AI 掘金头条–用户模块

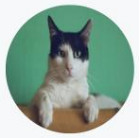
用户注册、用户登录、获取和修改用户信息



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

用户注册



新闻资讯

用户名 请输入用户名

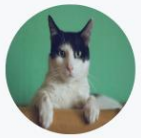
密码 请输入密码

确认密码 请再次输入密码

注册

已有账号？去登录

用户登录



新闻资讯

用户名 请输入用户名

密码 请输入密码

登录

测试账号：admin

测试密码：123456

个人信息

头像 

用户名 admin

账号ID ID: heima-0e9ae

个人简介 这是一个测试账号 >

修改密码 >

接口实现流程



- ✓ 定义 APIRouter 实例
- ✓ 注册路由 include_router()
- ✓ 参照接口规范文档




```
class Base(DeclarativeBase):
  pass

class Category(Base):
  __tablename__="news_category"
  ......
```

- ✓ 参照数据库表



- ✓ select(模型类)
- ✓ add()
- ✓ update()
- ✓ delete()



- ✓ Depends 注入数据库依赖
- ✓ 调用查询逻辑
- ✓ 响应结果

用户注册

1. 用户注册

- 接口地址：`POST /api/user/register`
- 请求参数：

参数名	类型	必填	说明
username	string	是	用户名
password	string	是	密码

- 请求示例：

```
{
  "username": "example_user",
  "password": "example_password"
}
```

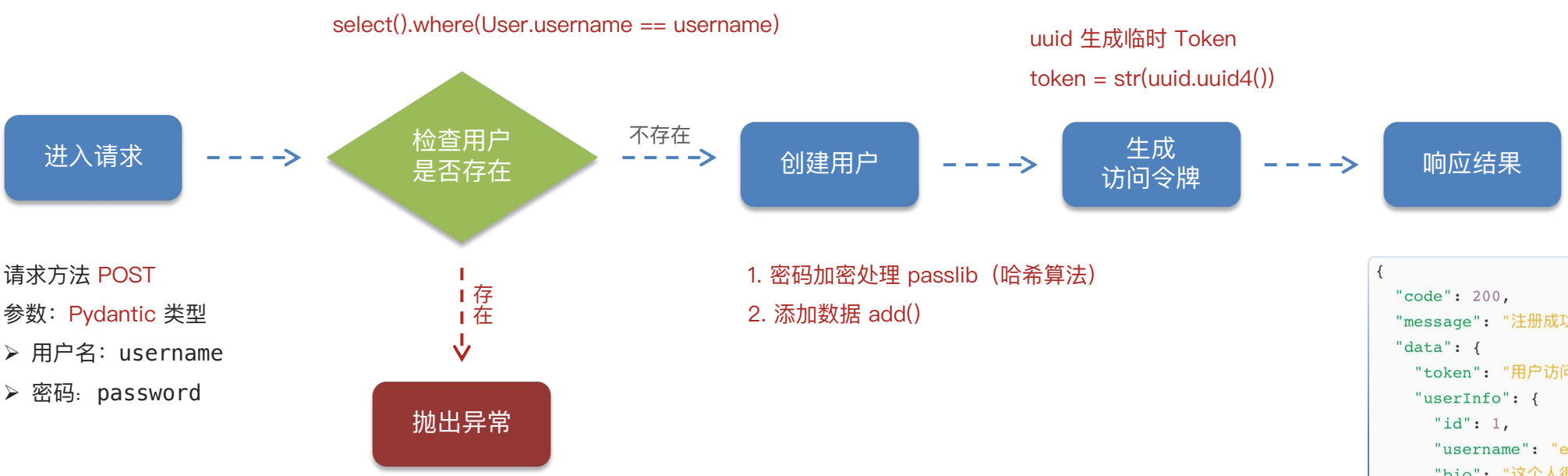
- 响应示例：

```
{
  "code": 200,
  "message": "注册成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "example_user",
      "bio": "这个人很懒，什么都没留下",
      "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg"
    }
  }
}
```

```python
router = APIRouter(prefix="/api/user", tags=["user"])

@router.post("/register")
async def register_user(user_data: UserRegisterRequest):
  return {
    "code": 200,
    "message": "注册成功",
    "data": {
      "token": "token",
      "userInfo": {
        "id": "user.id",
        "username": "user.username",
        "bio": "user.bio",
        "avatar": "user.avatar"
      }
    }
  }
```

