

**Scrapy项目计划书**

v1.4.1

赵正阳 SY1906428

郭浩隆 SY1906430

沈一聪 SY1906510

梁远志 SY1906503

宋冰晨 SY1906429

2020年3月26日

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 变更时间 | 修改人 | 审核人 | 备注 |
| 1.0 | 2020/3/18 | 赵正阳 | 沈一聪、梁远志、宋冰晨、郭浩隆 | 建立初始项目计划书文档 |
| 1.1 | 2020/3/20 | 宋冰晨 | 沈一聪、梁远志、赵正阳、郭浩隆 | 建立版本变更表格、更正页码 |
| 1.2 | 2020/3/20 | 赵正阳 | 沈一聪、梁远志、宋冰晨、郭浩隆 | 更新组员分工 |
| 1.3 | 2020/3/21 | 宋冰晨 | 沈一聪、梁远志、赵正阳、郭浩隆 | 修正甘特图 |
| 1.4 | 2020/3/26 | 郭浩隆 | 沈一聪、梁远志、赵正阳、宋冰晨 | 添加项目目标、项目标准、修改Web UI计划、添加项目时间表 |
| 1.4.1 | 2020/4/2 | 赵正阳 | 沈一聪、梁远志、宋冰晨、郭浩隆 | 修改贡献率计算公式，增加主观评价 |

**目录**

[**1.** **项目简介** 1](#_Toc36159357)

[1.1 整体架构 1](#_Toc36159358)

[1.2 工作流程 2](#_Toc36159359)

[1.3 项目目标 2](#_Toc36159360)

[1.4 项目标准 2](#_Toc36159361)

[**2.** **组员介绍** 3](#_Toc36159362)

[2.1 组员技能 3](#_Toc36159363)

[2.2 组员分工 3](#_Toc36159364)

[2.3 协作方式 4](#_Toc36159365)

[**3.** **项目计划** 5](#_Toc36159366)

[3.1 整体计划 5](#_Toc36159367)

[3.2 扩展功能 5](#_Toc36159368)

[3.2.1 应对反爬虫 5](#_Toc36159369)

[3.2.2 分布式 11](#_Toc36159370)

[3.2.3 Web UI 12](#_Toc36159371)

[3.3 进度安排 15](#_Toc36159372)

[**4.** **组员贡献率计算方式** 17](#_Toc36159373)

[**5.** **参考文献** 19](#_Toc36159374)

1. **项目简介**

Scrapy是一个快速的、高层次的网络爬虫框架，用于爬取网页并提取结构化的数据，可用于数据挖掘、监控、自动化测试等多种用途。

使用Scrapy编写的爬虫程序简单、快速、易用；可以很方便地增加自定义扩展功能，而无需修改核心代码；该框架使用Python编写，可以很容易地在不同的操作系统上移植。基于以上原因，我们选择Scrapy作为我们的开源项目。

1.1 整体架构

图1.1是Scrapy官方文档中的整体架构图。



图1.1 Scrapy整体架构图

从整体架构图中可以看到，Scrapy框架包含以下核心组件：

（1）引擎(Scrapy Engine)：负责Spider、Item Pipeline、Downloader、Scheduler中间的通讯，信号、数据传递等；

（2）调度器(Scheduler)：负责接受引擎发送过来的Request请求，并按照一定的方式进行整理排列、入队；

（3）下载器(Downloader)：负责下载引擎发送的所有请求，并将其获取到的Responses交还给引擎；

（4）爬虫(Spider)：负责处理所有Responses，从中分析提取数据，并将需要跟进的URL提交给引擎，再次进入调度器；

（5）管道(Item Pipeline)：负责处理Spider中获取到的Item，并进行进行后期处理（详细分析、过滤、存储等）。

1.2 工作流程

在爬虫工作过程中数据流由引擎控制，主要包括以下几个步骤。

（1）引擎从爬虫获取初始请求；

（2）引擎将请求交给调度器，并获取下一个请求；

（3）调度器将下一个请求返回给引擎；

（4）引擎通过下载器中间件将请求发送给下载器；

（5）页面下载完成后，下载器会生成一个（带有该页面的）响应，并通过下载器中间件将其发送给引擎；

（6）引擎从下载器接收响应，并通过爬虫中间件将其发送到爬虫进行处理；

（7）爬虫处理响应，并通过爬虫中间件将抓取的项目和新的（要跟踪的）请求返回给引擎；

（8）引擎将处理后的项目发送到管道，然后将处理后的请求发送到调度器，并获取下一个请求。

（9）（从步骤1开始）重复该过程，直到不再有来自调度器的请求为止。

1.3 项目目标

基于scrapy框架，对其功能进行调研，分析可以改进或扩展的方向，并完成软件需求规格说明书的编写。根据需求规格说明书完成对scrapy框架的改进和功能扩展，并完成对项目的测试。最终分析、总结项目开发过程与软件管理过程。

1.4 项目标准

项目遵循如下推荐性国家标准：

* GB/T 8567-2006 计算机软件文档编制规范
* GB/T 9386-2008 计算机软件测试文档编制规范
* GB/T 9385-2008 计算机软件需求规格说明规范

