Table of Contents

爬虫教程

知了课堂零基础：21天搞定Python分布式爬虫配套教案。

第一章：网络请求

虚拟环境

为什么需要虚拟环境：

到目前位置，我们所有的第三方包安装都是直接通过 `pip install xx` 的方式进行安装的，这样安装会将那个包安装到你的系统级的 `Python` 环境中。但是这样有一个问题，就是如果你现在用 `Django 1.10.x` 写了个网站，然后你的领导跟你说，之前有一个旧项目是用 `Django 0.9` 开发的，让你来维护，但是 `Django 1.10` 不再兼容 `Django 0.9` 的一些语法了。这时候就会碰到一个问题，我如何在我的电脑中同时拥有 `Django 1.10` 和 `Django 0.9` 两套环境呢？这时候我们就可以通过虚拟环境来解决这个问题。

虚拟环境原理介绍：

虚拟环境相当于一个抽屉，在这个抽屉中安装的任何软件包都不会影响到其他抽屉。并且在项目中，我可以指定这个项目的虚拟环境来配合我的项目。比如我们现在有一个项目是基于 `Django 1.10.x` 版本，又有一个项目是基于 `Django 0.9.x` 的版本，那么这时候就可以创建两个虚拟环境，在这两个虚拟环境中分别安装 `Django 1.10.x` 和 `Django 0.9.x` 来适配我们的项目。

安装 `virtualenv` ：

`virtualenv` 是用来创建虚拟环境的软件工具，我们可以通过 `pip` 或者 `pip3` 来安装：

```
pip install virtualenv
pip3 install virtualenv
```

创建虚拟环境：

创建虚拟环境非常简单，通过以下命令就可以创建了：

```
virtualenv [虚拟环境的名字]
```

如果你当前的 `Python3/Scripts` 的查找路径在 `Python2/Scripts` 的前面，那么将会使用 `python3` 作为这个虚拟环境的解释器。如果 `python2/Scripts` 在 `python3/Scripts` 前面，那么将会使用 `Python2` 来作为这个虚拟环境的解释器。

进入环境：

虚拟环境创建好了以后，那么可以进入到这个虚拟环境中，然后安装一些第三方包，进入虚拟环境在不同的操作系统中有不同的方式，一般分为两种，第一种是 `Windows` ，第二种是 `*nix` ：

1. `windows` 进入虚拟环境：进入到虚拟环境的 `Scripts` 文件夹中，然后执行 `activate` 。

2. `*nix` 进入虚拟环境：`source /path/to/virtualenv/bin/activate` 一旦你进入到了这个虚拟环境中，你安装包，卸载包都是在这个虚拟环境中，不会影响到外面的环境。

退出虚拟环境：

退出虚拟环境很简单，通过一个命令就可以完成：`deactivate`。

创建虚拟环境的时候指定 `Python` 解释器：

在电脑的环境变量中，一般是不会去更改一些环境变量的顺序的。也就是说比如你的 `Python2/Scripts` 在 `Python3/Scripts` 的前面，那么你不会经常去更改他们的位置。但是这时候我确实是想在创建虚拟环境的时候用 `Python3` 这个版本，这时候可以通过 `-p` 参数来指定具体的 `Python` 解释器：

```
virtualenv -p C:\Python36\python.exe [virutalenv name]
```

---

virtualenvwrapper：

`virtualenvwrapper` 这个软件包可以让我们管理虚拟环境变得更加简单。不用再跑到某个目录下通过 `virtualenv` 来创建虚拟环境，并且激活的时候也要跑到具体的目录下去激活。

安装 `virtualenvwrapper`：

1. *nix：`pip install virtualenvwrapper`。
2. windows：`pip install virtualenvwrapper-win`。

`virtualenvwrapper` 基本使用：

1. 创建虚拟环境：

```
mkvirtualenv my_env
```

那么会在你当前用户下创建一个 `Env` 的文件夹，然后将这个虚拟环境安装到这个目录下。 如果你电脑中安装了 `python2` 和 `python3`，并且两个版本中都安装了 `virtualenvwrapper`，那么将会使用环境变量中第一个出现的 `Python` 版本来作为这个虚拟环境的 `Python` 解释器。

1. 切换到某个虚拟环境：

```
workon my_env
```

1. 退出当前虚拟环境：

```
deactivate
```

1. 删除某个虚拟环境：

```
rmvirtualenv my_env
```

1. 列出所有虚拟环境：

```
lsvirtualenv
```

1. 进入到虚拟环境所在的目录：

```
cdvirtualenv
```

修改 **mkvirtualenv** 的默认路径：

在 我的电脑->右键->属性->高级系统设置->环境变量->系统变量 中添加一个参数 WORKON_HOME ，将这个参数的值设置为你需要的路径。

创建虚拟环境的时候指定 **Python** 版本：

在使用 mkvirtualenv 的时候，可以指定 --python 的参数来指定具体的 python 路径：

```
mkvirtualenv --python==C:\Python36\python.exe hy_env
```

爬虫前奏

爬虫的实际例子：

1. 搜索引擎（百度、谷歌、360搜索等）。
2. 伯乐在线。
3. 惠惠购物助手。
4. 数据分析与研究（数据冰山知乎专栏）。
5. 抢票软件等。

什么是网络爬虫：

1. 通俗理解：爬虫是一个模拟人类请求网站行为的程序。可以自动请求网页、并数据抓取下来，然后使用一定的规则提取有价值的数据。
2. 专业介绍：百度百科。

通用爬虫和聚焦爬虫：

1. 通用爬虫：通用爬虫是搜索引擎抓取系统（百度、谷歌、搜狗等）的重要组成部分。主要是将互联网上的网页下载到本地，形成一个互联网内容的镜像备份。
2. 聚焦爬虫：是面向特定需求的一种网络爬虫程序，他与通用爬虫的区别在于：聚焦爬虫在实施网页抓取的时候会对内容进行筛选和处理，尽量保证只抓取与需求相关的网页信息。

为什么用**Python**写爬虫程序：

1. PHP：PHP是世界是最好的语言，但他天生不是做这个的，而且对多线程、异步支持不是很好，并发处理能力弱。爬虫是工具性程序，对速度和效率要求比较高。
2. Java：生态圈很完善，是Python爬虫最大的竞争对手。但是Java语言本身很笨重，代码量很大。重构成本比较高，任何修改会导致代码大量改动。爬虫经常要修改采集代码。
3. C/C++：运行效率是无敌的。但是学习和开发成本高。写个小爬虫程序可能要大半天时间。
4. Python：语法优美、代码简洁、开发效率高、支持的模块多。相关的HTTP请求模块和HTML解析模块非常丰富。还有Scrapy和Scrapy-redis框架让我们开发爬虫变得异常简单。

准备工具：

1. Python3.6开发环境。
2. Pycharm 2017 professional版。
3. 虚拟环境。`virtualenv/virtualenvwrapper`。

http协议和Chrome抓包工具

什么是http和https协议：

HTTP协议：全称是 `HyperText Transfer Protocol` ，中文意思是超文本传输协议，是一种发布和接收HTML页面的方法。服务器端口号是 `80` 端口。 HTTPS协议：是HTTP协议的加密版本，在HTTP下加入了SSL层。服务器端口号是 `443` 端口。

在浏览器中发送一个http请求的过程：

1. 当用户在浏览器的地址栏中输入一个URL并按回车键之后，浏览器会向HTTP服务器发送HTTP请求。HTTP请求主要分为“Get”和“Post”两种方法。
2. 当我们在浏览器输入URL http://www.baidu.com 的时候，浏览器发送一个Request请求去获取 http://www.baidu.com 的html文件，服务器把Response文件对象发送回给浏览器。
3. 浏览器分析Response中的 HTML，发现其中引用了很多其他文件，比如Images文件，CSS文件，JS文件。 浏览器会自动再次发送Request去获取图片，CSS文件，或者JS文件。
4. 当所有的文件都下载成功后，网页会根据HTML语法结构，完整的显示出来了。

url详解：

`URL` 是 `Uniform Resource Locator` 的简写，统一资源定位符。 一个 `URL` 由以下几部分组成：

```
scheme://host:port/path/?query-string=xxx#anchor
```

- **scheme**：代表的是访问的协议，一般为 `http` 或者 `https` 以及 `ftp` 等。
- **host**：主机名，域名，比如 `www.baidu.com` 。
- **port**：端口号。当你访问一个网站的时候，浏览器默认使用80端口。
- **path**：查找路径。比如： `www.jianshu.com/trending/now` ，后面的 `trending/now` 就是 `path` 。
- **query-string**：查询字符串，比如： `www.baidu.com/s?wd=python` ，后面的 `wd=python` 就是查询字符串。
- **anchor**：锚点，后台一般不用管，前端用来做页面定位的。

在浏览器中请求一个 `url` ，浏览器会对这个url进行一个编码。除英文字母，数字和部分符号外，其他的全部使用百分号+十六进制码值进行编码。

常用的请求方法：

在 `Http` 协议中，定义了八种请求方法。这里介绍两种常用的请求方法，分别是 `get` 请求和 `post` 请求。

1. `get` 请求：一般情况下，只从服务器获取数据下来，并不会对服务器资源产生任何影响的时候会使用 `get` 请求。
2. `post` 请求：向服务器发送数据（登录）、上传文件等，会对服务器资源产生影响的时候会使用 `post` 请求。 以上是在网站开发中常用的两种方法。并且一般情况下都会遵循使用的原则。但是有的网站和服务器为了做反爬虫机制，也经常会不按常理出牌，有可能一个应该使用 `get` 方法的请求就一定要改成 `post` 请求，这个要视情况而定。

请求头常见参数：

在 `http` 协议中，向服务器发送一个请求，数据分为三部分，第一个是把数据放在url中，第二个是把数据放在 `body` 中（在 `post` 请求中），第三个就是把数据放在 `head` 中。这里介绍在网络爬虫中经常会用到的一些请求头参数：

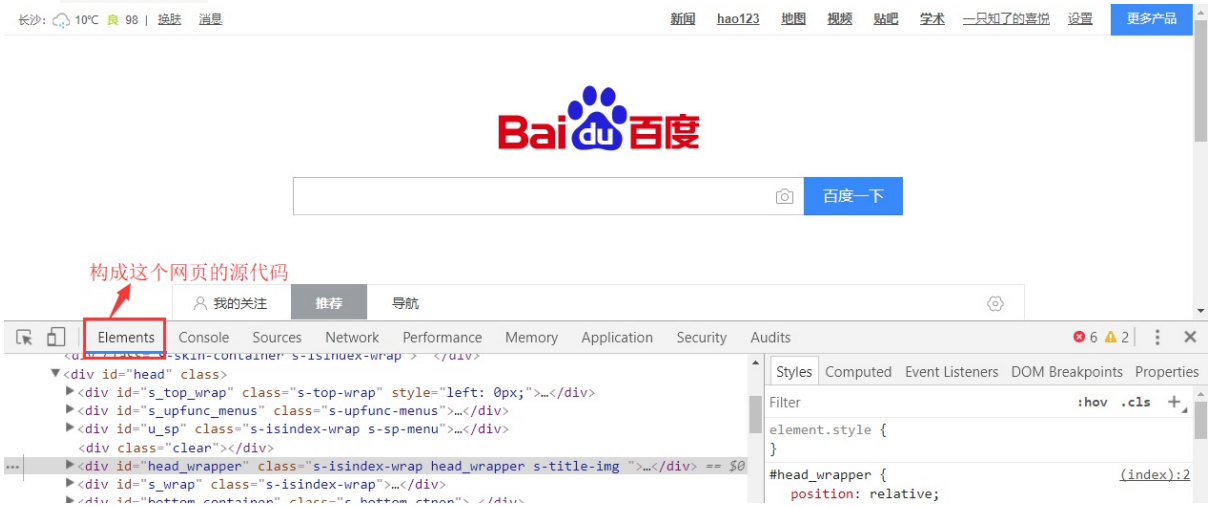
1. `User-Agent` ：浏览器名称。这个在网络爬虫中经常会被使用到。请求一个网页的时候，服务器通过这个参数就可以知道这个请求是由哪种浏览器发送的。如果我们是通过爬虫发送请求，那么我们的 `User-Agent` 就是 `Python` ，这对于那些有反爬虫机制的网站来说，可以轻易的判断你这个请求是爬虫。因此我们要经常设置这个值为一些浏览器的值，来伪装我们的爬虫。
2. `Referer` ：表明当前这个请求是从哪个 `url` 过来的。这个一般也可以用来做反爬虫技术。如果不是从指定页面过来的，那么就不做相关的响应。
3. `Cookie` ： `http` 协议是无状态的。也就是同一个人发送了两次请求，服务器没有能力知道这两个请求是否来自同一个人。因此这时候就用 `cookie` 来做标识。一般如果想要做登录后才能访问的网站，那么就需要发送 `cookie` 信息了。

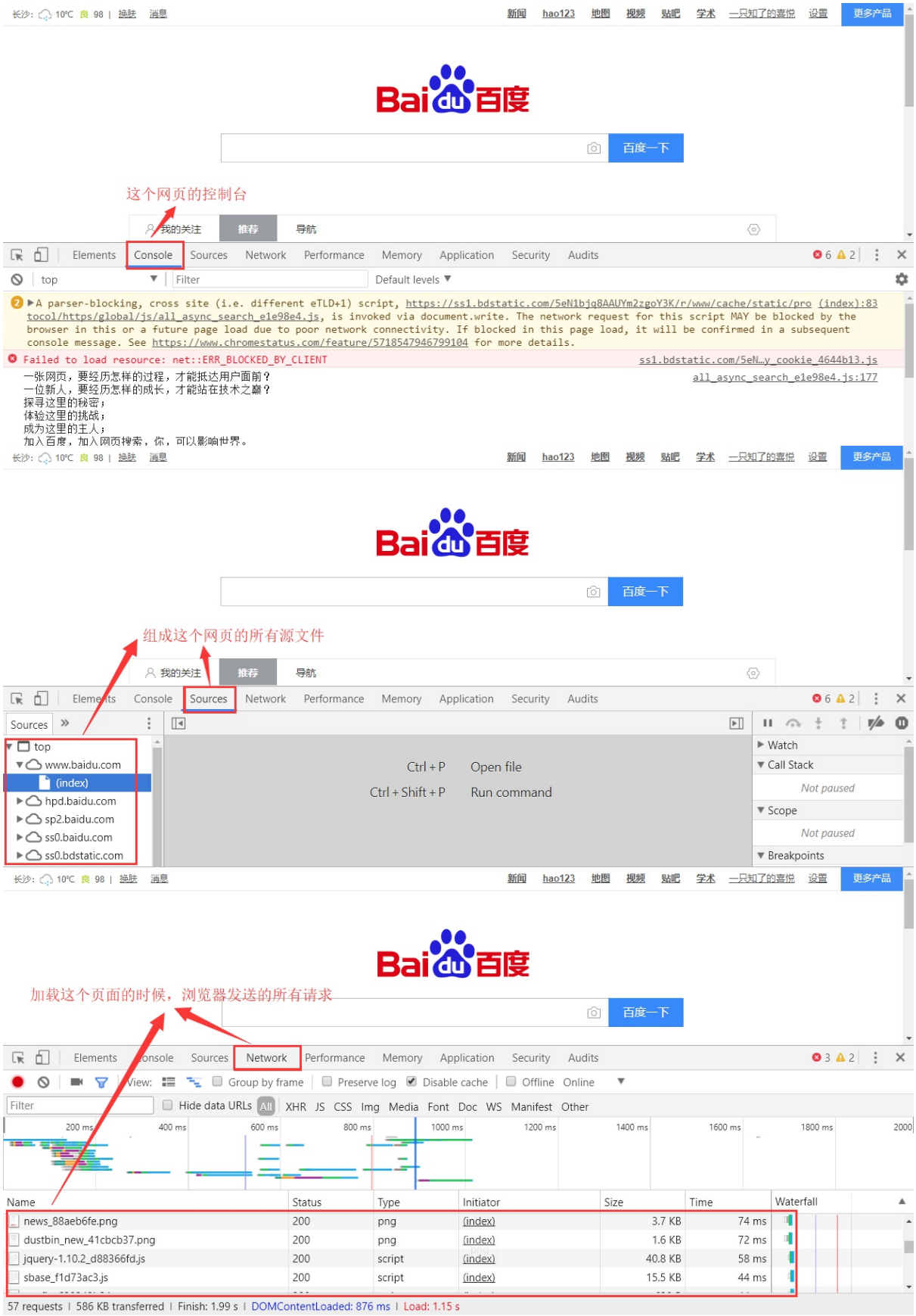
常见响应状态码：

1. `200` ：请求正常，服务器正常的返回数据。
2. `301` ：永久重定向。比如在访问 `www.jingdong.com` 的时候会重定向到 `www.jd.com` 。
3. `302` ：临时重定向。比如在访问一个需要登录的页面的时候，而此时没有登录，那么就会重定向到登录页面。
4. `400` ：请求的 `url` 在服务器上找不到。换句话说就是请求 `url` 错误。
5. `403` ：服务器拒绝访问，权限不够。
6. `500` ：服务器内部错误。可能是服务器出现 `bug` 了。

Chrome抓包工具：

`Chrome` 浏览器是一个非常亲近开发者的浏览器。可以方便的查看网络请求以及发送的参数。对着网页 `右键->检查` 。然后就可以打开开发者选项。以下用图片来说明。




这个网页的控制台
Elements
Console
Sources
Network
Performance
Memory
Application
Security
Audits
一张网页，要经历怎样的过程，才能抵达用户面前？
一位新人，要经历怎样的成长，才能站在技术之巅？
探寻这里的秘密；
体验这里的挑战；
成为这里的主人；
加入百度，加入网页搜索，你，可以影响世界。
组成这个网页的所有源文件
top
www.baidu.com
(index)
hpd.baidu.com
sp2.baidu.com
ss0.baidu.com
ss0.bdstatic.com
Ctrl + P Open file
Ctrl + Shift + P Run command
Watch
Call Stack
Not paused
Scope
Not paused
Breakpoints
加载这个页面的时候，浏览器发送的所有请求
Name
Status
Type
Initiator
Size
Time
Waterfall
news_88aeb6fe.png 200 png (index) 3.7 KB 74 ms
dustbin_new_41cbcb37.png 200 png (index) 1.6 KB 72 ms
jquery-1.10.2_d88366fd.js 200 script (index) 40.8 KB 58 ms
sbase_f1d73ac3.js 200 script (index) 15.5 KB 44 ms
57 requests | 586 KB transferred | Finish: 1.99 s | DOMContentLoaded: 876 ms | Load: 1.15 s

urllib库

`urllib` 库是 `Python` 中一个最基本的网络请求库。可以模拟浏览器的行为，向指定的服务器发送一个请求，并可以保存服务器返回的数据。

urlopen函数：

在 `Python3` 的 `urllib` 库中，所有和网络请求相关的方法，都被集到 `urllib.request` 模块下面了，以先来看下 `urlopen` 函数基本的使用：

```
from urllib import request
resp = request.urlopen('http://www.baidu.com')
print(resp.read())
```

实际上，使用浏览器访问百度，右键查看源代码。你会发现，跟我们刚才打印出来的数据是一模一样的。也就是说，上面的三行代码就已经帮我们把百度的首页的全部代码爬下来了。一个基本的url请求对应的python代码真的非常简单。
以下对 `urlopen` 函数的进行详细讲解：

1. `url`：请求的url。
2. `data`：请求的 `data`，如果设置了这个值，那么将变成 `post` 请求。
3. 返回值：返回值是一个 `http.client.HTTPResponse` 对象，这个对象是一个类文件句柄对象。有 `read(size)`、`readline`、`readlines` 以及 `getcode` 等方法。

urlretrieve函数：

这个函数可以方便的将网页上的一个文件保存到本地。以下代码可以非常方便的将百度的首页下载到本地：

```
from urllib import request
request.urlretrieve('http://www.baidu.com/','baidu.html')
```

urlencode函数：

用浏览器发送请求的时候，如果url中包含了中文或者其他特殊字符，那么浏览器会自动的给我们进行编码。而如果使用代码发送请求，那么就必须手动的进行编码，这时候就应该使用 `urlencode` 函数来实现。 `urlencode` 可以把字典数据转换为 `URL` 编码的数据。示例代码如下：

```
from urllib import parse
data = {'name':'爬虫基础','greet':'hello world','age':100}
qs = parse.urlencode(data)
print(qs)
```

parse_qs函数：

可以将经过编码后的url参数进行解码。示例代码如下：

```
from urllib import parse
qs = "name=%E7%88%AC%E8%99%AB%E5%9F%BA%E7%A1%80&greet=hello+world&age=100"
print(parse.parse_qs(qs))
```

urlparse和urlsplit：

有时候拿到一个url，想要对这个url中的各个组成部分进行分割，那么这时候就可以使用 `urlparse` 或者是 `urlsplit` 来进行分割。示例代码如下：

```
from urllib import request,parse

url = 'http://www.baidu.com/s?username=zhiliao'

result = parse.urlsplit(url)
# result = parse.urlparse(url)

print('scheme:',result.scheme)
print('netloc:',result.netloc)
print('path:',result.path)
print('query:',result.query)
```

`urlparse` 和 `urlsplit` 基本上是一模一样的。唯一不一样的地方是， `urlparse` 里面多了一个 `params` 属性，而 `urlsplit` 没有这个 `params` 属性。比如有一个 `url` 为： `url = 'http://www.baidu.com/s;hello?wd=python&username=abc#1'` ，那么 `urlparse` 可以获取到 `hello` ，而 `urlsplit` 不可以获取到。 `url` 中的 `params` 也用得比较少。

request.Request类：

如果想要在请求的时候增加一些请求头，那么就必须使用 `request.Request` 类来实现。比如要增加一个 `User-Agent` ，示例代码如下：

```
from urllib import request

headers = {
'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36'
}
req = request.Request("http://www.baidu.com/",headers=headers)
```

```
resp = request.urlopen(req)
print(resp.read())
```

糗事百科爬虫实战作业：

1. url链接：http://neihanshequ.com/bar/1/
2. 要求：能爬取一页的数据就可以了。

ProxyHandler处理器（代理设置）

很多网站会检测某一段时间某个IP的访问次数(通过流量统计，系统日志等)，如果访问次数多的不像正常人，它会禁止这个IP的访问。
所以我们可以设置一些代理服务器，每隔一段时间换一个代理，就算IP被禁止，依然可以换个IP继续爬取。
urllib中通过ProxyHandler来设置使用代理服务器，下面代码说明如何使用自定义opener来使用代理：

```
from urllib import request

# 这个是没有使用代理的
# resp = request.urlopen('http://httpbin.org/get')
# print(resp.read().decode("utf-8"))

# 这个是使用了代理的
handler = request.ProxyHandler({"http":"218.66.161.88:31769"})

opener = request.build_opener(handler)
req = request.Request("http://httpbin.org/ip")
resp = opener.open(req)
print(resp.read())
```

