urllib2

官网

URI介绍

Web上每种可用的资源，如HTML文档、图像、视频片段、程序等都由一个通用资源标志符(Universal Resource Identifier，URI)进行定位。

URI通常由三部分组成：

1. 访问资源的命名机制；
2. 存放资源的主机名；
3. 资源自身的名称，由路径表示。

如下面的URI： `http://www.why.com.cn/myhtml/html1223/`

我们可以这样解释它： 这是一个可以通过HTTP协议访问的资源，位于主机www.webmonkey.com.cn上,通过路径/html/html1223访问。

URL介绍

URL是URI的一个子集。它是Uniform Resource Locator的缩写，译为统一资源定位符。 通俗地说，URL是Internet上描述信息资源的字符串，主要用在各种WWW客户程序和服务器程序上。采用URL可以用一种统一的格式来描述各种信息资源，包括文件、服务器的地址和目录等。 URL的一般格式为(带方括号[]的为可选项)：

```
protocol://hostname[:port]/path/[;parameters][?query]#fragment
```

URL的格式由三部分组成：

- 第一部分是协议(或称为服务方式)。
- 第二部分是存有该资源的主机IP地址(有时也包括端口号)。
- 第三部分是主机资源的具体地址，如目录和文件名等。

第一部分和第二部分用://符号隔开。 第二部分和第三部分用/符号隔开。 第一部分和第二部分是不可缺少的，第三部分有时可以省略。

python爬虫

在Python中，我们使用urllib2这个组件来抓取网页。urllib2是Python的一个获取URLs(Uniform Resource Locators)的组件。它以urlopen函数的形式提供了一个非常简单的接口。

urllib2用一个Request对象来映射你提出的HTTP请求。在它最简单的使用形式中你将用你要请求的地址创建一个Request对象，通过调用urlopen并传入Request对象，将返回一个相关请求response对象，这个应答对象如同一个文件对象，所以你可以在Response中调用.read()。

```
import urllib2
req = urllib2.Request('http://www.baidu.com')
response = urllib2.urlopen(req)
the_page = response.read()
print the_page
```

发送data表单数据

有时候你希望发送一些数据到URL(通常URL与CGI[通用网关接口]脚本，或其他WEB应用程序挂接)。在HTTP中,这个经常使用熟知的POST请求发送。这个通常在你提交一个HTML表单时由你的浏览器来做。并不是所有的POSTs都来源于表单，你能够使用POST提交任意的数据到你自己的程序。一般的HTML表单，data需要编码成标准形式。然后做为data参数传到Request对象。编码工作使用urllib的函数而非urllib2。

```
import urllib
import urllib2

url = 'http://www.someserver.com/register.cgi'

values = {'name' : 'WHY',
          'location' : 'SDU',
          'language' : 'Python' }

data = urllib.urlencode(values) # 编码工作
req = urllib2.Request(url, data)  # 发送请求同时传data表单
response = urllib2.urlopen(req)  #接受反馈的信息
the_page = response.read()  #读取反馈的内容
```

如果没有传送data参数，urllib2使用GET方式的请求。 GET方法和POST方法的区别：详见w3school

设置Headers到http请求

某些网站反感爬虫的到访，于是对爬虫一律拒绝请求。这时候我们需要伪装成浏览器，这可以通过修改http包中的header来实现：

```
headers = {
    'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3
}
req = urllib2.Request(
    url = 'http://secure.verycd.com/signin/*/http://www.verycd.com/',
    data = postdata,
    headers = headers
)
```

反“反盗链”/修改Header

某些站点有所谓的反盗链设置，其实说穿了很简单，就是检查你发送请求的header里面，referer站点是不是他

自己。所以我们只需要把`headers`的`referer`改成该网站即可.

```
headers = {
'Referer':'http://www.cnbeta.com/articles'
}
```

