

数据库期中作业样例

本次上机任务

本次的上机任务主要是熟悉期中作业相关的一些技术，**并不完全和真正的大作业相同**，具体大作业的内容还请查看《数据管理技术期中大作业》文档：

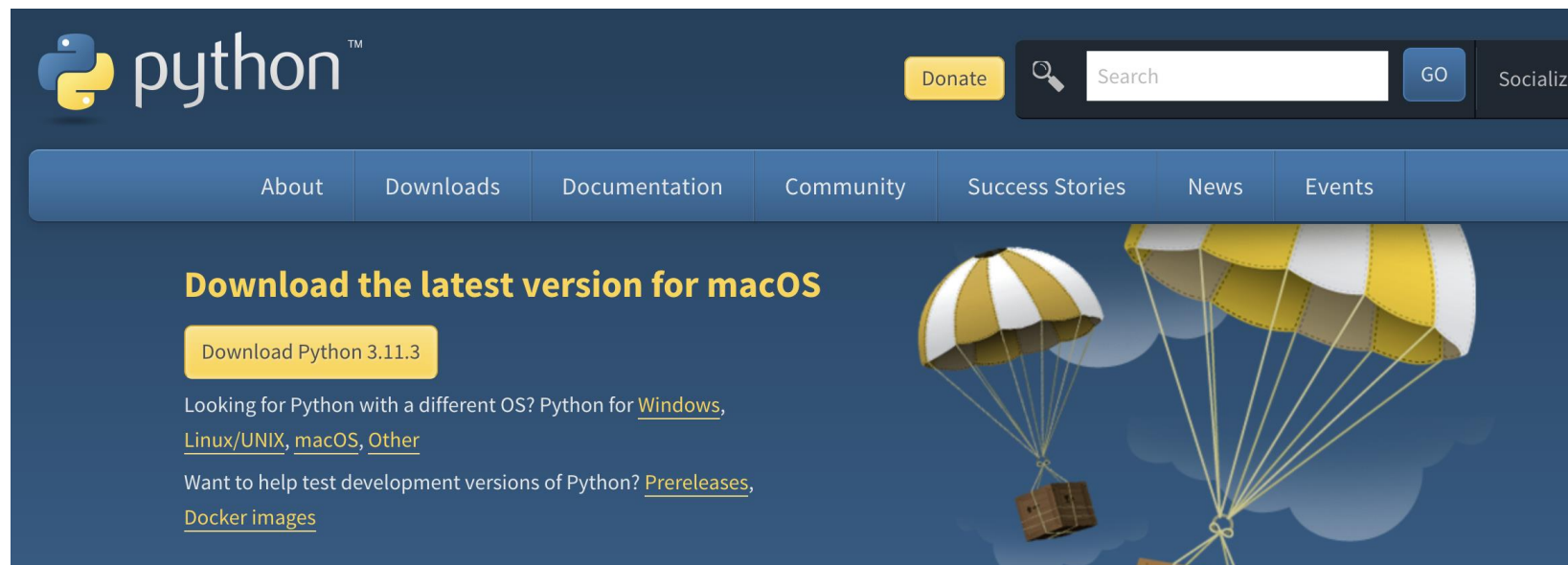
- ❑ 相关软件安装flask和PyMySQL
- ❑ 数据库连接和CURD
- ❑ 提供RESTful接口服务
- ❑ 测试RESTful服务

TASK 1 python包安装

1. 安装python

<https://www.python.org/downloads/>

下载对应操作系统的安装包并且安装，若已安装好python，直接到下一步



TASK 1 python包安装

2. 安装flask和PyMySQL

```
pip install flask
```

```
pip install PyMySQL (python3应该默认安装, 如果没有执行安装一下)
```

如果下载速度比较慢, 可以尝试使用清华镜像

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple package
```

例如:

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple flask
```

