

网络爬虫

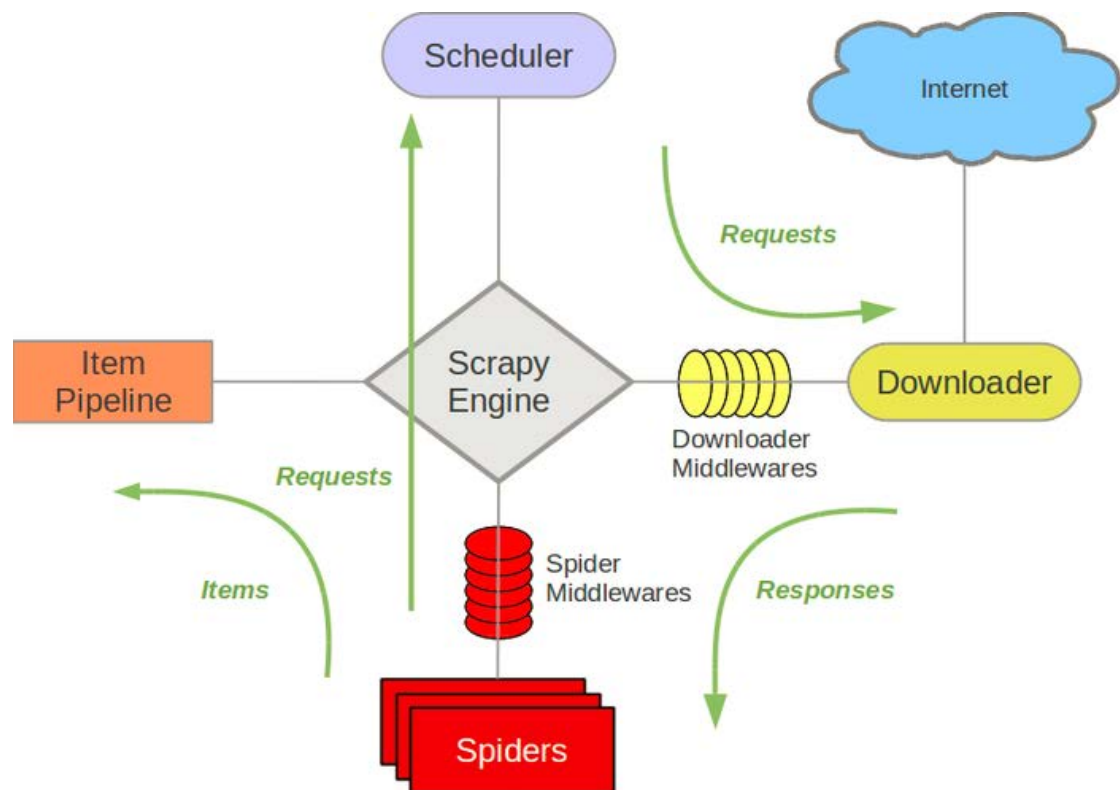
简要概述

该爬虫是基于 scrapy + redis + scrapyd 的分布式网络爬虫系统，实现从 4 个不同的网站：智联招聘、智联卓聘、Boss 直聘、以及 51 招聘上爬取职位招聘信息，包括：职位名称、职位链接、公司名称、工作地点、职位发布日期、职位招聘人数、职位类型、公司简介、职位具体信息、公司规模、公司类型、公司行业、公司地址、公司主页、专业要求等相关信息。最终将爬取数据存入到 MongoDB 数据库中，实现项目数据的获取任务。

具体实现

Scrapy

Scrapy 是一个为了爬取网站数据，提取结构性数据而编写的应用框架。其可以应用在数据挖掘，信息处理或存储历史数据等一系列的程序中。其最初是为了页面抓取（更确切来说，网络抓取）所设计的，也可以应用在获取 API 所返回的数据(例如 Amazon Associates Web Services) 或者通用的网络爬虫。Scrapy 用途广泛，可以用于数据挖掘、监测和自动化测试。Scrapy 使用了 Twisted 异步网络库来处理网络通讯。整体架构大致如下：



组件-Scrapy Engine

引擎负责控制数据流在系统中所有组件中流动，并在相应动作发生时触发事件。

组件-调度器（Scheduler）

调度器从引擎接受 request 并将他们入队，以便之后引擎请求他们时提供给引擎。

组件-下载器（Downloader）

下载器负责获取页面数据并提供给引擎，而后提供给 spider。

组件-Spiders

Spider 是 Scrapy 用户编写用于分析 response 并提取 item(即获取到的 item) 或额外跟进的 URL 的类。每个 spider 负责处理一个特定(或一些)网站。

组件-Item Pipeline

Item Pipeline 负责处理被 spider 提取出来的 item。典型的处理有清理、验证及持久化(例如存取到数据库中)。

组件-下载器中间件 (Downloader middlewares)

下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理 Downloader 传递给引擎的 response。其提供了一个简便的机制，通过插入自定义代码来扩展 Scrapy 功能。

组件-Spider 中间件 (Spider middlewares)

Spider 中间件是在引擎及 Spider 之间的特定钩子(specific hook)，处理 spider 的输入(response)和输出(items 及 requests)。其提供了一个简便的机制，通过插入自定义代码来扩展 Scrapy 功能。

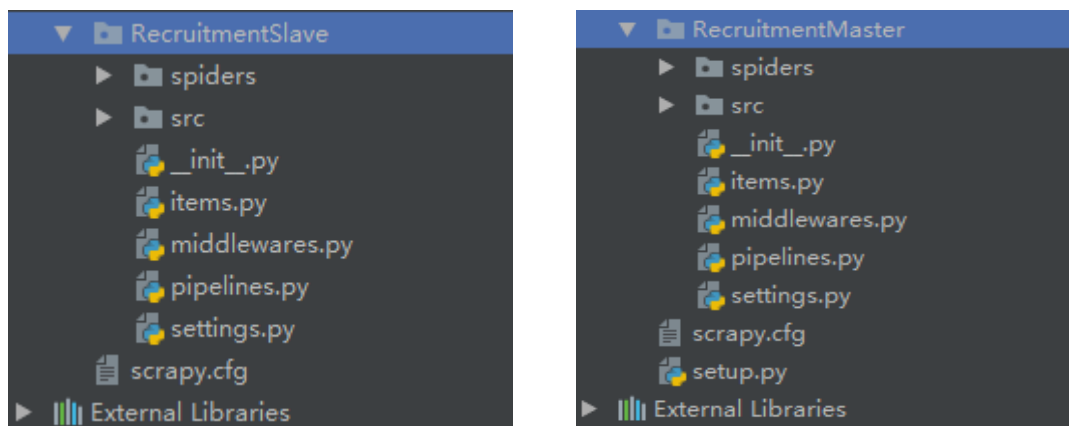
数据流 (Data flow)

Scrapy 中的数据流由执行引擎控制，其过程如下：

1. 引擎打开一个网站(open a domain)，找到处理该网站的 Spider 并向该 spider 请求第一个要爬取的 URL(s)。

2. 引擎从 Spider 中获取到第一个要爬取的 URL 并在调度器(Scheduler)以 Request 调度。
3. 引擎向调度器请求下一个要爬取的 URL。
4. 调度器返回下一个要爬取的 URL 给引擎，引擎将 URL 通过下载中间件(请求(request)方向)转发给下载器(Downloader)。
5. 一旦页面下载完毕，下载器生成一个该页面的 Response，并将其通过下载中间件(返回(response)方向)发送给引擎。
6. 引擎从下载器中接收到 Response 并通过 Spider 中间件(输入方向)发送给 Spider 处理。
7. Spider 处理 Response 并返回爬取到的 Item 及(跟进的)新的 Request 给引擎。
8. 引擎将(Spider 返回的)爬取到的 Item 给 Item Pipeline，将(Spider 返回的)Request 给调度器。
9. (从第二步)重复直到调度器中没有更多地 request，引擎关闭该网站。

