Restful Framework

一、restful api

可以总结为一句话：REST是所有Web应用都应该遵守的架构设计指导原则。Representational State Transfer，翻译是”表现层状态转化”。 **面向资源是REST最明显的特征**，对于同一个资源的一组不同的操作。资源是服务器上一个可命名的抽象概念，资源是以名词为核心来组织的，首先关注的是名词。REST要求，必须通过统一的接口来对资源执行各种操作。对于每个资源只能执行一组有限的操作。（7个HTTP方法：GET/POST/PUT/DELETE/PATCH/HEAD/OPTIONS）

1、Restful API设计规范

- **资源**。首先是弄清楚资源的概念。资源就是网络上的一个实体，一段文本，一张图片或者一首歌曲。资源总是要通过一种载体来反应它的内容。文本可以用TXT，也可以用HTML或者XML、图片可以用JPG格式或者PNG格式，JSON是现在最常用的资源表现形式。
- **统一接口**。RESTful风格的数据元操CRUD（create,read,update,delete）分别对应HTTP方法：GET用来获取资源，POST用来新建资源（也可以用于更新资源），PUT用来更新资源，DELETE用来删除资源，这样就统一了数据操作的接口。
- **URI**。可以用一个URI（统一资源标识符）指向资源，即每个URI都对应一个特定的资源。要获取这个资源访问它的URI就可以，因此URI就成了每一个资源的地址或识别符。一般的，每个资源至少有一个URI与之对应，最典型的URI就是URL。
- **无状态**。所谓无状态即所有的资源都可以URI定位，而且这个定位与其他资源无关，也不会因为其他资源的变化而变化。有状态和无状态的区别，举个例子说明一下，

例如要查询员工工资的步骤为第一步：登录系统。第二步：进入查询工资的页面。第三步：搜索该员工。第四步：点击姓名查看工资。这样的操作流程就是有状态的，查询工资的每一个步骤都依赖于前一个步骤，只要前置操作不成功，后续操作就无法执行。如果输入一个URL就可以得到指定员工的工资，则这种情况就是无状态的，因为获取工资不依赖于其他资源或状态，且这种情况下，员工工资是一个资源，由一个URL与之对应可以通过HTTP中的GET方法得到资源，这就是典型的

RESTful风格。

- **RESTful API还有其他一些规范。**
  - 应该将API的版本号放入URL。
    GET:http://www.xxx.com/v1/friend/123。或者将版本号放在HTTP头信息中。版本号取决于自己开发团队的习惯和业务的需要，不是强制的。

    ```
    http://www.example.com/app/1.0/foo

    http://www.example.com/app/1.1/foo

    http://www.example.com/app/2.0/foo
    ```

    - 另一种做法是，将版本号放在HTTP头信息中，但不如放入URL方便和直观。Github采用这种做法。

      因为不同的版本，可以理解成同一种资源的不同表现形式，所以应该采用同一个URL。版本号可以在HTTP请求头信息的Accept字段中进行区分（参见Versioning REST Services）：

    ```
    Accept: vnd.example-com.foo+json; version=1.0

    Accept: vnd.example-com.foo+json; version=1.1

    Accept: vnd.example-com.foo+json; version=2.0
    ```

  - **资源作为网址，只能有名词，不能有动词，而且所用的名词往往与数据库的表名对应。**

    举例来说，以下是不好的例子:

    ```
    /getProducts
    /listOrders
    /retreiveClientByOrder?orderId=1
    ```

    对于一个简洁结构，你应该始终用名词。 此外，利用的HTTP方法可以分离网址中的资源名称的操作。

```
GET /products ：将返回所有产品清单
POST /products ：将产品新建到集合
GET /products/4 ：将获取产品 4
PUT /products/4 ：将更新产品 4
```

- **API中的名词应该使用复数。无论子资源或者所有资源。**

```
获取单个产品：http://127.0.0.1:8080/AppName/rest/products/1
获取所有产品：http://127.0.0.1:8080/AppName/rest/products
```

