# 《面向对象与多线程综合实验》指导书

本课程旨在使学生了解软件开发的一般过程，深入理解面向对象语言的基本概念和基本原理，理解和掌握继承与多态、异常处理、输入输出流、GUI设计、JDBC数据库操作、网络编程、多线程等技术；熟练掌握在Java语言环境下，上述技术的具体实现方法，并深入领会Java程序设计实用开发技术。

为达到上述目标，本课程计划让学生以迭代方式逐步编程开发一个小型档案管理系统，实现上述7个部分的训练，从而培养学生分析和解决问题的能力。该小型档案系统要求如下：

1. 系统基于C/S模式，包括客户端子系统，服务器端子系统
2. 用户需登录，验证口令通过后才能使用系统。用户分为系统管理人员、档案录入人员，档案浏览人员
3. 系统中相关用户信息、档案属性存放于关系数据库中，档案本身以文件形式存放于服务器相关目录中
4. 客户端和服务器端基于Socket实现通信，服务器端需实现多线程功能，可同时处理、响应多个客户端的数据请求
5. 用户信息管理功能，普通用户登录后可对自己基本信息(姓名、性别、民族、出生日期、职务、密码等信息)的查询、修改。系统管理人员除上述功能外壳增加、删除用户、修改用户的相关属性和权限
6. 档案数据录入功能，档案录入人员可输入新的档案文件信息，并将档案文件上传至服务器
7. 档案数据查询功能，普通用户可按条件查询相应的档案文件信息，对相关档案文件可以下载保存

## 封装、继承与多态

### 1.1 知识要点

（1）封装

封装(Encapsulation)是类的三大特征之一，即将类的状态信息隐藏在类内部，不允许外部程序直接访问，必须通过该类提供的方法来实现对隐藏信息的操作和访问。简而言之就是隐藏对象的属性和实现细节，仅向外提供访问接口。

封装的好处主要有：隐藏类的实现细节；让使用者只能通过程序员规定的方法来访问数据；可以方便的加入存取控制语句，限制不合理操作。

封装的具体步骤：修改属性的可见性来限制对属性的访问；为每个属性创建一对赋值(setter)方法和取值(getter)方法，用于对这些属性的存取；在赋值方法中，加入对属性的存取控制、有效性检查等。

（2）继承

继承(Inheritance)是Java中实现代码重用的重要手段，它是一种软件复用形式。继承是使用已存在的类的定义作为基础建立新类的技术，新类可以吸收现有类的成员，并可以增加新的功能或修改原有的功能。这种技术使得复用以前的代码非常容易，能够大大缩短开发周期，降低开发费用。

继承表达的是is-a的关系，或者说是一种特殊和一般的关系。Java中只支持单继承，即每个类只能有一个父类。在Java中所有的类都直接或间接的继承了java.lang.Object类，即Object类是所有java类的祖先。

在Java中，子类可以从父类中继承的有：继承public和protected修饰的属性和方法，不管子类和父类是否在同一个包里；继承默认权限修饰符修饰的属性和方法,但子类和父类必须在同一个包里。子类无法继承父类的有：无法继承private修饰的属性和方法；无法继承父类的构造方法。

（3）多态

多态(Polymorphism) 即多种状态，是指计算机程序运行时，系统可依据对象所属类，引发对应类的方法，从而有不同的行为。简单来说，所谓多态意指相同的消息给予不同的对象会引发不同的动作。

多态又可分为设计时多态和运行时多态。重载又被称为设计时多态，而对于覆盖或继承的方法，Java运行时系统根据调用该方法的实例的类型来决定选择调用哪个方法则被称为运行时多态。多态的优点可归纳为五个方面：可替换性、可扩充性、接口性、灵活性和简化性。

在Java中实现多态的三个条件：继承的存在(继承是多态的基础,没有继承就没有多态)；子类重写父类的方法(多态下调用子类重写的方法)；父类引用变量指向子类对象(子类到父类的类型转换)。

特别注意：子类转换成父类时的规则：将一个父类的引用指向一个子类的对象，称为向上转型，自动进行类型转换。此时通过父类引用调用的方法是子类覆盖或继承父类的方法，不是父类的方法。此时通过父类引用变量无法调用子类特有的方法。如果父类要调用子类的特有方法就得将一个指向子类对象的父类引用赋给一个子类的引用，称为向下转型,此时必须进行强制类型转换。

### 实验目的

掌握类的定义、对象的创建、对象的属性的引用和方法的调用；熟悉Java中的继承机制，方法的重载与覆盖；掌握多态、抽象类、接口的使用。

### 1.3 实验内容

编写一个程序，实现档案管理系统中的用户管理模块。要求模块中实现用户的模拟登录过程。通过用户输入，获取用户名和口令；与事先记录在程序中的用户信息进行对比，通过口令验证后才能使用系统。用户分为系统管理人员、档案录入人员，档案浏览人员三类，相关类图如下所示。

（1）要求在用户类中实现封装，并构造setter、getter方法实现属性的访问。

（2）通过继承方式加以实现上述类。

（3）通过多态，实现用户菜单项的展示，根据用户角色不同，系统自动调用对应showMenu()方法。

（4）在未讲数据库之前，系统中已存在用户的信息放置在Hashtable中。提供DataProcessing类实现数据的查找、插入、更新和删除。（此类可提供给学生直接使用）

