# FastAPI 入门

## 概述

### 本课程特色

采用FastAPI+requests双端开发。FastAPI开发服务端接口，requests模拟客户端发起请求。

### 参考文档

FastAPI官方文档：<https://fastapi.tiangolo.com/zh/tutorial/>

### 安装

pip install fastapi

pip install uvicorn[standard]

### 入门案例

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index():

    return {"message": "Hello World"}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000/"

response=requests.get(target\_url)

print(response.text)

运行服务端：python main.py

运行客户端：python client.py

## 请求参数

### 请求头参数

服务器：main.py，获取请求头参数中的UserAgent，用户代理。

from fastapi import FastAPI

from typing import Optional

from fastapi import Header

app = FastAPI()

@app.get("/")

async def index(*user\_agent*: Optional[str] = Header(None)):

    return {"message": *user\_agent*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000/"

response=requests.get(target\_url)

print(response.text)

运行服务端：python main.py

运行客户端：python client.py

### 路径参数

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/{name}/{age}")

async def index(*name*: str, *age*: int):

    return {"name": *name*, "age": *age*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000/zhangdapeng/33"

response=requests.get(target\_url)

print(response.text)

### 查询参数

#### 基本用法

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index(*name*: str, *age*: int):

    return {"name": *name*, "age": *age*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000?name=zhangdapeng&age=33"

response=requests.get(target\_url)

print(response.text)

#### 可选参数

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index(*name*: str, *age*: int=33):

    return {"name": *name*, "age": *age*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000?name=zhangdapeng"

response=requests.get(target\_url)

print(response.text)

### 请求体参数

#### Body 作为参数

服务端：main.py

from fastapi import FastAPI, Body

app = FastAPI()

@app.get("/")

async def index(*name*: str=Body(), *age*: int=Body(33)):

    return {"name": *name*, "age": *age*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

data={"name":"zhangdapeng"}

response=requests.get(target\_url,*json*=data)

print(response.text)

#### 使用 pydantic 作为参数

服务端：main.py

from fastapi import FastAPI, Body

from pydantic import BaseModel

class Student(BaseModel):

    name:str

    age:int=33

app = FastAPI()

@app.get("/")

async def index(*student*: Student):

    return *student*.dict()

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

data={"name":"zhangdapeng"}

response=requests.get(target\_url,*json*=data)

print(response.text)

### 参数校验

服务端：main.py

from fastapi import FastAPI, Path, Query

app = FastAPI()

@app.get("/{name}")

async def index(

*name*:str=Path(*min\_length*=3,*max\_length*=32),

*age*:int=Query(0,*gt*=0,*lte*=200),

):

    return {"name":*name*, "age":*age*}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000/zhangdapeng?age=33"

response=requests.get(target\_url)

print(response.text)

## 请求响应

### 响应字符串

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index():

    return "ok"

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

response=requests.get(target\_url)

print(response.text)

### 响应字典

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index():

    return {"ok": True}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

response=requests.get(target\_url)

print(response.text)

### 响应列表

服务端：main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def index():

    return [{"name":"zs"},{"name":"ls"}]

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

response=requests.get(target\_url)

print(response.text)

### 响应pydantic

服务端：main.py

from fastapi import FastAPI

from pydantic import BaseModel

app = FastAPI()

class Student(BaseModel):

    name:str

    age:int

@app.get("/")

async def index():

    return Student(*name*="zs",*age*=33)

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run("main:app")

客户端：client.py

import requests

target\_url="http://localhost:8000"

response=requests.get(target\_url)

print(response.text)

# 商品管理API

## 基本增删改查

### 新增商品

### 接口设计

请求方法：

● POST

接口路径：

● /goods

● http://localhost:8000/goods

入参：

● name：商品名称，字符串

● price：商品价格，浮点数

返回值：

● code：编码

● status：状态

● msg：提示消息

### 代码实现

@app.post("/goods")  
async def post\_goods(goods: Goods):  
 global db\_id  
 db[db\_id] = goods.dict()  
 db\_id += 1  
 return get\_response()

## 根据ID获取指定商品

### 接口设计

请求方法：

● GET

接口路径：

● /goods/{gid}

● http://localhost:8000/goods/1

入参：无

返回值：

● code：编码

● status：状态

● msg：提示消息

● data: 商品信息

○ ID

○ name：商品名称

○ price：商品价格

### 代码实现

@app.get("/goods/{gid}")  
async def get\_goods\_by\_id(gid: int):  
 goods = db.get(gid)  
 return get\_response(data=goods)

## 分页获取所有商品

### 接口设计

接口路径：

● /goods

● http://localhost:8000/goods

入参：

● page：分页

● size：每页多少个元素

返回值：

● code：编码

● status：状态

● msg：提示消息

● data：数据

○ total：总共有多少条数据

○ data：商品列表，数组

■ ID

■ name：商品名称

■ price：商品价格

### 生成模拟数据

import random

db = [

    {"id": i, "name": f"《React学习手册{i}》", "price": random.randint(20, 200)}

    for i in range(1, 101)

]

print(type(db))

print(db)

### 接口代码实现

数据代码：

db = {

    i: {"name": f"《React学习手册{i}》", "price": random.randint(30,200)}

    for i in range(1,100)

}

接口代码：

@app.get("/goods")  
async def get\_goods(page: int = 1, size: int = 20):  
 start\_index = (page - 1) \* size  
 end\_index = start\_index + size  
 page\_data = [  
 db.get(i)  
 for i in range(start\_index, end\_index+1)  
 ]  
 data = {  
 "total": len(db),  
 "page": page,  
 "size": size,  
 "data": page\_data  
 }  
 return get\_response(data)

## 根据ID修改指定商品

### 接口设计

请求方法：

● PUT

接口路径：

● /goods/{gid}

● http://localhost:8000/goods/100

入参：

● name：商品名称

● price：商品价格

返回值：

● code：编码

● status：状态

● msg：提示消息

### 接口代码实现

@app.put("/goods/{gid}")

@app.put("/goods/{gid}")  
async def put\_goods(gid: int, goods: Goods):  
 target = db.get(gid)  
 if target is None:  
 del db[gid]  
 db[gid] = goods.dict()  
 return get\_response()

## 根据ID删除指定商品

### 接口设计

请求方法：

● DELETE

接口路径：

● /goods/{gid}

● http://localhost:8000/goods/100

返回值：只返回成功

● code：编码

● status：状态

● msg：提示消息

### 接口代码实现

@app.delete("/goods/{gid}")  
async def delete\_goods(gid: int):  
 target = db.get(gid)  
 if target is not None:  
 del db[gid]  
 return get\_response()

## 完整代码

### 服务端

服务端代码：main.py

import random  
from fastapi import FastAPI  
from pydantic import BaseModel  
  
app = FastAPI()  
  
# 模拟数据库  
db = {  
 i: {"name": f"《React学习手册{i}》", "price": random.randint(30, 200)}  
 for i in range(1, 100)  
}  
db\_id = 1 # 模拟ID  
  
  
def get\_response(data=None):  
 response = {"code": 10000, "msg": "success", "status": True}  
 if data is not None:  
 response["data"] = data  
 return response  
  
  
class Goods(BaseModel):  
 name: str # 商品名称  
 price: float # 商品价格  
  
  
@app.post("/goods")  
async def post\_goods(goods: Goods):  
 global db\_id  
 db[db\_id] = goods.dict()  
 db\_id += 1  
 return get\_response()  
  
  
@app.get("/goods/{gid}")  
async def get\_goods\_by\_id(gid: int):  
 goods = db.get(gid)  
 return get\_response(data=goods)  
  
  
@app.get("/goods")  
async def get\_goods(page: int = 1, size: int = 20):  
 start\_index = (page - 1) \* size  
 end\_index = start\_index + size  
 page\_data = [  
 db.get(i)  
 for i in range(start\_index, end\_index+1)  
 ]  
 data = {  
 "total": len(db),  
 "page": page,  
 "size": size,  
 "data": page\_data  
 }  
 return get\_response(data)  
  
  
@app.put("/goods/{gid}")  
async def put\_goods(gid: int, goods: Goods):  
 target = db.get(gid)  
 if target is None:  
 del db[gid]  
 db[gid] = goods.dict()  
 return get\_response()  
  
  
@app.delete("/goods/{gid}")  
async def delete\_goods(gid: int):  
 target = db.get(gid)  
 if target is not None:  
 del db[gid]  
 return get\_response()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
  
 uvicorn.run("main:app")

### 客户端

客户端代码：client.py

import requests  
  
# 新增商品  
target\_url = "http://localhost:8000/goods"  
data = {"name": "测试", "price": 33.33}  
response = requests.post(target\_url, json=data)  
print("测试新增商品：", response.text)  
  
# 根据id获取商品  
target\_url = "http://localhost:8000/goods/33"  
response = requests.get(target\_url)  
print("测试根据id获取商品：", response.text)  
  
# 根据id删除商品  
target\_url = "http://localhost:8000/goods/34"  
response = requests.delete(target\_url)  
print("测试根据id删除商品：", response.text)  
  
# 根据id修改商品  
target\_url = "http://localhost:8000/goods/33"  
data = {"name": "测试333", "price": 33.33}  
response = requests.put(target\_url, json=data)  
print("测试根据id修改商品：", response.text)  
  
# 分页获取商品  
target\_url = "http://localhost:8000/goods?page=1&size=20"  
response = requests.get(target\_url)  
print("测试分页获取商品：", response.text)

# 中间件开发

## 跨域中间件

服务端：main.py

from fastapi import FastAPI  
from fastapi.middleware.cors import CORSMiddleware  
  
app = FastAPI()  
  
app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=["\*"],  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
  
@app.get("/")  
async def main():  
 return {"message": "Hello World"}  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
  
 uvicorn.run("main:app")

客户端：client.py

import requests  
  
target\_url = "http://localhost:8000"  
response = requests.get(target\_url)  
print(response.text)

# 整合peewee

## 使用PHPStudy安装MySQL

下载：<https://www.xp.cn/download.html>

安装好以后启动MySQL：

图形用户界面, 应用程序

描述已自动生成

默认账号：root

默认密码：root

## peewee基本用法

### 安装

pip install peewee

pip install pymysql

### 创建模型

import pymysql  
from peewee import \*  
from playhouse.db\_url import connect  
  
pymysql.install\_as\_MySQLdb()  
  
db = connect('mysql://root:root@localhost:3306/test')  
  
  
class Goods(Model):  
 name = CharField()  
 price = FloatField()  
  
 class Meta:  
 database = db  
  
  
db.connect()

### 创建表

try:  
 db.create\_tables([Goods])  
except Exception as e:  
 pass

### 新增商品

goods = Goods(name='《React学习手册》', price=128)  
goods.save()

### 根据ID查询商品

goods = Goods.get(1)  
print(goods.name, goods.price)

### 修改商品

goods = Goods.get(1)  
goods.name = "《React学习手册111》"  
goods.save()

### 分页查询商品

data = Goods.select().paginate(1, 2)  
print(list(data))

## 基于MySQL的商品管理接口

### 数据准备

import pymysql  
from fastapi import FastAPI  
from peewee import \*  
from playhouse.db\_url import connect  
from pydantic import BaseModel  
  
pymysql.install\_as\_MySQLdb()  
  
db = connect('mysql://root:root@localhost:3306/test')  
app = FastAPI()  
  
  
class GoodsModel(Model):  
 name = CharField()  
 price = FloatField()  
  
 class Meta:  
 database = db  
 db\_table = "goods"  
  
  
class GoodsSchema(BaseModel):  
 name: str # 商品名称  
 price: float # 商品价格  
  
  
def get\_response(data=None):  
 response = {"code": 10000, "msg": "success", "status": True}  
 if data is not None:  
 response["data"] = data  
 return response  
  
  
def get\_response\_error(code, msg):  
 return {"code": code, "msg": msg, "status": False}

### 新增商品接口

@app.post("/goods")  
async def post\_goods(goods: GoodsSchema):  
 try:  
 goods\_model = GoodsModel(\*\*goods.dict())  
 goods\_model.save()  
 return get\_response()  
 except Exception as e:  
 print(e)  
 return get\_response\_error(10003, "保存商品数据失败")

### 根据ID获取商品接口

@app.get("/goods/{gid}")  
async def get\_goods\_by\_id(gid: int):  
 try:  
 goods = GoodsModel.get(gid)  
 return get\_response(goods.\_\_data\_\_)  
 except Exception as e:  
 print(e)  
 return get\_response\_error(10004, "商品不存在")

### 分页获取商品接口

@app.get("/goods")  
async def get\_goods(page: int = 1, size: int = 20):  
 query = GoodsModel.select()  
 count = query.count()  
 result = query.paginate(page, size).dicts()  
 data = {  
 "total": count,  
 "data": list(result)  
 }  
 return get\_response(data)

### 修改商品接口

@app.put("/goods/{gid}")  
async def put\_goods(gid: int, goods: GoodsSchema):  
 try:  
 goods\_model = GoodsModel.get(gid)  
 goods\_model.name = goods.name  
 goods\_model.price = goods.price  
 goods\_model.save()  
 return get\_response()  
 except Exception as e:  
 return get\_response\_error(10004, "商品不存在")

### 删除商品接口

@app.delete("/goods/{gid}")  
async def delete\_goods(gid: int):  
 try:  
 GoodsModel.get(gid).delete\_instance()  
 except Exception as e:  
 pass  
 return get\_response()

### 完整代码

import pymysql  
from fastapi import FastAPI  
from peewee import \*  
from playhouse.db\_url import connect  
from pydantic import BaseModel  
  
pymysql.install\_as\_MySQLdb()  
  
db = connect('mysql://root:root@localhost:3306/test')  
app = FastAPI()  
  
  
class GoodsModel(Model):  
 name = CharField()  
 price = FloatField()  
  
 class Meta:  
 database = db  
 db\_table = "goods"  
  
  
class GoodsSchema(BaseModel):  
 name: str # 商品名称  
 price: float # 商品价格  
  
  
def get\_response(data=None):  
 response = {"code": 10000, "msg": "success", "status": True}  
 if data is not None:  
 response["data"] = data  
 return response  
  
  
def get\_response\_error(code, msg):  
 return {"code": code, "msg": msg, "status": False}  
  
  
# http://localhost:8000/docs  
# http://localhost:8000/goods  
@app.post("/goods")  
async def post\_goods(goods: GoodsSchema):  
 try:  
 goods\_model = GoodsModel(\*\*goods.dict())  
 goods\_model.save()  
 return get\_response()  
 except Exception as e:  
 print(e)  
 return get\_response\_error(10003, "保存商品数据失败")  
  
  
@app.get("/goods/{gid}")  
async def get\_goods\_by\_id(gid: int):  
 try:  
 goods = GoodsModel.get(gid)  
 return get\_response(goods.\_\_data\_\_)  
 except Exception as e:  
 print(e)  
 return get\_response\_error(10004, "商品不存在")  
  
  
# http://localhost:8000/goods  
@app.get("/goods")  
async def get\_goods(page: int = 1, size: int = 20):  
 query = GoodsModel.select()  
 count = query.count()  
 result = query.paginate(page, size).dicts()  
 data = {  
 "total": count,  
 "data": list(result)  
 }  
 return get\_response(data)  
  
  
@app.put("/goods/{gid}")  
async def put\_goods(gid: int, goods: GoodsSchema):  
 try:  
 goods\_model = GoodsModel.get(gid)  
 goods\_model.name = goods.name  
 goods\_model.price = goods.price  
 goods\_model.save()  
 return get\_response()  
 except Exception as e:  
 return get\_response\_error(10004, "商品不存在")  
  
  
@app.delete("/goods/{gid}")  
async def delete\_goods(gid: int):  
 try:  
 GoodsModel.get(gid).delete\_instance()  
 except Exception as e:  
 pass  
 return get\_response()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import uvicorn  
  
 uvicorn.run("main:app")

### 客戶端代码

import requests  
  
# 新增商品  
target\_url = "http://localhost:8000/goods"  
data = {"name": "测试", "price": 33.33}  
response = requests.post(target\_url, json=data)  
print("测试新增商品：", response.text)  
  
# 根据id获取商品  
target\_url = "http://localhost:8000/goods/1"  
response = requests.get(target\_url)  
print("测试根据id获取商品：", response.text)  
  
# 根据id修改商品  
target\_url = "http://localhost:8000/goods/1"  
data = {"name": "测试333", "price": 33.33}  
response = requests.put(target\_url, json=data)  
print("测试根据id修改商品：", response.text)  
  
# 分页获取商品  
target\_url = "http://localhost:8000/goods?page=1&size=20"  
response = requests.get(target\_url)  
print("测试分页获取商品：", response.text)  
  
# 根据id删除商品  
target\_url = "http://localhost:8000/goods/1"  
response = requests.delete(target\_url)  
print("测试根据id删除商品：", response.text)

# JWT权限校验

## 接口设计

### 参考资料

FastAPI官方文档：<https://fastapi.tiangolo.com/zh/tutorial/security/oauth2-jwt/>

### 安装依赖

pip install fastapi[all]

pip install uvicorn[standard]

pip install python-jose[cryptography]

pip install passlib[bcrypt]

pip install python-multipart

### 功能设计

1. 让自带的接口文档需要用户登录以后才能够访问，确保接口的安全

2. 实现一个接口，能够生产JWT Token并返回

3. 实现一个接口，能够进行用户注册

4. 对注册用户的密码进行加密处理，确保数据的安全性

### 接口概要设计

1. 文档登录接口

2. 用户注册接口

3. 用户登录接口

4. 密码修改接口

5. 获取当前登录用户信息接口

### 文档登录接口

1. 请求方式：POST

2. 请求参数：

a. username：用户名

b. password：密码

3. 接口地址：/token

4. 响应数据：

a. access\_token：JWT Token

b. token\_type：Token类型

### 用户注册接口

1. 请求方式：POST

2. 请求参数：

a. username：用户名

b. password：密码

c. re\_password：确认密码

3. 接口地址：/register

4. 响应数据：

a. status：状态

b. code：响应编码

c. msg：响应消息

### 用户登录接口

1. 请求方式：POST

2. 请求参数：

a. username：用户名

b. password：密码

3. 接口地址：/login

4. 响应数据：

a. status：状态

b. code：响应编码

c. msg：响应消息

d. data：响应数据

ⅰ. id：用户ID

ⅱ. username：用户名

ⅲ. token：JWT Token

### 密码修改接口

1. 请求方式：PUT

2. 请求参数：

a. username：用户名

b. old\_password：旧密码

c. password：密码

d. re\_password：确认密码

3. 接口地址：/password

4. 响应数据：

a. status：状态

b. code：响应编码

c. msg：响应消息

### 获取当前登录用户信息接口

1. 请求方式：GET

2. 请求参数：在请求头中携带“Authorization：Bear token”

3. 接口地址：/luserinfo

4. 响应数据：

a. status：状态

b. code：响应编码

c. msg：响应消息

d. data：响应数据

ⅰ. id：用户ID

ⅱ. username：用户名

ⅲ. menu：菜单列表

ⅳ. auth：权限列表

## 快速入门

### 密码生成与校验

from passlib.context import CryptContext  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 生成密码  
 password\_str = "zhangdapeng520"  
 password = pwd\_context.encrypt(password\_str)  
 print(password)  
  
 # 校验密码  
 print(pwd\_context.verify(password\_str, password))

### 完整代码

from datetime import datetime, timedelta  
from typing import Union  
  
from fastapi import Depends, FastAPI, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm  
from jose import JWTError, jwt  
from passlib.context import CryptContext  
from pydantic import BaseModel  
  
# to get a string like this run:  
# openssl rand -hex 32  
SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"  
ALGORITHM = "HS256"  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30  
  
# 模拟数据库  
fake\_users\_db = {  
 "zhangdapeng": {  
 "username": "zhangdapeng",  
 "full\_name": "张大鹏",  
 "email": "zhangdapeng@zhangdapeng.com",  
 "hashed\_password": "$2b$12$4Te1//FDmF0YPYuskhsWdeT9vwgZB3v7O.OL3X64guEyaGL8/8sFK",  
 "disabled": False,  
 }  
}  
  
  
class Token(BaseModel):  
 access\_token: str  
 token\_type: str  
  
  
class TokenData(BaseModel):  
 username: Union[str, None] = None  
  
  
class User(BaseModel):  
 username: str  
 email: Union[str, None] = None  
 full\_name: Union[str, None] = None  
 disabled: Union[bool, None] = None  
  
  
class UserInDB(User):  
 hashed\_password: str  
  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")  
  
app = FastAPI()  
  
  
def verify\_password(plain\_password, hashed\_password):  
 return pwd\_context.verify(plain\_password, hashed\_password)  
  
  
def get\_password\_hash(password):  
 return pwd\_context.hash(password)  
  
  
def get\_user(db, username: str):  
 if username in db:  
 user\_dict = db[username]  
 return UserInDB(\*\*user\_dict)  
  
  
def authenticate\_user(fake\_db, username: str, password: str):  
 user = get\_user(fake\_db, username)  
 if not user:  
 return False  
 if not verify\_password(password, user.hashed\_password):  
 return False  
 return user  
  
  
def create\_access\_token(data: dict, expires\_delta: Union[timedelta, None] = None):  
 to\_encode = data.copy()  
 if expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 else:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 return encoded\_jwt  
  
  
async def get\_current\_user(token: str = Depends(oauth2\_scheme)):  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 try:  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 username: str = payload.get("sub")  
 if username is None:  
 raise credentials\_exception  
 token\_data = TokenData(username=username)  
 except JWTError:  
 raise credentials\_exception  
 user = get\_user(fake\_users\_db, username=token\_data.username)  
 if user is None:  
 raise credentials\_exception  
 return user  
  
  
async def get\_current\_active\_user(current\_user: User = Depends(get\_current\_user)):  
 if current\_user.disabled:  
 raise HTTPException(status\_code=400, detail="Inactive user")  
 return current\_user  
  
  
@app.post("/token", response\_model=Token)  
async def login\_for\_access\_token(form\_data: OAuth2PasswordRequestForm = Depends()):  
 """登录获取token"""  
 user = authenticate\_user(fake\_users\_db, form\_data.username, form\_data.password)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.username}, expires\_delta=access\_token\_expires  
 )  
 return {"access\_token": access\_token, "token\_type": "bearer"}  
  
  
@app.get("/users/me/", response\_model=User)  
async def read\_users\_me(current\_user: User = Depends(get\_current\_active\_user)):  
 return current\_user  
  
  
@app.get("/users/me/items/")  
async def read\_own\_items(current\_user: User = Depends(get\_current\_active\_user)):  
 return [{"item\_id": "Foo", "owner": current\_user.username}]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import uvicorn  
  
 uvicorn.run("main:app")

### 客户端

import requests  
  
# 登录获取token  
target\_url = "http://localhost:8000/token"  
data = {"username": "zhangdapeng", "password": "zhangdapeng520"}  
response = requests.post(target\_url, data=data)  
print("测试登录获取token：", response.text)  
  
# 获取当前用户  
token = response.json()["access\_token"]  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户：", response.text)  
  
# 获取当前数据  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/items/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户数据：", response.text)

## 代码分析

### 准备基础数据

from datetime import datetime, timedelta  
from typing import Union  
  
from fastapi import Depends, FastAPI, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm  
from jose import JWTError, jwt  
from passlib.context import CryptContext  
from pydantic import BaseModel  
  
# 生成私钥的命令：openssl rand -hex 32  
SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"  
ALGORITHM = "HS256" # 加密算法  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30 # token过期时间（分钟）  
  
# 模拟用户数据  
fake\_users\_db = {  
 "zhangdapeng": {  
 "username": "zhangdapeng",  
 "full\_name": "张大鹏",  
 "email": "zhangdapeng@zhangdapeng.com",  
 "hashed\_password": "$2b$12$4Te1//FDmF0YPYuskhsWdeT9vwgZB3v7O.OL3X64guEyaGL8/8sFK",  
 "disabled": False,  
 }  
}  
  
# 密码加密对象  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
# 文档OAuth2登录  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")  
# FastAPI实例对象  
app = FastAPI()

### 准备Schema

class Token(BaseModel):  
 """token模型"""  
 access\_token: str # jwt token  
 token\_type: str # token 类型  
  
  
class TokenData(BaseModel):  
 """token内部数据"""  
 username: Union[str, None] = None # 用户名  
  
  
class User(BaseModel):  
 """用户模型"""  
 username: str # 用户名  
 email: Union[str, None] = None # 邮箱  
 full\_name: Union[str, None] = None # 全名  
 disabled: Union[bool, None] = None # 是否禁用  
  
  
class UserInDB(User):  
 """入库的用户模型"""  
 hashed\_password: str # 加密密码

### 密码加密

核心代码如下：

def get\_password\_hash(password):  
 """密码加密"""  
 return pwd\_context.hash(password)

这里牵扯到我们导入的一个叫passlib的库，该库的核心用法如下：

from passlib.context import CryptContext  
  
# 密码加密对象  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
# 原始密码  
password\_str = "zhangdapeng"  
  
# 密码加密  
secret = pwd\_context.hash(password\_str)  
print("加密后的密码：", secret)  
  
# 校验密码是否正确  
is\_ok = pwd\_context.verify(password\_str, secret)  
print("密码是否正确：", is\_ok)

输出结果如下：

加密后的密码： $2b$12$GycmTRnJpDOjBoiLcNZnYe3tkyDUNVYr/WRWzK/6L60VdaDS1PQpq

密码是否正确： True

### 校验密码是否正确

def verify\_password(plain\_password, hashed\_password):  
 """校验密码是否正确"""  
 return pwd\_context.verify(plain\_password, hashed\_password)

### 获取用户信息

def get\_user(db, username: str):  
 """模拟从数据库获取用户信息"""  
 if username in db:  
 user\_dict = db[username]  
 return UserInDB(\*\*user\_dict)

### 校验用户信息

def authenticate\_user(fake\_db, username: str, password: str):  
 """校验用户"""  
 # 从数据库获取用户  
 user = get\_user(fake\_db, username)  
 # 校验用户是否存在  
 if not user:  
 return False  
 # 校验用户密码是否正确  
 if not verify\_password(password, user.hashed\_password):  
 return False  
 # 校验通过，返回数据库查到的用户信息  
 return user

### 生成JWT Token

def create\_access\_token(data: dict, expires\_delta: Union[timedelta, None] = None):  
 """创建JWT Token"""  
 # 提交要打包的token内部数据  
 to\_encode = data.copy()  
 # 设置token的过期时间  
 if expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 else:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 # 生成JWT Token  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 # 返回JWT Token  
 return encoded\_jwt

### 获取当前登录用户

async def get\_current\_user(token: str = Depends(oauth2\_scheme)):  
 """获取当前登录用户"""  
 # 异常对象  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
  
 try:  
 # 解析客户端传过来的JWT Token  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 # 获取用户名  
 username: str = payload.get("sub")  
 # 校验用户名是否存在  
 if username is None:  
 raise credentials\_exception  
 # 封装token数据  
 token\_data = TokenData(username=username)  
 except JWTError:  
 raise credentials\_exception  
 # 从数据库获取用户  
 user = get\_user(fake\_users\_db, username=token\_data.username)  
 # 校验用户是否存在  
 if user is None:  
 raise credentials\_exception  
 # 返回获取到的用户  
 return user

### 校验用户账户是否激活

async def get\_current\_active\_user(current\_user: User = Depends(get\_current\_user)):  
 """获取激活的用户"""  
 # 校验用户是否被禁用  
 if current\_user.disabled:  
 raise HTTPException(status\_code=400, detail="Inactive user")  
 # 返回用户信息  
 return current\_user

### 登录接口

@app.post("/token", response\_model=Token)  
async def login\_for\_access\_token(form\_data: OAuth2PasswordRequestForm = Depends()):  
 """用户登录获取Token接口"""  
 # 校验用户  
 user = authenticate\_user(fake\_users\_db, form\_data.username, form\_data.password)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 # 生成JWT Token  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.username}, expires\_delta=access\_token\_expires  
 )  
 # 返回token和token类型  
 return {"access\_token": access\_token, "token\_type": "bearer"}

### 获取用户信息接口

@app.get("/users/me/", response\_model=User)  
async def read\_users\_me(current\_user: User = Depends(get\_current\_active\_user)):  
 """获取当前登录用户"""  
 return current\_user

### 一个激活用户才能访问的接口

@app.get("/users/me/items/")  
async def read\_own\_items(current\_user: User = Depends(get\_current\_active\_user)):  
 """一个需要登录后才能访问接口"""  
 return [{"item\_id": "Foo", "owner": current\_user.username}]

### 完整代码

from datetime import datetime, timedelta  
from typing import Union  
  
from fastapi import Depends, FastAPI, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm  
from jose import JWTError, jwt  
from passlib.context import CryptContext  
from pydantic import BaseModel  
  
# 生成私钥的命令：openssl rand -hex 32  
SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"  
ALGORITHM = "HS256" # 加密算法  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30 # token过期时间（分钟）  
  
# 模拟用户数据  
fake\_users\_db = {  
 "zhangdapeng": {  
 "username": "zhangdapeng",  
 "full\_name": "张大鹏",  
 "email": "zhangdapeng@zhangdapeng.com",  
 "hashed\_password": "$2b$12$4Te1//FDmF0YPYuskhsWdeT9vwgZB3v7O.OL3X64guEyaGL8/8sFK",  
 "disabled": False,  
 }  
}  
  
# 密码加密对象  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
# 文档OAuth2登录  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")  
# FastAPI实例对象  
app = FastAPI()  
  
  
class Token(BaseModel):  
 """token模型"""  
 access\_token: str # jwt token  
 token\_type: str # token 类型  
  
  
class TokenData(BaseModel):  
 """token内部数据"""  
 username: Union[str, None] = None # 用户名  
  
  
class User(BaseModel):  
 """用户模型"""  
 username: str # 用户名  
 email: Union[str, None] = None # 邮箱  
 full\_name: Union[str, None] = None # 全名  
 disabled: Union[bool, None] = None # 是否禁用  
  
  
class UserInDB(User):  
 """入库的用户模型"""  
 hashed\_password: str # 加密密码  
  
  
def get\_password\_hash(password):  
 """密码加密"""  
 return pwd\_context.hash(password)  
  
  
def verify\_password(plain\_password, hashed\_password):  
 """校验密码是否正确"""  
 return pwd\_context.verify(plain\_password, hashed\_password)  
  
  
def get\_user(db, username: str):  
 """模拟从数据库获取用户信息"""  
 if username in db:  
 user\_dict = db[username]  
 return UserInDB(\*\*user\_dict)  
  
  
def authenticate\_user(fake\_db, username: str, password: str):  
 """校验用户"""  
 # 从数据库获取用户  
 user = get\_user(fake\_db, username)  
 # 校验用户是否存在  
 if not user:  
 return False  
 # 校验用户密码是否正确  
 if not verify\_password(password, user.hashed\_password):  
 return False  
 # 校验通过，返回数据库查到的用户信息  
 return user  
  
  
def create\_access\_token(data: dict, expires\_delta: Union[timedelta, None] = None):  
 """创建JWT Token"""  
 # 提交要打包的token内部数据  
 to\_encode = data.copy()  
 # 设置token的过期时间  
 if expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 else:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 # 生成JWT Token  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 # 返回JWT Token  
 return encoded\_jwt  
  
  
async def get\_current\_user(token: str = Depends(oauth2\_scheme)):  
 """获取当前登录用户"""  
 # 异常对象  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
  
 try:  
 # 解析客户端传过来的JWT Token  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 # 获取用户名  
 username: str = payload.get("sub")  
 # 校验用户名是否存在  
 if username is None:  
 raise credentials\_exception  
 # 封装token数据  
 token\_data = TokenData(username=username)  
 except JWTError:  
 raise credentials\_exception  
 # 从数据库获取用户  
 user = get\_user(fake\_users\_db, username=token\_data.username)  
 # 校验用户是否存在  
 if user is None:  
 raise credentials\_exception  
 # 返回获取到的用户  
 return user  
  
  
async def get\_current\_active\_user(current\_user: User = Depends(get\_current\_user)):  
 """获取激活的用户"""  
 # 校验用户是否被禁用  
 if current\_user.disabled:  
 raise HTTPException(status\_code=400, detail="Inactive user")  
 # 返回用户信息  
 return current\_user  
  
  
@app.post("/token", response\_model=Token)  
async def login\_for\_access\_token(form\_data: OAuth2PasswordRequestForm = Depends()):  
 """用户登录获取Token接口"""  
 # 校验用户  
 user = authenticate\_user(fake\_users\_db, form\_data.username, form\_data.password)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 # 生成JWT Token  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.username}, expires\_delta=access\_token\_expires  
 )  
 # 返回token和token类型  
 return {"access\_token": access\_token, "token\_type": "bearer"}  
  
  
@app.get("/users/me/", response\_model=User)  
async def read\_users\_me(current\_user: User = Depends(get\_current\_active\_user)):  
 """获取当前登录用户"""  
 return current\_user  
  
  
@app.get("/users/me/items/")  
async def read\_own\_items(current\_user: User = Depends(get\_current\_active\_user)):  
 """一个需要登录后才能访问接口"""  
 return [{"item\_id": "Foo", "owner": current\_user.username}]  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import uvicorn  
  
 uvicorn.run("main:app")

### 客户端

import requests  
  
# 登录获取token  
target\_url = "http://localhost:8000/token"  
data = {"username": "zhangdapeng", "password": "zhangdapeng520"}  
response = requests.post(target\_url, data=data)  
print("测试登录获取token：", response.text)  
  
# 获取当前用户  
token = response.json()["access\_token"]  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户：", response.text)  
  
# 获取当前数据  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/items/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户数据：", response.text)

## 连接数据库

### 创建用户表

import pymysql  
from peewee import \*  
from playhouse.db\_url import connect  
  
pymysql.install\_as\_MySQLdb()  
  
# 连接指定数据库  
db = connect("mysql://root:root@localhost:3306/test")  
  
  
# 创建用户表  
class User(Model):  
 username = CharField()  
 password = CharField()  
 role\_id = IntegerField()  
  
 class Meta:  
 database = db  
  
  
db.connect()  
  
  
def init\_db():  
 tables = [User]  
 db.create\_tables(tables)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 init\_db()

### 增加用户

user = User(username='zhangdapeng', password='zhangdapeng', role\_id=1)  
user.save()  
print(user.id)  
print(user.username)  
print(user.password)

### 根据ID查询用户

user = User.get(1)  
print(user.id)  
print(user.username)  
print(user.password)

### 修改用户密码

user = User.get(1)  
user.password = "zhangdapeng520"  
user.save()  
print(user.id)  
print(user.username)  
print(user.password)

### 根据用户名查询用户

user = User.get(username="zhangdapeng")  
print(user.id)  
print(user.username)  
print(user.password)

### 数据库模块

#### db/\_\_init\_\_.py

from db.user import User  
from db.base import db  
  
\_\_all\_\_ = [  
 "User",  
 "db",  
]

#### db/base.py

import pymysql  
from peewee import \*  
from playhouse.db\_url import connect  
  
pymysql.install\_as\_MySQLdb()  
  
# 连接指定数据库  
db = connect("mysql://root:root@localhost:3306/test")  
  
  
# 基础模型  
class BaseModel(Model):  
 class Meta:  
 database = db  
  
  
db.connect()

#### db/user.py

from peewee import \*  
from db.base import BaseModel  
  
  
# 创建用户表  
class User(BaseModel):  
 username = CharField()  
 password = CharField()  
 role\_id = IntegerField()

## 组件化开发

### 完整代码

#### 目录结构

文本

描述已自动生成

#### db/\_\_init\_\_.py

from .data import fake\_users\_db, get\_user, authenticate\_user  
  
\_\_all\_\_ = [  
 "fake\_users\_db",  
 "get\_user",  
 "authenticate\_user",  
]

#### db/data.py

from schema import UserInDB  
from utils import verify\_password  
  
# 模拟用户数据  
fake\_users\_db = {  
 "zhangdapeng": {  
 "username": "zhangdapeng",  
 "full\_name": "张大鹏",  
 "email": "zhangdapeng@zhangdapeng.com",  
 "hashed\_password": "$2b$12$4Te1//FDmF0YPYuskhsWdeT9vwgZB3v7O.OL3X64guEyaGL8/8sFK",  
 "disabled": False,  
 }  
}  
  
  
def get\_user(db, username: str):  
 """模拟从数据库获取用户信息"""  
 if username in db:  
 user\_dict = db[username]  
 return UserInDB(\*\*user\_dict)  
  
  
def authenticate\_user(fake\_db, username: str, password: str):  
 """校验用户"""  
 # 从数据库获取用户  
 user = get\_user(fake\_db, username)  
 # 校验用户是否存在  
 if not user:  
 return False  
 # 校验用户密码是否正确  
 if not verify\_password(password, user.hashed\_password):  
 return False  
 # 校验通过，返回数据库查到的用户信息  
 return user

#### dependencies/\_\_init\_\_.py

from .user import get\_current\_user, get\_current\_active\_user  
  
\_\_all\_\_ = [  
 "get\_current\_user",  
 "get\_current\_active\_user",  
]

#### dependencies/user.py

from fastapi import Depends, HTTPException, status  
from fastapi.security import OAuth2PasswordBearer  
from jose import JWTError, jwt  
  
from db import fake\_users\_db, get\_user  
from schema import TokenData, User  
from utils.jwt import SECRET\_KEY, ALGORITHM  
  
# 文档OAuth2登录  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="token")  
  
  
async def get\_current\_user(token: str = Depends(oauth2\_scheme)):  
 """获取当前登录用户"""  
 # 异常对象  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Could not validate credentials",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
  
 try:  
 # 解析客户端传过来的JWT Token  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 # 获取用户名  
 username: str = payload.get("sub")  
 # 校验用户名是否存在  
 if username is None:  
 raise credentials\_exception  
 # 封装token数据  
 token\_data = TokenData(username=username)  
 except JWTError:  
 raise credentials\_exception  
 # 从数据库获取用户  
 user = get\_user(fake\_users\_db, username=token\_data.username)  
 # 校验用户是否存在  
 if user is None:  
 raise credentials\_exception  
 # 返回获取到的用户  
 return user  
  
  
async def get\_current\_active\_user(current\_user: User = Depends(get\_current\_user)):  
 """获取激活的用户"""  
 # 校验用户是否被禁用  
 if current\_user.disabled:  
 raise HTTPException(status\_code=400, detail="Inactive user")  
 # 返回用户信息  
 return current\_user

#### router/\_\_init\_\_.py

from .user import router as user\_router  
  
\_\_all\_\_ = [  
 "user\_router",  
]

#### router/user.py

from datetime import timedelta  
  
from fastapi import APIRouter  
from fastapi import Depends, HTTPException, status  
from fastapi.security import OAuth2PasswordRequestForm  
# 导入JWT相关的代码  
from utils.jwt import ACCESS\_TOKEN\_EXPIRE\_MINUTES, create\_access\_token  
# 导入用户相关数据库方法  
from db import fake\_users\_db, authenticate\_user  
# 导入依赖注入相关代码  
from dependencies.user import get\_current\_active\_user  
# 导入用户相关schema  
from schema import Token, User  
  
router = APIRouter()  
  
  
@router.post("/token", response\_model=Token)  
async def login\_for\_access\_token(form\_data: OAuth2PasswordRequestForm = Depends()):  
 """用户登录获取Token接口"""  
 # 校验用户  
 user = authenticate\_user(fake\_users\_db, form\_data.username, form\_data.password)  
 if not user:  
 raise HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="Incorrect username or password",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
 # 生成JWT Token  
 access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 access\_token = create\_access\_token(  
 data={"sub": user.username}, expires\_delta=access\_token\_expires  
 )  
 # 返回token和token类型  
 return {"access\_token": access\_token, "token\_type": "bearer"}  
  
  
@router.get("/users/me/", response\_model=User)  
async def read\_users\_me(current\_user: User = Depends(get\_current\_active\_user)):  
 """获取当前登录用户"""  
 return current\_user  
  
  
@router.get("/users/me/items/")  
async def read\_own\_items(current\_user: User = Depends(get\_current\_active\_user)):  
 """一个需要登录后才能访问接口"""  
 return [{"item\_id": "Foo", "owner": current\_user.username}]

#### schema/\_\_init\_\_.py

from .user import Token, TokenData, User, UserInDB  
  
\_\_all\_\_ = [  
 "Token",  
 "TokenData",  
 "User",  
 "UserInDB",  
]

#### schema/user.py

from typing import Union  
  
from pydantic import BaseModel  
  
  
class Token(BaseModel):  
 """token模型"""  
 access\_token: str # jwt token  
 token\_type: str # token 类型  
  
  
class TokenData(BaseModel):  
 """token内部数据"""  
 username: Union[str, None] = None # 用户名  
  
  
class User(BaseModel):  
 """用户模型"""  
 username: str # 用户名  
 email: Union[str, None] = None # 邮箱  
 full\_name: Union[str, None] = None # 全名  
 disabled: Union[bool, None] = None # 是否禁用  
  
  
class UserInDB(User):  
 """入库的用户模型"""  
 hashed\_password: str # 加密密码

#### utils/\_\_init.py

from .password import get\_password\_hash, verify\_password  
from .jwt import create\_access\_token  
  
\_\_all\_\_ = [  
 "get\_password\_hash",  
 "verify\_password",  
 "create\_access\_token",  
]

#### utils/jwt.py

from datetime import timedelta, datetime  
from typing import Union  
  
from jose import jwt  
  
# 生成私钥的命令：openssl rand -hex 32  
SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"  
ALGORITHM = "HS256" # 加密算法  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30 # token过期时间（分钟）  
  
  
def create\_access\_token(data: dict, expires\_delta: Union[timedelta, None] = None):  
 """创建JWT Token"""  
 # 提交要打包的token内部数据  
 to\_encode = data.copy()  
 # 设置token的过期时间  
 if expires\_delta:  
 expire = datetime.utcnow() + expires\_delta  
 else:  
 expire = datetime.utcnow() + timedelta(minutes=15)  
 to\_encode.update({"exp": expire})  
 # 生成JWT Token  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 # 返回JWT Token  
 return encoded\_jwt

#### utils/password.py

from passlib.context import CryptContext  
  
# 密码加密对象  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
  
def verify\_password(plain\_password, hashed\_password):  
 """校验密码是否正确"""  
 return pwd\_context.verify(plain\_password, hashed\_password)  
  
  
def get\_password\_hash(password):  
 """密码加密"""  
 return pwd\_context.hash(password)

#### main.py

from fastapi import FastAPI  
  
from router import user\_router  
  
app = FastAPI()  
app.include\_router(user\_router)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import uvicorn  
  
 uvicorn.run("main:app")

#### client.py

import requests  
  
# 登录获取token  
target\_url = "http://localhost:8000/token"  
data = {"username": "zhangdapeng", "password": "zhangdapeng520"}  
response = requests.post(target\_url, data=data)  
print("测试登录获取token：", response.text)  
  
# 获取当前用户  
token = response.json()["access\_token"]  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户：", response.text)  
  
# 获取当前数据  
headers = {"Authorization": "Bearer " + token}  
target\_url = "http://localhost:8000/users/me/items/"  
response = requests.get(target\_url, headers=headers)  
print("测试获取当前用户数据：", response.text)

## 权限管理系统

### 安装依赖

pip install fastapi[all]

pip install uvicorn[standard]

pip install python-jose[cryptography]

pip install passlib[bcrypt]

pip install peewee

pip install pymysql

### 完整代码

#### 目录结构

文本

描述已自动生成

#### 安装依赖

pip install fastapi[all]

pip install uvicorn[standard]

pip install python-jose[cryptography]

pip install passlib[bcrypt]

pip install peewee

pip install pymysql

#### config/\_\_init\_\_.py

#### config/server.py

服务器相关配置：

# 接口前缀  
API\_PREFIX = "/api/v1"

#### config/jwt.py

JWT Token相关配置：

# 秘钥：openssl rand -hex 32  
SECRET\_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"  
# 加密算法  
ALGORITHM = "HS256"  
# JWT Token的过期时间  
ACCESS\_TOKEN\_EXPIRE\_MINUTES = 60 \* 20

#### config/db.py

参数说明：

● DB\_HOST：数据库IP地址

● DB\_PORT：数据库服务端口号

● DB\_USER：数据库用户名

● DB\_PASSWORD：数据库密码

● DB\_DATABASE：数据库名

● db\_config：数据库配置信息

数据库相关配置：

DB\_HOST = 'localhost'  
DB\_PORT = 3306  
DB\_USER = 'root'  
DB\_PASSWORD = 'root'  
DB\_DATABASE = 'test'  
  
db\_config = {  
 'host': DB\_HOST,  
 'port': DB\_PORT,  
 'user': DB\_USER,  
 'password': DB\_PASSWORD,  
 'database': DB\_DATABASE,  
}

#### utils/\_\_init\_\_.py

#### utils/common.py

判断两个变量是否相等的方法：

def is\_equal(a, b):  
 """判断是否相等"""  
 return a == b

#### utils/jwt.py

方法说明：

● get\_token：生成JWT Token字符串

● parse\_token：解析JWT Token字符串

代码：

from datetime import datetime, timedelta  
from jose import jwt  
from config.jwt import SECRET\_KEY, ALGORITHM, ACCESS\_TOKEN\_EXPIRE\_MINUTES  
  
  
def get\_token(data: dict, expires\_minute: int = ACCESS\_TOKEN\_EXPIRE\_MINUTES):  
 """  
 生成token  
 :param data: 要打包的数据  
 :param expires\_delta: 过期时间  
 :return: JWT Token  
 """  
 to\_encode = data.copy()  
 expire = datetime.utcnow() + timedelta(minutes=expires\_minute)  
 to\_encode.update({"exp": expire})  
 encoded\_jwt = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 return encoded\_jwt  
  
  
def parse\_token(token) -> dict:  
 """  
 解析Token  
 """  
 return jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

#### utils/password.py

方法说明：

● get\_password：密码加密

● check\_password：检查密码是否正确

代码：

from passlib.context import CryptContext  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
  
def check\_password(password\_text, password\_hash):  
 """密码校验"""  
 return pwd\_context.verify(password\_text, password\_hash)  
  
  
def get\_password(password\_text):  
 """密码加密"""  
 return pwd\_context.hash(password\_text)

#### utils/response.py

方法说明：

● get\_response：获取通用的响应，默认是成功的响应信息

● get\_error\_response：获取错误响应信息

● get\_param\_error\_response：获取参数错误响应信息

● get\_not\_found\_response：获取数据不存在响应信息

● get\_exitsts\_response：获取数据已存在响应信息

代码：

def get\_response(msg="success", code=10000, status=True, data=None):  
 """成功响应"""  
 response = {"status": status, "code": code, "msg": msg}  
 if data is not None:  
 response["data"] = data  
 return response  
  
  
def get\_error\_response(msg="服务器内部错误", code=10003):  
 """错误响应"""  
 return get\_response(code=code, status=False, msg=msg)  
  
  
def get\_param\_error\_response(msg):  
 """参数错误响应"""  
 return get\_error\_response(msg, code=10005)  
  
  
def get\_not\_found\_response(msg):  
 """数据不存在错误响应"""  
 return get\_error\_response(msg, code=10004)  
  
  
def get\_exists\_response(msg):  
 """数据已存在错误响应"""  
 return get\_error\_response(msg, code=10006)

#### db/\_\_init\_\_.py

#### db/base.py

核心功能：提供了断线重连机制，解决了MySQL重启后，API接口失效的问题。

import pymysql  
from peewee import \*  
from playhouse.shortcuts import ReconnectMixin  
from config.db import db\_config  
  
pymysql.install\_as\_MySQLdb()  
  
  
# 同步数据库  
# 同步数据库断线重连类  
class ReconnectMySQLDatabase(ReconnectMixin, MySQLDatabase):  
 pass  
  
  
# 数据库实例  
db = ReconnectMySQLDatabase(\*\*db\_config)  
  
  
# 基础模型  
class BaseModel(Model):  
 class Meta:  
 database = db  
  
  
db.connect()

#### db/user.py

方法说明：

● db\_register：创建一个新的用户

● db\_get\_user\_by\_username：根据用户名获取用户信息

代码：

from peewee import \*  
from db.base import BaseModel  
from utils.password import get\_password  
  
  
# 创建用户表  
class User(BaseModel):  
 username = CharField()  
 password = CharField()  
 role\_id = IntegerField()  
  
  
def db\_register(username, password, role\_id=0):  
 """用户注册"""  
 hashed\_password = get\_password(password)  
 user = User(username=username, password=hashed\_password, role\_id=role\_id)  
 user.save()  
  
  
def db\_get\_user\_by\_username(username):  
 """根据用户名获取用户信息"""  
 users = list(User.select().filter(User.username == username))  
 if len(users) > 0:  
 return users[0]  
 return None

#### schemas/\_\_init\_\_.py

#### schemas/user.py

对象说明：

● LoginSchema：用户登录对象

● RegisterSchema：用户注册对象

● UpdatePasswordSchema：更新密码对象

代码：

from pydantic import BaseModel  
  
  
class LoginSchema(BaseModel):  
 username: str  
 password: str  
  
  
class RegisterSchema(LoginSchema):  
 re\_password: str  
  
  
class UpdatePasswordSchema(RegisterSchema):  
 old\_password: str

#### depends/\_\_init\_\_.py

#### depends/user.py

from fastapi import Depends, HTTPException, status, Header  
from fastapi.security import OAuth2PasswordBearer  
from jose import JWTError  
from utils.jwt import parse\_token  
from db.user import db\_get\_user\_by\_username  
from config.server import API\_PREFIX  
  
# OAuth2登录，文档登录  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl=f"{API\_PREFIX}/token")  
  
  
def get\_user(token):  
 """根据Token解析登录用户信息"""  
 credentials\_exception = HTTPException(  
 status\_code=status.HTTP\_401\_UNAUTHORIZED,  
 detail="权限校验失败",  
 headers={"WWW-Authenticate": "Bearer"},  
 )  
  
 if not token:  
 raise credentials\_exception  
  
 try:  
 # 解析token  
 payload = parse\_token(token)  
  
 # 校验用户名  
 username: str = payload.get("username")  
 if username is None:  
 raise credentials\_exception  
 except JWTError:  
 raise credentials\_exception  
  
 # 获取用户  
 user = None  
 try:  
 user = db\_get\_user\_by\_username(username)  
 if user is None:  
 raise credentials\_exception  
 except Exception as e:  
 print(e)  
 raise credentials\_exception  
  
 return user  
  
  
def get\_login\_user(token: str = Depends(oauth2\_scheme)):  
 """获取登录用户"""  
 return get\_user(token)  
  
  
def get\_json\_login\_user(zdppy\_jwt\_token: str = Header()):  
 """JSON接口获取登录用户"""  
 return get\_user(zdppy\_jwt\_token)

#### routers/\_\_init\_\_.py

#### routers/common.py

from fastapi import APIRouter, Depends  
from fastapi.security import OAuth2PasswordRequestForm  
from utils.response import \*  
from router.user import get\_user\_data  
  
router = APIRouter()  
  
  
@router.post("/token", summary="Form登录")  
async def login\_form(form\_data: OAuth2PasswordRequestForm = Depends()):  
 print("Doc文档登录", form\_data.username, form\_data.password)  
 response = get\_user\_data(form\_data.username, form\_data.password)  
 data = response.get("data")  
 return {"access\_token": data.get("token"), "token\_type": "bearer"}

#### routers/user.py

from fastapi import APIRouter, Depends  
from db.user import db\_register, db\_get\_user\_by\_username  
from utils.common import is\_equal  
from utils.password import check\_password  
from utils.response import \*  
from schemas.user import RegisterSchema, LoginSchema, UpdatePasswordSchema  
from utils.jwt import get\_token  
from utils.password import get\_password  
from depends.user import get\_json\_login\_user  
  
router = APIRouter()  
  
  
# 注册接口  
@router.post("/register", summary="用户注册")  
async def register(schema: RegisterSchema):  
 """  
 注册接口  
 """  
 # 校验两次密码是否一致  
 if not is\_equal(schema.password, schema.re\_password):  
 return get\_param\_error\_response("两次密码不一致")  
  
 # 校验用户名是否已存在  
 try:  
 user = db\_get\_user\_by\_username(schema.username)  
 if user is not None:  
 return get\_exists\_response("该用户已存在")  
 except Exception as e:  
 print(e)  
 return get\_error\_response(msg="连接MySQL服务失败")  
  
 # 新增用户  
 try:  
 db\_register(schema.username, schema.password)  
 return get\_response()  
 except Exception as e:  
 print(e)  
  
 return get\_error\_response()  
  
  
def get\_user\_data(username, password):  
 # 获取用户  
 user = None  
 try:  
 user = db\_get\_user\_by\_username(username)  
 if user is None:  
 return get\_not\_found\_response("该用户不存在")  
 except Exception as e:  
 print(e)  
 return get\_error\_response(msg="连接MySQL服务失败")  
  
 # 校验密码是否正确  
 if not check\_password(password, user.password):  
 return get\_param\_error\_response("用户名或密码错误")  
  
 # 生成token  
 data = {  
 "id": user.id,  
 "username": user.username,  
 "role\_id": user.role\_id,  
 }  
 data["token"] = get\_token(data=data)  
 return get\_response(data=data)  
  
  
@router.post("/login", summary="JSON登录")  
async def login\_json(json\_data: LoginSchema):  
 return get\_user\_data(json\_data.username, json\_data.password)  
  
  
@router.put("/password", summary="密码修改")  
async def put\_password(schema: UpdatePasswordSchema):  
 # 获取用户get\_login\_user  
 user = None  
 try:  
 user = db\_get\_user\_by\_username(schema.username)  
 if user is None:  
 return get\_not\_found\_response("该用户不存在")  
 except Exception as e:  
 print(e)  
 return get\_error\_response(msg="连接MySQL服务失败")  
  
 # 校验旧密码  
 if not check\_password(schema.old\_password, user.password):  
 return get\_param\_error\_response("用户名或密码错误")  
  
 # 校验两次密码是否一致  
 if not is\_equal(schema.password, schema.re\_password):  
 return get\_param\_error\_response("两次密码不一致")  
  
 # 修改密码  
 user.password = get\_password(schema.password)  
 try:  
 user.save()  
 except Exception as e:  
 print(e)  
 return get\_error\_response(msg="连接MySQL服务失败")  
  
 return get\_response()  
  
  
@router.get("/userinfo", summary="获取用户信息")  
async def get\_userinfo(user=Depends(get\_json\_login\_user)):  
 """  
 获取用户信息  
 """  
 data = {  
 "id": user.id,  
 "username": user.username,  
 "role\_id": user.role\_id,  
 }  
 return get\_response(data=data)

#### routers/goods.py

from fastapi import APIRouter  
from utils.response import \*  
  
router = APIRouter()  
  
  
# 注册接口  
@router.get("/goods", summary="获取商品列表")  
async def get\_goods(page: int = 1, size: int = 20):  
 """获取商品信息"""  
 return get\_response()

#### init\_db.py

初始化数据库：

from db.user import User, db\_register  
from db.base import db  
  
  
def init\_user():  
 """初始化用户"""  
 db\_register("zhangdapeng", "zhangdapeng")  
  
  
def init\_db():  
 """初始化数据库"""  
 tables = [User]  
 db.drop\_tables(tables)  
 db.create\_tables(tables)  
  
 init\_user()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 init\_db()

#### main.py

入口程序主要引入子路由并挂载到APP。

from fastapi import FastAPI, Depends  
from config.server import API\_PREFIX  
from depends.user import get\_json\_login\_user  
from router.user import router as user\_router  
from router.goods import router as good\_router  
  
app = FastAPI()  
app.include\_router(user\_router, prefix=API\_PREFIX, tags=["用户管理"])  
app.include\_router(good\_router, prefix=API\_PREFIX, tags=["商品管理"],  
 dependencies=[Depends(get\_json\_login\_user)])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import uvicorn  
  
 uvicorn.run("main:app")

#### 测试

初始化数据库

python init\_db.py

启动服务

python main.py

访问接口文档：http://127.0.0.1:8000/docs

# 项目部署

## 部署简单项目

目录结构

图形用户界面, 文本, 应用程序

描述已自动生成

### Dockerfile

# 父容器  
FROM zhangdapeng520/fastapi:v1  
  
# 复制项目  
COPY ./ /app  
  
# 工作目录  
WORKDIR /app  
  
# 执行命令  
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

### main.py

from typing import Union  
  
from fastapi import FastAPI  
  
app = FastAPI()  
  
  
@app.get("/")  
def read\_root():  
 return {"Hello": "World"}  
  
  
@app.get("/items/{item\_id}")  
def read\_item(item\_id: int, q: Union[str, None] = None):  
 return {"item\_id": item\_id, "q": q}

### 构建容器

创建镜像：

docker build -t api .

创建容器：

docker run -d --name api -p 8000:8000 api

访问测试：http://localhost:8000/docs