1. **常用的网络数据的爬取方法？**

python爬虫的几种数据提取方式：正则 ，bs4，pyquery，xpath，cssselector。

1. **你遇到过得反爬虫策略有哪些？对应的解决方案有哪些？**

通过headers反爬虫：解决策略，伪造headers  
 基于用户行为反爬虫：动态变化去爬取数据，模拟普通用户的行为  
 基于动态页面的反爬虫：跟踪服务器发送的ajax请求，模拟ajax请求

1. **Urllib和urllib2的区别在哪？**

Python的urllib和urllib2模块都做与请求URL相关的操作，但他们提供不同的功能。他们两个最显着的差异如下：

总结1：urllib2可以接受一个Request对象，并以此可以来设置一个URL的headers，但是urllib只接收一个URL。这意味着，你不能伪装你的用户代理字符串等。

总结2：urllib模块可以提供进行urlencode的方法，该方法用于GET查询字符串的生成，urllib2的不具有这样的功能。这就是urllib与urllib2经常在一起使用的原因。

1. **设计一个基于session登录验证的爬虫方案**

from YDMHTTPDemo import parse\_code\_img  
from lxml import etree  
import requests  
  
# 创建一个session对象:会自动保存cookie  
session = requests.session()  
# 获取验证码的文本数据  
url = **'http://www.renren.com/'**login\_url = **'http://www.renren.com/ajaxLogin/login?1=1&uniqueTimestamp=2018113926761'**headers = {  
 **'User-Agent'**: **'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36'**}  
data = {  
 **"email"**: **"13120303891"**,  
 **"icode"**: **""**,  
 **"origURL"**: **"http://www.renren.com/home"**,  
 **"domain"**: **"renren.com"**,  
 **"key\_id"**: **"1"**,  
 **"captcha\_type"**: **"web\_login"**,  
 **"password"**: **"0fa20889fa1bf409dda8d0d684a8b168aa1850283dd96ebd37462b843610d1cb"**,  
 **"rkey"**: **"fa647ac3744264c8eece684183c36fe5"**,  
 **"f"**: **"http%3A%2F%2Fzhibo.renren.com%2Ftop"**}  
page\_text = requests.get(url=url, headers=headers).text  
# 解析验证码图片，保存本地  
tree = etree.HTML(page\_text)  
code\_img\_url = tree.xpath(**'//img[@id="verifyPic\_login"]/@src'**)[0]  
  
  
def personal\_info(url, headers, data):  
 # 使用session发起请求：将cookie存储到session。保证请求成功，将cookie存储到session中即可  
 session.post(url=url, headers=headers, data=data)  
 # 进行个人主页的页面数据的获取  
 personal\_url = **'http://www.renren.com/969091665/profile'** page\_text = session.get(url=personal\_url, headers=headers).text  
 with open(**'./renren.html'**, **'w'**, encoding=**'utf-8'**) as f:  
 f.write(page\_text)  
  
  
# 部分网站多次登录失败后会出现验证码情况  
if code\_img\_url:  
 # 识别验证码文本信息  
 code\_img\_data = requests.get(url=code\_img\_url, headers=headers).content  
 code\_path = **'./code.jpg'** with open(code\_path, **'wb'**) as f:  
 f.write(code\_img\_data)  
 code\_text = parse\_code\_img(filename=code\_path, username=**'bobo328410948'**, password=**'bobo328410948'**, appid=6003,  
 appkey=**'1f4b564483ae5c907a1d34f8e2f2776c'**, codetype=2004)  
 print(code\_text)  
 # 登录操作:只为了获取cookie  
 data[**"icode"**] = code\_text  
 personal\_info(url=login\_url, headers=headers, data=data)  
# 没有验证码出现的情况  
else:  
 personal\_info(url=login\_url, headers=headers, data=data)

1. **说一说爬虫所要用到的数据包和解析包**

****

1. **动态网站如何抓取呢？**

在爬虫过程中，一般情况下都是直接解析html源码进行分析解析即可。但是，有一种情况是比较特殊的：网页的数据采用异步加载的，比如ajax加载的数据，在我们“查看网页源代码”是查看不到的。采用常规的爬虫这一块是解析不到的。

