面向对象与多线程综合实验

结合华为云平台的基于JAVASE数据挖掘系统设计与实现

主要内容

一
- 实验目标和要求

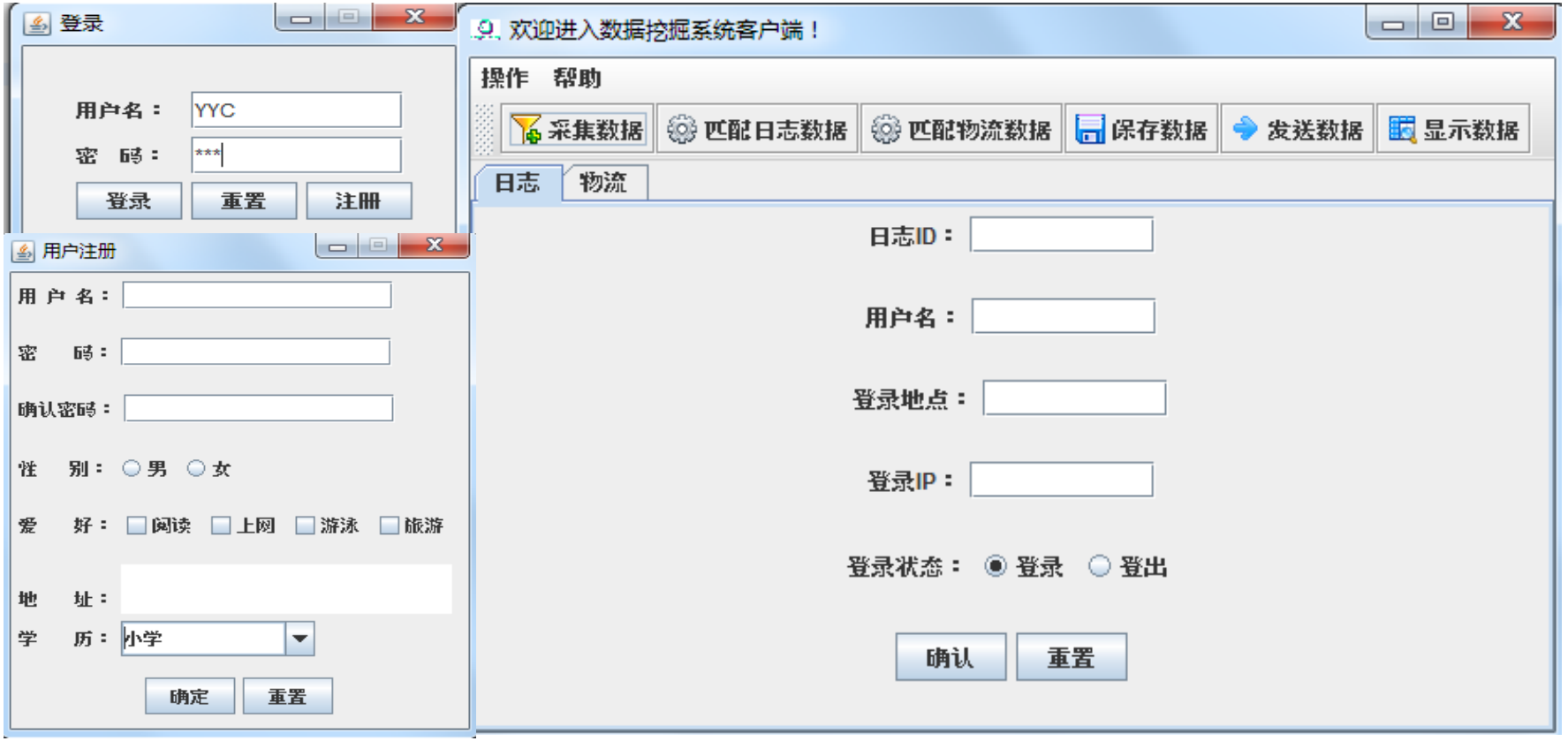
二
- 实验内容

三
- 知识要点

系统简介

- 基于Java SE 的数据挖掘系统

基于客户端服务器端（Client-Server，C-S）模式，实现日志与物流数据信息的采集、匹配、保存、显示等功能，为数据分析挖掘提供基础支撑。



需求分析

- 系统包括客户端应用程序、服务器端应用程序---C/S模式
- 用户和数据信息的保存---JDBC数据库保存和查询
- 用户能够进行注册和登录，授权后使用系统---GUI登录和注册界面设计与功能实现
- 能够实现日志和物流信息的数据采集（录入），登录登出对匹配、信息保存和数据显示等功能---GUI主界面设计与功能实现
- 系统能够进行数据自动刷新功能，与数据库保持同步---线程
- 客户端与服务器端交互，客户端能够将数据发送到服务器端，服务器端接收客户端发送的日志和物流信息，进行保存和处理---Socket通信
- 系统优化---JAVA高级应用与新特性

