<http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html>

# 问题及解决：

1.linux下爬取时报503可能是由于settings文件中的user\_agent设置有误

# 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提供 [**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item) 类来满足这样的需求。

[**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#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)](http://docs.python.org/library/stdtypes.html#dict) 的API以及用于声明可用字段的简单语法。

## 声明Item

Item使用简单的class定义语法以及 [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象来声明。例如:

**import** scrapy

**class** **Product**(scrapy**.**Item):

name **=** scrapy**.**Field()

price **=** scrapy**.**Field()

stock **=** scrapy**.**Field()

last\_updated **=** scrapy**.**Field(serializer**=**str)

## Item字段(Item Fields)

[**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象指明了每个字段的元数据(metadata)。例如上面例子中 last\_updated 中指明了该字段的序列化函数。

您可以为每个字段指明任何类型的元数据。 [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象对接受的值没有任何限制。也正是因为这个原因，文档也无法提供所有可用的元数据的键(key)参考列表。 [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象中保存的每个键可以由多个组件使用，并且只有这些组件知道这个键的存在。您可以根据自己的需求，定义使用其他的[**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 键。 设置 [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象的主要目的就是在一个地方定义好所有的元数据。 一般来说，那些依赖某个字段的组件肯定使用了特定的键(key)。您必须查看组件相关的文档，查看其用了哪些元数据键(metadata key)。

需要注意的是，用来声明item的 [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象并没有被赋值为class的属性。 不过您可以通过[**Item.fields**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item.fields) 属性进行访问。

## 与Item配合

**创建item：**

>>> product **=** Product(name**=**'Desktop PC', price**=**1000)

>>> **print** product

Product(name='Desktop PC', price=1000)

**获取字段值：**

>>> product['name']

Desktop PC

>>> product**.**get('name')

Desktop PC

>>> product['price']

1000

>>> product['last\_updated']

Traceback (most recent call last):

**...**

KeyError: 'last\_updated'

>>> product**.**get('last\_updated', 'not set')

not set

>>> product['lala'] *# getting unknown field*

Traceback (most recent call last):

**...**

KeyError: 'lala'

>>> product**.**get('lala', 'unknown field')

'unknown field'

>>> 'name' **in** product *# is name field populated?*

True

>>> 'last\_updated' **in** product *# is last\_updated populated?*

False

>>> 'last\_updated' **in** product**.**fields *# is last\_updated a declared field?*

True

>>> 'lala' **in** product**.**fields *# is lala a declared field?*

False

**设置字段值：**

>>> product['last\_updated'] **=** 'today'

>>> product['last\_updated']

today

>>> product['lala'] **=** 'test' *# setting unknown field*

Traceback (most recent call last):

**...**

KeyError: 'Product does not support field: lala'

**获取所有获取到的值：**

>>> product**.**keys()

['price', 'name']

>>> product**.**items()

[('price', 1000), ('name', 'Desktop PC')]

**其它任务：**

**1）复制item：**

>>> product2 **=** Product(product)

>>> **print** product2

Product(name='Desktop PC', price=1000)

>>> product3 **=** product2**.**copy()

>>> **print** product3

Product(name='Desktop PC', price=1000)

**2）根据item创建字典（dict）：**

>>> dict(product) *# create a dict from all populated values*

{'price': 1000, 'name': 'Desktop PC'}

**3）根据字典（dict）创建item：**

>>> Product({'name': 'Laptop PC', 'price': 1500})

Product(price=1500, name='Laptop PC')

>>> Product({'name': 'Laptop PC', 'lala': 1500}) *# warning: unknown field in dict*

Traceback (most recent call last):

**...**

KeyError: 'Product does not support field: lala'

## 扩展Item

可以通过继承原始的Item来扩展item(添加更多的字段或者修改某些字段的元数据)。

EX:

**class** **DiscountedProduct**(Product):

discount\_percent **=** scrapy**.**Field(serializer**=**str)

discount\_expiration\_date **=** scrapy**.**Field()

也可以通过使用原字段的元数据,添加新的值或修改原来的值来扩展字段的元数据:

**class** **SpecificProduct**(Product):

name **=** scrapy**.**Field(Product**.**fields['name'], serializer**=**my\_serializer)

