北京邮电大学计算机学院

软件工程专业实验报告

课程名称： Java 程 序 设 计

项目名称： java实验2

项目完成人：

姓名： 禄禄鱼

学号：

指导教师： 崔毅东

日 期： 2023 年 12 月 12 日

# 实验概述

## 实验目的

本实验的目的是通过使用Java语言进行基本程序的开发，练习继承、多态、接口、异常处理、多线程同步、Socket通信等。通过本实验，可以加深对Java语言的理解，提高编程能力和解决问题的能力。

## 实验任务

|  |  |
| --- | --- |
|  | 任务说明 |
|  | Access database：编写一个程序，可以查看、插入、更新数据库中的图书信息。使用任意数据库管理系统，创建一个Book表，包含id、title、pubDate、author等字段。程序可以通过命令行输入查询、插入、更新的指令，从数据库中获取或修改数据。 |
|  | Multi-Threading programming：编写一个模拟航空预订系统的程序，使用多个客户线程处理预订请求。使用wait/notify/notifyall机制处理预订冲突和座位可用性。创建一个Flight类，表示共享资源，包含座位数和预订记录。创建多个Customer类，表示线程，尝试在Flight类上预订座位。使用同步机制避免竞态条件，使用wait/notifyall机制实现线程间的通信。 |
|  | Socket programming：编写一个天气查询系统的程序，使用客户端-服务器模式，客户端向服务器发送查询城市的名称，服务器返回该城市的天气数据。使用TCP协议进行Socket通信，使用字符串格式传输数据。服务器可以处理多个顺序的客户端连接。 |
|  |  |
|  |  |
|  |  |
|  |  |

## 实验环境

JDK 17

### 硬件环境

Intel Core i5-11代 内存：16GB 硬盘：512GB SSD

### 网络环境

BUPT校园网

### 软件环境

OS:Windows 10

IDE:JetBrain Intellij

## 实验结果

 本实验完成了三个项目的要求，实现了相应的功能和性能。在实验过程中，遇到了一些问题和困难，通过查阅资料和调试程序，最终解决了。实验结果的截图和文字回答见后续章节

# 项目1

## 问题分析

### 目标

本项目的目标是编写一个可以操作数据库中的图书信息的程序，实现查看、插入、更新的功能。程序可以通过命令行输入指令，从数据库中获取或修改数据。

### 功能

本项目的功能如下：

* 查看功能：当用户输入view指令和一个图书的id时，程序从数据库中查询该id对应的图书记录，如果存在，显示该图书的所有信息，如果不存在，显示“没有找到该图书”的提示信息。
* 插入功能：当用户输入insert指令和一个图书的信息时，程序向数据库中插入一条新的图书记录，如果插入成功，显示“插入成功”的提示信息，如果插入失败，显示“插入失败”的提示信息。
* 更新功能：当用户输入update指令和一个图书的id和新的信息时，程序从数据库中更新该id对应的图书记录，如果更新成功，显示“更新成功”的提示信息，如果更新失败，显示“更新失败”的提示信息。

### 性能

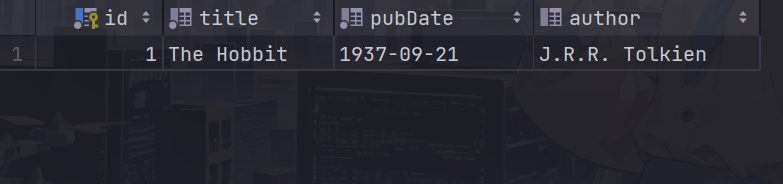
* 程序能够快速响应用户的输入，不超过1秒。
* 程序能够正确执行用户的指令，不出现逻辑错误或者数据丢失。
* 程序能够处理异常情况，如数据库连接失败、输入格式错误、数据冲突等，给出合理的提示信息。