任务分配

课时分配	实验任务	任务列表	主要知识点分解
第1-2次课	基于控制台的系统数据**采集、匹配、显示**和**记录**功能实现	✓ 注册华为软开云账户 ✓ 搭建数据挖掘系统**框架** ✓ 实现日志和物流数据信息的**采集**、**匹配**和**显示**功能 ✓ 实现匹配的物流数据信息的文件保存和读取**记录**功能	继承与多态/异常处理/集合/文件存储及IO流/华为软开云
第3次课	基于**JDBC**的控制台系统基本功能实现	✓ 创建项目所需的数据库表，并搭建数据访问基础环境 ✓ 实现并测试匹配的日志、物流信息的数据库保存和查询功能	JAVAJDBC/MySQL
第4次课	基于SwingGUI的系统**注册**和**登录界面**设计实现	✓ 创建用户数据库表、用户实体类和用户业务逻辑类 ✓ 创建用户注册窗口，并将用户注册信息保存到数据库 ✓ 创建用户登录窗口，登录成功则进入系统主界面	SwingGUI/事件驱动/WinBuilder
第5次课	基于SwingGUI系统**主界面**设计实现和系统**优化**	✓ 实现主界面中的菜单和工具栏 ✓ 实现主界面中的日志和物流**数据采集**、**匹配**、**保存**和**显示**功能 ✓ GUI系统优化	高级UI组件
第6次课	系统信息**自动刷新**功能实现	✓ 使用线程实现**一定时间**（如**每隔1分钟**）自动刷新日志和物流显示数据表格，即从数据库中获取并显示，与数据库保持同步	线程
第7次课	系统客户端/服务器端数据**发送(交互)**功能实现	✓ 客户端应用程序：修改主界面**发送数据**页面响应，即使用Socket实现数据由客户端发送到服务器 ✓ 服务器端应用程序：使用Server Socket实现接收客户端发送的日志和物流数据信息，并将信息保存到数据库	Socket网络编程
第8次课	系统验收	✓ 数据挖掘系统的基本功能实现与演示 ✓ **在华为软开云平台完成系统需求分析和设计反推**	PPT演示、系统运行
拓展	Java 高级应用以及Java8新特性	◆ 使用注解重新迭代升级系统代码 ◆ 使用格式化将输出的日期进行格式化输出 ◆ 使用Lambda表达式迭代升级主窗口中“帮助”菜单的事件处理 ◆ 使用Lambda表达式实现查找指定的匹配信息并显示	增加注解和格式化以及Lambda优化和查询

功能要求

系统基于C/S模式，包括客户端和服务器端应用程序

- 1. 用户登录和注册功能:用户验证口令通过后登录系统，新用户进行注册，并将注册信息保存数据库
- 2. 日志、物流数据的采集功能：对日志和物流数据进行采集，并保存到Mysql数据库；
- 3. 日志、物流数据的筛选匹配功能，以日志信息为例：
  - 根据日志的登录、登出状态，对日志进行分类，分别存放到**登录**日志集合（ArrayList<LogRec> logIns）和**登出**日志集合（ArrayList<LogRec> logOuts）中。
  - 在登录日志和登出日志中，根据**用户名**和IP地址进行匹配：如果存在相同的用户名和IP地址，则日志信息匹配成功，将匹配的日志数据封装到MatchedLogRec对象，并保存到匹配日志集合（ArrayList<MatchedLogRec> matchLogs）中。
- 4. 日志、物流数据的数据保存功能：在系统主界面中点击“保存数据”按钮时，将匹配的日志数据和物流数据保存到本地文件和数据库中。
- 5. 客户端服务器端交互功能：
  - ✓ 客户端的数据发送功能：在客户端通过Socket技术向服务器端发送匹配的日志数据和物流数据；当数据发送成功后，清空客户端暂时存放数据的集合，然后弹出信息提醒；
  - ✓ 服务器数据查询功能：当点击客户端显示数据功能时，服务器端从数据库中查找符合条件的数据，并发送到客户端；在客户端以表格的形式将数据显示。

实验目标和要求

实验目标

1. 理解与掌握知识点：
   - 理解线程的概念
   - 掌握JAVA多线程的创建和线程生命周期
   - 了解线程的同步和优先级
2. 熟悉工具与平台：Windowbuilder插件完成GUI界面开发
3. 开发程序：实现系统GUI图形界面的数据自动刷新功能

