

**基于Scrapy的模板化爬虫程序管理平台**

**实现方案**

v1.0

赵正阳 SY1906428

郭浩隆 SY1906430

沈一聪 SY1906510

梁远志 SY1906503

宋冰晨 SY1906429

2020年5月7日

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 日期 | 修改人 | 修改章节 | 备注 |
| 1.0 | 2020/5/7 | 赵正阳 | 1、3.1 | 完成概述，绘制系统组件图和ER图，整合初稿 |
| 宋冰晨 | 2.1、2.2.2~2.2.4、2.3 | 完成Web UI的模板浏览、任务操作和数据下载模块的流程图 |
| 郭浩隆 | 2.2.1、2.2.5 | 完成Web UI的登录注册和模板管理模块的流程图 |
| 梁远志 | 3.2、4 | 绘制系统状态图，完成系统调度器部分 |
| 沈一聪 | 5 | 完成爬虫模板部分 |

**目录**

[1. 概述 1](#_Toc39785521)

[2. Web UI 1](#_Toc39785522)

[2.1 前端框架AdminLTE 2](#_Toc39785523)

[2.2 流程图 2](#_Toc39785524)

[2.2.1 登录注册模块 2](#_Toc39785525)

[2.2.2 模板浏览模块 3](#_Toc39785526)

[2.2.3 任务操作模块 4](#_Toc39785527)

[2.2.4 数据下载模块 5](#_Toc39785528)

[2.2.5 模板管理模块 6](#_Toc39785529)

[2.3 前后端交互 7](#_Toc39785530)

[3. 数据库设计 9](#_Toc39785531)

[3.1 ER图 9](#_Toc39785532)

[3.2 系统状态图 12](#_Toc39785533)

[4. 系统的调度器设计 14](#_Toc39785534)

[5. 爬虫模板 16](#_Toc39785535)

[5.1 开发流程 16](#_Toc39785536)

[5.2 运行环境 17](#_Toc39785537)

[5.3 模板组成 17](#_Toc39785538)

[5.3.1 目录结构 17](#_Toc39785539)

[5.3.2 项目(item) 18](#_Toc39785540)

[5.3.3 用户代理中间件(useragent middleware) 19](#_Toc39785541)

[5.3.4 下载中间件(download middleware) 19](#_Toc39785542)

[5.3.5 项目管道(item pipeline) 21](#_Toc39785543)

[5.3.6 配置文件(config) 21](#_Toc39785544)

[5.3.7 传入参数 22](#_Toc39785545)

[5.3.8 利用Scrapyd打包项目 22](#_Toc39785546)

[5.3.9 反爬虫应对 22](#_Toc39785547)

1. **概述**

本系统主要由Web UI、调度器和爬虫模板三部分组成。用户通过前端界面进行创建爬虫任务等操作，网站后端将爬虫任务划分为多个爬虫作业，由调度器程序将这些爬虫作业分配到Scrapyd服务器节点上，由Scrapyd服务器执行实际的采集工作。网站后端使用的数据库为MySQL，Scrapy爬虫程序保存采集结果的数据库为MongoDB。

系统的整体组件图如图1.1所示



图1.1整体组件图

下面分别描述每个部分的具体实现方案。

1. **Web UI**

系统网站采用Django作为Web应用框架。该框架的软件设计模式为MVT，即模型（Model）、视图（View）和模板（Template）。图2.1显示了Django的MVT架构。



图2.1 Django的MVT架构

第一步，用户通过浏览器请求资源。Django将请求的资源URL与urls.py文件中的现有URL路径进行匹配。将用户请求的URL与urls.py中的URL进行匹配的过程称为URL映射。一旦URL映射成功，Django就会跳转到views.py文件并调用一个视图。被调用的视图查找需要交互的模型和模板，然后将响应返回给浏览器。在这一步，模型处理与用户请求相关联的数据；而模板处理的是渲染视图所需的HTML和其他静态文件，例如CSS文件、JS文件、图像等。

2.1 前端框架AdminLTE

系统的目标用户是不了解网络爬虫技术希望简易采集数据的用户，因此在可视化用户界面的设计上，采用常见的仪表盘前端模板。AdminLTE是一个完全响应式的后台管理模板。其基于Bootstrap 4.4框架以及JS / jQuery插件，高度可定制且易于使用，适合从小型移动设备到大型台式机的多种屏幕分辨率。

2.2 流程图

本系统的参与者一共分为两类：普通用户和管理员。对于普通用户，系统提供的功能包括登录注册、模板浏览、任务操作、数据下载四个模块；对于管理员，系统提供的功能包括登录注册和模板管理两个模块。

