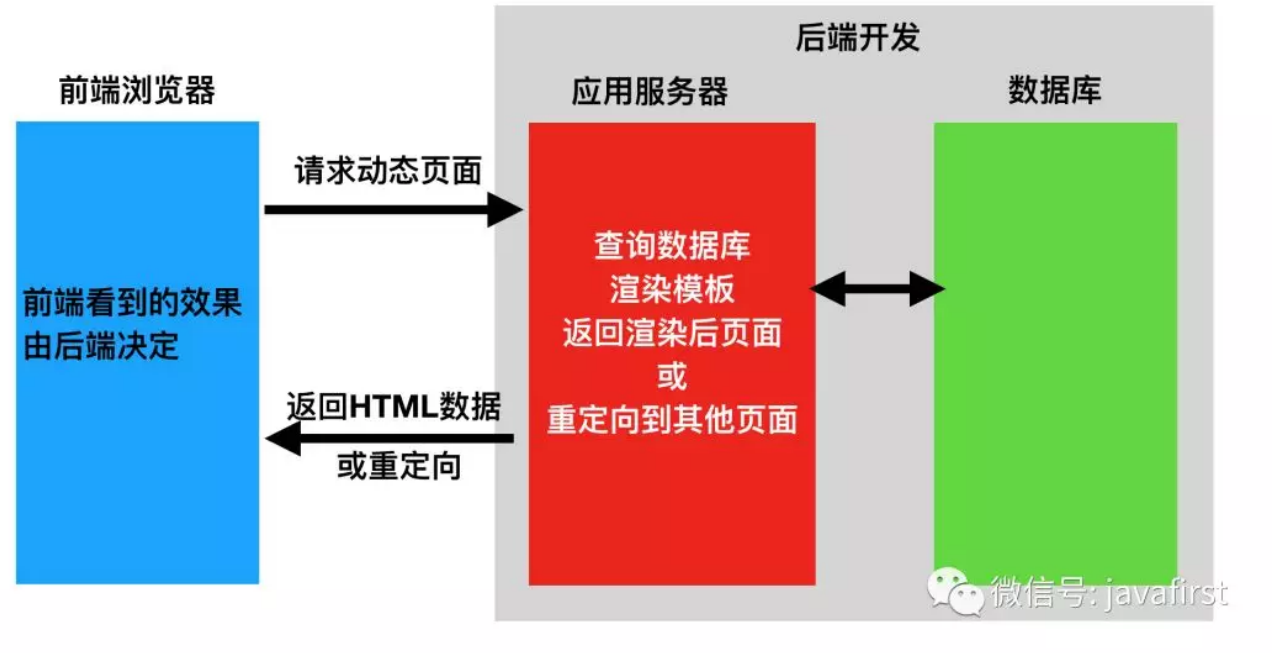
当前的Web开发领域，我们经常会听到前后端分离这个技术名词，顾名思义，就是前端页面的开发与后端服务器的开发分离开。这个技术方案的实现要借助API服务模式，API简单说就是开发人员提供编程接口被其他人调用，调用之后后端服务器会返回数据供调用者使用，或者接受调用者发来的数据，修改后端的状态。