用户注册



请求方法 POST

参数：Pydantic 类型

- 用户名：username
- 密码：password

1. 密码加密处理 passlib（哈希算法）
2. 添加数据 add()

```
{
  "code": 200,
  "message": "注册成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "example_user",
      "bio": "这个人很懒，什么都没留下",
      "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg"
    }
  }
}
```

passlib 密码加密

01

安装

```
pip install
"passlib[bcrypt]==1.7.4"
```

02

创建加密上下文

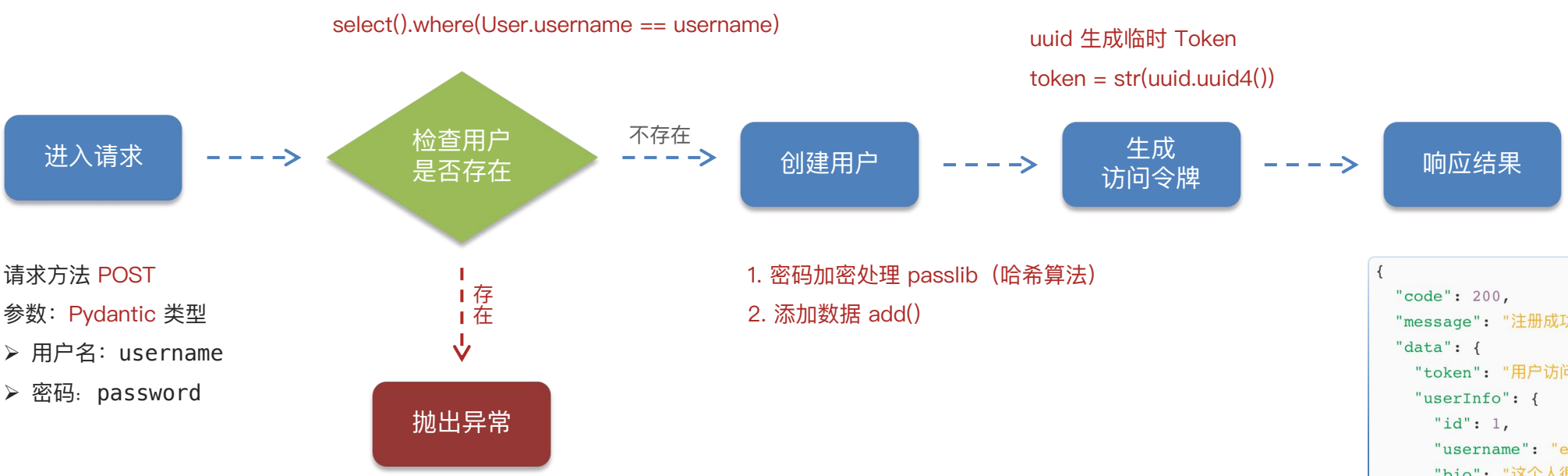
```
pwd_context = CryptContext(
    schemes=["bcrypt"],
  deprecated="auto"
)
```

03

加密或校验

- `pwd_context.hash(password)`
- `pwd_context.verify(plain_password, hashed_password)`