1. **组员介绍**

2.1 组员技能

每个组员所来自的实验室、熟悉的语言以及掌握的技能如表2.1所示。

表2.1 组员技能介绍

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 实验室 | 熟悉的语言 | 技能 |
| 赵正阳（组长） | 软国重实验室 | Java, Python | Spring Boot, TensorFlow |
| 郭浩隆 | 软国重实验室 | Java, Python | Spring Boot, PyTorch |
| 沈一聪 | 软件工程研究所  操作系统实验室 | C++, Rust | 偏好操作系统 |
| 梁远志 | 软件工程研究所  操作系统实验室 | C | 系统软件、Web开发 |
| 宋冰晨 | 软国重实验室 | Python, JavaScript | Django |

2.2 组员分工

初步确定组员的大致分工如表2.2所示。

表2.2 组员大致分工

|  |  |
| --- | --- |
| 组员 | 分工 |
| 赵正阳 | 1. 原型爬虫程序的编写 2. Web UI扩展功能的编码和测试 3. 会议记录 4. 实验8——统计数据表格设计 |
| 郭浩隆 | 1. 分布式扩展功能的编码和测试 2. 实验8——数据记录 |
| 沈一聪 | 1. 应对反爬虫扩展功能的编码和测试 2. 评审意见记录 |
| 梁远志 | 1. 分布式扩展功能的编码和测试 2. 实验7——软件配置管理分析报告 |
| 宋冰晨 | 1. Web UI扩展功能的编码和测试 2. 实验6——项目进度控制 |

2.3 协作方式

（1）小组交流：每周五和周一晚7:30在微信群或腾讯会议开小组会，交流各自的进展，安排下一周的工作计划，有问题随时在微信群中讨论；

（2）协同工作：使用GitHub进行文档和代码管理，使用Microsoft Project 进行进度控制和统计。

1. **项目计划**

3.1 整体计划

项目的整体计划如下：

（1）了解Scrapy框架和使用方法及原理；

（2）使用Scrapy编写爬虫程序，阅读源代码，熟悉Scrapy框架的几个主要组件；

（3）针对某个网站编写完整的爬虫程序并分析性能；

（4）对爬虫程序进行改进和测试。

3.2 扩展功能

对于爬虫程序的改进，初步计划实现三个扩展功能：应对反爬虫、分布式和Web UI。

3.2.1 应对反爬虫

（1）Scrapy应对反爬虫的主要难点

Scrapy应对反爬虫的最主要难点在于，在对一些采用了动态JavaScript加载的机制较为复杂的网页进行爬虫时，往往力不从心，网站的登录首当其冲，成为一个主要的难题。

很多时候，我们在爬取数据时，有些页面的数据需要登录才能看到，因此需要登录再爬取数据。实现login再爬取的手段很多，其核心都是通过cookie的方式来记录身份信息，因此模拟登录的核心在于对cookie的使用（获取、保存和使用）。

