浙江理工大学科技与艺术学院

KEYI COLLEGE OF ZHEJIANG SCI-TECH UNIVERSITY

程序设计基础课程设计

课程设计报告

课设名称 ： 基于 Python 爬虫+Java 全栈开发的 Tlias 智能管理辅助系统

课设时间： 2025.12.16——2025.12.23

专　　业　大数据管理与应用

班　　级　24 大数据管理与应用 1

学生姓名　王俊翰

学　　号　Xc24590101

指导老师　叶杰锋

需求分析

本次课程设计的核心任务是开发 “基于 Python 爬虫 + Java 全栈开发的 Tlias 智能管理辅助系统”，通过爬虫获取 HTML 页面中的班级与学员数据，结合 Java 后端与前端技术实现一站式管理功能，具体需求如下：

数据爬取需求：目标 爬取对应的 HTML 页面，需精准提取 30 条班级数据（含班级 ID、名称、教室、学科、开课 / 结课时间等）和 30 条学员数据（含姓名、学号、性别、学历、联系地址等），确保数据无重复、格式规范，符合 MySQL 数据库表结构定义。

数据存储需求：将爬取的结构化数据存入 MySQL 数据库，需适配 clazz（班级表）和 student（学员表）的字段约束，包括枚举类型映射（如学科名称转数字编码、性别文本转标识）、唯一字段去重（学号、手机号、身份证号）等，保证数据一致性。

系统功能需求：基于 Java 全栈技术实现多模块管理功能，包括班级学员管理（增删改查）、系统信息管理（部门、员工管理）、数据统计管理（员工性别 / 职位可视化）、文件上传（本地磁盘 + 阿里云 OSS）等，支持前后端异步交互，界面响应流畅。

合规与技术约束：爬取过程严格遵循合法原则，不涉及隐私信息；技术栈需符合课程设计要求,Python 爬虫采用 BeautifulSoup 解析 HTML、pymysql 操作数据库,Java 后端基于 SpringBoot 框架，前端结合 Vue+ECharts 实现可视化与交互。

一、设计思路

1. 开发环境与工具

开发环境：Python 3.9+、JDK 21、MySQL 8.0、IntelliJ IDEA、PyCharm

核心工具与类库：

爬虫部分：requests（请求处理）、BeautifulSoup4（HTML 解析）、pymysql（数据库连接）

Java 后端：SpringBoot、MyBatis、MySQL 驱动、Lombok（后续手动替换）、ECharts（可视化）

前端：HTML+CSS+JS、Vue.js、Ajax（异步交互）

其他：阿里云 OSS SDK（文件上传）、Logback（日志）

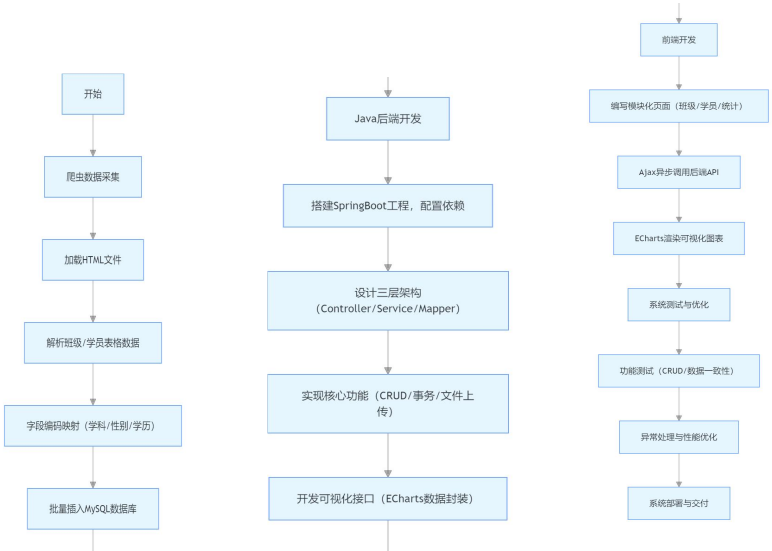
2. 核心设计思路

本系统采用 “爬虫数据采集→数据库存储→前后端全栈开发” 的三段式架构，各环节逻辑衔接紧密：

爬虫模块设计：采用类编程模式封装核心功能，流程为 “加载 HTML 文件→解析表格数据→字段编码映射→批量插入 MySQL”。通过 select 方法定位表格行与列，自定义映射函数将中文文本（如 “本科”“男”）转为数据库枚举编码，利用 executemany 批量插入提升效率，通过 ON DUPLICATE KEY UPDATE 处理主键冲突。

Java 后端设计：遵循三层架构模式，Controller 层接收前端请求并响应结果，Service 层处理业务逻辑（如事务管理、数据补全），Mapper 层负责数据库 CRUD 操作；封装统一响应类 Result，全局异常处理器捕获请求异常，保证系统稳定性；引入 ECharts 实现数据可视化，通过 XML 配置 MyBatis 处理复杂 SQL。

前后端交互设计：采用 Rest 风格 API 接口，前端通过 Ajax 异步发送 GET/POST/PUT/DELETE 请求，后端返回 JSON 格式数据；文件上传支持本地与 OSS 双方案，适配不同存储需求；前端页面按功能模块化划分（班级管理、学员管理、数据统计），保证界面简洁易用。



二、详细设计

1. 分析 HTML 结构

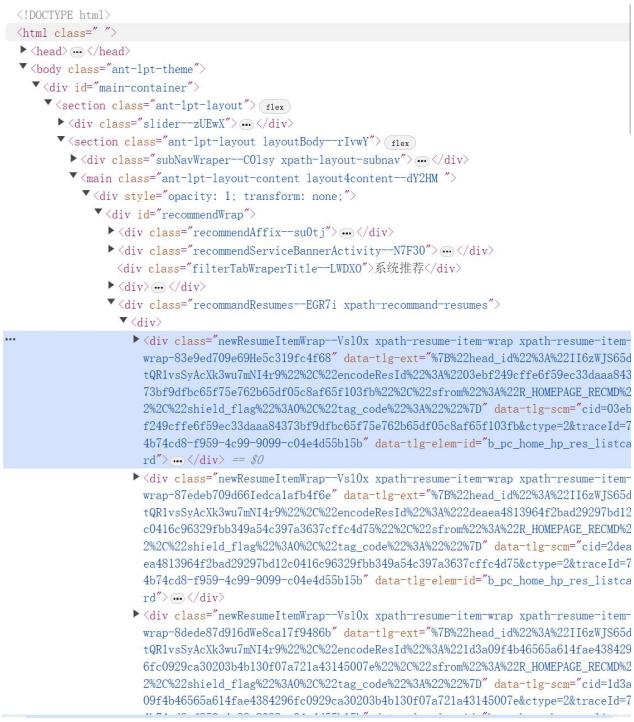


图 1-HTML 页面

2. 分析请求方式（如图：POST 方法）

图 2-请求方法

3.编写爬虫代码

导入所需库

```python
import requests
from bs4 import BeautifulSoup
import pymysql
import time
from datetime import datetime
import logging
```

