下载中间件

下载器中间件是Scrapy的请求/响应处理的钩子框架。它是一个轻量级的低级系统，用于全局改变Scrapy的请求和响应。

激活下载中间件

要激活下载程序中间件组件，请将其添加到 `DOWNLOADER_MIDDLEWARES` 设置中，该设置是一个dict，其键是中间件类路径，其值是中间件命令。

这是一个例子：

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
}
```

该 `DOWNLOADER_MIDDLEWARES` 设置与 `DOWNLOADER_MIDDLEWARES_BASE` Scrapy中定义的设置合并（并不意味着被覆盖），然后按顺序排序，以获得已启用的中间件的最终排序列表：第一个中间件是靠近引擎的中间件，最后一个是更接近引擎的中间件到下载器。换句话说，`process_request()` 将以增加的中间件顺序（100,200,300，...）`process_response()` 调用每个中间件的方法，并且将按递减顺序调用每个中间件的方法。

要确定分配给中间件的顺序，请参阅 `DOWNLOADER_MIDDLEWARES_BASE` 设置并根据要插入中间件的位置选择值。订单很重要，因为每个中间件执行不同的操作，您的中间件可能依赖于应用的某些先前（或后续）中间件。

如果要禁用内置中间件（ `DOWNLOADER_MIDDLEWARES_BASE` 默认情况下定义和启用的中间件 ），则必须在项目的 `DOWNLOADER_MIDDLEWARES` 设置中定义它并将*None*指定 为其值。例如，如果要禁用用户代理中间件：