（5）在未讲I/O之前，系统中的文件操作的方法可虚化，只用打印语句即可。

## 2. 异常处理

### 2.1 知识要点

（1）异常

异常指不期而至的各种状况，如：文件找不到、网络连接失败、数据库错误等。异常是一个事件，它发生在程序运行期间，干扰了正常的指令流程。Java通过API中Throwable类的众多子类描述各种不同的异常。因而，Java异常都是对象，是Throwable子类的实例，描述了出现在一段编码中的错误条件。当条件生成时，错误将引发异常。



Throwable：有两个重要的子类：Exception（异常）和Error（错误），二者都是Java异常处理的重要子类，各自都包含大量子类。

Error：是程序无法处理的错误，表示运行应用程序中较严重问题。大多数错误与代码编写者执行的操作无关，而表示代码运行时 JVM出现的问题。

Exception：是程序本身可以处理的异常。异常和错误的区别：异常能被程序本身可以处理，错误是无法处理。

通常，Java的异常(包括Exception和Error)分为可查的异常（checked exceptions）和不可查的异常（unchecked exceptions）。不可查异常是编译器不要求强制处置的异常：包括运行时异常（RuntimeException与其子类）和错误（Error）。运行时异常是不检查异常，程序中可以选择捕获处理，也可以不处理。除了RuntimeException及其子类以外，其他的Exception类及其子类都属于可查异常。这种异常的特点是Java编译器会检查它，也就是说，当程序中可能出现这类异常，要么用try-catch语句捕获它，要么用throws子句声明抛出它。

（2）处理异常机制

在 Java 应用程序中，异常处理机制为：抛出异常，捕捉异常。

A．抛出异常

当一个方法出现错误引发异常时，方法创建异常对象并交付运行时系统，异常对象中包含了异常类型和异常出现时的程序状态等异常信息。运行时系统负责寻找处置异常的代码并执行。

* throws抛出异常

如果一个方法可能会出现异常，但没有能力处理这种异常，可以在方法声明处用throws子句来声明抛出异常。throws语句的语法格式为：

methodname throws Exception1,Exception2,..,ExceptionN {…… }

* throw抛出异常

throw总是出现在方法体中，用来抛出一个Throwable类型的异常。程序会在throw语句后立即终止，它后面的语句执行不到。throw语句的语法格式为：

throw new exceptionname;

B．捕获异常

在方法抛出异常之后，运行时系统将转为寻找合适的异常处理器（exception handler）。当异常处理器所能处理的异常类型与方法抛出的异常类型相符时，即为合适的异常处理器。运行时系统从发生异常的方法开始，依次回查调用栈中的方法，直至找到含有合适异常处理器的方法并执行。当运行时系统遍历调用栈而未找到合适的异常处理器，则运行时系统终止。同时，意味着Java程序的终止。

* 捕获异常：try、catch 和 finally

try {

// 可能会发生异常的程序代码

} catch (Type1 id1) {

// 捕获并处理try抛出的异常类型Type1

} catch (Type2 id2) {

// 捕获并处理try抛出的异常类型Type2

} finally {

// 无论是否发生异常，都将执行的语句块

}

try 块：用于捕获异常。其后可接零个或多个catch块，如果没有catch块，则必须跟一个finally块。

catch 块：用于处理try捕获到的异常。

finally 块：无论是否捕获或处理异常，finally块里的语句都会被执行。当在try块或catch块中遇到return语句时，finally语句块将在方法返回之前被执行。

### 2.2 实验目的

理解异常的基本概念；了解Java异常的层次结构；熟悉并掌握Java异常的捕获处理方法。

### 2.3 实验内容

（1）阅读Java™ Platform, Standard Edition 8 API Specification文档，了解后续编程中将要处理的IOException及其子类FileNotFoundException、EOFException，SocketException，SQLException以及运行时异常RuntimeException与其子类IllegalStateException。

（2）根据新提供的DataProcessing类（因还未讲SQL，此类模拟异常出现情况，以一定概率随机产生异常），在所编写的Administrator、Operator和Browser类，增加异常处理功能。

## 3. 输入输出流

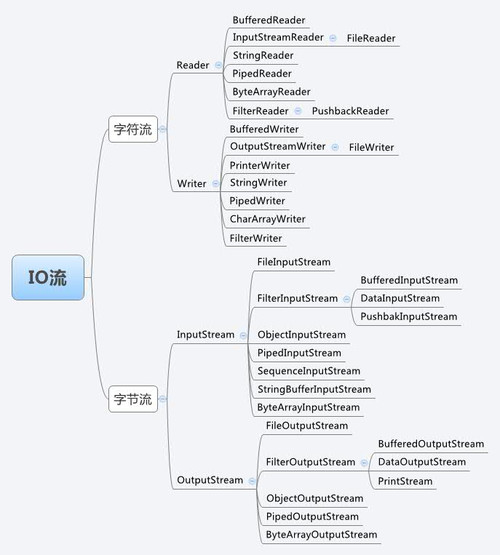
### 3.1 知识要点

（1）流的概念

流（Stream）的概念源于UNIX。流的单向的，根据流的方向，流可以分输入流和输出流。可以从输入流中读取信息，但不能写它。相反，只能往输出流写入信息，但不能读它。

