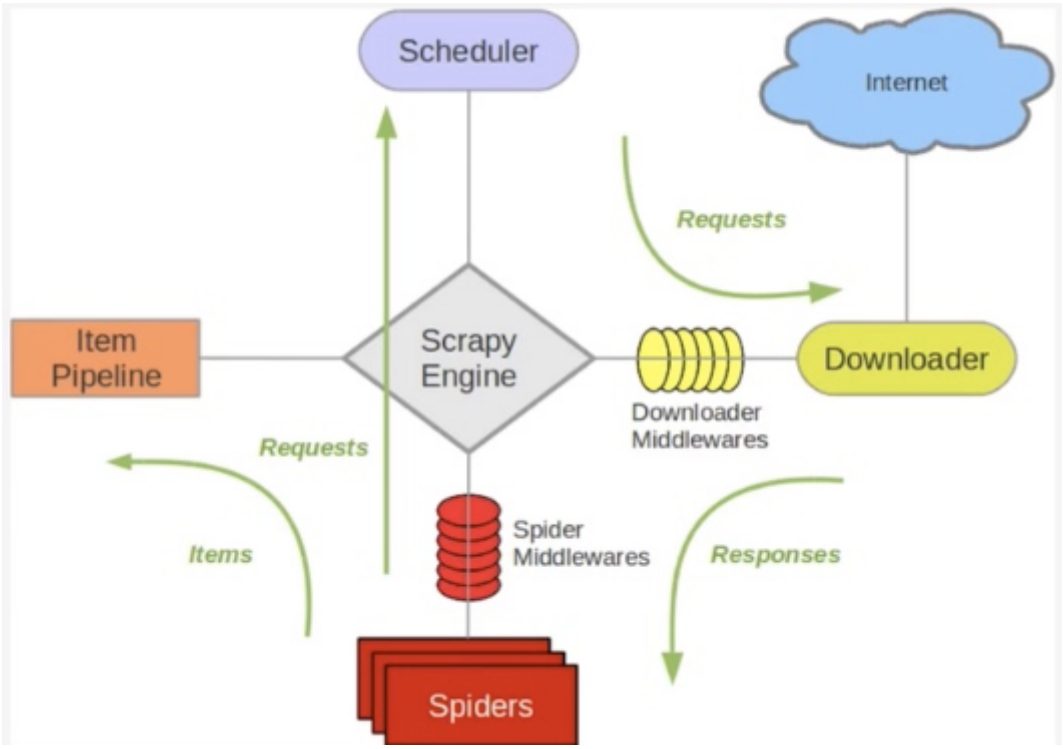
Scrapy爬虫框架分布式实现

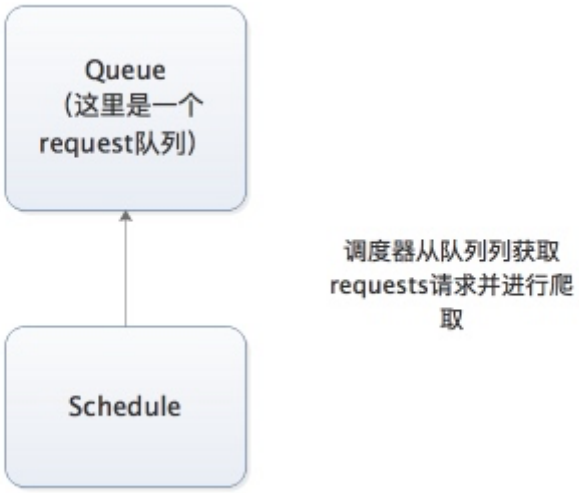
分布式爬虫原理

关于Scrapy工作流程回顾

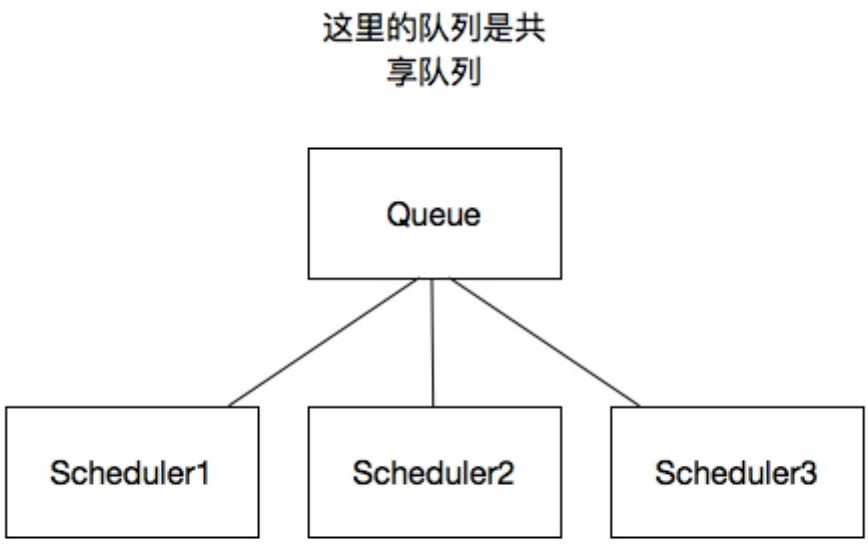
Scrapy单机架构



上图的架构其实就是一种单机架构，只在本机维护一个爬取队列，Scheduler进行调度，而要实现多态服务器共同爬取数据关键就是共享爬取队列。

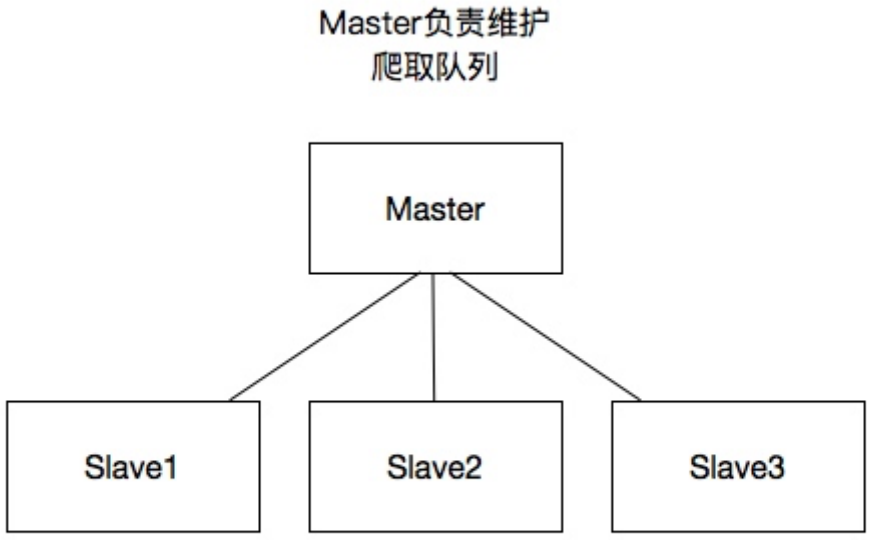


分布式架构



不同的主机的调度器从共享队列里获取

我将上图进行再次更改



各个Slave负责数据抓取，数据处理，数据存储

这里重要的就是我的队列通过什么维护?
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

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

如何防止中断？如果某个slave因为特殊原因宕机，如何解决?
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，第一台丛集执行爬取向队列中添加request

如何实现上述这种架构?
这里有一个scrapy-redis的库，为我们提供了上述的这些功能
scrapy-redis改写了Scrapy的调度器，队列等组件，利用他可以方便的实现Scrapy分布式架构
关于scrapy-redis的地址：https://github.com/rmax/scrapy-redis

搭建分布式爬虫

参考官网地址：https://scrapy-redis.readthedocs.io/en/stable/

前提是要安装scrapy_redis模块：pip install scrapy_redis
这里的爬虫代码是用的之前写过的爬取知乎用户信息的爬虫

修改该settings中的配置信息：

替换scrapy调度器

```
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
```

添加去重的class

```
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
```

添加pipeline
如果添加这行配置，每次爬取的数据也都会入到redis数据库中，所以一般这里不做这个配置

```
ITEM_PIPELINES = {
'scrapy_redis.pipelines.RedisPipeline': 300
}
```

