基于Scrapy的WebUI开发

——项目需求分析说明书

SY1906514 汪凌风 SY1906508 明　昊

SY1906513 汪丽萍 SY1906431 郑泽西

SY1806220 闫奕涛 SY1906509 邵志钧

2020/03/25

# 版本记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 修改说明 | 参与者 |
| 1.0 | 2020.3.26 | 完成第一版 | 全体成员 |
| 1.01 | 2020.3.26 | 增加版本记录，修改用例图及RUCM | 闫奕涛，明昊，邵志钧 |

目　录

[基于Scrapy的WebUI开发 1](#_Toc36125300)

[版本记录 2](#_Toc36125302)

[一、引言 5](#_Toc36125303)

[1.1 编写目的 5](#_Toc36125304)

[1.2 背景 5](#_Toc36125305)

[1.3 定义 5](#_Toc36125306)

[1.4 参考资料 6](#_Toc36125307)

[二、任务概述 7](#_Toc36125308)

[2.1 目标 7](#_Toc36125309)

[2.2 用户特点 7](#_Toc36125310)

[2.3 假定与约束 7](#_Toc36125311)

[三、需求与设计 8](#_Toc36125312)

[3.1 需求 8](#_Toc36125313)

[3.1.1业务需求 8](#_Toc36125314)

[3.1.2功能需求 9](#_Toc36125315)

[3.1.3 非功能性需求 9](#_Toc36125316)

[3.2 框架及组件概述 10](#_Toc36125317)

[3.3 用例图 11](#_Toc36125318)

[3.3.1 调度模块 12](#_Toc36125319)

[3.3.2 接受请求 13](#_Toc36125320)

[3.3.3 发送请求 14](#_Toc36125321)

[3.3.4 下载页面 15](#_Toc36125322)

[3.3.5 生成请求 16](#_Toc36125323)

[3.3.6 解析响应 16](#_Toc36125324)

[3.3.7 处理item 18](#_Toc36125325)

[四、运行环境规定 20](#_Toc36125326)

[4.1 设备 20](#_Toc36125327)

[4.2 支持软件 20](#_Toc36125328)

[4.3 接口 20](#_Toc36125329)

[4.3.1 硬件接口 20](#_Toc36125330)

[4.3.2 软件接口 20](#_Toc36125331)

[4.3.3 通信接口 20](#_Toc36125332)

[4.3.4 用户接口 20](#_Toc36125333)

[五、基于Scrapy的WebUI开发 21](#_Toc36125334)

[5.1 整体思路说明 21](#_Toc36125335)

[5.2 WebUI用例图 22](#_Toc36125336)

[5.3 功能详细设计 23](#_Toc36125337)

[5.3.1 管理脚本 23](#_Toc36125338)

[5.3.2 自动化生成脚本 24](#_Toc36125339)

[5.3.3 生成词云 25](#_Toc36125340)

[5.3.4 在线编辑调试 25](#_Toc36125341)

[5.3.5 选择优化 27](#_Toc36125342)

