密级：内部

Scrapy需求规格说明书

[V1.20]

|  |  |  |  |
| --- | --- | --- | --- |
| 编写 | 武丁泽宇 | 日期 | 2017年3月 29 日 |
| 校对 | 郭炜峰 | 日期 | 2017年4月2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 | 所修改页 | 注记 |
| 1.00 | 2017.3.21 |  |  |  |
| 1.10 | 2017.3.29 | 4 |  |  |
| 1.11 | 2017.3.29 | 4 、5 |  |  |
| 1.2 |  |  |  |  |
|  |  |  |  |  |

目录

[1. 引言 1](#_Toc914356955)

[1.1 文档标识与编写目的 1](#_Toc351941043)

[1.2 背景 1](#_Toc667209595)

[1.3 定义 1](#_Toc1949457975)

[1.4 参考资料 2](#_Toc1227568285)

[2. 任务概述 2](#_Toc848595644)

[2.1 目标 2](#_Toc1167970725)

[2.2 用户特点 2](#_Toc1916397728)

[2.3 假定与约束 2](#_Toc322282235)

[3. 需求规定 2](#_Toc773950122)

[3.1 功能要求 3](#_Toc1872184176)

[3.1.1 需求场景及需求功能 3](#_Toc2055420827)

[3.1.2 框架组件概述 3](#_Toc1558194657)

[3.1.3 Scapy框架用例 5](#_Toc616920281)

[3.1.4 Spiders 5](#_Toc1587800467)

[3.1.5 选择器(Selectors) 6](#_Toc513807113)

[3.1.6 Items 6](#_Toc1306943947)

[3.1.7 Item Loaders 7](#_Toc1085726628)

[3.1.8 Item Pipeline 7](#_Toc718399012)

[3.1.9 设置（Settings） 7](#_Toc1272621567)

[3.1.10 异常(Exceptions) 8](#_Toc48018294)

[3.1.11 日志（Logging） 8](#_Toc918253386)

[3.1.12 页面服务（Web Service） 8](#_Toc35611868)

[3.1.13 下载器中间件 8](#_Toc9388468)

[3.1.14 Spider中间件 9](#_Toc1877638797)

[3.1.15 扩展 9](#_Toc908751611)

[3.1.16 核心API 10](#_Toc936275729)

[3.1.17 信号(Signals) 10](#_Toc8411520)

[3.1.18 Item Exporters 11](#_Toc929326356)

[3.1.19 数据收集(Stats Collection) 12](#_Toc1766588777)

[3.1.20 自动限速(AutoThrottle)扩展 12](#_Toc1570477063)

[3.2 非功能性要求 12](#_Toc1843683311)

[3.3 故障处理要求 13](#_Toc2118529820)

[3.4 其他专门要求 13](#_Toc90203010)

[4. 运行环境规定 13](#_Toc1645657638)

[4.1 设备 13](#_Toc1198614458)

[4.2 支持软件 14](#_Toc938798655)

[4.3 接口 14](#_Toc666144715)

[4.3.1 硬件接口 14](#_Toc967528538)

[4.3.2 软件接口 14](#_Toc1261080890)

[4.3.3 通信接口 14](#_Toc1440094837)

[4.3.4 用户接口 14](#_Toc692229066)

# 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。

## 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。

## 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。

## 定义

爬虫:抓取网页内容的模块

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 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](http://twistedmatrix.com/trac/) 编写。

