物品出口商

删除项目后，通常需要保留或导出这些项目，以便在其他应用程序中使用这些项目。毕竟，这是刮擦过程的全部目的。

为此，Scrapy为不同的输出格式提供了一组项目导出器，例如XML，CSV或JSON。

使用项目导出器

如果您赶时间，并且只想使用项目导出器输出已删除的数据，请参阅Feed导出。否则，如果您想知道项目导出器如何工作或需要更多自定义功能（默认导出未涵盖），请继续阅读下面的内容。

要使用项目导出器，必须使用其必需的args对其进行实例化。每个项目导出器都需要不同的参数，因此请在内置项目导出器参考中检查每个导出器文档以确定。在实例化导出器之后，您必须：

1.调用方法`start_exporting()`以指示导出过程的开始

2. `export_item()`为要导出的每个项目调用方法

3.最后调用`finish_exporting()`to信号表示导出过程结束

在这里，您可以看到一个项目管道，它使用多个项目导出器根据其中一个字段的值将已删除的项目分组到不同的文件：

```
from scrapy.exporters import XmlItemExporter

class PerYearXmlExportPipeline(object):
    """Distribute items across multiple XML files according to their 'year' field"""

    def open_spider(self, spider):
        self.year_to_exporter = {}

    def close_spider(self, spider):
        for exporter in self.year_to_exporter.values():
            exporter.finish_exporting()
            exporter.file.close()

    def _exporter_for_item(self, item):
        year = item['year']
        if year not in self.year_to_exporter:
            f = open('{}.xml'.format(year), 'wb')
            exporter = XmlItemExporter(f)
            exporter.start_exporting()
            self.year_to_exporter[year] = exporter
        return self.year_to_exporter[year]

    def process_item(self, item, spider):
        exporter = self._exporter_for_item(item)
        exporter.export_item(item)
        return item
```

项目字段的序列化

默认情况下，字段值未经修改地传递给基础序列化库，并且如何序列化它们的决定被委托给每个特定的序列化库。

但是，您可以自定义每个字段值*在传递到序列化库之前的序列化方式*。

有两种方法可以自定义字段的序列化方式，下面将对此进行介绍。

1.在字段中声明序列化程序

如果使用`Item`，可以在字段元数据中声明序列化程序。序列化程序必须是可调用的，它接收一个值并返回其序列化形式。

例：

```
import scrapy

def serialize_price(value):
    return '$ %s' % str(value)

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field(serializer=serialize_price)
```

2.重写serialize_field（）方法

您还可以覆盖该`serialize_field()`方法以自定义字段值的导出方式。

确保 `serialize_field()` 在自定义代码后调用基类方法。

例：

```
from scrapy.exporter import XmlItemExporter

class ProductXmlExporter(XmlItemExporter):

    def serialize_field(self, field, name, value):
        if field == 'price':
            return '$ %s' % str(value)
        return super(Product, self).serialize_field(field, name, value)
```

内置项目导出器参考

以下是与Scrapy捆绑在一起的物品出口商列表。其中一些包含输出示例，假设您要导出这两个项：

```
Item(name='Color TV', price='1200')
Item(name='DVD player', price='200')
```

BaseItemExporter

class **scrapy.exporters.BaseItemExporter** (*fields_to_export = None , export_empty_fields = False , encoding ='utf-8' , indent = 0*)

这是所有项目导出器的（抽象）基类。它为所有（具体）项目导出程序使用的常用功能提供支持，例如定义要导出的字段，是否导出空字段或使用哪种编码。

这些功能可以通过该填充它们各自的实例属性的构造器参数进行配置： `fields_to_export` , `export_empty_fields` , `encoding` , `indent` 。

export_item (*项目*)

导出给定项目。此方法必须在子类中实现。

serialize_field (*字段* , *名称* , *值*)

返回给定字段的序列化值。如果要控制特定字段或值的序列化/导出方式，可以覆盖此方法（在自定义项目导出器中）。

默认情况下，此方法查找在item字段中声明的序列化程序，并返回将该序列化程序应用于该值的结果。如果未找到序列化程序，则除了使用属性中声明的编码 `unicode` 进行编码的值外，它将返回未更改的值 。 `str` `encoding`

参数：

- **字段**（`Field`对象或空字典）- 被序列化的字段。如果正在导出原始字典（不是`Item`）*字段*值是空字典。
- **name**（*str*）- 要序列化的字段的名称
- **value** - 要序列化的值

`start_exporting()`

发出导出过程的开始信号。一些出口商可能会使用它来生成一些必需的标题（例如，`XmlItemExporter`）。您必须在导出任何项目之前调用此方法。

`finish_exporting()`

发出导出过程结束的信号。一些出口商可能会使用它来生成一些必需的页脚（例如，`XmlItemExporter`）。在没有其他要导出的项目之后，必须始终调用此方法。

`fields_to_export`

包含要导出的字段名称的列表，如果要导出所有字段，则为None。默认为无。

某些导出器（例如`CsvItemExporter`）尊重此属性中定义的字段的顺序。

一些导出器可能需要fields_to_export list才能在蜘蛛返回dicts（非`Item`实例）时正确导出数据。

`export_empty_fields`

是否在导出的数据中包含空/未填充的项目字段。默认为`False`。某些导出器（例如`CsvItemExporter`）忽略此属性并始终导出所有空字段。

dict项目将忽略此选项。

`encoding`

将用于编码unicode值的编码。这仅影响unicode值（使用此编码始终将其序列化为str）。其他值类型将不变地传递给特定的序列化库。

`indent`

用于在每个级别上缩进输出的空间量。默认为`0`。

- `indent=None` 选择最紧凑的表示，同一行中的所有项目没有缩进
- `indent<=0` 每个项目都在自己的行上，没有缩进
- `indent>0` 每个项目在其自己的行上，使用提供的数值缩进