`headers`是一个`dict`数据结构，你可以放入任何想要的`header`，来做一些伪装。

`Proxy`（代理服务器）的设置

`urllib2`默认会使用环境变量`http_proxy`来设置`HTTP Proxy`。如果想在程序中明确控制 `Proxy`而不受环境变量的影响，可以使用代理。

这里要注意的一个细节，使用`urllib2.install_opener()`会设置`urllib2`的全局 `opener`。这样后面的使用会很方便，但不能做更细致的控制，比如想在程序中使用两个不同的 `Proxy`设置等。比较好的做法是不使用`install_opener`去更改全局的设置，而只是直接调用 `opener`的`open`方法代替全局的`urlopen`方法。

```
import urllib2
enable_proxy = True
proxy_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})
null_proxy_handler = urllib2.ProxyHandler({})
if enable_proxy:
    opener = urllib2.build_opener(proxy_handler)
else:
    opener = urllib2.build_opener(null_proxy_handler)
urllib2.install_opener(opener)
```

`Timeout`设置

超时可以通过`urllib2.urlopen()`的`timeout`参数直接设置。

```
import urllib2
response = urllib2.urlopen('http://www.google.com', timeout=10)
```

`geturl`方法

这个返回获取的真实的`URL`，这个很有用，因为`urlopen`(或者`opener`对象使用的)或许会有重定向。获取的`URL`或许跟请求`URL`不同。

```
import urllib2

old_url = 'http://rrurl.cn/b1UZuP'
req = urllib2.Request(old_url)
response = urllib2.urlopen(req)
print 'Old url :' + old_url
print 'Real url :' + response.geturl()
```

输出的结果为

```
Old url :http://rrurl.cn/b1UZuP
Real url :http://www.polyu.edu.hk/polyuchallenge/best_of_the_best_elevator_pitch_award/bbca_voting_pro
```

info方法

这个返回对象的字典对象，该字典描述了获取的页面情况。通常是服务器发送的特定头headers。目前是httplib.HTTPMessage实例。经典的headers包含Content-length，Content-type和其他内容。

```
from urllib2 import *

url = 'http://www.baidu.com'
req = Request(url)
response = urlopen(req)
print 'Info():'
print response.info()
```

Redirect

urllib2默认情况下会针对HTTP 3XX返回码自动进行redirect动作，无需人工配置。要检测是否发生了redirect动作，只要检查一下Response的URL和Request 的URL是否一致就可以了。

```
import urllib2
my_url = 'http://www.google.cn'
response = urllib2.urlopen(my_url)
redirected = response.geturl() == my_url
print redirected

my_url = 'http://rrurl.cn/b1UZuP'
response = urllib2.urlopen(my_url)
redirected = response.geturl() == my_url
print redirected
```

如果不想自动redirect，除了使用更低层次的httplib库之外，还可以自定义HTTPRedirectHandler类。

Cookie

urllib2对Cookie的处理也是自动的。如果需要得到某个Cookie项的值，可以这么做： import urllib2
import cookielib
cookie = cookielib.CookieJar()
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
response = opener.open('http://www.baidu.com')
for item in cookie: print 'Name = '+item.name print 'Value = '+item.value

使用HTTP的PUT和DELETE方法

urllib2只支持HTTP的GET和POST方法，如果要使用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：

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

得到HTTP的返回码

对于200 OK来说，只要使用urlopen返回的response对象的getcode()方法就可以得到HTTP的返回码。但对其它返回码来说，urlopen会抛出异常。这时候，就要检查异常对象的code属性了：

```
import urllib2
try:
    response = urllib2.urlopen('http://bbs.csdn.net/why')
except urllib2.HTTPError, e:
    print e.code
```

Debug Log

使用urllib2时，可以通过下面的方法把debug Log打开，这样收发包的内容就会在屏幕上打印出来，方便调试，有时可以省去抓包的工作。

```
import urllib2
httpHandler = urllib2.HTTPHandler(debuglevel=1)
httpsHandler = urllib2.HTTPSHandler(debuglevel=1)
opener = urllib2.build_opener(httpHandler, httpsHandler)
urllib2.install_opener(opener)
response = urllib2.urlopen('http://www.baidu.com')
```

