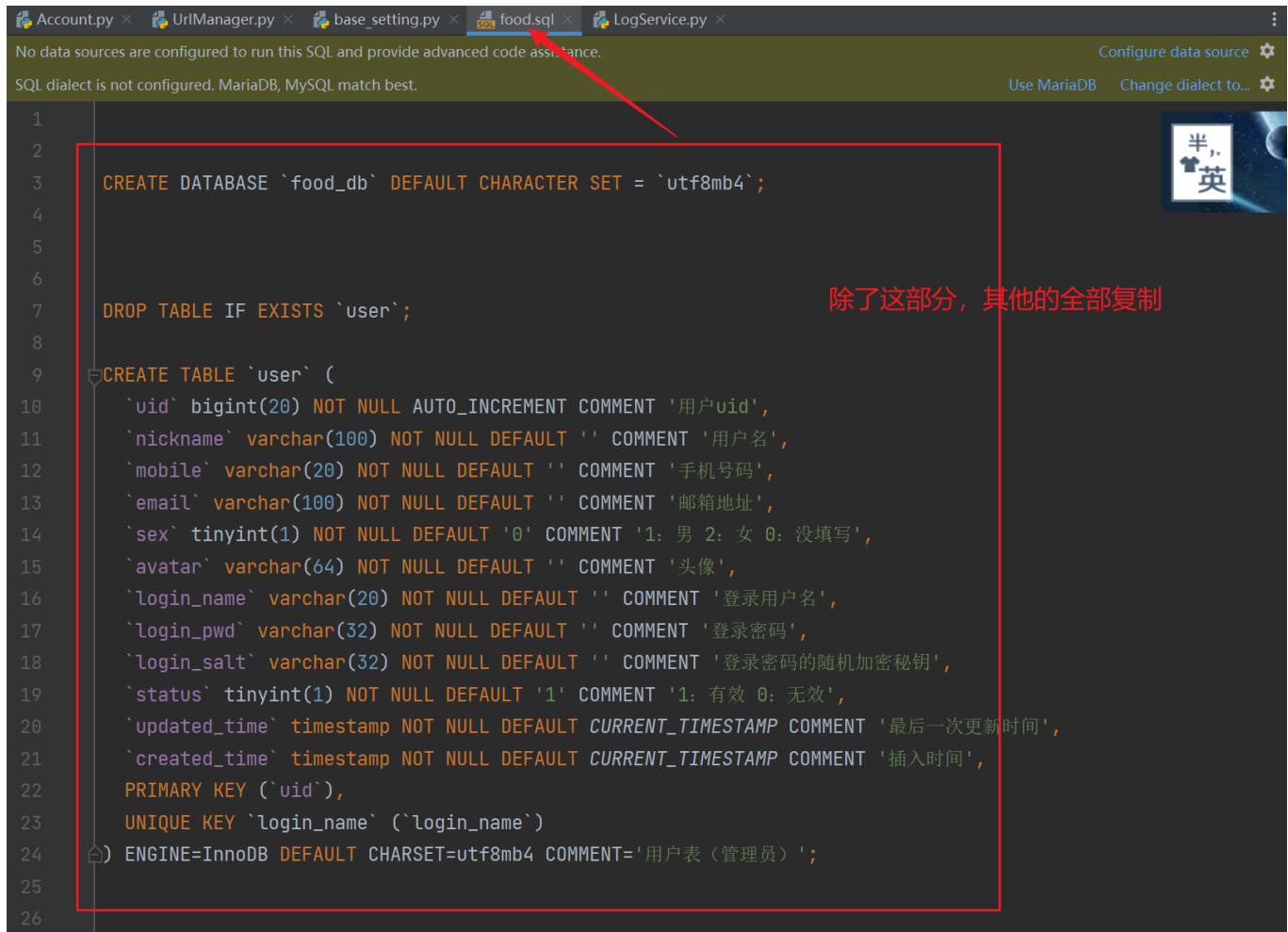


1.建立数据库 - 表

在初期的food.sql文件中复制以下数据 ---- 一定要从sql文件中去复制



```
1 CREATE DATABASE `food_db` DEFAULT CHARACTER SET = `utf8mb4`;
2
3 DROP TABLE IF EXISTS `user`;
4
5
6
7 CREATE TABLE `user` (
8     `uid` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '用户uid',
9     `nickname` varchar(100) NOT NULL DEFAULT '' COMMENT '用户名',
10    `mobile` varchar(20) NOT NULL DEFAULT '' COMMENT '手机号码',
11    `email` varchar(100) NOT NULL DEFAULT '' COMMENT '邮箱地址',
12    `sex` tinyint(1) NOT NULL DEFAULT '0' COMMENT '1: 男 2: 女 0: 没填写',
13    `avatar` varchar(64) NOT NULL DEFAULT '' COMMENT '头像',
14    `login_name` varchar(20) NOT NULL DEFAULT '' COMMENT '登录用户名',
15    `login_pwd` varchar(32) NOT NULL DEFAULT '' COMMENT '登录密码',
16    `login_salt` varchar(32) NOT NULL DEFAULT '' COMMENT '登录密码的随机加密秘钥',
17    `status` tinyint(1) NOT NULL DEFAULT '1' COMMENT '1: 有效 0: 无效',
18    `updated_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '最后一次更新时间',
19    `created_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '插入时间',
20    PRIMARY KEY (`uid`),
21    UNIQUE KEY `login_name` (`login_name`)
22 ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户表（管理员）';
```

[进入数据库](#)

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| anli |
| anli2 |
| db_books |
| demo_book |
| f1demo |
| food_db |
| mysql |
| performance_schema |
| py |
| test |
| v2ex |
| xiaodemo |
| xiaoyu_demo |
+-----+
14 rows in set (0.01 sec)
```

```
mysql> use food_db
Database changed
mysql> DROP TABLE IF EXISTS `app_access_log`;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
14 rows in set (0.01 sec)
```

```
mysql> use food_db
Database changed
mysql> DROP TABLE IF EXISTS `app_access_log`;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql>
mysql> CREATE TABLE `app_access_log` (
    -> `id` int(11) NOT NULL AUTO_INCREMENT,
    -> `uid` bigint(20) NOT NULL DEFAULT '0' COMMENT 'uid',
