Cookie和Session

> Cookie及Session一直以来都是Web开发中非常关键的一环，因为HTTP协议本身为无状态，每一次请求之间没有任何状态信息保持，往往我们的Web服务无法在客户端访问过程中得知用户的一些状态信息，比如是否登录等等；那么这里通过引入Cookie或者Seesion来解决这个问题。

> 当客户端访问时，服务端会为客户端生成一个Cookie键值对数据，通过Response响应给到客户端。当下一次客户端继续访问相同的服务端时，浏览器客户端就会将这个Cookie值连带发送到服务端。

> Cookie值存储在浏览器下，一般在你的浏览器安装目录的Cookie目录下，我们也可以通过F12或者各种浏览器的开发者工具来获取到

> 因为cookie是保存在浏览器中的一个纯明文字符串，所以一般来说服务端在生成cookie值时不建议存储敏感信息比如密码

Cookie

> 在django的代码中，我们可以使用一些提供Response响应的类，如：HttpResponse，redirect等实例的内置set_cookie函数来进行django项目中的Cookie设置

- `set_cookie(key, value='', max_age=None, expires=None, path='/',domain=None, secure=False, httponly=False)`

  > key：Cookie的key值，未来通过该key值获取到对应设置好的Cookie。
  >
  > value=''：对应Cookie的key值的value，比如：set_cookie(key='value',value='shuai')
  >
  > max_age=None：Cookie生效的时间，单位为秒，如果Cookie值只持续在客户端浏览器的会话时长，那么这个值应该为None。存在该值时，expires会被计算得到。
  >
  > expires=None：Cookie具体过期日期，是一个datetime.datetime对象，如果该值存在，那么max_age也会被计算得到

  ```
  import datetime
  current_time = datetime.datetime.now() # 当前时间
  expires_time = current_time + datetime.timedelta(seconds=10) # 向后推延十秒
  set_cookie('key','value',expires=expires_time) #设置Cookie及对应超时时间
  ```

  > path='/'：指定哪些url可以访问到Cookie，默认'/'为所有。
  >
  > domain=None：当我们需要设置的为一个跨域的Cookie值，那么可以使用该参数，比如：domain='.test.com'，那么这个Cookie值可以被www.test.com、bbs.test.com等主域名相同的域所读取，否则Cookie只被设置的它的域所读取。为None时，代表当前域名下全局生效。
  >
  > secure=False：https加密传输设置，当使用https协议时，需要设置该值，同样的，如果设置该值为True，如果不是https连接情况下，不会发送该Cookie值。
  >
  > httponly=False：HTTPOnly是包含在HTTP响应头部中Set-Cookie中的一个标记。为一个bool值，当设置为True时，代表阻止客户端的Javascript访问Cookie。这是一种降低客户端脚本访问受保护的Cookie数据风险的有效的办法

设置COOKIE

> 简单的实现一下`COOKIE`的设置

```
from django.shortcuts import render,HttpResponse

# Create your views here.
def set_cookie(request):
    # 在HTTPResponse部分设置COOKIE值
    cookie_reponse = HttpResponse('这是一个关于cookie的测试')
    cookie_reponse.set_cookie('test','hello cookie')
    return cookie_reponse
```

> 以上视图函数返回一个`HttpResponse`对象，并在该对象中集成`COOKIE`值的设定，设置`key`值为`test`，`value`值为`hello cookie`

获取COOKIE

> 再来简单的实现一下`COOKIE`的获取

```
def get_cookie(request):
    # 获取cookie值，从request属性中的COOKIE属性中
    cookie_data = request.COOKIES.get('test')
    return HttpResponse('Cookie值为:%s' % cookie_data)
```

> `Cookie`值存储在，`request`中的`COOKIES`属性中
>
> 并且该属性获取到的结果与字典类似，直接通过内置函数`get`获取即可

删除COOKIE

> 这里通过该视图函数路由进行COOKIE的删除

```
def delete_cookie(request):
    response = HttpResponseRedirect('/check_cookie/')
    response.delete_cookie('test')
    return response
```

