Scrapy 需求规格说明书

[V1.90]

编写	武丁泽宇	日期	2017 年 5 月 11 日
校对	郭炜锋	日期	2017 年 5 月 11 日

北京航空航天大学计算机学院

二〇一七年三月二十九日

文档修改记录

版本号	日期	所修改章节	修改说明	修改人	审核人
1.00	2017.3.21		完成第一版	武丁泽宇	胡勇
1.10	2017.3.29	4	修改第 4 章内容	武丁泽宇	胡勇
1.11	2017.3.29	4 、5	修改 4 章内容,删除 5 章	武丁泽宇	胡勇
1.20	2017.4.4	1、2 、3、 4	按照修改建议进行修订	武丁泽宇	胡勇
1.30	2017.4.5	3	完善业务需求和用例图	武丁泽宇	胡勇
1.40	2017.4.9	全文	排版修改	胡勇	胡勇
1.50	2017.4.16	1.3、全文字体、全文格式	将 1.3 定义使用表格表述，将正文的格式统一	郭炜锋	胡勇
1.60	2017.4.23	全文修改	根据老师批注进行修改	武丁泽宇	胡勇
1.70	2017.4.26	全文修改	根据各组评审意见进行修改和调整	武丁泽宇	胡勇
1.75	2017.5.3	批注章节修改增加第 5 章	按照批注进行修改完善	武丁泽宇	胡勇
1.80	2017.5.4	全文	格式化文本格式	胡勇	胡勇
1.90	2017.5.11	全文	修改 RUCM 和用例图	武丁泽宇	胡勇

目录

1 引言.....5
1.1 编写目的.....5
1.2 背景.....5
1.3 定义.....5
1.4 参考资料.....7
2 任务概述.....7
2.1 目标.....7
2.2 用户特点.....7
2.3 假定与约束.....7
3 需求与设计.....7
3.1 需求.....8
3.1.1 业务需求.....8
3.1.2 功能需求.....9
3.1.3 非功能性需求.....9
3.2 框架及组件概述.....10
3.3 用例图.....12
3.4 功能模块补充描述.....16
3.4.1 Scrapy Engine.....17
3.4.2 Spiders.....18
3.4.3 设置（Settings）.....18
3.4.4 Item Pipeline.....19
3.4.5 下载器中间件.....19
3.4.6 Spider 中间件.....19
3.5 扩展功能模块.....19
3.5.1 日志（Logging）.....19
3.5.2 页面服务（Web Service）.....19
3.5.3 Item Exporters.....20
3.5.4 自动限速(AutoThrottle)扩展.....20

3.5.5 自定义扩展.....20
3.6 故障处理要求.....21
3.7 其他专门要求.....22
4 运行环境规定.....22
4.1 设备.....22
4.2 支持软件.....22
4.3 接口.....22
4.3.1 硬件接口.....22
4.3.2 软件接口.....22
4.3.3 通信接口.....22
4.3.4 用户接口.....23
5 23
5.1 方案思路.....23
5.2 详细设计.....23
5.2.1 搭建测试环境.....23
5.2.2 反爬改进.....24
5.2.3 下载器改进.....24

1 引言

本软件 Scrapy 是开源爬虫框架。本文详细描述了 Scrapy 的功能、非功能性、业务、拓展和改进需求。

1.1 编写目的

本软件需求规格说明书，是为软件设计、软件测试人员和用户编写的。

本软件需求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 239 位贡献者

项目背景：本项目是用于开发一个高速并发的网络爬虫的框架，用于爬取网站的数据信息并导出其数据结构。

1.3 定义

术语	说明
爬虫	具有抓取网页内容功能的软件
爬虫引擎(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。
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。
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。
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。
Item Pipeline	Item Pipeline 负责处理被 spider 提取出来的 item。
下载器中间件(Downloader	下载器中间件是在引擎及下载器之间的特