    -> `referer_url` varchar(255) NOT NULL DEFAULT '' COMMENT '当前访问的refer',
    -> `target_url` varchar(255) NOT NULL DEFAULT '' COMMENT '访问的url',
    -> `query_params` text NOT NULL COMMENT 'get和post参数',
    -> `ua` varchar(255) NOT NULL DEFAULT '' COMMENT '访问ua',
    -> `ip` varchar(32) NOT NULL DEFAULT '' COMMENT '访问ip',
    -> `note` varchar(1000) NOT NULL DEFAULT '' COMMENT 'json格式备注字段',
    -> `created_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    -> PRIMARY KEY (`id`),
    -> KEY `idx_uid` (`uid`)
    -> ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='用户访问记录表';
Query OK, 0 rows affected (0.03 sec)
```

```
mysql>
mysql>
mysql> DROP TABLE IF EXISTS `app_error_log`;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

```
mysql> CREATE TABLE `app_error_log` (
    -> `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
    -> `referer_url` varchar(255) NOT NULL DEFAULT '' COMMENT '当前访问的refer',
    -> `target_url` varchar(255) NOT NULL DEFAULT '' COMMENT '访问的url',
    -> `query_params` text NOT NULL COMMENT 'get和post参数',
    -> `content` longtext NOT NULL COMMENT '日志内容',
```

```
    / PRIMARY KEY `idx_date`(`date`)  
    -> ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COMMENT='全站日统计';  
Query OK, 0 rows affected (0.03 sec)
```

```
mysql>  
mysql> show tables;  
+-----+  
| Tables_in_food_db |  
+-----+  
| app_access_log    |  
| app_error_log     |  
| food              |  
| food_cat          |  
| food_sale_change_log |  
| food_stock_change_log |  
| images            |  
| member            |  
| member_address    |  
| member_cart       |  
| member_comments   |  
| oauth_access_token |  
| oauth_member_bind |  
| pay_order         |  
| pay_order_callback_data |  
| pay_order_item    |  
| queue_list        |  
| stat_daily_food   |  
| stat_daily_member |  
| stat_daily_site   |  
| user              |  
| wx_share_history  |  
+-----+  
22 rows in set (0.00 sec)
```

