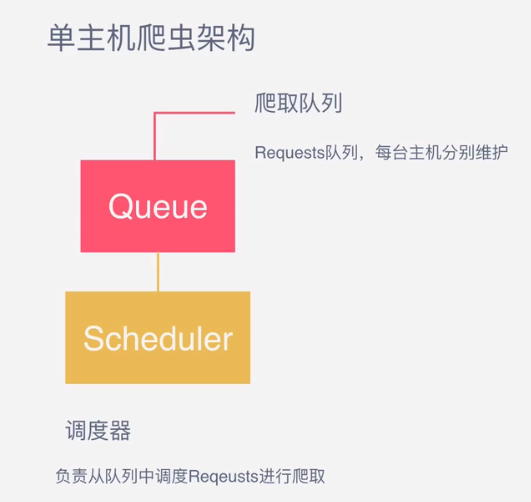
**分布式爬虫**

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。

**一、分布式爬虫的原理**

下面是单价版本的Scrapy框架：

Scrapy单机爬虫中只有一个本地爬取队列Queue，如果新的Request生成，就放到队列里面，随后Request被Scheduler调度。之后Request交给DownLoader执行爬取，简单的调度框架如下图所示：

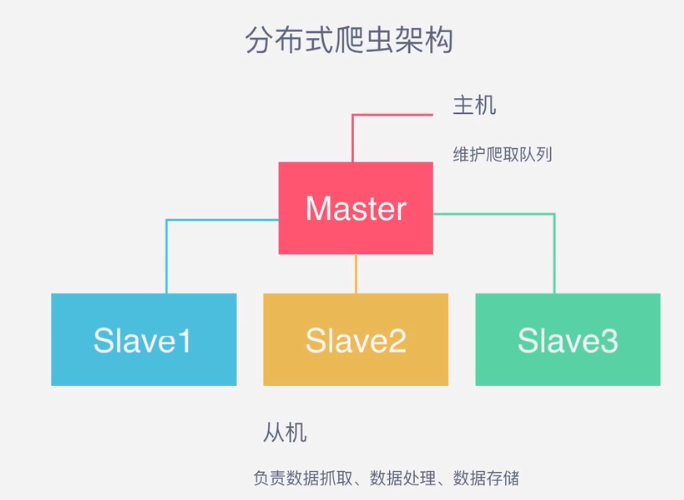


如果两个Scheduler同时从消息队里里面取Request，每个Scheduler都有对应的DownLoader，那么在带宽足够、正常爬取切不考虑队列压力的情况下，爬虫效率就会翻倍。

这样，Sheduler可以扩展为多个，DownLoader也是多个，而爬取队列维持为一个，也就是所谓的共享爬虫队列。这样才能保证Scheduler从队列里面调度某个Request之后，其他的Scheduler不会重复调用该Request，就可以保证多个Scheduler同步爬取。

Scheduler进行调度，而要实现多台服务器共同爬取数据关键就是共享爬取队列。

我们需要做的是在多台主机上同时运行爬虫任务协同爬取，而协同爬取的的前提就是共享爬虫队列。这样各台主机不需要各自维护排重队列，各台主机还是有各自的Sheduler和Downloader，所以调度和下载功能在各自的主机上完成。



**二、维护爬虫队列**

这里一般我们通过Redis为维护，Redis，非关系型数据库，Key-Value形式存储，结构灵活。

并且redis是内存中的数据结构存储系统，处理速度快，提供队列集合等多种存储结构，方便队列维护。

* 列表有lpush()、rpush()、lpop()、rpop()方法，可以实现先进先出式爬取队列，也可以实现先进后出堆栈式爬取队列；
* 集合的元素是无序，并且不重复的，这样就可以非常方便的实现随机排序且不重复的爬取队列；
* 有序集合带有分数表示，而Scrapy的Request也有优先级的控制，我们可以用它来实现优先级调度队列。

