**前后端分离基础**

**什么是前后端分离？**

<https://blog.csdn.net/fuzhongmin05/article/details/81591072>

简单来说就是实现了前后端互为黑箱。前端开发人员不必关心后端用的是什么，只需要拿到一个url作为接口即可。后端也不用学前端框架。比如这次使用的疫情信息api：<https://github.com/BlankerL/DXY-COVID-19-Crawler>（前端使用方法在本目录下另一篇FE文档尾部）。

应用场景举例：前端拿到用户所在位置信息，通过某restful接口实现信息传递，后端调出附近疫情情况，前端再用接口读取。

**Restful接口**

概念：<https://www.restapitutorial.com/>

方法：<https://spring.io/guides/gs/rest-service/>

**一些工具**

<https://github.com/yuan-xy/kaola> 似乎是实现了快捷生成restful接口

**Note**

前端需要后端提供的接口主要有：

如果需要用户注册，需给一个接口调用户信息，包括可能的论坛发言什么的（这个不急，前端同学也不太会搞）

疫情相关的信息前端可以直接调github那个接口，后端不必关心。当然如果想把数据下载下来再提供给前端也可行，毕竟那个接口常常不稳定

**使用Django REST Framework创建RESTful接口**

入门：<https://blog.stdioa.com/2017/03/DRF-demo/>

<https://www.jianshu.com/p/2538e8aa1ead>

文档：<https://www.django-rest-framework.org/>

参考视频：<https://www.youtube.com/watch?v=Uyei2iDA4Hs>

简介：

类比Django来介绍：

如果在 Django 中写一个页面，你需要：

1. 在 urls.py 中注册 view；
2. 在 views.py 中编写 view；
3. 在 templates 文件夹中编写模板。

相对地，如果使用 DRF 创建一组 API，你需要：

1. 在 urls.py 中定义并注册 router；
2. 在 views.py 中定义 ViewSet；
3. 在 serializers.py 中定义 serializer.

这里，urls.py负责路由（说明访问某个url对应调用哪个函数），views.py负责后端逻辑处理（查询数据库、处理数据等操作都在这里），model.py负责定义数据库中的数据类型，serializer.py负责转换数据库中的Model Data（数据库中数据存储的类型）和JSON类型的数据。

**实现一组api的基本流程**

1. 在**models.py**中创建自己需要的数据模型
2. 创建**serializers.py**在其中写出适合的serializer类，能够正确的序列化与反序列化
3. 在**views.py**中写出合适的api类，只需要继承**rest\_framework**中**generics**中的某个类，重写我们需要的方法实现合适的逻辑即可
4. 在**urls.py**中配置所需要的url

欢迎补充！

下面介绍一些有关REST架构的基本概念。

REST(Representational State Tranfer)架构由Roy Fielding于2000年首次提出，它是一个web服务中前后端交互的约定方案。REST由一组架构约束条件组成，并能够根据实际情况的需要自行调整。如果一个架构满足REST的一些原则，则该架构可以被称为是RESTful的。

**核心原则**

下面介绍一些REST架构的核心原则。

**以资源为中心**

在REST架构中，服务器维护的是资源。任何可能被客户端请求获取的事物都被服务器视为资源。因此，为了精确标识每一个资源的位置，需要使用URI(Uniform Resource Identifier)。

注意这里以URI标识的是资源而非调用资源的方法。很多时候开发者会使用动作相关的词语描述URI，例如GET /getUser/1 等等。这些是违反REST“以资源为中心”的原则的。

**资源的链接方式**

REST架构鼓励通过在资源内部加入链接，从而引导客户端执行下一步的操作。因此，客户端不需要记录资源之间的连接状态，而是交给服务器完成。客户端的任务只有记录每个资源的URL，通过查找资源内部链接进行进一步的调用。

**接口安全性与幂等性**

REST架构强调统一接口，即使用相同的接口访问不同的资源。在REST架构中，接口被限定为标准的HTTP方法，引入了“安全性”和“幂等性”两个概念衡量接口的特性。

**安全性**

安全性指的是调用接口是否会改变服务端资源状态。

**幂等性**

幂等性指的是多次重复调用某接口产生的总效果是否与仅调用一次该接口产生的效果相同。如果相同则表明该接口是幂等的。

|  |  |  |
| --- | --- | --- |
| 接口 | 安全性 | 幂等性 |
| GET | 是 | 是 |
| POST | 否 | 否 |
| PUT | 否 | 是 |
| DELETE | 否 | 是 |
| HEAD | 是 | 是 |
| PATCH | 否 | 否 |

**无状态通信**

状态可以分为应用状态和资源状态。客户端负责维护应用状态，服务端负责维护资源状态。

无状态通信原则指的是服务端在处理客户端的请求时，不需要记录与该客户端之前的会话状态，一切信息都包含在每次请求中。换言之，对服务端而言，每次请求都是独立的。这些会话状态在必要情况下作为应用状态的一部分，由应用端存储并跟踪，再根据服务端存储的资源关系进行后续操作。

**其他特性与细节**

**合理设计URL**

尽量使用可阅读的词标记服务端的资源，而不是无意义的字符串。如果资源过于复杂，可以使用句子描述，并用-或者\_作为分隔符。

同时，由于URL中大量使用名词，因此应提前约定名词单复数形式，最好都用单数或者都用复数，避免交互错误。

**利用状态码**

HTTP状态码指的是应用端向服务端发起请求后，服务端向用户返回的提示信息，由3位数字组成。状态码的数量非常多，具体情况可参照[常见的HTTP状态码](https://www.cnblogs.com/xflonga/p/9368993.html)。

|  |  |  |
| --- | --- | --- |
| 状态码 | 对应动词 | 语义 |
| 200 | OK | 一切正常 |
| 301 | Moved Permanently | 资源URL已更新 |
| 400 | Bad Request | 请求错误 |
| 404 | Not Found | 资源不存在 |
| 409 | Conflict | 通用冲突 |
| 500 | Internal Server Error | 通用错误响应 |

**参考链接**

1.         [RESTful架构详解](https://www.runoob.com/w3cnote/restful-architecture.html)

2.         [RESTful API 设计指南](http://www.ruanyifeng.com/blog/2014/05/restful_api.html)

3.         [聊聊RESTfu-接口设计篇（二）](https://howardwchen.com/2017/09/25/talk-about-restful-popular-api-design-2/?from=csdn)