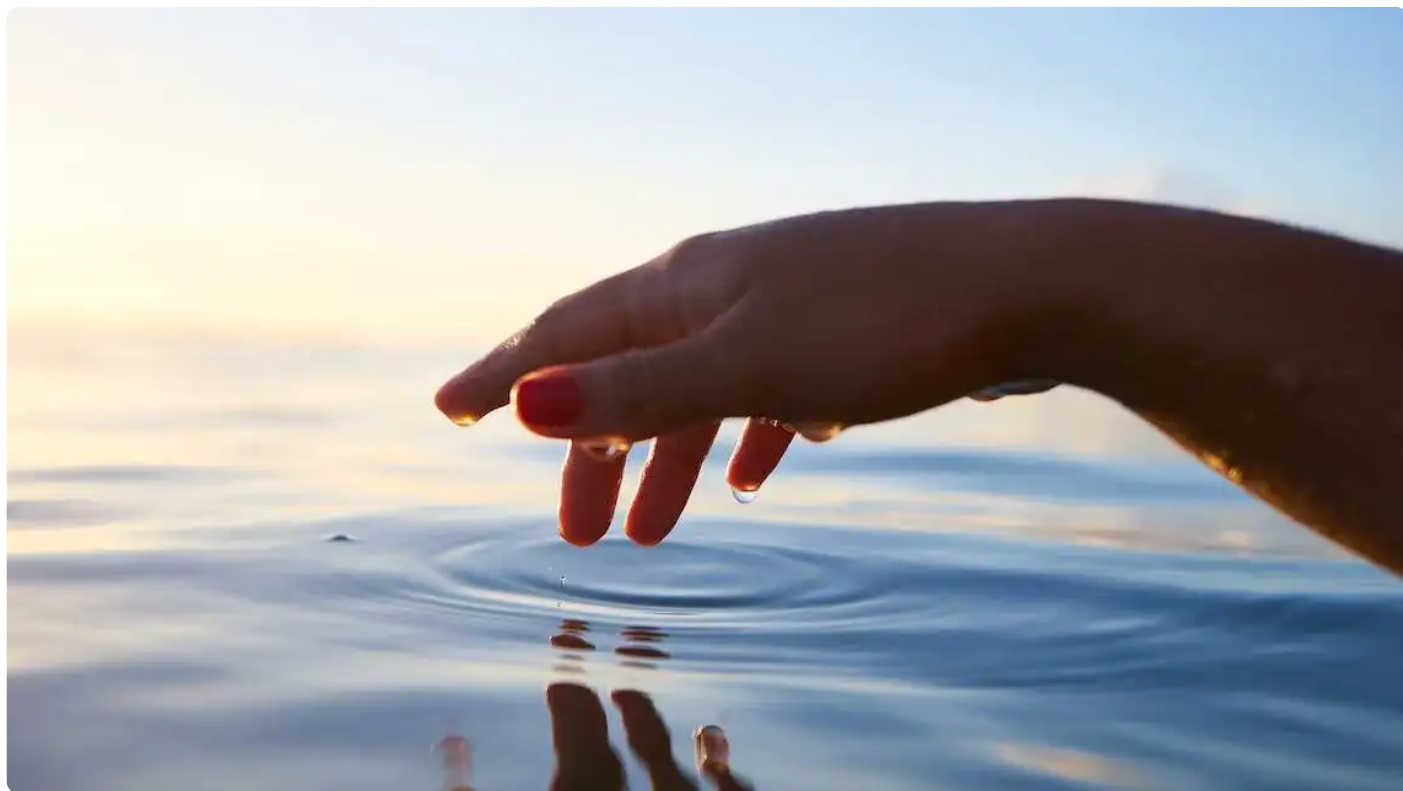


23 | 数据库应用（三）：项目数据库配置实战

2023-06-14 Barry 来自北京

《Python实战·从0到1搭建直播视频平台》



你好，我是 Barry。

在上节课，我们实现了数据库表的创建，也实现了项目和数据库的联动，这就相当于架设好了项目和数据库的桥梁。不过，我们前面都做的都是单点功能的实践。

为了让你加深理解，今后轻松应用数据库来完成项目的数据管理和业务处理，接下来的两节课我会带你完成数据库配置，并结合项目综合应用数据库。

在学习数据库配置前，我们需要先了解一下项目层级关系。只有明确项目的层级关系，数据库配置的脉络才会更加清晰，也能为我们之后的项目实战做好铺垫。

项目层级框架

我们先从项目层级框架开始说起。Flask 项目通常分为三层，我们分别来看看。

第一层是**前端展示层**，包括 HTML、CSS、JS 等文件，用于展示用户界面和与用户交互。这也是直接与用户接触的内容。

第二层是**后端业务逻辑层**，用来获取数据库的数据并进行业务逻辑处理，最后把处理好的数据交给前端去渲染展示。这层通常会涉及到 ORM 对象关系映射层，用来和数据库交互。

第三层就是**数据库层**，作用是存储和管理数据，通常会用到 MySQL 这样的关系型数据库。

当然，你也要明确一个 Flask 项目并不是都需要分为三层。具体的层级分布取决于项目的复杂度和需求。有些项目可能只需要一个简单的展示层和业务逻辑层，而有些项目可能需要更多的层次，才能支持更复杂的业务逻辑和数据管理。

随着项目变得越来越庞大，组织管理各个文件会越来越困难。为了更好地组织代码，我们会给每个项目设置许多文件夹，用来存放不同的内容。

在我们的在线直播视频平台项目中，这三个层级的代码都要放在 api 文件夹下。接下来，我们明确一下文件模块具体如何管理。前端资源放在 templates 文件夹下，ORM 放在 models 文件夹下，后端业务逻辑层放在 modules 文件夹下，而数据库的基本信息配置就放在 config 文件中。

数据库的配置

梳理好了层级关系，我们这就一起来看看在项目中如何实现对数据库的应用。

数据库核心配置项

前面说了，数据库的配置我们要对应放在 config 文件下。所以我们第一步要做的，就是直接在 config 文件下新建 config.py。

第二步就是配置数据库的一些基本信息。其中包含我们的数据库 IP 地址、端口以及用户名和密码等相关信息，具体的代码如下所示。


```
1 connection = pymysql.connect(
```

 复制代码

```
2     host='127.0.0.1', # 数据库IP地址
3     port=3306, # 端口
4     user='root', # 数据库用户名
5     password='flask_project', # 数据库密码
6     database='my_project' # 数据库名称
7 )