常用的代理有：

- 西刺免费代理IP：http://www.xicidaili.com/
- 快代理：http://www.kuaidaili.com/
- 代理云：http://www.dailiyun.com/

什么是cookie：

在网站中，http请求是无状态的。也就是说即使第一次和服务器连接后并且登录成功后，第二次请求服务器依然不能知道当前请求是哪个用户。`cookie`的出现就是为了解决这个问题，第一次登录后服务器返回一些数据（cookie）给浏览器，然后浏览器保存在本地，当该用户发送第二次请求的

时候，就会自动的把上次请求存储的 `cookie` 数据自动的携带给服务器，服务器通过浏览器携带的数据就能判断当前用户是哪个了。 `cookie` 存储的数据量有限，不同的浏览器有不同的存储大小，但一般不超过4KB。因此使用 `cookie` 只能存储一些小量的数据。

cookie的格式：

```
Set-Cookie: NAME=VALUE；Expires/Max-age=DATE；Path=PATH；Domain=DOMAIN_NAME；SECURE
```

参数意义：

- NAME：cookie的名字。
- VALUE：cookie的值。
- Expires：cookie的过期时间。
- Path：cookie作用的路径。
- Domain：cookie作用的域名。
- SECURE：是否只在https协议下起作用。

使用cookielib库和HTTPCookieProcessor模拟登录：

Cookie 是指网站服务器为了辨别用户身份和进行Session跟踪，而储存在用户浏览器上的文本文件，Cookie可以保持登录信息到用户下次与服务器的会话。
这里以人人网为例。人人网中，要访问某个人的主页，必须先登录才能访问，登录说白了就是要有cookie信息。那么如果我们想要用代码的方式访问，就必须要有正确的cookie信息才能访问。解决方案有两种，第一种是使用浏览器访问，然后将cookie信息复制下来，放到headers中。示例代码如下：

```
from urllib import request

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36',
    'Cookie': 'anonymid=jacdwz2x-8bjldx; depovince=GW; _r01_=1; _ga=GA1.2.1455063316.15
11436360; _gid=GA1.2.862627163.1511436360; wp=1; JSESSIONID=abczwY8ecd4xz8RJcyP-v; jebe
cookies=d4497791-9d41-4269-9e2b-3858d4989785|||||; ick_login=884e75d4-f361-4cff-94bb-81
fe6c42b220; _de=EA5778F44555C091303554EBBEB4676C696BF75400CE19CC; p=61a3c7d0d4b2d1e9910
95353f83fa2141; first_login_flag=1; ln_uact=970138074@qq.com; ln_hurl=http://hdn.xnimg.
cn/photos/hdn121/20170428/1700/main_nhiB_aebd0000854a1986.jpg; t=3dd84a3117737e819dd2c3
2f1cdb91d01; societyguester=3dd84a3117737e819dd2c32f1cdb91d01; id=443362311; xnsid=169e
fdc0; loginfrom=syshome; ch_id=10016; jebe_key=9c062f5a-4335-4a91-bf7a-970f8b86a64e%7Ca
022c303305d1b2ab6b5089643e4b5de%7C1511449232839%7C1; wp_fold=0'
}

url = 'http://www.renren.com/880151247/profile'

req = request.Request(url,headers=headers)
```

```
resp = request.urlopen(req)
with open('renren.html','w') as fp:
    fp.write(resp.read().decode('utf-8'))
```

但是每次在访问需要cookie的页面都要从浏览器中复制cookie比较麻烦。在Python处理Cookie，一般是通过 `http.cookiejar` 模块和 `urllib模块的HTTPCookieProcessor` 处理器类一起使用。 `http.cookiejar` 模块主要作用是提供用于存储cookie的对象。而 `HTTPCookieProcessor` 处理器主要作用是处理这些cookie对象，并构建handler对象。

http.cookiejar模块：

该模块主要的类有CookieJar、FileCookieJar、MozillaCookieJar、LWPCookieJar。这四个类的作用分别如下：

1. CookieJar：管理HTTP cookie值、存储HTTP请求生成的cookie、向传出的HTTP请求添加cookie的对象。整个cookie都存储在内存中，对CookieJar实例进行垃圾回收后cookie也将丢失。
2. FileCookieJar (filename,delayload=None,policy=None)：从CookieJar派生而来，用来创建FileCookieJar实例，检索cookie信息并将cookie存储到文件中。filename是存储cookie的文件名。delayload为True时支持延迟访问访问文件，即只有在需要时才读取文件或在文件中存储数据。
3. MozillaCookieJar (filename,delayload=None,policy=None)：从FileCookieJar派生而来，创建与Mozilla浏览器 cookies.txt兼容的FileCookieJar实例。
4. LWPCookieJar (filename,delayload=None,policy=None)：从FileCookieJar派生而来，创建与libwww-perl标准的 Set-Cookie3 文件格式兼容的FileCookieJar实例。

登录人人网：

利用 `http.cookiejar` 和 `request.HTTPCookieProcessor` 登录人人网。相关示例代码如下：

```
from urllib import request,parse
from http.cookiejar import CookieJar

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36'
}

def get_opener():
    cookiejar = CookieJar()
    handler = request.HTTPCookieProcessor(cookiejar)
    opener = request.build_opener(handler)
    return opener

def login_renren(opener):
    data = {"email": "970138074@qq.com", "password": "pythonspider"}
```

```
    data = parse.urlencode(data).encode('utf-8')
    login_url = "http://www.renren.com/PLogin.do"
    req = request.Request(login_url, headers=headers, data=data)
    opener.open(req)

def visit_profile(opener):
    url = 'http://www.renren.com/880151247/profile'
    req = request.Request(url,headers=headers)
    resp = opener.open(req)
    with open('renren.html','w') as fp:
        fp.write(resp.read().decode("utf-8"))

if __name__ == '__main__':
    opener = get_opener()
    login_renren(opener)
    visit_profile(opener)
```

保存**cookie**到本地：

保存 `cookie` 到本地，可以使用 `cookiejar` 的 `save` 方法，并且需要指定一个文件名：

```
from urllib import request
from http.cookiejar import MozillaCookieJar

cookiejar = MozillaCookieJar("cookie.txt")
handler = request.HTTPCookieProcessor(cookiejar)
opener = request.build_opener(handler)

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36'
}
req = request.Request('http://httpbin.org/cookies',headers=headers)

resp = opener.open(req)
print(resp.read())
cookiejar.save(ignore_discard=True,ignore_expires=True)
```

从本地加载**cookie**：

从本地加载 `cookie` ，需要使用 `cookiejar` 的 `load` 方法，并且也需要指定方法：

```
from urllib import request
from http.cookiejar import MozillaCookieJar

cookiejar = MozillaCookieJar("cookie.txt")
cookiejar.load(ignore_expires=True,ignore_discard=True)
```

```
handler = request.HTTPCookieProcessor(cookiejar)
opener = request.build_opener(handler)

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36'
}
req = request.Request('http://httpbin.org/cookies',headers=headers)

resp = opener.open(req)
print(resp.read())
```

requests库

虽然Python的标准库中 urllib模块已经包含了平常我们使用的大多数功能，但是它的 API 使用起来让人感觉不太好，而 Requests宣传是 “HTTP for Humans”，说明使用更简洁方便。

安装和文档地址：

利用 `pip` 可以非常方便的安装：

```
pip install requests
```

中文文档：http://docs.python-requests.org/zh_CN/latest/index.html
github地址：https://github.com/requests/requests

发送GET请求：

1. 最简单的发送 `get` 请求就是通过 `requests.get` 来调用：

```
response = requests.get("http://www.baidu.com/")
```

2. 添加headers和查询参数：
   如果想添加 headers，可以传入headers参数来增加请求头中的headers信息。如果要将参数放在url中传递，可以利用 params 参数。相关示例代码如下：

```
import requests

kw = {'wd':'中国'}

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}

# params 接收一个字典或者字符串的查询参数，字典类型自动转换为url编码，不需要urlencode()
response = requests.get("http://www.baidu.com/s", params = kw, headers = headers)

# 查看响应内容，response.text 返回的是Unicode格式的数据
print(response.text)

# 查看响应内容，response.content返回的字节流数据
print(response.content)

# 查看完整url地址
print(response.url)
```

```
# 查看响应头部字符编码
print(response.encoding)

# 查看响应码
print(response.status_code)
```

发送**POST**请求：

1. 最基本的POST请求可以使用 `post` 方法：

```
response = requests.post("http://www.baidu.com/",data=data)
```

2. 传入data数据：
   这时候就不要再使用 `urlencode` 进行编码了，直接传入一个字典进去就可以了。比如请求拉勾网的数据的代码：

```
import requests

url = "https://www.lagou.com/jobs/positionAjax.json?city=%E6%B7%B1%E5%9C%B3&needAddtionalResult=false&isSchoolJob=0"

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36',
    'Referer': 'https://www.lagou.com/jobs/list_python?labelWords=&fromSearch=true&suginput='
}

data = {
    'first': 'true',
    'pn': 1,
    'kd': 'python'
}

resp = requests.post(url,headers=headers,data=data)
# 如果是json数据，直接可以调用json方法
print(resp.json())
```

使用代理：

使用 `requests` 添加代理也非常简单，只要在请求的方法中（比如 `get` 或者 `post` ）传递 `proxies` 参数就可以了。示例代码如下：

```
import requests

url = "http://httpbin.org/get"

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36',
}

proxy = {
    'http': '171.14.209.180:27829'
}

resp = requests.get(url,headers=headers,proxies=proxy)
with open('xx.html','w',encoding='utf-8') as fp:
    fp.write(resp.text)
```

cookie：

如果在一个响应中包含了 `cookie` ，那么可以利用 `cookies` 属性拿到这个返回的 `cookie` 值：

```
import requests

url = "http://www.renren.com/PLogin.do"
data = {"email":"970138074@qq.com",'password':"pythonspider"}
resp = requests.get('http://www.baidu.com/')
print(resp.cookies)
print(resp.cookies.get_dict())
```

session：

之前使用 `urllib` 库，是可以使用 `opener` 发送多个请求，多个请求之间是可以共享 `cookie` 的。那么如果使用 `requests` ，也要达到共享 `cookie` 的目的，那么可以使用 `requests` 库给我们提供的 `session` 对象。注意，这里的 `session` 不是web开发中的那个session，这个地方只是一个会话的对象而已。还是以登录人人网为例，使用 `requests` 来实现。示例代码如下：

```
import requests

url = "http://www.renren.com/PLogin.do"
data = {"email":"970138074@qq.com",'password':"pythonspider"}
headers = {
```

```
    'User-Agent': "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/62.0.3202.94 Safari/537.36"
}

# 登录
session = requests.session()
session.post(url,data=data,headers=headers)

# 访问大鹏个人中心
resp = session.get('http://www.renren.com/880151247/profile')

print(resp.text)
```

处理不信任的SSL证书：

对于那些已经被信任的SSL整数的网站，比如 `https://www.baidu.com/` ，那么使用 `requests` 直接就可以正常的返回响应。示例代码如下：

```
resp = requests.get('http://www.12306.cn/mormhweb/',verify=False)
print(resp.content.decode('utf-8'))
```

XPath语法和lxml模块

什么是XPath？

xpath（XML Path Language）是一门在XML和HTML文档中查找信息的语言，可用来在XML和HTML文档中对元素和属性进行遍历。

XPath开发工具

1. Chrome插件XPath Helper。
2. Firefox插件Try XPath。

XPath语法

选取节点：

XPath 使用路径表达式来选取 XML 文档中的节点或者节点集。这些路径表达式和我们在常规的电脑文件系统中看到的表达式非常相似。

表达式	描述	示例	结果
nodename	选取此节点的所有子节点	bookstore	选取bookstore下所有的子节点
/	如果是在最前面，代表从根节点选取。否则选择某节点下的某个节点	/bookstore	选取根元素下所有的bookstore节点
//	从全局节点中选择节点，随便在哪个位置	//book	从全局节点中找到所有的book节点
@	选取某个节点的属性	//book[@price]	选择所有拥有price属性的book节点
.	当前节点	./a	选取当前节点下的a标签

谓语：

谓语用来查找某个特定的节点或者包含某个指定的值的节点，被嵌在方括号中。
在下面的表格中，我们列出了带有谓语的一些路径表达式，以及表达式的结果：

路径表达式	描述
/bookstore/book[1]	选取bookstore下的第一个子元素
/bookstore/book[last()]	选取bookstore下的倒数第二个book元素。
bookstore/book[position()<3]	选取bookstore下前面两个子元素。
//book[@price]	选取拥有price属性的book元素
//book[@price=10]	选取所有属性price等于10的book元素

通配符

*表示通配符。

通配符	描述	示例	结果
*	匹配任意节点	/bookstore/*	选取bookstore下的所有子元素。
@*	匹配节点中的任何属性	//book[@*]	选取所有带有属性的book元素。

选取多个路径：

通过在路径表达式中使用“|”运算符，可以选取若干个路径。
示例如下：

```
//bookstore/book | //book/title
# 选取所有book元素以及book元素下所有的title元素
```

运算符：

运算符	描述	实例	返回值
|	计算两个节点集	//book | //cd	返回所有拥有 book 和 cd 元素的节点集
+	加法	6 + 4	10
-	减法	6 - 4	2
*	乘法	6 * 4	24
div	除法	8 div 4	2
=	等于	price=9.80	如果 price 是 9.80，则返回 true。如果 price 是 9.90，则返回 false。
!=	不等于	price!=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
<	小于	price<9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
<=	小于或等于	price<=9.80	如果 price 是 9.00，则返回 true。如果 price 是 9.90，则返回 false。
>	大于	price>9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.80，则返回 false。
>=	大于或等于	price>=9.80	如果 price 是 9.90，则返回 true。如果 price 是 9.70，则返回 false。
or	或	price=9.80 or price=9.70	如果 price 是 9.80，则返回 true。如果 price 是 9.50，则返回 false。
and	与	price>9.00 and price<9.90	如果 price 是 9.80，则返回 true。如果 price 是 8.50，则返回 false。
mod	计算除法的余数	5 mod 2	1

lxml库

lxml 是 一个HTML/XML的解析器，主要的功能是如何解析和提取 HTML/XML 数据。

lxml和正则一样，也是用 C 实现的，是一款高性能的 Python HTML/XML 解析器，我们可以利用之前学习的XPath语法，来快速的定位特定元素以及节点信息。

lxml python 官方文档：http://lxml.de/index.html

需要安装C语言库，可使用 pip 安装：pip install lxml

基本使用：

我们可以利用他来解析HTML代码，并且在解析HTML代码的时候，如果HTML代码不规范，他会自动的进行补全。示例代码如下：

```
# 使用 lxml 的 etree 库
from lxml import etree

text = '''
<div>
    <ul>
         <li class="item-0"><a href="link1.html">first item</a></li>
         <li class="item-1"><a href="link2.html">second item</a></li>
         <li class="item-inactive"><a href="link3.html">third item</a></li>
         <li class="item-1"><a href="link4.html">fourth item</a></li>
         <li class="item-0"><a href="link5.html">fifth item</a> # 注意，此处缺少一个 </l
i> 闭合标签
     </ul>
 </div>
'''

#利用etree.HTML，将字符串解析为HTML文档
html = etree.HTML(text)

# 按字符串序列化HTML文档
result = etree.tostring(html)

print(result)
```

输入结果如下：

```
<html><body>
<div>
    <ul>
         <li class="item-0"><a href="link1.html">first item</a></li>
         <li class="item-1"><a href="link2.html">second item</a></li>
         <li class="item-inactive"><a href="link3.html">third item</a></li>
         <li class="item-1"><a href="link4.html">fourth item</a></li>
         <li class="item-0"><a href="link5.html">fifth item</a></li>
</ul>
 </div>
</body></html>
```

可以看到。lxml会自动修改HTML代码。例子中不仅补全了li标签，还添加了body，html标签。

从文件中读取**html**代码：

除了直接使用字符串进行解析，lxml还支持从文件中读取内容。我们新建一个hello.html文件：

```
<!-- hello.html -->
<div>
    <ul>
         <li class="item-0"><a href="link1.html">first item</a></li>
         <li class="item-1"><a href="link2.html">second item</a></li>
         <li class="item-inactive"><a href="link3.html"><span class="bold">third item</
span></a></li>
         <li class="item-1"><a href="link4.html">fourth item</a></li>
         <li class="item-0"><a href="link5.html">fifth item</a></li>
     </ul>
 </div>
```

然后利用 `etree.parse()` 方法来读取文件。示例代码如下：

```
from lxml import etree

# 读取外部文件 hello.html
html = etree.parse('hello.html')
result = etree.tostring(html, pretty_print=True)

print(result)
```

输入结果和之前是相同的。

在lxml中使用XPath语法：

1. 获取所有li标签：

   ```
   from lxml import etree

   html = etree.parse('hello.html')
   print type(html)  # 显示etree.parse() 返回类型

   result = html.xpath('//li')

   print(result)  # 打印<li>标签的元素集合
   ```

2. 获取所有li元素下的所有class属性的值：

   ```
   from lxml import etree

   html = etree.parse('hello.html')
   result = html.xpath('//li/@class')

   print(result)
   ```

3. 获取li标签下href为 `www.baidu.com` 的a标签：

```
from lxml import etree

html = etree.parse('hello.html')
result = html.xpath('//li/a[@href="www.baidu.com"]')

print(result)
```

4. 获取li标签下所有span标签：

```
from lxml import etree

html = etree.parse('hello.html')

#result = html.xpath('//li/span')
#注意这么写是不对的：
#因为 / 是用来获取子元素的，而 <span> 并不是 <li> 的子元素，所以，要用双斜杠

result = html.xpath('//li//span')

print(result)
```

5. 获取li标签下的a标签里的所有class：

```
from lxml import etree

html = etree.parse('hello.html')
result = html.xpath('//li/a//@class')

print(result)
```

6. 获取最后一个li的a的href属性对应的值：

```
from lxml import etree

html = etree.parse('hello.html')

result = html.xpath('//li[last()]/a/@href')
# 谓语 [last()] 可以找到最后一个元素

print(result)
```

7. 获取倒数第二个li元素的内容：

```
from lxml import etree

html = etree.parse('hello.html')
result = html.xpath('//li[last()-1]/a')

# text 方法可以获取元素内容
print(result[0].text)
```

8. 获取倒数第二个li元素的内容的第二种方式：

```
from lxml import etree

html = etree.parse('hello.html')
result = html.xpath('//li[last()-1]/a/text()')

print(result)
```

使用**requests**和**xpath**爬取电影天堂

示例代码如下：

```
import requests
from lxml import etree

BASE_DOMAIN = 'http://www.dytt8.net'
HEADERS = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
 like Gecko) Chrome/61.0.3163.100 Safari/537.36',
    'Referer': 'http://www.dytt8.net/html/gndy/dyzz/list_23_2.html'
}

def spider():
    url = 'http://www.dytt8.net/html/gndy/dyzz/list_23_1.html'
    resp = requests.get(url,headers=HEADERS)
    # resp.content：经过编码后的字符串
    # resp.text：没有经过编码，也就是unicode字符串
    # text：相当于是网页中的源代码了
    text = resp.content.decode('gbk')
    # tree：经过lxml解析后的一个对象，以后使用这个对象的xpath方法，就可以
    # 提取一些想要的数据了
    tree = etree.HTML(text)
    # xpath/beautifulsou4
    all_a = tree.xpath("//div[@class='co_content8']//a")
    for a in all_a:
        title = a.xpath("text()")[0]
        href = a.xpath("@href")[0]
```

```
        if href.startswith('/'):
            detail_url = BASE_DOMAIN + href
            crawl_detail(detail_url)
            break

def crawl_detail(url):
    resp = requests.get(url,headers=HEADERS)
    text = resp.content.decode('gbk')
    tree = etree.HTML(text)
    create_time = tree.xpath("//div[@class='co_content8']/ul/text()")[0].strip()
    imgs = tree.xpath("//div[@id='Zoom']//img/@src")
    # 电影海报
    cover = imgs[0]
    # 电影截图
    screenshoot = imgs[1]
    # 获取span标签下所有的文本
    infos = tree.xpath("//div[@id='Zoom']//text()")
    for index,info in enumerate(infos):
        if info.startswith("◎年　　代"):
            year = info.replace("◎年　　代","").strip()

        if info.startswith("◎豆瓣评分"):
            douban_rating = info.replace("◎豆瓣评分",'').strip()
            print(douban_rating)

        if info.startswith("◎主　　演"):
            # 从当前位置，一直往下面遍历
            actors = [info]
            for x in range(index+1,len(infos)):
                actor = infos[x]
                if actor.startswith("◎"):
                    break
                actors.append(actor.strip())
            print(",".join(actors))


if __name__ == '__main__':
    spider()
```

chrome相关问题：

在62版本（目前最新）中有一个bug，在页面302重定向的时候不能记录FormData数据。这个是这个版本的一个bug。详细见以下链接：https://stackoverflow.com/questions/34015735/http-post-payload-not-visible-in-chrome-debugger。

在金丝雀版本中已经解决了这个问题，可以下载这个版本继续，链接如下：https://www.google.com/chrome/browser/canary.html

作业：

使用requests和xpath爬取腾讯招聘网信息。要求为获取每个职位的详情信息。

BeautifulSoup4库

和 lxml 一样，Beautiful Soup 也是一个HTML/XML的解析器，主要的功能也是如何解析和提取 HTML/XML 数据。
lxml 只会局部遍历，而Beautiful Soup 是基于HTML DOM（Document Object Model）的，会载入整个文档，解析整个DOM树，因此时间和内存开销都会大很多，所以性能要低于lxml。
BeautifulSoup 用来解析 HTML 比较简单，API非常人性化，支持CSS选择器、Python标准库中的HTML解析器，也支持 lxml 的 XML解析器。
Beautiful Soup 3 目前已经停止开发，推荐现在的项目使用Beautiful Soup 4。

安装和文档：

1. 安装：`pip install bs4` 。
2. 中文文档：https://www.crummy.com/software/BeautifulSoup/bs4/doc/index.zh.html

几大解析工具对比：

解析工具	解析速度	使用难度
BeautifulSoup	最慢	最简单
lxml	快	简单
正则	最快	最难

简单使用：

```
from bs4 import BeautifulSoup

html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""

#创建 Beautiful Soup 对象
# 使用lxml来进行解析
```

```
soup = BeautifulSoup(html,"lxml")

print(soup.prettify())
```

四个常用的对象：

Beautiful Soup将复杂HTML文档转换成一个复杂的树形结构,每个节点都是Python对象,所有对象可以归纳为4种:

1. Tag
2. NavigatableString
3. BeautifulSoup
4. Comment

1. Tag：

Tag 通俗点讲就是 HTML 中的一个个标签。示例代码如下：

```
from bs4 import BeautifulSoup

html = """
<html><head><title>The Dormouse's story</title></head>
<body>
<p class="title" name="dromouse"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1"><!-- Elsie --></a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>
<p class="story">...</p>
"""

#创建 Beautiful Soup 对象
soup = BeautifulSoup(html,'lxml')


print soup.title
# <title>The Dormouse's story</title>

print soup.head
# <head><title>The Dormouse's story</title></head>

print soup.a
# <a class="sister" href="http://example.com/elsie" id="link1"><!-- Elsie --></a>

print soup.p
```

```
# <p class="title" name="dromouse"><b>The Dormouse's story</b></p>

print type(soup.p)
# <class 'bs4.element.Tag'>
```

我们可以利用 soup 加标签名轻松地获取这些标签的内容，这些对象的类型是bs4.element.Tag。但是注意，它查找的是在所有内容中的第一个符合要求的标签。如果要查询所有的标签，后面会进行介绍。
对于Tag，它有两个重要的属性，分别是name和attrs。示例代码如下：

```
print soup.name
# [document] #soup 对象本身比较特殊，它的 name 即为 [document]

print soup.head.name
# head #对于其他内部标签，输出的值便为标签本身的名称

print soup.p.attrs
# {'class': ['title'], 'name': 'dromouse'}
# 在这里，我们把 p 标签的所有属性打印输出了出来，得到的类型是一个字典。

print soup.p['class'] # soup.p.get('class')
# ['title'] #还可以利用get方法，传入属性的名称，二者是等价的

soup.p['class'] = "newClass"
print soup.p # 可以对这些属性和内容等等进行修改
# <p class="newClass" name="dromouse"><b>The Dormouse's story</b></p>
```

2. NavigableString：

如果拿到标签后，还想获取标签中的内容。那么可以通过 `tag.string` 获取标签中的文字。示例代码如下：

```
print soup.p.string
# The Dormouse's story

print type(soup.p.string)
# <class 'bs4.element.NavigableString'>thon
```

3. BeautifulSoup：

BeautifulSoup 对象表示的是一个文档的全部内容.大部分时候,可以把它当作 Tag 对象,它支持 遍历文档树 和 搜索文档树 中描述的大部分的方法.
因为 BeautifulSoup 对象并不是真正的HTML或XML的tag,所以它没有name和attribute属性.但有时

查看它的 .name 属性是很方便的,所以 BeautifulSoup 对象包含了一个值为 “[document]” 的特殊属性 .name

```
soup.name
# '[document]'
```

4. Comment：

Tag , NavigableString , BeautifulSoup 几乎覆盖了html和xml中的所有内容,但是还有一些特殊对象.容易让人担心的内容是文档的注释部分:

```
markup = "<b><!--Hey, buddy. Want to buy a used parser?--></b>"
soup = BeautifulSoup(markup)
comment = soup.b.string
type(comment)
# <class 'bs4.element.Comment'>
```

Comment 对象是一个特殊类型的 NavigableString 对象:

```
comment
# 'Hey, buddy. Want to buy a used parser'
```

遍历文档树：

1. contents和children：

```
html_doc = """
<html><head><title>The Dormouse's story</title></head>

<p class="title"><b>The Dormouse's story</b></p>

<p class="story">Once upon a time there were three little sisters; and their names were
<a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
<a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
<a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""

from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc,'lxml')
```

```
head_tag = soup.head
# 返回所有子节点的列表
print(head_tag.contents)

# 返回所有子节点的迭代器
for child in head_tag.children:
    print(child)
```

2. strings 和 stripped_strings

如果tag中包含多个字符串 [2] ,可以使用 .strings 来循环获取：

```
for string in soup.strings:
    print(repr(string))
    # u"The Dormouse's story"
    # u'\n\n'
    # u"The Dormouse's story"
    # u'\n\n'
    # u'Once upon a time there were three little sisters; and their names were\n'
    # u'Elsie'
    # u',\n'
    # u'Lacie'
    # u' and\n'
    # u'Tillie'
    # u';\nand they lived at the bottom of a well.'
    # u'\n\n'
    # u'...'
    # u'\n'
```

输出的字符串中可能包含了很多空格或空行,使用 .stripped_strings 可以去除多余空白内容：

```
for string in soup.stripped_strings:
    print(repr(string))
    # u"The Dormouse's story"
    # u"The Dormouse's story"
    # u'Once upon a time there were three little sisters; and their names were'
    # u'Elsie'
    # u','
    # u'Lacie'
    # u'and'
    # u'Tillie'
    # u';\nand they lived at the bottom of a well.'
    # u'...'
```

搜索文档树：

1. find和find_all方法：

搜索文档树，一般用得比较多的就是两个方法，一个是 `find` ，一个是 `find_all` 。 `find` 方法是找到第一个满足条件的标签后就立即返回，只返回一个元素。 `find_all` 方法是把所有满足条件的标签都选到，然后返回回去。使用这两个方法，最常用的用法是出入 `name` 以及 `attr` 参数找出符合要求的标签。

```
soup.find_all("a",attrs={"id":"link2"})
```

或者是直接传入属性的的名字作为关键字参数：

```
soup.find_all("a",id='link2')
```

2. select方法：

使用以上方法可以方便的找出元素。但有时候使用 `css` 选择器的方式可以更加的方便。使用 `css` 选择器的语法，应该使用 `select` 方法。以下列出几种常用的 `css` 选择器方法：

（1）通过标签名查找：

```
print(soup.select('a'))
```

（2）通过类名查找：

通过类名，则应该在类的前面加一个 `.` 。比如要查找 `class=sister` 的标签。示例代码如下：

```
print(soup.select('.sister'))
```

（3）通过id查找：

通过id查找，应该在id的名字前面加一个＃号。示例代码如下：

```
print(soup.select("#link1"))
```

（4）组合查找：

组合查找即和写 class 文件时，标签名与类名、id名进行的组合原理是一样的，例如查找 p 标签中，id 等于 link1的内容，二者需要用空格分开：

```
print(soup.select("p #link1"))
```

直接子标签查找，则使用 > 分隔：

```
print(soup.select("head > title"))
```

（5）通过属性查找：

查找时还可以加入属性元素，属性需要用中括号括起来，注意属性和标签属于同一节点，所以中间不能加空格，否则会无法匹配到。示例代码如下：

```
print(soup.select('a[href="http://example.com/elsie"]'))
```

（6）获取内容

以上的 select 方法返回的结果都是列表形式，可以遍历形式输出，然后用 get_text() 方法来获取它的内容。

```
soup = BeautifulSoup(html, 'lxml')
print type(soup.select('title'))
print soup.select('title')[0].get_text()

for title in soup.select('title'):
    print title.get_text()
```

正则表达式和**re**模块：

什么是正则表达式：

通俗理解：按照一定的规则，从某个字符串中匹配出想要的数据。这个规则就是正则表达式。
标准答案：https://baike.baidu.com/item/正则表达式/1700215?fr=aladdin

一个段子：

世界是分为两种人，一种是懂正则表达式的，一种是不懂正则表达式的。

正则表达式常用匹配规则：

匹配某个字符串：

```
text = 'hello'
ret = re.match('he',text)
print(ret.group())
>> he
```

以上便可以在`hello`中，匹配出`he`。

点（.）匹配任意的字符：

```
text = "ab"
ret = re.match('.',text)
print(ret.group())
>> a
```

但是点（.）不能匹配不到换行符。示例代码如下：

```
text = "ab"
ret = re.match('.',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute 'group'
```

\d匹配任意的数字：

```
text = "123"
```

```
ret = re.match('\d',text)
print(ret.group())
>> 1
```

\D匹配任意的非数字：

```
text = "a"
ret = re.match('\D',text)
print(ret.group())
>> a
```

而如果text是等于一个数字，那么就匹配不成功了。示例代码如下：

```
text = "1"
ret = re.match('\D',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute 'group'
```

\s匹配的是空白字符（包括：\n，\t，\r和空格）：

```
text = "\t"
ret = re.match('\s',text)
print(ret.group())
>> 空白
```

\w匹配的是 `a-z` 和 `A-Z` 以及数字和下划线：

```
text = "_"
ret = re.match('\w',text)
print(ret.group())
>> _
```

而如果要匹配一个其他的字符，那么就匹配不到。示例代码如下：

```
text = "+"
ret = re.match('\w',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute
```

\W匹配的是和\w相反的：

```
text = "+"
ret = re.match('\W',text)
print(ret.group())
>> +
```

而如果你的text是一个下划线或者英文字符，那么就匹配不到了。示例代码如下：

```
text = "_"
ret = re.match('\W',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute
```

[]组合的方式，只要满足中括号中的某一项都算匹配成功：

```
text = "0731-88888888"
ret = re.match('[\d\-]+',text)
print(ret.group())
>> 0731-88888888
```

之前讲到的几种匹配规则，其实可以使用中括号的形式来进行替代：

- \d：[0-9]
- \D：0-9
- \w：[0-9a-zA-Z_]
- \W：[^0-9a-zA-Z_]

匹配多个字符：

1. `*`：可以匹配0或者任意多个字符。示例代码如下：

   ```
   text = "0731"
   ret = re.match('\d*',text)
   print(ret.group())
   >> 0731
   ```

   以上因为匹配的要求是`\d`，那么就要求是数字，后面跟了一个星号，就可以匹配到0731这四个字符。

2. `+`：可以匹配1个或者多个字符。最少一个。示例代码如下：

   ```
   text = "abc"
   ret = re.match('\w+',text)
   print(ret.group())
   ```

```
>> abc
```

因为匹配的是 `\w` ，那么就要求是英文字符，后面跟了一个加号，意味着最少要有一个满足 `\w` 的字符才能够匹配到。如果text是一个空白字符或者是一个不满足\w的字符，那么就会报错。示例代码如下：

```
text = ""
ret = re.match('\w+',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute
```

3. `?` ：匹配的字符可以出现一次或者不出现（0或者1）。示例代码如下：

```
text = "123"
ret = re.match('\d?',text)
print(ret.group())
>> 1
```

4. `{m}` ：匹配m个字符。示例代码如下：

```
text = "123"
ret = re.match('\d{2}',text)
print(ret.group())
>> 12
```

5. `{m,n}` ：匹配m-n个字符。在这中间的字符都可以匹配到。示例代码如下：

```
text = "123"
ret = re.match('\d{1,2}',text)
prit(ret.group())
>> 12
```

如果text只有一个字符，那么也可以匹配出来。示例代码如下：

```
text = "1"
ret = re.match('\d{1,2}',text)
prit(ret.group())
>> 1
```

小案例：

1. 验证手机号码：手机号码的规则是以 `1` 开头，第二位可以是 `34587` ，后面那9位就可以随意了。示例代码如下：

```
text = "18570631587"
ret = re.match('1[34587]\d{9}',text)
print(ret.group())
>> 18570631587
```

而如果是个不满足条件的手机号码。那么就匹配不到了。示例代码如下：

```
text = "1857063158"
ret = re.match('1[34587]\d{9}',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute
```

2. 验证邮箱：邮箱的规则是邮箱名称是用`数字、数字、下划线`组成的，然后是`@`符号，后面就是域名了。示例代码如下：

```
text = "hynever@163.com"
ret = re.match('\w+@\w+\.[a-zA-Z\.]+',text)
print(ret.group())
```

3. 验证URL：URL的规则是前面是`http`或者`https`或者是`ftp`然后再加上一个冒号，再加上一个斜杠，再后面就是可以出现任意非空白字符了。示例代码如下：

```
text = "http://www.baidu.com/"
ret = re.match('(http|https|ftp)://[^\s]+',text)
print(ret.group())
```

4. 验证身份证：身份证的规则是，总共有18位，前面17位都是数字，后面一位可以是数字，也可以是小写的x，也可以是大写的X。示例代码如下：

```
text = "3113111890812323X"
ret = re.match('\d{17}[\dxX]',text)
print(ret.group())
```

^（脱字号）：表示以...开始：

```
text = "hello"
ret = re.match('^h',text)
print(ret.group())
```

如果是在中括号中，那么代表的是取反操作.

$：表示以...结束：

```
# 匹配163.com的邮箱
text = "xxx@163.com"
ret = re.search('\w+@163\.com$',text)
print(ret.group())
>> xxx@163.com
```

|：匹配多个表达式或者字符串：

```
text = "hello|world"
ret = re.search('hello',text)
print(ret.group())
>> hello
```

贪婪模式和非贪婪模式：

贪婪模式：正则表达式会匹配尽量多的字符。默认是贪婪模式。
非贪婪模式：正则表达式会尽量少的匹配字符。
示例代码如下：

```
text = "0123456"
ret = re.match('\d+',text)
print(ret.group())
# 因为默认采用贪婪模式，所以会输出0123456
>> 0123456
```

可以改成非贪婪模式，那么就只会匹配到0。示例代码如下：

```
text = "0123456"
ret = re.match('\d+?',text)
print(ret.group())
```

案例：匹配 `0-100` 之间的数字：

```
text = '99'
ret = re.match('[1-9]?\d$|100$',text)
print(ret.group())
>> 99
```

而如果 `text=101` ，那么就会抛出一个异常。示例代码如下：

```
text = '101'
ret = re.match('[1-9]?\d$|100$',text)
```

```
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute 'group'
```

转义字符和原生字符串：

在正则表达式中，有些字符是有特殊意义的字符。因此如果想要匹配这些字符，那么就必须使用反斜杠进行转义。比如 `$` 代表的是以...结尾，如果想要匹配 `$` ，那么就必须使用 `\$` 。示例代码如下：

```
text = "apple price is \$99,orange paice is $88"
ret = re.search('\$(\d+)',text)
print(ret.group())
>> $99
```

原生字符串：
在正则表达式中， `\` 是专门用来做转义的。在Python中 `\` 也是用来做转义的。因此如果想要在普通的字符串中匹配出 `\` ，那么要给出四个 `\` 。示例代码如下：

```
text = "apple \c"
ret = re.search('\\\c',text)
print(ret.group())
```

因此要使用原生字符串就可以解决这个问题：

```
text = "apple \c"
ret = re.search(r'\\c',text)
print(ret.group())
```

---

re模块中常用函数：

match：

从开始的位置进行匹配。如果开始的位置没有匹配到。就直接失败了。示例代码如下：

```
text = 'hello'
ret = re.match('h',text)
print(ret.group())
>> h
```

如果第一个字母不是 `h` ，那么就会失败。示例代码如下：

```
text = 'ahello'
ret = re.match('h',text)
print(ret.group())
>> AttributeError: 'NoneType' object has no attribute 'group'
```

如果想要匹配换行的数据，那么就要传入一个 `flag=re.DOTALL` ，就可以匹配换行符了。示例代码如下：

```
text = "abc\nabc"
ret = re.match('abc.*abc',text,re.DOTALL)
print(ret.group())
```

search：

在字符串中找满足条件的字符。如果找到，就返回。说白了，就是只会找到第一个满足条件的。

```
text = 'apple price $99 orange price $88'
ret = re.search('\d+',text)
print(ret.group())
>> 99
```

分组：

在正则表达式中，可以对过滤到的字符串进行分组。分组使用圆括号的方式。

1. `group` ：和 `group(0)` 是等价的，返回的是整个满足条件的字符串。
2. `groups` ：返回的是里面的子组。索引从1开始。
3. `group(1)` ：返回的是第一个子组，可以传入多个。
   示例代码如下：

```
text = "apple price is $99,orange price is $10"
ret = re.search(r".*(\$\d+).*(\$\d+)",text)
print(ret.group())
print(ret.group(0))
print(ret.group(1))
print(ret.group(2))
print(ret.groups())
```

findall：

找出所有满足条件的，返回的是一个列表。

```
text = 'apple price $99 orange price $88'
```

```
ret = re.findall('\d+',text)
print(ret)
>> ['99', '88']
```

sub：

用来替换字符串。将匹配到的字符串替换为其他字符串。

```
text = 'apple price $99 orange price $88'
ret = re.sub('\d+','0',text)
print(ret)
>> apple price $0 orange price $0
```

`sub` 函数的案例，获取拉勾网中的数据：

```
html = """
<div>
<p>基本要求：</p>
<p>1、精通HTML5、CSS3、 JavaScript等Web前端开发技术，对html5页面适配充分了解，熟悉不同浏览器间的差异，熟练写出兼容各种浏览器的代码；</p>
<p>2、熟悉运用常见JS开发框架，如JQuery、vue、angular，能快速高效实现各种交互效果；</p>
<p>3、熟悉编写能够自动适应HTML5界面，能让网页格式自动适应各款各大小的手机；</p>
<p>4、利用HTML5相关技术开发移动平台、PC终端的前端页面，实现HTML5模板化；</p>
<p>5、熟悉手机端和PC端web实现的差异，有移动平台web前端开发经验，了解移动互联网产品和行业，有在Android,iOS等平台下HTML5+CSS+JavaScript（或移动JS框架）开发经验者优先考虑；6、良好的沟通能力和团队协作精神，对移动互联网行业有浓厚兴趣，有较强的研究能力和学习能力；</p>
<p>7、能够承担公司前端培训工作，对公司各业务线的前端（HTML5\CSS3）工作进行支撑和指导。</p>
<p><br></p>
<p>岗位职责：</p>
<p>1、利用html5及相关技术开发移动平台、微信、APP等前端页面，各类交互的实现；</p>
<p>2、持续的优化前端体验和页面响应速度，并保证兼容性和执行效率；</p>
<p>3、根据产品需求，分析并给出最优的页面前端结构解决方案；</p>
<p>4、协助后台及客户端开发人员完成功能开发和调试；</p>
<p>5、移动端主流浏览器的适配、移动端界面自适应研发。</p>
</div>
"""

ret = re.sub('</?[a-zA-Z0-9]+>',"",html)
print(ret)
```

split：

使用正则表达式来分割字符串。

```
text = "hello world ni hao"
```

```
ret = re.split('\W',text)
print(ret)
>> ["hello","world","ni","hao"]
```

compile：

对于一些经常要用到的正则表达式，可以使用 `compile` 进行编译，后期再使用的时候可以直接拿过来用，执行效率会更快。而且 `compile` 还可以指定 `flag=re.VERBOSE` ，在写正则表达式的时候可以做好注释。示例代码如下：

```
text = "the number is 20.50"
r = re.compile(r"""
                \d+ # 小数点前面的数字
                \.? # 小数点
                \d* # 小数点后面的数字
                """,re.VERBOSE)
ret = re.search(r,text)
print(ret.group())
```

json文件处理：

什么是json：

JSON(JavaScript Object Notation, JS 对象标记) 是一种轻量级的数据交换格式。它基于ECMAScript (w3c制定的js规范)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。 易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。更多解释请见：https://baike.baidu.com/item/JSON/2462549?fr=aladdin

JSON支持数据格式：

1. 对象（字典）。使用花括号。
2. 数组（列表）。使用方括号。
3. 整形、浮点型、布尔类型还有null类型。
4. 字符串类型（字符串必须要用双引号，不能用单引号）。

多个数据之间使用逗号分开。
注意：**json**本质上就是一个字符串。

字典和列表转JSON：

```
import json

books = [
    {
        'title': '钢铁是怎样练成的',
        'price': 9.8
    },
    {
        'title': '红楼梦',
        'price': 9.9
    }
]

json_str = json.dumps(books,ensure_ascii=False)
print(json_str)
```

因为 `json` 在 `dump` 的时候，只能存放 `ascii` 的字符，因此会将中文进行转义，这时候我们可以使用 `ensure_ascii=False` 关闭这个特性。
在 `Python` 中。只有基本数据类型才能转换成 `JSON` 格式的字符串。也

即：`int`、`float`、`str`、`list`、`dict`、`tuple`。

将json数据直接`dump`到文件中：

`json`模块中除了`dumps`函数，还有一个`dump`函数，这个函数可以传入一个文件指针，直接将字符串`dump`到文件中。示例代码如下：

```
books = [
    {
        'title': '钢铁是怎样练成的',
        'price': 9.8
    },
    {
        'title': '红楼梦',
        'price': 9.9
    }
]
with open('a.json','w') as fp:
    json.dump(books,fp)
```

将一个json字符串load成Python对象：

```
json_str = '[{"title": "钢铁是怎样练成的", "price": 9.8}, {"title": "红楼梦", "price": 9.
9}]'
books = json.loads(json_str,encoding='utf-8')
print(type(books))
print(books)
```

直接从文件中读取json：

```
import json
with open('a.json','r',encoding='utf-8') as fp:
    json_str = json.load(fp)
    print(json_str)
```

csv文件处理

读取csv文件：

```
import csv

with open('stock.csv','r') as fp:
    reader = csv.reader(fp)
    titles = next(reader)
    for x in reader:
        print(x)
```

这样操作，以后获取数据的时候，就要通过下表来获取数据。如果想要在获取数据的时候通过标题来获取。那么可以使用 `DictReader` 。示例代码如下：

```
import csv

with open('stock.csv','r') as fp:
    reader = csv.DictReader(fp)
    for x in reader:
        print(x['turnoverVol'])
```

写入数据到csv文件：

写入数据到csv文件，需要创建一个 `writer` 对象，主要用到两个方法。一个是 `writerow` ，这个是写入一行。一个是 `writerows` ，这个是写入多行。示例代码如下：

```
import csv

headers = ['name','age','classroom']
values = [
    ('zhiliao',18,'111'),
    ('wena',20,'222'),
    ('bbc',21,'111')
]
with open('test.csv','w',newline='') as fp:
    writer = csv.writer(fp)
    writer.writerow(headers)
    writer.writerows(values)
```

也可以使用字典的方式把数据写入进去。这时候就需要使用 `DictWriter` 了。示例代码如下：

```
import csv

headers = ['name','age','classroom']
values = [
    {"name":'wenn',"age":20,"classroom":'222'},
    {"name":'abc',"age":30,"classroom":'333'}
]
with open('test.csv','w',newline='') as fp:
    writer = csv.DictWriter(fp,headers)
    writer = csv.writeheader()
    writer.writerow({'name':'zhiliao',"age":18,"classroom":'111'})
    writer.writerows(values)
```

Excel文件处理

在爬虫开发中，我们主要关注 `Excel` 文件的读写，不会过多关心 `Excel` 中的一些样式。如果想要读写 `Excel` 文件，需要借助到两个库 `xlrd` 和 `xlwt` ，其中 `xlrd` 是用于读的， `xlwt` 是用于写的，安装命令如下：

```
pip install xlrd
pip install xlwt
```

读取Excel文件：

```
import xlrd

workbook = xlrd.open_workbook("成绩表.xlsx")
sheet_names = workbook.sheet_names()
print(sheet_names) #打印所有的sheet的名称
```

