大数据的出现得益于互联网行业的快速发展、计算机硬件和软件能力的不断提升。大数据技术现已被应用到各行各业，而在招聘求职领域，我们希望通过爬虫技术、文本挖掘、统计分析等手段帮助求职者更好的了解市场需求，从而有一个清晰、明确的求职方向。

基于大数据技术的岗位画像设计

宋剑波
高伟
李浩翔

目录

简介

1.1 项目简介

通过爬虫技术对网站上的求职信息进行收集，再利用大数据平台，对网上收集下来的数据进行分析挖掘，挖掘出岗位、工资、学历、待遇等不同因素之间的关系，形成高价值信息。

网络爬虫

2.1、爬虫简介

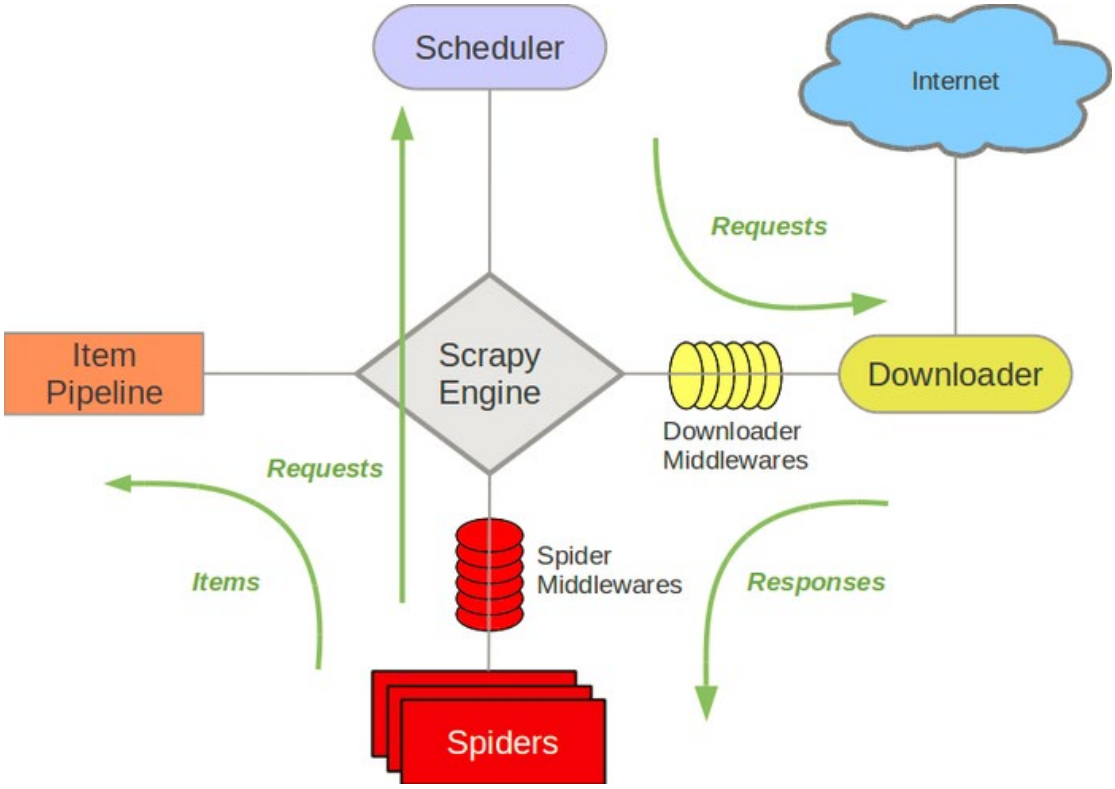
该爬虫是基于 scrapy +redis + scrapyd 的分布式网络爬虫系统，实现从 4 个不同的网站：智联招聘、智联卓聘、Boss 直聘、以及 51 招聘上爬取职位招聘信息，包括：职位名称、职位链接、公司名称、工作地点、职位发布日期、职位招聘人数、职位类型、公司简介、职位具体信息、公司规模、公司类型、公司行业、公司地址、公司主页、专业要求等相关信息。最终将爬取数据存入到 MongoDB 数据库中，实现项目数据的获取任务。

2.2、具体实现

见源代码。

2.2.1、scrapy 框架（高伟）

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）

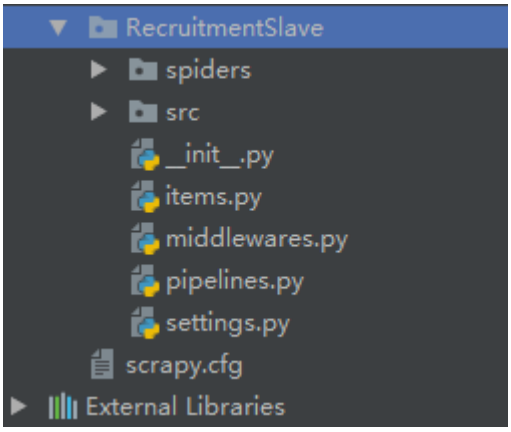
Scrapy 中的数据流由执行引擎控制，其过程如下:

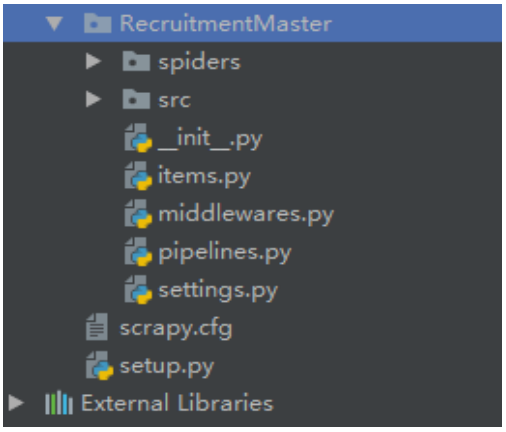
1. 引擎打开一个网站(open a domain)，找到处理该网站的 Spider 并向该 spider 请求第一

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。

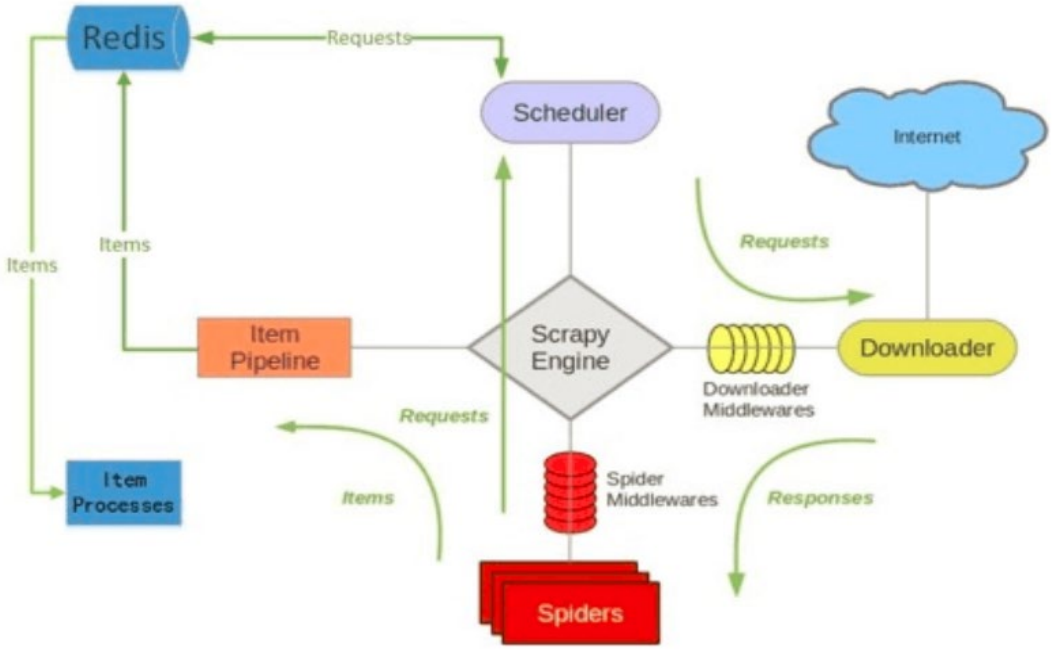
在 parse 方法中，可以利用正则表达式、xpath、beautifulsoup 等等解析网页的方法进行解析（可以混合使用），知道获取到我们所要找的信息，利用 yield 函数返回 item 就可以了。所以此项目是在这个基础上进行复杂化，但是基本原理没有变化，唯独要注意的是，在解析网页的时候，由于每一个网页的网页结构不一样，所以一个网站的解析方法只能试用与该网页，其余网页不可以。在每一个网页解析过程中，同一网站中相同的网页可能有不同的结构，所以需要多次与长期调试代码来尽可能适应所有页面。

2.2.2、scrapyd-redis 实现分布式爬虫（宋剑波 高伟）

2.2.2.1、架构

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 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：



基于 redis 的特性拓展了如下组件：

调度器（schedule）

scrapy-redis 调度器通过 redis 的 set 不重复的特性，巧妙的实现了 Duplication Filter 去重（DupeFilter set 存放爬取过的 request）。Spider 新生成的 request，将 request 的指纹到 redis 的 DupeFilter set 检查是否重复，并将不重复的 request push 写入 redis 的 request 队列。调度器每次从 redis 的 request 队列里根据优先级 pop 出一个 request，将此 request 发给 spider 处理。

Item Pipeline

将 Spider 爬取到的 Item 给 scrapy-redis 的 Item Pipeline，将爬取到的 Item 存入 redis 的 items 队列。可以很方便的从 items 队列中提取 item，从而实现 items processes 集群。

2.2.2.2、问题

1、 为什么使用 redis

redis 的特性体现在“内存数据库”和“KV”存储方式上，前者决定其性能，后者决定其存储内容的易于组织性。

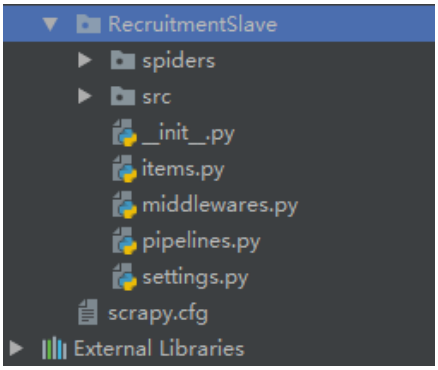
reidis 的使用适合链接的大量存取、快速调度的使用情景，最主要的是，由于链接需要的存储空间有限，内存的容量并不构成存储瓶颈，这时，存取速度（每秒十万次左右）便称为了 redis 的极大优势。