## 参考资料

GJB 438B 国军标开发通用文档

# 任务概述

## 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。

## 用户特点

熟悉网页抓取和Python的程序开发用户。

## 假定与约束

无

# 需求规定



图 1 框架图

## 功能要求

### 需求场景及需求功能

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。

软件框架必须提供以下功能：

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

调度功能——可调度的爬取请求队列

下载功能——获取页面数据并提供给引擎，而后提供给 spider

处理数据功能——从抓取到的网页中（HTML源码）提取数据。

日志功能——事件日志记录

扩展机制——允许开发者自定义

设置选项——用户可选运行模式

异常处理——应急、错误处理

### 框架组件概述

**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 。

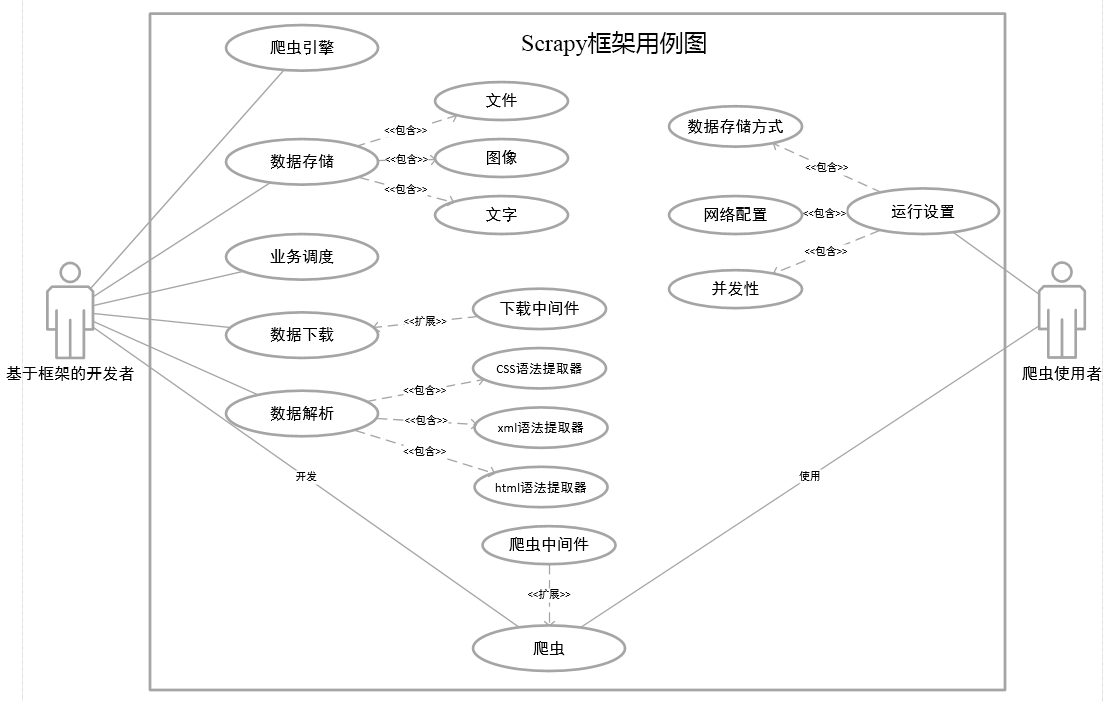
**Spider中间件(Spider middlewares)**

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。

**数据流(Data flow)**

Scrapy中的数据流由执行引擎控制，其过程如下:

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。



1. (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。

**事件驱动网络(Event-driven networking)**

Scrapy基于事件驱动网络框架 [Twisted](http://twistedmatrix.com/trac/) 编写。因此，Scrapy基于并发性考虑由非阻塞(即异步)的实现。

### Scapy框架用例

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。

### 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)的地方。

对spider来说，爬取的循环类似如下过程:

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()](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html" \l "scrapy.spiders.Spider.start_requests" \o "scrapy.spiders.Spider.start_requests) 来获取。 [start\_requests()](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html" \l "scrapy.spiders.Spider.start_requests" \o "scrapy.spiders.Spider.start_requests) 读取 [start\_urls](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html" \l "scrapy.spiders.Spider.start_urls" \o "scrapy.spiders.Spider.start_urls) 中的URL， 并以 [parse](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html" \l "scrapy.spiders.Spider.parse" \o "scrapy.spiders.Spider.parse) 为回调函数生成 [Request](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html" \l "scrapy.http.Request" \o "scrapy.http.Request) 。
2. 在回调函数内分析返回的(网页)内容，返回 [Item](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/items.html" \l "scrapy.item.Item" \o "scrapy.item.Item) 对象、dict、 [Request](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html" \l "scrapy.http.Request" \o "scrapy.http.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)](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/selectors.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/item-pipeline.html" \l "topics-item-pipeline) 处理)或使用 [Feed exports](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/feed-exports.html" \l "topics-feed-exports) 存入到文件中。