```

这里我们将配置信息封装到一个类中，这样的好处是让代码更易维护和分离，而且这样做还能方便我们配置不同环境，比如开发环境、测试环境和生产环境。此外为了保证数据的安全性，类中还可以存储敏感信息，比如数据库用户名和密码这些信息。

后面是具体的配置代码，供你参考。

 复制代码

```
1 class Config:
2     SQL_HOST = '127.0.0.1'
3     SQL_USERNAME = 'root'
4     SQL_PASSWORD = 'flask_project'
5     SQL_PORT = 3306
6     SQL_DB = 'my_project'
7     JSON_AS_ASCII = False
8     # 数据库的配置
9     SQLALCHEMY_DATABASE_URI = f"mysql+pymysql://{SQL_USERNAME}:{SQL_PASSWORD}@{SQL_HOST}:{SQL_PORT}/{SQL_DB}"
10    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

对照前面的代码，我们来梳理一下这些配置项的具体含义。

shikey.com转载分享

shikey.com转载分享

配置项	含义说明
DEBUG	表示是否处于调试模式，设置为 True 时表示该应用程序处于调试模式。在该模式下，Flask 会输出更多的日志信息，包括运行时错误和异常信息。此外，调试模式还会禁用应用程序的 HTTP 响应缓存，这有助于发现和修复与缓存相关的问题。
SQL_HOST、SQL_USERNAME、SQL_PASSWORD、SQL_PORT 和 SQL_DB	用于连接 MySQL 数据库的配置项。这几个配置项分别表示数据库的主机名、连接数据库的用户名、连接数据库的密码、数据库监听的端口号和连接哪个数据库的名字。
JSON_AS_ASCII	表示 JSON 序列化时是否使用 ASCII 编码，在项目开发过程中，JSON 数据可能包含非 ASCII 字符，例如中文字符或其他非 ASCII 字符集的字符。默认情况下，Python 的 JSON 模块序列化这些非 ASCII 字符时会报错，因为 JSON 使用的是 Unicode 编码。为了解决这个问题，可以将 JSON_AS_ASCII 设置为 False，这样 JSON 模块就会使用 UTF-8 编码来序列化 JSON 数据，从而支持非 ASCII 字符。
SQLALCHEMY_DATABASE_URI	用于连接数据库的 URI，其中包含了数据库的主机名、用户名、密码、端口号和数据库名等信息。
SQLALCHEMY_TRACK_MODIFICATIONS	表示是否跟踪数据库模型的修改，选择“不跟踪”可以避免消耗过多内存性能。



掌握了前面这些数据库核心配置项之后，你就能轻松地实现连接配置了。常规的开发流程里，项目包括开发、测试、生产这三个阶段，所以我们为了配置的方便，可以事先定义好这三类。

复制代码