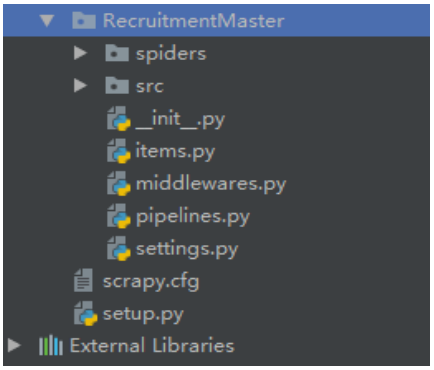
2、 redis master + slave

1、原理

在 Slave 启动并连接到 Master 之后，它将主动发送一个 SYNC 命令。此后 Master 将启动后台存盘进程，同时收集所有接收到的用于修改数据集的命令，在后台进程执行完毕后，Master 将传送整个数据库文件到 Slave，以完成一次完全同步。而 Slave 服务器在接收到数据库文件数据之后将其存盘并加载到内存中。此后，Master 继续将所有已经收集到的修改命令，和新的修改命令依次传送给 Slaves，Slave 将在本次执行这些数据修改命令，从而达到最终的数据同步。

将原来项目中 scrapy 框架拆分为两部分：Master 和 Slave，如下图所示：





2、配置

同时启动两个 Redis 服务器，分别监听不同的端口，如 6379 和 6380。长期保证这两个服务器之间的 Replication 关系，需要在 redis_6380 的配置文件中做如下修改 slaveof 127.0.0.1 6379（Master 和 Slave 在同一台主机，Master 的端口为 6379）。

3、 scrapy-redis 原理

scrapy-redis 是为了更方便地实现 scrapy 分布式爬取，Scrapy 本身是不支持分布式的，因为它的任务管理和去重全部是在机器内存中实现的。

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，相应的操作都要换，所以与队列相关的这些组件都做了更换。

scrapy-redis 提供了一个解决方法，把 deque 换成 redis 数据库，我们从同一个 redis 服

务器存放要爬取的 request，这样就能让多个 spider 去同一个数据库里读取，这样分布式的主要问题就解决了。

Scheduler 并不是直接就把 deque 拿来就粗暴的使用了，而且提供了一个比较高级的组织方法，它把待爬队列按照优先级建立了一个字典结构，比如：{priority0:队列 0priority1:队列 2priority2:队列 2}然后根据 request 中的 priority 属性，来决定该入哪个队列。而出列时，则按 priority 较小的优先出列。为了管理这个比较高级的队列字典，Scheduler 需要提供一系列的方法。但要是换了 redis 做队列，这个 scrapy 下的 Scheduler 就用不了。要有 scrapy-redis 的专用 scheduler。

4、验证 scrapy-redis 部署成功：

master 和 slave 的区别就是 settings.py 文件中对 scrapy-redis 的配置不同

要验证 scrapy-redis 是否究竟起作用，先启动 slave 的爬虫，然后观察 slave 的机器上的 redis 中究竟是否新生成了一个名为 dmoz:requests 的数据库，如果没有就证明 slave 正在爬取的 url 的确是从 master 中获取的，也就证明了 scrapy-redis 部署成功。

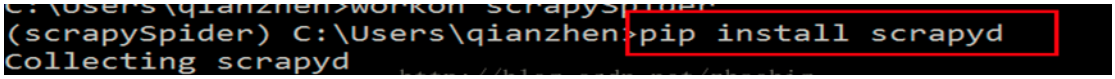
5、slave 机器上的爬虫是怎么读取 master 爬虫生成的 requests

Slava 爬虫读取 master 爬虫生成的 request 是通过 redis 数据库。redis 服务器存放着要爬取的 request，这样就能让多个 slave spider 去同一个数据库里读取 request。

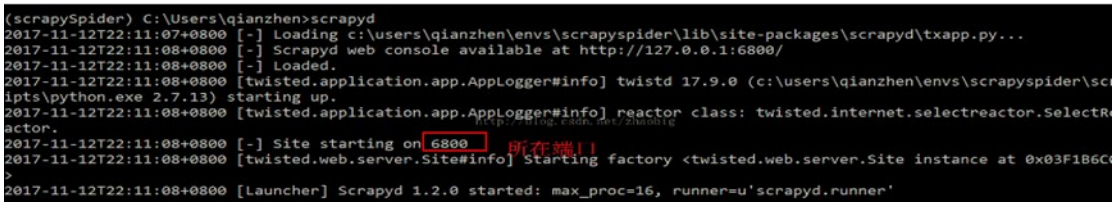
2.2.3、scrapyd 管理项目（高伟）

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。

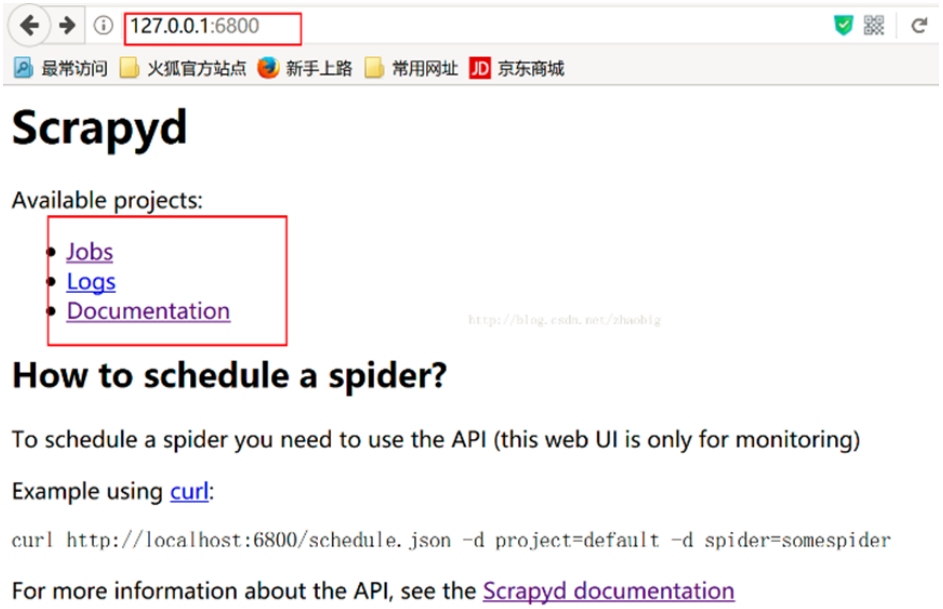
1. 打开 Aconada Prompt，新建虚拟环境（方便管理），在虚拟环境中安装 scrapy 项目需要使用到的包。除了下面提到的包之外，还需要自己在虚拟环境中安装要部署项目中使用的其他包：例如 pymongo（3.6.1）、parse（1.8.2）、bs4（0.0.1）、setuptools（39.0.1）、beautifulsoup（4.6.0）、pywin32（222）、requests（2.18.1）、scrapy（1.0.7）等等。
2. 在 Aconada Prompt 中新建虚拟环境的方法及操作虚拟环境所使用的命令请参考：https://blog.csdn.net/lyy14011305/article/details/59500819。
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。



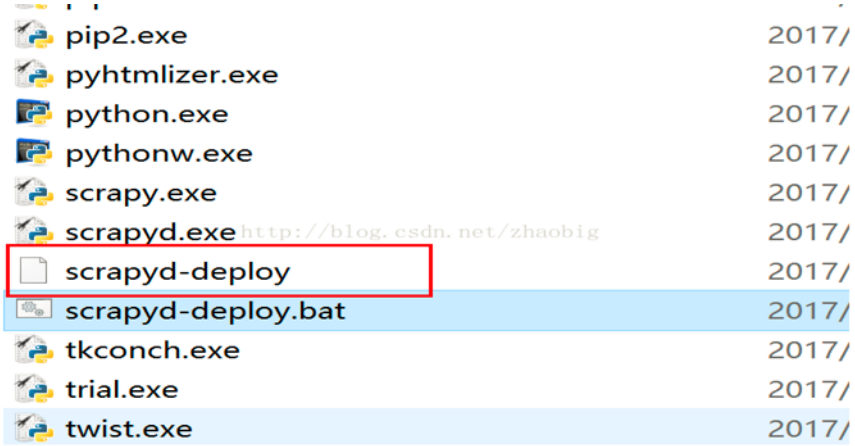
等待安装完成。然后在命令行中输入 scrapyd 启动 scrapyd 服务。



在浏览器地址栏中输入：127.0.0.1:6800。显示如下则安装成功。



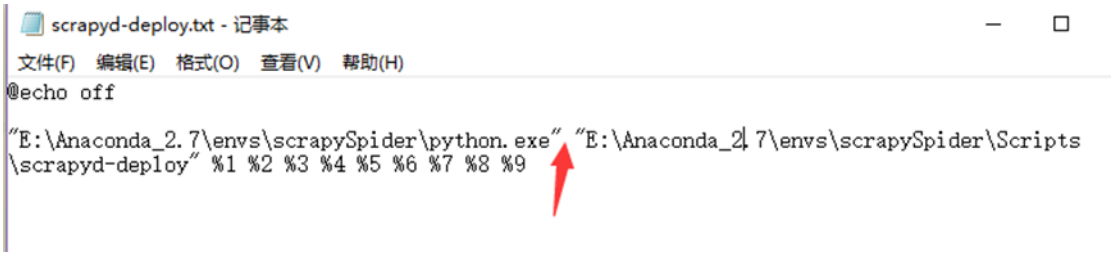
4. 重新打开新的 Aconada Prompt 进入虚拟环境中，安装 scrapy-client。scrapyd-client 模块是专门打包 scrapy 爬虫项目到 scrapyd 服务中的，进入虚拟环境，执行命令 pip install scrapyd-client==1.1.0，安装完成后，在虚拟环境的 scripts 中会出现 scrapyd-deploy 无后缀文件，这个 scrapyd-deploy 无后缀文件是启动文件，在 Linux 系统下可以远行，在 windows 下是不能运行的，所以我们需要编辑一下使其在 windows 可以运行。



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

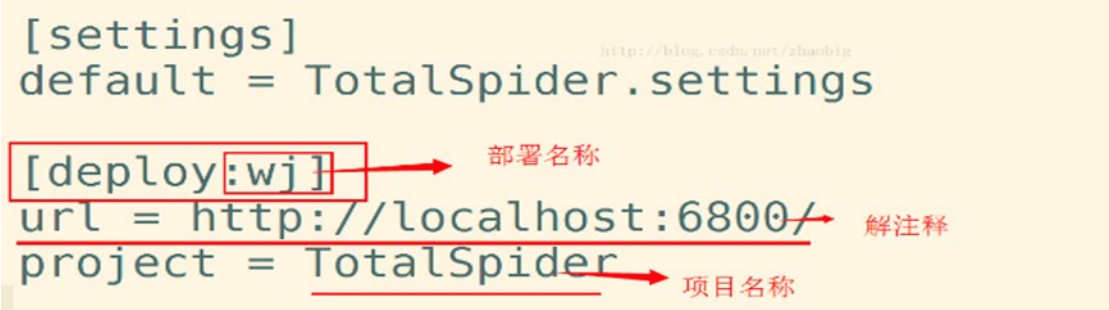
"E:\Anaconda_2.7\envs\scrapySpider\python.exe""E:\Anaconda_2.7\envs\scrapySpider\Scripts\scrapyd-deploy" %1 %2 %3 %4 %5 %6 %7 %8 %9



