掌握定向网络数据爬取和网页解析的基本能力

常用的 Python IDE 工具

文本工具类 IDE	集成工具类 IDE
IDLE	PyCharm
Notepad++	Wing
Sublime Text *	PyDev & Eclipse
Vim & Emacs	Visual Studio
Atom	Anaconda & Spyder
Komodo	Canopy

第一周内容导学

Requests库→robots.txt（网络爬虫排除标准）→Projects（实战项目）

Requests库入门

```
pip install scrapy -i https://pypi.douban.com/simple
```

阿里云 https://mirrors.aliyun.com/pypi/simple/

中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/

豆瓣(douban) https://pypi.douban.com/simple/

清华大学 https://pypi.tuna.tsinghua.edu.cn/simple/

中国科学技术大学 https://pypi.mirrors.ustc.edu.cn/simple/

Requests库的安装：

获取网页：r = requests.get(url,)

状态码：r.status_code，200是访问成功

编码：r.encoding = 'utf-8'

网页内容：r.text

Requests库的7个主要方法：

requuests.request()	构造一个请求，支撑以下各方法的基础方法
requests.get()	获取HTML网页的主要方法，对应于HTTP的GET
requests.head()	获取HTML网页头信息的方法，对应于HTTP的HEAD

requests.post()	向HTML网页提交POST请求的方法，对应于HTTP的POST
requests.put()	向HTML网页提交PUT请求的方法，对应于HTTP的PUT
requests.patch()	向HTML网页提交局部修改请求，对应于HTTP的PATCH
requests.delete()	向HTML网页提交删除请求，对应于HTTP的DELETE

Requests库的get()方法：

r = requests.get(url) 构造一个向服务器请求资源的Request对象，返回一个包含服务器资源的Response对象(r)

requests.get(url,params=None,**kwargs)

url: 拟获取页面的url链接

params：url中的额外参数，字典或字节流格式，可选

**kwargs：12个控制访问的参数

Response对象的属性

r.status_code	HTTP请求的返回状态，200表示连接成功，404表示失败
r.text	HTTP响应内容的字符串形式，即 url对应的页面内容
r.encoding	从HTTP header中猜测的响应内容编码方式
r.apparent_encoding	从内容中分析出的响应内容编码方式（备选编码方式）
r.content	HTTP响应内容的二进制形式

r.encoding: 如果header中不存在charset，则认为编码为ISO-8859-1

爬取网页的通用代码框架：

理解Requests库的异常

requests.ConnectionError	网络连接错误异常，如DNS查询失败、拒绝连接等
requests.HTTPError	HTTP错误异常
requests.URLRequired	URL缺失异常
requests.TooManyRedirects	超过最大重定向次数，产生重定向异常
requests.ConnectTimeout	连接远程服务器超时异常
requests.Timeout	请求URL超时，产生超时异常
r.raise_for_status()	如果不是200，产生异常 requests.HTTPError

爬取网页的通用代码框架:

```python
import requests

def getHTMLText(url):
    try:                                                # 网络连接有风险，异常处理很重要
        r = requests.get(url, timeout=30)
        r.raise_for_status()    #如果状态不是200，引发HTTPError异常
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return "产生异常"

if __name__ == "__main__":
    url = "http://www.baidu.com"
    print(getHTMLText(url))
```

HTTP协议及Requests库方法

HTTP，Hypertext Transfer Protocol，超文本传输协议

HTTP是一个基于“请求与响应”模式的、无状态的应用层协议

HTTP协议采用URL作为定位网络资源的标识

URL格式 http://host[:port][path]

host：合法的Internet主机域名或IP地址

port：端口号，缺省端口为80

path：请求资源的路径

URL是通过HTTP协议存取资源的Internet路径，一个URL对应一个数据资源

HTTP协议对资源的操作：

方法	说明
GET	请求获取URL位置的资源
HEAD	请求获取URL位置资源的响应消息报告，即获得该资源的头部信息
POST	请求向URL位置的资源后附加新的数据
PUT	请求向URL位置存储一个资源，覆盖原URL位置的资源
PATCH	请求局部更新URL位置的资源，即改变该处资源的部分内容
DELETE	请求删除URL位置存储的资源

Requests库的post()方法：

put()方法类似

```
payload = {'key1':'value1','key2':'value2'}
r = requests.post(url, data = payload)      # 向URL POST一个字典，自动编码为form（表单
r = requests.post(url, data = 'ABC')        # 向URL POST一个字符串，自动编码为data
```

Requests库主要方法解析

requests.request(method, url, **kwargs)

method: 请求方式，对应GET/ HEAD/ POST/ PUT/ PATCH/ delete/ OPTIONS等7种

例：r = requests.request('GET', url, **kwargs), OPTIONS主要是获取参数

url：拟获取页面的url链接

**kwargs：控制访问的参数，共13个，均为可选项

params：字典或字节序列，作为参数增加到url中

```
>>> kv = {'key1':'value1','key2':'value2'}
>>> r = requests.request('Get', 'http://python123.io/ws', params=kv)
>>> print(r.url)
https://python123.io/ws?key1=value1&key2=value2
```

data：字典、字节序列或文件对象，作为Request的内容

```
kv = {'key1':'value1','key2':'value2'}
r = requests.request('POST','http://python123.io/ws', data=kv)
body = '主体内容'
r = requests.request('POST','http://python123.io/ws', data=body)
```

json：JSON格式的数据，作为Request的内容

headers：字典，HTTP定制头

```
hd = {'user-agent': 'Chrome/10'}
r = requests.request('POST', 'http://python123.io/ws', headers=hd)
```

```
# 访问时服务器看到的user-agent字段的值是Chrome/10（Chrome浏览器的第十个版本）
# 也就是说我们可以模拟任何我们想模拟的浏览器向服务器发起访问
```

cookies：字典或CookieJar, Requests中的cookie

auth：元组，支持HTTP认证功能

files：字典类型，传输文件

```
fs = {'file': open('data.xls', 'rb')}       # 向某个链接提交某个文件
r = requests.request('POST', 'http://python123.io/ws', files=fs)
```

timeout：设定超时时间，秒为单位

proxies：字典类型，设定访问代理服务器，可以增加登录认证

```
pxs = {'http': 'http://user:pass@10.10.10.1:1234',
       'https': 'http://10.10.10.1:4321'}
r = requests.request('GET', 'http://www.baidu.com', proxies=pxs)
# 在增加的http代理中可以增加用户名和密码的设置，在访问百度时使用的IP地址是
# 代理服务器的IP地址，隐藏用户信息，防止爬虫的逆追踪
```