通过以上对于 Scrapy 的基本了解，此项目的爬虫结构(结合 redis 后)如下图所示：



(此项目是基于 Python2.7 版本)

在 spider 中编写不同网站所对应的爬虫代码, 在 items.py 中定义每一个爬虫的爬取字段, 在 middlewares.py 中定义所要使用的中间插件, 例如: 用户代理池、IP 代理池等等, 在 pipelines.py 中编写每一个爬虫爬取下来存储信息的代码, 在 settings.py 中设置相关的组件以及申明。

其中主要了解一下 spider 文件中爬虫文件的结构:

```
import scrapy

class MySpider(scrapy.Spider):
    name = 'example.com'
    allowed_domains = ['example.com']
    start_urls = [
        'http://www.example.com/1.html',
        'http://www.example.com/2.html',
        'http://www.example.com/3.html',
    ]

    def parse(self, response):
        self.log('A response from %s just arrived!' % response.url)
```

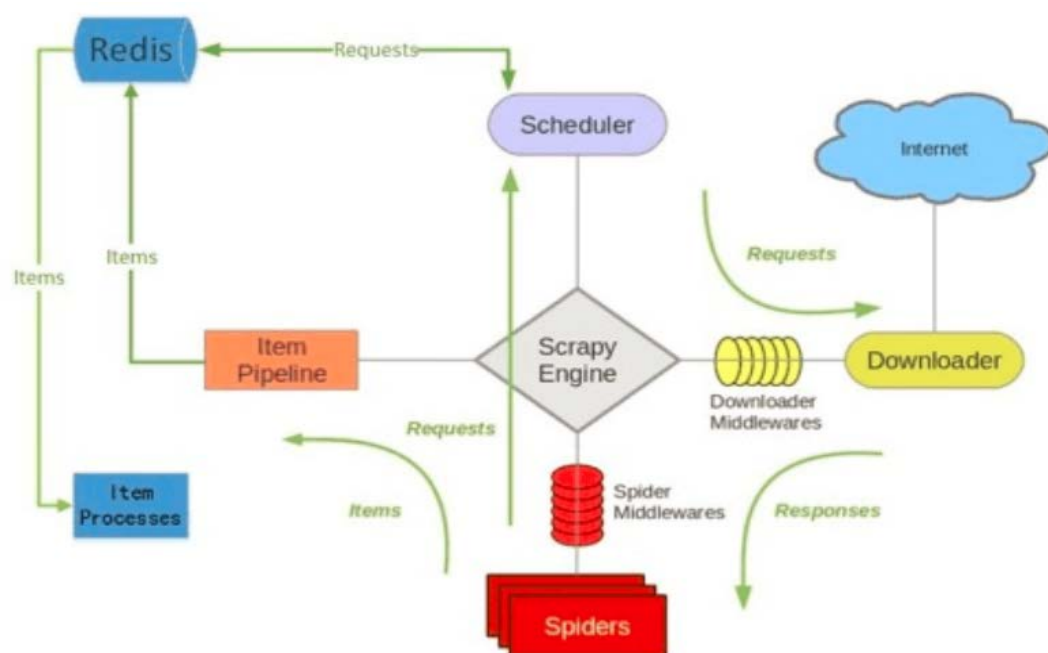
利用 import 引入 scrapy 包, 接着定义爬虫的 class 类, 满足 scrapy.Spider 或 crawl.Spider 方法, 然后为此爬虫设置一个独一无二的名字, 赋值 name。然后在 allowed_domains 中将所要爬取网站的域名填入。在 start_urls 中写入所要爬取网站的 url 列表, 之后的 parse 函数就会从 start_urls 的队列中取出 url 进行访问并且获取相关信息, 并解析。

在 parse 方法中, 可以利用正则表达式、xpath、beautifulsoup 等等解析网页的方法进行解析 (可以混合使用), 知道获取到我们所要找的信息, 利用 yield 函数返回 item 就可以了。所以此项目是在这个基础上进行复杂化, 但是基本原理没有变化, 唯独要注意的是, 在解析网页的时候, 由于每一个网页的网页结构不一样, 所以一个网站的解析方法只能试用与该网页, 其余网页不可以。在每一个网页解析过程中, 同一网站中相同的网页可能有不同的结构, 所以需要多次与长期调试代码来尽可能适应所有页面。

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 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。

加上 redis 后，上述 scrapy 的架构就变成下图所示：



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。

connect 文件引入了 redis 模块，这个是 redis-python 库的接口，用于通过 python 访问 redis 数据库，可见，这个文件主要是实现连接 redis 数据库的功能（返回的是 redis 库的 Redis 对象或者 StrictRedis 对象，这俩都是可以直接用来

进行数据操作的对象)。这些连接接口在其他文件中经常被用到。其中，我们可以看到，要想连接到 redis 数据库，和其他数据库差不多，需要一个 ip 地址、端口号、用户名密码（可选）和一个整形的数据库编号，同时我们还可以在 scrapy 工程的 setting 文件中配置套接字的超时时间、等待时间等。

组件-dupefilter.py

负责执行 request 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。

它重写了 scrapy 本身已经实现的 request 判重功能。因为本身 scrapy 单机跑的话，只需要读取内存中的 request 队列或者持久化的 request 队列（scrapy 默认的持久化似乎是 json 格式的文件，不是数据库）就能判断这次要发出的 request url 是否已经请求过或者正在调度（本地读就行了）。而分布式跑的话，就需要各个主机上的 scheduler 都连接同一个数据库的同一个 request 池来判断这次的请求是否是重复的了。

在这个文件中，通过继承 BaseDupeFilter 重写他的方法，实现了基于 redis 的判重。根据源代码来看，scrapy-redis 使用了 scrapy 本身的一个 fingerprint 接 request_fingerprint，这个接口很有趣，根据 scrapy 文档所说，他通过 hash 来判断两个 url 是否相同（相同的 url 会生成相同的 hash 结果），但是当两个 url 的地址相同，get 型参数相同但是顺序不同时，也会生成相同的 hash 结果，所以 scrapy-redis 依旧使用 url 的 fingerprint 来判断 request 请求是否已经出现过。这

个类通过连接 redis，使用一个 key 来向 redis 的一个 set 中插入 fingerprint（这个 key 对于同一种 spider 是相同的，redis 是一个 key-value 的数据库，如果 key 是相同的，访问到的值就是相同的，这里使用 spider 名字+DupeFilter 的 key 就是为了在不同主机上的不同爬虫实例，只要属于同一种 spider，就会访问到同一个 set，而这个 set 就是他们的 url 判重池），如果返回值为 0，说明该 set 中该 fingerprint 已经存在（因为集合是没有重复值的），则返回 False，如果返回值为 1，说明添加了一个 fingerprint 到 set 中，则说明这个 request 没有重复，于是返回 True，还顺便把新 fingerprint 加入到数据库中了。DupeFilter 判重会在 scheduler 类中用到，每一个 request 在进入调度之前都要进行判重，如果重复就不需要参加调度，直接舍弃就好了，不然就是白白浪费资源。