**三、如何去重**

这里借助redis的集合，redis提供集合数据结构，在redis集合中存储每个request的指纹，在向request队列中加入Request前先验证这个Request的指纹是否已经加入集合中。如果已经存在则不添加到request队列中，如果不存在，则将request加入到队列并将指纹加入集合。

**四、如何防止中断**

这里是做了启动判断，在每台slave的Scrapy启动的时候都会判断当前redis request队列是否为空。如果不为空，则从队列中获取下一个request执行爬取。如果为空则重新开始爬取，第一台从机执行爬取向队列中添加request。

**五、架构的实现**

接下来主要在需要在程序里面实现这个框架。首先实现一个共享队列，还要实现去重的功能。另外还有一个Sheduler的实现，确保可以从消息队列里面存取Request。

目前已经有人实现的框架scrapy-redis：

<https://github.com/rmax/scrapy-redis>

**六、Scrapy-Redis**

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-redi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。

参考资料：

<https://www.biaodianfu.com/scrapy-redis.html>

**七、如何配置**

**#启用Redis调度存储请求队列**

**SCHEDULER = "scrapy\_redis.scheduler.Scheduler"**

**#确保所有的爬虫通过Redis去重**

**DUPEFILTER\_CLASS = "scrapy\_redis.dupefilter.RFPDupeFilter"**

**#默认请求序列化使用的是pickle 但是我们可以更改为其他类似的。PS：这玩意儿2.X的可以用。3.X的不能用**

**#SCHEDULER\_SERIALIZER = "scrapy\_redis.picklecompat"**

**#不清除Redis队列、这样可以暂停/恢复 爬取**

**#SCHEDULER\_PERSIST = True**

**#使用优先级调度请求队列 （默认使用）**

**#SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.PriorityQueue'**

**#可选用的其它队列**

**#SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.FifoQueue'**

**#SCHEDULER\_QUEUE\_CLASS = 'scrapy\_redis.queue.LifoQueue'**

**#最大空闲时间防止分布式爬虫因为等待而关闭**

**#这只有当上面设置的队列类是SpiderQueue或SpiderStack时才有效**

**#并且当您的蜘蛛首次启动时，也可能会阻止同一时间启动（由于队列为空）**

**#SCHEDULER\_IDLE\_BEFORE\_CLOSE = 10**

**#将清除的项目在redis进行处理**

**ITEM\_PIPELINES = {**

**'scrapy\_redis.pipelines.RedisPipeline': 300**

**}**

**#序列化项目管道作为redis Key存储**

**#REDIS\_ITEMS\_KEY = '%(spider)s:items'**

**#默认使用ScrapyJSONEncoder进行项目序列化**

**#You can use any importable path to a callable object.**

**#REDIS\_ITEMS\_SERIALIZER = 'json.dumps'**

**#指定连接到redis时使用的端口和地址（可选）**

**#REDIS\_HOST = 'localhost'**

**#REDIS\_PORT = 6379**

**#指定用于连接redis的URL（可选）**

**#如果设置此项，则此项优先级高于设置的REDIS\_HOST 和 REDIS\_PORT**

**#REDIS\_URL = 'redis://user:pass@hostname:9001'**

**#自定义的redis参数（连接超时之类的）**

**#REDIS\_PARAMS = {}**

**#自定义redis客户端类**

**#REDIS\_PARAMS['redis\_cls'] = 'myproject.RedisClient'**

**#如果为True，则使用redis的'spop'进行操作。**

**#如果需要避免起始网址列表出现重复，这个选项非常有用。开启此选项urls必须通过sadd添加，否则会出现类型错误。**

**#REDIS\_START\_URLS\_AS\_SET = False**

**#RedisSpider和RedisCrawlSpider默认 start\_usls 键**

**#REDIS\_START\_URLS\_KEY = '%(name)s:start\_urls'**

**#设置redis使用utf-8之外的编码**

**#REDIS\_ENCODING = 'latin1'**

请各位小伙伴儿自行挑选需要的配置写到项目的settings.py文件中。