第一种解决方案是采用一些第三方的工具，模拟浏览器的行为，去加载数据。比如：Selenium、PhantomJs。

优点：不必考虑动态页面的各种变化多端（无论动态数据如何变化，最终呈现在页面上的效果是固定的，我们只关心最终结果。），我们只用关心最终的现实结果即可。可以统一处理。

缺点：性能低下，比如使用Selenium，每次我们都需要去启动一个浏览器进程；配置繁琐，不同的浏览器需要下载不同的驱动以及jar包，并且驱动和jar包之间有严格版本匹配关系，如果不匹配就不能使用（至少本人因为版本匹配的关系，花了很大的时间）。

第二种解决方案是分析页面，找到对应请求接口，直接获取数据。

优点：性能高，使用方便。我们直接获取原数据接口（换句话说就是直接拿取网页这一块动态数据的API接口），肯定会使用方便，并且改变的可能性也比较小。

缺点：缺点也是明显的，如何获取接口API?有些网站可能会考虑到数据的安全性，做各种限制、混淆等。这就需要看开发者个人的基本功了，进行各种分析了。

1. **如何实现AJAX请求的数据获取？**

答案如上

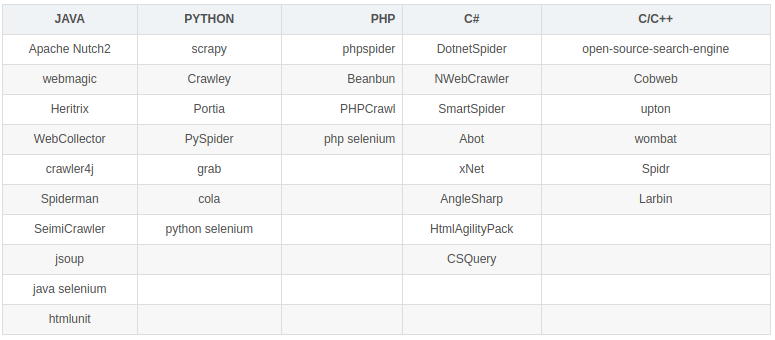
1. **爬虫携带cookie，但cookie过期怎么办？**

实例代码:

import os  
import pickle  
import time  
from selenium.webdriver.common.desired\_capabilities import DesiredCapabilities  
from selenium import webdriver  
def get\_cookie\_from\_network():  
 # 这个url比较关键，网络上比较老的办法是  
 # url\_login = 'https://login.taobao.com/member/login.jhtml'  
 # 测试之后发现，会报错“可能不是一个可交互的element”  
 # 在后面添加?style=mini后就可以了  
 url\_login = **'https://login.taobao.com/member/login.jhtml?style=mini'** # 这一段是为了给selenium添加user-agent。模拟浏览器  
 dcap = dict(DesiredCapabilities.PHANTOMJS)  
 dcap[**"phantomjs.page.settings.userAgent"**] = (  
 **"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0 "**)  
 driver = webdriver.PhantomJS(desired\_capabilities=dcap)  
 driver.implicitly\_wait(1)  
 driver.get(url\_login)  
 # driver.find\_element\_by\_id("J\_Static2Quick").click()  
 driver.find\_element\_by\_id(**"TPL\_username\_1"**).clear()  
 driver.find\_element\_by\_id(**"TPL\_password\_1"**).clear()  
 driver.find\_element\_by\_id(**'TPL\_username\_1'**).send\_keys(**u'张小呆920318'**)  
 driver.find\_element\_by\_id(**'TPL\_password\_1'**).send\_keys(**'plus820828'**)  
 driver.find\_element\_by\_id(**'J\_SubmitStatic'**).click()  
 # 获得 cookie信息  
 cookie\_list = driver.get\_cookies()  
 print  
 cookie\_list  
 cookie\_dict = {}  
 for cookie in cookie\_list:  
 # 写入文件  
 f = open(**'cookies/'** + cookie[**'name'**] + **'.taobao'**, **'w'**)  
 pickle.dump(cookie, f)  
 f.close()  
  
 if cookie.has\_key(**'name'**) and cookie.has\_key(**'value'**):  
 cookie\_dict[cookie[**'name'**]] = cookie[**'value'**]  
 return cookie\_dict

