

1，前后端分离

1.1 什么是前后端分离

前端：即客户端，负责渲染用户显示界面【如web的js动态渲染页面, 安卓，IOS，pc客户端等】

后端：即服务器端，负责接收http请求，处理数据

API: Application Programming Interface 是一些预先定义的函数，或指软件系统不同组成部分衔接的约定

前后端分离 完整请求过程

- 1，前端通过http请求后端API
- 2，后端以json形式返回前端数据
- 3，前端生成用户显示界面【如html , ios , android】

判断前后端分离得核心标准： 谁生成显示页面

1，后端生成【前后端未分离】 ex: flask->render_template django -> HttpResponse(html)

2，前端生成【前后端分离】

1.2 优点

- 1，各司其职

前端：视觉层面，兼容性，前端性能优化

后端：并发，可用性，性能

2，解耦，前端和后端均易于扩展

3，后端灵活搭配各类前端 - 如安卓等

4，提高用户体验

5，前端+后端可完全并行开发，加快开发效率

1.3 分离常见问题

| 问题 | 答案 |
|---|--|
| 如何解决http无状态? | 采用token(详情见下方章节) |
| 如果前端为JS, 如何解决跨域问题? | 采用CORS(详情见下方章节) |
| 如何解决csrf问题 | 采用token |
| Single Page web Application 是否会影响Search Engine Optimization效果 | 会, 前后端分离后, 往往页面不存在静态文字【例如新闻的详细内容】 |
| "老板, 这个逻辑到底是让前端做还是后端做啊?" | 底线原则: 数据校验需要前后端都做 |
| "老板, 前端工作压力太大了啊" | 团队协作不能只是嘴上说说 |
| 动静分离和前后端分离是一个意思么? | 动静分离指的是 css/js/img这类静态资源跟服务器拆开部署, 典型方案-静态资源交由CDN厂商处理 |

1.4 实现方式

1, Django/Flask 后端只返回json

2, 前端 -> ex: js向服务器发出ajax请求, 获取数据, 拿到数据后动态生成html

2, token - 令牌

学前须知:

1, base64 '防君子不防小人'

| 方法 | 作用 | 参数 | 返回值 |
|-------------------|---|---|--|
| b64encode | 将输入的参 数转化为 base64规则 的串 | 预加密的明文, 类型为 bytes; 例: b'guoxiaonao' | base64对应编码的密文, 类型为bytes; 例:b'Z3VveGlhb25hbW==' |
| b64decode | 将base64串 解密回 明文 | base64密文,类型为 bytes;例: b'Z3VveGlhb25hbW==' | 参数对应的明文, 类型为 bytes; 例: b'guoxiaonao' |
| urlsafe_b64encode | 作用同 b64encode, 但是会将 '+'替换成 '-', 将 '/' 替换 成 '_' | 同b64encode | 同b64encode |
| urlsafe_b64decode | 作用同 b64decode | 同b64decode | 同b64decode |

代码演示:

```
import base64
#base64加密
s = b'guoxiaonao'
b_s = base64.b64encode(s)
#b_s打印结果为 b'Z3VveG1hb25hbW=='

#base64解密
ss = base64.b64decode(b_s)
#ss打印结果为 b'guoxiaonao'
```

2, SHA-256 安全散列算法的一种 (hash)

hash三大特点:

1) 定长输出 2) 不可逆 3) 雪崩

```
import hashlib
s = hashlib.sha256() #创建sha256对象
s.update(b'xxxx') #添加欲hash的内容, 类型为 bytes
s.digest() #获取最终结果
```

3, HMAC-SHA256 是一种通过特别计算方式之后产生的消息认证码, 使用**散列算法**同时结合一个**加密密钥**。它可以用来保证数据的完整性, 同时可以用来作某个消息的身份验证

```
import hmac
#生成hmac对象
#第一个参数为加密的key， bytes类型，
#第二个参数为欲加密的串， bytes类型
#第三个参数为hmac的算法， 指定为SHA256
h = hmac.new(key, str, digestmod='SHA256 ')
h.digest() #获取最终结果
```

4, RSA256 非对称加密

1, 加密： 公钥加密， 私钥解密

2, 签名： 私钥签名， 公钥验签

2.1 JWT - json-web-token

1, 三大组成

1, header

格式为字典-元数据格式如下

```
{'alg': 'HS256', 'typ': 'JWT'}
#alg代表要使用的 算法
#typ表明该token的类别 - 此处必须为 大写的 JWT
```

