请求和回应

Scrapy使用`Request`和`Response`对象来抓取网站。

通常，`Request`对象在蜘蛛中生成并通过系统，直到它们到达下载器，下载器执行请求并返回一个`Response`对象，该对象返回发出请求的蜘蛛。

两者`Request`和`Response`类都有子类，它们添加了基类中不需要的功能。下面在Request子类和 Response子类中描述了这些。

请求对象

class **scrapy.http.Request** (*url* [, *callback* , *method* ='GET' , *headers* , *body* , *cookies* , *meta* , *encoding* ='utf-8' , *priority* = 0 , *dont_filter* = False , *errback* , *flags*])

一个`Request`对象表示一个HTTP请求，它通常在Spider中生成并由Downloader执行，从而生成一个`Response`。

参数：

- **url**（*字符串*）- 此请求的URL
- **callback**（*callable*）- 将使用此请求的响应（一旦下载）*调用*的函数作为其第一个参数。有关更多信息，请参阅下面将其他数据传递给回调函数。如果请求未指定回调，则将使用spider的`parse()`方法。请注意，如果在处理期间引发异常，则会调用errback。
- **method**（*string*）- 此请求的HTTP方法。默认为`'GET'`。
- **meta**（*dict*）- `Request.meta`属性的初始值。如果给定，则此参数中传递的dict将被浅层复制。
- **body**（*str 或unicode*）- 请求体。如果a `unicode`被传递，则`str`使用传递的*编码*（默认为`utf-8`）对其 进行*编码*。如果`body`未给出，则存储空字符串。无论此参数的类型如何，存储的最终值都是`str`（从不`unicode`或`None`）。
- **headers**（*dict*）- 此请求的标头。dict值可以是字符串（对于单值标头）或列表（对于多值标头）。如果`None`作为值传递，则根本不会发送HTTP标头。
- **饼干**（*字典或清单*）-
  请求cookie。这些可以以两种形式发送。

  1. 使用词典：

     ```
     request_with_cookies = Request(url="http://www.example.com",
                                    cookies={'currency': 'USD',
     'country': 'UY'})
     ```

  2. 使用dicts列表：

```
request_with_cookies = Request(url="http://www.example.com",
                               cookies=[{'name': 'currency',
                                         'value': 'USD',
                                         'domain': 'example.com',
                                         'path': '/currency'}])
```

后一种形式允许自定义 cookie的属性 `domain` 和 `path` 属性。这仅在保存cookie以供以后请求时才有用。

当某个站点返回cookie（在响应中）时，这些cookie存储在该域的cookie中，并将在将来的请求中再次发送。这是任何常规Web浏览器的典型行为。但是，如果由于某种原因，您希望避免与现有Cookie合并，则可以通过将 `dont_merge_cookies` 密钥设置为True 来指示Scrapy执行此操作 `Request.meta` 。

不合并cookie的请求示例：

```
request_with_cookies = Request(url="http://www.example.com",
                               cookies={'currency': 'USD', 'country':
'UY'},
                               meta={'dont_merge_cookies': True})
```

有关更多信息，请参阅CookiesMiddleware。

- **encoding**（*字符串*）- 此请求的编码（默认为 `'utf-8'` ）。此编码将用于对URL进行百分比编码并将正文转换为 `str` （如果给定 `unicode` ）。
- **priority**（*int*）- 此请求的优先级（默认为 `0` ）。调度程序使用优先级来定义用于处理请求的顺序。具有更高优先级值的请求将更早执行。允许使用负值以指示相对较低的优先级。
- **dont_filter**（*boolean*）- 表示调度程序不应过滤此请求。当您想要多次执行相同的请求时，可以使用此选项来忽略重复过滤器。小心使用它，否则您将进入爬行循环。默认为 `False` 。
- **errback**（*callable*）- 在处理请求时引发任何异常时将*调用*的函数。这包括因404 HTTP错误而失败的页面等。它接收Twisted Failure实例作为第一个参数。有关更多信息，请参阅下面的使用errbacks捕获请求处理中的异常。
- **flags**（*list*）- 发送给请求的标志，可用于日志记录或类似目的。