实验要求与交付

要求：

完成数据挖掘系统的数据自动刷新功能：

即使用线程实现每隔1分钟，或指定时间内，刷新显示数据的日志和物流表格数据，以便与数据库中的数据保持一致。

✱交付：

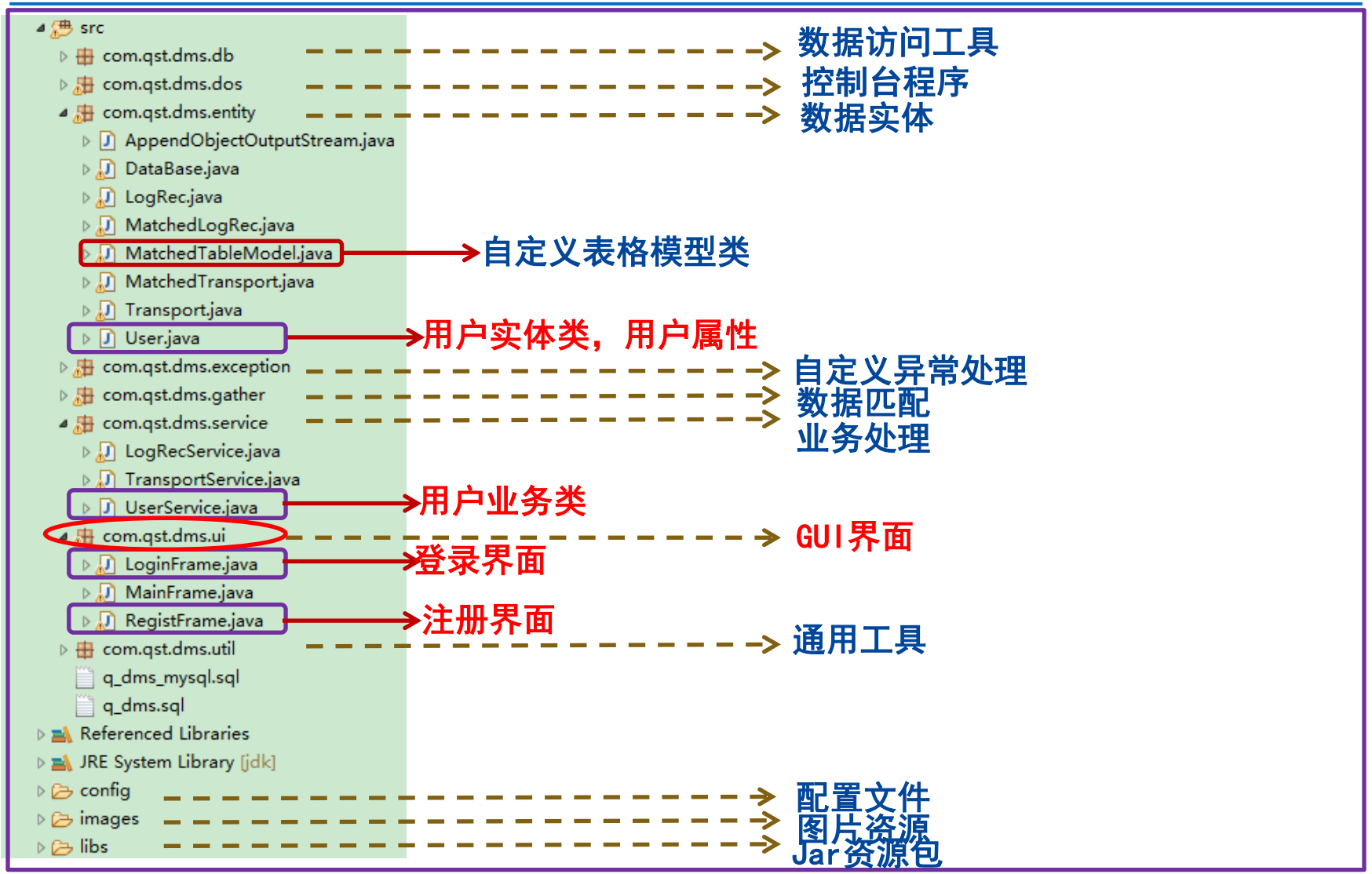
1. 代码与运行结果
2. 开发过程中遇到问题截图和解决办法

本次实验内容

实验内容

1. 做什么	2. 怎么做？	3. 对应项
◆ 每隔1分钟从数据库调取匹配后的日志和物流数据 ◆ 显示数据中的表格信息与数据库保持同步	◆ 主界面新增线程内部类，并调用刷新日志和物流数据信息表格方法，线程阻塞一分钟 ◆ 主界面中开启该线程类	◆ 线程的创建，启动 ◆ MainFrame.java及其中的UpdateTableThread内部类

实验内容系统框架


src
com.qst.dms.db
com.qst.dms.dos
com.qst.dms.entity
AppendObjectOutputStream.java
DataBase.java
LogRec.java
MatchedLogRec.java
MatchedTableModel.java
MatchedTransport.java
Transport.java
User.java
com.qst.dms.exception
com.qst.dms.gather
com.qst.dms.service
LogRecService.java
TransportService.java
UserService.java
com.qst.dms.ui
LoginFrame.java
MainFrame.java
RegistFrame.java
com.qst.dms.util
q_dms_mysql.sql
q_dms.sql
Referenced Libraries
JRE System Library [jdk]
config
images
libs
数据访问工具
控制台程序
数据实体
自定义表格模型类
用户实体类，用户属性
自定义异常处理
数据匹配
业务处理
用户业务类
GUI界面
登录界面
注册界面
通用工具
配置文件
图片资源
Jar资源包

主界面中新增线程类

MainFrame.java 中的内部类 UpdateTableThread.java

```
// 线程类，每隔1分钟刷新一次显示数据表格中的数据,与数据库保持同步
private class UpdateTableThread extends Thread {
    // 重写run()方法
    public void run() {
        while (true) {
            // 移除所有的选项卡
            showPane.removeAll();
            // 刷新日志信息
            flushMatchedLogTable();
            // 刷新物流信息
            flushMatchedTransTable();
            try {
                // 线程挂起1分钟
                Thread.sleep(1*60*1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
// 刷新日志选项卡，显示日志信息表格
private void flushMatchedLogTable() {
    // 创建tableModel，通过标志为区分日志和物流：1，日志；0，物流
    MatchedTableModel logModel = new MatchedTableModel(
            logRecService.readLogResult(), 1);
    // 使用tableModel创建JTable
    JTable logTable = new JTable(logModel);
    // 通过JTable对象创建JScrollPane，显示数据
    scrollPane = new JScrollPane(logTable);
    // 添加日志选项卡
    showPane.addTab("日志", scrollPane);
}
```

