cookie和session

HTTP被设计为”无态”，也就是俗称“脸盲”。 这一次请求和下一次请求 之间没有任何状态保持，我们无法根据请求的任何方面(IP地址，用户代理等)来识别来自同一人的连续请求。实现状态保持的方式：在客户端或服务器端存储与会话有关的数据（客户端与服务器端的一次通信，就是一次会话）

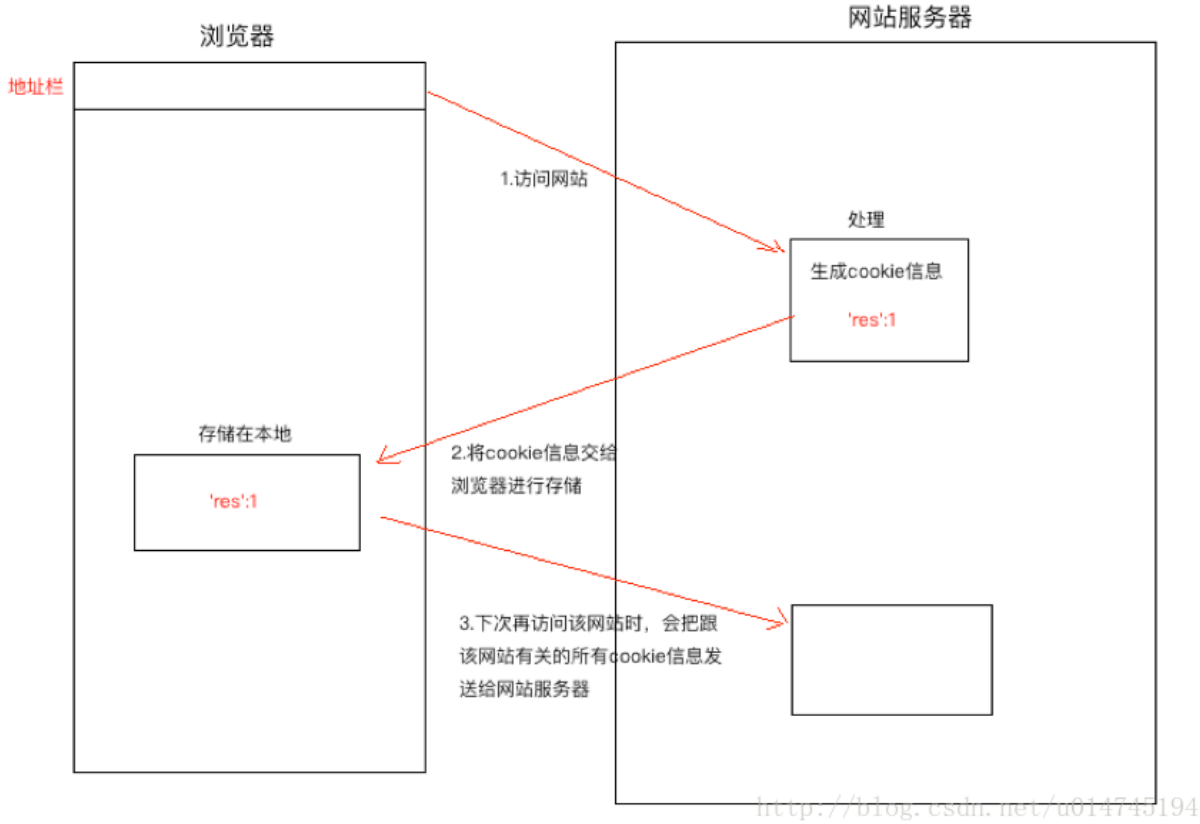
- cookie
- session

不同的请求者之间不会共享这些数据，cookie和session与请求者一一对应。

一、cookie

一种会话数据管理技术，该技术把会话数据保存在浏览器客户端。cookies 是浏览器为 Web 服务器存的一小信息。 每次浏览器从某个服务器请求页面 时，都会自动带上以前收到的cookie。Cookie具有不可跨域名性，不能跨浏览器。Cookie中只能保存ASCII字符串，不能保存汉字，但可以将飞ASCII转码保存