`url`

包含此请求的URL的字符串。请记住，此属性包含转义的URL，因此它可能与构造函数中传递的URL不同。

该属性是只读的。更改请求使用的URL `replace()` 。

`method`

表示请求中的HTTP方法的字符串。这保证是大写的。例如： `"GET"` ， `"POST"` ， `"PUT"` ，等

`headers`

类似字典的对象，包含请求标头。

body

包含请求正文的str。

该属性是只读的。更改请求使用的正文 `replace()` 。

meta

包含此请求的任意元数据的字典。对于新的请求，此dict为空，并且通常由不同的Scrapy组件（扩展，中间件等）填充。因此，此dict中包含的数据取决于您启用的扩展。

有关Scrapy识别的特殊元键列表，请参阅Request.meta特殊键。

当使用或方法克隆请求时， 此dict会被浅层复制，也可以在您的蜘蛛中从属性中访问。 `copy()` `replace()` `response.meta`

copy ( )

返回一个新请求，该请求是此请求的副本。另请参阅：将 其他数据传递给回调函数。

replace ([*URL , 方法 , 头 , 主体 , 饼干 , 元 , 编码 , dont_filter , 回调 , errback可*])

返回具有相同成员的Request对象，但通过指定的任何关键字参数给出新值的成员除外。 `Request.meta` 默认情况下复制该属性（除非在 `meta` 参数中给出新值）。另请参阅将 其他数据传递给回调函数。

将附加数据传递给回调函数

请求的回调是在下载该请求的响应时将被调用的函数。将使用下载的 `Response` 对象作为其第一个参数调用回调函数。

例：

```
def parse_page1(self, response):
    return scrapy.Request("http://www.example.com/some_page.html",
                          callback=self.parse_page2)

def parse_page2(self, response):
    # this would log http://www.example.com/some_page.html
    self.logger.info("Visited %s", response.url)
```

在某些情况下，您可能有兴趣将参数传递给那些回调函数，以便稍后在第二个回调中接收参数。您可以使用该 `Request.meta` 属性。

以下是如何使用此机制传递项目以填充不同页面中的不同字段的示例：

```
def parse_page1(self, response):
    item = MyItem()
    item['main_url'] = response.url
    request = scrapy.Request("http://www.example.com/some_page.html",
                             callback=self.parse_page2)
    request.meta['item'] = item
    yield request

def parse_page2(self, response):
    item = response.meta['item']
    item['other_url'] = response.url
    yield item
```

使用errbacks来捕获请求处理中的异常

请求的错误回送是在处理异常时将调用的函数。

它接收Twisted Failure实例作为第一个参数，可用于跟踪连接建立超时，DNS错误等。

这是蜘蛛记录所有错误并在需要时捕获一些特定错误的示例：

```
import scrapy

from scrapy.spidermiddlewares.httperror import HttpError
from twisted.internet.error import DNSLookupError
from twisted.internet.error import TimeoutError, TCPTimedOutError

class ErrbackSpider(scrapy.Spider):
    name = "errback_example"
    start_urls = [
        "http://www.httpbin.org/",              # HTTP 200 expected
        "http://www.httpbin.org/status/404",    # Not found error
        "http://www.httpbin.org/status/500",    # server issue
        "http://www.httpbin.org:12345/",        # non-responding host, timeout expected
        "http://www.httphttpbinbin.org/",       # DNS error expected
    ]

    def start_requests(self):
        for u in self.start_urls:
            yield scrapy.Request(u, callback=self.parse_httpbin,
                                 errback=self.errback_httpbin,
                                 dont_filter=True)

    def parse_httpbin(self, response):
        self.logger.info('Got successful response from {}'.format(response.url))
        # do something useful here...

    def errback_httpbin(self, failure):
        # log all failures
        self.logger.error(repr(failure))

        # in case you want to do something special for some errors,
        # you may need the failure's type:

        if failure.check(HttpError):
            # these exceptions come from HttpError spider middleware
            # you can get the non-200 response
            response = failure.value.response
            self.logger.error('HttpError on %s', response.url)

        elif failure.check(DNSLookupError):
            # this is the original request
            request = failure.request
            self.logger.error('DNSLookupError on %s', request.url)

        elif failure.check(TimeoutError, TCPTimedOutError):
            request = failure.request
            self.logger.error('TimeoutError on %s', request.url)
```

Request.meta特殊键

该`Request.meta`属性可以包含任意数据，但Scrapy及其内置扩展可识别一些特殊键。

那些是：

- `dont_redirect`
- `dont_retry`
- `handle_httpstatus_list`
- `handle_httpstatus_all`
- `dont_merge_cookies`
- `cookiejar`
- `dont_cache`
- `redirect_urls`