- 如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果。

```
?limit=10：指定返回记录的数量
?offset=10：指定返回记录的开始位置。
?page=2&per_page=100：指定第几页，以及每页的记录数。
?sortby=name&order=asc：指定返回结果按照哪个属性排序，以及排序顺序。
?animal_type_id=1：指定筛选条件
```

2、到底什么是RESTful架构

- 每一个URI代表一种资源
- 客户端和服务器之间，传递这种资源的某种表现层
- 客户端通过四个HTTP动词，对服务端资源进行操作，实现"表现层状态转换"

3、HTTP常用动词

对于资源的具体操作类型，由HTTP动词表示。

常用的HTTP动词有下面四个（括号里是对应的SQL命令）。

- GET（SELECT）：从服务器取出资源
- POST（CREATE or UPDATE）：在服务器创建资源或更新资源
- PUT（UPDATE）：在服务器更新资源（客户端提供改变后的完整资源）
- DELETE（DELETE）：从服务器删除资源

还有三个不常用的HTTP动词。

- HEAD：获取资源的元数据
- OPTIONS：获取信息，关于资源的哪些属性是客户端可以改变的

- PATCH（UPDATE）：在服务器更新资源（客户端提供改变的属性）

示例：

```
GET /students：获取所有学生
POST /students：新建学生
GET /students/id：获取某一个学生
PUT /students/id ：更新某个学生的信息（需要提供学生的全部信息）
PATCH /students/id：更新某个学生的信息（需要提供学生变更部分信息）
DELETE /students/id：删除某个学生
```

4、restful相关的网络请求状态码

- 200 OK - [GET]：服务器成功返回用户请求的数据
- 201 CREATED -[POST/PUT/PATCH]：用户新建或修改数据成功
- 202 Accepted - [*]：表示一个请求已经进入后台排队（异步任务）
- 204 NO CONTENT - [DELETE]：表示数据删除成功
- 400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误
- 401 Unauthorized - [*]：表示用户没有权限（令牌，用户名，密码错误）
- 403 Forbidden - [*]：表示用户得到授权，但是访问是被禁止的
- 404 NOT FOUND - [*]：用户发出的请求针对的是不存在的记录
- 406 Not Acceptable - [*]：用户请求格式不可得
- 410 Gone - [GET]：用户请求的资源被永久移除，且不会再得到的
- 422 Unprocesable entity -[POST/PUT/PATCH]：当创建一个对象时，发生一个验证错误
- 500 INTERNAL SERVER EROR - [*]：服务器内部发生错误

状态码的完全列表参见这里或这里。

5.错误处理（Error handling）

如果状态码是4xx，服务器就应该向用户返回出错信息。一般来说，返回的信息中将error作为键名，出错信息作为键值即可。

```
{
    error: "Invalid API key"
}
```

6.返回结果

针对不同操作，服务器向用户返回的结果应该符合以下规范。

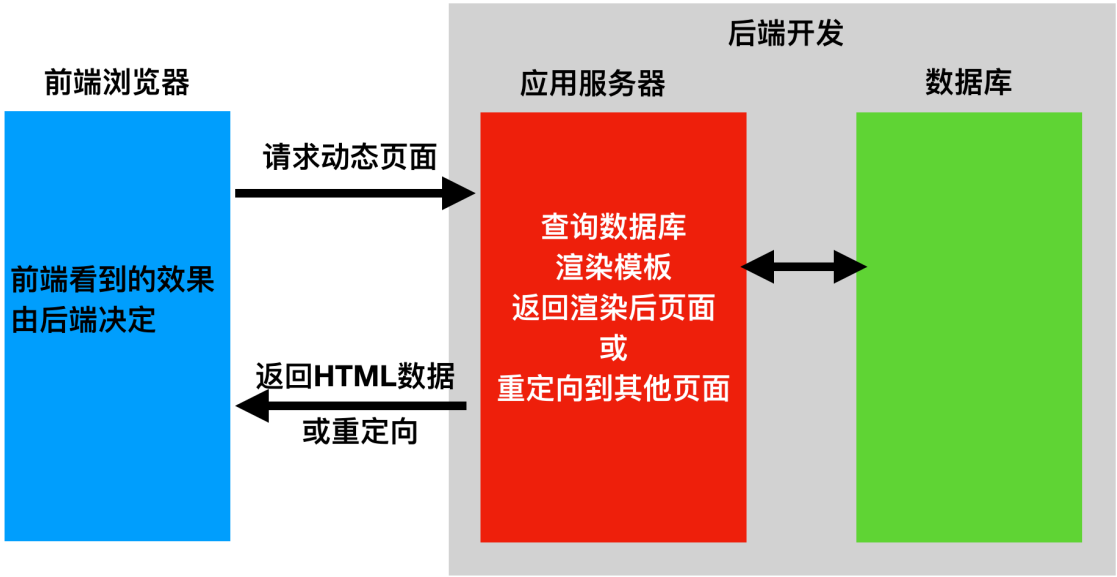
```
GET /collection：返回资源对象的列表（数组）
GET /collection/resource：返回单个资源对象
POST /collection：返回新生成的资源对象
PUT /collection/resource：返回完整的资源对象
PATCH /collection/resource：返回完整的资源对象
```