组件-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 LIFO 的 SpiderStack。默认使用的是第二种，这也就是出现之前文章中所分析情况的原因（[链接](#)）。

该文件实现了几个容器类，可以看这些容器和 redis 交互频繁，同时使用了我们上边 picklecompat 中定义的 serializer。这个文件实现的几个容器大体相同，只不过一个是队列，一个是栈，一个是优先级队列，这三个容器到时候会被 scheduler 对象实例化，来实现 request 的调度。比如我们使用 SpiderQueue 最为调度队列的类型，到时候 request 的调度方法就是先进先出，而实用 SpiderStack 就是先进后出了。

我们可以仔细看看 SpiderQueue 的实现，他的 push 函数就和其他容器的一

样，只不过 push 进去的 request 请求先被 scrapy 的接口 request_to_dict 变成了一个 dict 对象（因为 request 对象实在是比较复杂，有方法有属性不好串行化），之后使用 picklecompat 中的 serializer 串行化为字符串，然后使用一个特定的 key 存入 redis 中（该 key 在同一种 spider 中是相同的）。而调用 pop 时，其实就是从 redis 用那个特定的 key 去读其值（一个 list），从 list 中读取最早进去的那个，于是就先进先出了。

这些容器类都会作为 scheduler 调度 request 的容器，scheduler 在每个主机上都会实例化一个，并且和 spider 一一对应，所以分布式运行时会有一个 spider 的多个实例和一个 scheduler 的多个实例存在于不同的主机上，但是，因为 scheduler 都是用相同的容器，而这些容器都连接同一个 redis 服务器，又都使用 spider 名加 queue 来作为 key 读写数据，所以不同主机上的不同爬虫实例公用一个 request 调度池，实现了分布式爬虫之间的统一调度。

组件-picklecompat.py

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），所以我们存什么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 request 对象，至于为什么不实用 json 格式，我也不是很懂，item pipeline 的串行化默认用的就是 json。

组件-pipelines.py

实现分布式处理的作用。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。

pipeline 文件实现了一个 item pipeline 类, 和 scrapy 的 item pipeline 是同一个对象, 通过从 settings 中拿到我们配置的 REDIS_ITEMS_KEY 作为 key, 把 item 串行化之后存入 redis 数据库对应的 value 中 (这个 value 可以看出是个 list, 我们的每个 item 是这个 list 中的一个结点), 这个 pipeline 把提取出的 item 存起来, 主要是为了方便我们延后处理数据。

组件-scheduler.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。

scrapy-redis 所实现的两种分布式: 爬虫分布式以及 item 处理分布式就是由模块 scheduler 和模块 pipelines 实现。上述其它模块作为二者辅助的功能模块。

该组件重写了 scheduler 类, 用来代替 scrapy.core.scheduler 的原有调度器。其实对原有调度器的逻辑没有很大的改变, 主要是使用了 redis 作为数据存储的媒介, 以达到各个爬虫之间的统一调度。

scheduler 负责调度各个 spider 的 request 请求, scheduler 初始化时, 通过 settings 文件读取 queue 和 dupefilters 的类型 (一般就用上边默认的), 配置

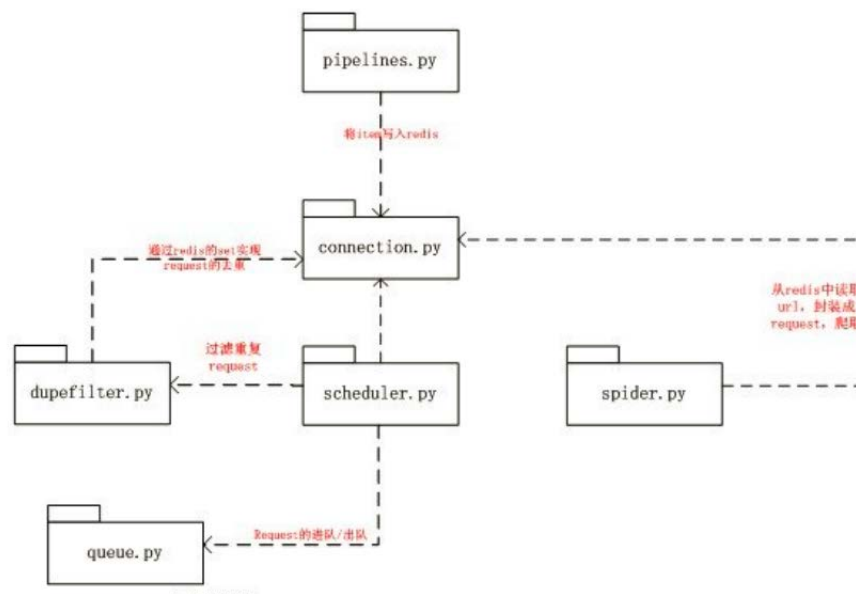
queue 和 dupefilters 使用的 key (一般就是 spider name 加上 queue 或者 dupefilters, 这样对于同一种 spider 的不同实例, 就会使用相同的数据块了)。每当一个 request 要被调度时, enqueue_request 被调用, scheduler 使用 dupefilters 来判断这个 url 是否重复, 如果不重复, 就添加到 queue 的容器中 (先进先出, 先进后出和优先级都可以, 可以在 settings 中配置)。当调度完成时, next_request 被调用, scheduler 就通过 queue 容器的接口, 取出一个 request, 把他发送给相应的 spider, 让 spider 进行爬取工作。

组件-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。

spider 的改动也不是很大, 主要是通过 connect 接口, 给 spider 绑定了 spider_idle 信号, spider 初始化时, 通过 setup_redis 函数初始化好和 redis 的连接, 之后通过 next_requests 函数从 redis 中取出 start url, 使用的 key 是 settings 中 REDIS_START_URLS_AS_SET 定义的 (注意了这里的初始化 url 池和我们上边的 queue 的 url 池不是一个东西, queue 的池是用于调度的, 初始化 url 池是存放入口 url 的, 他们都存在 redis 中, 但是使用不同的 key 来区分, 就当成是不同的表吧), spider 使用少量的 start_url, 可以发展出很多新的 url, 这些 url 会进入 scheduler 进行判重和调度。直到 spider 跑到调度池内没有 url 的时候, 会触发 spider_idle 信号, 从而触发 spider 的 next_requests 函数, 再次从 redis 的 start url 池中读取一些 url。

组件之间的关系

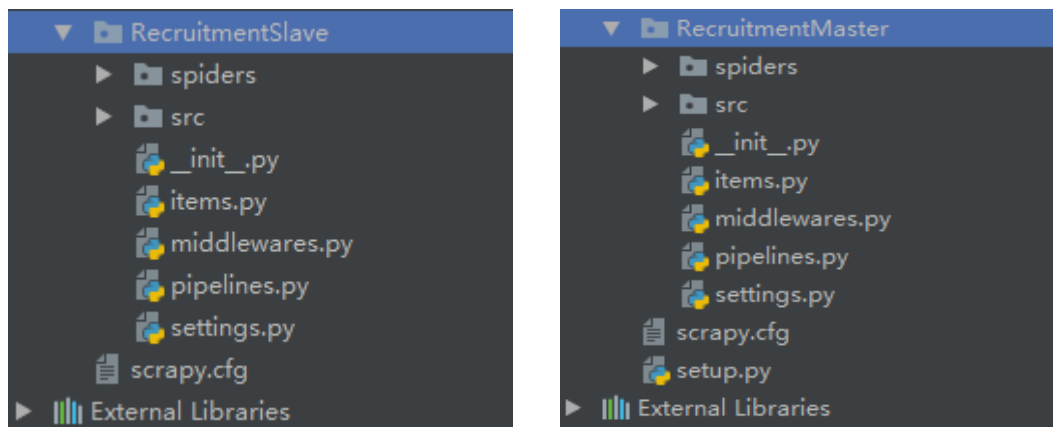


所以由以上可以知道 scrapy-redis 的总体思路：这个工程通过重写 scheduler 和 spider 类，实现了调度、spider 启动和 redis 的交互。实现新的 dupefilter 和 queue 类，达到了判重和调度容器和 redis 的交互，因为每个主机上的爬虫进程都访问同一个 redis 数据库，所以调度和判重都统一进行统一管理，达到了分布式爬虫的目的。

