

# URL 介绍

## URL 定义

- URL 即统一资源定位符 Uniform Resource Locator
- 作用:
  - 用来表示互联网上某个资源的地址。
- 说明:
  - 互联网上的每个文件都有一个唯一的URL，它包含的信息指出文件的位置以及浏览器应该怎么处理它。
- URL的一般语法格式为：

```
1 | protocol :// hostname[:port] / path [?query][#fragment]
```

- 如:

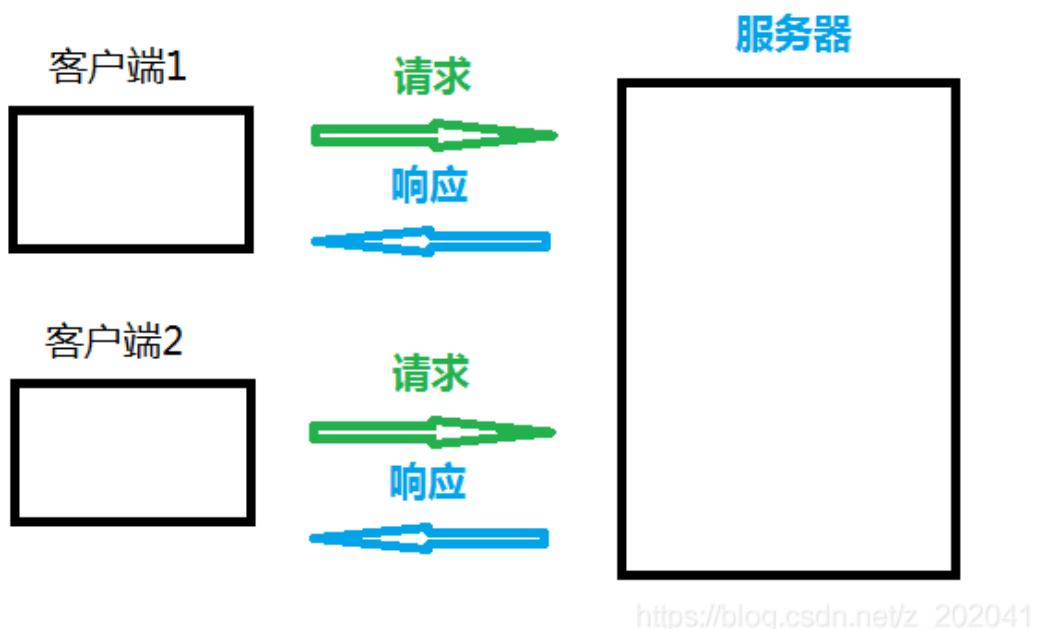
```
1 | http://tts.tmooc.cn/video/showvideo?  
menuId=657421&version=AID201908#subject
```

- 说明:

- protocol (协议)
  - http 通过 HTTP 访问该资源。格式 `HTTP://`
  - https 通过安全的 HTTPS 访问该资源。格式 `HTTPS://`
  - file 资源是本地计算机上的文件。格式: `file:///`
  - ...
- hostname (主机名)
  - 是指存放资源的服务器的域名系统(DNS) 主机名、域名 或 IP 地址。
- port (端口号)
  - 整数, 可选, 省略时使用方案的默认端口;
  - 各种传输协议都有默认的端口号, 如http的默认端口为80。
- path (路由地址)
  - 由零或多个“/”符号隔开的字符串, 一般用来表示主机上的一个目录或文件地址。路由地址决定了服务器端如何处理这个请求
- query(查询)
  - 可选, 用于给动态网页传递参数, 可有多个参数, 用“&”符号隔开, 每个参数的名和值用“=”符号隔开。
- fragment (信息片断)
  - 字符串, 用于指定网络资源中的片断。例如一个网页中有多个名词解释, 可使用 `fragment`直接定位到某一名词解释。
- 注: [] 代表其中的内容可省略

## HTTP协议的请求和响应

- 请求是指浏览器端通过HTTP协议发送给服务器端的数据
- 响应是指服务器端接收到请求后做相应的处理后再回复给浏览器端的数据



[https://blog.csdn.net/z\\_202041](https://blog.csdn.net/z_202041)

## HTTP 请求

```

1 POST /v1/tokens HTTP/1.1      -> 起始行
2
3 Host: 127.0.0.1:8000          -> headers
4 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:74.0) Gecko/20100101
   Firefox/74.0
5 Accept: */*
6 Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/json
9 Content-Length: 58
10 Origin: http://127.0.0.1:7000
11 Connection: keep-alive
12 Referer: http://127.0.0.1:7000/dadashop/templates/login.html
13
14 {"username":"guoxiaonao"}     -> body

```

- 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(最常用)
- HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。
- HTTP1.1 请求详述

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。

- HttpRequest对象
  - 视图函数的第一个参数是HttpRequest对象
  - 服务器接收到http协议的请求后，会根据请求数据报文创建HttpRequest对象
  - HttpRequest属性
    - path\_info: URL字符串
    - method: 字符串，表示HTTP请求方法，常用值: 'GET'、 'POST'
    - encoding: 字符串，表示提交的数据的编码方式
      - 如果为None则表示使用浏览器的默认设置，一般为'utf-8'
      - 这个属性是可写的，可以通过修改它来修改访问表单数据使用的编码，接下来对属性的任何访问将使用新的encoding值
    - GET: QueryDict查询字典的对象，包含get查询字符串请求方式的所有数据
    - POST: QueryDict查询字典的对象，只包含post表单提交请求方式的所有数据
    - FILES: 类似于字典的对象，包含所有的上传文件信息
    - COOKIES: Python字典，包含所有的cookie，键和值都为字符串
    - session: 似于字典的对象，表示当前的会话
    - body: 字符串，请求体的内容(POST或PUT)
    - scheme : 请求协议('http'/'https')
    - request.get\_full\_path() : 请求的完整路径
    - request.get\_host() : 请求的主机
    - request.META : 请求中的元数据(消息头)
      - request.META['REMOTE\_ADDR'] : 客户端IP地址

## HTTP 响应

```

1 HTTP/1.0 200 OK          -> 起始行
2
3 Date: Sat, 21 Mar 2020 09:44:15 GMT    -> headers
4 Server: WSGIServer/0.2 CPython/3.6.8
5 Content-Type: application/json
6 X-Frame-Options: SAMEORIGIN
7 Content-Length: 217
8 Vary: Origin
9 Access-Control-Allow-Origin: *
10
11 {"code": 200, "username": "guoxiaonao"} -> body

```

- 当浏览者访问一个网页时，浏览者的浏览器会向网页所在服务器发出请求。当浏览器接收并显示网页前，此网页所在的服务器会返回一个包含HTTP状态码的信息头用以响应浏览器的请求。
- HTTP状态码的英文为HTTP Status Code。
- 下面是常见的HTTP状态码：
  - 200 - 请求成功
  - 301 - 永久重定向-资源（网页等）被永久转移到其它URL  
Location: www..... 标明要跳转的目的地。
  - 302 - 临时重定向
  - 404 - 请求的资源（网页等）不存在
  - 500 - 内部服务器错误
- HTTP状态码分类
  - HTTP状态码由三个十进制数字组成，第一个十进制数字定义了状态码的类型，后两个数字没有分类的作用。HTTP状态码共分为5种类型：