middlewares)	定钩子(specific hook)，处理 Downloader 传递给引擎的 response。
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)。
数据流(Data flow)	Scrapy 中的数据流。
事件驱动网络(Event-driven networking)	Scrapy 基于事件驱动网络框架 Twisted 编写。
XML	可扩展标记语言，标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。
JSON	存储和交换文本信息的语法。类似 XML
CSV	逗号分隔值，文件以纯文本形式存储表格数据（数字和文本）。纯文本意味着该文件是一个字符序列，不含必须像二进制数字那样被解读的数据。
XPath	即 XML 路径语言（XML Path Language），它是一种用来确定 XML 文档中某部分位置的语言。XPath 基于 XML 的树状结构，提供在数据结构树中找寻节点的能力。
CSS	层叠样式表，又称串样式列表、级联样式表、串接样式表、层叠样式表、階層式樣式表，一种用来为结构化文档（如 HTML 文档或 XML 应用）添加样式（字体、间距和颜色等）的计算机语言，由 W3C 定义和维护。
Telnet	Telnet 协议是一种应用层协议，使用于互联网及局域网中，使用虚拟终端机的形式，提供双向、以文字字符串为主的交互功能。
cookies	指某些网站为了辨别用户身份而储存在用

	户本地终端上的数据（通常经过加密）。
session	会话（session）是一种持久网络协议，在用户（或用户代理）端和服务器端之间建立关联，从而起到交换数据包的作用机制。
DNS	互联网的一项服务。 它作为将域名和 IP 地址相互映射的一个分布式数据库，能够使人更方便地访问互联网。

1.4 参考资料

GJB 438B 国军标开发通用文档。

2 任务概述

2.1 目标

Scrapy 的出现是为了构建一整套便捷高效的爬取框架，便于进行站点数据爬取，提取结构性数据。可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。任何人都可以根据需求方便地修改框架内容。并且框架提供了多种类型爬虫的基类，同时使用 Twisted 这个异步网络库来处理网络通讯。构建的架构清晰，包含了各种中间件接口，可以灵活的完成各种需求。使用 Python 语言，以便达到简介高效的目的。