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
```


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。



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/
```


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：

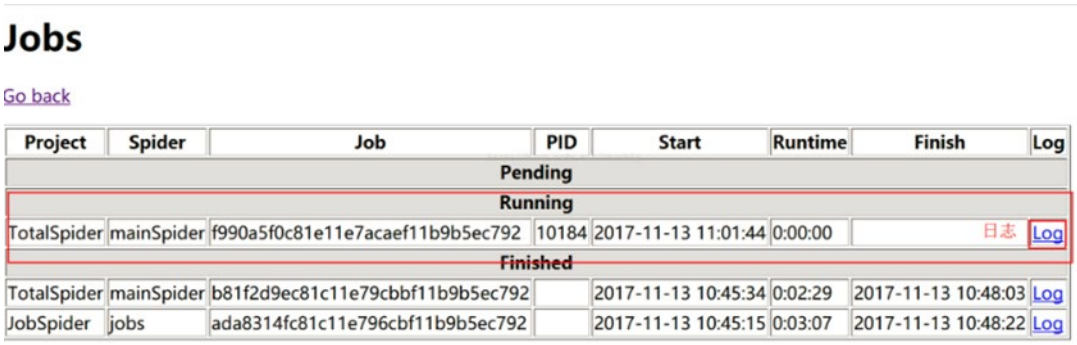
```
C:\Users\qianzhen\Desktop\TotalSpider>scrapyd-deploy wj -p TotalSpider
Packing version 1510540622
Deploying to project "TotalSpider" in http://localhost:6800/addversion.json
Server response (200):
{"status": "error", "message": "environment can only contain strings", "node_name": "wj"}
```


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。

10. 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。



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：scrapyd 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。

11. 每次执行一些列更改的措施后，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。

2.2.4、反爬虫机制实现（高伟）

2.2.4.1、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：

```python
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，这里就是用到了这个方法，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：

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

2.2.4.2、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。

2.2.5、docker 实现环境部署（宋剑波）

2.2.5.1、遇到问题

环境配置问题十分麻烦，包括如下几种情况。

1、我们在本地写好了一个 Scrapy 爬虫项目，想要把它放到服务器上运行，但是服务器上没有安装 Python 环境。

2、其他人给了我们一个 Scrapy 爬虫项目，项目使用包的版本和本地环境版本不一致，项目无法直接运行。

3、我们需要同时管理不同版本的 Scrapy 项目，如早期的项目依赖于 Scrapy 0.25，现在的项目依赖于 Scrapy 1.4.0。

在这些情况下，我们需要解决的就是环境的安装配置、环境的版本冲突解决等问题。对于 Python 来说，VirtualEnv 的确可以解决版本冲突的问题。但是，VirtualEnv 不太方便做项目部署，我们还是需要安装 Python 环境。

2.2.5.2、解决方案

Docker 可以用来解决上述问题。

Docker 可以提供操作系统级别的虚拟环境，一个 Docker 镜像一般都包含一个完整的操作系统，而这些系统内也有已经配置好的开发环境，如 Python 3.6 环境等。我们可以直接使用此 Docker 的 Python 3 镜像运行一个容器，将项目直接放到容器里运行，就不用再额外配置 Python 3 环境。这样就解决了环境配置的问题。

我们也可以进一步将 Scrapy 项目制作成一个新的 Docker 镜像，镜像里只包含适用于本项目的 Python 环境。如果要部署到其他平台，只需要下载该镜像并运行就好了，因为 Docker 运行时采用虚拟环境，和宿主机是完全隔离的，所以也不需要担心环境冲突问题。如果我们能够把 Scrapy 项目制作成一个 Docker 镜像，只要其他主机安装了 Docker，那么只要将镜像下载并运行即可，而不必再担心环境配置问题或版本冲突问题。

2.2.5.3、Docker 镜像使用

涉及到的相关使用有如下：

1、 拉取 Docker 镜像

Docker Hub 站点拉取公共的 Docker 镜像。

如：搜索 nginx ，拉取官方的 nginx 镜像。

docker pull nginx

2、 查看 Docker 镜像

docker images

3、 运行 Docker 镜像

docker run -it -v /Users/kwang/docker:/usr/share/nginx/html/hello -p 80:80 -d nginx:latest

-i 以交互模式运行容器，通常与 -t 同时使用

-t 为容器重新分配一个伪输入终端，通常与 -i 同时使用

-p 本机端口:容器端口 映射

-d 后台运行，并返回容器 ID

-v 可以将本机目录映射到容器内。如这里把本机的/home/kwang/docker 目录映射到/usr/share/nginx/html/hello/目录下

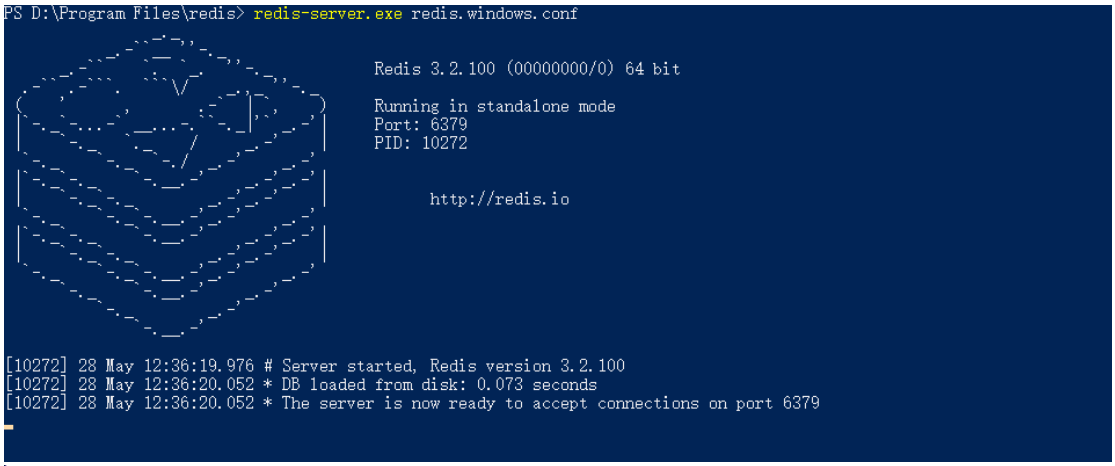
运行成功后，命令行返回一个 Docker 容器的 ID

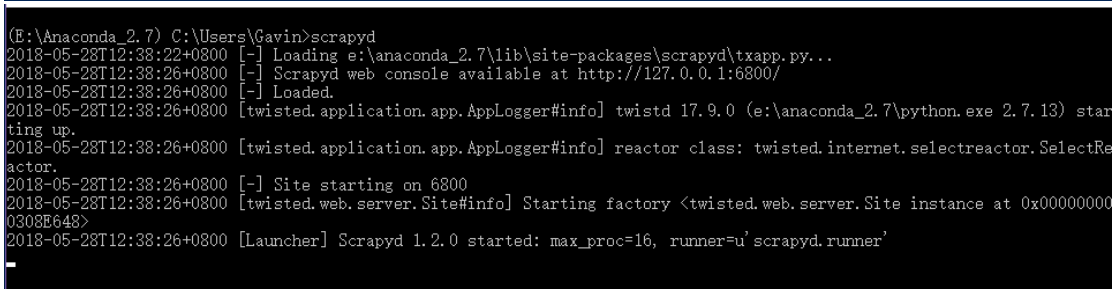
4、 查看运行中的 Docker 容器

2.3、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）：





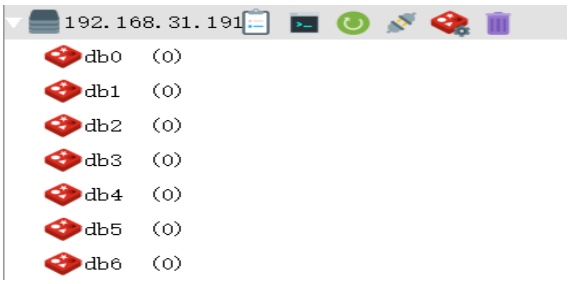
2. Win 上远程连接三台服务器：

```
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:



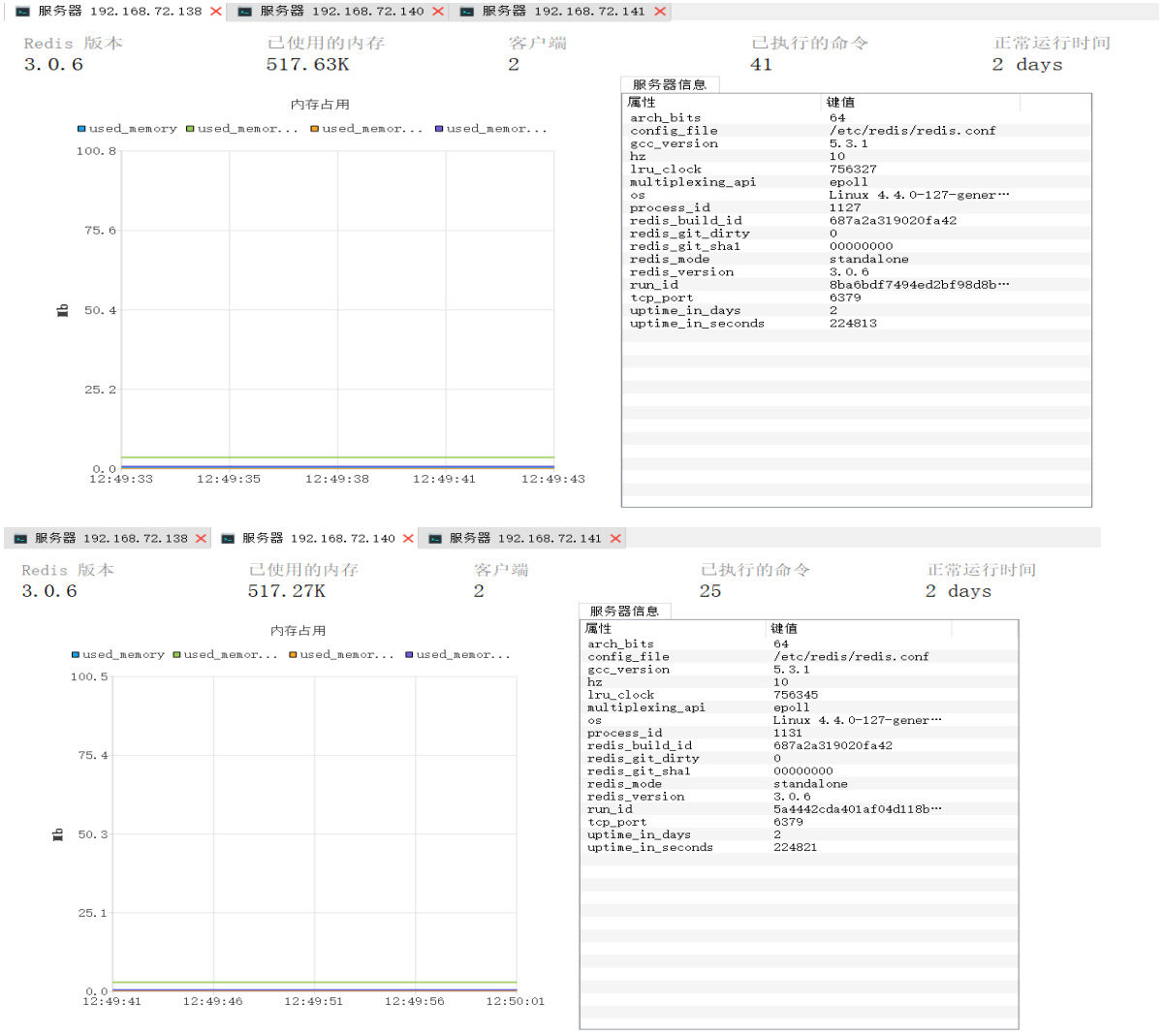
4. 在 linux 服务器上进行项目发布前的部署以及后期发布:

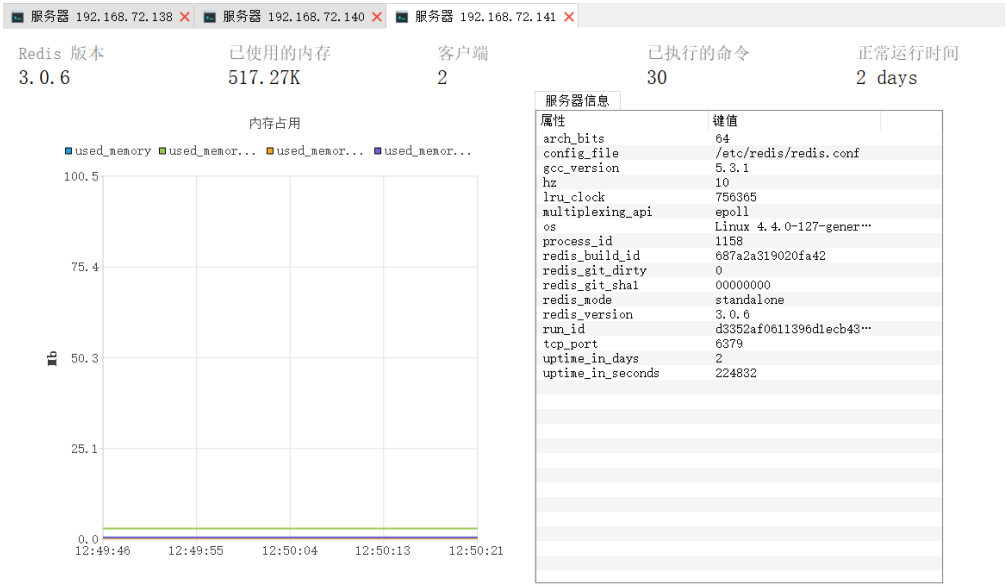
```
 * Support:        https://ubuntu.com/advantage
Last login: Sat May 26 07:31:08 2018 from 192.168.72.1
server@ubuntu:~$ cd spiderproject/RecruitmentSlave/
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapyd-deploy -l
slave                http://127.0.0.1:6800/
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapy list
Boss
ChinaHR
ZhiLian
ZhuoPin
server@ubuntu:~/spiderproject/RecruitmentSlave$ scrapyd-deploy slave -p Recruitment
Slave
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$
```

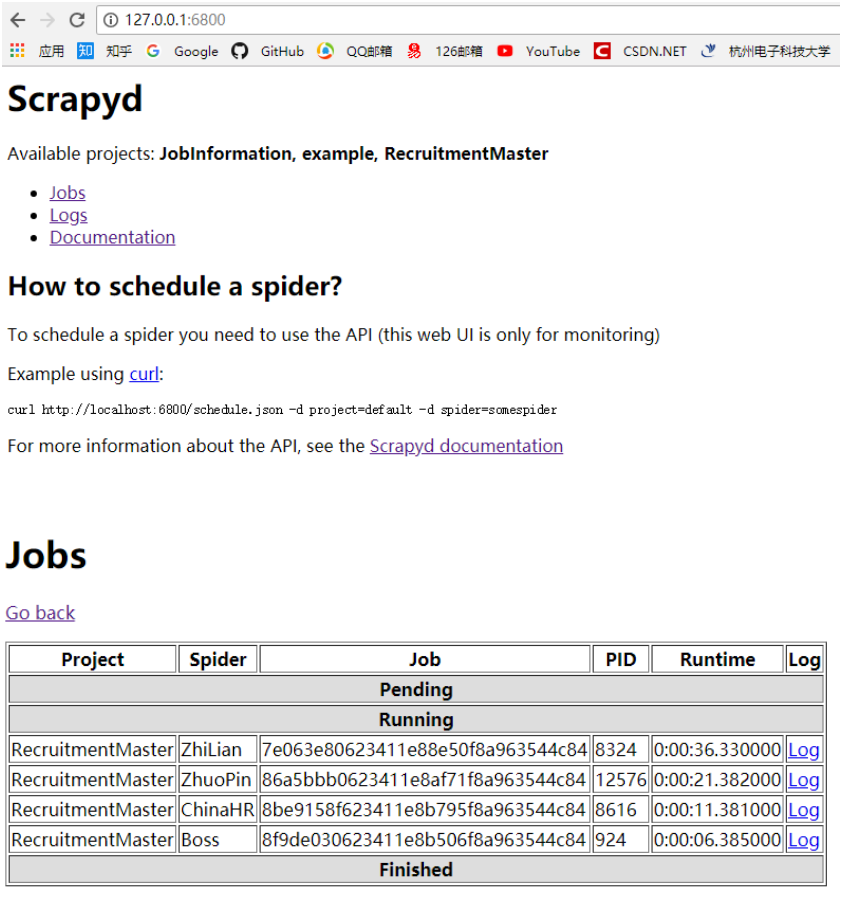
```
PS C:\Users\Gavin\Desktop> ssh server@192.168.72.141
server@192.168.72.141's password:
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * 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
roject=RecruitmentSlave -d spider=ZhiLian
{"status": "ok", "jobid": "3cbfc632623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
roject=RecruitmentSlave -d spider=ZhuoPin
{"status": "ok", "jobid": "4104a0fa623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
roject=RecruitmentSlave -d spider=Boss
{"status": "ok", "jobid": "44f5ea52623211e8bb21000c299a1751", "node_name": "ubuntu"}
server@ubuntu: /spiderproject/RecruitmentSlave$ curl http://localhost:6800/schedule.json -d p
roject=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：





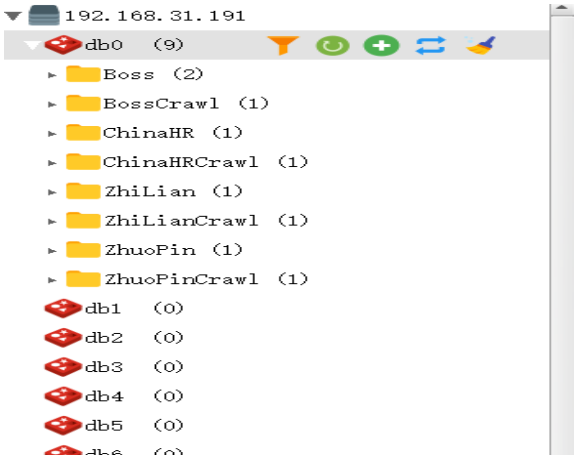
6. Scrapyd 服务起开起成功后在浏览器可以看到如下信息：

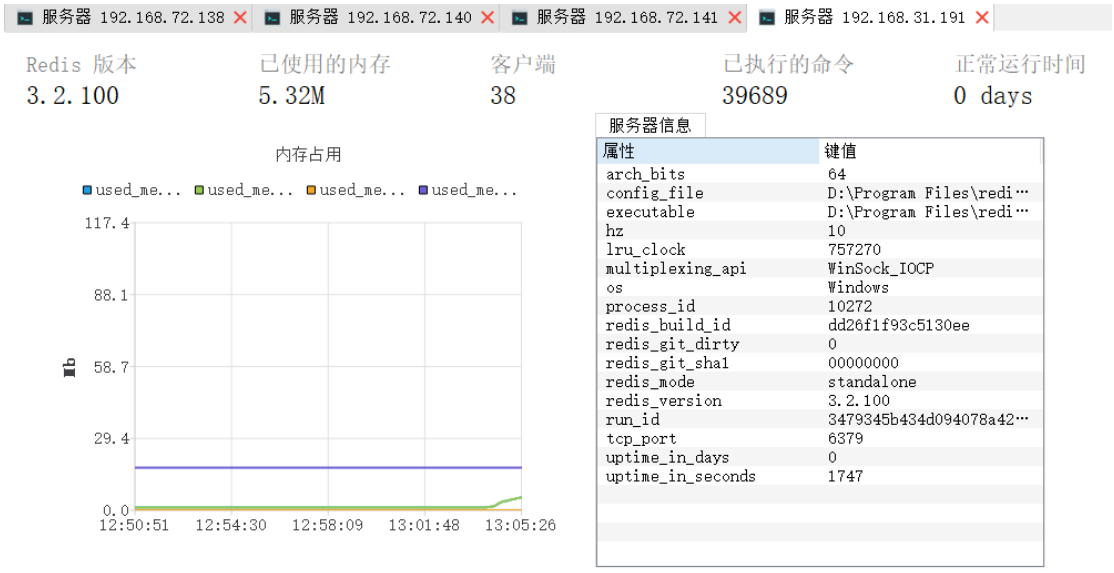


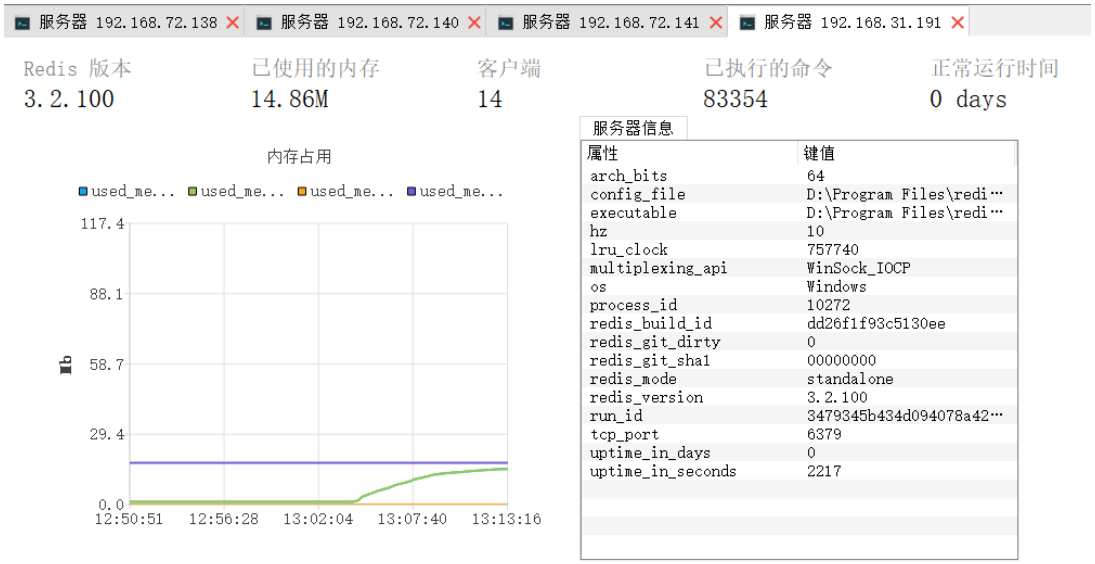
7. Master 在发布前的打包以及发布：

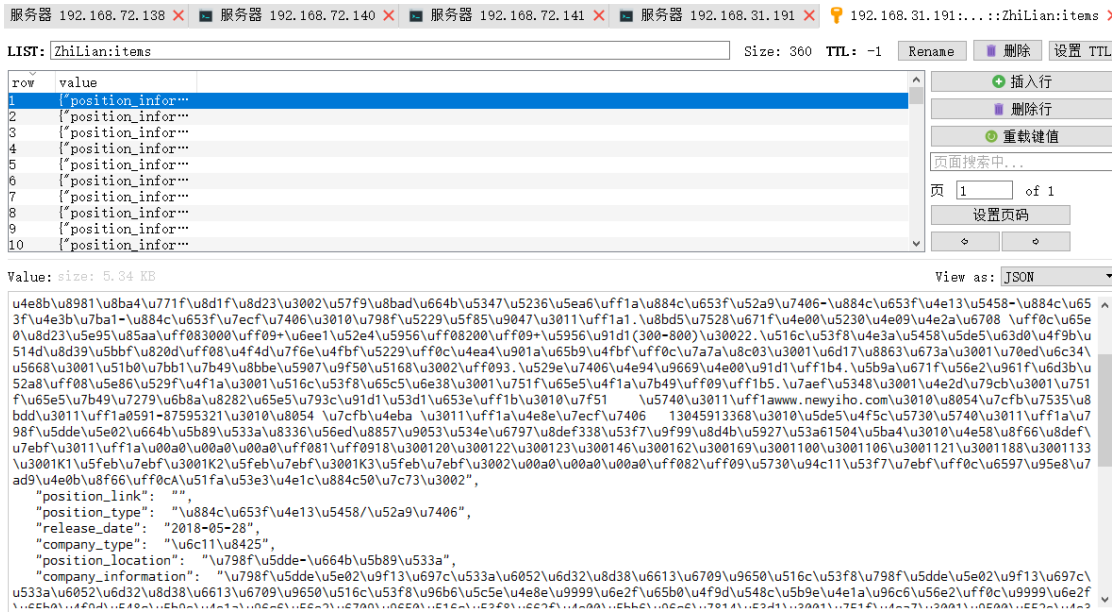
```
(scrapyspider) F:\SoftProgram\pycharm\RecruitmentMaster>scrapyd-deploy -l
master               http://127.0.0.1:6800/

