书和作者后端代码：

```
from flask import Flask,render_template,redirect,url_for
from flask import request
from flask_wtf import FlaskForm
from wtforms import StringField,SubmitField
from wtforms.validators import DataRequired
from flask_sqlalchemy import SQLAlchemy
app = Flask(__name__)

class Config(object):
    SECRET_KEY = "fADFASFAF"
    # 设置数据库的uri地址:SQLALCHEMY_DATABASE_URI:这个是固定的写法
    # mysql:表示数据库的类型
    # root:表示用户名
    # mysql:表示密码
    # 127.0.0.1:3306:数据库的ip地址和端口号
    # db:表示数据库的名字
    SQLALCHEMY_DATABASE_URI = "mysql://root:mysql@127.0.0.1:3306/auth_book_11"
    # 信号追踪,当我们的数据库进行更改的时候，系统会发送信号,如果不需要那么可以设置为flase
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    # 打印出来原始的数据
    app.config['SQLALCHEMY_ECHO'] = True

app.config.from_object(Config)
# 初始化数据库的对象
db = SQLAlchemy(app)
# 作者
class Author(db.Model):
    __tablename__ = "authors"
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    # 在一对多的关系中需要在一的一方设置关系
    books = db.relationship("Book",backref = "author")

# 书
class Book(db.Model):
```

```
    __tablename__ = "books"
    id = db.Column(db.Integer,primary_key=True)
    name = db.Column(db.String(128))
    # 在一对多的一方设置外键
    author_id = db.Column(db.Integer,db.ForeignKey("authors.id"))

# app.config["SECRET_KEY"] = "fADFASFAF"
class AuthorBookForm(FlaskForm):
    author_name = StringField(label="作者",validators=
[DataRequired("请输入作者的名字")])
    book_name = StringField(label="书名",validators=[DataRequired("请
输入书的名字")])
    submit = SubmitField("添加")


@app.route("/delete_author/<author_id>")
def delete_author(author_id):
    # 1 在删除作者之前，那么需要先找到作者
    author = Author.query.get(author_id)
    # 2 找到作者之后，那么不要着急先删除作者，因为作者和书是主外键关系，直接
删除作者是删除不掉
    Book.query.filter(Book.author_id == author.id).delete()
    # 3 找到作者之后，在进行删除书，最后才删除作者
    db.session.delete(author)
    db.session.commit()

    return redirect(url_for("index"))

# 删除作者的书
@app.route("/delete_book/<book_id>")
def delete_book(book_id):
    # 根据id查询到书
    # try:
    book = Book.query.get(book_id)


    db.session.delete(book)
    db.session.commit()
    # except Exception as e:
    #     print("")

    # if not book:
```

```
    #     return "当前的书不存在"
    # try:

    # except Exception as e:
    #     #  如果删除错误进行回滚
    #     db.session.rollback()
    return redirect(url_for("index"))


@app.route("/" ,methods=["GET","POST"])
def index():
    form = AuthorBookForm()
    # 提交并且验证
    if form.validate_on_submit():
        # 获取到作者的名字
        # 上面这种是普通的表单获取值
        # author_name = request.form.get("author_name")
        # book_name = request.form.get("book_name")
        # 这个地方是wtf的语法，只有在wtf里面才会这样获取值
        author_name = form.author_name.data
        book_name = form.book_name.data

        author = Author()
        # 获取到用户输入的名字之后，对数据库进行赋值
        author.name = author_name
        db.session.add(author)
        db.session.commit()

        book = Book()
        book.name = book_name
        book.author_id = author.id
        db.session.add(book)
        db.session.commit()


    # 获取到所有的作者
    authors = Author.query.all()
    return render_template("code_22_author_book.html",form =
form,authors = authors)

if __name__ == '__main__':
    db.drop_all()
```

```
    db.create_all()
    # 生成数据
    au1 = Author(name='老王')
    au2 = Author(name='老尹')
    au3 = Author(name='老刘')
    # 把数据提交给用户会话
    db.session.add_all([au1, au2, au3])
    # 提交会话
    db.session.commit()
    bk1 = Book(name='老王回忆录', author_id=au1.id)
    bk2 = Book(name='我读书少，你别骗我', author_id=au1.id)
    bk3 = Book(name='如何才能让自己更骚', author_id=au2.id)
    bk4 = Book(name='怎样征服美丽少女', author_id=au3.id)
    bk5 = Book(name='如何征服英俊少男', author_id=au3.id)
    # 把数据提交给用户会话
    db.session.add_all([bk1, bk2, bk3, bk4, bk5])
    # 提交会话
    db.session.commit()
    app.run(debug=True)
```

书和作者前端代码：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<form method="post">
    {{ form.csrf_token }}

    {{ form.author_name.label }}
    <p>{{ form.author_name }}</p>

     {{ form.book_name.label }}
    <p>{{ form.book_name }}</p>
```

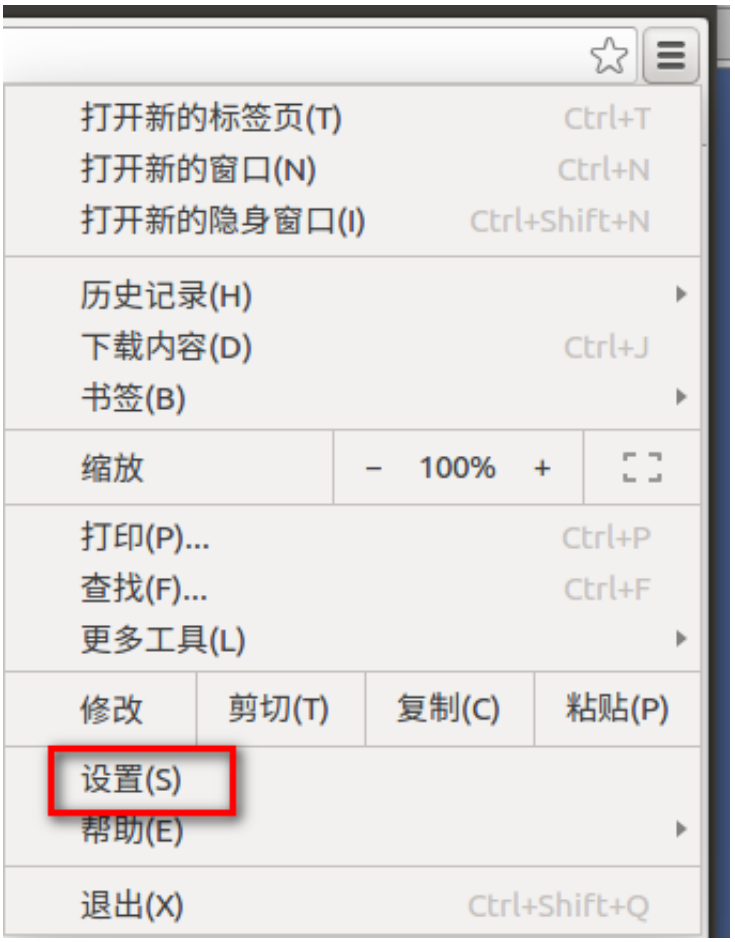
```
16
17
18      {{ form.submit }}
19
20     <h1>获取所有的作者和书：</h1>
21     {#迭代所有的作者#}
22     {% for author in authors %}
23         <li>{{ author.name }}<a href="/delete_author/{{ author.id }}">
删除</a></li>
24         <ul>
25            {% for book in author.books %}
26              <li>{{ book.name }}<a href="/delete_book/{{ book.id }}">
删除</a></li>
27            {% endfor %}
28         </ul>
29     {% endfor %}
30 </form>
31
32
33
34 </body>
35 </html>
```

设置cookie

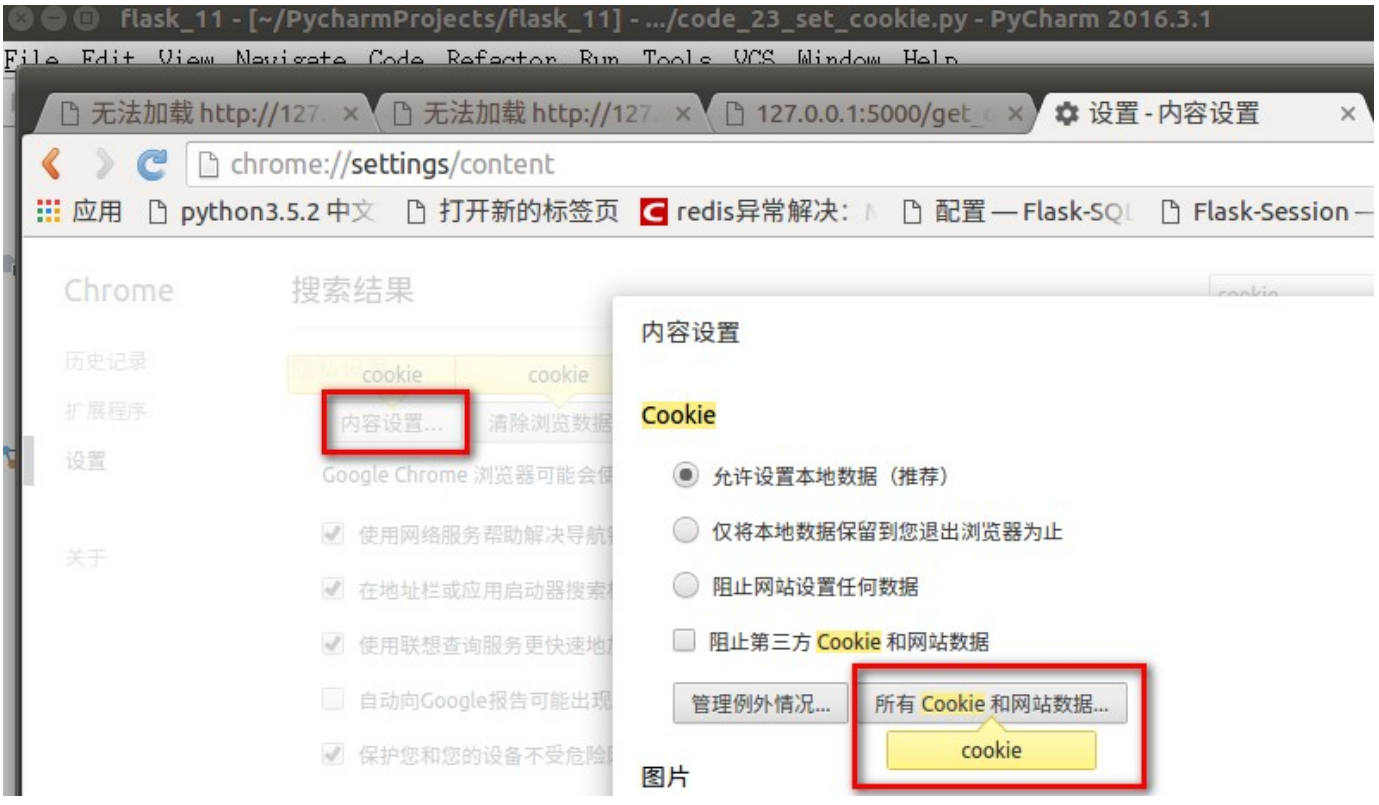
Cookie：指某些网站为了辨别用户身份、进行会话跟踪而储存在用户本地的数据

Cookie是存储在浏览器中的一段纯文本信息，建议不要存储敏感信息如密码

在做cookie的时候，需要先清除浏览器的cookie缓存，按照如下步骤清除浏览器缓存


打开新的标签页(T) Ctrl+T
打开新的窗口(N) Ctrl+N
打开新的隐身窗口(I) Ctrl+Shift+N
历史记录(H)
下载内容(D) Ctrl+J
书签(B)
缩放 - 100% +
打印(P)... Ctrl+P
查找(F)... Ctrl+F
更多工具(L)
修改 剪切(T) 复制(C) 粘贴(P)
设置(S)
帮助(E)
退出(X) Ctrl+Shift+Q

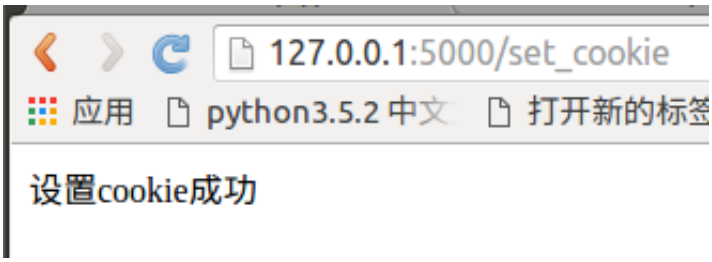

搜索结果
cookie
隐私设置
内容设置...
清除浏览数据...
cookie
cookie
了解详情


flask_11 - [~/PycharmProjects/flask_11] - .../code_23_set_cookie.py - PyCharm 2016.3.1
File Edit View Navigate Code Refactor Run Tools VCS Window Help
无法加载 http://127
无法加载 http://127
127.0.0.1:5000/get_c
设置 - 内容设置
chrome://settings/content
应用
python3.5.2 中文
打开新的标签页
redis异常解决:
配置 — Flask-SQL
Flask-Session —
Chrome
搜索结果
历史记录
扩展程序
设置
关于
cookie
cookie
内容设置...
清除浏览数据
Google Chrome 浏览器可能会
使用网络服务帮助解决导航
在地址栏或应用启动器搜索
使用联想查询服务更快速地
自动向Google报告可能出现
保护您和您的设备不受危险
内容设置
Cookie
允许设置本地数据（推荐）
仅将本地数据保留到您退出浏览器为止
阻止网站设置任何数据
阻止第三方 Cookie 和网站数据
管理例外情况...
所有 Cookie 和网站数据...
cookie
图片


```python
from flask import Flask,make_response
from flask import request

app = Flask(__name__)

@app.route("/set_cookie")
def set_cookie():
    # 获取到服务端返回过来的信息
    resp = make_response("设置cookie成功")
    # cookie的值是通过键值对的方式进行设置:
    resp.set_cookie("name","itheima",max_age= 7200)
    resp.set_cookie("city","sz")
    return resp


@app.route("/get_cookie")
def get_cookie():
    # 获取到所有的cookie
    name = request.cookies.get("name")
    return "获取到cookie的值 = " + name

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

运行效果：

设置cookie的值是itheima



获取到cookie的值是itheima

Session

- 对于敏感、重要的信息，建议要存储在服务器端，不能存储在浏览器中，如用户名、余额、等级、验证码等信息

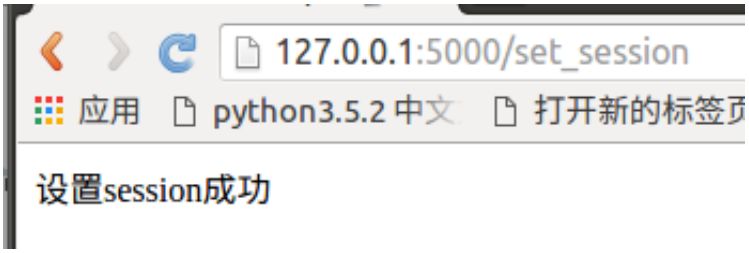
千万千万记得，在flask程序里面，只要是使用到了session，就必须设置SECRET_KEY:

app.config["SECRET_KEY"] = "fadffa"

```
from flask import Flask,session

app = Flask(__name__)
# 只要是在flask里面用到了session就必须设置SECRET_KEY
app.config["SECRET_KEY"] = "fadffa"

@app.route("/set_session")
def set_session():
    session["name"] = "itcast"
    return "设置session成功"

@app.route("/get_session")
def get_session():
    name = session.get("name")
```

```
15      return name
16
17  if __name__ == '__main__':
18      app.run(debug=True)
```

运行效果



防止 CSRF 攻击

步骤

1. 在客户端向后端请求界面数据的时候，后端会往响应中的 cookie 中设置 csrf_token 的值
2. 在 Form 表单中添加一个隐藏的的字段，值也是 csrf_token
3. 在用户点击提交的时候，会带上这两个值向后台发起请求
4. 后端接受到请求，以会以下几件事件：
    - 从 cookie中取出 csrf_token
    - 从 表单数据中取出来隐藏的 csrf_token 的值
    - 进行对比
5. 如果比较之后两值一样，那么代表是正常的请求，如果没取到或者比较不一样，代表不是正常的请求，不执行下一步操作