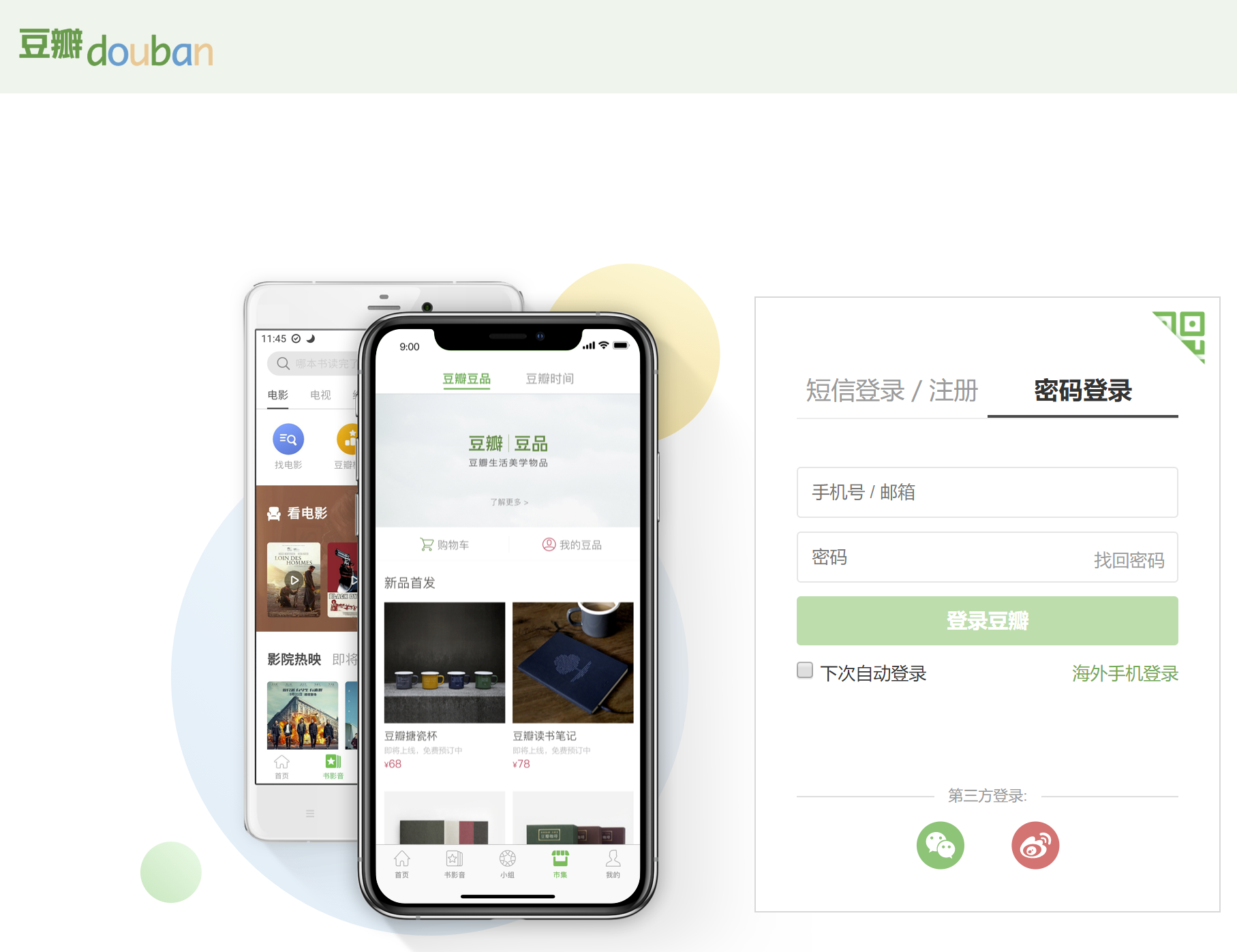


图3.1 豆瓣网页版的登录页面

可选的模拟登陆的方法有许多，其中最简单的是通过分析登录页面，获取登录接口和提交表单进行登录验证（可以用简单的Request库，结合cookielib方式或者Scrapy等）。然而，随着各个知名网站中反爬虫机制的不断升级，登录接口变得越来越复杂，大多都加入动态加载JavaScript的方法，单纯的使用上述方法，其实施难度在不断增加。对此，我们可以考虑采用Selenium方式来登录，同时通过session记录登录后的cookies，进一步对其他内的内容进行爬取。

（2）Selenium介绍

Selenium是最广泛使用的开源Web UI（用户界面）自动化测试套件之一。它最初由Jason Huggins于2004年开发，作为Thought Works的内部工具。Selenium支持跨不同浏览器，平台和编程语言的自动化。

Selenium可以轻松部署在Windows、Linux、Solaris和Macintosh等平台上。 此外，它支持iOS（iOS，Windows Mobile和Android）等移动应用程序的OS（操作系统）。

Selenium通过使用特定于每种语言的驱动程序支持各种编程语言。Selenium支持的语言包括C#、Java、Perl、PHP、Python和Ruby。目前，Selenium Web驱动程序最受Java和C#欢迎。Selenium测试脚本可以使用任何支持的编程语言进行编码，并且可以直接在大多数现代Web浏览器中运行。Selenium支持的浏览器包括Internet Explorer，Mozilla Firefox，Google Chrome和Safari。Selenium的基本特点如图3.2所示。

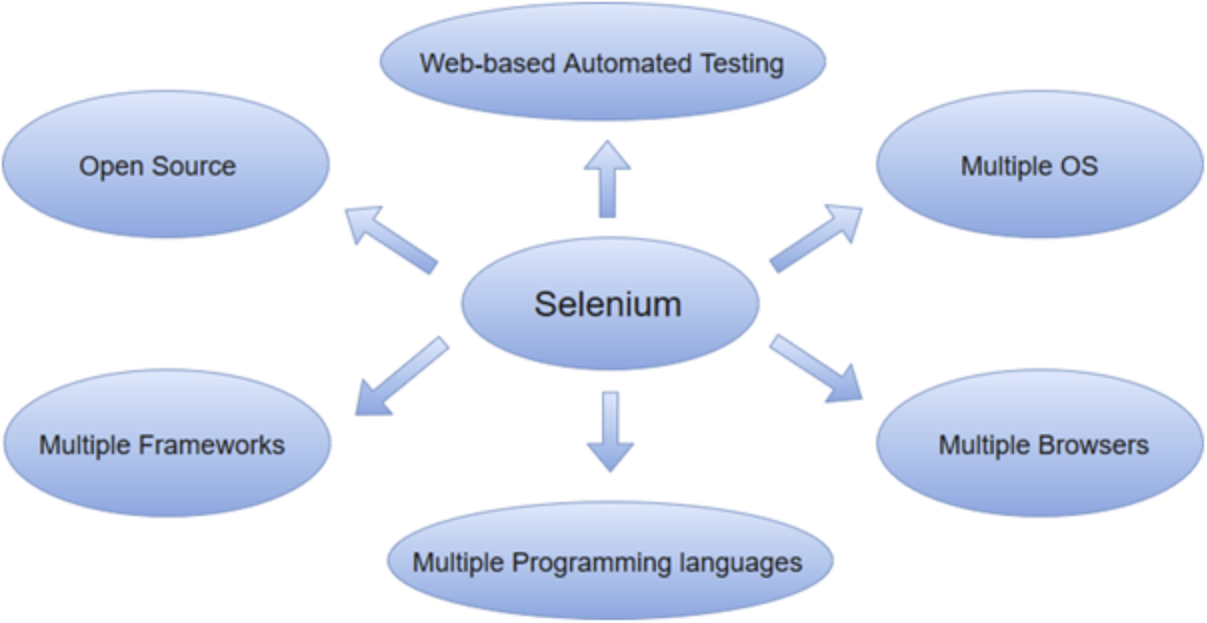


图3.2 Selenium相关特性

简单来说，Selenium的主要功能可以归纳为以下四点：

* 测试与浏览器的兼容性——测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上。
* 测试系统功能——创建回归测试检验软件功能和用户需求。
* 支持自动录制动作和自动生成.Net、Java、Perl、Python等不同语言的测试脚本。
* 爬虫，可以模拟人点击鼠标的过程，防止被发现是爬虫程序，缺点是速度较慢。

