Added Note

- 中间件(下载器中间件 & 爬虫中间件)
- 下载器中间件

  > 更换代理IP，更换Cookies，更换User-Agent，自动重试

- 开发实现

  > 中间件本身是Python的一个类，只要爬虫每次访问网站之前都先“经过”这个类，它就能给请求换新的代理IP，这样就能实现动态改变代理。 在创建一个Scrapy工程以后，工程文件夹下会有一个middlewares.py文件 名字后面的s表示复数，说明这个文件里面可以放很多个中间件。 Scrapy自动创建的这个中间件是一个**爬虫中间件**

- Singleton Pattern 单例模式

  > 某一个类只有一个实例存在

- pytesseract

  > 功能是识别图片文件中文字，并作为返回参数返回识别结果； 默认支持tiff、bmp格式图片，只有在安装PIL(Python Imaging Library)之后，才能支持jpeg、gif、png等其他图片格式；

- selenium - 浏览器自动化测试框架

  > 测试脚本执行时，浏览器自动按照脚本代码做出点击，输入，打开，验证等操作，就像真实用户所做的一样，从终端用户的角度测试应用程序。

- Lambda函数

匿名函数，即没有具体名称的函数，它允许快速定义单行函数，类似于C语言的宏，可以用在任何需要函数的地方。这区别于def定义的函数。 用些函数如果只是临时一用，而且它的业务逻辑也很简单，没必要非给它去个名字 def创建的方法是有名称的，而lambda没有。 lambda会返回一个函数对象，但这个对象不会赋给一个标识符，而def则会把函数对象赋值给一个变量（函数名）。 lambda只是一个表达式，而def则是一个语句。 lambda表达式" : "后面，只能有一个表达式，def则可以有多个。 像if或for或print等语句不能用于lambda中，def可以。 定义简单的函数 不能共享给别的程序调用 语法格式： lambda [arg1 [,agr2, ... argn]]:expression 单个参数的： g = lambda x: x ** 2 print g(3) 多个参数的： g = lambda x,y,z :(x+y) **z print g(1,2,2)

如果定义匿名函数，还要给它绑定一个名字的话，有点画蛇添足，通常是直接使用 lambda 函数。那么 lamdba 函数的正确使用场景在哪呢？ 1 - 函数式编程 尽管 Python 算不上是一门纯函数式编程语言，但它本身提供了很多函数式编程的特性，像 map、reduce、filter、sorted 这些函数都支持函数作为参数，lambda 函数就可以应用在函数式编程中。

EXP.

list1 = [3,5,-4,-1,0,-2,-6] sorted(list1,key= lambda x:abs(x))

2 - 闭包 在这里我们可以简单粗暴地理解为闭包就是一个定义在函数内部的函数，闭包使得变量即使脱离了该函数的作用域范围也依然能被访问到。

EXP.

def my_add(n): return lambda x:x+n

add_3 = my_add(3) add_3(7)

在爬虫中的应用
https://blog.csdn.net/weixin_40982642/article/details/78583560 (https://blog.csdn.net/weixin_40982642/article/details/78583560)
https://zhuanlan.zhihu.com/p/27607704 (https://zhuanlan.zhihu.com/p/27607704)

- K-means聚类算法

1 - 确定聚类的个数k

2 - 选定k的D维向量作为初始类的中心

3 - 对每个样本计算与聚类中心的距离，选择最近的作为该样本所属的类

4 - 在同一类内部，重新计算聚类中心（几何重心）不断迭代，直到收敛。

- KNN近邻算法：

如果一个样本在特征空间中的k个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。处理验证码问题

自动识别验证码 重新请求（前提是使用了代理IP） * 重试中间件 有时使用代理会被远程拒绝或超时错误，这时需要换代理IP重试 重写 scrapy.downloadrmiddlewares.retry.RetryMiddleware * Scrapy介绍 包含爬取数据、提取结构性数据、应用框架

底层通过Twisted异步网络框架处理网络通讯

可扩展、高性能、多线程、分布式爬虫框架 * 常用命令 startproject 创建一个新项目

genspider 根据模板生成一个新爬虫

crawl 执行爬虫

shell 启动交互式抓取控制台 * scrapy的去重原理 原理：

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**得到一个**指纹信息**，并且将指纹信息和set()集合中的指纹信息进行比对，如果set()集合中已经存在这个数据，就不在将这个Request放入队列中。如果set()集合中没有存在这个加密后的数据，就将这个Request对象放入队列中，等待被调度。