allow_redirects：True/Flase，默认为True，重定向开关

stream：True/False，默认为True，获取内容立即下载开关

verify：True/False，默认为True，认证SSL证书开关

cert：本地SSL证书路径

requests.get(url,params=None,**kwargs)

requests.head(url, **kwarg)　**kwarg：13个控制访问参数

requests.post(url, data=None, json=None, **kwargs)　　11个控制访问参数

requests.put(url, data=None, **kwargs)　　12个控制访问参数

requests.patch(url, data=None, **kwargs)　　12个控制访问参数

requests.delete(url, **kwargs)　　13个控制访问参数

网络爬虫的盗亦有道

网络爬虫的尺寸

爬取网页，玩转网页：小规模，数据量小，爬取速度不敏感；Requests库

爬取网站 爬取系列网站：中规模，数据量较大，爬取速度敏感；Scrapy库

爬取全网：大规模，搜索引擎，爬取速度关键；定制开发

网络爬虫引发的问题

网络爬虫的骚扰：受限于编写水平和目的，网络爬虫将会为Web服务器带来巨大的资源开销

网络爬虫的法律风险：服务器上的数据有产权归属，网络爬虫获取数据后牟利将带来法律风险

网络爬虫泄露隐私：网络爬虫可能具备突破简单访问控制的能力，获得被保护数据从而泄露个人隐私

网络爬虫的限制

来源审查：判断User-Agent进行限制

- 检查来访HTTP协议头的User-Agent域，只响应浏览器或友好爬虫的访问

发布公告：Robots协议

- 告知所有爬虫网站的爬取策略，要求爬虫遵守

Robots协议

Robots Exclusion Standard 网络爬虫排除标准

作用：网站告知网络爬虫哪些页面可以抓取，哪些不行

形式：在网站根目录下的robots.txt文件

Robots协议基本语法：

https://www.jd.com/robots.txt

```
User-agent: *              # 注释：
Disallow: /?*              *代表所有
Disallow: /pop/*.html      /代表根目录
```

Robots协议的遵守方式

网络爬虫使用Robots协议：自动或人工识别robots.txt，再进行内容爬取

约束性：Robots协议是建议但非约束性，网络爬虫可以不遵守，但存在法律风险

Requests库爬取实例

实例1：京东商品页面的爬取

```python
import requests
url = "https://item.jd.com/100004770249.html"
try:
    r = requests.get(url)
    r.raise_for_status()
    r.encoding = r.apparent_encoding
    print(r.text[:1000])
except:
    print("爬取失败")
```

实例2：亚马逊商品页面的爬取

```python
import requests
url = "https://www.amazon.cn/dp/B01MYH8A99"
try:
    kv = {'user-agent':'Mozilla/5.0'}        # 标准的浏览器身份标识
    r = requests.get(url,headers=kv)         # 模拟浏览器访问
    r.raise_for_status()
    r.encoding = r.apparent_encoding         # r.request.headers 查看报头信息
    print(r.text[1000:2000])
except:
    print("爬取失败")
```

实例3：百度360搜索关键词提交

百度的关键词接口： http://www.baidu.com/s?wd=keyword

360的关键词接口： http://www.so.com/s?q=keyword

```python
import requests
keyword = "Python"
try:
    kv = {'wd':keyword}
    r = requests.get("http://www.baidu.com/s",params=kv)
    print(r.request.url)
    r.raise_for_status()
    print(len(r.text))
except:
    print("爬取失败")
```

实例4：网络图片的爬取和存储

网络图片链接的格式：http://www.example.com/picture.jpg

图片爬取代码

```
import requests
import os
url = "http://img0.dili360.com/ga/M00/48/F7/wKgBy1llvmCAAQOVADC36j6n9bw622.tub.jpg"
root = "E://图片//爬虫//"
path = root + url.split('/')[-1]
try:
    if not os.path.exists(root):
        os.mkdir(root)
    if not os.path.exists(path):
        r = requests.get(url)
        r.raise_for_status()
        with open(path, 'wb') as f:
            f.write(r.content)      # r.content表示返回内容的二进制形式，
            f.close()               # 图片是以二进制形式存储的
            print("文件保存成功")
    else:
        print("文件已存在")
except:
    print("爬取失败")
```

实例5：IP地址归属地的自动查询

ip138 IP查询：http://m.ip138.com/ip.asp?ip=ipaddress

```
import requests
url = "http://m.ip138.com/ip.asp?ip="
try:
    r = requests.get(url + '202.204.80.112')
    r.raise_for_status()
    r.encoding = r.apparent_encoding
    print(r.text[:1000])
except:
    print("爬取失败")
```

第二周内容导学

Beautiful Soup：解析HTML页面信息标记与提取方法

Projects：实战项目—中国大学排名爬虫

Beautiful Soup库的安装

https://www.crummy.com/software/BeautifulSoup/

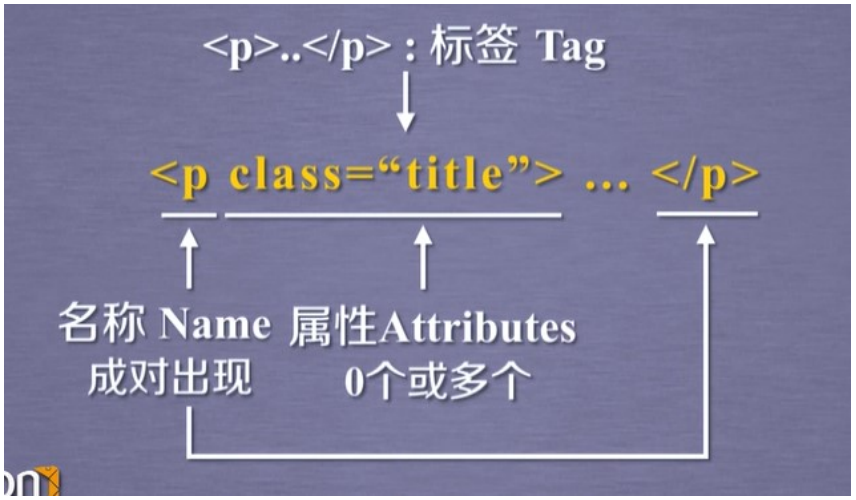
演示HTML页面地址：https://python123.io/ws/demo.html

```
<html><head><title>This is a python demo page</title></head>
<body>
<p class="title"><b>The demo python introduces several python courses.</b></p>
<p class="course">Python is a wonderful general-purpose programming language. You c
<a href="http://www.icourse163.org/course/BIT-268001" class="py1" id="link1">Basic
</body></html>
```

```
from bs4 import BeautifulSoup                          # BeautifulSoup是一个类，第一个参
soup = BeautifulSoup('<p>data</p>', 'html.parser')  # 第二个参数是需要用的解析器
```