用户注册



请求方法 POST

参数：Pydantic 类型

- 用户名：username
- 密码：password

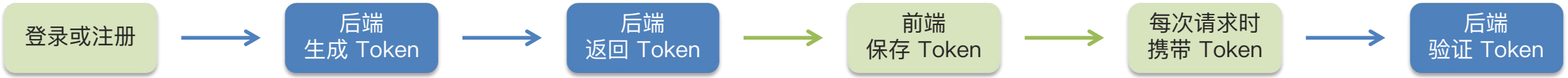
1. 密码加密处理 passlib（哈希算法）
2. 添加数据 add()

```
{
  "code": 200,
  "message": "注册成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "example_user",
      "bio": "这个人很懒，什么都没留下",
      "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg"
    }
  }
}
```

Token

Token：是服务器发给客户端的一段字符串，用来在后续请求中证明“你已经登录过了”

作用：解决 HTTP 是无状态的问题，在每次请求中“自我证明身份”



Token

Token 在请求中的位置：请求头

```
Authorization: Bearer <token>
```

- Authorization：专门用来放身份信息
- Bearer：表示“持有者令牌”
- <token>：真正的身份凭证

封装通用成功响应格式

```
{
  "code": 200,
  "message": "注册成功",
  "data": {
    "token": "用户访问
    "userInfo": {
      "id": 1,
      "username": "ex
      "bio": "这个人很
      "avatar": "http
    }
  }
}
```

```
{
  "code": 200,
  "message": "登录成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "exa
      "nickname": null
      "avatar": "https
      "bio": "这个人很懒
    }
  }
}
```

```
{
  "code": 200,
  "message": "success",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar": "https://fastly.
    "gender": "unknown",
    "bio": "这个人很懒，什么都没留
  }
}
```

```
{
  "code": 200,
  "message": "更新成功",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar": "https://fastly.jsdelivr.net/npm/@vant/assets/cat.jpeg",
    "gender": "unknown",
    "bio": "这是我的个人简介"
  }
}
```

封装通用成功响应格式

01

抽取响应结果
utils/xx.py

```python
def success_response(message: str="success", data=None):
  content = {"code": 200, "message": message, "data": data}
  # 把任何FastAPI、Pydantic、ORM 对象转换成可以被 JSON 安全序列化的数据结构
  return JSONResponse(content=jsonable_encoder(content))
```

02

定义数据类型

```python
class UserAuthResponse(BaseModel):
  token: str
  user_info: ...
  model_config = ConfigDict(
    from_attributes=True, # 自动序列化 user_info 中的 ORM 对象
  )
```

03

调用函数
响应结果

```python
response_data = UserAuthResponse(
  token=token,
  user_info=UserInfoResponse.model_validate(user)
)
return success_response(message="注册成功", data=response_data)
```

全局异常处理器

全局异常处理器（Global Exception Handler）是注册在 FastAPI 应用级别的异常处理函数，用于捕获业务层、数据库层以及系统层抛出的异常，并以统一的响应格式返回给前端。