这段代码在保留所有原来的元数据值的情况下添加(或者覆盖)了 name 字段的 serializer

## Item对象

class scrapy.item.Item([arg])

返回一个根据给定的参数可选初始化的item

Item复制了标准的 [dict API](http://docs.python.org/library/stdtypes.html" \l "dict) 。包括初始化函数也相同。Item唯一额外添加的属性是:fields

一个包含了item所有声明的字段的字典，而不仅仅是获取到的字段。该字典的key是字段(field)的名字，值是 [Item声明](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#topics-items-declaring) 中使用到的 [Field](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 对象

## 字段（Field）对象

class scrapy.item.Field([arg])

[**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field) 仅仅是内置的 [dict](http://docs.python.org/library/stdtypes.html" \l "dict) 类的一个别名，并没有提供额外的方法或者属性。换句话说， [**Field**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Field)对象完完全全就是Python字典(dict)。被用来基于类属性(class attribute)的方法来支持 [item声明语法](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#topics-items-declaring)

# 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，爬取的循环类似下文:

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.org/zh_CN/latest/topics/spiders.html" \l "scrapy.spider.Spider.start_requests" \o "scrapy.spider.Spider.start_requests) 来获取的。 [start\_requests()](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.start_requests) 读取[start\_urls](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.start_urls) 中的URL， 并以 [parse](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.parse) 为回调函数生成 [Request](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request)

1. 在回调函数内分析返回的(网页)内容，返回 [Item](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item) 对象或者 [Request](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 或者一个包括二者的可迭代容器。 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)
2. 在回调函数内，您可以使用 [选择器(Selectors)](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html#topics-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
3. 最后，由spider返回的item将被存到数据库(由某些 [Item Pipeline](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/item-pipeline.html#topics-item-pipeline) 处理)或使用 [Feed exports](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/feed-exports.html#topics-feed-exports) 存入到文件中

虽然该循环对任何类型的spider都(多少)适用，但Scrapy仍然为了不同的需求提供了多种默认spider。 之后将讨论这些spider。

## Spider参数

Spider可以通过接受参数来修改其功能。 spider参数一般用来定义初始URL或者指定限制爬取网站的部分。 您也可以使用其来配置spider的任何功能。

在运行 [**crawl**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/commands.html#std:command-crawl) 时添加 -a 可以传递Spider参数:

scrapy crawl myspider -a category=electronics

Spider在构造器(constructor)中获取参数:

**import** scrapy

**class** **MySpider**(Spider):

name **=** 'myspider'

**def** **\_\_init\_\_**(self, category**=**None, **\***args, **\*\***kwargs):

super(MySpider, self)**.**\_\_init\_\_(**\***args, **\*\***kwargs)

self**.**start\_urls **=** ['http://www.example.com/categories/%s' **%** category]

*# ...*

Spider参数也可以通过Scrapyd的 schedule.json API来传递。 参见 [Scrapyd documentation](http://scrapyd.readthedocs.org/).

## 内置Spider参考手册

Scrapy提供多种方便的通用spider供您继承使用。 这些spider为一些常用的爬取情况提供方便的特性， 例如根据某些规则跟进某个网站的所有链接、根据 [Sitemaps](http://www.sitemaps.org/) 来进行爬取，或者分析XML/CSV源。

### Spider

**class scrapy.spider.Spider**

Spider是最简单的spider。每个其他的spider必须继承自该类(包括Scrapy自带的其他spider以及您自己编写的spider)。 Spider并没有提供什么特殊的功能。 其仅仅请求给定的**start\_urls**/**start\_requests** ，并根据返回的结果(resulting responses)调用spider的 **parse** 方法。

#### name:

定义spider名字的字符串(string)。spider的名字定义了Scrapy如何定位(并初始化)spider，所以其必须是唯一的。 不过您可以生成多个相同的spider实例(instance)，这没有任何限制。 name是spider最重要的属性，而且是必须的。

如果该spider爬取单个网站(single domain)，一个常见的做法是以该网站(domain)(加或不加[后缀](http://en.wikipedia.org/wiki/Top-level_domain) )来命名spider。 例如，如果spider爬取 **mywebsite.com** ，该spider通常会被命名为**mywebsite**