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 request 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 start url 池，拿去新的一批 url 入口，然后再次重复上边的工作。

所以加入 redis 后，就需要将原来项目中 scrapy 框架拆分为两部分：Master 和 Slave，如下图所示：



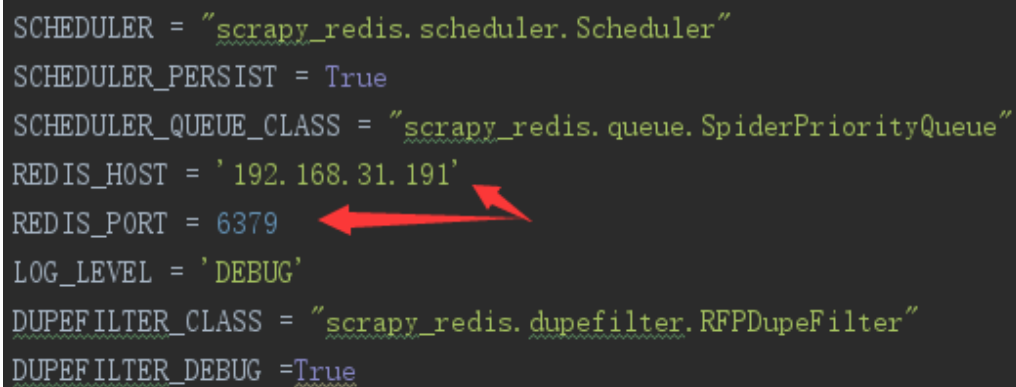
Master 端

Master 端主要是获取网站的 url，将其存入到 redis 的数据库中，让 redis 进行统一化调度管理。在此项目中只需要写入相应的 spider 文件，方法与 scrapy 中介绍的一致，只不过是只解析 url 并且返回 url。此外，我们只需要重写一个 InsertRedis.py 文件，来将我们所获取到的 url 插入到 redis 数据库中即可。

```
def InsertRequest(url, type):
    try:
        r = redis.Redis(host='127.0.0.1', port=6379, db=0)
        if type == 0:
            r.lpush('ZhiLianCrawl:requests', url)
        if type == 1:
            r.lpush('ZhuoPinCrawl:requests', url)
        if type == 2:
            r.lpush('BossCrawl:requests', url)
        if type == 3:
            r.lpush('ChinaHRCrawl:requests', url)
    except:
        logger.info("Redis open failed!!!")
```

Slave 端

Slave 端主要是从 redis 队列中取出 master 所获取的 url 进行请求即解析数据, 由于 redis 存在队列机制, 所以会保证每一个 slave 端取出的 url 不会有重复。首先我们需要将原来 spider 文件中的每一个的爬虫文件 class 方法改为 RedisSpider, 然后将原来的 start_url 替换为 redis_keys, 即每一个爬虫所对应的不同请求 url 队列, 之后 parse 中的解析方法不会发生变换, 最后任然是利用 yield 返回 item。在 slave 端中 item.py、pipelines.py、middlewares.py 和 scrapy 中一样, 在 settings.py 中加入下图所示内容:



```
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
SCHEDULER_PERSIST = True
SCHEDULER_QUEUE_CLASS = "scrapy_redis.queue.SpiderPriorityQueue"
REDIS_HOST = '192.168.31.191'
REDIS_PORT = 6379
LOG_LEVEL = 'DEBUG'
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
DUPEFILTER_DEBUG = True
```

其中 REDIS_HOST 是 master 端的主机 IP 地址, REDIS_PORT 是 master 端的 redis 所对应的端口号, 一般来说, redis 的统一端口号都是 6379。

Scrapyd

scrapyd 是运行 scrapy 爬虫的服务程序, 它支持以 http 命令方式发布、删除、启动、停止爬虫程序。而且 scrapyd 可以同时管理多个爬虫, 每个爬虫还可以有多个版本。此部署项目的前提: 操作系统-windows(安装有 Anaconda)、Linux 服务器 (Ubuntu_Server)、项目 python 版本-2.7。

1. 打开 Aconada Prompt, 新建虚拟环境 (方便管理), 在虚拟环境中安装 scrapy

项目需要使用到的包。除了下面提到的包之外，还需要自己在虚拟环境中安装要部署项目中使用的其他包：例如 pymongo (3.6.1)、parse (1.8.2)、bs4 (0.0.1)、setuptools (39.0.1)、beautifulsoup (4.6.0)、pywin32 (222)、requests (2.18.1)、scrapy (1.0.7) 等等。

2. 在 Aconda Prompt 中新建虚拟环境的方法及操作虚拟环境所使用的命令请参考：<https://blog.csdn.net/lyy14011305/article/details/59500819>。
3. 在虚拟环境命令框输入 activate XXX（所创建虚拟环境名称），接着安装 scrapyd 模块，scrapyd 模块是专门用于部署 scrapy 项目的，可以部署和管理 scrapy 项目。执行 pip install scrapyd==1.1.1。

```
C:\Users\qianzhen>workon scrapySpider
(scrapySpider) C:\Users\qianzhen>pip install scrapyd
Collecting scrapyd
```

等待安装完成。然后在命令行中输入 scrapyd 启动 scrapyd 服务。

```
(scrapySpider) C:\Users\qianzhen>scrapyd
2017-11-12T22:11:07+0800 [-] Loading c:\users\qianzhen\envs\scrapyspider\lib\site-packages\scrapyd\txapp.py...
2017-11-12T22:11:08+0800 [-] Scrapyd web console available at http://127.0.0.1:6800/
2017-11-12T22:11:08+0800 [-] Loaded.
2017-11-12T22:11:08+0800 [twisted.application.app.AppLogger#info] twisted 17.9.0 (c:\users\qianzhen\envs\scrapyspider\sc
ipts\python.exe 2.7.13) starting up.
2017-11-12T22:11:08+0800 [twisted.application.app.AppLogger#info] reactor class: twisted.internet.selectreactor.SelectR
actor.
2017-11-12T22:11:08+0800 [-] Site starting on 6800
2017-11-12T22:11:08+0800 [twisted.web.server.Site#info] Starting factory <twisted.web.server.Site instance at 0x03F186C
>
2017-11-12T22:11:08+0800 [Launcher] Scrapyd 1.2.0 started: max_proc=16, runner=u'scrapyd.runner'
```

在浏览器地址栏中输入：127.0.0.1:6800。显示如下则安装成功。



Scrapy

Available projects:

- [Jobs](#)
- [Logs](#)
- [Documentation](#)

<http://blog.csdn.net/zhaohig>

How to schedule a spider?

