密级：内部

Scrapy 需求规格说明书

[V1.30]

编写	武丁泽宇	日期	2017 年 3 月 29 日
校对	郭炜锋	日期	2017 年 4 月 6 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
1.00	2017.3.21		完成第一版	武丁泽宇
1.10	2017.3.29	4	修改第 4 章内容	武丁泽宇
1.11	2017.3.29	4 、5	修改 4 章内容,删除 5 章	武丁泽宇
1.2	2017.4.4	1、2 、3、 4	按照修改建议进行修订	武丁泽宇
1.3	2017.4.5	3	完善业务需求和用例图	武丁泽宇
1.4	2017.4.9	全文	排版修改	胡勇

目录

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

（1） 爬虫:具有抓取网页内容功能的软件

（2） 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。

（3） 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 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。

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。

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

构建一整套便捷高效地爬取框架，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的修改框架内容。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 需求设计

3.1 需求

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。

软件必要的功能：

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

其他软件功能需求：

1. 用 scrapy crawl 来启动 Scrapy，也可以使用 API 在脚本中启动 Scrapy。
2. 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。
3. 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。
4. 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)。
5. 默认可以进行设置全局并发进行同时处理多个 request。
6. 对 HTML, XML 源数据 选择及提取 的内置支持, 提供了 CSS 选择器(selector)以及 XPath 表达式进行处理，以及一些帮助函数(helper method)来使用正则表达式来提取数据。
7. 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。
8. 通过 feed 导出 提供了多格式(JSON、CSV、XML)，多存储后端(FTP、S3、本地文件系统)的内置支持。
9. 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。
10. 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。
11. 高扩展性。通过使用 signals ， 设计好的 API(中间件, extensions, pipelines)来定制实现功能。
12. 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、robots.txt 、爬取深度限制。
13. 内置 Telnet 终端 ， 通过在 Scrapy 进程中钩入 Python 终端，可以查看并且调试爬虫。
14. 其他一些特性，如可重用的，从 Sitemaps 及 XML/CSV feeds 中爬取网站的爬虫、 可以自动下载爬取到的数据中的图片(或者其他资源)的 media pipeline、带缓存的 DNS 解析器等性。

业务需求

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，当我们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。但是在抓取对象复杂时，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，将开发人员面临的共同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。因此，scrapy 设计出了图 1 所示的框架来满足设计需求。