让我们来详细了解一下：

前后端不分离

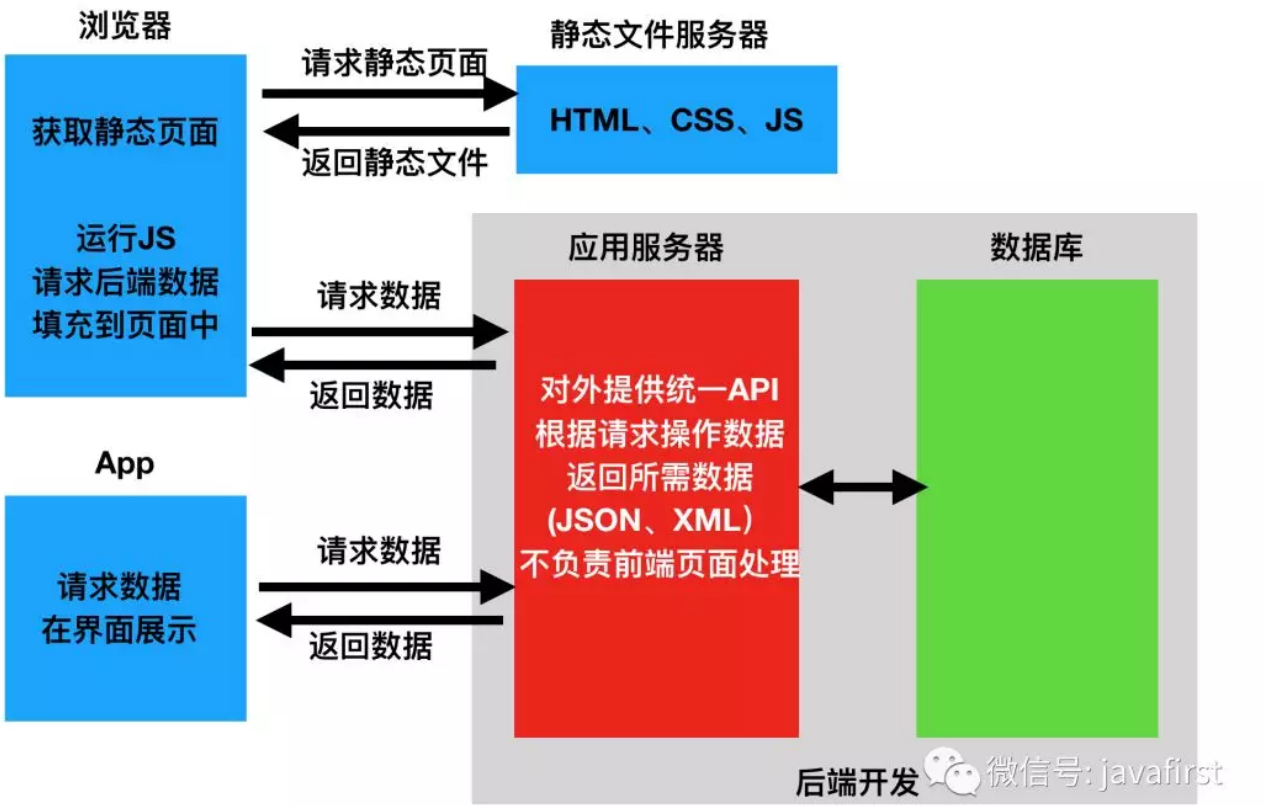
在前后端不分离的应用模式中，前端页面看到的效果都是由后端控制，由后端渲染页面或重定向，也就是后端需要控制前端的展示，前端与后端的耦合度很高。

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适用于前端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,我们通常将后端开发的每一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。

API的风格有多种，但是现在比较主流且实用的就是本文要说的RESTful API。

RESTful

RESTful是一种软件架构风格、设计风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

REST全称是Representational State Transfer，中文意思是 表征状态转移 。 它首次出现在2000年Roy Fielding的博士论文中，Roy Fielding是HTTP规范的主要编写者之一。 他在论文中提到："我这篇文章的写作目的，就是想在符合架构原理的前提下，理解和评估以网络为基础的应用软件的架构设计，得到一个功能强、性能好、适宜通信的架构。REST指的是一组架构约束条件和原则。" 如果一个架构符合REST的约束条件和原则，我们就称它为RESTful架构。

要好好地设计我们的url，不要乱搞!

REST本身并没有创造新的技术、组件或服务，而隐藏在RESTful背后的理念就是使用Web的现有特征和能力， 更好地使用现有Web标准中的一些准则和约束。虽然REST本身受Web技术的影响很深， 但是理论上REST架构风格并不是绑定在HTTP上，只不过目前HTTP是唯一与REST相关的实例。 所以我们这里描述的REST也是通过HTTP实现的REST。

RESTful的核心操作：URL定位资源，用HTTP动词（GET,POST,DELETE,DETC）描述操作。

那这种风格的接口有什么好处呢？可以前后端分离。前端拿到数据只负责展示和渲染，不对数据做任何处理。后端处理数据并以JSON格式传输出去，定义这样一套统一的接口，在web，ios，android三端都可以用相同的接口。

关于RESTful的技术内涵，阅读下面三篇博文即可：

https://www.runoob.com/w3cnote/restful-architecture.html

https://blog.csdn.net/qq_21383435/article/details/80032375

http://www.ruanyifeng.com/blog/2018/10/restful-api-best-practices.html

关于RESTful的核心，其实就是如何设计URL！！！

RESTful实践

- API与用户的通信协议，尽量使用HTTPs协议。
- 域名
  - **https://api.example.com** 最好不要用这种（会存在跨域问题）
  - **https://example.org/api/** API很简单
- 版本
  - URL，如： **https://api.example.com/v1/**
  - 包含在请求头中
- url，任何东西都是资源，均使用名词表示（可复数）
  - **https://api.example.com/v1/zoos**
  - **https://api.example.com/v1/animals**
  - **https://api.example.com/v1/employees**
- method
  - GET： 从服务器取出资源（一项或多项）
  - POST： 在服务器新建一个资源
  - PUT： 在服务器更新资源（客户端提供改变后的完整资源），完整更新