Beautiful Soup库的基本元素

Beautiful Soup库的理解：它是解析、遍历、维护“标签树”的功能库



Beautiful Soup库的引用：Beautiful Soup库，也叫beautifulsoup4或bs4

Beautiful Soup类：HTML文档↔标签树↔BeautifulSoup类

```
from bs4 import BeautifulSoup
soup = BeautifulSoup('<p>data</p>', 'html.parser')
soup2 = BeautifulSoup(open("D://demo.html"), "html.parser") # 打开一个html文件
```

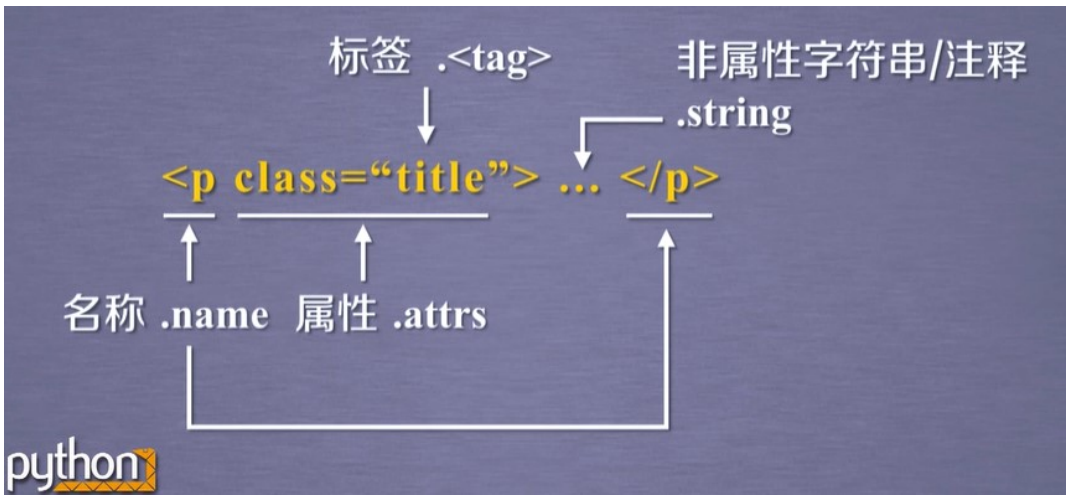
BeautifulSoup对应一个HTML/XML文档的全部内容

Beautiful Soup库解析器：

解析器	使用方法	条件
bs4的HTML解析器	BeautifulSoup(mk, 'html.parser')	安装bs4库
lxml的HTML解析器	BeautifulSoup(mk, 'lxml')	pip install lxml
lxml的XML解析器	BeautifulSoup(mk, 'xml')	pip install lxml
html5lib的解析器	BeautifulSoup(mk, 'html5lib')	pip install html5lib

Beautiful Soup类的基本元素：

基本元素	说明
Tag	标签，最基本的信息组织单元，分别用<>和</>标明开头和结尾
Name	标签的名字， ... 的名字是'p'，格式：<tag>.name
Attributes	标签的属性，字典形式组织，格式：<tag>.attrs
NavigableString	标签内非属性字符串，<>...</>中字符串，格式：<tag>.string
Comment	标签内字符串的注释部分，一种特殊的Comment类型



基于bs4库的HTML内容遍历方法

上行遍历， 下行遍历， 平行遍历

标签树的下行遍历：

属性	说明

.contents	子节点的列表，将<tag>所有儿子节点存入列表
.children	子节点的迭代类型，与.contents类似，用于循环遍历儿子节点
.descendants	子孙节点的迭代类型，包含所有子孙节点，用于循环遍历

```
for child in soup.body.children:
    print(child)                    # 遍历儿子节点
for child in soup.body.descendants:
    print(child)                    # 遍历子孙节点
```

标签树的上行遍历：

属性	说明
.parent	节点的父亲标签
.parents	节点先辈标签的迭代类型，用于循环遍历先辈节点

```
soup = BeautifulSoup(demo, "html.parser")
for parent in soup.a.parents:       # 遍历soup的a标签的先辈标签
    if parent is None:
        print(parent)
    else:
        print(parent.name)
```

标签树的平行遍历：

属性	说明
.next_sibling	返回按照HTML文本顺序的下一个平行节点标签
.previous_sibling	返回按照HTML文本顺序的上一个平行节点标签
.next_siblings	迭代类型，返回按照HTML文本顺序的后续所有平行结点标签
.previous_siblings	迭代类型，返回按照HTML文本顺序的前续所有平行节点标签

平行遍历发生在同一个父节点下的各节点间

```
for sibling in soup.a.next_siblings:        # 遍历后续节点
    print(sibling)
```

```
for sibling in soup.a.previous_siblings:    # 遍历前续节点
    print(sibling)
```

基于bs4库的HTML格式输出

如何能让HTML页面更加友好的显示?

```
>>> soup = BeautifulSoup("<p>中文</p>", "html.parser")
>>> soup.p.string                    # 默认使用UTF-8编码
'中文'
>>> print(soup.p.prettify())
<p>
 中文
</p>
```

信息标记的三种形式

信息的标记:

- 标记后的信息可形成信息组织结构，增加了信息维度
- 标记后的信息可用于通信、存储和展示
- 标记的结构与信息一样具有重要价值
- 标记后的信息有利于程序理解和运用

HTML的信息标记：HTML通过预定义的<>...</>标签形式组织不同类型的信息

信息标记的三种形式：XML，JSON，YAML

```
  XML               JSON                            YMAL
<name>...</name>    "key": "value"                  key: value          key: #Comme
<name />            "key": ["value1", "value2"]     key:                -value1
<!--   -->          "key": {"subkey", "subvalue"}     subkey: subvalue  -value2
```

三种信息标记形式的比较

XML实例:

```
<person>
   <firstName>Tian</firstName>
   <lastName>Song</lastName>
   <address>
        <streetAddr>中关村南大街5号</streetAddr>
        <city>北京市</city>
        <zipcode>100081</zipcode>
   </address>
   <prof>Computer System</prof><prof>Security</prof>
</person>
```

JSON实例：

```
{
   "firstName" : "Tian" ,
   "lastName"  : "Song" ,
   "address"   : {
                   "streetAddr" : "中关村南大街5号" ,
                   "city"       : "北京市" ,
                   "zipcode"    : "100081"
                 } ,
   "prof"      : [ "Computer System" , "Security" ]
}
```

YAML实例：

```
firstName : Tian
lastName  : Song
address   :
    streetAddr : 中关村南大街5号
    city       : 北京市
    zipcode    : 100081
prof      :
-Computer System
-Security
```

