# cookie和session

【python】Flask之session使用_python flask session-CSDN博客

### 1.cookie

cookie：在网站中，http请求是无状态的。也就是说即使第一次和服务器连接后并且登录成功后，第二次请求服务器依然不能知道当前请求是哪个用户。cookie的出现就是为了解决这个问题，第一次登录后服务器返回一些数据（cookie）给浏览器，然后浏览器保存在本地，当该用户发送第二次请求的时候，就会自动的把上次请求存储的cookie数据自动的携带给服务器，服务器通过浏览器携带的数据就能判断当前用户是哪个了。cookie存储的数据量有限，不同的浏览器有不同的存储大小，但一般不超过4KB。因此使用cookie只能存储一些小量的数据。

### 2.session

session: session和cookie的作用有点类似，都是为了存储用户相关的信息。不同的是，cookie是存储在本地浏览器，session是一个思路、一个概念、一个服务器存储授权信息的解决方案，不同的服务器，不同的框架，不同的语言有不同的实现。虽然实现不一样，但是他们的目的都是服务器为了方便存储数据的。session的出现，是为了解决cookie存储数据不安全的问题的。

### 3.cookie和session结合使用

cookie和session结合使用：web开发发展至今，cookie和session的使用已经出现了一些非常成熟的方案。在如今的市场或者企业里，一般有两种存储方式：

- 存储在服务端：通过cookie存储一个session_id，然后具体的数据则是保存在session中。如果用户已经登录，则服务器会在cookie中保存一个session_id，下次再次请求的时候，会把该session_id携带上来，服务器根据session_id在session库中获取用户的session数据。就能知道该用户到底是谁，以及之前保存的一些状态信息。这种专业术语叫做server side session。存储在服务器的数据会更加的安全，不容易被窃取。但存储在服务器也有一定的弊端，就是会占用服务器的资源，但现在服务器已经发展至今，一些session信息还是绰绰有余的。

- 将session数据加密，然后存储在cookie中。这种专业术语叫做client side session。flask采用的就是这种方式，但是也可以替换成其他形式。

### 4.flask里面的session机制

#### 4.1 session的作用

由于http协议是一个无状态的协议，但网站基本上有登录使用的功能，这要求有状态管理，而session机制实现的就是这个功能

session基于cookie实现， 保存在服务端的键值对(形式：{随机字符串：'xxxxxx'}), 同时在浏览器中的cookie中也对应一相同的随机字符串，用来再次请求的时候验证

#### 4.2 实现的原理

用户第一次请求后，将产生的状态信息保存在session中，可以把session当做一个容器，保存了正在使用的所有用户的状态信息，这段状态信息分配一个唯一的标识符用来标识用户的身份，将其保存在响应对象的cookie中 当第二次请求时，解析cookie中的标识符，拿到标识符后去session找到对应的用户的信息

#### 4.3 使用案例

##### 4.3.1 配置SECRET_KEY

flask的session是通过加密之后放到cookie中，有加密就有密钥用于解密，所以只要用到了flask的session模块就一定要配置"SECRET_KEY"这个全局宏 一般设置为24位的字符，配置方法一般有两种

#### 4.3.2 配置方法一

1.新建一个config.py的文件配置secret_key

config.py

```
SECRET_KEY = 'XXXXXXXXX'
```

2.在主运行文件里面添加config文件里面的内容 main.py

```
#encoding: utf-8
from flask import Flask,session
import config
app = Flask(__name__)
```

#### 4.3.3 配置方法二

直接在主运行文件里面配置。配置config的时候也是和操作字典是一样的 main.py

```
encoding: utf-8

from flask import Flask,session

app = Flask(__name__)
app.config['SECRET_KEY'] = 'XXXXX' 或者随机数（os.urandom(24)）
# 或者
#app.secret_key = 'why would I tell you my secret key?'
# key值可以使用随机数，或者自定义
```

### 4.4 操作session

#### 4.4.1 设置session

```
from flask import Flask,session
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)

# 设置session
 @app.route('/')
def set():
     session['username'] = 'liefyuan'
 # 设置“字典”键值对（正式开发时候，值需要session.get('user')获取）
     return 'success'

 if __name__ == '__main__':
     app.run()
```

#### 4.4.2 读取session

因为session就像字典一样所以，操作它的时候有两种方法：

1.result = session['key']： 如果内容不存在，将会报异常 2.result = session.get('key')：如果内容不存在，将返回None(推荐)

```
from flask import Flask,session
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)

 # 读取session
 @app.route('/get')
 def get():
     # session['username']
     # session.get('username')
     return session.get('username')

 if __name__ == '__main__':
     app.run()
```

#### 4.4.3 删除session

```
#encoding: utf-8
from flask import Flask,session
import os
app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)

# 删除session
@app.route('/delete/')
def delete():
     print session.get('username')
     session.pop('username',None) 或者 session['username'] = False
     print session.get('username')
     return 'success'
if __name__ == '__main__':
       app.run()
```

#### 4.4.4 清除session中所有数据

```
#encoding: utf-8

from flask import Flask,session
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = os.urandom(24)

# 清除session中所有数据
@app.route('/clear')
def clear():
     print session.get('username')
     # 清除session中所有数据
     session.clear
     print session.get('username')
     return 'success'

if __name__ == '__main__':
     app.run()
```