

Cookie和Session

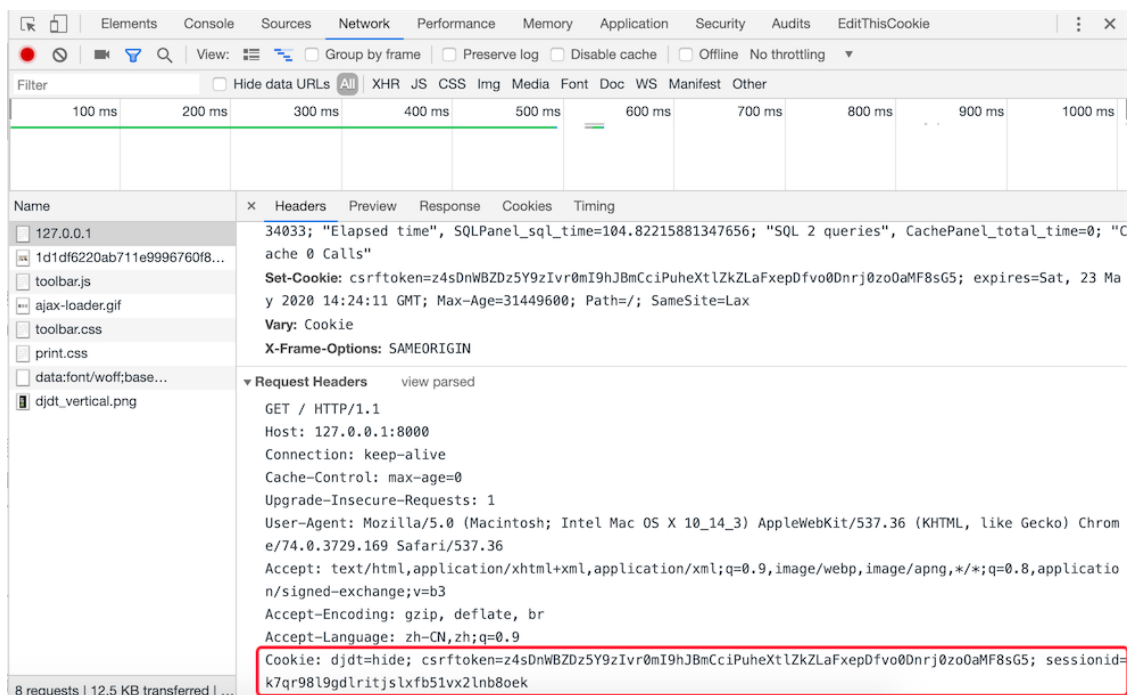
实现用户跟踪

如今，一个网站如果不通过某种方式记住你是谁以及你之前在网站的活动情况，失去的就是网站的可用性和便利性，继而很有可能导致网站用户的流失，所以记住一个用户（更专业的说法叫**用户跟踪**）对绝大多数Web应用来说都是必需的功能。

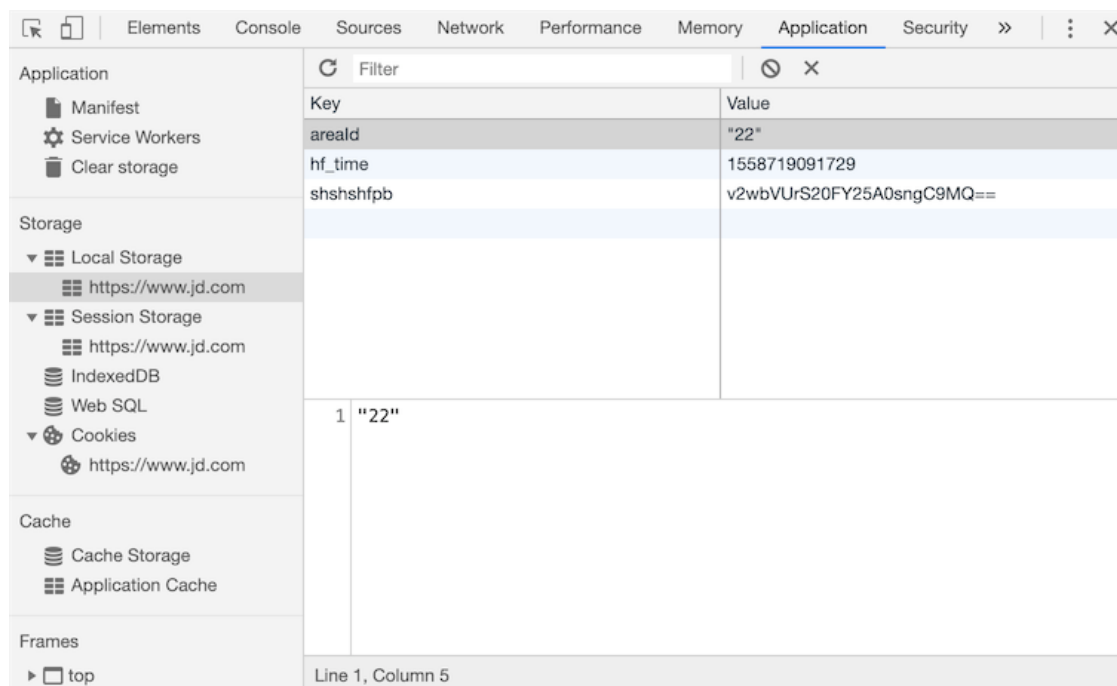
在服务器端，我们想记住一个用户最简单的办法就是创建一个对象，通过这个对象就可以把用户相关的信息都保存起来，这个对象就是我们常说的session（用户会话对象）。那么问题来了，HTTP本身是一个无连接（每次请求和响应的过程中，服务器一旦完成对客户端请求的响应之后就断开连接）、无状态（客户端再次发起对服务器的请求时，服务器无法得知这个客户端之前的任何信息）的协议，即便服务器通过session对象保留了用户数据，还得通过某种方式来确定当前的请求与之前保存过的哪一个session是有关联的。相信很多人都能想到，我们可以给每个session对象分配一个全局唯一的标识符来识别session对象，我们姑且称之为sessionid，每次客户端发起请求时，只要携带上这个sessionid，就有办法找到与之对应的session对象，从而实现在两次请求之间记住该用户的信息，也就是我们之前说的用户跟踪。