# 一、引言

本软件Scrapy是开源爬虫框架。本需求规格说明书详细描述了Scrapy框架的功能需求、非功能需求、业务流程、架构以及改进需求及技术路线。

## 1.1 编写目的

本软件需求规格说明书，是为了软件设计、软件测试人员和用户所编写的。

本软件规格说明书的适用读者，包括参加能力验证的开发测试人员、Scrapy技术人员，以及项目的其他相关人员。

## 1.2 背景

软件名称：Scrapy

项目的组织机构：Scrapy开源项目开发组

项目的实施机构：Github站点上的359位贡献者

项目背景：本项目是一个高速并发的网络爬虫框架，用于对网站进行爬网并从其页面提取结构化数据。

## 1.3 定义

|  |  |
| --- | --- |
| Engine | 引擎，处理整个系统的数据流处理、触发事务，是整个框架的核心。 |
| Item | 项目，它定义了爬取结果的数据结构，爬取的数据会被赋值成该Item对象。 |
| Selectors | Scrapy有自己的数据提取机制。它们被称为选择器，因为它们“选择”HTML文档的某些部分 XPath 或 CSS 表达。 |
| Scheduler | 调度器，接受引擎发过来的请求并将其加入队列中，在引擎再次请求的时候将请求提供给引擎。 |
| Downloader | 下载器，下载网页内容，并将网页内容返回给“蜘蛛”。 |
| Spider | “蜘蛛”，其内定义了爬取的逻辑和网页的解析规则，它主要负责解析响应并生成提取结果和新的请求。spider是定义一个特定站点（或一组站点）如何被抓取的类，包括如何执行抓取（即跟踪链接）以及如何从页面中提取结构化数据（即抓取项）。换言之，spider是为特定站点（或者在某些情况下，一组站点）定义爬行和解析页面的自定义行为的地方。 |
| Item Pipeline | 项目管道，负责处理由蜘蛛从网页中抽取的项目，它的主要任务是清洗、验证和存储数据。 |
| Downloader Middlewares | 下载器中间件，位于引擎和下载器之间的钩子框架，主要处理引擎与下载器之间的请求及响应，自定义扩展下载功能的组件。 |
| Spider Middlewares | 蜘蛛中间件，自定义扩展Engine和Spider中间通信的功能组件。在该机制中，可以插入自定义功能来处理发送到的响应。 |

表1 术语定义

## 1.4 参考资料

　　GB/T 9385-2008 计算机软件需求规格说明

　　GB/T 8567-2006 计算机软件文档编制规范

# 二、任务概述

## 2.1 目标

Scrapy是一个高速并发的网络爬虫框架，使用Python编写并适用于Python的快速高层次的屏幕抓取和网页抓取框架。其目的是为了构建一套方便高效的爬取框架，用于进行网络站点爬取并提取结构数据。Scrapy可以用于数据挖掘、监测及自动化测试中。Scrapy由于其扩展性优秀，因此任何用户都可以根据其用途需求修改框架内容。框架提供了多种爬虫基类，并使用Twisted异步网络库来处理网络通讯。同时框架具有广泛的内置扩展和用于处理的中间产品，可以灵活应用于各种不同的场景。

## 2.2 用户特点

熟悉Python和网页爬取的程序开发用户。

## 2.3 假定与约束

无

# 三、需求与设计

## 3.1 需求

### 3.1.1业务需求

　　对爬虫最典型业务场景——页面数据抓取来说，在需要大量下载网络数据以便后续分析的时候，往往通过手动下载是不能满足需求的，就需要有一个自动化的下载方法。一般的高级语言都会提供相关网络数据下载的功能函数，通过调用就可完成简单的下载。但是在被抓取对象的结构比较复杂时，就需要开发人员自己根据需要编写相应的抓取程序，而抓取程序很大程度上都有重复公用的代码，每次都重新开发编写显然非常不合适。因此，就需要有一个抽象的开发爬虫的框架方便开发人员进行开发，避免了重复造轮子的工作。

　　Scrapy就是在这种需求背景下开发出来的。它将可重用的代码部分抽象出来成为模块调用，使得开发人员只需关注程序要实现的具体功能对象而不用再关心基础模块的搭建，为快速开发爬虫软件提供便捷、节约时间。另外，在爬虫代码编写中最耗时的是反爬虫的问题。使用scrapy框架在开始写代码之前可以利用内置的shell去请求你抓取网站的数据页面，检测网站的反爬虫能力，再用框架提供的模块编写相应程序，就会使得编写代码的复用率很高，更换爬取对象时一般只需改改正则表达式和爬取队列即可，从而大大的提高了工作的效率。

　　为了使得框架使用起来具有一定的灵活性，又具有一定的便捷性，因而将框架设计成多个模块，简单的爬取只需下载器去下载页面数据。对于多任务同时进行的要求就需要引入并行的spider去并发执行，而这些并发任务就需要一个调度器去合理的分配任务执行顺序。对于数据也要有相应的处理模块，每个模块的任务分配和整个程序的数据流控制就需要有一个核心的控制引擎去负责。

