Flask-RESTFul API 和 Blueprint 的结合

原创 FatPuffer 最后发布于2019-03-28 15:29:11 阅读数 684 ☆ 收藏

首先展示一下项目目录树：

```
.
├── frf_demo  # 存放整个项目
│   ├── apps  # 存放应用模块包
│   │   ├── cart  # 购物车模块
│   │   │   ├── __init__.py  # 创建cart应用蓝图
│   │   │   ├── models.py  # 数据库模型
│   │   │   ├── urls.py  # 创建api，进行路由分发
│   │   │   └── views.py  # 处理请求视图类
│   │   ├── db  # 存放应用模型共用字段模型类
│   │   │   ├── base_model.py
│   │   │   └── __init__.py
│   │   ├── goods
│   │   │   ├── __init__.py
│   │   │   ├── models.py
│   │   │   ├── urls.py
│   │   │   └── views.py
│   │   ├── __init__.py
│   │   ├── order
│   │   │   ├── __init__.py
│   │   │   ├── models.py
│   │   │   ├── urls.py
│   │   │   └── views.py
│   │   └── user
│   │       ├── __init__.py
│   │       ├── models.py
│   │       ├── urls.py
│   │       └── views.py
│   ├── __init__.py  # 创建flask应用文件（工厂函数）
│   ├── libs  # 存放一些第三方源码包（方便我们对其修改）
```

```
30 │   ├── static  # 静态文件存储目录
31 │   ├── templates  # 模板目录
32 │   └── utils  # 存放全局共用文件包
33 │       ├── __init__.py
34 │       └──  user.py  # 用户应用所需函数
35 ├── config.py  # 配置文件
36 ├── logs  # 日志文件
37 └── manage.py  # 启动文件
```

详细说明

- **说在前面**：由于本篇文章仅为了说明如何将flask `restful-api` 和 `Blueprint` 结合，所以在此不涉及详细内容

（1）首先我们看一下配置文件 `config.py` 信息

```python
# coding:utf-8
import redis


class Config(object):
    """配置参数"""

    # 设置秘钥，在使用 CSRF 时必须
    SECRET_KEY = "%S-D*d\uk/j@51.65!_?"

    # 连接数据库
    SQLALCHEMY_DATABASE_URL = "myslq://root:123456@127.0.0.1:3306/db_flask"

    # 设置sqlalchemy自动跟踪数据库
    SQLALCHEMY_TRACK_MODIFICATIONS = True

    # redis
    REDIS_HOST = '127.0.0.1'
    REDIS_PORT = 6379

    # flask-session配置
    SESSION_TYPE = 'redis'
    SESSION_REDIS = redis.StrictRedis(host=REDIS_HOST, port=REDIS_PORT)
    SESSION_USE_SIGNER = True  # 对cookie中的session_id进行隐藏处理
    PERMANENT_SESSION_LIFETIME = 86499  # session数据的有效期，单位：秒

```

```
27
28 class DevelopementConfig(Config):
29     """开发模式的配置信息"""
30
31     BEBUG = True
32
33
34 class ProductConfig(Config):
35     """生产环境配置信息"""
36     pass
37
38
39 config_map = {
40     'develope': DevelopementConfig,
41     'product': ProductConfig,
42 }
43
```

(2) 首先我们看一下创建flask应用的文件（`frf_demo/ __init__.py`）内容

```
1  # coding:utf-8
2
3  from flask import Flask
4  import redis
5  import logging
6  from logging.handlers import RotatingFileHandler
7  from flask_session import Session
8  from flask_wtf import CSRFProtect
9  from config import config_map
10 from flask_sqlalchemy import SQLAlchemy
11
12
13 # 数据库
14 db = SQLAlchemy()
15
16 # 创建redis连接
17 redis_store = None
18
19 # 为flask补充csrf防护
20 csrf = CSRFProtect()
21
```

```python
# 设置日志的记录等级
logging.basicConfig(level=logging.DEBUG)
# 创建日志记录器，指明日志保存的路径，每个日志文件的最大大小，保存日志文件个数上限
file_log_handler = RotatingFileHandler('logs/log', maxBytes=1024*1024*1000, backupCount=10)
# 创建日志记录格式                    日志等级    输出日志信息的文件名    行数        日志信息
formatter = logging.Formatter('%(levelname)s %(filename)s:%(lineno)d %(message)s')
# 为刚创建的日志记录器设置日志记录格式
file_log_handler.setFormatter(formatter)
# 为全局的日志工具对象（flask app使用的）添加日志记录器
logging.getLogger().addHandler(file_log_handler)


def create_app(config_name):
    """工厂模式"""

    app = Flask(__name__,
                template_folder='template',
                static_folder='static')

    # 根据配置模式的名字获取配置参数类
    config_class = config_map.get(config_name)
    app.config.from_object(config_class)

    # 使用app初始化db
    db.init_app(app)

    # 初始化redis工具
    global redis_store
    redis_store = redis.StrictRedis(host=config_class.REDIS_HOST, port=config_class.REDIS_PORT)

    # 利用flask-session，将session数据保存到redis中
    Session(app)

    # 初始化
    csrf.init_app(app)

    from apps.cart import app_cart
    from apps.order import app_order
    from apps.goods import app_goods
    from apps.user import app_user

    app.register_blueprint(app_cart, url_prefix='/cart')
```

```
    app.register_blueprint(app_goods, url_prefix='/')
    app.register_blueprint(app_order, url_prefix='/order')
    app.register_blueprint(app_user, url_prefix='/user')

    return app

