

29 | 后端功能接口实战（一）：后端接口该如何开发？

2023-06-28 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。

在前面课程中，我们已经学过了后端接口开发的前置知识，并且我们也通过一些功能案例尝试了接口开发。

接下来的两节课，我们就结合直播视频平台的需求，来完成系统化的接口开发，让你掌握独立完成功能接口开发的能力。这节课我们先来梳理开发流程规范和接口需求，并通过创建 passport 认证模块编写登录注册相关的接口代码。

shikey.com 转载分享

接口开发流程规范

我们先了解一下接口开发的流程规范，让你对项目及开发整体的流程更加熟悉。企业级的项目开发都要遵循开发规范，目的是让开发操作更规范，提高开发效率。流程规范一共有五条，我们依次来看看。

一是需求明确。在开发接口之前，我们首先要明确接口的功能需求和数据需求。这包括接口需要实现的功能、需要接收的参数以及返回的数据格式。这些如果不在开发前明确，很容易导致我们后期返工。

二是使用框架。使用现有的框架，比如 Flask 或 Django 这样的框架可以大大提高开发效率。这些框架已经为你处理了许多底层的细节，你只需要关注业务逻辑即可。我们在项目中使用的就是 Flask 框架。

三是设计 RESTful API。RESTful 是一种常用的 API 设计风格，它简单易用，容易理解，也方便与前端进行交互。这一部分我们在 Flask-RESTful 这节课已经讲过了，也为你打好了基础。

四就是进行接口测试。接口测试能够保证开发功能的实用性和完整性。通过这一步，我们就能够在投入使用前提前检验我们的接口功能，保证功能顺利上线。具体就是使用 Postman 测试我们的 API 接口。这些工具无需等待前端的配合，用起来很方便。

最后是第五点，编写接口文档，这是企业里每个研发团队必须要做的规范化管理。良好的文档可以帮助其他开发者理解和使用你的接口，方便团队管理协作，也有利于接口的后期维护。

接口需求梳理

明确了开发规范，我们再结合直播视频平台的功能梳理一下接口需求的梳理，为后续开发实战做好准备。

只有理顺了接口需求，我们才能更清晰数据库表设计以及接口的功能实现方式。我们这就来看看在线视频平台里最有代表性的几个必备接口，你可以参考后面的表格来看看。

shikey.com转载分享

接口名称	需求梳理
登录与注册接口	用户通过注册接口创建账户，并提供相关的个人信息（如用户名、密码、电子邮件地址等）。一旦用户注册，即可使用登录接口，通过验证用户名和密码的方式访问其账户。
视频列表	视频列表接口通常返回可供用户观看的视频列表。根据平台内视频类别呈现，可根据发布时间先后、用户喜爱数等维度展示视频，供用户选择观看。
视频详情接口	当用户从列表点击某个视频，这时候会进入到视频详情页，接口需要返回视频地址、视频详细介绍、作者相关信息、视频相关数据等，保证用户对视频相关信息足够了解。
个人信息管理接口	主要作用是实现用户个人信息的管理，也有利于平台对用户分类管理。通过该接口，用户能够查看和更新个人信息（例如更改用户名、密码、电子邮件地址或者其他个人详情）。这个接口还可能包括安全设置，如两步验证或者密码重置功能。
点赞、关注功能实现接口	实现用户跟其他用户或内容的交流互动。点赞接口通常让用户能够对特定的内容（如视频、评论等）表达喜欢。关注接口能让用户关注其他用户或者频道，获得关注对象的最新更新动态。



明确了功能接口的开发规范和需求情况，接下来就是实操环节。

项目接口实战

在整个后端实战课程中，我们每一个模块的学习和应用都是为了最终项目接口开发做准备。课程里我放的是核心代码，完整的代码你可以通过 [Gitee 链接](#) 获取。

config 配置管理

首先需要搭建整体结构。我们在 config 文件中完成配置管理。后面是创建配置基类的代码实现。

shikey.com转载分享

复制代码

```

1 class Config:
2     DEBUG = True
3     LEVEL_LOG = logging.INFO
4     SECRET_KEY = 'slajfasfjkajfj'
5     SQL_HOST = '127.0.0.1'
6     SQL_USERNAME = 'root'
7     SQL_PASSWORD = 'root'
8     SQL_PORT = 3306
9     SQL_DB = 'my_project'
10    JSON_AS_ASCII = False
11    # 数据库的配置