#### allowed\_domains:

可选。包含了spider允许爬取的域名(domain)列表(list)。 当 **[OffsiteMiddleware](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spider-middleware.html" \l "scrapy.contrib.spidermiddleware.offsite.OffsiteMiddleware" \o "scrapy.contrib.spidermiddleware.offsite.OffsiteMiddleware)** 启用时， 域名不在列表中的URL不会被跟进。

start\_urls:

URL列表。当没有制定特定的URL时，spider将从该列表中开始进行爬取。 因此，第一个被获取到的页面的URL将是该列表之一。 后续的URL将会从获取到的数据中提取。

#### custom\_settings:

#### crawler:

#### settings:

#### from\_crawler(crawler.\*args.\*\*kwargs):

#### start\_requests:

该方法必须返回一个可迭代对象(iterable)。该对象包含了spider用于爬取的第一个Request。

当spider启动爬取并且未指定URL时，该方法被调用。 当指定了URL时，[**make\_requests\_from\_url()**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.make_requests_from_url) 将被调用来创建Request对象。 该方法仅仅会被Scrapy调用一次，因此您可以将其实现为生成器。

该方法的默认实现是使用 **[start\_urls](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html" \l "scrapy.spider.Spider.start_urls" \o "scrapy.spider.Spider.start_urls)** 的url生成Request。

如果您想要修改最初爬取某个网站的Request对象，您可以重写(override)该方法。 例如，如果您需要在启动时以POST登录某个网站，你可以这么写:

**def** **start\_requests**(self):

**return** [scrapy**.**FormRequest("http://www.example.com/login",

formdata**=**{'user': 'john', 'pass': 'secret'},

callback**=**self**.**logged\_in)]

**def** **logged\_in**(self, response):

*# here you would extract links to follow and return Requests for*

*# each of them, with another callback*

**pass**

#### make\_requests\_from\_url(url):

该方法接受一个URL并返回用于爬取的 [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 对象。 该方法在初始化request时被**[start\_requests()](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html" \l "scrapy.spider.Spider.start_requests" \o "scrapy.spider.Spider.start_requests)** 调用，也被用于转化url为request。

默认未被复写(overridden)的情况下，该方法返回的Request对象中， [**parse()**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.parse) 作为回调函数，dont\_filter参数也被设置为开启。 (详情参见 [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request)).

#### parse(response):

当response没有指定回调函数时，该方法是Scrapy处理下载的response的默认方法。

**parse** 负责处理response并返回处理的数据以及(/或)跟进的URL。 [**Spider**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider) 对其他的Request的回调函数也有相同的要求。

该方法及其他的Request回调函数必须返回一个包含 [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 及(或) [**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item) 的可迭代的对象。

|  |  |
| --- | --- |
| **参数:** | **response** ([**Response**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Response)) – 用于分析的response |

#### log(message[, level, component])

使用 [**scrapy.log.msg()**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/logging.html#scrapy.log.msg) 方法记录(log)message。 log中自动带上该spider的 [**name**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.spider.Spider.name) 属性。 更多数据请参见 [Logging](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/logging.html#topics-logging) 。

#### closed(*reason*)

当spider关闭时，该函数被调用。 该方法提供了一个替代调用signals.connect()来监听**[spider\_closed](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/signals.html" \l "std:signal-spider_closed)** 信号的快捷方式。

### CrawlSpider

**class scrapy.contrib.spiders.CrawlSpider**

爬取一般网站常用的spider。其定义了一些规则(rule)来提供跟进link的方便的机制。 也许该spider并不是完全适合您的特定网站或项目，但其对很多情况都使用。 因此您可以以其为起点，根据需求修改部分方法。当然您也可以实现自己的spider。

除了从Spider继承过来的(您必须提供的)属性外，其提供了一个新的属性:

#### rules:

一个包含一个(或多个) [**Rule**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.contrib.spiders.Rule) 对象的集合(list)。 每个 [**Rule**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html#scrapy.contrib.spiders.Rule) 对爬取网站的动作定义了特定表现。 Rule对象在下边会介绍。 如果多个rule匹配了相同的链接，则根据他们在本属性中被定义的顺序，第一个会被使用。

该spider也提供了一个可复写(overrideable)的方法:

#### parse\_start\_url(*response*)

当start\_url的请求返回时，该方法被调用。 该方法分析最初的返回值并必须返回一个 [**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item)对象或者 一个 [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 对象或者 一个可迭代的包含二者对象。