主窗口中开启线程类

在主界面窗口MainFrame.java中开启更新表格数据的线程

```
        //开启更新表格数据的线程
        new UpdateTableThread().start();
    }
```

*思考： 在MainFrame类中的哪里开启该线程

JTable

- JTable(TableModel dm) ：构造一个 JTable，使用数据模型 dm、默认的列模型和默认的选择模型对其进行初始化
- 在这个构造函数中，需要传递一个数据模型TableModel 用来存放数据，当表格显示时就直接通过这个TableModel来获取表格的信息以及数据
- TableMode是一个**接口**，需要实现**AbstractTableModel**的方法，而AbstractTableModel又是一个**抽象的类**，也就是说在使用自己的TableModel的时候要重写一个自己的TableMode类，通过这个Model来控制自己表格的数据，AbstractTableModel中有些方法是已经实现的了，所以我们只需要对自己需要的方法进行重写

JTable

entity 包中的自定义的MatchedTableModel类，继承AbstractTableModel类

学会使用JTable(TableModel dm)

```java
package com.qst.dms.entity;

import java.sql.ResultSet;

public class MatchedTableModel extends AbstractTableModel {

    // 使用ResultSet来创建TableModel
    private ResultSet rs;
    private ResultSetMetaData rsmd;
    // 标志位，区分日志和物流：1，日志；0，物流
    private int sign;

    public MatchedTableModel(ResultSet rs, int sign) {
        this.rs = rs;
        this.sign = sign;
        try {
            rsmd = rs.getMetaData();
        } catch (Exception e) {
            rsmd = null;
        }
    }

    // 获取表格的行数
    public int getRowCount() {
        try {
            rs.last();
            // System.out.println(count);
            return rs.getRow();
        } catch (Exception e) {
            return 0;
        }
    }

    // 获取表格的列数
    public int getColumnCount() {
        try {
            // System.out.println(rsmd.getColumnCount());
```

```java
// 获取表格的列数
public int getColumnCount() {
    try {
        // System.out.println(rsmd.getColumnCount());
        return rsmd.getColumnCount();
    } catch (Exception e) {
        return 0;
    }
}

// 获取指定位置的值
public Object getValueAt(int rowIndex, int columnIndex) {
    try {
        rs.absolute(rowIndex + 1);
        return rs.getObject(columnIndex + 1);
    } catch (Exception e) {
        return null;
    }
}

// 获取表头信息
public String getColumnName(int column) {
    String[] logArray = { "用户名", "登录时间", "登出时间","登录地点" };
    String[] tranArray = { "物流ID", "采集时间", "目的地", "状态", "经手人", "收货人",
            "物流类型" };
    return sign == 1 ? logArray[column] : tranArray[column];
}
```