2.2 用户特点

熟悉网页抓取和 Python 的程序开发用户。

2.3 假定与约束

无

3 需求与设计

3.1 需求

3.1.1 业务需求

典型业务需求场景：页面数据抓取

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

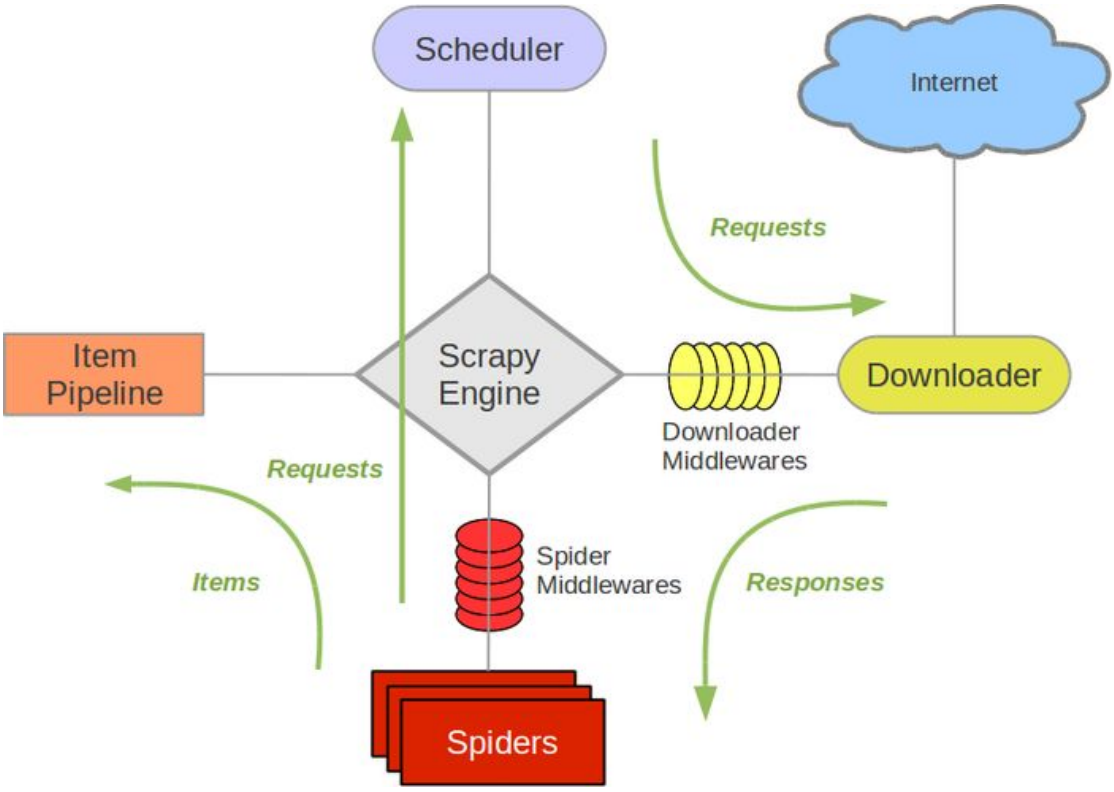


图 1 Scrapy 框架图

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)

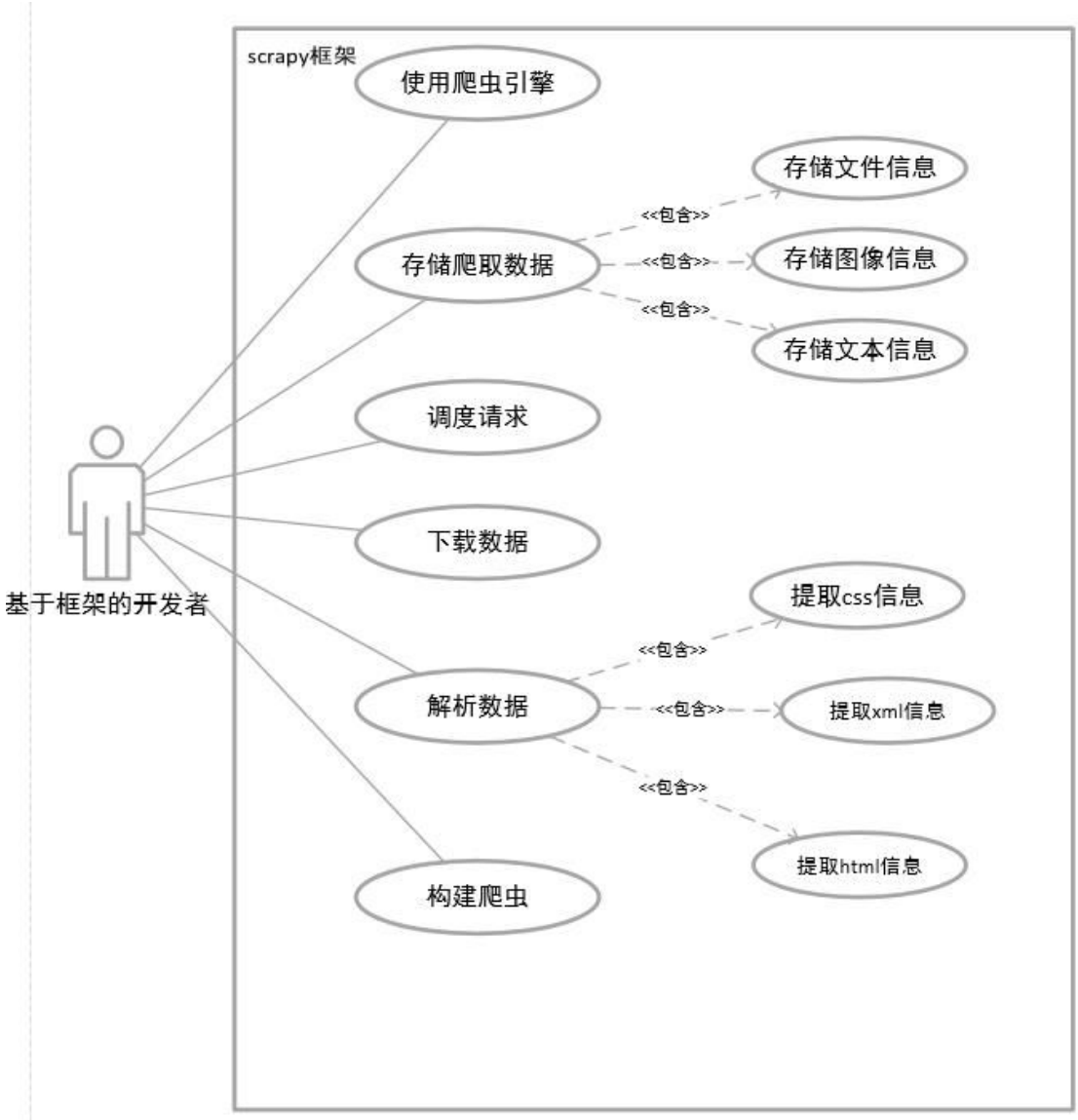
下载器中间件是在引擎及下载器之间的特定钩子(specific hook)，处理 Downloader 传递给引擎的 response。 其提供了一个简便的机制，通过插入自定义代码来扩展 Scrapy 功能 。功能

