核心

0.15版本的新功能。

本节介绍Scrapy核心API，它适用于扩展和中间件的开发人员。

抓取工具

Scrapy API的主要入口点是 `Crawler` 对象，通过 `from_crawler` 类方法传递给扩展。此对象提供对所有Scrapy核心组件的访问，这是扩展访问它们并将其功能挂钩到Scrapy的唯一方法。

Extension Manager负责加载和跟踪已安装的扩展，并通过 `EXTENSIONS` 设置配置它，其中包含所有可用扩展的字典及其顺序，类似于您配置下载中间件的方式。

class **scrapy.crawler.Crawler** (*spidercls*, *settings*)

必须使用 `scrapy.spiders.Spider` 子类和 `scrapy.settings.Settings` 对象实例化Crawler 对象。

settings

此搜寻器的设置管理器。

扩展和中间件使用它来访问此爬网程序的Scrapy设置。

有关Scrapy设置的介绍，请参阅设置。

对于API，请参阅 `Settings` 类。

signals

此爬虫的信号管理器。

扩展和中间件使用它来将自己挂钩到Scrapy功能。

有关信号的介绍，请参阅信号。

对于API，请参阅 `SignalManager` 类。

stats

此爬虫的统计信息收集器。

这可以从扩展和中间件中使用，以记录其行为的统计信息，或访问其他扩展程序收集的统计信息。

有关统计数据收集的介绍，请参阅统计数据集。

对于API，请参阅`StatsCollector`类。

`extensions`

跟踪已启用扩展的扩展管理器。

大多数扩展程序不需要访问此属性。

有关扩展的介绍和Scrapy上可用扩展的列表，请参阅扩展。

`engine`

执行引擎，它协调调度程序，下载程序和蜘蛛之间的核心爬行逻辑。

某些扩展可能希望访问Scrapy引擎，检查或修改下载程序和调度程序行为，尽管这是一种高级用法，并且此API尚不稳定。

`spider`

Spider目前正在被抓获。这是构造爬网程序时提供的spider类的实例，它是在`crawl()`方法中给出的参数之后创建的。

`crawl` (** args* , *** kwargs*)

通过使用给定的*args*和*kwargs*参数实例化其spider类，同时将执行引擎设置为运动来启动爬网程序 。

返回爬网结束时触发的延迟。

---

class **`scrapy.crawler.CrawlerRunner`** (*settings = None*)

这是一个方便的助手类，可以在已设置的Twisted 反应器中跟踪，管理和运行爬虫。

必须使用`Settings`对象实例化CrawlerRunner 对象。

除非编写手动处理爬网过程的脚本，否则不应该需要此类（因为Scrapy负责相应地使用它）。有关示例，请参阅脚本中的Run Scrapy。

`crawl` (*crawler_or_spidercls* , ** args* , *** kwargs*)

使用提供的参数运行爬网程序。

它会调用给定的Crawler `crawl()`方法，同时跟踪它，以便以后可以停止。

如果*crawler_or_spidercls*不是`Crawler`实例，则此方法将尝试使用此参数创建一个作为赋予它的spider类。

返回爬网结束时触发的延迟。

参数：
- **crawler_or_spidercls**（`Crawler`实例，`Spider`子类或字符串）- 已创建的爬虫，或项目内的蜘蛛类或蜘蛛的名称来创建它
- **args**（*list*）- 初始化蜘蛛的参数
- **kwargs**（*dict*）- 初始化蜘蛛的关键字参数

crawlers

由这个类`crawlers`开始`crawl()`并管理的集合。

create_crawler(*crawler_or_spidercls*)

返回一个`Crawler`对象。

- 如果*crawler_or_spidercls*是Crawler，*则按原样*返回。
- 如果*crawler_or_spidercls*是Spider子类，则为其构造新的Crawler。
- 如果*crawler_or_spidercls*是一个字符串，则此函数在Scrapy项目中使用此名称查找蜘蛛（使用蜘蛛加载程序），然后为其创建一个Crawler实例。

join()

返回当所有托管`crawlers`已完成执行时触发的延迟。

stop()

同时停止所有正在进行的爬行作业。

返回在它们全部结束时触发的延迟。

---

class **scrapy.crawler.CrawlerProcess**(*settings = None，install_root_handler = True*)