2.2.1 登录注册模块

用户登录及注册账号模块流程图如下图2.2所示。

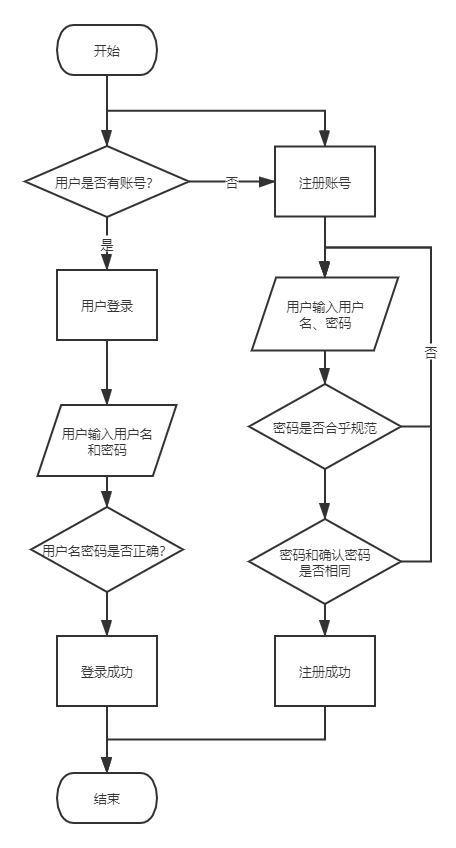


图2.2 登录注册模块流程图

2.2.2 模板浏览模块

模板浏览模块的流程图如图2.3所示。



图2.3 模板浏览模块流程图

2.2.3 任务操作模块

任务操作模块的流程图如图2.4所示。



图2.4 任务操作模块流程图

2.2.4 数据下载模块

数据下载模块的流程图如图2.5所示。



图2.5 数据下载模块流程图

2.2.5 模板管理模块

模板管理模块的流程图如图2.6所示。

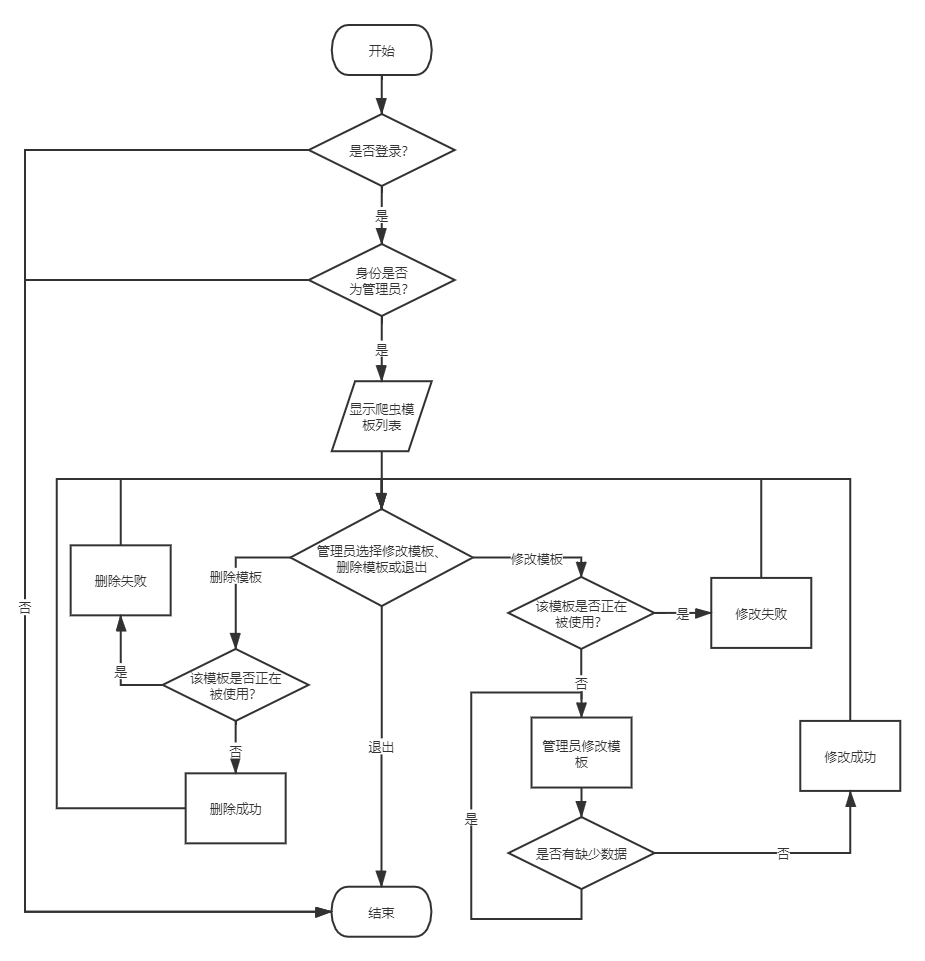


图2.6 模板管理模块流程图

2.3 前后端交互

Django的核心是基于请求和响应的简单概念，即浏览器发出请求，Django调用一个视图，该视图将响应发送回浏览器，如图2.5所示。模板浏览和数据下载模块使用的是这种前后端交互模式。



图2.5 Django自带的前后端交互模式

在任务操作模块中，用户可以手动刷新当前界面，以更新任务列表，显示最新的任务进度情况和任务状态。同时后端可以主动推送消息给用户，以异步的方式动态更新用户的任务列表。为了实现后端主动推送消息，系统采用Python第三方库Channels来扩展Django。Channels是官方的Django项目，其以Django的核心为基础，并在其下分层了一个完全异步的层，以同步模式运行Django本身，但异步处理了连接和套接字，并对应地提供了两种编码方式。如图2.6所示。



图2.6 Channels支持的前后端交互模式

1. **数据库设计**

3.1 ER图

本系统包含用户、网站（模板集合）、网站类型、爬虫模板、采集字段、模板参数、爬虫任务、爬虫作业、Scrapyd服务器节点9个实体，ER图如图3.1所示。



图3.1 ER图

每一个实体的具体作用如下所述。

1. User：表示系统的用户，使用is\_superuser字段区分普通用户和管理员。由于系统中使用Django提供的认证机制，因此该实体类即为Django认证机制中的User模型。
2. SiteType：表示网站类型（例如“本地生活”、“媒体阅读”等），用于在首页按类型筛选模板集合。name字段为英文名称，在代码逻辑中使用；display\_name字段为中文名称，用于在前端界面展示给用户。
3. Site：表示同网站的模板集合（例如“豆瓣”，以下简称为“网站”或“模板集合”），对应一个Scrapy项目，每个网站包含多个爬虫模板。site\_type\_id字段为关联SiteType实体的外键；name字段为英文名称，在代码逻辑中使用，同时对应Scrapy项目名；display\_name字段为中文名称，用于在前端界面展示给用户；egg字段为Scrapy项目打包egg文件的存放路径；settings字段为传递给Scrapy爬虫的其他选项；update\_time字段为模板集合的更新时间，当模板集合包含的任何子模板更新时更新该字段。在前端界面中，每个网站需要显示一个logo，对应的图片URL通过网站英文名拼接得到。
4. Template：表示爬虫模板（例如“豆瓣热门电影”），对应一个具体的Scrapy Spider。site\_id字段为关联Site实体的外键；name字段为英文名称，在代码逻辑中使用，同时对应Scrapy Spider名称；display\_name字段为中文名称，用于在前端界面展示给用户；update\_time字段为模板的更新时间，当管理员修改现有模板时将更新该字段。用户选择模板时将展示模板的简介、采集字段、模板参数和样例数据，其中简介和样例数据分别由introduction和sample\_data字段表示（均为HTML格式），采集字段和模板参数由通过外键关联的Field和Param实体表示。在每个模板的所有参数中，需要指定一个数字类型的参数用来将用户创建的任务划分为爬虫作业（例如“翻页次数”），该参数的英文名由split\_param字段指定。在前端界面中，每个网站需要显示一个logo，对应的图片URL通过网站英文名+模板英文名拼接得到。
5. Field：表示模板的采集字段。template\_id字段为关联Template实体的外键；name和display\_name字段分别为采集字段的英文和中文名称。用户选择模板时每个采集字段将展示一张预览图片，对应的URL通过网站英文名+模板英文名+字段英文名拼接得到。
6. Param：表示模板参数。template\_id字段为关联Template实体的外键；name和display\_name字段分别为模板参数的英文和中文名称；input\_label字段和input\_type字段分别表示用户填写模板参数的表单中该参数使用何种HTML标签（<input>或<textarea>）以及<input>标签允许输入的类型（text, number等）。用户选择模板时每个模板参数将展示一张预览图片，对应的URL通过网站英文名+模板英文名+参数英文名拼接得到。
7. Task：表示用户创建的爬虫任务，相当于一个爬虫模板的“实例”。user\_id字段为关联User实体的外键；template\_id字段为关联Template实体的外键；name字段为用户创建任务时填写的任务名称，长度应在3~20个字符，并且每个用户创建的任务不能有相同的名称；create\_time和finish\_time字段分别表示任务的创建时间和完成时间；args为JSON格式的模板参数，键为该任务所属模板的参数英文名，值为用户填写的具体参数值；status表示任务状态，有“等待运行”、“正在运行”、“已暂停”、“已完成”和“已终止”六种状态，状态之间的转移关系在3.2节中描述。
8. Node：表示Scrapyd服务器节点。其中id, port, username和password字段由调度器（3.3节）访问服务器上的Scrapyd服务使用；status字段表示节点状态，有ONLINE, OFFLINE和DISABLED三种状态，状态之间的转移关系在3.2节中描述。
9. Job：表示可由Scrapyd调度的爬虫作业。用户创建爬虫任务后，系统将根据其所属模板的划分参数（split\_param字段）将任务划分为多个爬虫作业，由调度器将作业分配到节点，由Scrapyd服务器来执行实际的采集工作。task\_id字段为关联Task实体的外键；node\_id字段为关联Node实体的外键；status字段表示作业状态，有created, pending, running和finished四种状态，状态之间的转移关系在3.2节中描述；args字段表示要传递给Scrapy Spider的参数，由其所属的任务的args字段得到，但其中的划分参数需要做相应调整。例如，模板有一个“翻页次数”参数，用户创建任务时填写的值为10，则划分作业时该参数分别置为1~10，其他参数与任务一致。