To schedule a spider you need to use the API (this web UI is only for monitoring)

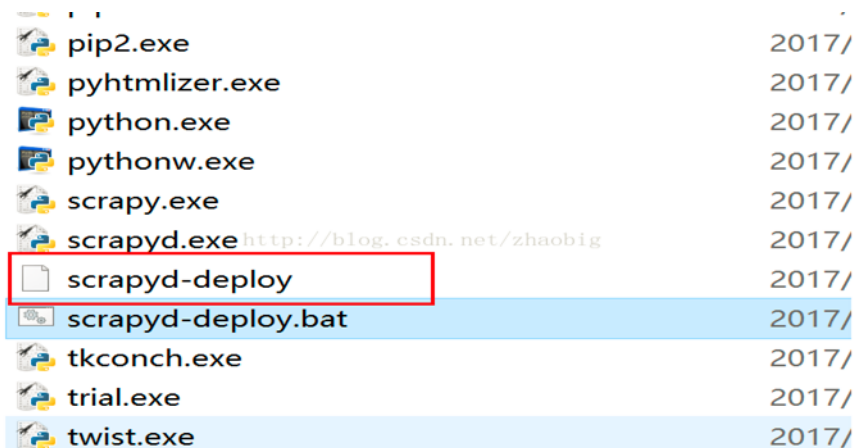
Example using [curl](#):

```
curl http://localhost:6800/schedule.json -d project=default -d spider=somespider
```

For more information about the API, see the [Scrapy documentation](#)

4. 重新打开新的 Aconada Prompt 进入虚拟环境中，安装 scrapy-client。

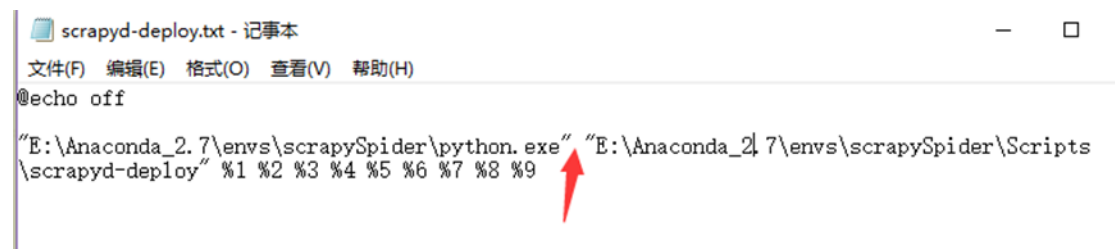
scrapyd-client 模块是专门打包 scrapy 爬虫项目到 scrapyd 服务中的，进入虚拟环境，执行命令 `pip install scrapyd-client==1.1.0`，安装完成后，在虚拟环境的 `scripts` 中会出现 `scrapyd-deploy` 无后缀文件，这个 `scrapyd-deploy` 无后缀文件是启动文件，在 Linux 系统下可以运行，在 windows 下是不能运行的，所以我们需要编辑一下使其在 windows 可以运行。



在含有 `scrapyd-deploy` 的文件夹中新建一个 `scrapyd-deploy.bat` 文件，右键选择编辑，输入以下配置，注意：两个路径之间是空格。

```
@echo off
```

```
"E:\Anaconda_2.7\envs\scrapySpider\python.exe" "E:\Anaconda_2.7\envs\scrapySpider\Scripts\scrapyd-deploy" %1 %2 %3 %4 %5 %6 %7 %8 %9
```



5. 进入虚拟环境，进入到你的爬虫项目中，进入带有 `scrapy.cfg` 文件的目录，执行命令 `scrapyd-deploy`，测试 `scrapyd-deploy` 是否可以运行，如果出现以下则正常。


```
(scrapySpider) C:\Users\qianzhen\Desktop\TotalSpider>scrapyd-deploy
Unknown target: default
http://blog.csdn.net/zhaobig
```

6. 打开爬虫项目中的 scrapy.cfg 文件, 这个文件就是给 scrapyd-deploy 使用的, 将 url 这行代码解掉注释, 并且给设置你的部署名称。

```
[settings]
default = TotalSpider.settings

[deploy:wj]
url = http://localhost:6800/
project = TotalSpider
```

部署名称
解注释
项目名称

7. 再次执行 scrapyd-deploy -l 启动服务, 可以看到设置的名称则正确。

```
(scrapySpider) C:\Users\qianzhen\Desktop\TotalSpider>scrapyd-deploy -l
wj
http://localhost:6800/
http://blog.csdn.net/zhaobig
```

8. 开始打包前, 执行一个命令: scrapy list, 这个命令执行成功说明可以打包了, 如果没执行成功说明还有工作没完成。执行 scrapy list 命令, 返回了爬虫名称说明一切 ok 了, 如下所示:

```
(scrapySpider) C:\Users\qianzhen\Desktop\TotalSpider>scrapy list
mainSpider
```

9. 到此我们就可以开始打包 scrapy 项目到 scrapyd 了, 用命令结合 scrapy 项目中的 scrapy.cfg 文件设置来打包。执行打包命令: scrapyd-deploy 部署名称 -p 项目名称。如: scrapyd-deploy wj -p TotalSpider。如下显示表示 scrapy 项目打包成功。

```
Packing version 1504715262
Deploying to project "adc" in http://localhost:6800/addversion.json
Server response (200):
{"version": "1504715262", "spiders": 1, "project": "adc", "node_name": "SKY-20160816NYP", "status": "ok"}
```

如果出现:

```
C:\Users\qianzhen\Desktop\TotalSpider>scrapydeploy wj -p TotalSpider
Packing version 1510540622
Deploying to project "TotalSpider" in http://localhost:6800/addversion.json
Server response (200):
{"status": "error", "message": "environment can only contain strings", "node_name": "wj"}
```

则重新多次提交，还是不可以的话重新启动 scrapyd 服务或者计算机。

- 然后在命令行工具（不用进入虚拟环境中）输入开启爬虫命令：
`curl http://localhost:6800/schedule.json -dproject=项目名称 -d spider=爬虫名称`
 如果出现如下所示可能会成功。

```
C:\Users\qianzhen>curl http://localhost:6800/schedule.json -d project=JobSpider -d spider=jobs
{"status": "ok", "jobid": "ada8314fc81c11e796cbf11b9b5ec792", "node_name": "wj"}
```

此时去网页中 127.0.0.1:6800 查看爬虫运行状态，如果如下所示则一切正常。

Jobs

[Go back](#)

Project	Spider	Job	PID	Start	Runtime	Finish	Log
Pending							
Running							
TotalSpider	mainSpider	f990a5f0c81e11e796cbf11b9b5ec792	10184	2017-11-13 11:01:44	0:00:00		日志 Log
Finished							
TotalSpider	mainSpider	b81f2d9ec81c11e796cbf11b9b5ec792		2017-11-13 10:45:34	0:02:29	2017-11-13 10:48:03	Log
JobSpider	jobs	ada8314fc81c11e796cbf11b9b5ec792		2017-11-13 10:45:15	0:03:07	2017-11-13 10:48:22	Log

以上所有都正常唯独在浏览器中没有正常显示出来，那么查看 scrapyd 服务中的信息，一般会有报错信息，如果出现：`scrapy spawnProcess not available since pywin32 is not installed`，认真查看自己是否已安装 pywin32 或者安装的是最新版的（卸载安装前期版本稳定性会好一点）。