要让客户端记住并在每次请求时带上sessionid又有以下几种做法：

1. URL重写。所谓URL重写就是在URL中携带sessionid，例如：
`http://www.example.com/index.html?sessionid=123456`，服务器通过获取sessionid参数的值来取到与之对应的session对象。
2. 隐藏域（隐式表单域）。在提交表单的时候，可以通过在表单中设置隐藏域向服务器发送额外的数据。例如：`<input type="hidden" name="sessionid" value="123456">`。
3. 本地存储。现在的浏览器都支持多种本地存储方案，包括：cookie、localStorage、sessionStorage、IndexedDB等。在这些方案中，cookie是历史最为悠久也是被诟病得最多的一种方案，也是我们接下来首先为大家讲解的一种方案。简单的说，cookie是一种以键值对方式保存在浏览器临时文件中的数据，每次请求时，请求头中会携带本站点的cookie到服务器，那么只要将sessionid写入cookie，下次请求时服务器只要读取请求头中的cookie就能够获得这个sessionid，如下图所示。



在HTML5时代要，除了cookie，还可以使用新的本地存储API来保存数据，就是刚才提到的localStorage、sessionStorage、IndexedDB等技术，如下图所示。



Django框架对session的支持

在创建Django项目时，默认的配置文件中 `settings.py` 文件中已经激活了一个名为 `SessionMiddleware` 的中间件（关于中间件的知识我们在下一个章节做详细的讲解，这里只需要知道它的存在即可），因为这个中间件的存在，我们可以直接通过请求对象的 `session` 属性来操作会话对象。`session` 属性是一个像字典一样可以读写数据的容器对象，因此我们可以使用“键值对”的方式来保留用户数据。与此同时，`SessionMiddleware` 中间件还封装了对cookie的操作，在cookie中保存了 `sessionid`，就如同我们之前描述的那样。

在默认情况下，Django将session的数据序列化后保存在关系型数据库中，在Django 1.6以后的版本中，默认的序列化数据的方式是JSON序列化，而在此之前一直使用Pickle序列化。JSON序列化和Pickle序列化的差别在于前者将对象序列化为字符串（字符形式），而后者将对象序列化为字节串（二进制形式），因为安全方面的原因，JSON序列化成为了目前Django框架默认序列化数据的方式，这就要求在我们保存在session中的数据必须是能够JSON序列化的，否则就会引发异常。还有一点需要说明的是，使用关系型数据库保存session中的数据在大多数时候并不是最好的选择，因为数据库可能会承受巨大的压力而成为系统性能的瓶颈，在后面的章节中我们会告诉大家如何将session的数据保存到缓存服务中。

我们继续完善之前的投票应用，前一个章节中我们实现了用户的登录和注册，下面我们首先完善登录时对验证码的检查。

```
def get_captcha(request):  
    """验证码"""  
    captcha_text = random_captcha_text()  
    request.session['captcha'] = captcha_text  
    image_data = Captcha.instance().generate(captcha_text)  
    return HttpResponse(image_data, content_type='image/png')
```

注意上面代码中的第4行，我们将随机生成的验证码字符串保存到session中，稍后用户登录时，我们要将保存在session中的验证码字符串和用户输入的验证码字符串进行比对，如果用户输入了正确的验证码才能够执行后续的登录流程，代码如下所示。

```
def login(request: HttpRequest):
```


session_key	session_data	expire_date
c9g2gt5cxo0k2evykgpejhc5ae7bfpl	Mml4YzViYjJhOGMyMDJkY2M5Yzg3...	2019-05-25 23:16:13.898522