TASK 2 数据库连接和CURD

1. 使用mysql-connector-python包进行数据库的连接。

运行下面的代码，连接成功可以看到数据库版本信息

```
import pymysql

# 打开数据库连接
db = pymysql.connect(host='localhost',
                    user='root',
                    password='password', # 填你自己的password
                    database='testdb',
                    port=3360 # 端口
                    )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# 使用 execute() 方法执行 SQL 查询
cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取单条数据.
data = cursor.fetchone()

print ("Database version : %s " % data)

# 关闭数据库连接
db.close()
```

✓ 0.0s Python

Database version : 8.0.32

TASK 2 数据库连接和CURD

1. 创建表

```
import pymysql

# 打开数据库连接
db = pymysql.connect(host='localhost',
                    user='root',
                    password='password', # 填你自己的password
                    database='testdb',
                    port=3360 # 端口
                    )

# 使用 cursor() 方法创建一个游标对象 cursor
cursor = db.cursor()

# 使用 execute() 方法执行 SQL，如果表存在则删除
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# 使用预处理语句创建表
sql = """CREATE TABLE EMPLOYEE (
    FIRST_NAME  CHAR(20) NOT NULL,
    LAST_NAME   CHAR(20),
    AGE         INT,
    SEX         CHAR(1),
    INCOME      DOUBLE )"""

cursor.execute(sql)

# 关闭数据库连接
db.close()
```

✓ 0.0s

Python

TASK 2 数据库连接和CURD

2. 插入数据

```
import pymysql

# 打开数据库连接
db = pymysql.connect(host='localhost',
                    user='root',
                    password='password', # 填你自己的password
                    database='testdb',
                    port=3360 # 端口
                    )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 插入语句
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
                        LAST_NAME, AGE, SEX, INCOME)
VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

# 以上语句也可以写成这样
"""
sql = "INSERT INTO EMPLOYEE(FIRST_NAME, \
    LAST_NAME, AGE, SEX, INCOME) \
    VALUES ('%s', '%s', %s, '%s', %s)" % \
    ('Mac', 'Mohan', 20, 'M', 2000)
"""

try:
    # 执行sql语句
    cursor.execute(sql)
    # 提交到数据库执行
    db.commit()
except:
    # 如果发生错误则回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

TASK 2 数据库连接和CURD

3. 更新数据

```
import pymysql

# 打开数据库连接
db = pymysql.connect(host='localhost',
                     user='root',
                     password='password', # 填你自己的password
                     database='testdb',
                     port=3360 # 端口
                     )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 更新语句
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')
try:
    # 执行SQL语句
    cursor.execute(sql)
    # 提交到数据库执行
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```


TASK 2 数据库连接和CURD

4. 删除数据

```
import pymysql

# 打开数据库连接
db = pymysql.connect(host='localhost',
                     user='root',
                     password='password', # 填你自己的password
                     database='testdb',
                     port=3360 # 端口
                     )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 删除语句
sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20)
try:
    # 执行SQL语句
    cursor.execute(sql)
    # 提交修改
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭连接
db.close()
```

TASK 2 数据库连接和CURD

可以看到，使用PyMySQL操作数据库其实还是需要写SQL语句的

如果是使用ORM框架的话，就可以避免这个问题。下面我们**以User表为例测试一下**

- ❑ 首先是连接并且建立对应的model。
- ❑ 使用db.create_all()创建上面定义的User模型(数据表)
- ❑ 这里需要安装新的依赖
- ❑ pip install Flask
- ❑ pip install Flask-SQLAlchemy

```
from flask import Flask, jsonify, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:123456@localhost:3306/test?charset=utf8mb4'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
print(db)

# 定义模型
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(128), nullable=False)
    age = db.Column(db.Integer, nullable=False)

    def to_dict(self):
        return {
            'id': self.id,
            'name': self.name,
            'age': self.age
        }

with app.app_context():
    # 创建数据库表
    db.create_all()
```

TASK 2 数据库连接和CURD

- ❑ 对于数据的插入，直接新建一个User对象，然后使用add()方法就可以添加到数据库
- ❑ 这里注意需要使用commit()方法确认提交
- ❑ 查询User表的时候可以使用query()来实现，all()代表查询全部的结果。
- ❑ 执行右边的代码可以看到User表中已有的数据
- ❑ 对于删除和更新操作，请同学们自行实现。

```
with app.app_context():
    # 创建数据库表
    db.create_all()

    # 插入新的数据
    new_user = User(name='test', age=18)
    db.session.add(new_user)
    db.session.commit()

    users = User.query.all()
    for user in users:
        print(user.to_dict())
```

TASK 3 提供RESTful接口服务

□ 提供一个数据插入的RESTful接口

假设右边代码保存在main.py文件中

运行代码 `python main.py`

看到下面的输出就说明服务已经正常启动了

还可以通过查看端口情况验证（自己尝试一下）

```
(base) ~ % test % python test.py
* Serving Flask app 'test'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 138-031-838
```

```
from flask import Flask, request, jsonify
import pymysql

app = Flask(__name__)
db = pymysql.connect(
    host="localhost",
    user="root",
    password="123456",
    database="test",
    port=3306
)

# 插入数据
@app.route('/data', methods=['POST'])
def insert_data():
    data = request.json
    table_name = data['table_name']
    rows = data['rows']
    cursor = db.cursor()
    for row in rows:
        keys = ', '.join(row.keys())
        values = ', '.join([f'"{value}"' for value in row.values()])
        cursor.execute(f'INSERT INTO {table_name}({keys}) VALUES({values})')
    db.commit()
    return jsonify({'message': f'{len(rows)} rows inserted into {table_name} successfully'}), 201

if __name__ == '__main__':
    app.run(debug=True)
```

TASK 4 测试RESTful服务

1. 使用curl测试

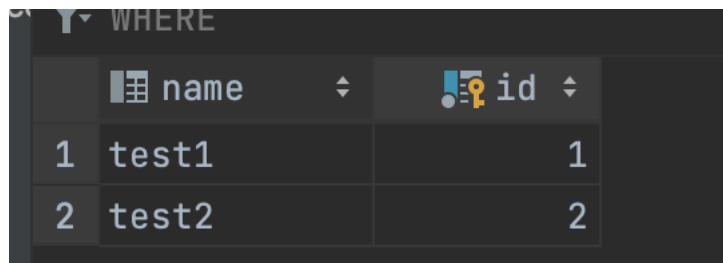
上一个task中，我们部署了数据插入的服务，这里对其进行验证

首先在数据库中**新建一个表** `table_test`，其columns有两列，分别是id: int和name: varchar(20)

```
curl -X POST -H "Content-Type: application/json" -d '{"table_name": "table_test", "rows": [ {"id": 1, "name": "test1"}, {"id": 2, "name": "test2"} ]}' http://127.0.0.1:5555/data
```

在命令行执行上述curl命令，成功后会看到返回信息。（部署服务那里也会有成功的信息）

查看数据库可以看到数据已经插入成功



The screenshot shows a database table with two columns: 'name' and 'id'. The first row contains 'test1' and '1', and the second row contains 'test2' and '2'.

	name	id
1	test1	1
2	test2	2

TASK 4 测试RESTful服务

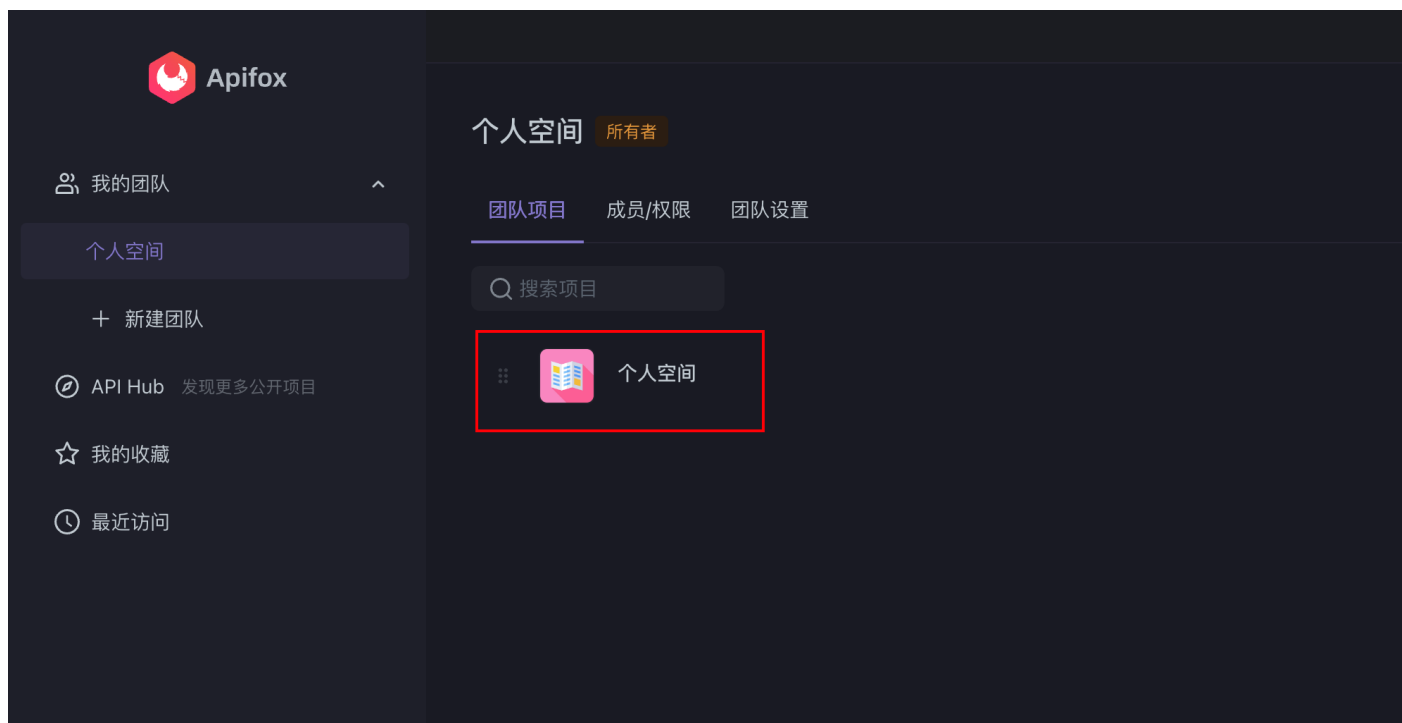
2. 使用Apifox测试

为了简便，这里演示使用网页端。大家可以下载应用端，具体使用流程类似

首先网页登陆Apifox

https://apifox.com/?utm_source=360&utm_medium=sem&utm_term=postman&qhclickid=bf7e410853238f8c

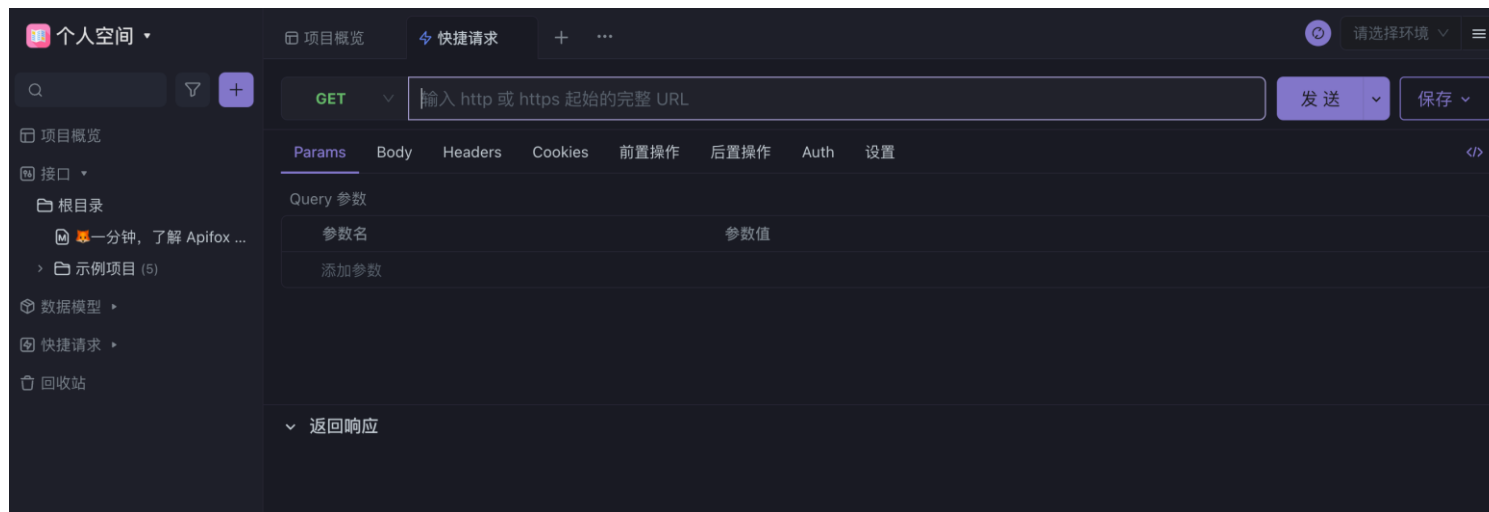
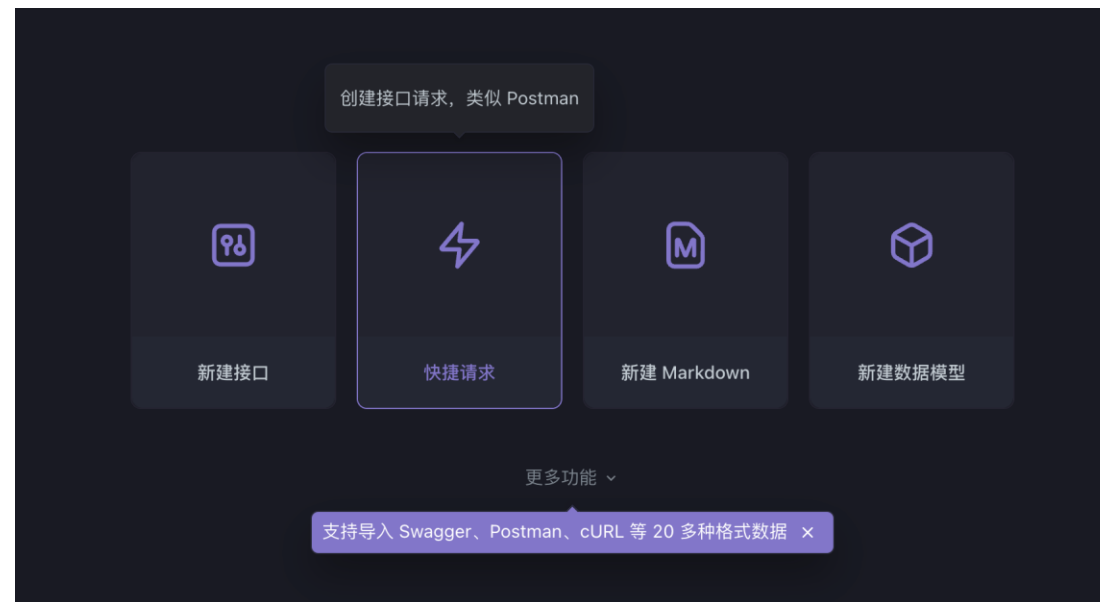
登陆后进入个人空间



TASK 4 测试RESTful服务

2. 使用Apifox测试

然后点击快捷请求，进入具体的请求页面

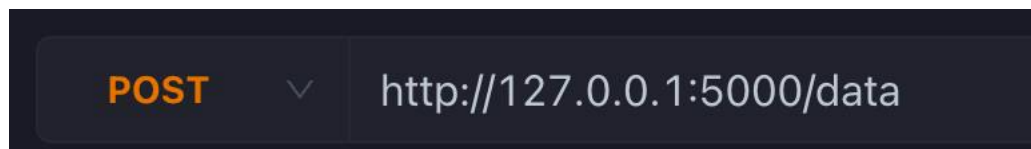


TASK 4 测试RESTful服务

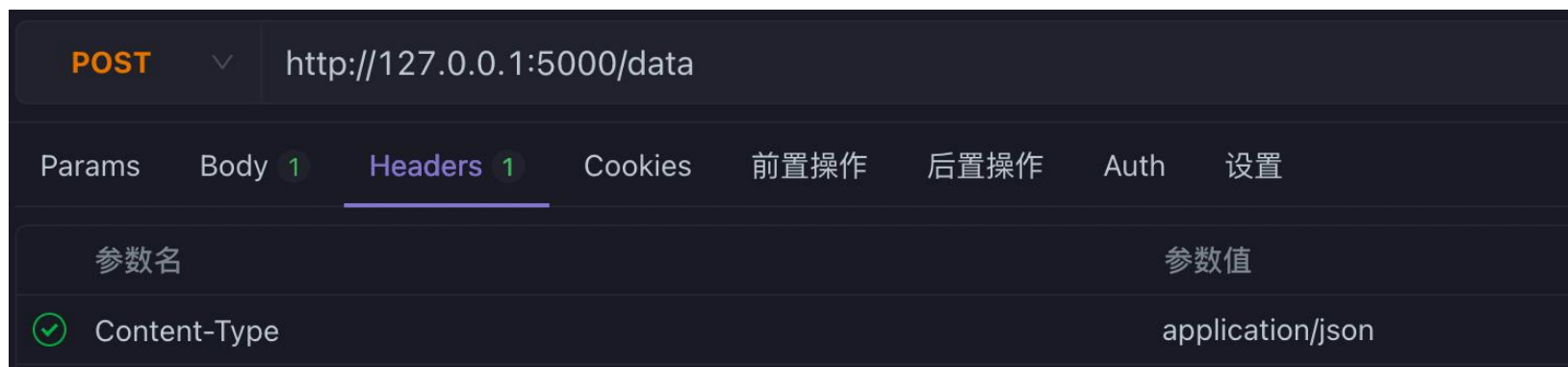
2. 使用Apifox测试

这里配置一些参数，以及请求的具体数据

第一步配置地址：http://127.0.0.1:5000/data，类型选择为POST



第二步配置Content-type: application/json

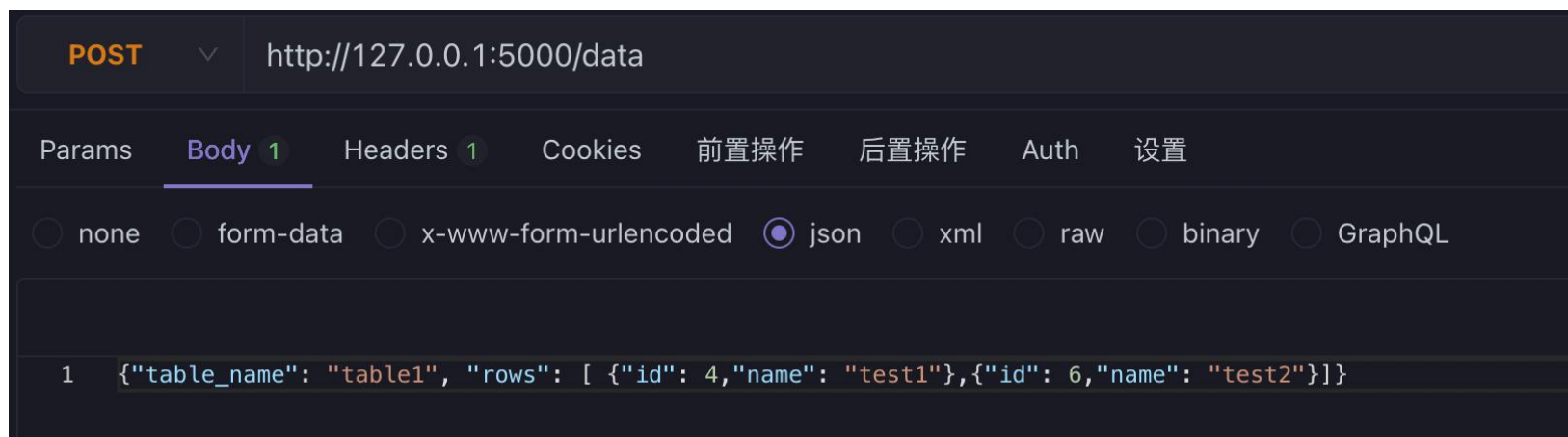


TASK 4 测试RESTful服务

2. 使用Apifox测试

这里配置一些参数，以及请求的具体数据

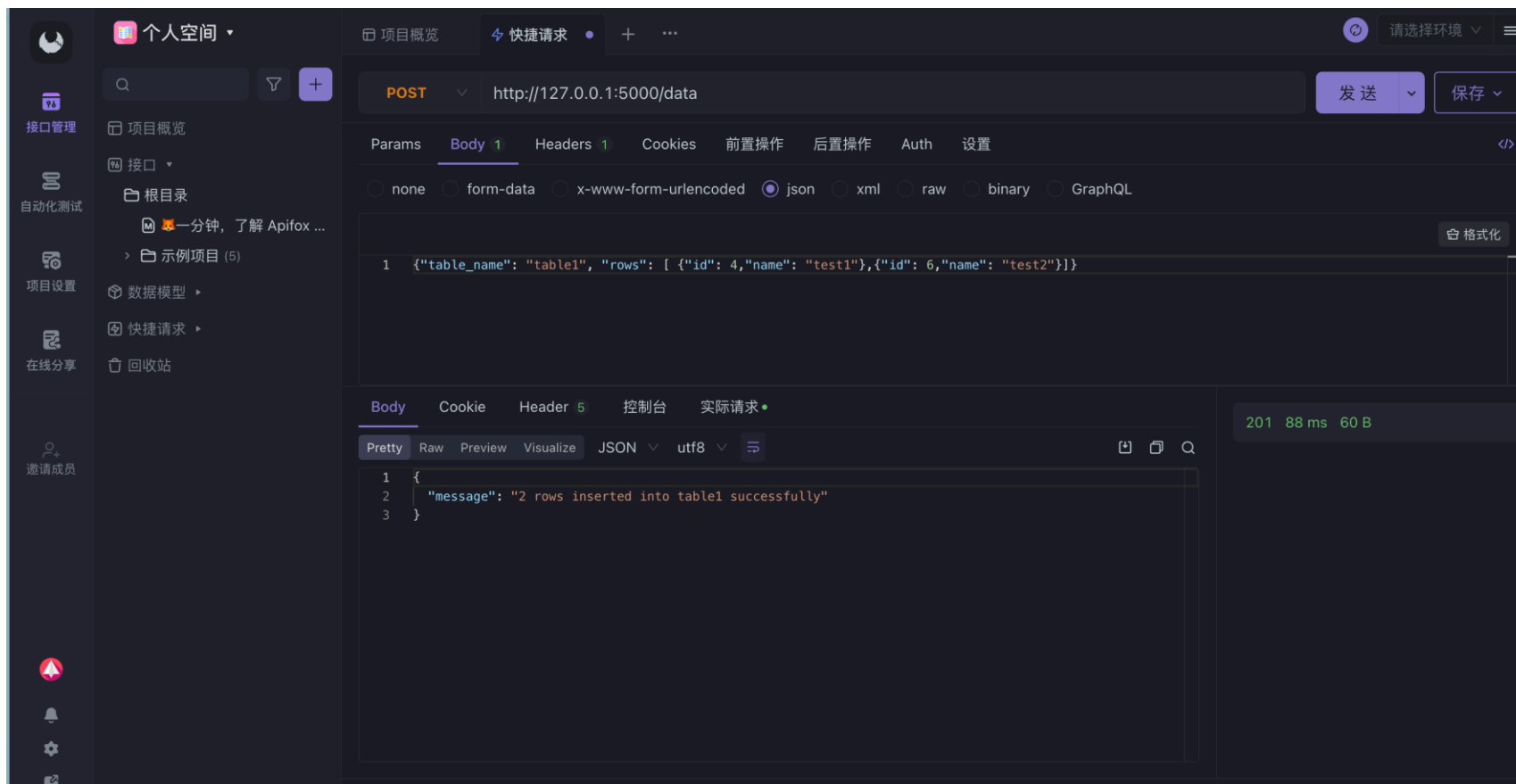
第三步配置数据，类型选择json



TASK 4 测试RESTful服务

2. 使用postman测试

点击发送，成功后可以看到返回的结果如下图



关于测试

■我们的测试分两类：

■SQL相关测试：

- 基本查询：针对数据表内容的MySQL语句查询；
- 索引查询：对特定的数据表进行索引查询（需要同学们预先使用代码对数据表进行索引建立）
- 触发器查询：针对触发器逻辑进行mysql查询

■ RESTful服务测试：

- 针对大家实现的RESTful服务进行测试

关于作业提交

□ 要求

使用ORM框架搭建一个简易的“前”后端系统，提供数据插入、更新、删除、查询等4个接口，并将其部署为RESTful服务。

□ 作业内容

本次大作业所提交的目录文件如图1，数据库的开发部分位于Test1-ORM.py和Test2-RESTful文件
我们已经提供了Departments数据库模型的定义代码（图2）、Departments表的数据注入（图3）、
数据插入接口（图4）

```
C:.\n|  insert.sh\n|  start.sh\n└─src\n\n    Test1-ORM.py\n    Test2-RESTful.py
```

图1 文件目录结构

```
class Departments(db.Model):\n    dept_no = db.Column(db.CHAR(4), primary_key=True)\n    dept_name = db.Column(db.String(40), nullable=False, unique=True)\n    dept_emps = db.relationship('Dept_emp', backref='department', lazy=True, cascade='all, delete')\n    dept_managers = db.relationship('Dept_manager', backref='department', lazy=True, cascade='all, delete')\n\n    def to_dict(self):\n        return {\n            'dept_no': self.dept_no,\n            'dept_name': self.dept_name\n        }
```

图2 Departments数据库模型

```
# department注入\nreader = read_csv_file('./departments.csv')\n# data = [{'dept_no': row['dept_no'], 'dept_name': row['dept_name']} for row in reader]\nrows = []\nfor row in reader:\n    department = Departments(\n        dept_no=row['dept_no'],\n        dept_name=row['dept_name']\n    )\n    rows.append(department)\n\nsession.bulk_save_objects(rows)\nsession.commit()
```

图3 Departments数据注入

```
# 插入数据\n@app.route('/api/v1/<table_name>', methods=['POST'])\ndef insert_data(table_name):\n    data = request.json\n    rows = data['rows']\n    cursor = db.cursor()\n    for row in rows:\n        keys = ', '.join(row.keys())\n        values = ', '.join([f'"{value}"' for value in row.values()])\n\n        sql = f'insert into {table_name}({keys}) values({values})'\n        cursor.execute(sql)\n    db.commit()\n    return jsonify({'message': 'data inserted successfully'}), 201
```

图4 RESTful插入接口样例

关于作业提交

□ TODO

- ✓ TODO1: 在Test1-ORM.py文件中, 参照样例, 完善实验全部内容
- ✓ TODO2: 在Test2-RESTful.py文件中, 参照样例, 完善实验全部内容
- ✓ TODO3: 实验报告: 叙述自己的开发思路以及使用的技术。报告格式不限, 但是需要包含上述的作业内容过程说明。

请保持原有的文件目录, 实验报告放置于根目录下, 所有文件打包命令为 “学号-姓名-数据管理技术期中大作业”, 请大家用tar.gz压缩格式, 提交到云平台。

截止时间: 2024/05/14 晚上12: 00

相关参考

<https://www.myfreax.com/curl-rest-api/>

<https://juejin.cn/post/7128307721954148366>

[RESTful 架构详解 | 菜鸟教程 \(runoob.com\)](#)

[Welcome to Flask — Flask Documentation \(2.2.x\) \(palletsprojects.com\)](#)

<https://docs.docker.com/engine/reference/commandline/run/>

[\(28条消息\) python Flask-ORM操作MYSQL数据库_flask mysql orm_笑得好虚伪的博客-CSDN博客](#)

关于Flask和RESTful相关的教程在百度和google上可以找到很多，请大家自行搜索。