二、DRF框架

1、web应用模式

- **前后端不分离**

  在前后端不分离的引用模式中，前端页面看到的效果都是由后端控制的，由后端页面渲染或者重定向，也就是后端需要控制前端的展示，前端与后端的耦合度很高，这种模式比较适合纯网页应用，但是后端对接APP时，App可能并不需要后端返回一个HTML网页,二仅仅是数据本身,所以后端原本返回网页的接口不再适用前端APP应用,为了对接APP后端还需再开发一套接口。

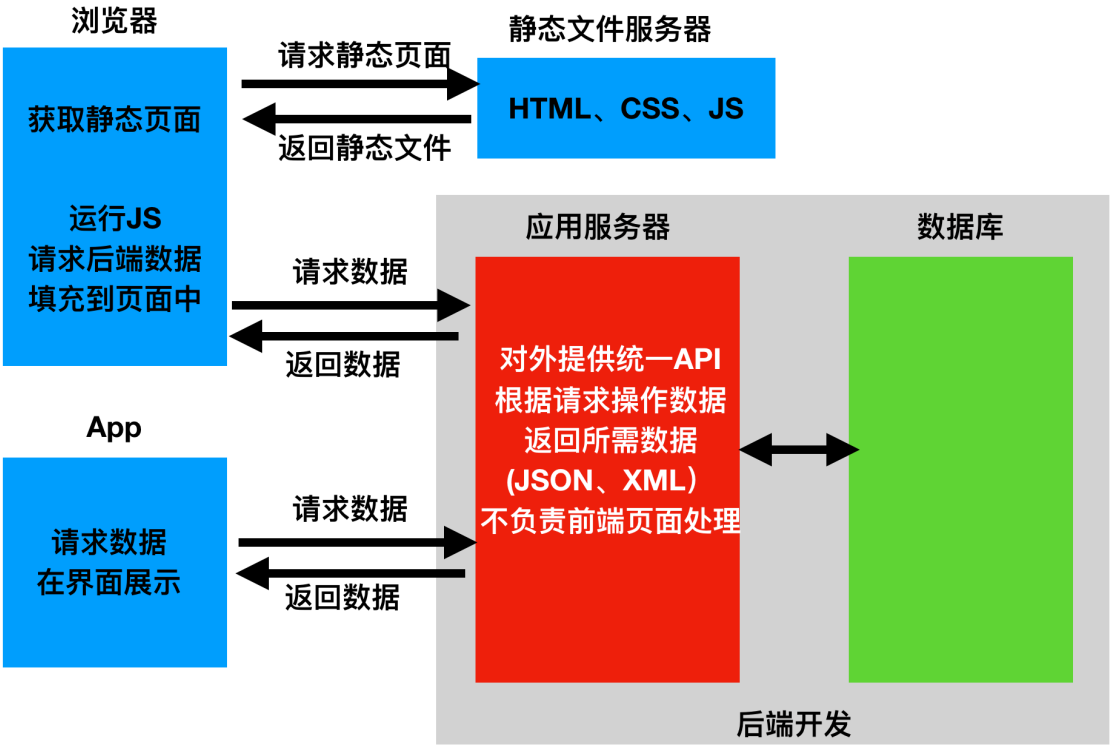


- **前后端分离**

  在前后端分离的应用模式中，后端仅返回前端所需要的数据，不再渲染HTML页面，不再控制前端的效果，只要前端用户看到什么效果，从后端请求的数据如何加载到前端中，都由前端自己决定，网页有网页自己的处理方式，APP有APP的处理方式，但无论哪种前端所需要的数据基本相同，后端仅需开发一套逻辑对外提供数据即可，在前后端分离的应用模式中,前端与后端的耦合度相对