```

1 # 开发环境的config
2 class DevConfig(Config):
3     pass
4 # 生产环境的config
5 class ProConfig(Config):
6     DEBUG = False
7     SQLALCHEMY_DATABASE_URI = "mysql+pymysql://root:my_project@127.0.0.1:3306/aaa"
8 # 测试环境的config
9 class TestConfig(Config):
10    pass
11 config_dict = {
12     'dev': DevConfig,
13     'pro': ProConfig,
14     'test': TestConfig,
15 }

```


这三个类均继承了 Config 类。同时，为了方便我们的转换，我们还要定义一个字典 config_dict，用它来设置这三个字段。**需要注意的是，针对项目的不同阶段，数据库中的数据是不一样的，只有这样我们才能实现对不同阶段代码的灵活调试。**

通过 SQLAlchemy 实现 ORM 应用

搞定了数据库配置项之后，我们下面就通过 SQLAlchemy 来实现 ORM 的应用。[🔗 上节课讲过](#)，SQLAlchemy 是一个 Python 编程语言下的 SQL 工具包和 ORM 库，它提供了 SQL 表达式操作和 ORM 映射操作的工具。


同理，在我们项目开发中也是一样的流程。我们先来安装 SQLAlchemy，具体的执行命令如下所示。

```
1 pip install flask-sqlalchemy
```

 复制代码

安装成功之后我们还需要完成实例化，实例化之后才能将一些全局配置应用到具体的数据库操作中。

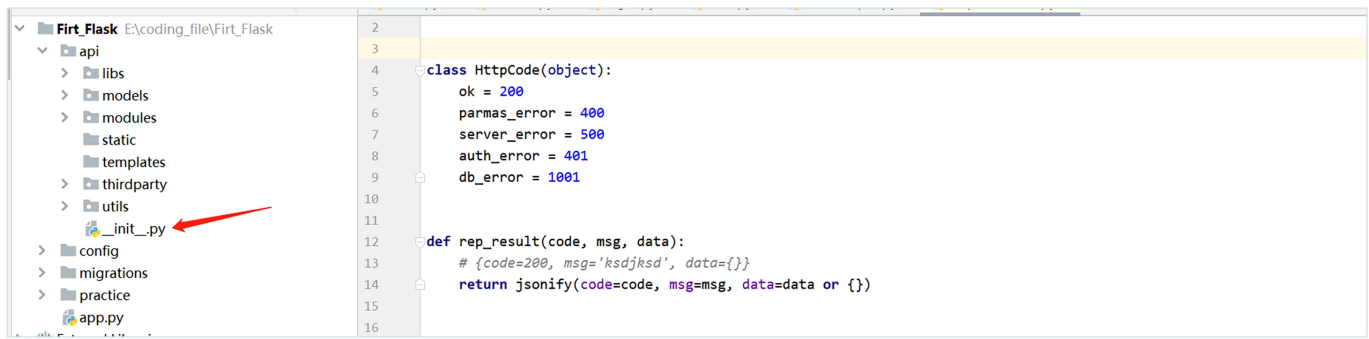
```
1 from flask import Flask
2 from flask_sqlalchemy import SQLAlchemy
3 app = Flask(__name__)
4 db = SQLAlchemy(app)
```

 复制代码

shikey.com转载分享

在项目当中，我们一般把 db 和 app 的实例化，放在 api 文件夹下的 __init__.py 文件中。

shikey.com转载分享



具体代码是后面这样，你可以边看代码，边听我为你梳理这段代码的用途。

复制代码