## 实验结果

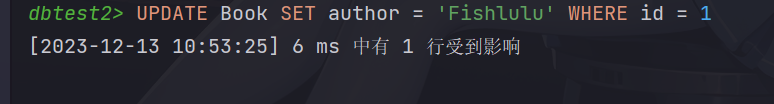
创建数据库表

插入一条记录

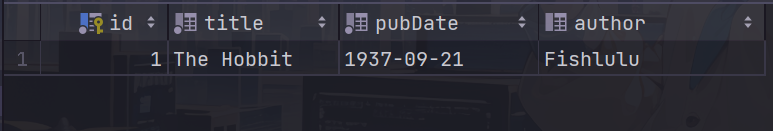
查看记录



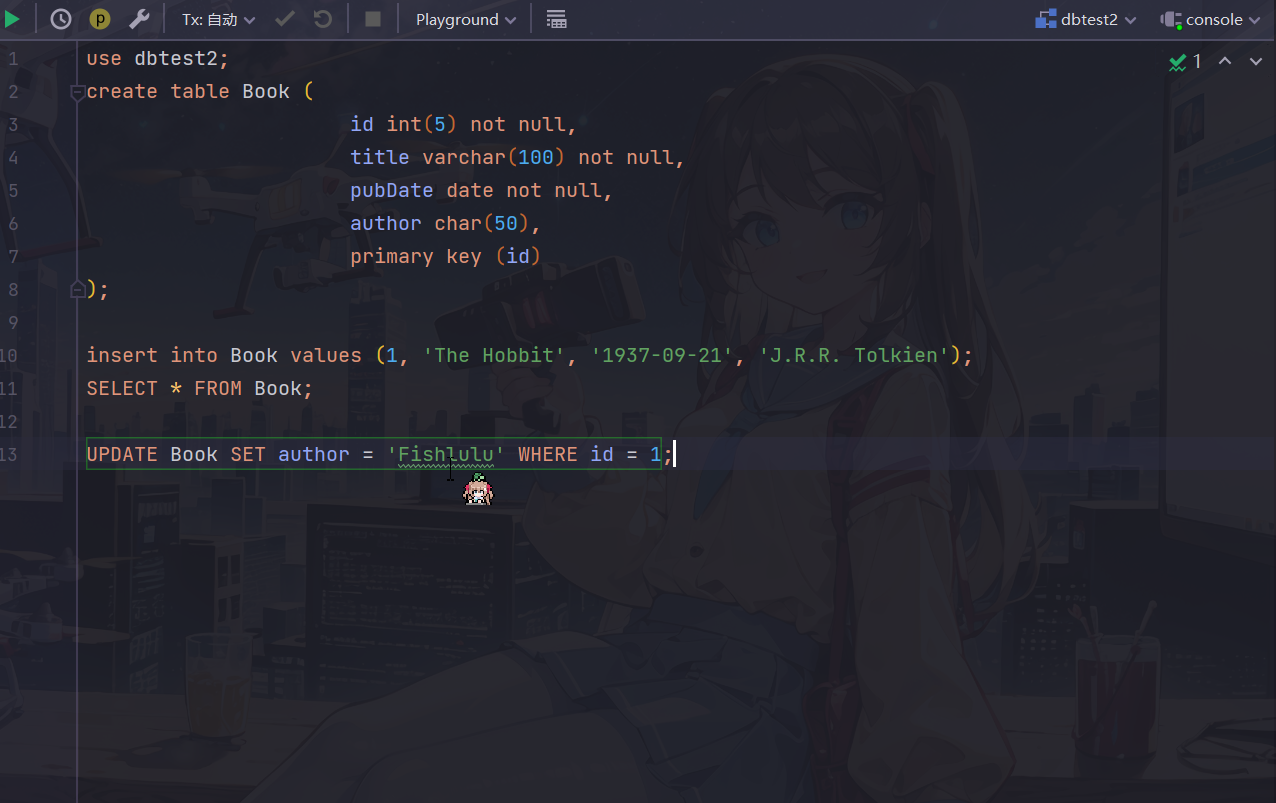
更新记录



再次查看记录



Dbm



## 调试心得

## 源程序

use dbtest2**;**create table Book (  
 id int(5) not null**,** title varchar(100) not null**,** pubDate date not null**,** author char(50)**,** primary key (id)  
)**;**insert into Book values (1**,** 'The Hobbit'**,** '1937-09-21'**,** 'J.R.R. Tolkien')**;**SELECT \* FROM Book**;**UPDATE Book SET author = 'Fishlulu' WHERE id = 1**;**