（3）Scrapy + Selenium的原理和方法

对于那些充斥着大量复杂JavaScript渲染的页面，请求非常难构造，或者构造方式经常发生变化，这种情况下采用Selenium直接模拟请求的方法，极大的减少了爬虫的项目难度和工作量。然而，若是所有的网页无论静态动态都采用Selenium来完成，整体效率又会变得十分低下。所以我们需要结合两者的优势，对于拥有大量静态元素的网页，直接采用Scrapy来完成爬虫；对于网页的登录界面，或者是拥有大量动态JS加载的网页，Selenium直截了当的风格成为我们的首选。

从图1中的Scrapy工作过程中来看，突破口就在下载中间件部分，可以用Selenium直接处理掉request请求，即部分取代Downloader的功能。如果只是为了应对网页登录，则只需要再利用Selenium完成登录之后，将cookies写入并保存成JSON文件以便之后调用即可。

（4）Scrapy + Selenium反爬虫的缺陷和改进方法

1. 浏览器相关的处理细节

因为用Scrapy + Selenium的方式，只有部分，甚至是一小部分页面会用到浏览器，而把chrome放到Middleware中具有诸多限制，所以可以考虑将浏览器相关代码转移到spider中实现。这样做的好处在于，项目中的每一个spider都有自己的浏览器，这样当启动多个spider时，就会存在多个浏览器，而并非所有的spider共用一个chrome，这对并发以及之后的分布式功能实现是有好处的。

1. Selenium的效率问题

同步Selenium效率低，可能需要结合Twisted形成异步方案以取代原本的同步方案。

1. Selenium本身的反爬虫应对

使用Selenium模拟浏览器进行数据抓取无疑是当下最通用的数据采集方案，它通吃各种数据加载方式，能够绕过客户JS加密，绕过爬虫检测，绕过签名机制。它的应用，使得许多网站的反采集策略形同虚设。由于Selenium不会在HTTP请求数据中留下指纹，因此无法被网站直接识别和拦截。

这是不是就意味着Selenium真的就无法被网站屏蔽了呢？非也。Selenium在运行的时候会暴露出一些预定义的JavaScript变量（特征字符串），例如"window.navigator.webdriver"，在非Selenium环境下其值为undefined，而在Selenium环境下，其值为true。

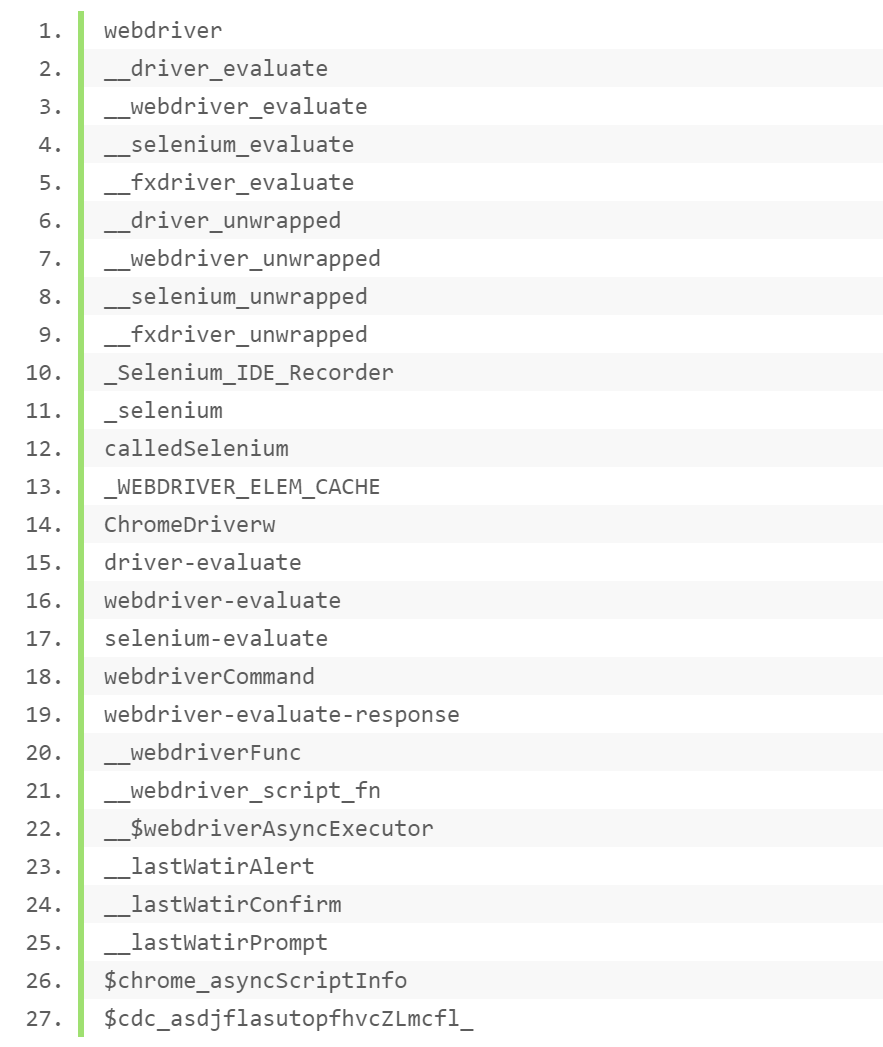


图3.3 Selenium的一些标志性字符串

除此之外，还有一些其它的标志性字符串（不同的浏览器可能会有所不同），常见的特征串如图3.3所示。

了解了这个特点之后，就可以在浏览器客户端JS中通过检测这些特征串来判断当前是否使用了Selenium，并将检测结果附加到后续请求之中，这样服务端就能识别并拦截后续的请求。