2.模型建立

Microsoft Windows [版本 10.0.19044.2251]
(c) Microsoft Corporation。保留所有权利。

```
E:\pythonfile\flpro>workon flpro  
(flpro) E:\pythonfile\flpro>
```

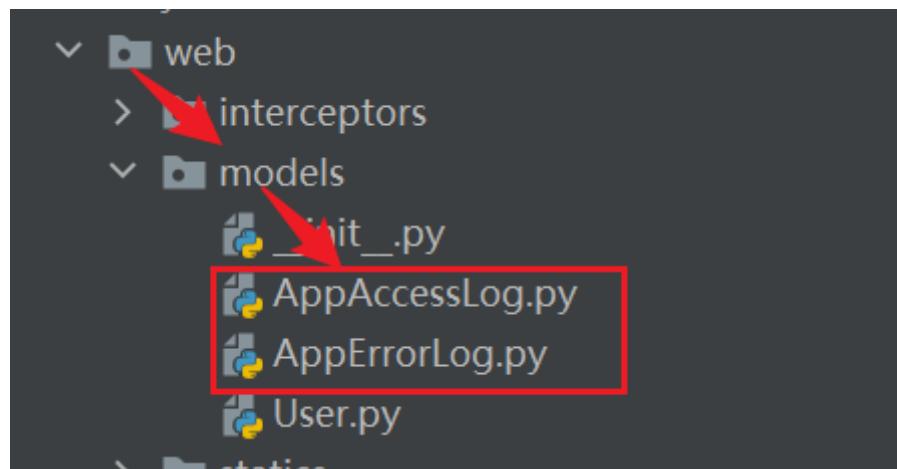
进入项目位置 --- 进入对应项目环境

```
flask-sqlacodegen mysql+pymysql://root:qwe123@127.0.0.1/food_db --tables app_access_log --outfile "web/models/AppAccessLog.py" --flask
```

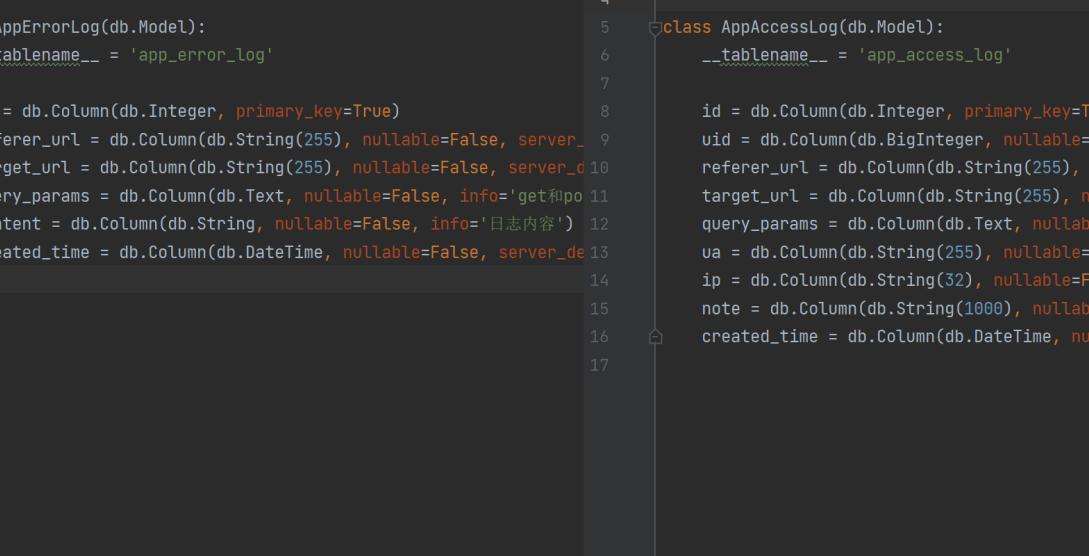
```
flask-sqlacodegen mysql+pymysql://root:qwe123@127.0.0.1/food_db --tables app_error_log --outfile "web/models/AppErrorLog.py" --flask
```