(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：

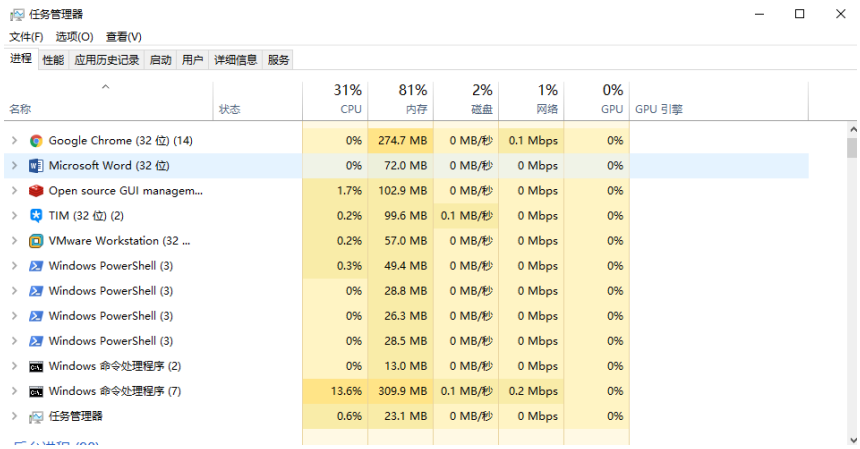








9. 此时主机的资源占用情况：



10. 终止项目部署或等待其自动结束：

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

数据分析与可视化

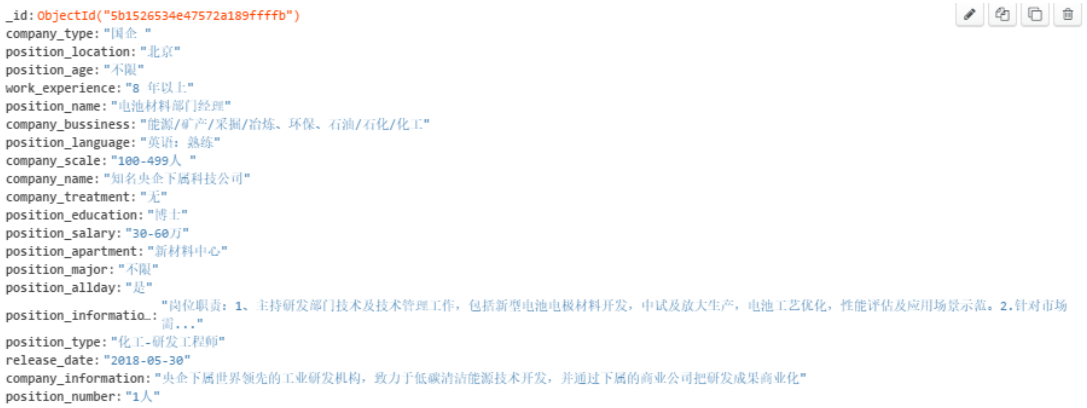
3.1、数据清洗（高伟）

由于从招聘网站上爬取下来的原始数据比较杂乱：部分字段缺失、部分字段格式不正确、大量的冗余数据、无用数据太多等等因素，在进行数据分析的时候，不利于项目的进行。因此进行数据清洗，将数据进行统一化，已达到数据分析时数据的要求：
清洗数据前数据库中数据如下：

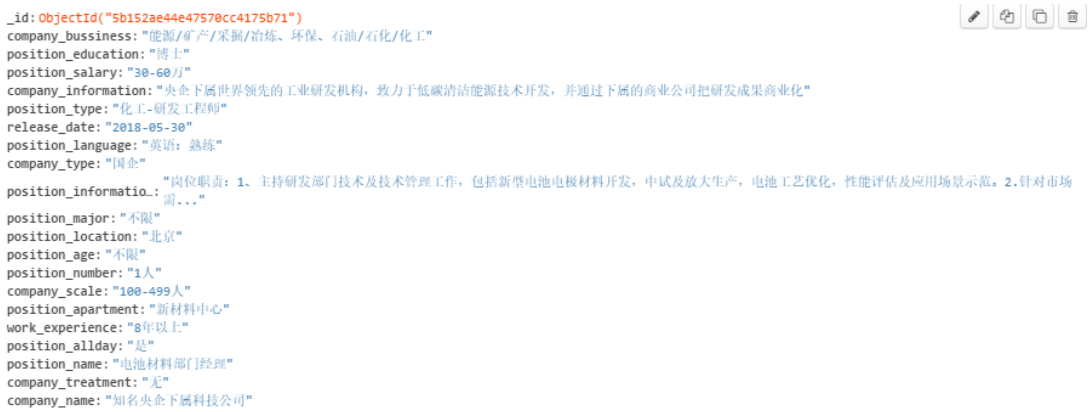

```
_id: ObjectId("5b107cc44e475722dc0476ac")
company_type: "国企 "
position_location: "北京"
position_age: "不限"
web_url: ""
work_experience: "8 年以上"
position_name: "电池材料部门经理"
company_bussiness: "能源/矿产/采掘/冶炼、环保、石油/石化/化工"
position_language: "英语：熟练"
company_scale: "100-499人 "
company_name: "知名央企下属科技公司"
company_treatment: ""
position_education: "博士"
position_salary: "30-60万"
position_apartment: "新材料中心"
position_major: "不限"
position_allday: "是"
position_informatio…: "岗位职责：1、主持研发部门技术及技术管理工作，包括新型电池电极材料开发，中试及放大生产，电池工艺优化，性能评估及应用场景示范。2.针对市场需...."
position_type: "化工-研发工程师"
release_date: "2018-05-30"
company_information: "央企下属世界领先的工业研发机构，致力于低碳清洁能源技术开发，并通过下属的商业公司把研发成果商业化"
company_address: ""
position_number: "1人"
position_overseas: ""
```


第一次清洗：去除 web_url、company_address、position_overseas 等无用数据，然后将数据

中 company_treatment 字段不存在数数据赋值为‘无’，之后存入数据库。
第一次数据清洗后数据库中数据如下：



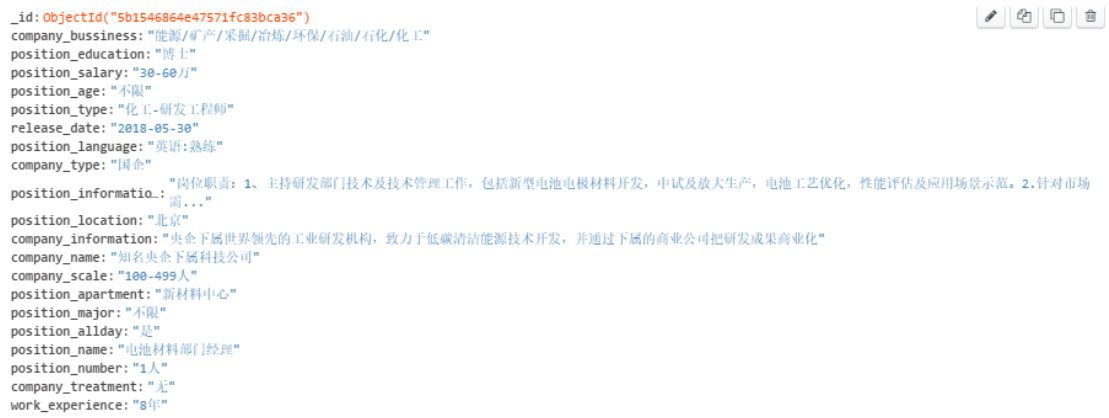
第二次清洗：将数据项中的每个数据规格化，将字段中的空格号全部清除，之后存入数据库。
第二次数据清洗后数据库中数据如下：



