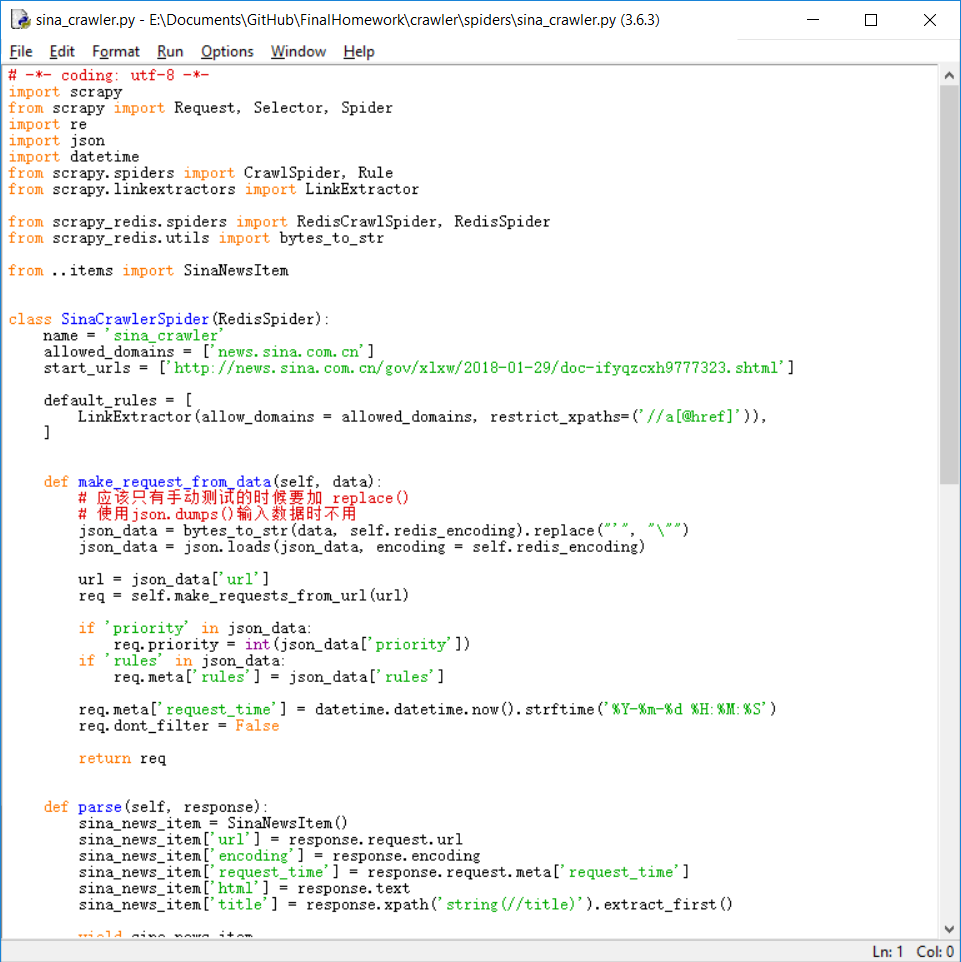
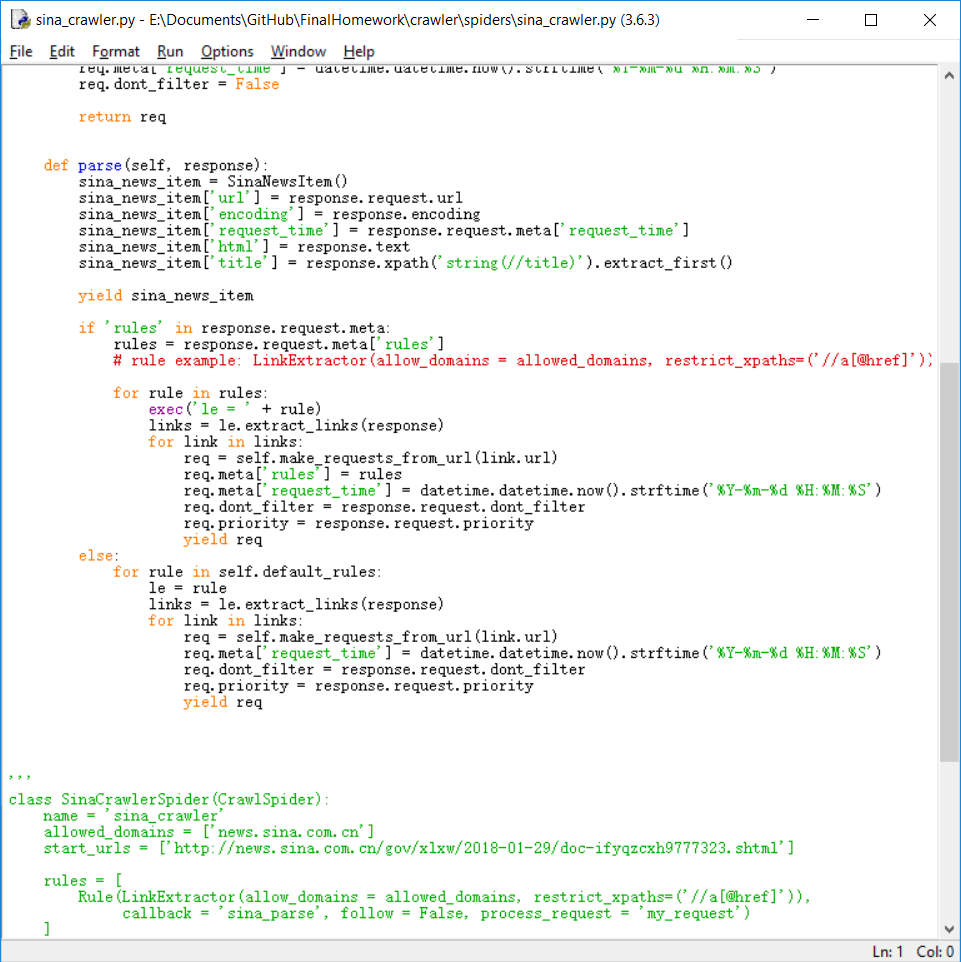
上周我们组已经完成了所有分布式框架部分，这周我们实现了自定义的分布式爬虫框架，并把爬虫与后端数据库相连，运行成功。