分类	分类描述
1**	信息，服务器收到请求，需要请求者继续执行操作
2**	成功，操作被成功接收并处理
3**	重定向，需要进一步的操作以完成请求
4**	客户端错误，请求包含语法错误或无法完成请求
5**	服务器错误，服务器在处理请求的过程中发生了错误

- Django中的响应对象HttpResponse:
  - 构造函数格式:
    - HttpResponse(content=响应体, content\_type=响应体数据类型, status=状态码)
  - 作用:
    - 向客户端浏览器返回响应，同时携带响应体内容
  - 参数:
    - content: 表示返回的内容。
    - status\_code: 返回的HTTP响应状态码(默认为200)。

- `content_type`: 指定返回数据的的MIME类型(默认为"text/html")。浏览器会根据这个属性, 来显示数据。如果是text/html, 那么就会解析这个字符串, 如果text/plain, 那么就会显示一个纯文本。

- 常用的Content-Type如下:

- `'text/html'` (默认的, html文件)
- `'text/plain'` (纯文本)
- `'text/css'` (css文件)
- `'text/javascript'` (js文件)
- `'multipart/form-data'` (文件提交)
- `'application/json'` (json传输)
- `'application/xml'` (xml文件)

注: 关键字MIME(Multipurpose Internet Mail Extensions)是指多用途互联网邮件扩展类型。

- `HttpResponse` 子类

类型	作用	状态码
<code>HttpResponseRedirect</code>	重定向	302
<code>HttpResponseNotModified</code>	未修改	304
<code>HttpResponseBadRequest</code>	错误请求	400
<code>HttpResponseNotFound</code>	没有对应的资源	404
<code>HttpResponseForbidden</code>	请求被禁止	403
<code>HttpResponseServerError</code>	服务器错误	500

## cookies 和 session

- 会话 - 从打开浏览器访问一个网站, 到关闭浏览器结束此次访问, 称之为一次会话
- HTTP协议是无状态的, 导致会话状态难以保持
- 试想一下, 如果不保持会话状态, 在电商网站购物的场景体验?

Cookies和Session就是为了保持会话状态而诞生的两个存储技术

## cookies

- cookies是保存在客户端浏览器上的存储空间
- Chrome 浏览器 可能通过开发者工具的 `Application` >> `Storage` >> `Cookies` 查看和操作浏览器端所有的 Cookies 值
- 火狐浏览器 可能通过开发者工具的 `存储 -> Cookie`
  - cookies 在浏览器上是以键-值对的形式进行存储的, 键和值都是以ASCII字符串的形式存储(不能是中文字符串)
  - cookies中的数据是按域存储隔离的, 不同的域之间无法访问
  - cookies 的内部的数据会在每次访问此网址时都会携带到服务器端, 如果cookies过大会降低响应速度
- 在Django 设置浏览器的COOKIE 必须通过 `HttpResponse` 对象来完成
  - 添加、修改COOKIE

- `HttpResponse.set_cookie(key, value='', max_age=None, expires=None)`
  - key:cookie的名字
  - value:cookie的值
  - max\_age:cookie存活时间, 秒为单位
  - expires:具体过期时间
  - 当不指定max\_age和expires时,关闭浏览器时此数据失效
- 删除COOKIE
  - `HttpResponse.delete_cookie(key)`
  - 删除指定的key 的Cookie。如果key不存在则什么也不发生。
  - 并不是删除key, 而是把KEY值置为“空字符串”, 再给一个已经过期的到期时间 (1970年), 再把存活时间设置为0.
- 获取cookie
  - 通过 `request.COOKIES` 绑定的字典(dict) 获取客户端的 COOKIES数据

```

1 | value = request.COOKIES.get('cookies名', '默认值')
2 | print("cookies名 = ", value)

```

- 示例
  - 以下示例均在视图函数中调用
  - 添加cookie
 

```

1 | # 为浏览器添加键为 my_var1, 值为123, 过期时间为1个小时的cookie
2 | responds = HttpResponse("已添加 my_var1, 值为123")
3 | responds.set_cookie('my_var1', 123, 3600)
4 | return responds

```
  - 修改cookie
 

```

1 | # 为浏览器添加键为 my_var1, 修改值为456, 过期时间为2个小时的cookie
2 | responds = HttpResponse("已修改 my_var1, 值为456")
3 | responds.set_cookie('my_var1', 456, 3600*2)
4 | return responds

```
  - 删除cookie
 

```

1 | # 删除浏览器键为 my_var1的cookie
2 | responds = HttpResponse("已删除 my_var1")
3 | responds.delete_cookie('my_var1')
4 | return responds

```
  - 获取cookie
 

```

1 | # 获取浏览器中 my_var变量对应的值value = request.COOKIES.get('my_var1',
2 |     '没有值!')print("cookie my_var1 = ", value)return
3 |     HttpResponse("my_var1:" + value)

```

# session

- session又名会话控制，是在服务器上开辟一段空间用于保留浏览器和服务器交互时的重要数据
- 实现方式
  - 使用 session 需要在浏览器客户端启动 cookie，且用在cookie中存储sessionid
  - 每个客户端都可以在服务器端有一个独立的Session
  - 注意：不同的请求者之间不会共享这个数据，与请求者一一对应
- Django中配置Session
  - 在 settings.py 文件中
  - 向 INSTALLED\_APPS 列表中添加：

```
1 | INSTALLED_APPS = [      # 启用 sessions 应用
                      'django.contrib.sessions',]
```

- 向 MIDDLEWARE 列表中添加：

```
1 | MIDDLEWARE = [      # 启用 Session 中间件
                   'django.contrib.sessions.middleware.SessionMiddleware',]
```

- session的基本操作:
  - session对象是一个类似于字典的SessionStore类型的对象,可以用类似于字典的方式进行操作
  - session 只能够存储能够序列化的数据,如字典,列表等。
- 1. 保存 session 的值到服务器
  - `request.session['KEY'] = VALUE`
- 2. 获取session的值
  - `VALUE = request.session['KEY']`
  - `VALUE = request.session.get('KEY', 缺省值)`
- 3. 删除session的值
  - `del request.session['KEY']``
- 在 settings.py (最下面添加) 中有关 session 的设置
  - 1. SESSION\_COOKIE\_AGE
    - 作用: 指定sessionid在cookies中的保存时长(默认是2周), 如下:
    - `SESSION_COOKIE_AGE = 60 * 60 * 24 * 7 * 2`
  - 2. SESSION\_EXPIRE\_AT\_BROWSER\_CLOSE = True
    - 设置只要浏览器关闭时,session就失效(默认为False)
- 注: 当使用session时需要迁移数据库,否则会出现错误

```
1 | python3 manage.py migrate
```

```
1 | django 原生session 问题: 1, django_session表是 单表设计; 且该表数据量持续增持【浏览器故意删掉sessionid&过期数据未删除】2, 可以每晚执行 python3 manage.py clearsessions 【该命令可删除已过期的session数据】
```