按流的数据单位不同分为：字节流和字符流。在Java中，字节流读写以字节为单位的流；而字符流用于读写Unicode字符组成的文本流。

（2）流的层次结构



A. File：用于文件或者目录的描述信息，例如生成新目录，修改文件名，删除文件，判断文件所在路径等。

构造函数：File( String path)、File(String path, String FileName)、File(File dir, String name)。

File类提供了一种与机器无关的方式来描述一个文件对象的属性，通过类File所提供的方法，可以得到文件或目录的描述信息，这主要包括名称、所在路经、可读性、可写性、文件的长度等，还可以生成新的目录、改变文件名、删除文件、列出一个目录中所有的文件等。

B. InputStream：抽象类，基于字节的输入操作，是所有输入流的父类。定义了所有输入流都具有的共同特征。

主要提供文件内容操作的基本功能函数read()、 skip()、close()等；一般都是创建出其派生类对象（完成指定的特殊功能）来实现文件读取。文件操作的应注意异常的捕获：由于包java.io中几乎所有的类都声明有I/O异常，因此程序应该对这些异常加以处理。

流结束的判断：方法read()的返回值为-1时；readLine()的返回值为null时。

C. OutputStream：抽象类。基于字节的输出操作。是所有输出流的父类。定义了所有输出流都具有的共同特征。

主要提供文件内容操作的基本功能函数write()、 flush()、close()等；一般都是创建出其派生类对象（完成指定的特殊功能）来实现文件写入。文件操作的应注意异常的捕获：由于包java.io中几乎所有的类都声明有I/O异常，因此程序应该对这些异常加以处理。

D. Reader：抽象类，基于字符的输入操作。

提供文件内容操作的基本功能函数read()、 skip()、close()等；文件操作的应注意异常的捕获； InputStreamReader 是字节流和字符流的桥梁，它使用指定的 charset 读取字节并将其解码为字符。

E. Writer：抽象类，基于字符的输出操作。

提供文件内容操作的基本功能函数write()、 flush()、close()等；文件操作的应注意异常的捕获； OutputStreamWriter是字符流和字节流的桥梁，它使用指定的charset将要写入流中的字符编码成字节。

F. RandomAccessFile：随机文件操作，可以从文件的任意位置进行存取（输入输出）操作。

RandomAccessFile类直接继承于Object类而非InputStream/OutputStream类，从而可以实现读写文件中任何位置中的数据（只需要改变文件的读写位置的指针）。由于RandomAccessFile类实现了DataOutput与DataInput接口，因而利用它可以读写Java中的不同类型的基本类型数据（比如采用readLong()方法读取长整数，而利用readInt()方法可以读出整数值等）。

关于上述内容具体描述，请参看Java教材与Java™ Platform, Standard Edition 8 API Specification文档。

### 3.2 实验目的

了解Java中I/O流的概念和种类；掌握字节流处理和字符流处理，包括File类，InputStream/OutputStream及其子类，Reader/Writer及其子类；熟练掌握文件的顺序处理，随机访问处理；熟悉对象串行化的概念和方法。

### 3.3 实验内容

编写程序，实现档案管理系统中的文件上传/下载模块。要求用户登录系统后，可根据系统存储数据，浏览已有档案资料信息；可根据档案号，下载对应档案文件至指定目录；可输入并在系统中记录新档案信息，并将对应的档案文件上传至指定目录。要求如下：

（1）完善showFileList()方法，实现档案信息浏览，在未讲数据库之前，系统中已存在档案信息放置在Hashtable中，提供新版DataProcessing类，该类实现了对应数据的查找、插入操作。

（2）完善uploadFile()方法，实现档案数据的上传，在未讲网络之前，该方法只需实现在指定目录中读取文件，并将其拷贝至其他目录中，此外还需将相关档案信息写入对应Hashtable中。