#### 爬取规则(Crawling rules)

**classscrapy.contrib.spiders.Rule(link\_extractor, callback=None, cb\_kwargs=None, follow=None,process\_links=None, process\_request=None)**

**link\_extractor** 是一个 [Link Extractor](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/link-extractors.html#topics-link-extractors) 对象。 其定义了如何从爬取到的页面提取链接。

**callback** 是一个callable或string(该spider中同名的函数将会被调用)。 从link\_extractor中每获取到链接时将会调用该函数。该回调函数接受一个response作为其第一个参数， 并返回一个包含[**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item) 以及(或) [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 对象(或者这两者的子类)的列表(list)。

Warning:

当编写爬虫规则时，请避免使用 **parse** 作为回调函数。 由于 **[CrawlSpider](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html" \l "scrapy.contrib.spiders.CrawlSpider" \o "scrapy.contrib.spiders.CrawlSpider)** 使用 **parse** 方法来实现其逻辑，如果 您覆盖了 **parse** 方法，crawl spider 将会运行失败。

**cb\_kwargs** 包含传递给回调函数的参数(keyword argument)的字典。

**follow** 是一个布尔(boolean)值，指定了根据该规则从response提取的链接是否需要跟进。 如果**callback** 为None， **follow** 默认设置为 **True** ，否则默认为 **False** 。

**process\_links** 是一个callable或string(该spider中同名的函数将会被调用)。 从link\_extractor中获取到链接列表时将会调用该函数。该方法主要用来过滤。

**process\_request** 是一个callable或string(该spider中同名的函数将会被调用)。 该规则提取到每个request时都会调用该函数。该函数必须返回一个request或者None。 (用来过滤request)

### XMLFeedSpider

**classscrapy.contrib.spiders.XMLFeedSpider**

XMLFeedSpider被设计用于通过迭代各个节点来分析XML源(XML feed)。 迭代器可以从**iternodes** ， **xml** ， **html** 选择。 鉴于 **xml** 以及 **html** 迭代器需要先读取所有DOM再分析而引起的性能问题， 一般还是推荐使用 **iternodes** 。 不过使用 **html** 作为迭代器能有效应对错误的XML。

您必须定义下列类属性来设置迭代器以及标签名(tag name):

#### iterator:

用于确定使用哪个迭代器的string。可选项有:

* **'iternodes'** - 一个高性能的基于正则表达式的迭代器
* **'html'** - 使用 [**Selector**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html#scrapy.selector.Selector) 的迭代器。 需要注意的是该迭代器使用DOM进行分析，其需要将所有的DOM载入内存， 当数据量大的时候会产生问题。
* **'xml'** - 使用 [**Selector**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html#scrapy.selector.Selector) 的迭代器。 需要注意的是该迭代器使用DOM进行分析，其需要将所有的DOM载入内存， 当数据量大的时候会产生问题。

默认值为 **iternodes** 。

#### Itertag:

一个包含开始迭代的节点名的string。例如: itertag **=** 'product'

#### namespaces:

一个由 **(prefix, url)** 元组(tuple)所组成的list。 其定义了在该文档中会被spider处理的可用的namespace。 **prefix** 及 **uri** 会被自动调用 **[register\_namespace()](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html" \l "scrapy.selector.Selector.register_namespace" \o "scrapy.selector.Selector.register_namespace)** 生成namespace。

您可以通过在 **[itertag](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html" \l "scrapy.contrib.spiders.XMLFeedSpider.itertag" \o "scrapy.contrib.spiders.XMLFeedSpider.itertag)** 属性中制定节点的namespace。

除了这些新的属性之外，该spider也有以下可以覆盖(overrideable)的方法:

#### adapt\_response(*response*):

该方法在spider分析response前被调用。您可以在response被分析之前使用该函数来修改内容(body)。 该方法接受一个response并返回一个response(可以相同也可以不同)。

#### parse\_node(*response*, *selector*):

当节点符合提供的标签名时(**itertag**)该方法被调用。 接收到的response以及相应的[**Selector**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html#scrapy.selector.Selector) 作为参数传递给该方法。 该方法返回一个 [**Item**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/items.html#scrapy.item.Item) 对象或者 [**Request**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html#scrapy.http.Request) 对象 或者一个包含二者的可迭代对象(iterable)。