停止爬虫：`curl http://localhost:6800/cancel.json -dproject=scrapy 项目名称 -d job=运行 ID`。例如：`curl http://localhost:6800/cancel.json -d project=ArticleSpider -d job=2a9b218a13e011e888cb28d2449bc99e`。

- 每次执行一些列更改的措施后，记得重新启动 scrapyd 服务器，否则可能会没有任何反应或者继续失败。

反爬虫机制

用户代理池

大多数情况下，网站都会根据我们的请求头信息来区分你是不是一个爬虫程序，如果一旦识别出这是一个爬虫程序，很容易就会拒绝我们的请求，因此我们需要给我们的爬虫手动添加请求头信息，来模拟浏览器的行为，但是当我们需要大量的爬取某一个网站的时候，一直使用同一个 User-Agent 显然也是不够的，因此，需要在 scrapy 中设置随机的 User-Agent。

在项目（Master 与 Slave）中的 settings.py 文件中加入大量用户代理如下图所示：

```
# 用户代理池
user_agent_list = [
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/22.0.1207.1 Safari/537.1",
    "Mozilla/5.0 (X11; CrOS i686 2268.111.0) AppleWebKit/536.11 (KHTML, like Gecko) Chrome/20.0.1132.57 Safari/536.11",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1092.0 Safari/536.6",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.6 (KHTML, like Gecko) Chrome/20.0.1090.0 Safari/536.6",
    "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/19.77.34.5 Safari/537.1",
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.9 Safari/536.5",
    "Mozilla/5.0 (Windows NT 6.0) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.36 Safari/536.5",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 5.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1063.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1062.0 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.2) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",
    "Mozilla/5.0 (Windows NT 6.1) AppleWebKit/536.3 (KHTML, like Gecko) Chrome/19.0.1061.1 Safari/536.3",

```

而后，在 middlewares.py 文件中添加如下的信息，这也是我们设置 User-Agent 的主要逻辑：

```
class RotateUserAgentMiddleware(object):
    def __init__(self, user_agent=''):
        self.user_agent = user_agent

    def process_request(self, request, spider):
        ua = random.choice(user_agent_list)
        if ua:
            spider.logger.info('User-Agent: %s' % ua)
            request.headers.setdefault('User-Agent', ua)
```

可以看到整个过程非常的简单，相关模块的导入就不说了，我们首先自定义了一个类，这个类继承自 RotateUserAgentMiddleware。之前已经说过，scrapy 为我们提供了 from_crawler() 的方法，用于访问相关的设置信息，这里就是用了这个方法，从 settings 里面取出我们的 USER_AGENT 列表，而后就是随机从列表中选择一个，添加到 headers 里面，最后默认返回了 None。

最后一步，就是将我们自定义的这个 RotateUserAgentMiddleware 类添加到 DOWNLOADER_MIDDLEWARES，像下面这样：

```
DOWNLOADER_MIDDLEWARES = {  
    # 'RecruitmentSlave.middlewares.RecruitmentSlaveDownloaderMiddleware': 543,  
    'scrapy.contrib.downloadermiddleware.useragent.UserAgentMiddleware': None,  
    'RecruitmentSlave.middlewares.RotateUserAgentMiddleware': 400,  
}
```

IP 代理池

除了切换 User-Agent 之外，另外一个重要的方式就是设置 IP 代理，以防止我们的爬虫被拒绝。有一些网站会设置访问阈值，也就是说，如果一个 IP 访问速度超过这个阈值，那么网站就会认为，这是一个爬虫程序，而不是用户行为。为了避免远程服务器封锁 IP，或者想加快爬取速度，一个可行的方法就是使用代理 IP，我们需要做的就是创建一个自己的代理 IP 池。

思路是：编写 Proxies.py 文件，专门用来爬取代理网站的代理：西刺代理、66 代理等，然后利用 ping 来从获取 IP 队列中可以 ping 的通的 IP，接着再将 ping 的通的 IP 队列中的 IP 访问百度页面，再次筛选出可以访问百度页面的 IP，这经过两次筛选，已经从原来的 IP 队列中获取到了实用以及有效的 IP。接着再 middlewares.py 中编写 HttpProxyMiddleware 方法。在该方法中如果主机的 IP 没有被封，那么利用主机的 IP 进行访问页面，如果主机的 IP 被封，那么执行该方

法。首先从 proxies.dat 文件中读取有效 IP，如果文件为空，那么执行 fetch_new_proxyes 方法，利用 Proxies.py 从网上获取有效 IP，接着将这些获取到的 IP 进行赋予有效定义："valid": True, "count": 0。前者为此 IP 是否有效，后者是记录此 IP。将获取到的 IP 全部赋予这样的定义，那么此时从这对 IP 队列中随机挑选一个 IP 进行访问，如果可以访问网站，那么将其 valid 赋值为 True，否者为 False，相应 count 赋值为 0 或 1。如果此 IP 可用，则将其写入 proxies.dat 文件中，一遍下一次继续使用。如果 IP 不可用，将此 IP 从 IP 队列中删除，再从其队列中获取一个 IP，进行上述任务，如果一个 IP 长期有效，那么就一直使用此 IP，直到失效。如果经过一段时间主机的 IP 可以继续访问，那么将不会在使用 IP 代理池中的 IP。最后将所有可用的 IP 以覆盖写的方式写入 proxies.dat 文件中，以便做到实时更新 IP 代理池。如果长时间 IP 代理池没有更新，那么自动更新 IP 代理池，防止出现循环 retrying。最后将此方法在 DOWNLOADER_MIDDLEWARES 配置即可。

运行截图

以 win 为 Mster，以三台 linux 虚拟机为相应的 Slave。

1. 在 win 上开启 redis 服务以及 scrapyd 服务 (linux 以默认打开):

```
PS D:\Program Files\redis> redis-server.exe redis.windows.conf

Redis 3.2.100 (00000000/0) 64 bit

Running in standalone mode
Port: 6379
PID: 10272

http://redis.io

[10272] 28 May 12:36:19.976 # Server started, Redis version 3.2.100
[10272] 28 May 12:36:20.052 * DB loaded from disk: 0.073 seconds
[10272] 28 May 12:36:20.052 * The server is now ready to accept connections on port 6379
```

```
(E:\Anaconda_2.7) C:\Users\Gavin>scrapyd
2018-05-28T12:38:22+0800 [-] Loading e:\anaconda_2.7\lib\site-packages\scrapyd\txapp.py...
2018-05-28T12:38:26+0800 [-] Scrapyd web console available at http://127.0.0.1:6800/
2018-05-28T12:38:26+0800 [-] Loaded.
2018-05-28T12:38:26+0800 [twisted.application.app.AppLogger#info] twistd 17.9.0 (e:\anaconda_2.7\python.exe 2.7.13) starting up.
2018-05-28T12:38:26+0800 [twisted.application.app.AppLogger#info] reactor class: twisted.internet.selectreactor.SelectReactor.
2018-05-28T12:38:26+0800 [-] Site starting on 6800
2018-05-28T12:38:26+0800 [twisted.web.server.Site#info] Starting factory <twisted.web.server.Site instance at 0x000000000308E648>
2018-05-28T12:38:26+0800 [Launcher] Scrapyd 1.2.0 started: max_proc=16, runner='scrapyd.runner'
```

2. Win 上远程连接三台服务器:

```
PS C:\Users\Gavin\Desktop> ssh server@192.168.72.138
server@192.168.72.138's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Sat May 26 07:31:08 2018 from 192.168.72.1
server@ubuntu: $
```