如何操作？

dont_filter默认为False,即开启去重; * 用过什么中间件？

下载器中间件：

主要使用下载中间件处理请求，一般会对请求设置随机的User-Agent ，设置随机的代理。目的在于防止爬取网站的反爬虫策略。

引擎将请求传递给下载器过程中， 下载中间件可以对请求进行一系列处理。比如设置请求的 User-Agent，设置代理等

在下载器完成将Response传递给引擎中，下载中间件可以对响应进行一系列处理。比如进行gzip解压等。

爬虫中间件（没用过）：

用法与下载器中间件非常相似，只是它们的作用对象不同。下载器中间件的作用对象是请求request和返回response；爬虫中间件的作用对象是爬虫，更具体地来说，就是写在spiders文件夹下面的各个文件。

在中间件处理爬虫本身的异常

实际操作：

在setting.py中间设置好，随机的user-agent和proxy

创建middlewares.py来新建中间件 * 爬虫为什么用到代理？ 如果同一个IP频繁的抓取某个网站，很容易造成IP被封 代理IP质量好，速度快，就能提高爬取效率；代理IP干净，使用的人少，就不会被反爬虫策略发现，成功率就高。]

- 怎么使用代理？ 把代理的信息放在header里面 `i_headers = {'User-Agent': 'Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.48'}  req = urllib2.Request(new_url, headers=i_headers)`

px = request.ProxyHandler({‘http’:‘0.0.0.0’}) opener = request.build_opener(px) req = request.Request(‘网址’) res = opener.open(req) request.install_opener(opener) res = request.urlopen(req)

- 代理失效问题如何解决？ process_request 在请求之前会被调用，来设置代理

process_exception 在请求失败的时候被调用，用来判断代理失效

自己编写下载器中间件，scrapy每次连接失败都会重新执行激活的中间件

- 登陆验证码问题： 开源的OCR库 (Optical Character Recognition) 验证码截图人工识别 或者找第三方公司
- 爬取速度过快带来的验证码处理 换一个IP， 慢一点
- 模拟登陆 为什么有这个需求？ 很多网站都必须要登陆权限，爬虫如果想获取有用的信息就必须带有登陆后的cookie。

社交性质的网站只有用户登陆之后，才会出现有价值的内容

在手动登陆的时候有个小技巧，那就是故意把密码填错，这样可以很容易看到用户名和密码正确的提交路径和提交方式。

Python中的cookielib库

- 还没有自己写过分布式，但是基本的机理和流程知道。
- 构建分布式爬虫是时考虑的问题： 如何能保证多台机器同时抓取同一个URL？

如果某个节点挂掉，会不会影响其它节点，任务如何继续？

既然是分布式，如何保证架构的可伸缩性和可扩展性？不同优先级的抓取任务如何进行资源分配和调度？

celery 作为分布式调度工具

里面的一些概念

broker - 中间人，每当应用程序调用celery的异步任务的时候，会向broker传递消息，而后celery的worker将会取到消息，执行相应程序。

backend - 通常程序发送的消息，发完就完了，可能都不知道对方时候接受了。为此，celery实现了一个backend，用于存储这些消息以及celery执行的一些消息和结果。

worker - Celery类的实例，作用就是执行各种任务。

producer - 发送任务，将其传递给broker

beat - celery实现的定时任务。可以将其理解为一个producer，因为它也是通过网络调用定时将任务发送给worker执行。

celery只是任务队列，而不是真正意义上的消息队列，它自身不具有存储数据的功能，所以broker和backend需要通过第三方工具来存储信息

会把所有任务都通过消息队列发送给各个分布式节点进行执行，所以可以很好的保证url不会被重复抓取；

它在检测到worker挂掉的情况下，会尝试向其他的worker重新发送这个任务信息

- 解决编码问题 encode() decode()
- 爬下来的数据是如何处理的 python 自带的pymysql

理论上是非关系型数据库的效率更高 索引项的格式： 页面号码 页面大小 URL文本大小 时间文本大小 URL文本 时间文本

- Py基础问题
- py2 py3 的区别

  性能： py3 的性能差一点

  编码： py3 默认的编码方式是utf-8 ；py2 ASCII编码

  语法：py3 print() py2 print

  py3字符串只有str一种

  py3去除了long类型，只有一种整形

  input()

  py3 - str

  py2 - int

  还有很多其他的但是平常没有遇到过

- py2代码迁移到py3

2to3 列示需要修改的内容： 2to3 pythonfile.py 直接修改： 2to3 -w pythonfile.py 之后会出现以下文件，是原先代码的备份文件 pythonfile.py.bak