3.2 系统状态图

本系统的节点实体的状态转移如图3.2所示。



图3.2 节点实体的UML状态图

与节点（node）实体状态转移相关的操作者包括管理员和调度器。管理员在管理面板中填写信息添加节点后，节点处于禁用（DISABLED）状态；管理员可以在管理面板上启用该节点使其进入在线（ONLINE）状态；如需进行节点维护，管理员可以在管理面板上禁用该节点，使其进入禁用状态；管理员可以在面板上删除处于任意状态的节点；调度器会向所有处于在线状态的节点轮询状态和派发任务，当调度器调用节点接口发生错误时，则意味着发生潜在的节点失效，调度器将节点置为离线（OFFLINE）状态，并通知管理员进行排查和修理工作；管理员在检查修复后，在管理面板上重新启用这一节点，节点回到在线状态。

本系统的作业实体的状态转移如图3.3所示。



图3.3 作业实体的UML状态图

与作业（job）实体状态转移相关的操作者包括用户、调度器和节点。在收到用户请求创建任务后，系统后端切分这一任务，并创建若干个作业，此时作业进入已创建（CREATED）状态；此时调度器会轮询所属任务非暂停和终止状态的已创建作业，并将该作业分配调度到负载最低（等待运行作业数量最少）的节点，作业进入等待运行（PENDING）状态；调度器轮询节点，节点报告作业正在运行时，作业进入运行中（RUNNING）状态；调度器轮询节点，节点报告作业完成运行时，节点进入完成运行（FINISHED）状态；在用户暂停包含该作业的任务时，调度器轮询任务状态，将处于暂停状态的任务包含的处于等待运行状态的作业转移到已创建的状态，从而阻止作业被运行；当发生运行节点失效的情况时，因为无法取得节点返回的作业运行状态信息，失效的节点上处于等待运行和运行中状态的作业会回到已创建的状态等待重新被调度运行。

本系统的任务实体的状态转移如图3.4所示。



图3.4 任务实体的UML状态图

与任务实体状态转移相关的操作者包括调度器和用户；用户在系统界面上创建一个任务后，任务进入就绪（READY）状态，此时系统后端创建了多个作业；当调度器更新作业状态时，检测到一作业处于运行中状态且该任务处于就绪状态时，会将改任务置入运行中（RUNNING）状态；若调度器更新任务状态时检测到任务包含的所有作业都已完成时，会将处于运行状态的任务转移到完成运行（FINISHED）状态；用户可以在除完成运行状态外的状态在面板上终止任务使其进入已取消（CANCELED）状态；用户可以在面板上暂停处于就绪和运行中状态的任务，使其进入已暂停（PAUSED）状态；用户随后可以在面板上继续处于已暂停状态的作业。

1. **系统的调度器设计**

调度器设计上是一个独立与Django网站后端的程序，调度器在系统整体中处于系统后端和Scrapyd之间，以一个守护进程（daemon）的方式负责作业在多个节点上的分配调度，同时将作业状态信息存储到系统的数据库中。调度器的组件图如图4.1所示。



图4.1 调度器顺序图

一方面，调度器依赖于系统后端提供的创建的作业信息，因此Django网站后端对调度器提供了一组专用的数据接口，用于获取和更新作业节点信息。另一方面，调度器会主动调用Scrapyd的接口，从而获取节点上运行作业的状态信息。

调度器在启动后，会分别轮询三类信息：

（1）节点的运行状态信息：用于确认节点工作正常；

（2）已创建状态的作业信息：调度器会将这类作业优化地分配到一个节点上运行；

（3）等待运行和运行中状态的作业信息：调度器会向这些作业分配到的节点查询并更新作业的运行状态。

调度器的顺序图如图4.2所示。



图4.2 调度器顺序图

1. **爬虫模板**

5.1 开发流程

Scrapy爬虫脚本是EasySpider项目的重要组成部分，是实际完成目标网站关键数据爬取任务的相关业务代码，通过Scrapyd来完成部署。

如图5.1所示，Scrapy爬虫脚本的开发流程如下：

1. 首先由小组成员在本地编写目标网站的爬虫代码，在完成所有代码的编写之后，进行本地爬虫任务的测试并将网站爬取结果存入本地的服务器；
2. 通过本地测试之后，将本地的Scrapy项目通过Scrapyd-client进行封装打包，生成.egg文件并将其上传到代码仓库中；
3. 之后由负责easyspider-scheduler的同学获取.egg文件，根据easyspider-site-xxx仓库的环境配置要求，将.egg部署到工作服务器进行线上测试，测试数据存入项目实际的数据库中；
4. 线上测试通过之后，负责Scrapy爬虫脚本的小组成员再将爬虫使用的相关实例和说明（爬取数据示例图片、使用说明实例html、爬取结果html等）存入example目录，以供负责easyspider-web开发的小组成员在网页和后端中完成相关模板的添加；
5. 最后，easyspider-web中的相关开发工作完成之后，进行模拟实际的用户行为的全局的爬虫任务测试，该测试的成功完成标志着一次爬虫脚本开发流程的完成。

