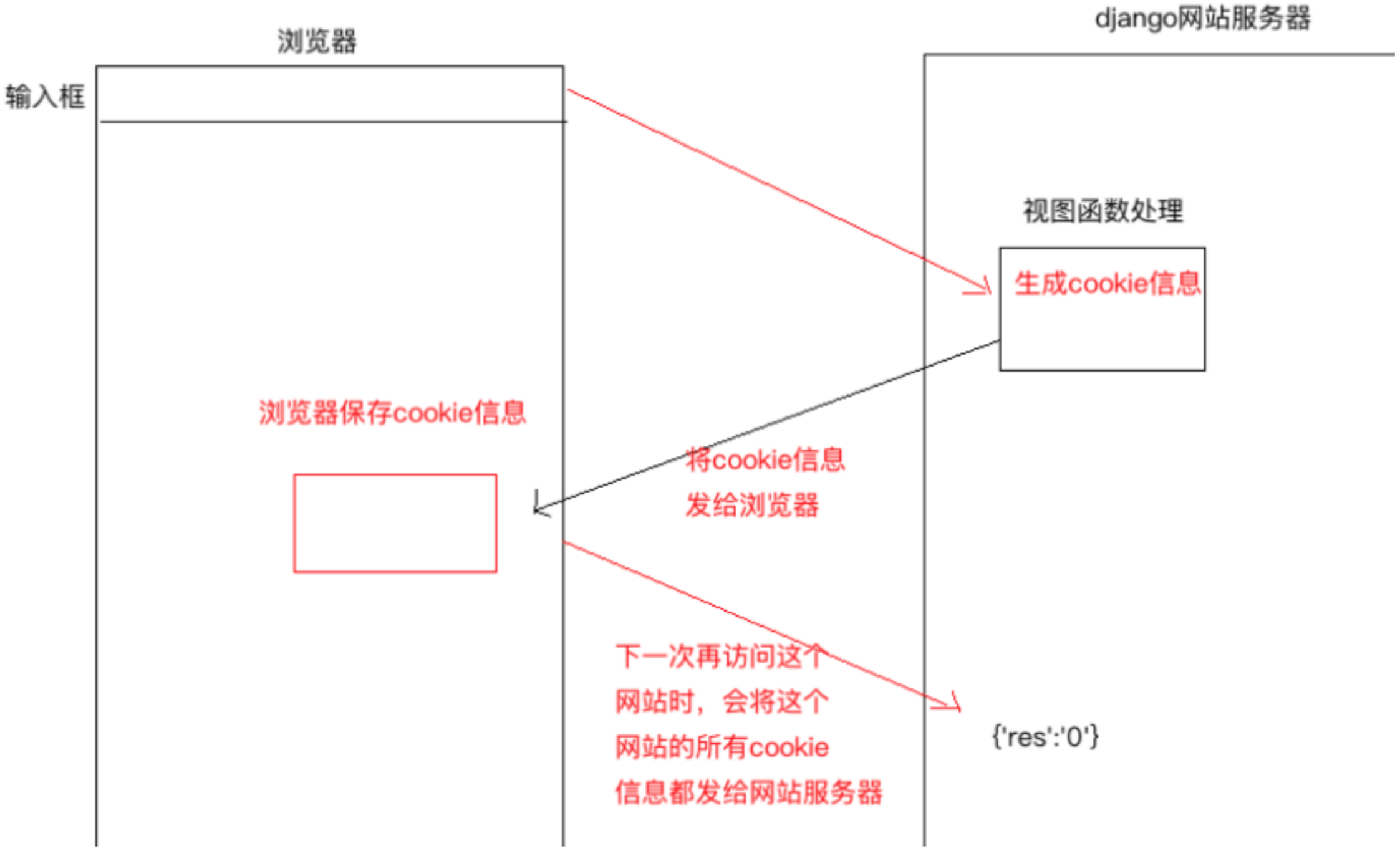
7.django的cookie与session

一.Http的无状态特性：
  http是无状态协议，是指协议对于事务处理没有记忆能力，缺少状态意味着如果后续处理需要前面的信息，则它必须重传； HTTP协议本身是无状态的，这与HTTP协议本来的目的是相符的，客户端只需要简单的向服务器请求下载某些文件，无论是客户端还是服务器都没有必要纪录彼此过去的行为，每一次请求之间都是独立的，但是如果希望几个请求的页面要有关联，比如:我在www.a.com/login里面登陆了，我在www.a.com/index也希望是登陆状态；但是这是2个不同的页面也就是2个不同的HTTP请求，由于这2个HTTP请求是无状态的也就是无关联的，所以无法单纯的在index中读取到它在login中已经登陆了.

  那怎么办呢？我不可能这2个页面我都去登陆一遍，或者用笨方法这2个页面都去查询数据库，如果有登陆状态，就判断是登陆的了；这种查询数据库的方案虽然可行，但是每次都要去查询数据库不是个事，会造成数据库的压力. 所以正是这种诉求，这个时候一个新的客户端存储数据方式出现了，那就是cookie. cookie把少量的信息存储在用户自己的电脑上，它在一个域名下是一个全局的，只要设置它的存储路径在域名www.a.com下，那么当用户用浏览器访问时，django就可以从这个域名的任意页面读取cookie中的信息；所以就很好的解决了我在www.a.com/login页面登陆了之后，我也可以在www.a.com/index获取到这个登陆信息了，同时又不用反复去查询数据库. 虽然这种方案很不错，也很快速方便，但是由于cookie 是存在用户端，而且它本身存储的尺寸大小也有限，最关键是用户可以是可见的并可以随意的修改，很不安全.

二. Cookie的特点：
2.1 cookie是由服务器生成，存储在浏览器端的一小段文本信息.
2.2 以键值对方式进行存储.
2.3 通过浏览器访问一个网站时，会将本地存储的跟网站相关的所有cookie信息发送给该网站的服务器.
2.4 cookie是基于域名安全的.
2.5 Cookie是有过期时间的，如果不指定默认关闭浏览器之后cookie就会过期.