　　一个典型的案例就是亚马逊公司，亚马逊利用爬虫技术，在scrapy框架的基础上进行扩展，实现了其商品广告接口功能，通过灵活的API接口，为客户提供其商品的实时广告链接数据。对抓取的商品交易信息、价格信息、折扣信息等数据进行分析处理后自动化地实时更新商品广告，从而省去很多的人力成本。

### 3.1.2功能需求

当用户需要开发爬虫系统时，可以调用本框架，通过本框架提供的组件，搭建好一套高效的爬虫系统。用户花费更多的时间在逻辑实现上，而不必关心底层的实现细节。基于以上业务需求的分析与相应的业务要求,我们认为软件有如下的功能需求。

1. 爬虫引擎——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；
2. 调度功能——可调度的爬取请求队列。
3. 下载功能——获取页面数据并提供给引擎，而后提供给 spider。
4. 处理数据功能——从抓取到的网页中（HTML源码）提取数据。
5. 日志功能——事件日志记录。
6. 扩展机制——允许开发者自定义。
7. 设置选项——用户可选运行模式。
8. 异常处理——应急、错误处理。

### 3.1.3 非功能性需求

Scrapy 需要支持以下非功能性需求以便更好了使用框架编写爬虫。

1. 高适用性。可以用scrapy crawl 来启动Scrapy，也可以使用 API 在脚本中启动Scrapy。默认情况下，执行 scrapy crawl 时，Scrapy每个进程运行一个spider。 Scrapy通过内部(internal)API 也支持单进程多个spider。除了支持单机爬取，也可以进行分布式爬取，支持启动多个Scrapyd，并分配到不同机器上。
2. 高可靠性。有些网站实现了特定的机制，以一定规则来避免被爬虫爬取。 框架需要能够实现避免被禁止(ban)，并且可以设置全局并发以便同时处理多个request。
3. 高便捷性。提供交互式shell终端 , 为测试CSS及XPath表达式，编写和调试爬虫提供了极大的方便。提供数据导出功能，提供对多格式(JSON、CSV、XML)、多存储后端(FTP、S3、本地文件系统)的支持。提供了一系列在spider之间共享的可复用的过滤器(即 Item Loaders)，对智能处理爬取数据提供了内置支持。
4. 高通用性。针对非英语语系中不标准或者错误的编码声明, 提供了自动检测以及健壮的编码支持。
5. 高扩展性。通过使用信号、设计好的API(中间件, extensions, pipelines)来定制实现功能。内置的中间件及扩展为下列功能提供支持: cookies and session 处理、HTTP 压缩 、HTTP 认证 、 HTTP 缓存 、user-agent模拟、爬取深度限制。

## 3.2 框架及组件概述



图1 Scrapy框架图

Scrapy框架如图1所示（来源：http://www.jianshu.com/p/078ad2067419），各部分组件介绍如下。

1. 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。
2. 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。
3. 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。
4. Spiders：每个spider负责处理一个特定(或一些)网站。
5. 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)。
6. 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。
7. 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。

## 3.3 用例图

　　通过对本项目中原生Scrapy框架的学习与分析，Scrapy框架的用例图如图2所示。其中包括Engine、Scheduler、Downloader、Spider和Pipeline五位用户，以及调度模块、调度Request、下载页面、生成请求、解析响应和处理item六个用例。

