**第9课 Java 数据库编程**

|  |  |  |
| --- | --- | --- |
| **课 题** | Java 数据库编程 | |
| **课 时** | 5课时（225 min）。 | |
| **教学目标** | **知识技能目标：**  1．了解 JDBC 的原理和功能。  2．掌握使用 Java 连接数据库的基本步骤和方法。  **思政育人目标：**  让学生通过学习Java 数据库编程，了解Java 通过 JDBC 连接和管理各种类型的数据库。 | |
| **教学重难点** | **教学重点：**连接数据库  **教学难点：**访问记录 | |
| **教学方法** | 讲授法、问答法、讨论法 | |
| **教学用具** | 电脑、投影仪、多媒体课件、教材 | |
| **教学设计** | 第1节课：考勤（2min）--知识讲解（40min）--作业布置（3min）  第2节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第3节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第4节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第5节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min） | |
| **教学过程** | **主 要 教 学 内 容 及 步 骤** | **设计意图** |
| **考勤**  **（2min）** | ■【教师】清点上课人数，记录好考勤  ■【学生】班干部报请假人员及原因 | 培养学生的组织纪律性,掌握学生的出勤情况 |
| **知识讲解**  （40min） | **【教师】**展示连接数据库（一）  **任务描述**  1. 创建一个 SQL Server 数据库，并为其配置 ODBC 数据源。  2. 编写程序，通过 JDBC-ODBC 桥连接前面创建的 SQL Server 数据库。  **任务目标**  1. 会配置 ODBC 数据源。  2. 会连接 Java 中的数据库。  3. 能够掌握“芝麻开门”口诀，做一个能打开数据“宝库”的人。  **预备知识**  **一、JDBC 原理**  JDBC（Java Database Connectivity，Java 数据库连接）是一种用于执行 SQL 语句的Java API，它由一组用 Java 语言编写的类和接口组成。JDBC 为开发人员提供了一个标准的 API，使他们能够用纯 Java API 来编写数据库应用程序。  因为 JDBC 可以使程序独立于具体的数据库，因此不必为访问 Oracle 的数据库专门写一个程序，为访问 SQL Server 数据库又专门写一个程序，或者为访问 Access 数据库又写另一个程序。只需要用 JDBC API 写一个程序就够了，它负责向相应数据库发送SQL 语句。如果将 Java 和 JDBC 结合起来，则程序员就只用写一个程序就可以让它在任何平台上运行。  **（一）JDBC 的功能**  JDBC 的功能，简单来说就是做三件事：  （1）与数据库建立连接；  （2）发送 SQL 语句到数据库；  （3）存放 SQL 语句的结果。  下列代码给出了以上三步的基本示例：  Connection con = DriverManager.getConnection ("jdbc:odbc:dbname",  "username","password");  // 与数据库建立连接  Statement stmt = con.createStatement();  ResultSet res = stmt.executeQuery ("select a,b,c from table");  /\* 发送 SQL 语句并接收返回结果 \*/  while(res.next()) // 处理返回结果  {  int x = res.getInt("a");  String s = res.getString("b");  double d = res.getDouble("c");  }  **（二）JDBC 的驱动程序管理器**  Java 应用程序只通过 JDBC API 与数据库保持连接，在 JDBC 内部，驱动程序管理器 DriverManager 通过相应的驱动程序与具体的数据库系统进行连接。DriverManager 是JDBC 的管理层，作用于应用程序和驱动程序之间，它跟踪可用的驱动程序，并在数据库和相应的驱动程序之间建立连接。  **（三）JDBC 的驱动程序类型**  目前的 JDBC 驱动程序可分为四类。  1. JDBC−ODBC 桥  JDBC−ODBC 桥利用 ODBC 驱动程序提供了 JDBC 访问功能。它是 JDK 的一部分，即是 sun.jdbc.odbc 包的一部分。JDBC−ODBC 桥驱动程序简单，但由于需要本地 ODBC 方法，所以限制了它只能在 Windows 平台上使用。  2. Java to Native API  Java to Native API 用于将 JDBC 调用为 Oracle、Sybase、Informix、DB2 或其他DBMS 本身所带驱动程序的调用。