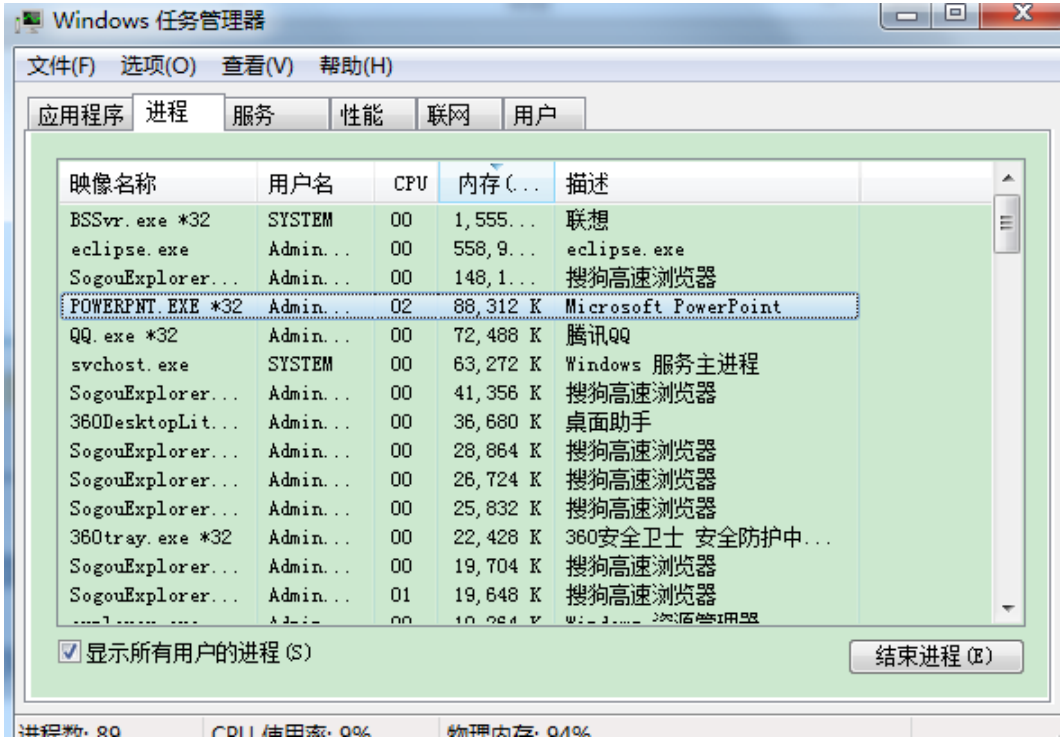
知识点

- 线程的概述
- JAVA线程模型（创建方法）
- 线程的生命周期
- 线程的优先级
- 线程同步

线程概述

进程

- 在操作系统中，每个独立运行的程序就是一个进程（Process），操作系统为每个加载到内存执行的程序创建一个进程，进程在执行过程中拥有独立的内存单元
- 进程是操作系统进行资源分配和调度的一个独立单位



Windows进程管理

线程概述

线程（Thread）在多任务处理应用程序中起着至关重要的作用

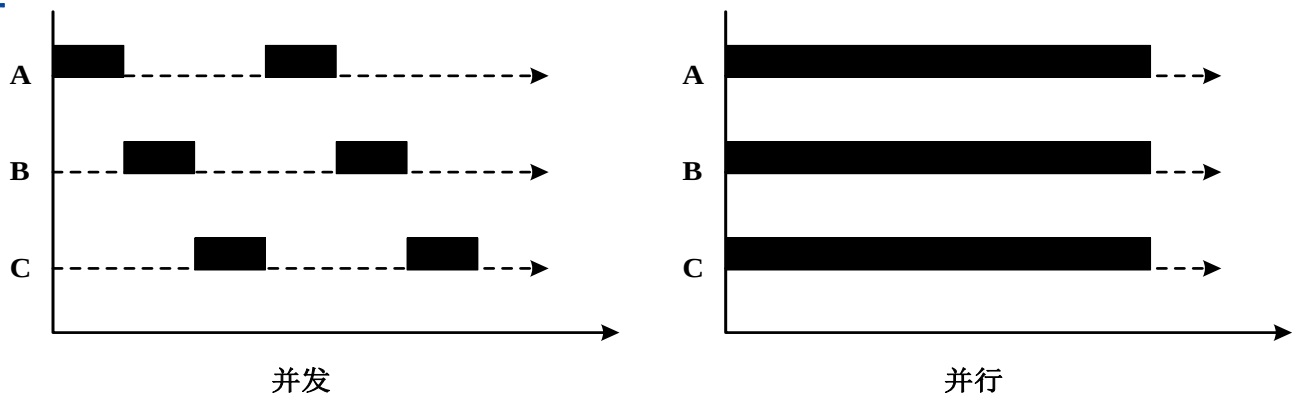
- 线程，是指进程中的一段独立的执行代码（指令序列流）。一个进程中有多个这样独立执行的程序段，该进程的执行程序就是多线程的程序
- 一个线程运行一个算法，一个程序可以划分成多个算法。将算法分散交由不同的线程去执行，实现多线程并发。程序员负责创建线程，并为线程指定要运行的算法
- 多个线程共享内存，各个线程共享资源

  进程是资源分配和调度的最小单位，线程是CPU调度和分派的基本单位，是进程的一个实体。多线程不是为了提高程序运行速度，而是提高应用程序使用率

线程概述

线程与进程

- 线程是进程中的组成部分，一个进程可以包含多个线程，多线程可以在一个程序中同时完成多个任务
- 同一进程中的多个线程之间虽然各自独立运行，但需要共享数据，才能实现多线程协同工作
- 对于一个CPU而言，在某一时间点只能执行一个进程，CPU会不断在多个进程之间来回轮换执行，CPU分时技术
- 单核CPU：分时技术并发执行各个线程；多核CPU：在不同的运行核或CPU上运行线程，并行执行



Java线程

Java主线程

- 每个进程至少包含一个线程，即主线程，其有两个特点：

  1. 一个进程肯定包含一个主线程
  2. 主线程用来执行main()方法

- Java程序启动Java虚拟机JVM，等于启动了一个应用程序，表示启动了一个进程。该进程会自动启动一个“主线程”，主线程会调用某个类的main方法

- 在main()方法中调用Thread类的静态方法currentThread()来获取主线程。

Java虚拟机的主线程入口是Main方法，用户自己创建新线程？⟹类和接口

Java线程

Java线程类

- Java线程有关的类和接口：

1. Thread类
2. Runnable接口
3. Callable接口
4. Future接口等

Java线程

Thread类:Java.lang包中一个专门创建线程和对线程进行操作的类

- Java的线程是通过Java软件包java. lang中定义的类Thread来实现的。当生成一个Thread类的对象，就产生一个线程，通过该对象实例，可以启动线程，终止线程等
- Java在Thread中定义了方法，将线程所必需的功能进行封装，用来操作和处理线程
- 其常用的方法如下：
  - ① 构造方法
  - ② run方法
  - ③ 改变线程状态方法
  - ④ 其他方法

java.lang包是Java类库中常用的包，Java程序会自动导入lang包，因此java.lang包中提供的类或接口可以直接使用，而无需import导入

Java线程

Thread类:Thread 线程类用于创建线程对象

方法	功能描述
Thread()	不带参数的构造方法，用于构造缺省的线程对象
Thread(Runnable target)	使用传递的Runnable构造线程对象
Thread(Runnable target,String name)	使用传递的Runnable构造名为name的线程对象
Thread(ThreadGroup group,Runnable target,String name)	使用传递的Runnable在group线程组内构造名为name的线程对象
final String getName()	获取线程的名称
final boolean isAlive()	判断线程是否处于激活状态，如果是，则返回true，否则返回false
final void setName(String name)	设置线程的名称为指定的name名
long getId()	获取线程

