下载和处理文件和图像

Scrapy提供可重复使用的项目管道，用于下载附加到特定项目的文件（例如，当您刮取产品并且还想在本地下载其图像时）。这些管道共享一些功能和结构（我们将它们称为媒体管道），但通常您将使用文件管道或图像管道。

两个管道都实现了这些功能：

- 避免重新下载最近下载的媒体
- 指定存储介质的位置（文件系统目录，Amazon S3存储桶，Google云端存储存储桶）

图像管道具有一些用于处理图像的额外功能：

- 将所有下载的图像转换为通用格式（JPG）和模式（RGB）
- 缩略图生成
- 检查图像宽度/高度以确保它们符合最小约束

管道还保留当前正在计划下载的那些媒体URL的内部队列，并将那些到达包含相同媒体的响应连接到该队列。这样可以避免在多个项目共享时多次下载同一媒体。

使用文件管道

使用时的典型工作流程`FilesPipeline`如下：

1. 在Spider中，您抓取一个项目并将所需的URL放入一个`file_urls`字段中。
2. 该项目从蜘蛛返回并转到项目管道。
3. 当项目到达时`FilesPipeline`，`file_urls`使用标准Scrapy调度程序和下载程序（这意味着重新使用调度程序和下载程序中间件）计划下载字段中的URL，但具有更高的优先级，在其他页面被删除之前处理它们。该项目在该特定管道阶段保持“锁定”，直到文件完成下载（或由于某种原因失败）。
4. 下载文件后，`files`将使用结果填充另一个字段（）。该字段将包含一个dicts列表，其中包含有关下载文件的信息，例如下载的路径，原始的已删除URL（从`file_urls`字段中获取）和文件校验和。`files`字段列表中的文件将保留原始`file_urls`字段的相同顺序。如果某些文件下载失败，将记录错误，该文件将不会出现在该`files`字段中。

使用图像流水线

使用它`ImagesPipeline`很像使用`FilesPipeline`，除了使用的默认字段名称不同：您使用`image_urls`项目的图像URL，它将填充一个`images`字段以获取有关下载图像的信息。

使用 `ImagesPipeline` for images文件的优点是，您可以配置一些额外的功能，如生成缩略图和根据图像大小过滤图像。

图像管道使用Pillow进行缩略图并将图像标准化为JPEG / RGB格式，因此您需要安装此库才能使用它。 Python映像库（PIL）在大多数情况下也应该可以工作，但是已知它会在某些设置中引起麻烦，因此我们建议使用Pillow而不是PIL。

启用媒体管道

要启用媒体管道，必须先将其添加到项目 `ITEM_PIPELINES` 设置中。

对于图像管道，请使用：

```
ITEM_PIPELINES = {'scrapy.pipelines.images.ImagesPipeline': 1}
```

对于文件管道，使用：

```
ITEM_PIPELINES = {'scrapy.pipelines.files.FilesPipeline': 1}
```

注意

您也可以同时使用“文件”和“图像管道”。

然后，将目标存储设置配置为将用于存储下载的图像的有效值。否则，即使您将管道包含在 `ITEM_PIPELINES` 设置中，管道也将保持禁用状态。

对于“文件管道”，请设置以下 `FILES_STORE` 设置：

```
FILES_STORE = '/path/to/valid/dir'
```

对于图像管道，请设置以下 `IMAGES_STORE` 设置：

```
IMAGES_STORE = '/path/to/valid/dir'
```

支持的存储

文件系统是目前唯一官方支持的存储，但也支持在Amazon S3和Google Cloud Storage中存储文件。

文件系统存储

使用文件名的URL 的SHA1哈希存储文件。

例如，以下图片网址：

```
http://www.example.com/image.jpg
```

谁的*SHA1哈希*是：

```
3afec3b4765f8f0a07b78f98c07b83f013567a0a
```

将下载并存储在以下文件中：

```
<IMAGES_STORE>/full/3afec3b4765f8f0a07b78f98c07b83f013567a0a.jpg
```

哪里：

- `<IMAGES_STORE>` 是 `IMAGES_STORE` 图像管道设置中定义的目录。
- `full` 是一个子目录，用于将完整图像与缩略图分开（如果使用）。有关详细信息，请参阅缩略图生成图像。

亚马逊S3存储

`FILES_STORE` 并且 `IMAGES_STORE` 可以代表Amazon S3存储桶。Scrapy会自动将文件上传到存储桶。

例如，这是一个有效值 `IMAGES_STORE` ：

```
IMAGES_STORE = 's3://bucket/images'
```

您可以修改用于存储文件的访问控制列表（ACL）策略，该策略由 `FILES_STORE_S3_ACL` 和 `IMAGES_STORE_S3_ACL` 设置定义。默认情况下，ACL设置为 `private` 。要使文件公开可用，请使用以下 `public-read` 策略：

```
IMAGES_STORE_S3_ACL = 'public-read'
```

