 'stack_crawler'
9     allowed_domains = ['stackoverflow.com']
10    start_urls = ['http://www.stackoverflow.com/']
11
12    rules = (
13        Rule(LinkExtractor(allow=r'Items/'), callback='parse_item', follow=True),
14    )
15
16    def parse_item(self, response):
17        i = StackItem()
18        #i['domain_id'] = response.xpath('//input[@id="sid"]/@value').extract()
19        #i['name'] = response.xpath('//div[@id="name"]').extract()
20        #i['description'] = response.xpath('//div[@id="description"]').extract()
21
22        return i
```

我们只需要对这个样板做一些更新。

更新“start_urls”列表

首先，添加问题的第一个页面链接到start_urls列表：

Python

```
start_urls = [
1 start_urls = [
2     'http://stackoverflow.com/questions?pagesize=50&sort=newest'
3 ]
```

更新“rules”列表

接下来，我们需要添加一个正则表达式到“rules”属性中，以此告诉爬虫在哪里可以找到下一个页面链接：

Python

```
rules = [  
    Rule(LinkExtractor(allow=r'questions?page=[0-9]&sort=newest'),  
        callback='parse_item', follow=True)  
]
```

现在爬虫能根据那些链接自动请求新的页面，并将响应传递给“parse_item”方法，以此来提取问题和对应的标题。

如果你仔细查看的话，可以发现这个正则表达式限制了它只能爬取前9个网页，因为在这个demo中，我们不想爬取所有的176234个网页。

更新“parse_item”方法

现在我们只需编写如何使用xpath解析网页，这一点我们已经在上一篇教程中实现过了，所以直接复制过来。

Python

```
def parse_item(self, response):  
  
    questions = response.xpath('//div[@class="summary"]/h3')  
  
    for question in questions:  
        item = StackItem()  
        item['url'] = question.xpath(  
            'a[@class="question-hyperlink"]/@href').extract()[0]  
        item['title'] = question.xpath(  
            'a[@class="question-hyperlink"]/text()').extract()[0]  
        yield item
```

这就是为爬虫提供的解析代码，但是现在先不要启动它。

添加一个下载延迟

我们需要通过在settings.py文件中设定一个下载延迟来善待StackOverflow（和任何其他网站）。

Python

```
DOWNLOAD_DELAY = 5  
  
1 DOWNLOAD_DELAY = 5
```

这告诉爬虫需要在每两个发出的新请求之间等待5秒钟。你也很有必要做这样的限制，因为如果你不这么做的话，StackOverflow将会限制你的访问流量，如果你继续不加限制地爬取该网站，那么你的IP将会被禁止。所有，友好点—要像对待自己的网站一样对待任何你爬取的网站。

现在只剩下一件事要考虑—存储数据。

MongoDB

上次我们仅仅下载了50个问题，但是因为这次我们要爬取更多的数据，所有我们希望避免向数据库中添加重复的问题。为了实现这一点，我们可以使用一个MongoDB的 [upsert](#)方法，它意味着如果一个问题已经存在数据库中，我们将更新它的标题；否则我们将新问题插入数据库中。

修改我们前面定义的MongoDBPipeline：

Python

```
class MongoDBPipeline(object):
    def __init__(self):
        connection = pymongo.Connection(
            settings['MONGODB_SERVER'],
            settings['MONGODB_PORT']
        )
        db = connection[settings['MONGODB_DB']]
        self.collection = db[settings['MONGODB_COLLECTION']]

    def process_item(self, item, spider):
        for data in item:
            if not data:
                raise DropItem("Missing data!")
        self.collection.update({'url': item['url']}, dict(item), upsert=True)
        log.msg("Question added to MongoDB database!",
                level=log.DEBUG, spider=spider)
        return item
```

为简单起见，我们没有优化查询，也没有处理索引值，因为这不是一个生产环境。

测试

启动爬虫！

Python

```
$ scrapy crawl questions
```

1 \$ scrapy crawl questions

现在你可以坐下来，看着你的数据库渐渐充满数据。

结论

你可以从[Github库](#)下载整个源代码，也可以在下面评论或提问。

1 赞 12 收藏 [3 评论](#)

关于作者： [PyPer](#)



一名就读于羊城某高校的学生，主要关注 Python、Perl、PowerShell等脚本技术，熟悉MSSQL、Oracle、Redis等数据库，新浪微博：<http://weibo.com/LiwianwplO>，微信公众号“NETEC”。[个人主页](#) · [我的文章](#) · [11](#)

Python下用Scrapy和MongoDB构建爬虫系统(1)

本文由[伯乐在线 - 木羊同学](#)翻译, [笑虎](#)校稿。未经许可, 禁止转载!

英文出处: realpython.com。欢迎加入[翻译组](#)。

这篇文章将根据真实的兼职需求编写一个爬虫, 用户想要一个Python程序从Stack Overflow抓取数据, 获取新的问题(问题标题和URL)。抓取的数据应当存入MongoDB。值得注意的是, Stack Overflow已经提供了可用于读取同样数据的API。但是用户想要一个爬虫, 那就给他一个爬虫。

像往常一样, 在开始任何抓取工作前, 一定要先查看该网站的使用/服务条款, 要尊重 robots.txt 文件。抓取行为应该遵守道德, 不要在很短时间内发起大量请求, 从而导致网站遭受泛洪攻击。对待那些你要抓取的网站, 要像对待自己的一样。

安装

我们需要Scrapy库(v0.24.4), 以及用于在MongoDB中存储数据的PyMongo库(v2.7.2)。同样需要安装MongoDB。

Scrapy

如果使用OSX或某种Linux, 使用pip安装Scrapy(激活命令行) :

Python

```
$ pip install Scrapy
```

```
1 $ pip install Scrapy
```

如果使用Windows的机器, 你需要手动安装一堆依赖库(木羊吐槽: Win下也是有pip的po主你不要黑她, 经测可以用上面命令直接安装成功)。请参考官方文档详细说明以及我创建的Youtube视频。

一旦Scrapy安装完毕, 可在Python命令行中使用这个命令验证:

Python

```
>>> import scrapy
```

```
>>>
```

```
1 >>> import scrapy
```

```
2 >>>
```

如果没有出错, 安装就完成了。

PyMongo

下一步, 使用pip安装PyMongo:

Python

```
$ pip install pymongo
```

```
1 $ pip install pymongo
```

现在可以开始构建爬虫了。

Scrapy工程

先创建一个新的Scrapy工程：

Python

```
$ scrapy startproject stack
```

```
1 $ scrapy startproject stack
```

这条命令创建了许多文件和文件夹，其中包含一套有助于你快速开始的基本模板：

Python

```
scrapy.cfg  
stack
```

```
1   scrapy.cfg  
2   stack  
3     __init__.py  
4     items.py  
5     pipelines.py  
6     settings.py  
7     spiders  
8       __init__.py
```

提取数据

items.py文件用于定义存储“容器”，用来存储将要抓取的数据。

StackItem()类继承自Item ([文档](#))，主要包含一些Scrapy已经为我们创建好的预定义对象：

Python

```
import scrapy
```

```
1 import scrapy  
2  
3 class StackItem(scrapy.Item):  
4     # define the fields for your item here like:  
5     # name = scrapy.Field()  
6     pass
```

添加一些想要收集的项。用户想要每条问题的标题和URL。那么，照这样更新items.py：

Python

```
from scrapy.item import Item, Field  
1 from scrapy.item import Item, Field  
2  
3 class StackItem(Item):  
4     title = Field()  
5     url = Field()
```

创建蜘蛛

在“spiders”目录下建立一个名为stack_spider.py的文件。这里是见证奇迹发生的地方—比如在这里告诉Scrapy怎么去找到我们想要的指定数据。正如你想的那样，对于每一个独立的网页，stack_spider.py都是不同的。

我们从定义一个类开始，这个类继承Scrapy的Spider，并添加一些必须的属性：

Python

```
from scrapy import Spider  
1 from scrapy import Spider  
2  
3  
4 class StackSpider(Spider):  
5     name = "stack"  
6     allowed_domains = ["stackoverflow.com"]  
7     start_urls = [  
8         "http://stackoverflow.com/questions?pagesize=50&sort=newest",  
9     ]
```

最初一些变量的含义很容易理解 ([文档](#)) :

- 定义蜘蛛的名字。
- allowed_domains 包含构成许可域的基础URL，供蜘蛛去爬。
- start_urls 是一个URL列表，蜘蛛从这里开始爬。蜘蛛从start_urls中的URL下载数据，所有后续的URL将从这些数据中获取。

XPath选择器

接下来，Scrapy使用XPath选择器在一个网站上提取数据。也就是说，我们可以通过一个给定的XPath选择HTML数据的特定部分。正如Scrapy所称，“XPath是一种选择XML节点的语言，也可以用于HTML。”

使用Chrome的开发者工具，可以很容易找到一个特定的Xpath。简单地检查一个特定的HTML元素，复制XPath，然后修改（如有需要）。

Scrapy

Search docs

Scrapy at a glance
Installation guide
Scrapy Tutorial
Examples
Command line tool
Items
Spiders
Selectors
Item Loaders
Scrapy shell

Read the Docs v: latest ▾

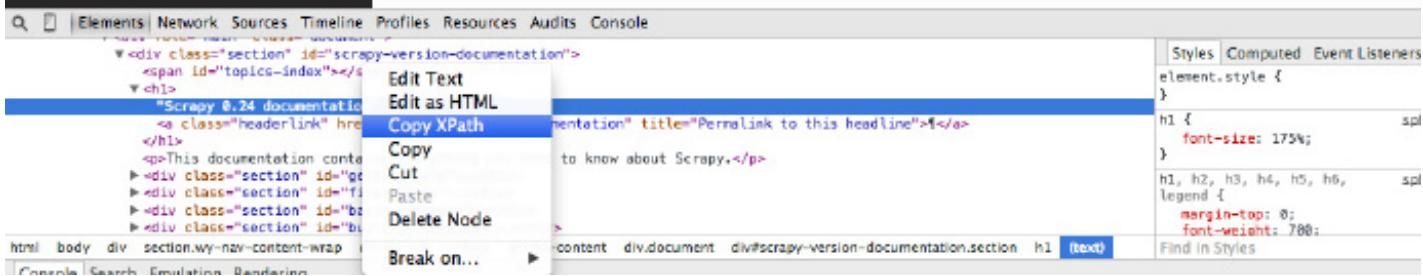
Scrapy 0.24 documentation

This documentation contains everything you need to know about Scrapy.

Getting help

Having trouble? We'd like to help!

- Try the [FAQ](#) – it's got answers to some common questions.
- Looking for specific information? Try the [Index](#) or [Module Index](#).
- Search for information in the [archives](#) of the [scrapy-users mailing list](#), or [post a question](#).
- Ask a question in the [#scrapy IRC channel](#).
- Report bugs with Scrapy in our [issue tracker](#).



开发者工具同时为用户提供在JavaScript控制台测试XPath选择器的功能，使用\$*x*，如\$*x*("//img")：

Scrapy

Search docs

Scrapy at a glance
Installation guide
Scrapy Tutorial
Examples
Command line tool
Items
Spiders
Selectors
Item Loaders
Scrapy shell

Read the Docs v: latest ▾

Elements Network Sources Timeline Profiles Resources Audits Console

```
<top frame> ↗ Preserve log
> $x("//*[@id='scrapy-version-documentation']/h1/text()")*
< "Scrapy 0.24 documentation"
```

Scrapy 0.24 documentation

This documentation contains everything you need to know about Scrapy.

Getting help

Having trouble? We'd like to help!

- Try the [FAQ](#) – it's got answers to some common questions.
- Looking for specific information? Try the [Index](#) or [Module Index](#).
- Search for information in the [archives](#) of the [scrapy-users mailing list](#), or [post a question](#).
- Ask a question in the [#scrapy IRC channel](#).
- Report bugs with Scrapy in our [issue tracker](#).

继续，通过定义的XPath告诉Scrapy去哪里寻找信息。在Chrom中导航至Stack Overflow网址，寻找XPath选择器。



stackoverflow

- [Questions](#)
- [Tags](#)
- [Users](#)
- [Badges](#)
- [Unanswered](#)

All Questions

newest

414 featured

frequent

votes

active

unanswered

0
votes

Spring data @transactional not rolling back with SQL Server and after runtimeexception

I've enabled my spring application to use transactions and annotated my service method accordingly but the changes to my DB persist when a RuntimeException is thrown. My Spring configuration looks ...

2 views

sql-server spring-mvc spring-data spring-transactions runtimeexception

asked 13 mins ago

 Henrique Ordine
1,104 ● 9 ● 24

Elements Network Sources Timeline Profiles Resources Audits Console

```

<div class="subheader">...</div>
<div id="questions">
  <div class="question-summary" id="question-summary-27624141">
    <div class="statscontainer">...</div>
    <div class="summary">
      <h3>...</h3>
      <div class="excerpt">...</div>
      <div class="tags t-sql-server t-spring-mvc t-spring-data t-spring-transactions t-runtimeexception">...</div>
      <div class="started fr">...</div>
    </div>
  </div>
</div>

```

右键点击第一条问题，选择“插入元素”：

现在从<div class="summary">, //*[@id="question-summary-27624141"]/div[2]中抓取XPath，然后在JavaScript控制台测试它：



stackoverflow

- [Questions](#)
- [Tags](#)
- [Users](#)
- [Badges](#)
- [Unanswered](#)

All Questions

newest

414 featured

frequent

votes

active

unanswered

0
votes

Spring data @transactional not rolling back with SQL Server and after runtimeexception

I've enabled my spring application to use transactions and annotated my service method accordingly but the changes to my DB persist when a RuntimeException is thrown. My Spring configuration looks ...

2 views

sql-server spring-mvc spring-data spring-transactions runtimeexception

asked 19 mins ago

 Henrique Ordine
1,104 ● 9 ● 24

Elements Network Sources Timeline Profiles Resources Audits Console

↳ <div class="summary">...</div>

```

$xpath('//*[@id="question-summary-27624141"]/div[2])
<div class="summary">...</div>

```

也许你会说，这只选择了一条问题。现在需要改变XPath去抓取所有的问题。有什么想法？很简单：`//div[@class="summary"]/h3`。

什么意思呢？本质上，这条XPath是说：抓取`<div>`的子树中所有这一类`<h3>`元素的总集。在JavaScript控制台中测试XPath。

请注意我们不会使用Chrome开发者工具的实际输出。在大多数案例中，这些输出仅仅是一个参考，便于直接找到能用的XPath。

现在更新`stack_spider.py`脚本：

Python

```
from scrapy import Spider
Spider
1 from scrapy import Spider
2 from scrapy.selector import Selector
3
4
5 class StackSpider(Spider):
6     name = "stack"
7     allowed_domains = ["stackoverflow.com"]
8     start_urls = [
9         "http://stackoverflow.com/questions?pagesize=50&sort=newest",
10    ]
11
12 def parse(self, response):
13     questions = Selector(response).xpath('//div[@class="summary"]/h3')
```

提取数据

我们仍然需要解析和抓取想要的数据，它符合`<div class="summary"><h3>`。继续，像这样更新`stack_spider.py`：

Python

```
from scrapy import Spider
Spider
1 from scrapy import Spider
2 from scrapy.selector import Selector
3
4 from stack.items import StackItem
5
6
7 class StackSpider(Spider):
8     name = "stack"
9     allowed_domains = ["stackoverflow.com"]
10    start_urls = [
11        "http://stackoverflow.com/questions?pagesize=50&sort=newest",
12    ]
13
14 def parse(self, response):
15     questions = Selector(response).xpath('//div[@class="summary"]/h3')
16
17     for question in questions:
18         item = StackItem()
```

```
19     item['title'] = question.xpath(
20         'a[@class="question-hyperlink"]/text()').extract()[0]
21     item['url'] = question.xpath(
22         'a[@class="question-hyperlink"]/@href').extract()[0]
23     yield item
```

我们将遍历问题，从抓取的数据中分配标题和URL的值。一定要利用Chrome开发者工具的JavaScript控制台测试XPath的选择器，例如`$x('//div[@class="summary"]/h3/a[@class="question-hyperlink"]/text()')` 和 `$x('//div[@class="summary"]/h3/a[@class="question-hyperlink"]/@href')`。

测试

准备好第一次测试了吗？只要简单地在“stack”目录中运行下面命令：

Python

```
$ scrapy crawl stack
```

```
1 $ scrapy crawl stack
```

随着Scrapy堆栈跟踪，你应该看到50条问题的标题和URL输出。你可以用下面这条小命令输出一个JSON文件：

Python

```
$ scrapy crawl stack -o
items.json -t json
```

```
1 $ scrapy crawl stack -o items.json -t json
```

我们已经基于要寻找的数据实现了爬虫。现在需要将抓取的数据存入MongoDB。

在MongoDB中存储数据

每当有一项返回，我们想验证数据，然后添加进一个Mongo集合。

第一步是创建一个我们计划用来保存所有抓取数据的数据库。打开`settings.py`，指定管道然后加入数据库设置：

Python

```
ITEM_PIPELINES = [stack.pipelines.
```

```
1 ITEM_PIPELINES = [stack.pipelines.MongoDBPipeline, ]
2
3 MONGODB_SERVER = "localhost"
4 MONGODB_PORT = 27017
5 MONGODB_DB = "stackoverflow"
6 MONGODB_COLLECTION = "questions"
```

管道管理

我们建立了爬虫去抓取和解析HTML，而且已经设置了数据库配置。现在要在pipelines.py中通过一个管道连接两个部分。

连接数据库

首先，让我们定义一个函数去连接数据库：

Python

```
import pymongo  
1 import pymongo  
2  
3 from scrapy.conf import settings  
4  
5  
6 class MongoDBPipeline(object):  
7  
8     def __init__(self):  
9         connection = pymongo.Connection(  
10             settings['MONGODB_SERVER'],  
11             settings['MONGODB_PORT'])  
12         )  
13         db = connection[settings['MONGODB_DB']]  
14         self.collection = db[settings['MONGODB_COLLECTION']]
```

这里，我们创建一个类，MongoDBPipeline()，我们有一个构造函数初始化类，它定义Mongo的设置然后连接数据库。

处理数据

下一步，我们需要定义一个函数去处理被解析的数据：

Python

```
import pymongo  
1 import pymongo  
2  
3 from scrapy.conf import settings  
4 from scrapy.exceptions import DropItem  
5 from scrapy import log  
6  
7  
8 class MongoDBPipeline(object):  
9  
10    def __init__(self):  
11        connection = pymongo.Connection(  
12            settings['MONGODB_SERVER'],  
13            settings['MONGODB_PORT'])  
14        )  
15        db = connection[settings['MONGODB_DB']]  
16        self.collection = db[settings['MONGODB_COLLECTION']]  
17  
18    def process_item(self, item, spider):  
19        valid = True  
20        for data in item:
```

```

21     if not data:
22         valid = False
23         raise DropItem("Missing {0}!".format(data))
24     if valid:
25         self.collection.insert(dict(item))
26         log.msg("Question added to MongoDB database!", 
27                 level=log.DEBUG, spider=spider)
28     return item

```

我们建立一个数据库连接，解包数据，然后将它存入数据库。现在再测试一次！

测试

再次，在“stack”目录下运行下面命令：

Python

```
$ scrapy crawl stack
```

```
1 $ scrapy crawl stack
```

万岁！我们已经成功将我们爬下了的数据存入数据库：

Key	Value	Type
(1) ObjectId("54a735443386b710c9685b6...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b68")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... how to get listboxitem's value on listbox hold ev...	String
title	Object	
(2) ObjectId("54a735443386b710c9685b6...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b6f")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(3) ObjectId("54a735443386b710c9685b7...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b7")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(4) ObjectId("54a735443386b710c9685b7...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b7")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(5) ObjectId("54a735443386b710c9685b7...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b7")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(6) ObjectId("54a735443386b710c9685b7...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b7")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(7) ObjectId("54a735443386b710c9685b7...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b7")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(8) ObjectId("54a735443386b710c9685b4...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b4")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(9) ObjectId("54a735443386b710c9685b4...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b4")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(10) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(11) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(12) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(13) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(14) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(15) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(16) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(17) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(18) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(19) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(20) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(21) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(22) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(23) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(24) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(25) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(26) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(27) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	
(28) ObjectId("54a735443386b710c9685b...")	{ 3 fields }	Object
__id	ObjectId("54a735443386b710c9685b")	ObjectId
url	/questions/27750340/how-to-get-listboxitem's... Object	String
title	Object	

总结

这是一个用Scrapy爬取网页的简单示例。真实兼职工作需要能跟踪分页链接的脚本，用CrawlSpider（[文档](#)）抓取每一个页面，非常容易实现。自己动手实现下，在下面Github仓库链接中

写一个评论，快速查看代码。需要帮助？从[这个脚本](#)开始，它已经很接近完成了。然后查看第二部分，它包含完整的解决方案。

你可以从Github repo中下载完整的代码。如果有问题请跟贴评论。谢谢阅读！

新年快乐

2 赞 22 收藏 [5 评论](#)

关于作者：[木羊同学](#)



hackos.py [个人主页](#) · [我的文章](#) · 12

Python指南（1.1）：挑选解释器

本文由 [伯乐在线 - mtunique](#) 翻译, [艾凌风](#) 校稿。未经许可, 禁止转载!

英文出处: docs.python-guide.org。欢迎加入[翻译组](#)。

1.1 挑选解释器

1.1.1 Python的现状（2 vs 3）

当选择python解释器的时候，一个首先要面对的问题是：“我应该选择Python 2还是Python 3？”答案并不像人们想象的那么明显。

现状的基本要点如下：

1. Python 2.7 作为标准已经很长时间了
2. Python 3 将重大变换引入到语言中，其中有不少开发者不满意。
3. 几年内Python 2.7 将得到必要的安全更新。
4. Python 3正在不断发展，就像Python 2在过去几年一样。

所以，你现在可以看到为什么这不是一个简单的决定了。

1.1.2 建议

那我直言不讳：

用Python 3, 如果:

- 你不在乎。
- 你爱Python 3。
- 你不漠不关心2 vs 3。
- 你不知道用哪一个。
- 你接受变化。

用Python 2, 如果:

- 你爱Python 2, 对未来的Python 3感到悲伤
- 你的软件的稳定性对语言有要求, 运行时永远不会改变
- 你依赖的软件需要它

1.1.3 所以.... 3?

如果你选择了一种Python的解释器来用，你不是固执己见的人，我推荐你用最新的Python 3.x，因为每个版本都带来了新的改进的标准库模块，安全性和bug修复。

鉴于这样，如果你有一个强有力的理由只用Python 2，比如Python 3无法足够替代的Python 2特有库，或者你（像我）非常喜欢，受Python 2启发。

查看[Can I Use Python 3?](#)来看看是否有你依赖的软件阻止你用Python 3。

延伸阅读

[写同时能够兼容Python 2.6, 2.7, 和3.3上工作的代码](#)是可能的。这包括从简单到困难的各种难度，这取决于你写的软件的类型；如果你是初学者，其实有更重要的东西要操心。

1.1.4 实现方式

当人们谈论起Python，他们往往意味着的不仅是语言本身，还包括其CPython实现。Python实际上是一个语言规范，可以用许多不同的方式来实现语言。

CPython

[CPython](#)是Python的参考实现，用C编写的。它把Python代码编译成中间态的字节码，然后由虚拟机解释。CPython为Python包和C扩展模块提供了最大限度的兼容。

如果你正在写开源的Python代码，并希望有尽可能广泛的用户，用CPython是最好的。用依赖于C扩展的包，CPython是你唯一的选择。

所有版本的Python语言都用C实现，因为CPython是参考实现。

PyPy

[PyPy](#)是用RPython实现的解释器，RPython是Python的子集，具有静态类型这个解释器的特点是即时编译，支持多重后端（C, CLI, JVM）。

PyPy旨在最大兼容性（参考CPython的实现），同时提高性能。

如果你正在寻找提高你的Python代码的性能方法，值得试一试PyPy。在一套的基准测试下，它比CPython的速度目前超过5倍。

PyPy支持Python 2.7。[PyPy3](#)，发布的测试版，支持Python 3。

Jython

[Jython](#)是一个将Python代码编译成Java字节码的实现，运行在JVM(Java Virtual Machine)上。另外，它可以导入并用任何Java类就像Python模块一样。

如果你需要与现有的Java代码库对接或者其他原因需要为JVM编写Python代码，那Jython是最好的选择。

Jython现在支持到Python 2.5。[\[2\]](#)

IronPython

[IronPython](#) 是一个针对 .NET framework的Python实现。它可以用Python和.NET framework的库，而且可以用.NET framework暴露Python代码给.NET框架中的其他语言。

[Python Tools for Visual Studio](#) 直接集成IronPython到Visual Studio开发环境中，使之成为Windows开发者的理想选择。

IronPython支持Python 2.7。[\[3\]](#)

PythonNet

[Python for .NET](#)是一个包，它提供给本机已安装的Python一个.NET公共语言运行时（CLR），接近无缝集成。这所采取是与IronPython（见上文）中相反的方式，比相互竞争要更互补些。

与Mono相结合，PythonNet能使非windows操作系统的原生的Python在.NET框架中操作，比如OS X和Linux。它可以在除外IronPython的环境中无冲突运行。

PythonNet支持Python 2.3到2.7.[\[4\]](#)

1 赞 1 收藏 [1 评论](#)

关于作者：[mtunique](#)



微博：@孟涛_hustGithub:mtunique个人网站：mtunique.com [个人主页](#) · [我的文章](#) · 12

机器学习之用Python从零实现贝叶斯分类器

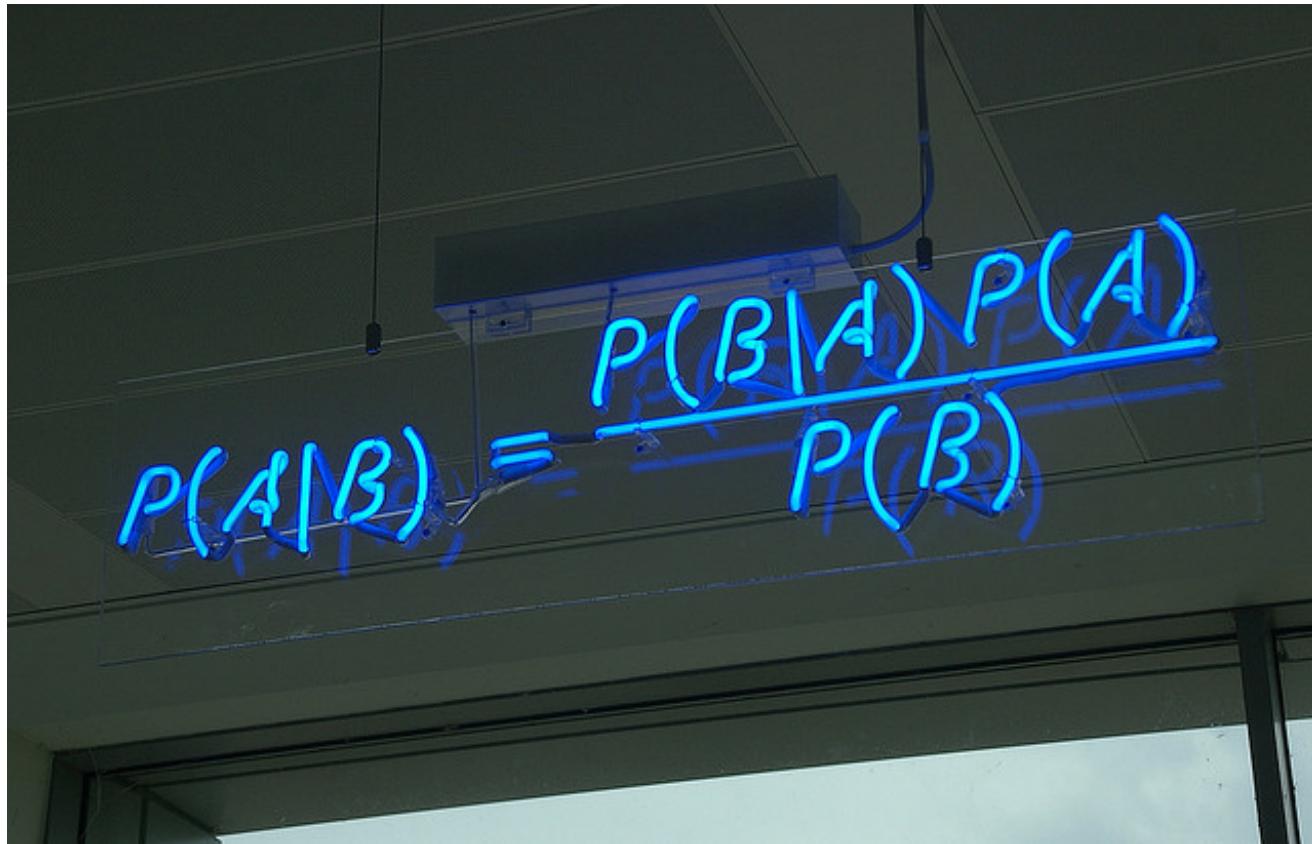
本文由 [伯乐在线 - Angus 翻译](#), [toolate 校稿](#)。未经许可, 禁止转载!
英文出处: machinelearningmastery.com。欢迎加入[翻译组](#)。

机器学习之用Python从零实现贝叶斯分类器

朴素贝叶斯算法简单高效，在处理分类问题上，是应该首先考虑的方法之一。

通过本教程，你将学到朴素贝叶斯算法的原理和Python版本的逐步实现。

更新：查看后续的关于朴素贝叶斯使用技巧的文章“[Better Naive Bayes: 12 Tips To Get The Most From The Naive Bayes Algorithm](#)”



朴素贝叶斯分类器, [Matt Buck](#)保留部分版权

关于朴素贝叶斯

朴素贝叶斯算法是一个直观的方法，使用每个属性归属于某个类的概率来做预测。你可以使用这种监督性学习方法，对一个预测性建模问题进行概率建模。

给定一个类，朴素贝叶斯假设每个属性归属于此类的概率独立于其余所有属性，从而简化了概率的计算。这种强假定产生了一个快速、有效的方法。

给定一个属性值，其属于某个类的概率叫做条件概率。对于一个给定的类值，将每个属性的条件概率

相乘，便得到一个数据样本属于某个类的概率。

我们可以通过计算样本归属于每个类的概率，然后选择具有最高概率的类来做预测。

通常，我们使用分类数据来描述朴素贝叶斯，因为这样容易通过比率来描述、计算。一个符合我们目的、比较有用的算法需要支持数值属性，同时假设每一个数值属性服从正态分布（分布在钟形曲线上），这又是一个强假设，但是依然能够给出一个健壮的结果。

预测糖尿病的发生

本文使用的测试问题是“皮马印第安人糖尿病问题”。

这个问题包括768个对于皮马印第安患者的医疗观测细节，记录所描述的瞬时测量取自诸如患者的年纪，怀孕和血液检查的次数。所有患者都是21岁以上（含21岁）的女性，所有属性都是数值型，而且属性的单位各不相同。

每一个记录归属于一个类，这个类指明以测量时间为止，患者是否是在5年之内感染的糖尿病。如果是，则为1，否则为0。

机器学习文献中已经多次研究了这个标准数据集，好的预测精度为70%-76%。

下面是[pima-indians.data.csv](#)文件中的一个样本，了解一下我们将要使用的数据。

注意：下载[文件](#)，然后以.csv扩展名保存（如：pima-indians-diabetes.data.csv）。查看[文件](#)中所有属性的描述。

Python

```
6,148,72,35,0,33.6,0.62
7,50,1
1 6,148,72,35,0,33.6,0.627,50,1
2 1,85,66,29,0,26.6,0.351,31,0
3 8,183,64,0,0,23.3,0.672,32,1
4 1,89,66,23,94,28.1,0.167,21,0
5 0,137,40,35,168,43.1,2.288,33,1
```

朴素贝叶斯算法教程

教程分为如下几步：

1.处理数据：从CSV文件中载入数据，然后划分为训练集和测试集。

2.提取数据特征：提取训练数据集的属性特征，以便我们计算概率并做出预测。

3.单一预测：使用数据集的特征生成单个预测。

4.多重预测：基于给定测试数据集和一个已提取特征的训练数据集生成预测。

5.评估精度：评估对于测试数据集的预测精度作为预测正确率。

6.合并代码：使用所有代码呈现一个完整的、独立的朴素贝叶斯算法的实现。

1.处理数据

首先加载数据文件。CSV格式的数据没有标题行和任何引号。我们可以使用csv模块中的open函数打开文件，使用reader函数读取行数据。

我们还需要将以字符串类型加载进来属性转换为我们可以使用的数字。下面是用来加载匹马印第安人数据集（Pima Indians dataset）的loadCsv()函数。

Python

```
import csv
def loadCsv(filename):
    import csv
    def loadCsv(filename):
        lines = csv.reader(open(filename, "rb"))
        dataset = list(lines)
        for i in range(len(dataset)):
            dataset[i] = [float(x) for x in dataset[i]]
        return dataset
```

我们可以通过加载皮马印第安人数据集，然后打印出数据样本的个数，以此测试这个函数。

Python

```
filename = 'pima-
indians-
diabetes.data.csv'
dataset = loadCsv(filename)
print('Loaded data file {0} with {1} rows'.format(filename, len(dataset)))
```

运行测试，你会看到如下结果：

Python

```
Loaded data file
iris.data.csv with 150 rows
1 Loaded data file iris.data.csv with 150 rows
```

下一步，我们将数据分为用于朴素贝叶斯预测的训练数据集，以及用来评估模型精度的测试数据集。我们需要将数据集随机分为包含67%的训练集合和包含33%的测试集（这是在此数据集上测试算法的通常比率）。

下面是splitDataset()函数，它以给定的划分比例将数据集进行划分。

Python

```
import random
def splitDataset(dataset,
splitRatio):
    import random
    def splitDataset(dataset, splitRatio):
        trainSize = int(len(dataset) * splitRatio)
```

```
4 trainSet = []
5 copy = list(dataset)
6 while len(trainSet) < trainSize:
7     index = random.randrange(len(copy))
8     trainSet.append(copy.pop(index))
9 return [trainSet, copy]
```

我们可以定义一个具有5个样例的数据集来进行测试，首先它分为训练数据集和测试数据集，然后打印出来，看看每个数据样本最终落在哪个数据集。

Python

```
dataset = [[1], [2], [3],
[4], [5]]
1 dataset = [[1], [2], [3], [4], [5]]
2 splitRatio = 0.67
3 train, test = splitDataset(dataset, splitRatio)
4 print('Split {} rows into train with {} and test with {}'.format(len(dataset), train, test))
```

运行测试，你会看到如下结果：

Python

```
Split 5 rows into train
with [[4], [3], [5]] and
1 Split 5 rows into train with [[4], [3], [5]] and test with [[1], [2]]
```

提取数据特征

朴素贝叶斯模型包含训练数据集中数据的特征，然后使用这个数据特征来做预测。

所收集的训练数据的特征，包含相对于每个类的每个属性的均值和标准差。举例来说，如果有2个类和7个数值属性，然后我们需要每一个属性（7）和类（2）的组合的均值和标准差，也就是14个属性特征。

在对特定的属性归属于每个类的概率做计算、预测时，将用到这些特征。

我们将数据特征的获取划分为以下的子任务：

1. 按类别划分数据
2. 计算均值
3. 计算标准差
4. 提取数据集特征
5. 按类别提取属性特征

按类别划分数据

首先将训练数据集中的样本按照类别进行划分，然后计算出每个类的统计数据。我们可以创建一个类别到属于此类别的样本列表的映射，并将整个数据集中的样本分类到相应的列表。

下面的**SeparateByClass()**函数可以完成这个任务：

Python

```
def separateByClass(datas):
    1 def separateByClass(dataset):
    2     separated = {}
    3     for i in range(len(dataset)):
    4         vector = dataset[i]
    5         if (vector[-1] not in separated):
    6             separated[vector[-1]] = []
    7             separated[vector[-1]].append(vector)
    8     return separated
```

可以看出，函数假设样本中最后一个属性（-1）为类别值，返回一个类别值到数据样本列表的映射。

我们可以用一些样本数据测试如下：

Python

```
dataset = [[1,20,1],
           [2,21,0], [3,22,1]]
1 dataset = [[1,20,1], [2,21,0], [3,22,1]]
2 separated = separateByClass(dataset)
3 print('Separated instances: {0}'.format(separated))
```

运行测试，你会看到如下结果：

Python

```
Separated instances:
{0: [[2, 21, 0]], 1: [[1, 20, 1], [3, 22, 1]]}
1 Separated instances: {0: [[2, 21, 0]], 1: [[1, 20, 1], [3, 22, 1]]}
```

计算均值

我们需要计算在每个类中每个属性的均值。均值是数据的中点或者集中趋势，在计算概率时，我们用它作为高斯分布的中值。

我们还需要计算每个类中每个属性的标准差。标准差描述了数据散布的偏差，在计算概率时，我们用它来刻画高斯分布中，每个属性所期望的散布。

标准差是方差的平方根。方差是每个属性值与均值的离差平方的平均数。注意我们使用N-1的方法
（译者注：参见无偏估计），也就是在计算方差时，属性值的个数减1。

Python

```
import math
def mean(numbers):
    1 import math
    2 def mean(numbers):
    3     return sum(numbers)/float(len(numbers))
    4
    5 def stdev(numbers):
    6     avg = mean(numbers)
```

```
7 variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
8 return math.sqrt(variance)
```

通过计算从1到5这5个数的均值来测试函数。

Python

```
numbers = [1,2,3,4,5]
print('Summary of {0}:
1 numbers = [1,2,3,4,5]
2 print('Summary of {0}: mean={1}, stdev={2}').format(numbers, mean(numbers), stdev(numbers))
```

运行测试，你会看到如下结果：

Python

```
Summary of [1, 2, 3, 4,
5]: mean=3.0,
1 Summary of [1, 2, 3, 4, 5]: mean=3.0, stdev=1.58113883008
```

提取数据集的特征

现在我们可以提取数据集特征。对于一个给定的样本列表（对应于某个类），我们可以计算每个属性的均值和标准差。

zip函数将数据样本按照属性分组为一个个列表，然后可以对每个属性计算均值和标准差。

Python

```
def summarize(dataset):
    summaries =
1 def summarize(dataset):
2     summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
3     del summaries[-1]
4     return summaries
```

我们可以使用一些测试数据来测试这个summarize()函数，测试数据对于第一个和第二个数据属性的均值和标准差显示出显著的不同。

Python

```
dataset = [[1,20,0],
[2,21,1], [3,22,0]]
1 dataset = [[1,20,0], [2,21,1], [3,22,0]]
2 summary = summarize(dataset)
3 print('Attribute summaries: {0}'.format(summary))
```

运行测试，你会看到如下结果：

Python

```
Attribute summaries: [(2.0,
1.0), (21.0, 1.0)]
```

```
1 Attribute summaries: [(2.0, 1.0), (21.0, 1.0)]
```

按类别提取属性特征

合并代码，我们首先将训练数据集按照类别进行划分，然后计算每个属性的摘要。

Python

```
def summarizeByClass(dataset):
    separated = separateByClass(dataset)
    summaries = {}
    for classValue, instances in separated.iteritems():
        summaries[classValue] = summarize(instances)
    return summaries
```

使用小的测试数据集来测试summarizeByClass()函数。

Python

```
dataset = [[1,20,1],
           [2,21,0], [3,22,1],
           [4,22,0]]
summary = summarizeByClass(dataset)
print('Summary by class value: {0}'.format(summary))
```

运行测试，你会看到如下结果：

Python

```
Summary by class
1 Summary by class value:
2 {0: [(3.0, 1.4142135623730951), (21.5, 0.7071067811865476)],
3 1: [(2.0, 1.4142135623730951), (21.0, 1.4142135623730951)]}
```

预测

我们现在可以使用从训练数据中得到的摘要来做预测。做预测涉及到对于给定的数据样本，计算其归属于每个类的概率，然后选择具有最大概率的类作为预测结果。

我们可以将这部分划分成以下任务：

1. 计算高斯概率密度函数
2. 计算对应类的概率
3. 单一预测
4. 评估精度

计算高斯概率密度函数

给定来自训练数据中已知属性的均值和标准差，我们可以使用高斯函数来评估一个给定的属性值的概

率。

已知每个属性和类值的属性特征，在给定类值的条件下，可以得到给定属性值的条件概率。

关于高斯概率密度函数，可以查看参考文献。总之，我们要把已知的细节融入到高斯函数（属性值，均值，标准差），并得到属性值归属于某个类的似然（译者注：即可能性）。

在**calculateProbability()**函数中，我们首先计算指数部分，然后计算等式的主干。这样可以将其很好地组织成2行。

Python

```
import math
def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
```

使用一些简单的数据测试如下：

Python

```
x = 71.5
mean = 73
stdev = 6.2
probability = calculateProbability(x, mean, stdev)
print('Probability of belonging to this class: {0}'.format(probability))
```

运行测试，你会看到如下结果：

Python

```
Probability of belonging to this class:
1 Probability of belonging to this class: 0.0624896575937
```

计算所属类的概率

既然我们可以计算一个属性属于某个类的概率，那么合并一个数据样本中所有属性的概率，最后便得到整个数据样本属于某个类的概率。

使用乘法合并概率，在下面的**calculClassProbabilities()**函数中，给定一个数据样本，它所属每个类别的概率，可以通过将其属性概率相乘得到。结果是一个类值到概率的映射。

Python

```
def calculateClassProbabilities(summaries, inputVector):
    1 def calculateClassProbabilities(summaries, inputVector):
```

```
2 probabilities = {}
3 for classValue, classSummaries in summaries.iteritems():
4     probabilities[classValue] = 1
5 for i in range(len(classSummaries)):
6     mean, stdev = classSummaries[i]
7     x = inputVector[i]
8     probabilities[classValue] *= calculateProbability(x, mean, stdev)
9 return probabilities
```

测试**calculateClassProbabilities()**函数。

Python

```
summaries = {0:[(1, 0.5)], 1:[(20, 5.0)]}
1 summaries = {0:[(1, 0.5)], 1:[(20, 5.0)]}
2 inputVector = [1.1, '?']
3 probabilities = calculateClassProbabilities(summaries, inputVector)
4 print('Probabilities for each class: {0}'.format(probabilities))
```

运行测试，你会看到如下结果：

Python

```
Probabilities for each
class: {0:
1 Probabilities for each class: {0: 0.7820853879509118, 1: 6.298736258150442e-05}
```

单一预测

既然可以计算一个数据样本属于每个类的概率，那么我们可以找到最大的概率值，并返回关联的类。

下面的**predict()**函数可以完成以上任务。

Python

```
def predict(summaries, inputVector):
1 def predict(summaries, inputVector):
2     probabilities = calculateClassProbabilities(summaries, inputVector)
3     bestLabel, bestProb = None, -1
4     for classValue, probability in probabilities.iteritems():
5         if bestLabel is None or probability > bestProb:
6             bestProb = probability
7             bestLabel = classValue
8     return bestLabel
```

测试**predict()**函数如下：

Python

```
summaries = {'A':[(1, 0.5)], 'B':[(20, 5.0)]}
1 summaries = {'A':[(1, 0.5)], 'B':[(20, 5.0)]}
2 inputVector = [1.1, '?']
```

```
3 result = predict(summaries, inputVector)
4 print('Prediction: {0}'.format(result))
```

运行测试，你会得到如下结果：

Python

```
Prediction: A
```

```
1 Prediction: A
```

多重预测

最后，通过对测试数据集中每个数据样本的预测，我们可以评估模型精度。**getPredictions()**函数可以实现这个功能，并返回每个测试样本的预测列表。

Python

```
def getPredictions(summar
1 def getPredictions(summaries, testSet):
2 predictions = []
3 for i in range(len(testSet)):
4 result = predict(summaries, testSet[i])
5 predictions.append(result)
6 return predictions
```

测试**getPredictions()**函数如下。

Python

```
summaries = {'A':[(1,
1 summaries = {'A':[(1, 0.5)], 'B':[(20, 5.0)]}
2 testSet = [[1.1, '?'], [19.1, '?']]
3 predictions = getPredictions(summaries, testSet)
4 print('Predictions: {0}'.format(predictions))
```

运行测试，你会看到如下结果：

Python

```
Predictions: ['A', 'B']
```

```
1 Predictions: ['A', 'B']
```

计算精度

预测值和测试数据集中的类别值进行比较，可以计算得到一个介于0%~100%精确率作为分类的精确度。**getAccuracy()**函数可以计算出这个精确率。

Python

```
def  
getAccuracy(testSet,  
1 def getAccuracy(testSet, predictions):  
2 correct = 0  
3 for x in range(len(testSet)):  
4 if testSet[x][-1] == predictions[x]:  
5 correct += 1  
6 return (correct/float(len(testSet))) * 100.0
```

我们可以使用如下简单的代码来测试**getAccuracy()**函数。

Python

```
testSet = [[1,1,1,'a'],  
[2,2,2,'a'], [3,3,3,'b']]  
1 testSet = [[1,1,1,'a'], [2,2,2,'a'], [3,3,3,'b']]  
2 predictions = ['a', 'a', 'a']  
3 accuracy = getAccuracy(testSet, predictions)  
4 print('Accuracy: {0}'.format(accuracy))
```

运行测试，你会得到如下结果：

Python

```
Accuracy: 66.6666666667  
1 Accuracy: 66.6666666667
```

合并代码

最后，我们需要将代码连贯起来。

下面是朴素贝叶斯Python版的逐步实现的全部代码。

Python

```
# Example of Naive  
Bayes implemented  
1 # Example of Naive Bayes implemented from Scratch in Python  
2 import csv  
3 import random  
4 import math  
5  
6 def loadCsv(filename):  
7 lines = csv.reader(open(filename, "rb"))  
8 dataset = list(lines)  
9 for i in range(len(dataset)):  
10 dataset[i] = [float(x) for x in dataset[i]]  
11 return dataset  
12  
13 def splitDataset(dataset, splitRatio):  
14 trainSize = int(len(dataset) * splitRatio)  
15 trainSet = []  
16 copy = list(dataset)  
17 while len(trainSet) < trainSize:  
18 index = random.randrange(len(copy))
```