- 优点： 数据存在在客户端，减轻服务器端的压力，提高网站的性能。
- 缺点：安全性不高：在客户端机很容易被查看或破解用户会话信息,容易被用户禁用
- 典型应用：
    - 网站登录
    - 购物车



用法：

```
# 1.设置cookie
HttpResponse.set_cookie(key, value='', max_age=None, expires=None,
path='/', domain=None, secure=None, httponly=False)
参数:
    key:     cookie的名称(*)
    value:   cookie的值,默认是空字符
    max_age: cookies的持续有效时间（以秒计），如果设置为 None，cookies 在
浏览器关闭的时候就失效了。
    expires: cookies的过期时间，格式:"Wdy, DD-Mth-YY HH:MM:SS GMT" 如果
设置这个参数，它将覆盖max_age。
    path:    cookie生效的路径前缀，浏览器只会把cookie回传给带有该路径的页
面，这样你可以避免将cookie传给
             站点中的其他的应用。/ 表示根路径，特殊的：根路径的cookie可以
被任何url的页面访问
    domain:   cookie生效的站点。你可用这个参数来构造一个跨站cookie。如，
domain=".example.com" 所构造的
              cookie对下面这些站点都是可 读的： www.example.com 、
www2.example.com。
              如果该参数设置为None，cookie只能由设置它的站点读取。
    secure:   如果设置为 True ，浏览器将通过HTTPS来回传cookie。
    httponly: 仅http传输    不能使用js获取cookie
```

```
#同set_cookie,不同点在于设置salt，即加盐，加密存储cookie数据
HttpResponse.set_signed_cookie(key, value, salt='', max_age=None,
expires=None, path='/', domain=None, secure=None, httponly=False)
```

```
#2 获取cookie
HttpRequest.COOKIES.get(key)

#获取加“盐”的cookie
HttpRequest.get_signed_cookie(key, default=RAISE_ERROR, salt='',
max_age=None)
```

```
# 3删除cookie
HttpResponse.delete_cookie(key, path='/', domain=None)
```

二、session

cookie看似解决了HTTP（短连接、无状态）的会话保持问题，但把全部用户数据保存在客户端，存在安全隐患，

于是session出现了。我们可以 把关于用户的数据保存在服务端，在客户端cookie里加一个sessionID（随机字符串）。Session中可以保存任意类型的数据。

- django中实现方式
  - 数据库（默认）
  - 缓存
  - 文件
  - 缓存+数据库
  - 加密cookie
  - django-redis-sessions(第三方)

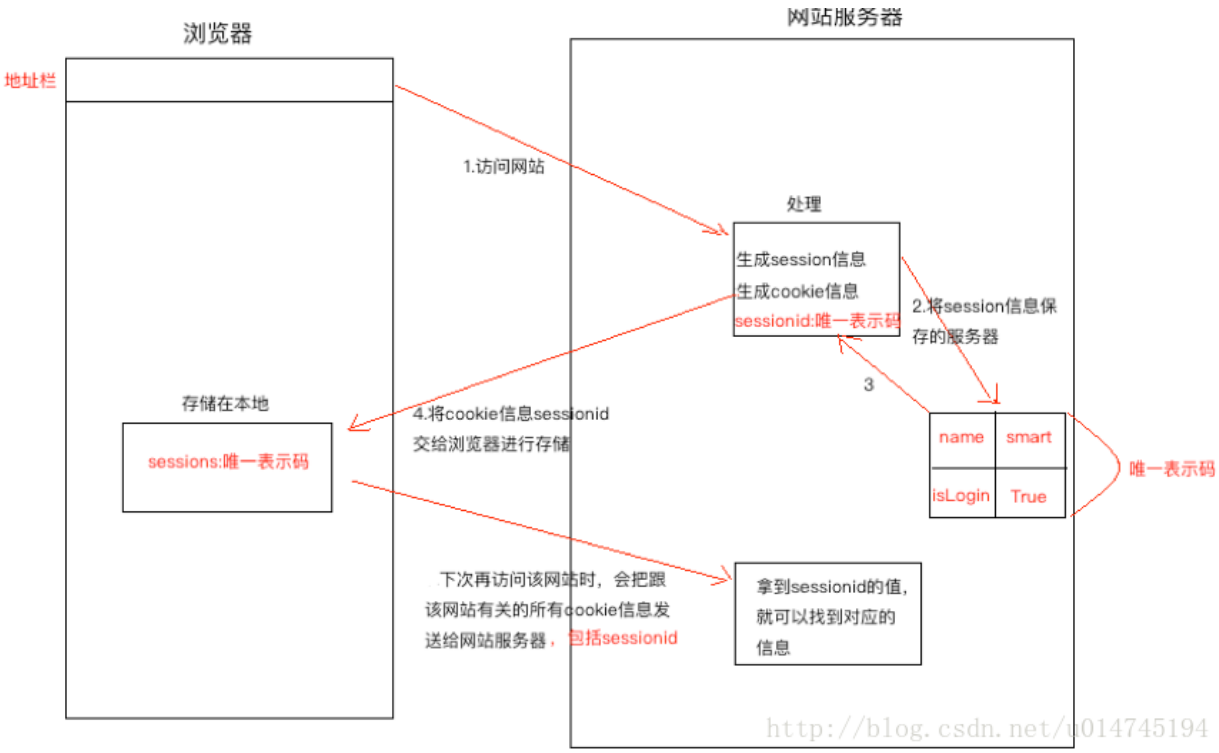
其**工作流程**:

(1)、当用户来访问服务端时,服务端会生成一个随机字符串；

(2)、当用户登录成功后 把 {sessionID :随机字符串} 组织成键值对加到cookie里发送给用户；

(3)、服务器以发送给客户端 cookie中的随机字符串做键，用户信息做值，保存用户信息；

(4)、再访问服务时客户端会带上sessionid，服务器根据sessionid来确认用户是否访问过网站



2.1 cookie和session的区别与联系

- 区别
  - Cookie中只能保存ASCII字符串，Session中可以保存任意类型的数据
  - 隐私策略不同：Cookie存储在客户端，对客户端是可见的，可被客户端窥探、复制、修改。而Session存储在服务器上，不存在敏感信息泄露的风险
  - 有效期不同：Cookie的过期时间可以被设置很长。Session依赖于名为SESSIONID的Cookie，其过期时间默认为-1，只要关闭了浏览器窗口，该Session就会过期，因此Session不能完成信息永久有效。如果Session的超时时间过长，服务器累计的Session就会越多，越容易导致内存溢出。
  - cookie在客户端存储值有大小的限制，大约几kb。session没有限制
  - 跨域支持不同：Cookie支持跨域访问（设置domain属性实现跨子域），Session不支持跨域访问
- 联系
  - session 基于cookie

2.2 session配置

1. **首先在settings.py中有如下配置（系统默认），**

```
INSTALLED_APPS = [
    'django.contrib.sessions',
]
MIDDLEWARE = [
    'django.contrib.sessions.middleware.SessionMiddleware',
]

#其他常用配置
SESSION_COOKIE_NAME="sessionid"  # Session的cookie保存在浏览器上时的key，即：sessionid＝随机字符串
SESSION_COOKIE_PATH="/"  # Session的cookie保存的路径
SESSION_COOKIE_DOMAIN = None  # Session的cookie保存的域名
SESSION_COOKIE_SECURE = False  # 是否Https传输cookie
SESSION_COOKIE_HTTPONLY = True  # 是否Session的cookie只支持http传输
SESSION_COOKIE_AGE = 7*24*60*60  # Session的cookie失效日期（2周） 默认1209600秒
SESSION_EXPIRE_AT_BROWSER_CLOSE =True  # 是否关闭浏览器使得Session过期

SESSION_SAVE_EVERY_REQUEST = True
#如果你设置了session的过期时间 30分钟后，这个参数是False 30分钟过后，session准时失效
#如果设置 True，在30分钟期间有请求服务端，就不会过期!
```