异常的处理

先来说一说HTTP的异常处理问题。当urlopen不能够处理一个response时，产生URLError。不过通常的Python APIs异常如ValueError,TypeError等也会同时产生。 HTTPError是URLError的子类，通常在特定HTTP URLs中产生。

URLError

通常，URLError在没有网络连接(没有路由到特定服务器)，或者服务器不存在的情况下产生。 这种情况下，异常同样会带有reason属性，它是一个tuple（可以理解为不可变的数组）， 包含了一个错误号和一个错误信息。

```
import urllib2

req = urllib2.Request('http://www.baibai.com')
try: urllib2.urlopen(req)
```

```
except urllib2.URLError,e:
    print e.reason
```

运行后可以看到打印出来的内容是：`[Errno 11001] getaddrinfo failed`。也就是说，错误号是`11001`，内容是`getaddrinfo failed`。

HTTPError

服务器上每一个`HTTP`应答对象`response`包含一个数字`状态码`。有时状态码指出服务器无法完成请求。默认的处理器会为你处理一部分这种应答。例如:假如`response`是一个"重定向"，需要客户端从别的地址获取文档，urllib2将为你处理。其他不能处理的，`urlopen`会产生一个`HTTPError`。

典型的错误包含`404`(页面无法找到)，`403`(请求禁止)，和`401`(带验证请求)。 HTTP状态码表示HTTP协议所返回的响应的状态。比如客户端向服务器发送请求，如果成功地获得请求的资源，则返回的状态码为`200`，表示响应成功。如果请求的资源不存在， 则通常返回`404`错误。 HTTP状态码通常分为5种类型，分别以1～5五个数字开头，由3位整数组成。（可从官网上看）

Error Codes错误码

因为默认的处理器处理了重定向(300以外号码)，并且100-299范围的号码指示成功，所以你只能看到400-599的错误号码。`BaseHTTPServer.BaseHTTPRequestHandler.response`是一个很有用的应答号码字典，显示了`HTTP`协议使用的所有的应答号。

当一个错误号产生后，服务器返回一个HTTP错误号，和一个错误页面。你可以使用`HTTPError`实例作为页面返回的应答对象`response`。这表示和错误属性一样，它同样包含了`read`,`geturl`,和`info`方法。

```
import urllib2
req = urllib2.Request('http://bbs.csdn.net/callmewhy')

try:
    urllib2.urlopen(req)
except urllib2.URLError, e:
    print e.code
```

运行后可以看见输出了`404`的错误码，也就说没有找到这个页面。

`Openers`与`Handles`

当你获取一个`URL`你使用一个`opener`(一个`urllib2.OpenerDirector`的实例)。正常情况下，我们使用默认`opener`:通过`urlopen`。但你能够创建个性的`openers`。

`Openers`使用处理器`handlerS`，所有的“繁重”工作由`handlers`处理。每个`handlers`知道如何通过特定协议打开URLs，或者如何处理URL打开时的各个方面。例如HTTP重定向或者HTTP cookies。

如果你希望用特定处理器获取URLs你会想创建一个`openers`，例如获取一个能处理`cookie`的`opener`，或者获取一个不重定向的`opener`。

要创建一个opener，可以实例化一个OpenerDirector，然后调用.add_handler(some_handler_instance)。同样，可以使用build_opener，这是一个更加方便的函数，用来创建opener对象，他只需要一次函数调用。build_opener默认添加几个处理器，但提供快捷的方法来添加或更新默认处理器。其他的处理器handlers你或许会希望处理代理，验证，和其他常用但有点特殊的情况。

install_opener用来创建（全局）默认opener。这个表示调用urlopen将使用你安装的opener。 Opener对象有一个open方法。该方法可以像urlopen函数那样直接用来获取urls：通常不必调用install_opener，除了为了方便。