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。

3.2 设计框架及组件概述

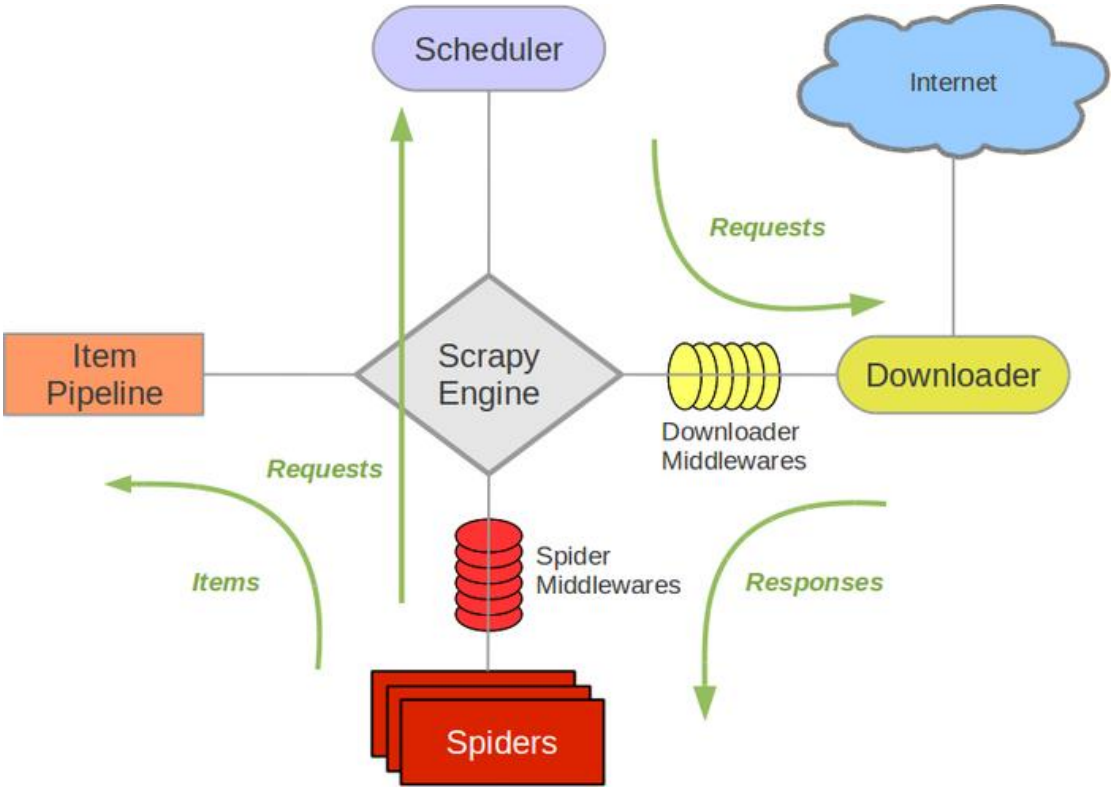

图 1 Scrapy 框架图

Scrapy 框架如图 1 所示。

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

URL 的类。 每个 spider 负责处理一个特定(或一些)网站。

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)。

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 。

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。更多复杂的业务逻辑，将在日后的实验中， 通过类图、甘特图、流程图等更直观的形式表现。Scrapy 框架用例图如图 2 所示。