```

19 trainSet.append(copy.pop(index))
20 return [trainSet, copy]
21
22 def separateByClass(dataset):
23     separated = {}
24     for i in range(len(dataset)):
25         vector = dataset[i]
26         if (vector[-1] not in separated):
27             separated[vector[-1]] = []
28             separated[vector[-1]].append(vector)
29     return separated
30
31 def mean(numbers):
32     return sum(numbers)/float(len(numbers))
33
34 def stdev(numbers):
35     avg = mean(numbers)
36     variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
37     return math.sqrt(variance)
38
39 def summarize(dataset):
40     summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
41     del summaries[-1]
42     return summaries
43
44 def summarizeByClass(dataset):
45     separated = separateByClass(dataset)
46     summaries = {}
47     for classValue, instances in separated.items():
48         summaries[classValue] = summarize(instances)
49     return summaries
50
51 def calculateProbability(x, mean, stdev):
52     exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
53     return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
54
55 def calculateClassProbabilities(summaries, inputVector):
56     probabilities = {}
57     for classValue, classSummaries in summaries.items():
58         probabilities[classValue] = 1
59         for i in range(len(classSummaries)):
60             mean, stdev = classSummaries[i]
61             x = inputVector[i]
62             probabilities[classValue] *= calculateProbability(x, mean, stdev)
63     return probabilities
64
65 def predict(summaries, inputVector):
66     probabilities = calculateClassProbabilities(summaries, inputVector)
67     bestLabel, bestProb = None, -1
68     for classValue, probability in probabilities.items():
69         if bestLabel is None or probability > bestProb:
70             bestProb = probability
71             bestLabel = classValue
72     return bestLabel
73
74 def getPredictions(summaries, testSet):
75     predictions = []
76     for i in range(len(testSet)):
77         result = predict(summaries, testSet[i])
78         predictions.append(result)
79     return predictions
80
81 def getAccuracy(testSet, predictions):

```

```

82 correct = 0
83 for i in range(len(testSet)):
84     if testSet[i][-1] == predictions[i]:
85         correct += 1
86 return (correct/float(len(testSet))) * 100.0
87
88 def main():
89     filename = 'pima-indians-diabetes.data.csv'
90     splitRatio = 0.67
91     dataset = loadCsv(filename)
92     trainingSet, testSet = splitDataset(dataset, splitRatio)
93     print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))
94     # prepare model
95     summaries = summarizeByClass(trainingSet)
96     # test model
97     predictions = getPredictions(summaries, testSet)
98     accuracy = getAccuracy(testSet, predictions)
99     print('Accuracy: {0}%'.format(accuracy))
100
101 main()

```

运行示例，得到如下输出：

Python

Split 768 rows into
train=514 and test=254

1 Split 768 rows into train=514 and test=254 rows
2 Accuracy: 76.3779527559%

实现扩展

这一部分为你提供了扩展思路，你可以将其作为教程的一部分，使用你已经实现的Python代码，进行应用研究。

到此，你已经使用Python一步步完成了高斯版本的朴素贝叶斯。

你可以进一步扩展算法实现：

计算所归属类的概率：将一个数据样本归属于每个类的概率更新为一个比率。计算上就是将一个样本数据归属于某个类的概率，比上其归属于每一个类的概率的和。举例来说，一个样本属于类A的概率时0.02，属于类B的概率时0.001，那么样本属于类A的可能性是 $(0.02/(0.02+0.001))*100$ 大约为95.23%。

对数概率：对于一个给定的属性值，每个类的条件概率很小。当将其相乘时结果会更小，那么存在浮点溢出的可能（数值太小，以至于在Python中不能表示）。一个常用的修复方案是，合并其概率的对数值。可以研究实现下这个改进。

名词属性：改进算法实现，使其支持名词属性。这是十分相似的，你所收集的每个属性的摘要信息是对于每个类的类别值的比率。潜心学习参考文献来获取更多信息。

不同的密度函数（伯努利或者多项式）：我们已经尝试了高斯朴素贝叶斯，你也可以尝试下其他分布。实现一个不同的分布诸如多项分布、伯努利分布或者内核朴素贝叶斯，他们对于属性值的分布

和/或与类值之间的关系有不同的假设。

学习资源及深入阅读

这一部分提供了一些用于学习更多朴素贝叶斯算法的资源，包括算法理论和工作原理，以及代码实现中的实际问题。

问题

更多学习预测糖尿病发作问题的资源

- [Pima Indians Diabetes Data Set](#):这个页面提供数据集文件，同时描述了各个属性，也列出了使用该数据集的论文列表
- [Dataset File](#):数据集文件
- [Dataset Summary](#):数据集属性的描述
- [Diabetes Dataset Results](#):许多标准算法在该数据集上的精度

代码

这一部分包含流行的机器学习库中的朴素贝叶斯的开源实现。如果你在考虑实现自己的用于实际使用的版本，可以查阅这些

- [Naive Bayes in Scikit-Learn](#):scikit-learn库中朴素贝叶斯的实现
- [Naive Bayes documentation](#):scikit-learn库中关于朴素贝叶斯的文档和样例代码
- [Simple Naive Bayes in Weka](#):朴素贝叶斯的Weka实现

书籍

你应该有几本机器学习应用的书籍。这一部分高亮出了常用机器学习书籍中关于朴素贝叶斯的章节。

- [Applied Predictive Modeling](#), page 353
- [Data Mining: Practical Machine Learning Tools and Techniques](#), page 94
- [Machine Learning for Hackers](#), page 78
- [An Introduction to Statistical Learning: with Applications in R](#), page 138
- [Machine Learning: An Algorithmic Perspective](#), page 171
- [Machine Learning in Action](#), page 61 (Chapter 4)
- [Machine Learning](#), page 177 (chapter 6)

下一步

行动起来。

跟着教程，从头开始实现朴素贝叶斯。将这个例子适用到其他问题。按照扩展改进实现。

评论分享你的经验。

更新：查看后续的关于朴素贝叶斯使用技巧的文章“Better Naive Bayes: 12 Tips To Get The Most From The Naive Bayes Algorithm”

2 赞 10 收藏 [13 评论](#)

关于作者： [Angus](#)



什么都不想说；新浪微博：@yannpzhao [个人主页](#) · [我的文章](#) · 12

Y分钟学会Python

原文出处：[Learn Python in Y Minutes](#) 译文出处：[夏永锋](#)

Python由Guido Van Rossum发明于90年代初期，是目前最流行的编程语言之一，因其语法的清晰简洁我爱上了Python，其代码基本上可以说是可执行的伪代码。

非常欢迎反馈！你可以通过推特[@louiedinh](#)或louiedinh AT gmail联系我。

备注：本文是专门针对Python 2.7的，但应该是适用于Python 2.x的。很快我也会为Python 3写这样的一篇文章！

Python

```
# 单行注释以井字符开头
1 # 单行注释以井字符开头
2 """ 我们可以使用三个双引号 ("") 或单引号 ('')
3 来编写多行注释
4 """
5
6 #####
7 ## 1. 基本数据类型和操作符
8 #####
9
10 # 数字
11 3 #=> 3
12
13 # 你预想的数学运算
14 1 + 1 #=> 2
15 8 - 1 #=> 7
16 10 * 2 #=> 20
17 35 / 5 #=> 7
18
19 # 除法略显诡异。整数相除会自动向下取小于结果的最大整数
20 11 / 4 #=> 2
21
22 # 还有浮点数和浮点数除法（译注：除数和被除数两者至少一个为浮点数，结果才会是浮点数）
23 2.0 # 这是一个浮点数
24 5.0 / 2.0 #=> 2.5 额...语法更明确一些
25
26 # 使用括号来强制优先级
27 (1 + 3) * 2 #=> 8
28
29 # 布尔值也是基本类型数据
30 True
31 False
32
33 # 使用not来求反
34 not True #=> False
35 not False #=> True
36
37 # 相等比较使用 ==
38 1 == 1 #=> True
39 2 == 1 #=> False
40
41 # 不相等比较使用 !=
42 1 != 1 #=> False
43 2 != 1 #=> True
44
45 # 逻辑与使用 and
46 1 and 2 #=> 1
47 1 and 0 #=> 0
48
49 # 逻辑或使用 or
50 1 or 2 #=> 1
51 0 or 2 #=> 1
52
53 # 逻辑非使用 not
54 not 1 #=> False
55 not 0 #=> True
56
57 # 循环语句
58 for i in range(10):
59     print(i)
60
61 # if语句
62 if 1 > 0:
63     print("1 > 0")
64
65 # else语句
66 if 1 > 0:
67     print("1 > 0")
68 else:
69     print("1 <= 0")
70
71 # elif语句
72 if 1 > 0:
73     print("1 > 0")
74 elif 1 < 0:
75     print("1 < 0")
76 else:
77     print("1 == 0")
78
79 # while语句
80 i = 0
81 while i < 10:
82     print(i)
83     i += 1
84
85 # break语句
86 for i in range(10):
87     if i == 5:
88         break
89     print(i)
90
91 # continue语句
92 for i in range(10):
93     if i == 5:
94         continue
95     print(i)
96
97 # pass语句
98 for i in range(10):
99     pass
100
101 # 函数
102 def add(a, b):
103     return a + b
104
105 # 类
106 class Point:
107     def __init__(self, x, y):
108         self.x = x
109         self.y = y
110
111     def move(self, dx, dy):
112         self.x += dx
113         self.y += dy
114
115     def __str__(self):
116         return "Point(%d, %d)" % (self.x, self.y)
117
118 # 列表推导式
119 points = [Point(x, y) for x in range(10) for y in range(10)]
120
121 # 字典推导式
122 squares = {x: x*x for x in range(10)}
123
124 # set推导式
125 unique_numbers = {x for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}  
126
127 # 生成器表达式
128 even_numbers = (x for x in range(10) if x % 2 == 0)
```

```
42 1 != 1 #=> False
43 2 != 1 #=> True
44
45 # 更多的比较方式
46 1 < 10 #=> True
47 1 > 10 #=> False
48 2 <= 2 #=> True
49 2 >= 2 #=> True
50
51 # 比较操作可以串接!
52 1 < 2 < 3 #=> True
53 2 < 3 < 2 #=> False
54
55 # 可以使用"或'创建字符串
56 "This is a string."
57 'This is also a string.'
58
59 # 字符串也可以相加!
60 "Hello " + "world!" #=> "Hello world!"
61
62 # 字符串可以看作是一个字符列表
63 "This is a string"[0] #=> 'T'
64
65 # None是一个对象
66 None #=> None
67
68 #####2. 变量与数据容器#####
69 ## 2. 变量与数据容器
70 #####
71
72 # 打印输出非常简单
73 print "I'm Python. Nice to meet you!"
74
75 # 赋值之前不需要声明变量
76 some_var = 5 # 约定使用 小写_字母_和_下划线 的命名方式
77 some_var #=> 5
78
79 # 访问之前未赋值的变量会产生一个异常
80 try:
81     some_other_var
82 except NameError:
83     print "Raises a name error"
84
85 # 赋值时可以使用条件表达式
86 some_var = a if a > b else b
87 # 如果a大于b，则将a赋给some_var,
88 # 否则将b赋给some_var
89
90 # 列表用于存储数据序列
91 li = []
92 # 你可以一个预先填充的列表开始
93 other_li = [4, 5, 6]
94
95 # 使用append将数据添加到列表的末尾
96 li.append(1) #li现在为[1]
97 li.append(2) #li现在为[1, 2]
98 li.append(4) #li现在为[1, 2, 4]
99 li.append(3) #li现在为[1, 2, 4, 3]
100
101 # 使用pop从列表末尾删除数据
102 li.pop() #=> 3, li现在为[1, 2, 4]
103 # 把刚刚删除的数据存回来
104 li.append(3) # 现在li再一次为[1, 2, 4, 3]
```

```
105
106 # 像访问数组一样访问列表
107 li[0] #=> 1
108 # 看看最后一个元素
109 li[-1] #=> 3
110
111 # 越界访问会产生一个IndexError
112 try:
113     li[4] # 抛出一个IndexError异常
114 except IndexError:
115     print "Raises an IndexError"
116
117 # 可以通过分片(slice)语法来查看列表中某个区间的数据
118 # 以数学角度来说，这是一个闭合/开放区间
119 li[1:3] #=> [2, 4]
120 # 省略结束位置
121 li[2:] #=> [4, 3]
122 # 省略开始位置
123 li[:3] #=> [1, 2, 4]
124
125 # 使用del从列表中删除任意元素
126 del li[2] #li现在为[1, 2, 3]
127
128 # 列表可以相加
129 li + other_li #=> [1, 3, 3, 4, 5, 6] - 注意：li和other_li并未改变
130
131 # 以extend来连结列表
132 li.extend(other_li) # 现在li为[1, 2, 3, 4, 5, 6]
133
134 # 以in来检测列表中是否存在某元素
135 1 in li #=> True
136
137 # 以len函数来检测列表长度
138 len(li) #=> 6
139
140 # 元组类似列表，但不可变
141 tup = (1, 2, 3)
142 tup[0] #=> 1
143 try:
144     tup[0] = 3 # 抛出一个TypeError异常
145 except TypeError:
146     print "Tuples cannot be mutated."
147
148 # 可以在元组上使用和列表一样的操作
149 len(tup) #=> 3
150 tup + (4, 5, 6) #=> (1, 2, 3, 4, 5, 6)
151 tup[:2] #=> (1, 2)
152 2 in tup #=> True
153
154 # 可以将元组解包到变量
155 a, b, c = (1, 2, 3) # 现在a等于1, b等于2, c等于3
156 # 如果你省略括号，默认也会创建元组
157 d, e, f = 4, 5, 6
158 # 看看两个变量互换值有多简单
159 e, d = d, e #现在d为5, e为4
160
161 # 字典存储映射关系
162 empty_dict = {}
163 # 这是一个预先填充的字典
164 filled_dict = {"one": 1, "two": 2, "three": 3}
165
166 # 以[]语法查找值
167 filled_dict['one'] #=> 1
```

```
168
169 # 以列表形式获取所有的键
170 filled_dict.keys() #=> ["three", "two", "one"]
171 # 注意 - 字典键的顺序是不确定的
172 # 你的结果也许和上面的输出结果并不一致
173
174 # 以in来检测字典中是否存在某个键
175 "one" in filled_dict #=> True
176 1 in filled_dict #=> False
177
178 # 试图使用某个不存在的键会抛出一个KeyError异常
179 filled_dict['four'] #=> 抛出KeyError异常
180
181 # 使用get方法来避免KeyError
182 filled_dict.get("one") #=> 1
183 filled_dict.get("four") #=> None
184
185 # get方法支持一个默认参数，不存在某个键时返回该默认参数值
186 filled_dict.get("one", 4) #=> 1
187 filled_dict.get("four", 4) #=> 4
188
189 # setdefault方法是一种添加新的键-值对到字典的安全方式
190 filled_dict.setdefault("five", 5) #filled_dict["five"]设置为5
191 filled_dict.setdefault("five", 6) #filled_dict["five"]仍为5
192
193 # 集合
194 empty_set = set()
195 # 以几个值初始化一个集合
196 filled_set = set([1, 2, 2, 3, 4]) # filled_set现为set([1, 2, 3, 4, 5])
197
198 # 以&执行集合交运算
199 other_set = set([3, 4, 5, 6])
200 filled_set & other_set #=> set([3, 4, 5])
201 # 以|执行集合并运算
202 filled_set | other_set #=> set([1, 2, 3, 4, 5, 6])
203 # 以-执行集合差运算
204 set([1, 2, 3, 4]) - set([2, 3, 5]) #=> set([1, 4])
205
206 # 以in来检测集合中是否存在某个值
207 2 in filled_set #=> True
208 10 in filled_set #=> False
209
210 #####3. 控制流程#####
211 ## 3. 控制流程
212 #####
213
214 # 创建个变量
215 some_var = 5
216
217 # 以下是一个if语句。缩进在Python是有重要意义的。
218 # 打印 "some_var is smaller than 10"
219 if some_var > 10:
220     print "some_var is totally bigger than 10."
221 elif some_var < 10:
222     print "some_var is smaller than 10."
223 else:
224     print "some_var is indeed 10."
225
226 """
227 For循环在列表上迭代
228 输出：
229 dog is a mammal
230 cat is a mammal
```

```
231     mouse is a mammal
232 """
233 for animal in ["dog", "cat", "mouse"]:
234     # 可以使用%来插补格式化字符串
235     print "%s is a mammal" % animal
236
237 """
238 while循环直到未满足某个条件。
239 输出:
240     0
241     1
242     2
243     3
244 """
245 x = 0
246 while x < 4:
247     print x
248     x += 1    # x = x + 1的一种简写
249
250 # 使用try/except块来处理异常
251
252 # 对Python 2.6及以上版本有效
253 try:
254     # 使用raise来抛出一个错误
255     raise IndexError("This is an index error")
256 except IndexError as e:
257     pass    # pass就是什么都不干。通常这里用来做一些恢复工作
258
259 # 对于Python 2.7及以下版本有效
260 try:
261     raise IndexError("This is an index error")
262 except IndexError, e:    # 没有"as"，以逗号替代
263     pass
264
265 #####4. 函数#####
266 ## 4. 函数
267 #####
268
269 # 使用def来创建新函数
270 def add(x, y):
271     print "x is %s and y is %s" % (x, y)
272     return x + y    # 以一个return语句来返回值
273
274 # 以参数调用函数
275 add(5, 6) #=> 11 并输出 "x is 5 and y is 6"
276 # 另一种调用函数的方式是关键字参数
277 add(x=5, y=6) # 关键字参数可以任意顺序输入
278
279 # 可定义接受可变数量的位置参数的函数
280 def varargs(*args):
281     return args
282
283 varargs(1, 2, 3) #=> (1, 2, 3)
284
285 # 也可以定义接受可变数量关键字参数的函数
286 def keyword_args(**kwargs):
287     return kwargs
288
289 # 调用一下该函数看看会发生什么
290 keyword_args(big="foot", loch="ness") #=> {"big": "foo", "loch": "ness"}
291
292 # 也可以一次性接受两种参数
293 def all_the_args(*args, **kwargs):
```

```
294     print args
295     print kwargs
296 """
297 all_the_args(1, 2, a=3, b=4)输出:
298 [1, 2]
299 {"a": 3, "b": 4}
300 """
301
302 # 在调用一个函数时也可以使用*和**
303 args = (1, 2, 3, 4)
304 kwargs = {"a": 3, "b": 4}
305 foo(*args) #等价于foo(1, 2, 3, 4)
306 foo(**kwargs) # 等价于foo(a=3, b=4)
307 foo(*args, **kwargs) # 等价于foo(1, 2, 3, 4, a=3, b=4)
308
309 # Python的函数是一等函数
310 def create_adder(x):
311     def adder(y):
312         return x + y
313     return adder
314
315 add_10 = create_adder(10)
316 add_10(3) #=> 13
317
318 # 也有匿名函数
319 (lamda x: x > 2)(3) #=> True
320
321 # 有一些内置的高阶函数
322 map(add_10, [1, 2, 3]) #=> [11, 12, 13]
323 filter(lamda x: x > 5, [3, 4, 5, 6, 7]) #=>[6, 7]
324
325 # 可以使用列表推导来实现映射和过滤
326 [add_10(i) for i in [1, 2, 3]] #=> [11, 13, 13]
327 [x for x in [3, 4, 5, 6, 7] if x > 5] #=> [6, 7]
328 #####
329 ## 5. 类
330 #####
331 #####
332
333 # 创建一个子类继承自object来得到一个类
334 class Human(object):
335
336     # 类属性。在该类的所有示例之间共享
337     species = "H. sapiens"
338
339     # 基本初始化构造方法
340     def __init__(self, name):
341         # 将参数赋值给实例的name属性
342         self.name = name
343
344     # 实例方法。所有示例方法都以self为第一个参数
345     def say(self, msg):
346         return "%s: %s" % (self.name, msg)
347
348     # 类方法由所有实例共享
349     # 以调用类为第一个参数进行调用
350     @classmethod
351     def get_species(cls):
352         return cls.species
353
354     # 静态方法的调用不需要一个类或实例的引用
355     @staticmethod
356     def grunt():
```

```
357     return "*grunt*"
358
359 # 实例化一个类
360 i = Human(name="Ian")
361 print i.say("hi")      # 输出"Ian: hi"
362
363 j = Human("Joel")
364 print j.say("hello")    # 输出"Joel: hello"
365
366 # 调用类方法
367 i.get_species() #=> "H. sapiens"
368
369 # 修改共享属性
370 Human.species = "H. neanderthalensis"
371 i.get_species() #=> "H. neanderthalensis"
372 j.get_species() #=> "H. neanderthalensis"
373
374 # 调用静态方法
375 Human.grunt() #=> "*grunt*"
376 {%- endhighlight %}
```

进一步阅读

想要学习更多？试试[笨办法学习Python](#)。

1 赞 2 收藏 [3 评论](#)

用Pandas完成Excel中常见的任务（2）

本文由[伯乐在线 - 艾凌风](#)翻译, [Daetalus](#)校稿。未经许可, 禁止转载!
英文出处: [pbpython](#)。欢迎加入[翻译组](#)。

介绍

读者对于本系列[第一篇文章](#)的回应, 让我感到很兴奋。感谢大家正面的反馈。我想把本系列继续下去, 重点介绍其他的一些你经常使用Excel完成的任务, 并且展示给你如何在[pandas](#)中使用相同的功能。

在第一篇文章中, 我着重介绍了Excel中常见的数学计算工作, 以及在pandas如何完成这些工作。在本文中, 我们将着重介绍一些常见的选择和筛选任务, 并且介绍如何在pandas中完成同样的事情。

设置

如果您想要继续下去, 您可以下载本[excel文件](#)。

导入pandas和numpy模块。

Python

```
import pandas as pd
import numpy as np

1 import pandas as pd
2 import numpy as np
```

导入我们样本公司销售年销售额的Excel文件。

Python

```
df =
pd.read_excel("sample-
1 df = pd.read_excel("sample-salesv3.xlsx")
```

快速浏览一下数据类型, 以确保所以事情都能如预期一样运行。

Python

```
df.dtypes
1 df.dtypes
```

Python

```
account number int64
name object
1 account number int64
2 name object
3 sku object
4 quantity int64
5 unit price float64
6 ext price float64
7 date object
8 dtype: object
```

你会注意到, 我们的date列, 显示的是一个通用对象。我们准备把它转换为日期对象, 来简化将来会用到的一些

选择操作。

Python

```
df['date'] = pd.to_datetime(df['date'])  
1 df['date'] = pd.to_datetime(df['date'])  
2 df.head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

Python

```
df.dtypes  
1 df.dtypes
```

Python

```
account number int64  
name object  
1 account number int64  
2 name object  
3 sku object  
4 quantity int64  
5 unit price float64  
6 ext price float64  
7 date datetime64[ns]  
8 dtype: object
```

现在，data变成了一个datetime类型的对象，这对于将来的操作是很有用的。

筛选数据

我认为在Excel中最方便的功能是筛选。我想几乎每一次有人拿到一个任意大小的Excel文件，当他们想要筛选数据的时候，都会使用这个功能。

如图，对本数据集使用该功能：

	A	B	C	D	E	F	G	H
1	account number	name	sku	quantity	unit price	ext price	date	
2	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51	
3	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47	
4	218895	Kulas Inc	B1-69924	23	90.7	2086.1	2014-01-01 13:24:58	
5	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22	
6	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55	
7	714466	Trantow-Barrows	S2-77896	17	87.63	1489.71	2014-01-02 10:07:15	
8	218895	Kulas Inc	B1-65551	2	31.1	62.2	2014-01-02 10:57:23	
9	729833	Koeppe Ltd	S1-30248	8	33.25	266	2014-01-03 06:32:11	
10	714466	Trantow-Barrows	S1-50961	22	84.09	1849.98	2014-01-03 11:29:02	
11	737550	Fritsch, Russel and Anderson	S2-82423	14	81.92	1146.88	2014-01-03 19:07:37	
12	146832	Kiehn-Spinka	S2-82423	15	67.74	1016.1	2014-01-03 19:39:53	
13	688981	Keeling LLC	S2-00301	7	20.26	141.82	2014-01-04 00:02:36	

同Excel中的筛选功能一样，你可以使用pandas来筛选和选择某个特定数据的子集。

比方说，如果我们仅仅想查看一个特定的账号，我们可以简单是在Excel中完成，或是使用pandas完成操作。

下面是Excel的筛选解决方案：

	A	B	C	D	E	F	G	H
1	account number	name	sku	quantity	unit price	ext price	date	
5	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22	
15	307599	Kassulke, Ondricka and Metz	S2-10342	17	12.44	211.48	2014-01-04 07:53:01	
36	307599	Kassulke, Ondricka and Metz	S2-78676	35	33.04	1156.4	2014-01-10 05:26:31	
60	307599	Kassulke, Ondricka and Metz	B1-20000	22	37.87	833.14	2014-01-15 16:22:22	
72	307599	Kassulke, Ondricka and Metz	S2-10342	44	96.79	4258.76	2014-01-18 06:32:31	

在pandas中执行相关操作比Excel中更加直观。注意，我将会使用head函数来显示前面几个结果。这仅仅是为了让本文保持简短。

Python

```
df[df["account  
number"]==307599].head()  
1 df[df["account number"]==307599].head()
```

你还可以以数值为基准来进行筛选。我就不再举任何Excel的例子了。我相信你能明白。

Python

```
df[df["quantity"] >  
22].head()  
1 df[df["quantity"] > 22].head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
14	737550	Fritsch, Russel and Anderson	B1-53102	23	71.56	1645.88	2014-01-04 08:57:48
15	239344	Stokes LLC	S1-06532	34	71.51	2431.34	2014-01-04 11:34:58

如果我们想要更多复杂的筛选，我们可以使用map来以多重标准进行筛选。在这个例子中，从B1中查找以“sku”中起始的项目。

Python

```
df[df["sku"].map(lambda x:  
x.startswith('B1'))].head()  
1 df[df["sku"].map(lambda x: x.startswith('B1'))].head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
6	218895	Kulas Inc	B1-65551	2	31.10	62.20	2014-01-02 10:57:23
14	737550	Fritsch, Russel and Anderson	B1-53102	23	71.56	1645.88	2014-01-04 08:57:48
17	239344	Stokes LLC	B1-50809	14	16.23	227.22	2014-01-04 22:14:32

把两个或更多的语句连接起来很简单，用&就可以。

Python

```
df[df["sku"].map(lambda x:  
x.startswith('B1')) &  
(df["quantity"] > 22)].head()  
1 df[df["sku"].map(lambda x: x.startswith('B1')) && (df["quantity"] > 22)].head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
14	737550	Fritsch, Russel and Anderson	B1-53102	23	71.56	1645.88	2014-01-04 08:57:48
26	737550	Fritsch, Russel and Anderson	B1-53636	42	42.06	1766.52	2014-01-08 00:02:11
31	714466	Trantow-Barrows	B1-33087	32	19.56	625.92	2014-01-09 10:16:32

pandas支持的另外一个很有用的函数是`isin`。它使得我们可以定义一个列表，里面包含我们所希望查找的值。在这个例子中，我们查找包含两个特定account number值的全部项目。

Python

```
df[df['account number'].isin([714466, 218895])].head()
```

	account number	name	sku	quantity	unit price	ext price	date
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
5	714466	Trantow-Barrows	S2-77896	17	87.63	1489.71	2014-01-02 10:07:15
6	218895	Kulas Inc	B1-65551	2	31.10	62.20	2014-01-02 10:57:23
8	714466	Trantow-Barrows	S1-50961	22	84.09	1849.98	2014-01-03 11:29:02

pandas支持的另外一个函数叫做`query`，它使得我们可以有效的再数据集中选择数据。使用它需要安装[numexpr](#)，所以请确保你在进行下面步骤前已经进行了安装。

如果你想要通过名字来得到一个消费者列表，你可以使用`query`来完成，和前面展示的python语法类似。

Python

```
df.query('name == ["Kulas Inc", "Barton LLC"]').head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
6	218895	Kulas Inc	B1-65551	2	31.10	62.20	2014-01-02 10:57:23
33	218895	Kulas Inc	S1-06532	3	22.36	67.08	2014-01-09 23:58:27
36	218895	Kulas Inc	S2-34077	16	73.04	1168.64	2014-01-10 12:07:30

这里只是做个简单的示例，query函数能做到的还不止这些。我在此展示这些函数的用法，以便当你有需要的时候，会意识到可以用它。

处理日期

使用pandas，你可以对日期进行更加复杂的筛选。在我们处理日期前，我建议你把日期栏进行一个排序，以便返回的结果如你所愿。

Python

```
df = df.sort('date')
df.head()
```

```
1 df = df.sort('date')
2 df.head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

在操作日期前，为您展示python的筛选语法。

Python

```
df[df['date']
   &gt;='20140905'].head()
```

```
1 df[df['date'] &gt;='20140905'].head()
```

	account number	name	sku	quantity	unit price	ext price	date
1042	163416	Purdy-Kunde	B1-38851	41	98.69	4046.29	2014-09-05 01:52:32
1043	714466	Trantow-Barrows	S1-30248	1	37.16	37.16	2014-09-05 06:17:19
1044	729833	Koeppe Ltd	S1-65481	48	16.04	769.92	2014-09-05 08:54:41
1045	729833	Koeppe Ltd	S2-11481	6	26.50	159.00	2014-09-05 16:33:15
1046	737550	Fritsch, Russel and Anderson	B1-33364	4	76.44	305.76	2014-09-06 08:59:08

pandas的一个特别棒的特性是它能够理解日期，所以它允许我们进行部分筛选。如果我只想要查看最近几个月的日期数据，我可以这样做。

Python

```
df[df['date'] >='2014-03'].head()
1 df[df['date'] >='2014-03'].head()
```

	account number	name	sku	quantity	unit price	ext price	date
242	163416	Purdy-Kunde	S1-30248	19	65.03	1235.57	2014-03-01 16:07:40
243	527099	Sanford and Sons	S2-82423	3	76.21	228.63	2014-03-01 17:18:01
244	527099	Sanford and Sons	B1-50809	8	70.78	566.24	2014-03-01 18:53:09
245	737550	Fritsch, Russel and Anderson	B1-50809	20	50.11	1002.20	2014-03-01 23:47:17
246	688981	Keeling LLC	B1-86481	-1	97.16	-97.16	2014-03-02 01:46:44

当然，你可以把筛选标准链接起来。

Python

```
df[(df['date'] >='20140701') &
1 df[(df['date'] >='20140701') & (df['date'] <= '20140715')].head()
```

	account number	name	sku	quantity	unit price	ext price	date
778	737550	Fritsch, Russel and Anderson	S1-65481	35	70.51	2467.85	2014-07-01 00:21:58
779	218895	Kulas Inc	S1-30248	9	16.56	149.04	2014-07-01 00:52:38
780	163416	Purdy-Kunde	S2-82423	44	68.27	3003.88	2014-07-01 08:15:52
781	672390	Kuhn-Gusikowski	B1-04202	48	99.39	4770.72	2014-07-01 11:12:13
782	642753	Pollich LLC	S2-23246	1	51.29	51.29	2014-07-02 04:02:39

由于pandas可以理解日期列，所以可以将日期值设为不同的格式，都会得到正确的结果。

	account number	name	sku	quantity	unit price	ext price	date
1168	307599	Kassulke, Ondricka and Metz	S2-23246	6	88.90	533.40	2014-10-08 06:19:50
1169	424914	White-Trantow	S2-10342	25	58.54	1463.50	2014-10-08 07:31:40
1170	163416	Purdy-Kunde	S1-27722	22	34.41	757.02	2014-10-08 09:01:18
1171	163416	Purdy-Kunde	B1-33087	7	79.29	555.03	2014-10-08 15:39:13
1172	672390	Kuhn-Gusikowski	B1-38851	30	94.64	2839.20	2014-10-09 00:22:33

Python

```
df[df['date'] >= '10-10-2014'].head()
1 df[df['date'] >= '10-10-2014'].head()
```

	account number	name	sku	quantity	unit price	ext price	date
1174	257198	Cronin, Oberbrunner and Spencer	S2-34077	13	12.24	159.12	2014-10-10 02:59:06
1175	740150	Barton LLC	S1-65481	28	53.00	1484.00	2014-10-10 15:08:53
1176	146832	Kiehn-Spinka	S1-27722	15	64.39	965.85	2014-10-10 18:24:01
1177	257198	Cronin, Oberbrunner and Spencer	S2-16558	3	35.34	106.02	2014-10-11 01:48:13
1178	737550	Fritsch, Russel and Anderson	B1-53636	10	56.95	569.50	2014-10-11 10:25:53

当操作时间序列数据时，如果你把数据进行转化，以日期作为索引，我们可以做一些变相的筛选。

使用set_index 来设置新的索引。

Python

```
df2 = df.set_index(['date'])
df2.head()
1 df2 = df.set_index(['date'])
2 df2.head()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-01-01 07:21:51	740150	Barton LLC	B1-20000	39	86.69	3380.91
2014-01-01 10:00:47	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16
2014-01-01 13:24:58	218895	Kulas Inc	B1-69924	23	90.70	2086.10
2014-01-01 15:05:22	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05
2014-01-01 23:26:55	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26

你可以通过切分数据来获取一段区间。

Python

```
df2["20140101":"20140201"]
"].head()
1 df2["20140101":"20140201"].head()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-01-01 07:21:51	740150	Barton LLC	B1-20000	39	86.69	3380.91
2014-01-01 10:00:47	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16
2014-01-01 13:24:58	218895	Kulas Inc	B1-69924	23	90.70	2086.10
2014-01-01 15:05:22	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05
2014-01-01 23:26:55	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26

再一次的，我们可以使用不同的日期表示方法来避免模棱两可的日期命名惯例。

Python

```
df2["2014-Jan-1":"2014-Feb-1"].head()
1 df2["2014-Jan-1":"2014-Feb-1"].head()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-01-01 07:21:51	740150	Barton LLC	B1-20000	39	86.69	3380.91
2014-01-01 10:00:47	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16
2014-01-01 13:24:58	218895	Kulas Inc	B1-69924	23	90.70	2086.10
2014-01-01 15:05:22	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05
2014-01-01 23:26:55	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26

Python

```
df2["2014-Jan-1":"2014-Feb-1"].tail()
1 df2["2014-Jan-1":"2014-Feb-1"].tail()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-01-31 22:51:18	383080	Will LLC	B1-05914	43	80.17	3447.31
2014-02-01 09:04:59	383080	Will LLC	B1-20000	7	33.69	235.83
2014-02-01 11:51:46	412290	Jerde-Hilpert	S1-27722	11	21.12	232.32
2014-02-01 17:24:32	412290	Jerde-Hilpert	B1-86481	3	35.99	107.97
2014-02-01 19:56:48	412290	Jerde-Hilpert	B1-20000	23	78.90	1814.70

Python

```
df2["2014"].head()
```

```
1 df2["2014"].head()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-01-01 07:21:51	740150	Barton LLC	B1-20000	39	86.69	3380.91
2014-01-01 10:00:47	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16
2014-01-01 13:24:58	218895	Kulas Inc	B1-69924	23	90.70	2086.10
2014-01-01 15:05:22	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05
2014-01-01 23:26:55	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26

Python

```
df2["2014-Dec"].head()
```

```
1 df2["2014-Dec"].head()
```

	account number	name	sku	quantity	unit price	ext price
date						
2014-12-01 20:15:34	714466	Trantow-Barrows	S1-82801	3	77.97	233.91
2014-12-02 20:00:04	146832	Kiehn-Spinka	S2-23246	37	57.81	2138.97
2014-12-03 04:43:53	218895	Kulas Inc	S2-77896	30	77.44	2323.20
2014-12-03 06:05:43	141962	Herman LLC	B1-53102	20	26.12	522.40
2014-12-03 14:17:34	642753	Pollich LLC	B1-53636	19	71.21	1352.99

正如你所见到的那样，在进行基于日期的排序或者筛选时，可以有很多选择。

额外的字符串方法

Pandas同样已经支持了矢量字符串方法。

如果我们想识别出sku栏中包含某一特定值的全部值。我们可以使用str.contains。在这个例子中，我们已知sku总是以一种相同的方式表示，所以B1仅会出现在sku的前面。你需要理解你的数据来保证你能够得到你想要的结果。

Python

```
df[df['sku'].str.contains('B1')]  
)].head()  
  
1 df[df['sku'].str.contains('B1')].head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
6	218895	Kulas Inc	B1-65551	2	31.10	62.20	2014-01-02 10:57:23
14	737550	Fritsch, Russel and Anderson	B1-53102	23	71.56	1645.88	2014-01-04 08:57:48
17	239344	Stokes LLC	B1-50809	14	16.23	227.22	2014-01-04 22:14:32

我们可以把查询连接起来并且使用排序来控制数据的顺序。

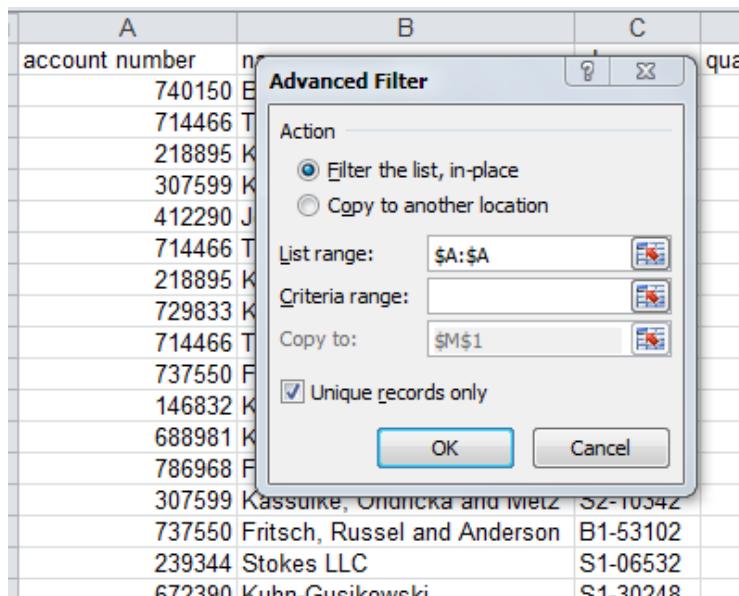
Python

```
df[(df['sku'].str.contains('B1-531')) &  
; (df['sku'].str.contains('B1-531')) & (df['quantity']>40)].sort(columns=['quantity','name'],ascending=[0,1])
```

	account number	name	sku	quantity	unit price	ext price	date
684	642753	Pollich LLC	B1-53102	46	26.07	1199.22	2014-06-08 19:33:33
792	688981	Keeling LLC	B1-53102	45	41.19	1853.55	2014-07-04 21:42:22
176	383080	Will LLC	B1-53102	45	89.22	4014.90	2014-02-11 04:14:09
1213	604255	Halvorson, Crona and Champlin	B1-53102	41	55.05	2257.05	2014-10-18 19:27:01
1215	307599	Kassulke, Ondricka and Metz	B1-53102	41	93.70	3841.70	2014-10-18 23:25:10
1128	714466	Trantow-Barrows	B1-53102	41	55.68	2282.88	2014-09-27 10:42:48
1001	424914	White-Trantow	B1-53102	41	81.25	3331.25	2014-08-26 11:44:30

彩蛋任务

在Excel中，我发现我自己经常会尝试从一个冗长的列表中，得到一个包含不重复项的小列表。在Excel中这件事情需要分几步来完成，但是在Pandas中却非常简单。有一种方式是使用Excel中提供的高级筛选工具来完成。



在pandas中，我们对某列使用这个unique函数来获取这个列表。

Python

```
df["name"].unique()
```

```
1 df["name"].unique()
```

Python

```
array([u'Barton LLC',  
      u'Trantow-Barrows',  
      ])
```

```
1 array([u'Barton LLC', u'Trantow-Barrows', u'Kulas Inc',  
2 u'Kassulke, Ondricka and Metz', u'Jerde-Hilpert', u'Koepp Ltd',  
3 u'Fritsch, Russel and Anderson', u'Kiehn-Spinka', u'Keeling LLC',  
4 u'Frami, Hills and Schmidt', u'Stokes LLC', u'Kuhn-Gusikowski',  
5 u'Herman LLC', u'White-Trantow', u'Sanford and Sons',  
6 u'Pollich LLC', u'Will LLC', u'Cronin, Oberbrunner and Spencer',  
7 u'Halvorson, Crona and Champlin', u'Purdy-Kunde'], dtype=object)  
8 If we wanted to include the account number, we could use drop_duplicates .
```

如果我们想要包含账户号，我们可以使用drop_duplicates。

Python

```
df.drop_duplicates(subs  
et=["account",  
    ])
```

```
1 df.drop_duplicates(subset=["account number", "name"]).head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

很显然我们放入了的数据超过了我们的需要，得到了一些无用的信息，因此，使用ix 来仅仅选择第一第二列。

Python

```
df.drop_duplicates(subs  
et=["account",  
    ])
```

```
1 df.drop_duplicates(subset=["account number", "name"]).ix[:,[0,1]]
```

	account number	name
0	740150	Barton LLC
1	714466	Trantow-Barrows
2	218895	Kulas Inc
3	307599	Kassulke, Ondricka and Metz
4	412290	Jerde-Hilpert
7	729833	Koeppe Ltd
9	737550	Fritsch, Russel and Anderson
10	146832	Kiehn-Spinka
11	688981	Keeling LLC
12	786968	Frami, Hills and Schmidt
15	239344	Stokes LLC
16	672390	Kuhn-Gusikowski
18	141962	Herman LLC
20	424914	White-Trantow
21	527099	Sanford and Sons
30	642753	Pollich LLC
37	383080	Will LLC
51	257198	Cronin, Oberbrunner and Spencer
67	604255	Halvorson, Crona and Champlin
106	163416	Purdy-Kunde

我认为这个记住这个单独的命令比记忆Excel的各步操作更容易。

如果你想要查看我的[笔记](#) 请随意下载。

结论

在我发表了我的第一篇文章之后，Dave Proffer在Twitter上转发了我的文章并评论到“打破你#excel沉迷的一些好技巧”。我觉得这句话非常准确，它描述了在我们的生活中使用Excel是多么的频繁。大多数的人只管伸手去用却从来没有意识到它的局限性。我希望这个系列的文章可以帮助大家认识到我们还有其他的替代工具，Python+Pandas是一个极其强大的组合。

打赏支持我翻译更多好文章，谢谢！

[打赏译者](#)

打赏支持我翻译更多好文章，谢谢！

任选一种支付方式

微信扫一扫转账



向摇钱少年ai转账

朕看的过瘾，赏小艾！

¥ 1.98

支付宝扫一扫，向我付款



¥1.99

赞赏你在伯乐在线的文章

1 赞 3 收藏 [1 评论](#)

关于作者：[艾凌风](#)



初入职场小码农;翻译组的勤务员;C/Python/在线教育/英文翻译 [个人主页](#) · [我的文章](#) · 71 ·

用Python和OpenCV创建一个图片搜索引擎的完整指南

本文由 [伯乐在线 - Daetalus](#) 翻译, [sunbiaobiao](#) 校稿。未经许可, 禁止转载!

英文出处: www.pyimagesearch.com。欢迎加入[翻译组](#)。

大家都知道, **通过文本或标签来搜索图片的体验非常糟糕。**

无论你是将个人照片贴标签并分类, 或是在公司的网站上搜索一堆照片, 还是在为下一篇博客寻找合适的图片。在用文本和关键字来描述图片是非常痛苦的事。

我就遇到了这样的痛苦的事情, 上周二我打开了一个很老的家庭相册, 其中的照片是9年前扫描成电子档的。

我想找到我家在夏威夷海滩拍的照片。我用iPhoto打开相册, 慢慢的浏览。这个过程非常辛苦, 每个JPEG图像的元信息中的日期都是错的。我已经不记得文件夹中的图片是如何排列的, 我绝望的搜索海滩的照片, 但还是找不到。

也许是运气, 我跌跌撞撞的找到了其中一幅海滩上的照片。多美的一幅照片啊。蓝天中飘着棉花糖般的白云。晶莹透彻的海水, 像丝绸一样掠过在金色的沙滩上。我几乎可以感觉微风轻抚着面庞, 呼吸着海边湿润的空气。

找到这幅照片后, 我停止了手动搜索, 打开一个代码编辑器。

虽然iPhoto这样的应用能让你将相片分组, 甚至可以检测人脸, 但我们可以做的更多。

注意, 我并不是介绍如何手动给图片添加标签。我是在介绍更强大的东西。比如通过一幅图片来搜索一组相似的图片。

这是不是很酷? 只需鼠标点击一次就可以可视化搜索图片。

这就是我的工作内容。我用半个小时写好代码, 完成了一个针对家庭假期相册的图片搜索引擎。

然后用上面找到的那张海滩图片作为搜索源。几秒后我就找到了相册中其他的海滩图片, 其中没有任何**为某张图片添加标签的动作**。

感兴趣吗, 我们继续。

在本文的其他部分, 我将介绍如何自己创建一个图像搜索引擎。

想要文本中的代码?

直接跳到文本中的最后的“下载”一节。

什么是图像搜索引擎?

读者也许会问, 什么才是一个真正的图像搜索引擎?

我的意思是, 我们都熟悉基于文本的搜索引擎, 如Google、Bing、Baidu等。用户只需输入几个与内容相关的关键字, 接着就会获得搜索结果。但对于图像搜索引擎, 其工作方式就有点区别。搜索时使用的不是文字, 而是图片。

听起来很困难, 我的意思是, 如何量化图像的内容, 让其可搜索呢?