#### process\_results(*response*, *results*):

当spider返回结果(item或request)时该方法被调用。 设定该方法的目的是在结果返回给框架核心(framework core)之前做最后的处理， 例如设定item的ID。其接受一个结果的列表(list of results)及对应的response。 其结果必须返回一个结果的列表(list of results)(包含Item或者Request对象)。

### CSVFeedSpider

**classscrapy.contrib.spiders.CSVFeedSpider**

该spider除了其按行遍历而不是按节点遍历之外其他和XMLFeedSpider十分类似。 而其在每次迭代时调用的是 **[parse\_row()](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/spiders.html" \l "scrapy.contrib.spiders.CSVFeedSpider.parse_row" \o "scrapy.contrib.spiders.CSVFeedSpider.parse_row)** 。

#### delimiter:

在CSV文件中用于区分字段的分隔符。类型为string。 默认为 **','** (逗号)。

#### quotechar:

A string with the enclosure character for each field in the CSV file Defaults to **'"'** (quotation mark).

#### headers:

在CSV文件中包含的用来提取字段的行的列表。

#### parse\_row(*response*, *row*):

该方法接收一个response对象及一个以提供或检测出来的header为键的字典(代表每行)。 该spider中，您也可以覆盖 **adapt\_response** 及 **process\_results** 方法来进行预处理(pre-processing)及后(post-processing)处理。

### SitemapSpider

**classscrapy.contrib.spiders.SitemapSpider**

SitemapSpider使您爬取网站时可以通过 [Sitemaps](http://www.sitemaps.org/) 来发现爬取的URL。

其支持嵌套的sitemap，并能从 [robots.txt](http://www.robotstxt.org/) 中获取sitemap的url。

#### sitemap\_urls:

包含您要爬取的url的sitemap的url列表(list)。 您也可以指定为一个 [robots.txt](http://www.robotstxt.org/) ，spider会从中分析并提取url。

#### sitemap\_rules:

一个包含 **(regex, callback)** 元组的列表(list):

* **regex** 是一个用于匹配从sitemap提供的url的正则表达式。 **regex** 可以是一个字符串或者编译的正则对象(compiled regex object)。
* callback指定了匹配正则表达式的url的处理函数。 **callback** 可以是一个字符串(spider中方法的名字)或者是callable。

例如: sitemap\_rules **=** [('/product/', 'parse\_product')]

规则按顺序进行匹配，之后第一个匹配才会被应用。如果您忽略该属性，sitemap中发现的所有url将会被 **parse** 函数处理。

#### sitemap\_follow:

一个用于匹配要跟进的sitemap的正则表达式的列表(list)。其仅仅被应用在 使用 *Sitemap index files* 来指向其他sitemap文件的站点。

默认情况下所有的sitemap都会被跟进。

#### sitemap\_alternate\_links:

指定当一个 **url** 有可选的链接时，是否跟进。 有些非英文网站会在一个 **url** 块内提供其他语言的网站链接, 默认 **sitemap\_alternate\_links** 关闭。

# 选择器(Selectors)

## 使用选择器(selectors)

### 构造选择器(selectors)

Scrapy selector是以 **文字(text)** 或 **[TextResponse](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/request-response.html" \l "scrapy.http.TextResponse" \o "scrapy.http.TextResponse)** 构造的 [**Selector**](http://scrapy-chs.readthedocs.org/zh_CN/latest/topics/selectors.html#scrapy.selector.Selector) 实例。 其根据输入的类型自动选择最优的分析方法(XML vs HTML):

>>> **from** scrapy.selector **import** Selector

>>> **from** scrapy.http **import** HtmlResponse

以文字构造:

>>> body **=** '<html><body><span>good</span></body></html>'

>>> Selector(text**=**body)**.**xpath('//span/text()')**.**extract()

[u'good']

以response构造:

>>> response **=** HtmlResponse(url**=**'http://example.com', body**=**body)

>>> Selector(response**=**response)**.**xpath('//span/text()')**.**extract()

[u'good']

为了方便起见，response对象以 .selector 属性提供了一个selector， 您可以随时使用该快捷方法:

>>> response**.**selector**.**xpath('//span/text()')**.**extract()

[u'good']