```
1 from flask import Flask
2 from flask_sqlalchemy import SQLAlchemy
3 from config.config import config_dict
4 db = SQLAlchemy()
5 def create_app(config_name):
6     app = Flask(__name__)
7     config = config_dict.get(config_name)
8     app.config.from_object(config)
9     db.init_app(app)
10    return app
```

这段代码我们重点要关注 `create_app` 函数。我们一般把 `create_app` 函数称为工厂函数。在 `create_app` 函数中我们可以传入一个 `config_name` 参数，方便我们切换开发、测试的数据库配置。

其中 `app.config.from_object(config)` 的作用是将 `config` 对象中定义的配置参数应用到 Flask 应用程序中，包括数据库连接信息、密钥、调试模式等等。

`db.init_app(app)` 则是将 Flask 应用程序对象传递给 SQLAlchemy 的 `db` 对象，并且绑定这两个对象，这样就能通过 SQLAlchemy 的 `db` 对象来处理与数据库有关的操作（如创建模型、查询数据、执行事务等等）了。

建立数据库表

完成了前面的各种配置，我们这就来建立数据库表。首先，我们需要在 models 文件夹中新建一个 base.py 文件，这里主要用来存放我们的模型基类。


类的创建

模型基类可以用于定义数据库表的结构和关系。在模型中，我们可以定义表的列、主键、外键等，并使用其提供的方法执行数据库操作。这样模型基类就可以通过导入的方式，被其他文件引用。具体的代码实现是后面这样。

 复制代码

```
1 from datetime import datetime
2 from api import db
3 class BaseModel:
4     """模型基类"""
5     create_time = db.Column(db.DateTime, default=datetime.now) # 创建时间
6     update_time = db.Column(db.DateTime, default=datetime.now, onupdate=datetime.now)
7     status = db.Column(db.SmallInteger, default=1) # 记录存活状态
```

下一步，我们需要在 models 文件下新建 user.py 文件，用于存放用户信息表。

 复制代码

```
1 from api import db
2 from api.models.base import BaseModel
3 class UserInfo(BaseModel, db.Model):
4     """用户信息表"""
5     __tablename__ = "user_info"
6     id = db.Column(db.Integer, primary_key=True, autoincrement=True) # 用户id
7     nickname = db.Column(db.String(64), nullable=False) # 用户昵称
8     mobile = db.Column(db.String(16)) # 手机号
9     avatar_url = db.Column(db.String(256)) # 用户头像路径
10    signature = db.Column(db.String(256)) # 签名
11    sex = db.Column(db.Enum('0', '1', '2'), default='0') # 1男 2女 0 暂不填写
12    birth_date = db.Column(db.DateTime) # 出生日期
13    role_id = db.Column(db.Integer) # 角色id
14    last_message_read_time = db.Column(db.DateTime)
15    def to_dict(self):
16        return {
17            'id': self.id,
18            'nickname': self.nickname,
19            'mobile': self.mobile,
20            'avatar_url': self.avatar_url,
```