目前整个分布式爬虫框架挂在学院内网中，进行百度百科内容的爬取，并且已经爬取了超过800万个界面。

我们的目标是对数据库插入一条初始爬取数据（url，爬取规则，优先级），然后爬虫从数据库中读取到这个种子，然后进行爬取。并且爬虫需要有去重与错误重试功能，另外对于相同的url，如果相隔时间过长，将会重新爬取，并且不会覆盖掉原先的数据。

首先我们使用sina新闻作为测试爬取对象。





在这里我们首先使用RedisSpider，这个类从当前文件中读取start\_urls，是用来测试的。最终版我们使用RedisCrawlSpider，这个类从数据库读取数据进行爬取。

另外经过研究scrapy-redis的源代码，我们发现它并不符合我们的要求。我们需要输入种子url，爬取规则，与优先级，然而scrapy-redis只保存了url的hash值来去重。所以我们继承了scrapy-redis中的各种类，自己实现了功能。

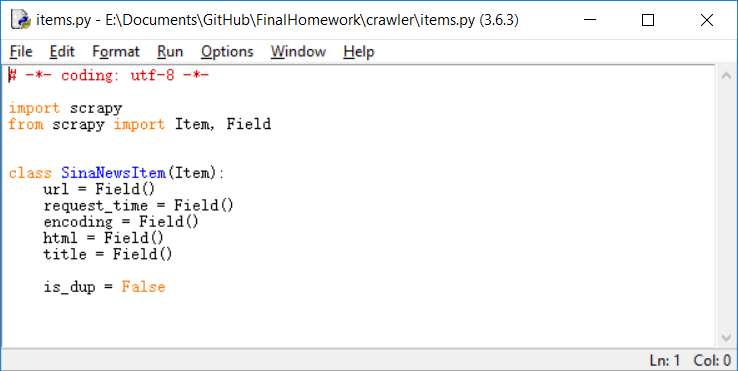
通过研究，我们认为应该使用json格式保存在redis数据库中对爬取进行控制。

如下：

'{"url":<https://baike.baidu.com/item/%E7%99%BE%E5%BA%A6%E7%99%BE%E7%A7%91/85895>,”priority”:2,”rules”:[“lxp1”,”lxp2”]}'

Json字典中保存一个string类型url,一个int类型priority与一个string数组类型rules，其中priority与rules可以缺省，当缺省时使用默认值替代。