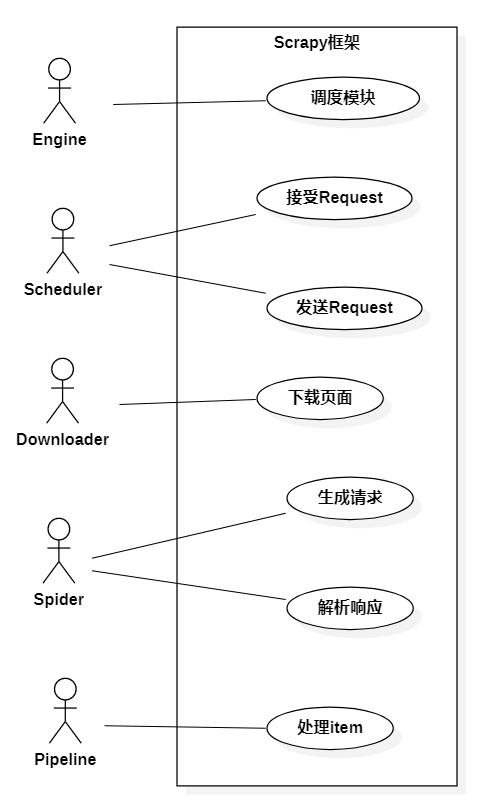


图2 Scrapy用例图

### 3.3.1 调度模块

　　Engine需要负责所有模块的调度，正常流程下是Spider发出一个request并启动Engine，并向其发送需要爬取的url，Engine得到url后，将其送给Scheduler等待调度，待Scheduler调度完成后，接收下一个需要爬取的url，将其送给下载器，下载器完成下载工作后，将内容封装成HttpResponse返回Engine，Engine再发给Spider进行处理（比如带入正则表达式，CSS选择器提取网页字段），Spider将结果封装成Item返回Engine，Engine最后通过Pipeline将结果一步一步的Piplines将数据持久化到不同存储体里，比如JSON，Mysql，ES等。



图3 调度模块RUCM图

### 3.3.2 接受请求

　　Scheduler在开始时会先初始化一个request队列，之后的工作就是维护这个队列，维护队列的方法是：接收Engine发来的request，放入过滤器中（进行去重等操作），通过过滤器的request则放入队列中等待调度。

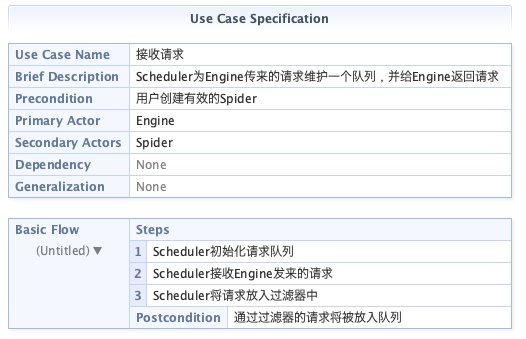


图4 接受请求RUCM图

### 3.3.3 发送请求

　　在处理完一个request后，Engine会向Scheduler再次请求一个request，这时如果Scheduler维护的队列中已无request，则代表已经完成用户的任务，程序结束，如果还有request，则将优先队列对首的request返回给Engine。



图5 发送请求RUCM图

### 3.3.4 下载页面

　　Downloader包含了从调度器调取url之后到scraper获取返回的网页内容之前的所有步骤，关系到网页如何下载，是最复杂的一部分内容。

　　传递的参数中，需要下载的资源是什么类型，就选用对应的下载处理器进行网络下载。下载处理器的工作就是：管理着各种资源对应的下载器，在真正发起网络请求时，选取对应的下载器进行资源下载。

　　中间件的职责是从某个组件流向另一个组件时，会经过一系列中间件，每个中间件都定义了自己的处理流程，相当于一个个管道，输入时可以针对数据进行处理，然后送达到另一个组件，另一个组件处理完逻辑后，又经过这一系列中间件，这些中间件可再针对这个相应结果进行处理，最终输出。



图6 下载页面RUCM图

### 3.3.5 生成请求

　　Spider中定义了爬取URL地址和爬取的方式，包括爬取的动作和如何从网页中提取结构化数据。其中Spider功能包括生成请求和解析响应。生成请求的过程中，首先Spider根据start\_urls内的URL地址生成请求对象并发送给Engine，之后若收到的响应中包含其他URL地址则继续生成请求对象并发送给Engine，直到收到的响应中不包含其他URL地址。



图7 生成请求RUCM图

### 3.3.6 解析响应

Spiders类最重要的一个功能就是在其中自定义回调函数，实现对Engine组件传来的响应对象的解析。包括了爬取的动作(例如:是否跟进链接)以及如何从网页的内容中提取结构化数据(爬取item)，同时它还负责从解析的数据中提取出继续需要爬取的url，然后利用这个url生成Request对象，将这个对象传递给Engine进行下一次请求的发送与响应的解析。这个用例的具体的流程如下：