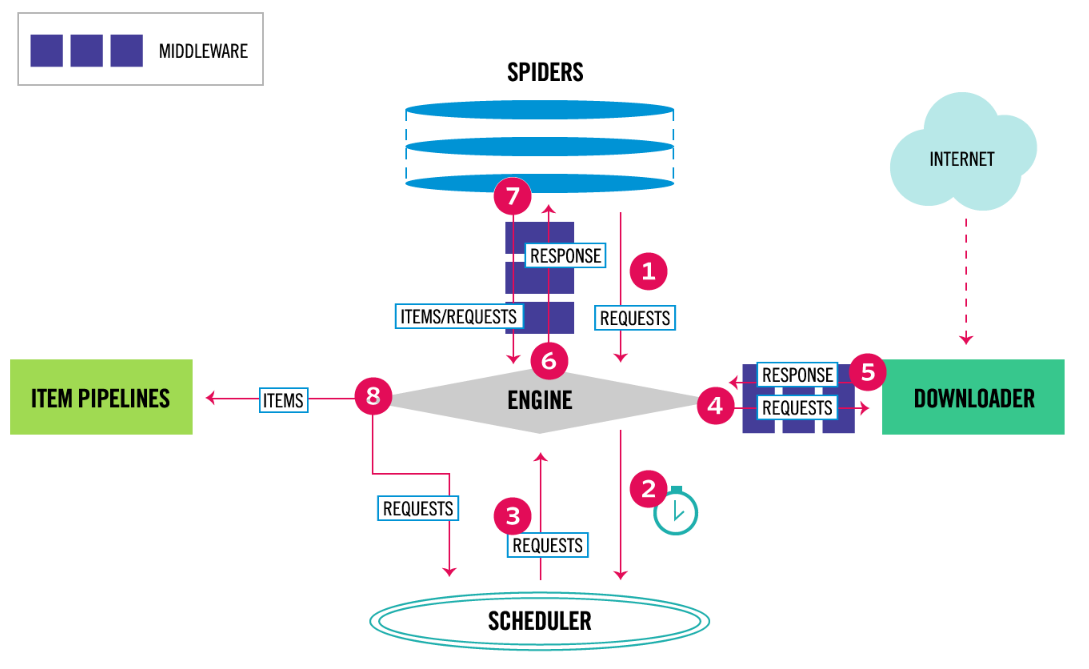
1. **你写爬虫时常用的框架是什么？为什么选择这个框架？**

各大主流编程语言爬虫框架介绍:

****

1. **Scrapy的基本结构都是什么？Scrapy中请求发送的整个流程是什么？**

Scrapy使用了Twisted作为框架，Twisted有些特殊的地方是它是事件驱动的，并且比较适合异步的代码。对于会阻塞线程的操作包含访问文件、数据库或者Web、产生新的进程并需要处理新进程的输出(如运行shell命令)、执行系统层次操作的代码(如等待系统队列),Twisted提供了允许执行上面的操作但不会阻塞代码执行的方法。



Scrapy数据流是由执行的核心引擎(engine)控制，流程是这样的：

1、爬虫引擎获得初始请求开始抓取。

2、爬虫引擎开始请求调度程序，并准备对下一次的请求进行抓取。

3、爬虫调度器返回下一个请求给爬虫引擎。

4、引擎请求发送到下载器，通过下载中间件下载网络数据。

5、一旦下载器完成页面下载，将下载结果返回给爬虫引擎。

6、引擎将下载器的响应通过中间件返回给爬虫进行处理。

7、爬虫处理响应，并通过中间件返回处理后的items，以及新的请求给引擎。

8、引擎发送处理后的items到项目管道，然后把处理结果返回给调度器，调度器计划处理下一个请求抓取。

9、重复该过程（继续步骤1），直到爬取完所有的url请求。

上图展示了scrapy的所有组件工作流程，下面单独介绍各个组件

爬虫引擎(ENGINE)