为了满足这一需求，我们定于item为

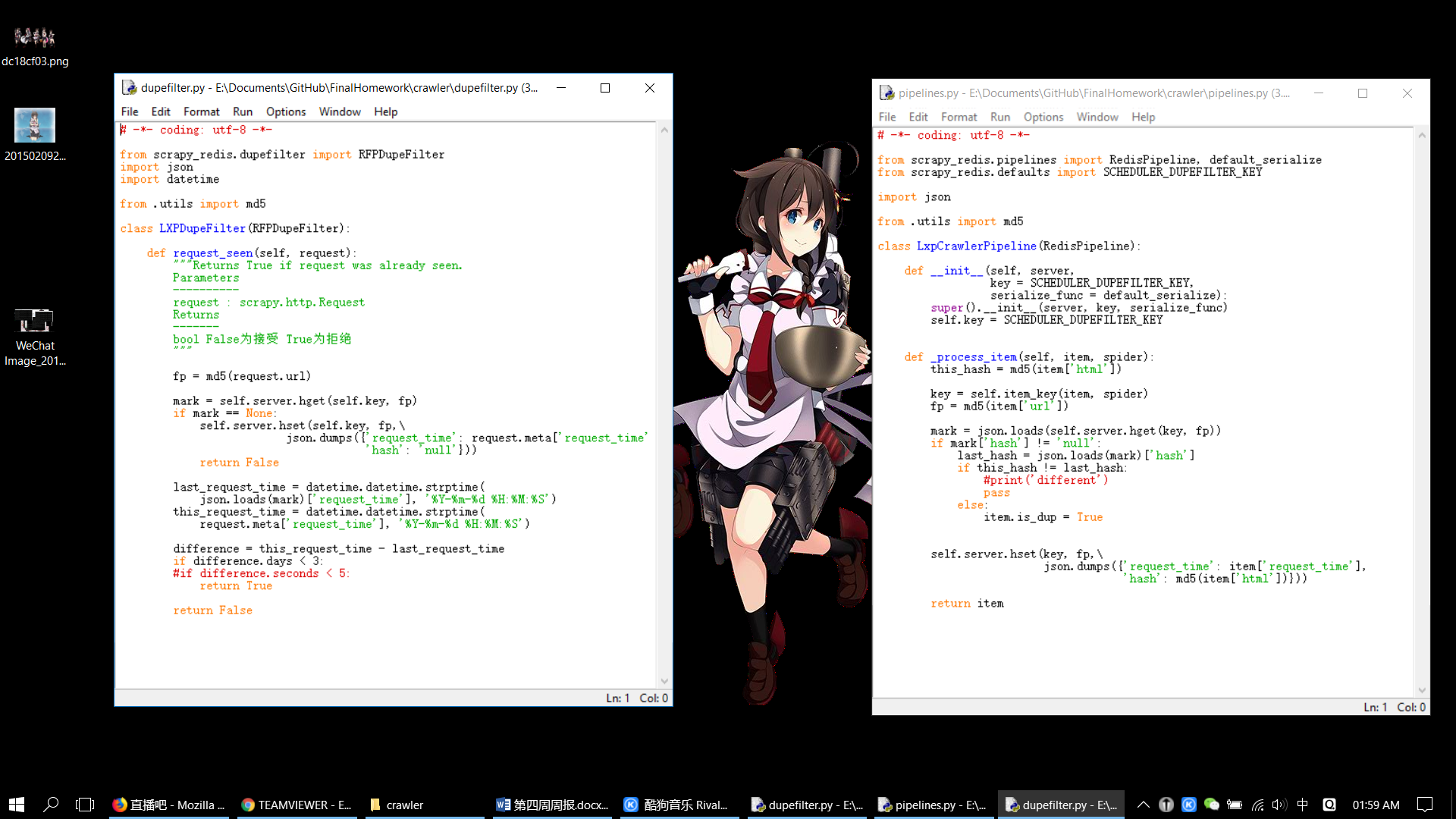


这其中request\_time用于记录请求时间，用于对页面请求时间进行记录。如果有间隔过长的同一界面的请求，将会重新爬取。

Encoding用于记录编码，以便最后从数据库中提取解析。

于是当爬虫开始运行的时候，每次解析完一个界面，就会抛出一个item数据包，并且会根据规则提取出下一轮迭代的url，并附加上rules与request\_time数据，重新发出请求。