XML 最早的通用信息标记语言，可扩展性好，但繁琐；Internet上的信息交互于传递

JSON 信息有类型，适合程序处理（js），较XML简洁；移动应用云端和节点的信息通信，无注释

YAML 信息无类型，文本信息比例最高，可读性好；各类系统的配置文件，有注释易读

信息提取的一般方法

方法一：完整解析信息的标记形式，再提取关键信息

需要标记解析器 例如：bs4库的标签树遍历

优点：信息解析准确

缺点：提取过程繁琐，速度慢

方法二：无视标记形式，直接搜索关键信息

对信息的文本查找函数即可

优点：提取过程简洁，速度较快

缺点：提取结果准确性与信息内容相关

融合方法：结合形势解析与搜索方法，提取关键信息

需要标记解析器及文本查找函数

实例：提取HTML中所有的URL链接

思路：1）搜索到所有<a>标签

2）解析<a>标签格式，提取href后的链接内容

```
import requests
r = requests.get("https://python123.io/ws/demo.html")
demo = r.text
from bs4 import BeautifulSoup
soup = BeautifulSoup(demo, "html.parser")
for link in soup.find_all('a'):
    print(link.get('href'))
>>> http://www.icourse163.org/course/BIT-268001
>>> http://www.icourse163.org/course/BIT-1001870001
```

基于bs4库的HTML内容查找方法

<>.find_all(name, attrs, recursive, string, **kwargs)

返回一个列表类型，存储查找的结果

name：对标签名称的检索字符串　soup.find_all(['a','b'])

attrs：对标签属性值的检索字符串，可标注属性检索　soup.find_all('p', 'course')
soup.find_all(id='link1')

recursive：是否对子孙全部检索，默认True　soup.find_all('a', recursive=False)

string：<>...</>中字符串区域的检索字符串　soup.find_all(string = re.compile('python'))

<tag>(...) 等价于 <tag>.find_all(...)　　soup(...) 等价于 soup.find_all(...)

扩展方法	说明
<>.find()	搜索且只返回一个结果，字符串类型，同 .find_all()参数
<>.find_parents()	在先辈节点中搜索，返回列表类型，同 .find_all()参数
<>.find_parent()	在先辈节点中返回一个结果，字符串类型，同 .find_all()参数

<>.find_next_siblings()	在后续平行节点中搜索，返回列表类型，同 .find_all()参数
<>.find_next_sibling()	在后续平行节点中返回一个结果，字符串类型，同 .find_all()参数
<>.find_previous_siblings()	在前序平行节点中搜索，返回列表类型，同 .find_all()参数
<>.find_previous_sibling()	在前序平行节点中返回一个结果，字符串类型，同 .find_all()参数

“中国大学排名定向爬虫”实例介绍

最好大学网：http://www.zuihaodaxue.cn/zuihaodaxuepaiming2016.html

功能描述

输入：大学排名URL链接

输出：大学排名信息的屏幕输出（排名，大学名称，总分）

技术路线：requests-bs4

定向爬虫：仅对输入URL进行爬取，不扩展爬取

程序的结构设计

步骤1：从网络上获取大学排名网页内容　　getHTMLText()

步骤2：提取网页内容中信息到合适的数据结构　　fillUnivList()

步骤3：利用数据结构展示并输出结果　　printUnivList()

“中国大学排名定向爬虫”实例编写

```
import requests
from bs4 import BeautifulSoup
import bs4

def getHTMLText(url):
    try:
        r = requests.get(url, timeout = 30)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""

def fillUnivList(ulist, html):
    soup = BeautifulSoup(html, "html.parser")
    for tr in soup.find('tbody').children:
        if isinstance(tr, bs4.element.Tag):     # 检测tr的类型是否是Tag标签
```

```
            tds = tr('td')
            ulist.append([tds[0].string, tds[1].string, tds[2].string])


def printUnivList(ulist, num):
    tplt = "{0:^10}\t{1:{3}^10}\t{2:^10}"
    print(tplt.format("排名", "学校名称", "总分", chr(12288)))
    for i in range(num):
        u = ulist[i]
        print(tplt.format(u[0], u[1], u[2],chr(12288)))

def main():
    uinfo = []
    url = 'http://www.zuihaodaxue.cn/zuihaodaxuepaiming2016.html'
    html = getHTMLText(url)
    fillUnivList(uinfo, html)
    printUnivList(uinfo, 20)  # 20 univs

main()
```

“中国大学排名定向爬虫”实例优化

中文对齐问题的原因：

:	<填充>	<对齐>	<宽度>	,	<精度>	<类型>
引号符号	用于填充的单个字符	<左对齐 >右对齐 ^居中对齐	槽的设定输出宽度	数字的千位分隔符适用于整数和浮点数	浮点数小数部分的精度或字符串的最大输出长度	整数类型 b,c,d,o,x,X 浮点数类型 e,E,f,%

当中文字符宽度不够时，采用西文字符填充；中西文字符占用宽度不同

采用中文字符的空格填充 chr(12288)

第三周内容导学

正则表达式详解提取页面关键信息 Re

正则表达式的概念

正则表达式：regular expression regex　RE

正则表达式是用来简洁表达一组字符串的表达式

- 通用的字符串表达框架

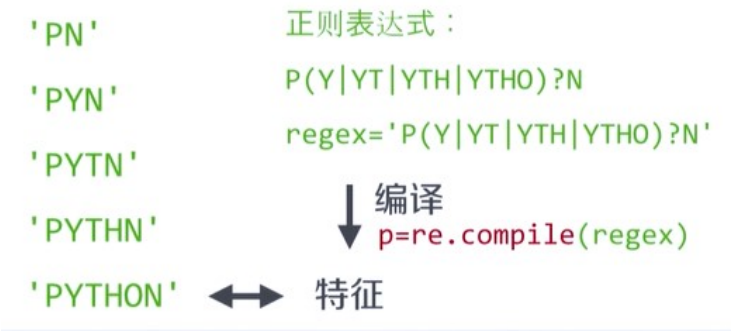
- 简洁表达一组字符串的表达式
- 针对字符串表达“简洁”和“特征”思想的工具
- 判断某字符串的特征归属