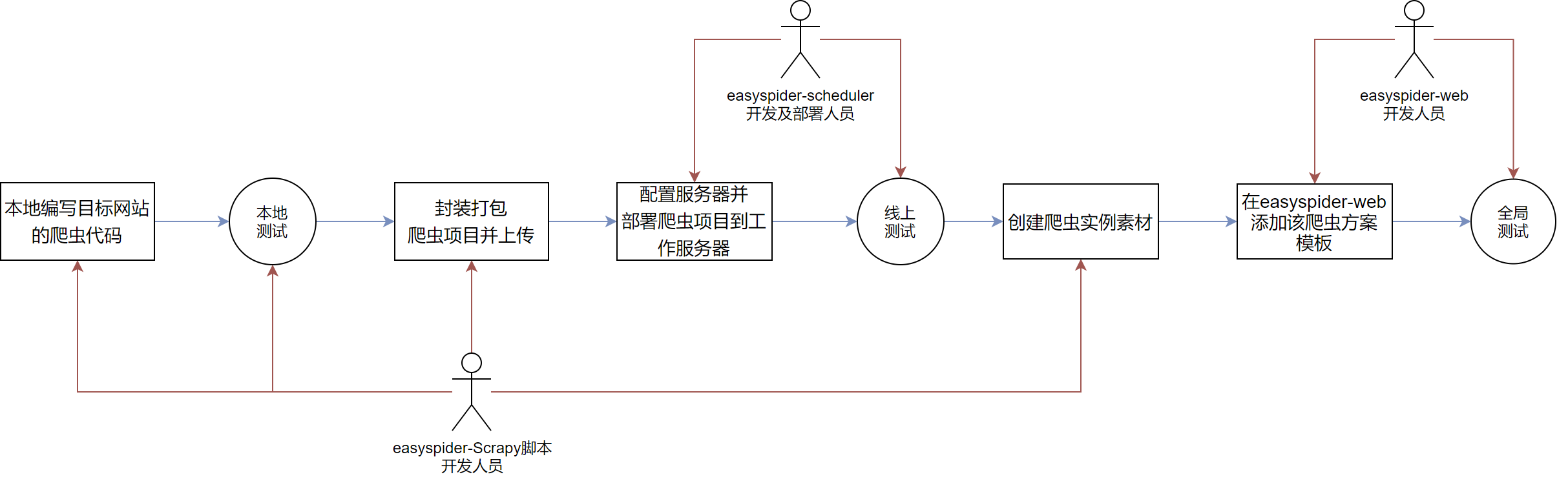


图5.1 Scrapy爬虫脚本开发流程

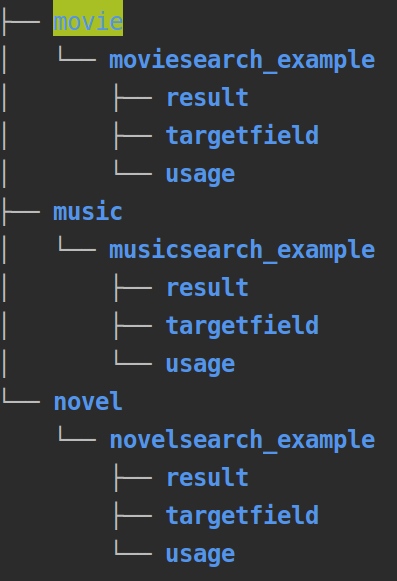
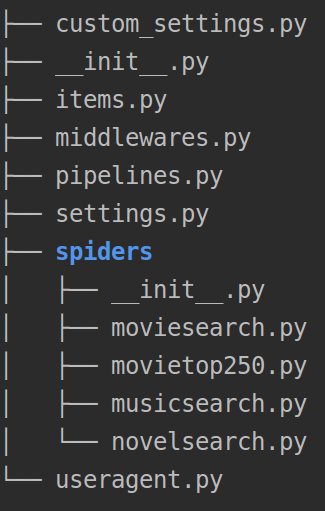
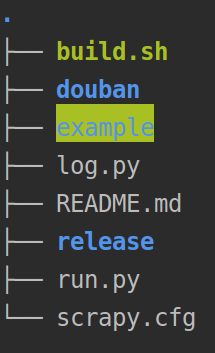
5.2 运行环境

* Python 3.7.0 (anaconda3)
* Scrapy 2.0.1
* selenium 3.141.0
* scrapyd-client 1.1.0
* pydispatcher 2.0.5
* pymongo 3.10.1
* Google Chrome 80.0.3987.122 & chromediver(same version)

5.3 模板组成

接下来以爬取豆瓣电影信息的脚本为例，介绍EasySpider中爬虫模板的通用设计及实现（spider的具体爬取逻辑因网页而异，此处略过）。

5.3.1 目录结构



左：爬虫模板目录 中：爬虫项目目录 右：爬虫示例素材目录

图5.2 爬虫模板的目录结构

图5.2展示了爬虫模板的目录结构。

爬虫模板目录下build.sh生成部署爬虫项目.egg以及一些配置信息到release目录中；douban目录中放置了爬虫项目的业务代码；example目录下保存着爬虫示例素材以满足前端开发的需要；log.py中实现本地log的管理；run.py通过将传入参数硬编码，方便了爬虫程序的调试。

爬虫项目目录下，custom\_setting.py保存了爬虫项目下每一个spider的独立配置参数；\_\_init\_\_.py中包含一些项目初始化时的操作；items.py中保存了爬取结果的项目数据结构；middlewares.py中保存了自定义的一些中间件，包括下载中间件、用户代理中间件等；pipelines.py保存了将爬取结果的项目数据结构存入mongodb的管道；settings.py中保存了项目的全局设置；spiders目录下保存了统一目标网站（爬虫项目）下针对不同爬取目标的爬虫；useragent.py保存了预设的一些用户代理备选项。

爬虫示例素材目录下，保存了各个爬虫的使用参数要求，爬取的目标字段以及爬取结果的实例信息，主要是jpg或者html的格式的文件。

5.3.2 项目(item)