（5）反爬虫的其他难点的应对



图3.4 爬虫和反爬虫对抗史

图3.4展示了爬虫以及反爬虫技术的对抗史。其场景以及解决方案归纳如下：

1. IP地址验证

场景：有些网站通过IP地址来识别是请求否是都来源于同一个客户端。

解决方案：不断地随机更换代理服务器的IP 地址。middlewares.py通过自定义的下载中间件为Scrapy设置了代理服务器。需要开发者事先准备好一系列代理服务器。settings.py文件设置启用自定义的下载中间件。

1. 禁用Cookie

场景：有些网站通过跟踪 Cookie 来识别是否是同一个客户端。

解决方案：在配置文件中关闭默认开启的cookie选项COOKIES\_ENABLED = False。

1. 违反爬虫规则文件

场景：有些网站robots.txt 文件，制定了爬虫规则。

解决方案：指定不遵守爬虫规则，ROBOTSTXT OBEY = False。

1. 限制访问频率

场景：当同一个 IP 地址、同一个客户端访问目标网站过于频繁时，很可能会被当成机器人。

解决方案：为了更好地模拟正常用户的访问速度，可以限制 Scrapy 的访问频率。比如开启频率限制，设置访问开始延迟，访问之间最大延迟，并行每台服务器请求数量，下载后的自动延迟。

1. 图形验证码

场景：为了防止机器程序访问，对同客户同IP达到一定访问次数，会要求输入图形验证码。

解决方案：让机器识别验证码。使用PIL、Libsvrn 等第三方库，自己开发程序来识别图形验证码。或是通过第三方图形验证码的在线识别网站进行识别，但识别率高的背后往往需要额外的收费。

3.2.2 分布式

Scrapy没有用于在分布式（多服务器）进行爬虫的内建设施。但是常规操作中有许多方法来进行分布式爬虫，而这些方法取决于编程者打算怎样去分布这些爬虫任务。一种情况是有许多的爬虫任务（Scrapy的Spider），显然的分布负载的方式是设置许多个Scrapyd实例，然后在这些实例上分布运行这些Spider。而另一种情况是如果需要将一个大的单独的爬虫任务分布到不同的机器上运行，则需要先准备一个需要爬取的URL的列表，然后将这个列表分成若干个部分文件，再通过提供给多个Scrapyd实例的不同的参数（用于标识哪一个部分），将任务分布在不同的机器上运行。

分布式爬虫具有如下的优点：

1. 能够提高整体的爬虫效率；
2. 多个服务器具有多个IP地址，能够减轻针对IP地址的反爬虫机制的影响；
3. 能够具有节点容错性，当一个爬虫服务器失效时也能够进行爬虫。

本项目也计划选择Scrapyd作为部署的软件，而不对Scrapyd做修改。项目的关注点在于编写一个用于调度（Schedule）这些Scrapyd实例的调度程序，并为这个调度程序提供一个用户操作的Web为基础的界面，提供爬虫过程的可视化以及爬虫结果的输出与可视化。对于用户而言，用户并不需要知道爬虫任务如何在多个Scrapyd实例上分布。本项目的分布式爬虫的示意图如下图所示。



图3.5 分布式爬虫

本项目的分布式爬虫部分的关注点在于怎么将用户输入切分成若干个爬虫（或者同一个具有不同参数的爬虫），进而将这些爬虫以一种负载均衡的方式分布到不同的Scrapyd实例运行，同时将不同Scrapyd实例产生的爬虫结果数据合并存储，最终将结果数据输出给用户。这一核心过程有这样几个关键点：

1. 将用户的输入转化为针对一个或者多个具有不同参数的爬虫程序；
2. 调度程序接受这些任务，并实现一种负载均衡的算法，通过调用Scrapyd的API向各个Scrapyd实例分配任务；
3. 调度程序调用Scrapyd的API获取各个实例的工作状态（性能以及进度）信息，并通过可视化的方式反馈给用户；
4. 在Scrapyd实例上，对爬取到的数据进行持久化存储；
5. 调度程序对Scrapyd实例存储的数据进行合并，并输出到用户界面。

3.2.3 Web UI

由于Scrapy是通过命令行来执行的，同时对爬虫进行不同的操作需要不同的命令，这就使得对爬虫的操作十分繁琐；因此我们决定为Scrapy添加一个可视化界面，使得我们能够很方便的监控和操作爬虫。同时可视化界面也极大简化了对于分布式爬虫的管理。我们调研了目前一些开源的scrapy可视化项目如scrapyWeb、Gerapy、SpiderKeeper等，它们均能够提供部署爬虫，定时执行爬虫，查看爬虫日志，查看爬虫执行情况等功能。我们根据调研的结果提出了本项目的Web UI开发计划。

本项目计划提供给用户根据系统内现有爬虫模板来爬取特定网站的功能，并能够管理自己的爬虫任务。同时管理员能够对爬虫模板进行管理。因此Web UI需要向用户和管理员展示以下几个功能：