获取 `Sheet` ：

一个 `Excel` 中可能有多个 `Sheet` ，那么可以通过以下方法来获取想要的 `Sheet` 信息：

1. `sheet_names` ：获取所有的 `sheet` 的名字。
2. `sheet_by_index` ：根据索引获取 `sheet` 对象。
3. `sheet_by_name` ：根据名字获取 `sheet` 对象。
4. `sheets` ：获取所有的 `sheet` 对象。
5. `sheet.nrows` ：这个 `sheet` 中的行数。
6. `sheet.ncols` ：这个 `sheet` 中的列数。

示例代码如下：

```
import xlrd

workbook = xlrd.open_workbook("成绩表.xlsx")
sheet_names = workbook.sheet_names()
print(sheet_names) #打印所有的sheet的名称

# 根据索引获取sheet
sheet = workbook.sheet_by_index(0)
print(sheet.name)

# 根据名称获取sheet
sheet = workbook.sheet_by_name("1班成绩")
```

```
print(sheet.name)

# 获取所有的sheet对象
sheets = workbook.sheets()
for sheet in sheets:
    print(sheet.name)

# 获取这个sheet中的行数和列数
nrows = sheet.nrows
ncols = sheet.ncols
```

获取Cell及其属性：

每个 `Cell` 代表的是表格中的一格。以下方法可以方便获取想要的 `cell` ：

1. `sheet.cell(row,col)` ：获取指定行和列的 `cell` 对象。
2. `sheet.row_slice(row,start_col,end_col)` ：获取指定行的某几列的cell对象。
3. `sheet.col_slice(col,start_row,end_row)` ：获取指定列的某几行的cell对象。
4. `sheet.cell_value(row,col)` ：获取指定行和列的值。
5. `sheet.row_values(row,start_col,end_col)` ：获取指定行的某几列的值。
6. `sheet.col_values(col,start_row,end_row)` ：获取指定列的某几行的值。

示例代码如下：

```
sheet = workbook.sheet_by_index(0)

# 使用cell方法获取指定的cell对象
for col in range(sheet.ncols):
    for row in range(sheet.nrows):
        print(sheet.cell(row,col))

# 使用row_slice获取第0行的1-2列的cell对象
cells = sheet.row_slice(0,1,3)
# 使用col_slice获取第0列的1-2行的cell对象
cells = sheet.col_slice(0,1,3)
```

另外在 `Cell` 上面也有一些常用的属性：

1. `cell.value` ：这个 `cell` 里面的值。
2. `cell.ctype` ：这个 `cell` 的数据类型。

Cell的数据类型：

1. `xlrd.XL_CELL_TEXT（Text）` ：文本类型。
2. `xlrd.XL_CELL_NUMBER（Number）` ：数值类型。
3. `xlrd.XL_CELL_DATE（Date）` ：日期时间类型。

4. `xlrd.XL_CELL_BOOLEAN（Bool）`：布尔类型。
5. `xlrd.XL_CELL_BLANK`：空白数据类型。

写入Excel：

写入 `Excel` 步骤如下：

1. 导入 `xlwt` 模块。
2. 创建一个 `Workbook` 对象。
3. 创建一个 `Sheet` 对象。
4. 使用 `sheet.write(row,col,data)` 方法把数据写入到 `Sheet` 下指定行和列中。如果想要在原来 `workbook` 对象上添加新的 `cell`，那么需要调用 `put_cell` 来添加。
5. 保存成 `Excel` 文件。

示例代码如下：

```
import xlwt
import random

workbook = xlwt.Workbook(encoding='utf-8')
sheet = workbook.add_sheet("成绩表")
# 添加表头
fields = ['数学','英语','语文']
for index,field in enumerate(fields):
    sheet.write(0,index,field)

# 随机的添加成绩
for row in range(1,10):
    for col in range(3):
        grade = random.randint(0,100)
        sheet.write(row,col,grade)

workbook.save("abc.xls")
```

另外，如果想要在原来已经存在的 `Excel` 文件中添加新的行或者新的列，那么需要采用 `put_cell(row,col,type,value,xf_index)` 来添加进去，最后再放到 `xlwt` 创建的 `workbook` 中，然后再保存进去。示例代码如下：

```
import xlrd
import xlwt

workbook = xlrd.open_workbook("成绩表.xlsx")
rsheet = workbook.sheet_by_index(0)
```

```
# 添加总分成绩
rsheet.put_cell(0,4,xlrd.XL_CELL_TEXT,"总分",None)
for row in range(1,rsheet.nrows):
    grade = sum(rsheet.row_values(row,1,4))
    rsheet.put_cell(row,4,xlrd.XL_CELL_NUMBER,grade,None)

# 添加每个科目的平均成绩
total_rows = rsheet.nrows
total_cols = rsheet.ncols
for col in range(1,total_cols):
    grades = rsheet.col_values(col,1,total_rows)
    avg_grade = sum(grades)/len(grades)
    print(type(avg_grade))
    rsheet.put_cell(total_rows,col,xlrd.XL_CELL_NUMBER,avg_grade,None)

# 重新写入一个新的excel文件数据
wwb = xlwt.Workbook(encoding="utf-8")
wsheet = wwb.add_sheet("1班学生成绩")
for row in range(rsheet.nrows):
    for col in range(rsheet.ncols):
        wsheet.write(row,col,rsheet.cell_value(row,col))

wwb.save("abc.xls")
```

MySQL数据库操作

安装mysql：

1. 在官网：https://dev.mysql.com/downloads/windows/installer/5.7.html
2. 如果提示没有 `.NET Framework` 框架。那么就在提示框中找到下载链接，下载一个就可以了。
3. 如果提示没有 `Microsoft Virtual C++ x64(x86)` ，那么百度或者谷歌这个软件安装即可。
4. 如果没有找到。那么私聊我。

navicat：

navicat是一个操作mysql数据库非常方便的软件。使用他操作数据库，就跟使用excel操作数据是一样的。

安装驱动程序：

Python要想操作MySQL。必须要有一个中间件，或者叫做驱动程序。驱动程序有很多。比如有 `mysqldb` 、 `mysqlclient` 、 `pymysql` 等。在这里，我们选择用 `pymysql` 。安装方式也是非常简单，通过命令 `pip install pymysql` 即可安装。

数据库连接：

数据库连接之前。首先先确认以下工作完成，这里我们以一个 `pymysql_test` 数据库.以下将介绍连接 `mysql` 的示例代码：

```
import pymysql

db = pymysql.connect(
    host="127.0.0.1",
    user='root',
    password='root',
    database='pymysql_test',
    port=3306
)
cursor = db.cursor()
cursor.execute("select 1")
data = cursor.fetchone()
print(data)
db.close()
```

插入数据：

```
import pymysql

db = pymysql.connect(
    host="127.0.0.1",
    user='root',
    password='root',
    database='pymysql_test',
    port=3306
)
cursor = db.cursor()
sql = """
insert into user(
    id,username,gender,age,password
  )
  values(null,'abc',1,18,'111111');
"""
cursor.execute(sql)
db.commit()
db.close()
```

如果在数据还不能保证的情况下，可以使用以下方式来插入数据：

```
sql = """
insert into user(
    id,username,gender,age,password
  )
  values(null,%s,%s,%s,%s);
"""

cursor.execute(sql,('spider',1,20,'222222'))
```

查找数据：

使用 `pymysql` 查询数据。可以使用 `fetch*` 方法。

1. `fetchone()`：这个方法每次之获取一条数据。
2. `fetchall()`：这个方法接收全部的返回结果。
3. `fetchmany(size)`：可以获取指定条数的数据。
   示例代码如下：

```
cursor = db.cursor()

sql = """
select * from user
"""
```

```
cursor.execute(sql)
while True:
    result = cursor.fetchone()
    if not result:
        break
    print(result)
db.close()
```

或者是直接使用 `fetchall`，一次性可以把所有满足条件的数据都取出来：

```
cursor = db.cursor()

sql = """
select * from user
"""

cursor.execute(sql)
results = cursor.fetchall()
for result in results:
    print(result)
db.close()
```

或者是使用 `fetchmany`，指定获取多少条数据：

```
cursor = db.cursor()

sql = """
select * from user
"""

cursor.execute(sql)
results = cursor.fetchmany(1)
for result in results:
    print(result)
db.close()
```

删除数据：

```
cursor = db.cursor()

sql = """
delete from user where id=1
"""
```

```
cursor.execute(sql)
db.commit()
db.close()
```

更新数据：

```
conn = pymysql.connect(host='localhost',user='root',password='root',database='pymysql_demo',port=3306)
cursor = conn.cursor()

sql = """
update user set username='aaa' where id=1
"""
cursor.execute(sql)
conn.commit()

conn.close()
```

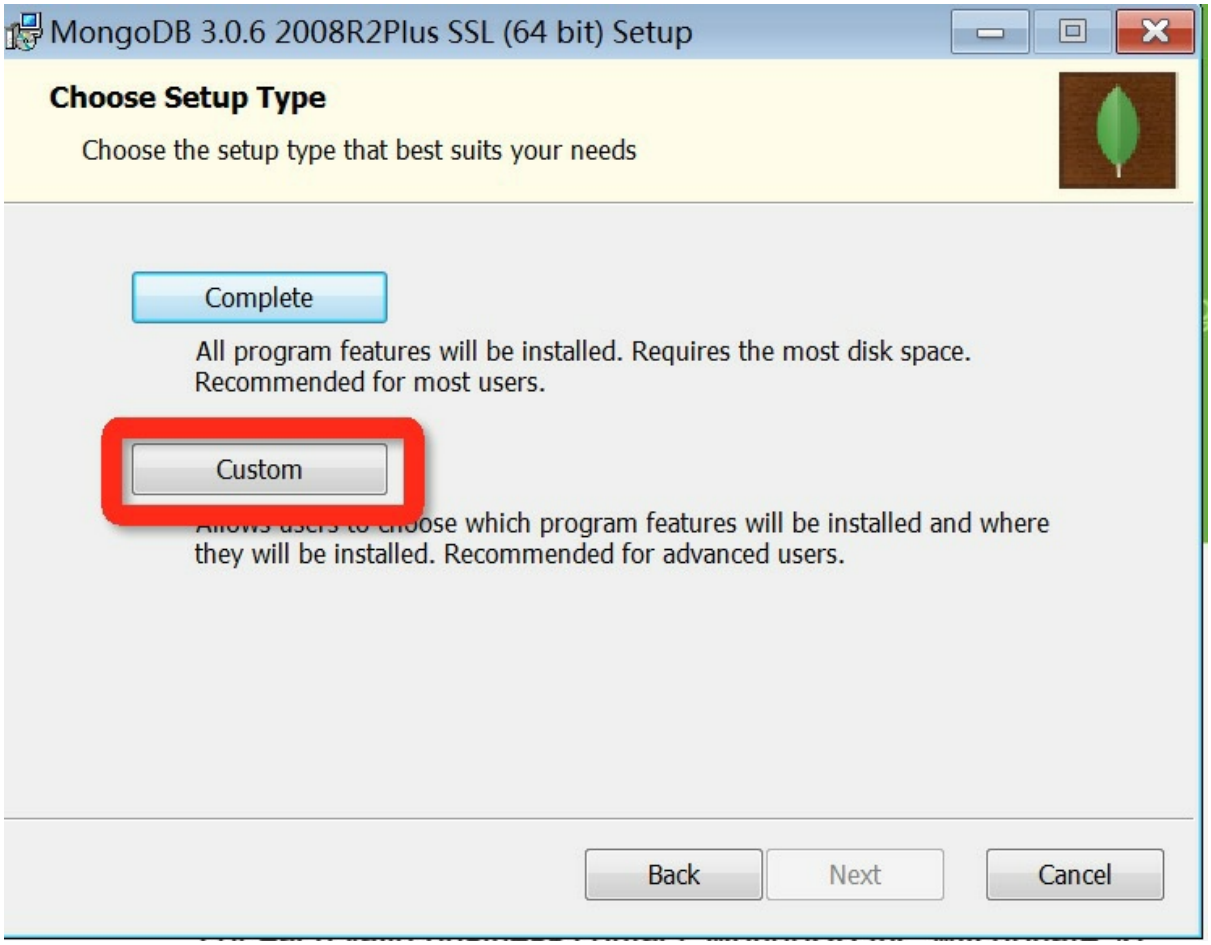
MongoDB数据库操作

`MongoDB` 是一个基于分布式文件存储的 `NoSQL` 数据库。在处理海量数据的时候会比 `MySQL` 更有优势。爬虫如果上了一个量级，可能就会比较推荐使用 `MongoDB` ，当然没有上量的数据也完全可以使用 `MongoDB` 来存储数据。因此学会使用 `MongoDB` 也是爬虫开发工程师必须掌握的一个技能。

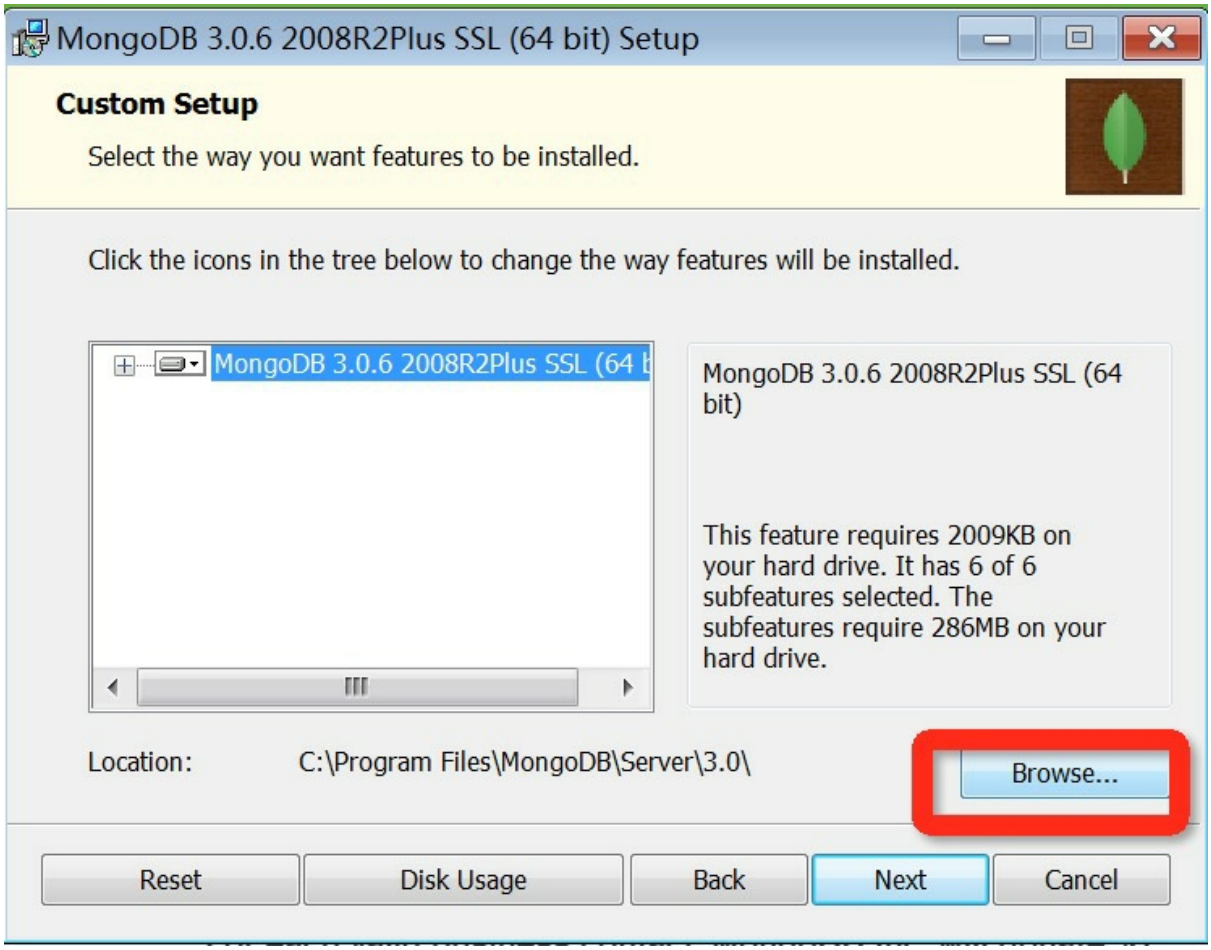
`Windows` 下安装 `MongoDB` 数据库：

在官网下载 `MongoDB` 数据库。是一个 `msi` 文件。官网地址如下：https://www.mongodb.com/download-center?ct=atlasheader#community

然后双击进行安装：



选择安装的地址。不要包含中文、空格。



在安装的时候选择安装 `Compass` 。这个软件是用图形化的界面操作 `MongoDB` ，使用起来非常方便。

运行 `MongoDB` ：

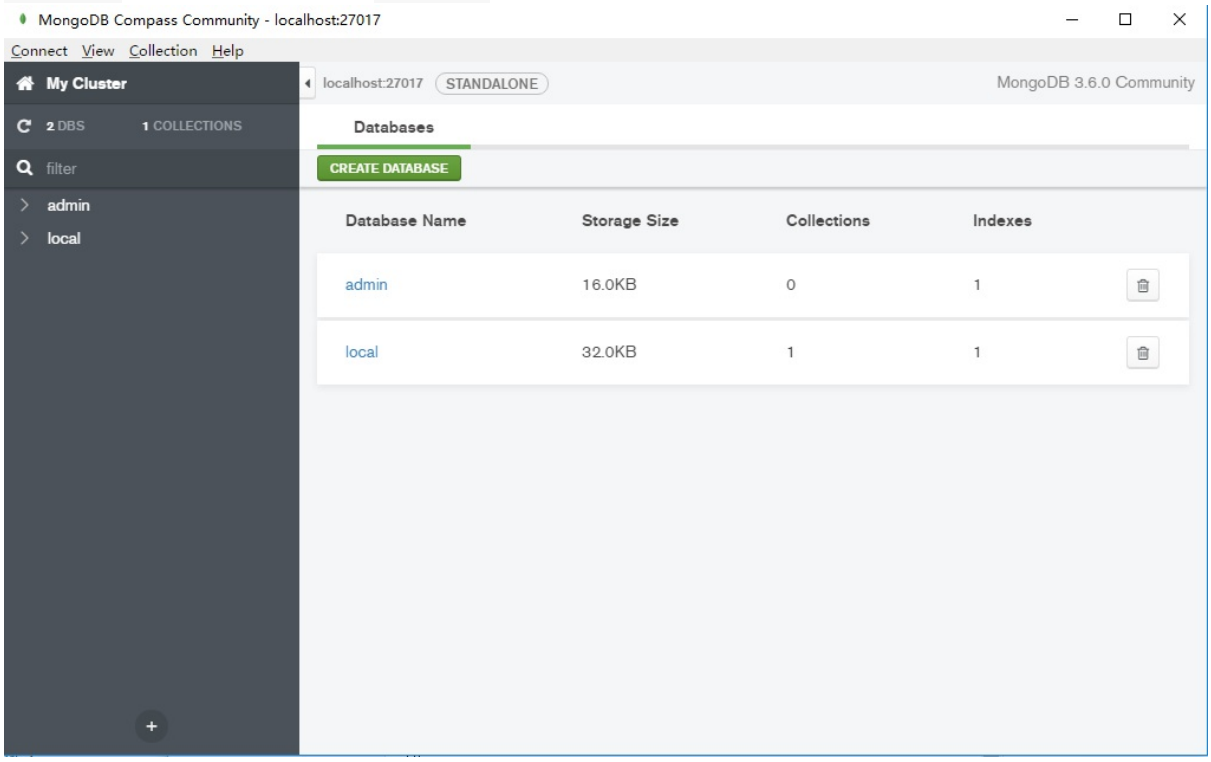
1. 创建数据目录： 启动 `MongoDB` 之前，首先要给他指定一个数据存储的路径。比如我在 `MongoDB` 的安装路径下创建一个 `data` 文件夹，专门用来存储数据的。 `D:\ProgramApp\mongodb\data` 。
2. 把 `mongodb` 的 `bin` 目录加入到环境变量中。方便后期调用。
3. 执行命令 `mongod --dbpath D:\ProgramApp\mongodb\data` 启动。

连接 `MongoDB` ：

在环境变量设置好的前提下，使用以下命令 `mongo` 就可以进入到 `mongo` 的操作终端了。

使用 `Compass` 软件连接 `MongoDB` ：

`Compass` 是一个图形化的操作 `MongoDB` 的客户端。使用他来操作会更加方便。



将 `MongoDB` 制作成 `windows` 服务：

启动 `mongodb` 后，如果想让 `mongodb` 一直运行，那么这个终端便不能关闭，而且每次运行的时候还需要指定 `data` 的路径。因此我们可以将 `mongodb` 制作成一个服务，以后就通过一行命令就可以运行了。以下将讲解如何制作服务：

1. 创建配置文件：在 `mongodb` 安装的路径下创建配置文件 `mongod.cfg` （路径和名字不是必须和我这的一样），然后在配置文件中添加以下代码：

   ```
   logpath=D:\ProgramApp\mongodb\data\log\mongod.log
   dbpath=D:\ProgramApp\mongodb\data\db
   ```

   `logpath` 是日志的路径。 `dbpath` 是 `mongodb` 数据库的存储路径。

2. 安装 `mongodb` 服务：
   使用以下命令即可将 `mongodb` 安装成一个服务：

   ```
   mongod --config "cfg配置文件所在路径" --install
   比如：
   mongod --config "D:\ProgramApp\mongodb\mongod.cfg" --install
   ```

3. 启动 `mongodb` ： `net start mongodb` 。

4. 关闭 `mongodb` ： `net stop mongodb` 。

5. 移除 `mongodb`： `"D:\ProgramApp\mongodb\bin\mongod.exe" --remove` 。

MongoDB 概念介绍：

SQL术语/概念	MongoDB术语/概念	解释/说明
database	database	数据库
table	collection	数据库表/集合
row	document	数据记录行/文档
column	field	数据字段/域
index	index	索引
joins	joins	表连接,MongoDB不支持
primary key	primary key	主键,MongoDB自动将_id字段设置为主键

MongoDB 三元素：

三元素：数据库、集合、文档。

1. 文档（document）：就是关系型数据库中的一行。文档是一个对象，由键值对构成，是 `json` 的扩展形式：

```
{'name':'abc','gender':'1'}
```

2. 集合（collection）：就是关系型数据库中的表。可以存储多个文档，结构可以不固定。如可以存储如下文档在一个集合中：

```
{"name":"abc","gender":"1"}
{"name":"xxx","age":18}
{"title":'yyy','price':20.9}
```

MongoDB基本操作命令：

1. `db`：查看当前的数据库。
2. `show dbs`：查看所有的数据库。
3. `use 数据库名`：切换数据库。如果数据库不存在，则创建一个。（创建完成后需要插入数据库才算创建成功）
4. `db.dropDatabase()`：删除当前指向的数据库。
5. `db.集合名.insert(value)`：添加数据到指定的集合中。
6. `db.集合名.find()`：从指定的集合中查找数据。 更多命令请见：http://www.runoob.com/mongodb/mongodb-tutorial.html