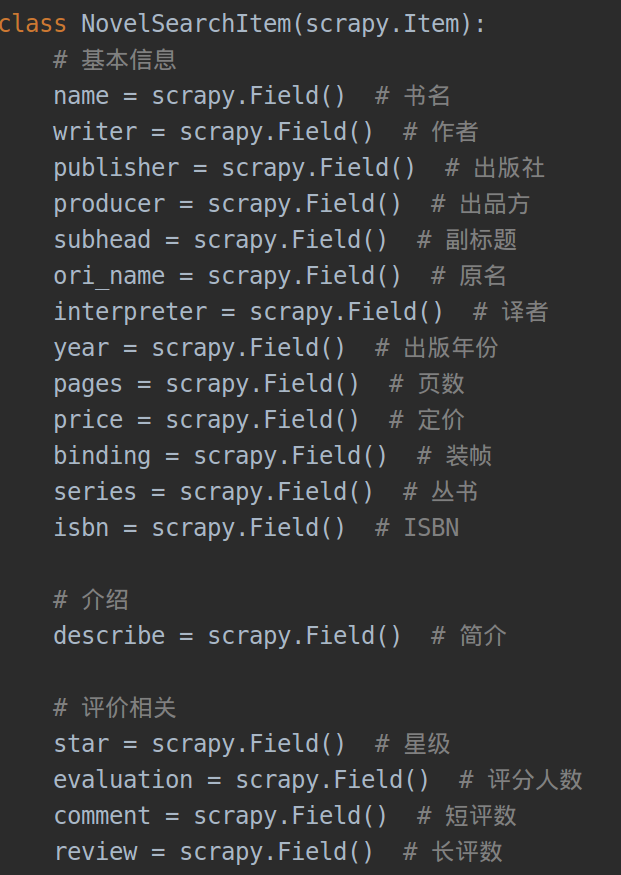
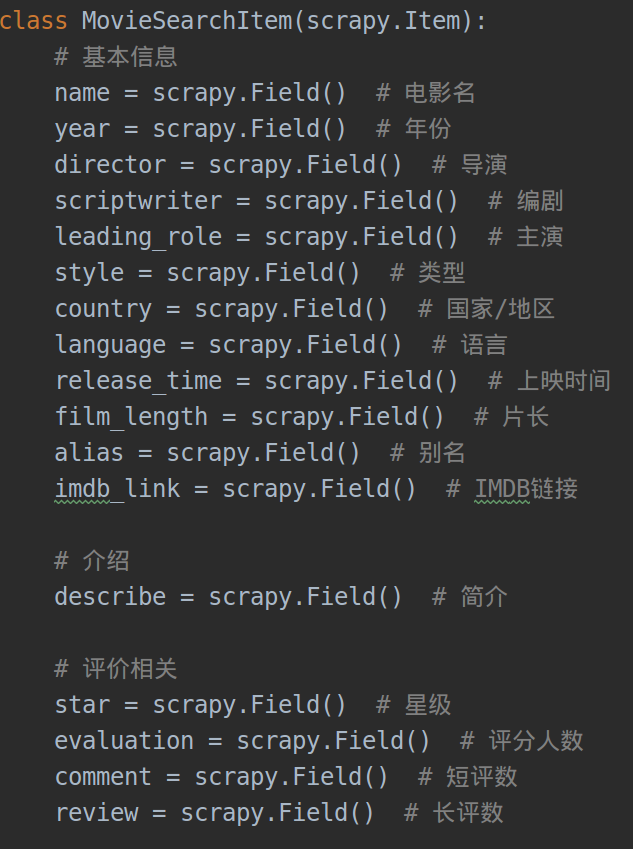


图5.3 项目（items）设计

根据豆瓣电影、读书和音乐三个部分单独设计的爬取项目（item），之后根据需要还可以进行弹性的扩充。

5.3.3 用户代理中间件(useragent middleware)

