数据库PJ1汇报文档

本文档为2024年春季数据库引论H课程的PJ1汇报

姓名：谢志康

学号：22307110187

实验验收时间：2024.5.10

目录：

1. **需求分析**（简要介绍整体PJ框架）
2. **概要设计**
3. **详细设计**
4. **部署方法**（该项目如何在您电脑上部署运行）
5. **Appendix**：Flask框架学习报告

正式内容：

1. **需求分析**（简要介绍整体PJ框架）——

项目主题是“校园二手物品线上交易平台”。旨在为校内学生提供便捷、安全的线上二手物品交易平台。

1. **概要设计**——

实现语言为python（后端），html、Javascript（前端）

后端为app.py文件，前端包括templates中所有的html界面（都是自己写的）和static中的css和js文件（css都是自己写的）

后端框架使用flask（appendix部分是flask的详细学习报告）

数据库使用MySQL关系型数据库

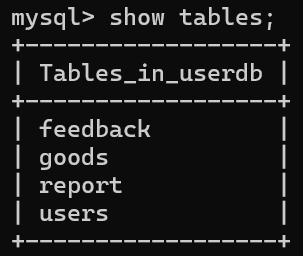
**主要功能**：

1. 用户信息管理：包括：注册账号、登录账号、查看个人信息、修改个人信息、查看本账号交易记录（包括曾经买过什么商品、发布（卖出）了什么商品）。
2. 商品信息管理：包括：用户上传要发布商品的信息，搜索想要商品的信息，下架不想卖（或已发售）的商品。
3. 举报功能（对于违反平台规章的用户，其余用户可以举报ta）
4. 意见反馈功能（用户对该平台的进一步发展有什么建议可以提供意见反馈）
5. 商品推荐界面（在index以及index\_after\_login界面，平台会根据商品库存信息随机推送八件商品显示在首页）
6. 商品分类界面：平台主要经营二手书籍、日用品、交通、体育物品、数码产品 以及跑腿（帮带早餐、帮拿外卖、帮取快递）等等功能。用户可以查看具体商品分类界面以了解平台主要经营什么商品。
7. 商品查询功能：支持模糊查询，用户输入想要的商品关键词，平台根据商品管理数据表返回符合要求的商品（对每个商品的商品名称和商品描述字段进行比较查询），用户可以查看这些商品的详细信息，并与卖家取得联系进行交易。
8. 支持图片上传功能，上传采用html支持的multi-file格式，商品发布可以附上商品图片，用户也可以更换自己的头像。
9. **详细设计**——

**数据库及数据表**部分：整体使用一个数据库userdb：

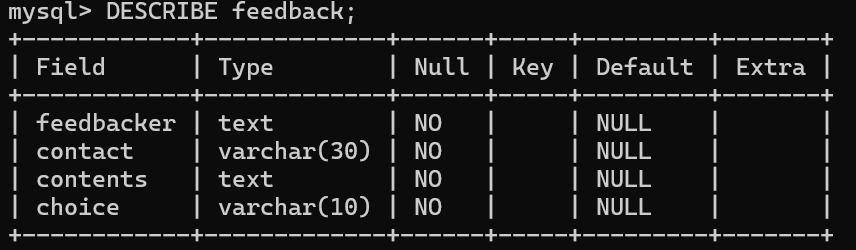


一共有四个表：分别是用户反馈表，商品信息表，举报内容表，用户信息表

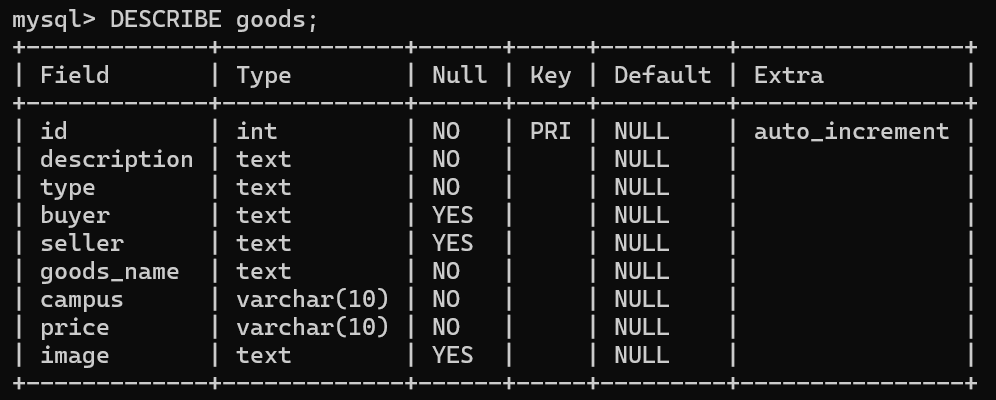


其各自对应的具体表属性如下：

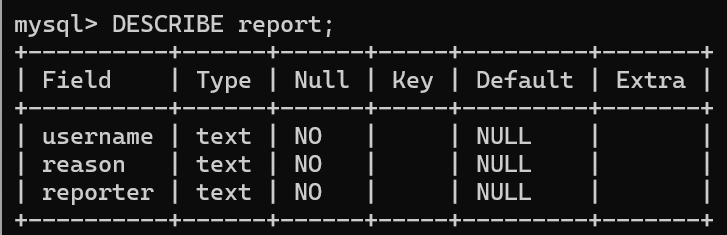
**feedback table**——反馈者（feedbacker），联系方式（contact），反馈内容（content），以及联系方式选择（choice）（限制选择email、电话、微信方式联系）



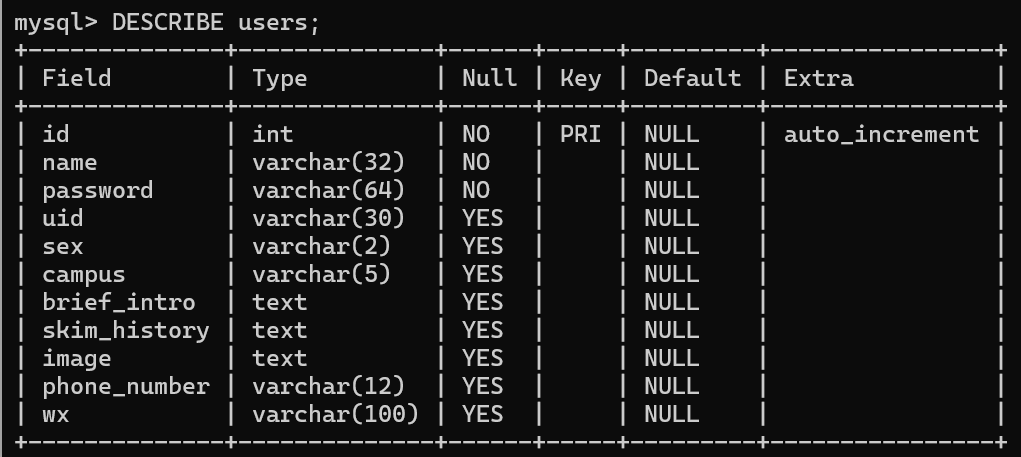
**goods table**——商品id（唯一标识商品，作为主键），商品描述（description），商品类型（type）（包括书籍、数码产品等……是选项框形式），买家（buyer），卖家（seller），商品名称（goods\_name），卖家所在校区（campus），商品价格（price），商品图片（image）



**report table**——举报对象用户名（username），举报理由（reason），举报者（reporter）



**users table**——用户id（唯一标识用户身份信息，作为主键），用户名（name），密码（password），学号（uid），性别（sex），校区（campus），简要个人介绍（brief\_intro），商品浏览历史（skim\_history），头像（image），手机号（phone\_number），微信（wx）



**主要函数功能实现**：（只介绍原理，不粘代码了，不然太多了）

index和index\_after

为登陆前以及登陆后的首页界面（登录后主页有显示更多操作，譬如，查看个人信息，发布商品等等）在首页，我进行一些商品推荐。方法是从数据库的goods表（管理商品信息）中随机抽取八个商品，在主页上显示出来，作为今日推荐。

register

为用户注册界面，用户在此新注册个人信息。使用post方法，用户注册信息发起request，后端取得用户的用户名以及密码，并进行安全性检验，如果符合要求（指用户名不与数据库users表中已存在的用户名相同，因为我以用户名作为主键唯一标识用户，不允许重复），则加入数据库的users表中。若不符合要求，使用flask的flash方式在前端提醒用户“该用户名太热门了需要换一个”。

Login

登录界面，与users表中用户信息匹配，用户名以及密码分别对应正确则登录成功。将用户登录信息存储在session中，以便后续跟踪确定用户行为。

Logout

登出界面，销毁session存储的用户信息，重定向到index（登陆前）

personal\_info

查看个人信息界面，从session中取得当前使用的用户的username，sql查询找到该用户的信息并显示。

mend\_info

修改个人信息界面，sql的update方式更新。实际上有很多需要考虑，比如，若用户修改了username，则新username不能与数据库中已存在的username相同，且session的信息需要更换，因此，在mend\_info之后我要求用户重新登录，更新session。

personal\_trade

查看个人交易信息界面，从session中取得username，在goods表中搜索商品发布者为该用户的商品信息，显示出来。

report

举报界面，post获取举报信息，插入report表中（安全性检验，被举报者的用户名必须存在）。

feedback

反馈界面，与report逻辑相同，使用feedback表。

Imageupload

头像上传界面，用户可以更换自己的头像。采用multi-file表单形式上传，文件自动保存到服务器端本地。Image数据加入到对应用户的个人信息表中。

Goodsupload

商品上传界面，用户上传商品，将商品信息post回后端（包括商品图片）一同存入goods表。

Goodsdownload

商品下架界面，用户下架自己的商品，delete from goods的方法实现，匹配该用户与该商品名称并进行删除。

Search

商品查询界面，支持模糊搜索（sql的like方式），从goods表中搜索出所有商品名称或商品描述带有关键词的商品信息，传回用户端显示出来。

Outcome

Search的结果的显示方式，这个函数与前端的JavaScript交互，为每个搜索结果动态生成一个url，用户点击搜索结果即可跳转到对应的商品详情界面。

Content

展示商品详细内容页面。商品id为主键，以此获得该商品详细信息，返回前端显示。

Books dailyexpense food sport transport electronic

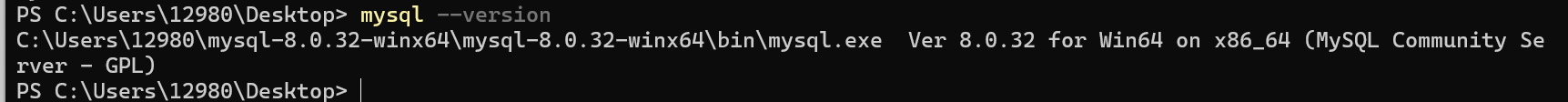
这几个函数为显示一些商品类别界面，在index\_after中有url跳转，帮助用户更好的了解我们平台主要经营什么。

1. **部署方法**（该项目如何在您电脑上部署运行）——

所有源代码（包括python代码：app.py（后端）以及所有前端html js代码还有css样式和对应的image按层级放好整体压缩后放在本实验报告同级文件夹下一同上交）

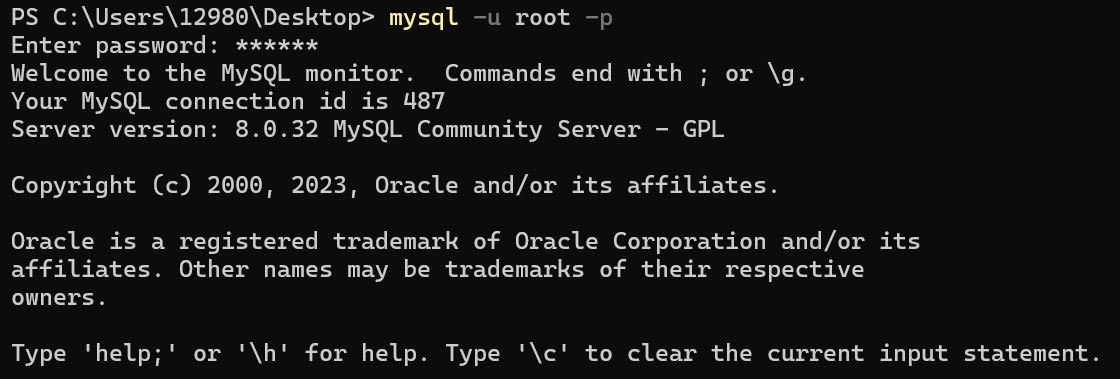
环境要求：

Mysql版本不低于——

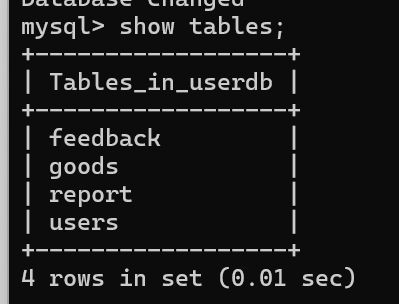
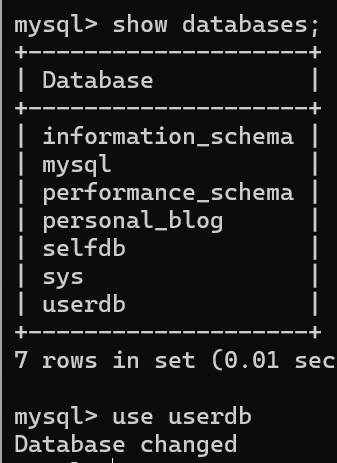


现在MySQL应该不会再比这版本低了，随便下一个就行

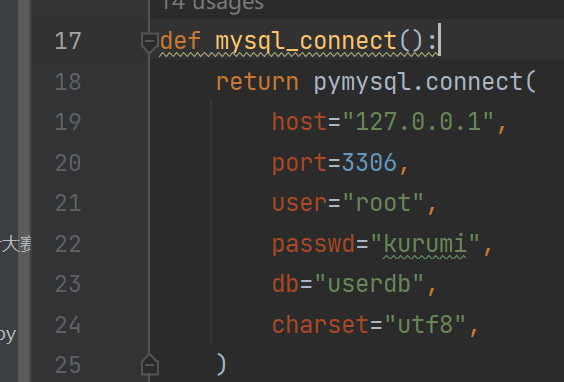
登录方式——



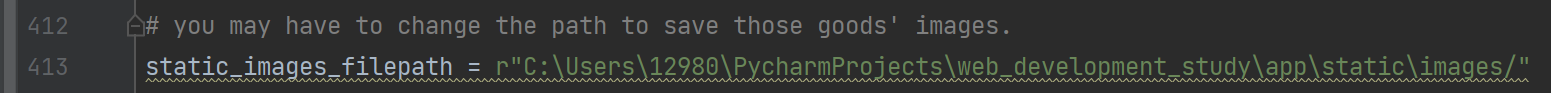
创建一个database名字为userdb，在里面创建四个表——



在python文件app.py中更改此处为您的数据库信息——



更改此处为您的文件路径——



完成

接下来直接运行app.py您就可以在端口映射中在前端浏览器访问本项目，一切前端操作都建立在userdb数据库基础上。

（事实上我曾经试过ngrox的内网穿透，可以将我的电脑作为服务器，您可以直接浏览器输入网址访问我的项目，但是这个要花钱，所以我没有搞）

以下为回答助教额外布置的任务，flask框架的个人学习报告。

1. **Appendix：**Flask框架学习报告——

**附上Flask框架的官网教学界面——**

[API — Flask Documentation (3.0.x) (palletsprojects.com)](https://flask.palletsprojects.com/en/3.0.x/api/#flask.scaffold.Scaffold.route)

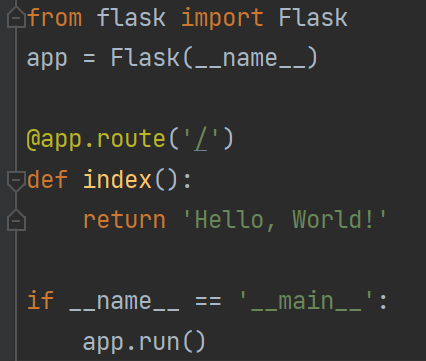
1. Flask框架与HTML交互实现图片上传——图片（文件）上传原理：

图片数据的格式是**多部分表单数据 (multipart/form-data)**，是一种在HTTP POST请求中用来上传文件的格式。在这种格式下，表单数据被划分为多个部分，每个部分包含一个字段的数据。每个部分都包含了字段的名称、数据类型以及数据本身。当用户选择图片并提交表单时，**浏览器**会**将**选择的**图片数据编码**成多部分表单数据格式，然后通过POST请求发送给服务器。在服务器端，flask框架通过**request**请求获取到这个多部分表单数据，并将其解析成文件对象，保存到服务器端本地。（request学习见第8条）

1. 路由：请求的**URL**（uniform resource location，网络链接，统一资源定位符）**如何映射到对应的响应函数**？

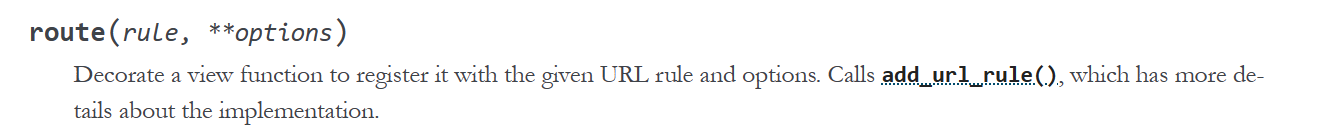
通过**@app.route**装饰器来实现。

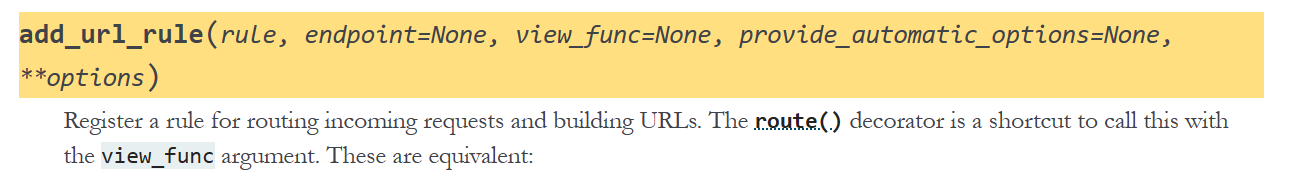
Example—



当用户在客户端访问根路径('/')时，Flask会调用index()函数来处理请求。

官方文档介绍——



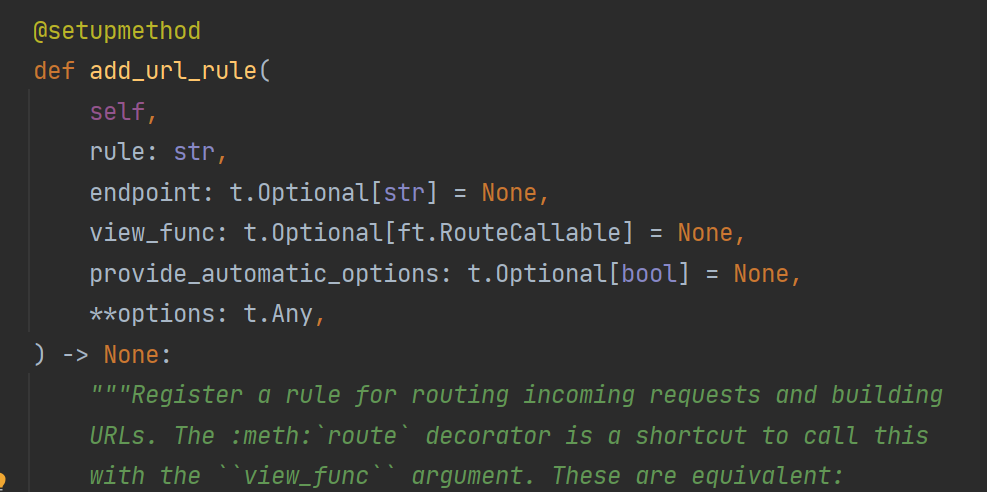


实际上对一个函数定义route装饰器就是调用了app（Flask实例）的add\_url\_rule()函数，因此，route装饰器相当于是add\_url\_rule()函数的简化写法，作用就是添加一个路由关系

route()源代码解释——

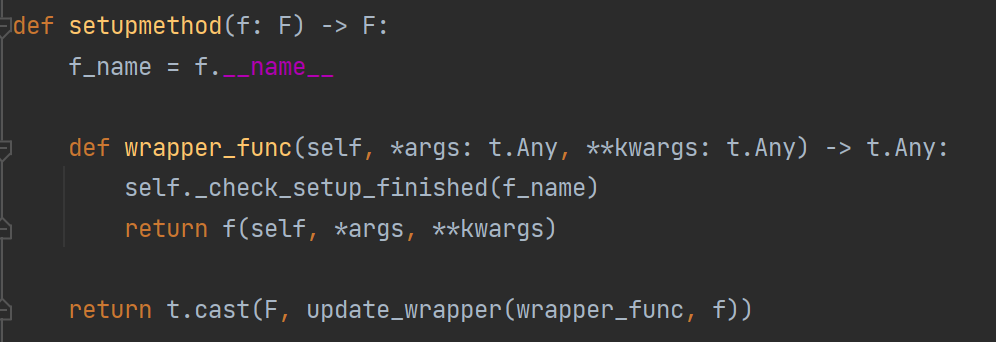


self就是app，传入decorator的函数f（view\_function）就是被装饰的功能函数index。本质上是一种映射，flask框架使用这种方式将每一个路由与具体的功能函数对应起来。如上例，当我们访问端口的根路径（‘/‘）时，实际上后端是index()函数在服务，index()函数决定用户在该路径下看到的内容。由这种方式可以想到，flask的前后端交互功能非常方便，逻辑清晰：一个后端功能函数对应一个前端界面（一般为html），对应唯一一个路由。



将self（传入的参数）封装起来，后面rule那些代码是限定类型。

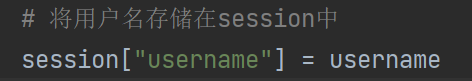
封装方式，调用setupmethod



1. **Session**：每个请求的session如何维护？

每个请求的Session是通过**客户端的Cookie**（Cookie类型为“小型文本文件”，是网站为了辨别用户身份，进行跟踪而储存在用户本地终端上的数据）来维护的（通常经过加密），由用户客户端计算机暂时或永久保存的信息。Flask使用session对象来处理Session。当你使用session对象存储数据时，Flask会自动将数据加密并存储在客户端的Cookie中。这样，即使在无状态的HTTP协议下，Flask也能够实现会话状态的保持。

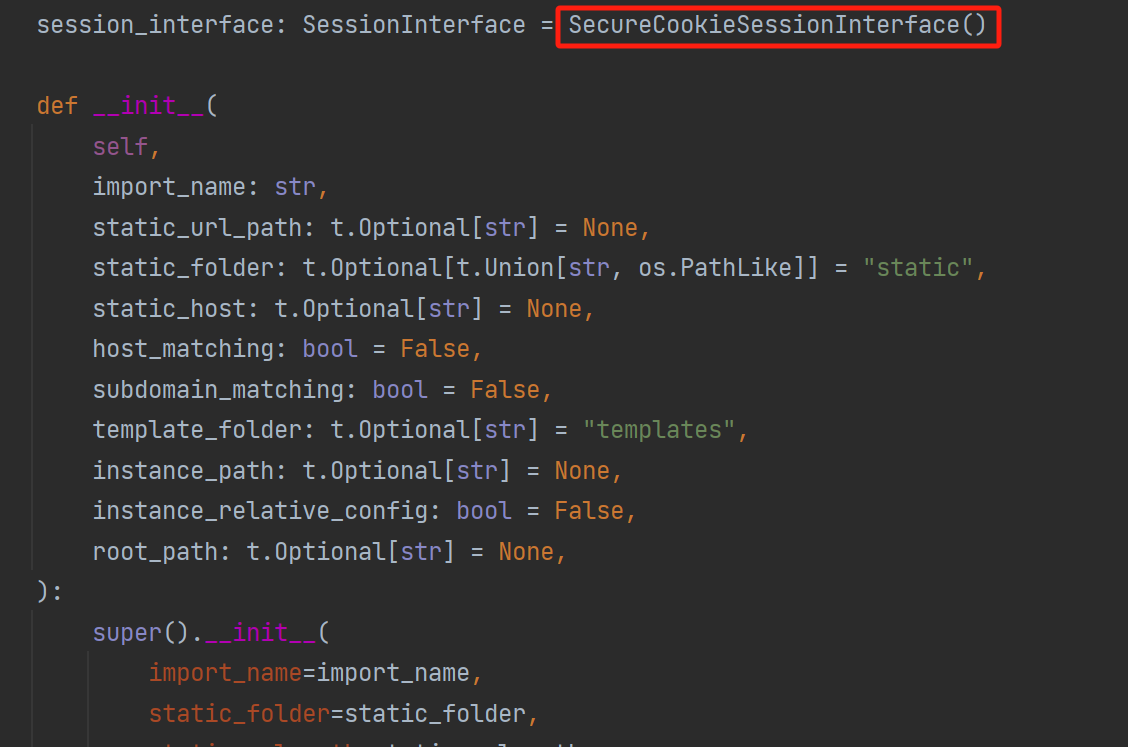
Example使用——



Username从request中取到（request在第8条介绍），这里将username保存在session中，此后用户端使用该函数的所有操作都对应该用户的行为。

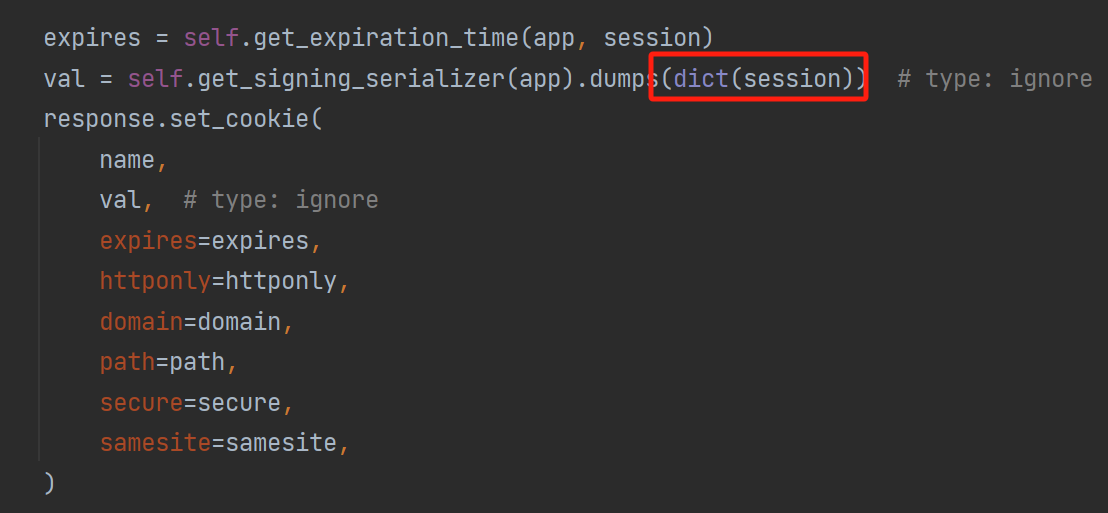
这种存储方式的本质是**字典存储**——

Session\_interface源代码：



（类中的执行步骤和第8条的request很像）

对session的操作实际上是调用了SecureCookieSessionInterface()这个函数，该函数源代码（部分截取）：



实际上是字典存储。并且最后设置了相当于标识的cookie（来自客户端），起到了对应唯一客户端的效果（每个客户端的cookie不一样，服务于他们的session也就不同，并且任意两个之间不产生冲突）

在后端函数编写中，session能存储一些实时的重要用户信息，管理用户状态（例如，登录后username存储在session中，之后直到登出（清空session的username字样），所有行为都能对应于该用户。譬如，执行个人信息修改时，以此保证修改的的确时该用户的个人信息（update……from……where username = username）

1. **database**：与数据库连接如何维护？（如何与数据库交互？）

Flask并没有内置数据库连接池或ORM（对象关系映射）。

但使用python语言，我们可以做到前端与数据库的交互。具体来说，以MySQL数据库为例，python中有一个扩展包名为pymysql，使得我们可以使用python语言操作MySQL数据库。数据库连接Example——

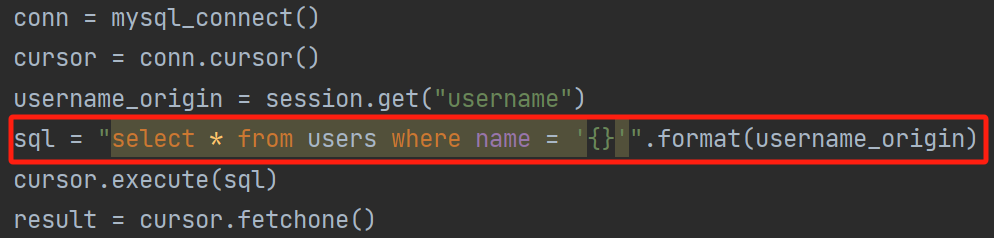


这一块我觉得逻辑应该是：**前端——交互——后端——数据库**

Flask为交互的部分，实际上和database并不直接相关联。

**一条通路**：数据从前端（客户端）传输过来，Flask的作用是交互，将前端用户的数据拿回到后端（使用get或post方式），后端为python代码实现核心逻辑，将这些数据进行处理，最后将需要的部分按SQL语言存储到本地数据库。

Example——



（以上代码解释：conn连接数据库，cursor使用游标方式，从session中取得当前用户的username，向数据库中查询name为username的所有条目，取第一个）

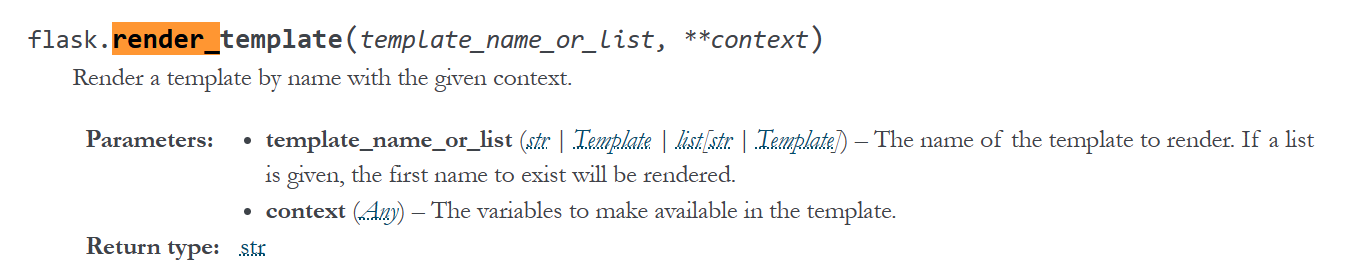
**另一条通路**：后端从数据库中取得所需的数据（使用SQL语言），通过Flask实现交互，将数据返回到前端显示在用户界面。

Example——

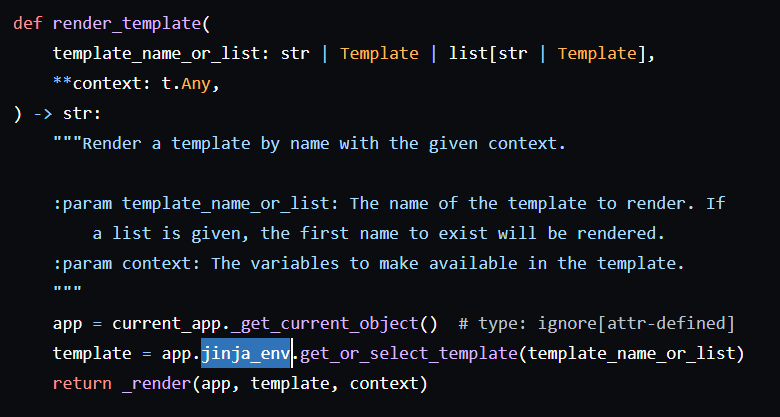


（以上代码解释：从数据库中的goods表中获取所有条目，随机抽取九个，通过flask框架的函数“render\_template”实现交互，返回到前端index.html中显示出来（显示方式由index.html的CSS样式具体定义，总之数据是传过去了））。

render\_template()函数是flask中的模板渲染函数，（一般为html界面）在客户端方对应的url下加载出对应的前端界面，支持数据传递。



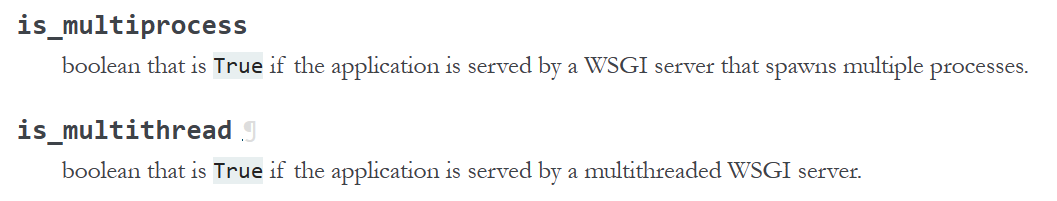
该函数是jinja2模板引擎下的一个函数



补充：

1. request请求是串行处理，还是并发处理？

Flask默认是**单线程**处理请求，请求是**串行处理**的，每个请求会在服务器上依次处理并完成。当一个请求到达服务器时，Flask会在一个线程中处理这个请求，这个线程会处理完整个请求的生命周期，包括调用相应的视图函数。



官方文档也只说明了，要被多线程WSGI server服务才能实现。

如果要并发处理请求，需要一些别的工具对Flask加以辅助。根据我查到的资料：

1. 可以使用类似于Gunicorn这样的WSGI服务器来部署Flask应用，它们支持多进程的并发处理。该网站有示例：[基于Flask的高并发部署方案\_flask高并发csdn-CSDN博客](https://blog.csdn.net/hhhhhhhhhhwwwwwwwwww/article/details/135539459)
2. 并发编程：python中的threading库和multiprocessing库。该网站有示例：[人生苦短，我用Python 九：Flask接口并发处理：多进程、多线程\_flask多线程例子-CSDN博客](https://blog.csdn.net/shdabai/article/details/131479178)
3. 异步在Flask中的使用

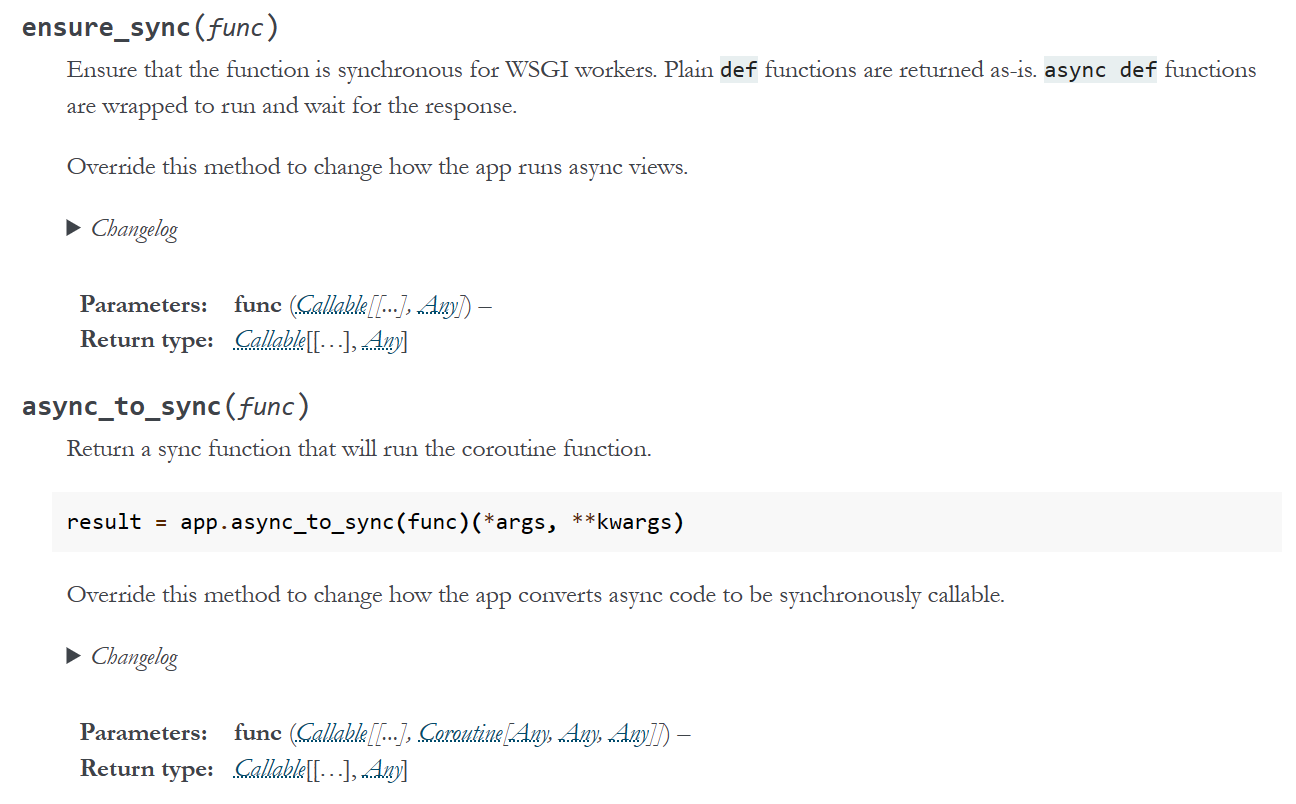
Flask本身并不直接支持异步。但是我在他官方网站上查到有一版Flask可以：



当然也有一些其它方法实现异步，基本思路与5相同，通过一些编程上的技巧或扩展包实现。例如：

1. 使用celery库：[flask异步编程之celery - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/659400775)
2. 使用异步函数库asyncio：[flask 异步编程 asyncio\_flask asyncio-CSDN博客](https://blog.csdn.net/cliffordl/article/details/134798427)

在Flask官网中，对asynchronous只有这一段说明：



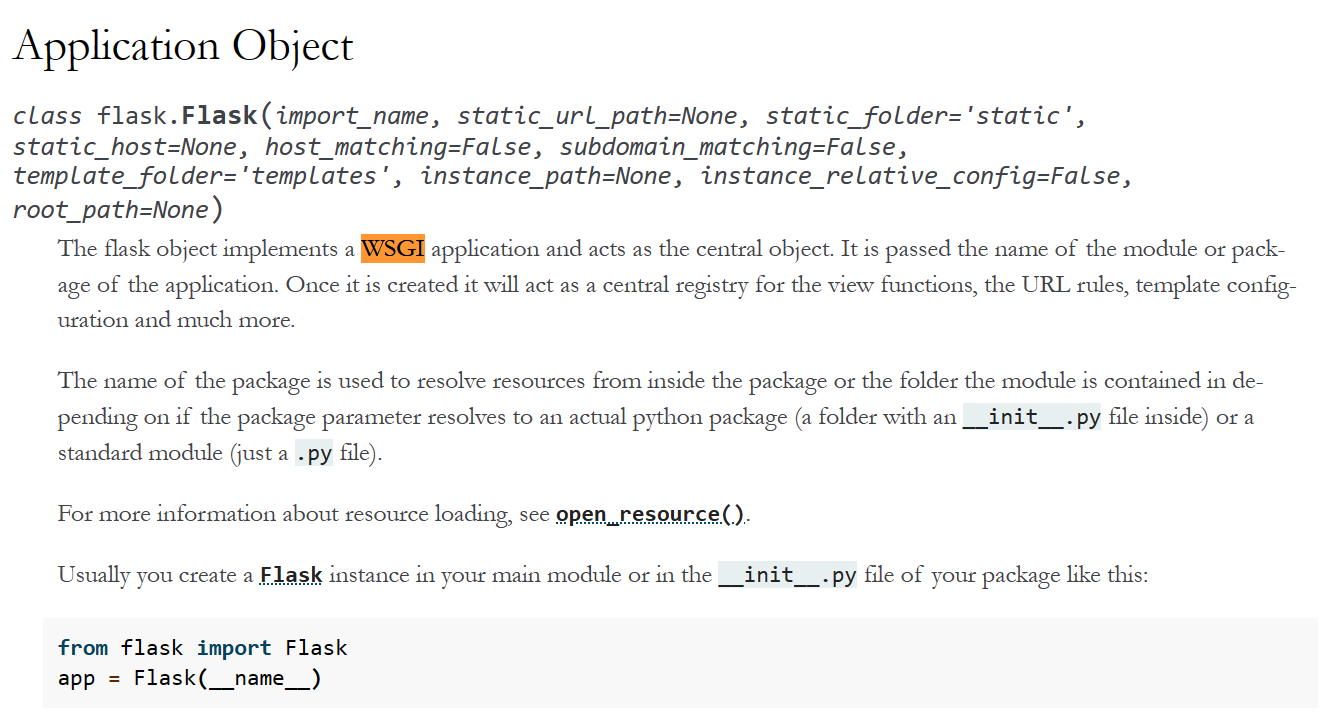
本身并不支持异步，如果遇到asyc def的函数，等其执行完。

将异步代码转换为可同步调用的方式。

1. uWSGI与Flask的交互

最底层的部分，Flask框架基础建立在WSGI协议上，flask的很多函数也是建立在WSGI上，例如第9部分上下文的request\_context，初始化时传入的environ就是一个WSGI Environment（一个字典，包含请求中的所有环境变量）。

官方文档说明——





官方文档使用uWSGI服务器部署Flask应用教学——

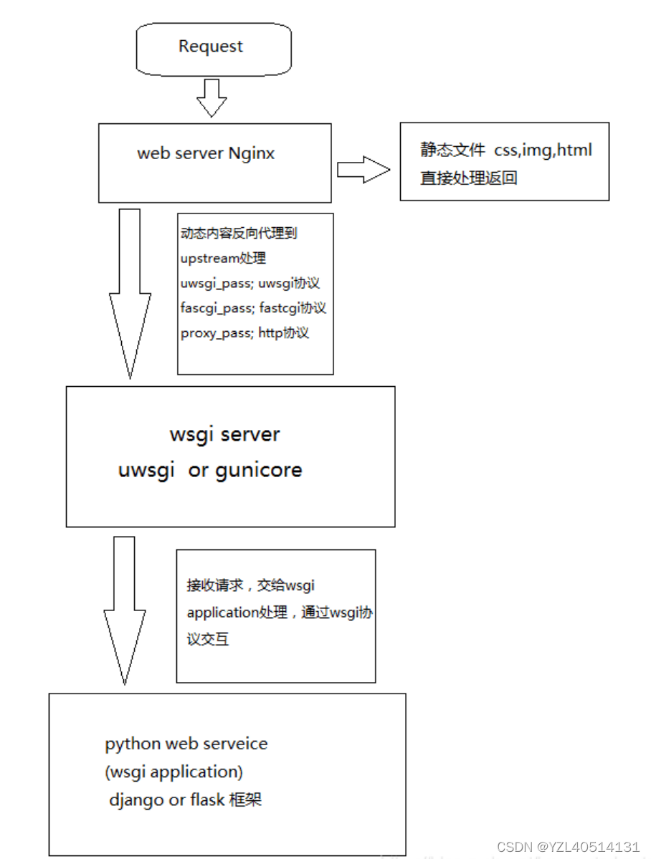
[uWSGI — Flask Documentation (3.0.x) (palletsprojects.com)](https://flask.palletsprojects.com/en/3.0.x/deploying/uwsgi/)

next

[Nginx和uWSGI和Flask的关系\_nginx和wsgi-CSDN博客](https://blog.csdn.net/YZL40514131/article/details/123461381)

WSGI: Web服务器网关接口（Web Server Gateway Interface）是为Python语言定义的Web服务器和Web应用程序或框架之间的一种简单而通用的接口。

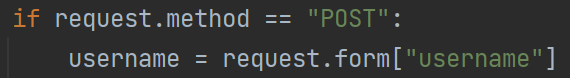
uWSGI是一个WSGI服务器实现，可以用来运行Python的Web应用程序，包括Flask应用。通常情况下，uWSGI会与Flask应用通过WSGI接口进行交互，WSGI接口定义了Web服务器和Web应用程序之间的通信协议。Flask应用会被uWSGI服务器加载和调度，uWSGI负责处理请求和响应，然后将它们传递给Flask应用进行处理。这种方式可以实现高效的并发处理和负载均衡。



1. Flask**请求（request）**相关问题。

Flask的request对象是一个全局变量，它封装了客户端发送的HTTP请求的所有信息。简单来说：工作原理是基于一个全局的请求上下文（global request context）。当Flask接收到一个请求时，它会自动处理请求，创建上下文，并将请求对象放入到上下文中，这样我们就可以在view\_function中通过request对象访问请求信息。

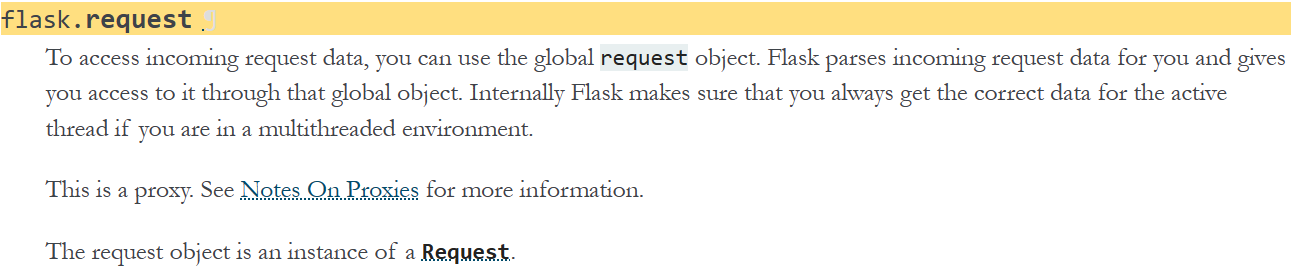
Example从客户端得到信息，取回变量名叫“username”的内容——

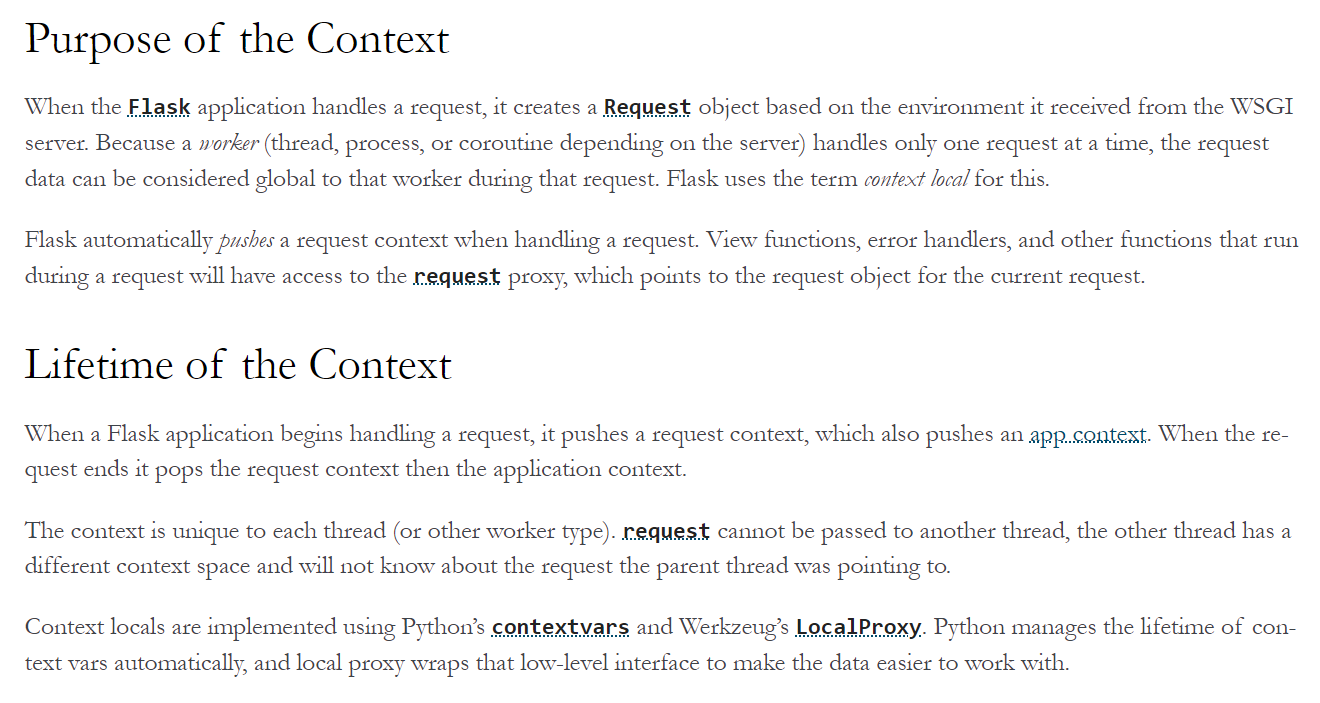


详解：[flask-源码-request-CSDN博客](https://blog.csdn.net/DALAOS/article/details/113566518)

官方文档介绍[https://flask.palletsprojects.com/en/3.0.x/reqcontext/](https://flask.palletsprojects.com/en/3.0.x/reqcontext/——)

（该文档不止有讲request的工作原理，也有讲一次request的全过程，上面那篇博客有更清晰的全过程总结）





重点1：一个进程在一个时间只能处理一个request，request的数据可以被认为是全局量。

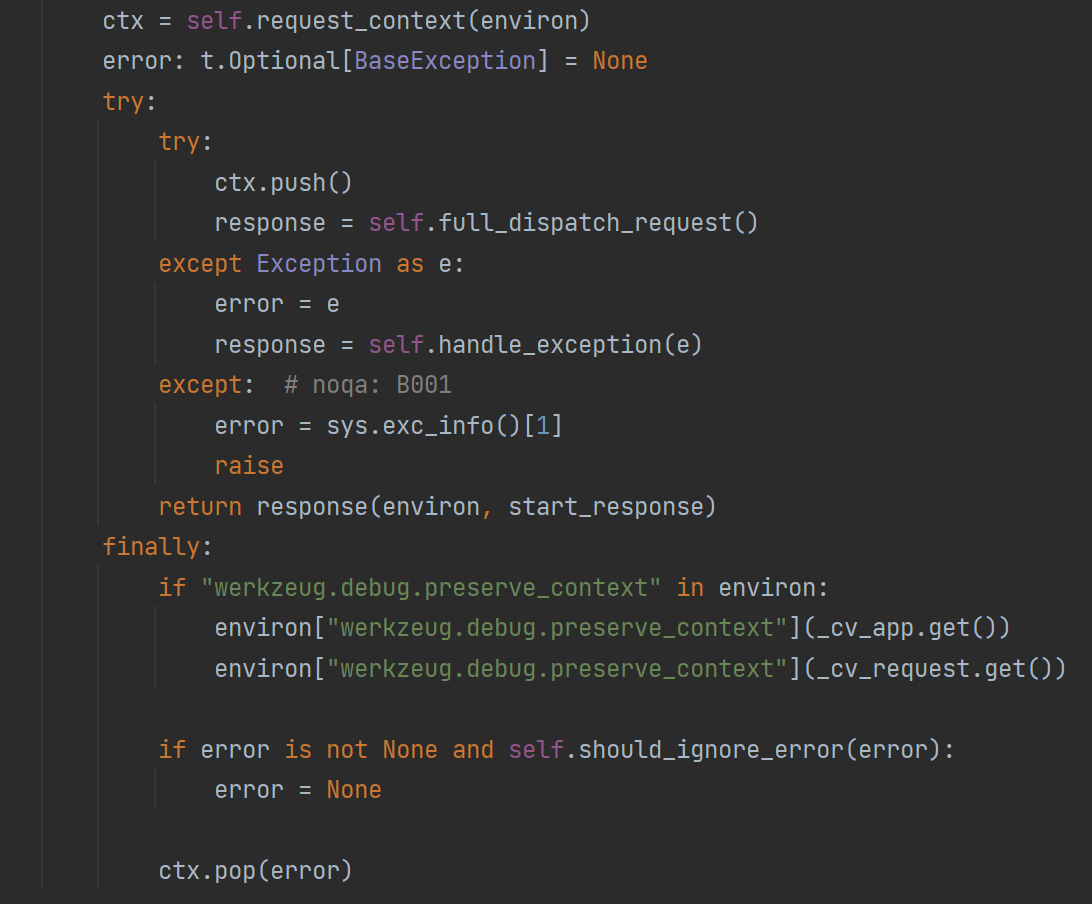
重点2：客户端request开始时app自动将内容push进一个专门服务request context的全局栈中，结束时pop。

重点3：每个线程各有自己的全局栈，不同线程间栈无关。

1. **上下文（context）**相关源代码详解

（这一块我是根据b站[day141-03 Flask请求上下文源码流程（一）\_哔哩哔哩\_bilibili](https://www.bilibili.com/video/BV1es411M7cM?p=31&spm_id_from=pageDriver&vd_source=c1a86248a42b5b39585a1befec584369)此视频学习。这一节讲的非常详细，从flask源码出发，一直递归学习到所有函数的最底层，讲清楚了flask处理上下文响应的全过程）以下是学习后我自己的理解——

源代码（在flask类的\_\_call\_\_中的wsgi\_app()函数中）：



首先：ctx = self,request\_context(environ) 将请求相关的数据封装到了RequestContext中，其次，将封装了请求相关数据的RequestContext对象添加到了8中所述的全局栈中。实际上：\_request\_stx\_stack.push(self) 源码中直接是ctx.push() （\_request\_stx\_stack实际上是flask中的一个全局栈LocalStack()）

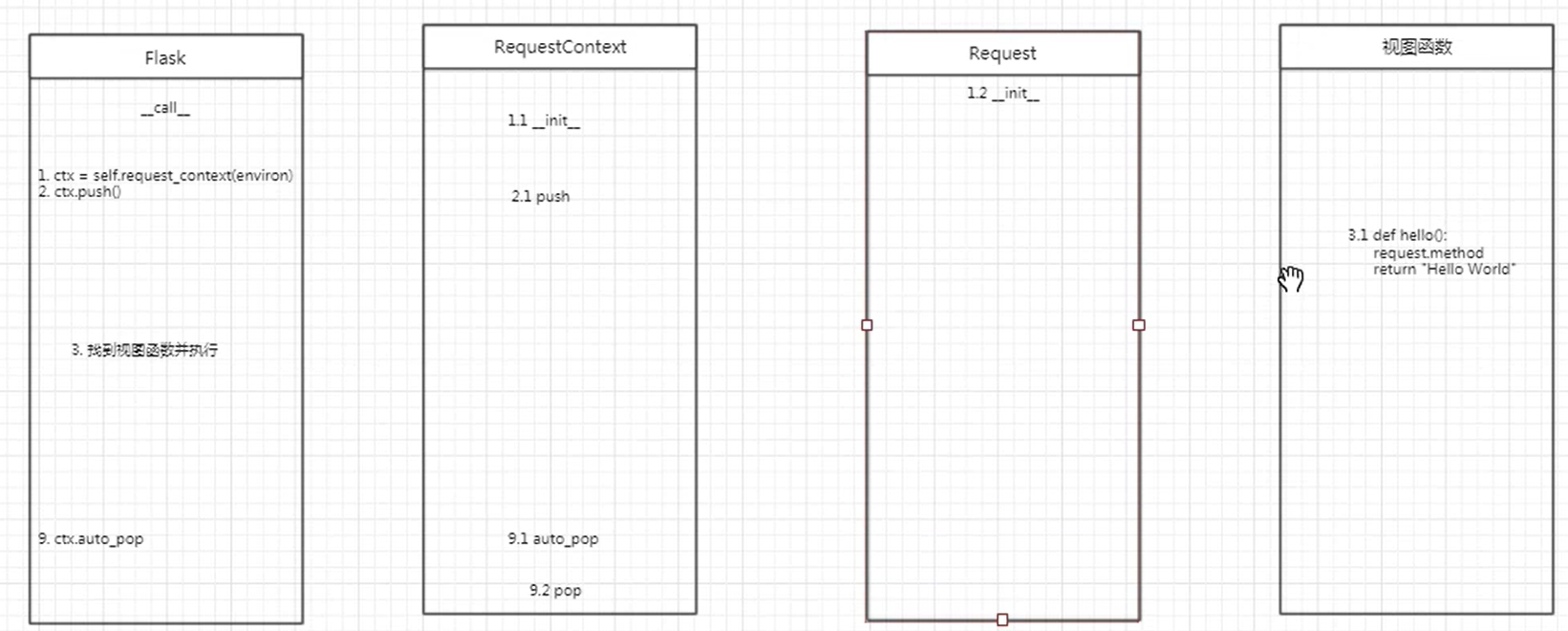
中间过程，就可以使用request的数据。

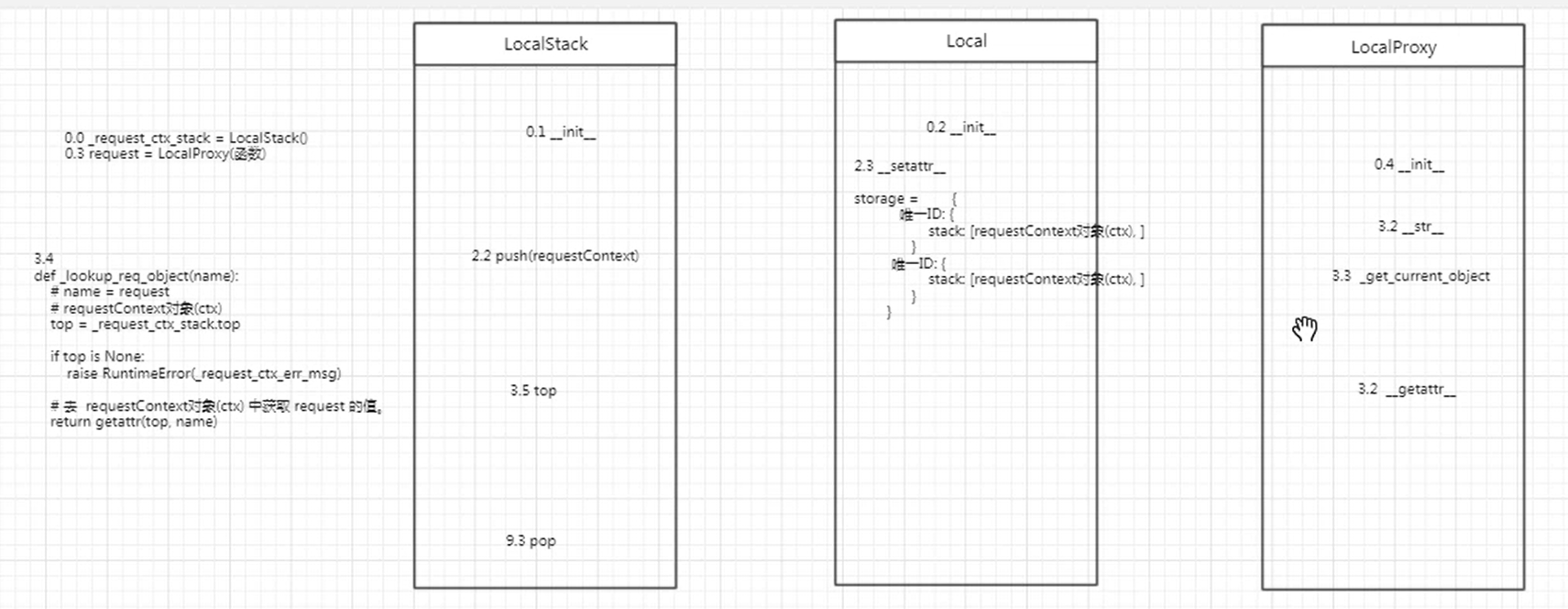
最后当请求结束，pop，将该对象清空。

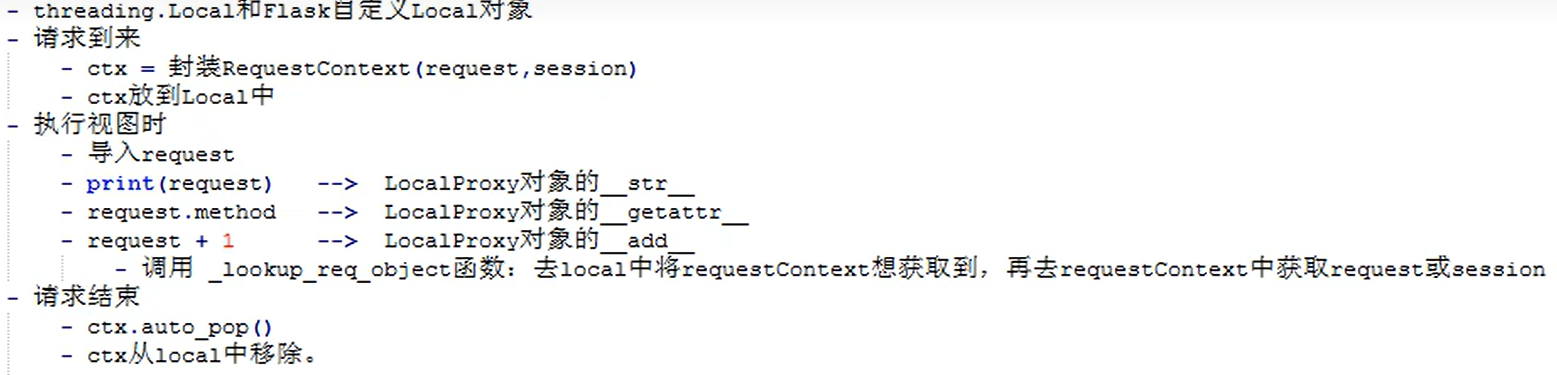
执行流程很清晰，主要是难在管理方式（这一块flask框架就自动帮忙管理了……）

全局变量为LocalStack, Local, LocalProxy，相当于在app运行初始就存在的全局量。Local中存放ctx，包含用户唯一id，起到标识作用，LocalProxy执行view\_function。在一个request发起时，Flask调用RequestContext，初始化，RequestContext又创建Request对象（在其中真正保存请求的各项数据）。

以下flask执行request全流程图——



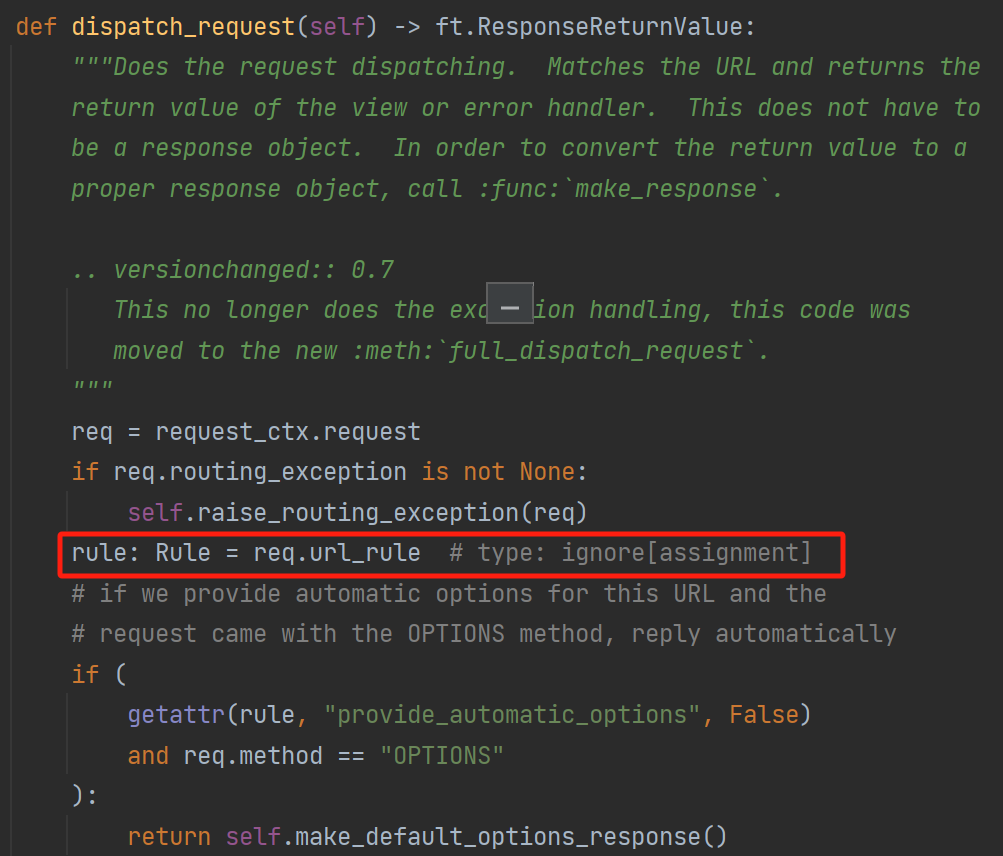




Supplement：

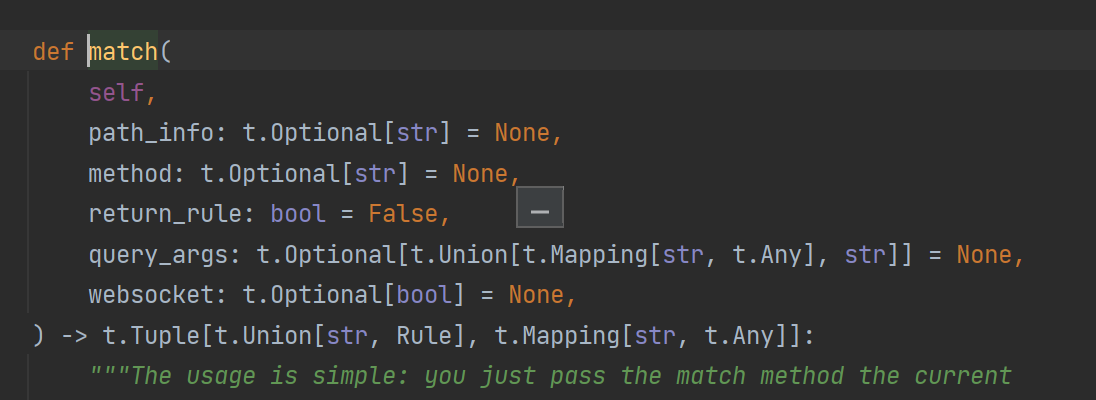
**url应该是用户在发request时封装在RequestContext中一起传回来的**，之后这个上下文就被存储在那个全局栈中。url和对应视图函数相关联是存放在一个url\_map中，在前面那个add\_url\_rule建立url规则时就添加进了这个url\_map

取得对应的视图函数，主要是dispatch\_request()函数做的事

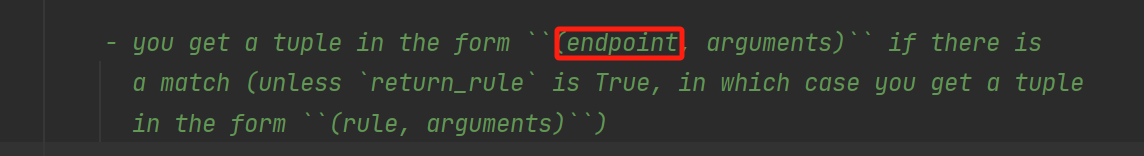


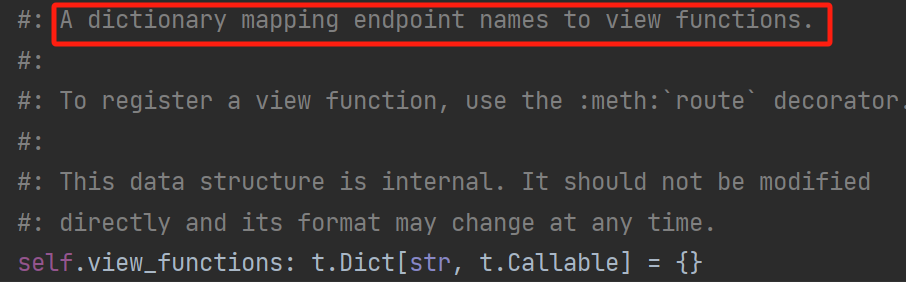
首先由它取得request传回的url

然后使用match()函数从路由映射表中匹配出对应的端点（endpoint）



关键是最后匹配（相当于遍历url\_map）找到url对应的endpoint



然后再根据endpoint找到对应的视图函数（view\_func），这一步由view\_function实现。

以上步骤找到对应与该url的视图函数。

以上操作的简化版代码大概就是——