Python操作 MongoDB：

安装 pymongo：

要用 python 操作 mongodb，必须下载一个驱动程序，这个驱动程序就是 pymongo：

```
pip install pymongo
```

连接 MongoDB：

```
import pymongo
# 获取连接的对象
client = pymongo.MongoClient('127.0.0.1',port=27017)
# 获取数据库
db = client.zhihu
# 获取集合（表）
collection = db.qa
# 插入一条数据到集合中
collection.insert_one({
    "username":"abc",
    "password":'hello'
})
```

数据类型：

类型	说明
Object ID	文档ID
String	字符串，最常用，必须是有效的UTF-8
Boolean	存储一个布尔值，true或false
Integer	整数可以是32位或64位，这取决于服务器
Double	存储浮点值
Arrays	数组或列表，多个值存储到一个键
Object	用于嵌入式的文档，即一个值为一个文档
Null	存储Null值
Timestamp	时间戳，表示从1970-1-1到现在的总秒数
Date	存储当前日期或时间的UNIX时间格式

操作 MongoDB：

操作 `MongoDB` 的主要方法如下：

1. `insert_one`：加入一条文档数据到集合中。示例代码如下：

```
collection.insert_one({
    "username":"abc",
    "password":'hello'
})
```

2. `insert_many`：加入多条文档数据到集合中。

```
collection.insert_many([
    {
        "username":'abc',
        'password':'111111'
    },
    {
        "username":'bbb',
        'password':'222222'
    },
])
```

3. `find_one`：查找一条文档对象。

```
result = collection.find_one()
print(result)
# 或者是指定条件
result = collection.find_one({"username":"abc"})
print(result)
```

4. `update_one`：更新一条文档对象。

```
collection.update_one({"username":"abc"},{"$set":{"username":"aaa"}})
```

5. `update_many`：更新多条文档对象。

```
collection.update_many({"username":"abc"},{"$set":{"username":"aaa"}})
```

6. `delete_one`：删除一条文档对象。

```
collection.delete_one({"username":"abc"})
```

7. `delete_many`：删除多条文档对象。

```
collection.delete_one({"username":"abc"})
```

多线程爬虫

有些时候，比如下载图片，因为下载图片是一个耗时的操作。如果采用之前那种同步的方式下载。那效率肯会特别慢。这时候我们就可以考虑使用多线程的方式来下载图片。

多线程介绍：

多线程是为了同步完成多项任务，通过提高资源使用效率来提高系统的效率。线程是在同一时间需要完成多项任务的时候实现的。
最简单的比喻多线程就像火车的每一节车厢，而进程则是火车。车厢离开火车是无法跑动的，同理火车也可以有多节车厢。多线程的出现就是为了提高效率。同时它的出现也带来了一些问题。更多介绍请参考：https://baike.baidu.com/item/多线程/1190404?fr=aladdin

threading模块介绍：

`threading` 模块是 `python` 中专门提供用来做多线程编程的模块。 `threading` 模块中最常用的类是 `Thread` 。以下看一个简单的多线程程序：

```
import threading
import time

def coding():
    for x in range(3):
        print('%s正在写代码' % x)
        time.sleep(1)

def drawing():
    for x in range(3):
        print('%s正在画图' % x)
        time.sleep(1)


def single_thread():
    coding()
    drawing()

def multi_thread():
    t1 = threading.Thread(target=coding)
    t2 = threading.Thread(target=drawing)

    t1.start()
    t2.start()

if __name__ == '__main__':
```

```
    multi_thread()
```

查看线程数：

使用 `threading.enumerate()` 函数便可以看到当前线程的数量。

查看当前线程的名字：

使用 `threading.current_thread()` 可以看到当前线程的信息。

继承自 `threading.Thread` 类：

为了让线程代码更好的封装。可以使用 `threading` 模块下的 `Thread` 类，继承自这个类，然后实现 `run` 方法，线程就会自动运行 `run` 方法中的代码。示例代码如下：

```
import threading
import time

class CodingThread(threading.Thread):
    def run(self):
        for x in range(3):
            print('%s正在写代码' % threading.current_thread())
            time.sleep(1)

class DrawingThread(threading.Thread):
    def run(self):
        for x in range(3):
            print('%s正在画图' % threading.current_thread())
            time.sleep(1)

def multi_thread():
    t1 = CodingThread()
    t2 = DrawingThread()

    t1.start()
    t2.start()

if __name__ == '__main__':
    multi_thread()
```

多线程共享全局变量的问题：

多线程都是在同一个进程中运行的。因此在进程中的全局变量所有线程都是可共享的。这就造成了一个问题，因为线程执行的顺序是无序的。有可能会造成数据错误。比如以下代码：

```
import threading

tickets = 0

def get_ticket():
    global tickets
    for x in range(1000000):
        tickets += 1
    print('tickets:%d'%tickets)

def main():
    for x in range(2):
        t = threading.Thread(target=get_ticket)
        t.start()

if __name__ == '__main__':
    main()
```

以上结果正常来讲应该是6，但是因为多线程运行的不确定性。因此最后的结果可能是随机的。

锁机制：

为了解决以上使用共享全局变量的问题。`threading`提供了一个`Lock`类，这个类可以在某个线程访问某个变量的时候加锁，其他线程此时就不能进来，直到当前线程处理完后，把锁释放了，其他线程才能进来处理。示例代码如下：

```
import threading

VALUE = 0

gLock = threading.Lock()

def add_value():
    global VALUE
    gLock.acquire()
    for x in range(1000000):
        VALUE += 1
    gLock.release()
    print('value: %d'%VALUE)

def main():
    for x in range(2):
        t = threading.Thread(target=add_value)
        t.start()

if __name__ == '__main__':
```

```
    main()
```

Lock版本生产者和消费者模式：

生产者和消费者模式是多线程开发中经常见到的一种模式。生产者的线程专门用来生产一些数据，然后存放到一个中间的变量中。消费者再从这个中间的变量中取出数据进行消费。但是因为要使用中间变量，中间变量经常是一些全局变量，因此需要使用锁来保证数据完整性。以下是使用 `threading.Lock` 锁实现的“生产者与消费者模式”的一个例子：

```
import threading
import random
import time

gMoney = 1000
gLock = threading.Lock()
# 记录生产者生产的次数，达到10次就不再生产
gTimes = 0

class Producer(threading.Thread):
    def run(self):
        global gMoney
        global gLock
        global gTimes
        while True:
            money = random.randint(100, 1000)
            gLock.acquire()
            # 如果已经达到10次了，就不再生产了
            if gTimes >= 10:
                gLock.release()
                break
            gMoney += money
            print('%s当前存入%s元钱，剩余%s元钱' % (threading.current_thread(), money, gM
oney))
            gTimes += 1
            time.sleep(0.5)
            gLock.release()

class Consumer(threading.Thread):
    def run(self):
        global gMoney
        global gLock
        global gTimes
        while True:
            money = random.randint(100, 500)
            gLock.acquire()
            if gMoney > money:
```

```
                gMoney -= money
                print('%s当前取出%s元钱，剩余%s元钱' % (threading.current_thread(), money
, gMoney))
                time.sleep(0.5)
            else:
                # 如果钱不够了，有可能是已经超过了次数，这时候就判断一下
                if gTimes >= 10:
                    gLock.release()
                    break
                print("%s当前想取%s元钱，剩余%s元钱，不足！" % (threading.current_thread(
),money,gMoney))
            gLock.release()

def main():
    for x in range(5):
        Consumer(name='消费者线程%d'%x).start()

    for x in range(5):
        Producer(name='生产者线程%d'%x).start()

if __name__ == '__main__':
    main()
```

Condition版的生产者与消费者模式：

`Lock` 版本的生产者与消费者模式可以正常的运行。但是存在一个不足，在消费者中，总是通过 `while True` 死循环并且上锁的方式去判断钱够不够。上锁是一个很耗费CPU资源的行为。因此这种方式不是最好的。还有一种更好的方式便是使用 `threading.Condition` 来实现。 `threading.Condition` 可以在没有数据的时候处于阻塞等待状态。一旦有合适的数据了，还可以使用 `notify` 相关的函数来通知其他处于等待状态的线程。这样就可以不用做一些无用的上锁和解锁的操作。可以提高程序的性能。首先对 `threading.Condition` 相关的函数做个介绍， `threading.Condition` 类似 `threading.Lock` ，可以在修改全局数据的时候进行上锁，也可以在修改完毕后进行解锁。以下将一些常用的函数做个简单的介绍：

1. `acquire` ：上锁。
2. `release` ：解锁。
3. `wait` ：将当前线程处于等待状态，并且会释放锁。可以被其他线程使用 `notify` 和 `notify_all` 函数唤醒。被唤醒后会继续等待上锁，上锁后继续执行下面的代码。
4. `notify` ：通知某个正在等待的线程，默认是第1个等待的线程。
5. `notify_all` ：通知所有正在等待的线程。 `notify` 和 `notify_all` 不会释放锁。并且需要在 `release` 之前调用。

`Condition` 版的生产者与消费者模式代码如下：

```
import threading
import random
import time

gMoney = 1000
gCondition = threading.Condition()
gTimes = 0
gTotalTimes = 5

class Producer(threading.Thread):
    def run(self):
        global gMoney
        global gCondition
        global gTimes
        while True:
            money = random.randint(100, 1000)
            gCondition.acquire()
            if gTimes >= gTotalTimes:
                gCondition.release()
                print('当前生产者总共生产了%s次'%gTimes)
                break
            gMoney += money
            print('%s当前存入%s元钱，剩余%s元钱' % (threading.current_thread(), money, gM
oney))
            gTimes += 1
            time.sleep(0.5)
            gCondition.notify_all()
            gCondition.release()

class Consumer(threading.Thread):
    def run(self):
        global gMoney
        global gCondition
        while True:
            money = random.randint(100, 500)
            gCondition.acquire()
            # 这里要给个while循环判断，因为等轮到这个线程的时候
            # 条件有可能又不满足了
            while gMoney < money:
                if gTimes >= gTotalTimes:
                    gCondition.release()
                    return
                print('%s准备取%s元钱，剩余%s元钱，不足！'%(threading.current_thread(),mo
ney,gMoney))
                gCondition.wait()
            gMoney -= money
            print('%s当前取出%s元钱，剩余%s元钱' % (threading.current_thread(), money, gM
oney))
```

```
            time.sleep(0.5)
            gCondition.release()

def main():
    for x in range(5):
        Consumer(name='消费者线程%d'%x).start()

    for x in range(2):
        Producer(name='生产者线程%d'%x).start()

if __name__ == '__main__':
    main()
```

Queue线程安全队列：

在线程中，访问一些全局变量，加锁是一个经常的过程。如果你是想把一些数据存储到某个队列中，那么Python内置了一个线程安全的模块叫做 `queue` 模块。Python中的queue模块中提供了同步的、线程安全的队列类，包括FIFO（先进先出）队列Queue，LIFO（后入先出）队列LifoQueue。这些队列都实现了锁原语（可以理解为原子操作，即要么不做，要么都做完），能够在多线程中直接使用。可以使用队列来实现线程间的同步。相关的函数如下：

1. 初始化Queue(maxsize)：创建一个先进先出的队列。
2. qsize()：返回队列的大小。
3. empty()：判断队列是否为空。
4. full()：判断队列是否满了。
5. get()：从队列中取最后一个数据。
6. put()：将一个数据放到队列中。

使用生产者与消费者模式多线程下载表情包：

```
import threading
import requests
from lxml import etree
from urllib import request
import os
import re
from queue import Queue

class Producer(threading.Thread):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/62.0.3202.94 Safari/537.36'
    }
    def __init__(self,page_queue,img_queue,*args,**kwargs):
        super(Producer, self).__init__(*args,**kwargs)
```

```
        self.page_queue = page_queue
        self.img_queue = img_queue


    def run(self):
        while True:
            if self.page_queue.empty():
                break
            url = self.page_queue.get()
            self.parse_page(url)

    def parse_page(self,url):
        response = requests.get(url,headers=self.headers)
        text = response.text
        html = etree.HTML(text)
        imgs = html.xpath("//div[@class='page-content text-center']//a//img")
        for img in imgs:
            if img.get('class') == 'gif':
                continue
            img_url = img.xpath(".//@data-original")[0]
            suffix = os.path.splitext(img_url)[1]
            alt = img.xpath(".//@alt")[0]
            alt = re.sub(r'[，。？?,/\\·]','',alt)
            img_name = alt + suffix
            self.img_queue.put((img_url,img_name))

class Consumer(threading.Thread):
    def __init__(self,page_queue,img_queue,*args,**kwargs):
        super(Consumer, self).__init__(*args,**kwargs)
        self.page_queue = page_queue
        self.img_queue = img_queue

    def run(self):
        while True:
            if self.img_queue.empty():
                if self.page_queue.empty():
                    return
            img = self.img_queue.get(block=True)
            url,filename = img
            request.urlretrieve(url,'images/'+filename)
            print(filename+'  下载完成！')

def main():
    page_queue = Queue(100)
    img_queue = Queue(500)
    for x in range(1,101):
        url = "http://www.doutula.com/photo/list/?page=%d" % x
        page_queue.put(url)
```

```
    for x in range(5):
        t = Producer(page_queue,img_queue)
        t.start()

    for x in range(5):
        t = Consumer(page_queue,img_queue)
        t.start()

if __name__ == '__main__':
    main()
```

GIL全局解释器锁：

Python自带的解释器是 `CPython` 。 `CPython` 解释器的多线程实际上是一个假的多线程（在多核CPU中，只能利用一核，不能利用多核）。同一时刻只有一个线程在执行，为了保证同一时刻只有一个线程在执行，在 `CPython` 解释器中有一个东西叫做 `GIL（Global Intepreter Lock）` ，叫做全局解释器锁。这个解释器锁是有必要的。因为 `CPython` 解释器的内存管理不是线程安全的。当然除了 `CPython` 解释器，还有其他的解释器，有些解释器是没有 `GIL` 锁的，见下面：

1. `Jython` ：用Java实现的Python解释器。不存在GIL锁。更多详情请见：https://zh.wikipedia.org/wiki/Jython
2. `IronPython` ：用 `.net` 实现的Python解释器。不存在GIL锁。更多详情请见：https://zh.wikipedia.org/wiki/IronPython
3. `PyPy` ：用 `Python` 实现的Python解释器。存在GIL锁。更多详情请见：https://zh.wikipedia.org/wiki/PyPy
   GIL虽然是一个假的多线程。但是在处理一些IO操作（比如文件读写和网络请求）还是可以在很大程度上提高效率的。在IO操作上建议使用多线程提高效率。在一些CPU计算操作上不建议使用多线程，而建议使用多进程。

多线程下载百思不得姐段子作业：

```
import requests
from lxml import etree
import threading
from queue import Queue
import csv


class BSSpider(threading.Thread):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/62.0.3202.94 Safari/537.36'
    }
    def __init__(self,page_queue,joke_queue,*args,**kwargs):
```

```
        super(BSSpider, self).__init__(*args,**kwargs)
        self.base_domain = 'http://www.budejie.com'
        self.page_queue = page_queue
        self.joke_queue = joke_queue

    def run(self):
        while True:
            if self.page_queue.empty():
                break
            url = self.page_queue.get()
            response = requests.get(url, headers=self.headers)
            text = response.text
            html = etree.HTML(text)
            descs = html.xpath("//div[@class='j-r-list-c-desc']")
            for desc in descs:
                jokes = desc.xpath(".//text()")
                joke = "\n".join(jokes).strip()
                link = self.base_domain+desc.xpath(".//a/@href")[0]
                self.joke_queue.put((joke,link))
            print('='*30+"第%s页下载完成! "%url.split('/')[-1]+"="*30)

class BSWriter(threading.Thread):
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/62.0.3202.94 Safari/537.36'
    }

    def __init__(self, joke_queue, writer,gLock, *args, **kwargs):
        super(BSWriter, self).__init__(*args, **kwargs)
        self.joke_queue = joke_queue
        self.writer = writer
        self.lock = gLock

    def run(self):
        while True:
            try:
                joke_info = self.joke_queue.get(timeout=40)
                joke,link = joke_info
                self.lock.acquire()
                self.writer.writerow((joke,link))
                self.lock.release()
                print('保存一条')
            except:
                break

def main():
    page_queue = Queue(10)
    joke_queue = Queue(500)
    gLock = threading.Lock()
```

```
    fp = open('bsbdj.csv', 'a',newline='', encoding='utf-8')
    writer = csv.writer(fp)
    writer.writerow(('content', 'link'))

    for x in range(1,11):
        url = 'http://www.budejie.com/text/%d' % x
        page_queue.put(url)

    for x in range(5):
        t = BSSpider(page_queue,joke_queue)
        t.start()

    for x in range(5):
        t = BSWriter(joke_queue,writer,gLock)
        t.start()

if __name__ == '__main__':
    main()
```

动态网页数据抓取

什么是**AJAX**：

AJAX（Asynchronouse JavaScript And XML）异步JavaScript和XML。过在后台与服务器进行少量数据交换，Ajax 可以使网页实现异步更新。这意味着可以在不重新加载整个网页的情况下，对网页的某部分进行更新。传统的网页（不使用Ajax）如果需要更新内容，必须重载整个网页页面。因为传统的在传输数据格式方面，使用的是 `XML` 语法。因此叫做 `AJAX` ，其实现在数据交互基本上都是使用 `JSON` 。使用AJAX加载的数据，即使使用了JS，将数据渲染到了浏览器中，在 `右键->查看网页源代码` 还是不能看到通过ajax加载的数据，只能看到使用这个url加载的html代码。

获取**ajax**数据的方式：

1. 直接分析ajax调用的接口。然后通过代码请求这个接口。
2. 使用Selenium+chromedriver模拟浏览器行为获取数据。

方式	优点	缺点
分析接口	直接可以请求到数据。不需要做一些解析工作。代码量少，性能高。	分析接口比较复杂，特别是一些通过js混淆的接口，要有一定的js功底。容易被发现是爬虫。
selenium	直接模拟浏览器的行为。浏览器能请求到的，使用selenium也能请求到。爬虫更稳定。	代码量多。性能低。

Selenium+chromedriver获取动态数据：

`Selenium` 相当于是一个机器人。可以模拟人类在浏览器上的一些行为，自动处理浏览器上的一些行为，比如点击，填充数据，删除cookie等。 `chromedriver` 是一个驱动 `Chrome` 浏览器的驱动程序，使用他才可以驱动浏览器。当然针对不同的浏览器有不同的driver。以下列出了不同浏览器及其对应的driver：

1. Chrome：https://sites.google.com/a/chromium.org/chromedriver/downloads
2. Firefox：https://github.com/mozilla/geckodriver/releases
3. Edge：https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
4. Safari：https://webkit.org/blog/6900/webdriver-support-in-safari-10/

安装**Selenium**和**chromedriver**：

1. 安装 `Selenium` ： `Selenium` 有很多语言的版本，有java、ruby、python等。我们下载python版本的就可以了。

```
pip install selenium
```

2. 安装 `chromedriver` ：下载完成后，放到不需要权限的纯英文目录下就可以了。

快速入门：

现在以一个简单的获取百度首页的例子来讲下 `Selenium` 和 `chromedriver` 如何快速入门：

```
from selenium import webdriver

# chromedriver的绝对路径
driver_path = r'D:\ProgramApp\chromedriver\chromedriver.exe'

# 初始化一个driver，并且指定chromedriver的路径
driver = webdriver.Chrome(executable_path=driver_path)
# 请求网页
driver.get("https://www.baidu.com/")
# 通过page_source获取网页源代码
print(driver.page_source)
```

selenium常用操作：

更多教程请参考：http://selenium-python.readthedocs.io/installation.html#introduction

关闭页面：

1. `driver.close()` ：关闭当前页面。
2. `driver.quit()` ：退出整个浏览器。

定位元素：

1. `find_element_by_id` ：根据id来查找某个元素。等价于：

```
submitTag = driver.find_element_by_id('su')
submitTag1 = driver.find_element(By.ID,'su')
```

2. `find_element_by_class_name` ：根据类名查找元素。 等价于：

```
submitTag = driver.find_element_by_class_name('su')
submitTag1 = driver.find_element(By.CLASS_NAME,'su')
```

3. `find_element_by_name` ：根据name属性的值来查找元素。等价于：

```
submitTag = driver.find_element_by_name('email')
submitTag1 = driver.find_element(By.NAME,'email')
```

4. `find_element_by_tag_name`：根据标签名来查找元素。等价于：

```
submitTag = driver.find_element_by_tag_name('div')
submitTag1 = driver.find_element(By.TAG_NAME,'div')
```

5. `find_element_by_xpath`：根据xpath语法来获取元素。等价于：

```
submitTag = driver.find_element_by_xpath('//div')
submitTag1 = driver.find_element(By.XPATH,'//div')
```

6. `find_element_by_css_selector`：根据css选择器选择元素。等价于：

```
submitTag = driver.find_element_by_css_selector('//div')
submitTag1 = driver.find_element(By.CSS_SELECTOR,'//div')
```

要注意，`find_element`是获取第一个满足条件的元素。`find_elements`是获取所有满足条件的元素。

操作表单元素：

1. 操作输入框：分为两步。第一步：找到这个元素。第二步：使用`send_keys(value)`，将数据填充进去。示例代码如下：

```
inputTag = driver.find_element_by_id('kw')
inputTag.send_keys('python')
```

使用`clear`方法可以清除输入框中的内容。示例代码如下：

```
inputTag.clear()
```

2. 操作checkbox：因为要选中`checkbox`标签，在网页中是通过鼠标点击的。因此想要选中`checkbox`标签，那么先选中这个标签，然后执行`click`事件。示例代码如下：

```
rememberTag = driver.find_element_by_name("rememberMe")
rememberTag.click()
```

3. 选择select：select元素不能直接点击。因为点击后还需要选中元素。这时候selenium就专门为select标签提供了一个类`selenium.webdriver.support.ui.Select`。将获取到的元素当成参数传到这个类中，创建这个对象。以后就可以使用这个对象进行选择了。示例代码如下：

```
from selenium.webdriver.support.ui import Select
# 选中这个标签，然后使用Select创建对象
selectTag = Select(driver.find_element_by_name("jumpMenu"))
# 根据索引选择
```

```
selectTag.select_by_index(1)
# 根据值选择
selectTag.select_by_value("http://www.95yueba.com")
# 根据可视的文本选择
selectTag.select_by_visible_text("95秀客户端")
# 取消选中所有选项
selectTag.deselect_all()
```

4. 操作按钮：操作按钮有很多种方式。比如单击、右击、双击等。这里讲一个最常用的。就是点击。直接调用 `click` 函数就可以了。示例代码如下：

```
inputTag = driver.find_element_by_id('su')
inputTag.click()
```

