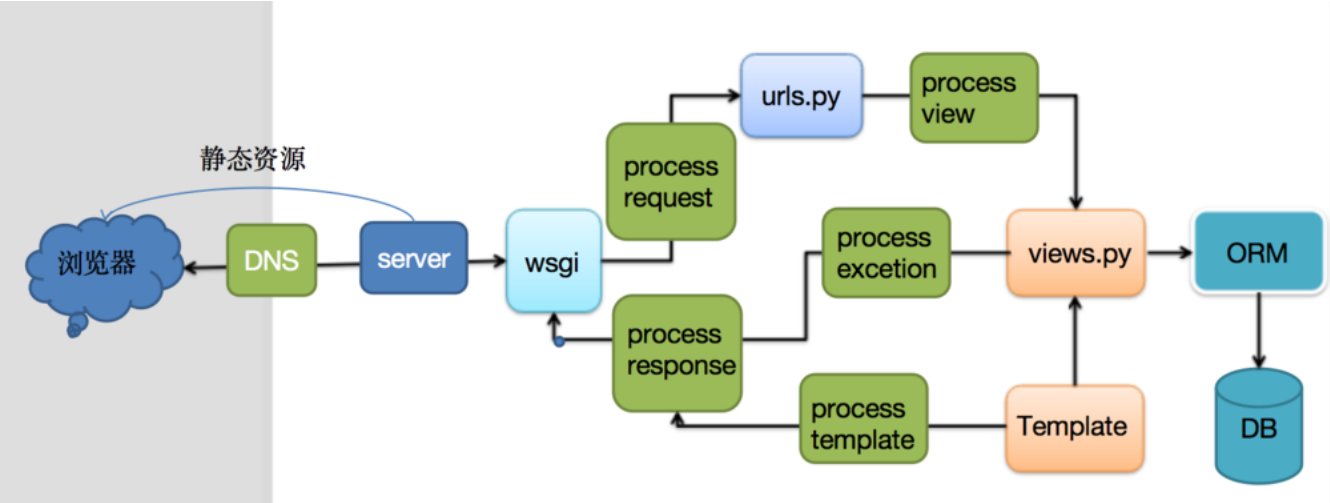
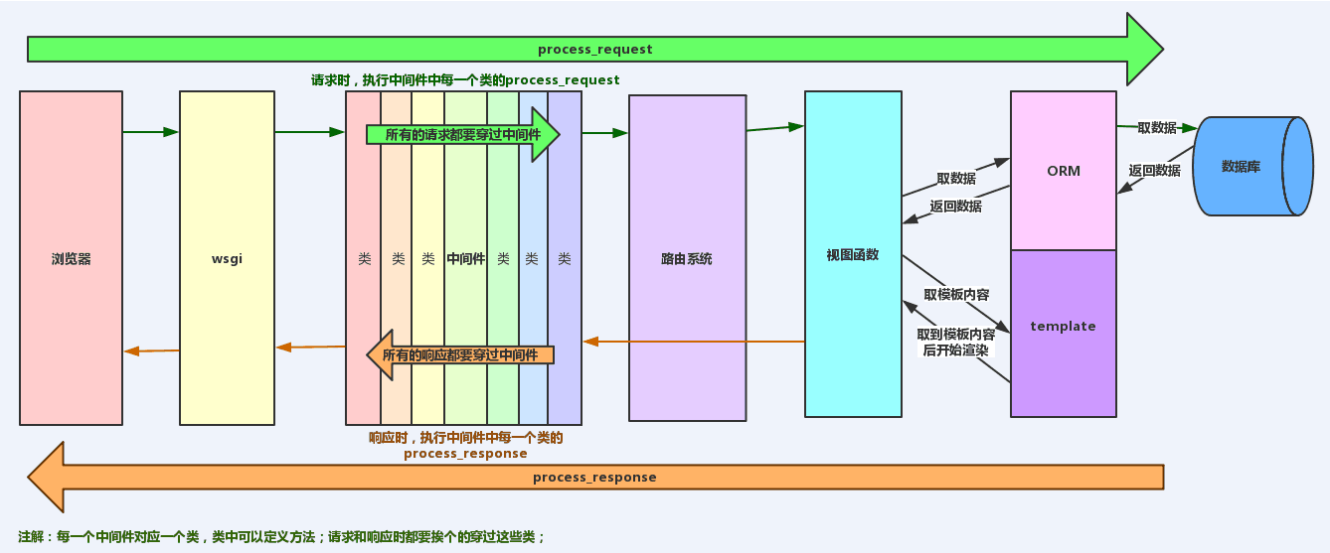
问题7：Django 如何实现 http 请求和发送？

Django 生命周期

1. 当用户在浏览器中输入url时,浏览器会生成请求头和请求体发给服务端请求头和请求体中会包含浏览器的动作(action),这个动作通常为get或者post,体现在url之中.
2. url经过Django中的wsgi,再经过Django的中间件,最后url到过路由映射表,在路由中一条一条进行匹配,一旦其中一条匹配成功就执行对应的视图函数,后面的路由就不再继续匹配了.
3. 视图函数根据客户端的请求查询相应的数据.返回给Django,然后Django把客户端想要的数据做为一个字符串返回给客户端.
4. 客户端浏览器接收到返回的数据,经过渲染后显示给用户.

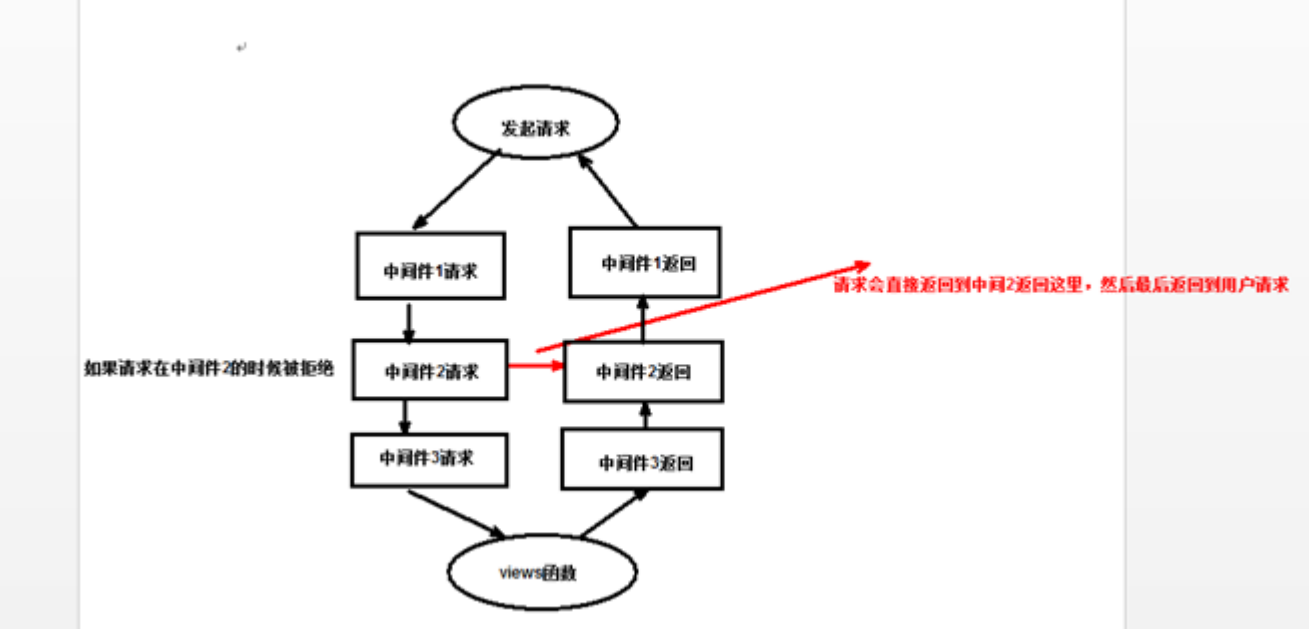




中间件执行顺序

总结：中间件的本质其实就是个装饰器，装饰器的本质其实就是个闭包

- 在请求视图被处理前，中间件由上至下依次执行
- 在请求视图被处理后，中间件由下至上依次执行



VUE的生命周期

生命周期解析：

https://juejin.im/post/5afd7eb16fb9a07ac5605bb3

https://segmentfault.com/a/1190000011381906?utm_source=tag-newest

http://www.php.cn/js-tutorial-412302.html

cookie、session、token

cookie 和 session： https://www.cnblogs.com/ITCodeMonkey/p/7874343.html

cookie、session、token 参考：

https://my.oschina.net/u/3908739/blog/1941216

https://www.liangzl.com/get-article-detail-16019.html

session相关： https://www.cnblogs.com/zhuifeng-mayi/p/9099811.html

1、cookie和session的区别

session是存储服务器端，cookie是存储在客户端，所以session的安全性比cookie高。

获取session里的信息是通过存放在会话cookie里的session id获取的。而session是存放在服务器的内存中里，所以session里的数据不断增加会造成服务器的负担，所以会把很重要的信息存储在session中，而把一些次要东西存储在客户端的cookie里。

cookie确切的说分为两大类：会话cookie和持久化cookie。

会话cookie是存放在客户端浏览器的内存中，他的生命周期和浏览器是一致的，当浏览器关闭会话cookie也就消失了

持久化cookie是存放在客户端硬盘中，持久化cookie的生命周期是我们在设置cookie时候设置的那个保存时间，session的信息是通过sessionid获取的，而sessionid是存放在会话cookie当中的，当浏览器关闭的时候会话cookie消失，所以sessionid也就消失了，但是session的信息还存在服务器端，只是查不到所谓的session但它并不是不存在。所以session在服务器关闭的时候，或者是sessio过期，又或者调用了invalidate()，再或者是session中的某一条数据消失调用session.removeAttribute()方法，session在通过调用session.getsession来创建的。

2、会话控制

所谓的会话,就是用户和浏览器中网站之间一次交互过程.

会话的开始是在用户打开浏览器以后第一次访问网站.

会话的结束时在用户关闭浏览器以后.

因为 http 是一种无状态协议，浏览器请求服务器是无状态的。

无状态：指一次用户请求时，浏览器、服务器无法知道之前这个用户做过什么，每次请求都是一次新的请求。

无状态原因：浏览器与服务器是使用 socket 套接字进行通信的，服务器将请求结果返回给浏览器之后，会关闭当前的 socket 连接，而且服务器也会在处理页面完毕之后销毁页面对象。

有时需要保持下来用户浏览的状态，比如用户是否登录过，浏览过哪些商品等

实现状态保持主要有两种方式：

- 在客户端存储信息使用 `Cookie,本地存储, token[jwt,oauth]`
- 在服务器端存储信息使用 `Session` ,redis

Cookie

Cookie是由服务器端生成，发送给客户端浏览器，浏览器会将Cookie的key/value保存，下次请求同一网站时就发送该Cookie给服务器（前提是浏览器设置为启用cookie）。Cookie的key/value可以由服务器端自己定义。

使用场景: 登录状态, 浏览历史, 网站足迹

Cookie是存储在浏览器中的一段纯文本信息，建议不要存储敏感信息如密码，因为电脑上的浏览器可能被其它人使用

Cookie基于域名安全，不同域名的Cookie是不能互相访问的

如访问luffy.com时向浏览器中写了Cookie信息，使用同一浏览器访问baidu.com时，无法访问到luffy.com写的Cookie信息

浏览器的同源策略针对cookie也有限制作用.

当浏览器请求某网站时，会将本网站下所有Cookie信息提交给服务器，所以在request中可以读取Cookie信息

（1）flask 中 设置和获取 Cookie

1. 设置 cookie

设置cookie需要通过flask的Response响应对象来进行设置,由flask内部提供了一个make_response函数给我们可以快速创建响应对象

```
from flask imoprt Flask,make_response
@app.route('/set_cookie')
def set_cookie():
    resp = make_response('this is to set cookie')
    resp.set_cookie('username', 'xiaoming', max_age=3600)
    return resp
```

2. 获取cookie

```
from flask import Flask,request
@app.route('/get_cookie')
def resp_cookie():
    resp = request.cookies.get('username')
    return resp
```

（2）flask 中 设置和获取 Session

对于敏感、重要的信息，建议要存储在服务器端，不能存储在浏览器中，如用户名、余额、等级、验证码等信息

在服务器端进行状态保持的方案就是 `Session`

注意: Session依赖于Cookie,而且flask中使用session,需要配置SECRET_KEY选项,否则报错.

1. 设置session

```
@app.route('/set_session')
def set_session():
    session['username'] = 'xiaoming'
    return 'ok!'
```

2. 获取session

```
@app.route('/get_session')
def get_session():
    return session.get('username')
```

（3）Django中设置与获取 Session

1、设置Sessions值

```
request.session['username'] = "yuan"
```

- 生成随机字符串
- session_key就是存随机字符串，转换成字典，存到django_session表里
- 在cookie中设置sessionid=session_key

2、获取Sessions值

```
request.session.get('is_login')
```

- 获取cookie获取sessionid
- 拿到这个id去django-session表中过滤对当前对象
- 获取值。

3、删除Sessions值

```
del request.session["session_name"]
```

4、flush()

