安装了Scrapy后，创建Scrapy工程需要使用命令行，例：

进入scrapy\_test文件夹后，执行：

scrapy startproject xiaozhu

创建xiaozhu工程，生成对应的python文件。

scrapy工程下有很大py文件，其中爬虫脚本的几个属性如下：

name属性：一个scrapy项目中可能有多个爬虫，每个爬虫的name属性是其自身的唯一标识，一个项目中不能有同名的爬虫。

start\_urls属性：设置爬虫的起始爬取点，会有一个或多个url，如果有多个url，scrapy会分别爬取。

parse方法：当一个页面下载完成后，scrapy会调用一个指定的页面解析函数（默认为parse），页面解析函数通常是一个生成器函数（yield返回）。

使用scrapy xpath语法时要避免<tbody>元素，因为无法返回结果：

<https://scrapy-chs.readthedocs.io/zh_CN/1.0/topics/firefox.html#topics-firefox>

scrapy框架测试：

例：程序scrapy\_test/xiaozhu

例：程序scrapy\_test/zhuanti

例：程序scrapy\_test/tieba

例：程序scrapy\_test/picture：

下载图片

scrapy的请求都被调度并异步处理，某个请求出错也不会影响其它请求被继续处理。

scrapy即使不创建工程也可以使用：

例：程序scrapy\_test/stackoverflow.py

import scrapy  
  
  
class StackOverflowSpider(scrapy.Spider):  
 name = "stackoverflow"  
 start\_urls = ['http://stackoverflow.com/questions?sort=votes']  
  
 def parse(self, response):  
 for href in response.css('.question-summary h3 a::attr(href)'):  
 full\_url = response.urljoin(href.extract())  
 yield scrapy.Request(full\_url, callback=self.parse\_question)  
  
 def parse\_question(self, response):  
 yield {'title': response.css('h1 a::text').extract()[0],  
 'votes': response.css('.question .vote-count-post::text').extract()[0],  
 'body': response.css('.question .post-text').extract()[0],  
 'tags': response.css('.question .post-tag::text').extract(),  
 'links': response.url  
 }

在命令行执行：

scrapy runspider stackoverflow.py -o stackoverflow-question.json

一个简单的scrapy爬虫例程：

1. 创建一个新的Scrapy工程
2. 定义你所需要要出去的Item对象
3. 编写一个spider来爬取某个网站并提取出所有的Item对象
4. 编写一个Item Pipeline来存储提取出来的Item对象

在命令行执行：scrapy startproject coolscrapy

例：程序scrapy\_test/coolscrapy

例：程序scrapy\_test/books

Scrapy爬虫框架的工作原理：



组件描述：

Scrapy Engine（内部组件）：引擎，负责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。

Scheduler（内部组件）：调度器，从引擎接受request并将他们入队，以便引擎请求他们时提供给引擎。

Downloader（内部组件）：下载器，负责获取页面数据并提供给引擎，而后提供给Spiders。

Spiders（用户实现）：爬虫，用户编写用于分析页面数据（response）并提取item，也会产生对新页面的下载请求。

Item Pipeline（可选组件）：数据管道，负责处理spider提取的数据item，典型处理有清理、验证及持久化。

Downloader middlewares（可选组件）：下载器中间件，引擎与下载器之间的特定钩子，处理下载器传递给引擎的response。

Spider middlewares（可选组件）：爬虫中间件，引擎与spider之间的特定钩子，处理spider的输入（response）和输出（item及request）。

Scrapy中 的数据流（Data flow）：

1. 引擎从Spider中获取到第一个要爬取的URL，Spider将该URL构造一个Request对象，提交给引擎
2. Request对象随后进入调度器（Scheduler）按某种算法进行排队，之后的某个时刻Scheduler将其出队，送我下载器（Downloader）。
3. 下载器（Downloader）根据Request对象中的URL地址发生HTTP请求到网站服务器，之后根据服务器返回的HTTP响应生成一个该页面的Response，并将其通过下载中间件（Downloader middlewares）返回给引擎。
4. 引擎从下载器接收到Response并通过Spider中间件（Spider middlewares）发送给Spider。
5. Spider处理Response并返回爬取到的Item以及新的Request给引擎。
6. 引擎将Spider返回的Item给Item Pipeline，将Request给调度器。
7. 重复直到调度器中没有更多的Request，引擎关闭。

Scrapy基于事件驱动网络框架Twisted编写，是异步实现的。