1. Engine接收到下载器返回的响应对象并将其发送给Spiders。
2. Spiders中间件处理Engine传入的响应对象，Spider中间件是一个钩子框架，可以钩住Scrapy的Spider处理机制，在该机制中，可以通过编写自己的Spiders插入自定义功能来处理发送到的响应，同时也可以使用Scrapy附带的Spiders中间件组件。要激活蜘蛛中间件组件，需要将其添加到 SPIDER\_MIDDLEWARES 设置。
3. 如果中间件收到的响应不符合标准，比如输出无法处理或网站的返回状态不正确，需要抛出错误并将错误类型记录到日志中。
4. Spiders接收到处理后的响应对象。
5. 使用选择器从响应对象中提取出数据，选择器是一种Scrapy自己的数据提取机制，因为它们“选择”HTML文档的某些部分 XPath 或 CSS 表达，同时也可以使用别的机制进行数据的提取，比如BeautifulSoup和lxml，都是常见的实现这个任务的框架。
6. 将目的数据存入Item对象传递给Pipeline，Item对象是scrapy提供的一个用于收集爬取数据的简单容器。他们提供了一个 dictionary-like 具有声明可用字段的方便语法的API，也可以python dicts的形式返回提取的数据。虽然方便和熟悉，但python dicts缺乏结构：很容易在字段名中输入错误或返回不一致的数据，特别是在有许多spider的大型项目中。
7. 从数据中提取的url生成Request对象。
8. 将Request对象传递给Engine，这样就可以进行下一次的数据迭代。



图8 解析响应RUCM图

### 3.3.7 处理item

　　在已经从指定网页爬取到需要的内容之后，开发者可能要对爬取到的结果做最后的处理，或是要清理HTML数据，或是要对已经抓取到的数据做验证，或是检查重复数据，或是对非文本数据做特殊处理。所以，开发者需要一个针对已抓取数据的管道Pipeline，Pipeline用来对数据做最后的处理。

　　Pipeline应该可以在整个爬取过程中一直运行直至脚本运行完毕。Pipeline的实际操作内容应该由开发者自己来定义，同时，它应该可以与Scrapy的其他核心组件进行交互。Pipeline应该提供对于非文本的爬取数据的默认处理支持，至少包括文件数据，图片数据和多媒体数据。

　　开发者应该可以针对一个Spider脚本定义多个Pipeline进行处理，这写Pipeline按照一个指定的序列顺序执行。



图9 处理item的RUCM图

# 四、运行环境规定

## 4.1 设备

　　系统：Linux、Mac、Windows

　　内存：256MB以上

硬盘：20G以上

用于开发python爬虫的PC机或网络服务器

## 4.2 支持软件

　　Python3或2.7，基于Windows、Linux、MacOs等平台

## 4.3 接口

### 4.3.1 硬件接口

　　无

### 4.3.2 软件接口

　　Python类库

### 4.3.3 通信接口

　　HTTP协议、TCP/IP协议、HTTPS协议

### 4.3.4 用户接口

　　命令行工具：通过 scrapy 命令行工具进行控制。

　　Scrapy终端：交互终端，提供未启动spider的情况下尝试及调试爬取代码。

telnet终端：以供检查和控制Scrapy运行的进程。

# 五、基于Scrapy的WebUI开发

## 5.1 整体思路说明

Scrapy这个框架并非完美，其本身是有一些缺陷的，如：与用户交互的方式依然是古老的命令行交互，没有自带的可视化用户操作界面，其性能上也有可以改进的地方，如在爬取大量信息时，容易出现内存泄漏的问题。本项目的主要工作在于对Scrapy进行功能扩展和性能改进，针对刚才所述的这些缺点，目前计划为Scrapy增加如下新的功能点：

* 为Scrapy定制可视化web用户界面。
* 在web界面中输入用户需要筛选的标签、目标网站等信息，为用户输出爬虫的内容，时长等，并对输出进行排版。
* 为用户显示爬取信息的词云。
* 将在web界面提供调试与编译python代码的功能。
* 针对一些特定网站给出性能优化方案和性能指标对比。