共享的爬取队列，这里用需要redis的连接信息
这里的user:pass表示用户名和密码，如果没有则为空就可以

```
REDIS_URL = 'redis://user:pass@hostname:9001'
```

设置为为True则不会清空redis里的dupefilter和requests队列
这样设置后指纹和请求队列则会一直保存在redis数据库中，默认为False，一般不进行设置

```
SCHEDULER_PERSIST = True
```

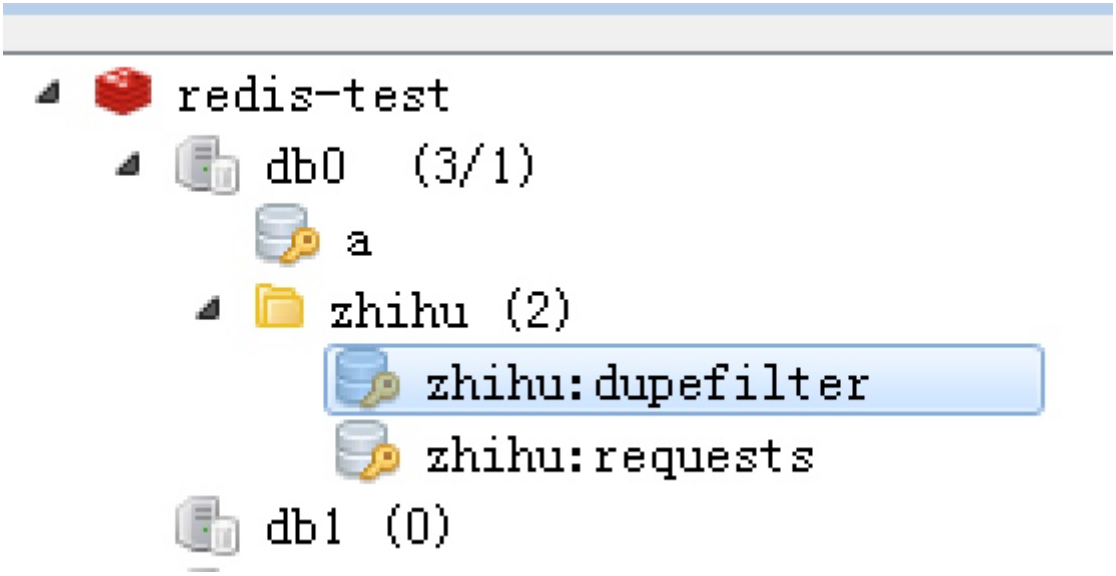
设置重启爬虫时是否清空爬取队列
这样每次重启爬虫都会清空指纹和请求队列,一般设置为False

```
SCHEDULER_FLUSH_ON_START=True
```

分布式

将上述更改后的代码拷贝的各个服务器，当然关于数据库这里可以在每个服务器上都安装数据，也可以共用一个数据，我这里方面是连接的同一个mongodb数据库，当然各个服务器上也不能忘记：
所有的服务器都要安装scrapy,scrapy_redis,pymongo

这样运行各个爬虫程序启动后，在redis数据库就可以看到如下内容，dupefilter是指纹队列，requests是请求队列



工欲善其事必先利其器！！！

之前写的爬虫，无论是单线程，多线程异步等都是在自己的电脑上运行。
好处是单个爬虫方便管理，调试；但当有了大量的URL需要爬取，用分布式爬虫无疑是最好的选择。
我的测试代码以实习僧网为目标网站，约2w个URL，单个scrapy与3个scrapy-redis分布式时间比约为5： 1
这篇文章会通过一个例子详细介绍scrapy-redis原理及其实现过程。