## Cookies vs session

1 | 存储位置: C- 浏览器中    S- 服务器中【mysql】 安全性: C - 不安全    S- 相对安全一些不管C还是S , 不要存储敏感数据 【密码】

## 缓存

### 什么是缓存?

缓存是一类可以更快的读取数据的介质统称，也指其它可以加快数据读取的存储方式。一般用来存储临时数据，常用介质的是读取速度很快的内存

### 为什么使用缓存?

视图渲染有一定成本，对于低频变动的页面可以考虑使用缓存技术，减少实际渲染次数

#### 案例分析

```
1 | from django.shortcuts import renderdef index(request):      # 时间复杂度极高的渲染  
    book_list = Book.objects.all()  #-> 此处假设耗时2s      return render(request,  
    'index.html', locals())
```

#### 优化思想

```
1 | given a URL, try finding that page in the cacheif the page is in the cache:  
    return the cached pageelse:    generate the page    save the generated page  
    in the cache (for next time)    return the generated page
```

### 使用缓存场景:

- 1, 博客列表页
- 2, 电商商品详情页
- 3, 缓存导航及页脚

### Django中设置缓存

Django中提供多种缓存方式，如需使用需要在settings.py中进行配置

#### 1,数据库缓存

Django可以将其缓存的数据存储在您的数据库中

```
1 CACHES = {    'default': {        'BACKEND':  
    'django.core.cache.backends.db.DatabaseCache',        'LOCATION':  
    'my_cache_table',        'TIMEOUT': 300, #缓存保存时间 单位秒, 默认值为300,  
    'OPTIONS':{            'MAX_ENTRIES': 300, #缓存最大数据条数  
    'CULL_FREQUENCY': 2,#缓存条数达到最大值时 删除1/x的缓存数据        }    }}
```

创建缓存表

```
1 python3 manage.py createcachetable
```

2,文件系统缓存

```
1 CACHES = {    'default': {        'BACKEND':  
    'django.core.cache.backends.filebased.FileBasedCache',        'LOCATION':  
    '/var/tmp/django_cache',#这个是文件夹的路径        #'LOCATION':  
    'c:\test\cache',#windows下示例    }}
```

3,本地内存缓存

```
1 CACHES = {    'default': {        'BACKEND':  
    'django.core.cache.backends.locmem.LocMemCache',        'LOCATION': 'unique-  
snowflake'    }}
```

## Django中使用缓存

- 在视图View中使用
- 在路由URL中使用
- 在模板中使用

在视图View中使用cache

```
1 from django.views.decorators.cache import cache_page# 这种装饰器的方式缓存数据只能  
以时间到期而失效, 很难手动删除@cache_page(30) -> 单位sdef my_view(request): ...
```

在路由中使用

```
1 from django.views.decorators.cache import cache_pageurlpatterns = [  
    path('foo/', cache_page(60)(my_view)),]
```

在模板中使用

```
1 {% load cache %}{% cache 500 sidebar request.user.username %} .. sidebar  
for logged in user ...{% endcache %}
```

- 缓存api

作用: 局部缓存部分结果

使用:

```

1 #指定配置引入from django.core.cache import cache
cache1 =
caches['myalias']cache2 = caches['myalias_2']#默认配置引入【指的配置中的
default项】等同于 caches['default']from django.core.cache import cache#常
规命令 set#key: 字符串类型#value: Python对象#timeout: 缓存存储时间 默认值为
settings.py CACHES对应配置的TIMEOUT#返回值: Nonecache.set('my_key',
'myvalue', 30)#常规命令 get#返回值: 为key的具体值, 如果没有数据, 则返回
Nonecache.get('my_key')#可添加默认值, 如果没取到返回默认值cache.get('my_key',
'default值')#常规命令 add 只有在key不存在的时候 才能设置成功#返回值 True or
Falsecache.add('my_key', 'value') #如果my_key已经存在, 则此次赋值失效#常规命令
get_or_set 如果未获取到数据 则执行set操作#返回值 key的值
cache.get_or_set('my_key', 'value', 10)#常规命令 get_many(key_list)
set_many(dict,timeout)#返回值 set_many:返回插入不成功的key数组 #
get_many:取到的key和value的字典>>> cache.set_many({'a': 1, 'b': 2, 'c':
3})>>> cache.get_many(['a', 'b', 'c']){'a': 1, 'b': 2, 'c': 3}#常规命令
delete#返回值 Nonecache.delete('my_key')#常规命令 delete_many#返回值 成功删
除的数据条数cache.delete_many(['a', 'b', 'c'])

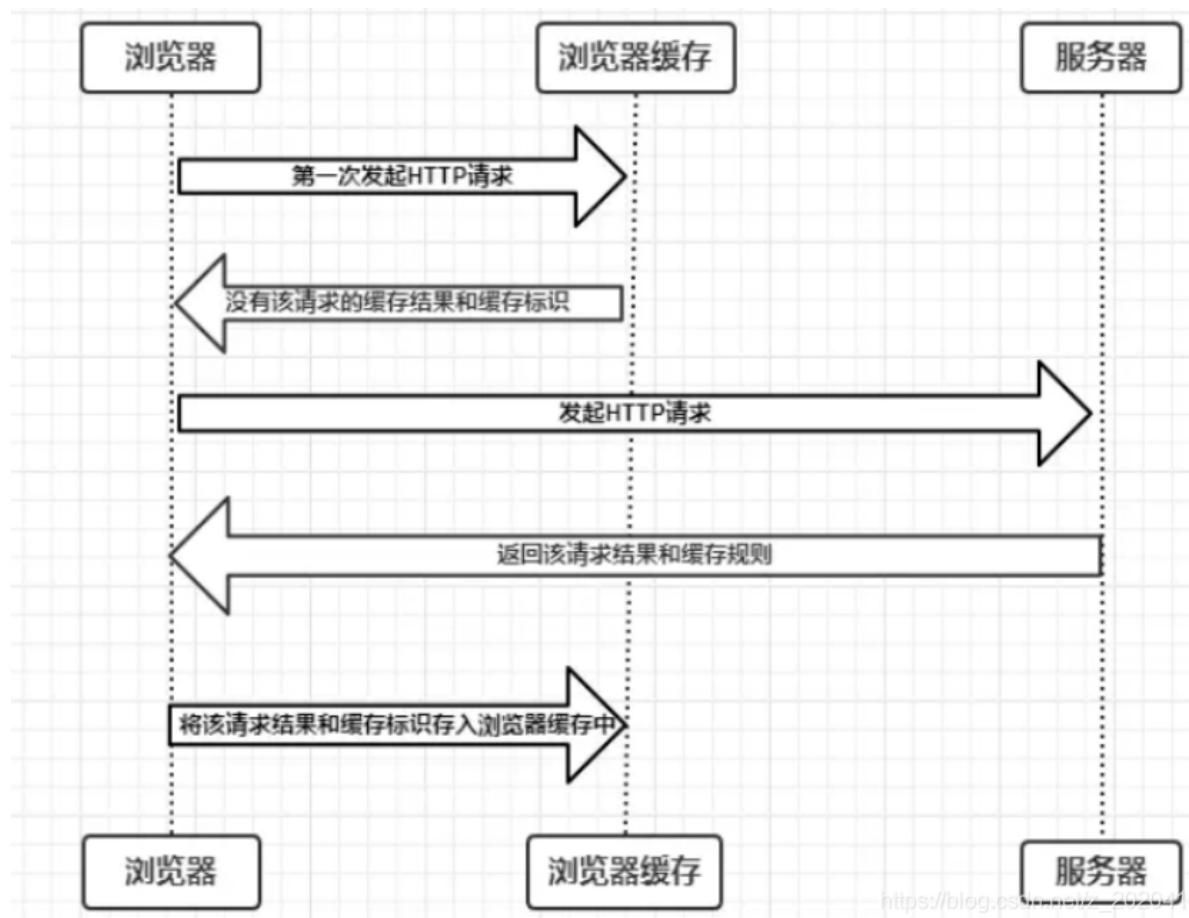
```

```

1 ##### 缓存过期处理: 1) 数量达到阈值: 直接删除 1/N (参照配置项) 2) 惰性删除, 获取一
个过期的数据时, 会被被动的删除掉该数据 (省资源) ####

