RESTful架构和DRF入门

Django Restful（DRF）框架学习（一）Restful架构概念
一、诞生背景
REST全称是Representational State Transfer，表征性状态转移。
如果架构符合REST的约束条件和原则，就成为RESTful架构
二、解释
2.1 资源与URI
资源可以是实体（例如手机号码），也可以只是一个抽象概念（例如资源）。
eg:
某用户的手机号
某用户的个人信息
两个产品指甲你的依赖关系
某手机号码的潜在价值

URI：Uniform Resource Identifier，统一资源标识符，用来唯一标识一个资源。资源必须由URI标识。
URI通常使用一下通用的符号
使用_或-来让URI可读性更好
使用/来表示资源的层级关系
使用?来过滤
使用,或;可以表示同级资源的关系
URI不应该使用动作来描述，例如如下的都是不符合规范的URI：
GET /getUser/1
POST /createUser
PUT /updateUser/1
DELETE /deleteUser/1
2.2 统一资源接口
RESTful架构应使用一组受限的预定义的接口。不论是什么资源，都要通过相同的接口进行资源的访问。接口应该使用标准的HTTP方法：

GET
安全且幂等
获取表示
变更时获取表示（缓存）
200（OK） - 表示已在响应中发出
204（无内容） - 资源有空表示
301（Moved Permanently） - 资源的URI已被更新
303（See Other） - 其他（如，负载均衡）
304（not modified）- 资源未更改（缓存）
400 （bad request）- 指代坏请求（如，参数错误）
404 （not found）- 资源不存在
406 （not acceptable）- 服务端不支持所需表示
500 （internal server error）- 通用错误响应
503 （Service Unavailable）- 服务端当前无法处理请求
POST
不安全且不幂等
使用服务端管理的（自动产生）的实例号创建资源
创建子资源
部分更新资源
如果没有被修改，则不过更新资源（乐观锁）

200（OK）- 如果现有资源已被更改
201（created）- 如果新资源被创建
202（accepted）- 已接受处理请求但尚未完成（异步处理）
301（Moved Permanently）- 资源的URI被更新
303（See Other）- 其他（如，负载均衡）
400（bad request）- 指代坏请求
404 （not found）- 资源不存在
406 （not acceptable）- 服务端不支持所需表示
409 （conflict）- 通用冲突
412 （Precondition Failed）- 前置条件失败（如执行条件更新时的冲突）
415 （unsupported media type）- 接受到的表示不受支持
500 （internal server error）- 通用错误响应
503 （Service Unavailable）- 服务当前无法处理请求
PUT
不安全但幂等
用客户端管理的实例号创建一个资源
通过替换的方式更新资源
如果未被修改，则更新资源（乐观锁）
200 （OK）- 如果已存在资源被更改
201 （created）- 如果新资源被创建
301（Moved Permanently）- 资源的URI已更改
303 （See Other）- 其他（如，负载均衡）
400 （bad request）- 指代坏请求
404 （not found）- 资源不存在
406 （not acceptable）- 服务端不支持所需表示
409 （conflict）- 通用冲突
412 （Precondition Failed）- 前置条件失败（如执行条件更新时的冲突）
415 （unsupported media type）- 接受到的表示不受支持
500 （internal server error）- 通用错误响应
503 （Service Unavailable）- 服务当前无法处理请求
DELETE
不安全但幂等
删除资源
200 （OK）- 资源已被删除
301 （Moved Permanently）- 资源的URI已更改
303 （See Other）- 其他，如负载均衡
400 （bad request）- 指代坏请求
404 （not found）- 资源不存在
409 （conflict）- 通用冲突
500 （internal server error）- 通用错误响应
503 （Service Unavailable）- 服务端当前无法处理请求
其他的HTTP方法
WebDAV的LOCK和UPLOCK，github的对issue进行更新的PATCH方法。
RESTful架构必须充分利用响应状态码说明状态，而不是通过响应体。
2.3 资源的表述
我们通过HTTP方法获取的资源是资源的表述，资源的表述包括数据和描述数据的元数据，例如HTTP头“Content-Type”就是元数据属性。
一些常见的设计：
在URI里边带上版本号
使用URI后缀来区分表述格式（通常不会这么做）
2.4 资源的链接
可以在响应头里边增加Link头告诉客户端怎么访问下一页和最后一页的记录。

2.5 应用状态和资源状态
客户端负责维护应用状态，服务端维护资源状态。
若Cookie保存的是服务器不依赖于会话状态既可验证的信息（如认证令牌），这样的Cookie也是符合REST原则的。

DRF以及Restful

Web应用模式

在开发Web应用中，有两种应用模式：

- 前后端不分离
- 前后端分离

1 前后端不分离

在前后端不分离的应用模式中，前端页面看到的效果都是由后端控制，由后端渲染页面或重定向，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。

这种应用模式比较适合纯网页应用，但是当后端对接App时，App可能并不需要后端返回一个HTML网页，而仅仅是数据本身，所以后端原本返回网页的接口不再适用于前端App应用，为了对接App后端还需再开发一套接口。