第三次清洗：在上述工作完成后，由于数据项中部分数据不规整，所以需要将其规整化，利用正则匹配将数据中的不利因素找出，然后将其替换。主要涉及到的字段：company_bussiness、position_type、position_language、position_major、position_location、position_age、company_scale、work_experience、position_name。其正则匹配的模式如下：

```
patten1 = re.compile(r'（.*?）|\(.*?\)')
patten2 = re.compile(r'、|及|+|,')
patten3 = re.compile(r',')
patten4 = re.compile(r'-其他')
patten5 = re.compile(r'支持-|支持/')
patten6 = re.compile(r'、|，|或')
patten7 = re.compile(r'相关专业|及相关|\.\.\.|类相关专业|类|相关|专业')
patten8 = re.compile(r'以下|以上|及以上')
patten9 = re.compile(r'\\|、|，|/ ')
```

然后将其存入数据库中。
经过三次数据清洗，数据库中的是数据如下：



3.2、岗位工资的影响因素（宋剑波）

3.2.1、分析

在岗位工资的影响因素方面，首先需要选择一个方面的职业与城市，我们选择北京的互联网行业作为分析对象，首先从清洗好的 json 数据中筛选出北京的互联网相关数据。

然后确定需要分析的因素。我选择的是：

总体薪酬情况（x:薪酬（万/年） y:value）

工作经验要求（x:工作经验（1-3 年等） y:工作数量）

不同工作经验的薪酬情况（x:工作经验（1-3 年等） y:薪酬（万/年））

学历要求（x:学历（本科、硕士等） y:工作数量）

不同学历的薪酬情况（x:学历（本科、硕士等）y:薪酬（万/年））

未实现：

技能关键词（x:工作数量 y:技能）

不同技能的薪酬水平（x:技能 y: 薪酬（万/年） 气泡图：气泡大小表示需求量的大小）

3.2.2、遇到问题

目前有丰富的数据分析函数库，比如基于 Python 数据分析和处理的类库：numpy，matplotlib，sklearn 和 networkx 等。不同的类库有各自的优势，我们该如何选择。

类库中具体的函数参数繁多，搞清楚怎么使用。

3.2.3、解决方案

Numpy 是科学计算包擅长数组处理，多维数组的产生与访问，还有庞大的函数库，包括：求和、平均值、方差、最值、排序、矩阵、矩阵类和线性代数模块等。

Matplotlib 是画图工具包，擅长画图，点和线多形态展示、坐标轴变换等。

Scikit-learn 是机器学习工具集，里面有很多机器学习相关的算法（如聚类算法，SVM

等)。

Networkx 是复杂网络分析库，内置了常用的图与复杂网络分析算法，可以方便的进行复杂网络数据分析、仿真建模等工作。

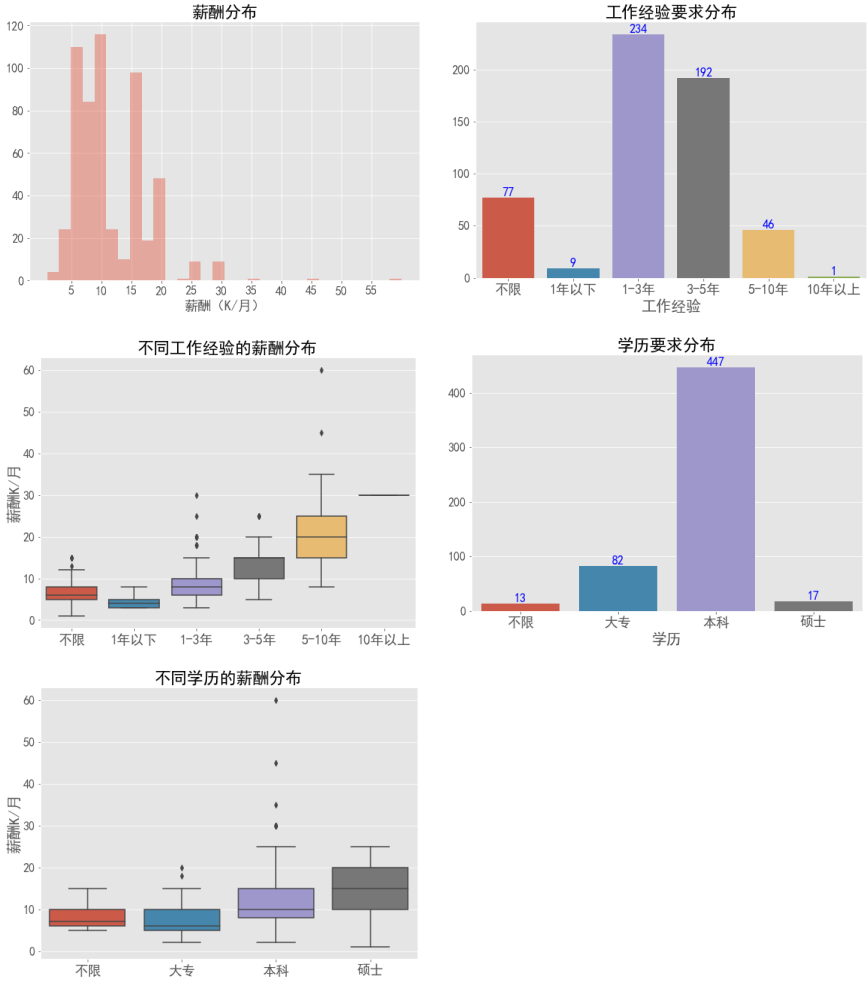
经过分析，我所需求的数据可视化多数是直方图。故我们选择 matplotlib 库进行数据可视化。

Matplotlib 应该是基于 python 语言最优秀的绘图库了，但是它也有一个十分令人头疼的问题，那就是太过于复杂了。3000 多页的官方文档，上千个方法以及数万个参数，属于典型的可以用它做任何事，但又无从下手。尤其是，想通过 Matplotlib 调出非常漂亮的效果时，往往会非常麻烦。Seaborn 基于 Matplotlib 核心库进行了更高级的 API 封装，可以轻松地画出更漂亮的图形。

故最后我们使用 matplotlib 和 seaborn 来实现可视化。

关于具体函数的使用，只能参考相关文档以及相关实例。

最后的实现效果如下:



3.3、岗位能力需求图谱 （李浩翔）

3.3.1 分析

首先选择要分析的岗位，以下的例子我选择了对软件工程师这个岗位进行分析。然后需

要对其职业要求进行处理，最后在饼状图中呈现出来。

3.3.2 遇到问题

如何从如此多的数据中获取到岗位所真正需要的能力，以及如何确定这些能力所占的比重

3.3.3 解决方案

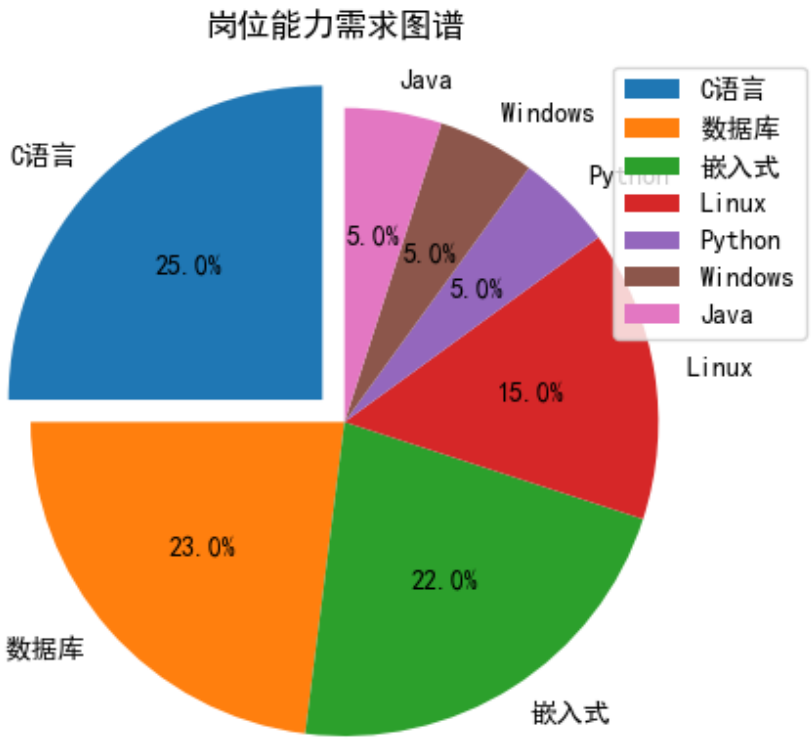
通过 jieba 分词，对岗位要求的文本进行分词并计算词频，删除没用的词语，选出前 7 个有效的能力要求，并按照词频分配权重。

Jieba 分词支持三种分词模式：

1．精确模式，试图将句子最精确的切开，适合文本分析；
2．全模式，把句子中所有的可以成词的词语都扫描出来，速度非常快，但是不能解决歧义；
3．搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎。