正则表达式在文本处理中十分常用

- 表达文本类型的特征（病毒、入侵等）
- 同时查找或替换一组字符串
- 匹配字符串的全部或部分

正则表达式的使用

- 编译：将符合正则表达式语法的字符串转换成正则式表达特征



正则表达式的语法

正则表达式语法由字符和操作符构成

正则表达式的常用操作符

操作符	说明	实例
.	表示任何单个字符	
[ ]	字符集，对单个字符给出取值范围	[abc]表示a、b、c，[a-z]表示a到z单个字符
[^]	非字符集，对单个字符给出排除范围	[^abc]表示非a或b或c的单个字符
*	前一个字符0次或无限次扩展	abc* 表示 ab、abc、abcc、abccc等
+	前一个字符1次或无限次扩展	abc+ 表示 abc、abcc、abccc等
?	前一个字符0次或1次扩展	abc? 表示 ab、abc
|	左右表达式任意一个	abc|def 表示 abc、def
{m}	扩展前一个字符m次	ab{2}c 表示 abbc
{m,n}	扩展前一个字符m至n次（含n）	ab{1,2}c 表示 abc、abbc
^	匹配字符串开头	^abc 表示abc且在一个字符串的开头
$	匹配字符串结尾	abc$ 表示abc且在一个字符串的结尾

( )	分组标记，内部只能使用 | 操作符	(abc) 表示abc，(abc|def) 表示 abc、def
\d	数字，等价于[0-9]	
\w	单词字符，等价于[A-Za-z0-9_]	

正则表达式实例

正则表达式	对应字符串
P(Y|YT|YTH|YTHO)?N	'PN'、'PYN'、'PYTN'、'PYTHN'、'PYTHON'
PYTHON+	'PYTHON'、'PYTHONN'、'PYTHONNN' …
PY[TH]ON	'PYTON'、'PYHON'
PY[^TH]?ON	'PYON'、'PYaON'、'PYbON'、'PYcON'…
PY{:3}N	'PN'、'PYN'、'PYYN'、'PYYYN'

经典正则表达式实例

^[A-Za-z]+$	由26个字母组成的字符串
^[A-Za-z0-9]+$	由26个字母和数字组成的字符串
^-?\d+$	整数形式的字符串
^[0-9]*[1-9][0-9]*$	正整数形式的字符串
[1-9]\d{5}	中国境内邮政编码，6位
[\u4e00-\u9fa5]	匹配中文字符
\d{3}-\d{8}|\d{4}-\d{7}	国内电话号码，010-68913536

匹配IP地址的正则表达式

IP地址字符串形式的正则表达式（IP地址分分4段，每段0-255）

\d+.\d+.\d+.\d+　　\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}　不精确

精确写法

　　0-99: [1-9]?\d　100-199: 1\d{2}　200-249: 2[0-4]\d　　250-255: 25[0-5]

(([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5]).){3}([1-9]?\d|1\d{2}|2[0-4]\d|25[0-5])

Re库的基本使用

Re库介绍：Re库是Python的标准库，主要用于字符串匹配

正则表达式的表示类型

- raw string类型（原生字符串类型）

re库采用raw string类型表示正则表达式，表示为：r ' text '

例如：r ' [1-9]\d{5} '　r '\d{3}- \d{8}|\d{4}-\d{7}'

raw string 是不包含转义符的字符串

- string 类型，更繁琐，要转义

Re库的主要功能函数

函数	说明
re.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象
re.match()	从一个字符串的开始位置起匹配正则表达式，返回match对象
re.findall()	搜索字符串，以列表类型返回全部能匹配的子串
re.split()	将一个字符串按照正则表达式匹配结果进行分割。返回列表类型
re.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象
re.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串

re.search(pattern, string,flags=0)

- pattern：正则表达式的字符串或原生字符串表示
- string：待匹配字符串
- flags：正则表达式使用时的控制标记

常用标记	说明
re.I re.IGNORECASE	忽略正则表达式的大小写， [A-Z]能够匹配小写字符
re.M re.MULTILINE	正则表达式中的^操作符能够将给定字符串的每行当作匹配开始
re.S re.DOTALL	正则表达式中的. 操作符能够匹配所有字符，默认匹配换行除外的所有字符

re.match(pattern, string,flags=0) 各参数与前述相同

re.findall(pattern, string, flags=0) 各参数与前述相同

re.split(pattern, string, maxsplit=0, flags=0)

- 最大分割数，剩余部分作为最后一个元素输出

re.sub(pattern, repl,string,count=0,flags=0)

- repl：替换匹配字符串的字符串
- count：匹配的最大替换次数

```
match = re.search(r'[1-9]\d{5}','BIT 100081')
if match:
    print(match.group(0))                                    #  100081
match = re.match(r'[1-9]\d{5}','100081 BIT')
if match:
    match.group(0)                                           #  100081
ls = re.findall(r'[1-9]\d{5}', 'BIT 100081 TSU100084')       #  ls ['100081', '1
re.split(r'[1-9]\d{5}', 'BIT 100081 TSU100084')              #  ['BIT ', ' TSU',
re.split(r'[1-9]\d{5}', 'BIT 100081 TSU100084', maxsplit=1)  # ['BIT ', ' TSU100
for m in re.finditer(r'[1-9]\d{5}', 'BIT100081 TSU100084'):
    if m:                                                    # 100081
        print(m.group(0))                                    # 100084
re.sub(r'[1-9]\d{5}', ':zipcode','BIT100081 TSU100084')      # 'BIT:zipcode TSU:
```

Re库的另一种等价用法

```
rst = re.search(r'[1-9]\d{5}','BIT 100081')         # 函数式用法：一次性操作
pat = re.compile(r'[1-9]\d{5}')                     # 面向对象用法：编译后的多次操作
rst = pat.search('BIT 100081')
```

regex = re.compile(pattern, flags=0)　将正则表达式的字符串形式编译成正则表达式对象

- pattern：正则表达式的字符串或原生字符串表示
- flags：正则表达式使用时的控制标记

函数	说明
regex.search()	在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象
regex.match()	从一个字符串的开始位置起匹配正则表达式，返回match对象
regex.findall()	搜索字符串，以列表类型返回全部能匹配的子串
regex.split()	将一个字符串按照正则表达式匹配结果进行分割。返回列表类型
regex.finditer()	搜索字符串，返回一个匹配结果的迭代类型，每个迭代元素是match对象
regex.sub()	在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串

Re库的match对象

match对象就是一次匹配的结果，包含了匹配的相关信息

Match对象的属性

属性	说明
.string	待匹配的文本
.re	匹配时使用的pattern对象（正则表达式）
.pos	正则表达式搜索文本的开始位置
.endpos	正则表达式搜索文本的结束位置

Match对象的方法

方法	说明
.group(0)	获得一次匹配后的字符串
.start()	匹配字符串在原始字符串的开始位置
.end()	匹配字符串在原始字符串的结束位置
.span()	返回(.start(), .end())

```
m = re.search(r'[1-9]\d{5}','BIT100081 TSU100084')
match.string        # 'BIT 100081 TSU 100084'
m.group(0)          # '100081'  返回一次匹配的结果，查看全部用finditer()
m.re                # re.compile('[1-9]\\d{5}')
m.pos               # 0
m.endpos            # 19
m.start()           # 3
m.end()             # 9
m.span()            # (3, 9)
```

Re库的贪婪匹配和最小匹配

同时匹配长短不同的多项，返回哪一个?