## 5.2 WebUI用例图

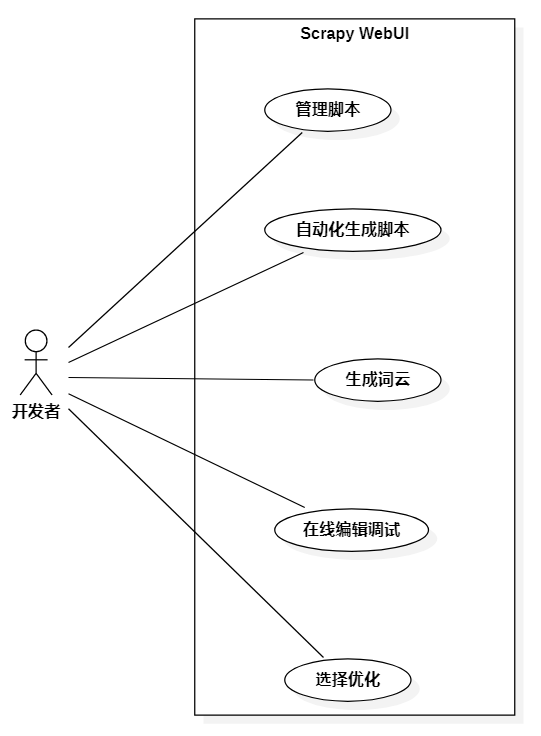


图10 WebUI用例图

图10描述了基于Scrapy的WebUI的用例图。一个可视化的管理开发界面对于初学者来说是友善的，可以显著提高开发心情与开发效率。对于熟练者来说，可视化的WebUI开发界面也能帮助开发者缩短开发时间。

WebUI界面主要提供的功能出于如下考虑。首先，一个开发界面，开发者必定需要在界面内进行代码的编写与调试。所以提供在线编辑调试功能。其次，可视化提供的便利方面开发者在可视化界面上进行简单的对于脚本的管理。所以提供管理脚本的功能。用户可以在这个界面观测脚本的运行状态与输出结果。同时，为了方便初学者上手和减少熟练者所需要花费的时间，根据输入的网址与标签自动化生成脚本也是需要提供的。

最后是两个Web界面提供的改进功能。生成词云提供给开发者查看1-N个网站的词云情况，是对于爬取内容的改进。选择优化会提供给开发者对于爬虫脚本的1-N个优化方案进行选择，这些优化方案会某种程度上提升脚本的爬取性能。

## 5.3 功能详细设计

### 5.3.1 管理脚本

　　脚本管理界面是一个基于web的，对于scrapy脚本进行可视化管理的界面。用户可以在当前界面选择自定义脚本进行运行，界面会显示脚本的运行状态与运行时间。同时，用户可以选择某个脚本查看脚本的结构化输出。界面会提供输出下载TXT文本功能，如果用户需要的话，经过在页面上简单的配置，也可以帮助用户将脚本的爬取内容保存入数据库。最后，用户的使用记录与爬取结果会在web界面上维持一段时间的保存。



图11 管理脚本RUCM图

　　脚本管理界面实现原理比较简单。用户选择自定义脚本进行运行，只需要在用户选择需要执行的脚本之后通过后端接口将脚本的对应路径传给后端API进行执行即可。同时在执行过程中与后端建立websocket长链接，通过规律的通信来监测脚本的运行情况与读取输出。对于脚本运行时间的计时可以在前端进行计算。对于脚本的结构化输出，由于scrapy爬取的脚本属于结构化数据，所以可以参考JSON的结构化输出工具进行开发。脚本爬取内容存入数据库目前仅准备对接Mongodb非关系型数据库。最后，用户的使用记录与爬取结果会通过txt形式保存在本地，需要的时候方便读取。

### 5.3.2 自动化生成脚本

　　用户可以通过择标签与目标网站来自动生成爬取脚本，同时将脚本托管给web界面运行。同时用户可以下载生成后的脚本文件。这个功能的实现原理也比较简单，主要需求开发者对scrapy足够熟悉。这是一个半自动化生成脚本的功能，开发者需要设定好一套模板，根据用户输入的网址与标签，套用模板生成简单可用的scrapy爬取脚本。脚本实现功能的多少取决于开发者对scrapy框架的理解程度。