- `delete_cookie(key, path='/', domain=None)`

  > 在`Cookie`中删除指定的`key`及对应的`value`，如果`key`值不存在，也不会引发任何异常。
  >
  > 由于`Cookie`的工作方式，`path`和`domain`应该与`set_cookie`时使用的值相同，否则`Cookie`值将不会被删除

> 通过`response`相应类的`delete_cookie`方法，本来应该在会话结束之后才消失的`Cookie`值，现在已经被直接删除掉。后台中通过`Request`中的`Cookie`字典获取到值也为`None`
>
> 不要忘记字典的get，获取不到结果时，返回`None`

> 但是，现在还有一个问题，我们在用户浏览器存储的Cookei值为明文，具有极大的安全隐患，django也提供了加密的Cookie值存储及获取方式

防止篡改COOKIE

> 通过`set_signed_cookie`函数进行持有签名的`COOKIE`值设置，避免用户在客户端进行修改
>
> 要记得，这个函数并不是对`COOKIE`值进行加密

- `HttpResonse.set_signed_cookie(key, value, salt='', max_age=None, expires=None, path='/', domain=None, secure=None, httponly=True)`

  > 为`cookie`值添加签名，其余参数与`set_cookie`相同

- `Request.get_signed_cookie(key, salt='', max_age=None)`

  > 从用户请求中获取通过salt盐值加了签名的`Cookie`值。
  >
  > 这里的`salt`要与之前存储时使用的`salt`值相同才可以解析出正确结果。
  >
  > 还要注意的是，如果对应的key值不存在，则会引发`KeyError`异常，所以要记得异常捕获来确定是否含有`Cookie`值

```
def check_salt_cookie(request):
    try:
        salt_cookie = request.get_signed_cookie(key='salt_cookie',salt='nice')
    except KeyError: #获取不到该key值的Cookie
        response = HttpResponse('正在设置一个salt Cookie值')
        response.set_signed_cookie(key='salt_cookie',salt='nice',value='salt_cookie')
        return response
    else: #获取到了对应key值，展示到新的HttpResonse中
        return HttpResponse('获取到的salt Cookie值:%s' % salt_cookie)
```

> 第一次访问的时候，还没有加`Cookie`值，所以我们在获取的时候会抛出`KeyError`异常
>
> 此时捕获异常，并且设置`Cookie`即可；
>
> 再次刷新的时候，因为这里已经给出了`Cookie`值，则不会引发异常，会在页面中展示获取到的加盐`Cookie`

Session

> 虽然说有了`Cookie`之后，我们把一些信息保存在客户端浏览器中，可以保持用户在访问站点时的状态，但是也存在一定的安全隐患，`Cookie`值被曝露，`Cookie`值被他人篡改，等等。我们将换一种更健全的方式，也就是接下来要说的`Session`。
>
> `Session`在网络中，又称会话控制，简称会话。用以存储用户访问站点时所需的信息及配置属性。当用户在我们的`Web`服务中跳转时，存储在`Session`中的数据不会丢失，可以一直在整个会话过程中存活。
>
> 在`django`中，默认的`Session`存储在数据库中`session`表里。默认有效期为**两个星期**。

session创建流程

1. 客户端访问服务端，服务端为每一个客户端返回一个唯一的`sessionid`，比如`xxx`。
2. 客户端需要保持某些状态，比如维持登陆。那么服务端会构造一个`{sessionid: xxx }`类似这样的字典数据加到`Cookie`中发送给用户。注意此时，只是一个随机字符串，返回给客户端的内容并不会像之前一样包含实际数据。

3. 服务端在后台把返回给客户端的`xxx`字符串作为`key`值，对应需要保存的服务端数据为一个新的字典，存储在服务器上，例如：`{xxx : {id:1}}`

> 之后的一些客户端数据获取，都是通过获取客户端向服务端发起的`HttpRequest`请求中里`Cookie`中的`sessionid`之后，再用该`sessionid`从服务端的`Session`数据中调取该客户端存储的`Session`数据

> **注意**：补充说明，默认存储在数据库的`Session`数据，是通过`base64`编码的，我们可以通过`Python`的`base64`模块下的`b64decode()`解码得到原始数据