1. 浏览爬虫列表

向用户展示系统内当前的爬虫模板列表，向用户展示模板样例，并将模板按类别进行分类，同时提供给用户按关键字检索的功能。

1. 管理爬虫任务

能够让用户对自己的爬虫进行管理，如添加爬虫任务、查看爬虫运行进度、启动爬虫任务、暂停爬虫任务、终止爬虫任务等。同时为爬虫任务选择模板和进行配置。

1. 管理爬虫结果

用户查看爬虫爬取的结果，并能够将爬取结果保存到本地。

1. 管理爬虫模板

管理员可以对系统内的爬虫模板进行管理，包括添加爬虫模板、删除爬虫模板和更新爬虫模板。

我们对目前scrapy可视化项目的调研情况如下：

（1）ScrapydWeb

ScrapydWeb是一个基于Scrapyd的可视化组件，可以用于Scrapyd集群管理、支持Scrapy日志分析和可视化。它集成并且提供更多了的可视化功能和更加优美的画面。

它包括以下几种功能：爬虫集群管理、爬虫任务管理、项目部署、爬虫启动、项目管理、日志分析。

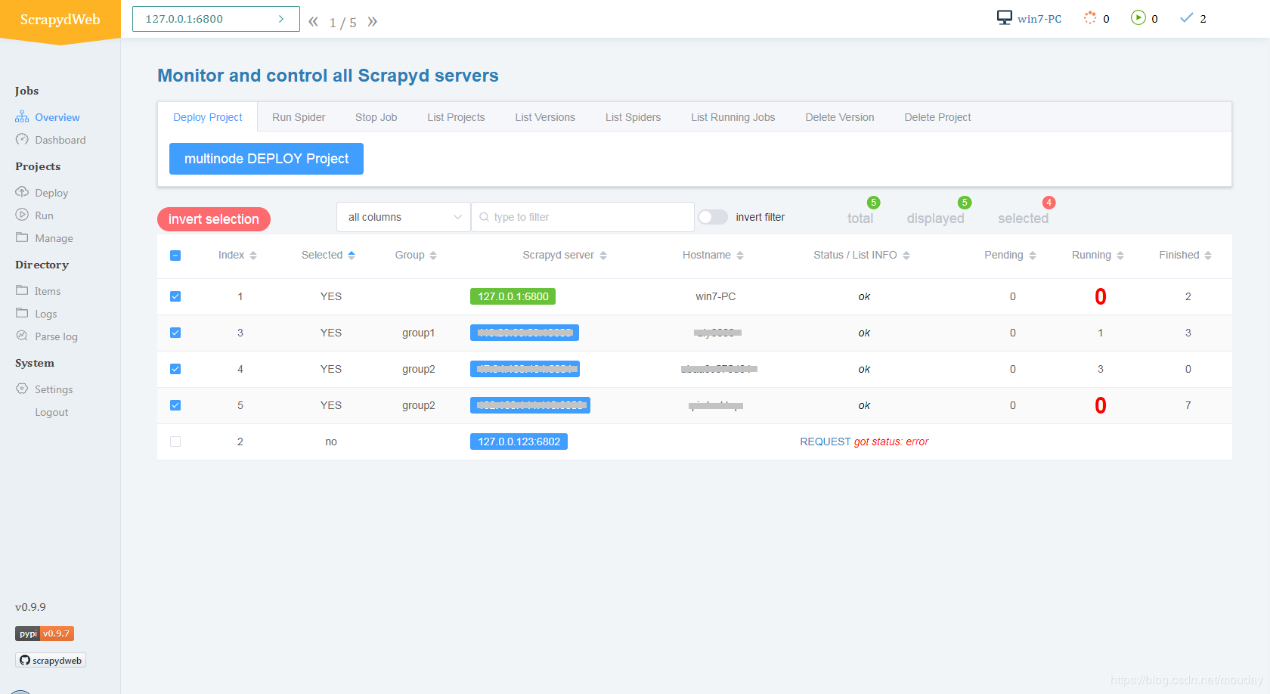


图3.7 ScrapydWeb界面

（2）Gerapy

Gerapy是一款分布式的爬虫管理框架，支持Python 3，基于Scrapy、Scrapyd、Scrapy-client开发，它包括以下几种功能：控制爬虫运行、监控爬虫状态、实时查看爬取结果、实现项目的部署、进行统一的主机管理、更便捷的生成爬虫代码。