本文将逐步回答这个问题。首先, 先了解一下图像搜索引擎的内容。

一般来说, 有三种类型的图像搜索引擎: **基于元数据、基于例子、混合模式**。

基于元数据

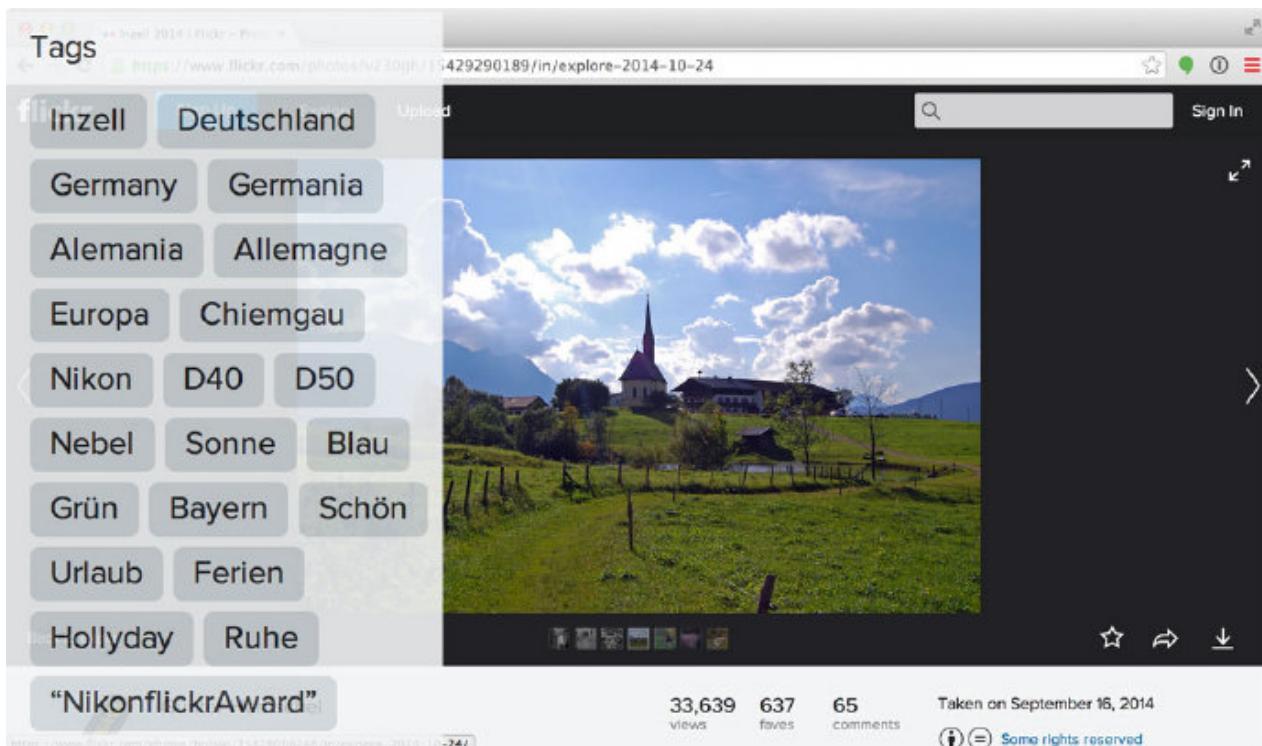


图1：基于元数据的图像搜索引擎的例子。注意其中关键字和标签是手动关联到图像上的。

通过元数据搜索与标准的关键字搜索引擎没有本质的不同。这种方式很少检测图像本身的内容。而是用相关的文本信息：如手动注释或添加标签；以及自动上下文提示（如网页中该图片附近的文字信息）。

当用户在基于元数据的系统上进行搜索时，与传统的文本搜索引擎其实差点不多的。得到的是含有类似标签或注释的图片。

再次说明，使用基于元数据系统的工具进行搜索，基本上是不查找图像本身的。

用基于元数据进行搜索的应用中，一个比较好的例子就是Flickr。将图像上传到Flickr后，输入一些文本作为标签描述这幅图像。接着Flickr会使用这些关键字进行搜索，查找并推荐其他相关的图像。

基于例子搜索

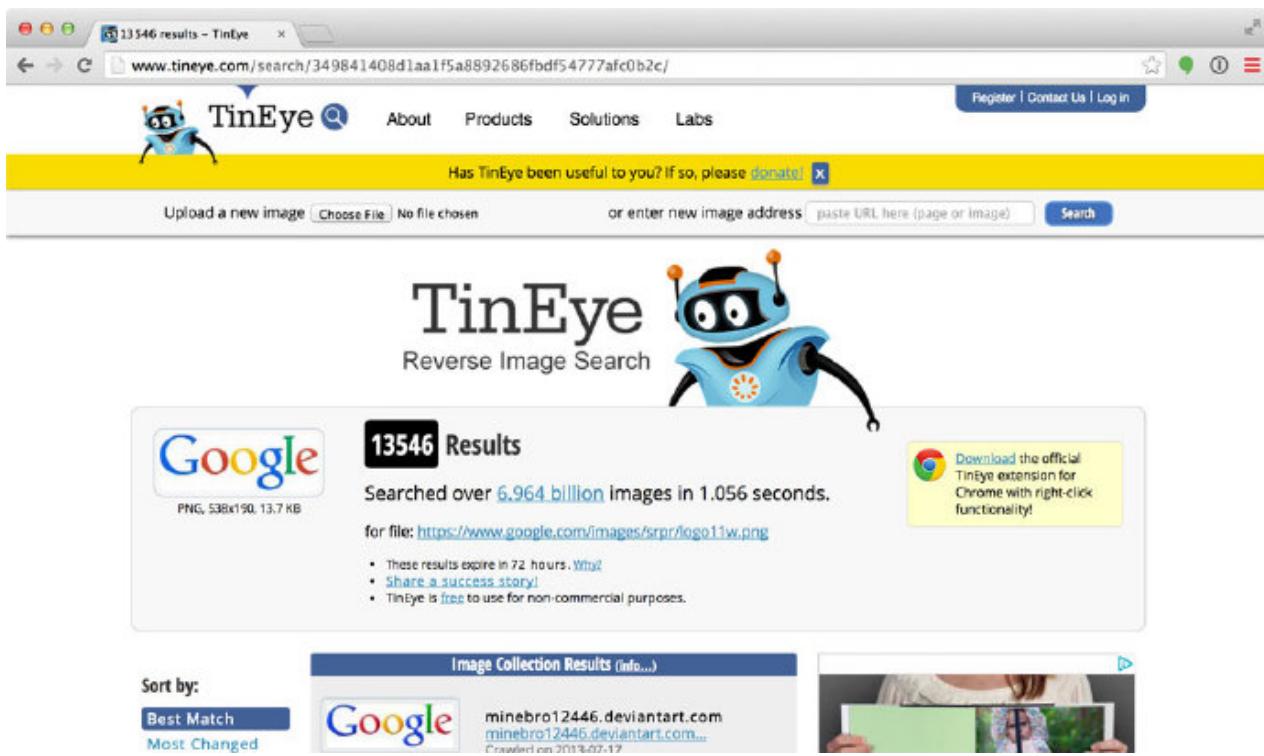


图2：TinEye就是一个基于例子的图像搜索引擎。该搜索引擎会使用图像本身的内容进行搜索，而不是使用文本搜索。

另一方面，基于例子搜索仅仅依赖于图像的内容，不需要提供关键字。引擎会分析、量化并存储图像，然后返回其他相关的图像。

图像搜索引擎量化图像内容的过程称为**基于内容的图像信息获取（Content-Based Image Retrieval, CBIR）**系统。术语CBIR通常用在学术文献中，但在实际上，这是“图像搜索引擎”的另一种表述，并特意指明该搜索引擎是严格基于图像的内容的，没有任何关于图像的文本的信息。

基于例子系统的一个比较好的例子就是TinEye。向TinEye提交待查找图像时，TinEye实际上是一个逆向图像搜索引擎，TinEye返回该图像最相近的匹配，以及该图像位于的原始网页地址。

看下本节刚开始的示例图像。我上传了一个Google logo图像。TinEye检测了图像的内容，在搜索了超过60亿幅图片后，返回了1.3万个含有Google logo图片的网页。

所以仔细想一下：你需要为TinEye中的60亿幅图像添加标签吗？**当然不需要**，为这么多图片手动添加标签需要庞大的人力物力。

取而代之，使用某些算法从图像本身中提取“特征”（如用一组数字来量化并抽象表示图像）。接着，当用户提交了需要查找的图像，从这幅图像中提取特征，将其与数据库中的特征进行比较，尝试返回相似的图像。

同样，依然需要强调，基于例子的搜索系统非常依赖图像的内容。这种类型的系统很难构建并扩展，但可以用算法全自动的搜索，无需人工干预。

混合方式

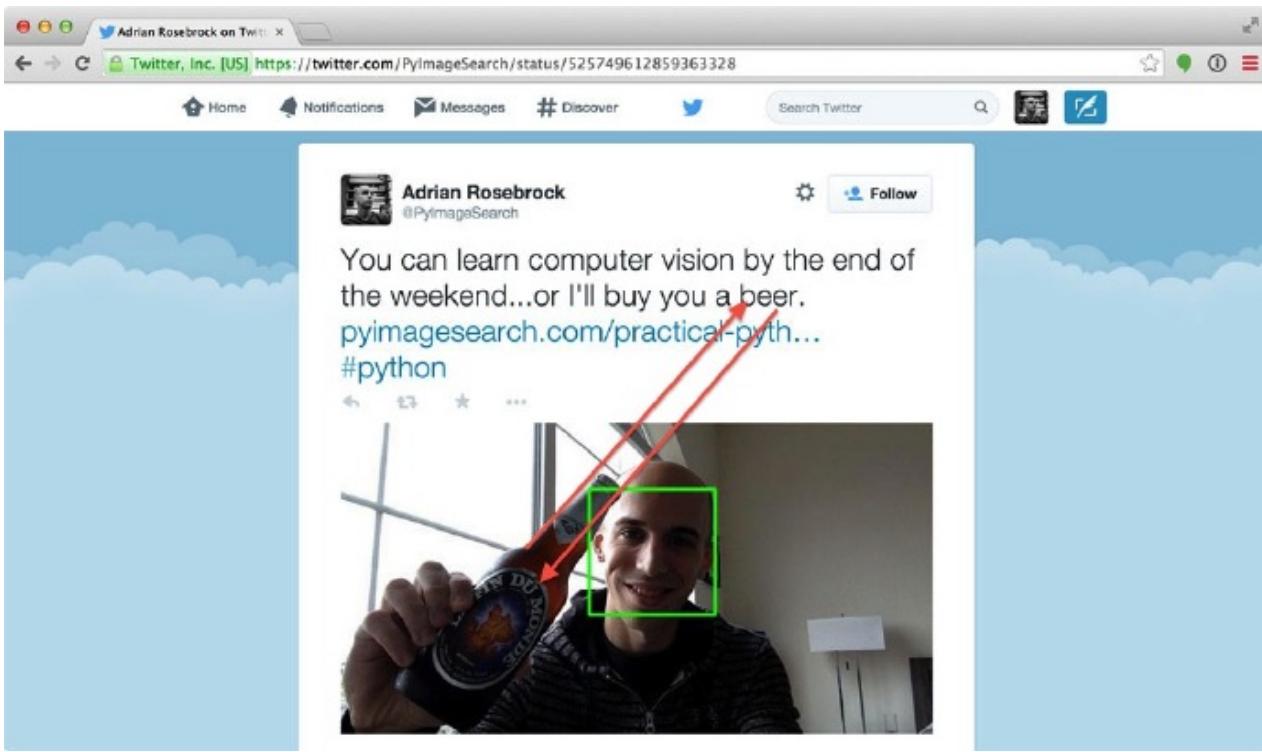


图3：混合式图像搜索引擎可以同时基于图像和文本描述搜索。

当然，除了前面介绍的两种方式，还有一种介于两者之间的方式，如Twitter使用的。

在Twitter上，可以与推文一起上传图像。这样就可以即使用提取图像本身的特征，也可以使用推文中的文本，从而诞生一种混合方式。基于这种方式可以构建一个即使用上下文关系，又使用基于例子搜索的策略的图像搜索引擎。

提示：有兴趣阅读更多关于不同类型的图像搜索引擎的资料？我有一篇完整的博客介绍比较这些搜索引擎的，链接[在此](#)。

在进一步描述和构建图像搜索引擎之前，让我们先来了解一些重要的术语。

一些重要的术语

在深入了解之前，先花点时间了解一些重要的术语。

在构建图像搜索引擎时，首先要对**数据集编列索引**。索引化数据集是量化数据集的过程，即通过**图像描述符**（image descriptor，也称描述子）提取每幅图像的**特征**。

图像描述符就是用来描述图像的算法。

例如：

- R、G、B三色通道的均值和标准差。
- 图像特征形状的统计矩。
- 形状和纹理的梯度和朝向。

这里最重要的是图像描述符确定了图像是如何量化的。

另一方面，**特征**是图像描述符的输出。当将一幅图像放入图像描述符中时，就会获得这幅图像的特征。

以基本的术语来说。**特征**（或**特征向量**）仅仅是一个用来抽象表示或量化的图像的数字列表。

来看下面这幅示例图像：

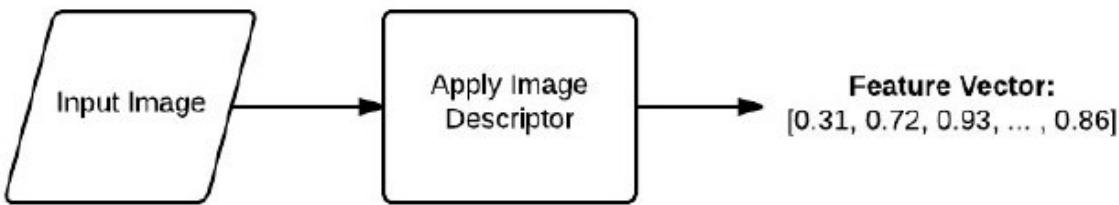


图4：图像描述符的管道。描述符中有一幅输入图像，使用图像描述符会返回一个特征向量（一个数字列表），这里对一幅输入图像使用图像描述符，输出是一组用来量化图像的数字。

通过距离量测或其他相似度比较函数，特征向量可以用来表示比较的相似度。**距离量测和相似度函数**采用两个特征向量作为输入，返回一个数值来描述着两个特征向量的相似度。

下图以可视化的方式比较了两幅图的比较过程：

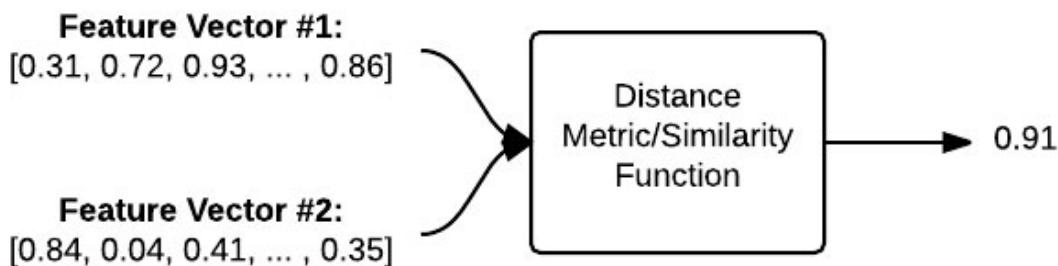


图5：为了比较两幅图，必须将对应的特征向量输入进距离量测/相似度比较函数。输出结果是一个数值，量化地描述两幅图下的相似度。

给定两个特征向量，使用距离函数来确定这两个特征向量的相似度。距离函数的输出是一个浮点数，用来描述两幅图像的相似度。

CBIR系统的4个步骤

无论构建的是什么样的CBIR系统，最终都可以分解成4个不同的步骤。

1. **定义图像描述符**：在这一阶段，需要决定描述图像的哪一方面。是关注图像的颜色，还是图像中的物体形状，或是图像中的纹理？
2. **索引化数据集**：现在有了图像描述符，接着就是将这个图像描述符应用得到数据集中的每幅图像，提取这些图像的特征，将其存储起来（如CSV文件、RDBMS、Redis数据库中，等），这样后续步骤就能使用以便比较。
3. **定义相似矩阵**：很好，现在有了许多特征向量。但如何比较这些特征向量呢？流行的方式是比较欧几里德距离、余弦距离、或卡方距离。但实际中取决于两点：1、数据集；2、提取的特征类型。
4. **搜索**：最后一步是进行实际的搜索。用户会向系统提交一幅需要搜索的图片（例如从上传窗口或通过移动App提交），而你的任务是：1、提取这幅图像的特征；2、使用相似度函数将这幅图像的特征与已经索引化的特征进行比较。这样，只需根据相似度函数的结果，返回相关的图像就可以了。

再次强调，这是所有CBIR系统中最基本的4步。如果使用的特征表示不同，则步骤数会增加，也会为每个步骤增加一定数量的子步骤。就目前而言，让我们关注并使用这4步。

下面通过图像来具体了解这4个大步骤。下图表述的是步骤1和2：

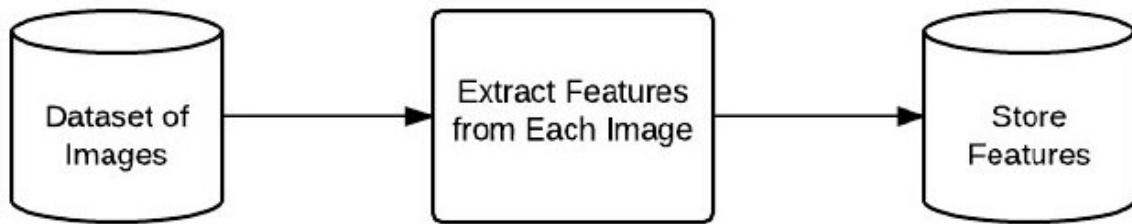


图6：处理并提取数据集中的每幅图像的流程图。

首先提取数据集中每幅图像的特征，将这些特征存入一个数据库。

接着可以执行搜索（步骤3和4）：

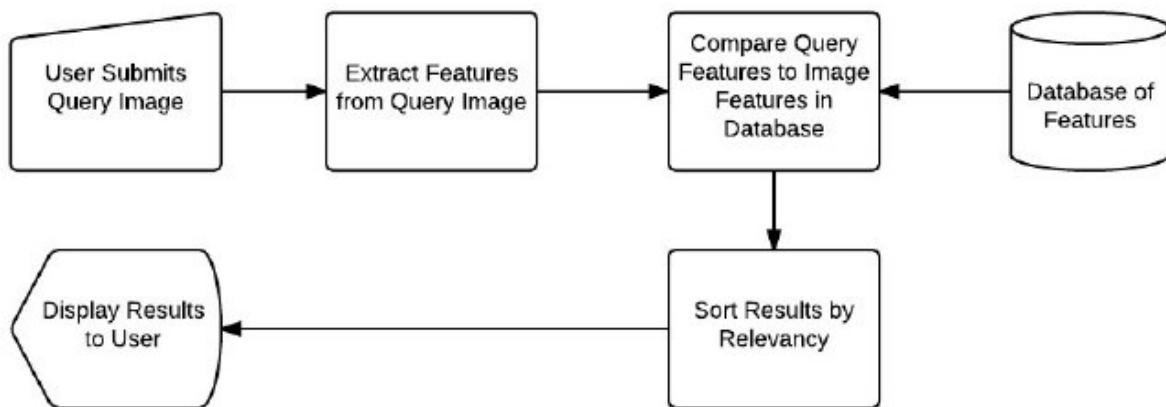


图7：在CBIR系统中执行搜索。用户提交一个搜索请求，系统对搜索图像进行描述，其特征会与数据集中已有的特征进行比较，并对结果根据相关度进行排序，返回给用户。

首先，用户必须像搜索引擎提交一幅需要查找的图像。接着对这幅图像提取特征信息。将这些特征信息与数据集中已有的图像的特征信息进行比较。最后，对结果根据相关度进行排序并返回给用户。

数据集——假期相册

这里将[INRIA假期数据集](#)作为图像搜索的数据集。

这个数据集含有全世界许多地方的假期旅行，包括埃及金字塔、潜水、山区的森林、餐桌上的瓶子和盘子、游艇、海面上的日落。

下面是数据集中的一些图片：



图8：数据集中的示例图像。我们将使用这些图像构建自己的图像搜索引擎。

在本例中，对于我们希望从旅行相册中找到某种景色的相片来说，用这幅数据集作为示例来说非常好。

目标

我们的目标是构建一个个人图像搜索引擎。将假期照片作为数据集，我们希望将这个数据集变成可搜索的，即一个“基于例子”的图像搜索引擎。例如，如果我提交了一幅在河中航行的帆船的照片，图像搜索引擎应该能找到并返回相册中码头和船坞拍摄的照片。

看下面的图，其中有我提交的照片，即一幅在水里的船。得到了假期照片集合中相关的图像。

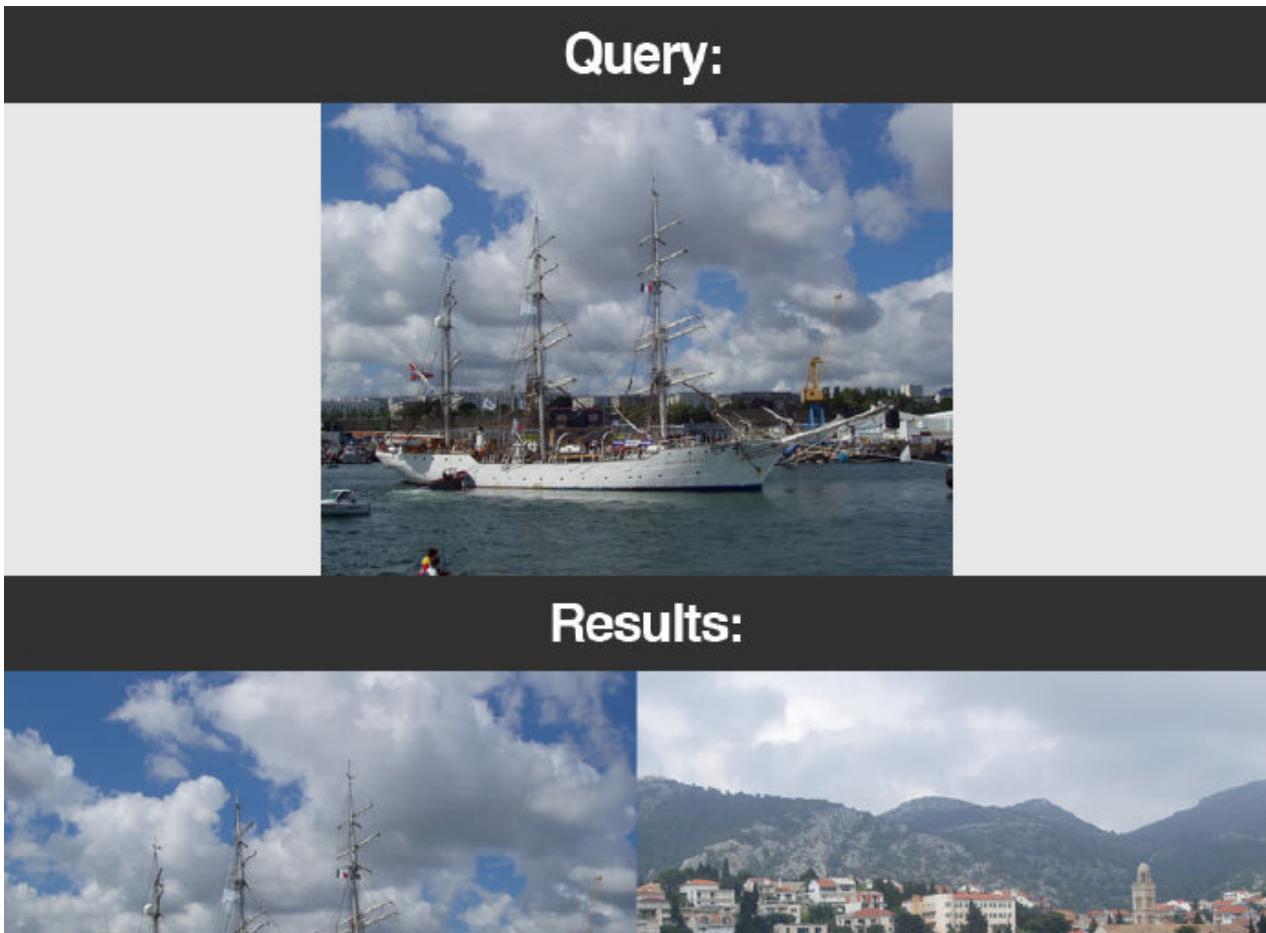






图9：图像搜索引擎的例子。提交了一幅含有海中船只的图像。返回相关的图像，这些图像都是在海中的船。

为了构建这个系统，将使用一个简单且有效的图像描述符：**颜色直方图**。

通过将颜色直方图作为我们的图像描述符，可以根据图像的色彩分布提取特征。由于这一点，我们可以对我们的图像搜索引擎做个重要的假设：

假设：如果图像含有相似的色彩分布，那么这两幅图像就认为是相似的。即使图像的内容差别非常大，依然会根据色彩分布而被认为是相近的。

这个假设非常重要，在使用颜色直方图作为图像描述符时，这是个公平且合理的假设。

第一步1：定义图像描述符

这里不使用标准的颜色直方图，而是对其进行一些修改，使其更加健壮和强大。

这个图像描述符是HSV颜色空间的3D颜色直方图（色相、饱和度、明度）。一般来说，图像由RGB构成的元组表示。通常将RGB色彩空间想象成一个立方体，如下图所示。

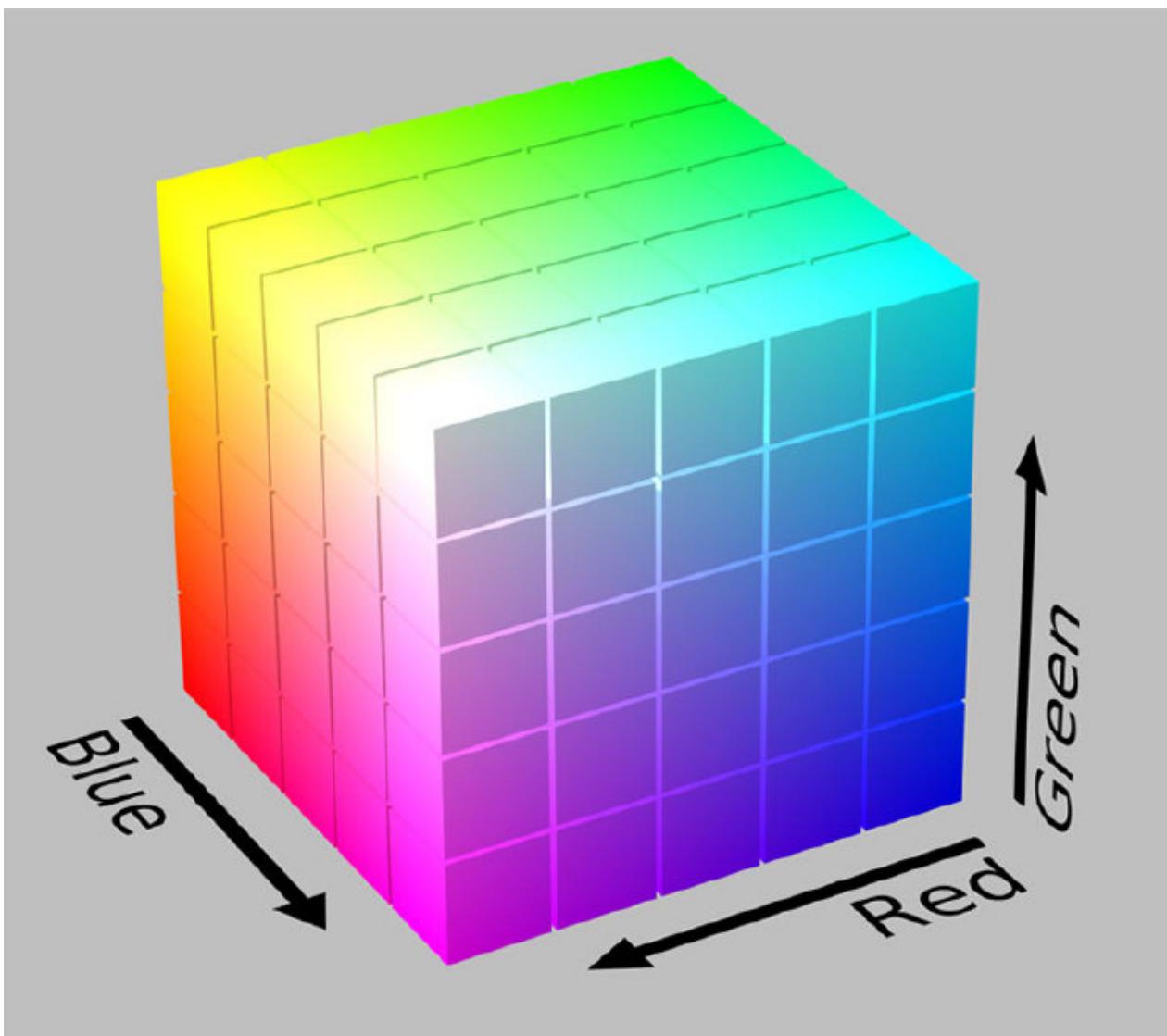


图10：RGB立方体的例子。

然而，虽然RGB值很容易理解，但RGB色彩空间无法模拟人眼接受到的色彩。取而代之，我们使用HSV色彩空间将像素点的映射到圆bin体上。

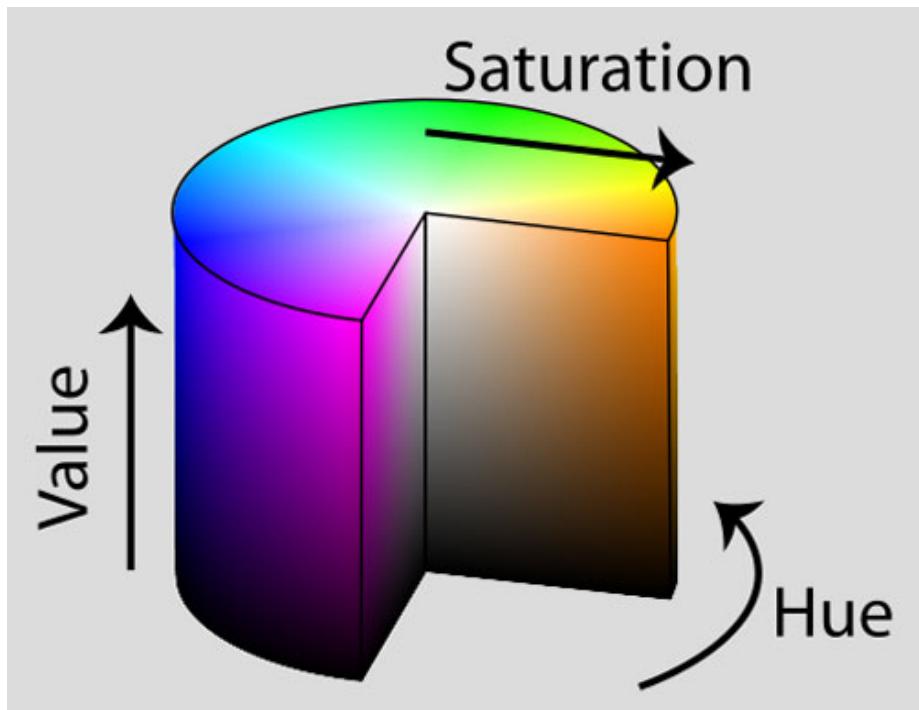


图11：HSV圆bin体的例子。

还有其他颜色空间能够更好的模拟人眼接收的颜色，如CIE L*a*b*和CIE XYZ颜色空间，但作为第一个图像搜索引擎的实现，先简化使用的色彩模型。

现在选定了颜色空间，接着需要定义直方图中bin的数量。直方图用来粗略的表示图像中各强度像素的密度。本质上，**直方图会估计底层函数的概率密度**。在本例中，P是图像I中像素色彩C出现的概率。

主要注意的是，为直方图选取bin的数目需要不断的权衡。如果选择的bin数目过少，那么直方图含有的数据量就不够，无法区分某些不同颜色分布的图像。反之，如果直方图选取的bin的数目过多，那么其中的组件就过多，导致内容很相近的图片也会判断成不相似。

下面是直方图bin过少的例子。

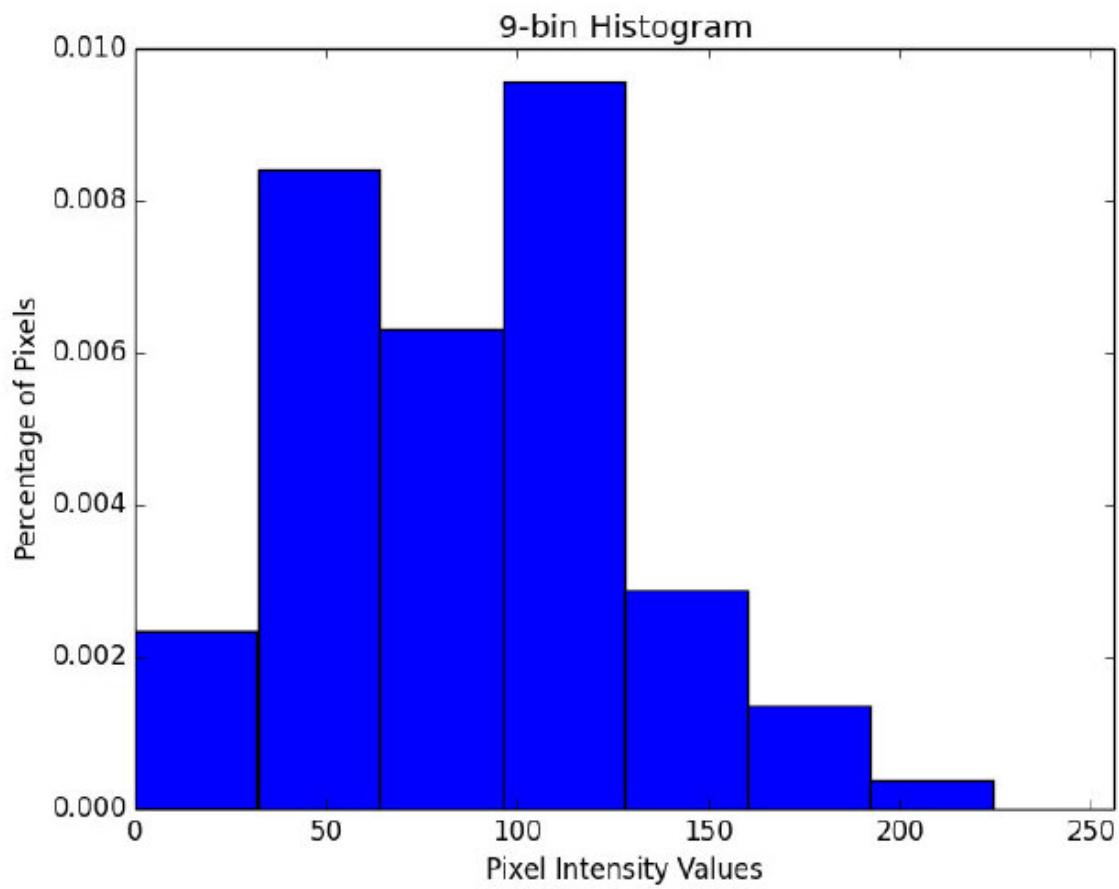


图12：9个bin直方图的例子。注意其中bin的数量很少，只有少数给定的像素值位列其中。

注意其中只有少数几个bin及相应的像素值。

下面是直方图bin过多的例子。

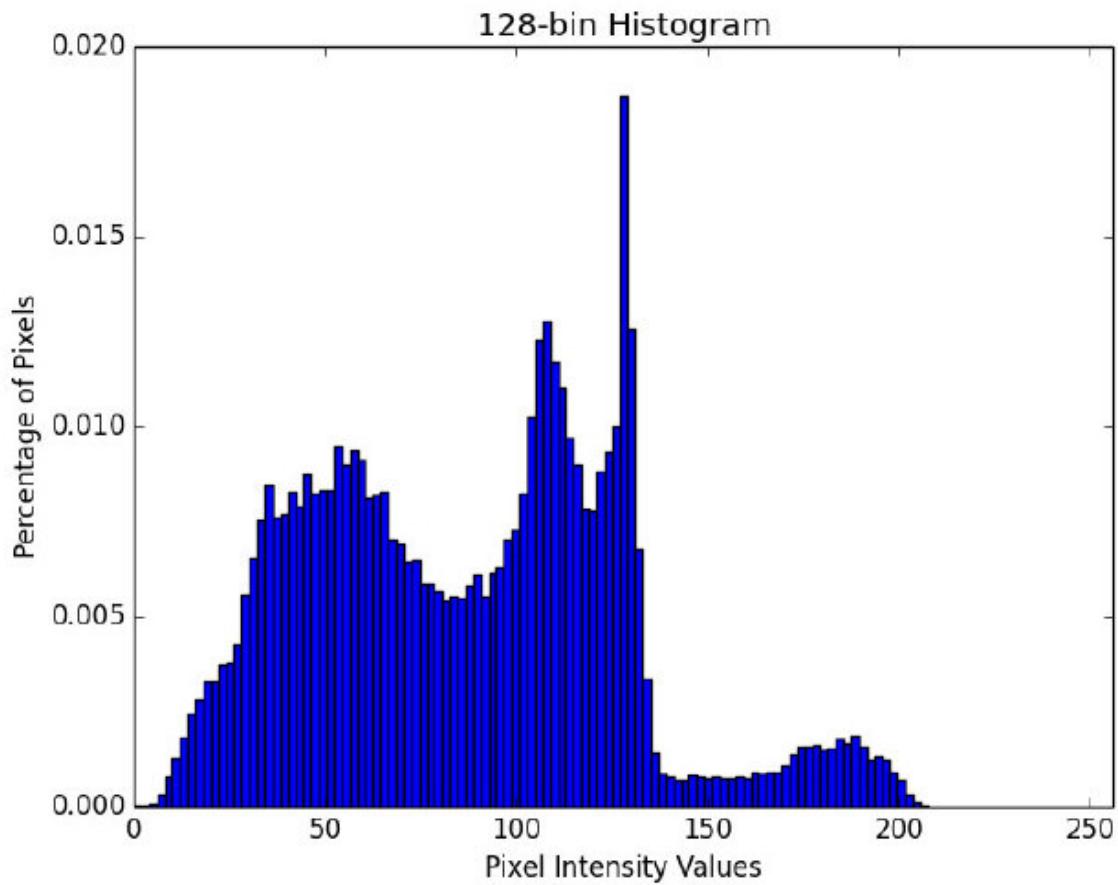


图13：128 bin直方图的例子。注意其中含有许多的柱和相应的像素值。

在上面的例子中使用了许多bin，bin的数目过多，由于需要直方图中每个“山峰”和“山谷”都需要匹配才能认为图像是“相似的”，所以就失去了“概括”图像的能力。

就我个人而言，我喜欢用迭代、实验性的方式来调整bin的数目。迭代方法一般基于数据集的大小调整。数据集越小，使用的bin的数目就越少。如果数据集非常大，则会使用更多的bin，这样可以让直方图更大，更能区分图像。

一般来说，读者需要为颜色直方图描述符实验bin的个数，具体取决于数据集的大小和数据集中图像之间色彩分布的差异。

对于我们的假期照片图像搜索引擎，将在HSV色彩空间中使用3D颜色直方图，8个bin用于色相通道、12个bin用于饱和度通道、3个bin用于明度通道，总共的特征向量有 $8 \times 12 \times 3 = 288$ 。

这意味着数据集中的每幅图像，无论其像素数目是 36×36 ，还是 2000×1800 。最终都会用288个浮点数构成的列表抽象并量化表示。

我认为解释3D直方图最好的方式是用连接词AND。一个3D HSV颜色描述符将查找指定图像中1号bin有多少像素含有色相值，AND有多少像素有饱和度值，AND有多少像素有明度值。计算出符合条件的像素值。虽然需要对每个bin重复这个操作，但可以非常高效的完成这个任务。

很酷，是吧！

理论讲解的够多了，下面来开始编码。

用你最喜欢的编辑器打开一个新文件，命名为colordescriptor.py。加入下面代码：

Python

```
# import the necessary packages
# import the necessary packages
# import numpy as np
# import cv2
#
5 class ColorDescriptor:
6     def __init__(self, bins):
7         # store the number of bins for the 3D histogram
8         self.bins = bins
9
10    def describe(self, image):
11        # convert the image to the HSV color space and initialize
12        # the features used to quantify the image
13        image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
14        features = []
15
16        # grab the dimensions and compute the center of the image
17        (h, w) = image.shape[:2]
18        (cX, cY) = (int(w * 0.5), int(h * 0.5))
```

首先导入所需的Python模块。用NumPy进行数值处理，用cv2使用OpenCV的Python绑定。

在第五行定义了ColorDescriptor类。该类用来封装所有用于提取图像中3D HSV颜色直方图的逻辑。

ColorDescriptor的__init__方法只有一个参数——bins，即颜色直方图中bin的数目。

在第10行定义describe方法，用于描述指定的图像。

在describe方法中，将图像从RGB颜色空间（或是BGR颜色空间，OpenCV以NumPy数组的形式反序表示RGB图像）转成HSV颜色空间。接着初始化用于量化图像的特征列表features。

17和18行获取图像的维度，并计算图像中心(x, y)的位置。

现在遇到难点。

这里不计算整个图像的3D HSV颜色直方图，而是计算图像中**不同区域**的3D HSV颜色直方图。

使用**基于区域**的直方图，而不是**全局**直方图的好处是：这样我们可以模拟各个区域的颜色分布。例如看下面的这幅图像：



图14：待搜索的图像。

在这幅图像中，很明显，蓝天在图像的上部，而沙滩在底部。使用全局搜索的话，就无法确定图像中“蓝色”区域和“棕色”沙子区域的位置。而是仅仅知道图像中有多少比例是蓝色，有多少比例是棕色。

为了消除这个问题，可以对图像中的不同区域计算颜色直方图：



图15：将图像分为5个不同区域的例子。

对于我们的图像描述符，将图像分为5个不同的区域：1、左上角；2、右上角；3、右下角；4、左下角；以及图像的中央。

使用这些区域，可以粗略模拟出不同的区域。能够表示出蓝天在左上角和右上角，沙滩在左下角和右下角。图像的中央是沙滩和蓝天的结合处。

下面的代码是创建基于区域的颜色描述符：

Python

```
# import the necessary packages
# import the necessary packages
# import numpy as np
# import cv2
#
# class ColorDescriptor:
#     def __init__(self, bins):
#         # store the number of bins for the 3D histogram
#         self.bins = bins
#
#     def describe(self, image):
#         # convert the image to the HSV color space and initialize
#         # the features used to quantify the image
#         image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
#         features = []
#
#         # grab the dimensions and compute the center of the image
#         (h, w) = image.shape[:2]
#         (cX, cY) = (int(w * 0.5), int(h * 0.5))
#
#         # divide the image into four rectangles/segments (top-left,
#         # top-right, bottom-right, bottom-left)
#         segments = [(0, cX, 0, cY), (cX, w, 0, cY), (cX, w, cY, h),
#                     (0, cY, h)]
```

```

24
25 # construct an elliptical mask representing the center of the
26 # image
27 (axesX, axesY) = (int(w * 0.75) / 2, int(h * 0.75) / 2)
28 ellipMask = np.zeros(image.shape[:2], dtype = "uint8")
29 cv2.ellipse(ellipMask, (cX, cY), (axesX, axesY), 0, 0, 360, 255, -1)
30
31 # loop over the segments
32 for (startX, endX, startY, endY) in segments:
33 # construct a mask for each corner of the image, subtracting
34 # the elliptical center from it
35 cornerMask = np.zeros(image.shape[:2], dtype = "uint8")
36 cv2.rectangle(cornerMask, (startX, startY), (endX, endY), 255, -1)
37 cornerMask = cv2.subtract(cornerMask, ellipMask)
38
39 # extract a color histogram from the image, then update the
40 # feature vector
41 hist = self.histogram(image, cornerMask)
42 features.extend(hist)
43
44 # extract a color histogram from the elliptical region and
45 # update the feature vector
46 hist = self.histogram(image, ellipMask)
47 features.extend(hist)
48
49 # return the feature vector
50 return features

```

22和23行用于分别定义左上、右上、右下、和左下区域。

这里，我们需要构建一个椭圆用来表示图像的中央区域。在代码的27行，定义一个长短轴分别为图像长宽75%的椭圆。

接着初始化一个空白图像（将图像填充0，表示黑色的背景），该图像与需要描述的图像大小相同，见28行。

最后，在29行使用cv2.ellipse函数绘制实际的椭圆。该函数需要8个不同的参数：

1. 需要绘制椭圆的图像。这里使用了下面会介绍的“掩模”的概念。
2. 两个元素的元组，用来表示图像的中心坐标。
3. 两个元组的元组，表示椭圆的两个轴。在这里，椭圆的长短轴长度为图像长宽的75%。
4. 椭圆的旋转角度。在本例中，椭圆无需旋转，所以值为0。
5. 椭圆的起始角。
6. 椭圆的终止角。看上一个参数，这意味着绘制的是完整的椭圆。
7. 椭圆的颜色，255表示的绘制的是白色椭圆。
8. 椭圆边框的大小。传递正数比会以相应的像素数目绘制椭圆边框。负数表示椭圆是填充模式。

在35行为每个角的掩模分配内存，在36行为图像的每个角绘制白色矩形，接着在37行将矩形减去中间的椭圆。

如果将这个过程用图像动态表示，看上去应该是这样的：

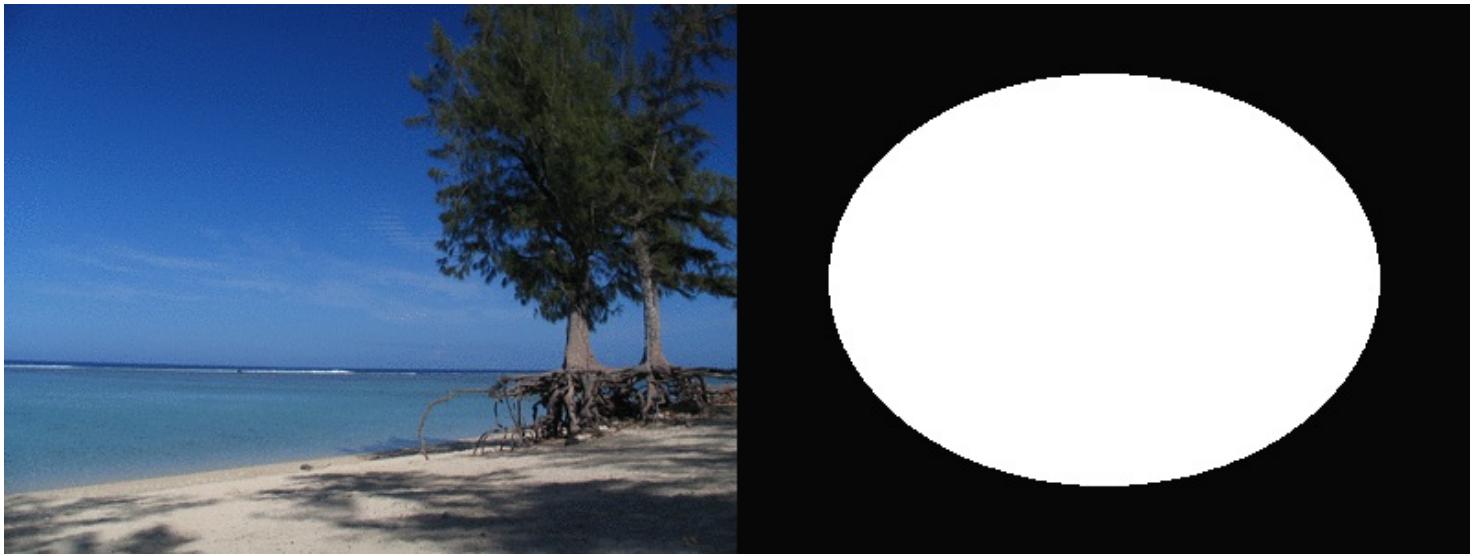


图16：为图像中每个需要提取特征的区域构建掩模。

如这个动画所示，我们独立检测每块区域，在迭代中移除每个矩形与图像中间的椭圆重叠的部分。

读者也许会奇怪，“我们不是要提取图像的颜色直方图吗？为什么要做这些掩模的事情？”