    - PATCH： 在服务器更新资源（客户端提供改变的属性），局部更新，可能不支持
    - DELETE： 从服务器删除资源
- 过滤，通过在url上传参的形式传递搜索条件
    - https://api.example.com/v1/zoos?limit=10： 指定返回记录的数量
    - https://api.example.com/v1/zoos?offset=10： 指定返回记录的开始位置
    - https://api.example.com/v1/zoos?page=2&per_page=100： 指定第几页，以及每页的记录数
    - https://api.example.com/v1/zoos?sortby=name&order=asc： 指定返回结果按照哪个属性排序，以及排序顺序
    - https://api.example.com/v1/zoos?animal_type_id=1： 指定筛选条件
- 状态码 + code信息

```
200 OK - [GET]：服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）。
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，该操作是幂等的。
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。
```

- 错误处理，状态码是4xx时，应返回错误信息。

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

- 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
```

- 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。

```
{"link": {
  "rel":   "collection https://www.example.com/zoos",
  "href":  "https://api.example.com/zoos",
  "title": "List of zoos",
  "type":  "application/vnd.yourformat+json"
}}
```

HTTP请求方法详解

请求方法：指定了客户端想对指定的资源/服务器作何种操作

根据HTTP标准，HTTP请求可以使用多种请求方法。

HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头
3	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。

GET：获取资源

GET方法用来请求已被URI识别的资源。指定的资源经服务器端解析后返回响应内容（也就是说，如果请求的资源是文本，那就保持原样返回；如果是CGI[通用网关接口]那样的程序，则返回经过执行后的输出结果）。 最常用于向服务器查询某些信息。必要时，可以将查询字符串参数追加到URL末尾，以便将信息发送给服务器。

POST：传输实体文本

POST方法用来传输实体的主体。 虽然用GET方法也可以传输实体的主体，但一般不用GET方法进行传输，而是用POST方法；虽然GET方法和POST方法很相似，但是POST的主要目的并不是获取响应的主体内容。 POST请求的主体可以包含非常多的数据，而且格式不限。

GET方法和POST方法本质上的区别：

1、GET方法用于信息获取，它是安全的（安全：指非修改信息，如数据库方面的信息），而POST方法是用于修改服务器上资源的请求；

2、GET请求的数据会附在URL之后，而POST方法提交的数据则放置在HTTP报文实体的主体里，所以POST方法的安全性比GET方法要高；

3、GET方法传输的数据量一般限制在2KB，其原因在于：GET是通过URL提交数据，而URL本身对于数据没有限制，但是不同的浏览器对于URL是有限制的，比如IE浏览器对于URL的限制为2KB，而Chrome，FireFox浏览器理论上对于URL是没有限制的，它真正的限制取决于操作系统本身；POST方法对于数据大小是无限制的，真正影响到数据大小的是服务器处理程序的能力。

HEAD：获得报文首部

HEAD方法和GET方法一样，只是不返回报文的主体部分，用于确认URI的有效性及资源更新的日期时间等。 具体来说：1、判断类型； 2、查看响应中的状态码，看对象是否存在（响应：请求执行成功了，但无数据返回）； 3、测试资源是否被修改过 HEAD方法和GET方法的区别：GET方法有实体，HEAD方法无实体。

PUT：传输文件

PUT方法用来传输文件，就像FTP协议的文件上传一样，要求在请求报文的主体中包含文件内容，然后保存在请求URI指定的位置。但是HTTP/1.1的PUT方法自身不带验证机制，任何人都可以上传文件，存在安全问题，故一般不用。

POST和PUT的区别

POST 方法用来传输实体的主体，PUT方法用来传输文件，自身不带验证机制。

这两个方法看起来都是讲一个资源附加到服务器端的请求，但其实是不一样的。一些狭窄的意见认为，POST方法用来创建资源，而PUT方法则用来更新资源。这个说法本身没有问题，但是并没有从根本上解释了二者的区别。事实上，它们最根本的区别就是：POST方法不是幂等的，而PUT方法则有幂等性。那这又衍生出一个问题，什么是幂等?

幂等（idempotent、idempotence）是一个抽象代数的概念。在计算机中，可以这么理解，一个幂等操作的特点就是其任意多次执行所产生的影响均与依次一次执行的影响相同。

POST在请求的时候，服务器会每次都创建一个文件，但是在PUT方法的时候只是简单地更新，而不是去重新创建。因此PUT是幂等的。

DELETE：删除资源

指明客户端想让服务器删除某个资源，与PUT方法相反，按URI删除指定资源

OPTIONS：询问支持的方法

OPTIONS方法用来查询针对请求URI指定资源支持的方法（客户端询问服务器可以提交哪些请求方法）

TRACE：追踪路径

客户端可以对请求消息的传输路径进行追踪，TRACE方法是让Web服务器端将之前的请求通信还给客户端的方法

CONNECT：要求用隧道协议连接代理

CONNECT方法要求在与代理服务器通信时建立隧道，实现用隧道协议进行TCP通信。主要使用SSL（安全套接层）和TLS（传输层安全）协议把通信内容加密后经网络隧道传输。