- 简介
  - 爬虫就是用 `http` 请求网页,然后网页返回页面信息的一种技术.
  - `HTTP` :每一次请求建立一次连接,是一种 `TCP/IP` 协议形式
  - 要掌握好爬虫技术需要会一点web前端知识,会python的语法,正则表达式和xpath基础. `这些我们之后都会讲到.`
- `example1`
  - python用来进行爬虫的最基本的库 `urllib` 和 `urllib2`
  - `example1.py`

```
# -*- coding:utf-8 -*-
import urllib2
req = urllib2.Request('http://neu.edu.cn')
response = urllib2.urlopen(req)
html = response.read()
print html
```

  - 第一步先发送一个request请求到 `http://neu.edu.cn`
  - 第二步接收从官网的反馈
  - 第三步为读取 `response` 里的内容
  - 最后打印得到html页面元素信息
- `example2` :请求
  - `get` 和 `post` 请求的区别:
    - `get` :表示所有的请求信息都是可见的,例如 `http://xxx?name=aaa&password=bbb` ,会以 `?xxx&xxx&xxx` 的形式显示在 `url` 上
    - `post` :表示相关的请求信息是不可见的,例如上面的 `?name=aaa&password=bbb` 就会不见了
  - `example2`
    - `get` :

```
# -*- coding:utf-8 -*-
import urllib
import urllib2
#下面是校园网控制网关
#https://ipgw.neu.edu.cn/srun_portal_pc.php?url=&ac_id=1
data={}
data['url']=''
data['ac_id'] = 1
url_values = urllib.urlencode(data)
url = 'https://ipgw.neu.edu.cn/srun_portal_pc.php'
url = url+'?'+url_values
req = urllib2.Request(url)
response = urllib2.urlopen(req)
html = response.read()
print html
```

- `example3` :Cookie的使用
  - Cookie:网站存储在用户本地的相关信息,例如你登陆一次之后下一次访问页面就不用再重复登陆了
  - Opener:我们可以使用urlopen,但只能传入3个参数(url,data,timeout),当要传入Cookie时,我们需要用到urllib2中更一般的opener
  - Cookielib:CookieJar->FileCookieJar->MozilaCookieJar | LWPCookieJar (用于存储Cookie的类)
  - 获取Cookie保存到变量

    ```
    # -*- coding:utf-8 -*-
    import urllib2
    import cookielib

    cookie = cookielib.CookieJar()
    #创建cookie处理器
    handler = urllib2.HTTPCookieProcessor(cookie)
    #构建opener
    opener = urllib2.build_opener(handler)
    response = opener.open('http://www.baidu.com')
    for item in cookie:
      print 'Name = ',item.name
      print 'Value = ',item.value
    ```

  - 将cookie保存到本地文件

```
# -*- coding:utf-8 -*-
import urllib2
import cookielib

filename = 'cookie.txt'
cookie = cookielib.MozillaCookieJar(filename)
handler = urllib2.HTTPCookieProcessor(cookie)
opener = urllib2.build_opener(handler)
response = opener.open('http://www.baidu.com')
cookie.save(ignore_discard=True,ignore_expires=True)
```

- 从文件中获取Cookie并访问

```
# -*- coding:utf-8 -*-
import urllib2
import cookielib

cookie = cookielib.MozillaCookieJar()
cookie.load('cookie.txt',ignore_expires=True,ignore_discard=True)
req = urllib2.Request('http://www.baidu.com')
opener = urllib2.build_opener(urllib2.HTTPCookieProcessor(cookie))
response = opener.open(req)
print response.read()
```

- `example4` :正则表达式简介
  - 正则表达式表

语法	说明	表达式实例	完整匹配的字符串
	字符		
一般字符	匹配自身	abc	abc
.	匹配任意除换行符"\n"外的字符。 在DOTALL模式中也能匹配换行符。	a.c	abc
\	转义字符，使后一个字符改变原来的意思。 如果字符串中有字符*需要匹配，可以使用*或者字符集[*]。	a\.c a\\c	a.c a\c
[...]	字符集（字符类）。对应的位置可以是字符集中任意字符。 字符集中的字符可以逐个列出，也可以给出范围，如[abc]或[a-c]。第一个字符如果是^则表示取反，如[^abc]表示不是abc的其他字符。 所有的特殊字符在字符集中都失去其原有的特殊含义。在字符集中如果要使用]、-或^，可以在前面加上反斜杠，或把]、-放在第一个字符，把^放在非第一个字符。	a[bcd]e	abe ace ade
	预定义字符集（可以写在字符集[...]中）		
\d	数字：[0-9]	a\dc	a1c
\D	非数字：[^\d]	a\Dc	abc
\s	空白字符：[<空格>\t\r\n\f\v]	a\sc	a c
\S	非空白字符：[^\s]	a\Sc	abc
\w	单词字符：[A-Za-z0-9_]	a\wc	abc
\W	非单词字符：[^\w]	a\Wc	a c