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

在实际环境中，开发者和使用者的身份一般重叠，但划分为两类，使得用例图更清晰。Scrapy 框架的用例图如图 2 所示。


scrapy框架
使用爬虫引擎
存储文件信息
<<包含>>
存储爬取数据
<<包含>>
存储图像信息
<<包含>>
存储文本信息
调度请求
下载数据
基于框架的开发者
提取css信息
<<包含>>
解析数据
<<包含>>
提取xml信息
<<包含>>
提取html信息
构建爬虫

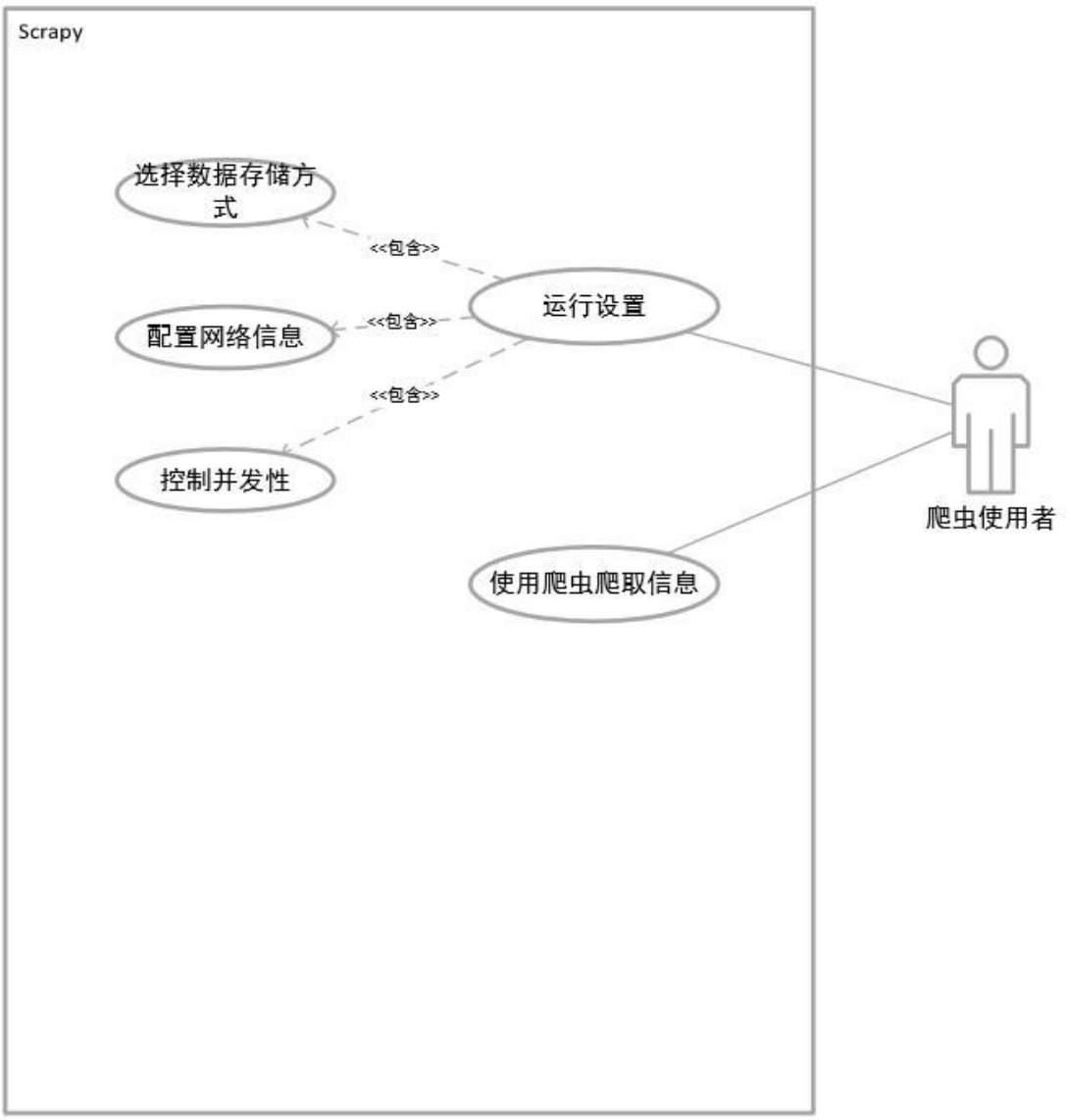


图 2 Scrapy 用例图

在主要的业务场景–页面数据抓取中，使用到的 scrapy 核心功能模块有：spider、设置、数据存储、下载器、调度器和 scrapy engine。所以我们对这些模块画出 RUCM 图依次如下：Spider 模块 RUCM 图如图 3 所示，设置模块 RUCM 如图 4 所示，数据存储模块 RUCM 如图 5 所示，下载器模块 RUCM 如图 6 所示，调度器模块 RUCM 如图 7 所示, scrapy engine 模块 RUCM 如图 8 所示。

Use Case Specification	
Use Case Name	spider
Brief Description	爬虫开发者构建爬虫软件；爬虫使用者爬取数据。
Precondition	爬虫使用者启动爬虫。
Primary Actor	初始化爬虫引擎、过滤器、调度器
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 由起始URL生成起始Request。
	2 爬虫引擎将生成的请求交给到调度器。
	3 调度器调度下载器下载Request。
	4 下载器下载页面数据。
	5 爬虫引擎将HTML原始页面交给spider。
	6 spider从原始HTML页面中抽取出URL和数据。
	7 spider生成新的Request和Item。
	8 爬虫引擎将Request传给调度器.
	9 爬虫引擎将Item传给数据处理管道.
	Postcondition 没有新的原始HTML页面传给spider

Specific Alternative Flow (Untitled) ▼	RFS 1
	1 在Request中添加cookie和登录信息。
	2 爬虫引擎将请求交给调度器。