（3）完善downloadFile()，实现档案数据下载，目前只需要实现根据档案号，在Hashtable中查找得到文件位置，然后读取文件并将其拷贝至指定目录中。

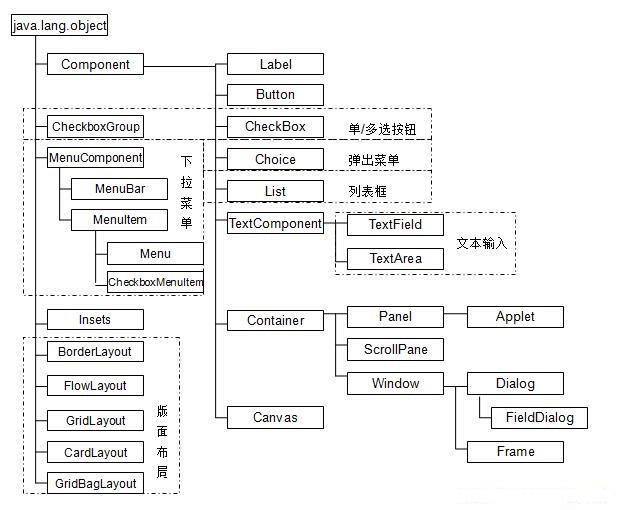
## 4. GUI设计

### 4.1 知识要点

（1）GUI-Graphics User Interface

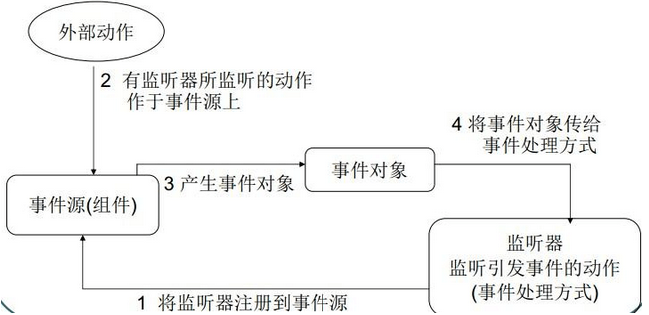
GUI，图形用户界面，又称图形用户接口，是指采用图形方式显示的计算机操作用户界面。允许用户使用鼠标等输入设备操纵屏幕上的图标或菜单选项，以选择命令、调用文件、启动程序或执行其它一些日常任务。与通过键盘输入文本或字符命令来完成例行任务的字符界面相比，图形用户界面有许多优点。图形用户界面由窗口、下拉菜单、对话框及其相应的控制机制构成，通过图形用户界面GUI，用户和程序之间可以方便的进行交互。

Java中包含了许多用于支持GUI设计的类，如按钮、菜单、列表、文本框等组件类，以及窗口、面板等容器类。Java GUI包主要有AWT包(java.awt.\*) 和Swing包(javax.swing.\*)。



（2）事件处理

Java采用委托事件模型来处理事件，其特点是将用户界面对象与程序处理逻辑分开，把事件的处理委托给独立的对象。委托事件模型由事件(Event)、事件源(Source)、事件监听器(Listeners)组成。事件源就是awt包或者swing包中的图形界面组件。每一个事件源都有自己特有的对应事件和共性事件。监听器将可以触发某一个事件的动作封装到监听器中。即，一个事件源产生一个事件并将它送给一个或者多个监听者，由其对象的相关方法进行处理。



### 4.2实验目的

了解Java图形界面程序的基本结构；掌握Java布局管理和常用组件的使用；掌握Java事件处理机制。

### 4.3 实验内容

编写程序，将前面课程所编写的档案管理系统改编为图形用户界面。要求程序界面选用合适的布局，综合使用菜单、按钮、文本框、密码框、下拉列表、文件对话框等组件，实现良好的人机接口。

## 5. JDBC数据库操作

### 5.1 知识要点

（1）JDBC简介

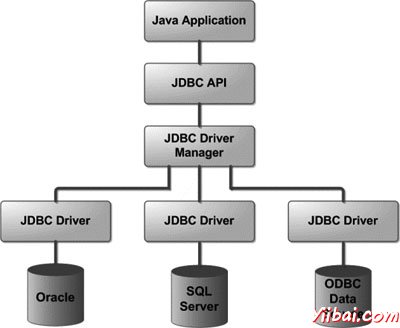
JDBC（Java Data Base Connectivity, java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

JDBC对Java编程语言和广泛的数据库之间独立于数据库的连接标准的Java API。JDBC库包含的API为每个通常与数据库的使用相关联的任务：

* 使得连接到数据库
* 创建SQL语句
* 执行SQL的查询数据库
* 查看和修改结果记录

JDBC API支持两层和三层处理模型进行数据库访问，但在一般的JDBC体系结构由两层组成：

* JDBC API: 提供了应用程序对JDBC的管理连接。
* JDBC Driver API: 支持JDBC管理到驱动器连接。



（2）常见的JDBC组件

JDBC API提供了以下接口和类：

DriverManager: 这个类管理数据库驱动程序的列表。内容是否符合从Java应用程序使用的通信子协议正确的数据库驱动程序的连接请求。识别JDBC在一定子协议的第一个驱动器将被用来建立数据库连接。

Driver: 此接口处理与数据库服务器通信。很少直接与驱动程序对象。相反，使用DriverManager中的对象，它管理此类型的对象。它也抽象与驱动程序对象工作相关的详细信息。

Connection : 此接口与接触数据库的所有方法。连接对象表示通信上下文，即，与数据库中的所有的通信是通过唯一的连接对象。

Statement : 可以使用这个接口创建的对象的SQL语句提交到数据库。一些派生的接口接受除执行存储过程的参数。

ResultSet: 这些对象保存从数据库后，执行使用Statement对象的SQL查询中检索数据。它作为一个迭代器，让您可以通过移动它的数据。

SQLException: 这个类处理发生在一个数据库应用程序的任何错误。

（3）JDBC连接数据库代码和步骤

创建一个以JDBC连接数据库的程序，包含7个步骤：

1. 加载JDBC驱动程序：

在连接数据库之前，首先要加载想要连接的数据库的驱动到JVM（Java虚拟机）， 这通过java.lang.Class类的静态方法forName(String className)实现。

try{

//加载MySql的驱动类

Class.forName("com.mysql.jdbc.Driver") ;

}catch(ClassNotFoundException e){

System.out.println("找不到驱动程序类 ，加载驱动失败！");

e.printStackTrace() ;