有关更多信息，请参阅Amazon S3开发人员指南中的预装ACL。

谷歌云存储

`FILES_STORE` 并且 `IMAGES_STORE` 可以代表Google云端存储分区。Scrapy会自动将文件上传到存储桶。（需要谷歌云存储）

例如，这些是有效的 `IMAGES_STORE` 和 `GCS_PROJECT_ID` 设置：

```
IMAGES_STORE = 'gs://bucket/images/'
GCS_PROJECT_ID = 'project_id'
```

有关身份验证的信息，请参阅此文档。

用法示例

要首先使用媒体管道，请启用它。

然后，如果一个蜘蛛返回带有URL键的dict（ `file_urls` 或者 `image_urls` ，分别对于Files或Images Pipeline），则管道将把结果放在相应的键（ `files` 或 `images` ）下。

如果您更喜欢使用 `Item` ，则使用必要的字段定义自定义项，例如图像管道的示例：

```
import scrapy

class MyItem(scrapy.Item):

    # ... other item fields ...
    image_urls = scrapy.Field()
    images = scrapy.Field()
```

如果要为URL键或结果键使用其他字段名称，也可以覆盖它。

对于文件管道，设置 `FILES_URLS_FIELD` 和/或 `FILES_RESULT_FIELD` 设置：

```
FILES_URLS_FIELD = 'field_name_for_your_files_urls'
FILES_RESULT_FIELD = 'field_name_for_your_processed_files'
```

对于图像管道，设置 `IMAGES_URLS_FIELD` 和/或 `IMAGES_RESULT_FIELD` 设置：

```
IMAGES_URLS_FIELD = 'field_name_for_your_images_urls'
IMAGES_RESULT_FIELD = 'field_name_for_your_processed_images'
```

如果您需要更复杂的内容并希望覆盖自定义管道行为，请参阅扩展媒体管道。

如果您有多个从ImagePipeline继承的图像管道，并且您希望在不同的管道中具有不同的设置，则可以设置前面带有管道类的大写名称的设置键。例如，如果您的管道名为MyPipeline，并且您想要自定义IMAGES_URLS_FIELD，则定义设置MYPIPELINE_IMAGES_URLS_FIELD并使用您的自定义设置。

附加功能

文件到期

Image Pipeline避免下载最近下载的文件。要调整此保留延迟，请使用`FILES_EXPIRES`设置（或`IMAGES_EXPIRES`在图像管道的情况下），该设置指定天数的延迟：

```
# 120 days of delay for files expiration
FILES_EXPIRES = 120

# 30 days of delay for images expiration
IMAGES_EXPIRES = 30
```

两个设置的默认值均为90天。

如果您具有子类FilesPipeline的管道，并且您希望为其设置不同的设置，则可以设置以大写类名称开头的设置键。例如，给定名为MyPipeline的管道类，您可以设置设置键：

MYPIPELINE_FILES_EXPIRES = 180

和管道类MyPipeline将到期时间设置为180。

图像的缩略图生成

图像管道可以自动创建下载图像的缩略图。

要使用此功能，您必须设置`IMAGES_THUMBS`为字典，其中键是缩略图名称，值是其尺寸。

例如：

```
IMAGES_THUMBS = {
    'small': (50, 50),
    'big': (270, 270),
}
```

使用此功能时，图像管道将使用以下格式创建每个指定大小的缩略图：

```
<IMAGES_STORE>/thumbs/<size_name>/<image_id>.jpg
```

哪里：

- `<size_name>` 是在指定的 `IMAGES_THUMBS` 字典键（ `small` ， `big` 等）
- `<image_id>` 是图像网址的SHA1哈希值

使用 `small` 和 `big` 缩略图名称存储的图像文件示例：

```
<IMAGES_STORE>/full/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/small/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
<IMAGES_STORE>/thumbs/big/63bbfea82b8880ed33cdb762aa11fab722a90a24.jpg
```

第一个是从网站下载的完整图像。

过滤小图像

使用图像管道时，您可以通过在 `IMAGES_MIN_HEIGHT` 和 `IMAGES_MIN_WIDTH` 设置中指定允许的最小尺寸来删除太小的图像。

例如：

```
IMAGES_MIN_HEIGHT = 110
IMAGES_MIN_WIDTH = 110
```

注意

大小限制根本不会影响缩略图生成。

可以只设置一个大小约束或两者。设置它们时，仅保存满足两个最小尺寸的图像。对于上面的例子，尺寸（105×105）或（105×200）或（200×105）的图像都将被丢弃，因为至少一个尺寸比约束短。