Microsoft Windows [版本 10.0.19044.2251]
(c) Microsoft Corporation。保留所有权利。

```
E:\pythonfile\flpro>workon flpro  
(flpro) E:\pythonfile\flpro>flask-sqlacodegen mysql+pymysql://root:qwe123@127.0.0.1/food_db --tables app_access_log --outfile "web/models/AppAccessLog.py" --flask  
(flpro) E:\pythonfile\flpro>flask-sqlacodegen mysql+pymysql://root:qwe123@127.0.0.1/food_db --tables app_error_log --outfile "web/models/AppErrorLog.py" --flask  
(flpro) E:\pythonfile\flpro>
```



3.调整模型db对象



```
# coding: utf-8
from application import db

class AppErrorLog(db.Model):
    __tablename__ = 'app_error_log'

    id = db.Column(db.Integer, primary_key=True)
    referer_url = db.Column(db.String(255), nullable=False, server_
    target_url = db.Column(db.String(255), nullable=False, server_
    query_params = db.Column(db.Text, nullable=False, info='get和po'
    content = db.Column(db.String, nullable=False, info='日志内容')
    created_time = db.Column(db.DateTime, nullable=False, server_de

# coding: utf-8
from application import db

class AppAccessLog(db.Model):
    __tablename__ = 'app_access_log'

    id = db.Column(db.Integer, primary_key=True)
    uid = db.Column(db.BigInteger, nullable=False, inde_
    referer_url = db.Column(db.String(255), nullable=False)
    target_url = db.Column(db.String(255), nullable=False)
    query_params = db.Column(db.Text, nullable=False, info=''
    ua = db.Column(db.String(255), nullable=False, server_
    ip = db.Column(db.String(32), nullable=False, server_
    note = db.Column(db.String(1000), nullable=False, serv_
    created_time = db.Column(db.DateTime, nullable=False,
```

4. 封装日志记录方法

The screenshot shows the PyCharm IDE interface with the following details:

- Project Tree:** The project is named "fipro" located at "E:\pythonfile\fipro". It contains several packages and files:
 - common**: Contains `__init__.py`, `Helper.py`, `LogService.py`, `UrlManager.py`, and `UserService.py`.
 - lib**: Contains `__init__.py`, `Helper.py`, `LogService.py`, `UrlManager.py`, and `UserService.py`.
 - config**
 - docs**
 - job**
 - web**: This folder is selected and highlighted in blue. It contains `application.py`, `manager.py`, and `www.py`.
 - External Libraries**
 - Scratches and Consoles**
- Code Editor:** The file `LogService.py` is open in the editor. A red arrow points from the file name in the Project tree to the tab bar in the editor.
- Code Content:** The code defines a class `LogService` with a static method `addAccessLog`. The method adds access logs by creating an `AppAccessLog` object and setting its properties based on the current user and request parameters. It also imports `getCurrentDate` from `common.lib.Helper`.

```
from common.lib.Helper import getCurrentDate

"""
用来管理访问记录，添加浏览记录和错误记录
"""

class LogService():
    @staticmethod
    def addAccessLog():
        """添加访问记录日志"""
        # 1. 实例访问记录日志模型对象
        target = AppAccessLog()

        # 2. 根据建立的钩子（拦截器-用户登录验证）中注入的g对象中的用户对象数据
        if "current_user" in g and g.current_user is not None:
            # 存在用户对象数据 -> 获取用户id -> 赋予给target的id属性
            target.uid = g.current_user.uid # 当前用户的yid
        # 获取请求对象中的 referer 请求来源地址（防盗链） -> 赋予给target的referer_url属性
        target.referer_url = request.referrer
        # 获取请求对象中的请求地址-路由 -> 赋予给target的target_url属性
        target.target_url = request.url
        # 获取请求对象中的查询参数（字典形式返回） -> 赋予给target的query_params属性
        target.query_params = json.dumps(request.values.to_dict())
        # 获取请求对象中的User-Agent -> 赋予给target的ua属性
        target.ua = request.headers.get("User-Agent")
        # 获取当前时间 -> 赋予给target的created_time 属性
        target.created_time = getCurrentDate()
```

```
# -*-coding:utf-8-*-
from flask import request, g
from application import db
```

```

import json
from web.models.AppAccessLog import AppAccessLog
from web.models.AppErrorLog import AppErrorLog
from common.lib.Helper import getCurrentDate

"""
用来管理访问记录，添加浏览记录和错误记录
"""

class LogService():
    @staticmethod
    def addAccesslog():
        """添加访问记录日志"""
        # 1. 实例访问记录日志模型对象
        target = AppAccessLog()

        # 2. 根据建立的钩子（拦截器-用户登录验证）中注入的g对象中的用户对象数据
        if "current_user" in g and g.current_user is not None:
            # 存在用户对象数据 -> 获取用户id -> 赋予给target的id属性
            target.uid = g.current_user.uid # 当前用户的yid
        # 获取请求对象中的 referer 请求来源地址（防盗链） -> 赋予给target的referer_url属性
        target.referer_url = request.referrer
        # 获取请求对象中的请求地址-路由 -> 赋予给target的target_url属性
        target.target_url = request.url
        # 获取请求对象中的查询参数（字典形式返回） -> 赋予给target的query_params属性
        target.query_params = json.dumps(request.values.to_dict())
        # 获取请求对象中的User-Agent -> 赋予给target的ua属性
        target.ua = request.headers.get("User-Agent")
        # 获取当前时间 -> 赋予给target的created_time 属性 from common.lib.Helper import
        getCurrentDate
        target.created_time = getCurrentDate()

        # 写入数据库
        db.session.add(target)
        db.session.commit()

        return True

    @staticmethod
    def addErrorlog(content):
        """
        添加错误日志
        :param content: 错误信息
        :return:
        """

        # 忽略favicon.ico请求引发的错误
        if "favicon.ico" in request.url:
            print("图标问题报404可以忽略")
            return
        # 实例错误日志模型对象
        target = AppErrorLog()

        # 获取请求对象中的 referer 请求来源地址（防盗链） -> 赋予给target的referer_url属性
        target.referer_url = request.referrer
        # 获取请求对象中的请求地址-路由 -> 赋予给target的target_url属性
        target.target_url = request.url