爬虫引擎负责控制各个组件之间的数据流，当某些操作触发事件后都是通过engine来处理。

调度器

调度接收来engine的请求并将请求放入队列中，并通过事件返回给engine。

下载器

通过engine请求下载网络数据并将结果响应给engine。

Spider

Spider发出请求，并处理engine返回给它下载器响应数据，以items和规则内的数据请求(urls)返回给engine。

管道项目(item pipeline)

负责处理engine返回spider解析后的数据，并且将数据持久化，例如将数据存入数据库或者文件。

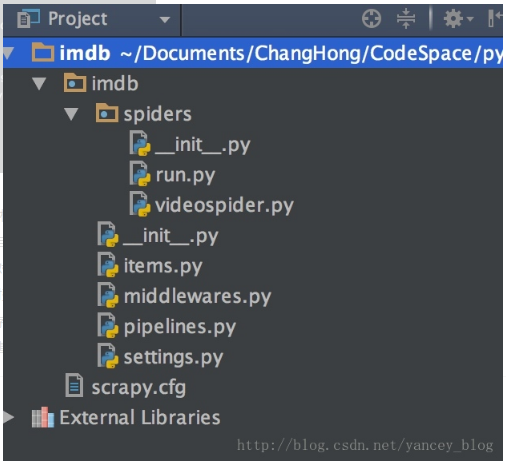
下载中间件

下载中间件是engine和下载器交互组件，以钩子(插件)的形式存在，可以代替接收请求、处理数据的下载以及将结果响应给engine。

spider中间件

spider中间件是engine和spider之间的交互组件，以钩子(插件)的形式存在，可以代替处理response以及返回给engine items及新的请求集。

**Scrapy项目结构**

****

items.py 负责数据模型的建立，类似于实体类。

middlewares.py 自己定义的中间件。

pipelines.py 负责对spider返回数据的处理。

settings.py 负责对整个爬虫的配置。

spiders目录 负责存放继承自scrapy的爬虫类。

scrapy.cfg scrapy基础配置

1. **Scrapy的去重原理简单说说**

对于每一个url的请求，调度器都会根据请求得相关信息加密（类似于MD5）得到一个指纹信息，并且将指纹信息和自己维护的一个集合中的url指纹信息进行比对，如果集合中已经存在这个指纹，就代表出现重复请求，就不再将这个Request放入队列中。如果集合中没有存在这个指纹，就将这个Request对象放入队列中，等待被调度。

 一般在request请求的参数中加入dont\_filter=False可以关闭去重，方便重复爬取同一网站下不同页面的信息。也可以通过修改爬取规则如start\_url等或自定义爬取url作为请求的参数以避免不必要的去重。

至于数据去重我一般有两种方法：自己利用python的集合类型在pipeline中加入去重的功能，是最简单无脑的方法；另外是在入库时进行去重，例如在Mysql中设置唯一索引或者在Redis中使用集合等等，这样的话对爬虫爬取效率的影响会比较小，但会增加数据库的负担。

1. **列举你使用过得scrapy中间件**

scrapy的中间件理论上有三种(Schduler Middleware,Spider Middleware,Downloader Middleware),在应用上一般有以下两种

1.爬虫中间件Spider Middleware，主要功能是在爬虫运行过程中进行一些处理.

2.下载器中间件Downloader Middleware，主要功能是在请求到网页后,页面被下载时进行一些处理.