问得好！

原因是因为我们需要告诉OpenCV直方图函数我们要提取的颜色直方图的区域。

记住，我们的目标是分开描述图像的每个区域。表述不同区域最高效的方法是使用掩模。对于图像中某个点(x, y)，只有掩模中该点位白色(255)时，该像素点才会用于计算直方图。如果该像素点对应的位置在掩模中是黑色(0)，将会被忽略。

通过下图可以更加深刻的了解这个概念。

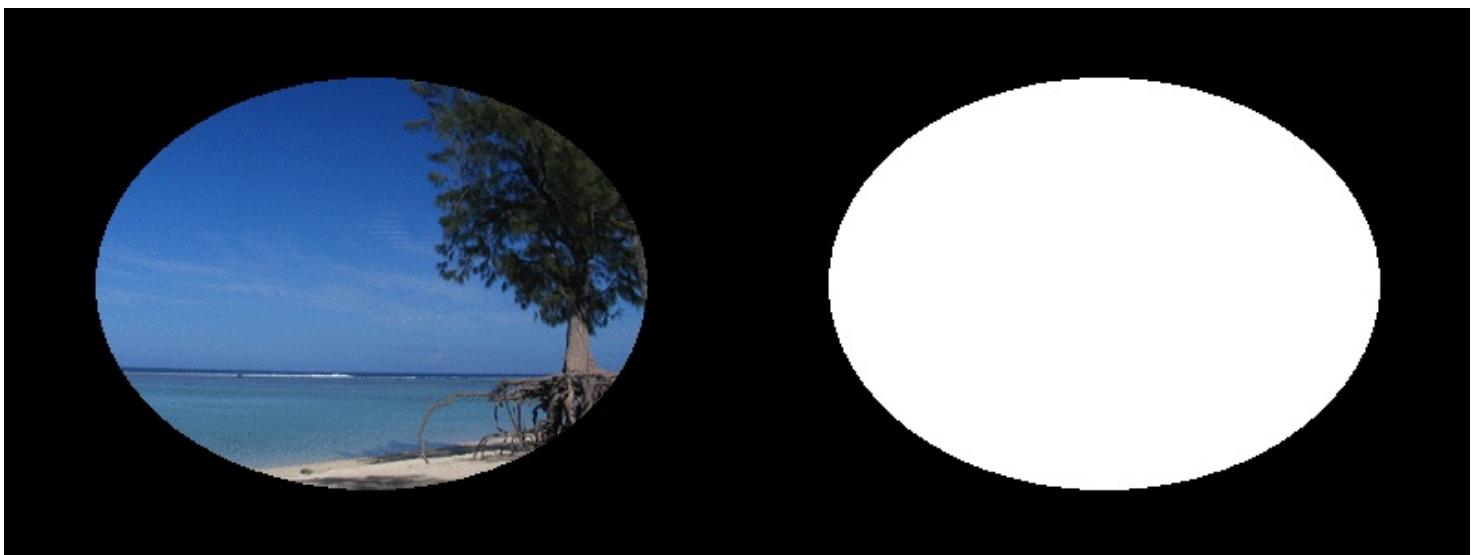


图17：对图像使用掩模。注意左图中只有右图掩模中对应的区域为白色才会显示。

可以看到，只有掩模的区域才会用于直方图的计算中。

很合理，是吧。

所以现在在41行针对每个区域都调用直方图方法。第一个参数是需要提取特征的图像，第二个参数是掩模区域，

这样来提取颜色直方图。

histogram方法会返回当前区域的颜色直方图表示，我们将其添加到特征列表中。

46和47行提取图像中间（椭圆）区域的颜色直方图并更新features列表。

最后，在50行像调用函数返回特征向量。

现在来快速浏览下实际的histogram方法：

Python

```
def histogram(self, image, mask):
    def histogram(self, image, mask):
        # extract a 3D color histogram from the masked region of the
        # image, using the supplied number of bins per channel; then
        # normalize the histogram
        hist = cv2.calcHist([image], [0, 1, 2], mask, self.bins,
                           [0, 180, 0, 256, 0, 256])
        hist = cv2.normalize(hist).flatten()
        # return the histogram
    return hist.
```

这里的histogram方法需要两个参数，第一个是需要描述的图像，第二个是mask，描述需要描述的图像区域。

在5和6行，通过调用cv2.calcHist计算图像掩模区域的直方图，使用构造器中的bin数目作为参数。

在7行对直方图归一化。这意味着如果我们计算两幅相同的图像，其中一幅比另一幅大50%，直方图会是相同的。对直方图进行归一化非常重要，这样每个直方图表示的就是图像中每个bin的所占的比例，而不是每个bin的个数。同样，归一化能保证不同尺寸但内容近似的图像也会在比较函数中认为是相似的。

最后，在10行向调用函数返回归一化后的3D HSV颜色直方图。

步骤2：从数据集提取特征

现在有了定义好的图像描述符，进入第二步，对数据集中的每幅图像提取特征（如颜色直方图）。提取特征并将其持久保存起来的过程一般称为“索引化”。

继续来看对假期照片数据集进行索引化的代码。创建一个新文件，命名为index.py，添加索引化所需的代码：

Python

```
# import the necessary packages
# import the necessary packages
from pyimagesearch.colordescriptor import ColorDescriptor
import argparse
import glob
import cv2
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required = True,
                help = "Path to the directory that contains the images to be indexed")
ap.add_argument("-i", "--index", required = True,
                help = "Path to where the computed index will be stored")
args = vars(ap.parse_args())
# initialize the color descriptor
```

```
16 cd = ColorDescriptor((8, 12, 3))
```

首先导入所需的模块。注意第一步中的ColorDescriptor类，这里将其放入pyimagesearch模块，以便更好的组织代码。

还需要argparse模块来处理命令行参数、glob来获取图像的文件路径，以及cv2来使用OpenCV的接口。

7-12行用来处理命令行指令。这里需要两个指令，–dataset，表示假期相册的路径。–index，表示输出的CSV文件含有图像文件名和对应的特征。

最后，在16行初始化ColorDescriptor，8 bin拥有色相、12 bin用于饱和度、3 bin用于明度。

现在所有内容都初始化了，可以从数据集提取特征了：

Python

```
# open the output index file for writing
1 # open the output index file for writing
2 output = open(args["index"], "w")
3
4 # use glob to grab the image paths and loop over them
5 for imagePath in glob.glob(args["dataset"] + "/*.png"):
6 # extract the image ID (i.e. the unique filename) from the image
7 # path and load the image itself
8 imageID = imagePath[imagePath.rfind("/") + 1:]
9 image = cv2.imread(imagePath)
10
11 # describe the image
12 features = cd.describe(image)
13
14 # write the features to file
15 features = [str(f) for f in features]
16 output.write("%s,%s\n" % (imageID, ",".join(features)))
17
18 # close the index file
19 output.close()
```

在2行代开输出文件，在5行遍历数据集中的所有图像。

对于每幅图像，可以提取一个imageID，即图像的文件名。对于这个作为示例的搜索引擎，我们假定每个文件名都是唯一的，也可以针对每幅图像生产一个UUID。在9行将从磁盘上读取图像。

现在图像载入内存了，在12行对图像使用图像描述符并提取特征。ColorDescriptor的describe方法返回由浮点数构成的列表，用来量化并表示图像。

这个数字列表，或者说特征向量，含有第一步中图像的5个区域的描述。每个区域由一个直方图表示，含有 $8 \times 12 \times 3 = 288$ 项。5个区域总共有 $5 \times 288 = 1440$ 维度。。。因此每个图像使用1440个数字量化并表示。

15和16行简单的将图像的文件名和管理的特征向量写入文件。

为了索引化我们的相册数据集，打开一个命令行输入下面的命令：

Python

```
$ python index.py --dataset dataset --index
1 $ python index.py --dataset dataset --index index.csv
```

这个脚本运行的很快，完成后将会获得一个名为index.csv的新文件。

使用你最喜欢的文本编辑器打开并查看该文件。

可以看到在.csv文件的每一行，第一项是文件名，第二项是一个数字列表。这个数字列表就是用来表示并量化图像的特征向量。

对index文件运行wc命令，可以看到已经成功对数据集中805幅图像索引化了：

Python

```
$ wc -l index.csv  
805 index.csv
```

```
1 $ wc -l index.csv  
2 805 index.csv
```

第3步：搜索

现在已经从数据集提取了特征了，接下来需要一个方法来比较这些特征，获取相似度。这就是第三步的内容，创建一个类来定义两幅图像的相似矩阵。

创建一个新文件，命名为searcher.py，让我们在这里做点神奇的事情：

Python

```
# import the necessary packages  
#  
1 # import the necessary packages  
2 import numpy as np  
3 import csv  
4  
5 class Searcher:  
6     def __init__(self, indexPath):  
7         # store our index path  
8         self.indexPath = indexPath  
9  
10    def search(self, queryFeatures, limit = 10):  
11        # initialize our dictionary of results  
12        results = {}
```

首先先导入NumPy用于数值计算，csv用于方便的处理index.csv文件。

在第5行定义Searcher类。Searcher类的构造器只需一个参数，indexPath，用于表示index.csv文件在磁盘上的路径。

要实际执行搜索，需要在第10行调用search方法。该方法需要两个参数，queryFeatures是提取自待搜索图像（如向CBIR系统提交并请求返回相似图像的图像），和返回图像的数目的最大值。

最后，在12行初始化results字典。在这里，字典有很用的用途，每个图像有唯一的imageID，可以作为字典的键，而相似度作为字典的值。

好了，现在将注意力放在这里。这里是发生神奇的地方：

Python

```
# open the index file  
for reading  
#  
1 # open the index file for reading  
2 with open(self.indexPath) as f:  
3     # initialize the CSV reader  
4     reader = csv.reader(f)
```

```

5
6 # loop over the rows in the index
7 for row in reader:
8 # parse out the image ID and features, then compute the
9 # chi-squared distance between the features in our index
10 # and our query features
11 features = [float(x) for x in row[1:]]
12 d = self.chi2_distance(features, queryFeatures)
13
14 # now that we have the distance between the two feature
15 # vectors, we can update the results dictionary -- the
16 # key is the current image ID in the index and the
17 # value is the distance we just computed, representing
18 # how 'similar' the image in the index is to our query
19 results[row[0]] = d
20
21 # close the reader
22 f.close()
23
24 # sort our results, so that the smaller distances (i.e. the
25 # more relevant images are at the front of the list)
26 results = sorted([(v, k) for (k, v) in results.items()])
27
28 # return our (limited) results
29 return results[:limit]

```

在1行打开index.csv文件，在3行获取CSV读取器的句柄，接着在6行循环读取index.csv文件的每一行。

对于每一行，提取出索引化后的图像的颜色直方图，用11行的chi2_distance函数将其与待搜索的图像特征进行比较，该函数在下面介绍。

在32行使用唯一的图像文件名作为键，用与待查找图像的与索引后的图像的相似度作为值来更新results字典。

最后，将results字典根据相似度升序排序。

卡方相似度为零的图片表示完全相同。相似度数值越高，表示两幅图像差别越大。

说到卡方相似度，看下面的源码：

Python

```

def chi2_distance(self,
histA, histB, eps = 1e-
1 def chi2_distance(self, histA, histB, eps = 1e-10):
2     # compute the chi-squared distance
3     d = 0.5 * np.sum([(a - b) ** 2 / (a + b + eps)
4         for (a, b) in zip(histA, histB)])
5
6     # return the chi-squared distance
7     return d

```

chi2_distance函数需要两个参数，即用来进行比较的两个直方图。可选的eps值用来预防除零错误。

这个函数的名称来自皮尔森的卡方测试统计，用来比较离散概率分布。

由于比较的是颜色直方图，根据概率分布的定义，卡方函数是个完美的选择。

一般来说，直方图两端的值的差别并不重要，可以使用权重对其进行处理，卡方距离函数就是这么做的。

还能跟的上吗？我保证，最后一步是最简单的，仅仅需要将前面的各部分组合在一起。

第四步：执行搜索

如果我告诉你，执行搜索是最简单的一步，你信吗？实际上，只需一个驱动程序导入前面定义的所有的模块，将其依次组合成具有完整功能的CBIR系统。

所以新建最后一个文件，命名为search.py，这样我们的例子就能完成了：

Python

```
# import the necessary packages
1 # import the necessary packages
2 from pyimagesearch.colordescriptor import ColorDescriptor
3 from pyimagesearch.searcher import Searcher
4 import argparse
5 import cv2
6
7 # construct the argument parser and parse the arguments
8 ap = argparse.ArgumentParser()
9 ap.add_argument("-i", "--index", required = True,
10 help = "Path to where the computed index will be stored")
11 ap.add_argument("-q", "--query", required = True,
12 help = "Path to the query image")
13 ap.add_argument("-r", "--result-path", required = True,
14 help = "Path to the result path")
15 args = vars(ap.parse_args())
16
17 # initialize the image descriptor
18 cd = ColorDescriptor((8, 12, 3))
```

首先导入所需的包，导入第一步的ColorDescriptor来提取待查找图像的特征；导入第三步定义的Searcher类，用于执行执行实际的搜索。

argparse和cv2模块一直会导入。

在8-15行处理命令行参数。我们需要用一个-index来表示index.csv文件的位置。

还需要-query来表示带搜索图像的存储路径。该图像将与数据集中的每幅图像进行比较。目标是找到数据集中欧给你与待搜索图像相似的图像。

想象一下，使用Google搜索并输入“Python OpenCV tutorials”，会希望获得与Python和OpenCV相关的信息。

与之相同，如果针对相册构建一个图像搜索引擎，提交了一副关于云、大海上的帆船的图像，希望通过图像搜索引擎获得相似的图像。

接着需要一个-result-path，用来表示相册数据集的路径。通过这个命令可以选择不同的数据集，向用户显示他们所需要的最终结果。

最后，在18行使用图像描述符提取相同的参数，就如同在索引化那一步做的一样。如果我们是为了比较图像的相似度（事实也正是如此），就无需改变数据集中颜色直方图的bin的数目。

直接将第三步中使用的直方图bin的数目作为参数在第四步使用。

这样会保证图像的描述是连续且可比较的。

现在到了进行真正比较的时候：

Python

```
# load the query image
1 # load the query image and describe it
```

```
2 query = cv2.imread(args["query"])
3 features = cd.describe(query)
4
5 # perform the search
6 searcher = Searcher(args["index"])
7 results = searcher.search(features)
8
9 # display the query
10 cv2.imshow("Query", query)
11
12 # loop over the results
13 for (score, resultID) in results:
14 # load the result image and display it
15 result = cv2.imread(args["result_path"] + "/" + resultID)
16 cv2.imshow("Result", result)
17 cv2.waitKey(0)
```

在2行从磁盘读取待搜索图像，在3行提取该图像的特征。

在6和7行使用提取到的特征进行搜索，返回经过排序后的结果列表。

到此，所需做的就是将结果显示给用户。

在9行显示出待搜索的图像。接着在13-17行遍历搜索结果，将相应的图像显示在屏幕上。

所有这些工作完成后，就可以实际操作了。

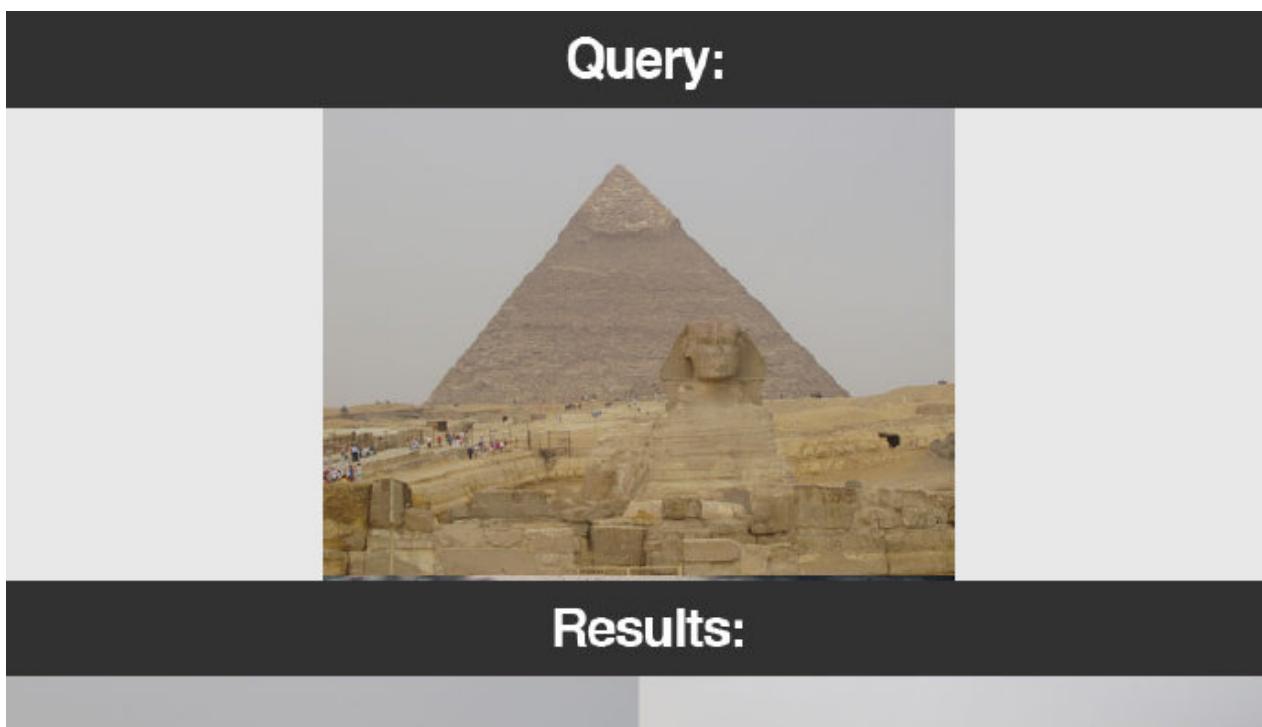
继续阅读，看最终效果如何。

CBIR系统实战

打开终端，切换到代码所在的目录，执行下面的命令：

Python

```
$ python search.py --index index.csv --query queries/108100.png --result-path dataset
```



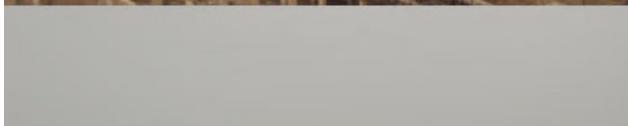




图18：在相册中搜索含有埃及金字塔的图像。

第一幅图像是待搜索的埃及金字塔。我们的目标是在相册中找到相似的图像。可以看到，在相册中找到了去金字塔游玩拍摄的照片。

我们还游览的埃及其他地方，所以用其他照片搜索试试：

Python

```
$ python search.py --index index.csv --query  
1 $ python search.py --index index.csv --query queries/115100.png --result-path dataset
```

Query:



Results:







图19：使用搜索引擎搜索埃及其他地方的图片，注意图中蓝天的位置。

注意我们的搜索图像中，上半部分是蓝天。中间和下半部分是褐色的建筑和土地。

可以肯定，图像搜索引擎会返回上半部分是蓝天，下半部分是棕褐色建筑和沙子的图像。

这是因为我们使用了本文开头介绍的基于区域的颜色直方图描述符。使用这种图像描述符可以粗略的针对每个区域执行，最后的结果中会含有图像每个区域的像素的密度。

旅途的最后一站是海滩，用下面的命令搜索海滩上的图像：

Python

```
$ python search.py --index index.csv --query  
1 $ python search.py --index index.csv --query queries/103300.png --result-path dataset
```

Query:

Results:



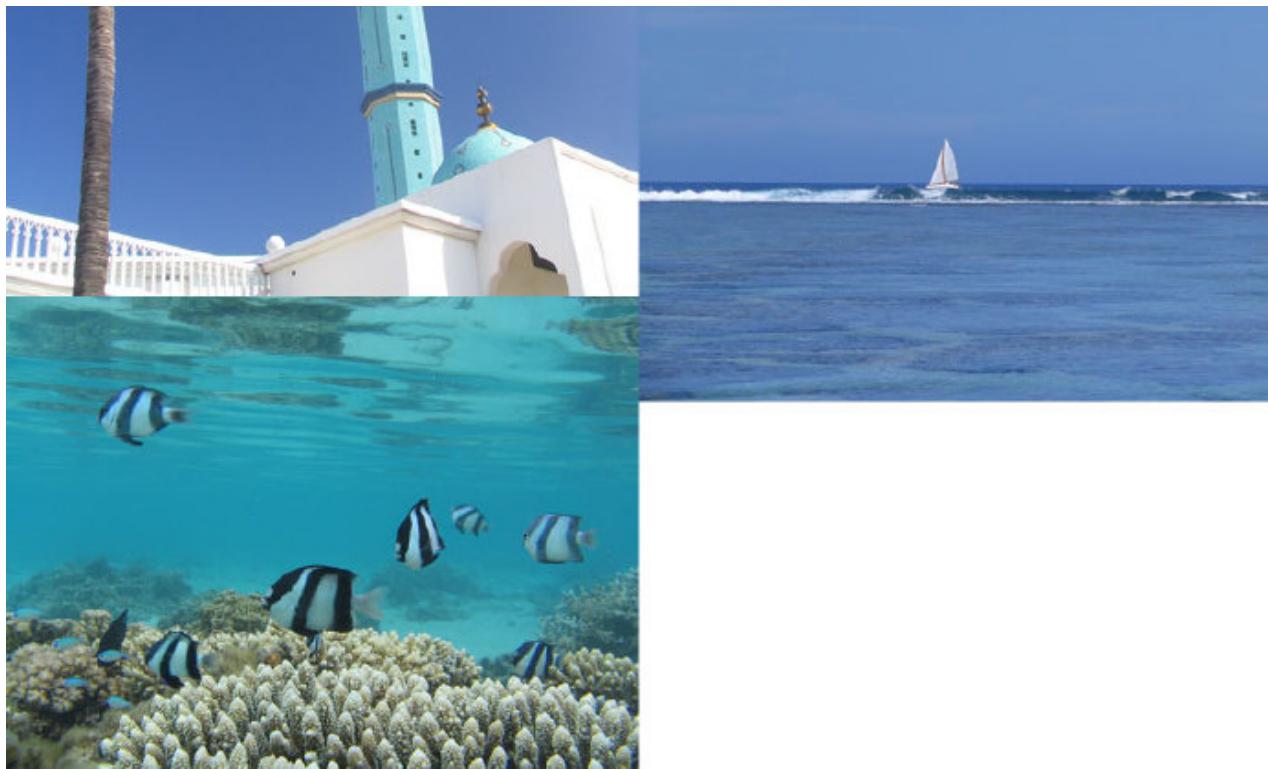


图20：使用OpenCV构建CBIR系统来搜索相册。

注意，前3个搜索结果是在相同地点拍摄到的图像。其他图像都含有蓝色的区域。

当然，没有潜水的海滩之旅是不完整的。

Python

```
$ python search.py --index index.csv --query queries/103100.png --result-path dataset  
1 $ python search.py --index index.csv --query queries/103100.png --result-path dataset
```

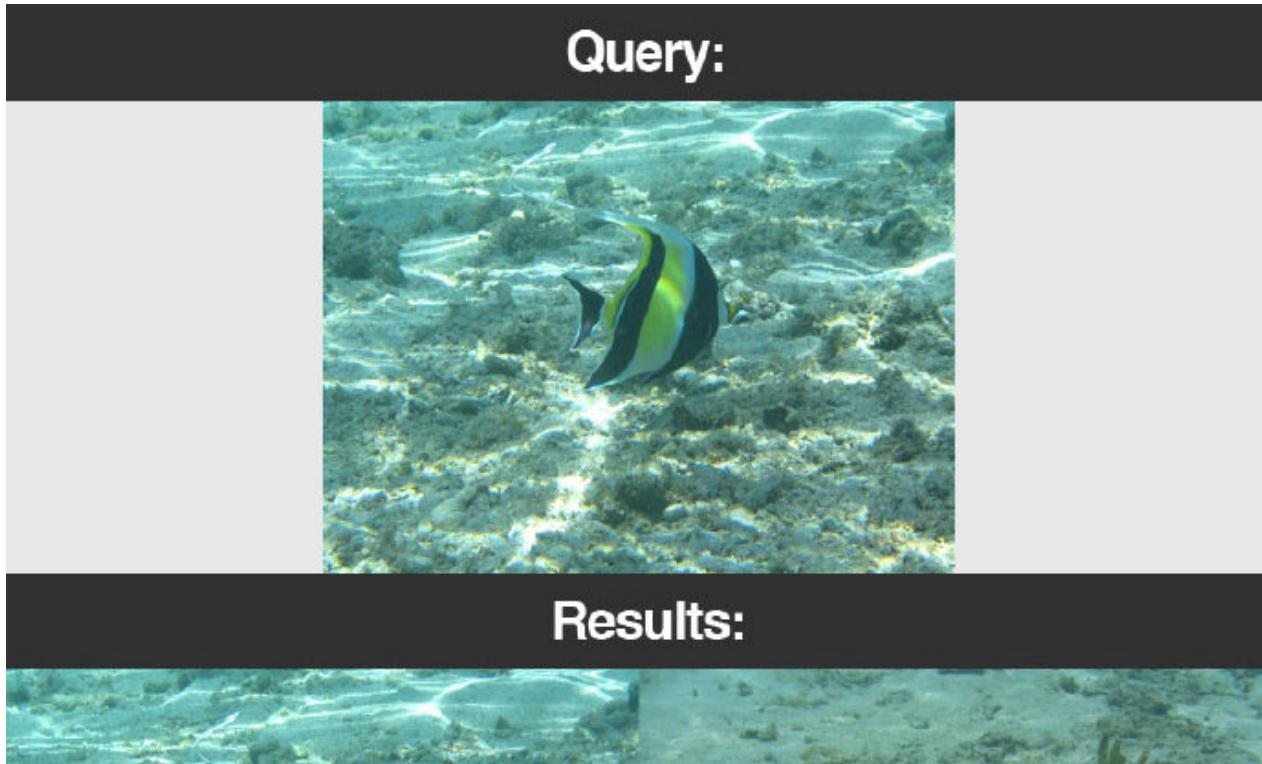






图21：图像搜索引起再次返回相关的结果。这次是水下冒险。

结果非常棒。前5个结果是同一条鱼，前10幅有9幅是水下探险。

最后，一天的旅途结束了，到了观看夕阳的时候：

Python

```
$ python search.py --index index.csv --query queries/127502.png --result-path dataset
```

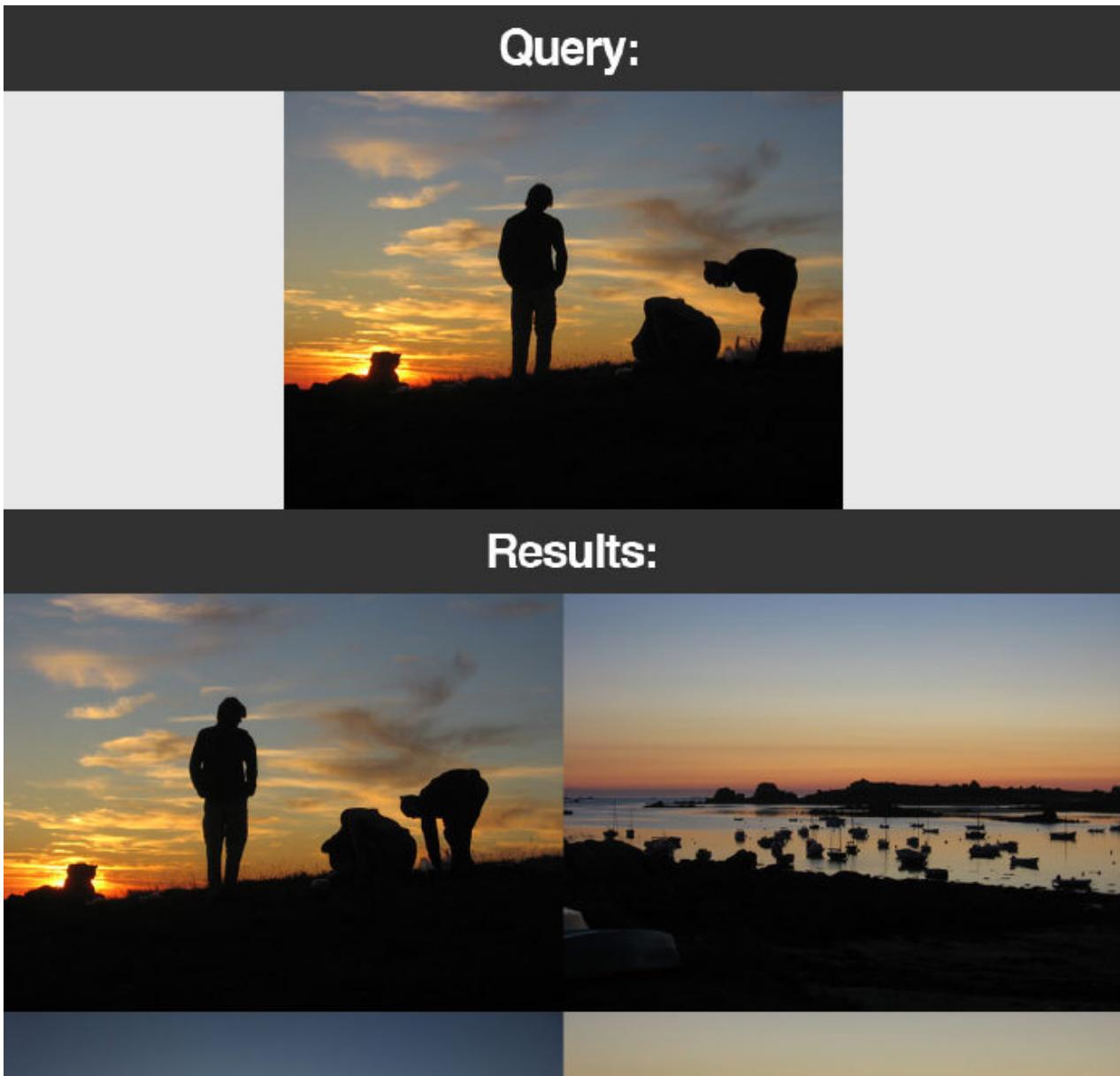




图22：这个OpenCV图像搜索引擎可以查找到相册集中含有夕阳的相片。

搜索结果非常棒，所有的结果都含有夕阳。

这样，你就有了第一个图像搜索引擎：

总结

本文介绍了如何构建一个图像搜索引擎，来查找相册中的图像。

使用颜色直方图对相册中的图像的颜色部分进行分类。接着，使用颜色描述符索引化相册，提取相册中每一副图像的颜色直方图。

使用卡方距离比较图像，这是比较离散概率分布最常见的选择。

接着，实现了提交待搜索图像和返回查找结果的逻辑。

下一步

接下来该干什么？

可以看到，使用命令行是与这个图像搜索引擎交互的唯一方式。这样还不是太吸引人

下一篇文章将探索如何将这个图像搜索引擎封装进一个Python网络框架中，让其更易于使用。

下载：

代码和数据集总共有约**200mb**。如果读者想要下载文中用到的代码和图像。请在原文中输入你的邮箱地址，我会给你一个下载代码和数据集的链接。这个链接不仅能下载到代码的zip压缩包，还会收到我送给你的11页关于计算机视觉和图像搜索引擎的资源指南，其中含有文中没有介绍到的一些技术！是不是很不错？别犹豫，赶快在下面输入你的邮箱地址，我会立即给你发送代码的！[在原文填写邮箱](#)（要查看两次邮箱，还得翻墙。不想来回倒腾的童鞋，请戳链接：<http://pan.baidu.com/s/1ntsoamP>）

打赏支持我翻译更多好文章，谢谢！

[打赏译者](#)

打赏支持我翻译更多好文章，谢谢！

任选一种支付方式

微信扫一扫转账



向孙波翔(Sun)转账

赞赏你的文章

¥1.00

支付宝扫一扫，向我付款



¥1.00

赞赏你发在伯乐在线的文章

4 赞 13 收藏 [22 评论](#)

关于作者：[Daetalus](#)



Pyston核心开源开发者。熟悉CPython实现，关注Python科学计算。[个人主页](#) · [我的文章](#) · 28 ·

用Pandas完成Excel中常见的任务

本文由 [伯乐在线 - 艾凌风](#) 翻译, [Daetalus](#) 校稿。未经许可, 禁止转载!
英文出处: [pbpython.com](#)。欢迎加入[翻译组](#)。

引言

本文的目的, 是向您展示如何使用pandas来执行一些常见的Excel任务。有些例子比较琐碎, 但我觉得展示这些简单的东西与那些你可以在其他地方找到的复杂功能同等重要。作为额外的福利, 我将会进行一些模糊字符串匹配, 以此来展示一些小花样, 以及展示pandas是如何利用完整的Python模块系统去做一些在Python中是简单, 但在Excel中却很复杂的事情的。

有道理吧? 让我们开始吧。

为某行添加求和项

我要介绍的第一项任务是把某几列相加然后添加一个总和栏。

首先我们将[excel数据](#)导入到pandas数据框架中。

Python

```
; html-script: false
import pandas as pd
1 ; html-script: false ]import pandas as pd
2 import numpy as np
3 df = pd.read_excel("excel-comp-data.xlsx")
4 df.head()
```

	account	name	street	city	state	postal-code	Jan	Feb	Mar
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000
3	109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000

我们想要添加一个总和栏来显示Jan、Feb和Mar三个月的销售总额。

在Excel和pandas中这都是简单直接的。对于Excel, 我在J列中添加了公式`=SUM(G2:I2)`。在Excel中看上去是这样的:

	J2	f2	=SUM(G2:I2)	A	B	C	D	E	F	G	H	I	J
1				account	name	street	city	state	postal-code	Jan	Feb	Mar	total
2				211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000	107000
3				320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000	175000
4				648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000	246000
5				109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000	175000
6				121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000	317000
7				132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000	305000
8				145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	62000	120000	70000	252000
9				205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	145000	95000	35000	275000
10				209744	Champlin-Morar	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	70000	95000	35000	200000
11				212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	70000	120000	35000	225000
12				214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	45000	120000	55000	220000
13				231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000	322000
14				242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000	317000
15				268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000	210000
16				273274	McDermott PLC	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	150000	120000	70000	340000
17													

下面, 我们是这样在pandas中操作的:

Python

```
; html-script: false
[|df["total"] = df["Jan"] +
1 ; html-script: false ]df["total"] = df["Jan"] + df["Feb"] + df["Mar"]
2 df.head()
```

	account	name	street	city	state	postal-code	Jan	Feb	Mar	total
0	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000	107000
1	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000	175000
2	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000	246000
3	109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000	175000
4	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000	317000

接下来，让我们对各列计算一些汇总信息以及其他值。如下Excel表所示，我们要做这些工作：

	A	B	C	D	E	F	G	H	I	J
1	account	name	street	city	state	postal-code	Jan	Feb	Mar	total
2	211829	Kerluke, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	10000	62000	35000	107000
3	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	95000	45000	35000	175000
4	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	91000	120000	35000	246000
5	109996	D'Amore, Gleichner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	45000	120000	10000	175000
6	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	162000	120000	35000	317000
7	132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	150000	120000	35000	305000
8	145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	62000	120000	70000	252000
9	205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	145000	95000	35000	275000
10	209744	Champlin-Morar	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	70000	95000	35000	200000
11	212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	70000	120000	35000	225000
12	214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	45000	120000	55000	220000
13	231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000	322000
14	242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000	317000
15	268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000	210000
16	273274	McDermott PLC	8917 Bergstrom Meadow	Kathyneborough	Delaware	27933	150000	120000	70000	340000
17							1462000	1507000	717000	3686000
18										

如你所见，我们在表示月份的列的第17行添加了SUM(G2:G16)，来取得每月的总和。

进行在pandas中进行列级别的分析很简单。下面是一些例子：

Python

```
; html-script: false
[df["Jan"].sum(),
1 ; html-script: false ]df["Jan"].sum(), df["Jan"].mean(),df["Jan"].min(),df["Jan"].max()
```

Python

```
; html-script: false ]
(1462000)
1 ; html-script: false ](1462000, 97466.66666666672, 10000, 162000)
```

现在我们要把每月的总和相加得到它们的和。这里pandas和Excel有点不同。在Excel的单元格里把每个月的总和相加很简单。由于pandas需要维护整个DataFrame的完整性，所以需要一些额外的步骤。

首先，建立所有列的总和栏

Python

```
; html-script: false
[sum_row=df[["Jan","Fe
1 ; html-script: false ]sum_row=df[["Jan","Feb","Mar","total"]].sum()
2 sum_row
```

Python

```
; html-script: false ]Jan
1462000
1 ; html-script: false ]Jan 1462000
2 Feb 1507000
```

```
3 Mar    717000  
4 total   3686000  
5 dtype: int64
```

这很符合直觉，不过如果你希望将总和值显示为表格中的单独一行，你还需要做一些微调。

我们需要把数据进行变换，把这一系列数字转换为DataFrame，这样才能更加容易的把它合并进已经存在的数据中。T函数可以让我们把按行排列的数据变换为按列排列。

Python

```
; html-script: false  
| df_sum=pd.DataFrame  
1 ; html-script: false ]df_sum=pd.DataFrame(data=sum_row).T  
2 df_sum
```

Jan	Feb	Mar	total
0 1462000	1507000	717000	3686000

在计算总和之前我们要做的最后一件事情是添加丢失的列。我们使用reindex来帮助我们完成。技巧是添加全部的列然后让pandas去添加所有缺失的数据。

Python

```
; html-script: false  
| df_sum=df_sum.reindex  
1 ; html-script: false ]df_sum=df_sum.reindex(columns=df.columns)  
2 df_sum
```

account	name	street	city	state	postal-code	Jan	Feb	Mar	total
0	NaN	NaN	NaN	NaN	NaN	1462000	1507000	717000	3686000

现在我们已经有了一个格式良好的DataFrame，我们可以使用append来把它加入到已有的内容中。

Python

```
; html-script: false  
| df_final=df.append(df_s  
1 ; html-script: false ]df_final=df.append(df_sum,ignore_index=True)  
2 df_final.tail()
```

	account	name	street	city	state	postal-code	Jan	Feb	Mar	total
11	231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000	322000
12	242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000	317000
13	268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000	210000
14	273274	McDermott PLC	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	150000	120000	70000	340000
15	NaN	NaN	NaN	NaN	NaN	NaN	1462000	1507000	717000	3686000

额外的数据变换

另外一个例子，让我们尝试给数据集添加状态的缩写。

对于Excel，最简单的方式是添加一个新的列，对州名使用vlookup函数并填充缩写栏。

我进行了这样的操作，下面是其结果的截图：

G2	B	C	D	E	F	G	H	I	J	K	
1	A	name	street	city	state	postal-code	abbrev	Jan	Feb	Mar	total
2	211829	Kerluge, Koepp and Hilpert	34456 Sean Highway	New Jaycob	Texas	28752	TX	10000	62000	35000	1070
3	320563	Walter-Trantow	1311 Alvis Tunnel	Port Khadijah	NorthCarolina	38365	#N/A	95000	45000	35000	1750
4	648336	Bashirian, Kunde and Price	62184 Schamberger Underpass Apt. 231	New Lilianland	Iowa	76517	IA	91000	120000	35000	2460
5	109996	D'Amore, Glechner and Bode	155 Fadel Crescent Apt. 144	Hyattburgh	Maine	46021	ME	45000	120000	10000	1750
6	121213	Bauch-Goldner	7274 Marissa Common	Shanahanchester	California	49681	CA	162000	120000	35000	3170
7	132971	Williamson, Schumm and Hettinger	89403 Casimer Spring	Jeremieburgh	Arkansas	62785	AR	150000	120000	35000	3050
8	145068	Casper LLC	340 Consuela Bridge Apt. 400	Lake Gabriellaton	Mississippi	18008	#N/A	62000	120000	70000	2520
9	205217	Kovacek-Johnston	91971 Cronin Vista Suite 601	Deronville	Rhodelsland	53461	#N/A	145000	95000	35000	2750
10	209744	Champlin-Morar	26739 Grant Lock	Lake Juliannton	Pennsylvania	64415	PA	70000	95000	35000	2000
11	212303	Gerhold-Maggio	366 Maggio Grove Apt. 998	North Ras	Idaho	46308	ID	70000	120000	35000	2250
12	214098	Goodwin, Homenick and Jerde	649 Cierra Forks Apt. 078	Rosaberg	Tennessee	47743	#N/A	45000	120000	55000	2200
13	231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	#N/A	150000	10000	162000	3220
14	242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	IA	162000	120000	35000	3170
15	268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	#N/A	55000	120000	35000	2100
16	273274	McDermott PLC	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	DE	150000	120000	70000	3400
17								1462000	1507000	717000	36860
18											

你可以注意到，在进行了vlookup后，有一些数值并没有被正确的取得。这是因为我们拼错了一些州的名字。在Excel中处理这一问题是一个巨大的挑战（对于大型数据集而言）

幸运的是，使用pandas我们可以利用强大的python生态系统。考虑如何解决这类麻烦的数据问题，我考虑进行一些模糊文本匹配来决定正确的值。

幸运的是其他人已经做了很多这方面的工作。[fuzzywuzzy](#)库包含一些非常有用的函数来解决这类问题。首先要确保你安装了他。

我们需要的另外一段代码是州名与其缩写的映射表。而不是亲自去输入它们，谷歌一下你就能找到这段代码[code](#)。

首先导入合适的fuzzywuzzy函数并且定义我们的州名映射表。

Python

```
; html-script: false ]from fuzzywuzzy import fuzz
state_to_code = {"VERMONT": "VT", "GEORGIA": "GA", "IOWA": "IA", "Armed Forces Pacific": "AP", "GUAM": "GU",
"KANSAS": "KS", "FLORIDA": "FL", "AMERICAN SAMOA": "AS", "NORTH CAROLINA": "NC", "HAWAII": "HI",
"NEW YORK": "NY", "CALIFORNIA": "CA", "ALABAMA": "AL", "IDAHO": "ID", "FEDERATED STATES OF MICRONESIA": "FM",
"Armed Forces Americas": "AA", "DELWARE": "DE", "ALASKA": "AK", "ILLINOIS": "IL",
"Armed Forces Africa": "AE", "SOUTH DAKOTA": "SD", "CONNECTICUT": "CT", "MONTANA": "MT", "MASSACHUSETTS": "MA",
"PUERTO RICO": "PR", "Armed Forces Canada": "AE", "NEW HAMPSHIRE": "NH", "MARYLAND": "MD", "NEW MEXICO": "NM",
"MISSISSIPPI": "MS", "TENNESSEE": "TN", "PALAU": "PW", "COLORADO": "CO", "Armed Forces Middle East": "AE",
"NEW JERSEY": "NJ", "UTAH": "UT", "MICHIGAN": "MI", "WEST VIRGINIA": "WV", "WASHINGTON": "WA",
"MINNESOTA": "MN", "OREGON": "OR", "VIRGINIA": "VA", "VIRGIN ISLANDS": "VI", "MARSHALL ISLANDS": "MH",
"WYOMING": "WY", "OHIO": "OH", "SOUTH CAROLINA": "SC", "INDIANA": "IN", "NEVADA": "NV", "LOUISIANA": "LA",
"NORTHERN MARIANA ISLANDS": "MP", "NEBRASKA": "NE", "ARIZONA": "AZ", "WISCONSIN": "WI", "NORTH DAKOTA": "ND",
"Armed Forces Europe": "AE", "PENNSYLVANIA": "PA", "OKLAHOMA": "OK", "KENTUCKY": "KY", "RHODE ISLAND": "RI",
"DISTRICT OF COLUMBIA": "DC", "ARKANSAS": "AR", "MISSOURI": "MO", "TEXAS": "TX", "MAINE": "ME"}
```

这里有些介绍模糊文本匹配函数如何工作的例子。

Python

```
; html-script: false ]process.extractOne("Mi
1 ; html-script: false ]process.extractOne("Minnesota", choices=state_to_code.keys())
```

Python

```
; html-script: false ](&#039;MINNESOTA&#
1 ; html-script: false ](&#039;MINNESOTA&#, 95)
1 ; html-script: false ]process.extractOne("AlaBAMMazzz", choices=state_to_code.keys(), score_cutoff=80)
```

现在我知道它是如何工作的了，我们创建自己的函数来接受州名这一列的数据然后把他转换为一个有效的缩写。这里我们使用score_cutoff的值为80。你可以做一些调整，看看哪个值对你的数据来说比较好。你会注意到，返回值要么是一个有效的缩写，要么是一个np.nan 所以域中会有一些有效的值。

Python

```
; html-script: false ]def convert_state(row):
    abbrev = process.extractOne(row["state"], choices=state_to_code.keys(), score_cutoff=80)
    if abbrev:
        return state_to_code[abbrev[0]]
    return np.nan
```

把这列添加到我们想要填充的单元格，然后用NaN填充它

Python

```
; html-script: false
| df_final.insert(6,
1 ; html-script: false ]df_final.insert(6, "abbrev", np.nan)
2 df_final.head()
```

	account	name	street	city	state	postal-code	Jan	Feb	Mar	total
11	231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	150000	10000	162000	322000
12	242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	162000	120000	35000	317000
13	268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	55000	120000	35000	210000
14	273274	McDermott PLC	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	150000	120000	70000	340000
15	NaN	NaN	NaN	NaN	NaN	NaN	1462000	1507000	717000	3686000

我们使用apply 来把缩写添加到合适的列中。

Python

```
; html-script: false
| df_final['abbrev'] =
1 ; html-script: false ]df_final['abbrev'] = df_final.apply(convert_state, axis=1)
2 df_final.tail()
```

	account	name	street	city	state	postal-code	abbrev	Jan	Feb	Mar	total
11	231907	Hahn-Moore	18115 Olivine Throughway	Norbertomouth	NorthDakota	31415	ND	150000	10000	162000	322000
12	242368	Frami, Anderson and Donnelly	182 Bertie Road	East Davian	Iowa	72686	IA	162000	120000	35000	317000
13	268755	Walsh-Haley	2624 Beatty Parkways	Goodwinmouth	Rhodelsland	31919	RI	55000	120000	35000	210000
14	273274	McDermott PLC	8917 Bergstrom Meadow	Kathryneborough	Delaware	27933	DE	150000	120000	70000	340000
15	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1462000	1507000	717000	3686000

我觉的这很酷。我们已经开发出了一个非常简单的流程来智能的清理数据。显然，当你只有15行左右数据的时候这没什么了不起的。但是如果是15000行呢？在Excel中你就必须进行一些人工清理了。