- 装饰器

装饰器本质上是一个Python函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象。它经常用于有切面需求的场景，比如：插入日志、性能测试、事务处理、缓存、权限校验等场景。装饰器是解决这类问题的绝佳设计，有了装饰器，我们就可以抽离出大量与函数功能本身无关的雷同代码并继续重用。概括的讲，装饰器的作用就是为已经存在的对象添加额外的功能。

- 迭代器

迭代器是一个可以记住遍历的位置的对象。 迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。 迭代器有两个基本的方法：iter() 和 next()。 字符串，列表或元组对象都可用于创建迭代器：

- 生成器

在 Python 中，使用了 yield 的函数被称为生成器（generator）。 跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。 在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。 调用一个生成器函数，返回的是一个迭代器对象。

- python的数据类型

字符串 str='this is a string'

布尔类型 bool = False

整数 int = 20

浮点数 float = 0.1

数字 包括整数浮点数

列表 列表 list = ['physics','chemistry',1997,2000]

元祖 Python的元组与列表类似，不同之处在于元组的元素不能修改；元组使用小括号()，列表使用方括号[]；元组创建很简单，只需要在括号中添加元素，并使用逗号(,)隔开即可 tup = {1,2,3,4,5} ; tup2 = "a","b","c","d"

字典 字典(dictionary)是除列表之外python中最灵活的内置数据结构类型。列表是有序的对象结合，字典是无序的对象集合。两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。 dict = {'abc':456}

日期

- 引号

  单引号 双引号的用法相同都是表示字符串

  利用三引号，表示多行的字符串，可以在三引号中自由的使用单引号和双引号

- 在function中设置一个全局的变量

  global i //在使用前初次声明 i=1 //给全局变量赋值

  def hanu(a) global i //再次声明，表示在这里使用的是全局变量，而不是局部变量

- 一个字符串是utf-8的字符，如何转换成gb18030的字符串

  string = string.decode('utf-8') string = string.encode('gb18030')

- 星号变量 * **

  单个星号代表这个位置接收任意多个非关键字参数，在函数的*b位置上将其转化成元组，

  而双星号代表这个位置接收任意多个关键字参数，在**b位置上将其转化成字典

- 怎么学习的

  BIT Mooc课程 + *Mining the Social Web* + "Web Scraping with Pyhton(O'Reilly"

- 平时用这个做什么？

  马拉松比赛 excel 体育比赛 比赛

- json类型的数据

  json.dumps() json.loads() json.dump() json.load()

- 数据库索引

  我们经常听到B+树就是这个概念，用这个树的目的和红黑树差不多，也是为了尽量保持树的平衡，当然红黑树是二叉树，但B+树就不是二叉树了，节点下面可以有多个子节点，数据库开发商会设置子节点数的一个最大值，这个值不会太小，所以B+树一般来说比较矮胖，而红黑树就比较瘦高了。 关于B+树的插入，删除，会涉及到一些算法以保持树的平衡，这里就不详述了。ORACLE的默认索引就是这种结构的。 如果经常需要同时对两个字段进行AND查询,那么使用两个单独索引不如建立一个复合索引，因为两个单独索引通常数据库只能使用其中一个，而使用复合索引因为索引本身就对应到两个字段上的，效率会有很大提高。

  第二种索引叫做散列索引，就是通过散列函数来定位的一种索引，不过很少有单独使用散列索引的，反而是散列文件组织用的比较多。 散列文件组织就是根据一个键通过散列计算把对应的记录都放到同一个槽中，这样的话相同的键值对应的记录就一定是放在同一个文件里了，也就减少了文件读取的次数，提高了效率。 散列索引呢就是根据对应键的散列码来找到最终的索引项的技术，其实和B树就差不多了，也就是一种索引之上的二级辅助索引，我理解散列索引都是二级或更高级的稀疏索引，否则桶就太多了，效率也不会很高。

  位图索引是一种针对多个字段的简单查询设计一种特殊的索引，适用范围比较小，只适用于字段值固定并且值的种类很少的情况，比如性别，只能有男和女，或者级别，状态等等，并且只有在同时对多个这样的字段查询时才能体现出位图的优势。 位图的基本思想就是对每一个条件都用0或者1来表示，如有5条记录，性别分别是男，女，男，男，女，那么如果使用位图索引就会建立两个位图，对应男的10110和对应女的01001,这样做有什么好处呢，就是如果同时对多个这种类型的字段进行and或or查询时，可以使用按位与和按位或来直接得到结果了。