2. 进行数据迁移，生成session使用的数据库表

2.3 session操作

- session设置

```
request.session['key'] = 123
request.session.setdefault('key','default') # 存在则不设置
```

```
def doregister(request):
    username = request.POST.get('username')
    password = request.POST.get('password')
    email = request.POST.get('email')
    user = User()
    user.username = username
    user.password = md5(password.encode('utf8')).hexdigest()
    user.email = email
    user.save()
```

```
    # 设置session
    request.session['username'] = username
    return render(request,"common/notice.html",context={
        'code':1,
        'msg':'注册成功',
        'url':'three:index',
        'wait':3
    })
```

- session获取

```
request.session['key'] #获取指定键，如果不存在则会报错
request.session.get('key',None) #获取指定键，如果没有键，会得到默认值
None，不会报错
```

```
def index(request):
    # session获取
    username = request.session.get('username')
    return render(request,'three/index.html',context=
{'username':username})
```

- session删除
  - clear() 清空所有session 但是不会将session表中的数据删除
  - flush() 清空所有 并删除表中的数据
  - logout() 退出登录 清除所有 并删除表中的数据
  - del req.session['key'] 删除某一个session的值

```
def logout(request):
    request.session.flush()
    return redirect(reverse("three:index"))
```

- 获取session所有的键值对
  - 获取所有的键 request.session.keys()
  - 获取所有的值 request.session.values()
  - 获取所有的键值对 request.session.items()
- session过期时间

```
request.session.set_expiry(value)

1. 如果value是个整数，session会在些秒数后失效。
2. 如果value是个datatime或timedelta，session就会在这个时间后失效。
3. 如果value是0,用户关闭浏览器session就会失效。
4. 如果value是None,session会依赖全局session失效策略。
```

用户认证系统

一、概要

auth模块是Django提供的标准权限管理系统,可以提供用户身份认证, 用户组和权限管理。

auth可以和admin模块配合使用， 快速建立网站的管理系统。

在INSTALLED_APPS中添加'django.contrib.auth'使用该APP, auth模块默认启用。

主要的操作包括:

1. create_user 创建用户
2. authenticate 验证登录
3. login 记住用户的登录状态
4. logout 退出登录
5. is_authenticated 判断用户是否登录
6. @login_required 判断用户是否登录的装饰器

二、前期配置

1、说明

Django 在新建工程时已经为使用用户认证系统做好了全部必要的配置。不过有可能你并非使用 django-admin 命令新建的工程，或者你使用的是一个正在开发中的项目，因此最好再检查一下 settings.py 文件中是否已经做好了全部必要配置。

2、配置

1. 在setting.py的INSTALLED_APPS

```
INSTALLED_APPS = [
    'django.contrib.auth',
    # 用户权限处理部分依赖的应用
    'django.contrib.contenttypes',
]
```

2. 在setting.py的MIDDLEWARE

```
MIDDLEWARE = [
# 会话支持中间件
'django.contrib.sessions.middleware.SessionMiddleware',
# 认证支持中间件
'django.contrib.auth.middleware.AuthenticationMiddleware',
]
```

三、User对象

1、user对象

1. 属性

说明	说明	备注
username	少于等于30个字符。 用户名可以包含字母、数字、_、@、+、.和- 字符	必选
first_name	少于等于30个字符	可选
last_name	少于30个字符	可选
email	邮箱地址	可选
password	密码的哈希及元数据。（Django 不保存原始密码）。原始密码可以无限长而且可以包含任意字符。参见密码相关的文档	必选
groups	与Group 之间的多对多关系	可选
user_permissions	与Permission 之间的多对多关系	可选
is_staff	布尔值。指示用户是否可以访问Admin 站点	可选
is_active	布尔值。指示用户的账号是否激活，缺省值为True	必选
is_superuser	布尔值。只是这个用户拥有所有的权限而不需要给他们分配明确的权限。	可选
last_login	用户最后一次登录的时间	默认值
date_joined	账户创建的时间。当账号创建时，默认设置为当前的date/time	默认值

2. 说明

User 对象属性：username， password（必填项）password用哈希算法保存到数据库

is_staff： 用户是否拥有网站的管理权限.