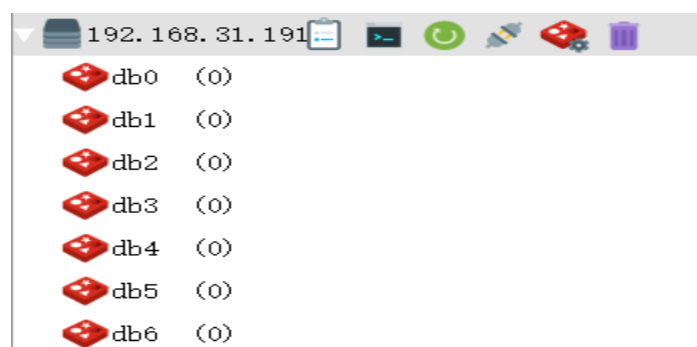
```
PS C:\Users\Gavin\Desktop> ssh server@192.168.72.140
server@192.168.72.140's password:
Permission denied, please try again.
server@192.168.72.140's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Fri May 25 07:29:12 2018
server@ubuntu: $
```

```
PS C:\Users\Gavin\Desktop> ssh server@192.168.72.141
server@192.168.72.141's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Fri May 25 07:29:28 2018
server@ubuntu: $
```

3. 此时 redis 数据库中 master 里面没有任何数据:



4. 在 linux 服务器上进行项目发布前的部署以及后期发布:

```
* Support:      https://ubuntu.com/advantage
Last login: Sat May 26 07:31:08 2018 from 192.168.72.1
server@ubuntu:~$ cd spiderproject/RecruitmentSlave/
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapy-deploy -l
slave
http://127.0.0.1:6800/
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapy list
Boss
ChinaHR
ZhiLian
ZhuoPin
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapy-deploy slave -p RecruitmentSlave
Packing version 1527482717
Deploying to project "RecruitmentSlave" in http://127.0.0.1:6800/addversion.json
Server response (200):
{"status": "ok", "project": "RecruitmentSlave", "version": "1527482717", "spiders":
4, "node_name": "ubuntu"}

server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ZhiLian
{"status": "ok", "jobid": "0dbe7b6c623211e8b037000c293652bb", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ZhuoPin
{"status": "ok", "jobid": "1384e6d0623211e8b037000c293652bb", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=Boss
{"status": "ok", "jobid": "17f2f068623211e8b037000c293652bb", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ChinaHR
{"status": "ok", "jobid": "1c593374623211e8b037000c293652bb", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$
```

```
PS C:\Users\Gavin\Desktop> ssh server@192.168.72.140
server@192.168.72.140's password:
Permission denied, please try again.
server@192.168.72.140's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Fri May 25 07:29:12 2018
server@ubuntu:~$ cd spiderproject/RecruitmentSlave/
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ZhiLian
{"status": "ok", "jobid": "2e3f52b2623211e88486000c2934e319", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ZhuoPin
{"status": "ok", "jobid": "31ca5aa8623211e88486000c2934e319", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=Boss
{"status": "ok", "jobid": "352f4410623211e88486000c2934e319", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule
.json -d project=RecruitmentSlave -d spider=ChinaHR
{"status": "ok", "jobid": "382d851e623211e88486000c2934e319", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$
```

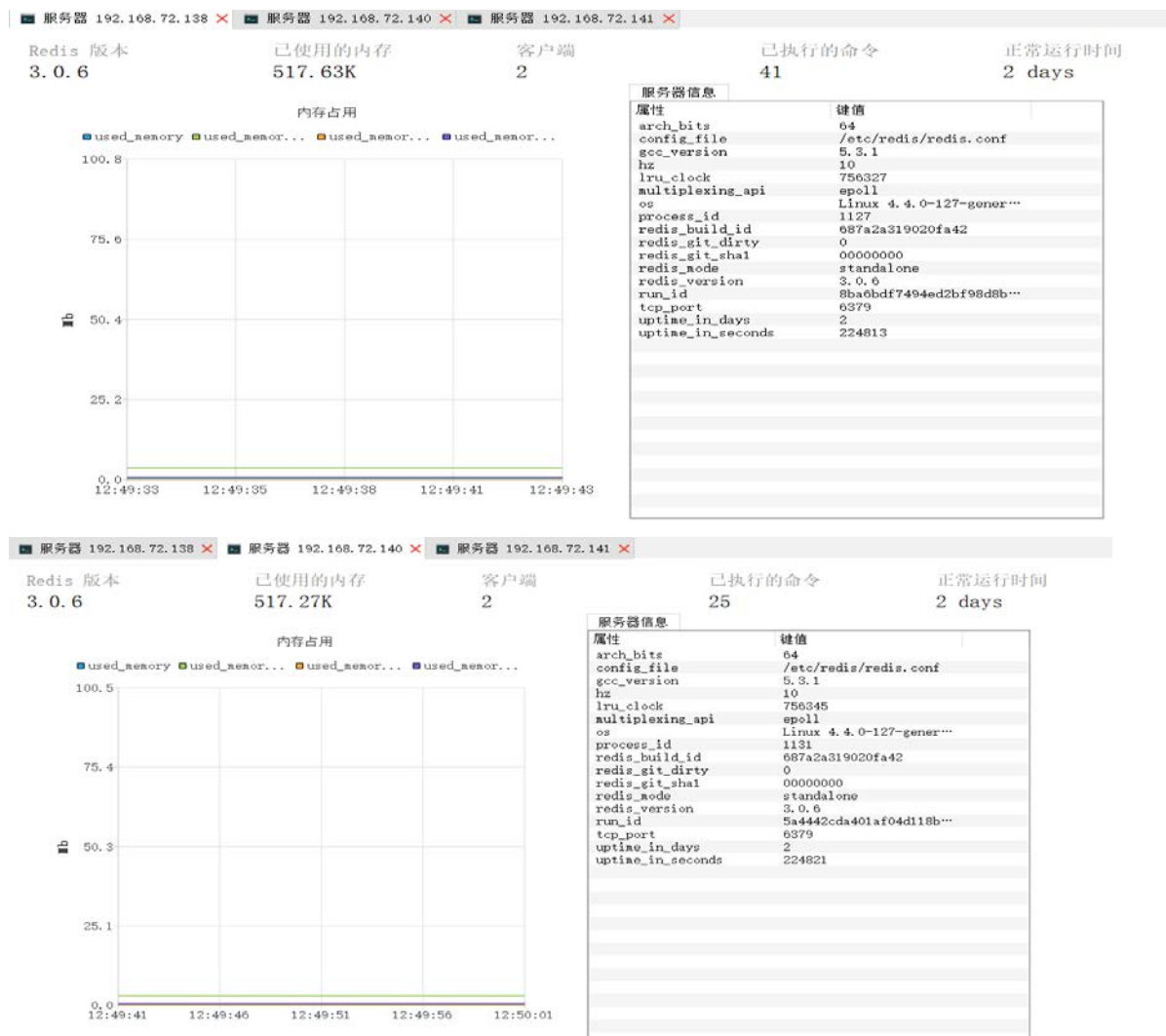
```

PS C:\Users\Gavin\Desktop> ssh server@192.168.72.141
server@192.168.72.141's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage
Last login: Fri May 25 07:29:28 2018
server@ubuntu:~$ cd spiderproject/RecruitmentSlave/
server@ubuntu:~/spiderproject/RecruitmentSlave$ ls
build  project.egg-info  RecruitmentSlave  scrapy.cfg  setup.py
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
project=RecruitmentSlave -d spider=Zhilian
{"status": "ok", "jobid": "3cbfc632623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
project=RecruitmentSlave -d spider=ZhuoPin
{"status": "ok", "jobid": "4104a0fa623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
project=RecruitmentSlave -d spider=Boss
{"status": "ok", "jobid": "44f5ea52623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
project=RecruitmentSlave -d spider=ChinaHR
{"status": "ok", "jobid": "49390af4623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu:~/spiderproject/RecruitmentSlave$

```

5. 发布后可以看到三台所占本机资源以及其他的情况，也就是发布成功：





6. Scrapyd 服务起开起成功后在浏览器可以看到如下信息：

← → ↻ 127.0.0.1:6800

应用 知乎 Google GitHub QQ邮箱 126邮箱 YouTube CSDN.NET 杭州电子科技大学

Scrapyd

Available projects: **JobInformation, example, RecruitmentMaster**

- [Jobs](#)
- [Logs](#)
- [Documentation](#)

How to schedule a spider?