### 选择器(Selectors)

当抓取网页时，做的最常见的任务是从HTML源码中提取数据。Scrapy提取数据有自己的一套机制。它们被称作选择器(seletors)，因为他们通过特定的 [XPath](http://www.w3.org/TR/xpath) 或者 [CSS](http://www.w3.org/TR/selectors) 表达式来“选择” HTML文件中的某个部分。

[XPath](http://www.w3.org/TR/xpath) 是一门用来在XML文件中选择节点的语言，也可以用在HTML上。 [CSS](http://www.w3.org/TR/selectors) 是一门将HTML文档样式化的语言。选择器由它定义，并与特定的HTML元素的样式相关连。

Scrapy选择器构建于 [lxml](http://lxml.de/) 库之上，这意味着它们在速度和解析准确性上非常相似。

不同于 [lxml](http://lxml.de/) API的臃肿，该API短小而简洁。这是因为 [lxml](http://lxml.de/) 库除了用来选择标记化文档外，还可以用到许多任务上。

### Items

爬取的主要目标就是从非结构性的数据源提取结构性数据，例如网页。 Scrapy spider可以以python的dict来返回提取的数据，虽然dict很方便，并且用起来也熟悉，但是其缺少结构性，容易打错字段的名字或者返回不一致的数据，尤其在具有多个spider的大项目中。

为了定义常用的输出数据，Scrapy提供了 [Item](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/items.html" \l "scrapy.item.Item" \o "scrapy.item.Item) 类。 [Item](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/items.html" \l "scrapy.item.Item" \o "scrapy.item.Item) 对象是种简单的容器，保存了爬取到得数据。 其提供了[类似于词典(dictionary-like)](https://docs.python.org/library/stdtypes.html" \l "dict) 的API以及用于声明可用字段的简单语法。

许多Scrapy组件使用了Item提供的额外信息: exporter根据Item声明的字段来导出数据、 序列化可以通过Item字段的元数据(metadata)来定义、 trackref 追踪Item实例来帮助寻找内存泄露 (see [使用 trackref 调试内存泄露](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/leaks.html" \l "topics-leaks-trackrefs)) 等等。

### Item Loaders

Item Loaders 提供了一种简便的构件（mechanism）来抓取。[Items](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/items.html" \l "topics-items) 提供了盛装抓取到的数据的\*容器\* , 而Item Loaders提供了构件\*装载populating\*该容器。

Item Loaders 被设计用来提供一个既弹性又高效简便的构件，以扩展或重写爬虫或源格式(HTML, XML之类的)等区域的解析规则，这将不再是后期维护的噩梦。

用Item Loaders装载Items

要使用Item Loader, 必须先将它实例化。可以使用类似字典的对象(例如: Item or dict)来进行实例化, 或者不使用对象也可以, 当不用对象进行实例化的时候,Item会自动使用 [ItemLoader.default\_item\_class](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/loaders.html" \l "scrapy.loader.ItemLoader.default_item_class" \o "scrapy.loader.ItemLoader.default_item_class) 属性中指定的Item 类在Item Loader constructor中实例化。当开始收集数值到Item Loader时,通常使用 [Selectors](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/selectors.html" \l "topics-selectors)。可以在同一个item field 里面添加多个数值；Item Loader将知道如何用合适的处理函数来“添加”这些数值。

### 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。

### 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)

### 异常(Exceptions)

下面是Scrapy提供的异常及其用法。

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（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。