接下来要介绍本爬虫框架的核心逻辑。



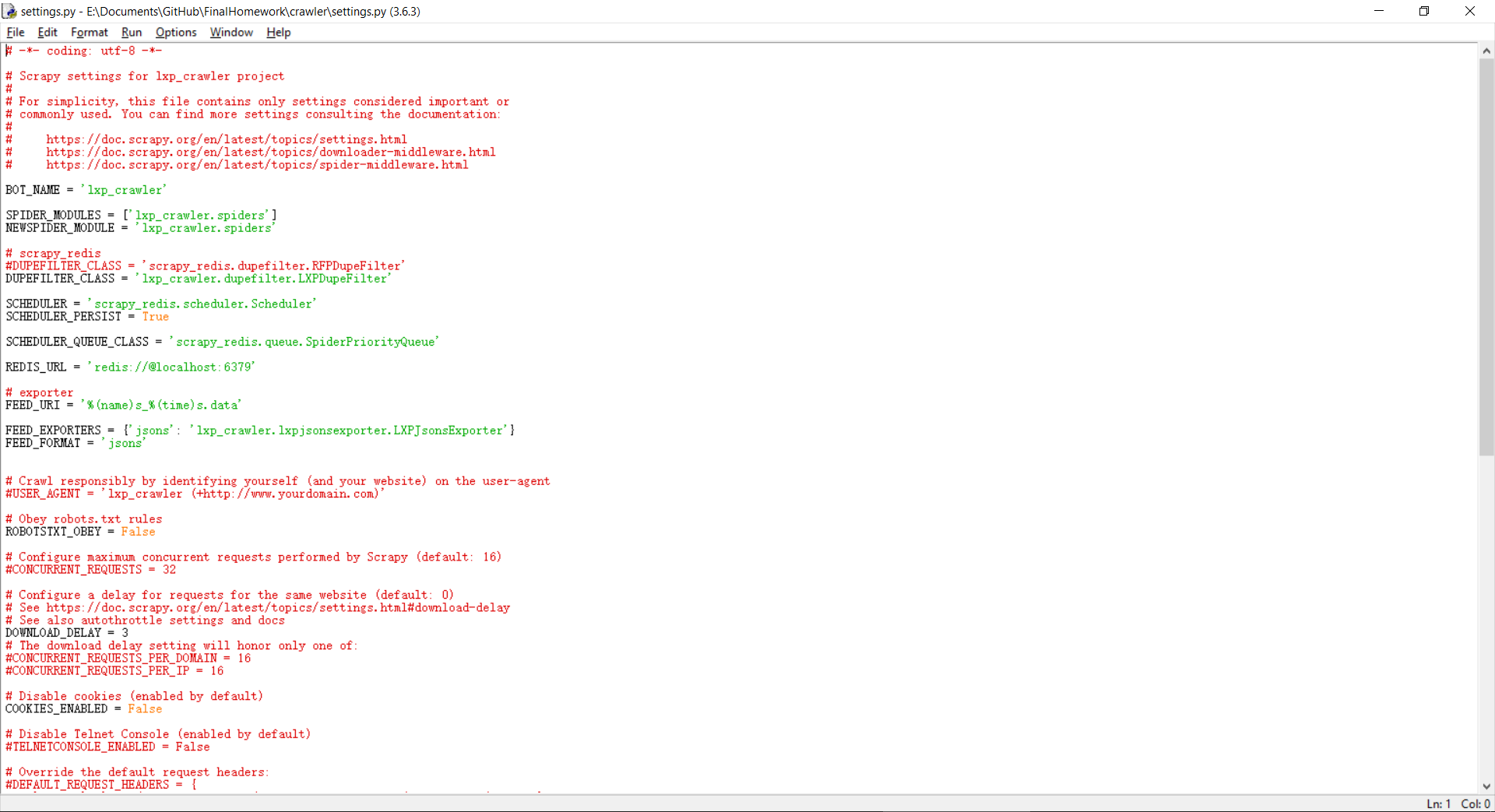
这是dupefilter与pipelines的代码。

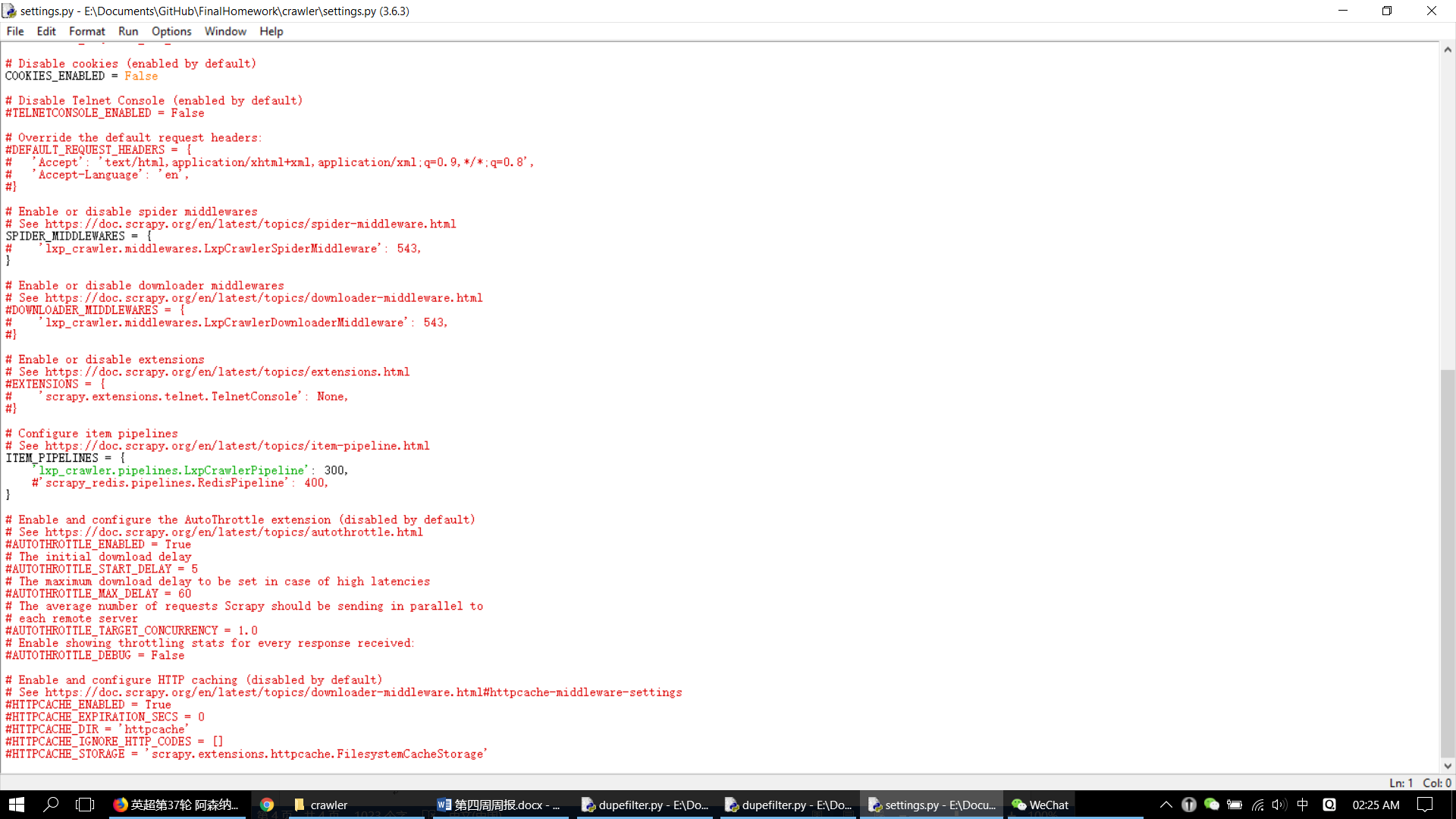
对于dupefilter.py，我们对每个url生成一个唯一的指针（内部实现为hash值），然后我们试图从控制爬取的数据库中查询这个url，这个数据库结构包括请求链接url，请求时间request\_time与界面内容的hash值hash。这个数据库被设计为广度优先搜索。

如果查询url失败，表示这个url从未被请求过，于是对控制数据库表进行insert操作。之后获取上次request\_time，与本次request\_time比较，如果距离上次请求时间过长（表示为有可能界面更新），则重新请求。

而当请求结束，item数据包被抛出后，使用pipelines进行处理。首先获得界面html的hash值与url的一个唯一的key。然后我们从通过这个url从控制数据库中获取相关控制信息。如果hash不为null，表示之前已经对这个url进行爬取，所要做的就是比对两个界面是否有不同，于是通过对比hash，如果不同，说明界面更新，我们保存新的界面hash，替换掉原有的hash，而具体html则被抛出进入flume并保存到数据库。然后我们再用新的控制数据update控制数据库。