Re库默认采用贪婪匹配，即输出匹配最长的子串

最小匹配：如何输出最短的字串?

最小匹配操作符

操作符	说明
*?	前一个字符0次或无限次扩展，最小匹配
+?	前一个字符1次或无限次扩展，最小匹配

??	前一个字符0次或1次扩展，最小匹配
{m,n}?	扩展前一个字符m至n次（含n），最小匹配

```
match = re.search(r'PY.*N', 'PYANBNCNDN')       # PY开头，N结尾的字符串
match.group(0)          # 'PYANBNCNDN'
match = re.search(r'PY.*?N', 'PYANBNCNDN')
match.group(0)          # 'PYAN'
```

“淘宝商品信息定向爬虫”实例介绍

功能描述

目标：获取淘宝搜索页面的信息，提取其中的商品名称和价格。

理解：淘宝的搜索接口

翻页的处理

技术路线：requests-re

程序的结构设计

步骤1：提交商品搜索请求，循环获取页面

步骤2：对于每个页面，提取商品名称和价格信息

步骤3：将信息输出到屏幕上

“淘宝商品信息定向爬虫”实例编写

```
import requests
import re

def getHTMLText(url):
    try:
        r = requests.get(url, timeout = 30)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""

def parsePage(ilt, html):
    try:
```

```
        plt = re.findall(r'\"view_price\":\"[\d\.]*\"',html)
        tlt = re.findall(r'\"raw_title\":\".*?\"',html)
        for i in range(len(plt)):
            price = eval(plt[i].split(':')[1])
            title = eval(tlt[i].split(':')[1])
            ilt.append([price, title])
    except:
        print("")

def printGoodsList(ilt):
    tplt = "{:4}\t{:8}\t{:16}"
    print(tplt.format("序号", "价格", "商品名称"))
    count = 0
    for g in ilt:
        count = count + 1
        print(tplt.format(count, g[0], g[1]))


def main():
    goods = '书包'
    depth = 2
    start_url = 'https://s.taobao.com/search?q=' + goods
    infoList = []
    for i in range(depth):
        try:
            url = start_url + '&s=' + str(44*i)
            html = getHTMLText(url)
            parsePage(infoList, html)
        except:
            continue
        printGoodsList(infoList)

main()
```

“股票数据定向爬虫”实例介绍

功能描述

目标：获取上交所和深交所所有股票的名称和交易信息

输出：保存到文件中

技术路线：requests-bs4-re

候选数据网站的选择

新浪股票：http://finance.sina.com.cn/stock/

百度股票：https://gupiao.baidu.com/stock/

候选数据网站的选择

选取原则：股票信息静态存在于HTML页面中，非js代码生成，没有Robots协议限制

选取方法：浏览器F12，源代码查看等

选取心态：不要纠结于某个网站，多找信息源尝试

程序的结构设计

步骤1：从东方财富网获取股票列表

步骤2：根据股票列表逐个到百度股票获取个股信息

步骤3：将结果存储到文件

“股票数据定向爬虫”实例编写

```
import requests
from bs4 import BeautifulSoup
import traceback
import re

def getHTMLText(url):
    try:
        r = requests.get(url, timeout = 30)
        r.raise_for_status()
        r.encoding = r.apparent_encoding
        return r.text
    except:
        return ""

def getStockList(lst, stockURL):
    html = getHTMLText(stockURL)
    soup = BeautifulSoup(html, 'html.parser')
    a = soup.find_all('a')
    for i in a:
        try:
            href = i.attrs['href']
            lst.append(re.findall(r"[s][hz]\d{6}",href)[0])
        except:
```

```
            continue

def getStockInfo(lst, stockURL, fpath):
    for stock in lst:
        url = stockURL + stock + ".html"
        html = getHTMLText(url)
        try:
            if html == "":
                continue
            infoDict = {}
            soup = BeautifulSoup(html, 'html.parser')
            stockInfo = soup.find('div', attrs={'class':'stock-bets'})

            name = stockInfo.find_all(attrs={'aclass':'bets-name'})[0]
            infoDict.update({'股票名称':name.text.split()[0]})

            keyList = stockInfo.find_all('dt')
            valueList = stockInfo.find_all('dd')
            for i in range(len(keyList)):
                key = keyList[i].text
                val = valueList[i].text
                infoDict[key] = val

            with open(fpath, 'a', encoding='utf-8') as f:
                f.write(str(infoDict) + '\n')
        except:
            traceback.print_exc()                    # 把错误信息显示出来
            continue

def main():
    stock_list_url = 'http://quote.eastmoney.com/stocklist.html'
    stock_info_url = 'https://gupiao.baidu.com/stock/'
    output_file = 'G://Python/网络爬虫//Python语言系列专题—北理工//BaiduStockInfo.txt
    slist = []
    getStockList(slist, stock_list_url)
    getStockInfo(slist, stock_info_url, output_file)

main()
```

“股票数据定向爬虫”实例优化

```
print('\r当前速度：{:.2f}%'.format(count*100/len(lst)),end='')
```