2 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的数据，不再渲染HTML页面，不再控制前端的效果。至于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，网页有网页的处理方式，App有App的处理方式，但无论哪种前端，所需的数据基本相同，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。

认识RESTful

即Representational State Transfer的缩写。维基百科称其为“具象状态传输”，国内大部分人理解为“表现层状态转化”。

RESTful是一种开发理念。维基百科说：REST是设计风格而不是标准。 REST描述的是在网络中client和server的一种交互形式；REST本身不实用，实用的是如何设计 RESTful API（REST风格的网络接口），一种万维网软件架构风格。

我们先来具体看下RESTful风格的url,比如我要查询商品信息，那么

> 非REST的url：http://.../queryGoods?id=1001&type=t01
>
> REST的url: http://.../t01/goods/1001

可以看出REST特点：url简洁，将参数通过url传到服务器，而传统的url比较啰嗦，而且现实中浏览器地址栏会拼接一大串字符，想必你们都见过吧。但是采用REST的风格就会好很多，现在很多的网站已经采用这种风格了，这也是潮流方向，典型的就是url的短化转换。

那么，到底什么是RESTful架构： 如果一个架构符合REST原则，就称它为RESTful架构。

要理解RESTful架构，理解Representational State Transfer这三个单词的意思。

- 具象的，就是指表现层，要表现的对象也就是“资源”，什么是资源呢？网站就是资源共享的东西，客户端（浏览器）访问web服务器，所获取的就叫资源。比如html，txt，json，图片，视频等等。
- 表现，比如，文本可以用txt格式表现，也可以用HTML格式、XML格式、JSON格式表现，甚至可以采用二进制格式；图片可以用JPG格式表现，也可以用PNG格式表现。

  浏览器通过URL确定一个资源，但是如何确定它的具体表现形式呢？应该在HTTP请求的头信息中用Accept和Content-Type字段指定，这两个字段才是对"表现层"的描述。
- 状态转换， 就是客户端和服务器互动的一个过程，在这个过程中, 势必涉及到数据和状态的变化，这种变化叫做状态转换。

  互联网通信协议HTTP协议，客户端访问必然使用HTTP协议，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。

  HTTP协议实际上含有4个表示操作方式的动词，分别是 GET,POST,PUT,DELETE,他们分别对应四种操作。GET用于获取资源，POST用于新建资源，PUT用于更新资源，DElETE用于删除资源。GET和POST是表单提交的两种基本方式，比较常见，而PUT和DElETE不太常用。

  而且HTTP协议是一种无状态协议，这样就必须把所有的状态都保存在服务器端。因此，如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）

总结

综合上面的解释，RESTful架构就是：

- 每一个URL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，对服务器端资源进行操作，实现"表现层状态转化"。

RESTful设计方法

1. 域名

应该尽量将API部署在专用域名之下。

```
https://api.example.com
```

如果确定API很简单，不会有进一步扩展，可以考虑放在主域名下。

```
https://example.org/api/
```

2. 版本（Versioning）

应该将API的版本号放入URL。

```
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
```

3. 路径（Endpoint）

路径又称"终点"（endpoint），表示API的具体网址，每个网址代表一种资源（resource）

(1) 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
PATCH（或）PUT /products/4 ：将更新产品 4
```

(2) API中的名词应该使用复数。无论子资源或者所有资源。

举例来说，获取产品的API可以这样定义

```
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
```

3. HTTP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（一项或多项）。
- POST（CREATE）：在服务器新建一个资源。
- 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。

- PATCH（UPDATE）：在服务器更新(更新)资源（客户端提供改变的属性）。
- 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。

下面是一些例子。

```
GET /zoos：列出所有动物园
POST /zoos：新建一个动物园（上传文件）
GET /zoos/ID：获取某个指定动物园的信息
PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）
PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）
DELETE /zoos/ID：删除某个动物园
GET /zoos/ID/animals：列出某个指定动物园的所有动物
DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物
```

4. 过滤信息（Filtering）

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。

下面是一些常见的参数。

```
?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
```

参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复。比如，GET /zoos/ID/animals 与 GET /animals?zoo_id=ID 的含义是相同的。

6. 状态码（Status Codes）

服务器向用户返回的状态码和提示信息，常见的有以下一些（方括号中是该状态码对应的HTTP动词）。

> - 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]：用户删除数据成功。
> - 400 INVALID REQUEST - [POST/PUT/PATCH]：用户发出的请求有错误，服务器没有进行新建或修改数据的操作
> - 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（与401错误相对），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，服务器没有进行操作，该操作是幂等的。
> - 406 Not Acceptable - [GET]：用户请求的格式不可得（比如用户请求JSON格式，但是只有XML格式）。
> - 410 Gone -[GET]：用户请求的资源被永久删除，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 ERROR - [*]：服务器发生错误，用户将无法判断发出的请求是否成功。

状态码的完全列表参见这里或这里。

7. 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"
}
```

8. 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
- DELETE /collection/resource：返回一个空文档