```

## 浏览器中的缓存



浏览器缓存分类:

## 强缓存

不会向服务器发送请求，直接从缓存中读取资源

1, Expires

缓存过期时间，用来指定资源到期的时间，是服务器端的具体的时间点

Expires:Thu, 02 Apr 2030 05:14:08 GMT

Expires 是 HTTP/1 的产物，受限于本地时间，如果修改了本地时间，可能会造成缓存失效

2, Cache-Control

在HTTP/1.1中，Cache-Control主要用于控制网页缓存。比如当 Cache-Control:max-age=120 代表请求创建时间后的120秒，缓存失效

## 协商缓存

协商缓存就是强制缓存失效后，浏览器携带缓存标识向服务器发起请求，由服务器根据缓存标识决定是否使用缓存的过程

1, Last-Modified和If-Modified-Since

1 | 第一次访问时，服务器会返回

Last-Modified: Fri, 22 Jul 2016 01:47:00 GMT

1 | 浏览器下次请求时 携带 If-Modified-Since 这个 header，该值为 Last-Modified 服务器接收请求后，对比结果，若资源未发生改变，则返回 304，否则返回 200 并将新资源返回给浏览器缺点：只能精确到秒，容易发生单秒内多次修改，检测不到

2, ETag和If-None-Match

1 | Etag 是服务器响应请求时，返回当前资源文件的一个唯一标识（由服务器生成），只要资源有变化，Etag 就会重新生成流程同上

对比 Last-Modified VS ETag

1, 精度不一样 - Etag 高

2, 性能上 - Last-Modifi 高

3, 优先级 - Etag 高

## 前后端分离

## 1.1 什么是前后端分离

1 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, 前端可以有效利用客户端处理数据，有效降低服务端压力3, 服务端错误不会直观的反馈到用户4, 后端灵活搭配各类前端 - 如安卓等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
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【蓝汛 网宿 阿里云 腾讯云】

## 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
- 3, 前端服务和后端服务 分开部署

# token

token (令牌) 的 **KEY** 存在setting里面，用的时候去导入

## 学前须知：

1 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:

1 import base64#base64加密s = b'guoxiaonao'b\_s = base64.b64encode(s)#b\_s打印结果为 b'Z3VveGlhb25hbw=='#base64解密ss = base64.b64decode(b\_s)#ss打印结果为 b'guoxiaonao'

1 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

1 import hashlib = 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s.hexdigest() #十六进制结果

1 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

1 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SHA256h = hmac.new(key, str, digestmod='SHA256') h.digest() #获取最终结果

1 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

# JWT - json-web-token

## 1, 三大组成

1 | 1, header 格式为字典-元数据格式如下

1 | `{'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

1 | 该部分数据需要转成json串并用base64 加密

1 | 2, payload 格式为字典-此部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：

1 | `{'exp':xxx, # Expiration Time 此token的过期时间的时间戳 time.time() + 有效期(s) 'iss':xxx, # (Issuer) Claim 指明此token的签发者 '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 有助于不同的客户端(web, android, apple)做数据统计}`

1 | 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：

1 | `{'username': 'guoxiaonao'}`

1 | 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

1 | `base64(header) + '.' + base64(payload) + '.' + base64(sign)` 最终结果如下：  
`b'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6Imd1b3hpYW9uYW8iLCJpc3MiOiJnZ2cifQ.Zzg1u55DCBqPRGF9z3-NAn4kba-MJN83SxyLFFfc5mmM'`

## 3, 校验jwt规则

1 | 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 | 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
	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

## CORS

### 1, 什么是CORS - Cross-origin resource sharing 跨域资源共享

1 | 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 | 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)

1 | \*\*满足以下全部条件\*\*的请求为 \*\*简单请求\*\*  
GET or HEAD or POST  
Accept-Language  
3, Content-Type 仅支持如下三种:  
multipart/form-data  
请求\*\*

1, 请求方法如下:  
2, 请求头仅包含如下:  
Content-Language  
Content-Type  
application/x-www-form-urlencoded  
text/plain  
\*\*不满足以上任意一点的请求都是 预检

## 4, 简单请求发送流程

1 | 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Cookie	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

1 | 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 | 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

1 | 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

1 | 3, 主请求阶段

请求头	作用	备注
Origin	表明此请求来自哪个域	

1 | 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

```
1 | 1.解压      tar -zxvf django-cors-headers-3.0.2.tar.gz2.      cd django-cors-headers-3.0.23.      sudo python3 setup.py install4.查看      pip3 freeze|grep -i 'cors'# pip3 install django-cors-headers -i https://pypi.tuna.tsinghua.edu.cn/simple/
```

配置 **setting** 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
CORS_ORIGIN_ALLOW_ALL 布尔值 如果为True 白名单不启用      3,
CORS_ORIGIN_WHITELIST =[ "https://example.com" ] # 白名单
ip:port      5, CORS_ALLOW_METHODS = ( 'DELETE',
'GET',           'OPTIONS',           'PATCH',           'POST',
'PUT',           )      6, CORS_ALLOW_HEADERS = ( 'accept-
encoding',       'authorization',       'content-type',
'dnt',           'origin',           'user-agent',       'x-
csrf-token',     'x-requested-with',     )      7,
CORS_PREFLIGHT_MAX_AGE 默认 86400s # options 有效期(默认1天)      8,
CORS_EXPOSE_HEADERS [] # 自定义响应头      9,
CORS_ALLOW_CREDENTIALS 布尔值, 默认False # 告诉浏览器, 要不要coocki, 默认不要

```

## RESTful

### 4.1, 什么是RESTful -Representational State Transfer

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/https2, 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

1 | <https://api.example.com/v1/users><https://api.example.com/v1/animals>

1 | 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) 。 (推荐使用)
- PATCH (UPDATE) : 在服务器更新资源 (客户端提供改变的属性) 。 (django对PATCH支持不是很好)

- **DELETE (DELETE)** : 从服务器删除资源。

具体案例如下：

```
1 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
```

1 | 6, 巧用查询字符串

```
1 ?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
```

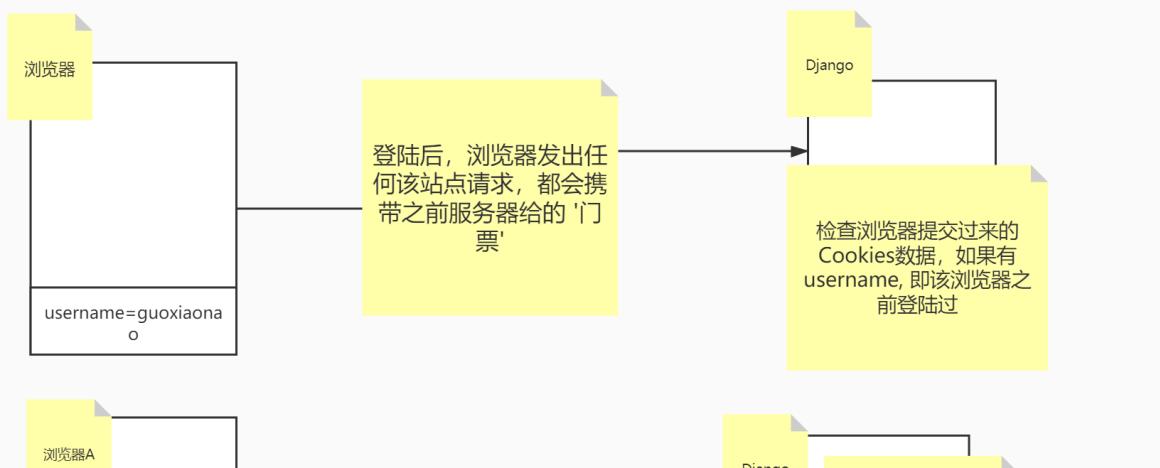
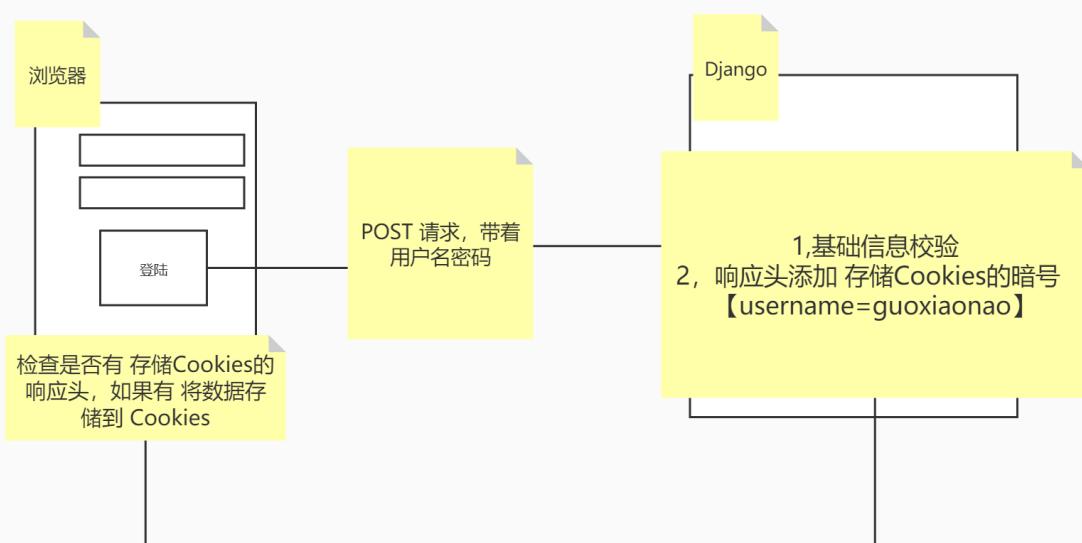
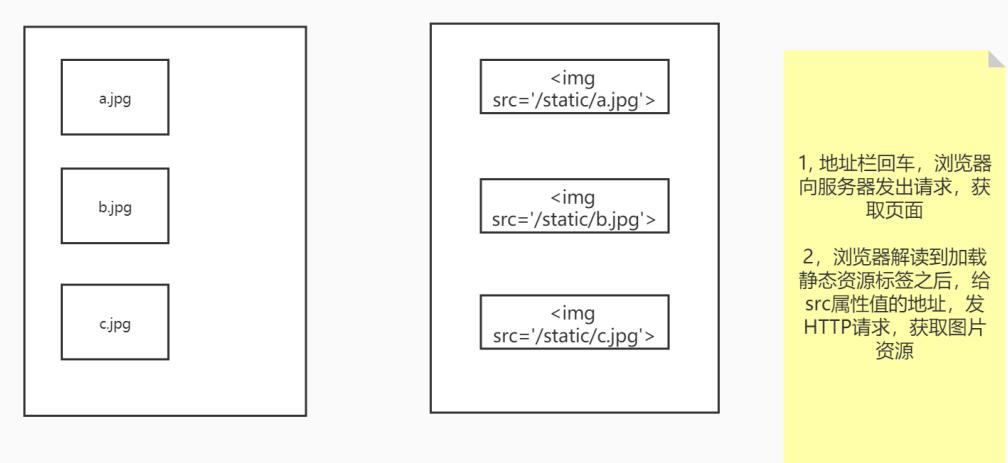
1 | 7, 状态码 1, 用HTTP响应码表达 此次请求结果，例如

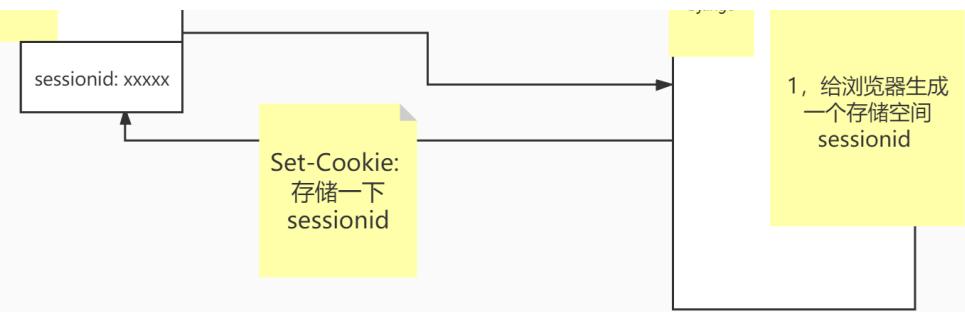
```
1 200 OK - [GET]: 服务器成功返回用户请求的数据
201 CREATED - [POST/PUT/PATCH]: 用于新建或修改数据成功。
202 Accepted - [*]: 表示一个请求已经进入后台排队（异步任务）
204 NO CONTENT - [DELETE]: 用户删除数据成功。
400 INVALID REQUEST - [POST/PUT/PATCH]: 用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的。