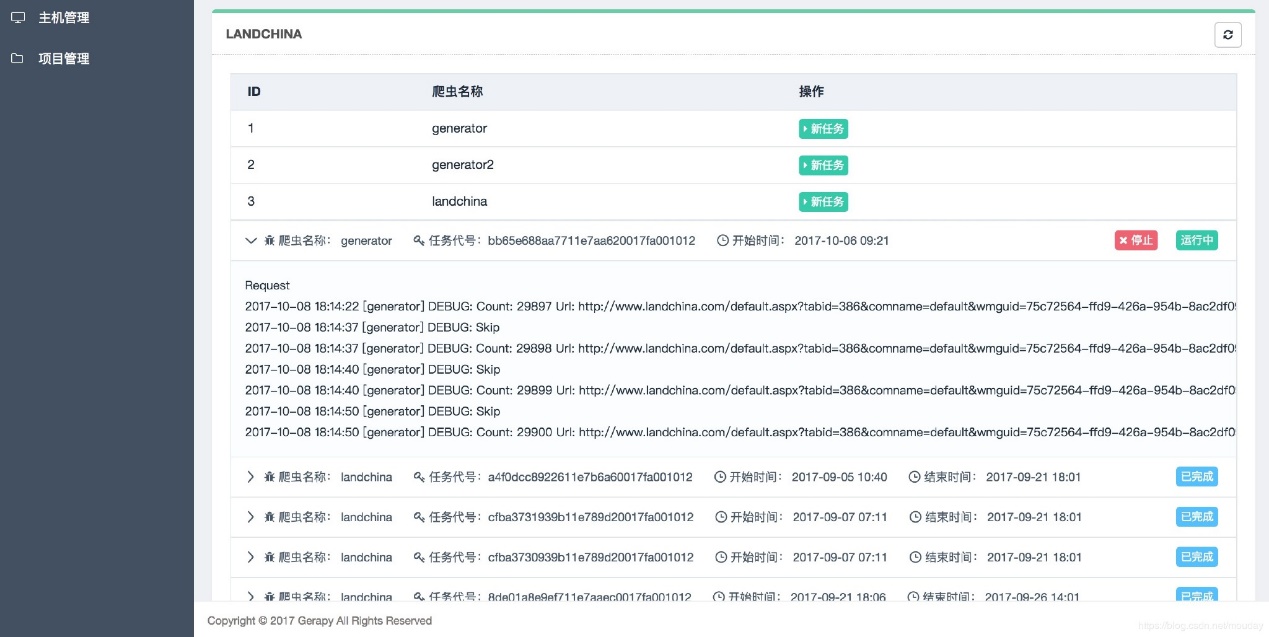


图3.8 Gerapy界面

（3）SpiderKeeper

SpiderKeeper是一款管理爬虫的软件，和上述几个可视化工具功能类似，能多台服务器部署爬虫，定时执行爬虫，查看爬虫日志，查看爬虫执行情况等。但是不足之处在于对于爬虫的爬取情况需要通过查看日志，因此爬虫的详细情况需要通过记录日志来实现。

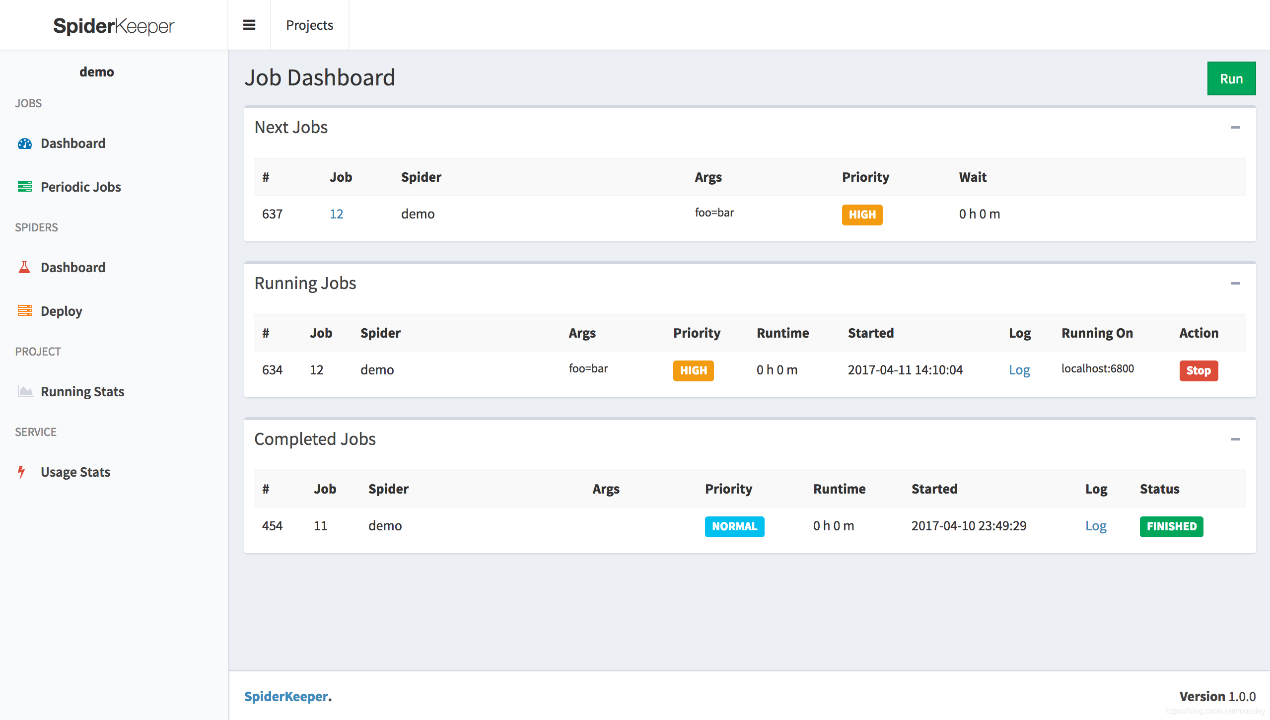


图3.9 SpiderKeeper界面

3.3 进度安排

|  |  |
| --- | --- |
| **主要工作计划** | **预计完成周数** |
| 完成实验的准备，包括项目目标、计划安排等 | 2 |
| 完成软件需求分析 | 3-4 |
| 完成软件需求的评审与复审 | 5-6 |
| 完成项目的管理、配置与分析 | 7 |
| 软件产品的实现与改进 | 8-10 |
| 实现反爬虫、分布式、可视化三个主要改进 | 8-9 |
| 功能改进的整合与封装 | 10 |
| 软件测试需求分析 | 11 |
| 软件测试评审、复审 | 12-13 |
| 软件测试演示与测评 | 14 |
| 软件项目管理、配置管理分析与项目总结 | 15-16 |