- `bindaddress`
- `dont_obey_robotstxt`
- `download_timeout`
- `download_maxsize`
- `download_latency`
- `download_fail_on_dataloss`
- `proxy`
- `ftp_user`（参见`FTP_USER`更多信息）
- `ftp_password`（参见`FTP_PASSWORD`更多信息）
- `referrer_policy`
- `max_retry_times`

bindaddress

用于执行请求的传出IP地址的IP。

download_timeout

下载器在超时之前等待的时间（以秒为单位）。另见：`DOWNLOAD_TIMEOUT`。

download_latency

自请求启动以来，获取响应所花费的时间，即通过网络发送的HTTP消息。只有在下载响应时，此元键才可用。虽然大多数其他元键用于控制Scrapy行为，但这个元素应该是只读的。

download_fail_on_dataloss

是否在破坏的回复中失败。见：`DOWNLOAD_FAIL_ON_DATALOSS`。

max_retry_times

元键用于每个请求的设置重试次数。初始化时，`max_retry_times`元键优先于`RETRY_TIMES`设置。

请求子类

这是内置`Request`子类的列表。您还可以将其子类化以实现您自己的自定义功能。

FormRequest对象

FormRequest类扩展了基础`Request`，具有处理HTML表单的功能。它使用lxml.html表单 来预先填充表单字段，其中包含来自`Response`对象的表单数据。

class **scrapy.http.FormRequest** (*url* [, *formdata* , ...])

本 `FormRequest` 类增加了新的参数构造函数。其余参数与 `Request` 类相同，此处未记录。

参数： **formdata**（*元组的dict 或iterable*）- 是一个包含HTML表单数据的字典（或（key，value）元组的迭代），它将被url编码并分配给请求的主体。

该 `FormRequest` 对象支持除标准以下类方法 `Request` 的方法：

> *classmethod* **from_response**(*response* [, *formname = None* , *formid = None* , *formnumber = 0* , *formdata = None* , *formxpath = None* , *formcss = None* , *clickdata = None* , *dont_click = False* , ...])

返回一个新 `FormRequest` 对象，其表单字段值预先填充 `<form>` 在给定响应中包含的HTML 元素中找到的对象。有关示例，请参阅 使用FormRequest.from_response（）来模拟用户登录。

该策略是默认情况下在任何看起来可点击的表单控件上自动模拟单击，如a 。尽管这非常方便，而且通常是所需的行为，但有时它可能会导致难以调试的问题。例如，在处理使用javascript填充和/或提交的表单时，默认行为可能不是最合适的。要禁用此行为，您可以将参数设置 为。此外，如果要更改单击的控件（而不是禁用它），您还可以使用该 参数。 `<input type="submit">` `from_response()` `dont_click` `True` `clickdata`

警告

将此方法与选项值中包含前导或尾随空格的select元素一起使用将不起作用，因为lxml中的 错误应该在lxml 3.8及更高版本中修复。

参数：

- **response**（ `Response` object）- 包含HTML表单的响应，该表单将用于预填充表单字段
- **formname**（*string*）- 如果给定，将使用name属性设置为此值的表单。
- **formid**（*string*）- 如果给定，将使用id属性设置为此值的表单。
- **formxpath**（*string*）- 如果给定，将使用与xpath匹配的第一个表单。
- **formcss**（*string*）- 如果给定，将使用与css选择器匹配的第一个表单。
- **formnumber**（*整数*）- 当响应包含多个表单时要使用的表单数。第一个（也是默认值）是 `0` 。
- **formdata**（*dict*）- 要在表单数据中覆盖的字段。如果响应 `<form>` 元素中已存在某个字段，则该值将被此参数中传递的值覆盖。如果此参数中传递的值为 `None` ，则该字段将不会包含在请求中，即使它存在于response `<form>` 元素中也是如此。
- **clickdata**（*dict*）- 用于查找单击控件的属性。如果没有给出，将提交表单数据，模拟第一个可点击元素的点击。除了html属性之外，还可以通过属性，通过其相对于表单内其他可提交输入的从零开始的索引来标识控件 `nr` 。
- **dont_click**（*boolean*）- 如果为True，将提交表单数据而不单击任何元素。