1. **我们为什么要使用代理？代理具体如何使用？**

为什么会用到代理？

1. 安全避免同一个代理IP访问同一个网页，对于长时间访问同一个网页的IP，极大可能性IP会被封掉。
2. 方便解决IP代理问题技术含量高，找代理处理方便省事。
3. 成本低自己去维护服务器成本过高，不低于长久持续发展

使用代理访问网页有四个步骤

1.设置代理地址

proxy = {"http":"80.211.168.221:8080"}

2.创建ProxyHandler

proxy\_handler = request.ProxyHandler(proxy)

3.创建Opener

opener = request.build\_opener(proxy\_handler)

4.安装Opener

request.install\_opener(opener)

之后就是urlopen访问网页

1. **代理失效的话该怎么办呢？**

1.将代理IP及其协议载入ProxyHandler赋给一个opener\_support变量；

2.将opener\_support载入build\_opener方法，创建opener；

3.安装opener。

1. **对于爬取过程中登录验证码，该如何处理？**
2. scrapy自带(但效率不高)
3. 验证码自动识别（手写识别算法）
4. 利用开源工具
5. 第三方收费接口
6. **模拟登陆流程简单说说**

1.通过 get 请求登陆界面获取 token 和图片验证码；

2.token 直接作为参数，验证码手动输入，加上账号、密码、区号作为post 请求登录的参数；

3.登陆成功后的 response header 里有我们需要的 cookie，将其取出拼到新的 request header 里，请求要爬取数据的页面，成功获取数据！

1. **如何处理网站传参加密的情况？**

加密的三种情况：

加密+访问次数限制+每个页面相关信息的条目需要点详情进行二次请求；

复杂的加密算法进行参数+时间戳+sig值，后台进行 参数+时间限制；

定时同步cookie+每个界面一个cookie。

破解方法：

使用selenium模拟点击获取详情页面；

获取其相应的api接口，GET接口URL，获取它的json表格内容；

反向分析网页JS加载内容；

1. **分布式爬虫原理简单说说**

分布式爬虫主要由主机与从机，我们把自己的核心服务器（主机）称为 master，而把用于跑爬虫程序的机器（从机）称为 slave。

我们首先给爬虫一些start\_urls，spider 最先访问 start\_urls 里面的 url，再根据我们的 parse 函数，对里面的元素、或者是其他的二级、三级页面进行抓取。而要实现分布式，只需要在这个starts\_urls里面做文章就行了。进一步描述如下：

master 产生 starts\_urls，url 会被封装成 request 放到 redis 中的 spider:requests，总的 scheduler 会从这里分配 request，当这里的 request 分配完后，会继续分配 start\_urls 里的 url。

slave 从 master 的 redis 中取出待抓取的 request，下载完网页之后就把网页的内容发送回 master 的 redis，key 是 spider:items。scrapy 可以通过 settings 来让 spider 爬取结束之后不自动关闭，而是不断的去询问队列里有没有新的 url，如果有新的 url，那么继续获取 url 并进行爬取，所以这一过程将不断循环。

master 里的 reids 还有一个 key 是 “spider:dupefilter” 用来存储抓取过的 url 的 fingerprint（使用哈希函数将url运算后的结果），防止重复抓取，只要 redis 不清空，就可以进行断点续爬。

1. **爬取下来的数据你会选择什么方式进行存储？**
2. 写入txt文件
3. 写入数据库（关系型或非关系型）
4. 写入excel表格
5. **XPath解析题**

**<?xml version=”1.0”encoding=’ISO-8859-1”?>**

**<bookstore>**

**<book>**

**<title lang=”eng”>Harry Potter<title>**

**<price>29.99</price>**

**<author>赵六</author>**

**</book>**

**<book>**

**<title lang=”eng”>Learning XML</title>**

**<price>39.95</price>**

**<author>张三</author>**

**</book>**

**<book>**

**<title lang=”eng”>ORACLE”</title>**

**<price>40.32</price>**

**<author>Lary</author>**

**</book>**

**</bookstore>**

**写出表达式：**

**1.选取bookstore元素中的book元素的所有title元素，且其中的price元素的值须大于35.0**

**2.选取属于bookstore元素的后代的所有book元素，而不管它们位于bookstore之下的什么位置**

**3.选择属于bookstore元素的后代的所有book元素，而不管它们位于bookstore之下的什么位置**

**4.选取所有带有属性的title元素**