图12 自动化生成脚本RUCM图

### 5.3.3 生成词云



图13 生成词云RUCM图

### 5.3.4 在线编辑调试

Scrapy原生是不具备WEB界面，爬虫的编写、调试都是通过代码和命令行操作，这对于有些人来说可能不那么方便，同时scrapy默认的debug模式信息量太大, 不容易调试, 可读性略差。因此考虑在我们的web框架的基础上增加在线编辑，编译与调试的模块。可以使得爬虫的编写、调试都在 Web界面中进行的，将在线编辑器集成在我们的可视化框架内，可以方便地进行可视化的编写，同时可以在线实时给出某些网页的爬取状态，比如当前正在爬取的网页，当前访问网站的运行状态，之前访问失败的网站列表，方便我们调试爬虫，简化了爬取数据的流程，使得可视化框架的功能更加完善。同时方便某些人编写，调试和运行代码。

这个用例的具体的流程如下：

1. 开发者登录网站，进入我们扩展的可视化界面。
2. 开发者在网站页面上选择在线编辑模块并进入。
3. 开发者在在线编辑器中实现计划项目的部分爬虫代码，这个在线编辑器应保证简单方便。
4. 开发者在编写完成计划代码后点击运行按键，运行编写完成的代码。
5. 系统编译并运行开发者在线编辑完成的代码，同时给出相应的运行信息与状态。
6. 开发者根据输出的运行信息进行代码的调整调试并重复上述过程。

表述这个流程的RUCM图如下图所示：



图14 在线编辑调试RUCM图

　　在经过在线调研后，我们暂定使用ACE框架来帮助我们实现这一扩展功能，ACE 是一个开源的、独立的、基于浏览器的代码编辑器，可以嵌入到任何web页面或JavaScript应用程序中。ACE支持超过60种语言语法高亮，并能够处理代码多达400万行的大型文档。ACE开发团队称，ACE在性能和功能上可以媲美本地代码编辑器（如Sublime Text、TextMate和Vim等）。同时它是Mozilla Skywriter（以前称为Bespin）项目的继任者，并作为Cloud9的主要在线编辑器。我们可以在它的基础上将在线编辑器模块集成到我们的可视化框架里。这只是在经过了初步调研之后的规划，在后续实现过程中如果对整体的实现有进一步的了解也可能使用别的方法。

　　在后续进行的过程中，如果时间和精力允许，我们也可利用这个在线编辑器实现可配置化爬虫的实现，即利用一些规则来完成爬取规则和解析规则的配置，这样可配置化程度就非常高，这样我们只需要维护爬取规则、提取逻辑就可以了。如果要新增一个爬虫，我们只需要写好对应的规则即可。我们可以在在线编辑器中增加一个可配置化爬虫的创建选项，接着我们便可以在添加提取实体、爬取规则、抽取规则了，例如选择什么解析器，我们可以配置解析成为哪个实体，每个字段使用怎样的解析方式，如 XPath 或 CSS 解析器、直接获取属性、直接添加值等多重方式，另外还可以指定处理器进行数据清洗，或直接指定正则表达式进行解析等等，通过这些流程我们可以做到任何字段的解析。再比如爬取规则，我们可以指定从哪个链接开始爬取，允许爬取的域名是什么，该链接提取哪些跟进的链接，用什么解析方法来处理等等配置。通过这些配置，我们可以完成爬取规则的设置。然后即可自动完成一些小规模代码的生成。

### 5.3.5 选择优化



图15 选择优化RUCM图

开发者首先登录web界面对爬虫进行设置，通过WebUI界面输入需要爬取的目标URL地址并设定爬取脚本模版，之后选择意向的优化方式并启动爬取程序，程序会根据开发者选择的优化选项对目标URL进行爬取，系统在完成本次爬取之后，会输出本次爬取的结果以及用时等信息。