此类方法的其他参数直接传递给 `FormRequest` 构造函数。

*在新版本0.10.3：*该 `formname` 参数。

*在新版本0.17：*该 `formxpath` 参数。

*新的版本1.1.0：*该 `formcss` 参数。

*新的版本1.1.0：*该 `formid` 参数。

请求用法示例

使用FormRequest通过HTTP POST发送数据

如果要在蜘蛛中模拟HTML表单POST并发送几个键值字段，可以返回一个 `FormRequest` 对象（来自您的蜘蛛），如下所示：

```
return [FormRequest(url="http://www.example.com/post/action",
                    formdata={'name': 'John Doe', 'age': '27'},
                    callback=self.after_post)]
```

使用FormRequest.from_response（）来模拟用户登录

网站通常通过元素提供预先填充的表单字段，例如会话相关数据或身份验证令牌（用于登录页面）。在抓取时，您需要自动预先填充这些字段，并仅覆盖其中的一些字段，例如用户名和密码。您可以将此 方法用于此作业。这是一个使用它的示例蜘蛛：`<input type="hidden">` `FormRequest.from_response()`

```
import scrapy

class LoginSpider(scrapy.Spider):
    name = 'example.com'
    start_urls = ['http://www.example.com/users/login.php']

    def parse(self, response):
        return scrapy.FormRequest.from_response(
            response,
            formdata={'username': 'john', 'password': 'secret'},
            callback=self.after_login
        )

    def after_login(self, response):
        # check login succeed before going on
        if "authentication failed" in response.body:
            self.logger.error("Login failed")
            return

        # continue scraping with authenticated session...
```

响应对象

class **scrapy.http.Response** (*url* [, *status = 200* , *headers = None* , *body = b''* , *flags = None* , *request = None*])

甲 `Response` 对象表示的HTTP响应，这通常是下载（由下载器）并馈送到蜘蛛进行处理。

参数：

- **url**（*字符串*）- 此响应的URL
- **status**（*整数*）- 响应的HTTP状态。默认为 `200`。
- **headers**（*dict*）- 此响应的标头。dict值可以是字符串（对于单值标头）或列表（对于多值标头）。
- **body**（*字节*）- 响应主体。要将解码后的文本作为str（Python 2中的unicode）访问，您可以使用 `response.text` 来自编码感知的 Response子类，例如 `TextResponse`。
- **flags**（*列表*）- 是包含 `Response.flags` 属性初始值的列表。如果给定，列表将被浅层复制。
- **request**（ `Request` object）- `Response.request` 属性的初始值。这表示 `Request` 生成此响应的内容。

`url`

包含响应URL的字符串。

该属性是只读的。更改响应使用的URL `replace()`。

`status`

表示响应的HTTP状态的整数。示例： `200`， `404`。

`headers`

类似字典的对象，包含响应头。可以使用 `get()` 返回具有指定名称的第一个标头值来访问值，或者 `getlist()` 返回具有指定名称的所有标头值。例如，此调用将为您提供标题中的所有Cookie：

```
response.headers.getlist('Set-Cookie')
```

`body`

这个回复的正文。请记住，Response.body始终是一个字节对象。如果您想使用unicode版本 `TextResponse.text`（仅在 `TextResponse` 和子类中可用）。

该属性是只读的。更改响应使用的正文 `replace()`。

`request`

`Request` 生成此响应的对象。在响应和请求通过所有Downloader Middleware之后，此属性在Scrapy引擎中分配。特别是，这意味着：

- HTTP重定向将导致原始请求（重定向前的URL）被分配给重定向的响应（重定向后的最终URL）。
- Response.request.url并不总是等于Response.url

- 此属性仅在蜘蛛代码和Spider Middleware可用，但在下载器中间件中不可用（尽管您通过其他方式可以获得请求）和 `response_downloaded` 信号的处理程序。

> **meta**
>
> 对象 `Request.meta` 属性的 快捷方式 `Response.request` （即 `self.request.meta` ）。
>
> 与 `Response.request` 属性不同，属性 `Response.meta` 沿着重定向和重试传播，因此您将获得 `Request.meta` 从蜘蛛发送的原始属性。
>
> **也可以看看**
>
> `Request.meta` 属性

> **flags**
>
> 包含此响应标志的列表。标志是用于标记响应的标签。例如：*'cached'*，*'redirected'*等。它们显示在响应（*__str__* 方法）的字符串表示中，引擎用于记录。

> **copy ( )**
>
> 返回一个新的Response，它是此Response的副本。

> **replace ([*url* , *status* , *headers* , *body* , *request* , *flags* , *cls*])**
>
> 返回具有相同成员的Response对象，但通过指定的任何关键字参数给定新值的成员除外。 `Response.meta` 默认情况下复制该属性。