```
DOWNLOADER_MIDDLEWARES = {
    'myproject.middlewares.CustomDownloaderMiddleware': 543,
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
}
```

最后，请记住，可能需要通过特定设置启用某些中间件。有关详细信息，请参阅每个中间件文档。

编写自己的下载中间件

每个中间件组件都是一个Python类，它定义了以下一个或多个方法：

类 `scrapy.downloadermiddlewares.DownloaderMiddleware`

> **注意**
>
> 任何下载器中间件方法也可能返回延迟。

> **process_request（*请求*，*蜘蛛*）**
>
> 对于通过下载中间件的每个请求，都会调用此方法。
>
> `process_request()` 应该：返回 `None` ，返回一个 `Response` 对象，返回一个 `Request` 对象，或者提升 `IgnoreRequest` 。
>
> 如果它返回 `None` ，Scrapy将继续处理此请求，执行所有其他中间件，直到最后，相应的下载程序处理程序被称为执行的请求（并且其响应已下载）。
>
> 如果它返回一个 `Response` 对象，Scrapy将不会打扰*任何*其他 `process_request()` 或 `process_exception()` 方法，或适当的下载功能; 它会回复那个回应。 `process_response()` 安装中间件的方法总是在每个响应上调用。
>
> 如果它返回一个 `Request` 对象，Scrapy将停止调用process_request方法并重新安排返回的请求。执行新返回的请求后，将在下载的响应上调用相应的中间件链。
>
> 如果它引发 `IgnoreRequest` 异常，将 `process_exception()` 调用已安装的下载中间件的 方法。如果它们都不处理异常， `Request.errback` 则调用request（ ）的errback函数。如果没有代码处理引发的异常，则会忽略它并且不会记录（与其他异常不同）。
>
> **参数：**
> - **request**（ `Request` object） - 正在处理的请求
> - **spider**（ `Spider` object） - 此请求所针对的蜘蛛

> **process_response（*请求*，*响应*，*蜘蛛*）**
>
> `process_response()` 应该：返回一个 `Response` 对象，返回一个 `Request` 对象或引发一个 `IgnoreRequest` 异常。
>
> 如果它返回a `Response` （它可能是相同的给定响应，或者是全新的响应），则该响应将继续与 `process_response()` 链中的下一个中间件一起处理。
>
> 如果它返回一个 `Request` 对象，则暂停中间件链，并重新安排返回的请求以便将来下载。这与从中返回请求的行为相同 `process_request()` 。
>
> 如果它引发 `IgnoreRequest` 异常， `Request.errback` 则调用request（ ）的errback函数。如果没有代码处理引发的异常，则会忽略它并且不会记录（与其他异常不同）。

参数：
- **request**（是一个 `Request` 对象） - 发起响应的请求
- **response**（ `Response` object） - 正在处理的响应
- **spider**（ `Spider` object） - 此响应所针对的蜘蛛

`process_exception`（*请求*，*异常*，*蜘蛛*）

`process_exception()` 下载处理程序或 `process_request()` （从下载中间件）引发异常（包括 `IgnoreRequest` 异常）时Scrapy调用

`process_exception()` 应该返回：要么 `None` ，一个 `Response` 对象或 `Request` 对象。