```

shikey.com转载分享


```

12  SQLALCHEMY_DATABASE_URI = f"mysql+pymysql://{SQL_USERNAME}:{SQL_PASSWORD}@{SQL_
13  SQLALCHEMY_TRACK_MODIFICATIONS = False
14  REDIS_HOST = '127.0.0.1'
15  REDIS_PORT = 6379
16
17
18  # 指定session使用什么来存储
19  SESSION_TYPE = 'redis'
20  # 指定session数据存储在后端的位置
21  SESSION_REDIS = StrictRedis(host=REDIS_HOST, port=REDIS_PORT)
22  # 是否使用secret_key签名你的session
23  SESSION_USE_SIGNER = True
24  # 设置过期时间, 要求'SESSION_PERMANENT', True。而默认就是31天
25  PERMANENT_SESSION_LIFETIME = 60 * 60 * 24 # 一天有效期

```

完成基本配置之后，在 config 文件中我们还需要做不同环境的配置。配置类的作用就是提供一个分离的环境配置逻辑的机制，让我们无需修改代码，就可以在不同的环境中轻松使用不同的配置参数。这里我们需要分成测试环境、开发环境和生产环境这三种模式来设置。

首先来看测试环境配置类的具体代码。


 复制代码

```

1  class TestConfig(Config):
2      pass

```

然后是开发环境配置类具体代码。


 复制代码

```

1  class DevConfig(Config):
2      pass

```

最后是生产环境配置类具体代码。

 复制代码

```

1  class ProConfig(Config):
2      LEVEL_LOG = logging.ERROR
3      DEBUG = False

```

主程序编写

搞定配置管理以后就可以编写主程序了。我们先完成 app 对象的实例化，这一步是创建 Flask 应用实例的重要步骤，它包含了应用的各种属性和方法，用于构建 Web 应用程序。通过 app 对象，我们可以定义应用的路由、添加蓝图、初始化扩展等功能，进而构建出完整的应用程序。

具体做法就是在项目中创建接口 api 包目录，然后在 `__init__.py` 中创建 app 对象，具体代码如下所示。

[复制代码](#)

```
1 def create_app(config_name):
2     app = Flask(__name__)
3     config = config_dict.get(config_name)
4     setup_log(log_file='logs/root.log', level=config.LEVEL_LOG)
5     app.config.from_object(config)
6     db.init_app(app)
7     global redis_store
8     redis_store = StrictRedis(host=config.REDIS_HOST, port=config.REDIS_PORT, dec
9     register_bp(app)
```

这段代码中，我们完成了后面这四步动作。

1. 工厂模式下，完成不同环境下配置信息导入。

2. 增加 app 日志管理。

3. 初始化数据库，并关联 app 对象。

4. 增加全局 redis 链接对象。

完成上面的操作之后，我们还需要增加注册蓝图方法，完成不同功能模块的管理。我以 passport_blu 模块为代表案例，带你看看具体的代码实现。其他模块增加蓝图注册方法和这里类似，只不过对应的模块名不一样。

```
1 def register_bp(app)
2     from api.modules.passport import passport_blu
3     app.register_blueprint(passport_blu)
4
```

到这里我们就完成了实例化 app 对象，完成了主程序的各个配置项，接下来我们就来完成 moduels 包的配置与开发。

创建数据库表

在这一部分我们将要完成创建 passport 认证模块，并且编写好登录和注册接口功能的代码。

首先我们需要在 models 下面的 base.py 文件中，创建模型基类。如何创建模型基类，我们在数据库实战 [那节课](#) 已经详细讲过了，这里我们直接看具体代码。

```
1 class BaseModels:
2     """模型基类"""
3     create_time = db.Column(db.DateTime, default=datetime.now) # 创建时间
4     update_time = db.Column(db.DateTime, default=datetime.now, onupdate=datetime.now)
5     status = db.Column(db.SmallInteger, default=1) # 记录存活状态
6
7     def add(self, obj):
8         db.session.add(obj)
9         return session_commit()
10
11     def update(self):
12         return session_commit()
13
14     def delete(self):
15         self.status = 0
16         return session_commit()
```