分类汇总

在本文的最后一节中，让我们按州来做一些分类汇总（subtotal）。

在Excel中，我们会用subtotal 工具来完成。

输出如下：

abbrev	Jan	Feb	Mar	total
AR Total	\$ 150,000	\$ 120,000	\$ 35,000	\$ 305,000
CA Total	\$ 162,000	\$ 120,000	\$ 35,000	\$ 317,000
DE Total	\$ 150,000	\$ 120,000	\$ 70,000	\$ 340,000
IA Total	\$ 253,000	\$ 240,000	\$ 70,000	\$ 563,000
ID Total	\$ 70,000	\$ 120,000	\$ 35,000	\$ 225,000
MC Total	\$ 62,000	\$ 120,000	\$ 70,000	\$ 252,000
ME Total	\$ 45,000	\$ 120,000	\$ 10,000	\$ 175,000
NC Total	\$ 95,000	\$ 45,000	\$ 35,000	\$ 175,000
ND Total	\$ 150,000	\$ 10,000	\$ 162,000	\$ 322,000
PA Total	\$ 70,000	\$ 95,000	\$ 35,000	\$ 200,000
RI Total	\$ 200,000	\$ 215,000	\$ 70,000	\$ 485,000
TN Total	\$ 45,000	\$ 120,000	\$ 55,000	\$ 220,000
TX Total	\$ 10,000	\$ 62,000	\$ 35,000	\$ 107,000
Grand Total	\$ 1,462,000	\$ 1,507,000	\$ 717,000	\$ 3,686,000

在pandas中创建分类汇总，是使用groupby 来完成的。

Python

```
; html-script: false
1 df_sub=df_final[["abbrev"]]
2 ; html-script: false df_sub=df_final[["abbrev","Jan","Feb","Mar","total"]].groupby('abbrev').sum()
2 df_sub
```

abbrev	Jan	Feb	Mar	total
AR	150000	120000	35000	305000
CA	162000	120000	35000	317000
DE	150000	120000	70000	340000
IA	253000	240000	70000	563000
ID	70000	120000	35000	225000
ME	45000	120000	10000	175000
MS	62000	120000	70000	252000
NC	95000	45000	35000	175000
ND	150000	10000	162000	322000
PA	70000	95000	35000	200000
RI	200000	215000	70000	485000
TN	45000	120000	55000	220000
TX	10000	62000	35000	107000

然后，我们想要通过对data frame中所有的值使用 applymap 来把数据单位格式化为货币。

Python

```
; html-script: false
1 def money(x):
2     pass
```

```

1 ; html-script: false ]def money(x):
2   return "${:.0f}".format(x)
3
4 formatted_df = df_sub.applymap(money)
5 formatted_df

```

	Jan	Feb	Mar	total
abbrev				
AR	\$150,000	\$120,000	\$35,000	\$305,000
CA	\$162,000	\$120,000	\$35,000	\$317,000
DE	\$150,000	\$120,000	\$70,000	\$340,000
IA	\$253,000	\$240,000	\$70,000	\$563,000
ID	\$70,000	\$120,000	\$35,000	\$225,000
ME	\$45,000	\$120,000	\$10,000	\$175,000
MS	\$62,000	\$120,000	\$70,000	\$252,000
NC	\$95,000	\$45,000	\$35,000	\$175,000
ND	\$150,000	\$10,000	\$162,000	\$322,000
PA	\$70,000	\$95,000	\$35,000	\$200,000
RI	\$200,000	\$215,000	\$70,000	\$485,000
TN	\$45,000	\$120,000	\$55,000	\$220,000
TX	\$10,000	\$62,000	\$35,000	\$107,000

格式化看上去进行的很顺利，现在我们可以像之前那样获取总和了。

Python

```

; html-script: false
]sum_row=df_sub[["Jan"]
1 ; html-script: false ]sum_row=df_sub[["Jan","Feb","Mar","total"]].sum()
2 sum_row

```

Python

```

; html-script: false ]Jan
1462000
1 ; html-script: false ]Jan    1462000
2 Feb    1507000
3 Mar    717000
4 total  3686000
5 dtype: int64

```

把值变换为列然后进行格式化。

Python

```

; html-script: false
]df_sub_sum=pd.DataFrame.T
1 ; html-script: false ]df_sub_sum=pd.DataFrame(data=sum_row).T
2 df_sub_sum=df_sub_sum.applymap(money)
3 df_sub_sum

```

Jan	Feb	Mar	total
\$1,462,000	\$1,507,000	\$717,000	\$3,686,000

最后，把总和添加到DataFrame中。

Python

```

; html-script: false
]final_table =
1 ; html-script: false ]final_table = formatted_df.append(df_sub_sum)
2 final_table

```

	Jan	Feb	Mar	total
AR	\$150,000	\$120,000	\$35,000	\$305,000
CA	\$162,000	\$120,000	\$35,000	\$317,000
DE	\$150,000	\$120,000	\$70,000	\$340,000
IA	\$253,000	\$240,000	\$70,000	\$563,000
ID	\$70,000	\$120,000	\$35,000	\$225,000
ME	\$45,000	\$120,000	\$10,000	\$175,000
MS	\$62,000	\$120,000	\$70,000	\$252,000
NC	\$95,000	\$45,000	\$35,000	\$175,000
ND	\$150,000	\$10,000	\$162,000	\$322,000
PA	\$70,000	\$95,000	\$35,000	\$200,000
RI	\$200,000	\$215,000	\$70,000	\$485,000
TN	\$45,000	\$120,000	\$55,000	\$220,000
TX	\$10,000	\$62,000	\$35,000	\$107,000
Total	\$1,462,000	\$1,507,000	\$717,000	\$3,686,000

你可以注意到总和行的索引号是'0'。我们想要使用rename来重命名它。

Python

```
; html-script: false
[final_table =
1 ; html-script: false ]final_table = final_table.rename(index={0:"Total"})
2 final_table
```

	Jan	Feb	Mar	total
AR	\$150,000	\$120,000	\$35,000	\$305,000
CA	\$162,000	\$120,000	\$35,000	\$317,000
DE	\$150,000	\$120,000	\$70,000	\$340,000
IA	\$253,000	\$240,000	\$70,000	\$563,000
ID	\$70,000	\$120,000	\$35,000	\$225,000
ME	\$45,000	\$120,000	\$10,000	\$175,000
MS	\$62,000	\$120,000	\$70,000	\$252,000
NC	\$95,000	\$45,000	\$35,000	\$175,000
ND	\$150,000	\$10,000	\$162,000	\$322,000
PA	\$70,000	\$95,000	\$35,000	\$200,000
RI	\$200,000	\$215,000	\$70,000	\$485,000
TN	\$45,000	\$120,000	\$55,000	\$220,000
TX	\$10,000	\$62,000	\$35,000	\$107,000
Total	\$1,462,000	\$1,507,000	\$717,000	\$3,686,000

结论

到目前为止，大部分人都已经知道使用pandas可以对数据做很多复杂的操作——就如同Excel一样。因为我一直在学习pandas，但我发现我还是会尝试记忆我是如何在Excel中完成这些操作的而不是在pandas中。我意识到把它俩作对比似乎不是很公平——它们是完全不同的工具。但是，我希望接触到哪些了解Excel并且想要学习一些可以满足分析他们数据需求的其他替代工具的那些人。我希望这些例子可以帮助到其他人，让他们有信心认为他们可以使用pandas来替换他们零碎复杂的Excel，进行数据操作。

我发现这个练习会帮助我加强记忆。我希望这对你来说同样有帮助。如果你有一些其他的Excel任务想知道如何用pandas来完成它，请通过评论来告诉我，我会尽力帮助你。

[打赏支持我翻译更多好文章，谢谢！](#)

[打赏译者](#)

[打赏支持我翻译更多好文章，谢谢！](#)

任选一种支付方式

微信扫一扫转账



向摆铁少年ai转账

朕看的过瘾，赏小艾！

¥1.98

支付宝扫一扫，向我付款



¥1.99

赞赏你在伯乐在线的文章

1 赞 5 收藏 [评论](#)

关于作者：[艾凌风](#)



初入职场小码农;翻译组的勤务员;C/Python/在线教育/英文翻译 [个人主页](#) · [我的文章](#) · 71 ·

13岁Python开发者写给青少年的多人游戏编程 (下)

本文由 [伯乐在线 - justyoung](#) 翻译, [黄利民](#) 校稿。未经许可, 禁止转载!
英文出处: [Julian Meyer](#)。欢迎加入[翻译组](#)。

欢迎回到我们面向青少年的多人游戏编程教程第二部分!

在教程的第一部分, 你完成了游戏客户端的大部分代码。你编写的代码在游戏界面上绘制了网格线, 并允许玩家在网格线上放置新的线条。

在教程的第二部分, 也是本教程的最后一部分, 你将完成游戏服务端的编写, 通过客户端与服务端的连接通信, 你将能实现多玩家游戏。让我们开始吧!

准备开始吧

制作一个多玩家游戏的基本思想是这样的, 你和你的朋友都需要一个游戏客户端, 你们的客户端程序需要与一个服务端程序连接, 这个服务端协调客户端程序之间的通信。

你将使用PodSixNet模块来为Boxes游戏创建服务器程序, PodSixNet是Chris McCormick创建的一个简单、轻量级的Python网络库。从外部看, PodSixNet能让玩家与服务器程序通信。你将会发现使用PodSixNet你能容易地完成你所要的网络操作。在[这里](#)你可以学习更多关于PodSixNet的知识。

PodSixNet是这样工作的: 首先你在一个指定的端口上运行定制的PodSixNet服务器程序 (你可以将它想象成一个带有指定的地址的邮箱)。然后你运行客户端程序, 客户端程序通过这个端口与服务器程序通信。

我已经将PodSixNet的源代码放置在你之前下载的项目资源文件里了。但是, 如果你想下载一个新的PodSixNet版本, 你可以在这里下载: [PodSixNet-78](#)

进入你解压的文件夹, 你将看到一个setup.py文件。按照下面的提示, 将PodSixNet安装到你的电脑上。

使用Mac的用户:

打开一个新的终端, 然后键入cd . (注意命令cd后有一个空格)。将包含PodSixNet解压文件的文件夹拖到终端窗口, 然后按回车。再终端上输入sudo python setup.py install, 然后按回车。这样就能在你的电脑上安装PodSixNet了, 安装完成后你就可以在电脑的任意位置使用PodSixNet了。

(下载的文件夹里没有包含setup.py文件, 需要下载最新的PodSixNet-78)

对于Windows用户:

在你解压PodSixNet文件的路径下, 按住shift键, 并且用鼠标右键点击目录空白处, 然后选择在此处打开命令行窗口。如果你已经配置好了Python的环境变量, 那么只需要在命令行窗口中输入python setup.py install这条命令就可以了。如果这条命令的返回结果是“Command Not Found”错误, 那么你

需要配置一下环境变量了。

按照如下的方法将Python添加到你的环境变量中，右键点击我的电脑，选择属性。然后选择高级选项卡，点击环境变量按钮。在系统变量下点击新建，将Python的安装路径添加进系统变量中如C:python2.7。这样，就可以使用python setup.py install命令来安装PodSixNet了。

来测试一下你的安装是否正确，通过在终端上输入python来启动Python解释器，或者进入DOS的命令提示符，然后按下回车键。输入import PodSixNet，再按下回车。如果没有得到什么提示，那么你的安装就是正确的。如果终端提示了一些错误，可以把错误提示贴在这篇博客的评论里，来寻求帮助。

PodSixNet架构

让我们开始创建一个PodSixNet的框架吧。在这个项目中，当你需要使用PodSixNet时，可以使用这个框架，请确保你的各个组件能正确运行。

新建一个名为server.py的文件，并将这个文件放在和boxes.py相同的目录下。然后在这个文件中加入如下的代码：

Python

```
import PodSixNet.Channel  
1 import PodSixNet.Channel  
2 import PodSixNet.Server  
3 from time import sleep  
4 class ClientChannel(PodSixNet.Channel.Channel):  
5     def Network(self, data):  
6         print data  
7  
8 class BoxesServer(PodSixNet.Server.Server):  
9  
10    channelClass = ClientChannel  
11  
12    def Connected(self, channel, addr):  
13        print 'new connection:', channel  
14  
15 print "STARTING SERVER ON LOCALHOST"  
16 boxesServe=BoxesServer()  
17 while True:  
18    boxesServe.Pump()  
19    sleep(0.01)
```

这段代码建立了一个简单的连接模型，它在默认的端口上监听连接的到来。当有人连接了服务器，它将打印一条消息。

这段代码本应是立即可以使用的，但是它目前还不能使用，因为你还没有编写客户端代码。客户端代码的修改工作是简单的。你需要做的只是初始化客户端的PodSixNet类，利用它连接服务器，然后你的客户端就能向服务器发送许多信息了。

将以下的代码添加到boxes.py文件的顶端，来导入一些需要使用的PodSixNet和timing库：

Python

```
from PodSixNet.Connection  
1 from PodSixNet.Connection import ConnectionListener, connection  
2 from time import sleep
```

将class BoxesGame():这行代码修改为如下这行代码，使得BoxesGame继承ConnectionListener：

Python

```
class BoxesGame(Connectio  
1 class BoxesGame(ConnectionListener):
```

将以下这行代码添加到BoxesGame类中init函数的最后一行，来初始化PodSixNet客户端：

Python

```
self.Connect()  
1 self.Connect()
```

将下面这段代码添加到update函数的开始处，来连接客户和服务器，等待新的事件或消息：

Python

```
connection.Pump()  
self.Pump()  
1 connection.Pump()  
2 self.Pump()
```

现在，打开两个终端窗口。使用Python在一个终端中运行boxes.py，在另一个终端中运行server.py。当你启动客户端时，服务器端会提示你，一个新的客户成功连接了。

Python

```
localhost:boxes rwenderlich$ python  
1 localhost:boxes rwenderlich$ python server.py  
2 STARTING SERVER ON LOCALHOST  
3 new connection: <socket._socketobject object at 0x37ff48>;
```

加入多玩家功能

现在你的服务器和客户端可以互相通信了，真棒！但是你还需要做一些工作来完善游戏。让我们给游戏添加一些功能吧。首先，当你在客户端界面放置一条线时，你需要将这个信息告诉服务器。在update方法里找到你绘制线条的代码，将那部分代码像下面这样修改：

Python

```
if
pygame.mouse.get_pre
    if pygame.mouse.get_pressed()[0] and not alreadyplaced and not isoutofbounds:
1   if is_horizontal:
2       self.boardh[ypos][xpos]=True
3       self.Send({"action": "place", "x":xpos, "y":ypos, "is_horizontal": is_horizontal, "gameid": self.gameid, "num":
4 self.num})
5   else:
6       self.boardv[ypos][xpos]=True
7       self.Send({"action": "place", "x":xpos, "y":ypos, "is_horizontal": is_horizontal, "gameid": self.gameid, "num":
self.num})
```

请注意这个代码引入了新的属性self.gameid和self.num。将以下这两行代码添加到`__init__`函数中，并把它们初始化为None：

Python

```
self.gameid = None
self.num = None
```

```
1 self.gameid = None
2 self.num = None
```

现在运行服务器和客户端代码，然后在客户端上点击一条横线。服务端会记录下你在客户端点击的线的信息。

Python

```
STARTING SERVER
ON LOCALHOST
```

```
STARTING SERVER ON LOCALHOST
1 new connection: <socket._socketobject object at 0x3adb90>;
2 new connection: <socket._socketobject object at 0x54db58>;
3 {>gameid<: None, >is_horizontal<: True, >action<: >place<;,
4 >num<: None, >y<: 5, >x<: 3}
5 {>gameid<: None, >is_horizontal<: False, >action<: >place<;,
6 >num<: None, >y<: 5, >x<: 3}
```

编写游戏类

接下来，你将实现游戏类，这个类将表示游戏的所有元素：一对客户端，游戏网格面板和现在该轮到谁操作了。

在server.py文件的BoxesServer类后添加如下的代码：

Python

```
class Game:
    def __init__(self,
```

```
1 class Game:
2     def __init__(self, player0, currentIndex):
3         # whose turn (1 or 0)
4         self.turn = 0
5         #owner map
```

```
6     self.owner=[[False for x in range(6)] for y in range(6)]
7     # Seven lines in each direction to make a six by six grid.
8     self.boardh = [[False for x in range(6)] for y in range(7)]
9     self.boardv = [[False for x in range(7)] for y in range(6)]
10    #initialize the players including the one who started the game
11    self.player0=player0
12    self.player1=None
13    #gameid of game
14    self.gameid=currentIndex
```

这个类表示了游戏的状态。服务器是这个游戏的“长官”，它会控制每个客户端界面的更新与显示。

当第一个用户连接时，服务器应新开始一个游戏。服务器将有一个游戏列表以及玩家的等待队列，这样当有一个客户连接时，服务器就能知道是应该新开始一个游戏，还是让新连接的玩家和一个正在等待的玩家一起玩。现在，让我们来添加这个功能吧。

在BoxesServer类的开始处添加以下这些代码吧：

Python

```
def __init__(self, *args, 
**kwargs):
1 def __init__(self, *args, **kwargs):
2     PodSixNet.Server.Server.__init__(self, *args, **kwargs)
3     self.games = []
4     self.queue = None
5     self.currentIndex=0
```

看到那行很奇怪的以PodSixNet开头的代码了吗？因为你在BoxesServer类里继承了PodSixNet.Server.Server类，所以，你需要在BoxesServer类的__init__方法里调用父类的__init__方法。这就是对PodSixNet的server类的初始化，初始化时需要把所有它需要的参数传给它。
currentIndex变量用来记录那些正在进行的游戏，当每个游戏开始时，它就会加1。

让我们添加下列这些代码吧，使新连接的玩家加入等待队列或让他与正在等待的玩家一起进行游戏。将下面这部分代码添加到Connected()方法的末尾。

Python

```
if self.queue==None: 
1 if self.queue==None:
2     self.currentIndex+=1
3     channel.gameid=self.currentIndex
4     self.queue=Game(channel, self.currentIndex)
5 else:
6     channel.gameid=self.currentIndex
7     self.queue.player1=channel
8     self.queue.player0.Send({"action": "startgame", "player":0, "gameid": self.queue.gameid})
9     self.queue.player1.Send({"action": "startgame", "player":1, "gameid": self.queue.gameid})
10    self.games.append(self.queue)
11    self.queue=None
```

正如你看到的那样，服务器首先检查是否有一个游戏在等待队列中。如果没有，服务器将新开始一个游戏，然后将新开始的游戏加入到队列中。这样，当下一个用户连接时，这个用户将被分配到刚创建

的游戏中。

考虑一下这个问题：当玩家在游戏的网格界面上选择了一条线，服务器将知道玩家放置线的位置。但是，当很多个游戏同时进行时，服务器并不知道当前的玩家属于哪一个游戏。因此，服务器就不知道该更新哪一个游戏的网格界面，也不知道该通知哪一个用户，他的网格界面应该改变了。

为了使服务器能得到上述的信息，在你将用户分配到一个游戏时，你首先应给用户一个gameid。你可将gameid作为“startgame”信息中的一个参数传递给用户。这也将作为你提醒用户游戏已经开始的信号。

在游戏开始前，你需要让用户等待“startgame”信息直到这个信息的到达，然后你需要决定，哪一个玩家先进行游戏。这将告知游戏的两个玩家，游戏开始了，他们都有一个特定的游戏id。让我们接下来做这个功能吧。

在客户端的代码中添加如下的方法：

Python

```
def Network_startgame(self):
    1 def Network_startgame(self, data):
    2     self.running=True
    3     self.num=data["player"]
    4     self.gameid=data["gameid"]
```

你希望客户端保持等待状态，直到收到开始游戏的消息。因此，在`__init__`方法的末尾添加如下的代码：

Python

```
self.running=False
while not self.running:
    1 self.running=False
    2 while not self.running:
    3     self.Pump()
    4     connection.Pump()
    5     sleep(0.01)
    6 #determine attributes from player #
    7 if self.num==0:
    8     self.turn=True
    9     self.marker = self.greenplayer
   10    self.othermarker = self.blueplayer
   11 else:
   12     self.turn=False
   13     self.marker=self.blueplayer
   14     self.othermarker = self.greenplayer
```

两个玩家，同一个游戏

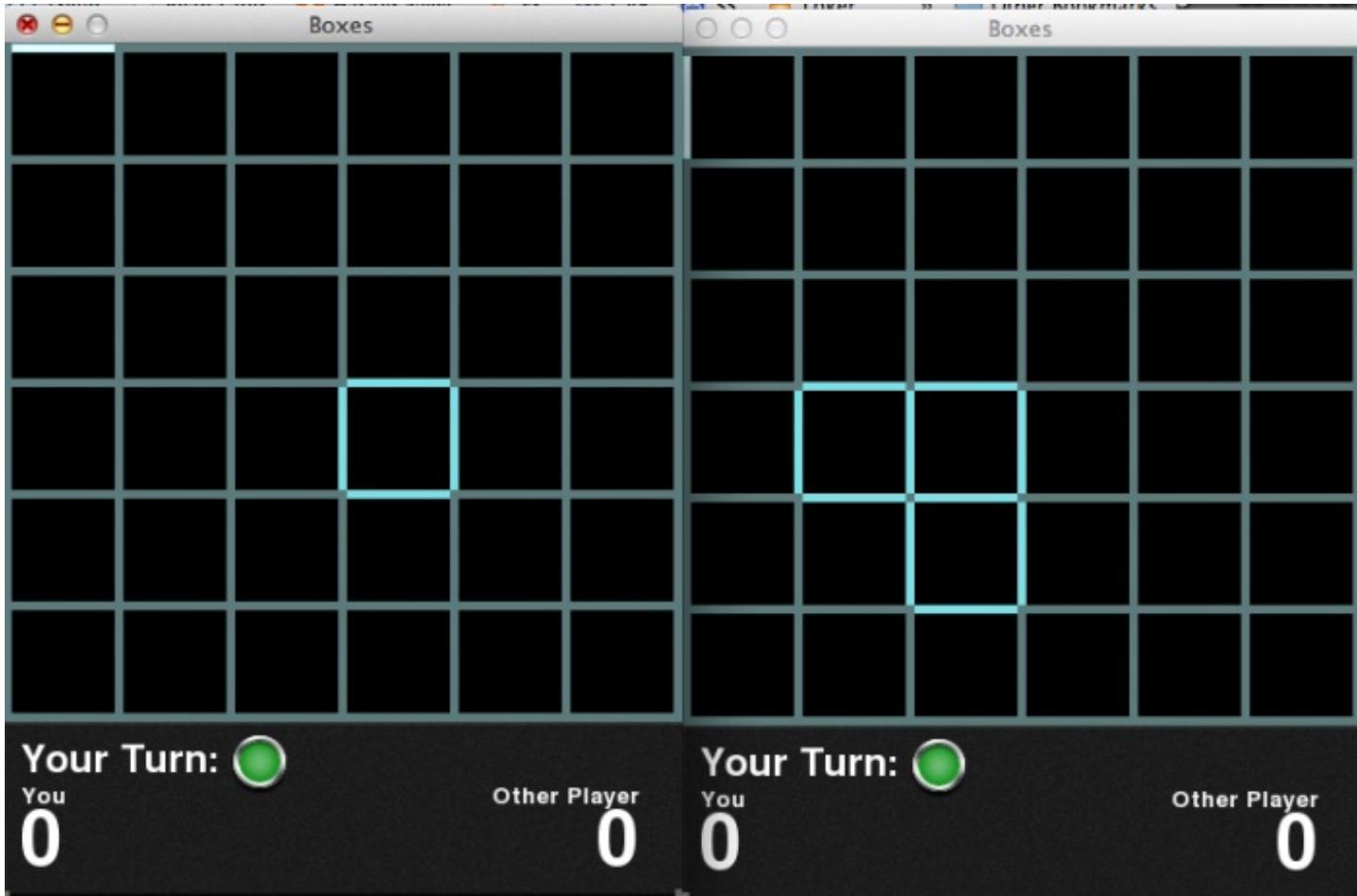
还记得我说的那个还不能使用的功能——`drawOwnermap`吗，它用指定的颜色绘制每个方格。现在它可以了，因为客户端能知道你在网格线上是蓝色的或绿色的了。

Python

```
self.drawOwnermap()
```

```
1 self.drawOwnermap()
```

现在运行游戏吧。这一次，你需要打开三个终端窗口——一个用来运行服务器程序，另外两个运行客户端程序，因为如果没有两个玩家同时在线，游戏就不会开始。现在还没有什么其他的效果，但是至少两个玩家的游戏连接到了同一个服务器。



现在我们来快速实现放置线条的功能。首先，你需要在server.py文件里的Game类中添加一个方法，当用户在网格上放置一条线时，它将发挥作用。Game类首先会检查游戏现在是否轮到当前玩家，如果是，那么它会更新两个玩家的游戏网格界面，将当前玩家放置的线条添加到两个游戏玩家的界面中。

将下面这个方法添加到server文件的Game类中：

Python

```
def placeLine(self, is_h,  x, y, data, num):
```

```
1 def placeLine(self, is_h, x, y, data, num):
2     #make sure it's their turn
3     if num==self.turn:
4         self.turn = 0 if self.turn else 1
5         #place line in game
```

```
6     if is_h:
7         self.boardh[y][x] = True
8     else:
9         self.boardv[y][x] = True
10    #send data and turn data to each player
11    self.player0.Send(data)
12    self.player1.Send(data)
```

这段代码首先检查玩家的动作是否有效，即游戏是否轮到该玩家，如果有效，则将玩家的动作发送给两个玩家，更新他们的网格界面以及轮次。接下来，你需要使服务器能够调用我们刚写的方法，添加下面这段代码到BoxesServer类中：

Python

```
def placeLine(self, is_h, [x, y, data, gameid]):
1 def placeLine(self, is_h, x, y, data, gameid, num):
2     game = [a for a in self.games if a.gameid==gameid]
3     if len(game)==1:
4         game[0].placeLine(is_h, x, y, data, num)
```

这段代码循环遍历所有游戏，找到gameid与当前玩家相同的游戏。然后它调用Game.placeLine()方法，将界面更新的消息发送给客户。

你还有最后一个方法需要添加到server文件的ClientChannel类中。

Python

```
def Network_place(self, [data]):
1 def Network_place(self, data):
2     #deconsolidate all of the data from the dictionary
3
4     #horizontal or vertical?
5     hv = data["is_horizontal"]
6     #x of placed line
7     x = data["x"]
8
9     #y of placed line
10    y = data["y"]
11
12    #player number (1 or 0)
13    num=data["num"]
14
15    #id of game given by server at start of game
16    self.gameid = data["gameid"]
17
18    #tells server to place line
19    self._server.placeLine(hv, x, y, data, self.gameid, num)
```

你已经看到了当玩家在客户端游戏界面上放置一条线时，服务器打印出来的信息，这段代码将会读这些信息，从这些信息中抽取出每一个参数，然后调用server的placeLine方法。

现在游戏的服务器能够向客户端发送信息了。但是，仍然还有一个大问题：客户端还不能处理这些信息。让我们给客户端代码添加一个方法来解决这个问题吧。

将下面的代码添加到客户端代码中：

Python

```
def Network_place(self, data):
    def Network_place(self, data):
        #get attributes
        x = data["x"]
        y = data["y"]
        hv = data["is_horizontal"]
        #horizontal or vertical
        if hv:
            self.boardh[y][x]=True
        else:
            self.boardv[y][x]=True
```

客户端收到放置线条信息时将调用这个方法。它从信息中读出参数，然后根据情况更新游戏状态。

现在，尝试运行我们的游戏。你放置的第一条线将会在另一个客户端界面上看到（但是后面放的线条将不会，别急，你将马上能解决这个问题）。现在，你已经完成了第一个多玩家服务器！你可以证明，这并不是一个简单的事情，回头看看到目前为止，你所做的工作吧！

轮流玩

接下来你需要实现游戏的轮流功能，这样玩家才不能在游戏中作弊。信不信，你已经为这个功能建立了一个变量（turn）。但首先，你需要一个延时功能，玩家放置一条线后，它将等待10帧才允许玩家在界面中放置下一条线。

在boxes.py的__init__方法中添加下面这个变量：

Python

```
self.justplaced=10
1 self.justplaced=10
```

你要使这个变量在每一帧后减1，在玩家放置一条线后重置为10。将以下的这个代码做一些改动：

Python

```
if
pygame.mouse.get_pre()
1 if pygame.mouse.get_pressed()[0] and not alreadyplaced and not isoutofbounds:
2 #-----to-----#
3 if pygame.mouse.get_pressed()[0] and not alreadyplaced and not isoutofbounds and self.turn==True and
3 self.justplaced<=0:
```

代码检查了当前是否轮到该玩家进行游戏，并且保证他在10帧内不能再次放置线条。

接下来，在update方法的顶部添加下面这行代码：

Python

```
self.justplaced-=1
```

```
1 self.justplaced-=1
```

这样在游戏的每一帧，`justplaced`变量都能减1。现在你还要保证当用户放置一条线时，需要将`justplaced`变量重置为10。将这行代码添加到刚才修改的if语句中：

Python

```
self.justplaced=10
```

```
1 self.justplaced=10
```

好的，现在你可能注意到了一个问题，两个玩家的轮次指示标志一直是绿色的！这是因为你还没有添加控制颜色转换的代码。

在`drawHUD()`方法中找到绘制指示标志`indicator`的方法`screen.blit`。将它改成下面这样：

Python

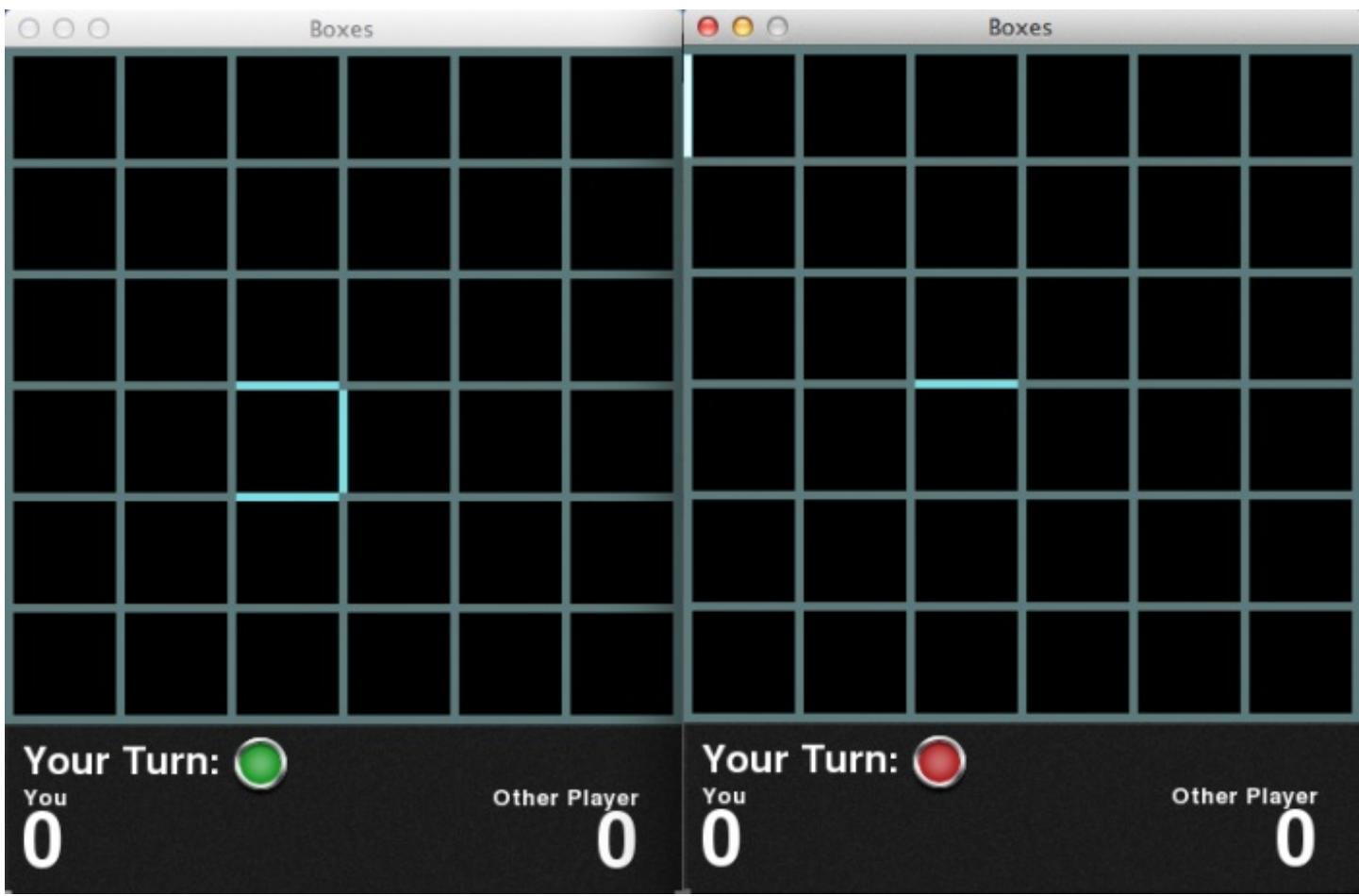
```
self.screen.blit(self.gree
```



```
nindicator if self.turn
```

```
1 self.screen.blit(self.greenindicator if self.turn else self.redindicator, (130, 395))
```

再一次运行游戏——其中一个玩家的`indicators`指示标志将是绿色或者红色。



这个指示标志是正确的，但是当你放置一条线时，游戏显然应当切换轮次。但目前，还没有实现。我们现在快速编写服务器代码来添加这个功能吧。

在服务端，当任何事件发生时，你都需要发送给客户一条信息，告诉客户端，现在轮到哪个玩家进行游戏了。你可以简单地在server.py的Game类中添加placeLine方法来完成这一功能。将下面这行代码添加到更新turn变量的if语句中：

Python

```
self.player1.Send({"action": "yourturn", "turf": True if self.turn == 1 else False})  
2 self.player0.Send({"action": "yourturn", "turf": True if self.turn == 0 else False})
```

turf代表了“true or false”。这个变量将告诉客户端，现在是否轮到他们进行游戏。在游戏开始时，你不需要发送这些信息，因为客户端根据玩家的number知道了谁应该第一个放置线条。

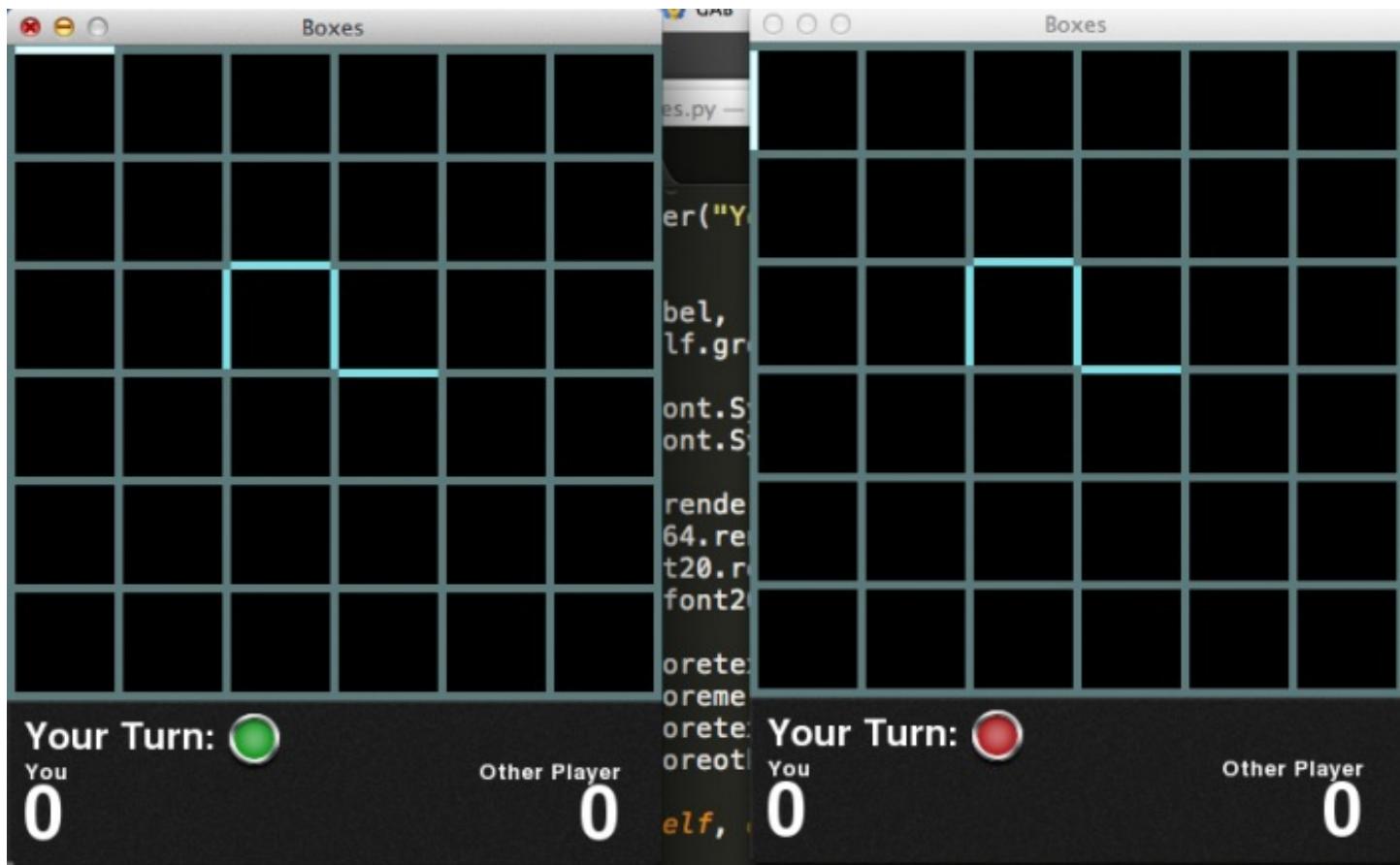
让我们在客户端实现“轮到你了”的指令。这个是一个非常简单的事情，在客户端的Game类中，添加这一方法。

Python

```
def Network_yourturn(self, data):
```

```
2 #torf = short for true or false  
3 self.turn = data["torf"]
```

现在再次运行服务器程序和两个客户程序。你必须重启一下服务器程序，因为你对它进行了修改。



现在你可以看到，玩家必须按照次序轮流进行游戏了，很不错对吗？现在你已经告诉客户端怎样按照顺序轮流进行游戏，你还应该奖励玩家“努力的成果”，那就是填充方块的颜色！

编写游戏的逻辑

这个游戏有一个简单的逻辑：在玩家进行游戏时，判断他是否完成了一个方格。服务器通过循环所有可能的方块来查找被玩家完成的方块。

在BoxesServer类中新建一个名为tick()的方法。将下面的代码添加到方法中：

Python

```
def tick(self):  
    # 1
```

```
1 def tick(self):  
2     # 1  
3     index=0  
4     change=3  
5     # 2  
6     for game in self.games:  
7         change=3  
8         for time in range(2):
```

```

9      # 3
10     for y in range(6):
11         for x in range(6):
12             # 4
13             if game.boardh[y][x] and game.boardv[y][x] and game.boardh[y+1][x] and game.boardv[y][x+1] and not
14 game.owner[x][y]:
15                 if self.games[index].turn==0:
16                     self.games[index].owner[x][y]=2
17                     game.player1.Send({"action":"win", "x":x, "y":y})
18                     game.player0.Send({"action":"lose", "x":x, "y":y})
19                     change=1
20             else:
21                 self.games[index].owner[x][y]=1
22                 game.player0.Send({"action":"win", "x":x, "y":y})
23                 game.player1.Send({"action":"lose", "x":x, "y":y})
24             change=0
25     # 5
26     self.games[index].turn = change if change!=3 else self.games[index].turn
27     game.player1.Send({"action":"yourturn", "torf":True if self.games[index].turn==1 else False})
28     game.player0.Send({"action":"yourturn", "torf":True if self.games[index].turn==0 else False})
29     index+=1
self.Pump()

```

哇，有许多代码啊！让我们拆开这些代码，逐个看看是什么意思吧。

1. 在方法的最顶部，你声明了一些变量：index，这个变量常用在for循环中，用来跟踪你遍历到的当前变量，change，告诉你现在是否要改变玩家的轮次，以及现在轮到谁玩游戏了。
2. 接下来循环遍历所有的游戏列表，将change变量重置为3，代表没有变化发生。
3. 然后遍历所有可能的方块。你需要遍历两次，因为一个玩家有可能在两个方块的中间放置了一条线，这样一次就同时完成了两个方块。
4. 对于每一个可能的方块，你要检查这个方块是否被绘制完成，如果是，还要确保这个方块不是在之前的游戏轮次中就被完成的。
5. 最后，你还要检查是哪一个玩家在完成一条线的放置后完成了一个方块，然后正确的设置variable变量。

现在，你已经有了tick这个方法，你需要将它添加到server中。这非常简单，到server.py的底部，找到下面这段代码：

Python

```
boxesServe.Pump()
```

```
1 boxesServe.Pump()
```

把它改为：

Python

```
boxesServe.tick()
```

```
1 boxesServe.tick()
```

现在，你已经有了一些游戏逻辑代码在server服务器上，让我们在client端添加一个方法，来告诉客户端是它赢了一个方块还是输了一个方块（也就是说对方赢了一个方块）。

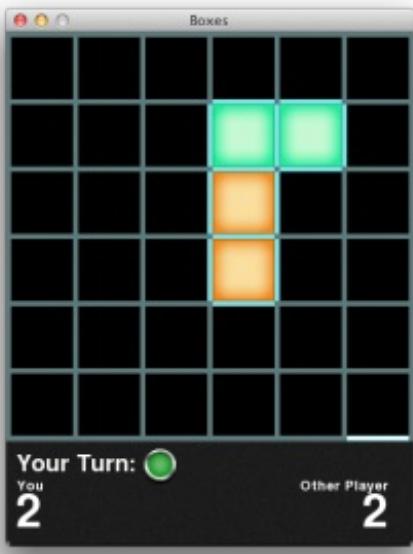
在客户端中添加以下两个方法：

Python

```
def Network_win(self, data):
    1 def Network_win(self, data):
    2     self.owner[data["x"]][data["y"]]="win"
    3     self.boardh[data["y"]][data["x"]]=True
    4     self.boardv[data["y"]][data["x"]]=True
    5     self.boardh[data["y"]+1][data["x"]]=True
    6     self.boardv[data["y"]][data["x"]+1]=True
    7     #add one point to my score
    8     self.me+=1
    9 def Network_lose(self, data):
   10     self.owner[data["x"]][data["y"]]="lose"
   11     self.boardh[data["y"]][data["x"]]=True
   12     self.boardv[data["y"]][data["x"]]=True
   13     self.boardh[data["y"]+1][data["x"]]=True
   14     self.boardv[data["y"]][data["x"]+1]=True
   15     #add one to other players score
   16     self.otherplayer+=1
```

这两个方法用来处理从网络上接收到的赢或输的信息，然后适当的更新游戏的状态。

再一次运行一个服务器和两个客户端吧，感受一下在游戏界面上同时显示两个客户端的信息。



游戏结束了！

等会儿，游戏什么时候才结束呢？你需要让服务器实现你在教程第一部分的最后添加的finish()方法。记得吗，这个方法在游戏的界面上显示玩家获胜或失败的画面，以及让玩家退出游戏。

将下面这几行代码添加到update方法的顶部：

Python

```
if self.me+self.otherplayer == 36:  
    self.didiwin=True if self.me>self.otherplayer else False  
    return 1
```

这段代码查看你以及你的对手一共获得了多少个方块。如果总共获得了36块（网格线中总共包含的方块），那么游戏结束。如果游戏结束，那么查看哪个玩家获得的方块数最多，获得方块数最多的玩家赢得游戏胜利，然后返回1。

最后，在文件底部，找到bg.update()，然后将它改为下面这样：

Python

```
if bg.update()==1:  
    break  
  
1 if bg.update()==1:  
2     break
```

在文件的最底部，添加这行代码，注意，这行代码不要有任何缩进：

Python

```
bg.finished()  
  
1 bg.finished()
```

现在，你能使游戏以某一玩家的胜利而终止了。但是，在测试这个功能之前，让我们给服务器和客户端再添加一个功能吧。当一个玩家退出了游戏，你要使游戏中的另一个玩家也退出游戏。

将下面的代码添加到ClientChannel类中：

Python

```
def Close(self):  
  
1 def Close(self):  
2     self._server.close(self.gameid)
```

然后将以下这部分代码添加到BoxesServer类中：

Python

```
def close(self, gameid):  
    try:  
  
1 def close(self, gameid):  
2     try:  
3         game = [a for a in self.games if a.gameid==gameid][0]  
4         game.player0.Send({"action":"close"})  
5         game.player1.Send({"action":"close"})  
6     except:
```

```
7     pass
```

要使客户端能够理解close()命令，需要在客户端的类中添加如下代码：

Python

```
def Network_close(self, data):  
1 def Network_close(self, data):  
2     exit()
```

再一次运行这个游戏吧，打开两个客户端，然后一直玩到你赢为止。这并不太难吧！

现在，你已经正式完成了这个游戏的开发。如果你想给游戏添加一些音乐和声音效果或让游戏更完美的话，那么就进入下一部分教程吧，教程的下一部分还将讲述游戏网络的连接。否则，可直接跳过本教程的下一部分，看看你的成果以及接下来可以做的工作。

最后的润色

在client客户端的主类(BoxesGame)里，添加下面这个方法：

Python

```
def initSound(self):  
1 def initSound(self):  
2     pygame.mixer.music.load("music.wav")  
3     self.winSound = pygame.mixer.Sound('win.wav')  
4     self.loseSound = pygame.mixer.Sound('lose.wav')  
5     self.placeSound = pygame.mixer.Sound('place.wav')  
6     pygame.mixer.music.play()
```

这个代码载入了音乐和声效文件，这样你就可以在需要的地方播放它们。这些.wav文件存放在你在教程（上）中下载的资源包里。我用[cxfr](#)制作了这些音效，这些音效来自于[Kevin MacLeod](#)。

现在在__init__中的initGraphics方法中添加如下这行代码：

Python

```
self.initSound()  
1 self.initSound()
```

然后在合适的地方添加下面这些代码：

Python

```
#anywhere in  
Network_place  
1 #anywhere in Network_place  
2 self.placeSound.play()
```

```
3
4 #anywhere in Network_win
5 self.winSound.play()
6
7 #anywhere in Network_lose
8 self.loseSound.play()
```

这些代码将会在合适的地方发出音效。

再一次运行游戏，并确保你的电脑打开了声音。享受groovy曲调吧（这里不太确定）！

到此，你已经给游戏增加了音效，接下来，我们让游戏的玩家可以通过网络连接进行游戏，而不是只能在同一台电脑上玩。

将BoxesGame类中的self.Connect()方法替代为如下的代码：

Python

```
address=raw_input("Ad 
dress of Server: ")
1 address=raw_input("Address of Server: ")
2 try:
3     if not address:
4         host, port="localhost", 8000
5     else:
6         host,port=address.split(":")
7     self.Connect((host, int(port)))
8 except:
9     print "Error Connecting to Server"
10    print "Usage:", "host:port"
11    print "e.g.", "localhost:31425"
12    exit()
13 print "Boxes client started"
```

这段代码让客户端确定如何查找服务器。在运行游戏客户端后，客户端将要求你输入游戏服务器的IP地址。

让我们修改一下server端的代码吧。将server.py文件中的boxesServe=BoxesServer()代码修改为如下：

Python

```
# try:
address=raw_input("Ho 
1 # try:
2 address=raw_input("Host:Port (localhost:8000): ")
3 if not address:
4     host, port="localhost", 8000
5 else:
6     host,port=address.split(":")
7 boxesServe = BoxesServer(localaddr=(host, int(port)))
```

最后再运行一次游戏，看看效果吧！

现在该做什么呢？

这是[finished sample project](#)教程里的最终代码。恭喜你！你用Python和PyGame完成了你的第一个多玩家游戏。希望你能在这个小项目中获得乐趣。

如果你对这个游戏有兴趣，还想继续完善游戏的话，以下这些建议你可以自己尝试一下：

- 添加一些随机的“坏块”，当一个玩家获得这种“坏块”时，它将失去一分。
- 随机地让一些方块的分值超过1分。
- 让第一个进入游戏的玩家能够决定网格的规模（即网格中有多少个方块）。
- 让玩家在他的轮次中，能移除其它玩家放置的线条。

对于这个教程，如果你有任何疑问或建议，可参与论坛的讨论。希望你愉快地用Python编程！

这篇博客的作者是一名13岁的Python开发者 [Julian Meyer](#)，你可以在[Google+](#) 和 [Twitter](#)上找到他。

1 赞 2 收藏 [评论](#)

关于作者： [justyoung](#)



可怜的小硕 [个人主页](#) · [我的文章](#) · 10

13岁Python开发者写给青少年的Python入门教程

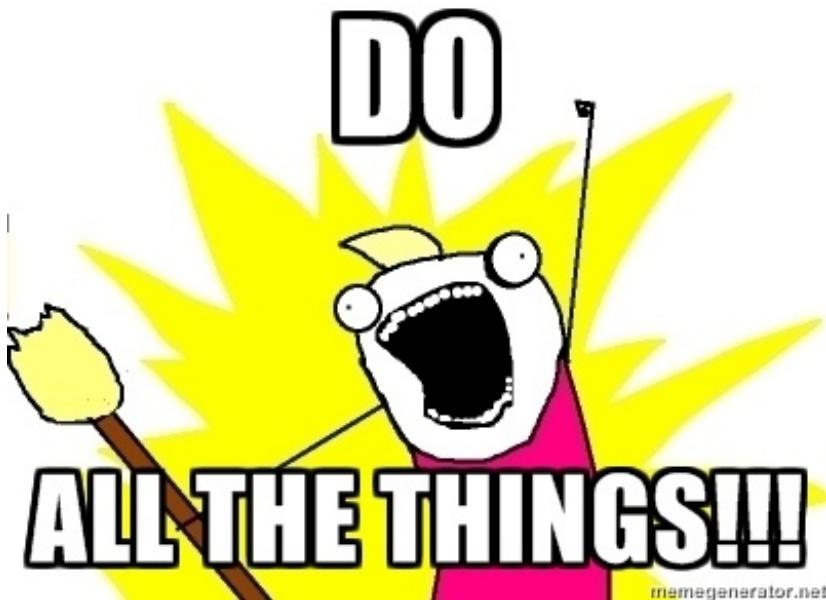
本文由 [伯乐在线 - 艾凌风](#) 翻译, [黄利民](#) 校稿。未经许可, 禁止转载!
英文出处: [Julian Meyer](#)。欢迎加入[翻译组](#)。

你曾经想知道计算机是如何工作的吗? 尽管我们不能在一篇文章里面教会你所有的东西, 但是可以通过学习如何写出你自己的程序来获得一个良好的开端。在这篇Python教程中, 你将会学到计算机编程的基础知识, 使用对新手来说最棒的编程语言之一。

什么是编程?