由于事先必须在客户机上安装其他一些附加软件，会使 Java 程序的兼容性大大降低。  3. Net Protocol All −Java  Net Protocol All−Java 驱动程序是由纯 Java 语言开发的，它可以用在一个第三方的解决方案中，而且可以在 Internet 上使用。它通过驱动程序厂商所创建的专有网络协议来和某种中间件通信，这个中间件通常位于 Web 服务器或数据库服务器上，并且可以和数据库通信。  4. Pure JDBC Driver  Pure JDBC Driver 驱动程序也是用纯 Java 语言开发的，它将 JDBC 调用直接转换为DBMS 所使用的网络协议。这将允许从客户机上直接调用 DBMS 服务器，是 Intranet 访问的一个很实用的解决方法。与其他类型的驱动程序相比，这种驱动的优点在于它的性能，在客户机和数据库之间没有任何本地代码或中间件，所以是 JDBC 访问数据库的首选方法。  本书为简单起见，所使用的 JDBC 驱动程序为 JDBC−ODBC 桥。  **【学生】**思考、讨论。 | **展示连接数据库（一），让学生更加仔细的阅读，从而激发学生的学习欲望。** |
| **作业布置**（3min） | **【教师】**布置课后作业  简述JDBC 的功能。 | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**  （40min） | **【教师】**展示连接数据库（二）  **二、连接数据库**  Java 通过 JDBC 连接和管理各种类型的数据库。要访问数据库，必须先配置好ODBC 数据源，并在程序中注册 JDBC 驱动程序，才可以使用 Java 连接并访问数据库。在本任务所涉及的内容中，将使用到一个 SQL Server 数据库 Book，在这个数据库中有一个名为 bookinfo 的表。  使用 SQL Server 企业管理器建立好数据库和相应的表后，在其中添加一些数据，以方便我们后续的试验。关于在 SQL Server 中建立数据库和表的操作，请参见有关书籍。数据库建立完成后，还需要建立 JDBC 可以访问的 ODBC 数据源。  （1）打开“控制面板”→“管理工具”→“数据源（ODBC）”，并选择“系统 DSN”选项卡，如图 9−2 所示。  （2）单击“添加”按钮，选择 SQL Server 为数据源驱动程序类型，如图 9−3 所示。    （3）输入要使用的数据源的名称和描述，并选择所在的 SQL Server 服务器，如图 9−4 所示。  （4）连续单击两次“下一步”按钮后，出现如图 9−5 所示的对话框。在这里选择新创建的数据库 Book 作为默认数据库。    （5）依次单击“下一步”和“完成”按钮后，出现如图 9−6 所示的对话框。到此，ODBC 数据源设置完成，可以使用“测试数据源”按钮对其进行测试。    **三、加载 JDBC 驱动**  加载 JDBC 驱动程序的方法有两种：  （1）通过 Class.forName 方法，显式地加载驱动程序类。由于这与外部设备无关，因此我们推荐使用这种加载驱动程序的方法。  下列代码用于加载 JDBC−ODBC 桥驱动程序：  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  （2）通过 java.lang.System 的属性将驱动程序添加到 jdbc.driver 中。这是一个由DriverManager 类加载的驱动程序类名的列表，由冒号分隔。初始化 DriverManager 类时，它搜索系统属性 jdbc.driver，如果用户已经输入了一个或多个驱动程序，则DriverManager 将尝试加载它们。这两种方法中，新加载的 Driver 类都要通过调用DriverManager.registerDriver 类进行注册，而调用 Class.forName 方法显式地加载每个驱动程序显然更为安全。  **四、建立数据库连接**  加载 Driver 类并在 DriverManager 类中注册后，就可以建立与数据库的连接了。当调用 DriverManager.getConnection 方法发出连接请求时，DriverManager 将检查每个驱动程序，查看它是否可以建立连接。  下面是使用 JDBC−ODBC 桥驱动程序，建立数据库连接的例子：  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // 加载驱动程序  String url = "jdbc:odbc:databasename"; // 数据库连接字符串  String user = ""; // 用户名  String password = ""; // 密码  DriverManager.getConnection(url,user,password); // 连接数据库  **任务实施**  /\* 源程序文件名：Ex0901.java\*/  import java.sql.