0.安装scrapy_redis

windows、ubuntu安装请参看：http://blog.fens.me/linux-redis-install/
centos7安装请参看：https://www.cnblogs.com/zjl6/p/6742673.html
注意：建议设置redis密码进行远程连接，或者添加安全组规则ip白名单，直接暴露端口容易被黑。

1.首先介绍一下：scrapy-redis框架

scrapy-redis：一个三方的基于redis的分布式爬虫框架，配合scrapy使用，让爬虫具有了分布式爬取的功能。
github地址： https://github.com/darkrho/scrapy-redis

一篇知乎介绍《scrapy-redis 和 scrapy 有什么区别？》

2.再介绍一下：分布式原理

scrapy-redis实现分布式，其实从原理上来说很简单，这里为描述方便，我们把自己的核心服务器称为master，而把用于跑爬虫程序的机器称为slave。

我们知 道，采用scrapy框架抓取网页，我们需要首先给定它一些start_urls，爬虫首先访问start_urls里面的url，再根据我们的具体逻辑，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，我们只需要在这个starts_urls里面做文章就行了。

我们在master上搭建一个redis数据库（注意这个数据库只用作url的存储，不关心爬取的具体数据，不要和后面的mongodb或者mysql混淆），并对每一个需要爬取的网站类型，都开辟一个单独的列表字段。通过设置slave上scrapy-redis获取url的地址为master地址。这样的结果就是，尽管有多个slave，然而大家获取url的地方只有一个，那就是服务器master上的redis数据库。

并且，由于scrapy-redis自身的队列机制，slave获取的链接不会相互冲突。这样各个slave在完成抓取任务之后，再把获取的结果汇总到服务器上（这时的数据存储不再在是redis，而是mongodb或者 mysql等存放具体内容的数据库了）

这种方法的还有好处就是程序移植性强，只要处理好路径问题，把slave上的程序移植到另一台机器上运行，基本上就是复制粘贴的事情。

3.分布式爬虫的实现：

1.使用两台机器，一台是win10，一台是ubuntu16.04，分别在两台机器上部署scrapy来进行分布式抓取一个网站

2.ubuntu16.04的ip地址为39.106.155.194，用来作为redis的master端，win10的机器作为slave

3.master的爬虫运行时会把提取到的url封装成request放到redis中的数据库：“dmoz:requests”，并且从该数据库中提取request后下载网页，再把网页的内容存放到redis的另一个数据库中“dmoz:items”

4.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

5.重复上面的3和4，直到master的redis中的“dmoz:requests”数据库为空，再把master的redis中的“dmoz:items”数据库写入到mongodb中

6.master里的reids还有一个数据“dmoz:dupefilter”是用来存储抓取过的url的指纹（使用哈希函数将url运算后的结果），是防止重复抓取的

（注：master与salve已经安装了MongoDB，Redis，scrapy，MySQL。）

4.完整实现过程

1、完成编码，多复制几份，把其中一份放到ubuntu作为master，其他几份留在windows作slave
2、启动master端scrapy，向master的redis中添加url，添加完成后master会继续运行爬虫，从redis取url进行抓取，数据存入master mongodb
3、启动多个slave爬虫，slave会远程向master redis中取url采集数据，采集数据会实时存入master mongodb中。

新建一个scrapy项目，完成常规的爬虫编码。

```
from redis import Redis
identity='slave'

from scrapy_redis.spiders import RedisSpider


class ShixiSpider(RedisSpider):
    if identity=='master':...
    name = 'slave_1'
    # 爬虫从连接的redis的该集合中读取url进行爬取，如果该集合为空，那么爬虫会一直等待..
    # 手动添加方法：lpush shixisheng:start_urls url
    redis_key = 'shixisheng:start_urls'
```