尽可能简单的讲, 编程是编写代码, 命令计算机去完成某项任务的艺术。这里讲的某项任务, 可以是简单的两数相加, 或是像把飞船送入轨道这样的复杂任务!

一个程序里面, 最小的组成部分被称作语句 (statement) ——代表了对计算机做出的一条指令。



当你完成了自己的程序后, 编译器会把你写的代码转换为机器码——计算机语言的最底层。机器码指示中央处理器工作 (central processing unit), 或者叫做CPU, 这里面包含一些诸如加载某个值或是做数学运算的步骤。

如果你曾经听过某人说: “我编译了我的程序”, 那代表他们已经把代码转换成了机器码。

为什么不直接写机器码呢? 原因很显然, 程序码具有可读性。下面比较了Python版的程序和其对应的机器码:

Python代码

Python

```
print "Hello, World!"  
...  
...
```

```
1 print "Hello, World!"  
2 ...  
3 "Hello, World!"
```

对应的机器码

Python

```
c7 3c 2a 3c 2a 2b 2a 5c  
3c 28 5c 2a 2b 2a 5c 3c
```

```
1 c7 3c 2a 3c 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c 3c  
2 28 5c 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c 3c 28 5c  
3 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c 3c 28 5c 2a 2b  
4 2a 5c 3c 28 5c 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c  
5 3c 28 5c 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c 3c 28  
6 5c 2a 2b 2a 5c 3c 28 5c 2a 2b 2a 5c 3c 28 5c 2a  
7 2b 2a 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00  
8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
11 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
12 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
13 00 00 00 00 00 00 00 00 64 48 65 6c 6c 6f 2c 20 57  
14 6f 72 6c 64 21 00 00 00 00 00 00 00 00 00 00 00 00  
15 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
17 ...  
18 "Hello, World!"
```

很明显为什么你不会想要直接编写机器码了。但是，确实有一些人会去写机器码——萝卜白菜各有所爱嘛！

上面我们忽略了一个小问题。Python是一种解释型语言；你并不会像上面我们暗示的那样，把它直接编译成为机器码。

实际上是这样的，Python使用一个叫做解释器的东西。解释器是另外一个程序，把代码编译成叫做二进制码的东西，然后再程序运行的时候再转换成机器码。你等下会学到更多和解释器有关的内容。

当你最终运行程序的时候，你刚编译的那些机器码会被加载到内存中，CPU会读取它并执行程序。

然而，在刚开始学习使用Python编程的时候，你并不需要完全理解编译器的内部工作原理，但是你必须确保你已经安装了Python。

准备开始

如果你在使用Mac，你走运了——Python已经预装在Mac中了。在Mac中使用Python解释器，打开**终端** (**Terminal.app**)；你可以再应用程序/工具文件夹下找到它，或者在Spotlight中输入，像这样：



打开终端后，输入下面的指令然后按下回车：

Python

```
$ python
```

```
1 $ python
```

你会看到类似下面这样的结果：

注意：如果你没有得到上面的结果，把输出结果复制到论坛，我们会尽力帮助你的！**

Windows

在Windows上，这一过程稍微有点复杂——但是还是要说，大部分的东西还是在Windows！上：]

首先，在浏览器中访问Python官网的[下载](#)页面。

The current production versions are [Python 3.4.0](#) and [Python 2.7.6](#).

Start with one of these versions for learning Python or if you want the most stability; they're both considered stable production releases.

If you don't know which version to use, try Python 3.4. Some existing third-party software is not yet compatible with Python 3, if you need to use such software, you can download Python 2.7.x instead.

For the MD5 checksums and OpenPGP signatures, look at the [detailed Python 3.4.0 page](#):

- [Python 3.4.0 Windows x86 MSI Installer](#) (Windows binary -- does not include source)
- [Python 3.4.0 Windows X86-64 MSI Installer](#) (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)
- [Python 3.4.0 Mac OS X 64-bit/32-bit x86-64/i386 Installer](#) (for Mac OS X 10.6 and later [2])
- [Python 3.4.0 Mac OS X 32-bit i386/PPC Installer](#) (for Mac OS X 10.5 and later [2])
- [Python 3.4.0 compressed source tarball](#) (for Linux, Unix or Mac OS X)
- [Python 3.4.0 xzipped source tarball](#) (for Linux, Unix or Mac OS X, better compression)

For the MD5 checksums and OpenPGP signatures, look at the [detailed Python 2.7.6 page](#):

- [Python 2.7.6 Windows Installer](#) (Windows binary -- does not include source)
- [Python 2.7.6 Windows X86-64 Installer](#) (Windows AMD64 / Intel 64 / X86-64 binary [1] -- does not include source)
- [Python 2.7.6 Mac OS X 64-bit/32-bit x86-64/i386 Installer](#) (for Mac OS X 10.6 and later [2])
- [Python 2.7.6 Mac OS X 32-bit i386/PPC Installer](#) (for Mac OS X 10.3 and later [2])

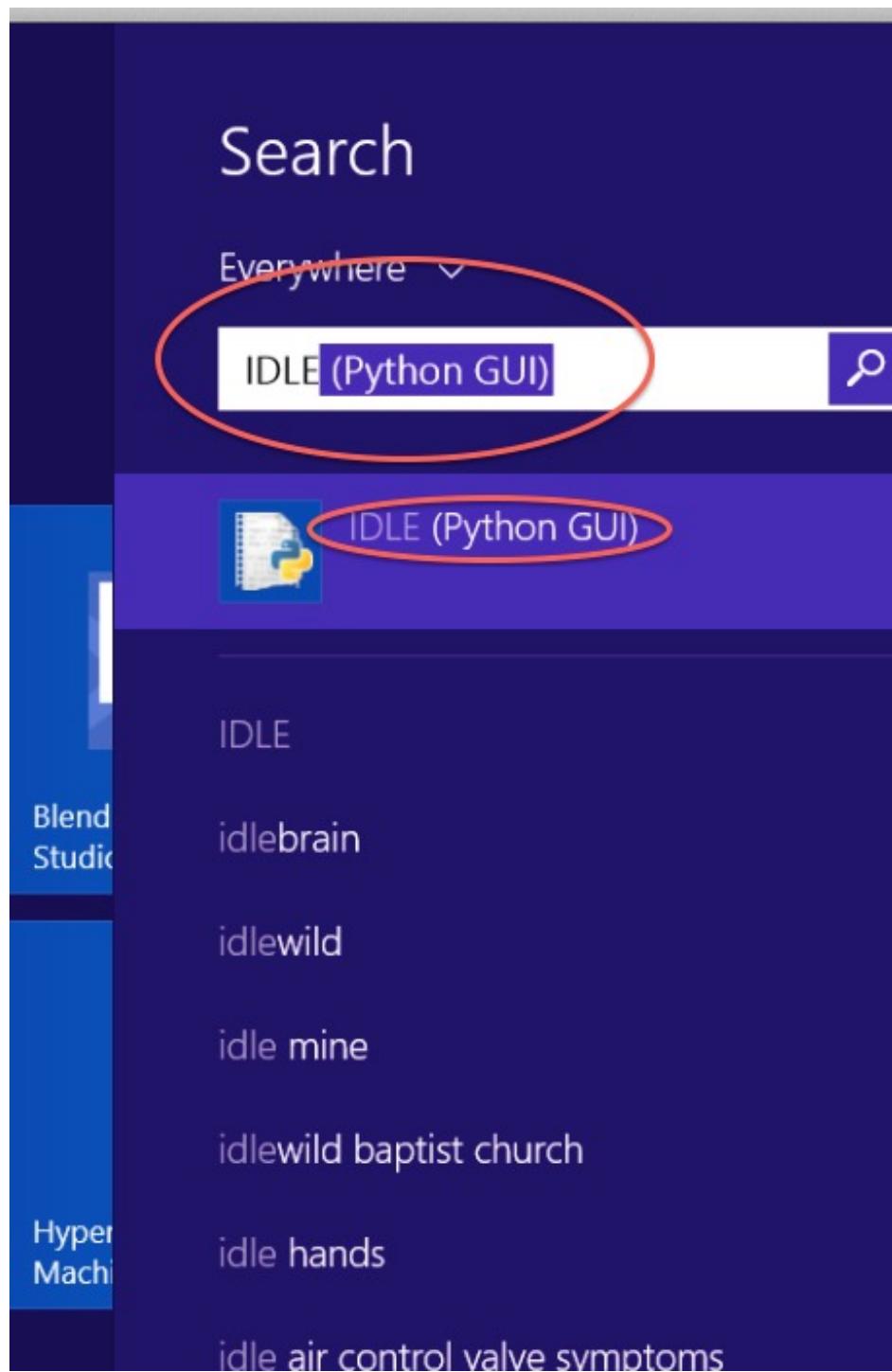
滚动页面跳过所有的Python 3.x.x版本，直接到Python 2.7.x。下载Windows安装包(Windows installer),运行，按照指引来做，接受默认的选项。
一旦安装包安装完成，你就可以启动它了。

在Windows Vista或者Windows 7上，这样启动：

1. 点击左下角的开始菜单。
2. 点击所有程序
3. 打开Python文件夹
4. 双击IDLE解释器

如果你使用Windows 8，这样启动：

1. 点击左下角开始菜单
2. 在搜索框中输入IDLE
3. 点击IDLE(Python GUI)



不管你是如何启动解释器的，你应该确保它能够工作，在终端中输入如下命令，或者是在Windows的命令提示符中输入如下指令并按回车：

Python

```
print "Hello World!"
```

```
1 print "Hello World!"
```

尽管看上去并不像，但是你已经写完你第一个Python程序了！在学习大多数语言的时候，打印出Hello, World被认为是一个起点。

python 命令指示计算机在屏幕上输出后面的字符——而不是在你的打印机上打印！注意“Hello World”两边的引号，引号中的任何东西都被看做常规的文本且不会被解释为一个指令。

变量

变量，是在计算机内存中存放数据的一种方式；在你的程序中你会经常用到它们。在一些语言中，变量有特定的类型，指明了这些变量属于那些类。

在Python中，你不需要声明变量的类型。现在暂时不需要在意这些细节；在本教程稍后的章节中你会学到关于这个内容的知识。

在解释器中输入如下命令并按回车：

Python

```
hello = "Hello World!"
```

```
1 hello = "Hello World!"
```

这么做声明了hello变量并且把Hello World赋值给了它。现在，你不需要在程序中需要“Hello World”的地方输入这个字符串了，取而代之的是，你可以使用hello这个变量。

在解释器中输入如下命令并按回车：

Python

```
print hello
```

```
1 print hello
```

这个指令会产生和Hello World例子相同的结果，但是它是打印出了hello这个变量的值。

变量同样也可以用来储存数字。在解释器中输入如下命令：

Python

```
x = 5  
print x
```

```
1 x = 5  
2 print x  
3 y = 10  
4 z = x + y  
5 print z
```

注意：从现在起，你可能要输入多行语句，只需要在每行末尾输入回车即可**

先猜猜看上面的代码会做些上面，再看下面的答案：

这段代码会打印5，然后打印15.第一个print语句打印了被你赋值为5的变量x。然后它打印了y+x的结果。因为y被赋值为10，x是5，所以打印了15。

对于你生命中的绝大部分程序变量是其核心。在你学习本教程的过程中，你会对变量变得非常熟悉。

变量类型

在之前的教程中你已经遇到了变量，但是我们并没有仔细的介绍它。不同的变量类型存储不同类型的值。

注意：对于全部的内建类型，请查看[Python官方文档](#)

到目前为止，你仅仅和Python中两个基本类型打过交道：整型(integers)和字符串型(strings)，你还会遇到布尔类型(boolean)，你可以用它来储存True或者False。

下面有关于这些变量类型的一点介绍：

整型

一个整型数，是一个整数。整型数的范围，在32位机上是-2147483648 到 2147483647，在64位机上是 -9223372036854775808 to 9223372036854775807。

你可以像这样简单的输入一个数字来创建整型，不需要任何引号：

Python

```
foo = 5
```

```
1 foo = 5
```

字符串型

字符串是一串字符；你可以使用字符串来表示很多东西，从屏幕上的任何文本到整个网页请求。

通过用引号包含一串字符来创建字符串，就像下面一样：

布尔类型

布尔类型代表了True或False。

你通过使用大写开头的True或False 来创建布尔类型，不需要引号，就像下面这样：

Python

```
`isFoo = True`
```

```
1 `isFoo = True`
```

变量两边没有引号；如果你用引号包含了True，你则是创建了一个字符串类型！

字符串连接和整数相加

Python让两个字符串钩在一起变得很容易，我们也称之为字符串连接。你可以使用str()把一个整数类型转换成字符串，相反，你也可以使用int()把一个字符串转换为整型。

在你的解释器中输入下面指令：

Python

```
"1" + "1"  
1 + 1
```

```
1 "1" + "1"  
2 1 + 1  
3 1 + int("1")
```

下面解释上面代码做了些什么：

- 第一条语句连接了两个字符串；引号确保了这两个数字被当做字符串除了。结果就是"11"。
- 第二条语句把两个数作为整数相加，结果是2。
- 最后一条语句把一个整数和另一个被转换为整数的字符串相加，结果还是2。

if 语句

if语句检查某个条件是不是真，如果是，则执行一段代码。条件语句通常是这样的形式值 - 操作符 - 值，通常是比较两个值。

比如，你可以使用`x == 0`评价一个值是不是等于0，`==`是等于操作符。

下面是Python中的一些常见比较

Python

```
a == b: #Checks if a and  
b are equal
```

```
1 a == b: #Checks if a and b are equal  
2 a != b: #Checks if a and b are not equal  
3 a > b: #Checks if a is greater than b  
4 a < b: #Checks if a is less than b  
5 a >= b: #Checks if a is greater than or equal to b  
6 a <= b: #Checks if a is less than or equal to b
```

if语句是如下的形式：

Python

```
if conditional:  
    do_statements_here
```

```
1 if conditional:  
2     do_statements_here
```

注意到在这里执行某些语句 这行是如何缩进的。这是你在Python声明代码块的方式。同一个代码块中的每一行都必须和其他所有行缩进相同的制表符或是空格；这是语言强制规定的。换句话说，不要混合使用制表符和空格。创建一个if语句，在解释器中输入如下命令：

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> if True:
... 
```

迎接你的是神秘的提示符...；这表示解释器在等待你输入代码块的内容，按下Tab键，输入第二行：

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> if True:
...     print "Hello, World!"
```

再次按下回车，你的光标回到了控制台的最左边。要想再输入一个代码块语句，只需要再按Tab。如果你输入完了，按下回车告诉解释器你的代码块已经完成。

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> if True:
...     print "Hello, World!"
...
Hello, World!
```

看一下下面的例子：

Python

```
x = 10
y = 10
```

```
1 x = 10
2 y = 10
3 if x == 10:
4     print "x equals 10!"
5 if x == y:
6     print "x equals y!"
```

第一条语句判断x是不是等于10，如果是的话，打印x equals 10!。另外一个if语句判断x是否等于y，如果是的话，打印"x equals y!".

for循环

for循环再Python中遍历列表中的项目并且把当前项目赋值给变量。一个列表几乎可以是任何东西的集合！

输入下面的代码，并且像下面这样缩进：

Python

```
for x in range(10):
    print x
```

```
1 for x in range(10):  
2     print x
```

在这个例子中，`range(10)` 生成了数字0到9的一个列表。for循环一次把范围内的数字赋值给x。就像if语句一样，for循环执行它下面缩进中的每个语句。在上面的例子中，缩进的代码块里面只包含了一个单独的语句。

因为`print`被调用了10次，对于列表中的每个项目调用一次，这个程序会打印0到9的数字。

函数

函数是可以重用的代码块，用来完成某个特定的代码块。比如说，你可以写一个函数用来把两个数相加，或是打印一个字符串。你可以像下面例子展示的一样定义并调用函数。

Python

```
def hello():  
    print "Hi"
```

```
1 def hello():  
2     print "Hi"  
3 for x in range(3):  
4     hello()
```

在运行前，你可以猜猜看这个程序的输出是什么吗？答案见下文：

它会打印出“Hi”三次，因为for循环会调用这个hello函数三次。

缩进的代码块定义了你调用函数时要执行的语句。因为`print "Hi"` 是这个函数里唯一缩进的语句，所以当调用函数时也是唯一被执行的语句，而不是下面的几行。

你可以通过输入结尾带有一对括号的函数名来调用函数，像之前展示的那样，`hello()` 调用了你上面调用的函数。函数有点类似有围墙的花园：它们不能看到它们那一亩三分地意外的世界。这个被称作变量作用域。如果你想要让一个函数和外界的数据协同工作，你需要把这些数据传递给函数。

使用参数(arguments)可以达到这一目的——不不不，函数之间是不会相互争吵(argue)的！

参数是一个你传递给函数的一个变量；然后函数就可以在内部使用这个参数的值了。

你可以像下面这样声明一个带参函数：

Python

```
def add(a, b):  
    print a + b
```

```
1 def add(a, b):  
2     print a + b
```

上面的函数定义了两个参数，`a`和`b`，放在括号里面，用逗号相互隔开。当你调用这个函数的时候，解释器会吧`a`和`b`的值设置为你传入变量的值。

看一下下面这个例子：

Python

```
def add(a, b):  
    print a + b
```

```
1 def add(a, b):  
2     print a + b  
3 add(1,2) # prints 3!
```

在上面的例子中，`add(1,2)` 这条语句把a和b分别设置为1和2。然后函数打印了你传入的两个数的和。上面的例子打印了计算结果——但是如果你想要用这个计算结果做些事情呢？如果你想要把这个函数的结果加上别的一个值呢？

为了做到这一点，你需要给你的函数加一个`return` 语句。

考虑如下代码：

Python

```
def add(a,b):  
    return a + b
```

```
1 def add(a,b):  
2     return a + b  
3 print add(2, 2) + 1
```

在上面的例子中，你的函数像之前一样把两数相加，但是`return`(返回)语句把两数和传回了函数调用语句。

这意味着上面的`print` 语句得到了`add(2,2)`返回的值然后把它加1， 最后会给你结果是5。

While 循环

`while`循环和`for`循环类似。`for`循环一直进行，直到到达列表的末尾，但是`while`循环会不定的循环，直到给定条件的值等于`False`。一个`while`循环的典型结构是下面这样：

Python

```
while (conditional):  
    run_statement
```

```
1 while (conditional):  
2     run_statement
```

通常，条件变量会在循环运行过程中更新。在解释器中输入下面的程序，来看看这一过程：

Python

```
x = 0  
while x < 10:
```

```
1 x = 0  
2 while x < 10:
```

```
3 print x  
4 x += 1
```

这个程序的行为类似于上面的for循环，但是使用了while循环。这是上面程序做的事情：

1. 给x赋值0
2. 检查是否满足 $x < 10$
3. 如果 $x < 10$ 的值是True, 执行下面的代码块. 如果是 False 退出循环
4. 打印x
5. 把 x 值加1

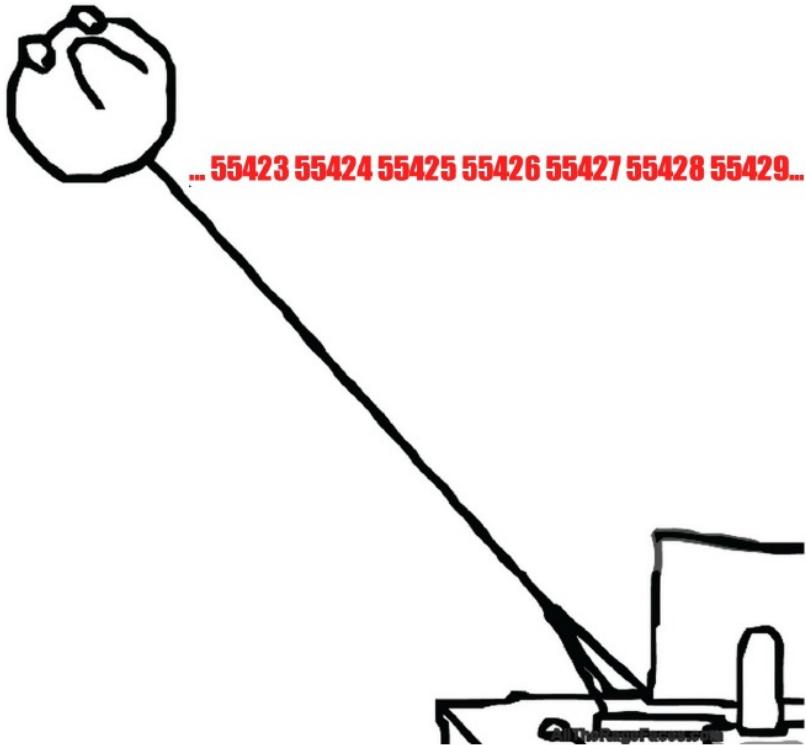
使用while循环需要注意一件事，就是不要创建死循环。

在解释器中输入下面的程序，来看看死循环是什么样的：

Python

```
x = 0  
while True:
```

```
1 x = 0  
2 while True:  
3     print x  
4     x += 1
```



哇啊——终端疯了吧？按Ctrl+C来终止程序吧。

刚才发生了什么？如果你仔细看，你会发现while循环的条件，True永远无法变成False，所以这段代码会以CPU能够执行的最快的速度打印数字。

这个故事告诉我们：当写while循环的时候要时刻保持小心，因为你代码里面的无限循环，对于真实世界来讲，不太会是什么好事！

使用True作为while循环的条件是有可能的，尤其是当你不知道要循环多少次的时候。但是你需要一些技巧来退出这个循环。

在解释器中输入下面的代码：

Python

```
x = 0  
while True:
```

```
1 x = 0  
2 while True:  
3     print x  
4     x += 1  
5     if x > 10:  
6         break
```

这样就好多了！上面的代码会打印0到10然后退出。这里用到的技巧是break语句，它会直接跳出循环。不论出于何种原因，如果你想要在for循环中提前跳出，也可以使用这个技巧。

捕获用户输入

Python中一件很酷的事情是可以非常简便的以文本的方式获取输入用户输入。输入是指任何从外部提供给程序的数据，比如文本或者是其他会影响程序行为的指令。

在你的解释器中输入下面代码：

Python

```
name =  
raw_input("What is your
```

```
1 name = raw_input("What is your name? ")  
2 print "Hello, " + name
```

上面的代码首先会让你进行输入；一旦用户输入了问题的答案，程序会把它赋值给name变量。这一过程完成后，程序会连接字符串“Hello，”和变量name 的内容。

raw_input()函数是Python内建函数；它完成打印从控制台输入字符串的工作，捕获用户输入的文本，然后把它们作为函数值返回。

利用上面捕获输入的技术，你可以创建简单的计算器。在你的解释器中输入下面的代码：

Python

```
a = raw_input("Number  
1: ")
```

```
1 a = raw_input("Number 1: ")  
2 b = raw_input("Number 2: ")  
3 print int(a) + int(b)
```

首先，你捕获了两个用户输入的值然后把它们分别赋值给a和b。然后你把他们转换成了整型并相加。

为什么要有转换这一步呢？仅仅是因为解释器中所有的输入都是以字符串形式进行的，而你是希望吧两个整数的值相加。

如果你不转换这些字符串为整数，你觉得会怎样呢？没错，程序会连接你输入的字符串，这并不是你想要的！

导入

在我们深入研究导入之前，介绍一下Python的模块是必要的。一个模块，是一组Python函数的集合，你希望在不同的程序中重用它们。导入一个模块等价于从一个模块中把所有的代码取出来放到你自己的程序里面所以你可以在任何时候去使用他们，但是又不需要去剪切和复制，哪怕是一行代码！

在Python社区中，存在着大量的模块，但是目前来讲，你仅会接触到随机数模块(random module)。

要导入模块，在解释器中输入下面的代码：

Python

```
import random
```

```
1 import random
```

一旦导入了模块，你就可以像下面这样使用它了：

Python

```
print random.randint(0,  
100)
```

```
1 print random.randint(0, 100)
```

这将会打印0到100间随机的整数；很直观的东西。到目前为止你已经学到很多知识了，足以把他们揉在一个小程序里来练习一下你目前学到的知识了！

猜谜游戏

这将是你辛苦学习Python编程得到的回报。你将创造属于你自己的猜谜游戏！

首先，你需要一个与在解释器中一句一句直接执行程序相比更好的方式。

为了做到这一点，你需要创建Python文件。

Mac

在Mac系统下创建Python文件，在终端中输入下面的命令：

Python

```
$ touch guess.py  
$ open -t guess.py
```

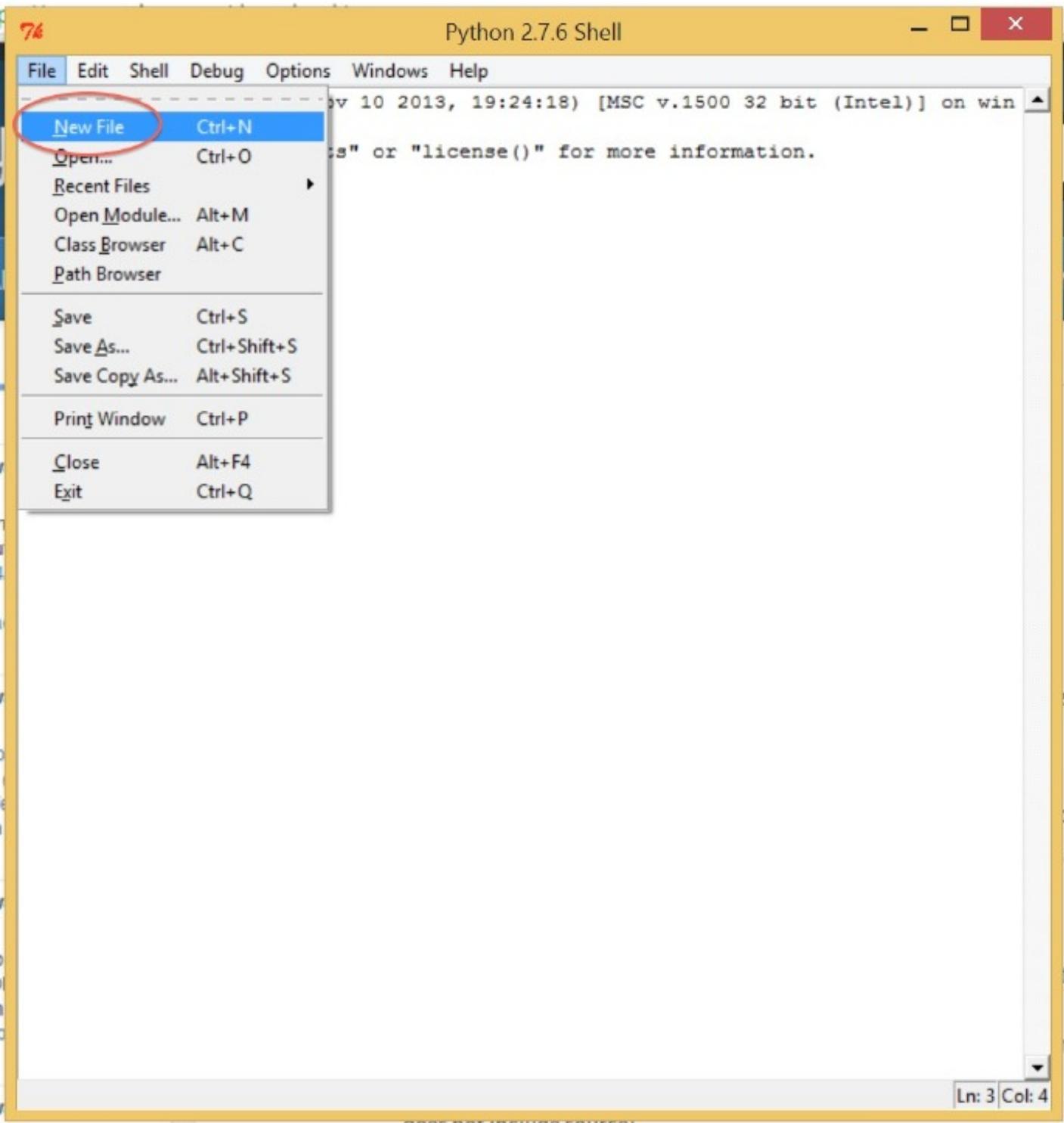
1 \$ touch guess.py
2 \$ open -t guess.py

这将会使用touch 命令创建一个空的guess.py 文件，然后通过open -t命令，使用默认的文本编辑器打开它，

一旦你的Python文件中有了一些内容后，你就可以在终端中输入python guess.py 来执行它。

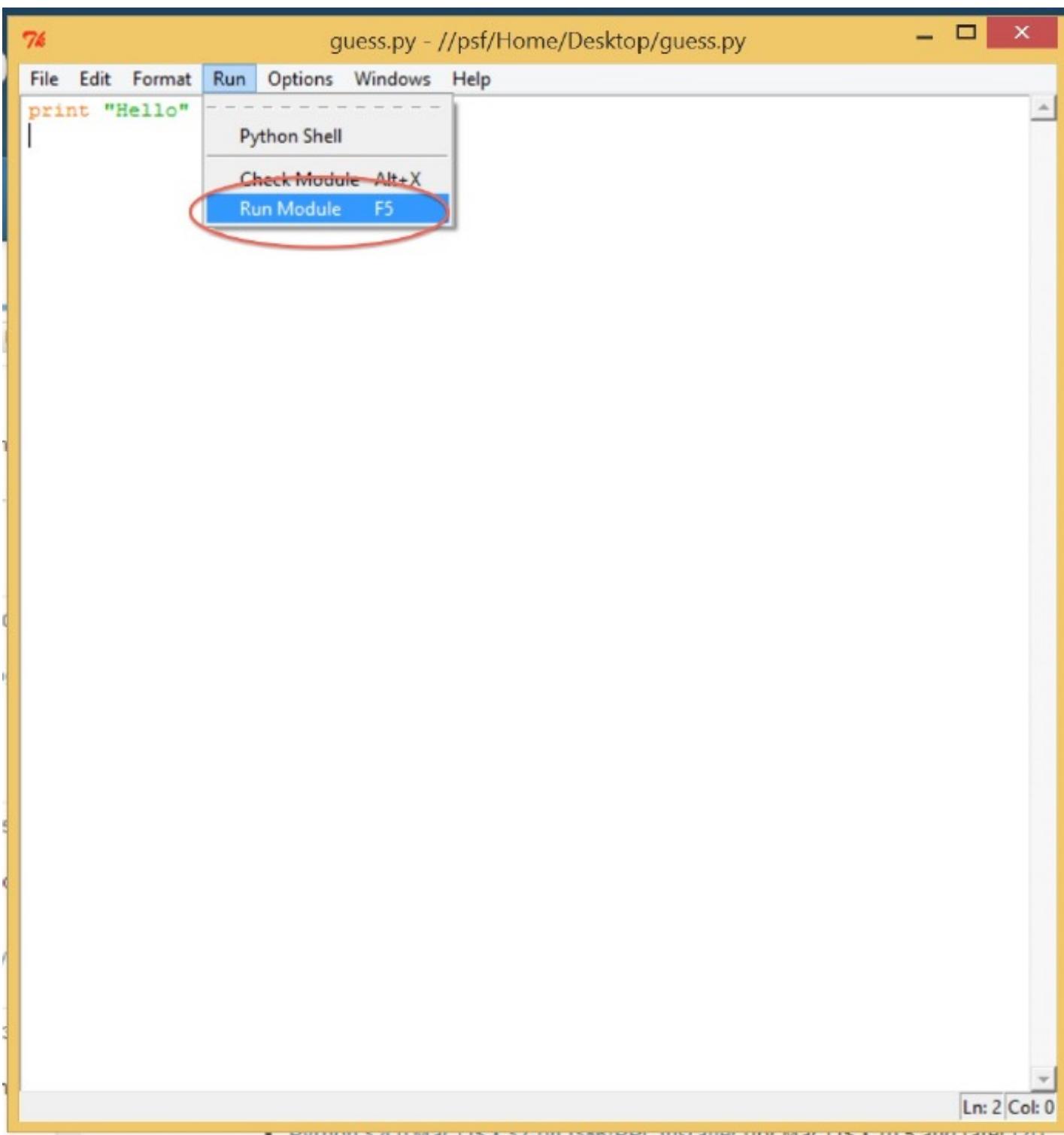
Windows

在Windows系统下，点击IDLE中的文件按钮(File) 找到新建文件(New File)。你会发现一个文本编辑器映入眼帘，你可以在里面输入你的新程序。



再次点击文件按钮，选择保存。把文件名命名为guess.py 然后把它保存在你想要保存的地方。

运行你的程序，在菜单中选择RunRun，就像这样：



游戏时间！

猜谜游戏将会生成一个随机数，然后在一个循环中，反复询问游戏者他们的猜测结果。如果用户猜中，则结束循环；否则程序会提醒用户他们的猜测是过高还是过低并继续询问结果直到猜中。

Python

```
import random  
number =  
1 import random
```

```
2 number = random.randint(0, 100)
```

上面这段程序会导入随机数模块，然后生成一个0到100之间的随机数并把它储存在变量number中。

接下来，你需要询问游戏者他们的答案。把这些代码加到程序的后面：

Python

```
guess = raw_input("Guess  
the Number: ")  
  
1 guess = raw_input("Guess the Number: ")
```

这段代码，和你猜的一样，让用户输入它们的答案并且把它保存在guess变量中。记住，现在这个变量是一个字符串，你需要把它转换为整型。

在程序中添加下面的代码，就加在上面的代码后面：

Python

```
guess_int = int(guess)  
  
1 guess_int = int(guess)
```

这个代码把用户输入的字符串转换为整型然后赋值给guess_int变量。

下面你需要比较用户猜测的值和随机数的大小。

在程序结尾添加下面的代码：

Python

```
if number > guess_int:  
    print "Too low!"  
  
1 if number > guess_int:  
2     print "Too low!"  
3 if number < guess_int:  
4     print "Too high!"  
5 if number == guess_int:  
6     print "You got it!"
```

点击菜单按钮的RunRun Module或是在终端中输入python guess.py 来运行程序；当程序提示你的时候，输入数字。发生了什么？程序在你输入数字后会把结果显示到屏幕上然后就停止了。哎呀！

你想让程序循环的询问你结果直到你猜中。你需要用一个running变量来添加这个功能。

Python

```
running = True  
  
1 running = True
```

running会被用在while循环中来控制你程序的循环。当用户输入正确的答案，程序会把running变量赋

值为False然后while就会停止。

在你让用户输入结果的前面添加下面这段代码：

Python

```
while running:  
    guess =  
  
1 while running:  
2     guess = raw_input("Guess the Number: ")  
3     ...etc
```

下面，缩进剩下的代码到同一层，这样while循环才能认出它们是循环中的代码块。

最后，你需要添加用户获胜时，把running赋值为False的语句：

Python

```
if number == guess_int:  
    print "You got it!"  
  
1 if number == guess_int:  
2     print "You got it!"  
3     running = False
```

确保if下面的两行代码被缩进两层。

运行你的程序，现在再试一下。你需要多少回才能猜中正确答案？

Python

```
Guess the number: 50  
Too Low!  
  
1 Guess the number: 50  
2 Too Low!  
3 Guess the number: 75  
4 Too Low!  
5 Guess the number: 87  
6 Too High!  
7 Guess the number: 80  
8 Too Low!  
9 Guess the number: 82  
10 Too Low!  
11 Guess the number: 84  
12 Too High!  
13 Guess the number: 83  
14 You Got It!
```

你最终的程序看上去是这个样子的：

Python

```
import random  
running = True  
  
1 import random  
2 running = True
```

```
3 number = random.randint(0, 100)
4
5 while running:
6     guess = raw_input("Guess the Number: ")
7
8     guess_int = int(guess)
9
10    if number > guess_int:
11        print "Too low!"?
12    if number < guess_int:
13        print "Too high!"
14    if number == guess_int:
15        print "You got it!"
16 running = False
```

恭喜你——你已经写出了你的第一个Python程序。别不是那么难，对吧？

从这里，到何方？

现在你已经完成了Python新手教程，我打赌你一定渴望一次挑战。尝试在你的程序中添加如下功能：

- 猜测次数统计
- 更大的随机数产生范围
- 一个电脑操控的游戏对手

如果你想要做一个稍微复杂一点的游戏，请看我的另一个教程：《[Beginning Game Programming for Teens with Python](#)》（伯乐在线翻译组已翻译[上篇](#)。）

如果你有任何的评论或是反馈，请到[论坛](#)分享！

打赏支持我翻译更多好文章，谢谢！

[打赏译者](#)

打赏支持我翻译更多好文章，谢谢！

任选一种支付方式

微信扫一扫转账



向摆铁少年ai转账

朕看的过瘾，赏小艾！

¥ 1.98

支付宝扫一扫，向我付款



¥1.99

赞赏你在伯乐在线的文章

1 赞 1 收藏 [1 评论](#)

关于作者：[艾凌风](#)



初入职场小码农,翻译组的勤务员;C/Python/在线教育/英文翻译 [个人主页](#) · [我的文章](#) · 71 ·

13岁Python开发者写给青少年的多人游戏编程 (上)

本文由 [伯乐在线 - justyoung](#) 翻译, [黄利民](#) 校稿。未经许可, 禁止转载!
英文出处: [Julian Meyer](#)。欢迎加入[翻译组](#)。

这篇博客的作者是一名13岁的Python开发者[Julian Meyer](#)。你可以在[Google+](#)和[Twitter](#)上找到他。

我确定, 你一定曾和你的朋友们一起玩过在线多人游戏。但是你是否想过这些游戏的内部是怎样实现的呢, 游戏是怎样在计算机中运行的呢?

在这个教程中, 你将通过编写一个简单的游戏来学习有关多人游戏编程。与此同时, 你也将学习到面向对象程序设计的思想。

在这个教程中, 你将会使用Python语言和它的PyGame模块。如果你刚接触Python或PyGame模块, 你应该[先阅读一下这篇文章](#), 将告诉你关于PyGame的基础知识。

着手开始吧

首先, 你需要安装PyGame模块。你可以在这个链接[here](#)上下载一个mac版的PyGame模块安装程序。如果你的系统是Mac OS 10.7或以上, 请下载Lion版的安装程序, 其它的, 则下载Snow Leopard版本。

你也可以按以下的方法下载和安装PyGame:

如果你使用MacPorts工具, 那么请用如下命令来安装: `sudo port install python2.7 py27-game`

如果你使用Fink, 那么请使用如下命令: `sudo fink install python27 pygame-py27`

如果你使用Homebrew和pip, 那么请在这个链接[here](#)中查找下载和安装PyGame的命令。

如果你使用的是windows操作系统, 那么你可以在这里[here](#)找到PyGame的安装程序。

提示: 如果你在上面的安装教程中遇到问题, 那么请确保你在系统上安装了32位版本的Python。如果你使用的是64位的系统, 那么你应该使用python2.7-32来运行Python。

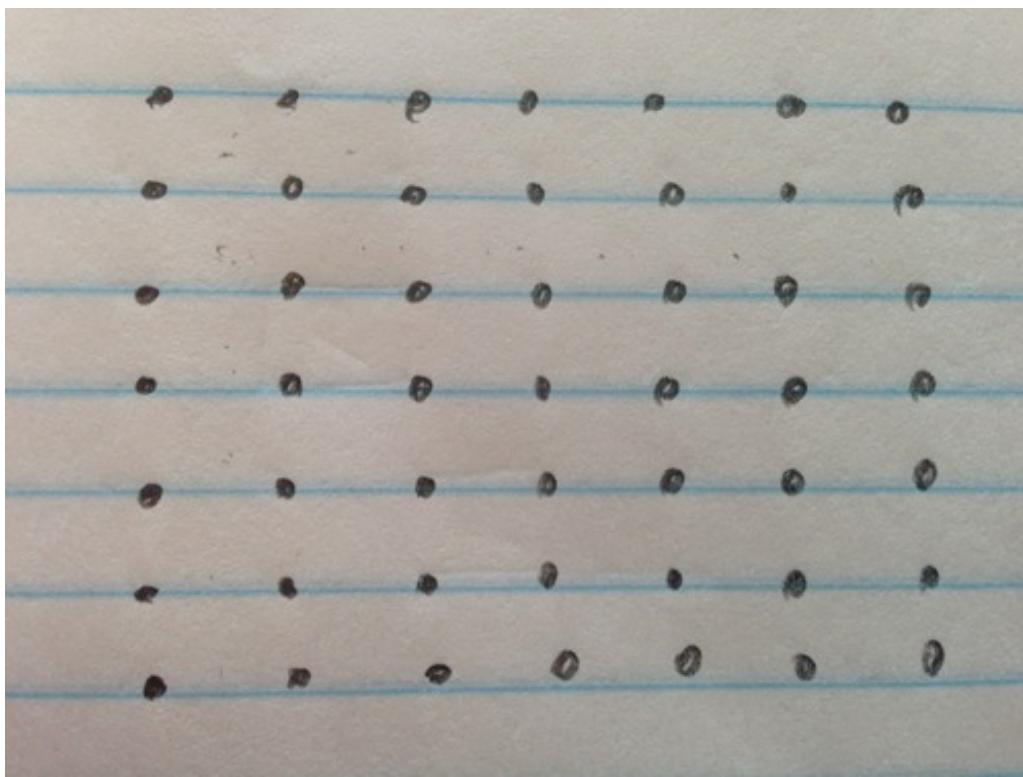
最后, 在这里下载我们这个项目所需的文件[download the resources for this project](#) (访问这个链接需要翻墙), 这些文件包括游戏所需的图像和声音。

我们的“游戏规则”

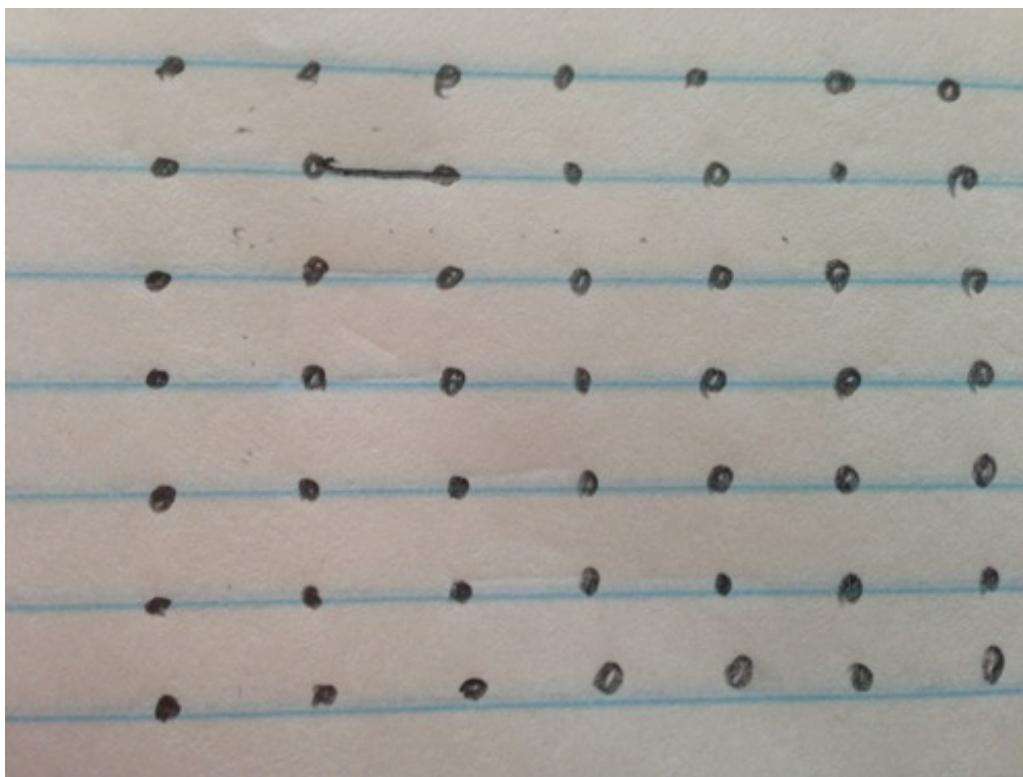
我们给教程中将要制作的游戏取名为“Boxes”。也许, 你在学校中已经和你的朋友们在纸上玩过这个游戏了。

也许你对游戏规则并不那么熟悉, 在这里, 就先介绍一下游戏规则:

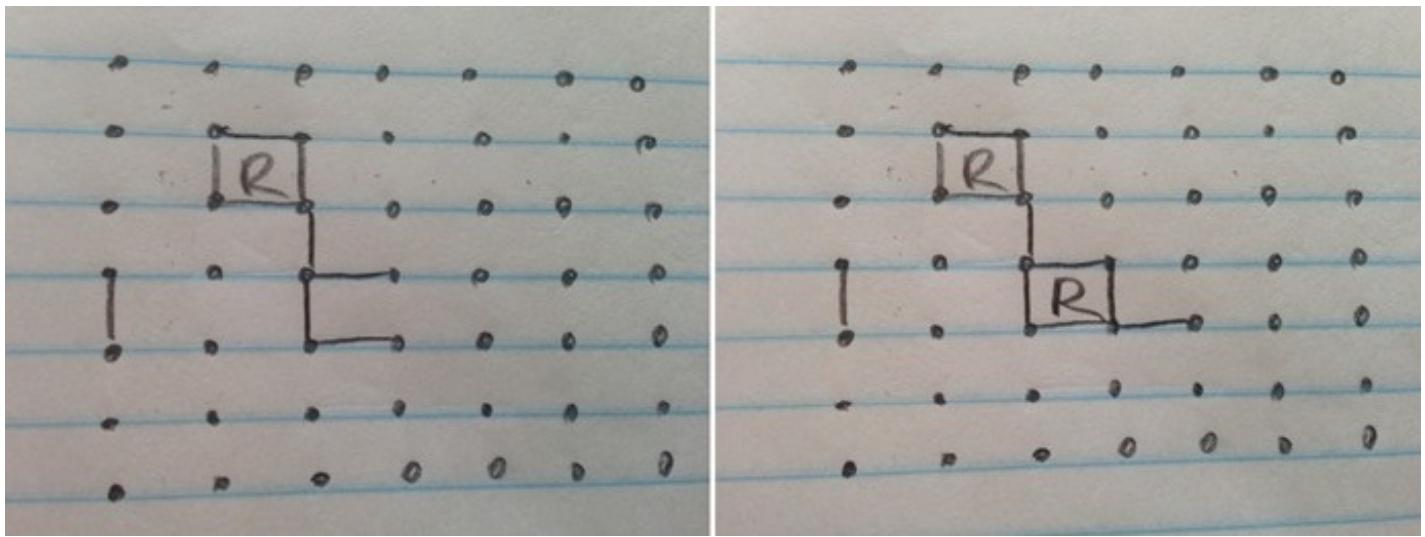
1. 游戏的棋盘包含 7×7 个网格点，如果你将这些点用线段连起来，将得到 6×6 个立方格。



2. 轮到每个玩家玩时，玩家将垂直或水平相邻的两个点用线段连接起来。



3. 如果一个玩家在网格中连接一条线后，网格中便组成了一个新格子，那么这个玩家就拥有这个新格子，并获得1分。



4 在游戏的最后，拥有格子数最多或分数最高的玩家就是胜利者。

虽然游戏的规则非常简单，但这个游戏玩起来却非常有趣，特别是当你无聊的时候。但是，如果能在线上和其他玩家玩这个游戏，是不是会更有趣呢？

面向对象程序设计简介

在我们开始编写游戏之前，先讨论一下你将在本教程中用到的面向对象程序设计思想。

面向对象程序设计，也被称为OOP，它是一个基于对象的编程方式。对象是由许多数据和与这些数据相关的逻辑组成。例如，你有一个“狗”对象，那么这个对象就包含了一些数据（例如，狗的名字、它最大的乐趣等），以及相关逻辑（例如，使狗发出叫声的指令）。

对象是由叫做“类”的模板实例化而成的，类定义了对象所包含的数据，以及这个对象能做的事情。对象的数据和它能做的事情分别称为对象的属性和方法。

方法即是一些函数，你可通过调用函数使对象完成某一任务。例如你可将car.drive()这行代码，理解为是告诉“汽车（car）”这个对象“开车（drive）”。属性是一些属于对象的变量。继续上一个例子，你的汽车“car”也许会有一个名为汽油（gas）的属性，代码car.gas = 100的意思是将汽车的汽油量设置为100。

刚才所说的两个代码操作的是一个已经存在的对象。回想我们刚才提到的，汽车（car）类是一个模板，它定义了怎样实例化一个汽车（car）对象，这个定义包括了对象的属性和方法。在对方法的定义中，你会看到对象内部的方法操作自己的代码。例如，你会看到这样的代码：self.gas=100，它不同于car.gas=100，self.gas=100的意思是汽车（car）对象告诉自己，将自己的汽油量（gas）设置为100。

OOP包含了许多内容，但是以上介绍的基础知识已经足够我们的教程了。我们用代码将Boxes游戏描述成许多对象的交互。这些对象都包含了我们在类中对其定义的若干属性和方法。当你在编写代码时，应注意，你写的类代码是从类的内部操作自身，还是操作其它外部对象。

编写一个简单的面向对象的游戏

有许多面向对象的框架可被用于我们的游戏设计。我们给Boxes游戏设计了两个对象，一个对象负责游戏的客户端，另一个对象负责游戏的服务器，现在，让我们来创建一个客户端的主类，这个类的代码将在用户启动游戏时运行。

在制作每一个游戏之前，我喜欢先为这个游戏程序创建一个文件夹。当你解压刚才下载的项目压缩文件时，将会看到一个名为boxes的文件夹。你需要将你的源文件和图像文件一起放在这个文件夹中。

在这个文件目录下，使用你最喜欢的编辑器创建一个名为boxes.py的文件（如果你没有最喜欢的编辑器，那么我推荐你在mac上使用TextEdit，或者在Windows上使用Notepad）。然后在这个文件中使用import语句：

Python