行为链：

有时候在页面中的操作可能要有很多步，那么这时候可以使用鼠标行为链类 `ActionChains` 来完成。比如现在要将鼠标移动到某个元素上并执行点击事件。那么示例代码如下：

```
inputTag = driver.find_element_by_id('kw')
submitTag = driver.find_element_by_id('su')

actions = ActionChains(driver)
actions.move_to_element(inputTag)
actions.send_keys_to_element(inputTag,'python')
actions.move_to_element(submitTag)
actions.click(submitTag)
actions.perform()
```

还有更多的鼠标相关的操作。

- click_and_hold(element)：点击但不松开鼠标。
- context_click(element)：右键点击。
- double_click(element)：双击。 更多方法请参考：http://selenium-python.readthedocs.io/api.html

Cookie操作：

1. 获取所有的 `cookie` ：

```
for cookie in driver.get_cookies():
    print(cookie)
```

2. 根据cookie的key获取value：

```
value = driver.get_cookie(key)
```

3. 删除所有的cookie：

```
driver.delete_all_cookies()
```

4. 删除某个 `cookie` ：

```
driver.delete_cookie(key)
```

页面等待：

现在的网页越来越多采用了 Ajax 技术，这样程序便不能确定何时某个元素完全加载出来了。如果实际页面等待时间过长导致某个dom元素还没出来，但是你的代码直接使用了这个WebElement，那么就会抛出NullPointer的异常。为了解决这个问题。所以 Selenium 提供了两种等待方式：一种是隐式等待、一种是显式等待。

1. 隐式等待：调用 `driver.implicitly_wait` 。那么在获取不可用的元素之前，会先等待10秒中的时间。示例代码如下：

```
driver = webdriver.Chrome(executable_path=driver_path)
driver.implicitly_wait(10)
# 请求网页
driver.get("https://www.douban.com/")
```

2. 显示等待：显示等待是表明某个条件成立后才执行获取元素的操作。也可以在等待的时候指定一个最大的时间，如果超过这个时间那么就抛出一个异常。显示等待应该使用 `selenium.webdriver.support.excepted_conditions` 期望的条件和 `selenium.webdriver.support.ui.WebDriverWait` 来配合完成。示例代码如下：

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Firefox()
driver.get("http://somedomain/url_that_delays_loading")
try:
    element = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "myDynamicElement"))
    )
finally:
    driver.quit()
```

3. 一些其他的等待条件：
    - presence_of_element_located：某个元素已经加载完毕了。

- presence_of_all_ememt_located：网页中所有满足条件的元素都加载完毕了。
- element_to_be_cliable：某个元素是可以点击了。

  更多条件请参考：http://selenium-python.readthedocs.io/waits.html

切换页面：

有时候窗口中有很多子tab页面。这时候肯定是需要进行切换的。`selenium`提供了一个叫做`switch_to_window`来进行切换，具体切换到哪个页面，可以从`driver.window_handles`中找到。示例代码如下：

```
# 打开一个新的页面
self.driver.execute_script("window.open('"+url+"')")
# 切换到这个新的页面中
self.driver.switch_to_window(self.driver.window_handles[1])
```

设置代理ip：

有时候频繁爬取一些网页。服务器发现你是爬虫后会封掉你的ip地址。这时候我们可以更改代理ip。更改代理ip，不同的浏览器有不同的实现方式。这里以`Chrome`浏览器为例来讲解：

```
from selenium import webdriver

options = webdriver.ChromeOptions()
options.add_argument("--proxy-server=http://110.73.2.248:8123")
driver_path = r"D:\ProgramApp\chromedriver\chromedriver.exe"
driver = webdriver.Chrome(executable_path=driver_path,chrome_options=options)

driver.get('http://httpbin.org/ip')
```

`WebElement`元素：

`from selenium.webdriver.remote.webelement import WebElement`类是每个获取出来的元素的所属类。
有一些常用的属性：

1. get_attribute：这个标签的某个属性的值。
2. screentshot：获取当前页面的截图。这个方法只能在`driver`上使用。
   `driver`的对象类，也是继承自`WebElement`。
   更多请阅读相关源代码。

图形验证码识别技术：

阻碍我们爬虫的。有时候正是在登录或者请求一些数据时候的图形验证码。因此这里我们讲解一种能将图片翻译成文字的技术。将图片翻译成文字一般被成为光学文字识别（Optical Character Recognition），简写为 `OCR` 。实现 `OCR` 的库不是很多，特别是开源的。因为这块存在一定的技术壁垒（需要大量的数据、算法、机器学习、深度学习知识等），并且如果做好了具有很高的商业价值。因此开源的比较少。这里介绍一个比较优秀的图像识别开源库：Tesseract。

Tesseract：

Tesseract是一个OCR库，目前由谷歌赞助。Tesseract是目前公认最优秀、最准确的开源OCR库。Tesseract具有很高的识别度，也具有很高的灵活性，他可以通过训练识别任何字体。

安装：

Windows系统：

在以下链接下载可执行文件，然后一顿点击下一步安装即可（放在不需要权限的纯英文路径下）：
https://github.com/tesseract-ocr/

Linux系统：

可以在以下链接下载源码自行编译。
https://github.com/tesseract-ocr/tesseract/wiki/Compiling
或者在 `ubuntu` 下通过以下命令进行安装：

```
sudo apt install tesseract-ocr
```

Mac系统：

用 `Homebrew` 即可方便安装：

```
brew install tesseract
```

设置环境变量：

安装完成后，如果想要在命令行中使用 `Tesseract` ，那么应该设置环境变量。 `Mac` 和 `Linux` 在安装的时候就默认已经设置好了。在 `Windows` 下把 `tesseract.exe` 所在的路径添加到 `PATH` 环境变量中。

还有一个环境变量需要设置的是，要把训练的数据文件路径也放到环境变量中。
在环境变量中，添加一个 `TESSDATA_PREFIX=C:\path_to_tesseractdata\teseractdata` 。

在命令行中使用**tesseract**识别图像：

如果想要在 `cmd` 下能够使用 `tesseract` 命令，那么需要把 `tesseract.exe` 所在的目录放到 `PATH` 环境变量中。然后使用命令： `tesseract 图片路径 文件路径` 。
示例：

```
tesseract a.png a
```

那么就会识别出 `a.png` 中的图片，并且把文字写入到 `a.txt` 中。如果不想写入文件直接想显示在终端，那么不要加文件名就可以了。

在代码中使用**tesseract**识别图像：

在 `Python` 代码中操作 `tesseract` 。需要安装一个库，叫做 `pytesseract` 。通过 `pip` 的方式即可安装：

```
pip install pytesseract
```

并且，需要读取图片，需要借助一个第三方库叫做 `PIL` 。通过 `pip list` 看下是否安装。如果没有安装，通过 `pip` 的方式安装：

```
pip install PIL
```

使用 `pytesseract` 将图片上的文字转换为文本文字的示例代码如下：

```
# 导入pytesseract库
import pytesseract
# 导入Image库
from PIL import Image

# 指定tesseract.exe所在的路径
pytesseract.pytesseract.tesseract_cmd = r'D:\ProgramApp\TesseractOCR\tesseract.exe'

# 打开图片
image = Image.open("a.png")
# 调用image_to_string将图片转换为文字
text = pytesseract.image_to_string(image)
print(text)
```

用 `pytesseract` 处理拉勾网图形验证码：

```
import pytesseract
from urllib import request
```

```
from PIL import Image
import time


pytesseract.pytesseract.tesseract_cmd = r"D:\ProgramApp\TesseractOCR\tesseract.exe"


while True:
    captchaUrl = "https://passport.lagou.com/vcode/create?from=register&refresh=1513081
451891"
    request.urlretrieve(captchaUrl,'captcha.png')
    image = Image.open('captcha.png')
    text = pytesseract.image_to_string(image,lang='eng')
    print(text)
    time.sleep(2)
```

字体反爬

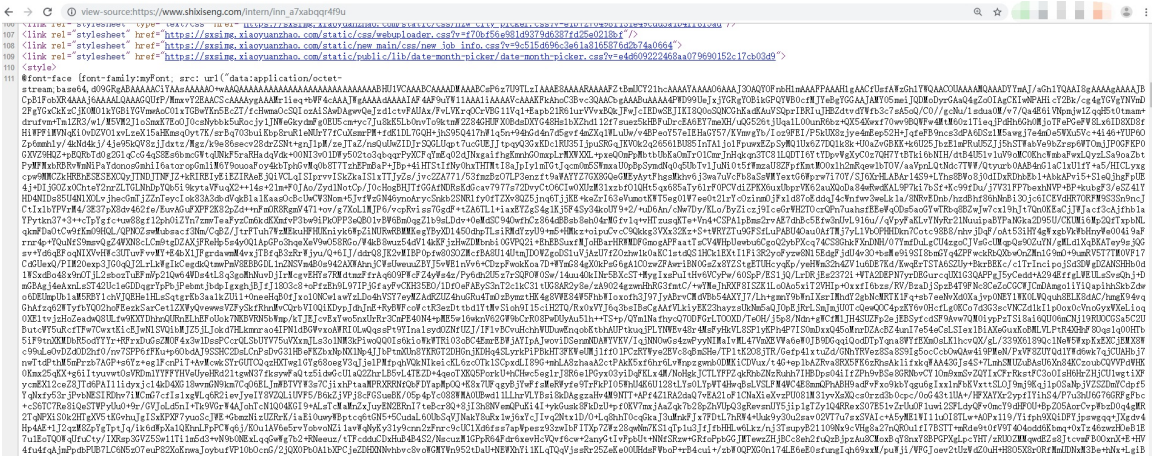
字体反爬原理：

1. 网页开发者自己创造一种字体，因为在字体中每个文字都有其代号，那么以后在网页中不会直接显示这个文字的最终的效果，而是显示的代号，因此即使获取到了网页中的文本内容，也只是获取到文字的代号，而不是文字本身。

2. 因为创造字体费时费力，并且如果把中国3000多常用汉字都实现，那么这个字体将达到几十兆，也会影响网页的加载。一般情况下为了反爬虫，仅会针对0-9以及少数汉字进行自己单独创建，其他的还是使用用户系统中自带的字体。

寻找自定义字体：

1. 一般情况下为了考虑网页渲染性能，通常网页开发者会把自定义的字体编码成 `base64` 的方式，因此我们可以到网页中找到 `@font-face` 属性，然后获取里面的 `base64` 代码，再用 `Python` 代码进行解码，然后再保存本地。示例： `view-source:https://www.shixiseng.com/intern/inn_a7xabqqr4f9u` 。示例图如下：



2. 如果没有使用 `base64` ，还有另外一种方式，就是直接把字体文件放到服务器上，然后前端通过 `@font-face` 中的url函数进行加载。示例： `https://developer.mozilla.org/zh-CN/docs/Web/CSS/@font-face` 。

分析字体：

1. 分析字体需要将字体转换成 `xml` 文件，然后查看其中的 `cmap` 和 `glyf` 中的属性。其中 `cmap` 存储的是 `code` 和 `name` 的映射，而 `glyf` 下存储的是每个 `name` 下的字体绘制规则。

cmap存储**code**和**name**的关系：

```
<cmap>
  <tableVersion version="0"/>
  <cmap_format_4 platformID="0" platEncID="3" language="0">
    <map code="0x9476" name="glyph00003"/><!-- CJK UNIFIED IDEOGRAPH-9476 -->
    <map code="0x958f" name="glyph00009"/><!-- CJK UNIFIED IDEOGRAPH-958F -->
    <map code="0x993c" name="glyph00005"/><!-- CJK UNIFIED IDEOGRAPH-993C -->
    <map code="0x9a4b" name="glyph00006"/><!-- CJK UNIFIED IDEOGRAPH-9A4B -->
    <map code="0x9e3a" name="glyph00007"/><!-- CJK UNIFIED IDEOGRAPH-9E3A -->
    <map code="0x9ea3" name="glyph00001"/><!-- CJK UNIFIED IDEOGRAPH-9EA3 -->
    <map code="0x9f64" name="glyph00008"/><!-- CJK UNIFIED IDEOGRAPH-9F64 -->
    <map code="0x9f92" name="glyph00002"/><!-- CJK UNIFIED IDEOGRAPH-9F92 -->
    <map code="0x9fa4" name="glyph00004"/><!-- CJK UNIFIED IDEOGRAPH-9FA4 -->
    <map code="0x9fa5" name="glyph00010"/><!-- CJK UNIFIED IDEOGRAPH-9FA5 -->
  </cmap_format_4>
  <cmap_format_12 platformID="0" platEncID="4" format="12" reserved="0" length="13
    <map code="0x9476" name="glyph00003"/><!-- CJK UNIFIED IDEOGRAPH-9476 -->
    <map code="0x958f" name="glyph00009"/><!-- CJK UNIFIED IDEOGRAPH-958F -->
    <map code="0x993c" name="glyph00005"/><!-- CJK UNIFIED IDEOGRAPH-993C -->
    <map code="0x9a4b" name="glyph00006"/><!-- CJK UNIFIED IDEOGRAPH-9A4B -->
    <map code="0x9e3a" name="glyph00007"/><!-- CJK UNIFIED IDEOGRAPH-9E3A -->
    <map code="0x9ea3" name="glyph00001"/><!-- CJK UNIFIED IDEOGRAPH-9EA3 -->
    <map code="0x9f64" name="glyph00008"/><!-- CJK UNIFIED IDEOGRAPH-9F64 -->
    <map code="0x9f92" name="glyph00002"/><!-- CJK UNIFIED IDEOGRAPH-9F92 -->
    <map code="0x9fa4" name="glyph00004"/><!-- CJK UNIFIED IDEOGRAPH-9FA4 -->
    <map code="0x9fa5" name="glyph00010"/><!-- CJK UNIFIED IDEOGRAPH-9FA5 -->
  </cmap_format_12>
```

glyf存储字体的形状：

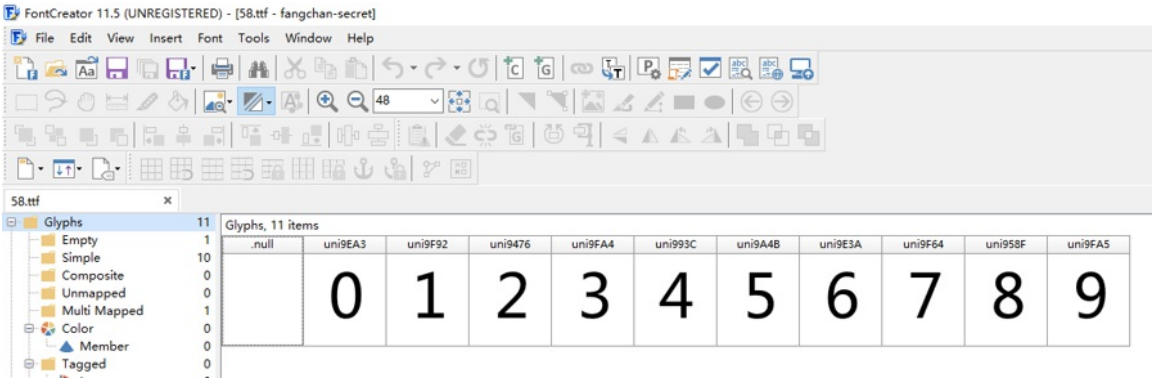
```
<glyf>

  <!-- The xMin, yMin, xMax and yMax values
       will be recalculated by the compiler. -->

  <TTGlyph name="glyph00000"/><!-- contains no outline data -->

  <TTGlyph name="glyph00001" xMin="0" yMin="-26" xMax="1113" yMax="1575">
    <contour>
      <pt x="91" y="744" on="1"/>
      <pt x="91" y="1154" on="0"/>
      <pt x="363" y="1575" on="0"/>
      <pt x="621" y="1575" on="1"/>
      <pt x="1113" y="1575" on="0"/>
      <pt x="1113" y="781" on="1"/>
      <pt x="1113" y="389" on="0"/>
      <pt x="836" y="-26" on="0"/>
      <pt x="586" y="-26" on="1"/>
      <pt x="350" y="-26" on="0"/>
      <pt x="91" y="368" on="0"/>
    </contour>
```

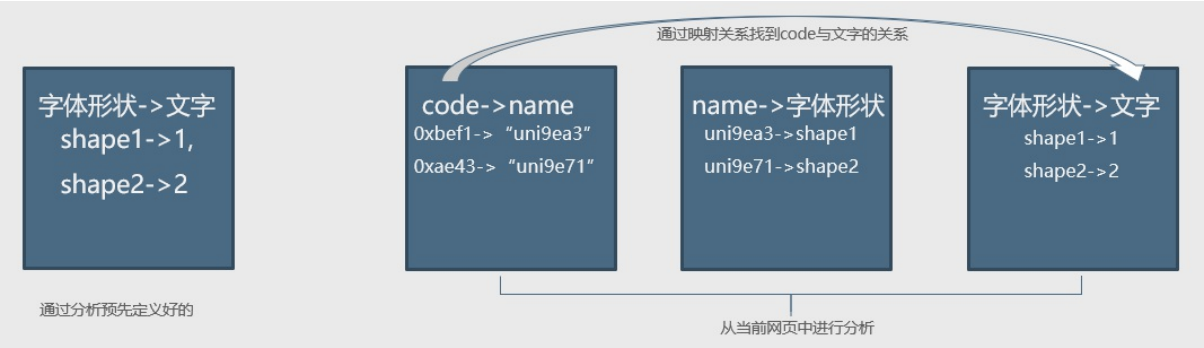
2. 从第1步中我们知道了 `name` 对应的字体的绘制规则，但是还是不知道字体是长什么样子，那么可以通过一款叫做 `FontCreator` 的软件来打开 `.tff` 的字体文件，这样就可以看到每个 `name` 对应的字体最终的呈现效果。（FontCreator是一款制作字体的工具，下载地址： `https://www.high-logic.com/FontCreatorSetup-x64.exe` ，这款软件有30天的试用期）。



字体反爬解决方案：

1. 在网页中，直接显示的是字体的 `code` ，而不是 `name` 。并且网页开发者为了增加爬虫的难度，有可能在多次请求之间 `code->name->最终字体` 的映射会发生改变。但是最终字体的形状是不会改变的，因此我们可以通过形状对比来进行判断。

2. 我们可以通过分析字体，得出每个字体形状对应的文字，然后保存到一个字典中。以后再请求网页的时候，就进行反向解析，先获取字体的形状，再通过字体形状反向获取代号所对应的具体文字内容。



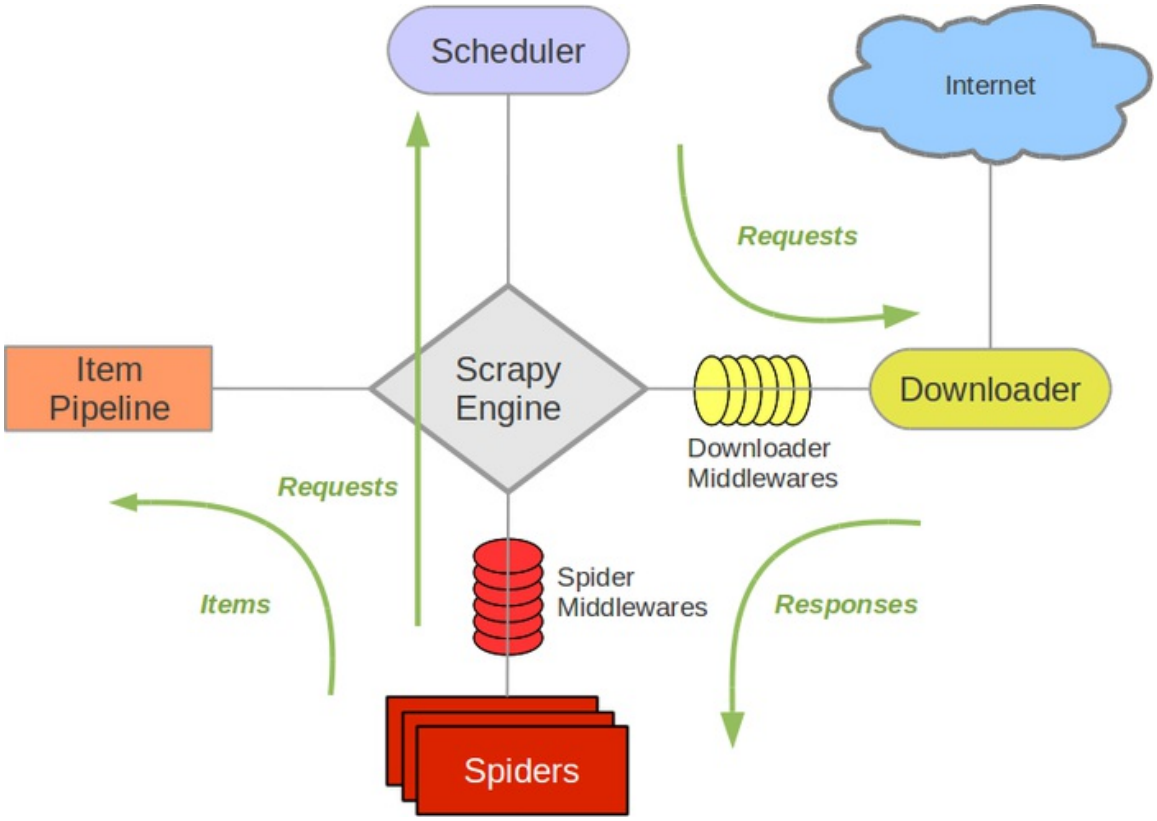
Scrapy框架架构

Scrapy框架介绍：

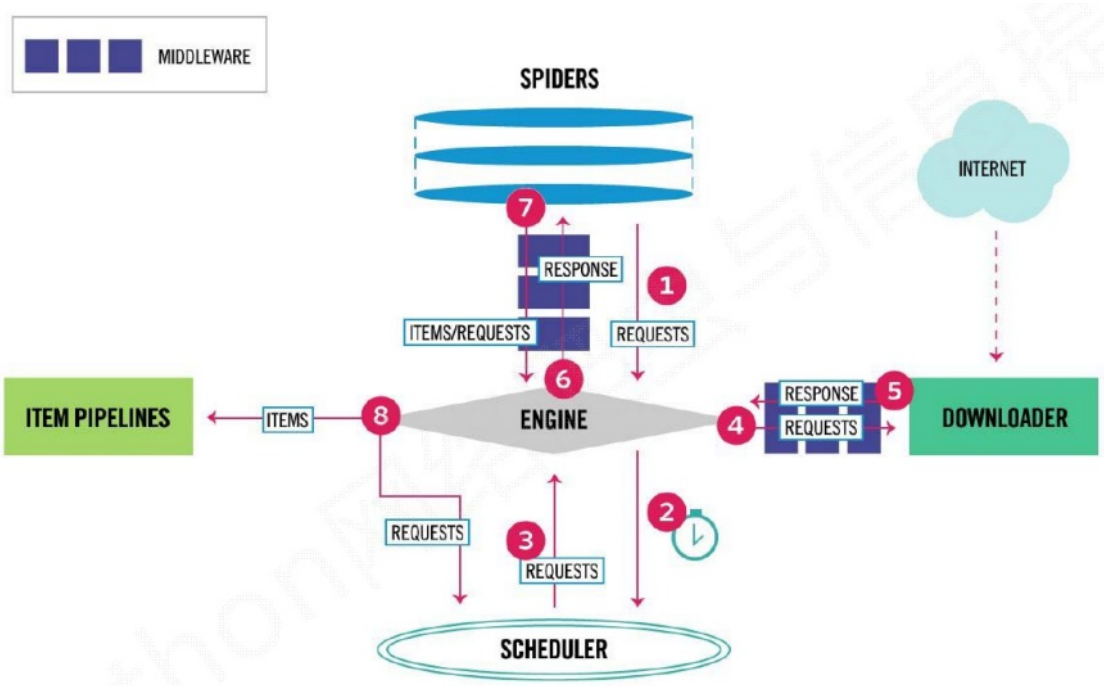
写一个爬虫，需要做很多的事情。比如：发送网络请求、数据解析、数据存储、反反爬虫机制（更换ip代理、设置请求头等）、异步请求等。这些工作如果每次都要自己从零开始写的话，比较浪费时间。因此 `Scrapy` 把一些基础的东西封装好了，在他上面写爬虫可以变的更加的高效（爬取效率和开发效率）。因此真正在公司里，一些上了量的爬虫，都是使用 `Scrapy` 框架来解决。

Scrapy架构图：

1. 流程图（1）：



2. 流程图（2）：



Scrapy框架模块功能：

1. `Scrapy Engine（引擎）`：`Scrapy`框架的核心部分。负责在`Spider`和`ItemPipeline`、`Downloader`、`Scheduler`中间通信、传递数据等。
2. `Spider（爬虫）`：发送需要爬取的链接给引擎，最后引擎把其他模块请求回来的数据再发送给爬虫，爬虫就去解析想要的数据。这个部分是我们开发者自己写的，因为要爬取哪些链接，页面中的哪些数据是我们需要的，都是由程序员自己决定。
3. `Scheduler（调度器）`：负责接收引擎发送过来的请求，并按照一定的方式进行排列和整理，负责调度请求的顺序等。
4. `Downloader（下载器）`：负责接收引擎传过来的下载请求，然后去网络上下载对应的数据再交还给引擎。
5. `Item Pipeline（管道）`：负责将`Spider（爬虫）`传递过来的数据进行保存。具体保存在哪里，应该看开发者自己的需求。
6. `Downloader Middlewares（下载中间件）`：可以扩展下载器和引擎之间通信功能的中间件。
7. `Spider Middlewares（Spider中间件）`：可以扩展引擎和爬虫之间通信功能的中间件。

Scrapy快速入门

安装和文档：

1. 安装：通过 `pip install scrapy` 即可安装。
2. Scrapy官方文档：http://doc.scrapy.org/en/latest
3. Scrapy中文文档：http://scrapy-chs.readthedocs.io/zh_CN/latest/index.html

> 注意：
>
> 1. 在 `ubuntu` 上安装 `scrapy` 之前，需要先安装以下依赖：`sudo apt-get install python3-dev build-essential python3-pip libxml2-dev libxslt1-dev zlib1g-dev libffi-dev libssl-dev` ，然后再通过 `pip install scrapy` 安装。
> 2. 如果在 `windows` 系统下，提示这个错误 `ModuleNotFoundError: No module named 'win32api'` ，那么使用以下命令可以解决： `pip install pypiwin32` 。

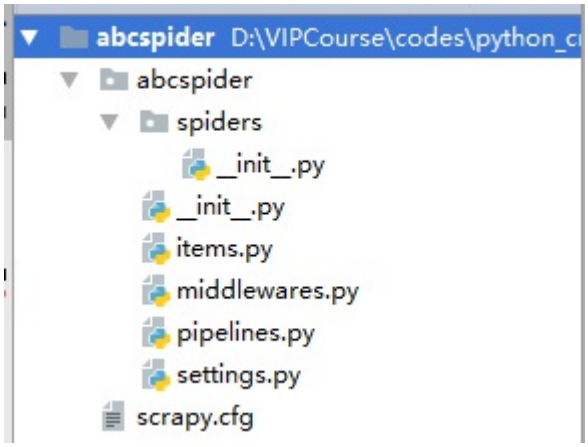
快速入门：

创建项目：

要使用 `Scrapy` 框架创建项目，需要通过命令来创建。首先进入到你想把这个项目存放的目录。然后使用以下命令创建：