	3 调度器调度下载器下载该请求页面。
	4 下载器下载页面。
	Postcondition 登录成功，cookie消息存放类在请求类中注册成功

Specific Alternative Flow (Untitled) ▼	RFS 1,2,3,4
	1 进行异常的日志记录。
	Postcondition 日志记录完成

图 3 Spider 模块 RUCM 图

Use Case Specification	
Use Case Name	运行设置
Brief Description	设置爬虫在爬取过程中的一些参数，包括并行数、日志级别等
Precondition	爬虫爬取逻辑部分已由代码实现完成
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 用户通过配置文件定义爬虫的一些参数。
	2 启动爬虫。
	3 初始化爬虫引擎、调度器、过滤器、爬虫实例。
	Postcondition 爬虫启动成功

Specific Alternative Flow (Untitled) ▼	RFS 1
	1 爬虫开发者或者爬虫使用者调整参数。
	2 启动爬虫
	Postcondition 爬虫启动成功

图 4 设置模块 RUCM 图

Use Case Specification

Use Case Name	数据存储
Brief Description	spider解析的实体通过实体管道的处理获取的数据进行存储
Precondition	spider将解析出的实体交付给实体管道
Primary Actor	实体管道
Secondary Actors	Spider
Dependency	包含文件、图片、文字
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 读取从spider获取的实体
	2 删除不需要的信息
	3 根据需要的格式将数据进行格式化处理
	4 将处理后的信息存储到数据库中
	Postcondition 数据以正确的格式存储到数据库中

Specific Alternative Flow (Untitled) ▼	RFS 3
	1 实体中没有找到所需数据
	Postcondition 返回空数据

Specific Alternative Flow (Untitled) ▼	RFS 4
	1 数据库操作出现异常
	Postcondition 显示数据库异常信息