说完了上面两个内容，下面我们来看一下基本认证的内容，这里会用到上面提及的Opener和Handler。

Basic Authentication 基本验证

为了展示创建和安装一个handler，我们将使用HTTPBasicAuthHandler。 当需要基础验证时，服务器发送一个header(401错误码) 请求验证。这个指定了scheme和一个realm，看起来像这样：Www-authenticate: SCHEME realm="REALM"。例如Www-authenticate: Basic realm="cPanel Users"客户端必须使用新的请求，并在请求头里包含正确的姓名和密码。这是“基础验证”，为了简化这个过程，我们可以创建一个HTTPBasicAuthHandler的实例，并让opener使用这个handler就可以啦。

HTTPBasicAuthHandler使用一个密码管理的对象来处理URLs和realms来映射用户名和密码。 如果你知道realm(从服务器发送来的头里)是什么，你就能使用HTTPPasswordMgr。

通常人们不关心realm是什么。那样的话，就能用方便的HTTPPasswordMgrWithDefaultRealm。 这个将在你为URL指定一个默认的用户名和密码。 这将在你为特定realm提供一个其他组合时得到提供。 我们通过给realm参数指定None提供给add_password来指示这种情况。

最高层次的URL是第一个要求验证的URL。你传给.add_password()更深层次的URLs将同样合适。

正则表达式

re模块

Python通过re模块提供对正则表达式的支持。 使用re的一般步骤是：

1. 先将正则表达式的字符串形式编译为Pattern实例。
2. 然后使用Pattern实例处理文本并获得匹配结果（一个Match实例）。
3. 最后使用Match实例获得信息，进行其他的操作。

Compile

re.compile(strPattern[, flag]): 这个方法是Pattern类的工厂方法，用于将字符串形式的正则表达式编译为Pattern对象。 第二个参数flag是匹配模式，取值可以使用按位或运算符|表示同时生效，比如re.I | re.M。另外，你也可以在regex字符串中指定模式，比如re.compile('pattern', re.I | re.M)与re.compile('(?im)pattern')是等价的。

可选值有：

- `re.I`(全拼：IGNORECASE): 忽略大小写（括号内是完整写法，下同）
- `re.M`(全拼：MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
- `re.S`(全拼：DOTALL): 点任意匹配模式，改变'.'的行为
- `re.L`(全拼：LOCALE): 使预定字符类 \w \W \b \B \s \S 取决于当前区域设定
- `re.U`(全拼：UNICODE): 使预定字符类 \w \W \b \B \s \S \d \D 取决于`unicode`定义的字符属性
- `re.X`(全拼：VERBOSE): 详细模式。这个模式下正则表达式可以是多行，忽略空白字符，并可以加入注释。

`Match`对象

`Match`对象是一次匹配的结果，包含了很多关于此次匹配的信息，可以使用`Match`提供的可读属性或方法来获取这些信息。

属性：

- string: 匹配时使用的文本。
- re: 匹配时使用的Pattern对象。
- pos: 文本中正则表达式开始搜索的索引。值与`Pattern.match()`和`Pattern.seach()`方法的同名参数相同。
- endpos: 文本中正则表达式结束搜索的索引。值与`Pattern.match()`和`Pattern.seach()`方法的同名参数相同。
- lastindex: 最后一个被捕获的分组在文本中的索引。如果没有被捕获的分组，将为`None`。
- lastgroup: 最后一个被捕获的分组的别名。如果这个分组没有别名或者没有被捕获的分组，将为`None`。

方法：

- `group([group1, …])`： 获得一个或多个分组截获的字符串；指定多个参数时将以元组形式返回。`group1`可以使用编号也可以使用别名；编号0代表整个匹配的子串；不填写参数时，返回`group(0)`；没有截获字符串的组返回`None`；截获了多次的组返回最后一次截获的子串。
- `groups([default])`： 以元组形式返回全部分组截获的字符串。相当于调用`group(1,2,…last)`。`default`表示没有截获字符串的组以这个值替代，默认为`None`。
- `groupdict([default])`： 返回以有别名的组的别名为键、以该组截获的子串为值的字典，没有别名的组不包含在内。`default`含义同上。
- `start([group])`： 返回指定的组截获的子串在`string`中的起始索引（子串第一个字符的索引）。`group`默认值为`0`。