Java线程

线程执行的代码在线程的run方法中定义，run方法称为线程体。实现线程体的特定对象是在初始化线程时传递给线程的

方法	功能描述
setPriority(int newPriority)	设置线程的优先级
getPriority()	获取线程的优先级
final void join()	等待线程死亡
static void sleep(long millis)	线程休眠，即将线程挂起一段时间，参数以毫秒为基本单位
void run()	线程的执行方法
void start()	启动线程的方法，启动线程后会自动执行run( )方法
void stop()	线程停止，该方法已过期，可以使用但不推荐用
void interrput()	中断线程
static int activeCount()	返回激活的线程数
static void yield()	临时暂停正在执行的线程，并允许执行其他线程

Java线程

Runnable接口:

- 用于标识某个Java类可否作为线程类，该接口只有一个抽象方法run()

```
package java.lang;
public interface Runnable {
        public abstract void run();
}
```

Java线程

Java创建多线程

- 在Java中创建线程有两种方法：

① 继承Thread类

② 实现Runnable接口

在使用Runnable接口时需要建立一个Thread实例。因此，无论是通过Thread类还是Runnable接口建立线程，都必须建立Thread类或它的子类的实例

总的原则：将算法对象放入线程对象，启动线程，并发执行算法

Java线程

通过继承Thread类创建并启动线程

- 步骤：

① 定义一个子类继承Thread类，并重写run()方法

② 创建子类的实例

③ 调用线程对象的start()方法启动该线程

Java线程

- 方法一：继承Thread类覆盖run方法

```
public class ThreadDemo1 {
    public static void main(String[] args){
        Demo d = new Demo();
        d.start();
        for(int i=0;i<60;i++){
            System.out.println(Thread.currentThread().getName()+i);
        }
    }
}
```

Java线程

- 方法一：继承Thread类覆盖run方法。

```
public class Demo extends Thread{
    public void run(){
        for(int i=0;i<60;i++){
            System.out.println(this.getName()+" :" +i);
        }
    }
}
```

Java线程

通过Runnable接口创建并启动线程

- 步骤：

① 定义一个类实现Runnable接口；

② 创建一个Thread类的实例，将Runnable接口的实现类所创建的对象作为参数传入Thread类的构造方法中；

③ 调用Thread对象的start()方法启动该线程

Java线程

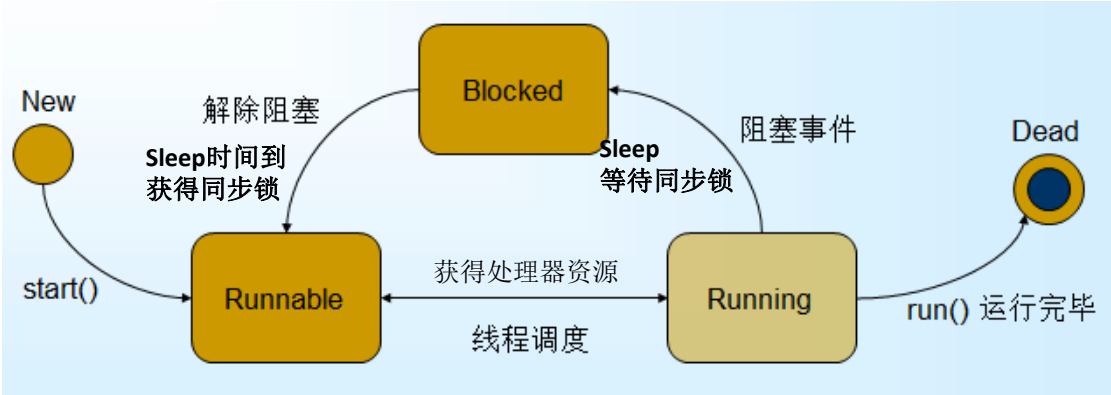
方法二：通过Runnable接口

```
public class ThreadDemo2 {
    public static void main(String[] args){
        Demo2 d =new Demo2();
        Thread t = new Thread(d);
        t.start();
        for(int x=0;x<60;x++){
            System.out.println(Thread.currentThread().getName()+x);
        }
    }
}
class Demo2 implements Runnable{
    public void run(){
        for(int x=0;x<60;x++){
            System.out.println(Thread.currentThread().getName()+x);
        }
    }
}
```

Java线程生命周期

线程的生命周期

Java线程也同样要经历新生New、就绪Runnable、运行Running、阻塞Blocked和终止Dead等5种不同状态。这些状态都可以通过Thread类中的方法进行控制。



方法	说明
start()	线程方法，新建的线程进入Runnable状态
wait()	对象方法，释放对象锁，线程进入blocked状态，等待被notify或时间到期
notify()/notifyAll()	对象方法，唤醒其他的线程
yield()	线程静态方法，放弃执行，回到Runnable状态，使其他优先级不低于此线程的线程有机会先运行
sleep()	线程静态方法，线程睡眠指定的一段时间，线程进入blocked状态
join()	线程方法，调用这个方法的线程，会等待加入的子线程完成
isAlive()	判断某一个线程是否还在运行run。Run方法结束返回false

Java线程生命周期

线程New和Runnable

- 当使用new关键字创建一个线程之后，该线程就处于新建状态
- 当线程对象调用start()方法之后，线程就处于就绪状态
- new完一个线程后，只能调用一次start()方法

如果调用start()方法后需要线程立即开始执行，可以使用Thread.sleep(1)来让当前运行的主线程休眠1毫秒，此时处于空闲状态的CPU会去执行处于就绪状态的线程，这样就可以让子线程立即开始执行。

Java线程生命周期

线程运行和阻塞状态

- 处于就绪状态的线程获得CPU后，开始执行run()方法，出现以下情况时线程会进入阻塞状态：

① 调用sleep()方法，主动放弃所占用的处理器资源；

② 调用了一个阻塞式IO方法，在该方法返回之前，该线程被阻塞；

③ 试图获得一个同步监视器，但该同步监视器正被其他线程所持有；

④ 执行条件还未满足，调用wait()方法使线程进入等待状态，等待其他线程的通知（notify）；

⑤ 程序调用线程的suspend()方法将该线程挂起

被阻塞的线程阻塞解除后，会进入就绪状态而不是运行状态，必须重新等待**线程调度器**再次调度。

Java线程生命周期

线程运行和阻塞状态

使用sleep()方法时需要注意

- sleep()方法的参数是以毫秒为基本单位
- sleep()方法声明了InterruptedException异常

```
try {
        Thread.sleep(1000);
} catch (InterruptedException e) {
        e.printStackTrace();
}
```

```
void myMethodName() throws
InterruptedException{
        Thread.sleep(1000);
        ...
}
```

如果一个线程包含了很长的循环，在循环的每次迭代之后把该线程切换到sleep休眠状态是一种很好的策略，这可以保证其他线程不必等待很长时间才能轮到处理器执行

Java线程生命周期

线程运行和阻塞状态

- 通过Thread类的isAlive()方法判断线程是否处于运行状态

① 当线程处于就绪、运行和阻塞三种状态时，其返回值为true

② 当线程处于新建、死亡两种状态时，其返回值为false

Java线程生命周期

死亡状态

- 线程结束后就处于死亡状态，结束线程有三种方式：

① 线程执行完成run()或call()方法
② 线程抛出一个未捕获的Exception或Error
③ 调用stop()方法直接停止线程

- 测试某个线程是否死亡，可通过线程对象的isAlive()方法来获得线程状态
- Thread类中join()方法可让一个线程等待另一个线程完成后继续执行原线程中的任务
- 两种中断子线程的方法

① 通过线程类Thread的方法成员interrupt()从外部中止运行
② 将子线程设为守护线程，守护线程在进程结束时会随之停止

主线程结束时，其他子线程不受任何影响，并不会随主线程的结束而结束。一旦子线程启动起来，子线程就拥有和主线程相同的地位，子线程不会受主线程的影响。

Java线程优先级

线程的优先级代表该线程的重要程度，优先级高的线程获得CPU时间的机会更多。

- Thread类提供三个静态常量来标识线程的优先级。

① MAX_PRIORITY：最高优先级，其值为10

② NORM_PRIORITY：普通优先级，其值为5

③ MIN_PRIORITY：最低优先级，其值为1

- Thread类提供了setPriority()方法来对线程的优先级进行设置，而getPriority()方法来获取线程的优先级

线程的优先级高度依赖于操作系统，并不是所有的操作系统都支持Java的10个优先级别，例如Windows 2000仅提供7个优先级别。因此，尽量避免直接使用整数给线程指定优先级，提倡使用MAX_PRIORITY、NORM_PRIORITY和MIN_PRIORITY三个优先级静态常量。另外，优先级并不能保证线程的执行次序，因此应避免使用线程优先级作为构建任务执行顺序的标准。

线程同步

- 多线程之间的互斥操作
- 多线程并发程序包含多个线程，每个线程运行一个算法，执行时各线程轮流切换，并发执行算法
- 如果两个线程中的算法不能重叠交叉执行，则这两个算法被称为是互斥操作

例如：多个线程共享一个对象，一个线程在更新该对象的同时，另一个线程也要更新或读取该对象，则在不同线程中同时访问共享数据就可能是互斥操作，该操作将破坏数据一致性

线程同步

- 多个线程同时访问一个共享对象，引发互斥操作，破坏数据一致性
- 引入线程同步：某时刻只允许一个线程独占性访问共享对象，其他线程只能处于阻塞状态，只有该线程访问操作结束后，才允许其他线程访问，这称为线程同步或叫互斥
- Java使用关键字Synchronized关键字控制对共享信息的并发访问，实现线程同步
- 实质是使用monitor监视器实现对共享数据操作的同步。共享对象都有一个监视器(对象锁)，监控器一次只允许一个线程执行对象的同步块
- 程序进入同步块，就将对象锁住，获得同步锁，当完成同步块，监控器打开该对象的锁，释放锁