is_active： 是否允许用户登录, 设置为`False`，可以不用删除用户来禁止 用户登录

2、拓展 User 模型

2.1、说明

> 用户可能还包含有头像、昵称、介绍等等其它属性，因此仅仅使用 Django 内置的 User 模型是不够。所有有些时候我们必须使用在系统的User上进行拓展

2.2、继承AbstractUser

1. 说明

   推荐方式、django.contrib.auth.models.User 也是继承自 `AbstractUser` 抽象基类，而且仅仅就是继承了 `AbstractUser`，没有对 `AbstractUser` 做任何的拓展

2. 在app的models.py中

   ```
   class User(AbstractUser):
       phone = models.CharField(
           max_length=12,
           null=True,
           verbose_name="手机号"
       )
       class Meta(AbstractUser.Meta):
           db_table='user'
           pass
   ```

3. 注意

   为了让 Django 用户认证系统使用我们自定义的用户模型，必须在 settings.py 里通过 `AUTH_USER_MODEL` 指定自定义用户模型所在的位置

   ```
   AUTH_USER_MODEL = 'app名字.User'
   ```

4. 迁移

```
python manage.py makemigrations
python manage.py migrate
```

3、 常用操作

1、验证登录

1. 说明

   当用户登录的时候用`authenticate(username=username,password=password)`验证登录，判断数据库中是否存在用户输入的账号和密码，返回一个user对象。底层将password用hash算法加密后和数据库中password进行对比

2. 示例代码

```
def myauthenticate(request):
    pwd = request.POST.get("pwd", "")
    u_name = request.POST.get("u_name", "")
    if len(pwd) <= 0 or len(u_name) <= 0:
        return HttpResponse("账号或密码不能为空")
    else:
        user = authenticate(username=u_name, password=pwd)
        if user:
            return HttpResponse("验证成功")
        else:
            return HttpResponse("账号或密码错误")
```

2、注册操作

1. 说明

   当用户注册的时候用`create_user(username,password,email)`默认情况下`is_active=True,is_staff=False,is_superuser=False`。

   底层将password用hash算法加密之后存储到数据库中

2. 示例代码

```
def register_view(request):
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
```

```
            phone = request.POST.get('phone')
            email = request.POST.get('email')
            # 验证用户是否存在
            user = User.objects.filter(username=username).first()
            if user:
                # 用户已经存在
                return render(request, 'register.html', {'msg':
'用户名已存在'})
            else:
                # 保存用户
                user =
User.objects.create_user(username=username,

 password=password,
                                                    phone=phone,
                                                    email=email)
                # 将用户信息保存到session中
                login(request, user)
                #发送激活邮件
                return redirect('/')
        except Exception as e:
            return render(request, 'register.html', {'msg': '注册
失败'})
    else:
        return render(request, 'register.html')
```

3、登录操作

1. 说明

   当用户登录的时候用`login(request,user)`来记住用户的登录状态

   该函数接受一个HttpRequest对象，以及一个认证了的User对象

   此函数使用django的session框架给某个已认证的用户附加上session id等信息。

2. 示例代码

```
def login_view(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
         # 验证用户是否存在
```

```
        user = authenticate(request, username=username,
password=password)
        if user:
             # 判断用户是否激活
            if user.is_active:
                login(request, user)
                return redirect('/')
            else:
                return render(request, 'test/login.html', {'msg':
'用户尚未激活'})
        else:
            return render(request, 'test/login.html', {'msg': '用
户密码错误'})
    else:
        return render(request, 'login.html')
```

4、登出操作

1. 说明

   当用户注销的时候用`logout(request)`,只需要一个参数request

2. 示例代码

```
from django.contrib.auth import logout
def logout_view(request):
    logout(request)
```

5、修改密码

1. 说明

2. 示例代码

```
user = auth.authenticate(username=username,
password=old_password)
if user:
    user.set_password(new_password)
    user.save()
```

6、只允许登录用户访问

1. 说明

   @login_required 修饰器修饰的view函数会先通过session key检查是否登录, 已登录用户可以正常的执行操作, 未登录用户将被重定向到login_url指定的位置. 若未指定login_url参数, 则重定向到settings.LOGIN_URL