9. 超媒体（Hypermedia API）

RESTful API最好做到Hypermedia（即返回结果中提供链接，连向其他API方法），使得用户不查文档，也知道下一步应该做什么。

比如，Github的API就是这种设计，访问api.github.com会得到一个所有可用API的网址列表。

```
{
"current_user_url": "https://api.github.com/user",
"authorizations_url": "https://api.github.com/authorizations",
// ...
}
```

从上面可以看到，如果想获取当前用户的信息，应该去访问api.github.com/user，然后就得到了下面结果。

```
{
  "message": "Requires authentication",
  "documentation_url": "https://developer.github.com/v3"
}
```

上面代码表示，服务器给出了提示信息，以及文档的网址。

10. 其他

服务器返回的数据格式，应该尽量使用JSON，避免使用XML。

明确REST接口开发的核心任务

分析一下上节的案例，可以发现，在开发REST API接口时，视图中做的最主要有三件事：

- 将请求的数据（如JSON格式）转换为模型类对象
- 操作数据库
- 将模型类对象转换为响应的数据（如JSON格式）

序列化Serialization

维基百科中对于序列化的定义：

序列化（serialization）在计算机科学的资料处理中，是指将数据结构或物件状态转换成可取用格式（例如存成档案，存于缓冲，或经由网络中传送），以留待后续在相同或另一台计算机环境中，能恢复原先状态的过程。依照序列化格式重新获取字节的结果时，可以利用它来产生与原始物件相同语义的副本。对于许多物件，像是使用大量参照的复杂物件，这种序列化重建的过程并不容易。面向对象中的物件序列化，并不概括之前原始物件所关联的函式。这种过程也称为物件编组（marshalling）。从一系列字节提取数据结构的反向操作，是反序列化（也称为解编组, deserialization, unmarshalling）。

序列化在计算机科学中通常有以下定义：

在数据储存与传送的部分是指将一个对象存储至一个储存媒介，例如档案或是记忆体缓冲等，或者透过网络传送资料时进行编码的过程，可以是字节或是XML等格式。而字节的或XML编码格式可以还原完全相等的对象。这程序被应用在不同应用程序之间传送对象，以及服务器将对象储存到档案或数据库。相反的过程又称为反序列化。

简而言之，我们可以将序列化理解为：

将程序中的一个数据结构类型转换为其他格式（字典、JSON、XML等），例如将Django中的模型类对象装换为JSON字符串，这个转换过程我们称为序列化。

如：

```
queryset = BookInfo.objects.all()
book_list = []
# 序列化
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)
```

反之，将其他格式（字典、JSON、XML等）转换为程序中的数据，例如将JSON字符串转换为Django中的模型类对象，这个过程我们称为反序列化。

如：

```
json_bytes = request.body
json_str = json_bytes.decode()

# 反序列化
book_dict = json.loads(json_str)
book = BookInfo.objects.create(
    btitle=book_dict.get('btitle'),
    bpub_date=datetime.strptime(book_dict.get('bpub_date'), '%Y-%m-%d').date()
)
```

我们可以看到，在开发REST API时，视图中要频繁的进行序列化与反序列化的编写。

总结

在开发REST API接口时，我们在视图中需要做的最核心的事是：

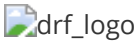
- 将数据库数据序列化为前端所需要的格式，并返回；
- 将前端发送的数据反序列化为模型类对象，并保存到数据库中。

Django REST framework 简介

1. 在序列化与反序列化时，虽然操作的数据不尽相同，但是执行的过程却是相似的，也就是说这部分代码是可以复用简化编写的。
2. 在开发REST API的视图中，虽然每个视图具体操作的数据不同，但增、删、改、查的实现流程基本套路化，所以这部分代码也是可以复用简化编写的：
    - 增：校验请求数据 -> 执行反序列化过程 -> 保存数据库 -> 将保存的对象序列化并返回
    - 删：判断要删除的数据是否存在 -> 执行数据库删除
    - 改：判断要修改的数据是否存在 -> 校验请求的数据 -> 执行反序列化过程 -> 保存数据库 -> 将保存的对象序列化并返回
    - 查：查询数据库 -> 将数据序列化并返回

Django REST framework可以帮助我们简化上述两部分的代码编写，大大提高REST API的开发速度。

认识Django REST framework



Django REST framework 框架是一个用于构建Web API 的强大而又灵活的工具。

通常简称为DRF框架 或 REST framework。

DRF框架是建立在Django框架基础之上，由Tom Christie大牛二次开发的开源项目。

特点

- 提供了定义序列化器Serializer的方法，可以快速根据 Django ORM 或者其它库自动序列化/反序列化；
- 提供了丰富的类视图、Mixin扩展类，简化视图的编写；
- 丰富的定制层级：函数视图、类视图、视图集合到自动生成 API，满足各种需要；
- 多种身份认证和权限认证方式的支持；
- 内置了限流系统；
- 直观的 API web 界面；
- 可扩展性，插件丰富

资料：

- 官方文档
- Github源码