}

成功加载后，会将Driver类的实例注册到DriverManager类中。

1. 提供JDBC连接的URL

连接URL定义了连接数据库时的协议、子协议、数据源标识。书写形式：协议：子协议：数据源标识。

协议：在JDBC中总是以jdbc开始；子协议：是桥连接的驱动程序或是数据库管理系统名称；数据源标识：标记找到数据库来源的地址与连接端口。

例如：（MySql的连接URL）jdbc:mysql://localhost:3306/test;

1. 创建数据库的连接

要连接数据库，需要向java.sql.DriverManager请求并获得Connection对象， 该对象就代表一个数据库的连接。

用DriverManager的getConnection(String url, String username, String password)方法传入指定的欲连接的数据库的路径、数据库的用户名和密码来获得。

例如：连接MySql数据库，假设用户名和密码都是root

String url = "jdbc:mysql://localhost:3306/test" ;

String username = "root" ;

String password = "root" ;

try{

Connection con =

DriverManager.getConnection(url , username , password ) ;

}catch(SQLException se){

System.out.println("数据库连接失败！");

se.printStackTrace() ;

}

1. 创建一个Statement

要执行SQL语句，必须获得java.sql.Statement实例，Statement实例分为以下3种类型：

* + 执行静态SQL语句。通常通过Statement实例实现。
  + 执行动态SQL语句。通常通过PreparedStatement实例实现。
  + 执行数据库存储过程。通常通过CallableStatement实例实现。

具体的实现方式：

Statement stmt = con.createStatement() ;

PreparedStatement pstmt = con.prepareStatement(sql) ;

CallableStatement cstmt = con.prepareCall("{CALL demoSp(? , ?)}") ;

1. 执行SQL语句

Statement接口提供了三种执行SQL语句的方法：executeQuery 、executeUpdate 和execute

* + - ResultSet executeQuery(String sqlString)：执行查询数据库的SQL语句，返回一个结果集（ResultSet）对象。
    - int executeUpdate(String sqlString)：用于执行INSERT、UPDATE或 DELETE语句以及SQL DDL语句，如：CREATE TABLE和DROP TABLE等
    - execute(sqlString):用于执行返回多个结果集、多个更新计数或二者组合的语句。

具体实现的代码：

ResultSet rs = stmt.executeQuery("SELECT \* FROM ...") ;

int rows = stmt.executeUpdate("INSERT INTO ...") ;

boolean flag = stmt.execute(String sql) ;

1. 处理结果

有两种情况：

* + 执行更新返回的是本次操作影响到的记录数。
  + 执行查询返回的结果是一个ResultSet对象。

ResultSet包含符合SQL语句中条件的所有行，并且它通过一套get方法提供了对这些行中数据的访问。使用结果集（ResultSet）对象的访问方法获取数据：

while(rs.next()){

String name = rs.getString("name") ;

String pass = rs.getString(1) ;

}

（列是从左到右编号的，并且从列1开始）

1. 关闭JDBC对象

操作完成以后要把所有使用的JDBC对象全都关闭，以释放JDBC资源，关闭顺序和声明顺序相反：

* + 关闭记录集
  + 关闭声明
  + 关闭连接对象

if (rs != null){ // 关闭记录集

try{

rs.close() ;

}catch(SQLException e){

e.printStackTrace() ;

}

}

if(stmt != null){ // 关闭声明

try{

stmt.close() ;

}catch(SQLException e){

e.printStackTrace() ;

}

}

if (conn != null){ // 关闭连接对象

try{

conn.close() ;

}catch(SQLException e){

e.printStackTrace() ;

}

}

（4）SQL简介

结构化查询语言(Structured Query Language)简称SQL，是一种特殊目的的编程语言，是一种数据库查询和程序设计语言，用于存取数据以及查询、更新和管理关系数据库系统；同时也是数据库脚本文件的扩展名。

可以把 SQL 分为两个部分：数据操作语言 (DML) 和数据定义语言 (DDL)。SQL (结构化查询语言)是用于执行查询的语法。但是 SQL 语言也包含用于更新、插入和删除记录的语法。

查询和更新指令构成了 SQL 的 DML 部分：

* SELECT - 从数据库表中获取数据

SELECT 列名称 FROM 表名称 WHERE 列 运算符 值

* UPDATE - 更新数据库表中的数据

UPDATE 表名称 SET 列名称 = 新值 WHERE 列名称 = 某值

* DELETE - 从数据库表中删除数据

DELETE FROM 表名称 WHERE 列名称 = 值

* INSERT INTO - 向数据库表中插入数据

INSERT INTO table\_name (列1, 列2,...) VALUES (值1, 值2,....)