异常：

- SQL 错误
- 外键关联失败
- 数据库连接异常
- 提交事务失败
- ……

单击此处添加标题

Step 01

定义异常处理器（函数）



Step 02

全局注册异常处理器

`app.add_exception_handler()`

注册顺序：

1. 业务异常
2. 数据约束异常(子类)
3. 数据库异常(父类)
4. 所有异常

用户登录

2. 用户登录

- 接口地址：`POST /api/user/login`
- 请求参数：

参数名	类型	必填	说明
username	string	是	用户名
password	string	是	密码

- 请求示例：

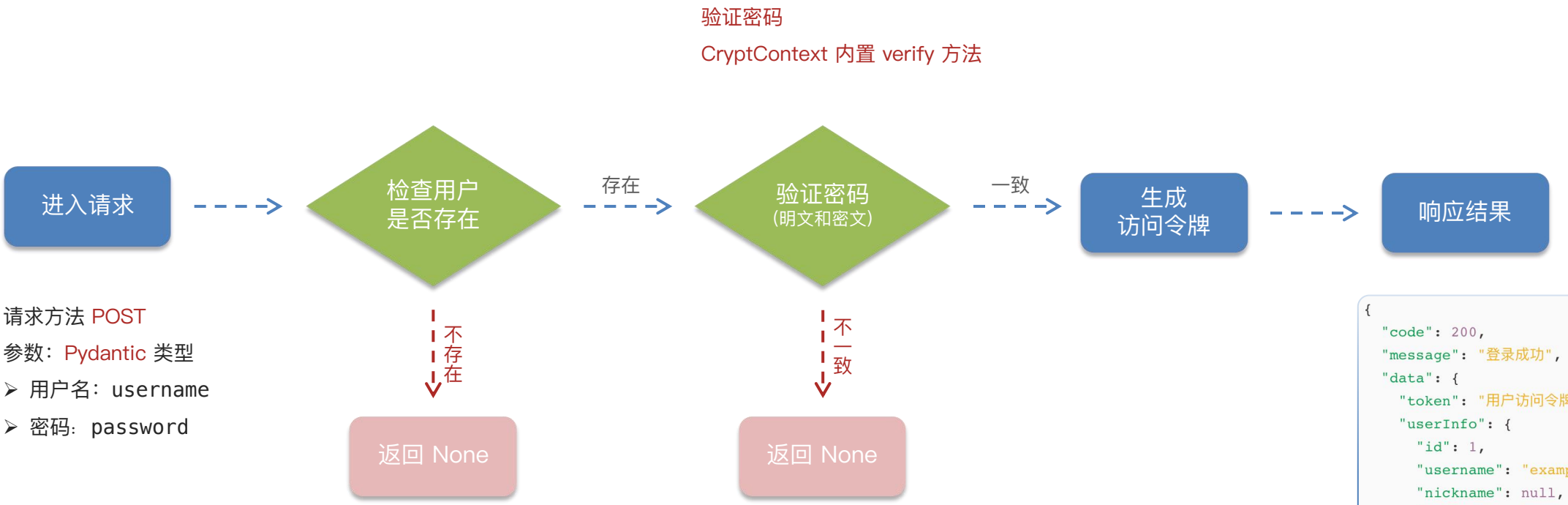
```
{
  "username": "example_user",
  "password": "example_password"
}
```

- 响应示例：

```
{
  "code": 200,
  "message": "登录成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "example_user",
      "nickname": null,
      "avatar":
"https://fastly.jsdelivr.net.../cat.jpg",
      "bio": "这个人很懒，什么都没留下"
    }
  }
}
```

```python
@router.post("/login")
async def login_user(login_data: UserRequest):
  return {
    "code":200,
    "message":"登录成功",
    "data":{
      "token": "token",
      "userInfo": {
      "id": "user.id",
      "username": "user.username",
      "nickname": "user.nickname",
      "avatar": "user.avatar",
      "bio": "user.bio"
      }
    }
  }
```

用户登录



请求方法 POST

参数：Pydantic 类型

- 用户名：username
- 密码：password

```
{
  "code": 200,
  "message": "登录成功",
  "data": {
    "token": "用户访问令牌",
    "userInfo": {
      "id": 1,
      "username": "example_user",
      "nickname": null,
      "avatar":
"https://fastly.jsdelivr.net.../cat.jpg",
      "bio": "这个人很懒，什么都没留下"
    }
  }
}
```

获取用户信息

3. 获取用户信息

- **接口地址**: `GET /api/user/info`
- **请求头**: 需要认证
- **响应示例**:

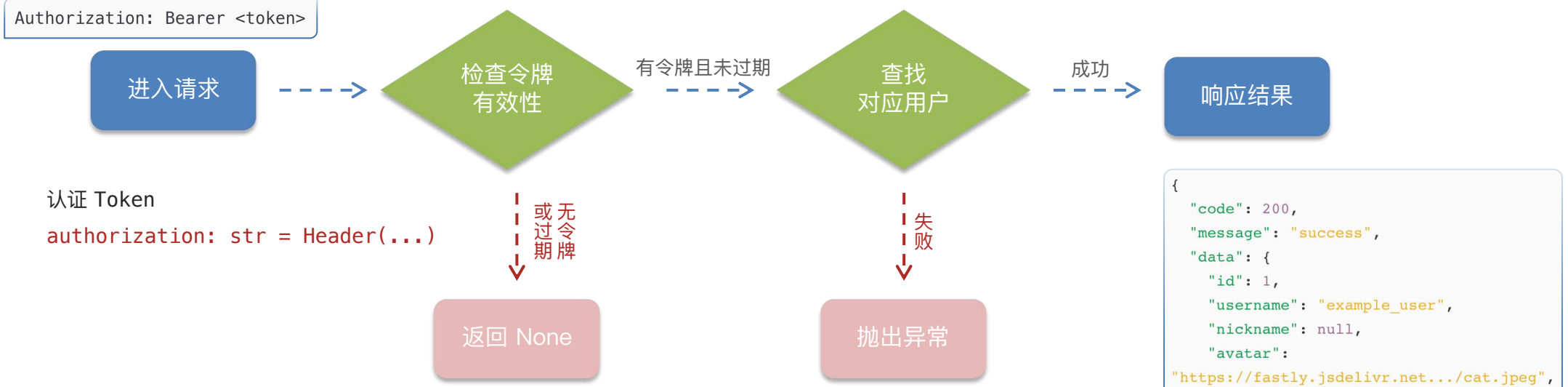
```
{
  "code": 200,
  "message": "success",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar":
"https://fastly.jsdelivr.net/npm/@vant/assets/cat.jpeg",
    "gender": "unknown",
    "bio": "这个人很懒，什么都没留下"
  }
}
```

```python
@router.get("/info")
async def get_user_info():
  return {
    "code": 200,
    "message": "success",
    "data": {
      "id": "user.id",
      "username": "user.username",
      "nickname": "user.nickname",
      "avatar": "user.avatar",
      "gender": "user.gender",
      "bio": "user.bio"
    }
  }
```

获取用户信息

```
select().where(UserToken.token == token)
```

```
Authorization: Bearer <token>
```



认证 Token

```
authorization: str = Header(...)
```

```
{
  "code": 200,
  "message": "success",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg",
    "gender": "unknown",
    "bio": "这个人很懒，什么都没留下"
  }
}
```

HTTP 头部参数

请求头（HTTP Header）主要包含身份信息、请求控制信息、数据格式说明和客户端环境信息。

Header 用来从 HTTP 请求头中获取数据，是 FastAPI 提供的参数依赖工具。

```python
from fastapi import Header

@app.get("/info")
async def get_token(authorization: str=Header(...)):
  pass
```

修改用户信息

4. 更新用户信息

- **接口地址**: `PUT /api/user/update`
- **请求头**: 需要认证
- **请求参数**:

参数名	类型	必填	说明
nickname	string	否	昵称
avatar	string	否	头像URL
gender	string	否	性别
bio	string	否	个人简介
phone	string	否	手机号

- **请求示例**:

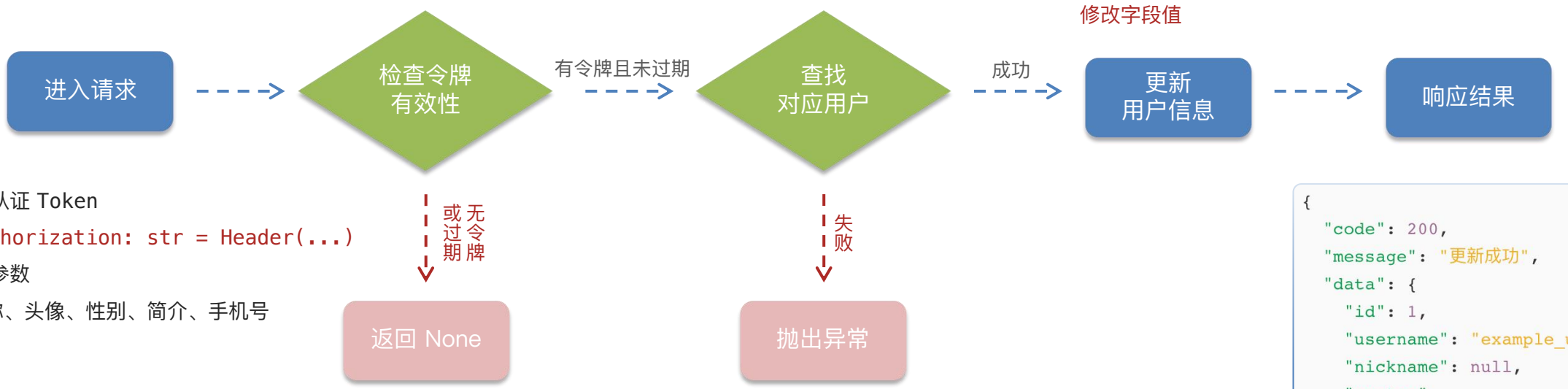
```
{"bio": "这是我的个人简介"}
```

- **响应示例**:

```
{
  "code": 200,
  "message": "更新成功",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg",
    "gender": "unknown",
    "bio": "这是我的个人简介"
  }
}
```

```python
@router.put("/update")
async def update_user_info(
                    user_data: UserUpdateRequest,
):
  return {
    "code": 200,
    "message": "更新成功",
    "data": {
      "id": "updated_user.id",
      "username": "updated_user.username",
      "nickname": "updated_user.nickname",
      "avatar": "updated_user.avatar",
      "gender": "updated_user.gender",
      "bio": "updated_user.bio"
    }
  }
```

修改用户信息



- 认证 Token

```
authorization: str = Header(...)
```

- ✓ 参数

昵称、头像、性别、简介、手机号

```
{
  "code": 200,
  "message": "更新成功",
  "data": {
    "id": 1,
    "username": "example_user",
    "nickname": null,
    "avatar":
"https://fastly.jsdelivr.net.../cat.jpeg",
    "gender": "unknown",
    "bio": "这是我的个人简介"
  }
}
```

修改密码

5. 修改用户密码

- **接口地址**: `PUT /api/user/password`
- **请求头**: 需要认证
- **请求参数**:

参数名	类型	必填	说明
oldPassword	string	是	当前密码
newPassword	string	是	新密码

- **请求示例**:

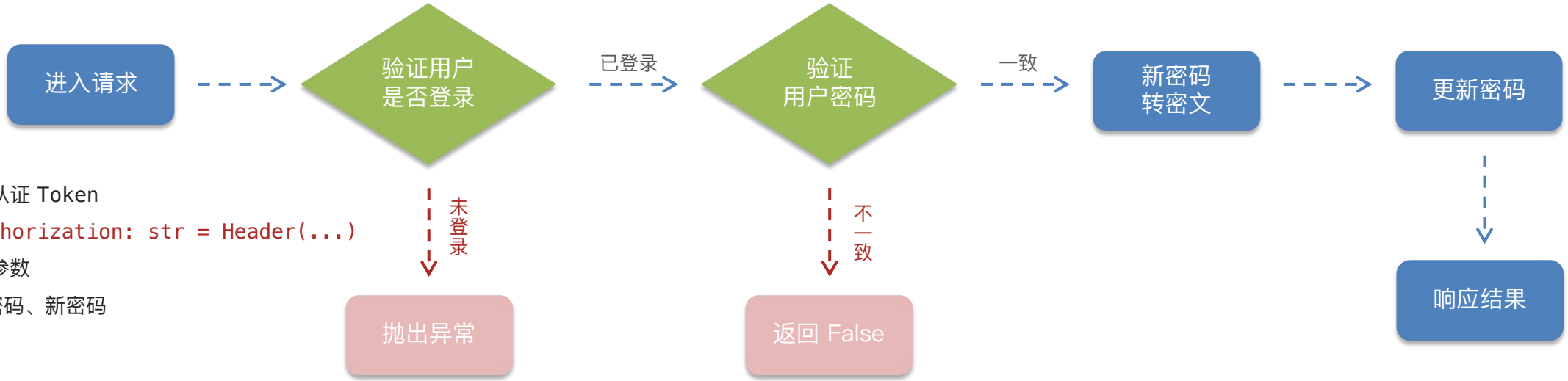
```
{
  "oldPassword": "current_password",
  "newPassword": "new_password"
}
```

