**3.3Scrapy-redis架构**

**为什么使用Scrapy+redis架构:**

Scrapy，Python开发的一个快速,高层次的屏幕抓取和web抓取框架，用于抓取web站点并从页面中提取结构化的数据。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。

Scrapy框架已经可以完成很大的一部分爬虫工作了。但是如果遇到比较大规模的数据爬取，直接可以用上python的多线程/多进程，如果你拥有多台服务器，分布式爬取是最好的解决方式，也是最有效率的方法。

Scrapy-redis是基于redis的一个scrapy组件，scrapy-redis提供了维持待爬取url的去重以及储存requests的指纹验证。原理是：redis维持一个共同的url队列，各个不同机器上的爬虫程序获取到的url都保存在redis的url队列，各个爬虫都从redis的URL队列获取url，并把数据统一保存在同一个数据库里面。

**3.3.1Scrapy+Redis架构定义:**

Scrapy是一个比较好用的Python爬虫框架，你只需要编写几个组件就可以实现网页数据的爬取。但是当我们要爬取的页面非常多的时候，单个主机的处理能力就不能满足我们的需求了（无论是处理速度还是网络请求的并发数），这时候分布式爬虫的优势就显现出来。

而Scrapy-Redis则是一个基于Redis的Scrapy分布式组件。它利用Redis对用于爬取的请求(Requests)进行存储和调度(Schedule)，并对爬取产生的项目(items)存储以供后续处理使用。scrapy-redis重写了scrapy一些比较关键的代码，将scrapy变成一个可以在多个主机上同时运行的分布式爬虫。

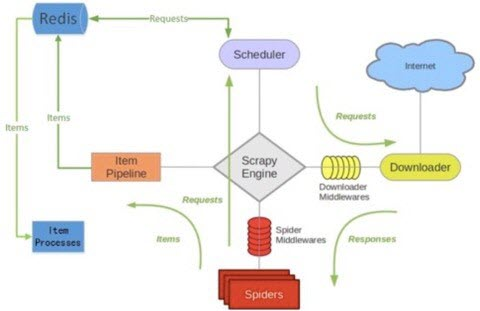
原生的Scrapy的架构是这样子的：



scrapy-redis的官方文档写的比较简洁，没有提及其运行原理，所以如果想全面的理解分布式爬虫的运行原理，还是得看scrapy-redis的源代码才行，不过scrapy-redis的源代码很少，也比较好懂，很快就能看完。

scrapy-redis工程的主体还是redis和scrapy两个库，工程本身实现的东西不是很多，这个工程就像胶水一样，把这两个插件粘结了起来。

加上了Scrapy-Redis之后的架构变成了：

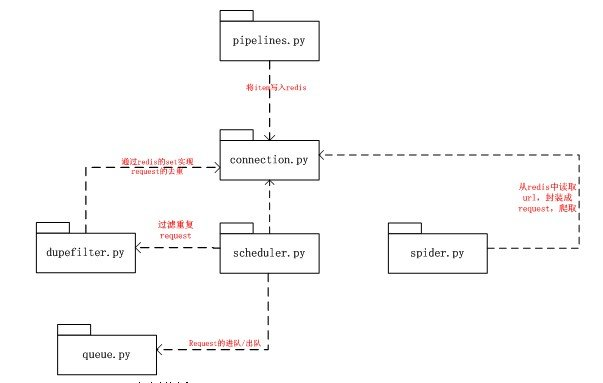


scrapy和scrapy-redis的主要区别在哪里？个人认为，scrapy和scrapy-redis不应该讨论区别。scrapy 是一个通用的爬虫框架，其功能比较完善，可以帮你迅速的写一个简单爬虫，并且跑起来。scrapy-redis是为了更方便地实现scrapy分布式爬取，而提供了一些以redis为基础的组件（注意，scrapy-redis只是一些组件，而不是一个完整的框架）。你可以这么认为，scrapy是一工厂，能够出产你要的spider。而scrapy-redis是其他厂商为了帮助scrapy工厂更好的实现某些功能而制造了一些设备，用于替换scrapy工厂的原设备。所以要想跑分布式，先让scrapy工厂搭建起来，再用scrapy-redis设备去更换scrapy的某些设备。那么这些scrapy-redis组件有什么突出特点呢？他们使用了redis数据库来替换scrapy原本使用的队列结构（deque），换了数据结构，那么相应的操作当然都要换啦，所以与队列相关的这些组件都做了更换

**Scrapy-Redis提供的组件:**

* connection.py 负责根据setting中配置实例化redis连接。被dupefilter和scheduler调用，总之涉及到redis存取的都要使用到这个模块。
* dupefilter.py 负责执行requst的去重，实现的很有技巧性，使用redis的set数据结构。但是注意scheduler并不使用其中用于在这个模块中实现的dupefilter键做request的调度，而是使用queue.py模块中实现的queue。当request不重复时，将其存入到queue中，调度时将其弹出。
* queue.py其作用如dupefilter.py所述，但是这里实现了三种方式的queue：FIFO的SpiderQueue，SpiderPriorityQueue，以及LIFI的SpiderStack。默认使用的是第二种
* picklecompat.py这里实现了loads和dumps两个函数，其实就是实现了一个serializer，因为redis数据库不能存储复杂对象（value部分只能是字符串，字符串列表，字符串集合和hash，key部分只能是字符串），所以我们存啥都要先串行化成文本才行。这里使用的就是python的pickle模块，一个兼容py2和py3的串行化工具。这个serializer主要用于一会的scheduler存reuqest对象，item pipeline的串行化默认用的就是json。
* pipelines.py这是是用来实现分布式处理(毕设这里是单机不做分布式)的作用。它将Item存储在redis中以实现分布式处理。另外可以发现，同样是编写pipelines，在这里的编码实现不同于文章中所分析的情况，由于在这里需要读取配置，所以就用到了from\_crawler()函数。
* cheduler.py 此扩展是对scrapy中自带的scheduler的替代（在settings的SCHEDULER变量中指出），正是利用此扩展实现crawler的分布式调度。其利用的数据结构来自于queue中实现的数据结构。
* spider.py 设计的这个spider从redis中读取要爬的url，然后执行爬取，若爬取过程中返回更多的url，那么继续进行直至所有的request完成。之后继续从redis中读取url，循环这个过程。

**3.3.2组件间关系:**



最后总结一下scrapy-redis的总体思路：这个工程通过重写scheduler和spider类，实现了调度、spider启动和redis的交互。实现新的dupefilter和queue类，达到了判重和调度容器和redis的交互，因为每个主机上的爬虫进程都访问同一个redis数据库，所以调度和判重都统一进行统一管理，达到了分布式爬虫的目的。

当spider被初始化时，同时会初始化一个对应的scheduler对象，这个调度器对象通过读取settings，配置好自己的调度容器queue和判重工具dupefilter。每当一个spider产出一个request的时候，scrapy内核会把这个reuqest递交给这个spider对应的scheduler对象进行调度，scheduler对象通过访问redis对request进行判重，如果不重复就把他添加进redis中的调度池。当调度条件满足时，scheduler对象就从redis的调度池中取出一个request发送给spider，让他爬取。当spider爬取的所有暂时可用url之后，scheduler发现这个spider对应的redis的调度池空了，于是触发信号spider\_idle，spider收到这个信号之后，直接连接redis读取strart url池，拿去新的一批url入口，然后再次重复上边的工作。