```

```

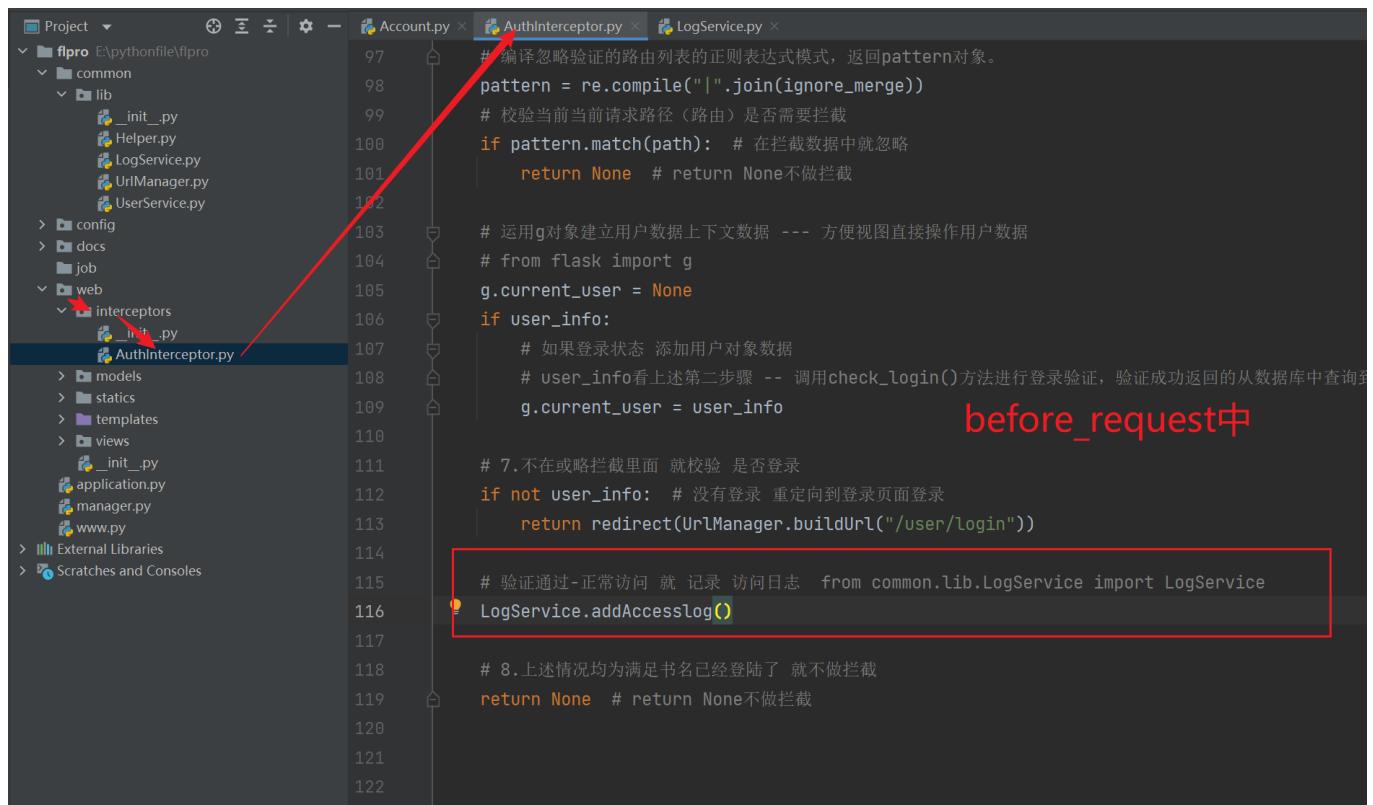
# 获取请求对象中的查询参数 (字典形式返回) -> 赋予给target的query_params属性
target.query_params = json.dumps(request.values.to_dict())
# 错误信息内容 -> 赋予给target的content属性
target.content = content
# 获取当前时间 -> 赋予给target的created_time 属性 from common.lib.Helper import
getCurrentDate
target.created_time = getCurrentDate()

# 写入数据库
db.session.add(target)
db.session.commit()
return True

