扩展

扩展框架提供了一种将自己的自定义功能插入Scrapy的机制。

扩展只是在Scrapy启动时实例化的常规类，当初始化扩展时。

扩展设置

扩展使用Scrapy设置来管理其设置，就像任何其他Scrapy代码一样。

扩展程序通常使用自己的名称为其设置添加前缀，以避免与现有（和将来）扩展冲突。例如，处理Google Sitemaps的假设扩展将使用诸如*GOOGLESITEMAP_ENABLED*，*GOOGLESITEMAP_DEPTH*等设置 。

加载和激活扩展

通过实例化扩展类的单个实例，在启动时加载和激活扩展。因此，所有扩展初始化代码必须在类构造函数（`__init__`方法）中执行。

要使扩展可用，请将其添加到`EXTENSIONS`Scrapy设置中的设置。在`EXTENSIONS`，每个扩展名由一个字符串表示：扩展名类名称的完整Python路径。例如：

```
EXTENSIONS = {
    'scrapy.extensions.corestats.CoreStats': 500,
    'scrapy.extensions.telnet.TelnetConsole': 500,
}
```

如您所见，该`EXTENSIONS`设置是一个dict，其中键是扩展路径，它们的值是订单，它们定义了扩展*加载*顺序。该`EXTENSIONS`设置与`EXTENSIONS_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扩展的最终排序列表。

由于扩展通常不相互依赖，因此在大多数情况下，它们的加载顺序无关紧要。这就是`EXTENSIONS_BASE`设置定义具有相同order（`0`）的所有扩展的原因。但是，如果您需要添加依赖于已加载的其他扩展的扩展，则可以利用此功能。

可用，启用和禁用扩展

并非所有可用的扩展程序都会启用。其中一些通常取决于特定的环境。例如，HTTP缓存扩展在默认情况下可用，但禁用，除非`HTTPCACHE_ENABLED`设置了该设置。

禁用扩展名

要禁用默认启用的扩展名（即 `EXTENSIONS_BASE` 设置中包含的扩展名），您必须将其顺序设置为 `None`。例如：

```
EXTENSIONS = {
    'scrapy.extensions.corestats.CoreStats': None,
}
```

编写自己的扩展

每个扩展都是一个Python类。Scrapy扩展的主要入口点（也包括中间件和管道）是 `from_crawler` 接收 `Crawler` 实例的类方法。通过Crawler对象，您可以访问设置，信号，统计信息，还可以控制爬网行为。

通常，扩展连接到信号并执行由它们触发的任务。

最后，如果 `from_crawler` 方法引发 `NotConfigured` 异常，则将禁用扩展。否则，将启用扩展程序。

样本扩展

在这里，我们将实现一个简单的扩展来说明上一节中描述的概念。此扩展程序每次都会记录一条消息：

- 一只蜘蛛被打开了
- 一只蜘蛛被关闭了
- 刮取特定数量的项目

将通过 `MYEXT_ENABLED` 设置启用扩展，并通过设置指定项目数 `MYEXT_ITEMCOUNT`。

这是这种扩展的代码：

```python
import logging
from scrapy import signals
from scrapy.exceptions import NotConfigured

logger = logging.getLogger(__name__)

class SpiderOpenCloseLogging(object):

    def __init__(self, item_count):
        self.item_count = item_count
        self.items_scraped = 0

    @classmethod
    def from_crawler(cls, crawler):
        # first check if the extension should be enabled and raise
        # NotConfigured otherwise
        if not crawler.settings.getbool('MYEXT_ENABLED'):
            raise NotConfigured

        # get the number of items from settings
        item_count = crawler.settings.getint('MYEXT_ITEMCOUNT', 1000)

        # instantiate the extension object
        ext = cls(item_count)

        # connect the extension object to signals
        crawler.signals.connect(ext.spider_opened, signal=signals.spider_opened)
        crawler.signals.connect(ext.spider_closed, signal=signals.spider_closed)
        crawler.signals.connect(ext.item_scraped, signal=signals.item_scraped)

        # return the extension object
        return ext

    def spider_opened(self, spider):
        logger.info("opened spider %s", spider.name)

    def spider_closed(self, spider):
        logger.info("closed spider %s", spider.name)

    def item_scraped(self, item, spider):
        self.items_scraped += 1
        if self.items_scraped % self.item_count == 0:
            logger.info("scraped %d items", self.items_scraped)
```