第四周内容导学

Scrapy专业爬虫框架介绍

爬虫框架的基本使用

Scrapy爬虫框架介绍

Scrapy的安装：执行pip install scrapy

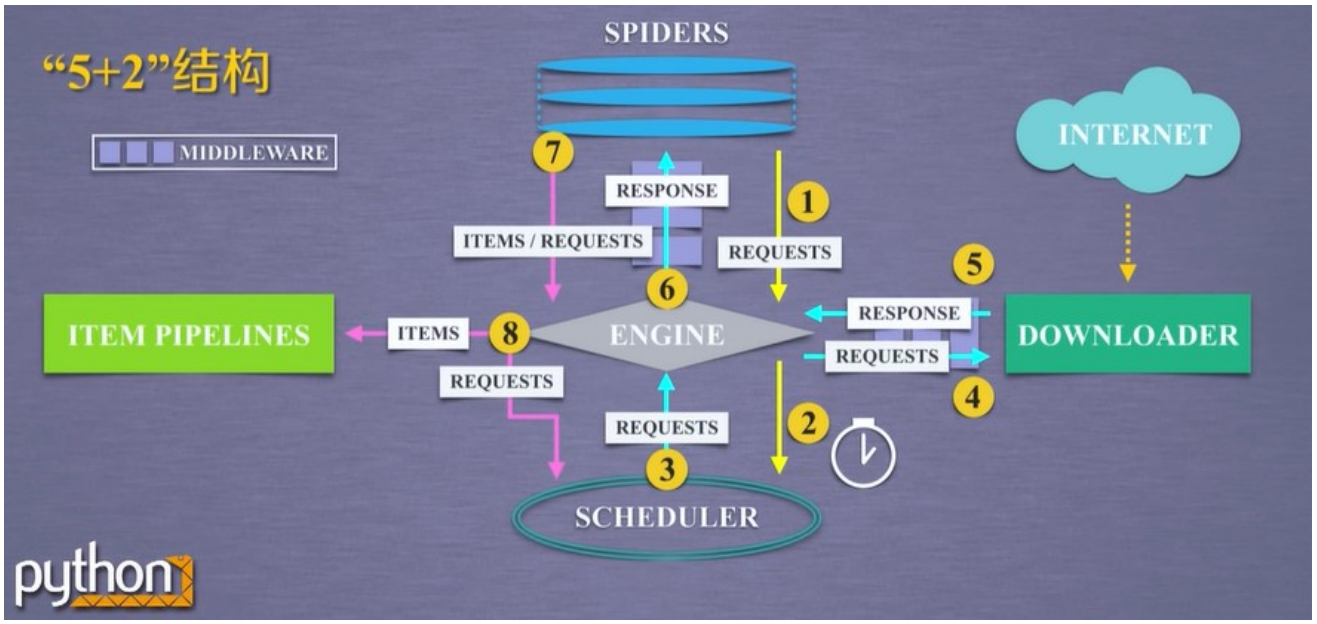
在使用pip的时候在后面加上-i参数，指定pip源
eg: pip install scrapy -i https://pypi.tuna.tsinghua.edu.cn/simple

Scrapy爬虫框架结构

- 爬虫框架：爬虫框架是实现爬虫功能的一个软件结构和功能组件集合

  爬虫框架是一个半成品，能够帮助用户实现专业网络爬虫

- “5+2”结构



Scrapy爬虫框架解析

Engine模块(不需要用户修改)：控制所有模块之间的数据流；根据条件触发事件

Downloader模块（不需要用户修改）：根据请求下载网页

Scheduler模块（不需要用户修改）：对所有爬取请求进行调度管理

Downloader Middleware中间键

目的：实施Engine、Scheduler和Downloader之间进行用户可配置的控制

功能：修改、丢弃、新增请求或响应

用户可以编写配置代码

Spider模块（需要用户编写配置代码）

- 解析Downloader返回的响应（Response）
- 产生爬取项（scraped item）
- 产生额外的爬取请求（Request）

Item Pipelines模块（需要用户编写配置代码）

- 以流水线方式处理Spider产生的爬取项
- 由一组操作顺序组成，类似流水线，每个操作是一个Item Pipeline类型
- 可能操作包括：清理、检验和查重爬取项中的HTML数据、将数据存储到数据库

Spider Middleware中间键

目的：对请求和爬取项的再处理

功能：修改、丢弃、新增请求或爬取项

用户可以编写配置代码

requests库和Scrapy爬虫的比较

requests vs. Scrapy

相同点

- 两者都可以进行页面请求和爬取，Python爬虫的两个重要技术路线
- 两者可用性都好，文档丰富，入门简单
- 两者都没用处理js、提交表单、应对验证码等功能（可扩展）

不同点

requests	Scrapy
页面级爬虫	网站级爬虫
功能库	框架
并发性考虑不足，性能较差	并发性好，性能较高
重点在于页面下载	重点在于爬虫结构
定制灵活	一般定制灵活，深度定制困难
上手十分简单	入门稍难

选用哪个技术路线开发爬虫

- 非常小的需求，requests库

- 不太小的需求，Scrapy框架
- 定制程度很高的需求（不考虑规模），自搭框架，requests>Scrapy

Scrapy爬虫的常用命令

Scrapy命令行

Scrapy是为持续运行设计的专业爬虫框架，提供操作的Scrapy命令行

Scrapy常用命令

命令	说明	格式
startproject	创建一个新工程	scrapy startproject <name> [dir]
genspider	创建一个爬虫	scrapy genspider [options] <name> <domain>
settings	获得爬虫配置信息	scrapy settings [options]
crawl	运行一个爬虫	scrapy crawl <spider>
list	列出工程中所有爬虫	scrapy list
shell	启动URL调试命令行	scrapy shell [url]

为什么Scrapy采用命令行创建和运行爬虫

- 命令行（不是图形界面）更容易自动化，适合脚本控制
- 本质上，Scrapy是给程序员用的，功能（而不是界面）更重要

Scrapy爬虫的第一个实例

演示HTML页面地址：http://python123.io/ws/demo.html

文件名称：demo.html

产生步骤

步骤1：建立一个Scrapy爬虫工程

scrapy startproject python123demo

生成的工程目录

python123demo/　　外层目录

　scrapy.cfg　　部署Scrapy爬虫的配置文件

　python123demo/　　Scrapy框架的用户自定义Python代码

　　__init__.py　　初始化脚本

　　items.py　　Items代码模板（继承类）

middlewares.py　　Middlewares代码模板（继承类）

pipelines.py　　Pipelines代码模板（继承类）

settings.py　　Scrapy爬虫的配置文件

spiders/　　Spiders代码模板目录（继承类）

__init__.py　　初始文件，无需修改

__pycache__/　　缓存目录，无需修改

步骤2：在工程中产生一个Scrapy爬虫

cd python123demo　　-->　scrapy genspider demo python123.io

demo.py文件

```
# -*- coding: utf-8 -*-
import scrapy

class DemoSpider(scrapy.Spider):
    name = 'demo'
    allowed_domains = ['python123.io']
    start_urls = ['http://python123.io/']

    def parse(self, response):
        pass
# parse()用于处理响应，解析内容形成字典，发现新的URL爬取请求
```

步骤3：配置产生的spider爬虫

```
# -*- coding: utf-8 -*-
import scrapy

class DemoSpider(scrapy.Spider):
    name = 'demo'
    #allowed_domains = ['python123.io']
    start_urls = ['http://python123.io/ws/demo.html']

    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
```

```
        self.log('Saved file %s.' % fname)
```

步骤4：运行爬虫，获取网页

scrapy crawl demo

yield关键字的使用

yield —— 生成器

- 生成器是一个不断产生值的函数
- 包含yield语句的函数是一个生成器
- 生成器每一次产生一个值（yield语句），函数被冻结，被唤醒后再产生一个值

```
def gen(n):
    for i in range(n):
        yield i**2
for i in gen(5):
    print(i, " ",end="")
0  1  4  9  16
```