```
scrapy startproject [项目名称]
```

目录结构介绍：



以下介绍下主要文件的作用：

1. items.py：用来存放爬虫爬取下来数据的模型。

2. middlewares.py：用来存放各种中间件的文件。
3. pipelines.py：用来将 `items` 的模型存储到本地磁盘中。
4. settings.py：本爬虫的一些配置信息（比如请求头、多久发送一次请求、ip代理池等）。
5. scrapy.cfg：项目的配置文件。
6. spiders包：以后所有的爬虫，都是存放到这个里面。

使用**Scrapy**框架爬取糗事百科段子：

使用命令创建一个爬虫：

```
scrapy gensipder qsbk "qiushibaike.com"
```

创建了一个名字叫做 `qsbk` 的爬虫，并且能爬取的网页只会限制在 `qiushibaike.com` 这个域名下。

爬虫代码解析：

```
import scrapy

class QsbkSpider(scrapy.Spider):
    name = 'qsbk'
    allowed_domains = ['qiushibaike.com']
    start_urls = ['http://qiushibaike.com/']

    def parse(self, response):
        pass
```

其实这些代码我们完全可以自己手动去写，而不用命令。只不过是不用命令，自己写这些代码比较麻烦。
要创建一个Spider，那么必须自定义一个类，继承自 `scrapy.Spider` ，然后在这个类中定义三个属性和一个方法。

1. name：这个爬虫的名字，名字必须是唯一的。
2. allow_domains：允许的域名。爬虫只会爬取这个域名下的网页，其他不是这个域名下的网页会被自动忽略。
3. start_urls：爬虫从这个变量中的url开始。
4. parse：引擎会把下载器下载回来的数据扔给爬虫解析，爬虫再把数据传给这个 `parse` 方法。这个是个固定的写法。这个方法的作用有两个，第一个是提取想要的数据。第二个是生成下一个请求的url。

修改 **`settings.py`** 代码：

在做一个爬虫之前，一定要记得修改 `setttings.py` 中的设置。两个地方是强烈建议设置的。

1. `ROBOTSTXT_OBEY` 设置为False。默认是True。即遵守机器协议，那么在爬虫的时候，scrapy首

先去找robots.txt文件，如果没有找到。则直接停止爬取。

2. `DEFAULT_REQUEST_HEADERS` 添加 `User-Agent` 。这个也是告诉服务器，我这个请求是一个正常的请求，不是一个爬虫。

完成的爬虫代码：

1. 爬虫部分代码：

```
import scrapy
from abcspider.items import QsbkItem

class QsbkSpider(scrapy.Spider):
    name = 'qsbk'
    allowed_domains = ['qiushibaike.com']
    start_urls = ['https://www.qiushibaike.com/text/']

    def parse(self, response):
        outerbox = response.xpath("//div[@id='content-left']/div")
        items = []
        for box in outerbox:
            author = box.xpath(".//div[contains(@class,'author')]//h2/text()").ext
ract_first().strip()
            content = box.xpath(".//div[@class='content']/span/text()").extract_fi
rst().strip()
            item = QsbkItem()
            item["author"] = author
            item["content"] = content
            items.append(item)
        return items
```

2. items.py部分代码：

```
import scrapy
class QsbkItem(scrapy.Item):
    author = scrapy.Field()
    content = scrapy.Field()
```

3. pipeline部分代码：

```
import json

class AbcspiderPipeline(object):
    def __init__(self):

        self.items = []
```

```
    def process_item(self, item, spider):
        self.items.append(dict(item))
        print("="*40)
        return item

    def close_spider(self,spider):
        with open('qsbk.json','w',encoding='utf-8') as fp:
            json.dump(self.items,fp,ensure_ascii=False)
```

运行**scrapy**项目：

运行scrapy项目。需要在终端，进入项目所在的路径，然后`scrapy crawl [爬虫名字]`即可运行指定的爬虫。如果不想每次都在命令行中运行，那么可以把这个命令写在一个文件中。以后就在pycharm中执行运行这个文件就可以了。比如现在新创建一个文件叫做`start.py`，然后在这个文件中填入以下代码：

```
from scrapy import cmdline

cmdline.execute("scrapy crawl qsbk".split())
```

CrawlSpider

在上一个糗事百科的爬虫案例中。我们是自己在解析完整个页面后获取下一页的url，然后重新发送一个请求。有时候我们想要这样做，只要满足某个条件的url，都给我进行爬取。那么这时候我们就可以通过 `CrawlSpider` 来帮我们完成了。 `CrawlSpider` 继承自 `Spider` ，只不过是在之前的基础之上增加了新的功能，可以定义爬取的url的规则，以后scrapy碰到满足条件的url都进行爬取，而不用手动的 `yield Request` 。

CrawlSpider爬虫：

创建CrawlSpider爬虫：

之前创建爬虫的方式是通过 `scrapy genspider [爬虫名字] [域名]` 的方式创建的。如果想要创建 `CrawlSpider` 爬虫，那么应该通过以下命令创建：

```
scrapy genspider -c crawl [爬虫名字] [域名]
```

LinkExtractors链接提取器：

使用 `LinkExtractors` 可以不用程序员自己提取想要的url，然后发送请求。这些工作都可以交给 `LinkExtractors` ，他会在所有爬的页面中找到满足规则的 `url` ，实现自动的爬取。以下对 `LinkExtractors` 类做一个简单的介绍：

```
class scrapy.linkextractors.LinkExtractor(
    allow = (),
    deny = (),
    allow_domains = (),
    deny_domains = (),
    deny_extensions = None,
    restrict_xpaths = (),
    tags = ('a','area'),
    attrs = ('href'),
    canonicalize = True,
    unique = True,
    process_value = None
)
```

主要参数讲解：

- allow：允许的url。所有满足这个正则表达式的url都会被提取。
- deny：禁止的url。所有满足这个正则表达式的url都不会被提取。
- allow_domains：允许的域名。只有在这个里面指定的域名的url才会被提取。

- deny_domains：禁止的域名。所有在这个里面指定的域名的url都不会被提取。
- restrict_xpaths：严格的xpath。和allow共同过滤链接。

Rule规则类：

定义爬虫的规则类。以下对这个类做一个简单的介绍：

```
class scrapy.spiders.Rule(
    link_extractor,
    callback = None,
    cb_kwargs = None,
    follow = None,
    process_links = None,
    process_request = None
)
```

主要参数讲解：

- link_extractor：一个 `LinkExtractor` 对象，用于定义爬取规则。
- callback：满足这个规则的url，应该要执行哪个回调函数。因为 `CrawlSpider` 使用了 `parse` 作为回调函数，因此不要覆盖 `parse` 作为回调函数自己的回调函数。
- follow：指定根据该规则从response中提取的链接是否需要跟进。
- process_links：从link_extractor中获取到链接后会传递给这个函数，用来过滤不需要爬取的链接。

微信小程序社区**CrawlSpider**案例

Scrapy Shell

我们想要在爬虫中使用xpath、beautifulsoup、正则表达式、css选择器等来提取想要的数据。但是因为 `scrapy` 是一个比较重的框架。每次运行起来都要等待一段时间。因此要去验证我们写的提取规则是否正确，是一个比较麻烦的事情。因此 `Scrapy` 提供了一个shell，用来方便的测试规则。当然也不仅仅局限于这一个功能。

打开Scrapy Shell：

打开cmd终端，进入到 `Scrapy` 项目所在的目录，然后进入到 `scrapy` 框架所在的虚拟环境中，输入命令 `scrapy shell [链接]` 。就会进入到scrapy的shell环境中。在这个环境中，你可以跟在爬虫的 `parse` 方法中一样使用了。

```
[s] Available Scrapy objects:
[s]   scrapy     scrapy module (contains scrapy.Request, scrapy.Selector, etc)
[s]   crawler    <scrapy.crawler.Crawler object at 0x058D3E50>
[s]   item       {}
[s]   request    <GET http://www.wxapp-union.com/article-3409-1.html>
[s]   response   <200 http://www.wxapp-union.com/article-3409-1.html>
[s]   settings   <scrapy.settings.Settings object at 0x06639C10>
[s]   spider     <WxappSpider 'wxapp' at 0x6a6c310>
[s] Useful shortcuts:
[s]   fetch(url[, redirect=True]) Fetch URL and update local objects (by default, redirects are followed)
[s]   fetch(req)                  Fetch a scrapy.Request and update local objects
[s]   shelp()           Shell help (print this help)
[s]   view(response)    View response in a browser
>>> from bs4 import BeautifulSoup
>>> title = response.xpath("//h1[@class='ph']/text()").extract_first()
>>> title
'周留存=月留存=半年留存，「忆年」相册小程序如何黏住千万用户？ ... ... '
>>>
```

Request和Response对象

Request对象：

```
class Request(object_ref):

    def __init__(self, url, callback=None, method='GET', headers=None, body=None,
                 cookies=None, meta=None, encoding='utf-8', priority=0,
                 dont_filter=False, errback=None, flags=None):

        self._encoding = encoding  # this one has to be set first
        self.method = str(method).upper()
        self._set_url(url)
        self._set_body(body)
        assert isinstance(priority, int), "Request priority not an integer: %r" % priority
        self.priority = priority

        assert callback or not errback, "Cannot use errback without a callback"
        self.callback = callback
        self.errback = errback

        self.cookies = cookies or {}
        self.headers = Headers(headers or {}, encoding=encoding)
        self.dont_filter = dont_filter

        self._meta = dict(meta) if meta else None
        self.flags = [] if flags is None else list(flags)
```

Request对象在我们写爬虫，爬取一页的数据需要重新发送一个请求的时候调用。这个类需要传递一些参数，其中比较常用的参数有：

1. `url`：这个request对象发送请求的url。
2. `callback`：在下载器下载完相应的数据后执行的回调函数。
3. `method`：请求的方法。默认为`GET`方法，可以设置为其他方法。
4. `headers`：请求头，对于一些固定的设置，放在`settings.py`中指定就可以了。对于那些非固定的，可以在发送请求的时候指定。
5. `meta`：比较常用。用于在不同的请求之间传递数据用的。
6. `encoding`：编码。默认的为`utf-8`，使用默认的就可以了。
7. `dont_filter`：表示不由调度器过滤。在执行多次重复的请求的时候用得比较多。
8. `errback`：在发生错误的时候执行的函数。

Response对象：

Response对象一般是由`Scrapy`给你自动构建的。因此开发者不需要关心如何创建`Response`对象，而是如何使用他。`Response`对象有很多属性，可以用来提取数据的。主要有以下属性：

1. meta：从其他请求传过来的`meta`属性，可以用来保持多个请求之间的数据连接。
2. encoding：返回当前字符串编码和解码的格式。
3. text：将返回来的数据作为`unicode`字符串返回。
4. body：将返回来的数据作为`bytes`字符串返回。
5. xpath：xapth选择器。
6. css：css选择器。

发送**POST**请求：

有时候我们想要在请求数据的时候发送post请求，那么这时候需要使用 `Request` 的子类 `FormRequest` 来实现。如果想要在爬虫一开始的时候就发送 `POST` 请求，那么需要在爬虫类中重写 `start_requests(self)` 方法，并且不再调用 `start_urls` 里的url。

模拟登录

案例一：模拟登录人人网

案例二：模拟登录豆瓣网（识别验证码）

图形验证码识别平台：https://market.aliyun.com/products/57126001/cmapi014396.html#sku=yuncode839600006

下载文件和图片

Scrapy为下载item中包含的文件(比如在爬取到产品时，同时也想保存对应的图片)提供了一个可重用的 `item pipelines` 。这些 `pipeline` 有些共同的方法和结构(我们称之为 `media pipeline`)。一般来说你会使用 `Files Pipeline` 或者 `Images Pipeline` 。

为什么要选择使用 `scrapy` 内置的下载文件的方法：

1. 避免重新下载最近已经下载过的文件。
2. 可以方便的指定文件存储的路径。
3. 可以将下载的图片转换成通用的格式。比如png或jpg。
4. 可以方便的生成缩略图。
5. 可以方便的检测图片的宽和高，确保他们满足最小限制。
6. 异步下载，效率非常高。

下载文件的 `Files Pipeline` ：

当使用 `Files Pipeline` 下载文件的时候，按照以下步骤来完成：

1. 定义好一个 `Item` ，然后在这个 `item` 中定义两个属性，分别为 `file_urls` 以及 `files` 。 `file_urls` 是用来存储需要下载的文件的url链接，需要给一个列表。
2. 当文件下载完成后，会把文件下载的相关信息存储到 `item` 的 `files` 属性中。比如下载路径、下载的url和文件的校验码等。
3. 在配置文件 `settings.py` 中配置 `FILES_STORE` ，这个配置是用来设置文件下载下来的路径。
4. 启动 `pipeline` ：在 `ITEM_PIPELINES` 中设置 `scrapy.pipelines.files.FilesPipeline:1` 。

下载图片的 `Images Pipeline` ：

当使用 `Images Pipeline` 下载文件的时候，按照以下步骤来完成：

1. 定义好一个 `Item` ，然后在这个 `item` 中定义两个属性，分别为 `image_urls` 以及 `images` 。 `image_urls` 是用来存储需要下载的图片的url链接，需要给一个列表。
2. 当文件下载完成后，会把文件下载的相关信息存储到 `item` 的 `images` 属性中。比如下载路径、下载的url和图片的校验码等。
3. 在配置文件 `settings.py` 中配置 `IMAGES_STORE` ，这个配置是用来设置图片下载下来的路径。
4. 启动 `pipeline` ：在 `ITEM_PIPELINES` 中设置 `scrapy.pipelines.images.ImagesPipeline:1` 。

汽车之家宝马**5**系高清图片下载实战。

Downloader Middlewares（下载器中间件）

下载器中间件是引擎和下载器之间通信的中间件。在这个中间件中我们可以设置代理、更换请求头等来达到反反爬虫的目的。要写下载器中间件，可以在下载器中实现两个方法。一个是 `process_request(self,request,spider)` ，这个方法是在请求发送之前会执行，还有一个是 `process_response(self,request,response,spider)` ，这个方法是数据下载到引擎之前执行。

process_request(self,request,spider)：

这个方法是下载器在发送请求之前会执行的。一般可以在这个里面设置随机代理ip等。

1. 参数：
    - request：发送请求的request对象。
    - spider：发送请求的spider对象。
2. 返回值：
    - 返回None：如果返回None，Scrapy将继续处理该request，执行其他中间件中的相应方法，直到合适的下载器处理函数被调用。
    - 返回Response对象：Scrapy将不会调用任何其他的 `process_request` 方法，将直接返回这个response对象。已经激活的中间件的process_response()方法则会在每个response返回时被调用。
    - 返回Request对象：不再使用之前的request对象去下载数据，而是根据现在返回的request对象返回数据。
    - 如果这个方法中抛出了异常，则会调用process_exception方法。

process_response(self,request,response,spider)：

这个是下载器下载的数据到引擎中间会执行的方法。

1. 参数：
    - request：request对象。
    - response：被处理的response对象。
    - spider：spider对象。
2. 返回值：
    - 返回Response对象：会将这个新的response对象传给其他中间件，最终传给爬虫。
    - 返回Request对象：下载器链被切断，返回的request会重新被下载器调度下载。
    - 如果抛出一个异常，那么调用request的 `errback` 方法，如果没有指定这个方法，那么会抛出一个异常。

随机请求头中间件：

爬虫在频繁访问一个页面的时候，这个请求头如果一直保持一致。那么很容易被服务器发现，从而禁止掉这个请求头的访问。因此我们要在访问这个页面之前随机的更改请求头，这样才可以避免爬虫被抓。随机更改请求头，可以在下载中间件中实现。在请求发送给服务器之前，随机的选择一个请求头。这样就可以避免总使用一个请求头了。示例代码如下：

```
class UserAgentDownloadMiddleware(object):
    # user-agent随机请求头中间件
    USER_AGENTS = [
        'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; en-us) AppleWebKit/534.50 (K
HTML, like Gecko) Version/5.1 Safari/534.50',
        'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Geck
o) Chrome/63.0.3239.84 Safari/537.36',
        'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0;',
        'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)',
        'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv,2.0.1) Gecko/20100101 Firefox/
4.0.1',
        'Mozilla/5.0 (Windows NT 6.1; rv,2.0.1) Gecko/20100101 Firefox/4.0.1'
    ]
    def process_request(self,request,spider):
        user_agent = random.choice(self.USER_AGENTS)
        request.headers['User-Agent'] = user_agent
```

user-agent列表：http://www.useragentstring.com/pages/useragentstring.php?typ=Browser

ip代理池中间件

购买代理：

在以下代理商中购买代理：

1. 芝麻代理：http://http.zhimaruanjian.com/
2. 太阳代理：http://http.taiyangruanjian.com/
3. 快代理：http://www.kuaidaili.com/
4. 讯代理：http://www.xdaili.cn/
5. 蚂蚁代理：http://www.mayidaili.com/
   等购买代理。

使用ip代理池：

1. 开放代理池设置：

```
class IPProxyDownloadMiddleware(object):
    PROXIES = [
        "5.196.189.50:8080",
        "134.17.141.44:8080",
```

```
            "178.49.136.84:8080",
            "45.55.132.29:82",
            "178.44.185.15:8080"
        ]
        def process_request(self,request,spider):
            # proxy = random.choice(self.PROXIES)
            # print('被选中的代理：%s' % proxy)
            # request.meta['proxy'] = "http://" + proxy
            proxy = "121.199.6.124:16816"
            user_password = "970138074:rcdj35ur"
            request.meta['proxy'] = proxy
            request.headers['Proxy-Authorization'] = 'Basic ' + base64.b64encode(user_
password.encode('utf-8')).decode('utf-8')
```

2. 独享代理池设置：