内置扩展参考

通用扩展

日志统计扩展

类 scrapy.extensions.logstats.LogStats

记录已爬网页面和已删除项目等基本统计信息。

核心统计扩展

类 scrapy.extensions.corestats.CoreStats

如果启用了统计信息收集，则启用核心统计信息的收集（请参阅统计信息收集）。

Telnet控制台扩展

类 scrapy.extensions.telnet.TelnetConsole

提供一个telnet控制台，用于在当前运行的Scrapy进程中进入Python解释器，这对于调试非常有用。

必须通过该 `TELNETCONSOLE_ENABLED` 设置启用telnet控制台，服务器将侦听指定的端口 `TELNETCONSOLE_PORT` 。

内存使用扩展

类 scrapy.extensions.memusage.MemoryUsage

> **注意**
>
> 此扩展在Windows中不起作用。

监视运行蜘蛛的Scrapy进程使用的内存：

1. 超过特定值时发送通知电子邮件
2. 当蜘蛛超过某个值时关闭蜘蛛

当达到某个警告值（ `MEMUSAGE_WARNING_MB` ）和达到最大值（ ）时，可以触发通知电子邮件 `MEMUSAGE_LIMIT_MB` ，这也将导致蜘蛛关闭并终止Scrapy进程。

此扩展由 `MEMUSAGE_ENABLED` 设置启用，可以使用以下设置进行配置：

- `MEMUSAGE_LIMIT_MB`
- `MEMUSAGE_WARNING_MB`
- `MEMUSAGE_NOTIFY_MAIL`
- `MEMUSAGE_CHECK_INTERVAL_SECONDS`

内存调试器扩展

类 scrapy.extensions.memdebug.MemoryDebugger

用于调试内存使用情况的扩展。它收集有关的信息：

- Python垃圾收集器未收集的对象
- 那些不应该活着的物品。有关更多信息，请参阅使用trackref调试内存泄漏

要启用此扩展程序，请启用该 `MEMDEBUG_ENABLED` 设置。信息将存储在统计数据中。

关闭蜘蛛扩展

类 scrapy.extensions.closespider.CloseSpider

在满足某些条件时，使用每种条件的特定结算原因自动关闭蜘蛛。

关闭蜘蛛的条件可以通过以下设置进行配置：

- `CLOSESPIDER_TIMEOUT`
- `CLOSESPIDER_ITEMCOUNT`
- `CLOSESPIDER_PAGECOUNT`
- `CLOSESPIDER_ERRORCOUNT`

CLOSESPIDER_TIMEOUT

默认： `0`

一个整数，指定秒数。如果蜘蛛保持打开的时间超过该秒数，它将自动关闭并显示原因 `closespider_timeout` 。如果为零（或未设置），则超时不会关闭蜘蛛。

CLOSESPIDER_ITEMCOUNT

默认： `0`

一个整数，指定多个项目。如果蜘蛛刮擦的数量超过了那个数量并且这些物品被物品管道传递，那么蜘蛛将被关闭 `closespider_itemcount` 。 `CONCURRENT_REQUESTS` 仍处理当前位于下载队列（直到 请求）的请求。如果为零（或未设置），则蜘蛛将不会被传递的项目数量关闭。

CLOSESPIDER_PAGECOUNT

版本0.11中的新功能。

默认： `0`

一个整数，指定要爬网的最大响应数。如果蜘蛛爬行的次数超过了，那么蜘蛛就会被关闭 `closespider_pagecount` 。如果为零（或未设置），蜘蛛将不会被已爬网响应的数量关闭。

CLOSESPIDER_ERRORCOUNT

版本0.11中的新功能。

默认： `0`

一个整数，指定关闭spider之前要接收的最大错误数。如果蜘蛛产生的错误数量超过这个数量，那么它将被关闭 `closespider_errorcount` 。如果为零（或未设置），蜘蛛将不会被错误数量关闭。

StatsMailer扩展

类 scrapy.extensions.statsmailer.StatsMailer

这个简单的扩展可用于每次域名完成抓取时发送通知电子邮件，包括收集的Scrapy统计数据。电子邮件将发送给 `STATSMAILER_RCPTS` 设置中指定的所有收件人。

调试扩展

堆栈跟踪转储扩展

类 scrapy.extensions.debug.StackTraceDump

收到SIGQUIT或SIGUSR2信号时，转储有关正在运行的进程的信息。转储的信息如下：

1. 发动机状态（使用 `scrapy.utils.engine.get_engine_status()` ）
2. 实时引用（请参阅使用trackref调试内存泄漏）
3. 堆栈所有线程的跟踪

在转储堆栈跟踪和引擎状态后，Scrapy进程继续正常运行。

此扩展仅适用于POSIX兼容平台（即非Windows），因为Windows上没有SIGQUIT和SIGUSR2信号。

至少有两种方法可以发送Scrapy SIGQUIT信号：

1. 在Scrapy进程运行时按Ctrl键（仅限Linux）？
2. 通过运行此命令（假设 `<pid>` 是Scrapy进程的进程ID）：

```
kill -QUIT <pid>
```

调试器扩展

类 scrapy.extensions.debug.Debugger

收到SIGUSR2信号时，在正在运行的Scrapy进程中调用Python调试器。退出调试器后，Scrapy进程继续正常运行。

有关更多信息，请参阅*Python中的调试*。

此扩展仅适用于POSIX兼容平台（即不是Windows）。