线程同步

- 线程同步保证了某个资源在某一时刻只能由一个线程访问
- 线程同步通常采用三种方式：

① 同步代码块

② 同步方法

③ 同步锁

线程同步

同步代码块

- 实现同步功能，只需将对实例的访问语句放入一个同步块中，其语法：

```
synchronized(object){
        // 需要同步的代码块
}
```

synchronized是同步关键字

object是同步监视器

任何时刻只能有一个线程可以获得对同步监视器的锁定；当同步代码块执行完成后，该线程会释放对该同步监视器的锁定。

线程同步

同步方法

- 使用synchronized关键字修饰需要同步的方法，其语法：

```
[访问修饰符] synchronized 返回类型 方法名([参数列表]) {
        // 方法体
}
```

synchronized锁定的是对象，而不是方法或代码块；synchronized也可以修饰类，当用synchronized修饰类时，表示这个类的所有方法都是synchronized的。

线程同步

同步锁

- 同步锁Lock通过显式定义同步锁对象来实现线程同步（JDK1.5提供）
- ReentrantLock类是常用的可重入同步锁，可以显式地加锁、释放锁，使用步骤如下：

① 定义一个ReentrantLock锁对象，该对象是final常量

② 在需要保证线程安全的代码之前增加“加锁”操作

```
private final ReentrantLock lock = new ReentrantLock();
```

① 在执行完线程安全的代码后“释放锁”

```
lock.unlock();
```

线程通讯

Java中提供了一些机制保证线程之间的协调运行，即线程通信

- 线程通信可使用Object类中定义的三个方法：

① wait()方法

② notify()方法

③ notifyAll()方法

notify()方法和notifyAll()方法只能在同步方法或同步块中使用。wait()方法区别于sleep()方法的是：wait()方法调用时会释放对象锁，而sleep()方法不会。

线程通讯

Timer和Swing Timer

- Java提供了Timer和Swing Timer控件，用于执行规划好的任务或循环任务
- 以按钮在窗口中移动为例，可以分别采用Thread、Timer和Swing Timer来实现
- 使用java.util.Timer具体步骤：

① 定义一个类继承TimerTask；
② 创建Timer对象；
③ 调用Timer对象的schedule()方法安排任务；
④ 调用Timer对象的cancel()方法，取消一个规划好的任务。

notify()方法和notifyAll()方法只能在同步方法或同步块中使用。wait()方法区别于sleep()方法的是：wait()方法调用时会释放对象锁，而sleep()方法不会。

线程通讯

Timer和Swing Timer

- Timer类的schedule()方法几种重载方式：

① schedule(TimerTask task, Date time)

② schedule(TimerTask task, Date firstTime, long period)

③ schedule(TimerTask task, long delay, long period)

- 使用javax.swing.Timer类具体步骤：

① 定义一个监听类，实现ActionListener监听接口，并重写actionPerformed()方法；

② 创建javax.swing.Timer对象；

③ 调用start()方法启动Swing Timer；

④ 调用stop()方法停止Swing Timer。

线程通讯

Swing Timer

- javax.swing.Timer只能在Swing应用程序中使用
- 其构造方法：

```
Timer(int  delay, ActionListener  liatener )
```

参数delay规定从调用start()方法开始到第一次执行该任务时的时间间隔

参数listener指定监听对象

总结

- 进程（Process）在执行过程中拥有独立的内存单元，是资源分配和调度的最小单位
- 线程（Thread）是进程的一个实体，多个线程共享数据和资源，线程的主要应用在于可以在一个程序中同时运行多个任务
- 创建线程：程序员通过继承Thread类或实现Runnable接口
- 线程状态：新建New、就绪Runnable、运行Running、阻塞Blocked和死亡Dead 5种状态
- 线程的执行次序：通过设置线程的优先级**PRIORITY()方法来控制
- 线程同步：Java引用“监视器”的概念，通过同步块和同步方法等两种方式来实现同步，线程同步可能导致死锁的产生
- Java多个线程之间通信：wait()、notify()和notifyAll()方法实现通信
- 执行定时任务：Timer和Swing Timer控件用于执行规划好的任务或循环任务

总结

- 单线程串行程序-----主线程多个算法串行执行
- 多线程并发程序-----多个线程之间通过分时技术并发执行，这样的程序被称为多线程并发程序
- 一个程序包含多个线程，其中一个是系统自动创建的主线程main thread，其余的则是由程序员创建的子线程sub thread，子线程在启动后将与主线程并发执行，分头执行各自的算法
- 如何启动线程：如何启动线程：调用线程对象的start()方法，计算机将在线程中执行算法对象的run()方法，该线程与所在进行的其他线程包括主线程并发执行
- 每个线程都有线程号ID，线程名称name，优先级priority，状态state等属性
- 如何将算法放入一个线程中运行？

① 将算法封装成一个可运行的算法对象

② 创建线程对象，在线程对象中运行算法对象