该部分数据需要转成json串并用base64 加密

2, payload

格式为字典-此部分分为公有声明和私有声明

公共声明： JWT提供了内置关键字用于描述常见的问题

此部分均为**可选项**，用户根据自己需求 按需添加key，常见公共声明如下：

```
{'exp':xxx, # Expiration Time 此token的过期时间的  
时间戳  
'iss':xxx, # (Issuer) Claim 指明此token的签发者  
'aud':xxx, #(Audience) Claim 指明此token的  
'iat':xxx, # (Issued At) Claim 指明此创建时间的时  
间戳  
'aud':xxx, # (Audience) Claim 指明此token签发面  
向群体  
}
```

私有声明：用户可根据自己业务需求，添加自定义的key，例如如下：

```
{'username': 'guoxiaonao'}
```

公共声明和私有声明均在同一个字典中；转成json串并用base64加密

3, signature 签名

签名规则如下：

根据header中的alg确定 具体算法，以下用 HS256为例

HS256(自定义的key, base64后的header + '.' +
base64后的payload)

解释：用自定义的key, 对base64后的header + '.' +
base64后的payload进行hmac计算

2, jwt结果格式

base64(header) + '.' + base64(payload) + '.' +
base64(sign)

最终结果如下：

b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1b3hpYW9uYW8iLCJpc3MiOiJnZ2cifQ.Zzg1u55DCBqPRGf9z3-NAn4kbA-MJN83SxyLFfc5mmM'

3， 校验jwt规则

1， 解析header, 确认alg

2， 签名校验 - 根据传过来的header和payload按 alg指明的算法进行签名， 将签名结果和传过来的sign进行对比， 若对比一致， 则校验通过

3， 获取payload自定义内容

4， pyjwt

1， 安装 pip3 install pyjwt

| 方法 | 参数说明 | 返回值 |
|---------------------------------|---|-------------------------|
| encode(payload, key, algorithm) | payload: jwt 三大组成中的 payload,需要 组成字典, 按 需添加公有声 明和私有声明 例如: {'username': 'guoxiaonao', 'exp': 1562475112} 参数类型: dict | token串 返回类型: bytes |
| | key: 自定义 的加密key 参数类型: str | |
| | algorithm: 需 要使用的加密 算法[HS256, RSA256等] 参数类型: str | |
| decode(token,key,algorithm,) | token: token 串 参数类型: bytes/str | payload明文 返回类型: dict |

| 方法 | 参数说明 | 返回值 |
|----|---|---|
| | key: 自定义的加密key,需要跟encode中的key保持一致 参数类型: str | |
| | algorithm: 同encode | |
| | issuer: 发布者, 若encode payload中添加 'iss' 字段, 则可针对该字段校验 参数类型: str | 若iss校验失败, 则抛出 jwt.InvalidIssuerError |
| | audience: 签发的受众群体, 若encode payload中添加 'aud' 字段, 则可针对该字段校验 参数类型: str | 若aud校验失败, 则抛出 jwt.InvalidAudienceError |

PS: 若encode得时候 payload中添加了exp字段; 则exp字段得值需为 当前时间戳+此token得有效期时间, 例如希望 token 300秒后过期 `{'exp': time.time() + 300}`; 在执行 decode时, 若检查到exp字段, 且token过期, 则抛出 `jwt.ExpiredSignatureError`

3, CORS - Cross-origin resource sharing - 跨域资源共享

1, 什么是CORS

允许浏览器向跨源(协议 + 域名 + 端口)服务器, 发出 XMLHttpRequest请求, 从而克服了AJAX只能同源使用的限制

2, 特点

1, 浏览器自动完成 (在请求头中加入特殊头 或 发送特殊请求)

2, 服务器需要支持 (响应头中需要有特殊头)

3, 简单请求(Simple requests)和预检请求(Preflighted requests)

满足以下全部条件的请求为 简单请求

1, 请求方法如下:

GET or HEAD or POST

2, 请求头仅包含如下:

Accept

Accept-Language

Content-Language

Content-Type

3, Content-Type 仅支持如下三种:

application/x-www-form-urlencoded

multipart/form-data

text/plain

不满足以上任意一点的请求都是 预检请求

4, 简单请求发送流程

1, 请求

请求头中 携带 Origin, 该字段表明自己来自哪个域

2, 响应

如果请求头中的Origin在服务器接受范围内, 则返回如下头

| 响应头 | 作用 | 备注 |
|----------------------------------|---|----|
| Access-Control-Allow-Origin | 服务器接受得域 | |
| Access-Control-Allow-Credentials | 是否接受Cooike | 可选 |
| Access-Control-Expose-Headers | 默认情况下，xhr只能拿到如下响应头：Cache-Control, Content-Language, Content-Type, Expires, Last-Modified；如果有需要获取其他头，需在此指定 | 可选 |