### 5.2实验目的

理解 JDBC的特点，结构，应用模型；了解JDBC驱动程序的类型；掌握通过JDBC访问数据库执行SQL 语句的方法。

### 5.3 实验内容

编写程序，将前面课程所编写的档案管理系统的数据管理方式改写为数据库存储。（建议将所有数据库操作封装在DataProcessing类中，并保留原来接口，这样可保持原有程序不做大的变化）

## 6. 网络编程

### 6.1 知识要点

（1）网络基础知识

网络编程的目的就是指直接或间接地通过网络协议与其他计算机进行通讯。网络编程中有两个主要的问题，一个是如何准确的定位网络上一台或多台主机，另一个就是找到主机后如何可靠高效的进行数据传输。在TCP/IP协议中IP层主要负责网络主机的定位，数据传输的路由，由IP地址可以唯一地确定Internet上的一台主机。而TCP层则提供面向应用的可靠的或非可靠的数据传输机制，这是网络编程的主要对象，一般不需要关心IP层是如何处理数据的。

目前较为流行的网络编程模型是客户机/服务器（C/S）结构。即通信双方一方作为服务器等待客户提出请求并予以响应。客户则在需要服务时向服务器提出申请。服务器一般作为守护进程始终运行，监听网络端口，一旦有客户请求，就会启动一个服务进程来响应该客户，同时自己继续监听服务端口，使后来的客户也能及时得到服务。

（2）TCP/IP和Socket

　　TCP/IP协议由网络层的IP协议和传输层的TCP协议组成。IP层负责网络主机的定位，数据传输的路由，由IP地址可以唯一的确定Internet上的一台主机。TCP层负责面向应用的可靠的或非可靠的数据传输机制，这是网络编程的主要对象。TCP是一种面向连接的保证可靠传输的协议。通过TCP协议传输，得到的是一个顺序的无差错的数据流。UDP是一种面向无连接的协议，每个数据报都是一个独立的信息，包括完整的源地址或目的地址，它在网络上以任何可能的路径传往目的地，因此能否到达目的地，到达目的地的时间以及内容的正确性都是不能被保证的。

Socket通常也称作"套接字"，用于描述IP地址和端口，是一个通信链的句柄。网络上的两个程序通过一个双向的通讯连接实现数据的交换，这个双向链路的一端称为一个Socket，一个Socket由一个IP地址和一个端口号唯一确定。应用程序通常通过"套接字"向网络发出请求或者应答网络请求。 Socket是TCP/IP协议的一个十分流行的编程界面，但是，Socket所支持的协议种类也不光TCP/IP一种，因此两者之间是没有必然联系的。在Java环境下，Socket编程主要是指基于TCP/IP协议的网络编程。

　　Socket通讯过程：服务端监听某个端口是否有连接请求，客户端向服务端发送连接请求，服务端收到连接请求向客户端发出接收消息，这样一个连接就建立起来了。客户端和服务端都可以相互发送消息与对方进行通讯。

　　Socket的基本工作过程包含以下四个步骤：

* + 创建Socket；
  + 打开连接到Socket的输入输出流；
  + 按照一定的协议对Socket进行读写操作；
  + 关闭Socket。

（3）Java中的Socket

在java.net包下有两个类：Socket和ServerSocket。ServerSocket用于服务器端，Socket是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个Socket实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服务器端或在客户端而产生不同级别。不管是Socket还是ServerSocket它们的工作都是通过SocketImpl类及其子类完成的。

下面列出Socket和ServerSocket几个常用的构造方法：

* Socket(InetAddress address,int port);//创建一个流套接字并将其连接到指定 IP 地址的指定端口号
* Socket(String host,int port);//创建一个流套接字并将其连接到指定主机上的指定端口号
* Socket(InetAddress address,int port, InetAddress localAddr,int localPort);//创建一个套接字并将其连接到指定远程地址上的指定远程端口
* Socket(String host,int port, InetAddress localAddr,int localPort);//创建一个套接字并将其连接到指定远程主机上的指定远程端口
* Socket(SocketImpl impl);//使用用户指定的 SocketImpl 创建一个未连接 Socket
* ServerSocket(int port);//创建绑定到特定端口的服务器套接字
* ServerSocket(int port,int backlog);//利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号
* ServerSocket(int port,int backlog, InetAddress bindAddr);//使用指定的端口、侦听 backlog 和要绑定到的本地 IP地址创建服务器

　　构造方法的参数中，address、host和port分别是双向连接中另一方的IP地址、主机名和端 口号，stream指明socket是流socket还是数据报socket，localPort表示本地主机的端口号，localAddr和bindAddr是本地机器的地址（ServerSocket的主机地址），impl是socket的父类，既可以用来创建serverSocket又可以用来创建Socket。count则表示服务端所能支持的最大连接数。

注意：必须小心选择端口号。每一个端口提供一种特定的服务，只有给出正确的端口，才 能获得相应的服务。0~1023的端口号为系统所保留，例如http服务的端口号为80,telnet服务的端口号为21,ftp服务的端口号为23, 所以我们在选择端口号时，最好选择一个大于1023的数以防止发生冲突。

下面列出几个重要的Socke方法：

* public InputStream getInputStream();//方法获得网络连接输入，同时返回一个IutputStream对象实例
* public OutputStream getOutputStream();//方法连接的另一端将得到输入，同时返回一个OutputStream对象实例
* public Socket accept();//用于产生"阻塞"，直到接受到一个连接，并且返回一个客户端的Socket对象实例。"阻塞"是一个术语，它使程序运行暂时"停留"在这个地方，直到一个会话产生，然后程序继续；通常"阻塞"是由循环产生的。

注意：其中getInputStream和getOutputStream方法均会产生一个IOException，它必须被捕获，因为它们返回的流对象，通常都会被另一个流对象使用。