To schedule a spider you need to use the API (this web UI is only for monitoring)

Example using [curl](#):

```
curl http://localhost:6800/schedule.json -d project=default -d spider=somespider
```

For more information about the API, see the [Scrapyd documentation](#)

Jobs

[Go back](#)

Project	Spider	Job	PID	Runtime	Log
Pending					
Running					
RecruitmentMaster	ZhiLian	7e063e80623411e88e50f8a963544c84	8324	0:00:36.330000	Log
RecruitmentMaster	ZhuoPin	86a5bbb0623411e8af71f8a963544c84	12576	0:00:21.382000	Log
RecruitmentMaster	ChinaHR	8be9158f623411e8b795f8a963544c84	8616	0:00:11.381000	Log
RecruitmentMaster	Boss	8f9de030623411e8b506f8a963544c84	924	0:00:06.385000	Log
Finished					

7. Master 在发布前的打包以及发布：

```

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>scrapyd-deploy -l
master
http://127.0.0.1:6800/

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>scrapy list
Boss
ChinaHR
ZhiLian
ZhuoPin

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster> scrapyd-deploy master -p Recruitment
Master
Packing version 1527483603
Deploying to project "RecruitmentMaster" in http://127.0.0.1:6800/addversion.json
Server response (200):
{"status": "ok", "project": "RecruitmentMaster", "version": "1527483603", "spiders": 4, "node
_name": "DESKTOP-M6VML7Q"}

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>curl http://localhost:6800/schedule.j
son -d project=RecruitmentSlave -d spider=ZhiLian
{"status": "error", "message": "Use \"scrapy\" to see available commands", "node_name": "DESK
TOP-M6VML7Q"}

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>curl http://localhost:6800/schedule.j
son -d project=RecruitmentMaster -d spider=ZhiLian
{"status": "ok", "jobid": "7e063e80623411e88e50f8a963544c84", "node_name": "DESKTOP-M6VML7Q"}

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>curl http://localhost:6800/schedule.j
son -d project=RecruitmentMaster -d spider=ZhuoPin
{"status": "ok", "jobid": "86a5bbb0623411e8af71f8a963544c84", "node_name": "DESKTOP-M6VML7Q"}

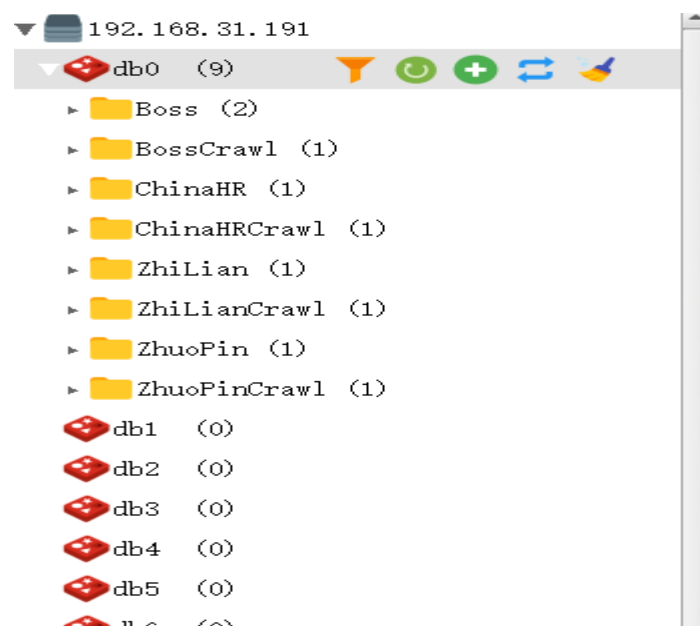
(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>curl http://localhost:6800/schedule.j
son -d project=RecruitmentMaster -d spider=ChinaHR
{"status": "ok", "jobid": "8be9158f623411e8b795f8a963544c84", "node_name": "DESKTOP-M6VML7Q"}

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>curl http://localhost:6800/schedule.j
son -d project=RecruitmentMaster -d spider=Boss
{"status": "ok", "jobid": "8f9de030623411e8b506f8a963544c84", "node_name": "DESKTOP-M6VML7Q"}

(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>

```

8. 此时 master 发布成功，会向 redis 数据库中添加 url，此时 slave 也开始从 master 这里获取 url，并返回解析后的数据：



任务管理器

文件(F) 选项(O) 查看(V)

进程 性能 应用历史记录 启动 用户 详细信息 服务

名称	状态	31% CPU	81% 内存	2% 磁盘	1% 网络	0% GPU	GPU 引擎
> Google Chrome (32 位) (14)		0%	274.7 MB	0 MB/秒	0.1 Mbps	0%	
> Microsoft Word (32 位)		0%	72.0 MB	0 MB/秒	0 Mbps	0%	
> Open source GUI managem...		1.7%	102.9 MB	0 MB/秒	0 Mbps	0%	
> TIM (32 位) (2)		0.2%	99.6 MB	0.1 MB/秒	0 Mbps	0%	
> VMware Workstation (32 ...		0.2%	57.0 MB	0 MB/秒	0 Mbps	0%	
> Windows PowerShell (3)		0.3%	49.4 MB	0 MB/秒	0 Mbps	0%	
> Windows PowerShell (3)		0%	28.8 MB	0 MB/秒	0 Mbps	0%	
> Windows PowerShell (3)		0%	26.3 MB	0 MB/秒	0 Mbps	0%	
> Windows PowerShell (3)		0%	28.5 MB	0 MB/秒	0 Mbps	0%	
> Windows 命令处理程序 (2)		0%	13.0 MB	0 MB/秒	0 Mbps	0%	
> Windows 命令处理程序 (7)		13.6%	309.9 MB	0.1 MB/秒	0.2 Mbps	0%	
> 任务管理器		0.6%	23.1 MB	0 MB/秒	0 Mbps	0%	

10. 终止项目部署或等待其自动结束:

```
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/cancel.json -d project=RecruitmentSlave -d job=fdc70cde602811e8b037000c293652bb
{"status": "ok", "prevstate": null, "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/cancel.json -d project=RecruitmentSlave -d job=0dbe7b6c623211e8b037000c293652bb
{"status": "ok", "prevstate": "running", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/cancel.json -d project=RecruitmentSlave -d job=17f2f068623211e8b037000c293652bb
{"status": "ok", "prevstate": "running", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/cancel.json -d project=RecruitmentSlave -d job=1c593374623211e8b037000c293652bb
{"status": "ok", "prevstate": "running", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$
```