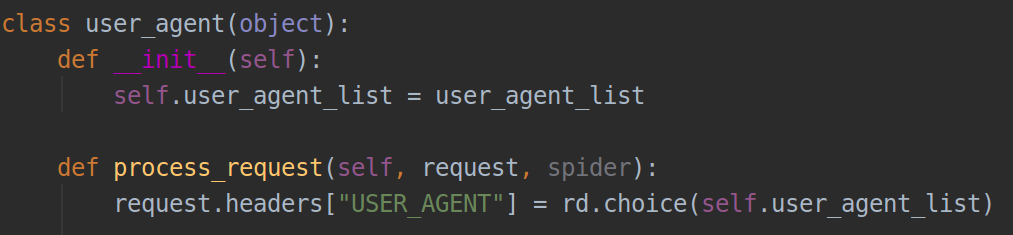


图5.4 用户代理中间件（useragent middleware）设计

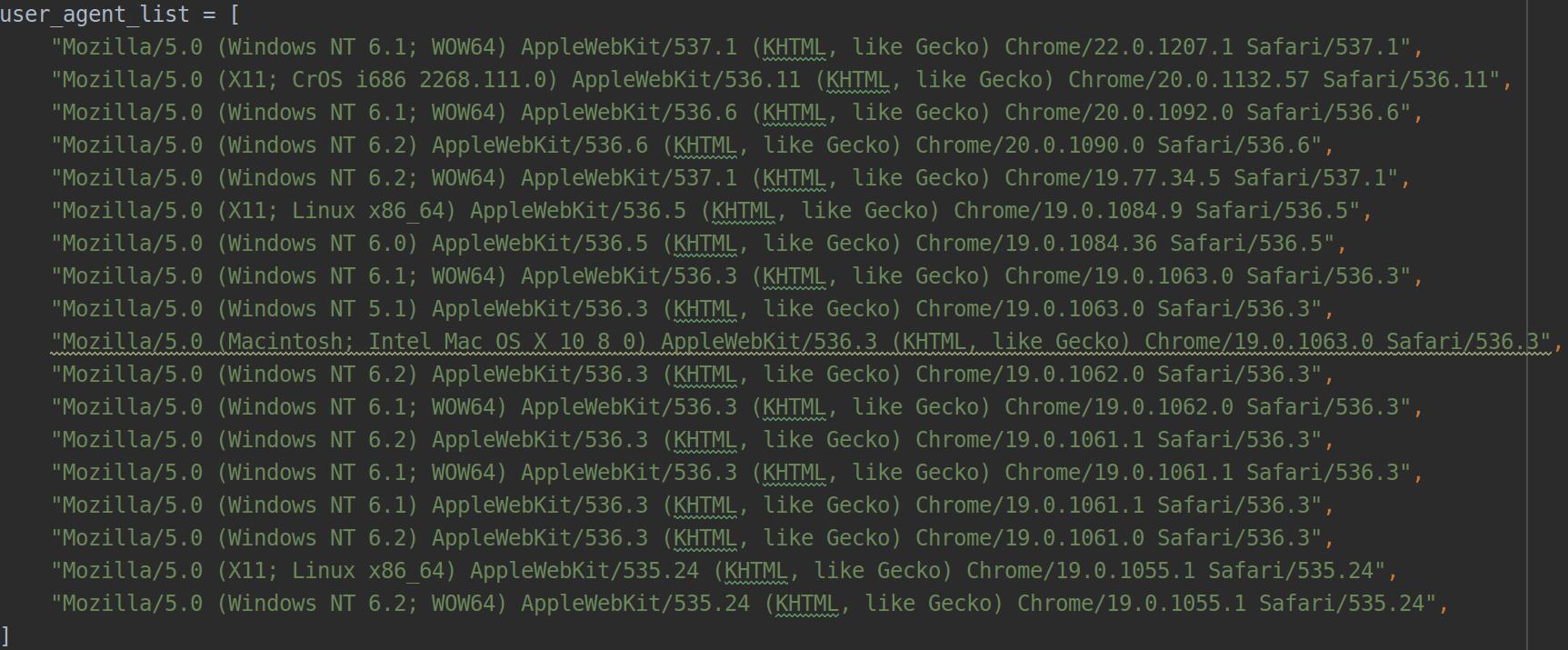
  
图5.5 用户代理

图5.4和图5.5展示了用户代理及其中间件的实现，目前大部分的网站都会对对Request 的header中发送请求的useragent进行解析从而判断是否是机器人，用户代理中间件的实现十分必要。

5.3.4 下载中间件(download middleware)

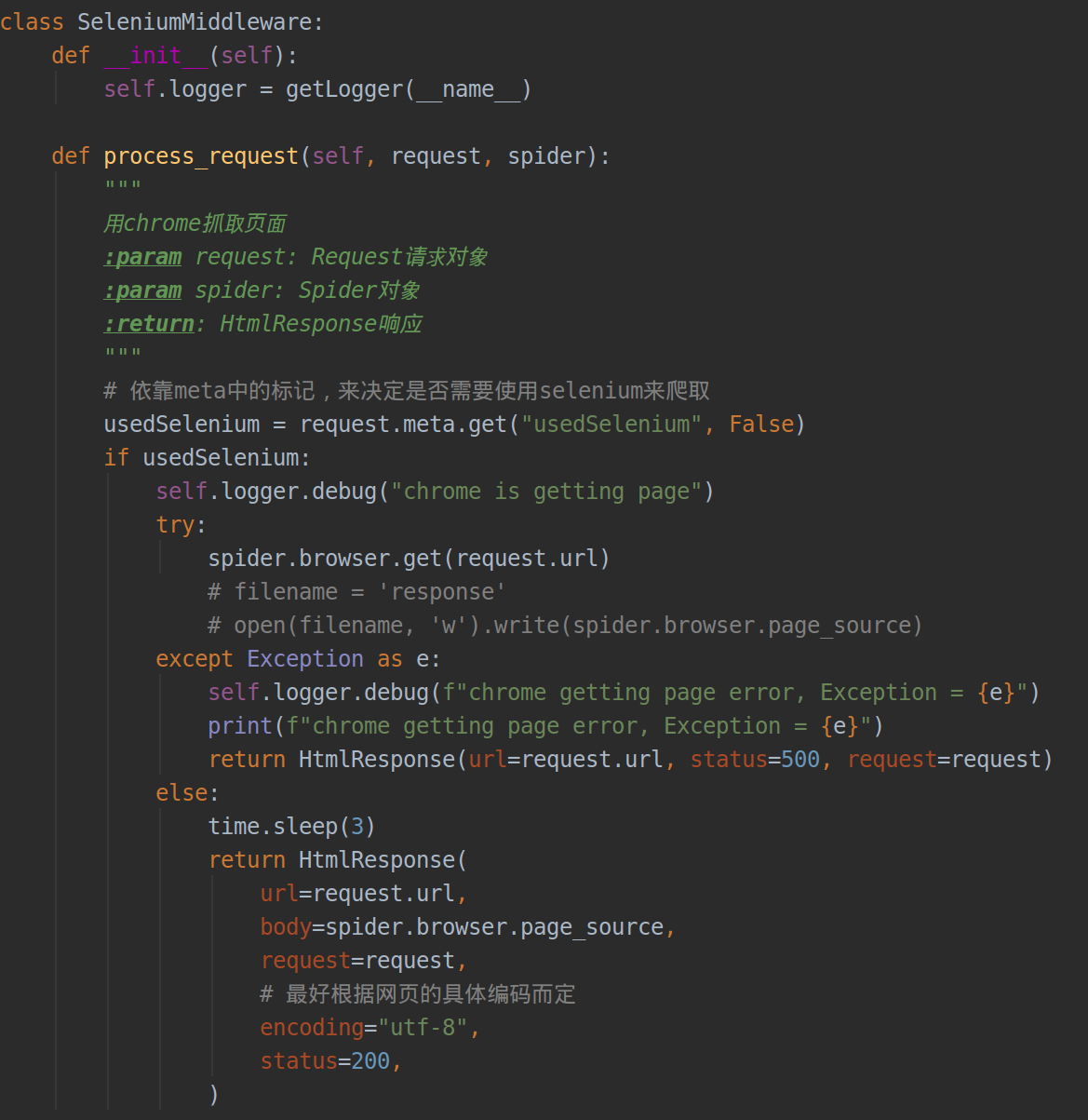


图5.6 Selenium实现的下载中间件

图5.6展示了基于Selenium实现的下载中间件，采用chrome-headless作为中间浏览器，对于一些动态JS生成的网页的爬取是必要的应对首选。如图1.7所示，browser的启动与spider的初始化相互绑定，这样的好处是统一爬虫项目下不同的spider一起运行时都会拥有不同的browser对象，不会发生抢夺或冲突；当spider结束运行之后，通过信号量自动关闭browser防止内存的溢出。