如果它返回 `None` ，Scrapy将继续处理此异常，执行任何其他 `process_exception()` 已安装的中间件方法，直到没有剩下中间件并且默认异常处理开始。

如果它返回一个 `Response` 对象， `process_response()` 则启动已安装的中间件的方法链，并且Scrapy不会打扰调用任何其他 `process_exception()` 中间件方法。

如果它返回一个 `Request` 对象，则重新安排返回的请求以便将来下载。这会停止执行 `process_exception()` 中间件的方法，就像返回响应一样。

参数：
- **request**（是一个 `Request` 对象） - 生成异常的请求
- **exception**（一个 `Exception` 对象） - 引发的异常
- **spider**（ `Spider` object） - 此请求所针对的蜘蛛

`from_crawler`（*cls*，*crawler*）

如果存在，则调用此类方法以从a创建中间件实例 `Crawler` 。它必须返回一个新的中间件实例。Crawler对象提供对所有Scrapy核心组件的访问，如设置和信号; 它是中间件访问它们并将其功能挂钩到Scrapy的一种方式。

参数： crawler（ `Crawler` object） - 使用此中间件的爬网程序

内置下载中间件参考

此页面描述了Scrapy附带的所有下载中间件组件。有关如何使用它们以及如何编写自己的下载程序中间件的信息，请参阅下载程序中间件使用指南。

有关默认启用的组件列表（及其订单），请参阅 `DOWNLOADER_MIDDLEWARES_BASE` 设置。

CookiesMiddleware

类 `scrapy.downloadermiddlewares.cookies.CookiesMiddleware`

此中间件可用于处理需要cookie的站点，例如使用会话的站点。它跟踪Web服务器发送的cookie，并在后续请求（来自该蜘蛛）上发回它们，就像Web浏览器一样。

以下设置可用于配置cookie中间件：

- `COOKIES_ENABLED`
- `COOKIES_DEBUG`

每个蜘蛛多个cookie会话

0.15版本的新功能。

通过使用`cookiejar` Request meta键，支持为每个蜘蛛保留多个cookie会话 。默认情况下，它使用单个cookie jar（会话），但您可以传递标识符以使用不同的cookie。

例如：

```
for i, url in enumerate(urls):
    yield scrapy.Request(url, meta={'cookiejar': i},
        callback=self.parse_page)
```

请记住，`cookiejar`元键不是“粘性”。您需要在后续请求中继续传递它。例如：

```
def parse_page(self, response):
    # do some processing
    return scrapy.Request("http://www.example.com/otherpage",
        meta={'cookiejar': response.meta['cookiejar']},
        callback=self.parse_other_page)
```

COOKIES_ENABLED

默认： `True`

是否启用cookie中间件。如果禁用，则不会将cookie发送到Web服务器。

请注意，尽管`COOKIES_ENABLED`设置值如果`Request.` `meta['dont_merge_cookies']`评估`True`请求，cookie将**不会**发送到Web服务器，并且收到的cookie `Response`将**不会**与现有cookie合并。

有关更多详细信息，请参阅`cookies`参数`Request`。

COOKIES_DEBUG

默认： `False`

如果启用，Scrapy将记录请求中发送的所有cookie（即`Cookie`标题）和响应中收到的所有cookie（即`Set-Cookie`标题）。

以下是 `COOKIES_DEBUG` 启用日志的示例：