图 3-python 库

配置爬虫模块（数据库存储用 MySQL）

```python
class Config:  10 用法
    MYSQL_HOST = "localhost"
    MYSQL_PORT = 3306
    MYSQL_USER = "root"
    MYSQL_PASSWORD = "1234"
    MYSQL_DB = "tlias"
```

图 4-爬虫模块

```python
# 爬虫
TARGET_URL = "https://bbs.itheima.com/forum-235-1.html"
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KH
    "Referer": "https://www.itheima.com/"
}
```

图 5-请求头

配置日志模块（方便 debug 进行后续问题排查）

```python
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler(Config.LOG_FILE, encoding="utf-8"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)
```

图 6-日志模块

连接数据库

```python
def connect(self):  1 个用法
    #连接 MySQL数据库
    try:
        self.conn = pymysql.connect(
            host=Config.MYSQL_HOST,
            port=Config.MYSQL_PORT,
            user=Config.MSQL_USER,
            password=Config.MYSQL_PASSWORD,
            db=Config.MYSQL_DB,
            charset="utf8mb4"
        )
        self.cursor = self.conn.cursor(pymysql.cursors.DictCursor)
        logger.info("MySQL 数据库连接成功")
    except Exception as e:
        logger.error( msg: f"MySQL 连接失败：{str(e)}", exc_info=True)
        raise
```

图 7-数据库连接

连接成功

在 DataGraph 用 SQL 语言进行表创建（Student 表、clazz 表）

```sql
create table clazz(
    id   int unsigned primary key auto_increment comment 'ID,主键',
    name  varchar(30) not null unique  comment '班级名称',
    room  varchar(20) comment '班级教室',
    begin_date date not null comment '开课时间',
    end_date date not null comment '结课时间',
    master_id int unsigned null comment '班主任ID, 关联员工表ID',
    subject tinyint unsigned not null comment '学科, 1:java, 2:前端, 3:大数据, 4:Python, 5:Go, 6: 嵌入式',
    create_time datetime  comment '创建时间',
    update_time datetime  comment '修改时间'
)comment '班级表';
```

图 8-表创建

```sql
create table student(
  id int unsigned primary key auto_increment comment 'ID,主键',
  name varchar(10)  not null comment '姓名',
  no char(10)  not null unique comment '学号',
  gender tinyint unsigned  not null comment '性别, 1: 男, 2: 女',
  phone  varchar(11)  not null unique comment '手机号',
  id_card  char(18)  not null unique comment '身份证号',
  is_college tinyint unsigned  not null comment '是否来自于院校, 1:是, 0:否',
  address  varchar(100)  comment '联系地址',
  degree  tinyint unsigned  comment '最高学历, 1:初中, 2:高中, 3:大专, 4:本科, 5:硕士, 6:博士',
  graduation_date date comment '毕业时间',
  clazz_id  int unsigned not null comment '班级ID, 关联班级表ID',
  violation_count tinyint unsigned default '0' not null comment '违纪次数',
  violation_score tinyint unsigned default '0' not null comment '违纪扣分',
  create_time  datetime  comment '创建时间',
  update_time  datetime  comment '修改时间'
) comment '学员表';
```

图 9-表创建

发送请求获取页面内容

```python
def fetch_page(self, url):  2 用法
    max_retries = 3
    for retry in range(max_retries):
        try:
            response = self.session.get(url, timeout=10)
            response.raise_for_status()  # 触发 HTTP 错误
            response.encoding = response.apparent_encoding  # 自动识别编码
            time.sleep(Config.DELAY)  # 延时防反爬
            logger.info(f"成功获取页面: {url}")
            return response.text
        except Exception as e:
            logger.warning(f"获取页面 {url} 失败（第 {retry+1} 次重试）: {str(e)}")
            time.sleep(2 * (retry + 1))  # 重试间隔递增
    logger.error(f"获取页面 {url} 失败（已重试 {max_retries} 次）")
    return None
```

图 10-发送请求

基于 beautifulsoup 解析班级列表页

```python
try:
    clazz_name = item.select_one("h3.class-name").get_text(strip=True)  # 班级名称
    begin_date = item.select_one("span.begin-date").get_text(strip=True)  # 开课时间
    end_date = item.select_one("span.end-date").get_text(strip=True)  # 结课时间
    subject_text = item.select_one("span.subject").get_text(strip=True)  # 学科文本
```

图 11-解析班级数据

解析单个班级的学院列表页

```python
for item in student_items:
    try:
        # 提取学员字段
        student_name = item.select_one("h4.student-name").get_text(strip=True)  # 姓名
        gender_text = item.select_one("span.gender").get_text(strip=True)  # 性别文本
        is_college_text = item.select_one("span.college").get_text(strip=True)  # 是否院校
        degree_text = item.select_one("span.degree").get_text(strip=True)  # 学历文本
        graduation_date = item.select_one("span.grad-date").get_text(strip=True, default=None)  # 毕业时间
        address = item.select_one("span.address").get_text(strip=True, default=None)  # 联系地址
```