完成模型基类的创建后，我们就要创建用户登录表，同样还是在 models 文件下的 user.py 文件中创建。

```

1 class UserLogin(BaseModels, db.Model):
2     """用户登录表"""
3     __tablename__ = "user_login"
4
5     id = db.Column(db.Integer, primary_key=True, autoincrement=True) # 用户id
6     mobile = db.Column(db.String(16), unique=True, nullable=False) # 手机号
7     password_hash = db.Column(db.String(128), nullable=False) # 加密的密码
8     user_id = db.Column(db.Integer) # 用户id
9     last_login = db.Column(db.DateTime, default=datetime.now) # 最后一次登录时间
10    last_login_stamp = db.Column(db.Integer) # 最后一次登录时间
11
12    @property
13    def password(self):
14        raise AttributeError('密码属性不能直接获取')
15
16    @password.setter
17    def password(self, value):
18        self.password_hash = generate_password_hash(value)
19
20    # 传入的是明文, 校验明文和数据库里面的hash之后密码 正确true
21    def check_password(self, password):
22        return check_password_hash(self.password_hash, password)

```

下一步是项目中的用户管理，在登录成功之后要展示用户信息。下面的 UserInfo 类主要用来创建用户信息表。

```

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

```



```

17         last_read_time = self.last_message_read_time or datetime(1900, 1, 1)
18         return Message.query.filter_by(recipient_id=self.id).filter(
19             Message.timestamp > last_read_time).count()
20
21     def to_dict(self):
22         return {
23             'id': self.id,
24             'nickname': self.nickname,
25             'mobile': self.mobile,
26             'avatar_url': self.avatar_url,
27             'sex': self.sex,
28         }


```

接口开发

完成了模型基类和表的创建之后，我们还要实现具体的功能接口。

我们先来梳理一下注册接口的具体功能实现。用户注册的核心逻辑就是，在用户完成一系列信息的录入后，点击注册按钮，然后将用户信息提交到数据库中。

我们直接在 modules 文件下的 view.py 文件中实现注册接口。

 复制代码

```

1 @passport_blu.route('/register', methods=['POST'])
2 def register():
3     """
4     注册接口
5     :return: code msg
6     """
7     data_dict = request.form
8     mobile = data_dict.get('mobile')
9     password = data_dict.get('password')
10    img_code_id = data_dict.get('img_code_id') # cur_id
11    img_code = data_dict.get('img_code') # 填写的code
12
13    if not all([mobile, password, img_code_id, img_code]):
14        return error(code=HttpCode.parmas_error, msg='注册所需参数不能为空')
15
16    # 2.1验证手机号格式
17    if not re.match('1[3456789]\\d{9}', mobile):
18        return error(code=HttpCode.parmas_error, msg='手机号格式不正确')
19
20    # 3.通过手机号取出redis中的验证码

```



```

21     redis_img_code = None
22     # 从redis取出img_code_id对应的验证码
23     try:
24         redis_img_code = redis_store.get(f'img_code: {img_code_id}')
25     except Exception as e:
26         current_app.logger.error(e)
27
28     if not redis_img_code:
29         return error(HttpStatusCode.parmas_error, 'redis图片验证码获取失败')
30
31     if img_code.lower() != redis_img_code.lower():
32         return error(HttpStatusCode.parmas_error, '图片验证码不正确')
33
34     user_info = UserInfo()
35     user_info.mobile = mobile
36     user_info.nickname = mobile
37     user_info.add(user_info)
38
39     user_login = UserLogin()
40     user_login.mobile = mobile
41     user_login.password = password
42     user_login.user_id = user_info.id
43     user_login.add(user_login)
44
45     return success('注册成功')

```

在前面这段代码中，通过 request.form 获取到用户信息之后，我们分别进行了表单非空验证、手机号格式认证以及图片验证码的认证。用户必须完成以上认证之后，才可以完成注册。

我们需要特别留意一下图片验证码的实现。这里我们主要借助三方包来生成简单的验证码接口。

shikey.com转载分享

 复制代码

```

1 @passport_blu.route('/image_code')
2 def img_code():
3     """
4     生成图像验证码
5     :return: 图片的响应
6     """
7     # 1.获取请求参数,args是获取?后面的参数
8     cur_id = request.args.get('cur_id')
9     pre_id = request.args.get('pre_id')
10    # 2.生成图片验证码