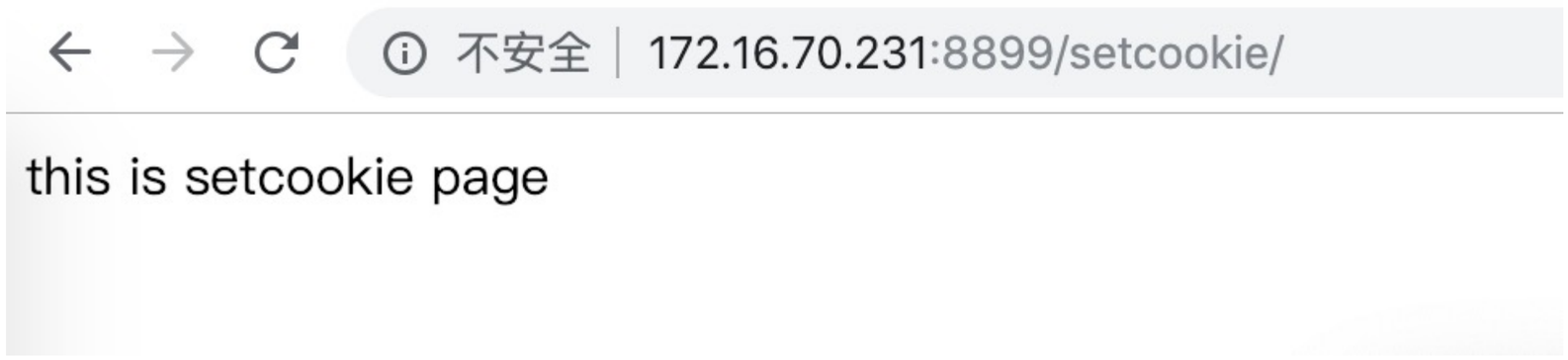


三. django创建cookie并发送给客户端

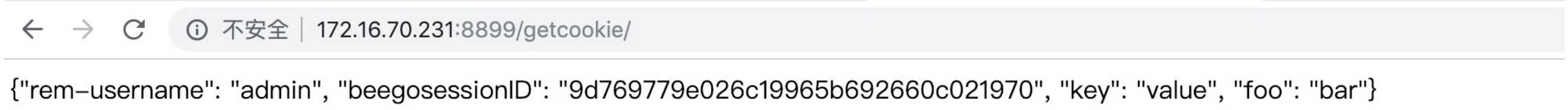
3.1 通过查看django源代码可知：

```
    def set_cookie(self, key, value='', max_age=None, expires=None, path='/',
                   domain=None, secure=False, httponly=False, samesite=None):
        """
        Set a cookie.

        ``expires`` can be:
        - a string in the correct format,
        - a naive ``datetime.datetime`` object in UTC,
        - an aware ``datetime.datetime`` object in any time zone.
        If it is a ``datetime.datetime`` object then calculate ``max_age``.
        """
        self.cookies[key] = value
        if expires is not None:
            if isinstance(expires, datetime.datetime):
                if timezone.is_aware(expires):
                    expires = timezone.make_naive(expires, timezone.utc)
                delta = expires - expires.utcnow()
                # Add one second so the date matches exactly (a fraction of
                # time gets lost between converting to a timedelta and
                # then the date string).
                delta = delta + datetime.timedelta(seconds=1)
                # Just set max_age - the max_age logic will set expires.
                expires = None
                max_age = max(0, delta.days * 86400 + delta.seconds)
            else:
                self.cookies[key]['expires'] = expires
        else:
            self.cookies[key]['expires'] = ''
        if max_age is not None:
            self.cookies[key]['max-age'] = max_age
            # IE requires expires, so set it if hasn't been already.
            if not expires:
                self.cookies[key]['expires'] = http_date(time.time() + max_age)
        if path is not None:
            self.cookies[key]['path'] = path
        if domain is not None:
            self.cookies[key]['domain'] = domain
        if secure:
            self.cookies[key]['secure'] = True
        if httponly:
            self.cookies[key]['httponly'] = True
        if samesite:
            if samesite.lower() not in ('lax', 'strict'):
                raise ValueError('samesite must be "lax" or "strict".')
            self.cookies[key]['samesite'] = samesite
```