基地：`scrapy.crawler.CrawlerRunner`

一个同时在进程中运行多个scrapy爬虫的类。

此类`CrawlerRunner`通过添加对启动Twisted reactor和处理关闭信号的支持来扩展，如键盘中断命令Ctrl-C。它还配置顶级日志记录。

与`CrawlerRunner`您未在应用程序中运行其他Twisted 反应器相比，此实用程序应该更合适。

必须使用`Settings`对象实例化CrawlerProcess 对象。

参数： **install_root_handler** - 是否安装root日志记录处理程序（默认值：True）

除非编写手动处理爬网过程的脚本，否则不应该需要此类（因为Scrapy负责相应地使用它）。有关示例，请参阅脚本中的Run Scrapy。

crawl(*crawler_or_spidercls，* *args，** *kwargs*)

使用提供的参数运行爬网程序。

它会调用给定的Crawler `crawl()`方法，同时跟踪它，以便以后可以停止。

如果*crawler_or_spidercls*不是`Crawler`实例，则此方法将尝试使用此参数创建一个作为赋予它的spider类。

返回爬网结束时触发的延迟。

参数：
- **crawler_or_spidercls**（`Crawler`实例，`Spider`子类或字符串）- 已创建的爬虫，或项目内的蜘蛛类或蜘蛛的名称来创建它
- **args**（*list*）- 初始化蜘蛛的参数
- **kwargs**（*dict*）- 初始化蜘蛛的关键字参数

`crawlers`

由这个类`crawlers`开始`crawl()`并管理的集合。

`create_crawler(crawler_or_spidercls)`

返回一个`Crawler`对象。

- 如果*crawler_or_spidercls*是Crawler，*则按原样*返回。
- 如果*crawler_or_spidercls*是Spider子类，则为其构造新的Crawler。
- 如果*crawler_or_spidercls*是一个字符串，则此函数在Scrapy项目中使用此名称查找蜘蛛（使用蜘蛛加载程序），然后为其创建一个Crawler实例。

`join()`

返回当所有托管`crawlers`已完成执行时触发的延迟。

`start(stop_after_crawl = True)`

此方法从一个扭曲的反应器，其调整池的大小来`REACTOR_THREADPOOL_MAXSIZE`，并安装基于DNS缓存`DNSCACHE_ENABLED`和`DNSCACHE_SIZE`。

如果*stop_after_crawl*为True，则在所有爬网程序完成后，反应堆将停止使用`join()`。

参数： **stop_after_crawl**（*boolean*）- 当所有爬虫都完成时停止或不停止反应器

`stop()`

同时停止所有正在进行的爬行作业。

返回在它们全部结束时触发的延迟。

设置

`scrapy.settings.SETTINGS_PRIORITIES`

设置Scrapy中使用的默认设置优先级的键名和优先级的字典。

每个项目定义一个设置入口点，为其提供标识的代码名称和整数优先级。在设置和检索`Settings`类中的值时，更高优先级优先于较小优先 级。

```
SETTINGS_PRIORITIES = {
    'default': 0,
    'command': 10,
    'project': 20,
    'spider': 30,
    'cmdline': 40,
}
```

有关每个设置源的详细说明，请参阅： 设置。

scrapy.settings.get_settings_priority（*优先*）

小辅助函数，它在`SETTINGS_PRIORITIES`字典中查找给定的字符串优先级 并返回其数值，或直接返回给定的数字优先级。

class scrapy.settings.Settings（*values = None，priority ='project'*）

基地： `scrapy.settings.BaseSettings`

此对象存储用于配置内部组件的Scrapy设置，并可用于任何进一步的自定义。

它是一个直接的子类，支持所有方法`BaseSettings`。此外，在实例化此类之后，新对象将具有已在内置设置引用中描述的全局默认设置。

class scrapy.settings.BaseSettings（*values = None，priority ='project'*）

此类的实例表现得像字典，但存储优先级及其对，并且可以被冻结（即标记为不可变）。`(key, value)`

键值条目可以在初始化时使用`values`参数传递，并且它们将采用该`priority`级别（除非`values`已经是实例`BaseSettings`，在这种情况下将保留现有的优先级）。如果`priority`参数是字符串，则将查找优先级名称`SETTINGS_PRIORITIES`。否则，应提供特定的整数。