```
21         'sex': self.sex,
22     }
```

这里的 `to_dict` 函数可以将 `UserInfo` 对象序列化为 JSON 数据，这样就能在查询时将查询到的数据放在 Flask 的响应中。这里的学习重点就是掌握创建方法，后面随着项目的深入，我们还会根据项目规模在 `models` 文件下灵活添加其他的文件。

生成数据库表

在创建数据库表类之后，数据库表的创建还差最后一个环节——执行代码生成数据库表。

首先，我们需要在项目的根目录下新建 `app.py` 文件。`app.py` 文件能将多个 Flask 脚本整合到一个文件中，简化 Flask 应用程序的管理和部署。

我们只要在 `app.py` 文件下做好相关函数的导入和初始化工作，就可以把 `app.py` 作为 Flask 的命令行接口，通过它运行不同的 Flask 命令，高效完成数据库迁移、创建超级用户、导出数据等操作。

紧接着，我们需要在 `api` 文件中导入 `db` 和 `create_app` 工厂函数。

 复制代码

```
1 from api import db, create_app
2 from flask_script import Manager
3 from flask_migrate import Migrate
4 app = create_app('dev')
5 # 管理器对象可以用来提供各种管理命令，例如启动应用程序、创建数据库表、导入数据等等
6 manager = Manager(app)
7 Migrate(app, db)
8 def create_db():
9     db.create_all()
10 if __name__ == '__main__':
11     manager.run()
```

对照前面的代码，我来讲解一下重点。我们导入了所需的库和模块。`api` 模块中包含了 Flask 应用程序和数据库连接，`create_app` 函数用来创建应用程序实例，`Manager` 类用于创建管

理命令的对象，Migrate 类用于处理数据库迁移。

之后，我们需要使用 create_app 函数创建一个名为 dev 的开发环境应用程序实例。同时使用 Manager 类创建一个名为 manager 的管理命令对象，该对象可以用来提供各种管理命令，例如启动应用程序、创建数据库表、导入数据等等。


第七行代码表示我们用 Migrate 类来初始化 Flask-Migrate，目的是方便处理数据库迁移。

之后第八行代码用到了 create_db 函数，它的主要作用是使用 db.create_all() 方法创建所有定义的数据库表。

最后是第十一行代码，我们在主程序中，使用 manager.run() 方法运行管理命令对象，目的是启动应用程序并监听管理命令。

理解了前面这些执行代码后，我们还需要生成 migrations 文件，这是为了方便后面变更和管理数据库结构，防止出现误操作或异常操作。执行命令和对应的执行效果图是后面这样。执行后面的语句后，就会自动生成 migrations 文件夹。

```
1 flask db init
```

 复制代码

```
(base) E:\coding_file\Firt_Flask>flask db init
Creating directory E:\coding_file\Firt_Flask\migrations ... done
Creating directory E:\coding_file\Firt_Flask\migrations\versions ... done
Generating E:\coding_file\Firt_Flask\migrations\alembic.ini ... done
Generating E:\coding_file\Firt_Flask\migrations\env.py ... done
Generating E:\coding_file\Firt_Flask\migrations\README ... done
Generating E:\coding_file\Firt_Flask\migrations\script.py.mako ... done
```

 极客时间

shikey.com转载分享

文件夹中包含文件 versions，未来每次修改数据库表结构时，这个文件都会生成一个脚本文件，用来记录每次数据库表的变化。其他的文件都是一些配置文件，你暂时不需要关心。

自动生成 migrations 文件夹后，我们还要在终端中输入后面的命令，命令的作用你可以参考代码注释。

```
1 flask db migrate # 生成迁移版本，保存到迁移文件夹中
2 flask db upgrade # 执行迁移
```

命令输入后，我们打开 Navicat，就会发现 my_project 数据库中多出了一个 alembic_version 数据库表。

```
Terminal: Local x +
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

(base) E:\coding_file\Firt_Flask>flask shell
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] on win32
App: api
Instance: E:\coding_file\Firt_Flask\instance
>>> db.create_all()
```

这个表中只有一个字段，是用来存储版本号。我们可以通过查询 alembic_version 表的版本号来确定当前数据库所处的状态。此外，alembic_version 表还可以用于在不同版本之间实现数据库迁移操作。

完成前面这些初步操作之后，我们还需要创建 user_info 数据库表。具体操作就是在 pycharm 的 Terminal 终端，输入后面的命令。