- 删除当前的会话数据并删除会话的Cookie。

- 这用于确保前面的会话数据不可以再次被用户的浏览器访问

3、基于Token的验证原理

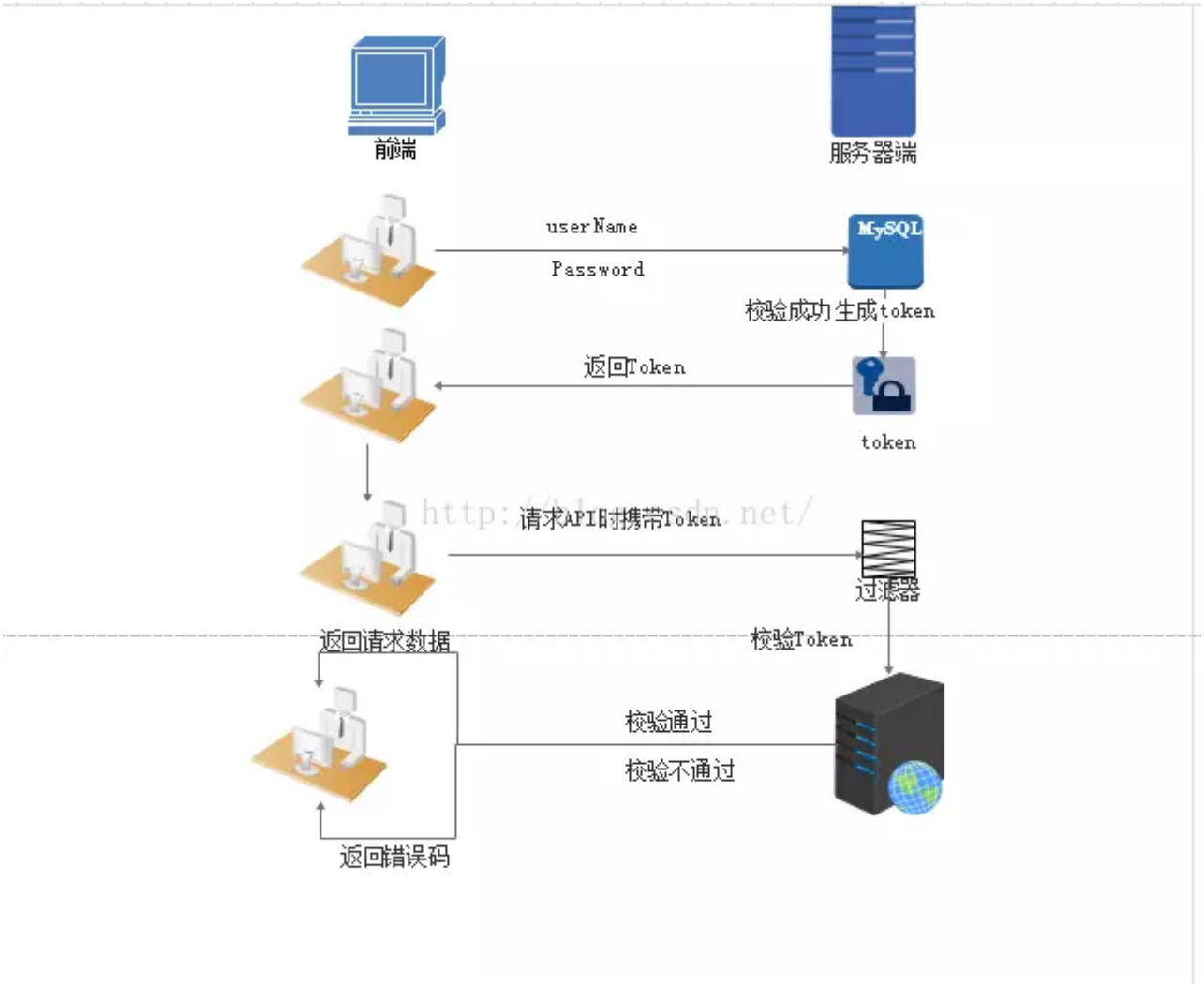
基于Token的身份验证是无状态的，我们不将用户信息存在服务器或Session中。

这种概念解决了在服务端存储信息时的许多问题

NoSession意味着你的程序可以根据需要去增减机器，而不用去担心用户是否登录。

基于Token的身份验证的过程如下:

1. 用户通过用户名和密码发送请求。
2. 程序验证。
3. 程序返回一个签名的token 给客户端。
4. 客户端储存token,并且每次用于每次发送请求。
5. 服务端验证token并返回数据。



Token检验过程：

1. 用户登录校验，校验成功后就返回Token给客户端。
2. 客户端收到数据后保存在客户端
3. 客户端每次访问API是携带Token到服务器端。

4. 服务器端采用filter过滤器校验。校验成功则返回请求数据，校验失败则返回错误码