创建对象后，可以使用该`set()`方法加载或更新新设置 ，并且可以使用字典的方括号表示法或`get()`实例的 方法及其值转换变体来访问。请求存储密钥时，将检索具有最高优先级的值。

copy（）

制作当前设置的深层副本。

此方法返回类的新实例，并`Settings`填充相同的值及其优先级。

对新对象的修改不会反映在原始设置上。

copy_to_dict（）

制作当前设置的副本并转换为dict。

此方法返回一个新的dict，其中填充了与当前设置相同的值及其优先级。

对返回的dict的修改不会反映在原始设置上。

例如，此方法可用于在Scrapy shell中打印设置。

`freeze()`

禁用对当前设置的进一步更改。

调用此方法后，设置的当前状态将变为不可变。试图通过该`set()`方法及其变体更改值是不可能的，并将收到警报。

`frozencopy()`

返回当前设置的不可变副本。

`freeze()`在返回的对象中调用的别名`copy()`。

`get(`*名称*`,` *默认=无*`)`

获取设置值而不影响其原始类型。

参数：
- name（*字符串*）- 设置名称
- default（*any*）- 如果未找到设置则返回的值

`getbool(`*name*`,` *default = False* `)`

获取设置值作为布尔值。

`1`，`'1'`，*TRUE*`和`'True'`回报`True`，同时`0`，`'0'`，`False`，`'False'`和`None`回报`False`。

例如，通过设置为的环境变量填充的设置`'0'`将`False`在使用此方法时返回。

参数：
- name（*字符串*）- 设置名称
- default（*any*）- 如果未找到设置则返回的值

`getdict(`*名称*`,` *默认=无*`)`

获取设置值作为字典。如果设置原始类型是字典，则将返回其副本。如果它是一个字符串，它将被评估为JSON字典。如果它是一个`BaseSettings`实例本身，它将被转换为一个字典，包含它们将返回的所有当前设置值`get()`，并丢失有关优先级和可变性的所有信息。

参数：
- **name**（*字符串*）- 设置名称
- **default**（*any*）- 如果未找到设置则返回的值

`getfloat(名称, 默认=0.0)`

获取设置值作为浮点数。

参数：
- **name**（*字符串*）- 设置名称
- **default**（*any*）- 如果未找到设置则返回的值

`getint(名称, 默认=0)`

获取设置值作为int。

参数：
- **name**（*字符串*）- 设置名称
- **default**（*any*）- 如果未找到设置则返回的值

`getlist(名称, 默认=无)`

获取设置值作为列表。如果设置原始类型是列表，则将返回其副本。如果它是一个字符串，它将被“，”拆分。

例如，通过设置为的环境变量填充的设置 `'one,two'` 将在使用此方法时返回列表['one'，'two']。

参数：
- **name**（*字符串*）- 设置名称
- **default**（*any*）- 如果未找到设置则返回的值

`getpriority(名字)`

返回设置的当前数字优先级值，或者 `None` 如果给定的 `name` 不存在。

参数： **name**（*字符串*）- 设置名称

`getwithbase(名字)`

获得类似字典的设置及其_BASE 对应的组合。

参数： **name**（*字符串*）- 类字典设置的名称

`maxpriority()`

返回所有设置中存在的最高优先级的数值 `default` ， `SETTINGS_PRIORITIES` 如果没有存储设置，则返回from 的数值。

set(*名称*, *值*, *优先级='项目'*)

存储具有给定优先级的键/值属性。

*在*配置Crawler对象（通过 `configure()` 方法）*之前*应该填充设置，否则它们将没有任何效果。

参数：
- name（*字符串*）- 设置名称
- value（*any*）- 与设置关联的值
- priority（*字符串或整数*）- 设置的优先级。应该是一个键 `SETTINGS_PRIORITIES` 或整数

setmodule(*模块*, *优先级='项目'*)

存储具有给定优先级的模块的设置。

这是一个辅助函数，它使用提供的函数调用 `set()` 每个全局声明的大写变量。 `module` `priority`

参数：
- module（*模块对象或字符串*）- 模块或模块的路径
- priority（*字符串或整数*）- 设置的优先级。应该是一个键 `SETTINGS_PRIORITIES` 或整数

update(*values*, *priority ='project'*)

