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 | 郭炜峰 | 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 |

目录

[1.引言 4](#__RefHeading___Toc1602_1957687751)

[1.1编写目的 4](#__RefHeading___Toc1604_1957687751)

[1.2背景 4](#__RefHeading___Toc1606_1957687751)

[1.3定义 4](#__RefHeading___Toc1608_1957687751)

[1.4参考资料 5](#__RefHeading___Toc1610_1957687751)

[2.任务概述 5](#__RefHeading___Toc1612_1957687751)

[2.1目标 5](#__RefHeading___Toc1614_1957687751)

[2.2用户特点 5](#__RefHeading___Toc1616_1957687751)

[2.3假定与约束 5](#__RefHeading___Toc1618_1957687751)

[3.需求设计 5](#__RefHeading___Toc1620_1957687751)

[3.1需求 5](#__RefHeading___Toc1622_1957687751)

[3.1.1功能需求 5](#__RefHeading___Toc1624_1957687751)

[3.1.2业务需求 7](#__RefHeading___Toc1626_1957687751)

[3.2 设计框架及组件概述 7](#__RefHeading___Toc1628_1957687751)

[3.3 用例图 9](#__RefHeading___Toc1630_1957687751)

[3.4 核心功能模块 12](#__RefHeading___Toc1632_1957687751)

[3.4.1Scrapy Engine 12](#__RefHeading___Toc1634_1957687751)

[3.4.2Spiders 13](#__RefHeading___Toc1636_1957687751)

[3.4.3Item Pipeline 13](#__RefHeading___Toc1638_1957687751)

[3.4.4下载器中间件 14](#__RefHeading___Toc1640_1957687751)

[3.4.5Spider中间件 14](#__RefHeading___Toc1642_1957687751)

[3.5扩展功能模块 14](#__RefHeading___Toc1644_1957687751)

[3.5.1设置（Settings） 14](#__RefHeading___Toc1646_1957687751)

[3.5.2日志（Logging） 15](#__RefHeading___Toc1648_1957687751)

[3.5.3页面服务（Web Service） 15](#__RefHeading___Toc1650_1957687751)

[3.5.4Item Exporters 15](#__RefHeading___Toc1652_1957687751)

[3.5.5自动限速(AutoThrottle)扩展 15](#__RefHeading___Toc1654_1957687751)

[3.5.6自定义扩展 16](#__RefHeading___Toc1656_1957687751)

[3.6故障处理要求 16](#__RefHeading___Toc1658_1957687751)

[3.7其他专门要求 16](#__RefHeading___Toc1660_1957687751)

[4.运行环境规定 17](#__RefHeading___Toc1662_1957687751)

[4.1设备 17](#__RefHeading___Toc1664_1957687751)

[4.2支持软件 17](#__RefHeading___Toc1666_1957687751)

[4.3接口 17](#__RefHeading___Toc1668_1957687751)

[4.3.1硬件接口 17](#__RefHeading___Toc1670_1957687751)

[4.3.2软件接口 17](#__RefHeading___Toc1672_1957687751)

[4.3.3通信接口 17](#__RefHeading___Toc1674_1957687751)

[4.3.4用户接口 17](#__RefHeading___Toc1676_1957687751)

# 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:具有抓取网页内容功能的软件

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。

## 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

# 需求设计

## 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。

## 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。

## 用例图

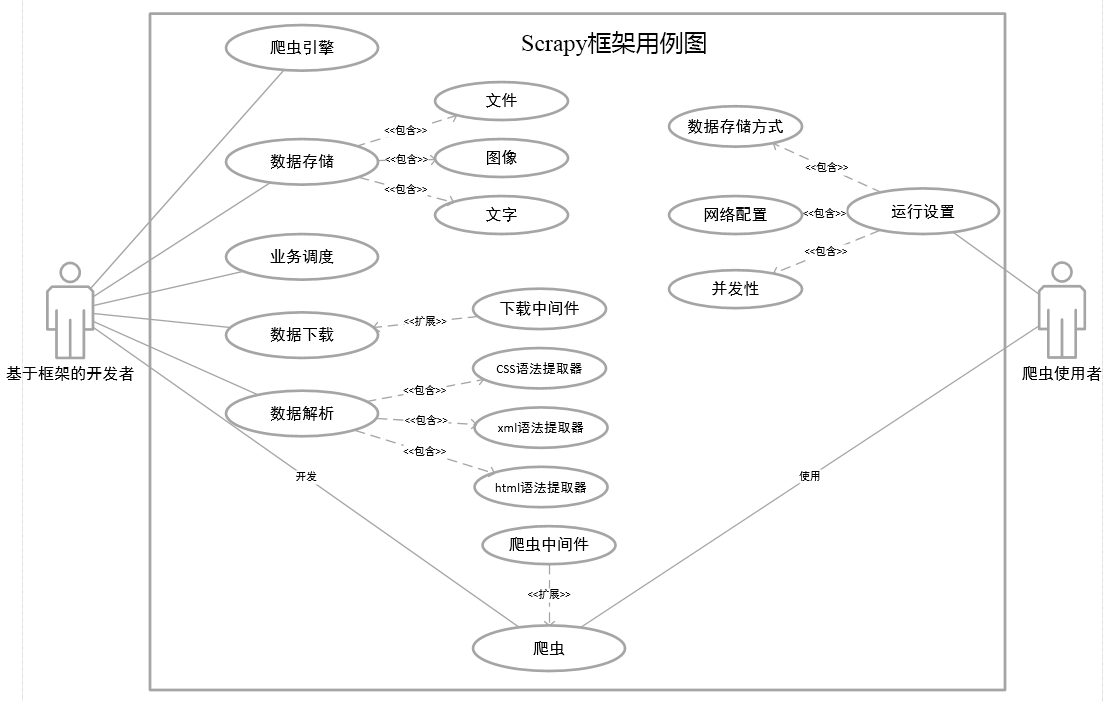
在实际环境中，开发者和使用者的身份一般重叠，但划分为两类，使得用例图更清晰。更多复杂的业务逻辑，将在日后的实验中， 通过类图、甘特图、流程图等更直观的形式表现。Scrapy框架用例图如图2所示。

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，。

图3 Spider模块 RUCM图



图4 设置模块RUCM图



图 5数据存储模块RUCM图



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

## 核心功能模块

### 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。

### 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。

### 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。

### 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。

### 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。

## 扩展功能模块

### 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)

### 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 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 。

### 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。

### 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。

### 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。

## 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。

## 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。