XmlItemExporter

class **scrapy.exporters.XmlItemExporter**(*file*, *item_element='item'*, *root_element='items'*, ***kwargs*)

参数：

- **file** - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）
- **root_element** （*str*） - 导出的XML中的根元素的名称。
- **item_element** （*str*） - 导出的XML中每个item元素的名称。

此构造函数的其他关键字参数将传递给 `BaseItemExporter` 构造函数。

此出口商的典型输出是：

```
<?xml version="1.0" encoding="utf-8"?>
<items>
  <item>
    <name>Color TV</name>
    <price>1200</price>
 </item>
  <item>
    <name>DVD player</name>
    <price>200</price>
 </item>
</items>
```

除非在 `serialize_field()` 方法中重写，否则通过序列化 `<value>` 元素内的每个值来导出多值字段。这是为了方便，因为多值字段非常常见。

例如，项目：

```
Item(name=['John', 'Doe'], age='23')
```

将序列化为：

```
<?xml version="1.0" encoding="utf-8"?>
<items>
  <item>
    <name>
      <value>John</value>
      <value>Doe</value>
    </name>
    <age>23</age>
  </item>
</items>
```

CsvItemExporter

class **scrapy.exporters.CsvItemExporter** (*file* , *include_headers_line = True* , *join_multivalued =' , ' , ** kwargs*)

将CSV格式的项目导出到给定的类文件对象。如果 `fields_to_export` 设置了该 属性，它将用于定义CSV列及其顺序。该 `export_empty_fields` 属性对此导出器没有影响。

参数：

- **file** - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）
- **include_headers_line** （*str*） - 如果启用，则使导出器输出一个标题行，其中包含从中获取的字段名称 `BaseItemExporter.fields_to_export` 或第一个导出的项目字段。
- **join_multivalued** - 将用于连接多值字段的char（或字符）（如果找到）。

此构造函数的其他关键字参数将传递给 `BaseItemExporter` 构造函数，以及csv.writer构造函数的剩余参数 ，因此您可以使用任何*csv.writer*构造函数参数来自定义此导出器。

此出口商的典型输出是：

```
product,price
Color TV,1200
DVD player,200
```

PickleItemExporter

class **scrapy.exporters.PickleItemExporter** (*file* , *protocol = 0* , *** kwargs*)

将pickle格式的项目导出到给定的类文件对象。

参数：

- **file** - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）
- **protocol** （*int*） - 要使用的pickle协议。

有关更多信息，请参阅pickle模块文档。

此构造函数的其他关键字参数将传递给 `BaseItemExporter` 构造函数。

Pickle不是人类可读的格式，因此不提供输出示例。

PprintItemExporter

class **scrapy.exporters.PprintItemExporter** (***文件*** , *** kwargs*)

将漂亮打印格式的项目导出到指定的文件对象。

参数： **file** - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）

此构造函数的其他关键字参数将传递给 `BaseItemExporter` 构造函数。

此出口商的典型输出是：

```
{'name': 'Color TV', 'price': '1200'}
{'name': 'DVD player', 'price': '200'}
```

较长的线条（如果存在）是格式化的。

JsonItemExporter

class **scrapy.exporters.JsonItemExporter**（***文件***，*** kwargs*）

将JSON格式的项目导出到指定的类文件对象，将所有对象写为对象列表。其他构造函数参数传递给 `BaseItemExporter` 构造函数，以及JSONEncoder构造函数的剩余参数，因此您可以使用任何 JSONEncoder构造函数参数来自定义此导出器。

参数： file - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）

此出口商的典型输出是：

```
[{"name": "Color TV", "price": "1200"},
{"name": "DVD player", "price": "200"}]
```

警告

JSON是一种非常简单和灵活的序列化格式，但它不能很好地扩展大量数据，因为JSON解析器（在任何语言中）都没有很好地支持增量（也称为流模式）解析（如果有的话），以及他们中的大多数只是在内存中解析整个对象。如果您希望JSON的功能和简单性具有更加流友好的格式，请考虑使用 `JsonLinesItemExporter` ，或者将输出拆分为多个块。

JsonLinesItemExporter

class **scrapy.exporters.JsonLinesItemExporter**（***文件***，*** kwargs*）

将JSON格式的项目导出到指定的类文件对象，每行编写一个JSON编码的项目。其他构造函数参数传递给 `BaseItemExporter` 构造函数，以及JSONEncoder构造函数的剩余参数，因此您可以使用任何JSONEncoder 构造函数参数来自定义此导出器。

参数： file - 用于导出数据的类文件对象。它的 `write` 方法应该接受 `bytes` （以二进制模式打开的磁盘文件， `io.BytesIO` 对象等）

此出口商的典型输出是：

```
{"name": "Color TV", "price": "1200"}
{"name": "DVD player", "price": "200"}
```

与生成 `JsonItemExporter` 的格式不同，此导出器生成的格式非常适合序列化大量数据。