```
2011-04-06 14:35:10-0300 [scrapy.core.engine] INFO: Spider opened
2011-04-06 14:35:10-0300 [scrapy.downloadermiddlewares.cookies] DEBUG: Sending cookies to: <GET
http://www.diningcity.com/netherlands/index.html>
        Cookie: clientlanguage_nl=en_EN
2011-04-06 14:35:14-0300 [scrapy.downloadermiddlewares.cookies] DEBUG: Received cookies from:
<200 http://www.diningcity.com/netherlands/index.html>
        Set-Cookie: JSESSIONID=B~FA4DC0C496C8762AE4F1A620EAB34F38; Path=/
        Set-Cookie: ip_isocode=US
        Set-Cookie: clientlanguage_nl=en_EN; Expires=Thu, 07-Apr-2011 21:21:34 GMT; Path=/
2011-04-06 14:49:50-0300 [scrapy.core.engine] DEBUG: Crawled (200) <GET
http://www.diningcity.com/netherlands/index.html> (referer: None)
[...]
```

DefaultHeadersMiddleware

类 scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware

此中间件设置设置中指定的所有默认请求标头 `DEFAULT_REQUEST_HEADERS`。

DownloadTimeoutMiddleware

类 scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware

此中间件为 `DOWNLOAD_TIMEOUT` 设置或 `download_timeout` spider属性中指定的请求设置下载超时。

注意

您还可以使用 `download_timeout` Request.meta键设置每个请求的下载超时；即使禁用DownloadTimeoutMiddleware，也支持此功能。

HttpAuthMiddleware

类 scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware

此中间件使用基本访问身份验证（也称为HTTP身份验证）对从某些蜘蛛生成的所有请求进行身份验证。

要从某些蜘蛛启用HTTP身份验证，请设置这些蜘蛛的 `http_user` 和 `http_pass` 属性。

例：

```
from scrapy.spiders import CrawlSpider

class SomeIntranetSiteSpider(CrawlSpider):

    http_user = 'someuser'
    http_pass = 'somepass'
    name = 'intranet.example.com'

    # .. rest of the spider code omitted ...
```

HttpCacheMiddleware

类 `scrapy.downloadermiddlewares.httpcache.HttpCacheMiddleware`

此中间件为所有HTTP请求和响应提供低级缓存。它必须与缓存存储后端以及缓存策略结合使用。

Scrapy附带三个HTTP缓存存储后端：

- 文件系统存储后端（默认）
- DBM存储后端
- LevelDB存储后端

您可以使用该 `HTTPCACHE_STORAGE` 设置更改HTTP缓存存储后端。或者您也可以实现自己的存储后端。

Scrapy附带两个HTTP缓存策略：

- RFC2616政策
- 虚拟政策（默认）

您可以使用该 `HTTPCACHE_POLICY` 设置更改HTTP缓存策略。或者您也可以实施自己的政策。

您还可以使用 `dont_cache` 元键等于*True*来避免在每个策略上缓存响应。

虚拟政策（默认）

此策略不了解任何HTTP Cache-Control指令。每个请求及其相应的响应都被缓存。当再次看到相同的请求时，将返回响应，而不从Internet传输任何内容。

Dummy策略对于更快地测试蜘蛛非常有用（无需每次都等待下载），并且当Internet连接不可用时，可以离线尝试蜘蛛。我们的目标是能够“重播”蜘蛛来看，*正是因为它跑前*。

要使用此政策，请设置：

- `HTTPCACHE_POLICY` 至 `scrapy.extensions.httpcache.DummyPolicy`

RFC2616策略

此策略提供符合RFC2616的HTTP缓存，即具有HTTP缓存控制感知，旨在生产并用于连续运行，以避免下载未修改的数据（以节省带宽和加速爬网）。

实施的内容：

- 不要尝试使用*no-store* cache-control指令集存储响应/请求

- 如果为新响应设置了*no-cache* cache-control指令，则不要从缓存提供响应
- 从*max-age* cache-control指令计算新鲜度生命周期
- 从*Expires*响应头计算新鲜度生命周期
- 从*Last-Modified*响应头开始计算新鲜度生命周期（Firefox使用的启发式）
- 从*Age*响应标头计算当前年龄
- 从*Date*标头计算当前年龄
- 根据*Last-Modified*响应头重新验证过时响应
- 根据*ETag*响应头重新验证过时响应
- 为任何收到的响应设置*日期*标题
- 支持请求中的*max-stale* cache-control指令

  这允许蜘蛛配置完整的RFC2616缓存策略，但避免在逐个请求的基础上重新验证，同时保持符合HTTP规范。

  例：

  添加*Cache-Control：max-stale = 600*到Request标头，以接受超过其到期时间不超过600秒的响应。

  另见：RFC2616,14.9.3

缺什么：

- *Pragma：无缓存*支持https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.1
- *不同的*标题支持https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.6
- 更新或删除后失效https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.10
- .....可能还有其他..

要使用此政策，请设置：

- `HTTPCACHE_POLICY` 至 `scrapy.extensions.httpcache.RFC2616Policy`

文件系统存储后端（默认）

文件系统存储后端可用于HTTP缓存中间件。

要使用此存储后端，请设置：

- `HTTPCACHE_STORAGE` 至 `scrapy.extensions.httpcache.FilesystemCacheStorage`

每个请求/响应对都存储在包含以下文件的不同目录中：

- `request_body` - 普通请求机构
- `request_headers` - 请求标头（原始HTTP格式）
- `response_body` - 简单的回应机构
- `response_headers` - 请求标头（原始HTTP格式）
- `meta` - Python `repr()` 格式的缓存资源的一些元数据（grep友好格式）
- `pickled_meta` - 相同的元数据，`meta` 但为更有效的反序列化而腌制

目录名称来自请求指纹（请参阅参考资料 `scrapy.utils.request.fingerprint` ），并且使用一个级别的子目录来避免在同一目录中创建太多文件（这在许多文件系统中效率低下）。示例目录可以是：