401 Unauthorized - [*]: 表示用户没有权限（令牌、用户名、密码错误）。
403 Forbidden - [*]: 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 Unprocessable 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
```

1 | 2, 自定义内部code 进行响应 如 返回结构如下 {'code':200, 'data': {}, 'error': xxx}

1 | 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

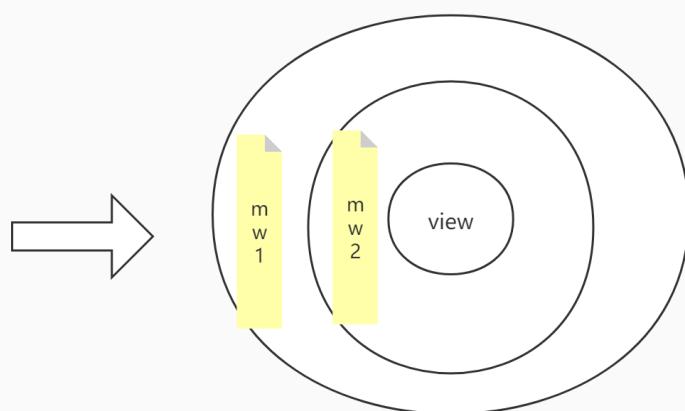
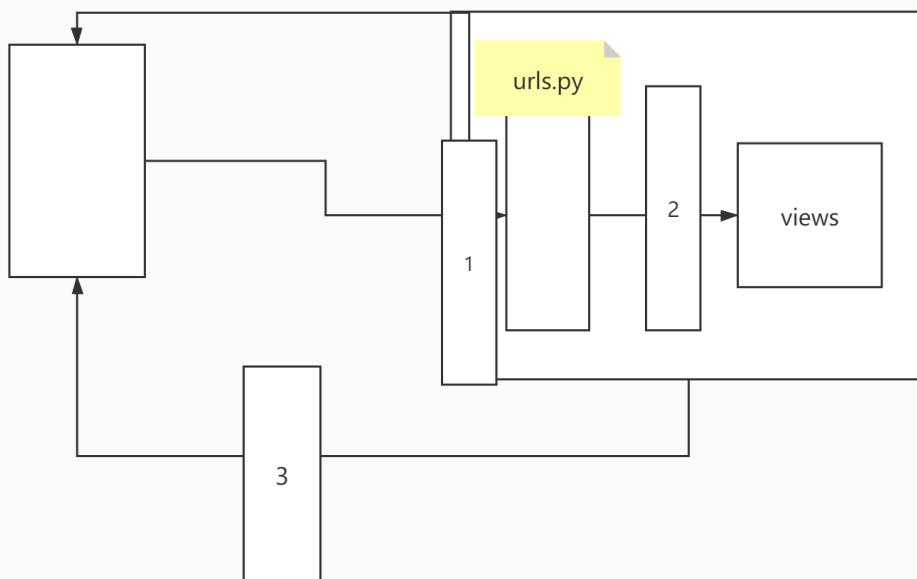
```
1 GET /users: 返回资源对象的列表（数组）
2 GET /users/guoxiaonao: 返回单个资源对象
3 POST /users: 返回新生成的资源对象
4 PUT /users/guoxiaonao: 返回完整的资源对象