```
1 flask shell
2 db.create_all()
```

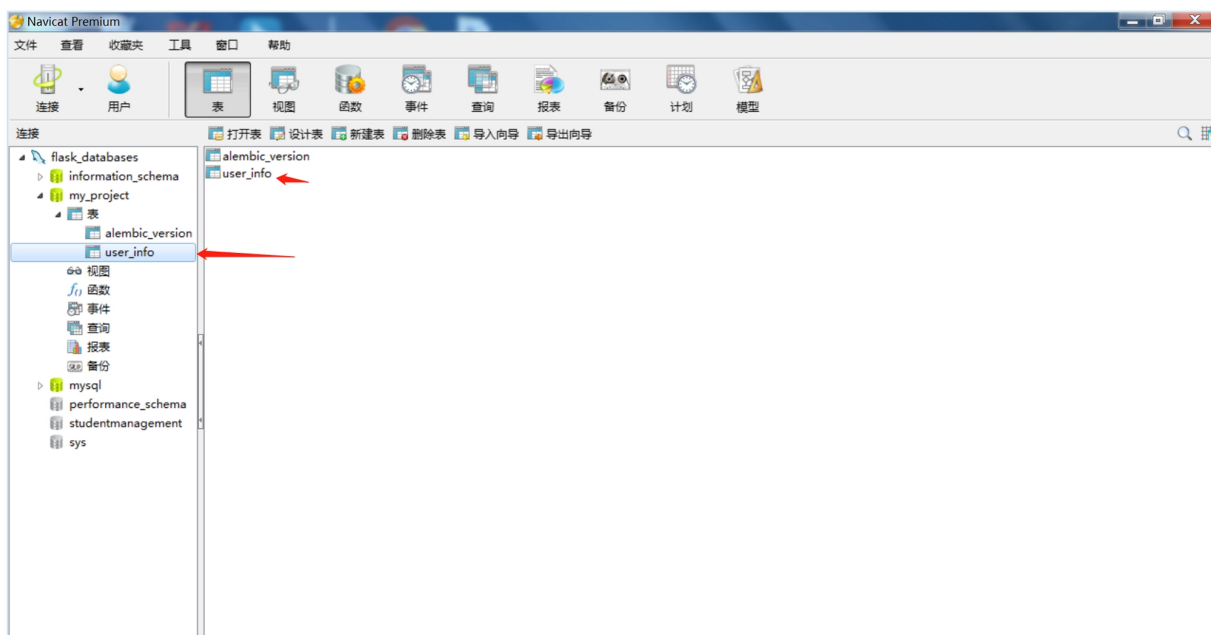
shikey.com转载分享

```
Terminal: Local x +
Microsoft Windows [版本 6.1.7601]
版权所有 (c) 2009 Microsoft Corporation。保留所有权利。