生成器相比一次列出所有内容的优势：

- 更节省存储空间
- 响应更加迅速
- 使用更灵活

```
import scrapy

class DemoSpider(scrapy.Spider):
    name = 'demo'

    def start_requests(self):
        urls = [
                    'http://python123.io/ws/demo.html'
                ]
        for url in urls:
            yield scrapy.Request(url=url, classback=self.parse)

```

```
    def parse(self, response):
        fname = response.url.split('/')[-1]
        with open(fname, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s.' % fname)
```

Scrapy爬虫的基本使用

Scrapy爬虫的使用步骤

步骤1：创建一个工程和Spider模板

步骤2：编写Spider

步骤3：编写Item Pipeline

步骤4：优化配置策略

Scrapy爬虫的数据类型

Request类；Response类：Item类

Request类

class scrapy.http.Request()

- Request对象表示一个HTTP请求
- 由Spider生成，由Downloader执行

属性或方法	说明
.url	Request对应的请求URL地址
.method	对应的请求方法，‘GET’ ‘POST’等
.headers	字典类型风格的请求头
.body	请求内容主体，字符串类型
.meta	用户添加的扩展信息，在Scrapy内部模块间传递信息使用
.copy()	复制该请求

Response类

class scrapy.http.Response()

- Response对象表示一个HTTP响应
- 由Downloader生成，由Spider处理

属性或方法	说明

.url	Response对应的URL地址
.status	HTTP状态码，默认是200
.headers	Response对应的头部信息
.body	Response对应的内容信息，字符串类型
.flags	一组标记
.request	产生Response类型对应的Requests对象
.copy()	复制该响应

Item类

class scrapy.item.Item()

- Item对象表示一个从HTML页面中提取的信息内容
- 由Spider生成，由Item Pipeline处理
- Item类似字典类型，可以按照字典类型操作

Scrapy爬虫提取信息的方法

Scrapy爬虫支持多种HTML信息提取方法

- Beautiful Soup
- lxml
- re
- XPath Selector
- CSS Selector

CSS Selector的基本使用

CSS Selector由W3C组织维护并规范

“股票数据Scrapy爬虫”实例介绍

功能描述

- 技术路线：scrapy
- 目标：获取上交所和深交所所有股票的名称和交易信息

- 输出：保存到文件中

数据网站的确定

获取股票列表：

东方财富网：http://quote.eastmoney.com/stocklist.html

获取个股信息：

百度股票：https://gupiao.baidu.com/stock/

单个股票：https://gupiao.baidu.com/stock/sz002439.html

“股票数据Scrapy爬虫”实例编写

步骤

步骤1：建立工程和Spider模板

- > scrapy startproject BaiduStocks
- > BaiduStocks
- > scrapy genspider stocks baidu.com
- > 进一步修改spiders/stocks.py

步骤2：编写Spider

- 配置stocks.py文件
- 修改对返回页面的处理
- 修改对新增URL爬取请求的处理（stocks.py）

```
# -*- coding: utf-8 -*-
import scrapy
import re

class StocksSpider(scrapy.Spider):
    name = 'stocks'
    start_urls = ['http://quote.eastmoney.com/stocklist.html']

    def parse(self, response):
        for href in response.css('a::attr(href)').extract():
            try:
                stock = re.findall(r"[s][hz]\d{6}", href)[0]
                url = 'https://gupiao.baidu.com/stock/' + stock + '.html'
                yield scrapy.Request(url, callback=self.parse_stock)
```

```
            except:
                continue

    def parse_stock(self, response):
        infoDict = {}
        stockInfo = response.css('.stock-bets')
        name = stockInfo.css('.bets-name').extract()[0]
        keyList = stockInfo.css('dt').extract()
        valueList = stockInfo.css('dd').extract()
        for i in range(len(keyList)):
            key = re.findall(r'>.*</dt>', keyList[i])[0][1:-5]
            try:
                val = re.findall(r'\d+\.?.*</dd>',valueList[i])[0][0:-5]
            except:
                val = '--'
            infoDict[key]=val

        infoDict.update(                          # 两个字典合并到一起
            {'股票名称': re.findall('\s.*\(', name)[0].split()[0] + \
             re.findall('\>.*\<', name)[0][1:-1]})
        yield infoDict
```

步骤3：编写ITEM Pipelines

- 配置pipelines.py文件
- 定义对爬取项（Scrapy Item）的处理类（pipelines.py）

```
# -*- coding: utf-8 -*-
# Define your item pipelines here
#
# Don't forget to add your pipeline to the ITEM_PIPELINES setting
# See: https://docs.scrapy.org/en/latest/topics/item-pipeline.html

class BaidustocksPipeline(object):
    def process_item(self, item, spider):
        return item

class BaidustocksInfoPipeline(object):
    def open_spider(self, spider):                        # 打开爬虫时进行的操作
        self.f = open('BaiduStockInfo.txt', 'w')
```

```
14
15      def close_spider(self, spider):
16          self.f.close()
17
18      def process_item(self, item, spider):
19          try:
20              line = str(dict(item)) + '\n'
21              self.f.write(line)
22          except:
23              pass
24          return item
```

- 配置ITEM_PIPELINES选项（settings.py）

```
1 # Configure item pipelines
2 # See https://docs.scrapy.org/en/latest/topics/item-pipeline.html
3 ITEM_PIPELINES = {
4     'BaiduStocks.pipelines.BaidustocksInfoPipeline': 300,
5 }
```

执行程序：scrapy crawl stocks

”股票数据Scrapy爬虫“实例优化

配置并发连接选项（settings.py）

选项	说明
CONCURRENT_REQUESTS	Downloader最大并发请求下载数量，默认32
CONCURRENT_ITEMS	Item Pipeline最大并发ITEM处理数量，默认100
CONCURRENT_REQUESTS_PER_DOMAIN	每个目标域名最大的并发请求数量，默认8
CONCURRENT_REQUESTS_PER_IP	每个目标IP最大的并发请求数量，默认0，非0有效

网络爬虫课程的未完待续

Scrapy爬虫的地位

- Python语言最好的爬虫框架
- 具备企业级专业爬虫的扩展性（7X24高可靠性）

- 千万级URL爬取管理与部署

Scrapy足以支撑一般商业服务所需的爬虫能力

Scrapy爬虫的应用展望

- 普通价值
    - 基于Linux，7X24，稳定爬取输出
    - 商业级部署和应用（scrapy-*）
    - 千万级规模内URL爬取、内容分析和存储
- 高阶价值
    - 基于docker，虚拟化部署
    - 中间键扩展，增加调度和监控
    - 各种反爬取对抗技术