图 5 数据存储模块 RUCM 图

Use Case Name	下载数据
Brief Description	本用例只要是根据引擎产生的url，去抓取网页信息，将至反馈给爬虫
Precondition	爬虫引擎将请求发送给下载器
Primary Actor	爬虫引擎
Secondary Actors	下载器
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 引擎根据调度返回的url将请求信息发送给下载器
	2 下载器根据request信息，传递给下载中间件
	3 下载中间件封装request请求，下载相应数据
	4 下载中间件将下载的数据反馈给下载器
	5 下载器将下载的数据反馈给爬虫引擎
	Postcondition 完成一次下载请求

Specific Alternative Flow (Untitled) ▼	RFS 4
	1 request请求返回错误状态码
	Postcondition 目标网页无法爬取

图 6 下载器模块 RUCM 图

Use Case Name	调度请求（schedule）
Brief Description	获得爬虫引擎提供的请求url，进行相应的排序，并将需要爬取的请求目标反馈给引擎模块
Precondition	在spider中存在有效的url，引擎进行调度请求
Primary Actor	spider
Secondary Actors	engine
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 爬虫引擎向调度器发送调度请求，询问下一个爬取目标
	2 调度器查询维护的url集合，选取合适的url
	3 调度器将下一个请求反馈给爬虫引擎
	Postcondition 完成一次调度请求

Specific Alternative Flow (Untitled) ▼	RFS 3
	1 调度器中的请求队列为空
	Postcondition 爬虫完成所有爬取任务，调度结束

Specific Alternative Flow (Untitled) ▼	RFS 4,5
	1 调度请求无法正常反馈引擎对应信息
	Postcondition 异常提醒

图 7 调度器 模块 RUCM 图

Use Case Specification

Use Case Name	Scrapy Engine
Brief Description	爬虫引擎，从调度器中读取URL，生成请求给下载器进行网页下载
Precondition	Spider中有可以爬取的URL
Primary Actor	爬虫开发者
Secondary Actors	调度器
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps
	1 引擎打开一个网站，找到处理该网站的Spider并向该Spider请求第一个要爬取的URL
	2 引擎从Spider中获取到需要爬取的URL，在调度器以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通过下载中间件转发给下载器
	5 下载器生成一个该页面的Response，并将其通过下载中间件返回发送给引擎
	6 引擎从下载器中接收到Response并通过Spider中间件发送给Spider处理
	7 Spider处理Response并返回爬取到的Item及新的Request给引擎
	8 引擎将爬取到的Item给Item Pipeline，将Request给调度器
	Postcondition 调度器中没有新的URL

Specific Alternative Flow (Untitled) ▼	RFS 2,8
	1 调度器中没有可用的URL
	Postcondition 爬虫爬取完成，引擎关闭网站

Specific Alternative Flow (Untitled) ▼	RFS 4,5
	1 封装的请求由于某种原因而获取不到信息,Response返回相应信息
	Postcondition Response返回异常数据

图 8 scrapy engine 模块 RUCM 图

3.4 功能模块补充描述

对应于前面画出的各个 Scrapy 核心功能模块 RUCM 图，我们给出 scrapy engine 和 spider 的更加详细补充描述，以表述 RUCM 未能展现的部分。由于在实际使用下载器、spider 和数据存储时，需要经过它们的中间件传输数据，所以我们也将各个中间件的功能用文字描述出来。他们作为各个模块的一部分存在，因而没有使用 RUCM 图描述。

3.4.1 Scrapy Engine

Scrapy 中的数据流由执行引擎 Scrapy Engine 控制，引擎控制的数据传输过程如下：

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。

3.4.2 Spiders

Spider 类定义了如何爬取某个(或某些)网站。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)。 换句话说，Spider 就是定义爬取的动作及分析某个网页(或者是有些网页)的模块。

对 spider 来说，爬取的循环过程如下:

1. 以初始的 URL 初始化 Request，并设置回调函数。 当该 request 下载完毕并返回时，将生成 response，并作为参数传给该回调函数。spider 中初始的 request 是通过调用 start_requests() 来获取。 start_requests() 读取 start_urls 中的 URL， 并以 parse 为回调函数生成 Request 。
2. 在回调函数内分析返回的(网页)内容，返回 Item 对象、dict、 Request 或者一个包括三者的可迭代容器。 返回的 Request 对象之后会经过 Scrapy 处理，下载相应的内容，并调用设置的 callback 函数(函数可相同)。
3. 在回调函数内，可以使用 选择器(Selectors) (也可以使用 BeautifulSoup, lxml 或者想用的任何解析器) 来分析网页内容，并根据分析的数据生成 item。
4. 最后，由 spider 返回的 item 将被存到数据库(由某些 Item Pipeline 处理)或使用 Feed exports 存入到文件中。