数量词（用在字符或(...)之后）			
*	匹配前一个字符0或无限次。	abc*	ab abccc
+	匹配前一个字符1次或无限次。	abc+	abc abccc
?	匹配前一个字符0次或1次。	abc?	ab abc
{m}	匹配前一个字符m次。	ab{2}c	abbc
{m,n}	匹配前一个字符m至n次。 m和n可以省略：若省略m，则匹配0至n次；若省略n，则匹配m至无限次。	ab{1,2}c	abc abbc
*? +? ?? {m,n}?	使 * + ? {m,n}变成非贪婪模式。	示例将在下文中介绍。	
边界匹配（不消耗待匹配字符串中的字符）			
^	匹配字符串开头。 在多行模式中匹配每一行的开头。	^abc	abc
$	匹配字符串末尾。 在多行模式中匹配每一行的末尾。	abc$	abc
\A	仅匹配字符串开头。	\Aabc	abc
\Z	仅匹配字符串末尾。	abc\Z	abc
\b	匹配\w和\W之间。	a\b!bc	a!bc
\B	[^\b]	a\Bbc	abc
逻辑、分组			
|	|代表左右表达式任意匹配一个。 它总是先尝试匹配左边的表达式，一旦成功匹配则跳过匹配右边的表达式。 如果|没有被包括在()中，则它的范围是整个正则表达式。	abc|def	abc def
(...)	被括起来的表达式将作为分组，从表达式左边开始每遇到一个分组的左括号'('，编号+1。 另外，分组表达式作为一个整体，可以后接数量词。表达式中的|仅在该组中有效。	(abc){2} a(123|456)c	abcabc a456c
(?P<name>...)	分组，除了原有的编号外再指定一个额外的别名。	(?P<id>abc){2}	abcabc
\<number>	引用编号为<number>的分组匹配到的字符串。	(\d)abc\1	1abc1 5abc5
(?P=name)	引用别名为<name>的分组匹配到的字符串。	(?P<id>\d)abc(?P=id)	1abc1 5abc5
特殊构造（不作为分组）			
(?:...)	(...)的不分组版本，用于使用'|'或后接数量词。	(?:abc){2}	abcabc
(?iLmsux)	iLmsux的每个字符代表一个匹配模式，只能用在正则表达式的开头，可选多个。匹配模式将在下文中介绍。	(?i)abc	AbC
(?#...)	#后的内容将作为注释被忽略。	abc(?#comment)123	abc123
(?=...)	之后的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	a(?=\d)	后面是数字的a
(?!...)	之后的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	a(?!\d)	后面不是数字的a
(?<=...)	之前的字符串内容需要匹配表达式才能成功匹配。 不消耗字符串内容。	(?<=\d)a	前面是数字的a
(?<!...)	之前的字符串内容需要不匹配表达式才能成功匹配。 不消耗字符串内容。	(?<!\d)a	前面不是数字的a
(?(id/name) yes-pattern |no-pattern)	如果编号为id/别名为name的组匹配到字符，则需要匹配yes-pattern，否则需要匹配no-pattern。 |no-patern可以省略。	(\d)abc(?(1)\d|abc)	1abc2 abcabc

- 接下来介绍Python的Re模块对正则表达式的支持
  - re.compile(string[,flag])返回pattern对象
  - re.match(pattern,string[,flags])
  - re.search(pattern,string[,flags])
  - re.split(pattern,string[,maxsplit])
  - re.findall(pattern,string[,flags])
  - re.finditer(pattern,string[,flags])
  - re.sub(pattern,repl,string[,count])
  - re.subn(pattern,repl,string[,count])

- 解释:

  - `pattern = re.compile(r'hello')`
  - flags表示匹配模式,使用|来表示同时生效,例如 `re.I|re.M`

    - re.I(GNORECASE):忽略大小写
    - re.M(ULTILINE):多行模式,改变'^'和'$'的行为
    - re.S(DOTALL):点任意匹配模式,改变'.'的行为
    - re.L(OCALE):使预定字符类\w \W \b \B \s \S取决于当前区域的设定
    - re.U(NICODE):使预定字符类\w \W \b \B \s \S \d \D取决于unicode定义的字符属性
    - re.X(VERBOSE):详细模式.这个模式下正则表达式可以是多行,忽略空白字符,并可以加入注释

  - re.match(pattern,string[,flags]):这个方法将会从字符串的开头开始匹配,当遇到无法匹配或者到达string尾部时还没有匹配结束,都会返回None,否则匹配pattern成功并且终止.
    - 例子详见example4_1.py

  - re.search(pattern,string[,flags]):这个方法是搜索整个字符串进行匹配
    - 例子详见example4_2.py

  - re.split(pattern,string[,maxsplit]):将字符串进行分割,其中maxsplit为最大分割的次数
    - 例子详见example4_3.py

  - re.findall(pattern,string[,flags]):以列表形式返回全部能匹配的子串
    - 例子详见example4_4.py

  - re.finditer(pattern,string[,flags]):搜索所有匹配的并且以迭代器的形式返回
    - 例子详见example4_5.py

  - re.sub(pattern,repl,string[,count]):使用pattern来匹配字符串,用repl声明的规则来改变字符串
    - 例子详见example4_6.py

  - re.subn(pattern,repl,string[,count]):执行re.sub(pattern,repl,string[,count])并加上替换次数
    - 例子详见example4_7.py

- example5:实战爬取静态网页 `http://www.qiushibaike.com/hot/page/1`
    - 用到的正则表达式:
        - ()表示里面是我们要匹配的东西,用group()来取出
        - .*?表示最小的匹配,即非贪婪的匹配
        - re.S 标志代表在匹配时为点任意匹配模式，点 . 也可以代表换行符