图 12-学员字段

成功连接 tlias 数据库并存入数据

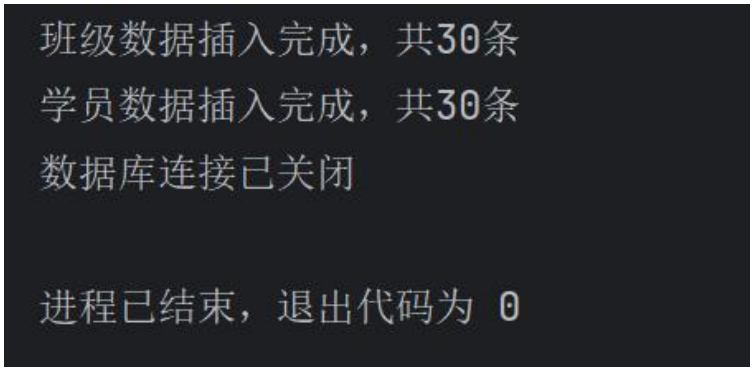


图 13-存取成功

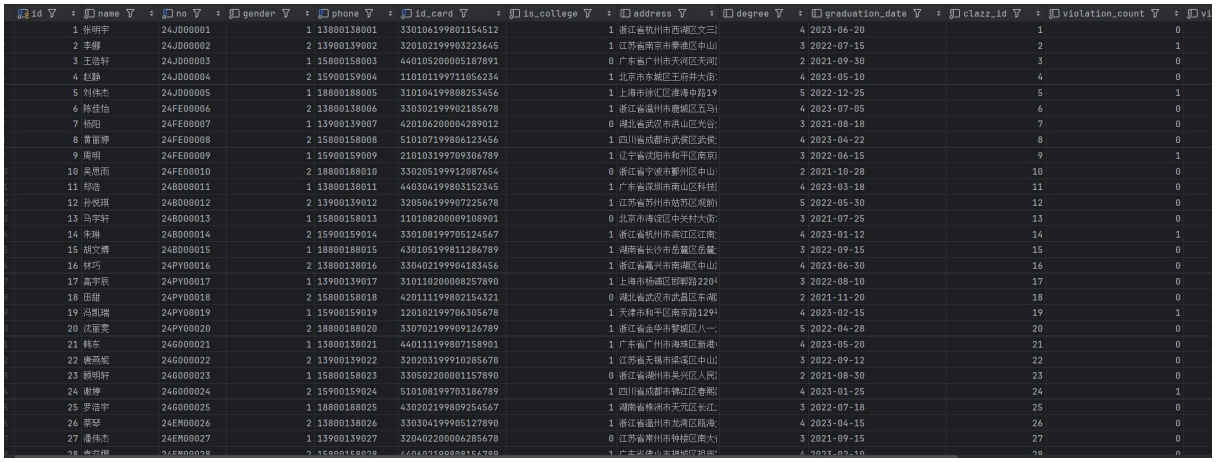


图 14-数据展示

	id	name	room	begin_date	end_date	master_id	subject	create_time	update_time
8	8	前端全栈就业班202403期	科技楼503	2024-03-10	2024-09-10	3	2	2024-03-03 14:30:00	2024-03-03 14:30:00
9	9	前端小程序开发班202404期	实训楼305	2024-04-25	2024-10-25	5	2	2024-04-18 10:50:00	2024-04-19 09:10:00
10	10	前端UI/UX设计班202405期	智慧楼302	2024-05-12	2024-11-12	9	2	2024-05-05 15:20:00	2024-05-05 15:20:00
11	11	大数据开发班202401期	实训楼401	2024-01-22	2024-08-22	2	3	2024-01-15 09:00:00	2024-01-16 13:40:00
12	12	大数据Spark班202402期	科技楼603	2024-02-28	2024-09-28	7	3	2024-02-21 10:30:00	2024-02-21 10:30:00
13	13	大数据Hadoop实战班2024(	智慧楼207	2024-03-25	2024-10-25	4	3	2024-03-18 14:00:00	2024-03-19 11:20:00
14	14	大数据数仓班202404期	实训楼403	2024-04-18	2024-11-18	6	3	2024-04-11 09:50:00	2024-04-11 09:50:00
15	15	大数据AI整合班202405期	科技楼505	2024-05-20	2024-12-20	8	3	2024-05-13 15:00:00	2024-05-14 10:40:00
16	16	Python全栈班202401期	智慧楼201	2024-01-10	2024-07-10	1	4	2024-01-03 09:15:00	2024-01-03 09:15:00
17	17	Python爬虫实战班202402	实训楼306	2024-02-12	2024-08-12	3	4	2024-02-05 10:40:00	2024-02-06 14:30:00
18	18	Python数据分析班202403	科技楼501	2024-03-18	2024-09-18	5	4	2024-03-11 09:30:00	2024-03-11 09:30:00
19	19	Python人工智能班202404	智慧楼301	2024-04-08	2024-10-08	9	4	2024-04-01 11:00:00	2024-04-02 09:20:00
20	20	Python自动化测试班2024(	实训楼402	2024-05-08	2024-11-08	2	4	2024-05-01 14:50:00	2024-05-01 14:50:00
21	21	Go后端开发班202401期	科技楼602	2024-01-28	2024-08-28	7	5	2024-01-21 10:00:00	2024-01-21 10:00:00
22	22	Go云原生实战班202402期	智慧楼204	2024-02-18	2024-09-18	4	5	2024-02-11 09:20:00	2024-02-12 15:10:00
23	23	Go微服务架构班202403期	实训楼307	2024-03-30	2024-10-30	6	5	2024-03-23 14:30:00	2024-03-23 14:30:00
24	24	Go全栈开发班202404期	科技楼504	2024-04-22	2024-11-22	8	5	2024-04-15 11:20:00	2024-04-16 10:30:00
25	25	Go区块链实战班202405期	智慧楼303	2024-05-25	2024-12-25	3	5	2024-05-18 09:40:00	2024-05-19 14:20:00
26	26	嵌入式开发班202401期	实训楼404	2024-01-12	2024-08-12	5	6	2024-01-05 15:00:00	2024-01-05 15:00:00
27	27	嵌入式Linux班202402期	科技楼506	2024-02-25	2024-09-25	9	6	2024-02-18 10:10:00	2024-02-19 09:50:00
28	28	嵌入式物联网班202403期	智慧楼206	2024-03-15	2024-10-15	2	6	2024-03-08 14:00:00	2024-03-08 14:00:00
29	29	嵌入式STM32班202404期	实训楼405	2024-04-12	2024-11-12	7	6	2024-04-05 11:30:00	2024-04-06 15:40:00
30	30	嵌入式智能硬件班202405期	科技楼604	2024-05-10	2024-12-10	4	6	2024-05-03 09:20:00	2024-05-03 09:20:00

图 15-数据展示 2

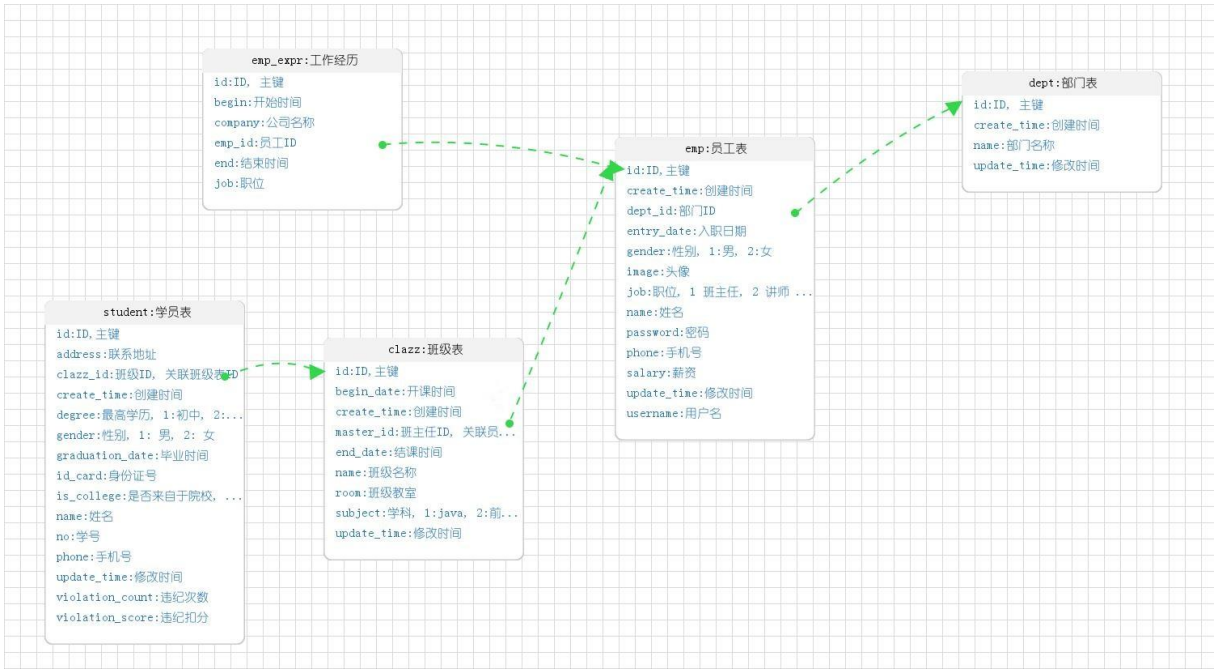


图 16-表结构

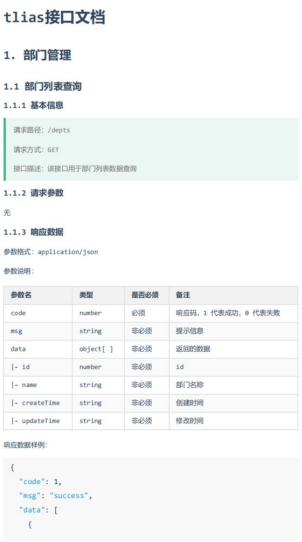
基于 HTML+CSS+JS+Vue 完成前端页面的开发，同时设置 Ajax 异步交互请求在不重新加载整个页面的情况下，与服务器交换数据更新部分网页

图 17-前端页面

基于 Java 开发后端，完成：班级学员管理（crud）、系统信息管理（crud）、数据统计管理（数据可视化）、文件上传 OSS

图 18-功能模块

查看接口文档，同时本次开发基于当前主流的 Rest 风格的 API 接口进行交互



tlias接口文档

1. 部门管理

1.1 部门列表查询

1.1.1 基本信息

请求路径：/depts

请求方式：GET

接口描述：该接口用于部门列表数据查询

1.1.2 请求参数

无

1.1.3 响应数据

参数格式：application/json

参数说明：

参数名	类型	是否必须	备注
code	number	必须	响应码，1 代表成功，0 代表失败
msg	string	非必须	提示信息
data	object[ ]	非必须	返回的数据
|- id	number	非必须	id
|- name	string	非必须	部门名称
|- createTime	string	非必须	创建时间
|- updateTime	string	非必须	修改时间

响应数据样例：

```
{
  "code": 1,
  "msg": "success",
  "data": [
    {
```

图 19-接口文档

工程搭建，创建 SpringBoot 工程，并引入 web 开发起步依赖、mybatis、MySQL 驱动、Lombok

注：这里我把 Lombok 依赖注入之后，后续前后端联调时会出现 406 的错误，排查后应该是 Lombok 没有起到作用，所以后面的操作全部都是手动添加有参无参函数、Getter、Setter 方法以及日志技术

准备基础包结构设置三层架构，Controller：接受前端请求、响应后端数据；Service：业务逻辑处理；Mapper：负责数据访问操作，同时也负责数据的 crud，并引入实体类 Dept 和统一的响应结果封装类 Result

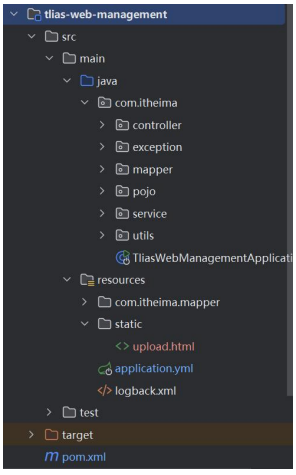


图 20-三层架构

4. 后端代码编写

完成第一个需求：查询部门数据

Controller 层：

```java
//@Slf4j  日志记录
@RequestMapping("/depts") //优化代码，将公共的请求路径前缀统一抽取到类上  嘎嘎嘎
@RestController
public class DeptController {
    //用@Slf4j注解，将日志记录功能封装到类中，不用再写日志记录代码，简化下面一长串代码
    private static final org.slf4j.Logger log  = LoggerFactory.getLogger(DeptController.class);  4 个用法

    @Autowired
    private DeptService deptService;
    //查询部门
    //@RequestMapping(value = "/depts",method = RequestMethod.GET) //method: 指定请求的方式
    //@GetMapping(value = "/depts", produces = MediaType.APPLICATION_JSON_VALUE)//spring框架中指定的请求方式，指定返回数据为json格式
    @GetMapping  嘎嘎嘎
    public Result list(){
        //System.out.println("查询全部部门数据");//测试代码,输出到控制台
        log.info("查询全部部门数据");
        List<Dept> deptList =  deptService.findAll();
        return Result.success(deptList);
    }
```

图 21-Controller 层

Service 层：添加@Service 注解，加入到 IOC 容器

在 DeptService 接口增加 findAll 方法：

```java
//部门管理业务接口
public interface DeptService {  4 个用法  1 个实现  嘎嘎嘎
    /*
    * 查询所有的部门数据
    * */
    List<Dept> findAll();  1 个用法  1 个实现  嘎嘎嘎
```

图 22-Service 层

在 DeptServiceImpl 中增加 findAll 方法：

```java
//部门管理Service接口实现类
@Service  嘎嘎嘎
public class DeptServiceImpl implements DeptService {

    @Autowired
    private DeptMapper deptMapper;

    @Override  1 个用法  嘎嘎嘎
    public List<Dept> findAll() {
        return deptMapper.findAll();
    }
```

图 23-Service 实现类

Mapper 层：添加@Mapper 注解，添加到 IOC 容器；增加 findAll 方法（由于 SQL 语

句较短，无需 XML 映射）

```java
@Mapper//告诉SpringBoot这是一个Mapper类，核心作用是定义对数据库中dept表的查询操作，并利用 MyBatis 的注解特性，将 Java 方法与 SQL 语句直接绑定
//增删改查
public interface DeptMapper {
    /*
    * 查询所有的部门数据
    * */
    @Select("select id,name,create_time,update_time from dept order by update_time desc ") 1 个用法  嘎嘎嘎
    List<Dept> findAll();//查询所有部门数据
```

图 24-Mapper 层

完成第二个需求：删除部门

Controller 层：通过 Spring 提供的@RequestParam 注解，将请求参数绑定给方法形参

```java
    * 方式一：基于HttpServletRequest对象获取参数(操作还需转化，企业用的不多)
    * */
//    @DeleteMapping("/depts")
//    public Result delete(HttpServletRequest request){
//        String idStr = request.getParameter("id");//通过getParameter方法获取的参数都是String类型
//        int i = Integer.parseInt(idStr);//但是对于ID指来说需要转换成int类型，拿到int类型的id值
//        System.out.println("删除部门：" + i);
//        return Result.success();
//    }
    /*
    * 方式二：基于Spring提供的@RequestParam注解获取参数，会自动进行类型转化，将请求参数绑定给方法参数
    * 注意事项：一旦声明了@RequestParam注解，那么这个参数的id值必须存在，否则会报错（400错误）
    * */
    @DeleteMapping  嘎嘎嘎
    public Result delete(@RequestParam("id") Integer deptId){
        //System.out.println("删除部门：" + deptId);
        log.info("删除部门：{}" ,deptId);
        deptService.deleteById(deptId);
        return Result.success();
    }
```

图 25-Controller 层

Service 层：

在 DeptService 接口中，增加 deleteById 方法

```java
//根据ID删除部门
void deleteById(Integer deptId);  1 个用法  1 个实现  嘎嘎嘎
```

图 26-Service 层

在 DeptServiceImpl 中，增加 deleteById 方法

```
@Override 1 个用法  嘎嘎嘎
public void deleteById(Integer id) {
    deptMapper.deleteById(id);
}
```

图 27-Service 实现类

Mapper 层：

```
/*
* 根据ID删除部门
* */
@Delete("delete  from  dept where id = #{id}")//#{id}：表示占位符，表示id参数的值,占位符的值会从方法参数中获取,占位符的值会替换掉#{id},
void deleteById(Integer id);
```

图 28-Mapper 层

完成第三个需求：新增部门

Controller 层：

```
/*
* 新增部门
* */
@PostMapping  嘎嘎嘎
public Result add(@RequestBody Dept dept){//@RequestBody注解可以将一个json格式的请求参数，直接封装到一个对象当中
    //System.out.println("新增部门：" + dept);
    deptService.add(dept);
    return Result.success();
}
```

图 29-Controller 层

Service 层：

在 DeptService 接口中增加接口方法 save：

```
/*
* 新增部门
* */
void add(Dept dept);  1 个实现  嘎嘎嘎
```

图 30-Service 层

在 DeptServiceImpl 实现类中增加 save 方法：

```
@Override  嘎嘎嘎
public void add(Dept dept) {
    //1.补全基础属性-createTime，updateTime
    dept.setCreateTime(LocalDateTime.now());//获取当前系统时间
    dept.setUpdateTime(LocalDateTime.now());
    //2.调用mapper接口方法插入数据
    deptMapper.insert(dept);
}
```

图 31-Service 实现类

Mapper 层：

```
/*
* 新增部门
* */ //字段名是下划线；属性名是驼峰命名
@Insert("insert into dept(name,create_time,update_time) values(#{name},#{createTime},#{updateTime})")//这里注意在#后面不能写cr
void insert(Dept dept);
```

图 32-Mapper 层

完成第四个需求：修改部门

Controller 层：用@PathVariable 注解来获取路径参数；查询回显

```
/*
 * 根据ID查询部门(如果这个路径参数名和这个方法的形参名称一致，即可省略一部分简化写)
 * */
@GetMapping("/{id}")  嘎嘎嘎
public Result getInfo(@PathVariable Integer id){
    //System.out.println("根据ID查询部门：" + id);
    log.info("根据ID查询部门：{}" ,id);
    Dept dept = deptService.getById(id);//调用service层方法查询部门数据有返回数据所以需要设置返回值
    return Result.success(dept);//返回值给前端
}
```

图 33-Controller 层

```
/*
 * 修改部门
 * */
@PutMapping    嘎嘎嘎
public Result update(@RequestBody Dept dept){
    //System.out.println("修改部门: " + dept);
    log.info("修改部门: {}" ,dept);
    deptService.update(dept);
    return Result.success();
}
```

图 34-Controller 层

Service 层：

在 Dept Service 接口中增加 getId 方法和 update 方法：

```
/*
* 根据ID查询部门
* */
Dept getById(Integer id);  1 个用法  1 个实现  嘎嘎嘎
```

图 35-Service 层

```
/*
* 修改部门
* */
void update(Dept dept);  1 个用法  1 个实现  嘎嘎嘎
```

图 36-Service 实现类

在 Dept ServiceImpl 实现类中增加 getId 方法和 update 方法：

图 37-getById 方法

图 38-update 方法

Mapper 层：

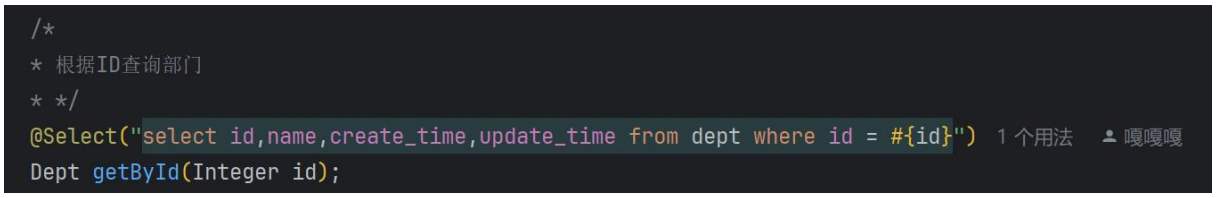


图 39-Mapper 层

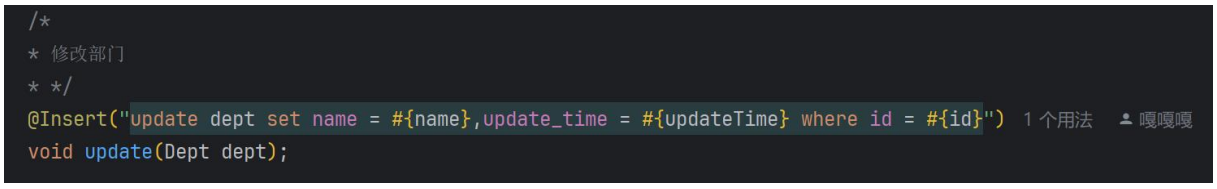


图 40-Mapper 层

注：由于后续员工管理的代码逻辑基本与上述一致，这边就不再重复展示，可参照部门管理的实现方式，依次完成员工的增删改查功能，不同点无非就是 SQL 语句的复杂度和长度提升，因此后续采用 XML 映射形式

5. 日志技术

引入日志技术，方便后续 debug 排查

引入 logback 依赖（SpringBoot 中无需引入，已经传递了此依赖）

引入配置文件

图 41-日志技术

6. 事务管理

由于后续可能会出现 emp 表数据保存成功后，但是 emp_expr 员工工作经历信息表数据保存失败的情况，由于事务的原子性（要么全部成功，要么全部失败），所以通过数据库中的事务来解决这个问题

在业务方法 save 上添加@Transactional 注解来控制事务，有任何异常程序即时回滚：

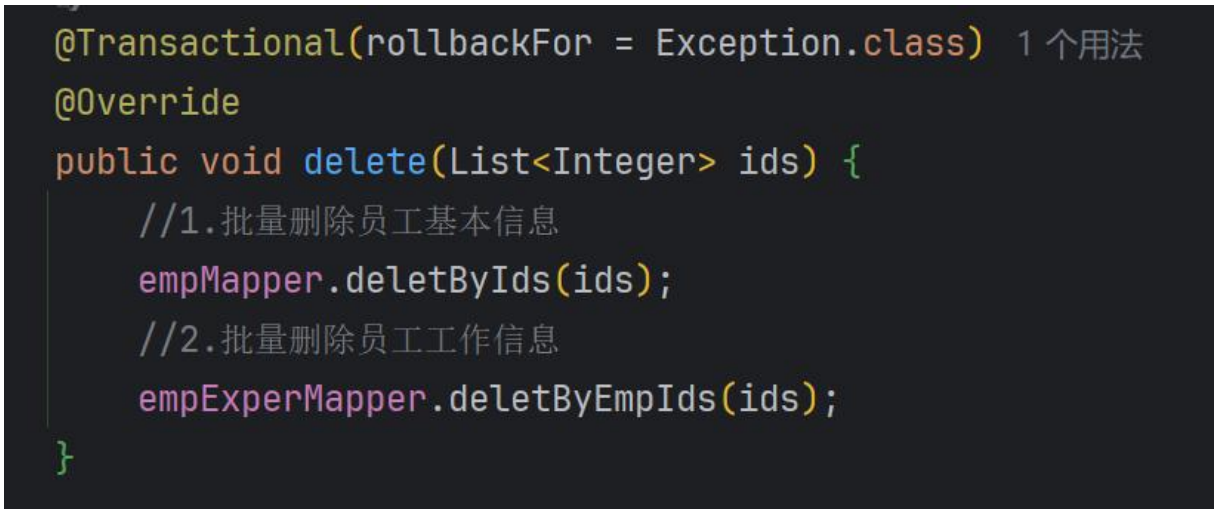


图 42-事务管理

7. 文件上传

这边展示两种形式：本地磁盘文件上传、阿里云 OSS 文件上传

本地磁盘文件上传：

```java
/*
 * 本地磁盘存储方案
 * */

private static final org.slf4j.Logger log  = LoggerFactory.getLogger(UploadController.class); 1 个用法
@PostMapping("/upload")
public Result upload(String name, Integer age, MultipartFile file) throws IOException {
    log.info("文件上传，参数：name={},age={},file={}",name,age,file);
    //获取上传的文件名
    String originalFilename = file.getOriginalFilename();
    //新的文件名
    String extension = originalFilename.substring(originalFilename.lastIndexOf( str: "."));
    String newFileName = UUID.randomUUID().toString() + extension;
    //保存文件
    file.transferTo(new File( pathname: "D:\\" + newFileName));
    return Result.success();
}
```

图 43-文件上传

阿里云 OSS 文件上传：

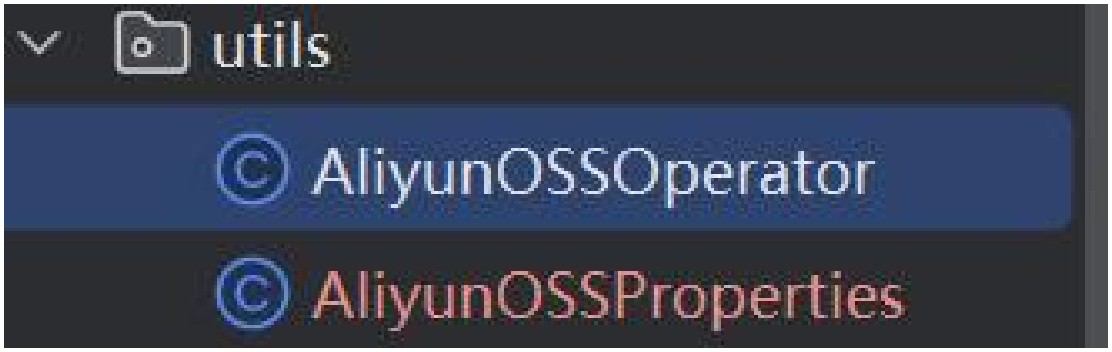


图 44-阿里云

```java
@Autowired
private AliyunOSSOperator aliyunOSSOperator;

@PostMapping("/upload")
public  Result upload(MultipartFile file) throws Exception{
    final org.slf4j.Logger log  = LoggerFactory.getLogger(UploadController.class);
    log.info("文件上传，参数：file={}",file);
    //将文件交给OSS存储服务
    String url = aliyunOSSOperator.upload(file.getBytes(), file.getOriginalFilename());
    log.info("文件上传成功，返回结果：{}",url);
    return Result.success(url);

}
```

图 45-阿里云 OSS 文件代码

```yaml
#阿里云
aliyun:
  oss:
    endpoint : https://oss-cn-beijing.aliyuncs.com
    bucketName : java-data-ai
    region : cn-beijing
```

图 46-OSS 配置代码

图 47-上传成功

8. 全局异常处理器

出现异常之后，异常会被全局异常处理器捕获，然后返回错误信息，被前端程序正常解析

```java
@RestControllerAdvice // 表示当前类是一个全局异常处理器
public class GlobalExceptionHandler {
    private static final org.slf4j.Logger log  = LoggerFactory.getLogger(GlobalExceptionHandler.class); 2 个用法

    @ExceptionHandler// 指定能够处理的异常类型
    public Result handleException(Exception e){
        log.error("程序出现异常",e);
        return Result.error( msg: "服务器异常,请联系王哥~");
    }

    //处理异常
    @ExceptionHandler
    public Result handleDuplicateKeyException(DuplicateKeyException e){ //方法形参中指定能够处理的异常类型
        log.error("程序出现异常",e); //记录异常信息
        String message = e.getMessage();
        int i = message.indexOf("Duplicate entry");
        String errMsg = message.substring(i);
        String[] arr = errMsg.split( regex: " ");
        return Result.error( msg: arr[2]  + "已存在"); //返回错误信息
    }
}
```

图 48-全局异常管理

9. 员工信息统计（数据可视化）

基于 Apache 公司的 Echarts 现成组件完成可视化开发：

图 49-ECharts

Controller 层：

```java
@RequestMapping("/report")
@RestController
public class ReportController {
    private static final org.slf4j.Logger log  = LoggerFactory.getLogger(ReportController.class); 2 个用法

    @Autowired
    private ReportService reportService;

    //统计员工职位人数
    @GetMapping("/empJobData")
    public Result getEmpJobData(){
        log.info("统计员工职位人数");
        JobOption jobOption = reportService.getEmpJobData();
        return Result.success(jobOption);
    }
```

图 50-Controller 层

```
    //统计员工性别人数
    @GetMapping("/empGenderData")
    public Result getEmpGenderData(){
        log.info("统计员工性别人数");
        List<Map<String,Object>> genderList = reportService.getEmpGenderData();
        return Result.success(genderList);
    }
}
```

图 51-Controller 层

Service 层：

定义 ReportService 接口，添加接口方法：

```
public interface ReportService {  4 个用法   1 个实现

    //统计员工职位人数
    JobOption getEmpJobData();  1 个用法   1 个实现
    //统计员工性别人数
    List<Map<String, Object>> getEmpGenderData();  1 个用法   1 个实现
}
```

图 52-Service 层

定义 ReportServiceImpl 实现类，添加接口方法：

```
@Service
public class ReportServiceImpl implements ReportService {

    @Autowired
    private EmpMapper empMapper;

    @Override  1 个用法
    public JobOption getEmpJobData() {
        //1.调用mapper接口，统计统计数据
        List<Map<String, Object>> maps = empMapper.countEmpJobData();//数据格式为 {pos=java, count=1}

        //2.创建两个集合，分别用于封装数据
        List<Object> jobList = maps.stream().map(dataMap -> dataMap.get("pos")).toList();
        List<Object> dataList = maps.stream().map(dataMap -> dataMap.get("total")).toList();

        return new JobOption(jobList, dataList);
    }
```

图 53-Service 实现类

```java
@Override  1 个用法
public List<Map<String, Object>> getEmpGenderData() {
    return empMapper.countEmpGenderData();
}
```

图 54-Service 实现类

Mapper 层：

```java
//统计员工职位人数
@MapKey("pos")  2 个用法
List<Map<String, Object>>countEmpJobData();

//统计员工性别人数
@MapKey("pos")  2 个用法
List<Map<String, Object>> countEmpGenderData();
```

图 55-Mapper 层

图 56-数据可视化

三、心得体会

本次课程设计从 Python 爬虫数据采集到 Java 全栈系统开发，覆盖了数据获取、后端架构、前后端交互等多个技术维度，不仅巩固了专业知识，更提升了问题解决与工程实践能力，具体心得如下：

技术融合与实践突破：通过本次项目，我深入掌握了 Python 爬虫与 Java 全栈的联动开发，理解了 “数据采集 - 存储 - 应用” 的完整链路。例如，爬虫模块中需精准匹配 HTML 表格的 DOM 结构，通过反复调试 select 选择器确保数据提取无误；Java 后端开发中，解决了 Lombok 依赖失效导致的 406 错误，通过手动编写 Getter/Setter 方法完成替代，深刻体会到依赖配置与代码兼容性的重要性。

问题解决与思维提升：项目实施中遇到多个关键问题，均通过逐步排查实现突破。爬虫阶段曾因学号长度超出 char(10)限制导致插入失败，通过缩短学号格式或修改字段长度解决；前后端联调时出现异步请求响应异常，排查发现是 Ajax 请求格式与后端接口不匹配，调整 contentType 为 application/json 后恢复正常；文件上传功能中，成功实现本地存储与阿里云 OSS 的双方案适配，学会了基于配置文件动态切换存储策略。这些经历让我养成了 “定位问题 - 分析原因 - 验证方案” 的逻辑思维。

工程规范与系统设计认知：本次项目严格遵循代码规范与架构设计原则，Python 爬虫采用类封装提升复用性，Java 后端通过三层架构降低耦合度，全局异常处理与事务管理保证系统稳定性。通过开发数据可视化模块，理解了 ECharts 与后端数据的适配逻辑；在班级与学员数据关联中，深刻体会到数据库外键约束的作用，学会了通过 clazz_id 建立表间关联，确保数据一致性。

不足与改进方向：项目仍存在可优化空间，，如：应对复杂网站时鲁棒性不足；前端页面交互较为基础，可增加表单验证、分页查询等功能；后端接口未实现权限控制，后续可引入 Spring Security 提升安全性。通过本次课程设计，我认识到软件开发是一个持续优化的过程，需在实践中不断完善功能与性能。

总体而言，本次课程设计不仅让我熟练掌握了 Python 爬虫与 Java 全栈的核心技术，更培养了工程实践能力与问题解决意识，为后续专业学习与项目开发奠定了坚实基础。

四、核心代码

```
import requests
from bs4 import BeautifulSoup
import pymysql
import time
from datetime import datetime
import logging

class Config:
    MYSQL_HOST = "localhost"
    MYSQL_PORT = 3306
    MYSQL_USER = "root"
    MYSQL_PASSWORD = "1234"
    MYSQL_DB = "tlias"

    # 爬虫配置
    TARGET_URL = "https://bbs.itheima.com/forum-235-1.html"
    HEADERS = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/120.0.0.0 Safari/537.36",
        "Referer": "https://www.itheima.com/"
    }
    DELAY = 1.5
    LOG_FILE = "crawler.log"

logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
    handlers=[
        logging.FileHandler(Config.LOG_FILE, encoding="utf-8"),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

class MySQLHandler:
    def __init__(self):
        self.conn = None
        self.cursor = None
        self.connect()
```

```
    def connect(self):
        try:
            self.conn = pymysql.connect(
                host=Config.MYSQL_HOST,
                port=Config.MYSQL_PORT,
                user=Config.MYSQL_USER,
                password=Config.MYSQL_PASSWORD,
                db=Config.MYSQL_DB,
                charset="utf8mb4"
            )
            self.cursor = self.conn.cursor(pymysql.cursors.DictCursor)
            logger.info("MySQL 数据库连接成功")
        except Exception as e:
            logger.error(f"MySQL 连接失败：{str(e)}", exc_info=True)
            raise

    def close(self):
        if self.cursor:
            self.cursor.close()
        if self.conn:
            self.conn.close()
        logger.info("MySQL 数据库连接关闭")

    def insert_clazz(self, clazz_data):
        sql = """
        INSERT INTO clazz (name, room, begin_date, end_date, master_id, subject, create_time,
update_time)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
        ON DUPLICATE KEY UPDATE
            room = VALUES(room), begin_date = VALUES(begin_date), end_date =
VALUES(end_date),
            master_id = VALUES(master_id), subject = VALUES(subject), update_time =
VALUES(update_time)
        """
        try:
            params = (
                clazz_data["name"],  # 班级名称
                clazz_data.get("room"),  # 教室
                clazz_data["begin_date"],  # 开课时间
                clazz_data["end_date"],  # 结课时间
                clazz_data.get("master_id"),  # 班主任 ID
                clazz_data["subject"],  # 学科
                datetime.now(),  # 创建时间
                datetime.now()   # 修改时间
```

```
            )
            self.cursor.execute(sql, params)
            self.conn.commit()
            clazz_id = self.cursor.lastrowid  # 获取插入的班级 ID（用于关联学员表）
            logger.info(f"班级 {clazz_data['name']} 插入成功，ID：{clazz_id}")
            return clazz_id
        except Exception as e:
            self.conn.rollback()
            logger.error(f"班级 {clazz_data.get('name')} 插入失败：{str(e)}", exc_info=True)
            return None

    def insert_student(self, student_data):
        """插入学员数据到 student 表"""
        sql = """
        INSERT INTO student (name, no, gender, phone, id_card, is_college, address, degree,
                             graduation_date, clazz_id, violation_count, violation_score,
create_time, update_time)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
        ON DUPLICATE KEY UPDATE
            gender = VALUES(gender), phone = VALUES(phone), is_college = VALUES(is_college),
            address = VALUES(address), degree = VALUES(degree), graduation_date =
VALUES(graduation_date),
            clazz_id = VALUES(clazz_id), violation_count = VALUES(violation_count),
            violation_score = VALUES(violation_score), update_time = VALUES(update_time)
        """
        try:
            params = (
                student_data["name"],  # 姓名
                student_data.get("no"),  # 学号
                student_data["gender"],  # 性别
                student_data.get("phone"),  # 手机号
                student_data.get("id_card"),  # 身份证号
                student_data["is_college"],  # 是否院校
                student_data.get("address"),  # 联系地址
                student_data.get("degree"),  # 最高学历
                student_data.get("graduation_date"),  # 毕业时间
                student_data["clazz_id"],  # 关联班级 ID（
                student_data.get("violation_count", 0),  # 违纪次数
                student_data.get("violation_score", 0),  # 违纪扣分
                datetime.now(),  # 创建时间
                datetime.now()  # 修改时间
            )
            self.cursor.execute(sql, params)
            self.conn.commit()
```

```
                logger.info(f"学员 {student_data['name']} 插入成功")
            except Exception as e:
                self.conn.rollback()
                logger.error(f"学员 {student_data.get('name')} 插入失败：{str(e)}", exc_info=True)
class HeimaCrawler:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update(Config.HEADERS)
        self.mysql_handler = MySQLHandler()

    def fetch_page(self, url):
        max_retries = 3
        for retry in range(max_retries):
            try:
                response = self.session.get(url, timeout=10)
                response.raise_for_status()
                response.encoding = response.apparent_encoding
                time.sleep(Config.DELAY)
                logger.info(f"成功获取页面：{url}")
                return response.text
            except Exception as e:
                logger.warning(f"获取页面 {url} 失败（第 {retry+1} 次重试）：{str(e)}")
                time.sleep(2 * (retry + 1))
        logger.error(f"获取页面 {url} 失败（已重试 {max_retries} 次）")
        return None

    def parse_clazz_list(self, html):
        soup = BeautifulSoup(html, "html.parser")
        clazz_list = []
        clazz_items = soup.select("div.class-item")
        for item in clazz_items:
            try:
                # 提取班级字段（根据实际页面标签调整）
                clazz_name = item.select_one("h3.class-name").get_text(strip=True)  # 班级名称
                begin_date = item.select_one("span.begin-date").get_text(strip=True)  # 开课时间
                end_date = item.select_one("span.end-date").get_text(strip=True)  # 结课时间
                subject_text = item.select_one("span.subject").get_text(strip=True)  # 学科文本

                subject_map = {"Java": 1, "前端": 2, "大数据": 3, "Python": 4, "Go": 5, "嵌入式": 6}
                subject = subject_map.get(subject_text, 4)  # 默认 Python（可调整）

                # 构造班级数据
                clazz_data = {
                    "name": clazz_name,
```

```
                    "room": None,
                    "begin_date": begin_date,
                    "end_date": end_date,
                    "master_id": None,
                    "subject": subject
                }
                clazz_list.append(clazz_data)
            except Exception as e:
                logger.error(f"解析班级数据失败：{str(e)}", exc_info=True)
        return clazz_list

    def parse_student_list(self, clazz_id, clazz_url):
        html = self.fetch_page(clazz_url)
        if not html:
            return
        soup = BeautifulSoup(html, "html.parser")
        student_items = soup.select("div.student-item")

        for item in student_items:
            try:
                # 提取学员字段（根据实际页面标签调整）
                student_name = item.select_one("h4.student-name").get_text(strip=True)  # 姓名
                gender_text = item.select_one("span.gender").get_text(strip=True)  # 性别文本
                is_college_text = item.select_one("span.college").get_text(strip=True)  # 是否院校
                degree_text = item.select_one("span.degree").get_text(strip=True)  # 学历文本
                graduation_date = item.select_one("span.grad-date").get_text(strip=True, default=None)
# 毕业时间
                address = item.select_one("span.address").get_text(strip=True, default=None)  # 联
系地址

                gender = 1 if gender_text == "男" else 2
                is_college = 1 if is_college_text == "是" else 0
                degree_map = {"初中": 1, "高中": 2, "大专": 3, "本科": 4, "硕士": 5, "博士": 6}
                degree = degree_map.get(degree_text)

                # 构造学员数据
                student_data = {
                    "name": student_name,
                    "no": None,
                    "gender": gender,
                    "phone": None,
                    "id_card": None,
                    "is_college": is_college,
                    "address": address,
```

```
                    "degree": degree,
                    "graduation_date": graduation_date,
                    "clazz_id": clazz_id
                }
                # 插入学员数据到 MySQL
                self.mysql_handler.insert_student(student_data)
            except Exception as e:
                logger.error(f"解析学员数据失败：{str(e)}", exc_info=True)

    def run(self):
        try:

            logger.info("开始爬取黑马程序员班级信息...")
            clazz_list_html = self.fetch_page(Config.TARGET_URL)
            if not clazz_list_html:
                logger.error("班级列表页获取失败，爬虫终止")
                return


            clazz_list = self.parse_clazz_list(clazz_list_html)
            for clazz_data in clazz_list:
                clazz_id = self.mysql_handler.insert_clazz(clazz_data)
                if not clazz_id:
                    continue
                clazz_url = Config.TARGET_URL + f"/clazz/{clazz_id}"
                logger.info(f"开始爬取班级 {clazz_data['name']} 的学员信息...")


                self.parse_student_list(clazz_id, clazz_url)

            logger.info("爬虫任务执行完成！")
        except Exception as e:
            logger.error(f"爬虫执行异常终止：{str(e)}", exc_info=True)
        finally:

            self.mysql_handler.close()


if __name__ == "__main__":
    crawler = HeimaCrawler()
    crawler.run()
```

指导老师评价：					
项目评价	**优**	**良**	**中**	**一般**	**较差**
课堂纪律：按时上课，无迟到早退					
完成态度：认真完成项目设计，勤奋好学，积极思考					
项目完成度：按照要求完成项目设计					
课程质量：代码正确，按时完成，界面美观					
课程总结：思路清晰，调理清楚，全面总结了心得体会和不足					
总评成绩	**签名：** **时间：**				