```
 class IPProxyDownloadMiddleware(object):
    def process_request(self,request,spider):
        proxy = '121.199.6.124:16816'
        user_password = "970138074:rcdj35ur"
        request.meta['proxy'] = proxy
        # bytes
        b64_user_password = base64.b64encode(user_password.encode('utf-8'))
        request.headers['Proxy-Authorization'] = 'Basic ' + b64_user_password.deco
de('utf-8')
```

下载中间件实战案例

Settings配置文件

常用的配置项：

1. `BOT_NAME`：项目名称。
2. `ROBOTSTXT_OBEY`：是否遵守爬虫协议。默认不遵守。
3. `CONCURRENT_ITEMS`：代表 `pipeline` 同时处理的 `item` 数的最大值。默认是100
4. `CONCURRENT_REQUESTS`：代表下载器并发请求的最大是，默认是16。
5. `DEFAULT_REQUEST_HEADERS`：默认请求头。可以将一些不会经常变化的请求头放在这个里面。
6. `DEPTH_LIMIT`：爬取网站最大允许的深度。默认为0，如果为0，则没有限制。
7. `DOWNLOAD_DELAY`：下载器在下载某个页面前等待多长的时间。该选项用来限制爬虫的爬取速度，减轻服务器压力。同时也支持小数。
8. `DOWNLOAD_TIMEOUT`：下载器下载的超时时间。
9. `ITEM_PIPELINES`：处理 `item` 的 `Pipeline`，是一个字典，字典的key这个pipeline所在包的绝对路径，值是一个整数，优先级，值越小，优先级越高。
10. `LOG_ENABLED`：是否启用logging。默认是True。
11. `LOG_ENCODING`：log的编码。
12. `LOG_LEVEL`：log的级别。默认为 `DEBUG`。可选的级别有 `CRITICAL`、`ERROR`、`WARNING`、`INFO`、`DEBUG`。
13. `USER_AGENT`：请求头。默认为 `Scrapy/VERSION (+http://scrapy.org)`。
14. `PROXIES`：代理设置。
15. `COOKIES_ENABLED`：是否开启cookie。一般不要开启，避免爬虫被追踪到。如果特殊情况也可以开启。

Scrapy实战

糗事百科爬虫。

入门级爬虫。使用 `Spider` 。

小程序社区爬虫。

数据保存到json文件中。使用 `CrawlSpider` 。

模拟登录豆瓣网爬虫。

发送post请求模拟登录。

图片下载爬虫。

汽车之家宝马5系爬虫。

BOSS直聘爬虫。

BOSS直聘有很高的反爬虫机制，只要用同一个ip访问多个职位列表页，就会被封掉ip。采用代理ip的方式可解决问题。

简书网站整站爬虫。

数据保存到mysql数据库中。
将selenium+chromedriver集成到scrapy。

redis教程：

概述

`redis` 是一种支持分布式的 `nosql` 数据库,他的数据是保存在内存中，同时 `redis` 可以定时把内存数据同步到磁盘，即可以将数据持久化，并且他比 `memcached` 支持更多的数据结构（ `string` , `list列表[队列和栈]` , `set[集合]` , `sorted set[有序集合]` , `hash(hash表)`)。相关参考文档：http://redisdoc.com/index.html

redis使用场景：

1. 登录会话存储：存储在 `redis` 中，与 `memcached` 相比，数据不会丢失。
2. 排行版/计数器：比如一些秀场类的项目，经常会有一些前多少名的主播排名。还有一些文章阅读量的技术，或者新浪微博的点赞数等。
3. 作为消息队列：比如 `celery` 就是使用 `redis` 作为中间人。
4. 当前在线人数：还是之前的秀场例子，会显示当前系统有多少在线人数。
5. 一些常用的数据缓存：比如我们的 `BBS` 论坛，板块不会经常变化的，但是每次访问首页都要从 `mysql` 中获取，可以在 `redis` 中缓存起来，不用每次请求数据库。
6. 把前200篇文章缓存或者评论缓存：一般用户浏览网站，只会浏览前面一部分文章或者评论，那么可以把前面200篇文章和对应的评论缓存起来。用户访问超过的，就访问数据库，并且以后文章超过200篇，则把之前的文章删除。
7. 好友关系：微博的好友关系使用 `redis` 实现。
8. 发布和订阅功能：可以用来做聊天软件。

`redis` 和 `memcached` 的比较：

	memcached	redis
类型	纯内存数据库	内存磁盘同步数据库
数据类型	在定义value时就要固定数据类型	不需要
虚拟内存	不支持	支持
过期策略	支持	支持
存储数据安全	不支持	可以将数据同步到dump.db中
灾难恢复	不支持	可以将磁盘中的数据恢复到内存中
分布式	支持	主从同步
订阅与发布	不支持	支持

`redis` 在 `ubuntu` 系统中的安装与启动

1. 安装：

   ```
   sudo apt-get install redis-server
   ```

2. 卸载：

   ```
   sudo apt-get purge --auto-remove redis-server
   ```

3. 启动：`redis` 安装后，默认会自动启动，可以通过以下命令查看：

   ```
   ps aux|grep redis
   ```

   如果想自己手动启动，可以通过以下命令进行启动：

   ```
   sudo service redis-server start
   ```

4. 停止：

   ```
   sudo service redis-server stop
   ```

redis在windows系统中的安装与启动：

1. 下载：redis官方是不支持windows操作系统的。但是微软的开源部门将redis移植到了windows上。因此下载地址不是在redis官网上。而是在github上：https://github.com/MicrosoftArchive/redis/releases。
2. 安装：点击一顿下一步安装就可以了。
3. 运行：进入到 `redis` 安装所在的路径然后执行 `redis-server.exe redis.windows.conf` 就可以运行了。
4. 连接：`redis` 和 `mysql` 以及 `mongo` 是一样的，都提供了一个客户端进行连接。输入命令 `redis-cli` （前提是redis安装路径已经加入到环境变量中了）就可以连接到 `redis` 服务器了。

其他机器访问本机redis服务器：

想要让其他机器访问本机的redis服务器。那么要修改redis.conf的配置文件，将bind改成 `bind [自己的ip地址或者0.0.0.0]` ，其他机器才能访问。
注意：**bind**绑定的是本机网卡的**ip**地址，而不是想让其他机器连接的**ip**地址。如果有多块网卡，那么可以绑定多个网卡的**ip**地址。如果绑定到额是**0.0.0.0**，那么意味着其他机器可以通过本机所有的**ip**地址进行访问。

对 `redis` 的操作

对 `redis` 的操作可以用两种方式，第一种方式采用 `redis-cli` ，第二种方式采用编程语言，比如 `Python` 、 `PHP` 和 `JAVA` 等。

1. 使用 `redis-cli` 对 `redis` 进行字符串操作：
2. 启动 `redis` ：

   ```
   sudo service redis-server start
   ```

3. 连接上 `redis-server` ：

   ```
   redis-cli -h [ip] -p [端口]
   ```

4. 添加：

   ```
   set key value
   如：
   set username xiaotuo
   ```

   将字符串值 `value` 关联到 `key` 。如果 `key` 已经持有其他值， `set` 命令就覆写旧值，无视其类型。并且默认的过期时间是永久，即永远不会过期。

5. 删除：

   ```
   del key
   如：
   del username
   ```

6. 设置过期时间：

   ```
   expire key timeout(单位为秒)
   ```

   也可以在设置值的时候，一同指定过期时间：

   ```
   set key value EX timeout
   或：
   setex key timeout value
   ```

7. 查看过期时间：

   ```
   ttl key
   如：
   ```

```
ttl username
```

8. 查看当前 `redis` 中的所有 `key` ：

```
keys *
```

9. 列表操作：
   - 在列表左边添加元素：

     ```
     lpush key value
     ```

     将值 `value` 插入到列表 `key` 的表头。如果 `key` 不存在，一个空列表会被创建并执行 `lpush` 操作。当 `key` 存在但不是列表类型时，将返回一个错误。

   - 在列表右边添加元素：

     ```
     rpush key value
     ```

     将值value插入到列表key的表尾。如果key不存在，一个空列表会被创建并执行RPUSH操作。当key存在但不是列表类型时，返回一个错误。

   - 查看列表中的元素：

     ```
     lrange key start stop
     ```

     返回列表 `key` 中指定区间内的元素，区间以偏移量 `start` 和 `stop` 指定,如果要左边的第一个到最后的一个 `lrange key 0 -1` 。

   - 移除列表中的元素：
     - 移除并返回列表 `key` 的头元素：

       ```
       lpop key
       ```

     - 移除并返回列表的尾元素：

       ```
       rpop key
       ```

     - 移除并返回列表 `key` 的中间元素：

       ```
       lrem key count value
       ```

       将删除 `key` 这个列表中， `count` 个值为 `value` 的元素。

- 指定返回第几个元素：

```
lindex key index
```

将返回 `key` 这个列表中，索引为 `index` 的这个元素。

- 获取列表中的元素个数：

```
llen key
如：
llen languages
```

- 删除指定的元素：

```
lrem key count value
如：
lrem languages 0 php
```

根据参数 count 的值，移除列表中与参数 value 相等的元素。 `count` 的值可以是以下几种：

  - count > 0：从表头开始向表尾搜索，移除与 `value` 相等的元素，数量为 `count` 。
  - count < 0：从表尾开始向表头搜索，移除与 `value` 相等的元素，数量为 `count` 的绝对值。
  - count = 0：移除表中所有与 `value` 相等的值。

10. `set` 集合的操作：

- 添加元素：

```
sadd set value1 value2....
如：
sadd team xiaotuo datuo
```

- 查看元素：

```
smembers set
如：
smembers team
```

- 移除元素：

```
srem set member...
如：
srem team xiaotuo datuo
```

- 查看集合中的元素个数：

```
scard set
如：
scard team1
```

- 获取多个集合的交集：

```
sinter set1 set2
如：
sinter team1 team2
```

- 获取多个集合的并集：

```
sunion set1 set2
如：
sunion team1 team2
```

- 获取多个集合的差集：

```
sdiff set1 set2
如：
sdiff team1 team2
```

11. `hash` 哈希操作：

- 添加一个新值：

```
hset key field value
如：
hset website baidu baidu.com
```

将哈希表 `key` 中的域 `field` 的值设为 `value` 。
如果 `key` 不存在，一个新的哈希表被创建并进行 `HSET` 操作。如果域 `field` 已经存在于哈希表中，旧值将被覆盖。

- 获取哈希中的 `field` 对应的值：

```
hget key field
如：
hget website baidu
```

- 删除 `field` 中的某个 `field` ：

```
hdel key field
如：
```

```
hdel website baidu
```

- 获取某个哈希中所有的 `field` 和 `value`：

```
hgetall key
如：
hgetall website
```

- 获取某个哈希中所有的 `field`：

```
hkeys key
如：
hkeys website
```

- 获取某个哈希中所有的值：

```
hvals key
如：
hvals website
```

- 判断哈希中是否存在某个 `field`：

```
hexists key field
如：
hexists website baidu
```

- 获取哈希中总共的键值对：

```
hlen field
如：
hlen website
```

12. 事务操作：Redis事务可以一次执行多个命令，事务具有以下特征：

- 隔离操作：事务中的所有命令都会序列化、按顺序地执行，不会被其他命令打扰。
- 原子操作：事务中的命令要么全部被执行，要么全部都不执行。
- 开启一个事务：

```
multi
```

以后执行的所有命令，都在这个事务中执行的。

- 执行事务：

```
exec
```

会将在 `multi` 和 `exec` 中的操作一并提交。

- 取消事务：

```
discard
```

会将 `multi` 后的所有命令取消。

- 监视一个或者多个 `key` ：

```
watch key...
```

监视一个(或多个)key，如果在事务执行之前这个(或这些) key被其他命令所改动，那么事务将被打断。

- 取消所有 `key` 的监视：

```
unwatch
```

13. 发布/订阅操作：

- 给某个频道发布消息：

```
publish channel message
```

- 订阅某个频道的消息：

```
subscribe channel
```

Scrapy-Redis分布式爬虫组件

`Scrapy`是一个框架，他本身是不支持分布式的。如果我们想要做分布式的爬虫，就需要借助一个组件叫做`Scrapy-Redis`，这个组件正是利用了`Redis`可以分布式的功能，集成到`Scrapy`框架中，使得爬虫可以进行分布式。可以充分的利用资源（多个ip、更多带宽、同步爬取）来提高爬虫的爬行效率。

分布式爬虫的优点：

1. 可以充分利用多台机器的带宽。
2. 可以充分利用多台机器的ip地址。
3. 多台机器做，爬取效率更高。

分布式爬虫必须要解决的问题：

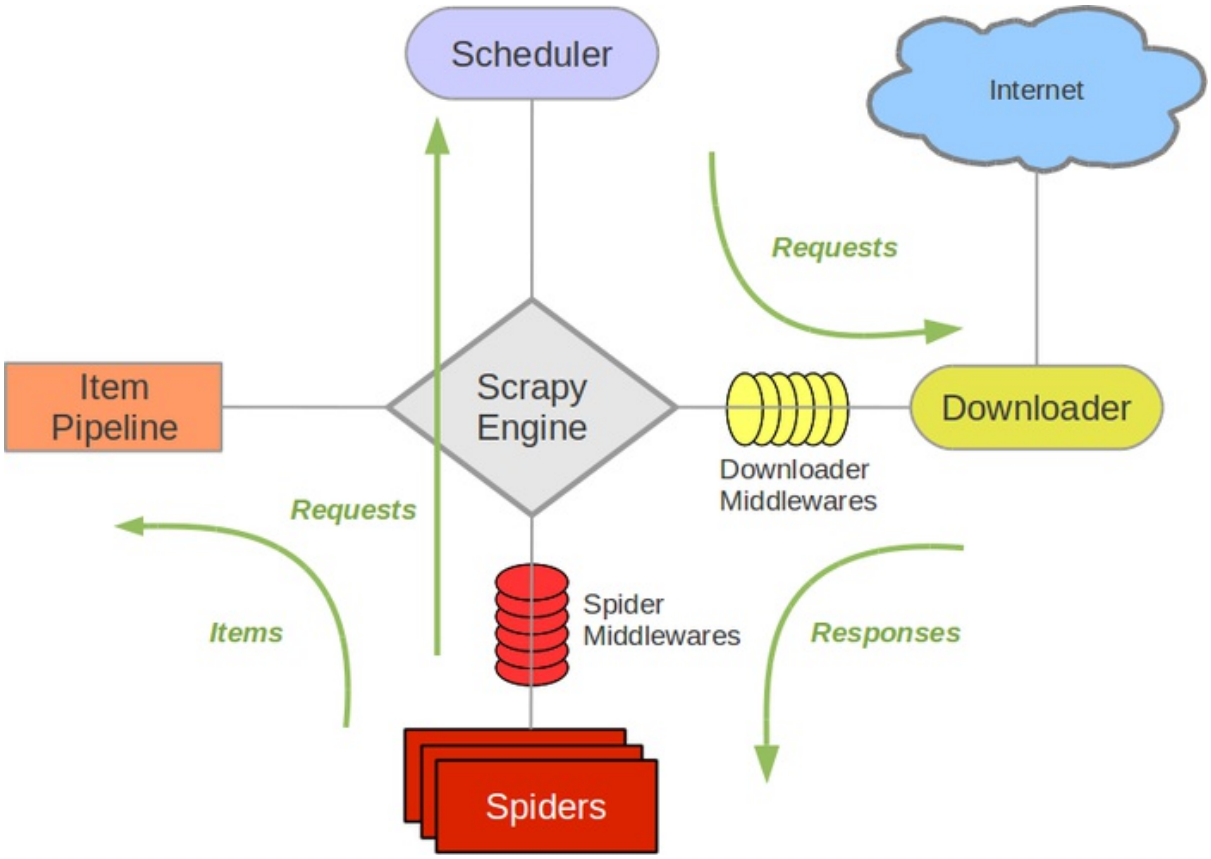
1. 分布式爬虫是好几台机器在同时运行，如何保证不同的机器爬取页面的时候不会出现重复爬取的问题。
2. 同样，分布式爬虫在不同的机器上运行，在把数据爬完后如何保证保存在同一个地方。

安装：

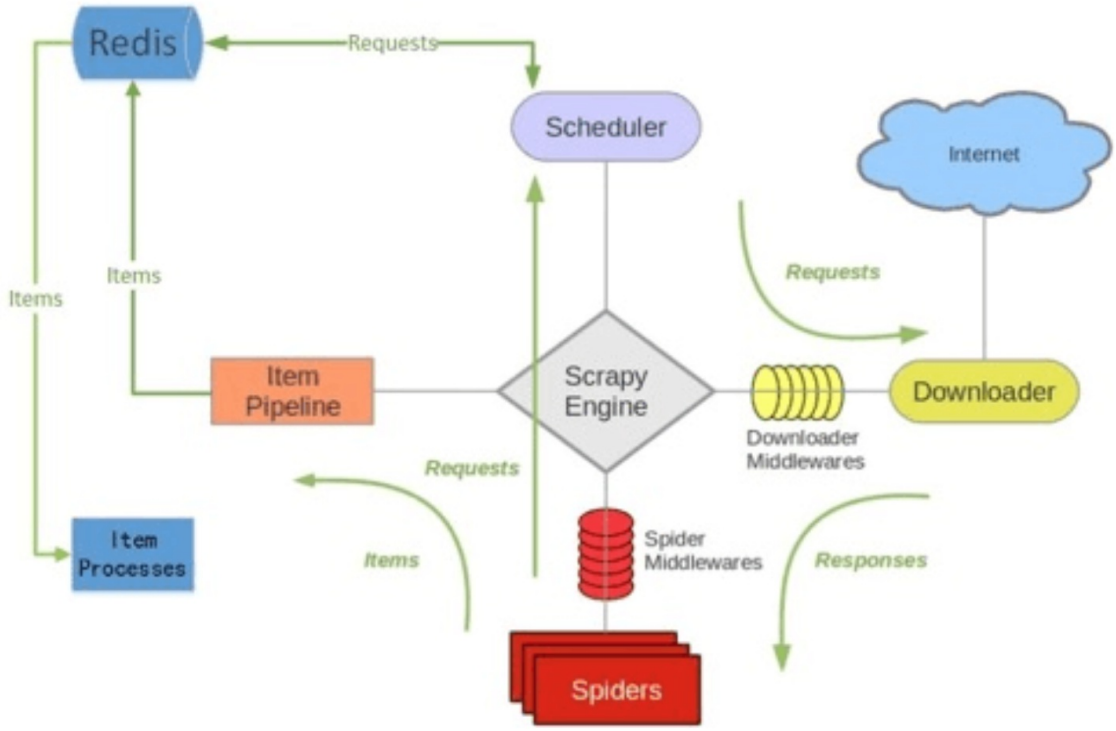
通过`pip install scrapy-redis`即可安装。

Scrapy-Redis架构：

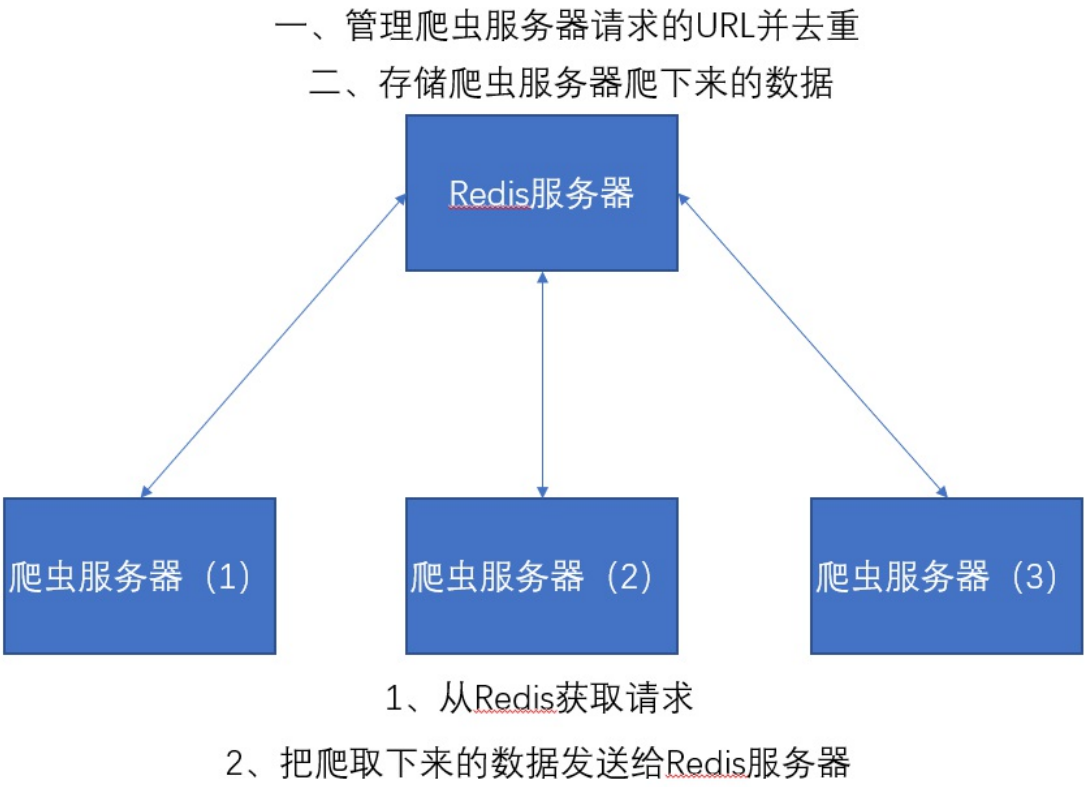
Scrapy架构图：



Scrapy-Redis架构图：



分布式爬虫架构图：



以上两个图片对比我们可以发现。`Item Pipeline`在接收到数据后发送给了`Redis`、`Scheduler`调度器调度数据也是从`Redis`中来的、并且其实数据去重也是在`Redis`中做的。

编写Scrapy-Redis分布式爬虫：

要将一个`Scrapy`项目变成一个`Scrapy-redis`项目只需修改以下三点就可以了：

1. 将爬虫的类从`scrapy.Spider`变成`scrapy_redis.spiders.RedisSpider`；或者是从`scrapy.CrawlSpider`变成`scrapy_redis.spiders.RedisCrawlSpider`。
2. 将爬虫中的`start_urls`删掉。增加一个`redis_key="xxx"`。这个`redis_key`是为了以后在`redis`中控制爬虫启动的。爬虫的第一个url，就是在redis中通过这个发送出去的。
3. 在配置文件中增加如下配置：

```
# Scrapy-Redis相关配置
# 确保request存储到redis中
SCHEDULER = "scrapy_redis.scheduler.Scheduler"

# 确保所有爬虫共享相同的去重指纹
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

# 设置redis为item pipeline
ITEM_PIPELINES = {
```

```
        'scrapy_redis.pipelines.RedisPipeline': 300
    }

    # 在redis中保持scrapy-redis用到的队列，不会清理redis中的队列，从而可以实现暂停和恢复的功能。
    SCHEDULER_PERSIST = True

    # 设置连接redis信息
    REDIS_HOST = '127.0.0.1'
    REDIS_PORT = 6379
```

1. 运行爬虫：
    i. 在爬虫服务器上。进入爬虫文件所在的路径，然后输入命令：`scrapy runspider [爬虫名字]`。
    ii. 在`Redis`服务器上，推入一个开始的url链接：`redis-cli> lpush [redis_key] start_url`开始爬取。