较低。

在前后端分离的应用模式中，我们通常将后端开发的每个视图都称为一个**接口**，或者**API**，前端通过访问接口来对数据进行增删改查。



2、Django Rest Framework

Django Rest Framework （DRF）是一个强大且灵活的工具包，用以构建Web API。 Django REST Framework可以在Django的基础上迅速实现API，并且自身还带有WEB的测试页面，可以方便的测试自己的API。

- 特性
  - 可浏览API
  - 提供丰富认证
  - 支持数据序列化
  - 可以轻量嵌入，仅使用fbv
  - 强大的社区支持

官方网站：

```
https://www.django-rest-framework.org/
```

中文翻译网站:

```
https://q1mi.github.io/Django-REST-framework-documentation/
```

- 环境的安装和配置

  DRF依赖于：

  - Python (3.5, 3.6, 3.7, 3.8)
  - Django (1.11, 2.0, 2.1, 2.2, 3.0)

  DRF是以Django扩展应用的方式提供的，所以我们可以直接利用已有的Django环境而无需从新创建。（若没有Django环境，需要先创建环境安装Django）

  - 安装djangorestframework

  ```
  pip install djangorestframework
  ```

  - 添加rest_framework应用

  在**settings.py**的**INSTALLED_APPS**中添加'rest_framework'。

  ```
  INSTALLED_APPS = [
      ...
      'rest_framework',
  ]
  ```

3、使用Django开发REST 接口

我们以在Django框架中使用的图书英雄案例来写一套支持图书数据增删改查的REST API接口，来理解REST API的开发。

在此案例中，前后端均发送JSON格式数据。

```
# views.py

from datetime import datetime

class BooksAPIVIew(View):
    """
    查询所有图书、增加图书
    """
    def get(self, request):
        """
```

```
        查询所有图书
        路由：GET /books/
        """
        queryset = BookInfo.objects.all()
        book_list = []
        for book in queryset:
            book_list.append({
                'id': book.id,
                'btitle': book.btitle,
                'bpub_date': book.bpub_date,
                'bread': book.bread,
                'bcomment': book.bcomment,
                'image': book.image.url if book.image else ''
            })
        return JsonResponse(book_list, safe=False)

    def post(self, request):
        """
        新增图书
        路由：POST /books/
        """
        json_bytes = request.body
        json_str = json_bytes.decode()
        book_dict = json.loads(json_str)

        # 此处详细的校验参数省略

        book = BookInfo.objects.create(
            btitle=book_dict.get('btitle'),
            bpub_date=datetime.strptime(book_dict.get('bpub_date'),
'%Y-%m-%d').date()
        )

        return JsonResponse({
            'id': book.id,
            'btitle': book.btitle,
            'bpub_date': book.bpub_date,
            'bread': book.bread,
            'bcomment': book.bcomment,
            'image': book.image.url if book.image else ''
        }, status=201)
```

```
class BookAPIView(View):
    def get(self, request, pk):
        """
        获取单个图书信息
        路由：  GET  /books/<pk>/
        """
        try:
            book = BookInfo.objects.get(pk=pk)
        except BookInfo.DoesNotExist:
            return HttpResponse(status=404)

        return JsonResponse({
            'id': book.id,
            'btitle': book.btitle,
            'bpub_date': book.bpub_date,
            'bread': book.bread,
            'bcomment': book.bcomment,
            'image': book.image.url if book.image else ''
        })

    def put(self, request, pk):
        """
        修改图书信息
        路由：  PUT  /books/<pk>
        """
        try:
            book = BookInfo.objects.get(pk=pk)
        except BookInfo.DoesNotExist:
            return HttpResponse(status=404)

        json_bytes = request.body
        json_str = json_bytes.decode()
        book_dict = json.loads(json_str)

        # 此处详细的校验参数省略

        book.btitle = book_dict.get('btitle')
        book.bpub_date =
datetime.strptime(book_dict.get('bpub_date'), '%Y-%m-%d').date()
        book.save()
```

```
        return JsonResponse({
            'id': book.id,
            'btitle': book.btitle,
            'bpub_date': book.bpub_date,
            'bread': book.bread,
            'bcomment': book.bcomment,
            'image': book.image.url if book.image else ''
        })

    def delete(self, request, pk):
        """
        删除图书
        路由： DELETE /books/<pk>/
        """
        try:
            book = BookInfo.objects.get(pk=pk)
        except BookInfo.DoesNotExist:
            return HttpResponse(status=404)

        book.delete()

        return HttpResponse(status=204)
```

```
# urls.py

urlpatterns = [
    url(r'^books/$', views.BooksAPIVIew.as_view()),
    url(r'^books/(?P<pk>\d+)/$', views.BookAPIView.as_view())
]
```