存储具有给定优先级的键/值对。

这是一个辅助函数调用 `set()` 为每一个项目 `values` 与所提供的 `priority` 。

如果 `values` 是字符串，则假定它是JSON编码的并且首先解析为dict `json.loads()` 。如果是 `BaseSettings` 实例，将使用每个键的优先级并 `priority` 忽略该参数。这允许使用单个命令插入/更新具有不同优先级的设置。

参数：
- values（dict或string或 `BaseSettings` ）- 设置名称和值
- priority（*字符串或整数*）- 设置的优先级。应该是一个键 `SETTINGS_PRIORITIES` 或整数

SpiderLoader API

类 scrapy.loader.SpiderLoader

该类负责检索和处理项目中定义的spider类。

可以通过在 `SPIDER_LOADER_CLASS` 项目设置中指定其路径来使用自定义蜘蛛装载程序 。它们必须完全实现 `scrapy.interfaces.ISpiderLoader` 接口以保证无错执行。

from_settings(*设定*)

Scrapy使用此类方法来创建类的实例。它是使用当前项目设置调用的，它会加载在`SPIDER_MODULES`设置模块中递归发现的蜘蛛。

参数： **设置**（`Settings`实例）- 项目设置

load(*spider_name*)

获取具有给定名称的Spider类。它将查看以前加载的名为*spider_name*的蜘蛛类的蜘蛛，如果找不到则会引发*KeyError*。

参数： spider_name（*str*）- 蜘蛛类名

list()

获取项目中可用蜘蛛的名称。

find_by_request(***要求***)

列出可以处理给定请求的蜘蛛名称。将尝试将请求的URL与蜘蛛的域匹配。

参数： request（`Request`instance）- 查询请求

信号

class **scrapy.signalmanager.SignalManager**(*sender = _Anonymous*)

connect(***接收器***, ***信号***, *** kwargs*)

将接收器功能连接到信号。

信号可以是任何对象，尽管Scrapy附带了一些预定信号，这些信号记录在信号 部分中。

参数：
- **receiver**（*callable*）- 要连接的函数
- **signal**（*object*）- 要连接的信号

disconnect(***接收器***, ***信号***, *** kwargs*)

断开接收器功能与信号的连接。这与`connect()`方法具有相反的效果，并且参数是相同的。

disconnect_all(***信号***, *** kwargs*)

断开所有接收器与给定信号的连接。

参数： signal（*object*）- 要断开的信号

send_catch_log(*信号*, *** kwargs*)

发送信号，捕获异常并记录它们。

关键字参数传递给信号处理程序（通过该 `connect()` 方法连接）。

send_catch_log_deferred(*信号*, *** kwargs*)

喜欢 `send_catch_log()` 但支持从信号处理程序返回延迟。

返回在触发所有信号处理程序延迟后触发的Deferred。发送信号，捕获异常并记录它们。

关键字参数传递给信号处理程序（通过该 `connect()` 方法连接）。

统计收集器

`scrapy.statscollectors` 模块下有几个Stats收集器 ，它们都实现了 `StatsCollector` 类定义的Stats Collector API（它们都是从它们继承的）。

类 **scrapy.statscollectors.StatsCollector**

get_value(*键*, *默认=无*)

返回给定统计信息键的值，如果不存在则返回默认值。

get_stats( )

从当前正在运行的蜘蛛中获取所有统计数据作为词典。

set_value(*键*, *值*)

设置给定统计数据键的给定值。

set_stats(*统计*)

使用 `stats` 参数中传递的dict覆盖当前统计信息。

inc_value(*key*, *count = 1*, *start = 0*)

假定给定的起始值（当它未设置时），按给定的计数递增给定统计量键的值。

max_value(*键*, *值*)

仅当同一键的当前值低于值时，才设置给定键的给定值。如果给定键没有当前值，则始终设置该值。

min_value(*键*, *值*)

仅当同一键的当前值大于值时，才设置给定键的给定值。如果给定键没有当前值，则始终设置该值。

clear_stats( )

清除所有统计数据。

以下方法不是stats集合api的一部分，而是在实现自定义统计信息收集器时使用：

open_spider(*蜘蛛*)

打开给定的蜘蛛进行统计数据收集。

close_spider(*蜘蛛*)

关闭给定的蜘蛛。调用此方法后，无法访问或收集更多特定统计信息。