05-Flask类视图和RESTful

Flask-RESTful

https://flask-restful.readthedocs.io/en/latest/

基本使用

- 安装

```
# pip 安装
pip install flask-restful

# 源码安装
# git clone https://github.com/flask-restful/flask-restful.git
# python setup.py develop
```

- 创建Resource实现类

```
class HelloRESTful(Resource):
    def get(self):
        return {"data":"Hello GET"}
    def post(self):
        return {'data':'hello POST'}
```

- 创建API对象，并注册路由

```
# 方式一： 创建并初始化
api = API(app)
# 方式二： 创建，之后初始化
api = API()
api.init_app(app)

# 注册路由
api.add_resource(HelloRESTFul,"/")
```

字段格式化

```
fields进行定义
marshal_with进行使用
  特性
        显示我们设计的数据结构
        默认返回的数据如果在预定义结构中不存在，数据会被自动过滤
```

如果返回的数据在预定义的结构中存在，数据会正常返回
如果返回的数据比预定义结构中的字段少，预定义的字段会呈现一个默认值

定义字段输出
使用字典进行定义
常用都是基本类型：String, Integer

```
# 格式化字段
user_fields = {
    'msg': fields.String,
    'status': fields.Integer,
    'data': fields.String(attribute='private_data'),
    'default_data': fields.String(default='1')
}
```

定义好的格式通过装饰器进行使用
@marshal_with(需要返回的数据格式)， return返回字典就ok了

```
class Users(Resource):
    @marshal_with(user_fields)
    def get(self):
        return {'msg':'呵呵', 'data':'没有数据', 'age':'22',
'private_data':'表中数据'}
```

级联数据：嵌套字典
Nested

```
# 格式化字段
usermodel_fileds = {
    'id': fields.Integer,
    'name': fields.String,
}

user2_fields = {
    'msg': fields.String(default='ok'),
    'status': fields.Integer(default=1),
    'data': fields.Nested(usermodel_fileds)
}
```

结构允许嵌套列表
fields.List(fields.Nested)

```
# 格式化字段
usermodel_fileds = {
    'id': fields.Integer,
    'name': fields.String,
}
users3_fields = {
```

```
            'status': fields.String(default=1),
            'msg': fields.String,
            'data': fields.List(fields.Nested(usermodel_fileds))
        }
```

URL

连接字段
    就是将当前数据的操作api暴露出来
    根据提供的url和唯一标识进行数据操作

```
# 格式化字段
usermodel_fileds = {
    'id': fields.Integer,
    'name': fields.String,
    'url': fields.Url('id', absolute=True)
}

# 在add_resource中提供对应的endpoint
api.add_resource(Users4, '/user4/', endpoint='id')
```

参数解析

可以不通过request.form或request.args获取参数，而是通过reqparse.RequestParser来解析

```
    # 参数转换器
    parser = reqparse.RequestParser()
    parser.add_argument('name', type=str, action='append')  # 支持多个name
    parser.add_argument('id', type=int, required=True, help='id是必须的') # 必需参数
    parser.add_argument('fldt_active', type=str, location='cookies')  # 获取cookies中的数据

    class Users4(Resource):
        def get(self):
            parse = parser.parse_args()
            user_name = parse.get('name')
            id = parse.get('id')
            fldt_active = parse.get('fldt_active')
            return {'name': user_name, 'id': id, 'fldt_active':fldt_active}
```