- `end([group])`： 返回指定的组截获的子串在`string`中的结束索引（子串最后一个字符的索引+1）。`group`默认值为`0`。
- `span([group])`： 返回`(start(group), end(group))`。
- `expand(template)`： 将匹配到的分组代入`template`中然后返回。`template`中可以使用`\id`或`\g<id>`、`\g<name>`引用分组，但不能使用编号0。`\id`与`\g<id>`是等价的；但`\10`将被认为是第10个分组，如果你想表达`\1`之后是字符`0`，只能使用`\g<1>0`。

例：

```
import re
# 匹配如下内容：单词+空格+单词+任意字符
m = re.match(r'(\w+) (\w+)(?P<sign>.*)', 'hello world!')

print "m.string:", m.string
print "m.re:", m.re
print "m.pos:", m.pos
print "m.endpos:", m.endpos
print "m.lastindex:", m.lastindex
print "m.lastgroup:", m.lastgroup

print "m.group():", m.group()
print "m.group(1,2):", m.group(1, 2)
print "m.groups():", m.groups()
print "m.groupdict():", m.groupdict()
print "m.start(2):", m.start(2)
print "m.end(2):", m.end(2)
print "m.span(2):", m.span(2)
print r"m.expand(r'\g<2> \g<1>\g<3>'):", m.expand(r'\2 \1\3')

### output ###
# m.string: hello world!
# m.re: <_sre.SRE_Pattern object at 0x016E1A38>
# m.pos: 0
# m.endpos: 12
# m.lastindex: 3
# m.lastgroup: sign
# m.group(1,2): ('hello', 'world')
# m.groups(): ('hello', 'world', '!')
# m.groupdict(): {'sign': '!'}
# m.start(2): 6
# m.end(2): 11
# m.span(2): (6, 11)
# m.expand(r'\2 \1\3'): world hello!
```

`Pattern`对象

属性：

`Pattern`对象是一个编译好的正则表达式，通过`Pattern`提供的一系列方法可以对文本进行匹配查找。 `Pattern`不能直接实例化，必须使用`re.compile()`进行构造，也就是`re.compile()`返回的对象。 Pattern提供了几个可读属性用于获取表达式的相关信息：

- `pattern`: 编译时用的表达式字符串。
- `flags`: 编译时用的匹配模式。数字形式。
- `groups`: 表达式中分组的数量。
- `groupindex`: 以表达式中有别名的组的别名为键、以该组对应的编号为值的字典，没有别名的组不包含在内。

方法：

- match：

  `match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])`：这个方法将从`string`的`pos`下标处起尝试匹配`pattern`；如果`pattern`结束时仍可匹配，则返回一个`Match`对象；如果匹配过程中`pattern`无法匹配，或者匹配未结束就已到达`endpos`，则返回`None`。`pos`和`endpos`的默认值分别为`0`和`len(string)`；`re.match()`无法指定这两个参数，参数`flags`用于编译`pattern`时指定匹配模式。

  注意：这个方法并不是完全匹配。当`pattern`结束时若`string`还有剩余字符，仍然视为成功。想要完全匹配，可以在表达式末尾加上边界匹配符`$`。下面来看一个Match的简单案例：

  ```
  import re

  # 将正则表达式编译成Pattern对象
  pattern = re.compile(r'hello')

  # 使用Pattern匹配文本，获得匹配结果，无法匹配时将返回None
  match = pattern.match('hello world!')

  if match:
      # 使用Match获得分组信息
      print match.group()

  ### 输出 ###
  # hello
  ```