最后我们在 settings.py里面配置了我们使用的组件

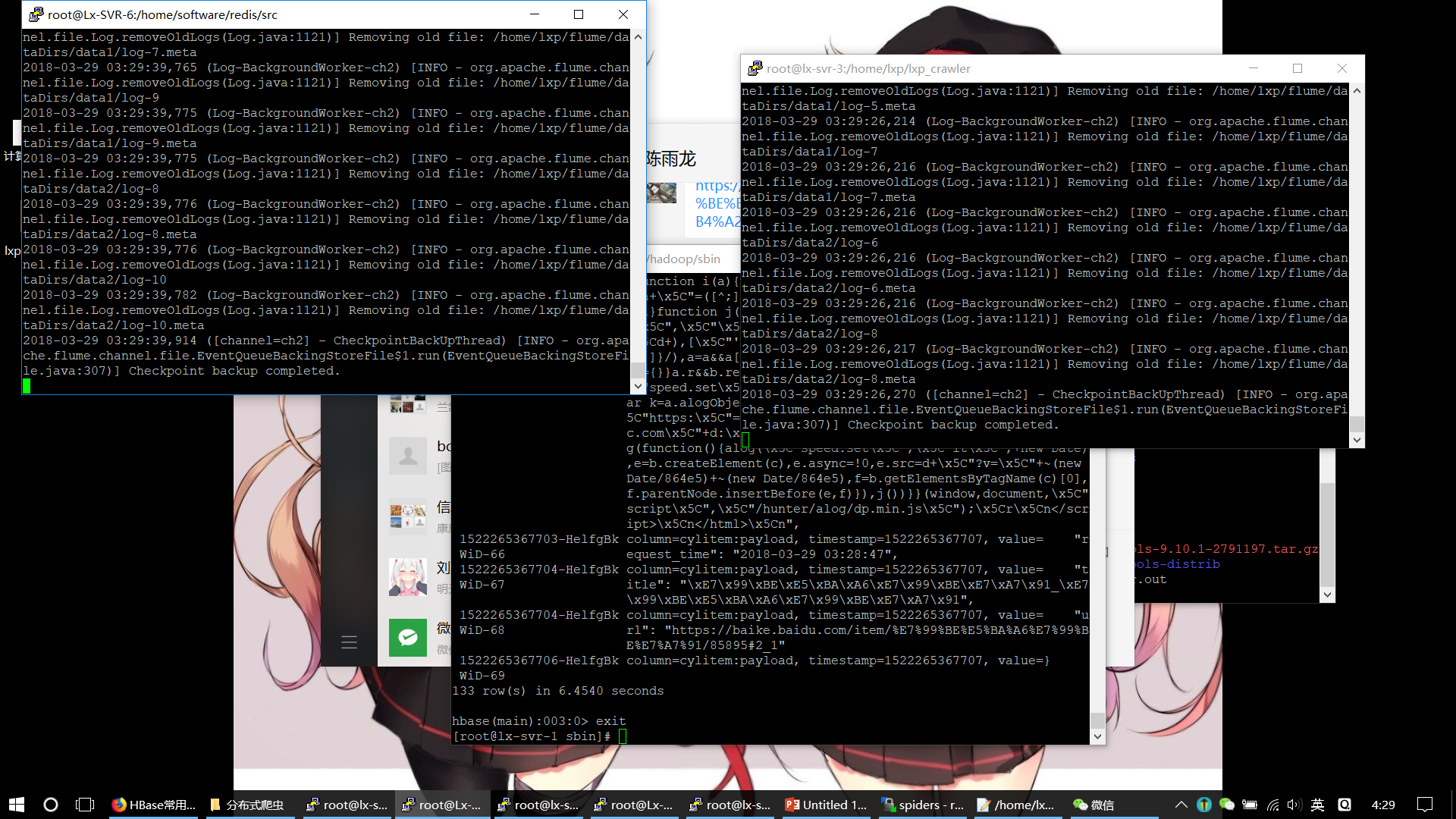




这样我们就完成了分布式爬虫框架的实现。

之后我们把这个sina爬虫改成了baidu爬虫，并放到学院服务器进行爬取。遗憾的是，在寝室无法连接学院内网，无论是堡垒机还是teamviewer都无法连接（**学校网更新后就不行了，不是我在甩锅（笑）**）。

这里有一张整个分布式框架运行起来，爬取baidu并把数据最终放入hbase数据库中的图片



我们将于周二之前从学院服务器中将全部代码取出，并更新周报。