（4）基本的Client/Server程序示例

以下是一个基本的客户端/服务器端程序代码。主要实现了服务器端一直监听某个端口，等待客户端连接请求。客户端根据IP地址和端口号连接服务器端，从键盘上输入一行信息，发送到服务器端，然后接收服务器端返回的信息，最后结束会话。这个程序一次只能接受一个客户连接。

客户端程序：

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class SocketClient {

public static void main(String[] args) {

try {

/\*\* 创建Socket\*/

// 创建一个流套接字并将其连接到指定 IP 地址的指定端口号

Socket socket =new Socket("127.0.0.1",2013);

// 60s超时

socket.setSoTimeout(60000);

/\*\* 发送客户端准备传输的信息 \*/

// 由Socket对象得到输出流，并构造PrintWriter对象

PrintWriter printWriter =new

PrintWriter(socket.getOutputStream(),true);

// 将输入读入的字符串输出到Server

BufferedReader sysBuff =new BufferedReader(

new InputStreamReader(System.in));

printWriter.println(sysBuff.readLine());

// 刷新输出流，使Server马上收到该字符串

printWriter.flush();

/\*\* 用于获取服务端传输来的信息 \*/

// 由Socket对象得到输入流，并构造相应的BufferedReader对象

BufferedReader bufferedReader =new BufferedReader(

new InputStreamReader(socket.getInputStream()));

// 输入读入一字符串

String result = bufferedReader.readLine();

System.out.println("Server say : " + result);

/\*\* 关闭Socket\*/

printWriter.close();

bufferedReader.close();

socket.close();

}catch (Exception e) {

System.out.println("Exception:" + e);

}

}

}

服务器端程序：

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class SocketServer {

public static void main(String[] args) {

try {

/\*\* 创建ServerSocket\*/

// 创建一个ServerSocket在端口2013监听客户请求

ServerSocket serverSocket =new ServerSocket(2013);

while (true) {

// 侦听并接受到此Socket的连接,请求到来则产生一个Socket

//对象，并继续执行

Socket socket = serverSocket.accept();

/\*\* 获取客户端传来的信息 \*/

// 由Socket得到输入流，并构造相应的BufferedReader对象

BufferedReader bufferedReader =new BufferedReader(

new InputStreamReader(socket.getInputStream()));

// 获取从客户端读入的字符串

String result = bufferedReader.readLine();

System.out.println("Client say : " + result);

/\*\* 发送服务端准备传输的 \*/

// 由Socket对象得到输出流，并构造PrintWriter对象

PrintWriter printWriter =new

PrintWriter(socket.getOutputStream());

printWriter.print("hello Client, I am Server!");

printWriter.flush();

/\*\* 关闭Socket\*/

printWriter.close();

bufferedReader.close();

socket.close();

}

}catch (Exception e) {

System.out.println("Exception:" + e);

}finally{

// serverSocket.close();

}

}

}

（5）文件传输示例

客户端向服务器端传送文件，服务端可获取文件名用于保存，获取文件大小计算传输进度，比较简单。

客户端代码：

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.net.Socket;

public class Client extends Socket{

private static final String SERVER\_IP ="127.0.0.1";

private static final int SERVER\_PORT =2013;

private Socket client;

private FileInputStream fis;

private DataOutputStream dos;

public Client(){

try {

try {

client =new Socket(SERVER\_IP, SERVER\_PORT);

//向服务端传送文件

File file =new File("c:/test.doc");

fis =new FileInputStream(file);

dos =new DataOutputStream(client.getOutputStream());

//文件名和长度

dos.writeUTF(file.getName());

dos.flush();

dos.writeLong(file.length());

dos.flush();

//传输文件

byte[] sendBytes =new byte[1024];

int length =0;

while((length = fis.read(sendBytes,0, sendBytes.length)) >0){

dos.write(sendBytes,0, length);

dos.flush();

}

}catch (Exception e) {

e.printStackTrace();

}finally{

if(fis !=null)

fis.close();

if(dos !=null)

dos.close();

client.close();

}

}catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args)throws Exception {

new Client();

}

}

服务器端代码：

import java.io.DataInputStream;

import java.io.File;

import java.io.FileOutputStream;

import java.net.ServerSocket;

import java.net.Socket;

public class Server extends ServerSocket{

private static final int PORT =2013;

private ServerSocket server;

private Socket client;

private DataInputStream dis;

private FileOutputStream fos;

public Server()throws Exception{

try {

try {

server =new ServerSocket(PORT);

while(true){

client = server.accept();

dis =new DataInputStream(client.getInputStream());

//文件名和长度

String fileName = dis.readUTF();

long fileLength = dis.readLong();

fos =new FileOutputStream(new File("d:/" + fileName));

byte[] sendBytes =new byte[1024];

int transLen =0;

System.out.println("----开始接收文件<" + fileName +">,文件大小为<" + fileLength +">----");

while(true){

int read =0;

read = dis.read(sendBytes);

if(read == -1)

break;

transLen += read;

System.out.println("接收文件进度" +100 \* transLen/fileLength +"%...");

fos.write(sendBytes,0, read);

fos.flush();

}

System.out.println("----接收文件<" + fileName +">成功-------");

client.close();

}

}catch (Exception e) {

e.printStackTrace();

}finally {

if(dis !=null)

dis.close();

if(fos !=null)

fos.close();

server.close();

}

}catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args)throws Exception {

new Server();