- **响应示例**:

```
{
  "code": 200,
  "message": "密码修改成功",
  "data": null
}
```

```python
@router.put("/password")
async def change_password(
      password_data: UserChangePwdRequest,
):
  return {
    "code": 200,
    "message": "密码修改成功",
    "data": null
  }
```

修改密码



- 认证 Token

```
authorization: str = Header(...)
```

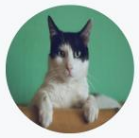
✓ 参数

旧密码、新密码

本章总结



用户注册



新闻资讯

用户名 请输入用户名

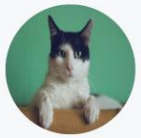
密码 请输入密码

确认密码 请再次输入密码

注册

已有账号？去登录

用户登录



新闻资讯

用户名 请输入用户名

密码 请输入密码

登录

测试账号：admin

测试密码：123456

个人信息

头像 

用户名 admin

账号ID ID: heima-0e9ae

个人简介 这是一个测试账号 >

修改密码 >

本章总结



- ✓ 定义 APIRouter 实例
- ✓ 注册路由 include_router()
- ✓ 参照接口规范文档




```
class Base(DeclarativeBase):
  pass

class Category(Base):
  __tablename__="news_category"
  ......
```

- ✓ 参照数据库表



- ✓ select(模型类)
- ✓ add()
- ✓ update()
- ✓ delete()



- ✓ Depends 注入数据库依赖
- ✓ 调用查询逻辑
- ✓ 响应结果

本章总结

- 封装通用成功响应格式

① 封装函数，返回统一响应格式

```
def success_response(message: str="success", data=None):
  content = {"code": 200, "message": message, "data": data}
  # 把任何FastAPI、Pydantic、ORM 对象转换成可以被 JSON 安全序列化的数据结构
  return JSONResponse(content=jsonable_encoder(content))
```

② 定义模型类

```
class UserAuthResponse(BaseModel):
  token: str
  user_info: ...
  model_config = ConfigDict(
    from_attributes=True, # 自动序列化 user_info 中的 ORM 对象
 )
```

③ 调用函数，响应结果

```
response_data = UserAuthResponse(
  token=token,
  user_info=UserInfoResponse.model_validate(user)
)
return success_response(message="注册成功", data=response_data)
```

本章总结

- 全局异常处理器：用于捕获业务层、数据库层以及系统层抛出的异常，并以统一的响应格式返回给前端。

① 定义异常处理器（函数）

② 全局注册异常处理器：`app.add_exception_handler(异常类, 异常处理函数名)`

本章总结

- passlib 加密

① 安装 passlib：`passlib[bcrypt]==1.7.4`

② 创建加密上下文

```
pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
```

③ 加密或校验

- `pwd_context.hash(password)`
- `pwd_context.verify(plain_password, hashed_password)`

本章总结

- 用户访问令牌 Token

Token：是服务器发给客户端的一段字符串，用来在后续请求中证明“你已经登录过了”

作用：解决 HTTP 是无状态的问题，在每次请求中“自我证明身份”

① 后端生成 Token：str(uuid.uuid4())

② 从请求头获取 Token：`authorization: str=Header(...)`



传智教育旗下高端IT教育品牌