3.4.3 设置（Settings）

Scrapy 设定(settings)提供了定制 Scrapy 组件的方法。可以控制包括核心(core)，

插件(extension)，pipeline 及 spider 组件。设定为代码提供了提取以 key-value 映射的配置值的的全局命名空间(namespace)。 设定可以通过多种方式设置，每个方式具有不同的优先级。

3.4.4 Item Pipeline

当 Item 在 Spider 中被收集之后，它将会被传递到 Item Pipeline，按照先后顺序执行对 Item 的处理。

3.4.5 下载器中间件

下载器中间件是介于 Scrapy 的 request/response 处理的钩子框架。 是用于全局修改 Scrapy request 和 response 的一个轻量、底层的系统。

3.4.6 Spider 中间件

Spider 中间件是介入到 Scrapy 的 spider 处理机制的钩子框架，可以添加代码来处理发送给 Spiders 的 response 及 spider 产生的 item 和 request。

3.5 扩展功能模块

由于实际使用中我们可能还会需要用到日志、页面服务、Item Exporters、自动限速设置和自定义扩展这几个扩展模块以便编写实际的爬虫或者调试程序，因此我们对这些模块功能也给出了如下的描述。

3.5.1 日志（Logging）

Scrapy 使用 Python 的内置日志记录系统进行事件日志记录。日志记录可以立即使用，并且可以在记录设置中列出的 Scrapy 设置在某种程度上进行配置。Scrapy 调用 scrapy.utils.log.configure_logging（）设置一些合理的默认值，并在运行命令时在日志设置中处理这些设置。

3.5.2 页面服务（Web Service）

Scrapy 提供的用于监控及控制运行中的爬虫的 web 服务(service)，并且服务是可扩展的并且默认是启用的。

3.5.3 Item Exporters

当抓取了需要的数据(Items)，就会想要将他们持久化或导出它们，并应用在其他的程序。这是整个抓取过程的目的。为此，Scrapy 提供了 Item Exporters 来创建不同的输出格式，如 XML，CSV 或 JSON。

3.5.4 自动限速(AutoThrottle)扩展

该扩展能根据 Scrapy 服务器及爬取的网站的负载自动限制爬取速度。

设计目标：

1. 更友好的对待网站，而不使用默认的下载延迟 0。
2. 自动调整 scrapy 来优化下载速度，使得用户不用调节下载延迟及并发请求数来找到优化的值。用户只需指定允许的最大并发请求数，剩下的都交给扩展来完成。

下载延迟是通过计算建立 TCP 连接到接收到 HTTP 包头(header)之间的时间来测量的。

限速算法根据以下规则调整下载延迟及并发数:

1. spider 永远以 1 并发请求数及 AUTOTHROTTLE_START_DELAY 中指定的下载延迟启动。
2. 当接收到回复时，下载延迟会调整到该回复的延迟与之前下载延迟之间的平均值。

3.5.5 自定义扩展

自定义扩展框架提供一个机制，使得能将自定义功能绑定到 Scrapy。扩展只是正常的类，它们在 Scrapy 启动时被实例化、初始化。

扩展使用 Scrapy settings 管理它们的设置，通常扩展需要给它们的设置加上前缀，以避免跟已有(或将来)的扩展冲突。

扩展在扩展类被实例化时加载和激活。 因此，所有扩展的实例化代码必须在类的构造函数(__init__)中执行。要使得扩展可用，需要把它添加到 Scrapy 的 EXTENSIONS 配置中。

为了禁用一个默认开启的扩展(比如，包含在 EXTENSIONS_BASE 中的扩展)，需要将其顺序(order)设置为 None 。

每个扩展是一个单一的 Python class。 Scrapy 扩展(包括 middlewares 和 pipelines)的主要入口是 from_crawler 类方法， 它接收一个 Crawler 类的实例。通过这个对象访问 settings，signals，stats，控制爬虫的行为。通常来说，扩展关联到 signals 并执行它们触发的任务。如果 from_crawler 方法抛出 NotConfigured 异常， 扩展会被禁用。否则，扩展会被开启。