# 项目2

## 问题分析

### 目标

本项目的目标是使用Java语言和多线程编程，模拟航空公司的订票系统，使用多线程处理客户的预订请求，并使用wait/notify/notifyall机制解决预订冲突和座位可用性问题。通过本项目，可以理解多线程编程的基本概念和方法，掌握线程的创建、启动、同步、通信等技术，了解线程安全和死锁等问题的原因和解决方案

### 功能

* 航班信息：创建一个Flight类，表示共享资源，包含航班号、起飞和降落地点、座位数量等信息。
* 顾客线程:创建Customer类,多个顾客操作同一个Flight资源
* 启动类:负责运行主函数,调用各个模块

### 性能

* 多线程：使用多线程处理预订请求，提高系统的并发处理能力。
* 同步：使用synchronized关键字或者Lock接口实现线程同步，保证对共享资源的访问是互斥的，避免数据不一致的问题。
* 通信：使用wait/notify/notifyall机制实现线程间的通信，当座位数量不足时，让预订处理线程等待，当有座位被取消时，唤醒等待的预订处理线程。

## 设计方案

### 模块划分

四个模块

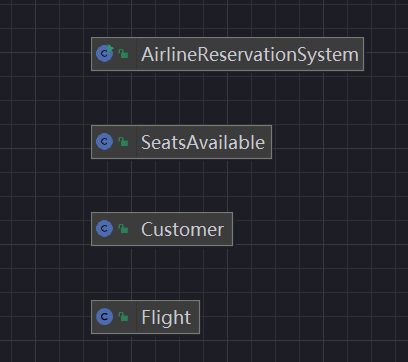
AirlineReservationSystem:启动类模块,加载线程并执行主函数

Customer:顾客线程,可以创建多个顾客来进行抢票操作

Flight:航空系统,使用了单例模式中的饿汉模式,多个顾客对同一个航空系统进行操作

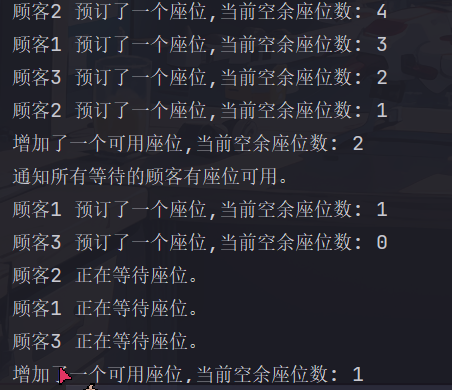
SeatsAvailable:额外开辟的定时补充座位的线程

### 类图



## 实验结果

初始航班可用座位为10,每三秒可用座位数+1,有三位顾客,每一秒执行一次订票操作,控制台输出如下



## 调试心得

在调试本项目时，我遇到了以下几类警告和错误，以及相应的处理方法：

出现了线程安全问题,这段代码出现了问题

public synchronized void bookSeat(Customer customer) throws InterruptedException {  
 if (> 0) {  
 // 座位可用，预订座位  
 System..println(customer.getName() + " 预订了一个座位。")**;** --**;** ++**;** } else {  
 // 没有可用座位，等待座位  
 System..println(customer.getName() + " 正在等待座位。")**;** wait()**;** }  
}

可以看出,是由于两个客户在同一时间均判断出availableSeats>0导致的

解决方案:

将方法synchronize属性抽离出来,用单独的类锁同步代码块包裹

线程安全：这可能是由于多个线程同时访问和修改共享资源，导致数据不一致的问题。解决方法是使用synchronized关键字或者Lock接口实现线程同步，保证对共享资源的访问是互斥的。

