物品

抓取的主要目标是从非结构化源（通常是网页）中提取结构化数据。Scrapy蜘蛛可以像Python一样返回提取的数据。虽然方便和熟悉，但Python缺乏结构：很容易在字段名称中输入拼写错误或返回不一致的数据，尤其是在具有许多蜘蛛的较大项目中。

为了定义通用输出数据格式，Scrapy提供了`Item`类。 `Item`对象是用于收集抓取数据的简单容器。它们提供类似字典的 API，并具有用于声明其可用字段的方便语法。

各种Scrapy组件使用Items提供的额外信息：导出器查看声明的字段以确定要导出的列，可以使用Item字段元数据自定义序列化，`trackref` 跟踪Item实例以帮助查找内存泄漏（请参阅使用trackref调试内存泄漏）等。

声明项目

使用简单的类定义语法和`Field`对象声明项。这是一个例子：

```
import scrapy

class Product(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    stock = scrapy.Field()
    last_updated = scrapy.Field(serializer=str)
```

注意

那些熟悉Django的人会注意到Scrapy Items被宣告类似于Django Models，除了Scrapy Items更简单，因为没有不同字段类型的概念。

项目字段

`Field`对象用于指定每个字段的元数据。例如，`last_updated`上面示例中说明的字段的序列化函数。

您可以为每个字段指定任何类型的元数据。`Field`对象接受的值没有限制。出于同样的原因，没有所有可用元数据键的参考列表。`Field`对象中定义的每个键可以由不同的组件使用，只有那些组件知道它。您也可以根据`Field`自己的需要定义和使用项目中的任何其他 键。`Field`

对象的主要目标 是提供一种在一个地方定义所有字段元数据的方法。通常，行为取决于每个字段的那些组件使用某些字段键来配置该行为。您必须参考其文档以查看每个组件使用的元数据键。

请务必注意， `Field` 用于声明项目的对象不会保留为类属性。相反，可以通过 `Item.fields` 属性访问它们。

使用项目

以下是使用上面声明的 `Product` 项目对项目执行的常见任务的一些示例 。您会注意到API与 dict API非常相似。

创建项目

```
>>> product = Product(name='Desktop PC', price=1000)
>>> print product
Product(name='Desktop PC', price=1000)
```

获取字段值

```
>>> product['name']
Desktop PC
>>> product.get('name')
Desktop PC

>>> product['price']
1000

>>> product['last_updated']
Traceback (most recent call last):
    ...
KeyError: 'last_updated'

>>> product.get('last_updated', 'not set')
not set

>>> product['lala'] # getting unknown field
Traceback (most recent call last):
    ...
KeyError: 'lala'

>>> product.get('lala', 'unknown field')
'unknown field'

>>> 'name' in product  # is name field populated?
True

>>> 'last_updated' in product  # is last_updated populated?
False

>>> 'last_updated' in product.fields  # is last_updated a declared field?
True

>>> 'lala' in product.fields  # is lala a declared field?
False
```

设定字段值

```
>>> product['last_updated'] = 'today'
>>> product['last_updated']
today

>>> product['lala'] = 'test' # setting unknown field
Traceback (most recent call last):
    ...
KeyError: 'Product does not support field: lala'
```

访问所有填充值

要访问所有填充值，只需使用典型的dict API：

```
>>> product.keys()
['price', 'name']

>>> product.items()
[('price', 1000), ('name', 'Desktop PC')]
```

其他常见任务

复制项目：

```
>>> product2 = Product(product)
>>> print product2
Product(name='Desktop PC', price=1000)

>>> product3 = product2.copy()
>>> print product3
Product(name='Desktop PC', price=1000)
```

从项目创建dicts：

```
>>> dict(product) # create a dict from all populated values
{'price': 1000, 'name': 'Desktop PC'}
```

从dicts创建项目：

```
>>> Product({'name': 'Laptop PC', 'price': 1500})
Product(price=1500, name='Laptop PC')

>>> Product({'name': 'Laptop PC', 'lala': 1500}) # warning: unknown field in dict
Traceback (most recent call last):
    ...
KeyError: 'Product does not support field: lala'
```

扩展项目

您可以通过声明原始Item的子类来扩展Items（以添加更多字段或更改某些字段的某些元数据）。

例如：

```
class DiscountedProduct(Product):
    discount_percent = scrapy.Field(serializer=str)
    discount_expiration_date = scrapy.Field()
```

您还可以使用先前的字段元数据扩展字段元数据，并附加更多值或更改现有值，如下所示：

```
class SpecificProduct(Product):
    name = scrapy.Field(Product.fields['name'], serializer=my_serializer)
```

这会添加（或替换）字段的`serializer`元数据键`name`，保留所有先前存在的元数据值。

物品对象

class **scrapy.item.Item** **(** [*arg*] **)**

返回一个可选择从给定参数初始化的新Item。

Items复制标准dict API，包括其构造函数。Items提供的唯一附加属性是：

fields

包含此Item的*所有已声明字段的字*典，不仅包括已填充的*字段*。键是字段名称，值是Item声明中`Field`使用的 对象。

字段对象

class **scrapy.item.Field** **(** [*arg*] **)**

该`Field`班只是一个别名内置的字典类，并没有提供任何额外功能或属性。换句话说，`Field`对象是普通的Python dicts。一个单独的类用于支持 基于类属性的项声明语法。