5 PATCH /users/guoxiaonao: 返回完整的资源对象
6 DELETE /users/guoxiaonao: 返回一个空文档
```



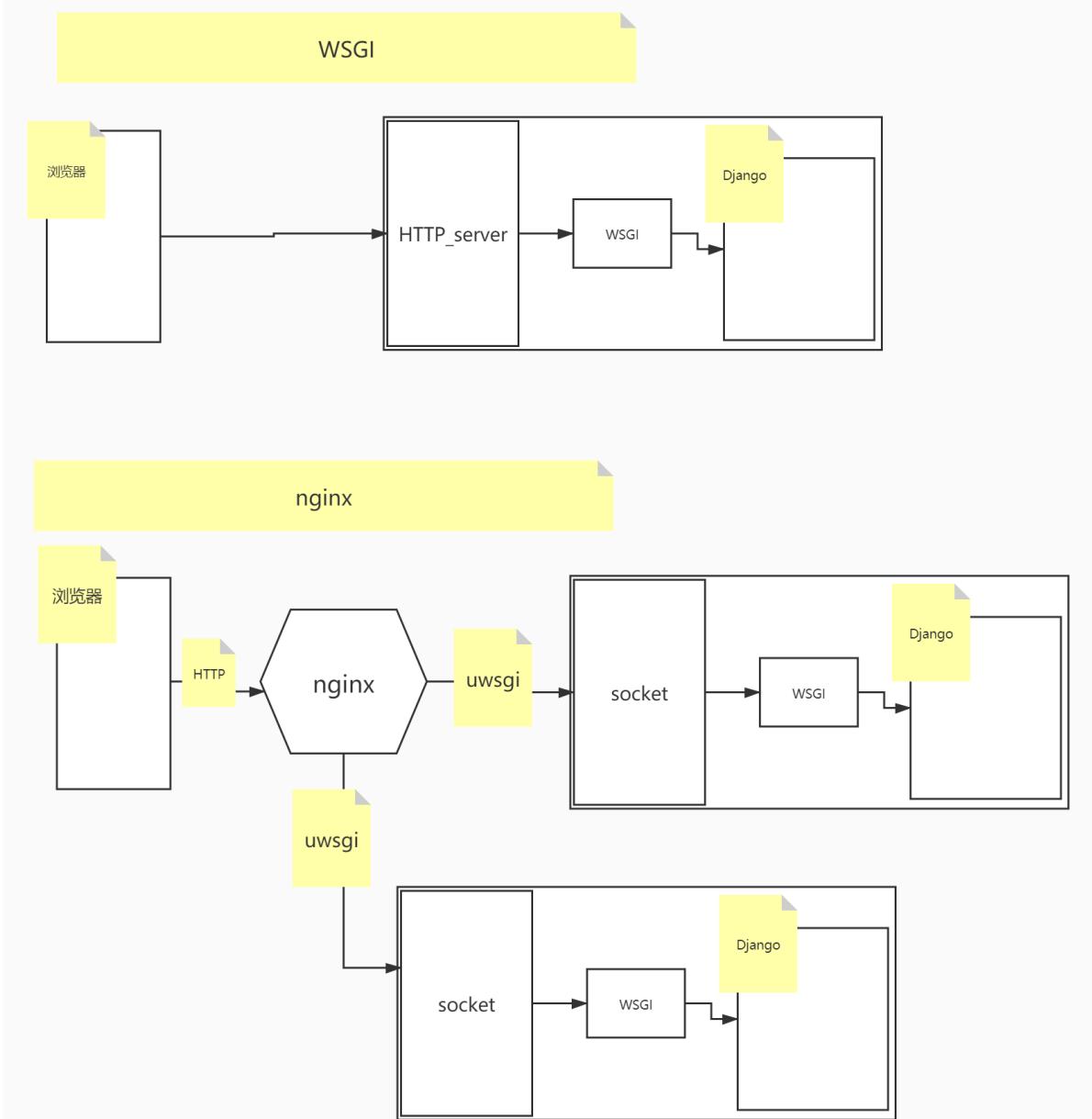
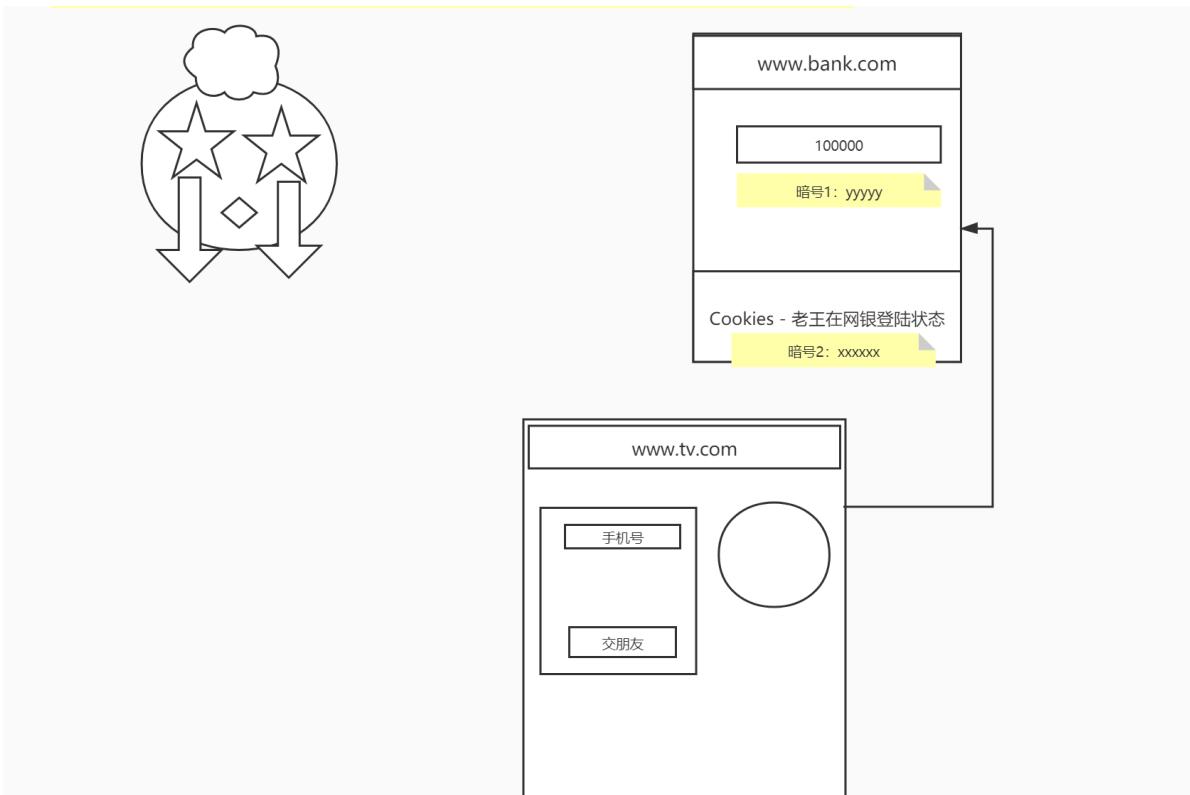


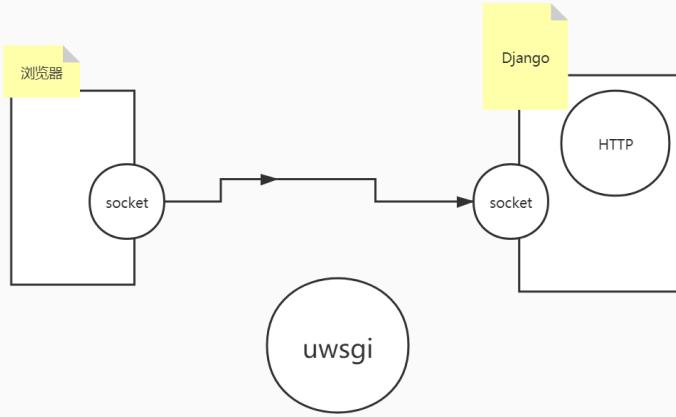
随机sessionid-a:  
{username:'guoxiaonao'}  
随机sessionid-b: {}

## 中间件

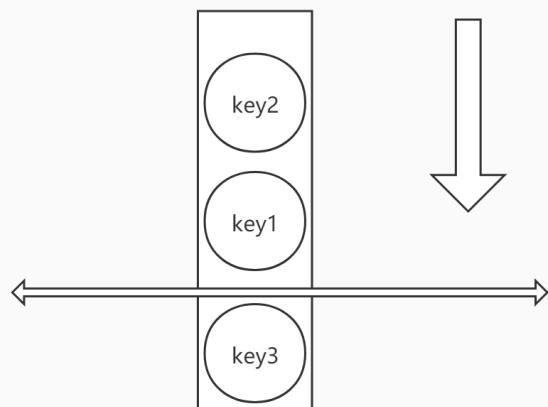
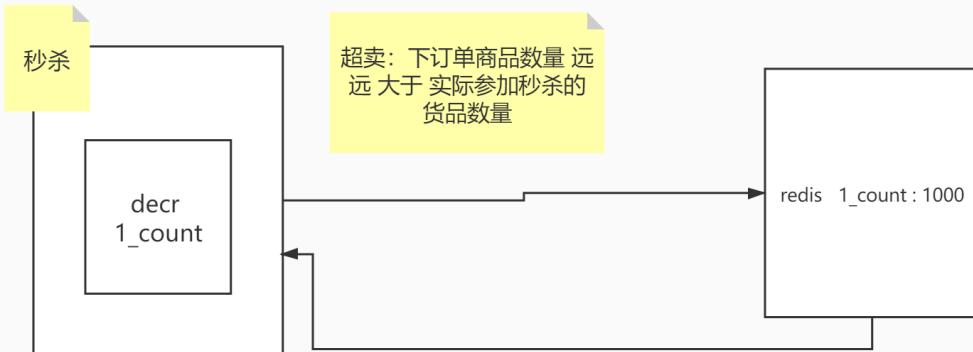


## CSRF

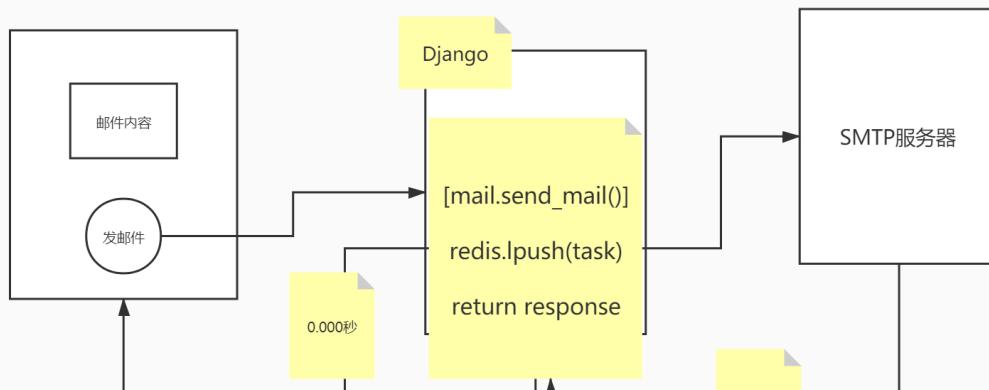


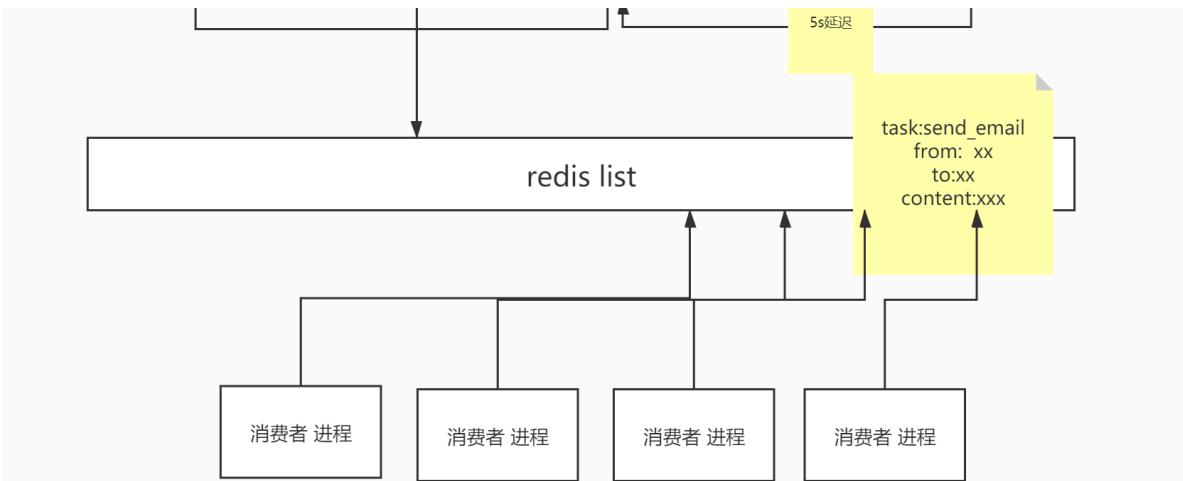


### 并发计数

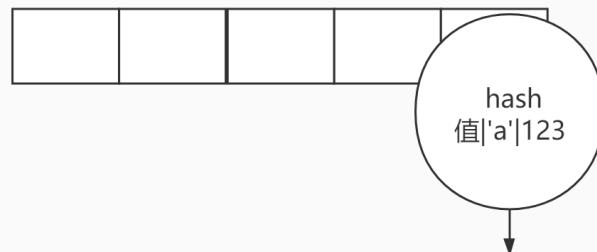


### 生产者 消费者模型





### python + redis hash 的实现



Python