- 开始改动代码实现分布式爬虫，首先引入RedisSpider，把原来继承自scrapy.spider改为继承RedisSpider。
- 添加redis_key = 'shixisheng:start_urls' ;这里的redis_key实际上就是一个变量名，master爬虫爬到的所有URL都会保存到redis中这个名为“readcolorspider:start_urls”的列表下面，slave爬虫同时也会从这个列表中读取后续页面的URL。这个变量名可以任意修改。
- 修改设置settings.py

```
①Scheduler，首先是Scheduler的替换，这个东西是Scrapy中的调度员。在settings.py中添加以下代码：
SCHEDULER="scrapy_redis.scheduler.Scheduler"
②去重
DUPEFILTER_CLASS="scrapy_redis.dupefilter.RFPDupeFilter"
③不清理Redis队列
SCHEDULER_PERSIST=True
```

如果这一项为True，那么在Redis中的URL不会被Scrapy_redis清理掉，这样的好处是：爬虫停止了再重新启动，它会从上次暂停的地方开始继续爬取。但是它的弊端也很明显，如果有多个爬虫都要从这里读取URL，需要另外写一段代码来防止重复爬取。
如果设置成了False，那么Scrapy_redis每一次读取了URL以后，就会把这个URL给删除。这样的好处是：多个服务器的爬虫不会拿到同一个URL，也就不会重复爬取。但弊端是：爬虫暂停以后再重新启动，它会重新开始爬。
④设置redis地址

如果这一项为True，那么在Redis中的URL不会被Scrapy_redis清理掉，这样的好处是：爬虫停止了再重新启动，它会从上次暂停的地方开始继续爬取。但是它的弊端也很明显，如果有多个爬虫都要从这里读取URL，需要另外写一段代码来防止重复爬取。
如果设置成了False，那么Scrapy_redis每一次读取了URL以后，就会把这个URL给删除。这样的好处是：多个服务器的爬虫不会拿到同一个URL，也就不会重复爬取。但弊端是：爬虫暂停以后再重新启动，它会重新开始爬。
④设置redis地址

```
启用本地redis:   REDIS_URL = 'redis://127.0.0.1:6379'
启用远程redis:   REDIS_URL = 'redis://39.106.155.194:6379'
```

- 其他设置（可选）
  爬虫请求的调度算法
  爬虫的请求调度算法，有三种情况可供选择：
  ①队列

```
SCHEDULER_QUEUE_CLASS='scrapy_redis.queue.SpiderQueue'
```

如果不配置调度算法，**默认就会使用这种方式。**它实现了一个先入先出的队列，先放进Redis的请求会优先爬取。
②栈

如果不配置调度算法，**默认就会使用这种方式。**它实现了一个先入先出的队列，先放进Redis的请求会优先爬取。
②栈

```
SCHEDULER_QUEUE_CLASS='scrapy_redis.queue.SpiderStack'
```

这种方式，后放入到Redis的请求会优先爬取。
③优先级队列

这种方式，后放入到Redis的请求会优先爬取。
③优先级队列

```
SCHEDULER_QUEUE_CLASS='scrapy_redis.queue.SpiderPriorityQueue'
```

这种方式，会根据一个优先级算法来计算哪些请求先爬取，哪些请求后爬取。这个优先级算法比较复杂，会综合考虑请求的深度等各个因素。

这种方式，会根据一个优先级算法来计算哪些请求先爬取，哪些请求后爬取。这个优先级算法比较复杂，会综合考虑请求的深度等各个因素。

呼~~配置完这些，爬虫就可以正常工作了，slave从master取url采集数据，当master redis中"shixisheng:start_urls"和"slave_1:requests"都为空时，爬虫会暂停等待，直到redis中有新的url。若再无新url添加进来，就可以在此刻结束程序。

分布式爬虫状态与对应的redis中集合的变化

image.png

该聊聊数据存储的问题了

两种方法：

- 1.各存各的，master仅提供待爬取url，slave采集后数据存在slave本地，最后把数据汇总即可，这样一来数据就存在了master一部分，各slave一部分，如果slave比较多，数据的汇总也比较麻烦。
- 2.各个slave采集数据的同时把数据实时的发送到master，这样数据只存在于master，而slave就充当了真正意义上的“slave”————“干完就走，两袖清风”