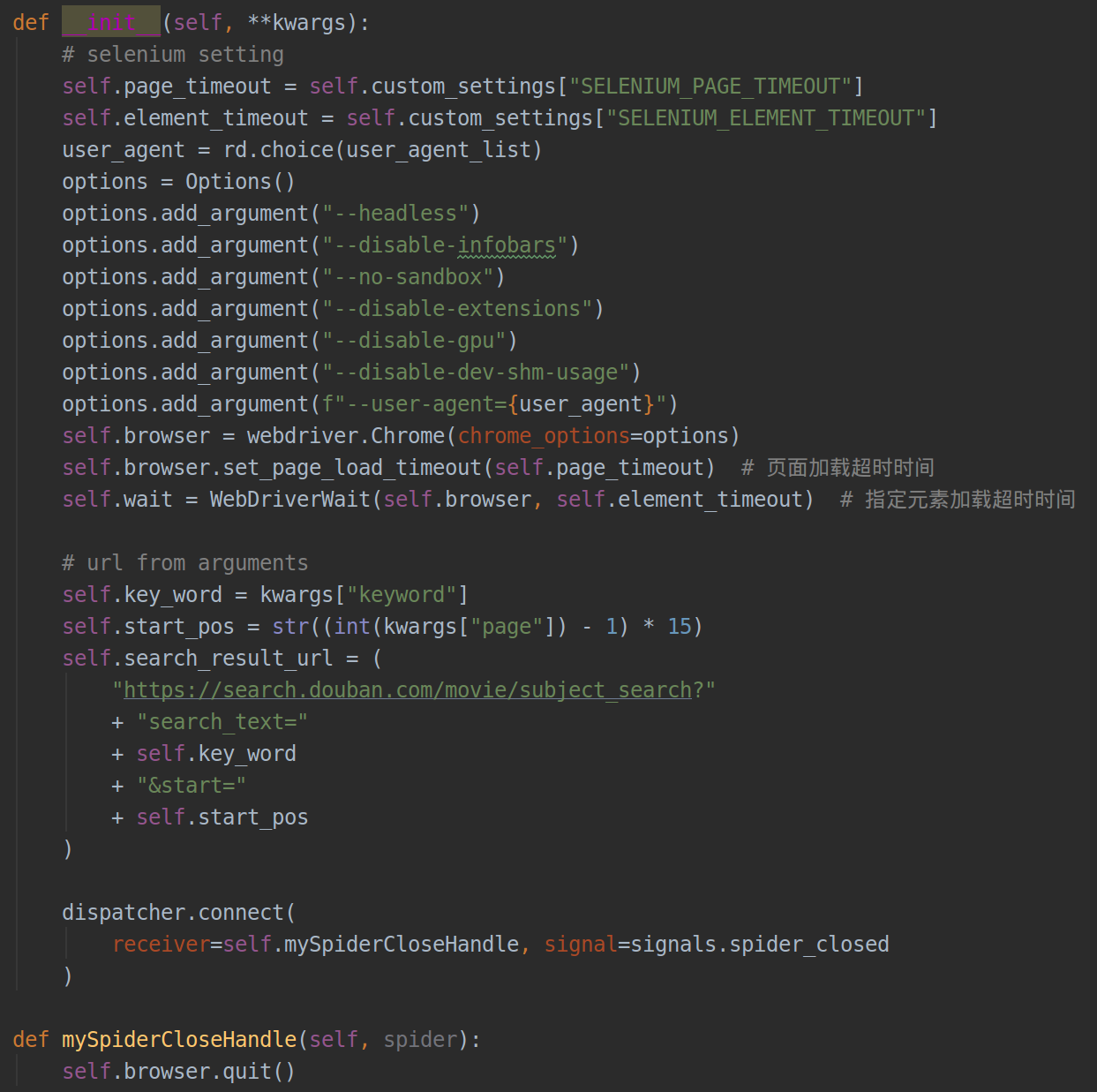


图5.7 Spider中Selenium browser的配置

5.3.5 项目管道(item pipeline)

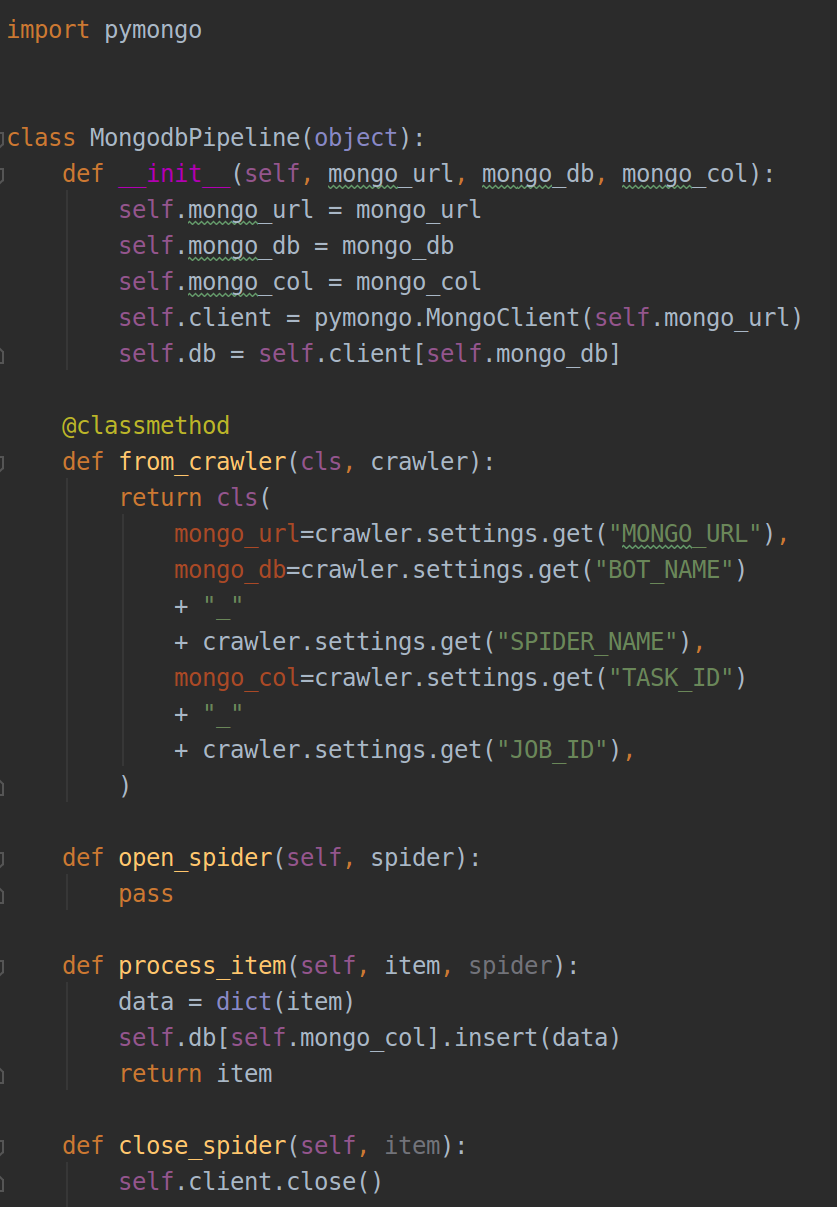


图5.8 MongoDB项目管道

爬虫的爬取结果通过项目管道中pymongo提供的API保存到MongoDB数据库中，MongoDB数据库的所在服务器位置、数据库名称、集合的名称都通过外部传参的方法在运行时确定，做到灵活可变。