死锁：这可能是由于多个线程相互等待对方释放锁，导致所有线程都无法继续执行的问题。解决方法是避免循环等待条件，即按照一定的顺序获取和释放锁。

线程饥饿：这可能是由于某些线程长时间无法获取到锁，导致无法执行的问题。解决方法是使用公平锁，保证每个线程都有机会获取到锁。

通过本项目的调试，我学习了如何使用Java语言进行多线程编程，以及如何处理可能发生的线程安全、死锁和线程饥饿等问题，提高了Java语言的编程能力和问题解决能力。

## 源程序

package school.expr02.proj02**;**public class AirlineReservationSystem {  
 public static void main(String[] args) {  
 // 创建一个带有有限座位的航班,这里假设有10个座位  
 Flight flight = Flight.()**;** // 创建多个顾客线程,这里假设有3个顾客  
 Customer customer1 = new Customer("顾客1"**,** flight)**;** Customer customer2 = new Customer("顾客2"**,** flight)**;** Customer customer3 = new Customer("顾客3"**,** flight)**;** // 启动顾客线程  
 customer1.start()**;** customer2.start()**;** customer3.start()**;** //启动一个线程模拟一段时间后座位可用  
 SeatsAvailable seatsAvailable = new SeatsAvailable()**;** seatsAvailable.start()**;** }  
}

package school.expr02.proj02**;**//客户类，代表一个客户，可以预订座位。  
public class Customer extends Thread {  
 private final Flight flight**;** public Customer(String name**,** Flight flight) {  
 super(name)**;** this.flight = flight**;** }  
  
 @Override  
 public void run() {  
 try {  
 while (true) {  
 // 顾客反复尝试预订座位  
 flight.bookSeat(this)**;** // 模拟一些延迟，然后进行下一次尝试  
 Thread.(1000)**;** }  
 } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
}

package school.expr02.proj02**;**// 代表共享资源。包含一定数量的座位和已订的预订。  
// Flight为饿汉式单例模式  
public class Flight {  
 private static final Flight = new Flight()**;** private static int **;** private static int **;** private static final Object = new Object()**;** public static final int = 10**;** private Flight() {  
 // 在构造方法中初始化座位和预订数  
 = **;** // 为示例设置初始座位数为10  
 = 0**;** }  
  
 // 获取Flight实例的方法  
 public static Flight getInstance() {  
 return **;** }  
  
 public void bookSeat(Customer customer) throws InterruptedException {  
 synchronized () {  
 if (> 0) {  
 // 座位可用，预订座位  
 System..println(customer.getName() + " 预订了一个座位,当前空余座位数: " + --)**;** ++**;** } else {  
 // 没有可用座位，等待座位  
 System..println(customer.getName() + " 正在等待座位。")**;** .wait()**;** }  
 }  
 }  
  
 // 该方法让空余座位数加1  
 public void increaseSeats() {  
 synchronized () {  
 // 增加座位  
 System..println("增加了一个可用座位,当前空余座位数: " + (++))**;** // 通知所有等待的顾客有座位可用  
 System..println("通知所有等待的顾客有座位可用。")**;** .notifyAll()**;** }  
 }  
}

package school.expr02.proj02**;**//模拟一段时间后座位可用  
public class SeatsAvailable extends Thread{  
 //每3秒调用一次Flight的increaseSeats方法  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 Thread.(3000)**;** Flight.().increaseSeats()**;** } catch (InterruptedException e) {  
 e.printStackTrace()**;** }  
 }  
 }  
}

# 项目3

## 问题分析

### 目标

本项目的目标是使用Java语言和Socket API，实现客户端和服务器之间的通信，客户端向服务器查询某个城市的天气信息，服务器根据城市名称返回天气数据。通过本项目，可以理解网络编程的基本概念和方法，掌握Socket的创建、连接、读写、关闭等技术，了解TCP/IP协议的工作原理和应用。

### 功能

* 服务器：创建一个Server类，表示服务器，接收客户端的连接请求，根据客户端发送的城市名称，查询天气数据，并将天气数据发送给客户端。
* 客户端：创建一个Client类，表示客户端，向服务器发送连接请求，输入城市名称，接收服务器返回的天气数据，并显示在控制台上。