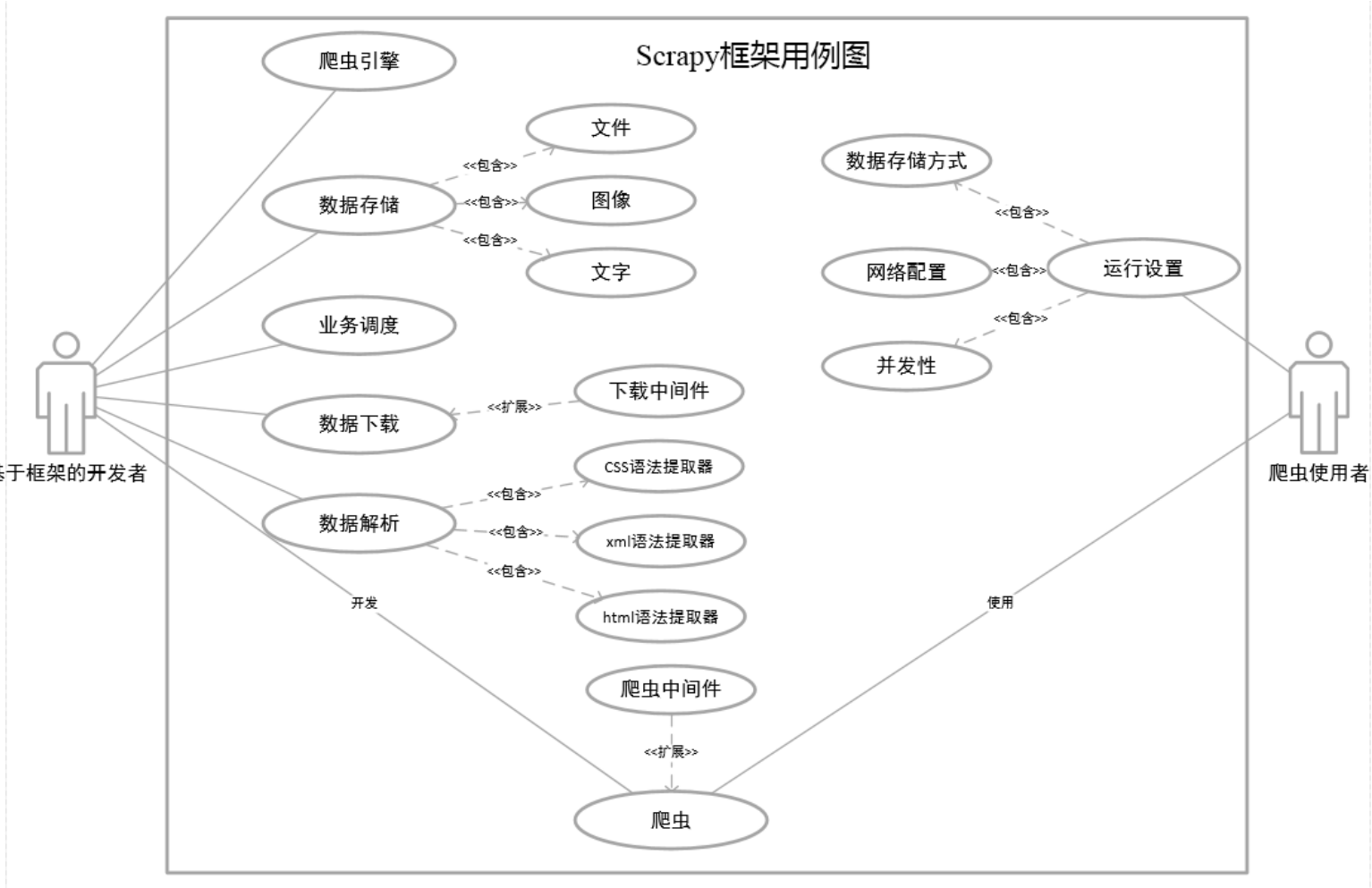

图 2 Scrapy 用例图

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	爬虫开发者定义起始URL以及从爬取的页面中解析后续的URL和抽取数据的方法行为的地方。
Precondition	正确安装了spider类库和相应的Python版本，有开发Python程序所需的编辑环境。
Primary Actor	爬虫的开发者
Secondary Actors	爬虫的使用者
Dependency	用于抽取URL和数据的原始HTML页面由下载器下载完成，然后经由爬虫引擎管理控制，交给spider；抽取的URL和数据会由爬虫引擎管理控制，分别交给调度器和数据处理管道。
Generalization	无

Basic Flow (Untitled) ▼	Steps
1	由起始URL生成起始Request，通过爬虫引擎调度，将生成的请求交给到调度器。
2	下载器根据起始Request下载好的HTML原始页面，经由爬虫引擎调度，将HTML原始页面交给spider
3	spider根据爬虫开发者定义好的URL和数据抽取方法，从原始HTML页面中抽取出URL和数据，并生成新的Request和Item。
4	spider生成新的Request和Item，通过爬虫引擎调度，将Request传给调度器，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	所爬取的页面需要登录，以及cookie验证时，在Request中添加cookie和登录信息
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	当运行过程中出现异常时，根据严重性进行日志记录
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	爬虫使用者
Secondary Actors	爬虫开发者
Dependency	spider对爬虫爬取逻辑的代码定义，数据处理的Item管道开发
Generalization	无

Basic Flow (Untitled) ▼	Steps
1	用户通过配置文件定义爬虫的一些参数，包括并行数、输出的日志级别等，爬虫引擎根据这些定义好的参数进行爬虫各部件的初始化
Postcondition	爬虫启动成功

Specific Alternative Flow (Untitled) ▼	RFS 1
1	爬虫启动失败，由爬虫开发者或者爬虫使用者重新调整参数
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	将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	spder将解析出的实体交付给实体管道
Primary Actor	实体管道
Secondary Actors	爬虫的使用者
Dependency	无
Generalization	无