\*;  public class Ex0901  {  public static void main(String args[])  {  String driver = "sun.jdbc.odbc.JdbcOdbcDriver";  String url = "jdbc:odbc:book\_shop";  String user = "";  String password = "";  try  {  Class.forName(driver);  }  catch(Exception e)  {  System.out.println(" 无法加载驱动程序："+driver);  e.printStackTrace();  }  try  {  Connection con = DriverManager.getConnection(url,user,password);  if(!con.isClosed())  System.out.println(" 数据库连接成功！ ");  con.close();  }  catch(SQLException ee)  {  System.out.println(" 数据库连接失败！ ");  ee.printStackTrace();  }  }  }  在前面的代码中，使用了两组 try...catch 语句。第一组 try...catch 语句是判断 JDBC驱动是否成功加载。我们使用 Class.forName（驱动名）方法加载驱动，如果驱动名有误，或出现其他不能加载驱动的情况，该方法就会抛出一个异常。第二组 try...catch 语句用于跟踪和判断在数据库连接和操作的过程中，是否出现数据库无法连接、SQL 语句错误等情况，其后的 catch 语句捕获数据库操作中可能出现的 SQLException 类异常，并使用printStackTrace( ) 方法输出异常的具体提示，程序员可根据提示判断错误是源自 Java 程序中还是 SQL 语句中，并进行相应的修改。  **【学生】**思考、讨论。 | **通过教师讲解，了解连接数据库（二）的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了连接数据库（二），让学生知道加载 JDBC 驱动程序的方法。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | **【教师】**布置课后作业  论述加载 JDBC 驱动。 | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**展示访问记录（一）  任务描述  编写一个程序，对数据库中的记录进行修改和查询操作。  任务目标  1. 能够掌握 Java 中对数据库记录的基本操作方法。  2. 能够掌握 Java 中与数据库处理相关的类 Statement、PreparedStatement、ResultSet  的使用。  3. 学会利用“宝库”，做一个知识“大富翁”。成为一个对社会、对国家有用的人。  预备知识  在连接到数据库之后，可以使用多种方法向数据库发送 SQL 语句，并接收其返回  结果，以完成数据的增加、删除、修改和查询操作。  一、增加、删除和修改记录  增加、删除和修改记录这三个数据库基本操作有一个共同点，就是这些操作都是向  数据库发送一个 SQL 语句就可以了，几乎不用理会数据库返回的结果。在这些操作中，  我们通常使用 Statement 类及其子类 PreparedStatement。  （一）Statement 类  Statement 类用于将 SQL 语句发送到数据库中。实际上有三种 Statement 对象：Statement、PreparedStatement（继承自 Statement）和 CallableStatement（继承自 PreparedStatement），  它们都专用于发送特定类型的 SQL 语句。Statement 对象用于执行不带参数的简单 SQL  语句；PreparedStatement 对象用于执行带或不带 IN 参数的预编译 SQL 语句；CallableStatement 对象用于执行对数据库已存储过程的调用。  1. 创建 Statement 对象  建立了到数据库的连接之后，就可以使用该连接发送 SQL 语句。Statement 对象用  Connection 的 createStatement 方法创建。如下列代码所示：  Connection con = DriverManager.getConnection (url,user,password);  Statement smt = con.createStatement();  2. 使用 Statement 对象执行语句  Statement 提供了三种执行 SQL 语句的方法：executeQuery、executeUpdate 和 execute。  executeQuery 方法用于产生单个结果集的语句。如 SELECT 语句。  executeUpdate 方法用于执行 INSERT、UPDATE 或 DELETE 语句以及 SQL DDL 语  句，例如 CREATE TABLE 和 DROP TABLE。INSERT、UPDATE 和 DELETE 语句的效  果是修改表中零行或多行中的一列或多列。它的返回值是一个整数，表示受到影响的行  数，即更新计数。对于 CREATE TABLE 或 DROP TABLE 等不操作行的语句，该方法  的返回值总为零。  