测试

使用Postman测试上述接口

1） 获取所有图书数据

GET 方式访问 http://127.0.0.1:8000/books/,_ 返回状态码200，数据如下：

```
[
    {
        "id": 1,
        "btitle": "射雕英雄传",
```

```
        "bpub_date": "1980-05-01",
        "bread": 12,
        "bcomment": 34,
        "image": ""
    },
    {
        "id": 2,
        "btitle": "天龙八部",
        "bpub_date": "1986-07-24",
        "bread": 36,
        "bcomment": 40,
        "image": ""
    },
    {
        "id": 3,
        "btitle": "笑傲江湖",
        "bpub_date": "1995-12-24",
        "bread": 20,
        "bcomment": 80,
        "image": ""
    },
    {
        "id": 4,
        "btitle": "雪山飞狐",
        "bpub_date": "1987-11-11",
        "bread": 58,
        "bcomment": 24,
        "image": ""
    },
    {
        "id": 5,
        "btitle": "西游记",
        "bpub_date": "1988-01-01",
        "bread": 10,
        "bcomment": 10,
        "image": "booktest/xiyouji.png"
    },
    {
        "id": 6,
        "btitle": "水浒传",
        "bpub_date": "1992-01-01",
        "bread": 10,
```

```
        "bcomment": 11,
        "image": ""
    },
    {
        "id": 7,
        "btitle": "红楼梦",
        "bpub_date": "1990-01-01",
        "bread": 0,
        "bcomment": 0,
        "image": ""
    }
]
```

2）获取单一图书数据

GET 访问 http://127.0.0.1:8000/books/5/ ， 返回状态码200， 数据如下：

```
{
    "id": 5,
    "btitle": "西游记",
    "bpub_date": "1988-01-01",
    "bread": 10,
    "bcomment": 10,
    "image": "booktest/xiyouji.png"
}
```

GET 访问http://127.0.0.1:8000/books/100/，返回状态码404

3）新增图书数据

POST 访问http://127.0.0.1:8000/books/，发送JSON数据：

```
{
    "btitle": "三国演义",
    "bpub_date": "1990-02-03"
}
```

返回状态码201，数据如下：

```
{
    "id": 8,
    "btitle": "三国演义",
    "bpub_date": "1990-02-03",
    "bread": 0,
    "bcomment": 0,
    "image": ""
}
```

4）修改图书数据

PUT 访问http://127.0.0.1:8000/books/8/，发送JSON数据：

```
{
    "btitle": "三国演义（第二版）",
    "bpub_date": "1990-02-03"
}
```

返回状态码200，数据如下：

```
{
    "id": 8,
    "btitle": "三国演义（第二版）",
    "bpub_date": "1990-02-03",
    "bread": 0,
    "bcomment": 0,
    "image": ""
}
```

5）删除图书数据

DELETE 访问http://127.0.0.1:8000/books/8/，返回204状态码