Scrapy Tutorial #9: How To Use Scrapy Item

Last updated on Feb 25 2019 by Michael Yin

Introduction:

This is the #9 post of my Scrapy Tutorial Series, in this Scrapy tutorial, I will talk about how to define Scrapy item, how to use Scrapy item, and how to create a custom Item Pipeline to save the data of Item into DB. I hope you have a good understanding after reading this article if you have any question when reading, just leave me message here, I will respond ASAP.

You can get the source code of this project at the end of this tutorial.

Scrapy Item VS Python Dict

In most scraping jobs, our goal is to extract structured data from web pages, Python Dict can indeed be used to return scraped data, however, it is easy to make a typo in field name or set inconsistent data, which would make some disaster in large project with many spiders. If you want one field only accept string type, there is no good solution to do this.

To solve this problem, Scrapy has provided `Scrapy Item` class. `Item` instances are simple container used to collected data, you can see it as wrapper of Python dict, but it has more useful feature compared native Python Dict.

If you are new to Scrapy, it is ok to use Python Dict in your project, however, if your project is large or you have some experience in Scrapy, please use Scrapy Item to make your project consistent.

How To Define Scrapy Item & How To Use It

Every Scrapy project have items definition file called `items.py`, so in this project, you need edit `scrapy_spider/items.py`

```
import scrapy

class QuoteItem(scrapy.Item):
    auther = scrapy.Field()
    quote = scrapy.Field()
```

Now let's try to use this item in Scrapy Shell, type `scrapy shell` to enter.

```
>>> from scrapy_spider.items import QuoteItem
>>> item = QuoteItem()
>>> item["quote"] = "test"
>>> item["quote"]
'test'

>>> item["wrong_field"] = "test"
KeyError: 'QuoteItem does not support field: wrong_field'

>>> 'name' in item  # is name in the item?
False

>>> 'quote' in item  # is quote in the item?
True
```

From the code above, you can see there is no big difference between the usage of Scrapy item and Python dict, but Scrapy item can help you remove the typo and make your project consistent, this is the reason why I recommend to use Scrapy Item rather than Python dict to collect data.

Item Pipeline

After an item has been scraped by spider, it will be sent to Item Pipeline to be processed by several components. Each pipeline component is a Python class which implements a simple method. The component can perform an action over the item, modify it, drop it, or send it to next component.

Item Pipeline can be used to validate scraped data, check duplicate data, or insert the data into databases such as Mysql, PostgreSQL or MongoDB.

In this section, I would like to show you how to use item pipeline to save the scraped quote data into database by using SQLAlchemy-The Database Toolkit for Python.

Setting up SQLAlchemy

First, we need to make sure we have installed Mysql Server

1. If you are on Mac, you can use `homebrew` to help you quickly install mysql.
2. If you are on Ubuntu or other Linux distribution, you can use package manager such as `apt`.
3. If you are on Windows, just install by using binary installer.

```
$ mysql -h localhost -u root -p

mysql> CREATE DATABASE quote_toscrape CHARACTER SET utf8 COLLATE utf8_general_ci;
```

The `localhost` in the command is the IP address of the MySQL server, remember to change it if your Mysql Server is on another machine. `quote_toscrape` is the database name.

After installing Mysql, now you need to install SQLAlchemy by using command `pip install SQLAlchemy`.

Settings.py

Now we add Mysql Server connection string in our project settings.py, add code below into `scrapy_spider/settings.py`

```
CONNECTION_STRING = "{drivername}://{user}:{passwd}@{host}:{port}/{db_name}?charset=utf8
    drivername="mysql",
    user="YOUR_USERNAME",
    passwd="YOUR_PASSWORD",
    host="localhost",
    port="3306",
    db_name="quote_toscrape",
)
```

Since we are using SQLAlchemy to connect Mysql, so the drivername here is `mysql`, if you are connecting Postgres, then the drivername would be `postgres`. You should change the `user`, `passwd` value here.

Models.py

Now create `scrapy_spider/models.py`, this file would contain database models created by `SQLAlchemy`.

First, we define a function which helps us connect to the database. For this, we'll need to import SQLAlchemy as well as our scrapy settings instance:

```
from sqlalchemy import create_engine, Column, Table, ForeignKey
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import (
    Integer, SmallInteger, String, Date, DateTime, Float, Boolean, Text, LargeBinary)

from scrapy.utils.project import get_project_settings

DeclarativeBase = declarative_base()

def db_connect():
    """
    Performs database connection using database settings from settings.py.
    Returns sqlalchemy engine instance
    """
    return create_engine(get_project_settings().get("CONNECTION_STRING"))

def create_table(engine):
    DeclarativeBase.metadata.create_all(engine)

class QuoteDB(DeclarativeBase):
    __tablename__ = "quote_table"

    id = Column(Integer, primary_key=True)
    quote = Column('quote', Text())
    author = Column('author', String(100))
```

There are some points I want to poin out with this example.

1. First, many online resources like to use `from sqlalchemy import *` to imports everything into our models.py, however, this is not a good practice because it can make people who read your code confused and have some bad impact on the performance. So to be better developers we need to be more explicit with what we are importing.

2. We use `get_project_settings` shipped with Scrapy to get the settings object, which can give us access to the `CONNECTION_STRING`.