```
import pygame
```

```
1 import pygame
```

这条语句导入了PyGame模块，我们在后续编码中，将使用这个模块。在进入下一步之前，你应该首先测试一下，这个模块是否已经正确导入并可以使用了。打开终端，并用cd命令，进入我们的项目文件夹，然后在终端里输入python boxes.py命令，例如，在我的机器上应输入这样的命令：

Python

```
cd  
/Users/jmeyer/Downloads
```

```
1 cd /Users/jmeyer/Downloads/boxes
```

```
2
```

```
3 python boxes.py
```

如果你能成功执行这个命令，就说明PyGame模块已经正确的安装在你的电脑上了。我们就可以进入下一步教程了。

提示：如果你运行上述命令时，收到“No module named pygame”的错误提示，则说明你没有安装PyGame，或者你安装的PyGame版本和你系统中的Python版本不兼容。例如，如果你使用MacPorts来安装Python 2.7和PyGame，那么你将使用这个命令来安装他们：port install python2.7 py27-game PyGame，安装好之后，你需要保证在终端里调用2.7版本的Python：python2.7。如果运行上述代码得到这个错误：

Python

```
ImportError:  
/Library/Frameworks/SD
```

```
1 ImportError: /Library/Frameworks/SDL.framework/Versions/A/SDL: no appropriate 64-bit architecture (see "man  
python" for running in 32-bit mode)
```

就说明你需要运行32位模式的Python，例如：python2.7-32

接下来，像定义每一个类一样，在boxes.py文件中添加类定义代码：

Python

```
class BoxesGame():
    def __init__(self):
1 class BoxesGame():
2     def __init__(self):
3         pass
4     #put something here that will run when you init the class.
```

这段代码的第一行告诉编译器，你将创建一个名叫BoxesGame的新类。第二行定义了一个名叫__init__的方法。init两边的双下划线暗示着这是一个特殊的方法名字。事实上，这个名称__init__确定了这个类的构造方法，这个方法将在你创建或实例化一个类的对象时调用。

现在，你将在init方法中编写初始化PyGame的代码。将以下的代码紧接在原来代码中的注释部分，#put something here...:

Python

```
#1
pygame.init()
1 #1
2 pygame.init()
3 width, height = 389, 489
4 #2
5 #initialize the screen
6 self.screen = pygame.display.set_mode((width, height))
7 pygame.display.set_caption("Boxes")
8 #3
9 #initialize pygame clock
10 self.clock=pygame.time.Clock()
```

请注意输入代码的缩进格式，即刚才输入的代码都要和“#put

something here...”这段代码左对齐。你可以在这个链接中查看更多

关于Python代码缩进的内容：[Python Indentation](#).

接下来我们一段一段解释刚才添加的代码：

1. 首先，你初始化了PyGame和两个变量width、height，这两个变量是用来设置我们游戏窗体的大小的。
2. 接着，用width和height变量设置窗体的宽和高。这段代码也设置了窗体的标题。
3. 最后，你初始化了PyGame的时钟，这个时钟将会用来追踪游戏中的时间。

接下来，我们添加update()方法，这个方法每隔一段时间更新一次游戏，包括重绘界面和接收用户输入。将以下的代码添加在__init__方法之后就能实现这些功能（update代码的左缩进必须和__init__相同）：

Python

```
def update(self):
    #sleep to make the
1 def update(self):
2     #sleep to make the game 60 fps
3     self.clock.tick(60)
4
5     #clear the screen
6     self.screen.fill(0)
7
8     for event in pygame.event.get():
9         #quit if the quit button was pressed
10        if event.type == pygame.QUIT:
11            exit()
12
13    #update the screen
14    pygame.display.flip()
```

这是一个简单的循环方法，这个方法定期清除窗体中的内容并检查用户是否想退出游戏。稍后，你会在这个方法中添加更多的内容。目前，你运行这个Python文件并不会见到什么效果，因为现在你做的仅是定义了一个名为BoxesGame的类。你还需要创建这个类的对象，然后使用这个对象运行游戏！

现在，我们已经有了update方法，让我们添加运行游戏主类的方法吧。之后，你会在这个游戏中添加一些基本的图片，例如绘制游戏中的棋盘。

在源文件的末尾添加这些代码来运行我们编写的游戏（代码的左缩进必须与文件的左缩进相同）：

Python

```
bg=BoxesGame()
#__init__ is called right
1 bg=BoxesGame() #__init__ is called right here
2 while 1:
3     bg.update()
```

这三行代码体现了面向对象程序设计的一个优点：真正让程序运行的代码其实只有三行。

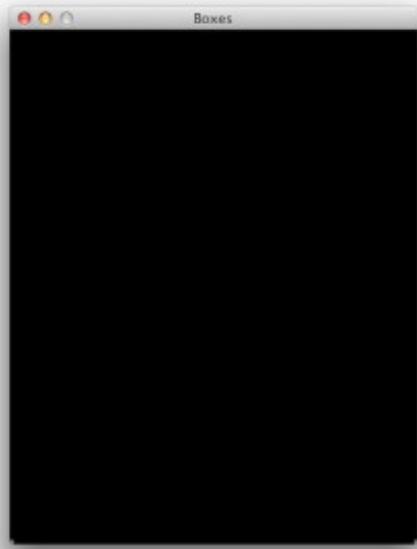
至此，整个源文件的内容应该是这样的：

Python

```
import pygame
1 import pygame
2
3 class BoxesGame():
4     def __init__(self):
5         pass
6         #1
7         pygame.init()
8         width, height = 389, 489
9         #2
10        #initialize the screen
11        self.screen = pygame.display.set_mode((width, height))
12        pygame.display.set_caption("Boxes")
13        #3
14        #initialize pygame clock
```

```
15     self.clock=pygame.time.Clock()
16 def update(self):
17     #sleep to make the game 60 fps
18     self.clock.tick(60)
19
20     #clear the screen
21     self.screen.fill(0)
22
23     for event in pygame.event.get():
24         #quit if the quit button was pressed
25         if event.type == pygame.QUIT:
26             exit()
27
28     #update the screen
29     pygame.display.flip()
30
31 bg=BoxesGame() #__init__ is called right here
32 while 1:
33     bg.update()
```

就这样，是不是很简单？现在，让我们来运行这个游戏：



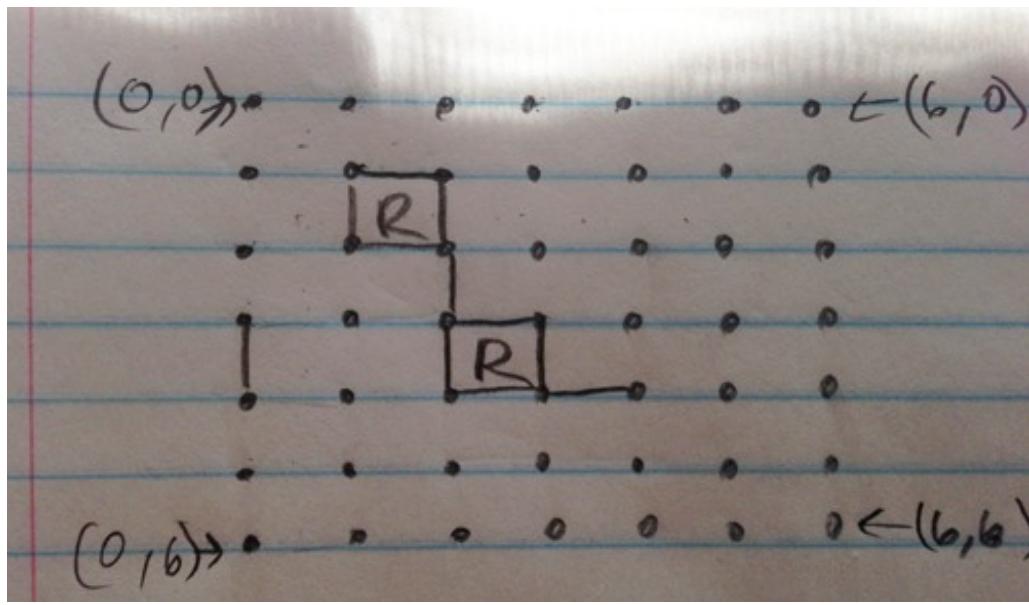
正如你看到的那样，游戏运行的结果就是看到一个令人非常印象深刻的黑色界面。

也许你现在并不理解这些代码，但是游戏的编写就像是一个战略过程。把自己想象成一个建筑设计师来编写游戏。你刚刚就为你的建筑打下了一个坚固的地基。每个雄伟的建筑都有一个良好的奠基，所以在你开始编写游戏之前，需要首先做好规划。

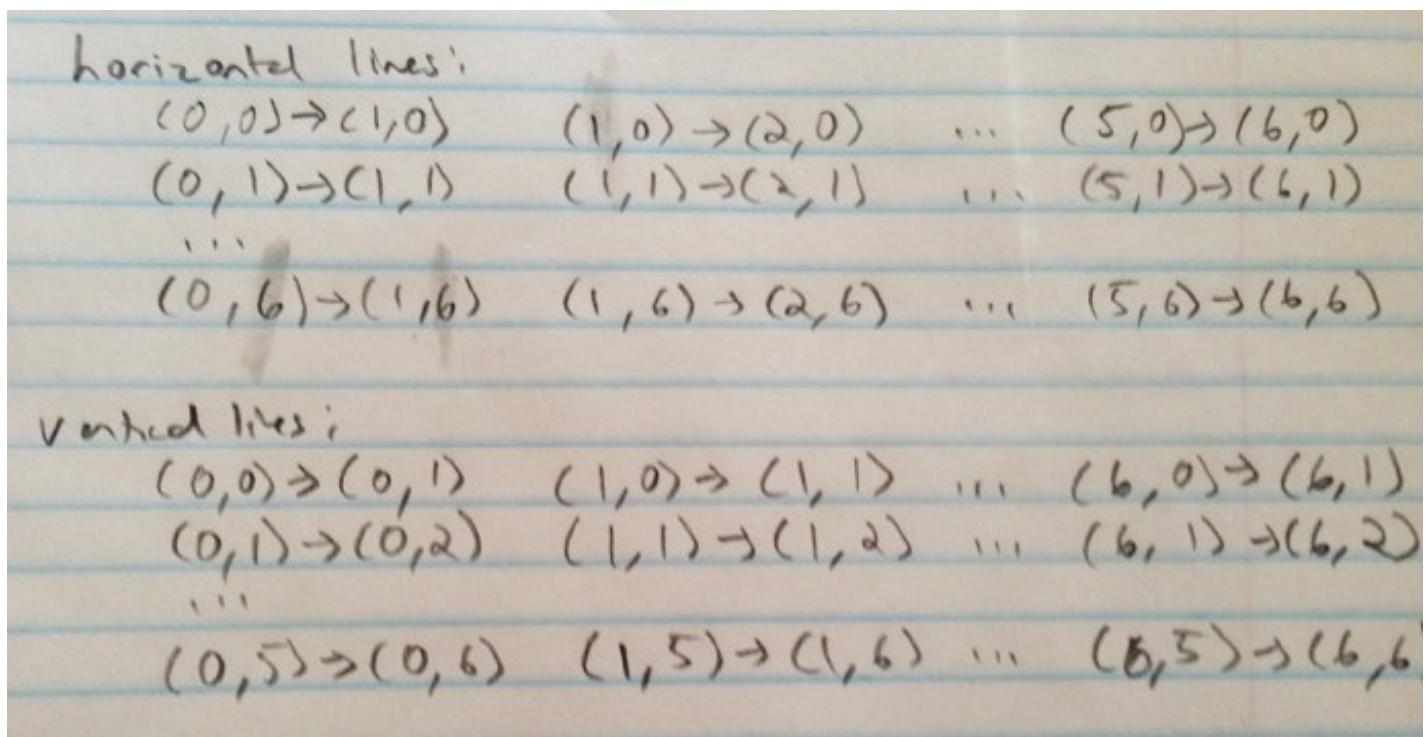
让我们添加另一个方法。如果你不记得什么是方法的话，那么你可以再看看教程中的“面向对象程序设计简介”这部分内容。

在游戏界面中绘制棋盘和线段

PyGame将窗体的左上角的坐标定义为(0,0)。所以我们为Boxes中的网格点定义一个坐标系统，其中(0,0)表示左上角的点，(6,6)表示右下角的点：



需要用某种方法来表示游戏中所有可能的线段。在游戏中，我们有垂直和水平的两种线段。让我们考虑一下，如何用一个列表来表示所有的线段集合，这是一个可以表示所有垂直和水平线段集合的方法：



从程序设计的角度来说，一个列表也被称为一个数组。当你的列表元素也是列表时，这个列表就被称为二维数组，例如水平和垂直线段的集合。

举一个例子，如果要表示从(0,0)点到(1,1)点水平方向上的路线，那么，就应该在“horizontal lines”这个列表中选择第0行，第0列的元素来表示。

请注意，“horizontal lines”列表有6行7列，而“vertical lines”有7行6列。（这里按照作者的代

码，horizontal lines应该是7行6列，vertical lines是6行7列)

将下面两行代码添加到__init__代码中，来定义“horizontal lines”和“vertical lines”这两个数组：

Python

```
self.boardh = [[False for x in range(6)] for y in range(7)]
1 self.boardh = [[False for x in range(6)] for y in range(7)]
2 self.boardv = [[False for x in range(7)] for y in range(6)]
```

[valuePerItem for x in y]，这个语句可以快速创建一个数组。上述代码在创建数组的同时还将数组的元素初始化为False。False代表一个空的区域。

至此，你就有了表示棋盘的方法了，接下来我们来看看怎样用代码画出这个棋盘。

首先，新建一个名为initGraphics()的方法。这个方法将被__init__方法调用，但为了使你的代码结构保持清晰，我们将载入图片的代码封装到其它方法中。在__init__方法之前，添加这个代码：

Python

```
def initGraphics(self):
1 def initGraphics(self):
2     self.normallinev=pygame.image.load("normalline.png")
3     self.normallineh=pygame.transform.rotate(pygame.image.load("normalline.png"), -90)
4     self.bar_donev=pygame.image.load("bar_done.png")
5     self.bar_doneh=pygame.transform.rotate(pygame.image.load("bar_done.png"), -90)
6     self.hoverlinev=pygame.image.load("hoverline.png")
7     self.hoverlineh=pygame.transform.rotate(pygame.image.load("hoverline.png"), -90)
```

正如你看到的那样，我们有三个垂直的线条图像：一个普通（空）线条，一个已被画过（占用）的线条和一个悬浮效果线条。将这些垂直的线条图像旋转90度，就可以表示水平的线条了。线条的图像保存在你下载的项目资源文件夹里，并且它们必须和你的python文件同处一个目录下。

你已经有了一个载入图像的方法，但是你需要调用它。猜一猜应该在哪添加调用的代码？

当你有了答案时，点击下面的“show”按钮，看看你是否答对了。

在__init__方法的末尾添加这个代码：

Python

```
#initialize the graphics
self.initGraphics()
1 #initialize the graphics
2 self.initGraphics()
```

接下来，你应该添加绘制棋盘的代码。若要循环整个棋盘上的x坐标和y坐标，你就必须在一个for循环中再嵌套一个for循环。每一个for循环可以循环x或y方向上的所有点。在__init__方法之后添加这段代码：

Python

```
def drawBoard(self):
    for x in range(6):
        def drawBoard(self):
            for x in range(6):
                for y in range(7):
                    if not self.boardh[y][x]:
                        self.screen.blit(self.normallineh, [(x)*64+5, (y)*64])
                    else:
                        self.screen.blit(self.bar_doneh, [(x)*64+5, (y)*64])
            for x in range(7):
                for y in range(6):
                    if not self.boardv[y][x]:
                        self.screen.blit(self.normallinev, [(x)*64, (y)*64+5])
                    else:
                        self.screen.blit(self.bar_donev, [(x)*64, (y)*64+5])
```

这段代码分别循环了垂直的和水平的线段组成的列表，并检查每个线段是否被点击选中了。
self.boardv[y][x]和self.boardh[y][x]返回true或false取决于这条线段是否被玩家选中了。

现在执行这个程序仍然不会有任何作用。现在你完成的代码仅定义了在drawBoard这个方法被调用时，drawBoard所要做的事情。现在让我们在update方法中添加对drawBoard的调用吧。在清空窗体的语句screen.fill(0)后添加下面的代码：

Python

```
#draw the board
self.drawBoard()
```

```
1 #draw the board
2 self.drawBoard()
```

当然，作为一个优秀的程序员，请记住在你的代码中添加注释，解释你刚才写的代码。

现在可以运行你刚写的代码了，程序运行后，你将看到界面上出现了你绘制的网格：



每次我写绘制图片的代码时，我都会对这些代码做一点测试，因为这样很有趣，而且也能发现一些BUG。在定义self.boardh和self.boardy的代码的后面添加这个代码：

Python

```
self.boardh[5][3]=True
```

```
1 self.boardh[5][3]=True
```

运行修改后的代码，你将看到，从(5,3)到(5,4)这条水平的线将会亮起：



很酷对吧？现在可以删除我们刚才添加的测试代码了。好，现在你已经成功完成棋盘的绘制了，这是我们游戏编程中的难点之一。

添加其它类型的线条

下一步，你需要找到离鼠标指针最近的一个线条，然后在那个位置绘制一个有悬浮效果的线条。

首先，在代码源文件的顶部写下这行代码，来导入我们马上用到的数学库：

Python

```
import math
```

```
1 import math
```

然后在pygame.display.flip()之前，添加以下这一大段代码：

Python

```
#1  
mouse =
```

```
1 #1  
mouse = pygame.mouse.get_pos()
```

```
2
3 #2
4 xpos = int(math.ceil((mouse[0]-32)/64.0))
5 ypos = int(math.ceil((mouse[1]-32)/64.0))
6
7 #3
8 is_horizontal = abs(mouse[1] - ypos*64) < abs(mouse[0] - xpos*64)
9
10 #4
11 ypos = ypos - 1 if mouse[1] - ypos*64 < 0 and not is_horizontal else ypos
12 xpos = xpos - 1 if mouse[0] - xpos*64 < 0 and is_horizontal else xpos
13
14 #5
15 board=self.boardh if is_horizontal else self.boardv
16 isoutofbounds=False
17
18 #6
19 try:
20     if not board[ypos][xpos]: self.screen.blit(self.hoverlineh if is_horizontal else self.hoverlinev, [xpos*64+5 if
21 is_horizontal else xpos*64, ypos*64 if is_horizontal else ypos*64+5])
22 except:
23     isoutofbounds=True
24     pass
25 if not isoutofbounds:
26     alreadyplaced=board[ypos][xpos]
27 else:
28     alreadyplaced=False
```

哇，好长一段代码。我们一段一段来看看这些代码吧：

1 首先，你通过PyGame的内建函数来获得鼠标指针的位置。

2 接下来，在前面的代码中，我们将网格的大小设置成了 64×64 ，所以我们可以计算获得鼠标指针在棋盘网格中的位置。

3 你需要检查你的鼠标指针是更接近方块的上边、下边还是是方块的左边、右边。因为你需要判断用户的鼠标是悬浮在一条水平的线条上还是垂直的线条上。

4 通过`is_horizontal`这个变量，计算线条的新位置。

5 用`boardh`或`boardv`初始化`board`变量。

6 最后，你需要将悬浮效果的线画到界面上，你必须考虑是画水平的线还是垂直的线，以及这个线条画在一个方块的上边、下边、左边或是右边。你还必须检查你画的线条是否超出了棋盘的边界。如果线条超出了边界或线条已经被画过了，那么你就不需要在这样的线条上添加悬浮效果。

运行这个程序，你会惊喜地发现，当你的鼠标靠近一个线条时，线条就会亮起。

如果你像我一样，现在就一定会激动得将鼠标在屏幕上移来移去。现在就请尽情享受你的成果吧！

好，现在当用户鼠标移到网格上的某线条时，那条线就会亮起。但是，你所写的不仅仅是一个一直移动鼠标的游戏。你还需要增加这样一个功能：当鼠标点击一个线条时，这个线条就表示被玩家画过，也就是玩家拥有了这个线条。

要完成这个功能，你需要使用PyGame的内建函数：`pygame.mouse.get_pressed()[0]`。这个函数返

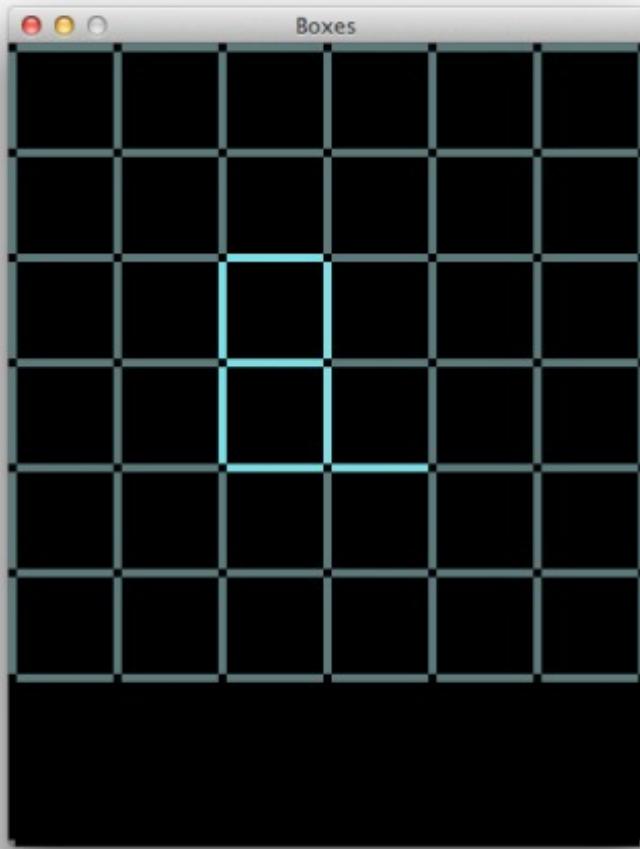
回1或0取决于玩家是否点击了这个线条。在我告诉你如何实现这个功能前，你可以仔细想一想应该怎样做呢？回想一下，我们刚才是怎样使用if语句的，以及怎样在界面上绘制网格的。

将这些代码直接添加到刚才那段代码的后面：

Python

```
if
pygame.mouse.get_pre()
1 if pygame.mouse.get_pressed()[0] and not alreadyplaced:
2     if is_horizontal:
3         self.boardh[ypos][xpos]=True
4     else:
5         self.boardv[ypos][xpos]=True
```

现在运行修改过的程序，如果你点击鼠标，你将会在线条悬浮的地方画上这个线条。正如你看到的，你的代码完成了这些工作：你放置的线条取决于鼠标是否被单击，以及线条是水平的还是垂直的。



这个代码有一个问题，如果你在棋盘网格的下方点击鼠标，那么我们写的游戏将崩溃。让我们看看造成这个现象的原因吧。通常，当一个错误发生时，你能在终端上看到一个错误报告，这里我们遇到的错误报告是这样的：

Python

```
Traceback (most recent call last):
```

```
1 Traceback (most recent call last):
2   File "/Users/school/Desktop/Dropbox/boxes/WIPBoxes.py", line 103, in <module>
3     bg.update()
4   File "/Users/school/Desktop/Dropbox/boxes/WIPBoxes.py", line 69, in update
5     self.boardh[ypos][xpos]=True
6 IndexError: list index out of range
```

这个错误是因为boardh数组越界了。还记得那个isoutofbounds变量吗？这个变量可以为我们解决数组越界的问题，只要简单地像下面这样修改一行代码即可：

Python

```
if
pygame.mouse.get_pre
1 if pygame.mouse.get_pressed()[0] and not alreadyplaced:
2
3 #-----to-----
4
5 if pygame.mouse.get_pressed()[0] and not alreadyplaced and not isoutofbounds:
```

现在，如果你在棋盘网格外点击鼠标，游戏也不会崩溃。好的，你刚才就算做了调试的工作！

在你实现游戏的服务端逻辑代码前，我们先在客户端添加一些最后的润色代码。

最后的润色

有一个问题一直困扰着我，那就是界面上网格线交叉部分的空缺。幸运的是，你可以很容易地用一个 7×7 大小的图片来填补这个空缺。当然，你需要一个图像文件，因此，我们现在就一口气载入这张图片以及项目中你需要使用的所有图片吧。

在initGraphics()后添加这些代码：

Python

```
self.separators=pygame
.image.load("separators"
1 self.separators=pygame.image.load("separators.png")
2 self.redindicator=pygame.image.load("redindicator.png")
3 self.greenindicator=pygame.image.load("greenindicator.png")
4 self.greenplayer=pygame.image.load("greenplayer.png")
5 self.blueplayer=pygame.image.load("blueplayer.png")
6 self.winningscreen=pygame.image.load("youwin.png")
7 self.gameover=pygame.image.load("gameover.png")
8 self.score_panel=pygame.image.load("score_panel.png")
```

现在，你载入了需要的图片文件，让我们将载入的图片绘制到这49个空缺点上吧。将以下的代码添加到drawBoard()：

Python

```
#draw separators
```

```
for x in range(7):  
    self.screen.blit(self.separators, [x*64, 0])
```

好的，现在我们来测试一下游戏吧。游戏运行后，你将会看到一个更好看的网格棋盘。



接下来，让我们在游戏界面的下方放置一个平视显示器（HUD）吧。首先，你需要新建一个 drawHUD() 方法。

Python

```
def drawHUD(self):  
    #draw the
```

```
1 def drawHUD(self):  
2     #draw the background for the bottom:  
3     self.screen.blit(self.score_panel, [0, 389])
```

这段代码是在游戏界面的背景上绘制得分的面板。

让我说明一下PyGame处理字体的三个步骤：

1 首先，你需要设置一个字体以及这个字体的大小。

2 接下来，你调用font.render("your text here")，使你输入的文本以你设置的字体呈现。

3 然后，将这些字像图片那样绘制到游戏的界面上。

现在，你已经知道怎样在界面上放置文本了，我们就可以绘制HUD的另一部分：“Your Turn”提示文字。在drawHUD()方法的尾部添加以下的代码：

Python

```
#create font  
myfont =  
  
1 #create font  
2 myfont = pygame.font.SysFont(None, 32)  
3  
4 #create text surface  
5 label = myfont.render("Your Turn:", 1, (255,255,255))  
6  
7 #draw surface  
8 self.screen.blit(label, (10, 400))
```

同样在pygame.init()的后面添加这行代码：

Python

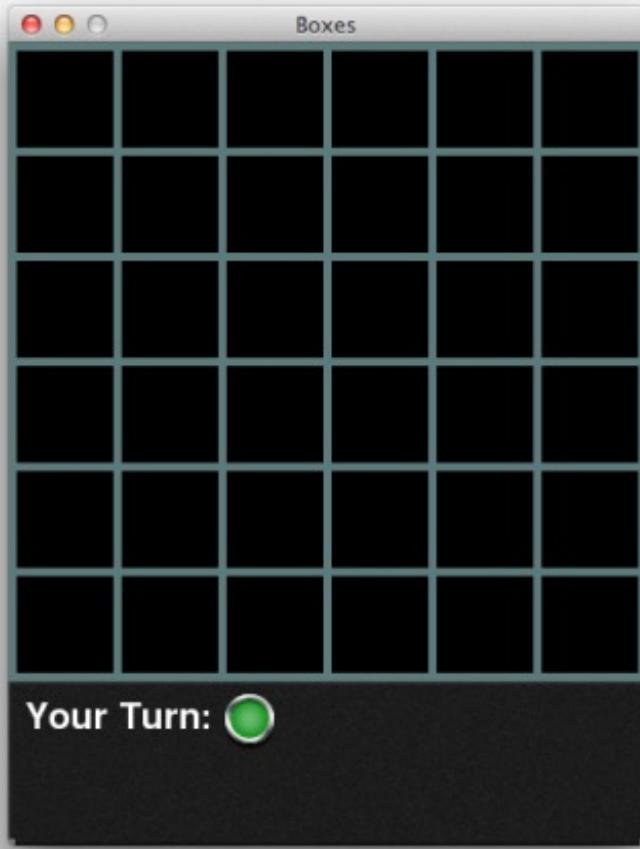
```
pygame.font.init()  
  
1 pygame.font.init()
```

刚才添加的代码建立了字体，将字体的颜色设置成白色，然后将这些字绘制到游戏界面上。在运行游戏之前，将这行代码添加到self.drawBoard()的后面：

Python

```
self.drawHUD()  
  
1 self.drawHUD()
```

运行这个程序后，你会看到“Your Turn”这个文本出现在游戏界面的下方。如果你仔细看游戏界面的下方，你会发现精细的背景质地。



这很棒不是吗，但是你仍然需要在“Your Turn”之后添加一个指示图标，来提醒玩家轮到他们了。

在做这之前，你需要让游戏知道，这个游戏轮到谁了。所以你需要在`__init__`代码的尾部添加这行代码。

Python

```
self.turn = True
```

```
1 self.turn = True
```

将这行代码添加到`drawHUD()`的尾部，以在界面上绘制指示图片。

Python

```
self.screen.blit(self.greenin  
dicator, (130, 395))
```

```
1 self.screen.blit(self.greenindicator, (130, 395))
```

现在运行游戏后，你将会看到绿色的指示器。好了，你可以把这个任务从你的未完成事项中去除了。

接下来，让我们创建每个玩家的分数文本。使用如下的代码初始化两个玩家的分数变量，将这些代码添加在`__init__`的末尾：

Python

```
self.me=0  
self.otherplayer=0
```

```
1 self.me=0  
2 self.otherplayer=0  
3 self.didiwin=False
```

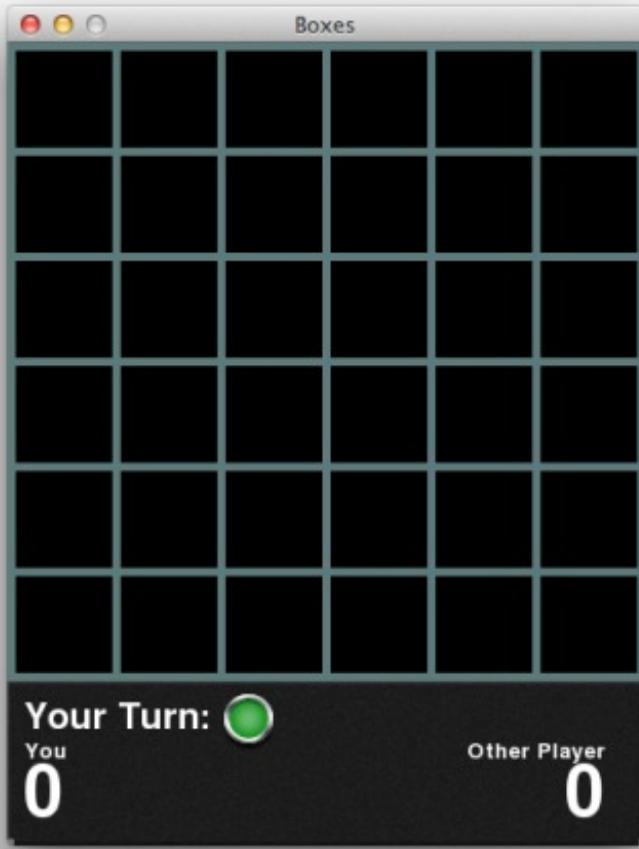
这里，你需要添加一些其他你会在本阶段使用的变量。还记得怎样添加文本吗？现在你将重复写一些刚才写过的代码，但是使用的是不同大小的字体。将这些代码添加到drawHUD()的尾部：

Python

```
#same thing here  
myfont64 =
```

```
1 #same thing here  
2 myfont64 = pygame.font.SysFont(None, 64)  
3 myfont20 = pygame.font.SysFont(None, 20)  
4  
5 scoreme = myfont64.render(str(self.me), 1, (255,255,255))  
6 scoreother = myfont64.render(str(self.otherplayer), 1, (255,255,255))  
7 scoretextme = myfont20.render("You", 1, (255,255,255))  
8 scoretextother = myfont20.render("Other Player", 1, (255,255,255))  
9  
10 self.screen.blit(scoretextme, (10, 425))  
11 self.screen.blit(scoreme, (10, 435))  
12 self.screen.blit(scoretextother, (280, 425))  
13 self.screen.blit(scoreother, (340, 435))
```

运行程序，检查一下你的成果吧。



现在你已经完成了HUD的功能。现在还需要做一些工作来完善客户端，原谅我吧.....

接下来，我们添加一个简单的变量来表示玩家所拥有的网格线。这些变量能让你跟踪玩家所拥有的方格。你需要使用这个变量正确地给玩家所拥有的方块上色并记下玩家得分。请记住，拥有最多方格的玩家将获得游戏的胜利！

首先，在`__init__`的末尾初始化一个数组：

Python

```
self.owner = [[0 for x in  
range(6)] for y in range(6)]  
  
1 self.owner = [[0 for x in range(6)] for y in range(6)]
```

和之前画网格线的方法一样，循环二维数组在屏幕上绘制玩家所拥有的网格。在类的底部添加这个方法。

Python

```
def  
drawOwnermap(self):  
  
1 def drawOwnermap(self):  
2   for x in range(6):
```

```
3     for y in range(6):
4         if self.owner[x][y]!=0:
5             if self.owner[x][y]=="win":
6                 self.screen.blit(self.marker, (x*64+5, y*64+5))
7             if self.owner[x][y]=="lose":
8                 self.screen.blit(self.othermarker, (x*64+5, y*64+5))
```

这个方法判断每一个给定的方块是否需要着色，如果需要，方法将给方块画上指定的颜色（每一个玩家都有自己的颜色）。

你还需要给游戏增加一个胜利和失败的界面。下面的方法就能完成这个功能，将它添加到类的底部：

Python

```
def finished(self):
```



```
1 def finished(self):
2     self.screen.blit(self.gameover if not self.didiwin else self.winningscreen, (0,0))
3     while 1:
4         for event in pygame.event.get():
5             if event.type == pygame.QUIT:
6                 exit()
7         pygame.display.flip()
```



当然，现在你还没有办法在游戏中触发这个界面。在下一部分的教程里，当你实现了游戏的服务端后，你就可以查看这个界面了。

请记住，在你添加完这些界面后，游戏的服务端就可以任意操纵客户端了。除了服务端和客户端之间的通信问题需要一点处理外，从现在开始，你不需要对客户端进行改动了。

但是，为了确保我们刚才编写的方法是正确的，我们在`__init__`方法的尾部调用`finished()`。你将看到一个向上面那幅图一样的game over的画面。



接下来要做什么呢？

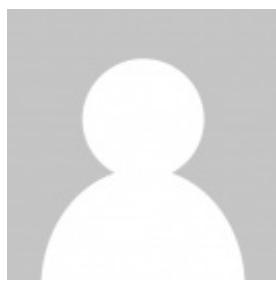
这是到目前为止，本教程的源代码可在这里找到[source code](#)。

恭喜你！你已经完成了一个整洁而漂亮的游戏客户端。当然，虽然你出色地完成了游戏的客户端代码，但是并没有实现任何游戏的逻辑，所以我们的开发工作并没有结束。

现在你需要[进入我们教程的第二部分](#)，教程的第二部分主要讲述游戏的服务端，随着教程，你将开始制作真正的多玩家游戏。

1 赞 2 收藏 [评论](#)

关于作者：[justyoung](#)



可怜的小硕 [个人主页](#) · [我的文章](#) · 10

高性能的Python扩展（3）

本文由[伯乐在线 - Janzou](#)翻译，[黄利民](#)校稿。未经许可，禁止转载！

英文出处：[crumpington](#)。欢迎加入[翻译组](#)。

- [高性能的Python扩展（1）](#)
- [高性能的Python扩展（2）](#)

简介

本文是这个系列的第三篇，我们关注于使用NumPy API为Python编写高性能的C扩展模块。在本文中，我们将使用OpenMP来并行[第二部分](#)中的实现。

回顾

`Wrold`是存储N体状态的一个类。我们的模拟将演化一系列时间步长下的状态。

Python

```
class World(object):
    """World is a
1 class World(object):
2     """World is a structure that holds the state of N bodies and
3     additional variables.
4
5     threads : (int) The number of threads to use for multithreaded
6         implementations.
7     dt    : (float) The time-step.
8
9     STATE OF THE WORLD:
10
11    N : (int) The number of bodies in the simulation.
12    m : (1D ndarray) The mass of each body.
13    r : (2D ndarray) The position of each body.
14    v : (2D ndarray) The velocity of each body.
15    F : (2D ndarray) The force on each body.
16
17    TEMPORARY VARIABLES:
18
19    Ft : (3D ndarray) A 2D force array for each thread's local storage.
20    s : (2D ndarray) The vectors from one body to all others.
21    s3 : (1D ndarray) The norm of each s vector.
22
23    NOTE: Ft is used by parallel algorithms for thread-local
24        storage. s and s3 are only used by the Python
25        implementation.
26    """
27    def __init__(self, N, threads=1,
28                 m_min=1, m_max=30.0, r_max=50.0, v_max=4.0, dt=1e-3):
29        self.threads = threads
30        self.N = N
31        self.m = np.random.uniform(m_min, m_max, N)
32        self.r = np.random.uniform(-r_max, r_max, (N, 2))
33        self.v = np.random.uniform(-v_max, v_max, (N, 2))
```

```
34     self.F = np.zeros_like(self.r)
35     self.Ft = np.zeros((threads, N, 2))
36     self.s = np.zeros_like(self.r)
37     self.s3 = np.zeros_like(self.m)
38     self.dt = dt
```

在开始模拟时，N体被随机分配质量m，位置r和速度v。对于每个时间步长，接下来的计算有：

1. 合力F，每个体上的合力根据所有其他体的计算。
2. 速度v，由于力的作用每个体的速度被改变。
3. 位置r，由于速度每个体的位置被改变。

计算力：串行代码

下面是[之前文章](#)实现中（全部的源代码在[这里](#)）的compute_F函数。这个函数计算模拟中每对体之间的相互作用力，其复杂度为O(N^2)。

Python

```
static inline void
{
    static inline void compute_F(npy_int64 N,
                                npy_float64 *m,
                                __m128d *r,
                                __m128d *F) {
        npy_int64 i, j;
        __m128d s, s2, tmp;
        npy_float64 s3;
        // Set all forces to zero.
        for(i = 0; i < N; ++i) {
            F[i] = _mm_set1_pd(0);
        }
        // Compute forces between pairs of bodies.
        for(i = 0; i < N; ++i) {
            for(j = i + 1; j < N; ++j) {
                s = r[j] - r[i];
                s2 = s * s;
                s3 = sqrt(s2[0] + s2[1]);
                s3 *= s3 * s3;
                tmp = s * m[i] * m[j] / s3;
                F[i] += tmp;
                F[j] -= tmp;
            }
        }
    }
}
```

性能

使用这一系列的实现，我们的代码在i5台式机上能每秒执行26427个时间步长。

计算力：并行代码

下面是重新实现的compute_F函数，它使用OpenMP来并行循环。完整的源文件src/simdomp1.c可以在[github](#)上获得。

Python

```
static inline void
{
    static inline void compute_F(npy_int64 threads,
                                npy_int64 N,
                                npy_float64 *m,
                                __m128d *r,
                                __m128d *F,
                                __m128d *Ft) {
        npy_int64 id, i, j, Nid;
        __m128d s, s2, tmp;
        npy_float64 s3;
    }

    #pragma omp parallel private(id, i, j, s, s2, s3, tmp, Nid)
    {
        id = omp_get_thread_num();
        Nid = N * id; // Zero-index in thread-local array Ft.