```
/path/to/cache/dir/example.com/72/72811f648e718090f041317756c03adb0ada46c7
```

DBM存储后端

版本0.13中的新功能。

一个DBM存储后端也可用于HTTP缓存中间件。

默认情况下，它使用anydbm模块，但您可以使用该 `HTTPCACHE_DBM_MODULE` 设置更改它。

要使用此存储后端，请设置：

- `HTTPCACHE_STORAGE` 至 `scrapy.extensions.httpcache.DbmCacheStorage`

LevelDB存储后端

版本0.23中的新功能。

一个性LevelDB存储后端也可用于HTTP缓存中间件。

建议不要将此后端用于开发，因为只有一个进程可以同时访问LevelDB数据库，因此您无法运行爬网并为同一个蜘蛛并行打开scrapy shell。

要使用此存储后端：

- 设置 `HTTPCACHE_STORAGE` 为 `scrapy.extensions.httpcache.LeveldbCacheStorage`
- 安装LevelDB python绑定之类的 `pip install leveldb`

HTTPCache中间件设置

的 `HttpCacheMiddleware` 可以通过以下设置来配置：

HTTPCACHE_ENABLED

版本0.11中的新功能。

默认： `False`

是否启用HTTP缓存。

*在版本0.11中更改：*在0.11之前， `HTTPCACHE_DIR` 用于启用缓存。

HTTPCACHE_EXPIRATION_SECS

默认： `0`

缓存请求的到期时间，以秒为单位。

超过此时间的缓存请求将被重新下载。如果为零，则缓存的请求将永不过期。

*在版本0.11中更改：*在0.11之前，零表示缓存的请求始终过期。

HTTPCACHE_DIR

默认： `'httpcache'`

用于存储（低级）HTTP缓存的目录。如果为空，则将禁用HTTP缓存。如果给出相对路径，则相对于项目数据dir。有关详细信息，请参阅：Scrapy项目的默认结构。

HTTPCACHE_IGNORE_HTTP_CODES

版本0.10中的新功能。

默认： `[]`

不要使用这些HTTP代码缓存响应。

HTTPCACHE_IGNORE_MISSING

默认： `False`

如果启用，将忽略未在缓存中找到的请求而不是下载。

HTTPCACHE_IGNORE_SCHEMES

版本0.10中的新功能。

默认： `['file']`

不要使用这些URI方案缓存响应。

HTTPCACHE_STORAGE

默认： `'scrapy.extensions.httpcache.FilesystemCacheStorage'`

实现缓存存储后端的类。

HTTPCACHE_DBM_MODULE

版本0.13中的新功能。

默认： `'anydbm'`

要在DBM存储后端中使用的数据库模块。此设置特定于DBM后端。

HTTPCACHE_POLICY

版本0.18中的新功能。

默认： `'scrapy.extensions.httpcache.DummyPolicy'`

实现缓存策略的类。

HTTPCACHE_GZIP

版本1.0中的新功能。

默认： `False`

如果启用，将使用gzip压缩所有缓存的数据。此设置特定于Filesystem后端。

HTTPCACHE_ALWAYS_STORE

版本1.1中的新功能。

默认： `False`

如果启用，将无条件地缓存页面。

蜘蛛可能希望在缓存中提供所有响应，以便将来使用*Cache-Control：max-stale*。DummyPolicy缓存所有响应，但从不重新验证它们，有时需要更细微的策略。

此设置仍然遵循*Cache-Control：*响应中的*no-store*指令。如果您不想这样做，请在您提供给缓存中间件的响应中的Cache-Control标头中过滤*no-store*。

HTTPCACHE_IGNORE_RESPONSE_CACHE_CONTROLS

版本1.1中的新功能。

默认： `[]`

要忽略的响应中的Cache-Control指令列表。

站点通常设置“无存储”，“无缓存”，“必须重新验证”等，但如果它遵守这些指令，蜘蛛可以生成的流量会感到不安。这允许有选择地忽略已知对正在被爬网的站点不重要的Cache-Control指令。

我们假设蜘蛛不会在请求中发出Cache-Control指令，除非它实际需要它们，因此不会过滤请求中的指令。

HttpCompressionMiddleware

类 **scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware**

该中间件允许从网站发送/接收压缩（gzip，deflate）流量。

如果安装了brotlipy，此中间件还支持解码brotli压缩响应。

HttpCompressionMiddleware设置

COMPRESSION_ENABLED

默认： `True`

是否启用压缩中间件。

HttpProxyMiddleware

版本0.8中的新功能。

类 **scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware**

此中间件通过设置对象的 `proxy` 元值来设置用于请求的HTTP代理 `Request`。

与Python标准库模块urllib和urllib2一样，它遵循以下环境变量：

- `http_proxy`
- `https_proxy`
- `no_proxy`

您还可以将 `proxy` 每个请求的元键设置为类似 `http://some_proxy_server:port` 或的值 `http://username:password@some_proxy_server:port` 。请记住，此值将优先于 `http_proxy` / `https_proxy` environment变量，并且还将忽略 `no_proxy` 环境变量。

RedirectMiddleware

类 scrapy.downloadermiddlewares.redirect.RedirectMiddleware

该中间件根据响应状态处理请求的重定向。

可以在 `redirect_urls` `Request.meta` 密钥中找到请求经过的URL（在重定向时）。

该 `RedirectMiddleware` 可通过以下设置进行配置（详情参见设置文档）：

- `REDIRECT_ENABLED`
- `REDIRECT_MAX_TIMES`

如果 `Request.meta` 将 `dont_redirect` 键设置为True，则此中间件将忽略该请求。

如果要在spider中处理一些重定向状态代码，可以在 `handle_httpstatus_list` spider属性中指定这些代码。

例如，如果您希望重定向中间件忽略301和302响应（并将它们传递给您的蜘蛛），您可以这样做：

