

# 🚀 功能扩展学习指南：新字段、Docker、用户认证

**前置知识：**已完成前三篇指南（项目结构、开发流程、项目规范化） **最后更新：**2026-02-14

## 目录

- [一、添加新字段（Priority）：端到端修改练习](#)
- [二、Docker 化：一句话部署](#)
- [三、用户认证：JWT 登录系统](#)
- [四、自测清单](#)

## 一、添加新字段（Priority）：端到端修改练习

### 1.1 为什么这个练习很重要

在真实项目中，“给表加个字段”是你做得最多的事情之一。看起来简单，但需要同步修改的地方比你想象的多：

改一个字段 → 需要同时改 3 个地方：

1. Model (数据库层) → 数据库表长什么样
2. Schema (验证层) → API 接受和返回什么数据
3. Tests (测试层) → 测试要验证新字段

 **习惯：**改了一层，必须检查其他层是否也需要同步修改。

### 1.2 我们做了什么

文件	改了什么
models/todo.py	加了 <code>priority</code> 列 (Integer, 默认 2)
schemas/todo.py	三个 Schema 都加了 <code>priority</code> (Create 有默认值, Update 可选, Response 必有)
tests/test_todo.py	现有测试加了 <code>priority</code> 断言 + 3 个新测试

### 1.3 关键代码解读

**Schema 中的验证约束：**

```
priority: int = Field(
    default=2,      # 不传就是中优先级
    ge=1,          # ge = greater or equal (≥1)
    le=3,          # le = less or equal (≤3)
)
```

ge 和 le 是 Pydantic 的验证规则。传入 priority: 5 时，Pydantic 会自动拒绝并返回 422 错误，你的业务代码完全不需要写 if priority > 3 这种判断。

### 为什么 Create、Update、Response 三个 Schema 的 priority 定义不同？

Schema	定义	原因
TodoCreate	priority: int = Field(default=2)	有默认值，不传就是 2
TodoUpdate	priority: int \  None = Field(default=None)	可选，不传就不改
TodoResponse	priority: int	必有，返回给用户一定包含

思考题：如果你要给 Todo 加一个 due\_date（截止日期）字段，三个 Schema 分别该怎么定义？创建时是必填还是可选？

## 二、Docker 化：一句话部署

### 2.1 Docker 解决什么问题

场景：你开发完了，想让朋友试试你的 API。

没有 Docker：

朋友：怎么运行？  
你：先装 Python 3.13，然后 pip install...  
朋友：我是 Mac 啊，安装失败了  
你：那你试试这个命令...  
(半小时后还在折腾环境)

有了 Docker：

朋友：怎么运行？  
你：docker compose up  
朋友：好了，跑起来了  
(30 秒)

Docker 把你的代码 + Python + 所有依赖打包成一个“容器”，在任何装了 Docker 的机器上都能一模一样地运行。

### 2.2 三个文件的职责

文件	作用	类比
Dockerfile	描述如何“打包”应用	做菜的食谱
docker-compose.yml	描述如何“运行”容器	上菜的流程
.dockerignore	打包时忽略哪些文件	不需要带上的东西

### 2.3 Dockerfile 逐行解读

```
# 1. 选基础镜像 (安装好了 Python 的精简 Linux 系统)
FROM python:3.13-slim

# 2. 设工作目录 (容器里的 /app 就是项目根)
WORKDIR /app

# 3. 先装依赖 (利用 Docker 分层缓存)
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 4. 再复制代码
COPY src/ ./src/

# 5. 暴露端口
EXPOSE 8000

# 6. 启动命令
CMD ["python", "-m", "uvicorn", "app.main:app", "--host", "0.0.0.0", "--port",
"8000"]
```

### 第3步为什么先复制 requirements.txt 而不是一次性复制所有文件?

Docker 是**分层缓存**的。每条指令生成一层：

层 1: FROM python:3.13-slim	→ 很少变
层 2: COPY requirements.txt	→ 偶尔变 (加新依赖时)
层 3: RUN pip install...	→ 跟上一层绑定
层 4: COPY src/ ./src/	→ 经常变 (每次改代码)

当你只改了代码 (层 4) 时, 前三层都命中缓存, 构建只需要几秒。如果把所有文件一起复制, 改一行代码就要重新 `pip install` 所有依赖 (几分钟)。

 **记住:** Dockerfile 里变化频率低的指令放上面, 变化频率高的放下面。

## 2.4 使用方法

```
# 构建并启动 (加 -d 表示后台运行)
docker compose up --build -d

# 查看日志
docker compose logs -f

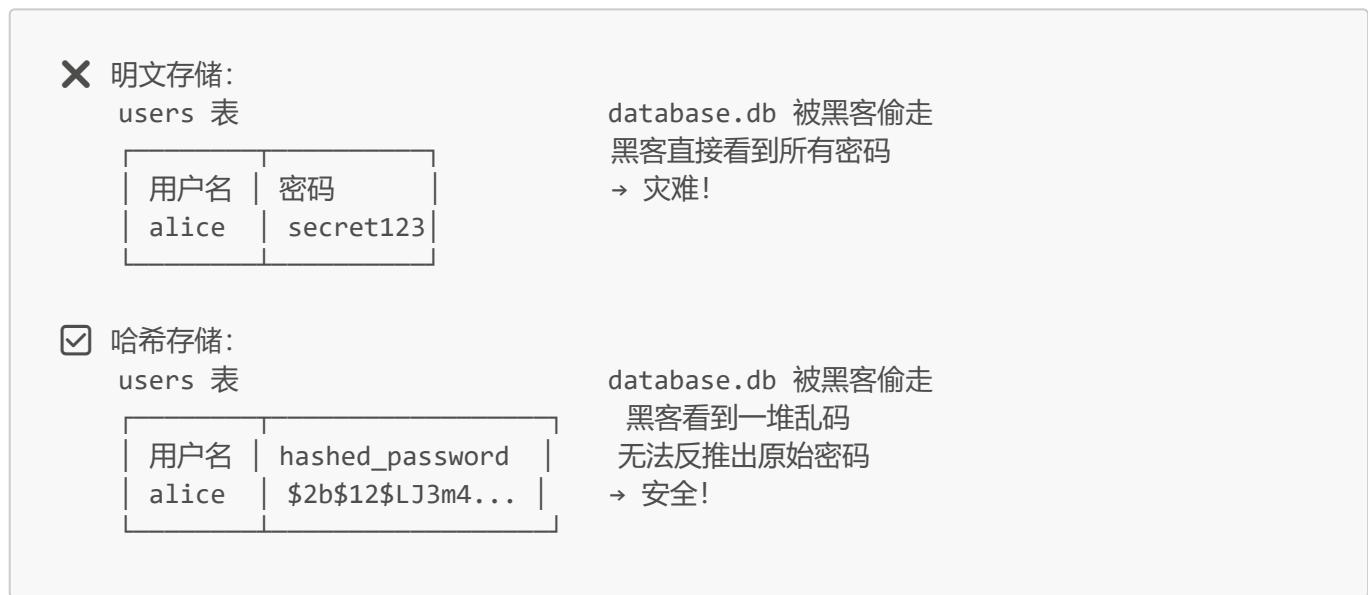
# 停止并清理
docker compose down
```

## 三、用户认证：JWT 登录系统

### 3.1 认证系统全景图



### 3.2 密码安全：为什么不能存明文



我们用 **bcrypt** 算法做哈希——它是单向的，无法从哈希值反推出密码。

```

import bcrypt

# 注册时：密码 → 哈希
hashed = bcrypt.hashpw("secret123".encode(), bcrypt.gensalt())
# 结果: b'$2b$12$LJ3m4ys...' (每次生成的结果都不同!)

# 登录时：验证密码
bcrypt.checkpw("secret123".encode(), hashed) # True
bcrypt.checkpw("wrongpass".encode(), hashed) # False
  
```

### 3.3 JWT Token 是什么

JWT = JSON Web Token, 一个包含用户信息的加密字符串。

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxIiwidXhwIjoxNzA4MH0.abc123...
|          |          |
| 头部 (算法信息) | 载荷 (用户 ID + 过期时间) | 签名
```

- **载荷**里存了 `{"sub": "1", "exp": 1708000000}` → 用户 ID 是 1, 到期时间
- **签名**用 SECRET\_KEY 加密 → 防止被篡改

**为什么用 Token 而不是每次都传用户名密码?**

1. **安全**: 密码只在登录时传一次, 之后用 token 代替
2. **高效**: 服务器不需要每次都查数据库验证密码
3. **标准化**: Token 放在 HTTP 头里, 所有客户端都支持

### 3.4 依赖注入保护端点

以前的端点:

```
async def create_todo(todo_in: TodoCreate, db = Depends(get_db)):
    todo = Todo(**todo_in.model_dump())
    ...
    ...
```

加了认证之后:

```
async def create_todo(
    todo_in: TodoCreate,
    current_user: User = Depends(get_current_user), # ← 新增!
    db = Depends(get_db),
):
    todo = Todo(**todo_in.model_dump(), user_id=current_user.id) # ← 关联用户
    ...
    ...
```

加一个 `Depends(get_current_user)` 就够了。FastAPI 会自动:

1. 从请求头取出 `Authorization: Bearer <token>`
2. 解码 token 获取 `user_id`
3. 从数据库查找 `User` 对象
4. 注入到 `current_user` 参数

如果 token 无效或过期, 自动返回 401 错误。**你的业务代码完全不需要处理认证逻辑。**

### 3.5 新文件结构

```

src/app/
├── security.py           ← 新增: 密码哈希 + JWT + get_current_user
└── models/
    ├── todo.py            ← 修改: 加了 user_id 外键
    └── user.py             ← 新增: 用户表
└── schemas/
    ├── todo.py            ← 新增: UserCreate, UserResponse, Token
    └── user.py
└── routers/
    ├── todo.py            ← 修改: 所有端点都需要登录
    └── auth.py             ← 新增: 注册 + 登录
└── main.py               ← 修改: 注册 auth 路由

tests/
├── conftest.py           ← 修改: 新增 auth_client fixture
└── test_todo.py          ← 修改: 使用 auth_client
    └── test_auth.py        ← 新增: 注册/登录测试

```

### 3.6 测试中的 auth\_client

```

@pytest.fixture
async def auth_client(client):
    # 1. 注册测试用户
    await client.post("/auth/register", json={...})
    # 2. 登录获取 token
    login_resp = await client.post("/auth/login", data={...})
    token = login_resp.json()["access_token"]
    # 3. 设置请求头 (之后所有请求自动带 token)
    client.headers["Authorization"] = f"Bearer {token}"
    yield client

```

有了这个 fixture，测试函数只需要把参数从 `client` 改成 `auth_client`，所有认证逻辑就自动处理了，测试代码保持简洁。

## 四、自测清单

### ④ 新字段

- 给一个表加新字段时，需要同步修改哪三个地方？
- `Field(ge=1, le=3)` 做了什么？如果传入 0 会发生什么？
- 创建 Schema 中 `priority: int = Field(default=2)` 和更新 Schema 中 `priority: int | None = Field(default=None)` 的区别是什么？

### ⑤ Docker

- Dockerfile 里先 COPY requirements.txt 再 COPY 源代码的好处是什么？
- `--host 0.0.0.0` 有什么作用？不加会怎样？

- Docker volume 解决了什么问题？

## ④ 认证

- 为什么密码不能以明文存储？
- bcrypt 哈希是单向的，那登录时怎么“验证”密码？
- JWT token 里包含了哪些信息？
- `Depends(get_current_user)` 是怎么保护 API 端点的？
- 如果不带 token 访问 `/todos/`，会返回什么状态码？

---

❖ 完成这三个功能后，你已经掌握了后端开发的核心技能：

- 数据库设计与 ORM 操作
- RESTful API 设计
- 数据验证与序列化
- 自动化测试
- 容器化部署
- 用户认证与授权

这些技能可以直接迁移到任何后端项目中。恭喜！ 🎉