**方案一：突破反爬改进**

Scrapy作为一个爬虫框架，由于会在短时间内对一个网站多个URL进行访问、有着相同IP地址且User Agent均为Python，很容易被网站的反爬虫机制识别并暂时限制访问或使用登录机制和验证码来阻止爬虫。使用原生Scrapy框架可以通过使用修改DownloaderMiddleware更改User Agent伪装为浏览器、在每次的请求中带上cookie或token实现登录或使用IP地址代理来突破网站的反爬虫机制。网站常见的反爬虫机制包括监测访问量与访问IP和User Agent、要求访问前登录、限制相同IP访问频率、通过账号权限限制单一账号访问内容、验证码登录、监测请求内容等。因此基于Scrapy框架的突破反爬改进可以从改进DownloaderMiddleware以模拟浏览器操作并实现登录，或通过集成Selenium完全模仿浏览器操作，或通过随机使用多个免费代理来减少单个IP的访问次数，或针对特定网站集成二维码识别类库，四种方式针对不同的反爬机制均可以提升原生Scrapy框架的突破反爬性能。

**方案二：用Bloom Filter改进Scrapy的去重算法**

　　Scrapy本身采用的去重机制是利用Set去重，这是一种原始有效的方法，它的优点在于算法简单且运行速度较快，但面对大数据量的时候，这一算法的弊端就开始显现。本算法将Request的指纹存储到了Redis集合中，每个指纹的长度为40，且内存中不仅仅要存储指纹，还要存储爬取队列，当数据量很大时，集合去重已经不能满足我们的要求。所以我们需要使用一个更加节省内存的去重算法：Bloom Filter。

　　在Bloom Filter中使用位数组来辅助实现检测判断。在初始状态下，我们声明一个包含m（m的值有具体要求）位的位数组，它的所有位都是0。空间复杂度被降至O(1)。

　　该算法的实现思路是：将集合中的每个元素用k个散列函数得出k个值，将k作为数组下标，将位数组中对应下标置为1，若待测元素经过散列函数计算得到的下标位都为1，那么这个元素在集合中就已经存在了。

现在我们有了一个待检测集合，其表示为S={x1, x2, …, xn}。接下来需要做的就是检测一个x是否已经存在于集合S中。在Bloom Filter算法中，首先使用k个相互独立、随机的散列函数来将集合S中的每个元素x1, x2, …, xn映射到长度为m的位数组上，散列函数得到的结果记作位置索引，然后将位数组该位置索引的位置1。例如，我们取k为3，表示有三个散列函数，x1经过三个散列函数映射得到的结果分别为1、4、8，x2经过三个散列函数映射得到的结果分别为4、6、10，那么位数组的1、4、6、8、10这五位就会置为1，如下图所示。

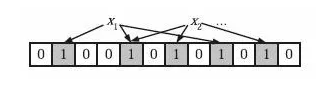
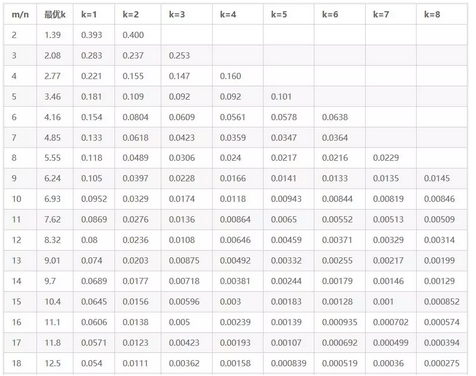


图16 案例解释图

该方法存在误判率，但如果取足够大的m和k，误判率可以忽略不计（下图中n表示集合的元素个数）。



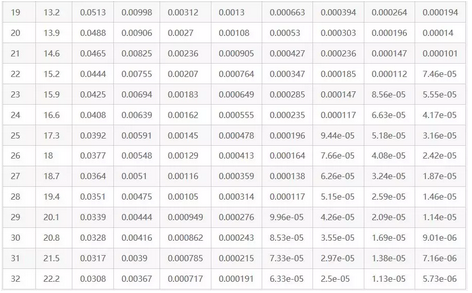


图17 误判率