- search：

  `search(string[, pos[, endpos]]) | re.search(pattern, string[, flags])`: 这个方法用于查找字符串中可以匹配成功的子串。从`string`的`pos`下标处起尝试匹配`pattern`，如果`pattern`结束时仍可匹配，则返回一个`Match`对象；若无法匹配，则将`pos`加1后重新尝试匹配；直到`pos=endpos`时仍无法匹配则返回`None`。`pos`和`endpos`的默认值分别为`0`和`len(string)`；`re.search()`无法指定这两个参数，参数`flags`用于编译`pattern`时指定匹配模式。

那么它和`match`有什么区别呢？

`match()`函数只检测`re`是不是在`string`的开始位置匹配，`search()`会扫描整个`string`查找匹配，`match()`只有在0位置匹配成功的话才有返回，如果不是开始位置匹配成功的话，`match()`就返回none。

例如：`print(re.match('super', 'superstition').span())`会返回`(0,5)`，而`print(re.match('super', 'insuperable'))`则返回`None`

`search()`会扫描整个字符串并返回第一个成功的匹配 例如：
`print(re.search('super', 'superstition').span())`返回`(0,5)`
`print(re.search('super', 'insuperable').span())`返回`(2,7)`

```
import re

# 将正则表达式编译成Pattern对象
pattern = re.compile(r'world')

# 使用search()查找匹配的子串，不存在能匹配的子串时将返回None
# 这个例子中使用match()无法成功匹配
match = pattern.search('hello world!')

if match:
    # 使用Match获得分组信息
    print match.group()

### 输出 ###
# world
```

- split：

`split(string[, maxsplit]) | re.split(pattern, string[, maxsplit])`: 按照能够匹配的子串将`string`分割后返回列表。`maxsplit`用于指定最大分割次数，不指定将全部分割。

```
import re

p = re.compile(r'\d+')
print p.split('one1two2three3four4')

### output ###
# ['one', 'two', 'three', 'four', '']
```

- findall：

`findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags])`:搜索`string`，以列表形式返回全部能匹配的子串。

```
import re
```

```
p = re.compile(r'\d+')
print p.findall('one1two2three3four4')

### output ###
# ['1', '2', '3', '4']
```

- finditer： `finditer(string[, pos[, endpos]]) | re.finditer(pattern, string[, flags])`: 搜索string，返回一个顺序访问每一个匹配结果（Match对象）的迭代器。

```
import re

p = re.compile(r'\d+')
for m in p.finditer('one1two2three3four4'):
    print m.group(),

### output ###
# 1 2 3 4
```

- sub：

`sub(repl, string[, count]) | re.sub(pattern, repl, string[, count])`: 使用`repl`替换`string`中每一个匹配的子串后返回替换后的字符串。 当`repl`是一个字符串时，可以使用`\id`或`\g<id>`、`\g<name>`引用分组，但不能使用编号0。 当`repl`是一个方法时，这个方法应当只接受一个参数（`Match`对象），并返回一个字符串用于替换（返回的字符串中不能再引用分组）。 `count`用于指定最多替换次数，不指定时全部替换。

```
import re

p = re.compile(r'(\w+) (\w+)')
s = 'i say, hello world!'

print p.sub(r'\2 \1', s)

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print p.sub(func, s)

### output ###
# say i, world hello!
# I Say, Hello World!
```

- subn：

`subn(repl, string[, count]) |re.sub(pattern, repl, string[, count])`: 返回(`sub(repl, string[, count])`, 替换次数)。

```
import re
```

```
p = re.compile(r'(\w+) (\w+)')
s = 'i say, hello world!'

print p.subn(r'\2 \1', s)

def func(m):
    return m.group(1).title() + ' ' + m.group(2).title()

print p.subn(func, s)

### output ###
# ('say i, world hello!', 2)
# ('I Say, Hello World!', 2)
```

附录

本文整理自：

Python爬虫入门教程

Python正则表达式指南

Python Documentation