项目计划时间表

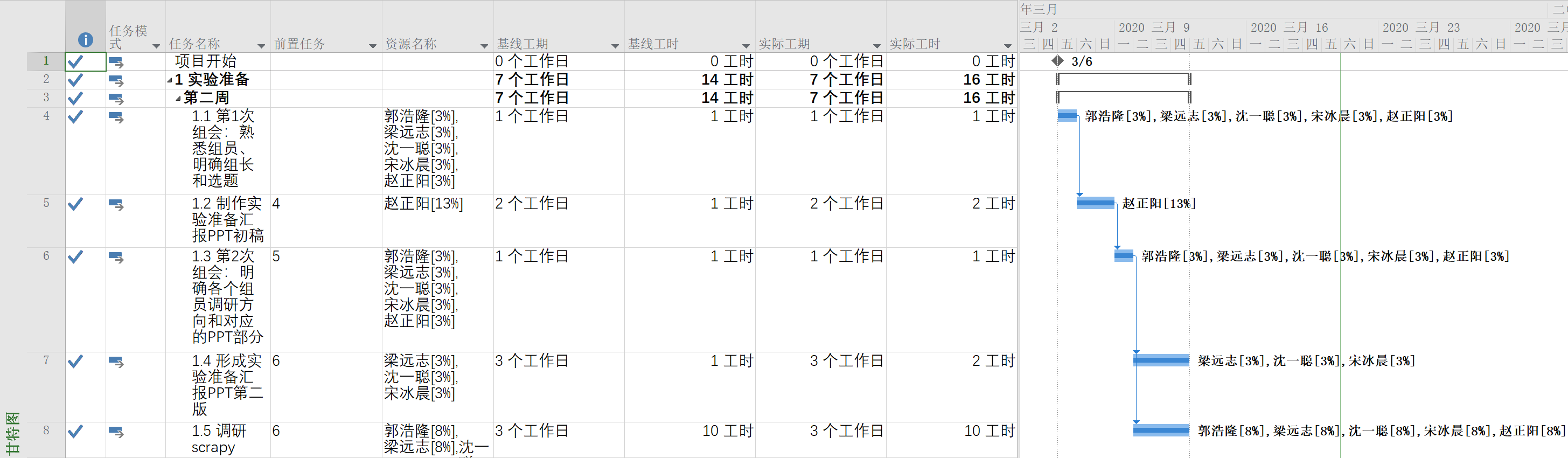


图3.10 进度安排甘特图

1. **组员贡献率计算方式**

组员贡献率包括工作量比例和互评得分两部分：

其中工作量比例由客观数据统计得出，互评得分由其他组员对本人的平均评分得出，暂定*w*1 = 0.8, *w*2 = 0.2。

软件工程综合实验包含了实验准备和实验1~8一共9大任务，组员的客观工作量数据由9个任务所包含文档、制图、开发和其他工作总计得出。

（1）文档类

说明：文档的撰写、修订

涉及任务：9个任务

计算指标：字数

（2）制图类

说明：文档所必需的各类制图

涉及任务：9个任务

计算指标：拟定一个图折算为200字

（3）开发类

说明：软件实现、数据统计环节所必需的代码

涉及任务：实验3软件设计与实现，实验6-8在统计数据时可能也会涉及到代码开发

要求：组员衡量缩写代码的难度，提供0-1的一个难度估计值

计算指标：拟定一行代码折算为10 \* (1 + 难度)字

（4）其他

说明：其他难以折算为字数的工作，比如文档的整合排版、MS Project文档的维护、调研等

涉及任务：9个任务

计算指标：工时

拟定按照表4.1进行各个任务的工作量统计。对于文档类、制图类、开发类工作，个人工作量的计算指标是字数，其中需要将制图和代码折算为字数后和文档字数进行累加。对于其他类工作，个人工作量的计算指标是工时，其中需要将各项其他类工作耗费的工时进行累加。因此，个人工作量比例的计算公式如下：

其中暂定*w*1 = *w*2 = 0.5。

表4.1 工作量统计表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 代码行数 | 代码难度  （0-1） | 文档字数  （W-字 P-图） | 其他 | | 贡献率 |
| 工作内容简述 | 耗费工时 |
| 组员1 |  |  |  |  |  |  |
| 组员2 |  |  |  |  |  |  |
| … |  |  |  |  |  |  |

1. **参考文献**

[1] <https://scrapy.org/>

[2] <https://docs.scrapy.org/en/latest/index.html>

[3] <https://pypi.org/project/selenium/>

[4] 李培.基于Python的网络爬虫与反爬虫技术研究[J].计算机与数字工程,2019,47(06):1415-1420+1496.

[5] 余豪士,匡芳君.基于Python的反反爬虫技术分析与应用[J].智能计算机与应用,2018,8(04):112-115.

[6] <https://blog.csdn.net/mouday/article/details/84926296>