> 整个过程结束之后：客户端浏览器存储的其实也只是一个**识别会话**的随机字符串`(xxx)`
>
> 而服务器中是通过这个随机的字符串`(xxx:value)`进行真正的存储

> `Session`的使用必须在`Settings`配置下

```
INSTALLED_APPS = (
    ...
    'django.contrib.sessions',
    ...
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    ...
)
```

> 当`settings.py`中`SessionMiddleware`激活后
>
> 在视图函数的参数`request`接收到的客户端发来的`HttpResquest`请求对象中都会含有一个`session`属性
>
> 这个属性和之前所讨论的`Cookie`类似，是一个类字典对象，首先支持如下常用字典内置属性

获取Session

- `session_data = request.session.get(Key)`
- `session_data = request.session[Key]`

  > 在`Session`中获取对应值，`get`方法获取时，如不存在该`Key`值，不会引发异常，返回`None`
  >
  > 而第二种直接通过字典获取，如`Key`值不存在，引发`KeyErro`

删除Session

- `del request.seesion[Key]`

  > 删除对应session，`Key`值不存在时，引发`KeyError`

- `request.session.clear()`

  > 清空`Session`中的所有数据。这里客户端还会保留`sessionid`
  >
  > 只不过在服务端`sessionid`对应的数据没有了。

- `request.session.flush()`

> 直接删除当前客户端的的Seesion数据。这里不光服务端sessionid对应的数据没有了，客户端的sessionid也会被删除

设置有效期

- `request.session.set_expiry(value)`:

  > 设置Session的有效时间。
  >
  > > `value`：有效时间。
  > >
  > > **为整数时**：将在value为秒单位之后过期
  > >
  > > **为0时**：将在用户关闭浏览器之后过期。
  > >
  > > **为None时**：使用全局过期的设置，默认为两个星期，14天。
  > >
  > > **为datetime时**：在这个指定时间后过期。

- `request.session.get_expiry_age()`

  > 返回距离过期还剩下的秒数。

- `request.session.clear_expired()`

  > 清除过期的`Session`会话。

> 编写一个简单的视图函数来玩耍`Session`吧

```
from django.shortcuts import render,HttpResponse
import datetime
def set_session(request):
    if request.session.get('test_id'):
        session_data = request.session.get('test_id')# 用户拿到的的session随机字符串
        session_key = request.session.session_key # 获取客户端浏览器中的SessionID值
        session_expire = request.session.get_expiry_age()
        now = datetime.datetime.now()
        expire_time = now + datetime.timedelta(seconds=session_expire)
        response = '<div>SessionID : %s</div>' % session_key + \
                   '<div>Session : %s</div>' % session_data + \
                   '<div>ExpireTime : %s</div>' % expire_time
        return HttpResponse(response)
    else:
        request.session['test_id'] = 'TEST'
        request.session.set_expiry(None)
        return HttpResponse('已设置好Session')
```

> 用户在第一次访问时，会走`else`分支，此时还没有任何服务端的`Session`及客户端的`Cookie`值设定
>
> 那么我们会通过`request.session[Key]`的方式来设置一个`Session`值，值为`TEST`
>
> 当用户第二次访问时将展示出所设置好的Session值及在客户端浏览器中存储的sessionid

> 在编写一个删除`Session`的视图函数吧

```python
def delete_session(request):
    if request.session.get('test_id'):
        del request.session['test_id']
        return HttpResponse('Session被删了')
    else:
        return HttpResponse('目前没有任何需要删除的session')
```

> 这里温柔的使用`del request.session[Key]`的方式来进行`Session`的删除
>
> 如果存在对应`test_id`的`Session`值则删除，反之返回一个字符串

Session删除总结

> 使用的是`del`的针对性删除方式，这样不会将整个客户端的`session`删除掉
>
> 使用`request.session.clear()`，只是清空了服务端`Session`中的数据，但是客户端的`Cookie`中还会保存`sessionid`，只不过这个值对应的字符串所对应的用户数据是一个空
>
> 使用`request.session.flush()`，那么客户端`Cookie`中保存的`sessionid`首先会被删除，其次服务端通过`sessionid`值保存的用户数据也会被全部删除。