### 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](http://www.jsonrpc.org/) 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/webservice.html" \l "std:setting-WEBSERVICE_ENABLED) 来启用服务。服务将会监听 [WEBSERVICE\_PORT](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/webservice.html" \l "std:setting-WEBSERVICE_PORT) 的端口，并将记录写入到 [WEBSERVICE\_LOGFILE](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/webservice.html" \l "std:setting-WEBSERVICE_LOGFILE) 指定的文件中。web服务是默认启用的[内置Scrapy扩展](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/extensions.html" \l "topics-extensions-ref) 。

### 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-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。

### 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/spiders.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-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。

### 扩展

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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-EXTENSIONS) 配置中。

为了禁用一个默认开启的扩展(比如，包含在 [EXTENSIONS\_BASE](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/signals.html" \l "topics-signals) 并执行它们触发的任务。如果 from\_crawler 方法抛出 [NotConfigured](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/exceptions.html" \l "scrapy.exceptions.NotConfigured" \o "scrapy.exceptions.NotConfigured) 异常， 扩展会被禁用。否则，扩展会被开启。

### 核心API

Scrapy核心API的目标用户是开发Scrapy扩展(extensions)和中间件(middlewares)的开发人员。

1. Crawler API

Scrapy API的主要入口是 [Crawler](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "scrapy.crawler.Crawler" \o "scrapy.crawler.Crawler) 的实例对象， 通过类方法 from\_crawler 将它传递给扩展(extensions)。 该对象提供对所有Scrapy核心组件的访问， 也是扩展访问Scrapy核心组件和挂载功能到Scrapy的唯一途径。Extension Manager负责加载和跟踪已经安装的扩展， 它通过 [EXTENSIONS](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-EXTENSIONS) 配置，包含一个所有可用扩展的字典， 字典的顺序与 [onfigure the downloader middlewares](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/downloader-middleware.html" \l "topics-downloader-middleware-setting) 配置的顺序一致。

1. 设置(Settings) API

设置Scrapy中使用的默认设置优先级的键名称和优先级的字典。每个项目定义一个设置入口点，给它一个用于识别的代码名称和一个整数优先级。 在Settings类中设置和检索值时，较高的优先级比较小的优先级具有更高的优先级。

1. SpiderLoader API

这个类负责检索和处理项目中定义的spider类。可以通过在SPIDER\_LOADER\_CLASS项目设置中指定其路径来使用自定义蜘蛛装载程序。 他们必须完全实现scrapy。interfaces。ISpiderLoader接口，以保证无错误的执行。

1. 信号(Signals) API

链接一个接收器函数(receiver function) 到一个信号(signal)。

1. 状态收集器(Stats Collector) API

收集统计spider的所有值。

### 信号(Signals)

Scrapy使用信号来通知事情发生。可以在Scrapy项目中捕捉一些信号(使用 [extension](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/extensions.html" \l "topics-extensions))来完成额外的工作或添加额外的功能，扩展Scrapy。信号提供了一些参数，不过处理函数不用接收所有的参数 - 信号分发机制(singal dispatching mechanism)仅仅提供处理器(handler)接受的参数。可以通过 [信号(Signals) API](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "topics-api-signals) 来连接(或发送自己的)信号。

以下给出Scrapy内置信号的列表及其意义。

engine\_started当Scrapy引擎启动爬取时发送该信号。

engine\_stopped当Scrapy引擎停止时发送该信号(例如，爬取结束)。

item\_scraped当item被爬取，并通过所有 [Item Pipeline](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/item-pipeline.html" \l "topics-item-pipeline) 后(没有被丢弃(dropped)，发送该信号。

item\_dropped当item通过 [Item Pipeline](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/item-pipeline.html" \l "topics-item-pipeline) ，有些pipeline抛出 [DropItem](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/exceptions.html" \l "scrapy.exceptions.DropItem" \o "scrapy.exceptions.DropItem) 异常，丢弃item时，该信号被发送。

spider\_closed当某个spider被关闭时，该信号被发送。该信号可以用来释放每个spider在 [spider\_opened](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/signals.html" \l "std:signal-spider_opened) 时占用的资源。