显然第二种办法比较好，master即master，slave即slave。
该项目数据库选用了mongodb。（也实现了存Mysql，下文会讲）
针对不同类型数据可以根据具体需求来选择不同的数据库存储。结构化数据可以使用mysql节省空间，非结构化、文本等数据可以采用mongodb等非关系型数据提高访问速度。具体选择可以自行百度谷歌，有很多关于sql和nosql的对比文章。

写入master mongodb代码（piplines.py）

```python
# 存到master的MongoDB
class MongoDBPipeline(object):
    def __init__(self):
        if identity=='master':
            self.client = MongoClient('127.0.0.1', 27017)
        else:
            self.client = MongoClient('39.106.155.194', 27017)
        self.shixisheng = self.client['test']['shixisheng']

    def process_item(self, item, spider):
        self.insert_to_mongodb(item)
        return item   # 返回给终端

    def insert_to_mongodb(self, item):
        self.shixisheng.insert_one({"name":item['name'], "salary":item['salary'], "location":item['location'], "xueli":item['x
                    "work":item['work'], "time":item['time'], "category":item['category'], "required":item['required']})
```

save to master mongodb

写入slave mysql代码（piplines.py）

```python
# 存到slave的Mysql
class MySQLStorePipeline(object):
    def __init__(self):...
    def process_item(self, item, spider):
        query = self.dbpool.runInteraction(self._conditional_insert, item)
        return item

    def _conditional_insert(self, tx, item):
        try:
            tx.execute('insert into shiziseng values(%s,%s,%s,%s,%s,%s,%s,%s)',(item['name'],item['salary'],item['location'],
                                                                                item['work'],item['time'],item['category'],ite
            print u"写入成功"
        except Exception,e:
            print e
            print u"写入失败"
```

save to slave mysql

上图是save to slave mysql ,如需save to master请在MySQLStorePipeline更改数据库连接配置。

settings.py文件

```python
# Obey robots.txt rules
ROBOTSTXT_OBEY = False
ITEM_PIPELINES = {
    # 'shixiseng.pipelines.MySQLStorePipeline': 300,    # 存到slave的Mysql
    'shixiseng.pipelines.MongoDBPipeline': 300,         # 存到master的MongoDB
}


SCHEDULER="scrapy_redis.scheduler.Scheduler"
DUPEFILTER_CLASS="scrapy_redis.dupefilter.RFPDupeFilter"
SCHEDULER_QUEUE_CLASS='scrapy_redis.queue.SpiderQueue'
# REDIS_URL = 'redis://127.0.0.1:6379'
REDIS_URL = 'redis://39.106.155.194:6379'
SCHEDULER_PERSIST=True
```

image.png

master mongodb部分截图：

master mongodb

本项目已上传到github，欢迎友情加star: https://github.com/TimeAshore/scrapy_redis_sxs/

本人也比较小白 学术尚浅，如有什么问题，欢迎提出来共同进步。

Scrapy分布式实现

1. 安装Scrapy-Redis。
2. 配置Redis服务器。
3. 修改配置文件。
    - SCHEDULER = 'scrapy_redis.scheduler.Scheduler'
    - DUPEFILTER_CLASS = 'scrapy_redis.dupefilter.RFPDupeFilter'
    - REDIS_HOST = '1.2.3.4'
    - REDIS_PORT = 6379
    - REDIS_PASSWORD = '1qaz2wsx'
    - SCHEDULER_QUEUE_CLASS = 'scrapy_redis.queue.FifoQueue'
    - SCHEDULER_PERSIST = True（通过持久化支持接续爬取）
    - SCHEDULER_FLUSH_ON_START = True（每次启动时重新爬取）

Scrapyd分布式部署

1. 安装Scrapyd
2. 修改配置文件
    - mkdir /etc/scrapyd
    - vim /etc/scrapyd/scrapyd.conf
3. 安装Scrapyd-Client
    - 将项目打包成Egg文件。
    - 将打包的Egg文件通过addversion.json接口部署到Scrapyd上。