支持繁体分词，支持自定义词典，MIT 授权协议。

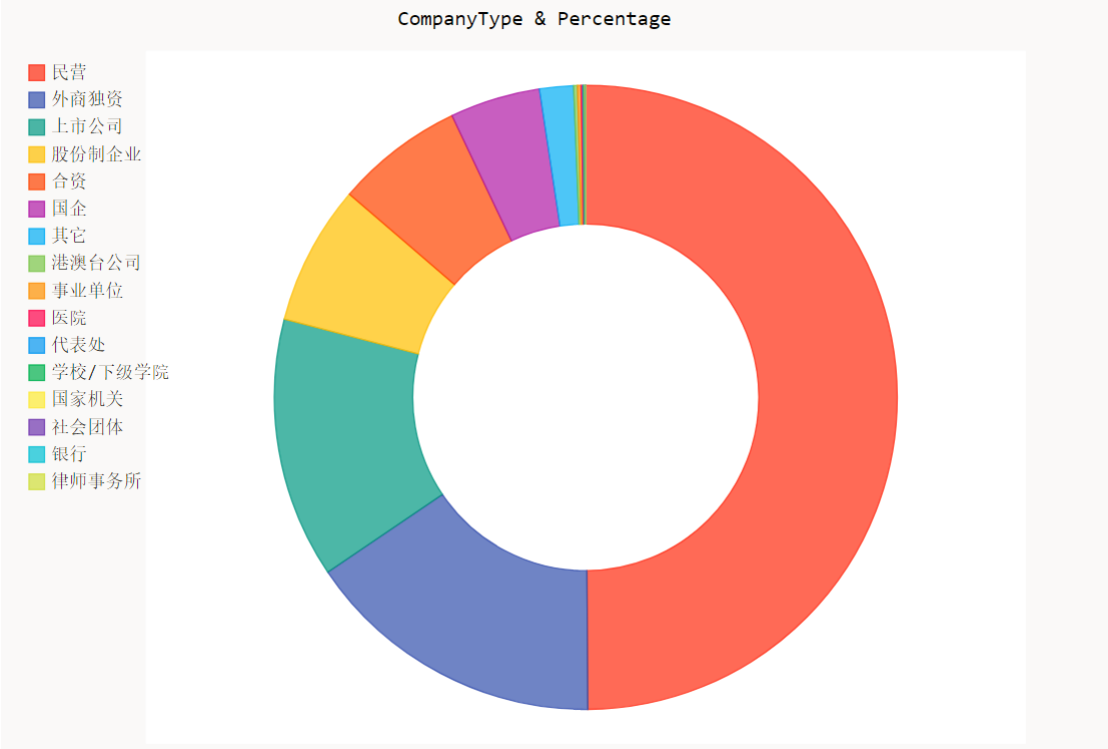
最后实现效果如下：

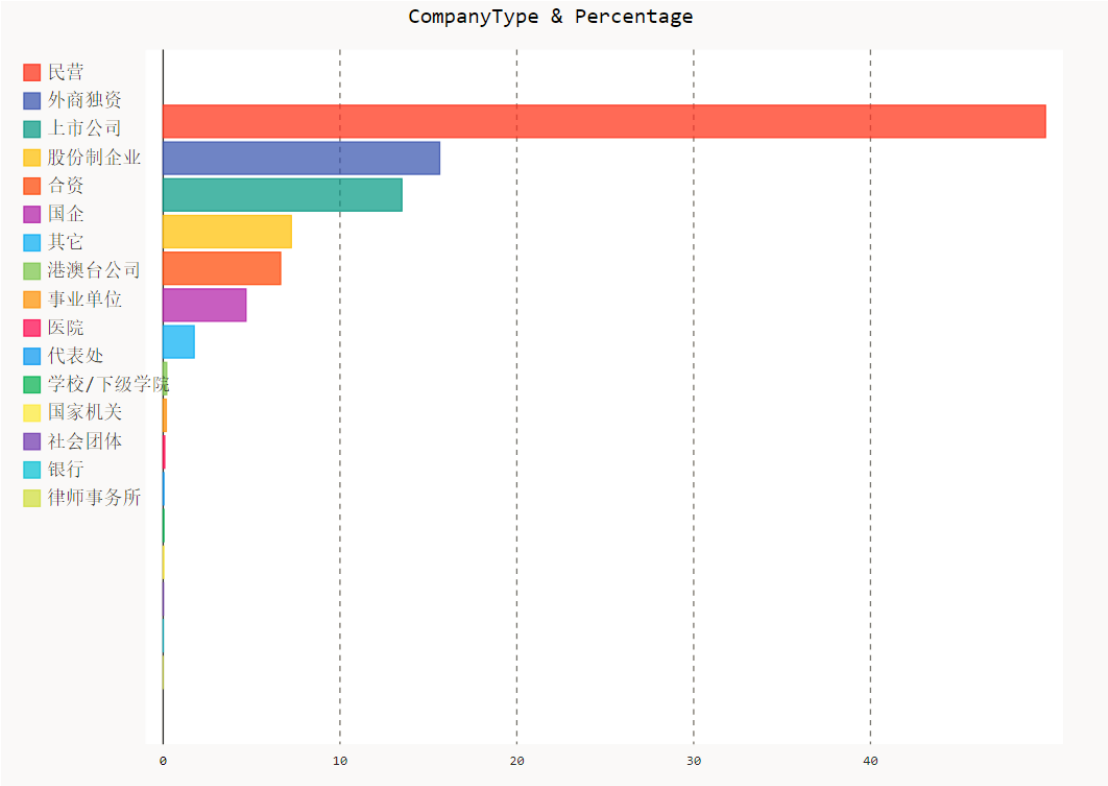


3.4、岗位的招聘企业画像（高伟）

3.4.1 公司行业及其所占百分比

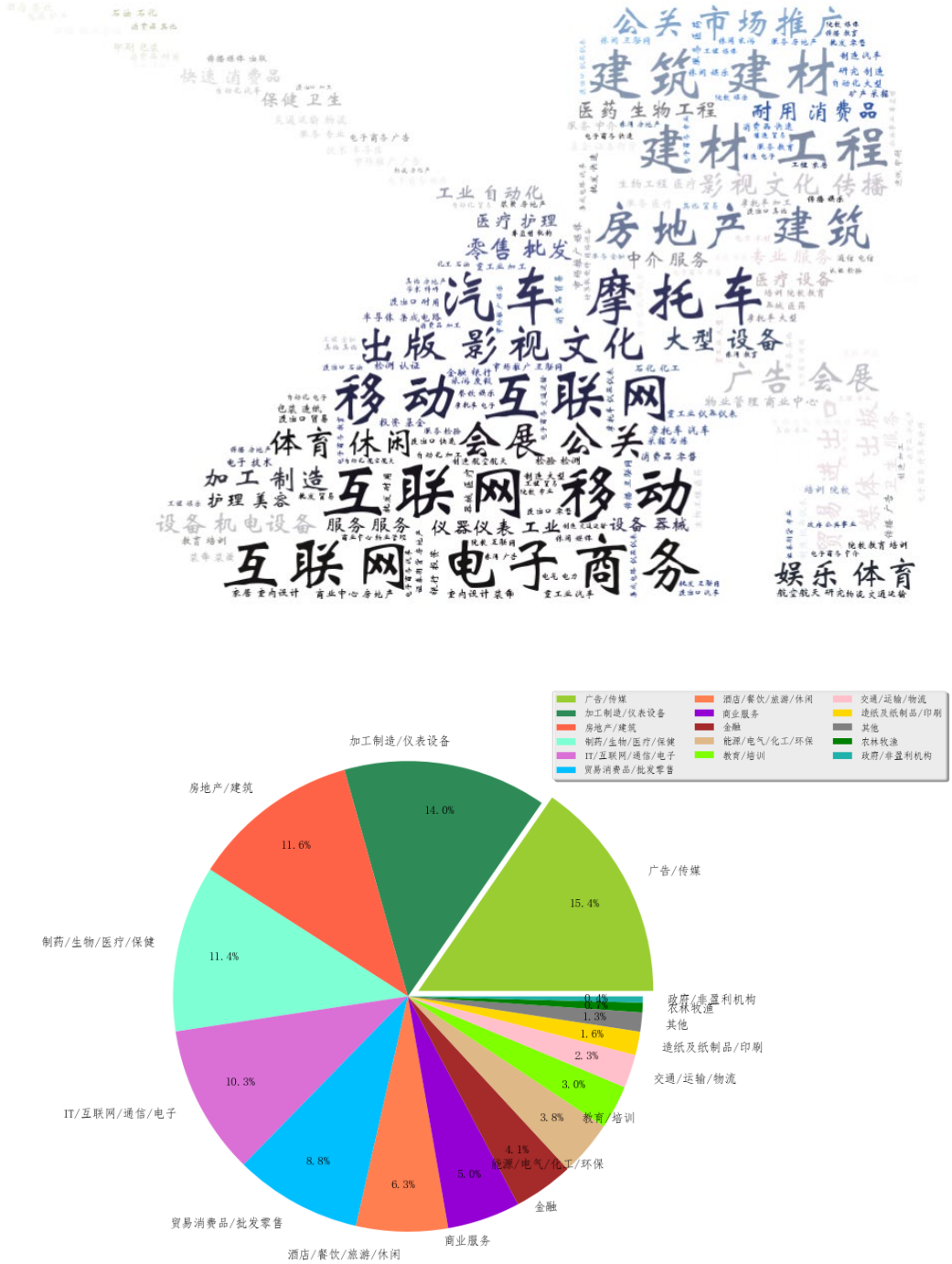
公司行业及其所占百分比-饼状图/柱状图

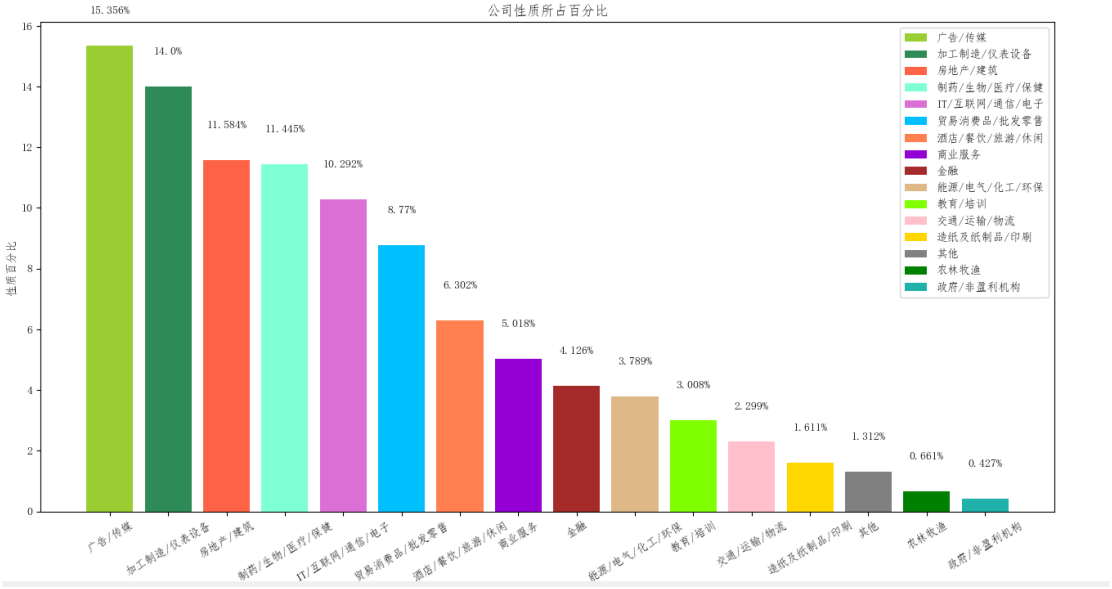




3.4.2 公司类型词频统计显示

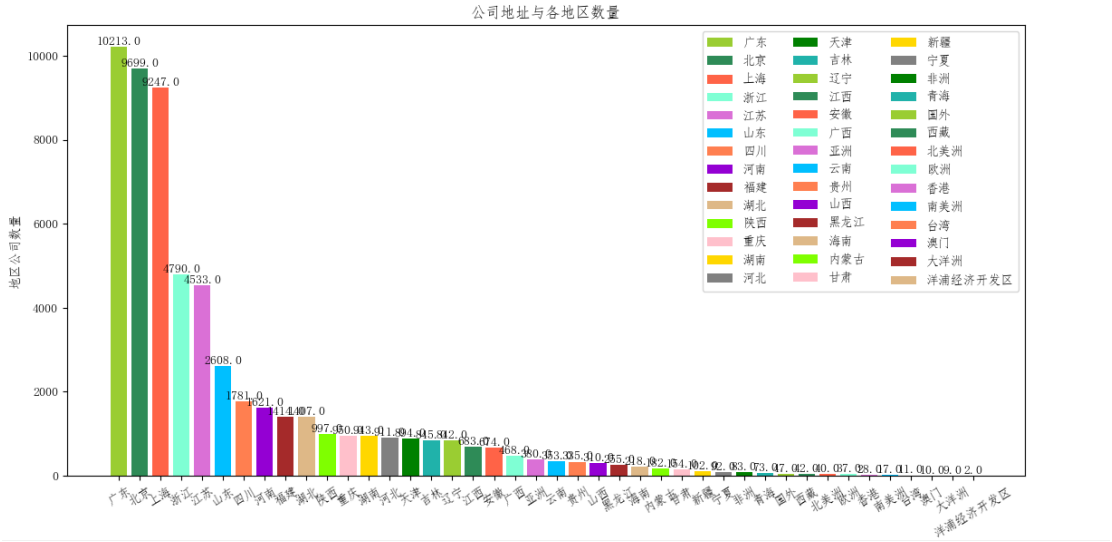
公司类型词频统计显示-词云/饼状图/柱状图





3.4.3 公司地址所占百分比

公司地址所占百分比-柱状图



职位推荐系统概述（高伟 宋建波）

4.1 推荐系统概述

随着信息技术和互联网的发展，人们逐渐从信息匮乏的时代走入了信息过载的时代。为了解决**信息过载**的问题，先后出现了如下的解决方案：

- **分类目录** 代表公司：雅虎、Hao123，但是随着互联网规模不断扩大，分类目录网站只能覆盖少量的热门网站。
- **搜索引擎** 代表公司：Google、百度，用户通过关键词搜索找到自己需要的信息。
- **推荐系统** 由于搜索引擎需要用户提供明确的关键词，但是有时候用户无法找到准确描述自己需求的关键词，这时候就需要推荐系统了。

推荐系统不需要用户提供明确的需求，通过分析用户的历史行为给用户的兴趣进行建模，从而主动给用户推荐能够满足他们兴趣和需求的信息。
所以它存在的两个前提点：

- **信息过载；**
- **用户需求不明确；**

4.2 推荐系统中的用户行为

个性化推荐算法通过对用户行为的深度分析，可以给用户带来更好的使用体验。用户的行为不是随机的，而是蕴含着很多模式。数据挖掘领域最著名的就是“啤酒与尿布”的故事了。从算法设计的角度来看，它说明了用户行为数据中蕴含着很多不是那么显而易见的规律。利用用户行为数据进行分析的算法一般是协同过滤算法。

用户行为一般分两种：

- 显性反馈行为：用户明确表示对物品喜好的行为。主要方式是评分和喜欢/不喜欢。例如 YouTube 早期的 5 分评分系统以及后来改进的两档评分系统
- 隐形反馈行为：不能明确反应用户喜好的行为。最有代表性的就是页面浏览行为

下面列出各领域的网站中这两种行为的例子：

表2-2 各代表网站中显性反馈数据和隐性反馈数据的例子

	显性反馈	隐性反馈
视频网站	用户对视频的评分	用户观看视频的日志、浏览视频页面的日志
电子商务网站	用户对商品的评分	购买日志、浏览日志
门户网站	用户对新闻的评分	阅读新闻的日志
音乐网站	用户对音乐/歌手/专辑的评分	听歌的日志

4.3 推荐系统的评测指标

- **用户满意度**。最重要的评测指标。可以通过购买行为、点击率、用户停留时间、转化率等 指标度量
- **预测准确度**。度量系统预测用户行为的能力。评分预测（预测用户对物品会打多少分）的预测准确度通过 RMSE（均方根误差）和 MAE（平均绝对误差）计算；
  TOPN 推荐（给用户推荐包含 N 个物品的列表）预测准确度通过 precision（准确率）和 recall（召回率）计算。

- **覆盖率**。推荐系统对长尾物品的发掘能力，可简单理解为推荐系统能够推荐出来的物品站总物品集合的比例。
- **多样性**。推荐列表需要能够覆盖用户不同的兴趣领域（比如某用户既喜欢看动漫又喜欢看武侠片）
- **新颖性**。给用户推荐那些他们以前没有听过的物品
- **惊喜度**。推荐结果和用户历史上喜欢的物品不相似，但是用户又觉得满意的推荐
- **信任度**。以电商平台为例，同样的推荐结果，以让用户信任的方式推荐给用户更能让用户产生购买欲，而以广告的形式推荐给用户就可能难以让用户产生购买的意愿。
- **实时性**。由于物品（新闻、微博等）具有很强的时效性，需要将物品在还具有时效性时就将它们推荐给用户。
- **健壮性**。衡量推荐系统抗击作弊的能力，通过模拟攻击来评测。
- **商业目标**。最本质的商业目标是平均一个用户给公司带来的盈利。不同网站商业目标不同，电商网站可能是销售额，基于展示广告盈利的网站可能是广告展示数，基于点击广告盈利的网站可能是广告点击总数。