5.3.6 配置文件(config)

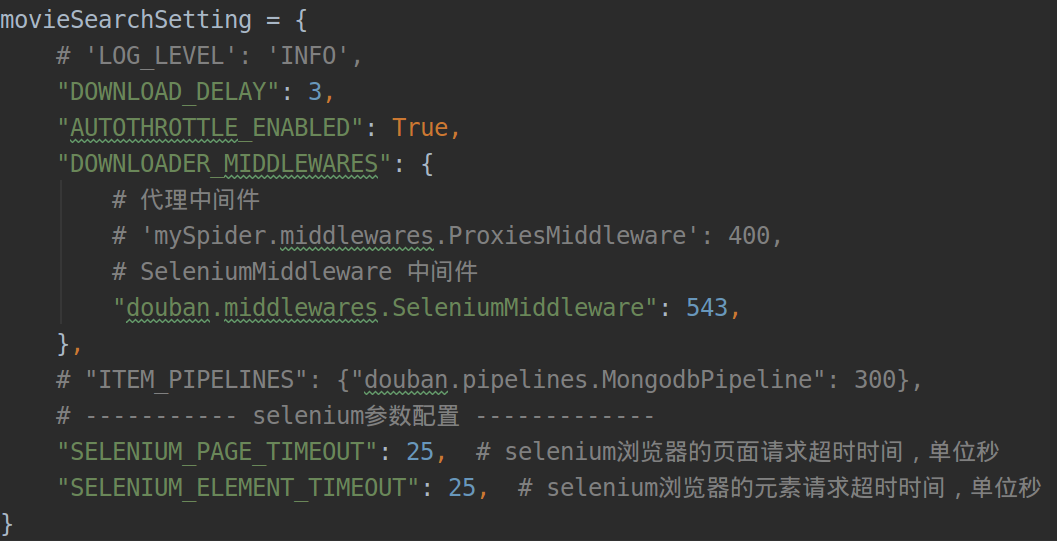


图5.9 配置文件

Scrapy中的参数优先级顺序为：程序启动时通过flag传入参数、用户自定义配置参数、项目全局参数、预设默认参数。图5.9展示了为每一个爬虫单独定义的用户自定义配置参数，可以覆盖setting.py中的全局配置参数。通过配置文件可以灵活地为每一个spider单独进行配置和调试，提高了项目的可扩展性。

5.3.7 传入参数

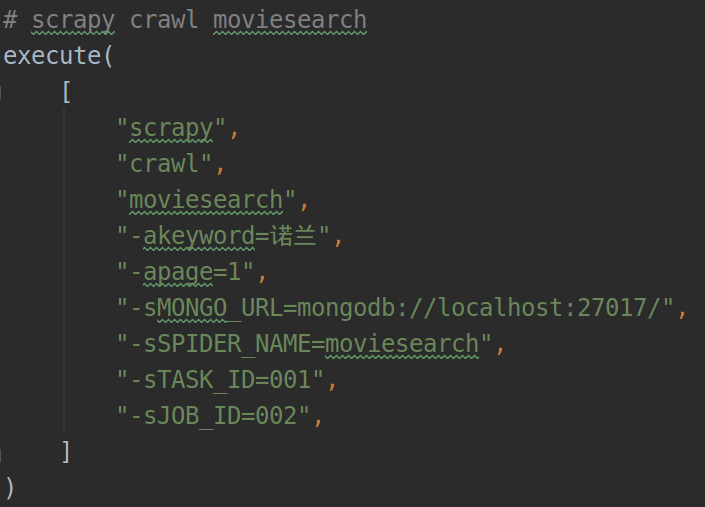


图5.10 运行时传入参数

爬虫任务的运行需要实际情况下用户选择的模板、填写爬虫信息、系统自动生成的Task号和Job号、数据库的URL等参数。这里选择使用运行时传入参数的方法进行动态的配置，需要预先与easyspider-web以及easy-scheduler的开发人员约定好传入参数的格式。实际传入参数的格式如图5.10所示。

5.3.8 利用Scrapyd打包项目

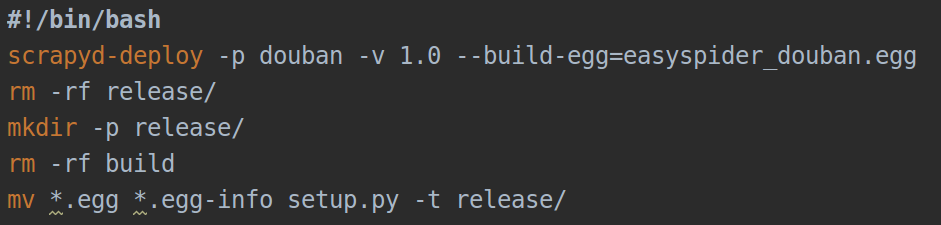


图5.11 利用Scrapyd client对爬虫项目进行打包

为了实现分布式爬虫并且可以对爬虫任务进行细化拆分，不能简单的在服务器上直接运行爬虫任务，而是需要通过Scrapyd对爬虫任务进行统一的调度。这就需要将一般的爬虫任务打包成Scrapyd所需要的.egg格式并且进行发布。与.egg一同产生的还有一些项目信息、版本号以及配置需求信息等，方便easyspider-scheduler的部署和迭代更新。

5.3.9 反爬虫应对

为了应对目标网站的反爬虫机制，目前已经对下述机制进行了支持：

* 随机更换user-agent
* 禁用robot规则
* 禁用cookie
* 自动限速
* Selenium模拟浏览器操作

目前的网页爬取效果良好，之后还会根据需要，对模拟登陆，动态IP代理（分布式爬虫已经实现部分替代）以及验证码识别进行支持。