```

shikey.com转载分享

```

11     name, text, img_data = captcha.captcha.generate_captcha()
12     # 3.保存到redis
13     try:
14         redis_store.set(f'img_code: {cur_id}', text, IMAGE_CODE_REDIS_EXPIRES)
15         # 判断是否有上一个uuid,如果存在则删除
16         if pre_id:
17             redis_store.delete(f'img_code: {pre_id}')
18     except Exception as e:
19         current_app.logger.error(e)
20         return error(HttpCode.db_error, 'redis存储失败')
21     # 4. 返回图片验证码
22     response = make_response(img_data)
23     response.headers["Content-Type"] = "image/jpg"
24
25     return response

```

前面这段代码的作用是生成一个验证码，并将其存储在变量 name、text 和 img_data 中。通过调用 captcha.captcha.generate_captcha() 方法（这是一个生成验证码的函数），返回一个元组 (name, text, img_data)。其中 name 是验证码的名称，text 是验证码的文本，img_data 是验证码的图像数据。

当然，我们也不能忽略后面对应的异常处理，这样才能保证程序稳定执行。

用户注册接口的开发中，有个非常重要的功能——用户判重。

比方说，一个手机号我们不支持多次注册。这一步实现的逻辑是这样的：我们要在用户点击注册或完成手机号输入之后，就通过查询现有用户手机号来判断是否重合。如果查询到相同的则注册失败，相反直接注册成功。你可以结合后面的代码实现来加深理解。

shikey.com转载分享

复制代码

```

1 @passport_blu.route('/check_mobile', methods=['POST'])
2 def check_mobile():
3     """
4     验证手机号
5     # 请求路径: /passport/check_mobile
6     # 请求方式: POST
7     # 请求参数: mobile
8     :return: code,msg
9     """
10    data_dict = request.form
11    mobile = data_dict.get('mobile')

```

```
12
13     try:
14         users = UserLogin.query.all()
15     except Exception as e:
16         current_app.logger.error(e)
17         return error(code=HttpCode.db_error, msg='查询用户信息异常')
18
19     if mobile in [i.mobile for i in users]:
20         return error(code=HttpCode.parmas_error, msg='手机号已存在, 请重新输入')
21
22     return success(msg=f'{mobile}, 此手机号可以使用')
```

这段代码中，最核心的部分就是在获取到 mobile 之后，通过 UserLogin.query.all() 方法查询数据，根据返回的数据来判断注册手机号。

到这里我们就完成了用户的注册功能，至于登录功能。我们上节课讲认证机制的时候已经详细说过，从生成 Token 到用户鉴权的全过程相信你已经非常熟悉了，完整代码你同样可以参考 [Gitee](#)。

总结

又到了课程的尾声，接下来我们一起对这节课学到的内容做个总结。

前面的课程中我们或多或少的都有涉及到接口实现的方法，这节课我们以用户注册的接口实现为例，更加体系化地实现了完整的功能开发。

具体实战前，我们先梳理了开发流程规范和接口需求。规范的流程不但能提高效率，也能更好地实现团队合作。接口需求能让我们明确之后要做哪些功能，为之后的实现环节做好预热。

接口开发阶段，我们从 config 项目配置管理到主程序编写，你必须要掌握实例化 app 对象的创建代码中每一项的作用。而在模型基类和用户表创建这一部分里，我们要注意**提前梳理好每一模块的字段信息以及对应的字段类型**。接口实现过程中一定要注意有业务逻辑实现和异常处理，只有全面考虑，才能保证程序的稳定执行。

这节课，我们以注册接口为案例带你体验了接口系统化开发的过程。用户相关的接口开发也是一样的实现方法，相信你在掌握了注册的接口实现之后，应对其他用户相关的接口实现也会非常轻松。希望你课后对照配套代码多多练习，巩固学习效果。

思考题

在课程中注册的时候我们做了图形验证，如果通过短信认证的方式来实现注册，你有什么好的想法分享么？

欢迎你在留言区和我交流互动，如果这节课对你有启发，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

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。

shikey.com转载分享

shikey.com转载分享