默认情况下，没有大小限制，因此处理所有图像。

允许重定向

默认情况下，媒体管道会忽略重定向，即HTTP重定向到媒体文件URL请求将意味着媒体下载被视为失败。

要处理媒体重定向，请将此设置设置为 `True` ：

```
MEDIA_ALLOW_REDIRECTS = True
```

扩展媒体管道

请在此处查看可在自定义文件管道中覆盖的方法：

类 scrapy.pipelines.files.FilesPipeline

get_media_requests(*项目*, *信息*)

如工作流程所示，管道将获取要从项目下载的图像的URL。为此，您可以覆盖该 `get_media_requests()` 方法并为每个文件URL返回一个Request：

```
def get_media_requests(self, item, info):
    for file_url in item['file_urls']:
        yield scrapy.Request(file_url)
```

这些请求将由管道处理，并且当它们完成下载后，结果将 `item_completed()` 作为2元素元组的列表发送到 方法。每个元组将包含以下内容： `(success, file_info_or_error)`

- `success` 是一个布尔值， `True` 如果图像已成功下载或 `False` 由于某种原因失败
- `file_info_or_error` 是一个包含以下键的dict（如果成功 `True` ）或者如果出现问题则为Twisted Failure。
  - `url` - 从中下载文件的URL。这是从 `get_media_requests()` 方法返回的请求的url。
  - `path` - `FILES_STORE` 存储文件的路径（相对于）
  - `checksum` - 图像内容的MD5哈希

收到的元组列表 `item_completed()` 保证保留与 `get_media_requests()` 方法返回的请求相同的顺序。

这是 `results` 参数的典型值：

```
[(True,
  {'checksum': '2b00042f7481c7b056c4b410d28f33cf',
   'path': 'full/0a79c461a4062ac383dc4fade7bc09f1384a3910.jpg',
   'url': 'http://www.example.com/files/product1.pdf'}),
 (False,
  Failure(...))]
```

默认情况下，该 `get_media_requests()` 方法返回 `None` ，这意味着没有要为该项目下载的文件。

item_completed(*结果*, *项目*, *信息*)

`FilesPipeline.item_completed()` 当单个项目的所有文件请求都已完成（完成下载或由于某种原因失败）时调用的方法。

该 `item_completed()` 方法必须返回将发送到后续项目管道阶段的输出，因此您必须返回（或删除）该项目，就像在任何管道中一样。

下面是 `item_completed()` 我们在 `file_paths` 项目字段中存储下载的文件路径（在结果中传递）的方法示例，如果项目不包含任何文件，则删除该项目：

```
from scrapy.exceptions import DropItem

def item_completed(self, results, item, info):
    file_paths = [x['path'] for ok, x in results if ok]
    if not file_paths:
        raise DropItem("Item contains no files")
    item['file_paths'] = file_paths
    return item
```

默认情况下，该 `item_completed()` 方法返回该项。

在此处查看您可以在自定义图像管道中覆盖的方法：

类 scrapy.pipelines.images.ImagesPipeline

它 `ImagesPipeline` 是 `FilesPipeline` 自定义字段名称和添加图像自定义行为的扩展。

get_media_requests（*项目*，*信息*）

与方法的工作方式相同 `FilesPipeline.get_media_requests()` ，但为图像URL使用不同的字段名称。

必须为每个图像URL返回一个请求。

item_completed（*结果*，*项目*，*信息*）

`ImagesPipeline.item_completed()` 当单个项目的所有图像请求都已完成（完成下载或由于某种原因失败）时，将调用该方法。

与方法的工作方式相同 `FilesPipeline.item_completed()` ，但使用不同的字段名称来存储图像下载结果。

默认情况下，该 `item_completed()` 方法返回该项。

自定义图像管道示例

以下是图像管道的完整示例，其方法在上面举例说明：

```python
import scrapy
from scrapy.pipelines.images import ImagesPipeline
from scrapy.exceptions import DropItem

class MyImagesPipeline(ImagesPipeline):

    def get_media_requests(self, item, info):
        for image_url in item['image_urls']:
            yield scrapy.Request(image_url)

    def item_completed(self, results, item, info):
        image_paths = [x['path'] for ok, x in results if ok]
        if not image_paths:
            raise DropItem("Item contains no images")
        item['image_paths'] = image_paths
        return item
```