密级：内部

Scrapy 需求规格说明书

[V1.00]

编	武丁泽宇	日	2017 年 3 月 21
校		日	
审		日	
批		日	

北京航空航天大学计算机学院

二〇一七年三月二十一日

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			

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 文档标识与编写目的

文档标识：XXXXX－XXXXX－XXXXX－XXXXX－1.00

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

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 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. 需求规定

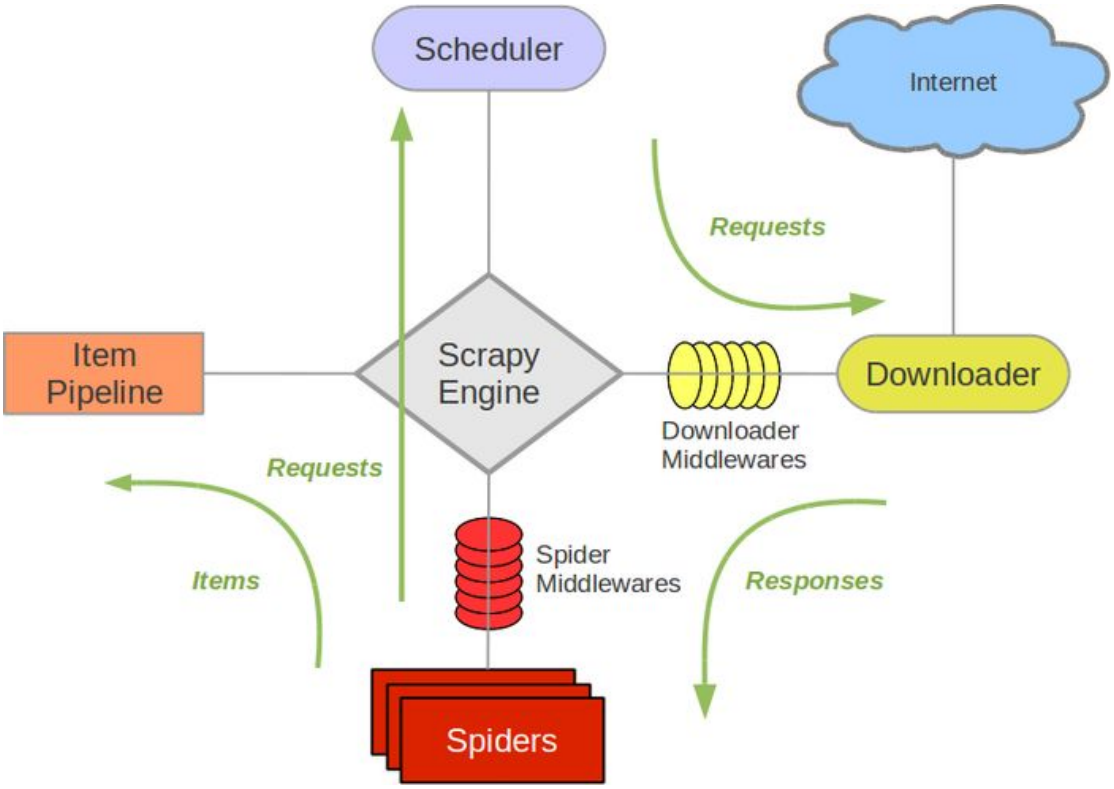

图 1 框架图

3.1 功能要求

3.1.1 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。
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。

事件驱动网络(Event-driven networking)

Scrapy 基于事件驱动网络框架 Twisted 编写。因此,Scrapy 基于并发性考虑由非阻塞(即异步)的实现。

3.1.2 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 就是您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() 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，您可以使用 选择器(Selectors) (您也可以使用 BeautifulSoup, lxml 或者您想用的任何解析器) 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。

3.1.3 选择器(Selectors)

当抓取网页时，你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 或者 CSS 表达式来“选择” HTML 文件中的某个部分。

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 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 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 API 的臃肿，该 API 短小而简洁。这是因为 lxml 库除了用来选择标记化文档外，还可以用到许多任务上。

3.1.4 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 类。 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) 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) 等等。

3.1.5 Item Loaders

Item Loaders 提供了一种简便的构件（mechanism）来抓取。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，你必须先将它实例化．你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 属性中指定的 Item 类在 Item Loader constructor 中实例化。当你开始收集数值到 Item Loader 时,通常使用 Selectors。 你可以在同一个 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。

3.1.6 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。

3.1.7 设置（Settings）

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

3.1.8 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。

3.1.9 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。

3.1.10 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 。

3.1.11 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。

3.1.12 Spider中间件

Spider 中间件是介入到 Scrapy 的 spider 处理机制的钩子框架，您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，您可以将其加入到 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。

3.1.13 扩展

扩展框架提供一个机制，使得你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。

3.1.14 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 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 配置，包含一个所有可用扩展的字典， 字典的顺序跟你在 configure the downloader middlewares 配置的顺序一致。

2. 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。

3. 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。

4. 信号(Signals) API

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

5. 状态收集器(Stats Collector) API

收集统计 spider 的所有值。

3.1.15 信号(Signals)

Scrapy 使用信号来通知事情发生。您可以在您的 Scrapy 项目中捕捉一些信号(使用 extension)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 来连接(或发送您自己的)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 后(没有被丢弃(dropped)，发送该信号。

item_dropped 当 item 通过 Item Pipeline ，有些 pipeline 抛出 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 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 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 时发送该信号。

response_downloaded 当一个 HTTPResponse 被下载时,由 downloader 发送该信号。

3.1.16 Item Exporters

当你抓取了你要的数据(Items)，你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。 你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。

3.1.17 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 的属性 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 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 启用这个收集器，来关闭数据收集，提高效率。

3.1.18 自动限速(AutoThrottle)扩展

该扩展能根据 Scrapy 服务器及您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。

3.2 性能要求

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。

3.3 故障处理要求

爬取数据日志完整，便于调试和查错。

3.4 其他专门要求

无

4. 运行环境规定

4.1 设备

PC 机。

显示器分辨率为 800×600 以上。

4.2 支持软件

Python3.0 或 2.7。

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.数据字典

序号	数据含义	输入（I）输出（O）常数（C）	类型	范围区间	约束
1	信号	I/O	文本	字母或汉字	长度[1。，30]个半角字符
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

注：文中所有边界均为闭区间。