> **urljoin (*url*)**
>
> 通过将Response `url` 与可能的相对URL 组合来构造绝对URL。
>
> 这是urlparse.urljoin的包装器，它只是进行此调用的别名：
>
> ```
> urlparse.urljoin(response.url, url)
> ```

> **follow (*url* , *callback = None* , *method ='GET'* , *headers = None* , *body = None* , *cookies = None* , *meta = None* , *encoding ='utf-8'* , *priority = 0* , *dont_filter = False* , *errback = None*)**
>
> 返回一个 `Request` 实例以关注链接 `url` 。它接受与 `Request.__init__` 方法相同的参数，但 `url` 可以是相对URL或 `scrapy.link.Link` 对象，而不仅是绝对URL。
>
> `TextResponse` `follow()` 除了绝对/相对URL和链接对象之外，还提供了一种支持选择器的方法。

响应子类

以下是可用的内置Response子类列表。您还可以将Response类子类化以实现您自己的功能。

TextResponse对象

class scrapy.http.TextResponse (*url* [, *encoding* [, ...]])

`TextResponse`对象将编码功能添加到基`Response`类，该基 类仅用于二进制数据，例如图像，声音或任何媒体文件。

`TextResponse`除了基础`Response`对象之外，对象还支持新的构造函数参数。其余功能与`Response`课程相同，此处未记录。

参数： **encoding**（*string*） - 是一个字符串，其中包含用于此响应的编码。如果`TextResponse`使用unicode主体创建对象，则将使用此编码对其进行编码（请记住body属性始终为字符串）。如果`encoding`是`None`（默认值），则将在响应标头和正文中查找编码。

`TextResponse`除了标准属性之外，对象还支持以下属性`Response`：

text

响应体，作为unicode。

与之相同`response.body.decode(response.encoding)`，但结果在第一次调用后缓存，因此您可以`response.text`多次访问 而无需额外开销。

> **注意**
>
> `unicode(response.body)`不是将响应体转换为unicode的正确方法：您将使用系统默认编码（通常为*ascii*）而不是响应编码。

encoding

带有此响应编码的字符串。通过尝试以下机制解决编码，按顺序：

1. 在构造函数*编码*参数中传递的*编码*
2. 在Content-Type HTTP标头中声明的编码。如果此编码无效（即未知），则忽略该编码并尝试下一个解析机制。
3. 响应正文中声明的编码。TextResponse类不为此提供任何特殊功能。但是，`HtmlResponse`和`XmlResponse`类一样。
4. 通过查看响应体来推断编码。这是一个更脆弱的方法，但最后一个尝试。

selector

甲`Selector`使用响应作为目标实例。选择器在第一次访问时被懒惰地实例化。

`TextResponse`除了标准方法之外，对象还支持以下方法`Response`：

xpath（*查询*）

快捷方式 `TextResponse.selector.xpath(query)` ：

```
response.xpath('//p')
```

css (*查询*)

快捷方式 `TextResponse.selector.css(query)` ：

```
response.css('p')
```

follow (*url* , *callback = None* , *method ='GET'* , *headers = None* , *body = None* , *cookies = None* , *meta = None* , *encoding = None* , *priority = 0* , *dont_filter = False* , *errback = None*)

返回一个 `Request` 实例以关注链接 `url` 。它接受与 `Request.__init__` 方法相同的参数，但 `url` 不仅可以是绝对URL，还可以

- 相对URL;
- scrapy.link.Link对象（例如链接提取器结果）;
- 属性选择器（不是SelectorList） - 例如 `response.css('a::attr(href)')[0]` 或 `response.xpath('//img/@src')[0]` 。
- 选择器 `<a>` 或 `<link>` 元素，例如 `response.css('a.my_link')[0]` 。

请参阅创建用法示例请求的快捷方式。

body_as_unicode ( )

相同 `text` ，但作为方法可用。保留此方法以实现向后兼容; 请更喜欢 `response.text` 。

HtmlResponse对象

class **scrapy.http.HtmlResponse** (*url* [, ...])

该 `HtmlResponse` 班是的子类 `TextResponse` ，其增加了通过查看HTML编码自动发现支持 META HTTP-EQUIV属性。见 `TextResponse.encoding` 。

XmlResponse对象

class **scrapy.http.XmlResponse** (*url* [, ...])

该 `XmlResponse` 班是的子类 `TextResponse` ，其增加了通过查看XML声明行编码自动发现支持。见 `TextResponse.encoding` 。