2. 示例代码

```
# settings 配置
LOGIN_URL = '/day05/login/'
# views
@login_required
def find_user_info(request):
    pass
```

```
@login_required(login_url='/day05/phone/login')
def find_user_info(request):
    pass
```

7、验证登录

1. 说明

   如果是真正的 User 对象，返回值恒为 True 。 用于检查用户是否已经通过了认证。 通过认证并不意味着用户拥有任何权限，甚至也不检查该用户是否处于激活状态，这只是表明用户成功的通过了认证。 这个方法很重要, 在后台用request.user.is_authenticated()判断用户是否已经登录，如果true则可以向前台展示request.user.name

2. 示例代码

```
在后台的视图函数里可以用request.user.is_authenticated()判断用户是否登录
在前端页面中可以用
{% if user.is_authenticated %}
{% endif %}
判断用户是否登录
```

8、修改密码

- make_password(password)
- check_password(password,encoded,salt=None)

四、邮箱验证

1. 说明
    1. 处理用户注册数据，存入数据库，is_active字段设置为False，用户未认证之前不允许登陆
    2. 产生token，生成验证连接URL
    3. 发送验证邮件
    4. 用户通过认证邮箱点击验证连接，设置is_active字段为True，可以登陆
    5. 若验证连接过期，删除用户在数据库中的注册信息，允许用户重新注册（username、email字段具有唯一性）
2. 注册发送邮箱

```
def register_view(request):
    if request.method == 'POST':
        try:
            username = request.POST.get('username')
            password = request.POST.get('password')
            phone = request.POST.get('phone')
            email = request.POST.get('email')
            # 验证用户是否存在
            user = authenticate(username=username,
password=password)
            if user:
                # 用户已经存在
                return render(request, 'register.html', {'msg':
'用户名已存在'})
            else:
                # 保存用户
                user =
User.objects.create_user(username=username,

 password=password,
                                                    phone=phone,
                                                    email=email)
                # 将用户信息保存到session中
                # login(request, user)
                # 发送邮件验证
              return render(request, 'message.html', {'message':
u"请登录到注册邮箱中验证用户，有效期为1个小时"})
        except Exception as e:
```

```
            return render(request, 'register.html', {'msg': '注册
失败'})
    else:
        return render(request, 'register.html')
```

3. 发送邮件
4. 邮件验证连接主要有两步
    - 一是产生token，即加密，
    - 二是处理验证链接。这里采用base64加密，及itsdangerous序列化（自带时间戳)

```
from itsdangerous import URLSafeTimedSerializer as utsr
import base64
import re
from django.conf import settings as django_settings

class Token:
    def __init__(self, security_key):
        self.security_key = security_key
        self.salt = base64.encodestring(security_key)
    def generate_validate_token(self, username):
        serializer = utsr(self.security_key)
        return serializer.dumps(username, self.salt)
    def confirm_validate_token(self, token, expiration=3600):
        serializer = utsr(self.security_key)
        return serializer.loads(token, salt=self.salt,
max_age=expiration)
    def remove_validate_token(self, token):
        serializer = utsr(self.security_key)
        print(serializer.loads(token, salt=self.salt))
        return serializer.loads(token, salt=self.salt)

token_confirm = Token(django_settings.SECRET_KEY)    # 定义为全局
变量
```

5. 激活用户

```
def active_user(request, token):
    try:
        username = token_confirm.confirm_validate_token(token)
    except:
```

```
            username = token_confirm.remove_validate_token(token)
            users = User.objects.filter(username=username)
            for user in users:
            user.delete()
            return render(request, 'message.html', {'message': u'对不
起，验证链接已经过期，请重新<a href=\"' +
unicode(django_settings.DOMAIN) + u'/signup\">注册</a>'})
        try:
            user = User.objects.get(username=username)
        except User.DoesNotExist:
            return render(request, 'message.html', {'message': u"对不
起，您所验证的用户不存在，请重新注册"})
        user.is_active = True
        user.save()
        message = u'验证成功，请进行<a href=\"' +
unicode(django_settings.DOMAIN) + u'user/login\">登录</a>操作'
        return render(request, 'message.html', {'message':message})
```