```
class MySpider(CrawlSpider):
    handle_httpstatus_list = [301, 302]
```

所述 `handle_httpstatus_list` 的键 `Request.meta` 也可以被用于指定的响应代码，以允许在每个请求基础。您还可以设置meta键 `handle_httpstatus_all` 来 `True` ，如果你想以允许请求的任何响应代码。

RedirectMiddleware设置

REDIRECT_ENABLED

版本0.13中的新功能。

默认： `True`

是否启用Redirect中间件。

REDIRECT_MAX_TIMES

默认： `20`

单个请求将遵循的最大重定向数。

MetaRefreshMiddleware

类 scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware

此中间件基于元刷新html标记处理请求的重定向。

该 `MetaRefreshMiddleware` 可通过以下设置进行配置（详情参见设置文档）：

- `METAREFRESH_ENABLED`
- `METAREFRESH_MAXDELAY`

该中间件服从 `REDIRECT_MAX_TIMES` 设置， `dont_redirect` 并 `redirect_urls` 按照描述请求元密钥 `RedirectMiddleware`

MetaRefreshMiddleware设置

METAREFRESH_ENABLED

版本0.17中的新功能。

默认： `True`

是否启用Meta Refresh中间件。

METAREFRESH_MAXDELAY

默认： `100`

跟随重定向的最大元刷新延迟（以秒为单位）。有些站点使用元刷新来重定向到会话过期页面，因此我们将自动重定向限制为最大延迟。

RetryMiddleware

类 scrapy.downloadermiddlewares.retry.RetryMiddleware

一种中间件，用于重试可能由临时问题（如连接超时或HTTP 500错误）引起的失败请求。

一旦蜘蛛完成了对所有常规（非失败）页面的爬行，在抓取过程中收集失败的页面并在结束时重新安排。一旦没有更多的失败页面重试，该中间件就会发送一个信号（retry_complete），因此其他扩展可以连接到该信号。

该 `RetryMiddleware` 可通过以下设置进行配置（详情参见设置文档）：

- `RETRY_ENABLED`
- `RETRY_TIMES`
- `RETRY_HTTP_CODES`

如果 `Request.meta` 将 `dont_retry` 键设置为True，则此中间件将忽略该请求。

重试中间件设置

RETRY_ENABLED

版本0.13中的新功能。

默认： `True`

是否启用重试中间件。

RETRY_TIMES

默认： `2`

除第一次下载外，最多重试次数。

每个请求也可以使用 `max_retry_times` 属性指定最大重试次数 `Request.meta`。初始化时， `max_retry_times` 元键优先于 `RETRY_TIMES` 设置。

RETRY_HTTP_CODES

默认： `[500, 502, 503, 504, 408]`

要重试的HTTP响应代码。始终重试其他错误（DNS查找问题，连接丢失等）。

在某些情况下，您可能希望添加400， `RETRY_HTTP_CODES` 因为它是用于指示服务器过载的公共代码。默认情况下不包括它，因为HTTP规范说明了这一点。

RobotsTxtMiddleware

类 scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware

要确保Scrapy尊重robots.txt，请确保已启用中间件并启用该 `ROBOTSTXT_OBEY` 设置。

如果 `Request.meta` 将 `dont_obey_robotstxt` 密钥设置为True，则即使 `ROBOTSTXT_OBEY` 启用了该中间件，该请求也将被忽略 。

DownloaderStats

类 `scrapy.downloadermiddlewares.stats.DownloaderStats`

中间件，用于存储通过它的所有请求，响应和异常的统计信息。

要使用此中间件，您必须启用该 `DOWNLOADER_STATS` 设置。

UserAgentMiddleware

类 `scrapy.downloadermiddlewares.useragent.UserAgentMiddleware`

允许蜘蛛覆盖默认用户代理的中间件。

为了使蜘蛛覆盖默认用户代理， 必须设置其*user_agent*属性。

AjaxCrawlMiddleware

类 `scrapy.downloadermiddlewares.ajaxcrawl.AjaxCrawlMiddleware`

基于元片段html标记找到“AJAX可抓取”页面变体的中间件。有关 详细信息，请参阅 https://developers.google.com/webmasters/ajax-crawling/docs/getting-started。

注意

`'http://example.com/!#foo=bar'` 即使没有这个中间件，Scrapy也会为URL找到“AJAX可抓取”页面 。当URL不包含时，AjaxCrawlMiddleware是必需的 `'!#'` 。这通常是“索引”或“主要”网站页面的情况。

AjaxCrawlMiddleware设置

AJAXCRAWL_ENABLED

版本0.21中的新功能。

默认： `False`

是否启用AjaxCrawlMiddleware。您可能希望为广泛爬网启用它。

HttpProxyMiddleware设置

HTTPPROXY_ENABLED

默认： `True`

是否启用`HttpProxyMiddleware`。

HTTPPROXY_AUTH_ENCODING

默认： `"latin-1"`

代理身份验证的默认编码`HttpProxyMiddleware`。