        // Zero out the thread-local force arrays.
        for(i = 0; i < N; i++) {
            Ft[Nid + i] = _mm_set1_pd(0);
        }

        // Compute forces between pairs of bodies.
        #pragma omp for schedule(dynamic, 8)
        for(i = 0; i < N; ++i) {
            F[i] = _mm_set1_pd(0);

            for(j = i + 1; j < N; ++j) {
                s = r[j] - r[i];
                s2 = s * s;
                s3 = sqrt(s2[0] + s2[1]);
                s3 *= s3 * s3;

                tmp = s * m[i] * m[j] / s3;

                Ft[Nid + i] += tmp;
                Ft[Nid + j] -= tmp;
            }
        }

        // Sum the thread-local forces computed above.
        #pragma omp for
        for(i = 0; i < N; ++i) {
            for(id = 0; id < threads; ++id) {
                F[i] += Ft[N * id + i];
            }
        }
    }
}
```

线程局部存储

串行版本和并行版本之间最明显不同在于Ft数组的出现。Ft数组通过每个同时运行的线程为力的计

算提供线程局部存储。这些局部的数组然后在一个单独的循环中相加，从而得到一个力F的数组。局部存储对于避免竞争条件是有必要的。可以想象一下，如果一个线程正在执行 $i=0$ 和 $j=1$ ，而此时另一个线程正在使用 $i=1$ 和 $j=2$ 。在我们的算法中，它们都试图修改 $F[1]$ ，这导致了一种竞争。你可以在[维基百科](#)上阅读有关竞争条件（race conditions）的内容。

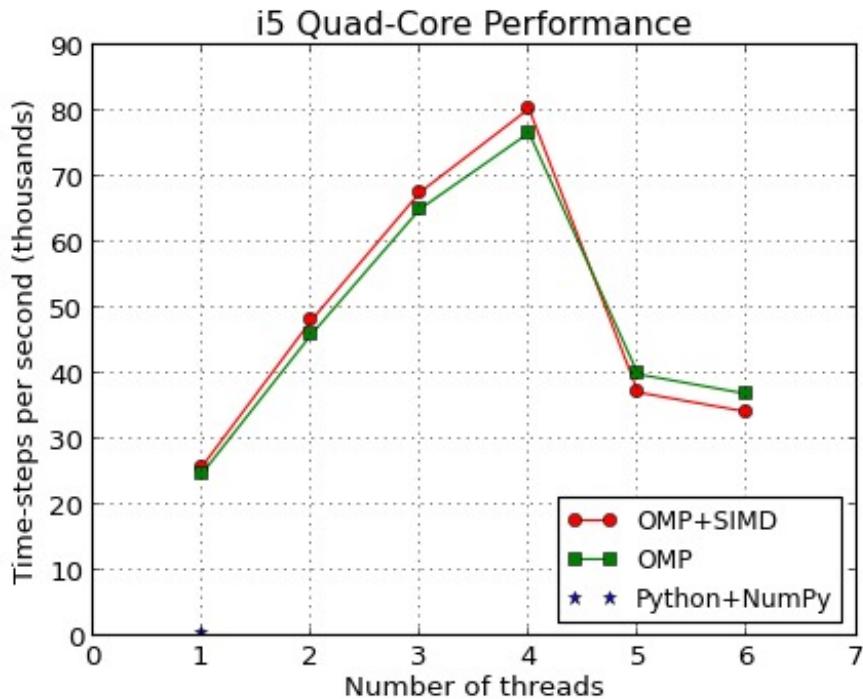
OpenMP

OpenMP API通过使用如`omp_get_thread_num`这样的函数和如`#pragma omp parallel`这样的指令来执行。在上面的代码中，我们创建了许多有`#pragma omp parallel`指令的线程。`private`指令列出了每个线程的私有变量。所有其他变量是公有的。`#pragma omp`指令允许`for`循环并行进行，每个线程处理不同的指数。我们使用`schedule(dynamic, 8)`指令来告诉编译器使用8块大小的动态调度。当每个循环花费的时间不同时，动态调度是有用的，正如这里的这种情况。需要注意的是，在每个并行`for`循环的末尾有一个隐式屏障。所有的线程在任何可以提前之前都必须完成循环。在[OpenMP 网站](#)上参见有关该API的更多信息。

性能

因为我们增加了一个额外的`for`循环来相加局部线程的力的数组，我们可以预期单核性能会稍受损失。在我们的测试中，在使用单线程时，OpenMP版本的运行速度比没有OpenMP的版本慢约3%。在Intel i5台式机上，使用四核及四线程，OpenMP使用SIMD指令执行，每秒80342个时间步长。这比我们原来使用NumPy的Python实现快312倍，比我们最快的串行实现快3倍。

扩展性



上图显示OpenMP代码是如何在四核i5台式机上扩展线程数量的。标有“OMP+SIMD”的点对应的是本文中的实现。一个类似的版本可以在[这里](#)获得，没有明确的向量指令标有“OMP”。使用Python和NumPy的原始版本被用于比较（左下）。性能可以很好的达到物理核的数量，并大幅减少额外线程的增加。请注意，这不是一般的事，所以使用不同数量的线程来衡量你的代码总是一个好主意。

环境变量

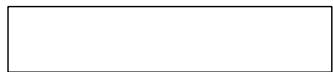
这里有许多环境变量可以改变使用OpenMp程序的行为。这些都对性能有重大的影响，特别是对有多个物理处理器的计算机。请参阅[GNU libgomp 文档](#)了解详细信息。

性能总结：所有实现

这里是这个系列文章中所有实现版本的基准。这个运行在一个四核Intel i5台式机上，在Debian Wheezy系统下使用gcc 4.7.2。

实现	时间步长/每秒 加速	
Python+Numpy	257	1
Simple C 1	17974	70
Simple C 2	26136	102
SIMD	26423	103
OMP 4 cores	76657	298
OMP+SIMD 4 cores	80342	313

Python

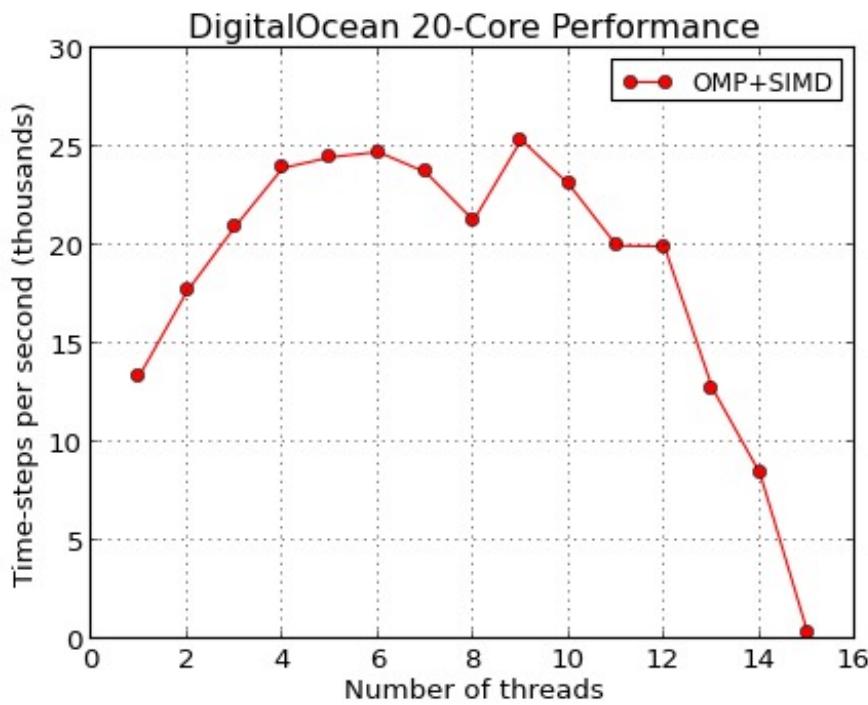


1

附加测试

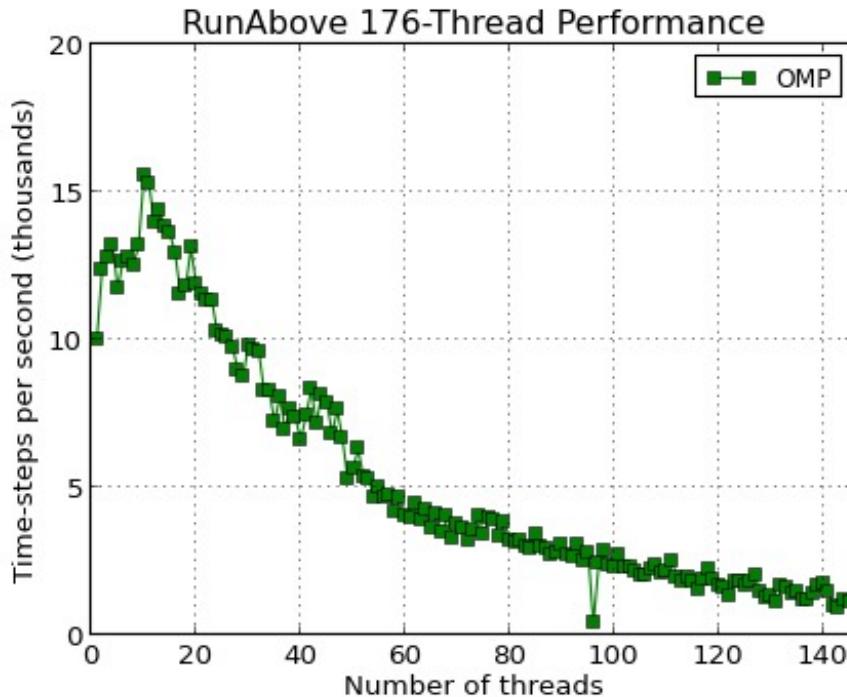
DigitalOcean20核虚拟机

在[DigitalOcean](#)20核虚拟机实例上运行上面相同的最快的实现。3个因素的最好的性能比在四核四线程的i5台式机上运行时的性能差。我不知道这个差异的原因是否是因为虚拟化和共享内核。



RunAbove176线程虚拟机

[RunAbove](#)对他们IBM Power 8架构免费一个月的使用进行推广。为了好玩，我在他们176线程的实例上运行了相同的实现。之前我从来没有使用过PowerPC架构，我很高兴代码运行不需要任何修改。



Cython

[Matthew Honnibal](#)提交了一个Cython的实现，可以在[这里](#)获得。这个实现和我们当初在[第一部分](#)中的C语言实现有些类似。

结论

当性能至关重要时，一个C语言扩展可以显著提升性能。使用SIMD指令可以使性能受益并简化代码，但有移植的成本。OpenMP的支持可以轻松地增加现有代码，并能提高在多核系统中的性能。如果您有任何疑问，意见，建议或更正，请通过联系链接告诉我。

1 赞 收藏 [评论](#)

关于作者： [Janzou](#)



做一直倦懒的小猫咪，享受午后阳光（新浪微博：[@Jan_zou](#)） [个人主页](#) · [我的文章](#) · [13](#)

高性能的Python扩展 (2)

本文由 [伯乐在线 - Janzou](#) 翻译, [黄利民](#) 校稿。未经许可, 禁止转载!

英文出处: [J. David Lee](#)。欢迎加入[翻译组](#)。

简介

这篇文章是这系列文章的第二篇, 我们的关注点在使用Numpy API为Python编写C扩展模块的过程。在第一部分中, 我们建立了一个简单的N体模拟, 并发现其瓶颈是计算体之间的相互作用力, 这是一个复杂度为 $O(N^2)$ 的操作。通过在C语言中实现一个时间演化函数, 我们大概能以大约70倍来加速计算。

[如果你还没有看过第一篇文章, 你应该在继续看这篇文章之前先看一下。](#)

在这篇文章中, 我们将牺牲我们代码中的一些通用性来提升性能。

回顾

World是存储N体状态的一个类。我们的模拟将演化一系列时间步长下的状态。

Python

```
class World(object):
    """World is a
1 class World(object):
2     """World is a structure that holds the state of N bodies and
3     additional variables.
4
5     threads : (int) The number of threads to use for multithreaded
6         implementations.
7     dt      : (float) The time-step.
8
9     STATE OF THE WORLD:
10
11    N : (int) The number of bodies in the simulation.
12    m : (1D ndarray) The mass of each body.
13    r : (2D ndarray) The position of each body.
14    v : (2D ndarray) The velocity of each body.
15    F : (2D ndarray) The force on each body.
16
17    TEMPORARY VARIABLES:
18
19    Ft : (3D ndarray) A 2D force array for each thread's local storage.
20    s : (2D ndarray) The vectors from one body to all others.
21    s3 : (1D ndarray) The norm of each s vector.
22
23    NOTE: Ft is used by parallel algorithms for thread-local
24        storage. s and s3 are only used by the Python
25        implementation.
26    """
27    def __init__(self, N, threads=1,
28                 m_min=1, m_max=30.0, r_max=50.0, v_max=4.0, dt=1e-3):
29        self.threads = threads
```

```

30     self.N = N
31     self.m = np.random.uniform(m_min, m_max, N)
32     self.r = np.random.uniform(-r_max, r_max, (N, 2))
33     self.v = np.random.uniform(-v_max, v_max, (N, 2))
34     self.F = np.zeros_like(self.r)
35     self.Ft = np.zeros((threads, N, 2))
36     self.s = np.zeros_like(self.r)
37     self.s3 = np.zeros_like(self.m)
38     self.dt = dt

```

在开始模拟时，N体被随机分配质量m，位置r和速度v。对于每个时间步长，接下来的计算有：

1. 合力F，每个体上的合力根据所有其他体的计算。
2. 速度v，由于力的作用每个体的速度被改变。
3. 位置R，由于速度每个体的位置被改变。

访问宏

我们在第一部分创建的扩展模块使用宏来获取C语言中NumPy数组的元素。下面是这些宏中的一些宏的形式：

Python

```

#define r(x0, x1) (*
(npy_float64*)
1 #define r(x0, x1) (*npy_float64*)((PyArray_DATA(py_r) +
2                         (x0) * PyArray_STRIDES(py_r)[0] + \
3                         (x1) * PyArray_STRIDES(py_r)[1]))

```

像这样使用宏，我们能使用像r(i, j)这样的简单标记来访问py_r数组中的元素。不管数组已经以某种形式被重塑或切片，索引值将匹配你在Python中看到的形式。

对于通用的代码，这就是我们想要的。在我们模拟的情况下，我们知道我们的数组的特点：它们是连续的，并且从未被切片或重塑。我们可以利用这一点来简化和提升我们代码的性能。

简单的C扩展 2

在本节中，我们将看到一个修改版本的C扩展，它摈弃了访问宏和NumPy数组底层数据的直接操作。本节中的代码src/simple2.c可在[github](#)上获得。

为了进行比较，之前的实现也可在[这里](#)获得。

演化函数

从文件的底部开始，我们可以看到，evolve函数与之前的版本一样，以相同的方式解析Python参数，但现在我们利用PyArray_DATA宏来获得一个纸箱底层的内存。我们将这个指针命名为npy_float64，作为double的一个别名。

Python

```

static PyObject *evolve(PyObject *self, PyObject *args) {
    // Declare variables.
    npy_int64 N, threads, steps, step, i, xi, yi;
    npy_float64 dt;
    PyArrayObject *py_m, *py_r, *py_v, *py_F;
    npy_float64 *m, *r, *v, *F;
    // Parse arguments.
    if (!PyArg_ParseTuple(args, "IdlOO!O!O!O!",
        &threads,
        &dt,
        &steps,
        &N,
        &PyArray_Type, &py_m,
        &PyArray_Type, &py_r,
        &PyArray_Type, &py_v,
        &PyArray_Type, &py_F)) {
        return NULL;
    }
    // Get underlying arrays from numpy arrays.
    m = (npy_float64*)PyArray_DATA(py_m);
    r = (npy_float64*)PyArray_DATA(py_r);
    v = (npy_float64*)PyArray_DATA(py_v);
    F = (npy_float64*)PyArray_DATA(py_F);
    // Evolve the world.
    for(step = 0; step < steps; ++step) {
        compute_F(N, m, r, F);
        for(i = 0; i < N; ++i) {
            // Compute offsets into arrays.
            xi = 2 * i;
            yi = xi + 1;
            v[xi] += F[xi] * dt / m[i];
            v[yi] += F[yi] * dt / m[i];
            r[xi] += v[xi] * dt;
            r[yi] += v[yi] * dt;
        }
    }
    Py_RETURN_NONE;
}

```

从我们以前使用宏的实现来看，唯一的变化是我们必须明确地计算数组索引。我们可以将底层数组描述为二维npy_float64矩阵，但这需要一定的前期成本和内存管理。

计算力

与之前的实现一样，力以相同的方式进行计算。唯一的不同在符号，以及在for-loops循环中显式索引的计算。

Python

```
static inline void
```

```
1 static inline void compute_F(npy_int64 N,
2                             npy_float64 *m,
3                             npy_float64 *r,
4                             npy_float64 *F) {
5     npy_int64 i, j, xi, yi, xj, yj;
6     npy_float64 sx, sy, Fx, Fy, s3, tmp;
7
8     // Set all forces to zero.
9     for(i = 0; i < N; ++i) {
10         F[2*i] = F[2*i + 1] = 0;
11     }
12
13    // Compute forces between pairs of bodies.
14    for(i = 0; i < N; ++i) {
15        xi = 2*i;
16        yi = xi + 1;
17        for(j = i + 1; j < N; ++j) {
18            xj = 2*j;
19            yj = xj + 1;
20
21            sx = r[xj] - r[xi];
22            sy = r[yj] - r[yi];
23
24            s3 = sqrt(sx*sx + sy*sy);
25            s3 *= s3*s3;
26
27            tmp = m[i] * m[j] / s3;
28            Fx = tmp * sx;
29            Fy = tmp * sy;
30
31            F[xi] += Fx;
32            F[yi] += Fy;
33            F[xj] -= Fx;
34            F[yj] -= Fy;
35        }
36    }
37 }
```

性能

我很惊讶地发现，相比于我们原来的C扩展，现在这种实现性能提升了45%。在相同的i5台式机上，以前的版本每秒执行17972个时间步长，而现在每秒执行26108个时间步长。这代表101倍提升了原始Python和NumPy的实现。

使用SIMD指令

在上面的实现中，我们在x和y方向上明确地计算出矢量分量。如果我们愿意放弃一些可移植性，并希望加快速度，我们可以使用x86的SIMD（单指令多数据）指令来简化我们的代码。

本节中的代码src/simd1.c，可在[github](#)上获得。

x86内部

在下面的代码中，我们将利用矢量数据类型 `_m128d`。数字128代表矢量中的字节数，而字母“d”表示矢量分量的数据类型。在这种情况下，分量的类型是double（64字节浮点）。其他矢量的宽度和类型可根据不同的架构获得。

多种内部函数都可以使用矢量数据类型。英特尔在其网站上提供了一个方便的参考。

可移植性

我的工作环境是使用GNU gcc编译器的Debian Wheezy系统。在这种环境下，定义可用的内部数据类型和函数的头文件是x86intrin.h。这可能不能跨平台移植。

演化函数

这里的`evolve`函数比之前的版本更加紧凑。二维数组转换为`_mm128d`指针，并利用矢量来排除显式计算矢量分量的需要。

Python

```
static PyObject
*evolve(PyObject *self,
1 static PyObject *evolve(PyObject *self, PyObject *args) {
2 // Variable declarations.
3     npy_int64 N, threads, steps, step, i;
4     npy_float64 dt;
5
6     PyArrayObject *py_m, *py_r, *py_v, *py_F;
7     npy_float64 *m;
8     __m128d *r, *v, *F;
9
10 // Parse arguments.
11 if (!PyArg_ParseTuple(args, "ldllO!O!O!O!",
12                     &threads,
13                     &dt,
14                     &steps,
15                     &N,
16                     &PyArray_Type, &py_m,
17                     &PyArray_Type, &py_r,
18                     &PyArray_Type, &py_v,
19                     &PyArray_Type, &py_F)) {
20     return NULL;
21 }
22
23 // Get underlying arrays from numpy arrays.
24 m = (npy_float64*)PyArray_DATA(py_m);
25 r = (__m128d*)PyArray_DATA(py_r);
26 v = (__m128d*)PyArray_DATA(py_v);
27 F = (__m128d*)PyArray_DATA(py_F);
28
29 // Evolve the world.
30 for(step = 0; step < steps; ++step) {
31     compute_F(N, m, r, F);
32
33     for(i = 0; i < N; ++i) {
34         v[i] += F[i] * dt / m[i];
35         r[i] += v[i] * dt;
36     }
37 }
```

```
38
39 Py_RETURN_NONE;
40 }
```

计算力

这里力的计算也比之前的实现更加紧凑。

Python

```
static inline void
1 static inline void compute_F(npy_int64 N,
2                             npy_float64 *m,
3                             __m128d *r,
4                             __m128d *F) {
5     npy_int64 i, j;
6     __m128d s, s2, tmp;
7     npy_float64 s3;
8
9     // Set all forces to zero.
10    for(i = 0; i < N; ++i) {
11        F[i] = _mm_set1_pd(0);
12    }
13
14    // Compute forces between pairs of bodies.
15    for(i = 0; i < N; ++i) {
16        for(j = i + 1; j < N; ++j) {
17            s = r[j] - r[i];
18
19            s2 = s * s;
20            s3 = sqrt(s2[0] + s2[1]);
21            s3 *= s3 * s3;
22
23            tmp = s * m[i] * m[j] / s3;
24            F[i] += tmp;
25            F[j] -= tmp;
26        }
27    }
28 }
```

性能

虽然这个明确的向量化代码更清晰，并比以前的版本更加紧凑，它的运行速度只提升了约1%，它每秒执行26427个时间步长。这可能是因为编译器能够使用相同的矢量指令优化之前的执行情况。

结论

如果全通用性是没有必要的，经常的性能提升，可以通过直接用C语言访问NumPy数组的底层内存来实现。

而在现在这个实例中，显式使用矢量内部没有提供多少好处，相对性能差异将在使用OpenMP的多芯设置中增大。

下一部分

在本系列文章的下一部分，我们将利用OpenMP来并行使用这里的实现，以及看如何利用多内核来扩展性能。

如果您有任何疑问，意见，建议或更正，请通过联系链接告诉我。

1 赞 收藏 [评论](#)

关于作者： [Janzou](#)



做一直倦懒的小猫咪，享受午后阳光（新浪微博：[@Jan_zou](#)） [个人主页](#) · [我的文章](#) · 13

高性能的Python扩展（1）

本文由[伯乐在线 - Janzou](#)翻译。未经许可，禁止转载！

英文出处：[J. David Lee](#)。欢迎加入[翻译组](#)。

简介

通常来说，Python不是一种高性能的语言，在某种意义上，这种说法是真的。但是，随着以[NumPy](#)为中心的数学和科学软件包的[生态圈](#)的发展，达到合理的性能不会太困难。

当性能成为问题时，运行时间通常由几个函数决定。用C重写这些函数，通常能极大的提升性能。

在本系列的第一部分中，我们来看看如何使用NumPy的C API来编写C语言的Python扩展，以改善模型的性能。在以后的文章中，我们将在这里提出我们的解决方案，以进一步提升其性能。

文件

这篇文章中所涉及的文件可以在[Github](#)上获得。

模拟

作为这个练习的起点，我们将在像重力的力的作用下为N体来考虑二维N体的模拟。

以下是将用于存储我们世界的状态，以及一些临时变量的类。

Python

```
# lib/sim.py
```

```
1 # lib/sim.py
2
3 class World(object):
4     """World is a structure that holds the state of N bodies and
5     additional variables.
6
7     threads : (int) The number of threads to use for multithreaded
8         implementations.
9
10    STATE OF THE WORLD:
11
12    N : (int) The number of bodies in the simulation.
13    m : (1D ndarray) The mass of each body.
14    r : (2D ndarray) The position of each body.
15    v : (2D ndarray) The velocity of each body.
16    F : (2D ndarray) The force on each body.
17
18    TEMPORARY VARIABLES:
19
20    Ft : (3D ndarray) A 2D force array for each thread's local storage.
21    s : (2D ndarray) The vectors from one body to all others.
22    s3 : (1D ndarray) The norm of each s vector.
```

```

23
24     NOTE: Ft is used by parallel algorithms for thread-local
25         storage. s and s3 are only used by the Python
26         implementation.
27     """
28     def __init__(self, N, threads=1,
29                  m_min=1, m_max=30.0, r_max=50.0, v_max=4.0, dt=1e-3):
30         self.threads = threads
31         self.N = N
32         self.m = np.random.uniform(m_min, m_max, N)
33         self.r = np.random.uniform(-r_max, r_max, (N, 2))
34         self.v = np.random.uniform(-v_max, v_max, (N, 2))
35         self.F = np.zeros_like(self.r)
36         self.Ft = np.zeros((threads, N, 2))
37         self.s = np.zeros_like(self.r)
38         self.s3 = np.zeros_like(self.m)
39         self.dt = dt

```

在开始模拟时，N体被随机分配质量m，位置r和速度v。对于每个时间步长，接下来的计算有：

1. 合力F，每个体上的合力根据所有其他体的计算。
2. 速度v，由于力的作用每个体的速度被改变。
3. 位置R，由于速度每个体的位置被改变。

第一步是计算合力F，这将是我们的瓶颈。由于世界上存在的其他物体，单一物体上的力是所有作用力的总和。这导致复杂度为O (N^2)。速度v和位置r更新的复杂度都是O (N)。

如果你有兴趣，这篇[维基百科的文章](#)介绍了一些可以加快力的计算的近似方法。

纯Python

在纯Python中，使用NumPy数组是时间演变函数的一种实现方式，它为优化提供了一个起点，并涉及测试其他实现方式。

Python

```

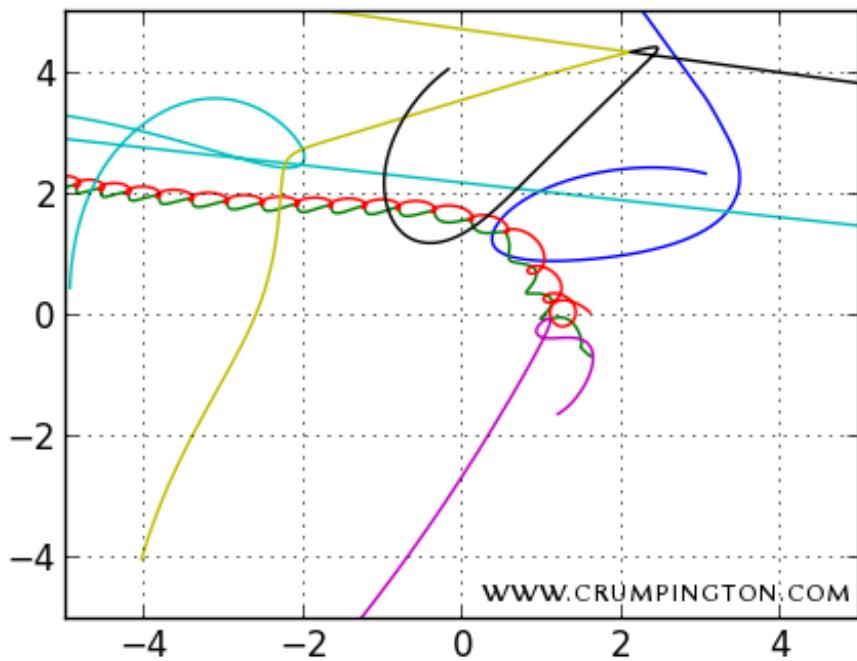
# lib/sim.py
1  # lib/sim.py
2
3  def compute_F(w):
4      """Compute the force on each body in the world, w."""
5      for i in xrange(w.N):
6          w.s[:] = w.r - w.r[i]
7          w.s3[:] = (w.s[:,0]**2 + w.s[:,1]**2)**1.5
8          w.s3[i] = 1.0 # This makes the self-force zero.
9          w.F[i] = (w.m[i] * w.m[:,None] * w.s / w.s3[:,None]).sum(0)
10
11 def evolve(w, steps):
12     """Evolve the world, w, through the given number of steps."""
13     for _ in xrange(steps):
14         compute_F(w)
15         w.v += w.F * w.dt / w.m[:,None]
16         w.r += w.v * w.dt

```

合力计算的复杂度为O (N^2) 的现象被NumPy的数组符号所掩盖。每个数组操作遍历数组元素。

可视化

这里是7个物体从随机初始状态开始演化的路径图：



性能

为了实现这个基准，我们在项目目录下创建了一个脚本，包含如下内容：

Python

```
import lib  
w = lib.World(101)  
  
1 import lib  
2 w = lib.World(101)  
3 lib.evolve(w, 4096)
```

我们使用cProfile模块来测试衡量这个脚本。

Python

```
python -m cProfile -  
scum bench.py  
  
1 python -m cProfile -scum bench.py
```

前几行告诉我们，compute_F确实是我们的瓶颈，它占了超过99%的运行时间。

Python

```
428710 function calls (428521 primitive calls)  
  
1 428710 function calls (428521 primitive calls) in 16.836 seconds  
2  
3 Ordered by: cumulative time
```

```
4
5      ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
6          1    0.000   0.000   16.837   16.837  bench.py:2(<module>)
7          1    0.062   0.062   16.756   16.756  sim.py:60(evolve)
8      4096   15.551   0.004   16.693   0.004  sim.py:51(compute_F)
9     413696   1.142   0.000   1.142   0.000 {method 'sum' ...
10         3    0.002   0.001   0.115   0.038 __init__.py:1(<module>)
11         ...
```

在Intel i5台式机上有101体，这种实现能够通过每秒257个时间步长演化世界。

简单的C扩展 1

在本节中，我们将看到一个C扩展模块实现演化的功能。当看完这一节时，这可能帮助我们获得一个C文件的副本。文件src/simple1.c，可以在[GitHub](#)上获得。

关于NumPy的C API的其他文档，请参阅[NumPy的参考](#)。Python的C API的详细文档在[这里](#)。

样板

文件中的第一件事情是先声明演化函数。这将直接用于下面的方法列表。

Python

```
static PyObject
*evolve(PyObject *self,
```

1 static PyObject *evolve(PyObject *self, PyObject *args);

接下来是方法列表。

Python

```
static PyMethodDef
methods[] = {
1 static PyMethodDef methods[] = {
2     { "evolve", evolve, METH_VARARGS, "Doc string."},
3     { NULL, NULL, 0, NULL } /* Sentinel */
4};
```

这是为扩展模块的一个导出方法列表。这只有一个名为evolve方法。

样板的最后一部分是模块的初始化。

Python

```
PyMODINIT_FUNC
initsimple1(void) {
1 PyMODINIT_FUNC initsimple1(void) {
2     (void) Py_Initialize("simple1", methods);
3     import_array();
4 }
```

另外，正如这里显示，`initsimple1`中的名称必须与`Py_InitModule`中的第一个参数匹配。对每个使用NumPy API的扩展而言，调用`import_array`是有必要的。

数组访问宏

数组访问的宏可以在数组中被用来正确地索引，无论数组被如何重塑或分片。这些宏也使用如下的代码使它们有更高的可读性。

Python

```
#define m(x0) (*  
(npy_float64*)  
1 #define m(x0) (*(npy_float64*)((PyArray_DATA(py_m) + \  
2 (x0) * PyArray_STRIDES(py_m)[0])))  
3 #define m_shape(i) (py_m->dimensions[(i)])  
4  
5 #define r(x0, x1) (*(npy_float64*)((PyArray_DATA(py_r) + \  
6 (x0) * PyArray_STRIDES(py_r)[0] + \  
7 (x1) * PyArray_STRIDES(py_r)[1])))  
8 #define r_shape(i) (py_r->dimensions[(i)])
```

在这里，我们看到访问宏的一维和二维数组。具有更高维度的数组可以以类似的方式被访问。

在这些宏的帮助下，我们可以使用下面的代码循环：

Python

```
for(i = 0; i < r_shape(0); ++i) {  
1 for(i = 0; i < r_shape(0); ++i) {  
2   for(j = 0; j < r_shape(1); ++j) {  
3     r(i, j) = 0; // Zero all elements.  
4   }  
5 }
```

命名标记

上面定义的宏，只在匹配NumPy的数组对象定义了正确的名称时才有效。在上面的代码中，数组被命名为`py_m`和`py_r`。为了在不同的方法中使用相同的宏，NumPy数组的名称需要保持一致。

计算力

特别是与上面五行的Python代码相比，计算力数组的方法显得颇为繁琐。

Python

```
static inline void  
compute_F(npy_int64  
1 static inline void compute_F(npy_int64 N,  
2                           PyArrayObject *py_m,  
3                           PyArrayObject *py_r,  
4                           PyArrayObject *py_F) {  
5   npy_int64 i, j;
```

```

6     npy_float64 sx, sy, Fx, Fy, s3, tmp;
7
8     // Set all forces to zero.
9     for(i = 0; i < N; ++i) {
10        F(i, 0) = F(i, 1) = 0;
11    }
12
13    // Compute forces between pairs of bodies.
14    for(i = 0; i < N; ++i) {
15        for(j = i + 1; j < N; ++j) {
16            sx = r(j, 0) - r(i, 0);
17            sy = r(j, 1) - r(i, 1);
18
19            s3 = sqrt(sx*sx + sy*sy);
20            s3 *= s3 * s3;
21
22            tmp = m(i) * m(j) / s3;
23            Fx = tmp * sx;
24            Fy = tmp * sy;
25
26            F(i, 0) += Fx;
27            F(i, 1) += Fy;
28
29            F(j, 0) -= Fx;
30            F(j, 1) -= Fy;
31        }
32    }
33}

```

请注意，我们使用牛顿第三定律（成对出现的力大小相等且方向相反）来降低内环范围。不幸的是，它的复杂度仍然为 $O(N^2)$ 。

演化函数

该文件中的最后一个函数是导出的演化方法。

Python



```

static PyObject *
1 static PyObject *evolve(PyObject *self, PyObject *args) {
2
3     // Declare variables.
4     npy_int64 N, threads, steps, step, i;
5     npy_float64 dt;
6     PyArrayObject *py_m, *py_r, *py_v, *py_F;
7
8     // Parse arguments.
9     if (!PyArg_ParseTuple(args, "ldllO!O!O!O!",
10                           &threads,
11                           &dt,
12                           &steps,
13                           &N,
14                           &PyArray_Type, &py_m,
15                           &PyArray_Type, &py_r,
16                           &PyArray_Type, &py_v,
17                           &PyArray_Type, &py_F)) {
18         return NULL;
19     }
20

```

```
21 // Evolve the world.
22 for(step = 0; step < steps; ++step) {
23     compute_F(N, py_m, py_r, py_F);
24
25     for(i = 0; i < N; ++i) {
26         v(i, 0) += F(i, 0) * dt / m(i);
27         v(i, 1) += F(i, 1) * dt / m(i);
28
29         r(i, 0) += v(i, 0) * dt;
30         r(i, 1) += v(i, 1) * dt;
31     }
32 }
33
34 Py_RETURN_NONE;
35 }
```

在这里，我们看到了Python参数如何被解析。在该函数底部的时间步长循环中，我们看到的速度和位置向量的x和y分量的显式计算。

性能

C版本的演化方法比Python版本更快，这应该不足为奇。在上面提到的相同的i5台式机中，C实现的演化方法能够实现每秒17972个时间步长。相比Python实现，这方面有70倍的提升。

观察

注意，C代码一直保持尽可能的简单。输入参数和输出矩阵可以进行类型检查，并分配一个Python装饰器函数。删除分配，不仅能加快处理，而且消除了由Python对象不正确的引用计数造成的内存泄露（或更糟）。

下一部分

在本系列文章的下一部分，我们将通过发挥C-相邻NumPy矩阵的优势来提升这种实现的性能。之后，我们来看看使用英特尔的SIMD指令和OpenMP来进一步推进。

如果您有任何疑问，意见，建议或更正，请通过联系链接告诉我。

1 赞 收藏 [1 评论](#)

关于作者：[Janzou](#)



做一直倦懒的小猫咪，享受午后阳光（新浪微博：@Jan_zou） [个人主页](#) · [我的文章](#) · 13

零基础自学用Python 3开发网络爬虫(四): 登录

原文出处: [Jecvay Notes \(@Jecvay\)](#)

- 《零基础自学用Python 3开发网络爬虫(一)》
- 《零基础自学用Python 3开发网络爬虫(二)》
- 《零基础自学用Python 3开发网络爬虫(三)》

今天的工作很有意思, 我们用 Python 来登录网站, 用Cookies记录登录信息, 然后就可以抓取登录之后才能看到的信息. 今天我们拿知乎网来做示范. 为什么是知乎? 这个很难解释, 但是肯定的是知乎这么大这么成功的网站完全不用我来帮他打广告. 知乎网的登录比较简单, 传输的时候没有对用户名和密码加密, 却又不失代表性, 有一个必须从主页跳转登录的过程.

不得不说一下, Fiddler 这个软件是 Tpircsboy 告诉我的. 感谢他给我带来这么好玩的东西.

第一步: 使用 Fiddler 观察浏览器行为

在开着 Fiddler 的条件下运行浏览器, 输入知乎网的网址 <http://www.zhihu.com> 回车后到 Fiddler 中就能看到捕捉到的连接信息. 在左边选中一条 200 连接, 在右边打开 Inspectors 透视图, 上方是该条连接的请求报文信息, 下方是响应报文信息.

其中 Raw 标签是显示报文的原文. 下方的响应报文很有可能是没有经过解压或者解码的, 这种情况他会在中间部位有一个小提示, 点击一下就能解码显示出原文了.

The screenshot shows the Fiddler Web Debugger interface. A single connection is selected in the left pane, showing a 200 HTTP response from www.zhihu.com. The right pane displays the request and response details. The request pane shows the raw GET request with various headers and a cookie. The response pane shows the raw HTML content of the page, which is the homepage of Zhihu. The HTML includes standard meta tags, a title, and links to static assets.

以上这个截图是在未登录的时候进入 <http://www.zhihu.com> 得到的. 现在我们来输入用户名和密码登陆知乎网, 再看看浏览器和知乎服务器之间发生了什么.



点击登陆后,回到 Fiddler 里查看新出现的一个 200 链接. 我们浏览器携带者我的帐号密码给知乎服务器发送了一个 POST, 内容如下:

```
POST http://www.zhihu.com/login HTTP/1.1
1 POST http://www.zhihu.com/login HTTP/1.1
2 Content-Type: application/x-www-form-urlencoded; charset=UTF-8
3 Accept: */*
4 X-Requested-With: XMLHttpRequest
5 Referer: http://www.zhihu.com/#signin
6 Accept-Language: en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3
7 Accept-Encoding: gzip, deflate
8 User-Agent: Mozilla/5.0 (Windows NT 6.4; WOW64; Trident/7.0; rv:11.0) like Gecko
9 Content-Length: 97
10 DNT: 1
11 Host: www.zhihu.com
12 Connection: Keep-Alive
13 Pragma: no-cache
14 Cookie: __utma=51854390.1539896551.1412320246.1412320246.1412320246.1; __utmb=51854390.6.10.1412320246; __utmc=51854390;
15 __utmz=51854390.1412320246.1.1.utmcsr=(direct)|utmccn=(direct)|utmcmd=(none); __utmv=51854390.000|3=entry_date=20141003=1
16 _xsrf=4b41f6c7a9668187ccd8a610065b9718&email=此处涂黑%40gmail.com&password=此处不可见&rememberme=y
```

截图如下:

The screenshot shows the Fiddler Web Debugger interface. On the left, a list of network requests is displayed. Request #35 is highlighted, showing a POST request to `http://www.zhihu.com/login`. On the right, the 'Inspectors' tab bar has 'WebForms' selected. The 'QueryString' and 'Body' sections are visible, with the 'Body' section containing form fields like 'rememberme' set to 'y'. Below the inspectors, the server response is shown with various headers, including several 'Set-Cookie' entries and a 'Content-Security-Policy' header.

我的浏览器给 `http://www.zhihu.com/login` 这个网址(多了一个`/login`)发送了一个POST, 内容都已经在上面列出来了, 有用户名, 有密码, 有一个"记住我"的 yes, 其中这个 WebForms 标签下 Fiddler 能够比较井井有条的列出来 POST 的内容. 所以我们用 Python 也发送相同的内容就能登录了. 但是这里出现了一个 Name 为 `_xsrf` 的项, 他的值是 `4b41f6c7a9668187ccda8a610065b9718`. 我们要先获取这个值, 然后才能给他发.

浏览器是如何获取的呢, 我们刚刚是先访问了 `http://www.zhihu.com/` 这个网址, 就是首页, 然后登录的时候他却给 `http://www.zhihu.com/login` 这个网址发信息. 所以用侦探一般的思维去思考这个问题, 就会发现肯定是首页把 `_xsrf` 生成发送给我们, 然后我们再把这个 `_xsrf` 发送给 `/login` 这个 url. 这样一会儿过后我们就要从第一个 GET 得到的响应报文里面去寻找 `_xsrf`

截图下方的方框说明, 我们不仅登录成功了, 而且服务器还告诉我们的浏览器如何保存它给出的 Cookies 信息. 所以我们也要用 Python 把这些 Cookies 信息记录下来.

这样 Fiddler 的工作就基本结束了!

第二步: 解压缩

简单的写一个 GET 程序, 把知乎首页 GET 下来, 然后 decode() 一下解码, 结果报错. 仔细一看, 发现知乎网传给我们的是经过 gzip 压缩之后的数据. 这样我们就需要先对数据解压. Python 进行 gzip 解压很方便, 因为内置有库可以用. 代码片段如下:

Python

```
import gzip
def ungzip(data):
    import gzip
    def ungzip(data):
        try: # 尝试解压
            print('正在解压....')
            data = gzip.decompress(data)
            print('解压完毕!')
        except:
            print('未经压缩, 无需解压')
        return data
```

通过 `opener.read()` 读取回来的数据, 经过 `ungzip` 自动处理后, 再来一遍 `decode()` 就可以得到解码后的 str 了

第二步: 使用正则表达式获取沙漠之舟

_xsrf 这个键的值在茫茫无际的互联网沙漠之中指引我们用正确的姿势来登录知乎, 所以 _xsrf 可谓沙漠之舟. 如果没有 _xsrf, 我们或许有用户名和密码也无法登录知乎(我没试过, 不过我们学校的教务系统确实如此) 如上文所说, 我们在第一遍 GET 的时候可以从响应报文中的 HTML 代码里面得到这个沙漠之舟. 如下函数实现了这个功能, 返回的 str 就是 _xsrf 的值.

Python

```
import re
def getXSRF(data):
    1 import re
    2 def getXSRF(data):
    3     cer = re.compile('name=\"_xsrf\" value=(\".*\")', flags = 0)
    4     strlist = cer.findall(data)
    5     return strlist[0]
```

第三步: 发射 POST !!

集齐 _xsrf, id, password 三大法宝, 我们可以发射 POST 了. 这个 POST 一旦发射过去, 我们就登陆上了服务器, 服务器就会发给我们 Cookies. 本来处理 Cookies 是个麻烦的事情, 不过 Python 的 http.cookiejar 库给了我们很方便的解决方案, 只要在创建 opener 的时候将一个 HTTPCookieProcessor 放进去, Cookies 的事情就不用我们管了. 下面的代码体现了这一点.

Python

```
import http.cookiejar
import urllib.request
1 import http.cookiejar
2 import urllib.request
3 def getOpener(head):
4     # deal with the Cookies
5     cj = http.cookiejar.CookieJar()
6     pro = urllib.request.HTTPCookieProcessor(cj)
7     opener = urllib.request.build_opener(pro)
8     header = []
9     for key, value in head.items():
10        elem = (key, value)
11        header.append(elem)
12     opener.addheaders = header
13     return opener
```

getOpener 函数接收一个 head 参数, 这个参数是一个字典. 函数把字典转换成元组集合, 放进 opener. 这样我们建立的这个 opener 就有两大功能:

自动处理使用 opener 过程中遇到的 Cookies

自动在发出的 GET 或者 POST 请求中加上自定义的 Header

第四部: 正式运行

正式运行还差一点点, 我们要把要 POST 的数据弄成 opener.open() 支持的格式. 所以还要 urllib.parse 库里的 urlencode() 函数. 这个函数可以把字典或者元组集合类型的数据转换成 & 连接的 str.

str 还不行, 还要通过 encode() 来编码, 才能当作 opener.open() 或者 urlopen() 的 POST 数据参数来使用. 代码如下:

Python

```
url =
"http://www.zhihu.com/"
1 url = 'http://www.zhihu.com/'
2 opener = getOpener(header)
3 op = opener.open(url)
4 data = op.read()
5 data = ungzip(data)  # 解压
6 _xsrf = getXSRF(data.decode())
7
8 url += 'login'
9 id = '这里填你的知乎帐号'
10 password = '这里填你的知乎密码'
11 postDict = {
12     '_xsrf': _xsrf,
13     'email': id,
14     'password': password,
15     'rememberme': 'y'
16 }
17 postData = urllib.parse.urlencode(postDict).encode()
18 op = opener.open(url, postData)
19 data = op.read()
20 data = ungzip(data)
21
22 print(data.decode()) # 你可以根据你喜欢来处理抓取回来的数据了!
```

代码运行后, 我们发现自己关注的人的动态(显示在登陆后的知乎首页的那些), 都被抓取回来了. 下一步做一个统计分析器, 或者自动推送器, 或者内

容分级自动分类器, 都可以.

Python

```
import gzip
import re

1 import gzip
2 import re
3 import http.cookiejar
4 import urllib.request
5 import urllib.parse
6
7 def ungzip(data):
8     try:      # 尝试解压
9         print('正在解压....')
10        data = gzip.decompress(data)
11        print('解压完毕!')
12    except:
13        print('未经压缩, 无需解压')
14    return data
15
16 def getXSRF(data):
17     cer = re.compile('name=__xsrf" value="(.*")', flags = 0)
18     strlist = cer.findall(data)
19     return strlist[0]
20
21 def getOpener(head):
22     # deal with the Cookies
23     cj = http.cookiejar.CookieJar()
24     pro = urllib.request.HTTPCookieProcessor(cj)
25     opener = urllib.request.build_opener(pro)
26     header = []
27     for key, value in head.items():
28         elem = (key, value)
29         header.append(elem)
30     opener.addheaders = header
31     return opener
32
33 header = {
34     'Connection': 'Keep-Alive',
35     'Accept': 'text/html, application/xhtml+xml, */*',
36     'Accept-Language': 'en-US,en;q=0.8,zh-Hans-CN;q=0.5,zh-Hans;q=0.3',
37     'User-Agent': 'Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko',
38     'Accept-Encoding': 'gzip, deflate',
39     'Host': 'www.zhihu.com',
40     'DNT': '1'
41 }
42
43 url = 'http://www.zhihu.com/'
44 opener = getOpener(header)
45 op = opener.open(url)
46 data = op.read()
47 data = ungzip(data)  # 解压
48 _xsrft = getXSRF(data.decode())
49
50 url += 'login'
51 id = '这里填你的知乎帐号'
52 password = '这里填你的知乎密码'
53 postDict = {
54     '_xsrf': _xsrft,
55     'email': id,
56     'password': password,
57     'rememberme': 'y'
58 }
59 postData = urllib.parse.urlencode(postDict).encode()
60 op = opener.open(url, postData)
61 data = op.read()
62 data = ungzip(data)
63
64 print(data.decode())
```

2 赞 6 收藏 [1评论](#)