3. To create table using Sqlalchemy, we have to import `declarative_base` from SQLAlchemy to map a class which defines the table structure to Mysql. We make `QuoteDB` inherit from `DeclarativeBase` and set the fields in the class. For each field, we define the type of field that it is, Integer for our primary key field, and the rest are Strings, Text.

After sqlalchemy has been setup, we can test if it works as expect by entering `scrapy shell`. Remember to use `%paste` command to paste code block below to test in Scrapy shell. You can check this post for more tips about Scrapy shell Scrapy Tutorial #6: Scrapy Shell Overview & Tips.

```
from sqlalchemy.orm import sessionmaker
from scrapy_spider.models import QuoteDB, db_connect, create_table

engine = db_connect()
create_table(engine)
Session = sessionmaker(bind=engine)

session = Session()
quotedb = QuoteDB()
quotedb.quote = "test qutoe"
```

```
quotedb.author = "test author"

try:
    session.add(quotedb)
    session.commit()

    #query again
    obj = session.query(QuoteDB).first()
    print(obj.quote)
except:
    session.rollback()
    raise
finally:
    session.close()
```

If you are not clear what this code block for, do not worry, I would explain in detail in the next section. What you should know here is that if there is no error raised, then you would see the test quote text printed in terminal, which means SqlAlchemy can work as expected in your env. Now we can move this code block to our pipeline, to make it process item scraped by spider and save the data into Mysql Database.

Pipelining Scraped Data to Database

There is a file `pipelines.py` in scrapy project, which defines pipelines we use in Scrapy project. We can modify `ScrapySpiderPipeline` in the file to make it save data to database as we expect.

```
from sqlalchemy.orm import sessionmaker
from scrapy_spider.models import QuoteDB, db_connect, create_table

class ScrapySpiderPipeline(object):
    def __init__(self):
        """
        Initializes database connection and sessionmaker.
        Creates deals table.
        """
        engine = db_connect()
        create_table(engine)
        self.Session = sessionmaker(bind=engine)

    def process_item(self, item, spider):
        """Save deals in the database.

        This method is called for every item pipeline component.
        """
        session = self.Session()
        quotedb = QuoteDB()
        quotedb.quote = item["quote"]
        quotedb.author = item["author"]

        try:
            session.add(quotedb)
            session.commit()
        except:
            session.rollback()
            raise
        finally:
            session.close()

        return item
```

Here, we have a constructor function, `def __init__(self)` to initialize the class by defining the engine, the data tables in DB. `create_table` function would try to create data table for us if there is no data table in DB.

Every pipeline component class must have a method `process_item`, which would be called sequentially to process scraped item. So in process_item, we establish session with database and set the data to our `QuoteDB` model. Then add it to our session, what you should notice is that the data is not saved to database at this point until you call the `commit` method.

If there is something wrong in the commit method, then `rollback` would help us rollback the transaction in process.

In the end of process_item, we return the item back to the next pipeline to process.

Activate Pipeline

After we finished our custom pipeline, we need to activate it in our settings.py, `ITEM_PIPELINES` is the only one you should modify.

```
ITEM_PIPELINES = {
    'scrapy_spider.pipelines.ScrapySpiderPipeline': 300,
}
```

The number here determines the order in which they run, items go through from lower value to higher value classes. You can set the number from 0-1000.

Run Spider & Check Databse

Now we can run our spider and check the results in DB, `scrapy crawl quotes_spider`, and after the spider finish job, you can login in mysql server to check the results.

```
$ mysql -h localhost -u root -p

mysql> use quote_toscrape;
mysql> select * from quote_table limit 1;

+----+-------------------------------------------------------------------------------
| id | quote
+----+-------------------------------------------------------------------------------
|  1 | test qutoe
|  2 | test qutoe
|  3 | “The world as we have created it is a process of our thinking. It cannot be chang
+----+-------------------------------------------------------------------------------
```

As you can see, since first and second row are inserted in when testing SqlAchemy, so you need to delete the data in production. The recommended way do to this is to add some code in `def __init__` of our Pipeline component class. You can treat this as a little exercise and send me message if you have any problem.

TroubleShoot:

This section would list some errors you might meet if you are trying to insert scraped item to Database using SqSqlalchemy.

`ImportError: No module named 'MySQLdb'`

This error is usually caused by missing mysql connecoter, we can install the issue by installing third-party package.

`pip install mysqlclient` can solve your problem no matter your Python is Python2 or Python3.

Some online old resource would tell you to use `pip install mysql-python` but this package can only work in Python2. So `mysqlclient` is the better option here because it can work on both Python versions.

Conclusion

In this Scrapy tutorial, I talked about how to define Scrapy item, how to use Scrapy item, and how to create a custom Item Pipeline to save the data of Item into DB. You have also learned how to use SQLAlchemy (the most popular ORM in python world) to do some work with Database for you. Since I mainly focus on Scrapy in this post, so I did not talk much about SQLAlchemy in this tutorial, if you have any question when importing the code into your project, just leave me message and I will respond ASAP.

To help user focus on the key part, I only paste part of the source code instead of the whole file in this tutorial, If you want source code which can run in your local env directly, just

```
git clone git@github.com:michael-yin/scrapy-spider-example.git

cd scrapy-spider-example

git checkout ee32e7d

# If you can star my project, I would appreciate that
```