在评测时，还要考虑到评测维度，目的是知道一个算法在什么情况下性能最好，常用评测维度有如下 3 种:

- **用户维度**。主要包括用户的人口统计信息、活跃度及是不是新用户等
- **物品维度**。包括物品的属性信息、流行度、平均分以及是不是新加入的物品等
- **时间维度**。包括季节、是工作日还是周末、是白天还是晚上等

4.4 职位推荐系统

4.4.1 Mahout 协同过滤

我们从建立一个基于行为的推荐引擎开始，因为我们想要利用我们已有的求职用户和他们的点击数据，我们的首次个性化推荐尝试是基于 Apache Mahout 提供的基于用户间的协同过滤算法的实现。我们将点击流数据喂给一个运行在 Hadoop 集群上的 Mahout 构建器并产生一个用户到推荐职位的映射。我们建立一个新的服务使得可以运行时访问这个模型，且多个客户端应用可以同时访问这个服务来获取推荐的职位。

4.4.2 产品原型的结果和障碍

作为一个产品原型，基于行为的推荐引擎向我们展示了一步步迭代前进的重要性。通过快速建立起这个系统并将其展示给用户，我们发现这种推荐对于求职者来说是有用的。然而，我们很快就遇到了一些在我们的数据流上使用 Mahout 的问题:

模型构建器需要花大约 18 个小时的时间来处理 Indeed 网站 2013 年的点击流数据，这个数据量要比今日的数据小了三倍。

我们只能一天执行一个模型构造器，这意味着每天新加入的用户直到第二天为止看不到任何推荐。

几百万新汇总的职位在模型构造器再次运行之前是不能作为可见的推荐的。

我们产生的模型是一个很大的映射，大约占了 50 吉字节的空间，需要花费数个小时将其通过广域网从其生成的数据中心拷贝到全球各地的数据中心。

Mahout 的实现的提供露了一些可配置参数，比如相似度阈值。我们可以调节算法的参数，但是我们想要测试整个不同的算法这样的灵活性。

4.4.3 为推荐实现最小哈希

我们先解决最重要的问题：构建器太慢了。我们发现在 Mahout 中的用户间相似度是通过在 n^2 复杂度下的用户间两两比较的来实现的。仅对于美国的网站用户来说（五千万的访问量），这个比较的数量将达到 15 * 10^15 次，这是难以接受的。而且这一计算本身也是批量处理的，新加一个用户或者一个新的点击事件就要求重新计算所有的相似度。

我们意识到推荐是一个非精确匹配问题。我们是在寻求方法来发现给定用户最相近的用户们，但是我们并不需要 100%准确。我们找了一些估算相似度的方法，不需要准确的计算。

主要贡献者戴夫格里菲思从一篇谷歌新闻学术论文上看到了最小哈希方法。最小哈希（或者最小独立序列）允许近似计算杰卡德相似度。将这一方法应用到两个用户都点击过的职位上，我们发现两个用户有更多共同的职位点击，那么他们的杰卡德相似度就越高。为所有的用户对计算杰卡德相似度的复杂度是 O(n^2)，而有了最小哈希后，我们可以将复杂度降到 O(n)。

给定一个哈希函数 h1，一个项目集合的最小哈希被定义为将集合中所有项用该哈希函数分别哈希，然后取其中最小的值。由于方差太大，单个哈希函数不足以计算近似杰卡德相似度。我们必须使用一个哈希函数族来合理地计算近似的杰卡德距离。通过使用一个哈希函数族，最小哈希可被用来实现可调节杰卡德相似度阈值的个性化推荐。

4.4.4 为新用户推荐职位

最小哈希的数学属性使得我们可以解决为新用户推荐职位的问题，并且可以向所有的用户推荐新的职位。当新的点击数据到来的时候，我们增量地更新最小哈希签名。我们也在内存中维护新职位和他们的最小哈希族的映射。通过在内存中保留这两部分数据，我们就可以在新用户点击了一些职位后为他们推荐职位。只要任何新发布的职位被点击访问过，它就可以被推荐给用户。

在转移到最小哈希方法后，我们有了一个混合的推荐模型，由一个在 Hadoop 上的构建的离线每日被更新的组件和一个由当日的点击数据组成、在内存缓存中实现的在线组件。这两个模型被整合起来用以计算每一个用户的最终推荐集合。在我们做了这些改变之后，每一个用户的推荐变得更加得动态，因为它们将随着用户点击感兴趣的职位的同时发生变化。

这些改变中我们认识到，我们可以在性能和准确度上做出权衡，用小的误差增加来换大的性能提升。我们也认识到可以将较慢的离线模型通过在线的模型进行补充，从而得到更新的推荐结果。

推荐系统前端页面展示

搜索框:

展示页面: