Table of Contents

接口测试课程

课程目标

- 能够根据接口API文档编写接口测试用例
- 能够使用Postman工具进行接口测试，并能够对大量接口用例进行管理、对接口响应结果进行断言、处理多接口的依赖及生成测试报告
- 能够使用Python+Requests封装的接口测试框架，实现接口对象封装、测试用例编写、测试数据管理及生成测试报告

课程大纲

章节	知识点	
第1章 接口测试基础	1.接口及接口测试概念 2.HTTP协议 3.接口规范	4.接口测试流程 5.项目环境说明
第2章 Postman实现接口测试	1.Postman介绍和安装 2.Postman基本用法 3.Postman高级用法	4.Postman测试报告 5.项目实战
第3章 数据库操作	1. 数据库介绍 2. 数据库基本操作	3. 数据库事务操作
第4章 代码实现接口测试	1. Requests库 2. 集成UnitTest	3. 接口测试框架开发 4. 项目实战
第5章 持续集成	1. 持续集成介绍 2. Git与Git代码托管平台 3. Jenkins	4. 持续集成之Postman 5. 持续集成之代码
第6章 接口测试扩展	1. 接口Mock测试	2. 接口测试总结

数据库操作

目标

1. 了解数据库的概念
2. 掌握操作数据库的基本流程
3. 掌握使用PyMySQL对数据库的增、删、改、查
4. 掌握使用PyMySQL对数据库的事务操作

数据库介绍

目标

1. 了解数据库的概念
2. 了解Python操作MySQL的方式

---

1. 数据库的概念

数据库(Database)就是一个存放数据的仓库，这个仓库是按照一定的数据结构来组织、存储和管理数据的。

1.1 数据库的分类

- 关系型数据库：MySQL、Oracle、SQL Server、SQLite等
- 非关系型数据库：Redis、MongoDB等

---

2. Python操作MySQL的方式

使用Python操作MySQL数据库的方式(驱动)有很多种，比如：PyMySQL、MySQLdb、mysqlclient、SQLAlchemy等等

1. MySQLdb
    - 是 Python 连接 MySQL 最流行的一个驱动，很多框架都也是基于此库进行开发
    - 只支持 Python2.x，并且和Windows 平台的兼容性不好
    - 年久失修，现在基本不推荐使用
2. mysqlclient
    - 是MySQLdb的衍生版本，完全兼容 MySQLdb，同时支持 Python3.x
    - 是 Django ORM的依赖工具
3. PyMySQL
    - 是纯 Python 实现的驱动，同时也兼容 MySQLdb
    - 安装、使用比较简单
4. SQLAlchemy
    - 是一种既支持原生 SQL，又支持 ORM 的工具
    - 类似于 Java 中的 Hibernate 框架

数据库基本操作

目标

1. 掌握操作数据库的基本流程
2. 掌握使用PyMySQL对数据库的增、删、改、查

---

1. 使用PyMySQL操作MySQL

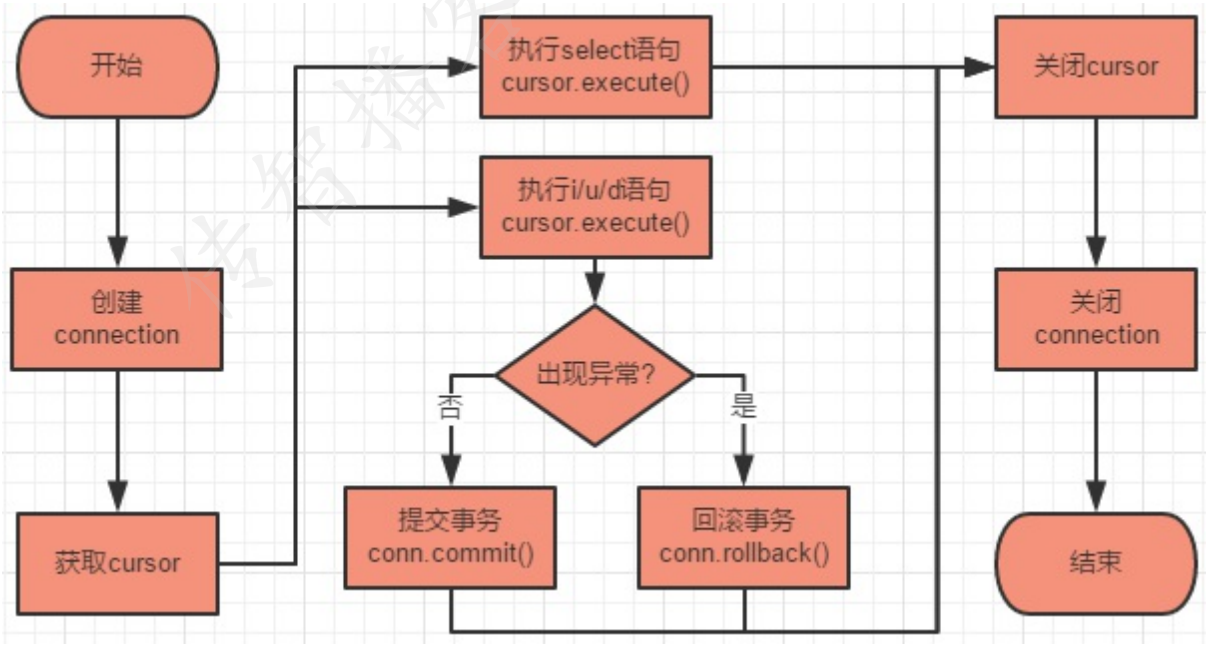
1.1 安装PyMySQL

下载地址：https://github.com/PyMySQL/PyMySQL

使用pip安装：

```
pip install PyMySQL
```

1.2 操作数据库的基本流程



代码实现步骤

1. 导包
2. 创建数据库连接

3. 获取游标对象
4. 执行操作
5. 关闭游标对象
6. 关闭数据库连接

2. 案例数据准备

初始化数据库

```
CREATE DATABASE IF NOT EXISTS books default charset utf8;
USE books;

DROP TABLE IF EXISTS `t_book`;
CREATE TABLE `t_book` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `title` varchar(20) NOT NULL COMMENT '图书名称',
  `pub_date` date NOT NULL COMMENT '发布日期',
  `read` int(11) NOT NULL DEFAULT '0' COMMENT '阅读量',
  `comment` int(11) NOT NULL DEFAULT '0' COMMENT '评论量',
  `is_delete` tinyint(1) NOT NULL DEFAULT '0' COMMENT '逻辑删除',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='图书表';

INSERT INTO `t_book` VALUES ('1', '射雕英雄传', '1980-05-01', '12', '34', '0');
INSERT INTO `t_book` VALUES ('2', '天龙八部', '1986-07-24', '36', '40', '0');
INSERT INTO `t_book` VALUES ('3', '笑傲江湖', '1995-12-24', '20', '80', '0');

DROP TABLE IF EXISTS `t_hero`;
CREATE TABLE `t_hero` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL COMMENT '姓名',
  `gender` smallint(6) NOT NULL COMMENT '性别',
  `description` varchar(200) DEFAULT NULL COMMENT '描述',
  `is_delete` tinyint(1) NOT NULL DEFAULT '0' COMMENT '逻辑删除',
  `book_id` int(11) NOT NULL COMMENT '所属图书ID',
  PRIMARY KEY (`id`),
  KEY `t_hero_book_id` (`book_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COMMENT='英雄人物表';

INSERT INTO `t_hero` VALUES ('1', '郭靖', '1', '降龙十八掌', '0', '1');
INSERT INTO `t_hero` VALUES ('2', '黄蓉', '0', '打狗棍法', '0', '1');
INSERT INTO `t_hero` VALUES ('3', '乔峰', '1', '降龙十八掌', '0', '2');
INSERT INTO `t_hero` VALUES ('4', '令狐冲', '1', '独孤九剑', '0', '3');
INSERT INTO `t_hero` VALUES ('5', '任盈盈', '0', '弹琴', '0', '3');
```

3. 数据库的基本操作

> 数据库的基本操作包括：增、删、改、查
> 对数据库进行操作之前，首先要获取到数据库的连接

3.1 连接数据库

调用 `pymysql.connect()` 方法创建数据库连接

```
conn = pymysql.connect(host=None, user=None, password="", database=None, port=0, autoco
mmit=False)
    host: 数据库服务器地址
    user: 登录用户名
    password: 密码
    database: 要连接的数据库名称
    port: 数据库连接端口(默认值: 3306)
    autocommit: 是否开启自动提交事务(默认值: False)
```

案例

```
1).连接到数据库（host:localhost user:root password:root database:books）
2).获取数据库服务器版本信息
```

示例代码

```
# 导包
import pymysql

# 创建数据库连接
conn = pymysql.connect("localhost", "root", "root", "books")

# 创建游标对象
cursor = conn.cursor()

# 执行操作：查询数据库版本信息
cursor.execute("select version()")

# 获取查询结果
result = cursor.fetchone()
print("result=", result)

# 关闭游标对象
cursor.close()
```

```
# 关闭数据库连接
conn.close()
```

3.2 数据库查询操作

常用的查询方法

- fetchone(): 获取下一个查询结果集，结果集是一个对象
- fetchall(): 获取全部的返回结果行
- rowcount: 获取execute()方法执行后影响的行数

案例

```
1).连接到数据库（host:localhost user:root password:root database:books）
2).查询图书表的数据（包括：图书id、图书名称、阅读量、评论量）
3).获取查询结果的总记录数
4).获取查询结果的第一条数据
5).获取全部的查询结果
```

示例代码

```
# 导包
import pymysql

# 创建数据库连接
conn = pymysql.connect("localhost", "root", "root", "books")

# 创建游标对象
cursor = conn.cursor()

# 执行操作：查询全部的图书数据
cursor.execute("select id,title,`read`,`comment` from t_book")

# 获取查询结果的总记录数
print("rowcount=", cursor.rowcount)

# 获取下一条数据
one = cursor.fetchone()
print("one=", one)

# 获取所有行
book_list = cursor.fetchall()
print("book_list=", book_list)
for book in book_list:
```

```
    print("book=", book)

# 关闭游标对象
cursor.close()

# 关闭数据库连接
conn.close()
```

3.3 数据库插入操作

案例

```
1).连接到数据库（host:localhost user:root password:root database:books autocommit:True）
2).新增一条图书数据（id:4 title:西游记 pub_date:1986-01-01 ）
```

示例代码

```
# 导包
import pymysql

# 创建数据库连接
conn = pymysql.connect("localhost", "root", "root", "books", autocommit=True)

# 创建游标对象
cursor = conn.cursor()

# 执行操作：新增一条图书数据
sql = "insert into t_book(id,title,pub_date) values(4,'西游记','1986-01-01')"
cursor.execute(sql)

# 关闭游标对象
cursor.close()

# 关闭数据库连接
conn.close()
```

3.4 数据库更新操作

案例

```
1).连接到数据库（host:localhost user:root password:root database:books autocommit:True）
2).把图书名称为‘西游记’的阅读量加一
```

示例代码

```
# 导包
import pymysql

# 创建数据库连接
conn = pymysql.connect("localhost", "root", "root", "books", autocommit=True)

# 创建游标对象
cursor = conn.cursor()

# 执行操作：把图书名称为‘西游记’的阅读量加一
sql = "update t_book set `read`=`read`+1 where title='西游记'"
cursor.execute(sql)

# 关闭游标对象
cursor.close()

# 关闭数据库连接
conn.close()
```

3.5 数据库删除操作

案例

```
1).连接到数据库（host:localhost user:root password:root database:books autocommit:True）
2).删除名称为‘西游记’的图书
```

示例代码

```
# 导包
import pymysql

# 创建数据库连接
conn = pymysql.connect("localhost", "root", "root", "books", autocommit=True)

# 创建游标对象
cursor = conn.cursor()

# 执行操作：删除名称为‘西游记’的图书
sql = "delete from t_book where title='西游记'"
cursor.execute(sql)
```

```
# 关闭游标对象
cursor.close()

# 关闭数据库连接
conn.close()
```

数据库事务操作

目标

1. 了解数据库事务的概念
2. 掌握使用PyMySQL对数据库的事务操作

---

1. 案例

1.1 需求

1).连接到数据库（host:localhost user:root password:root database:books），并开启自动提交事务
2).新增一条图书数据（id:4 title:西游记 pub_date:1986-01-01 ）
3).故意抛出一个异常（模拟代码出现异常）
4).新增一条英雄人物数据（name:孙悟空 gender:1 book_id:4）

1.2 出现的问题

- 图书数据添加成功，英雄人物数据添加失败
- 出现数据不一致的情况（不完整）

---

2. 事务

事务**(Transaction)**：是并发控制的基本单位。所谓的事务，它是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。

> 例如：银行转账，从一个账号扣款并使另一个账号增款，这两个操作要么都执行，要么都不执行。

数据库事务：是指一个逻辑工作单元中执行的一系列操作，要么完全地执行，要么完全地不执行。

2.1 事务的特征(ACID)

- 原子性(Atomicity)：事务中包含的操作被看做一个逻辑单元，这个逻辑单元中的操作要么全部成功，要么全部失败

- 一致性(Consistency)：事务的结果保留不变，即事务的运行并不改变数据的一致性
- 隔离性(Isolation)：又称孤立性，事务的中间状态对其它事务是不可见的
- 持久性(Durability)：指一个事务一旦提交成功，它对数据库中数据的改变就应该是永久性的

2.2 事务提交机制

- 自动提交
- 手工提交

---

3. 数据库事务操作

3.1 事务相关的操作方法

- conn.autocommit(False): 设置是否开启自动提交事务，默认不开启
- conn.commit(): 提交数据库事务
- conn.rollback(): 回滚事务

示例代码

```
# 导包
import pymysql

conn, cursor = None, None
try:
    # 创建数据库连接，不开启自动提交事务
    conn = pymysql.connect("localhost", "root", "root", "books", autocommit=False)

    # 获取游标对象
    cursor = conn.cursor()

    # 新增一条图书数据
    cursor.execute("insert into t_book(id,title,pub_date) values(4,'西游记','1986-01-01
')")

    # 抛出一个异常
    raise Exception("故意抛出了一个异常")

    # 新增一条英雄人物数据
    cursor.execute("insert into t_hero(name,gender,book_id) values('孙悟空',1,4)")

    # 提交事务
    conn.commit()
except Exception as e:
    print("error:", e)
```

```
    # 回滚事务
    conn.rollback()
finally:
    # 关闭游标对象
    if cursor:
        cursor.close()

    # 关闭数据库连接
    if conn:
        conn.close()
```

---

4. 封装数据库操作工具类

为了减少代码的冗余，提高测试效率，可以对数据库的相关操作封装成工具类。

实现的功能：

- 获取数据库连接对象
- 关闭数据库连接对象
- 获取游标对象
- 关闭游标对象
- 查询一条记录

示例代码

```
import pymysql

class DBUtil:
    _conn = None  # 数据库连接对象

    @classmethod
    def get_conn(cls):
        """获取数据库连接对象"""
        if cls._conn is None:
            cls._conn = pymysql.connect("localhost", "root", "root", "books")
        return cls._conn

    @classmethod
    def close_conn(cls):
        """关闭数据库连接"""
        if cls._conn:
            cls._conn.close()
            cls._conn = None

    @classmethod
```

```
    def get_cursor(cls):
        """获取游标对象"""
        return cls.get_conn().cursor()

    @classmethod
    def close_cursor(cls, cursor):
        """关闭游标对象"""
        if cursor:
            cursor.close()

    @classmethod
    def get_one(cls, sql):
        """查询一条记录"""
        data = None
        cursor = None
        try:
            cursor = cls.get_cursor()
            cursor.execute(sql)
            data = cursor.fetchone()
        except Exception as e:
            print("get_one error: ", e)
        finally:
            cls.close_cursor(cursor)
            cls.close_conn()
        return data
```