```

(3) 我们看一下启动文件 `manage.py` 内容

```
# coding:utf-8
from flask_script import Manager
from flask_migrate import Migrate, MigrateCommand
from frf_demo import create_app, db

# 调用创建flask应用文件中的应用生成函数，加入参数说明当前环境
app = create_app('develope')

manager = Manager(app)
Migrate(app, db)
manager.add_command("db", MigrateCommand)

if __name__ == '__main__':
    manager.run()

```

- 说明：当我们通过 `manage.py` 启动flask应用程序时，首先会调用项目 `frf_demo` 下的初始化文件 `__init__.py` 进行创建应用， `__init__.py` 文件在创建flask应用时会根据我们启动文件时所传入的参数从 `config.py` 配置文件中选择对应的配置参数进行项目启动。

(4) 运用蓝图关联我们项目下的每个应用

- 首先我们每个 应用下 都有一个初始化文件 `__init__.py` ， 此文件用来 创建蓝图 ， 将我们创建好的蓝图注册到创建应用的文件下 （ `frf_demo/ __init__.py` ） 。

```
from flask import Blueprint

app_user = Blueprint('user', __name__)

import urls
```

举报

- 其次在我们每个应用下有个 `urls.py` 文件，该文件被我们用来创建 `api`，将我们应用下的蓝图加入 `api` 进行管理，同时进行 路由分发，将试图处理类与请求路径相关联

```python
from flask_restful import Api
from . import app_user
from .views import RegisterView

# 将 user 模块蓝图加入Api进行管理
api = Api(app_user)

# 进行路由分发，类似于Django中的url工作内容（视图处理类， 请求路径）
api.add_resource(RegisterView, '/register')

```

- 在我们的应用模块下还有一个 `models.py` 文件，用来存放该应用下的所有 数据库模型类

```python
from frf_demo import db
from werkzeug.security import generate_password_hash, check_password_hash
from ..db.base_model import BaseModel


class User(BaseModel, db.Model):
    """User"""
    __tablename__ = "frf_user_profile"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(32), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    mobile = db.Column(db.String(11), unique=True, nullable=False)

    @property
    def password(self):
        """called on get password attributes"""
        raise AttributeError('not readable')

    @password.setter
    def password(self, passwd):
        """called on set password attributes , set encrypted password"""
        self.password_hash = generate_password_hash(passwd)

```

```
    def check_password(self, passwd):
        """check password correctness"""
        return check_password_hash(self.password_hash, passwd)

```

- 最后就是 `views.py` 文件，主要用来存放我们的 视图处理类

```
# coding:utf-8

from flask_restful import Resource
from flask_restful import reqparse
from sqlalchemy.exc import IntegrityError
from frf_demo import db
from flask import current_app
from models import User
from frf_demo.utils.user import check_iterate  # 我们自己定义的检测请求参数检验函数


# 验证请求参数
parser = reqparse.RequestParser()
parser.add_argument('uname', type=check_iterate, required=True, location='form', help='用户名重复')
parser.add_argument('password', required=True, type=str, location='form', help='密码不能为空')
parser.add_argument('mobile', type=check_iterate, required=True, location='form', help='密码重复')


class RegisterView(Resource):
    def get(self):
        return 'get retister page'

    def post(self):
        args = parser.parse_args()
        user = User(name=args['uname'], password=args['upwd'], mobile=args['mobile'])
        try:
            db.session.add(user)
            db.commit()

        except IntegrityError as e:
            # 数据库出错回滚
            db.session.rollback()
            # 手机号重复，记录错误日志
            current_app.logger.error(e)
```

```
            return {"errno": 0, "errmsg": "手机号已注册"}

        except Exception as e:
            # 数据库出错回滚
            db.session.rollback()
            current_app.logger.error(e)
            return {"errno": 0, "errmsg": "数据库查询异常"}

        return {"errno": 1, "errmsg": "注册成功"}
```

- 用户应用下请求参数校验文件（ `frf_demo/utils/user.py` ）内容

```
# coding: utf-8

from frf_demo.apps.user.models import User


def check_iterate(value):

    if value == 'mobile':
        mobile = User.query.filter_by(mobile=value)
        if mobile:
            raise ValueError("手机号已注册")
        else:
            return value
    elif value == 'uname':
        uname = User.query.filter_by(name=value)
        if uname:
            raise ValueError("用户名已注册")
        else:
            return value

```