```

5.日志记录插入

访问日志



The screenshot shows the PyCharm IDE interface with the project structure on the left and the code editor on the right. The code editor is displaying the `AuthInterceptor.py` file. A red arrow points from the project tree to the `AuthInterceptor.py` file in the editor. A red box highlights the line of code `LogService.addAccesslog()` at line 116. A red annotation text `before_request中` is placed to the right of the highlighted code.

```

# 编译忽略验证的路由列表的正则表达式模式，返回pattern对象。
pattern = re.compile("|".join(ignore_merge))
# 校验当前请求路径（路由）是否需要拦截
if pattern.match(path): # 在拦截数据中就忽略
    return None # return None不做拦截
# 运用g对象建立用户数据上下文数据 --- 方便视图直接操作用户数据
# from flask import g
g.current_user = None
if user_info:
    # 如果登录状态 添加用户对象数据
    # user_info看上述第二步骤 -- 调用check_login()方法进行登录验证，验证成功返回的从数据库中查询到
    g.current_user = user_info
# 7.不在或略拦截里面 就校验 是否登录
if not user_info: # 没有登录 重定向到登录页面登录
    return redirect(UrlManager.buildUrl("/user/login"))
# 验证通过-正常访问 就 记录 访问日志 from common.lib.LogService import LogService
LogService.addAccesslog()
# 8.上述情况均为满足书名已经登陆了 就不做拦截
return None # return None不做拦截