}

}

### 6.2实验目的

了解Java网络编程基础知识；掌握java.net包中关于网络的基本类及其属性和方法；掌握基于Socket的客户和服务器编程方法。

### 6.3 实验内容

编写程序，将前面课程所编写的档案管理系统从单机版改编成为客户机/服务器模式，实现档案文件在客户机和服务器之间的上传、下载。

## 7. 多线程

### 7.1 知识要点

（1）Java多线线程

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程序。

A．创建线程

在Java中创建线程有两种方法：使用Thread类和使用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。Thread构造函数：

* public Thread( );
* public Thread(Runnable target);
* public Thread(String name);
* public Thread(Runnable target, String name);
* public Thread(ThreadGroup group, Runnable target);
* public Thread(ThreadGroup group, String name);
* public Thread(ThreadGroup group, Runnable target, String name);
* public Thread(ThreadGroup group, Runnable target, String name, long stackSize);

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);

}

}

}

class Demo extends Thread{

public void run(){

for(int i=0;i<60;i++){

System.out.println(Thread.currentThread().getName()+i);

}

}

}

方法二：使用Runnable接口

提供一个实现接口Runnable的类,实现run()方法；创建一个这个新类的对象A；New 一个Thread对象，同时A作为参数传入；调用start启动线程。

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);

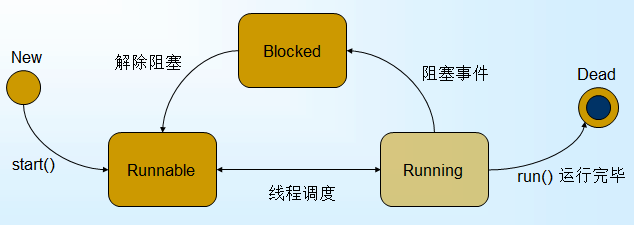
}

}

}

B．线程的生命周期

Java线程也同样要经历新生、就绪、运行、阻塞和终止等不同的状态。这些状态都可以通过Thread类中的方法进行控制。



相关方法介绍如下

|  |  |
| --- | --- |
| **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 |

（2）基于Socket的多客户端连接服务器

上次设计的程序，服务器端程序一次只能连接一个客户端，这在实际应用中显然是不可能的。通常的网络环境是多个客户端连接到某个主机进行通讯，所以需要对程序进行改造。下面举例如何实现多客户端连接服务器。

设计思路：服务器端主程序监听某一个端口，客户端发起连接请求，服务器端主程序接收请求，同时构造一个线程类，用于接管会话。当一个Socket会话产生后，这个会话就会交给线程进行处理，主程序继续进行监听。客户端和服务器建立连接，客户端发送消息，服务端根据消息进行处理并返回消息，若客户端申请关闭，则服务器关闭此连接，双方通讯结束。

客户端程序：

import java.io.BufferedReader;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class SocketClient {

public static void main(String[] args) {

try {

Socket socket =new Socket("127.0.0.1",2013);

socket.setSoTimeout(60000);

PrintWriter printWriter =new

PrintWriter(socket.getOutputStream(),true);

BufferedReader bufferedReader =new BufferedReader(new

InputStreamReader(socket.getInputStream()));

String result ="";

while(result.indexOf("bye") == -1){

BufferedReader sysBuff =new BufferedReader(new

InputStreamReader(System.in));

printWriter.println(sysBuff.readLine());

printWriter.flush();

result = bufferedReader.readLine();

System.out.println("Server say : " + result);

}

printWriter.close();

bufferedReader.close();

socket.close();

}catch (Exception e) {

System.out.println("Exception:" + e);

}

}

}

服务器端程序：

import java.io.\*;

import java.net.\*;

public class Server extends ServerSocket {

private static final int SERVER\_PORT =2013;

public Server()throws IOException {

super(SERVER\_PORT);

try {

while (true) {

Socket socket = accept();

new CreateServerThread(socket);//当有请求时，启一个线程处理

}

}catch (IOException e) {

}finally {

close();

}

}

//线程类

class CreateServerThread extends Thread {

private Socket client;

private BufferedReader bufferedReader;

private PrintWriter printWriter;

public CreateServerThread(Socket s)throws IOException {

client = s;

bufferedReader =new BufferedReader(new

InputStreamReader(client.getInputStream()));

printWriter =new PrintWriter(client.getOutputStream(),true);

System.out.println("Client(" + getName() +") come in...");

start();

}

public void run() {

try {

String line = bufferedReader.readLine();

while (!line.equals("bye")) {

printWriter.println("continue, Client(" + getName() +")!");

line = bufferedReader.readLine();

System.out.println("Client(" + getName() +") say: " + line);

}

printWriter.println("bye, Client(" + getName() +")!");

System.out.println("Client(" + getName() +") exit!");

printWriter.close();

bufferedReader.close();

client.close();

}catch (IOException e) {

}

}

}

public static void main(String[] args)throws IOException {

new Server();

}

}

### 7.2实验目的

了解线程的基本概念；掌握线程的构造，调度，控制策略；掌握多线程互斥和同步处理方法。

### 7.3 实验内容

编写程序，将前面课程所编写的档案管理系统的服务器端改写为多线程方式，使得其能同时处理多个客户端的请求。