如果服务器不接受此域，则响应头中不包含 Access-Control-Allow-Origin

5，预检请求发送流程

1，OPTION 请求发起，携带如下请求头

| 请求头 | 作用 | 备注 |
|--------------------------------|------------|----|
| Origin | 表明此请求来自哪个域 | 必选 |
| Access-Control-Request-Method | 此次请求使用方法 | 必选 |
| Access-Control-Request-Headers | 此次请求使用的头 | 必选 |

2, OPTION 接受响应阶段, 携带如下响应头

| 响应头 | 作用 | 备注 |
|----------------------------------|---|----|
| Access-Control-Allow-Origin | 同简单请求 | 必选 |
| Access-Control-Allow-Methods | 告诉浏览器, 服务器接受得跨域请求方法 | 必选 |
| Access-Control-Allow-Headers | 返回所有支持的头部, 当request有'Access-Control-Request-Headers'时, 该响应头必然回复 | 必选 |
| Access-Control-Allow-Credentials | 同简单请求 | 可选 |
| Access-Control-Max-Age | OPTION请求缓存时间, 单位s | 可选 |

3, 主请求阶段

| 请求头 | 作用 | 备注 |
|--------|------------|----|
| Origin | 表明此请求来自哪个域 | |

4, 主请求响应阶段

| 响应头 | 作用 | 备注 |
|-----------------------------|-----------|----|
| Access-Control-Allow-Origin | 当前服务器接受得域 | |

6, Django支持

django-cors-headers官网 <https://pypi.org/project/django-cors-headers/>

直接pip 将把django升级到2.0以上, 强烈建议用离线安装方式

配置流程

```
1, INSTALLED_APPS 中添加 corsheaders
2, MIDDLEWARE 中添加
corsheaders.middleware.CorsMiddleware
    位置尽量靠前, 官方建议
'django.middleware.common.CommonMiddleware' 上方
3, CORS_ORIGIN_ALLOW_ALL 布尔值 如果为
True 白名单不启用
```

```
4, CORS_ORIGIN_WHITELIST = [
    "https://example.com"
]
5, CORS_ALLOW_METHODS = (
    'DELETE',
    'GET',
    'OPTIONS',
    'PATCH',
    'POST',
    'PUT',
)
6, CORS_ALLOW_HEADERS = (
    'accept-encoding',
    'authorization',
    'content-type',
    'dnt',
    'origin',
    'user-agent',
    'x-csrftoken',
    'x-requested-with',
)
7, CORS_PREFLIGHT_MAX_AGE 默认 86400s
8, CORS_EXPOSE_HEADERS []
9, CORS_ALLOW_CREDENTIALS 布尔值, 默认
False
```

4, RESTful - Representational State Transfer

4.1, 什么是RESTful

1, 资源 (Resources)

网络上的一个实体, 或者说是网络上的一个具体信息, 并且每个资源都有一个独一无二得URI与之对应; 获取资源-直接访问URI即可

2, 表现层 (Representation)

如何去表现资源 - 即资源得表现形式; 如HTML, xml, JPG, json等

3, 状态转化 (State Transfer)

访问一个URI即发生了一次 客户端和服务端得交互; 此次交互将会涉及到数据和状态得变化

客户端需要通过某些方式触发具体得变化 - HTTP method 如 GET, POST, PUT, PATCH, DELETE 等

4.2 RESTful的特征

1, 每一个URI代表一种资源

2, 客户端和服务端之前传递着资源的某种表现

3, 客户端通过HTTP的几个动作 对 资源进行操作 - 发生'状态转化'

4.3 如何设计符合RESTful 特征的API

1, 协议 - http/https

2, 域名:

域名中体现出api字样, 如

<https://api.example.com>

or

<https://example.org/api/>

3, 版本:

<https://api.example.com/v1/>

4, 路径 -

路径中避免使用动词, 资源用名词表示, 案例如下

```
https://api.example.com/v1/users  
https://api.example.com/v1/animals
```

5, HTTP动词语义

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

- DELETE (DELETE) : 从服务器删除资源。

具体案例如下:

GET /zoos: 列出所有动物园
POST /zoos: 新建一个动物园
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

6, 巧用查询字符串

?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。
?type_id=1: 指定筛选条件

7, 状态码

1, 用HTTP响应码表达 此次请求结果, 例如

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，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

2, 自定义内部code 进行响应

如 返回结构如下 {'code':200, 'data': {}, 'error': xxx}

8, 返回结果

根据HTTP 动作的不同，返回结果的结构也有所不同

GET /users: 返回资源对象的列表（数组）

GET /users/guoxiaonao: 返回单个资源对象

POST /users: 返回新生成的资源对象

PUT /users/guoxiaonao: 返回完整的资源对象

PATCH /users/guoxiaonao: 返回完整的资源对象

DELETE /users/guoxiaonao: 返回一个空文档