```

6.日志记录展示

首先你知道 日志信息在哪展示的 -- <http://127.0.0.1:8888/account/info?id=1> --- 也就用户详情页中展示

管理后台

127.0.0.1:8888/account/info?id=1

Gmail YouTube 地图 mysql JSON在线解析及格... 浏览器控制台 python 工作 解决SSH登录无响应 爬虫 web HTML格式

然后就可以进行视图操作了

```
Account.py x AuthInterceptor.py x LogService.py x
98 req = request.args
99
100     # 3.获取查询字符串 id键数据
101     uid = int(req.get("id", 0)) # 没有就设置默认值0, 代表没有
102
103     # 4.构建请求路由 -- 列表页
104     reback_url = UrlManager.buildUrl("/account/index")
105
106     # 5. uid 小于1 代表没有该用户数据
107     if uid < 1:
108         return redirect(reback_url) # 回到列表页
109
110     # 6.根据uid查询对应用户 -- 获取用户数据
111     info = User.query.filter_by(uid=uid).first()
112
113     # 7. 如果没有用户数据 回到列表页
114     if not info:
115         return redirect(reback_url)
116
117     # 8. 有用户数据 就响应用户详情页
118     context["info"] = info
119
120     # 9.个人用户访问记录 , 基于uid查询, 然后根据id 降序-desc() 排列, 取前10条-limit(10)  from web.models.AppAccessLog
121     acces_list = AppAccessLog.query.filter_by(uid=uid).order_by(AppAccessLog.id.desc()).limit(10).all()
122     context["acces_list"] = acces_list
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
```

```
class Info(MethodView):
    def get(self):
        # 1.建立渲染数据字典
        context = {}

        # 2.获取请求携带的查询字符串数据
        req = request.args

        # 3.获取查询字符串 id键数据
        uid = int(req.get("id", 0)) # 没有就设置默认值0, 代表没有

        # 4.构建请求路由 -- 列表页
        reback_url = UrlManager.buildUrl("/account/index")

        # 5. uid 小于1 代表没有该用户数据
        if uid < 1:
            return redirect(reback_url) # 回到列表页

        # 6.根据uid查询对应用户 -- 获取用户数据
        info = User.query.filter_by(uid=uid).first()

        # 7. 如果没有用户数据 回到列表页
        if not info:
            return redirect(reback_url)

        # 8. 有用户数据 就响应用户详情页
        context["info"] = info

        # 9.个人用户访问记录 , 基于uid查询, 然后根据id 降序-desc() 排列, 取前10条-limit(10)  from web.models.AppAccessLog
        acces_list = AppAccessLog.query.filter_by(uid=uid).order_by(AppAccessLog.id.desc()).limit(10).all()
        context["acces_list"] = acces_list
```

```
return render_template("account/info.html", **context)
```

视图数据渲染后 -- 调整模板

The screenshot shows a code editor with several tabs open. On the left is a file tree for a Python project named 'fipro'. The 'templates/account' directory contains three files: 'index.html', 'info.html', and 'set.html'. A red arrow points from the file tree to the 'info.html' tab. The 'info.html' tab displays an HTML template with a red box highlighting the table section. The table has a single row with two columns, both containing placeholder text: '访问时间' and '访问Url'. Below the table, there is some Jinja2 templating code. The code editor interface includes a status bar at the bottom.

```
<div class="panel-body">
    <div class="tab-content">
        <div class="tab-pane active">
            <table class="table table-bordered">
                <thead>
                    <tr>
                        <th>访问时间</th>
                        <th>访问Url</th>
                    </tr>
                </thead>
                <tbody>
                    {% if acces_list %}
                        {% for item in acces_list %}
                            <tr>
                                <th>{{ item.created_time }}</th>
                                <th>{{ item.target_url }}</th>
                            </tr>
                        {% endfor %}
                    {% else %}
                        <tr><td colspan="2">暂无数据</td></tr>
                    {% endif %}
                </tbody>
            </table>
        </div>
    </div>
</div>
```

7.启动测试

<http://127.0.0.1:8888/account/info?id=1>

The screenshot shows a web browser window with the URL 'http://127.0.0.1:8888/account/info?id=1'. The page displays account information for a user named '风助'. It includes a sidebar with navigation links like '仪表盘', '账号管理', '美食管理', '会员列表', '财务管理', and '统计管理'. The main content area shows a summary of the user's details and a table of their access records. The table has two columns: '访问时间' (Access Time) and '访问Url' (Access URL). The data in the table is highlighted with a red box, matching the red box in the code editor's template. The browser's address bar shows the full URL, and the status bar at the bottom indicates the connection is secure.

访问时间	访问Url
2022-11-24 16:27:11	http://127.0.0.1:8888/account/info?id=1
2022-11-24 16:25:21	http://127.0.0.1:8888/account/info?id=1
2022-11-24 16:25:18	http://127.0.0.1:8888/account/info?id=1