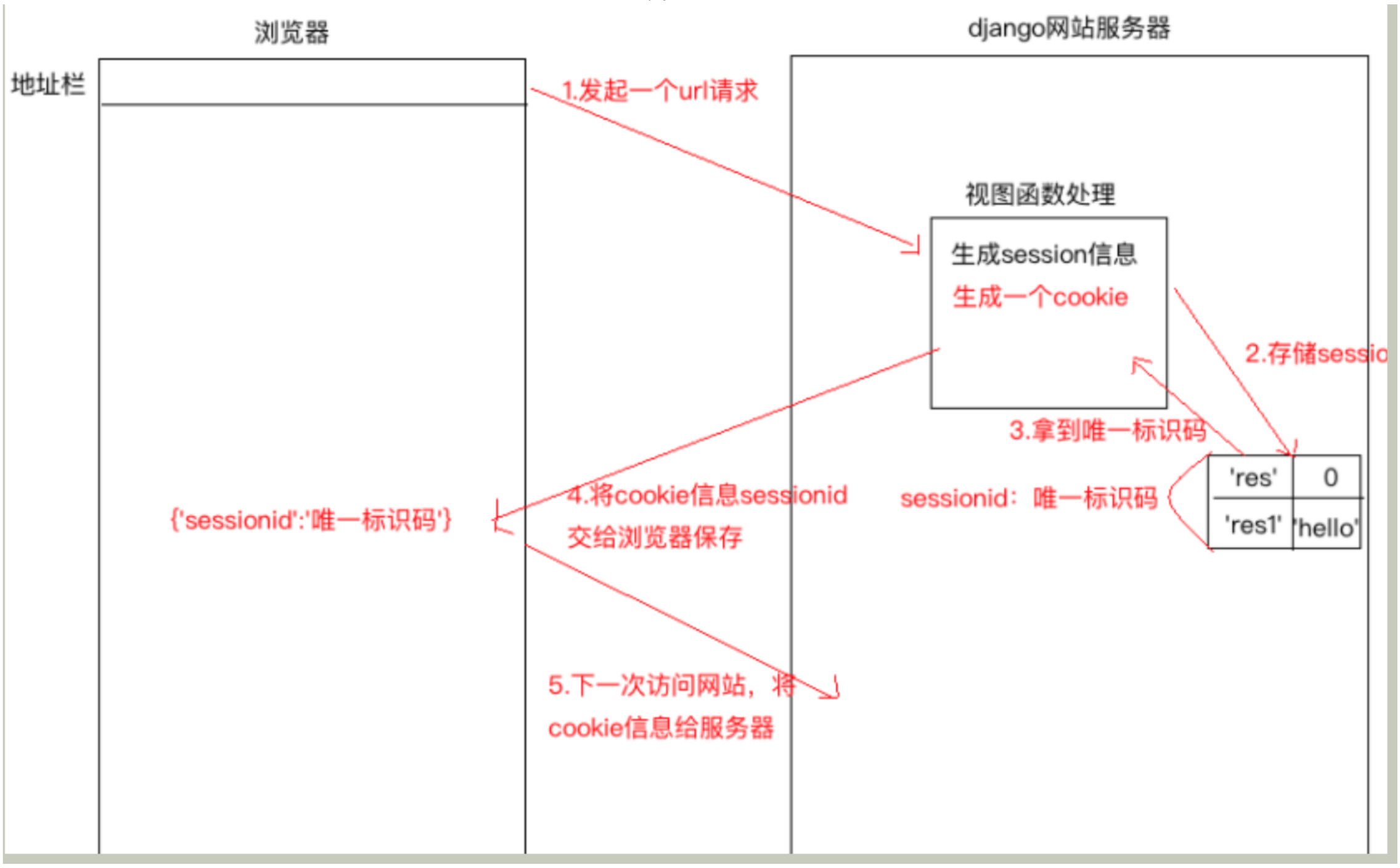
```
    set_cookie方法所接收的参数为:
    key:value ##cookie中的数据以键值对的形式存储(必填)
    max_age   ##cookie生效时间
    expires   ##明确的cookie超时时间
    path      ##指定哪个URL可以访问到cookie
    domain    ##指定哪个域名以及二级域名可以访问到cookie
    secure    ##https相关
    httponly  ##限制只能通过HTTP传输



3.2 后台代码
    def setCookies(request):
        response = HttpResponse("this is setcookie page")
        response.set_cookie("key", 'value')
        response.set_cookie("foo", "bar")
        return response

3.3 定义urls与视图映射关系
    urlpatterns = [
        path('admin/', admin.site.urls),
        path('setcookie/', login_views.setCookies), #新增
    ]


3.4 访问页面并查看浏览器中的cookie数据
```




```
3.5 django获取cookie数据

3.5.1 后台代码
    import json
    def getCookies(request):
        cookies = request.COOKIES
        return HttpResponse(json.dumps(cookies))

3.5.2 关联url与视图函数
    urlpatterns = [
        path('admin/', admin.site.urls),
        path('setcookie/', login_views.setCookies),
        path('getcookie/', login_views.getCookies),
        ]

3.5.3 请求页面测试获取cookie.
```




```
四. 什么是session?

   session是浏览器和服务器的一次交互的会话. 会话是什么呢?  就是我问候你好吗, 你回答很好! 就是一次会话, 那么对话完成后, 这次会话就结束了
.上面我们说cookie因为它存储在客户端, 并且客户端可以对其内容进行修改, 所以cookie很不安全, 那如何又要安全, 又可以方便的全局读取信息呢? 于是就需要用到一种新的存储会话机制, 那就是session, 我们可以将一个变量存入SESSION['name']中, 这样项目的各个页面和逻辑都能访问到, 所以很轻松的用来判断上面我们所介绍到的判断用户是否登陆的这个场景.

   Session对象存储特定用户会话所需的属性及配置信息, 这样当用户在应用程序的Web页之间跳转时, 存储在Session对象中的变量将不会丢失, 而是在整个用户会话中一直存在下去, 当用户请求来自应用程序的Web页时, 如果该用户还没有会话, 则Web服务器将自动创建一个Session对象; 当会话过期或被放弃后, 服务器将终止该会话; 对于敏感  重要的信息, 建议要储在服务器端, 不能存储在浏览器中; 如用户名  余额  等级  验证码等信息.
```




```
4.1 Session的特点:
    在服务器端进行状态保持的方案就是Session.
    session是以键值对进行存储的.
    session依赖于cookie.
    session也是有过期时间, 如果不指定默认两周就会过期.

4.2 Session依赖于Cookie
    问题: 所有请求者的Session都会存储在服务器中, 服务器如何区分请求者和Session数据的对应关系呢?
    答: 在使用Session后, 会在Cookie中存储一个sessionid的数据, 每次请求时浏览器都会将这个数据发给服务器, 服务器在接收到sessionid后, 会根据这个值找出这个请求者的Session.
    结果: 如果想使用Session, 浏览器必须支持Cookie, 否则就无法使用Session了.
    问题: session与cookie存储位置的区别?
    答: session存储在服务器端, cookie存储在客户端

4.3 django session配置

4.3.1.修改项目配置文件setting.py确保MIDDLEWARE_CLASSES 确保其中包含以下内容:
    'django.middleware.clickjacking.XFrameOptionsMiddleware'

4.3.2.确保INSTALLED_APPS中确保有以下内容:
    'django.contrib.sessions',

4.3.3.在settings.py中指定session的存储位置:
    SESSION_ENGINE='django.contrib.sessions.backends.file'
    SESSION_FILE_PATH='/tmp/django'  ##目录不存在要进行创建

4.3.4.Session基本操作
    #创建或修改 session:
    request.session[key] = value
    # 获取 session:
    request.session.get(key,default=None)
    # 删除 session
    del request.session[key] # 不存在时报错

4.4 通过session实现用户登录状态保持:

4.4.1 用户从login.html页面登录-->跳转至index.html页面-->跳转至home.html页面-->logout, 在login页面成功登录账号后, 在index页面和home页面都会显示当前登录的账号, 点击logout安全退出网站!

4.3.1

    前端代码
    login.html
    <form action="/login/" method="post"  enctype="multipart/form-data">
        USER:<input type="text" name="username"><br/>
        PSWD: <input type="password" name="password"><br/>
        <button type="submit">login</button>
    </form>


    后端代码:
    from django.http import HttpResponseRedirect
    from django.shortcuts import reverse
    def login(request):
        print(request.META)
        username = request.POST.get("username", "")
        password = request.POST.get("password", "")
        request.session["username"] = username
        return HttpResponseRedirect('/index')

    def index(request):
        print(request.META)
        username =request.session.get("username", 'err')
        s = '欢迎你: %s, <a href="/home">跳转到个人页面</a>' %username
        return HttpResponse(s)

    def home(request):
        username = request.session.get("username", 'err')
        s = 'this is  %s home page! <a href="/logout">安全退出</a>' %username
        return HttpResponse(s)

    def logout(request):
        request.session.clear()
        s = '您已安全退出!'
        return HttpResponse(s)

4.3.2 程序urls
```

```
    urlpatterns = [
        path('login/', login),
        path('index/', index),
        path('logout/', logout),
        path('home/', home),
    ]
```

4.3.3 项目urls

```
    urlpatterns = [
        path('admin/', admin.site.urls),
        path('', include('login.urls')),
        ]
```

4.3.4 测试


```
设置session超时时间
SESSION_COOKIE_AGE=60*30
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```