spider\_opened当spider开始爬取时发送该信号。该信号一般用来分配spider的资源，不过其也能做任何事。

spider\_idle当spider进入空闲(idle)状态时该信号被发送。

spider\_error当spider的回调函数产生错误时(例如，抛出异常)，该信号被发送。 request\_scheduled当引擎调度一个 [Request](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html" \l "scrapy.http.Request" \o "scrapy.http.Request) 对象用于下载时，该信号被发送。

response\_received当引擎从downloader获取到一个新的 [Response](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/request-response.html" \l "scrapy.http.Response" \o "scrapy.http.Response) 时发送该信号。

response\_downloaded当一个 HTTPResponse 被下载时，由downloader发送该信号。

### 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。

下面是一些Scrapy内置的 Item Exporters类。

BaseItemExporter这是一个对所有 Item Exporters 的(抽象)父类。它对所有(具体) Item Exporters 提供基本属性，如定义export什么fields。， 是否export空fields。， 或是否进行编码。

export\_item(item)输出给定item。 此方法必须在子类中实现。

serialize\_field(field， name， value)返回给定field的序列化值。可以覆盖此方法来控制序列化或输出指定的field。

start\_exporting()表示exporting过程的开始。

finish\_exporting()表示exporting过程的结束。

fields\_to\_export列出export什么fields值。， None表示export所有fields。 默认值为None。

export\_empty\_fields是否在输出数据中包含为空的item fields。 默认值是 False。

Encoding Encoding 属性将用于编码 unicode 值。

XmlItemExporter以XML格式 exports Items 到指定的文件类。

CsvItemExporter输出 csv 文件格式。

PickleItemExporter输出 pickle 文件格式。

PprintItemExporter输出整齐打印的文件格式。

JsonItemExporter输出 JSON 文件格式。， 所有对象将写进一个对象的列表。

JsonLinesItemExporter输出 JSON 文件格式。， 每行写一个 JSON-encoded 项。

### 数据收集(Stats Collection)

Scrapy提供了方便的收集数据的机制。数据以key/value方式存储，值大多是计数值。 该机制叫做数据收集器(Stats Collector)，可以通过 [Crawler API](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "topics-api-crawler) 的属性 [stats](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "scrapy.crawler.Crawler.stats" \o "scrapy.crawler.Crawler.stats) 来使用。无论数据收集(stats collection)开启或者关闭，数据收集器永远都是可用的。 数据收集器对每个spider保持一个状态表。当spider启动时，该表自动打开，当spider关闭时，自动关闭。

Scrapy提供了基本的 StatsCollector 作为数据收集器，也提供了基于 StatsCollector 的数据收集器。 可以通过 [STATS\_CLASS](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-STATS_CLASS) 设置来选择。默认使用的是 MemoryStatsCollector 。也可选择DummyStatsCollector，该数据收集器并不做任何事情。可以通过设置 [STATS\_CLASS](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/settings.html" \l "std:setting-STATS_CLASS) 启用这个收集器，来关闭数据收集，提高效率。

### 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/autothrottle.html" \l "std:setting-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。

## 非功能性要求

1. 用 scrapy crawl 来启动Scrapy，也可以使用 [API](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/api.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/selectors.html" \l "topics-selectors) 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/shell.html" \l "topics-shell) , 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/feed-exports.html" \l "topics-feed-exports) 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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/loaders.html" \l "topics-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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/signals.html" \l "topics-signals) ，设计好的API(中间件, [extensions](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/extensions.html" \l "topics-extensions), [pipelines](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/item-pipeline.html" \l "topics-item-pipeline))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](http://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/telnetconsole.html" \l "topics-telnetconsole) ，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](http://www.sitemaps.org/) 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。

## 故障处理要求

爬取数据日志完整，便于调试和查错。

## 其他专门要求

无

# 运行环境规定

## 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

## 支持软件

Python3.0+ 或 2.7。

基于Windows、Linux、MacOs等平台

## 接口

### 硬件接口

无。

### 软件接口

Python类库

### 通信接口

HTTP协议、TCP/IP协议、HTTPS协议

### 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。