(base) E:\coding_file\Firt_Flask>flask shell
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)] on win32
App: api
Instance: E:\coding_file\Firt_Flask\instance
>>> db.create_all()
```

这里我们输入 Flask Shell，就会进入一个交互式的 Python 编程，自由使用所有的 Flask 应用程序中定义的变量、函数和对象以及数据库模型、路由和视图函数等等。这个环节一般用来测试和调试应用程序。

进入交互式命令后，还需要输入 `db.create_all()` 来创建所有的数据库表。执行完成后，只需要 `Ctrl+Z` 退出执行即可。此时我们再次打开 Navicat，就会看到数据库表 `user_info` 已经创建好了。



不过仔细看一下就会发现，此时 `user_info` 中还没有任何信息，因为我们目前只完成了表的创建，还没有新增数据，这部分内容下节课我们再继续探索。

总结

又到了课程的尾声，我们来回顾一下今天的学习重点。

想要在项目中灵活应用数据库，熟悉项目层级框架并做好每个模块的配置管理这些准备步骤非常重要。通常一个 Flask 项目会分成三层：前端展示层、后端业务逻辑层和数据库层，明确了它们的作用，我们才能更好地管理项目。

对比前面课程的单点实践，你会发现今天在数据库配置阶段，有些配置存在一些细微变动，这些只有落地项目的时候才能感受到。这个过程你需要重点关注 `create_app` 函数的用法，尤其是其中各个参数的作用，这样才能高效配置数据库。

另一个学习重点就是数据库表建立。结合 user.py 文件的案例，我们学习了如何通过 db.Model 模型基类中的 db.Column 类创建数据库。你需要明确 app.py 文件在项目中的定位和作用，并且牢牢掌握数据库表创建的实现代码，把握其思路，这样才能从容应对类似的建表需求。

思考题

Flask 项目开发包括开发、测试、生产这三个阶段，你觉得是否需要对应有三个数据库来支撑开发？还是说单个数据库就可以满足？

欢迎你在留言区和我交流互动，如果这节课的内容对你有帮助，别忘了分享给身边更多朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (6)



Forest

2023-06-20 来自四川

在项目开发中，生产环境的数据库肯定是独立的，开发和测试可以使用同一个数据库；在开发和测试过程中，会添加很多测试数据，对于生产环境而言，这部分是脏数据且没有任何实际意义

作者回复：总结的非常准确，感谢分享，为你点赞！



👍 1



闻道

2023-07-07 来自广东

遇到是这个报错ModuleNotFoundError: No module named 'flask._compat'，看着是Flask版本问题，通过这里面的方法解决<https://stackoverflow.com/questions/67538056/flask-script-from-flask-compat-import-text-type-modulenotfounderror-no-module>



Pick Monster ...

2023-07-05 来自陕西

flask db init

报错Error: No such command 'db'.

提示在api __init__中没有db，这个为什么

作者回复: 你好同学，db 和 app 的实例化要在 api 文件夹下的 __init__.py 文件中，在此之前要安装 SQLAlchemy，出现这个问题是因为没有正确导入模块，在__init__函数中使用了未定义的数据库命令。请检查代码中是否存在语法错误，特别是与数据库相关的语句。



未来已来

2023-06-29 来自广东

不知道为啥，执行完`flask shell`及`db.create_all()`之后，没有正常建表

作者回复: 你可以检查一下是否正确配置了数据库连接，确保你正确设置了数据库连接数据库地址以及用户名和密码。其次，如果数据库中已经存在其他表或约束，例如外键约束、唯一约束等，可能影响新表的创建。你可以尝试在创建新表之前删除或修改这些约束，或者创建一个新的数据库进行测试。如有问题可以追评。

共 2 条评论 >



Geek_840593

2023-06-21 来自重庆

老师：app.py这里不是有段代码创建数据库表

.....

```
def create_db():
    db.create_all()
```

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

为什么我们还是手动在Terminal输入一次：

flask shell

db.create_all()

作者回复: 是这样的，在 app.py 文件中，确实有create_db()方法，它用于创建数据库表，通过 db.create_all() 方法来创建所有定义的模型类的表。然而，当你运行 Flask 应用程序时，create_db() 函数并不会自动执行。你需要在 Terminal 中手动输入 flask shell 命令，然后在 Python shell 中执行 db.create_all() 语句来创建数据库表。手动执行 db.create_all() 的原因是在 Flask 应用程序启动时，它需要加载整个应用程序，包括数据库模型和相关的表。在这个过程中，Flask 会自动检测数据库中是否

存在相关的表，如果不存在，则会抛出一个异常。因此，为了确保应用程序能够正确启动，我们需要手动执行 `db.create_all()` 来创建数据库表。



peter

2023-06-15 来自北京

Q1: 前缀f是什么意思?

```
SQLALCHEMY_DATABASE_URI = f"mysql+pymysql://{SQL_USERNAME}:{SQL_PASSWORD}@{SQL_HOST}:{SQL_PORT}/{SQL_DB}"
```

字符串前面的前缀f表示什么?

Q2: pass是什么意思?

```
# 开发环境的configclass DevConfig(Config): pass
```

函数体的pass是什么意思? 省略吗?

作者回复: 1、f 前缀是一种字符串格式化语法，它允许我们在运行时将变量插入到字符串中。

2、pass是一个空语句，它什么也不做，只是占据一个语句位置。

共 2 条评论 >



shikey.com转载分享

shikey.com转载分享