```
d = {} -> 初始化了长度为8的数组
d['a'] = 123
```

```
d['z'] = 123
```

```
hash('a') % 8 = 4
hash('z') % 8 = 4
```

哈希碰撞：

Python 开放地址法

```
hash(4+偏移量) = new_value
```

由于使用开放地址法，py删除 key时，采用伪删除，保留完整的 探测链

扩容：已经使用的座位超过三分之二开始扩容 4倍/2倍[>50000]

扩容后 触发 座位重排(rehash) - 旧数组的元素重新计算位置，用新位置存储到新数组中

Redis hash

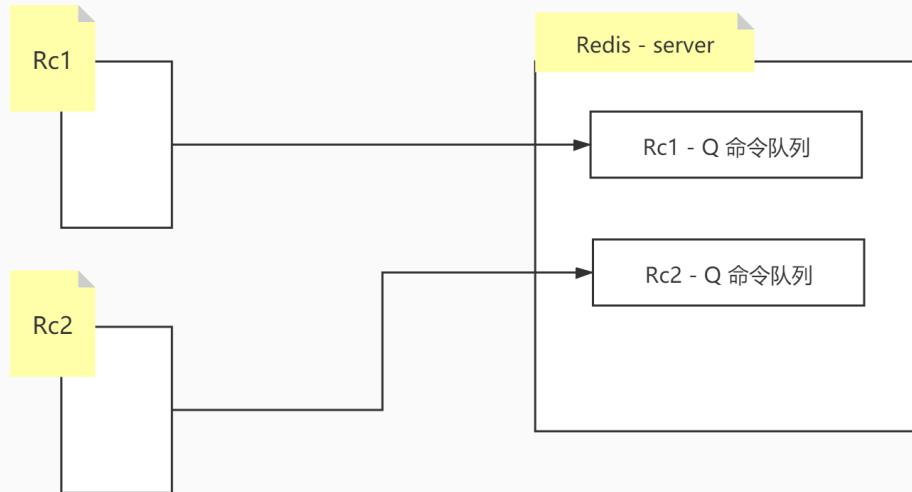
哈希碰撞 - 单链法

扩容 - userd/len(数组) > 1 可能扩 >5 肯定扩

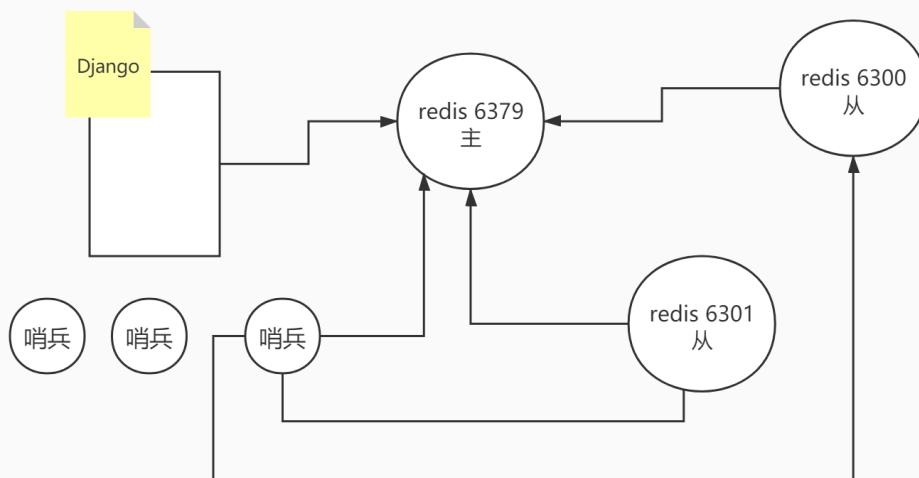
第一个大于 userd \* 2 的 2的n次方 以4为例 2的3次方大于4，则  
使用2的3次方

rehash -> 渐进式rehash

## Redis 事务



## redis 主从复制



[https://blog.csdn.net/z\\_202041](https://blog.csdn.net/z_202041)