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	Downloader
Brief Description	下载器，根据引擎发出的请求url,用于抓取相关的网页信息，并将之反馈给爬虫
Precondition	引擎经过下载中间件，将请求发送给下载器
Primary Actor	引擎
Secondary Actors	下载中间件
Dependency	请求url
Generalization	网页信息

Basic Flow (Untitled)	Steps
	1 引擎将请求信息通过下载中间件发送给下载器
	2 下载完胜后，生成response，将其通过下载中间件发送给引擎
	Postcondition 完成一次下载请求

Specific Alternative Flow (Untitled)	3
	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

图 7 调度器 模块 RUCM 图

图 8 scrapy engine 模块 RUCM 图

3.3 核心功能模块

3.3.1 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。

3.3.2 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。

3.3.3 Item Pipeline

当 Item 在 Spider 中被收集之后，它将会被传递到 Item Pipeline，一些组件会按照一定的顺序执行对 Item 的处理。

每个 item pipeline 组件(有时称之为“Item Pipeline”)是实现了简单方法的 Python 类。他们接收到 Item 并通过它执行一些行为，同时也决定此 Item 是否继续通过 pipeline，或是被丢弃而不再进行处理。

3.3.4 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。要激活下载器中间件组件，将其加入到 DOWNLOADER_MIDDLEWARES 设置中。 该设置是一个字典(dict)，键为中间件类的路径，值为其中间件的顺序(order)。

Scrapy 定义和实现了以下下载中间件：CookiesMiddleware、DefaultHeadersMiddleware、DownloadTimeoutMiddleware、HttpAuthMiddleware、HttpCacheMiddleware、HttpCompressionMiddleware、ChunkedTransferMiddleware、HttpProxyMiddleware、RedirectMiddleware、MetaRefreshMiddleware settings、RetryMiddleware、RobotsTxtMiddleware、DownloaderStats、UserAgentMiddleware、AjaxCrawlMiddleware。

3.3.5 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。

要启用 spider 中间件，可以将其加入到 SPIDER_MIDDLEWARES 设置中。 该设置是一个字典，键位中间件的路径，值为中间件的顺序(order)。

Scrapy 定义的 spider 中间件如下：

DepthMiddleware 是一个用于追踪每个 Request 在被爬取的网站的深度的中间件。 其可以用来限制爬取深度的最大深度或类似的事情。

HttpErrorMiddleware 过滤出所有失败(错误)的 HTTP response，因此 spider 不需要处理这些 request。

OffsiteMiddleware 过滤出所有 URL 不由该 spider 负责的 Request。

RefererMiddleware 根据生成 Request 的 Response 的 URL 来设置 Request Referer 字段。

UrlLengthMiddleware 过滤出 URL 长度比 URLLENGTH_LIMIT 的 request。

3.4 扩展功能模块

3.4.1 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。 下面以优先级降序的方式给出方式列表:

1. 命令行选项(Command line Options)(最高优先级)
2. 每个 spider 的设定
3. 项目设定模块(Project settings module)
4. 命令默认设定模块(Default settings per-command)
5. 全局默认设定(Default global settings) (最低优先级)

3.4.2 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。

3.4.3 页面服务（Web Service）

Scrapy 提供用于监控及控制运行中的爬虫的 web 服务(service)。 服务通过 JSON-RPC 2.0 协议提供大部分的资源，不过也有些(只读)资源仅仅输出 JSON 数据。Scrapy 为管理 Scrapy 进程提供了一个可扩展的 web 服务。可以通过 WEBSERVICE_ENABLED 来启用服务。服务将会监听 WEBSERVICE_PORT 的端口，并将记录写入到 WEBSERVICE_LOGFILE 指定的文件中。web 服务是默认启用的内置 Scrapy 扩展 。

3.4.4 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。

3.4.5 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。

3.4.6 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。

3.5 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。

3.5 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：256m 或 512m

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。