### 性能

* 网络连接：使用TCP协议进行网络连接，保证数据的可靠传输。
* 数据传输：使用InputStream和OutputStream进行数据的读写，保证数据的正确性。
* 异常处理：使用try-catch-finally语句处理可能发生的异常，如网络连接失败、数据读写错误、输入不合法等，保证程序的健壮性

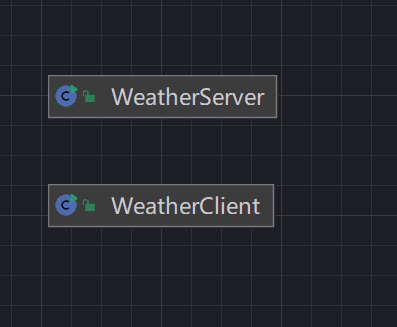
## 设计方案

### 模块划分

客户端 : 发送天气查询请求

服务端 : 接受并处理客户端的请求,返回天气数据

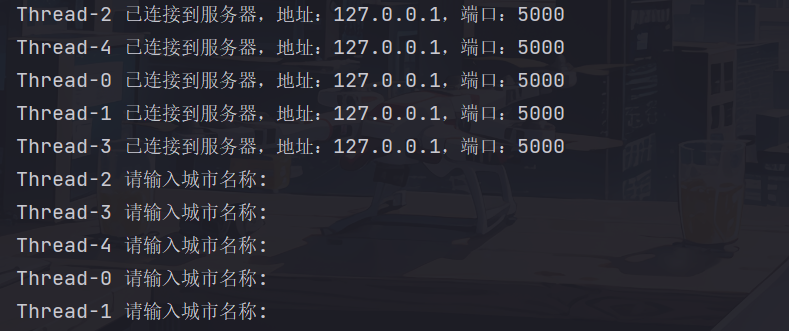
### 类图

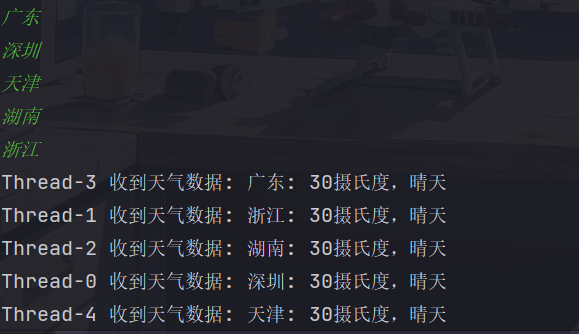


## 实验结果

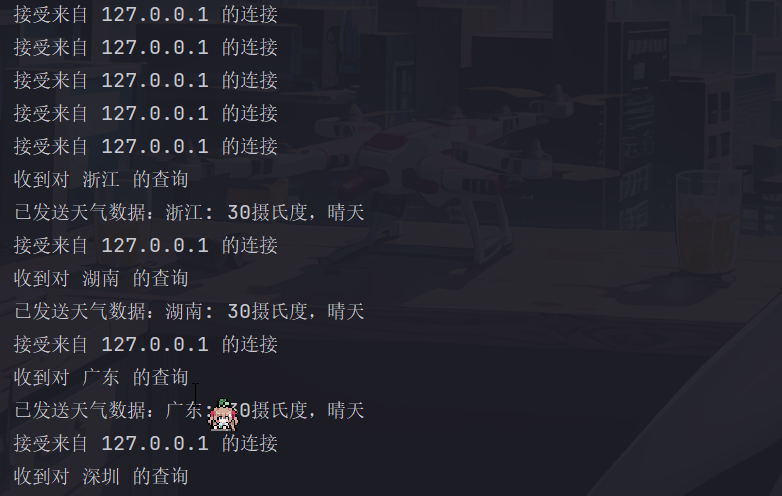
多个客户端连接服务端

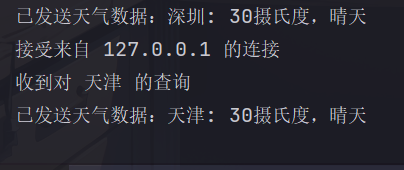
全部接受数据后可以批量发出





服务端返回的数据:





## 调试心得

* 网络连接失败：这可能是由于服务器的地址、端口等参数设置错误，或者网络连接异常等原因导致的。解决方法是检查网络的配置信息，确保网络连接正常。
* 数据读写错误：这可能是由于数据的格式不正确，或者数据的长度超过了缓冲区的大小，或者网络连接中断等原因导致的。解决方法是检查数据的格式和长度，以及网络连接的稳定性。
* 输入不合法：这可能是由于用户输入的城市名称不符合预期的格式或范围等原因导致的。解决方法是在程序中对用户的输入进行合法性检查，如果不合法，提示用户重新输入，或者抛出异常。

通过本项目的调试，我学习了如何使用Java语言进行网络编程，以及如何处理可能发生的网络连接失败、数据读写错误和输入不合法等问题，提高了Java语言的编程能力和问题解决能力。

## 源程序

客户端

package school.expr02.proj03**;**import java.io.IOException**;**import java.io.InputStream**;**import java.io.OutputStream**;**import java.net.Socket**;**import java.util.Scanner**;**public class WeatherClient {  
 public static void main(String[] args) {  
 try {  
 // 创建多个客户端并启动  
 for (int i = 0**;** i < 5**;** i++) {  
 Thread clientThread = new Thread(new ClientTask())**;** clientThread.start()**;** }  
 } catch (Exception e) {  
 e.printStackTrace()**;** }  
 }  
  
 // 客户端任务  
 static class ClientTask implements Runnable {  
 @Override  
 public void run() {  
 try {  
 Socket socket = new Socket(WeatherServer.**,** WeatherServer.)**;** System..println(Thread.().getName() + " 已连接到服务器，地址：" + WeatherServer.+ "，端口：" + WeatherServer.)**;** Scanner scanner = new Scanner(System.)**;** System..print(Thread.().getName() + " 请输入城市名称: \n")**;** String city = scanner.nextLine()**;** queryWeather(socket**,** city)**;** } catch (IOException e) {  
 e.printStackTrace()**;** }  
 }  
  
 private void queryWeather(Socket socket**,** String city) {  
 try {  
 InputStream input = socket.getInputStream()**;** OutputStream output = socket.getOutputStream()**;** output.write(city.getBytes())**;** byte[] buffer = new byte[1024]**;** int bytesRead = input.read(buffer)**;** String weatherData = new String(buffer**,** 0**,** bytesRead)**;** System..println(Thread.().getName() + " 收到天气数据: " + weatherData)**;** socket.close()**;** } catch (IOException e) {  
 e.printStackTrace()**;** }  
 }  
 }  
}

服务端

package school.expr02.proj03**;**import java.io.IOException**;**import java.io.InputStream**;**import java.io.OutputStream**;**import java.net.ServerSocket**;**import java.net.Socket**;**public class WeatherServer {  
 public static final String = "127.0.0.1"**;** public static final int = 5000**;** public static void main(String[] args) {  
 try {  
 ServerSocket serverSocket = new ServerSocket()**;** System..println("服务器正在监听端口 " + )**;** while (true) {  
 Socket clientSocket = serverSocket.accept()**;** System..println("接受来自 " + clientSocket.getInetAddress().getHostAddress() + " 的连接")**;** (clientSocket)**;** }  
 } catch (IOException e) {  
 e.printStackTrace()**;** }  
 }  
  
 private static void handleClient(Socket clientSocket) {  
 try {  
 InputStream input = clientSocket.getInputStream()**;** OutputStream output = clientSocket.getOutputStream()**;** byte[] buffer = new byte[1024]**;** int bytesRead = input.read(buffer)**;** String city = new String(buffer**,** 0**,** bytesRead)**;** System..println("收到对 " + city + " 的查询")**;** // 占位符：实际上应该根据城市名称查询天气数据  
 String weatherData = city + ": 30摄氏度，晴天"**;** output.write(weatherData.getBytes())**;** System..println("已发送天气数据：" + weatherData)**;** clientSocket.close()**;** } catch (IOException e) {  
 e.printStackTrace()**;** }  
 }  
}