其中，第1列就是浏览器cookie中保存的sessionid；第2列是经过BASE64编码后的session中的数据，如果使用Python的 `base64` 对其进行解码，解码的过程和结果如下所示。

```
>>> import base64
>>>
base64.b64decode('Mml4YzViYjJhOGMyMDJkY2M5Yzg3ZWlyZGVlZmUzYmYxNzd1NDdmZjp7ImNhCH
RjaGEiOiJzS3d0Iiwibm8iOjEsInVzZXJlIjoiaWJja2Zydwvkn0=')
'2b8c5bb2a8c202dcc9c87eb2debfe3bf177e47ff:
{"captcha":"skwt","no":1,"username":"jackfrued"}'
```

第3列是session的过期时间，session过期后浏览器保存的cookie中的sessionid就会失效，但是数据库中的这条对应的记录仍然会存在，如果想清除过期的数据，可以使用下面的命令。

```
python manage.py clearsessions
```

Django框架默认的session过期时间为两周（1209600秒），如果想修改这个时间，可以在项目的配置文件中添加如下所示的代码。

```
# 配置会话的超时时间为1天（86400秒）
SESSION_COOKIE_AGE = 86400
```

有很多对安全性要求较高的应用都必须在关闭浏览器窗口时让会话过期，不再保留用户的任何信息，如果希望在关闭浏览器窗口时就让会话过期（cookie中的sessionid失效），可以加入如下所示的配置。

```
# 设置为True在关闭浏览器窗口时session就过期
SESSION_EXPIRE_AT_BROWSER_CLOSE = True
```

如果不希望将session的数据保存在数据库中，可以将其放入缓存中，对应的配置如下所示，缓存的配置和使用我们在后面讲解。

```
# 配置将会话对象放到缓存中存储
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
# 配置使用哪一组缓存来保存会话
SESSION_CACHE_ALIAS = 'default'
```

如果要修改session数据默认的序列化方式，可以将默认的 `JSONSerializer` 修改为 `PickleSerializer`。

```
SESSION_SERIALIZER = 'django.contrib.sessions.serializers.PickleSerializer'
```

在视图函数中读写cookie

Django封装的 `HttpRequest` 和 `HttpResponse` 对象分别提供了读写cookie的操作。

`HttpRequest`封装的属性和方法：

1. `COOKIES` 属性 - 该属性包含了HTTP请求携带的所有cookie。
2. `get_signed_cookie` 方法 - 获取带签名的cookie，如果签名验证失败，会产生 `BadSignature` 异常。

HttpResponse封装的方法：

1. `set_cookie` 方法 - 该方法可以设置一组键值对并将其最终将写入浏览器。
2. `set_signed_cookie` 方法 - 跟上面的方法作用相似，但是会对cookie进行签名来达到防篡改的作用。因为如果篡改了cookie中的数据，在不知道**密钥**和**盐**的情况下是无法生成有效的签名，这样服务器在读取cookie时会发现数据与签名不一致从而产生 `BadSignature` 异常。需要说明的是，这里所说的密钥就是我们在Django项目配置文件中指定的 `SECRET_KEY`，而盐是程序中设定的一个字符串，你愿意设定为什么都可以，只要是一个有效的字符串。

上面提到的方法，如果不清楚它们的具体用法，可以自己查阅一下Django的[官方文档](#)，没有什么资料比官方文档能够更清楚的告诉你这些方法到底如何使用。

刚才我们说过了，激活 `SessionMiddleware` 之后，每个 `HttpRequest` 对象都会绑定一个 `session` 属性，它是一个类似字典的对象，除了保存用户数据之外还提供了检测浏览器是否支持cookie的方法，包括：

1. `set_test_cookie` 方法 - 设置用于测试的cookie。
2. `test_cookie_worked` 方法 - 检测测试cookie是否工作。
3. `delete_test_cookie` 方法 - 删除用于测试的cookie。
4. `set_expiry` 方法 - 设置会话的过期时间。
5. `get_expire_age` / `get_expire_date` 方法 - 获取会话的过期时间。
6. `clear_expired` 方法 - 清理过期的会话。

下面是在执行登录之前检查浏览器是否支持cookie的代码。

```
def login(request):
    if request.method == 'POST':
        if request.session.test_cookie_worked():
            request.session.delete_test_cookie()
            # Add your code to perform login process here
        else:
            return HttpResponse("Please enable cookies and try again.")
    request.session.set_test_cookie()
    return render_to_response('login.html')
```

Cookie的替代品

之前我们说过了，cookie的名声一直都不怎么好，当然我们在实际开发中是不会在cookie中保存用户的敏感信息（如用户的密码、信用卡的账号等）的，而且保存在cookie中的数据一般也会做好编码和签名的工作。即便如此，HTML5中还是给出了用于替代cookie的技术方案，其中使用得最为广泛的的就是 `localStorage` 和 `sessionStorage`，相信从名字上你就能听出二者的差别，存储在 `localStorage` 的数据可以长期保留；而存储在 `sessionStorage` 的数据会在浏览器关闭时会被清除。关于这些cookie替代品的用法，建议大家查阅[MDN](#)来进行了解。