3.6 故障处理要求

下面是 Scrapy 提供的故障异常抛出及其产生的场景。

DropItem 该异常由 item pipeline 抛出，用于停止处理 item。

CloseSpider 该异常由 spider 的回调函数(callback)抛出，来暂停/停止 spider。

IgnoreRequest 该异常由调度器(Scheduler)或其他下载中间件抛出，声明忽略该 request。

NotConfigured 该异常由 Extensions、Item pipelines、Downloader middlwares、Spider middlewares 组件抛出，声明其仍然保持关闭。该异常必须由组件的构造

器(constructor)抛出。

NotSupported 该异常声明一个不支持的特性。

日志 log 记录完整详细的操作动作与操作数据，用于调试和查错。

3.7 其他专门要求

无

4 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

显示器分辨率：800×600 以上

用于开发 python 爬虫的 PC 机或网络服务器

4.2 支持软件

Python3.0+ 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:一个交互终端，提供在未启动 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。

5 拓展和改进

5.1 方案思路

1. 首先在框架的基础上实现一个具体的爬虫，完整爬取一个网站。通过演示将整个爬取的过程展现出来，给出最终的爬取结果。并且以实际爬取过程测试框架的并发量和负载能力，尝试给出优化方案。

2. 在实际爬取中需要对使用框架后的爬取行为过程做限定和约束，避免重复爬取和循环爬取，同时避免爬取中数据丢失或者遗漏。爬取时也需要尝试多种反爬取措施，模拟人为动作行为，避免爬取失败。

3. 根据爬取的实际效果，进一步对爬虫的下载器做优化，加速爬取下载。

5.2 详细设计

5.2.1 搭建测试环境

为了显示的相关方案的改进效果，需要搭建测试环境。在改进前和改进后，分别统计爬取同一网站所有目标的 url 所花费的起止时间，并将爬取到的内容输出到一个文档记录中，最后根据统计结果，证明改进方案是否成功。爬取对

象为北航计算机学院官方网站主站（http://scse.buaa.edu.cn/buaa-css-web/initAction.action）。

5.2.2 反爬改进

Scrapy 本身也提供了一个 ProxyMiddleware，但是它只能使用固定的 IP 地址。由于固定的 IP 地址可能遭到 IP 被禁用的情况，这么一来，你就不能使用本地 IP 进行相应的爬虫工作了。此时就需要进行反爬虫的实现，使用到的两种基本策略：伪装 user agent 和使用免费代理。伪装 user agent 方式通过准备若干个浏览器的 user agent，每次发送请求的时候就从这几个 user agents 中随机选一个填上去，从而达到伪装的目的。但是如果对方用某段时间内某 IP 的访问次数来判定爬虫，然后将这些爬虫的 IP 都封掉的话，以上伪装就失效了。通过使用免费的代理减小单个 IP 的访问量，从而规避反爬。因而需要使用 python 代码进行代理 IP 地址的抓取和可用性测试，之后将可用的代理保存到相应的文件中提供给 scrapy 使用。

5.2.3 下载器改进

Scrapy 采用 Python 内置的集合（set）实现内存过滤器，同时提供文件过滤器让用户选择。两者并不排斥，文件过滤器的设置是为了可以停止爬虫（结束爬虫进程），然后在之后的某一个时间内重新接着上次的爬取继续时，可以直接从文件中读取已经爬取的 url。实际运行期间的过滤都是在内存过滤器（set 集合实现的过滤器）进行的。默认的 Python 集合需要存储网页本身的 url，同时为了实现管理集合以及其他一些针对集合的操作，Python 需要保存的内容除了 URL 之外还有许多，因此，当爬取大量网页的时候占用内存会比较大。而分析发现，我们实际上只需要判断一个 url 是否已经出现过，而至于 url 是什么或者

何时出现并不在乎。其次，我们也不会对已经爬取的 url 进行索引等操作，所以 Python 内置的 set 结合本身的许多操作相对来说是冗余的。根据以上分析，可以使用布隆过滤器（布隆过滤器可以用于检索一个元素是否在一个集合中，它的优点是空间效率和查询时间都远远超过一般的算法）来实现对已下载网页的过滤。