execute 方法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。  3. 语句完成  当连接处于自动提交模式时，其中所执行的语句在完成时将自动提交或还原。语句  在已执行且所有结果返回时，即认为已完成。对于返回一个结果集的 executeQuery 方法，在检索完 ResultSet 对象的所有行时该语句完成。对于 executeUpdate 方法，当它执  行时语句即完成。但在少数调用 execute 方法的情况下，可能在检索所有结果集或它生  成的更新计数之后，语句才完成。  4. 关闭 Statement 对象  Statement 对象可以由 Java 垃圾收集程序自动关闭。但作为一种良好的编程习惯，应  当在不需要 Statement 对象时调用 close 方法显式地关闭它们。这样可以立即释放 DBMS  资源，有助于避免潜在的问题。  下面的代码是用于向数据库中插入一条记录：  Connection con = DriverManager.getConnection (url,user,password);  Statement smt = con.createStatement();  smt.executeUpdate("INSERT into bookinfo (no,name,price,editor,publish  com,publishdate,quantiry)"+"values ('7040125361',' 计算机文化基础 ',26,' 李富  贵 ',' 清华大学出版社 ','2002-01-01',50)");System.out.println(" 记录插入完毕！ ");  smt.close(); con.close();  （二）PreparedStatement 类  PreparedStatement 继承自 Statement，在 PreparedStatement 对象中包含已经经过预编  译处理的 SQL 语句，这些 SQL 语句可以具有一个或多个 IN 参数。在创建 SQL 语句时，  IN 参数的值并未被指定，而是用问号“？”为每个 IN 参数保留一个占位符。在执行该  语句之前，必须通过适当的 setXXX 方法为每个问号提供具体的值。  由于 PreparedStatement 对象已经经过预编译处理，因此其执行速度要快于 Statement  对象。所以，通常把需要多次执行的 SQL 语句创建为 PreparedStatement 对象，以便提  高该语句的执行效率。  作为 Statement 的子类，PreparedStatement 在继承了 Statement 的所有功能的同时，  还添加了一系列与参数相关的方法，这些方法主要用于设置发送给数据库以取代 IN 参  数占位符的具体值。  1. 创建 PreparedStatement 对象  下面代码创建了包含两个 IN 参数占位符的 SQL 语句的 PreparedStatement 对象（其  中 con 是 Connection 对象）：  PreparedStatement psmt = con.prepareStatement("UPDATE tablename SET m  = ? WHERE x = ?");  其中包含的 SQL 语句“UPDATE tablename SET m= ? WHERE x = ?”，在对象创建  完毕后，就已经发送到了 DBMS，并做好了执行准备。  2. 传递 IN 参数  在执行 PreparedStatement 对象之前，通常需要调用 setXXX 方法来完成对每个“？”  参数的设置。其中 XXX 是与该参数相应的类型。例如，如果一个“？”参数具有 Java  类型 long，则使用的方法就是 setLong。setXXX 方法有两个参数，其中第一个参数是要设置的“？”的序数位置，第二个参数是设置给“？”的值。以下代码演示了如何设置两  个“？”参数占位符的值并执行 psmt 10 次：  psmt.setString(1,"Hi");  for(int i = 0; i<10; i++)  {  psmt.setInt(2,i);  int rowCount = psmt.executeUpdate();  }  在这段代码中，第一个“？”参数占位符被设置为“Hi”并保持，在 for 循环中，每  次都将第二个“？”参数设置为不同的值，从 0 开始，到 9 结束。  3. 参数中数据类型的一致性  setXXX 方法中的 XXX 是 Java 的类型。在向 DBMS 发送参数时，JDBC 驱动程序  会把 Java 的 XXX 类型映射为 JDBC 对应的数据类型，并将该 JDBC 类型发送给数据库。  例如下面这一条语句将 PreparedStatement 对象 psmt 的第二个参数设置为 44，Java 类型  为 short：  psmt.setShort(2,44);  这条语句中 44 是 Java 的 short 类型，JDBC 驱动程序将 44 映射为 JDBC 的 SMALLINT  类型并发送给数据库。  程序员的责任是确保将每个 IN 参数的 Java 数据类型映射为数据库所需的或兼  容的 JDBC 数据类型。JDBC 驱动程序承诺一定的兼容性，例如数据库需要 JDBC 的  SMALLINT 的数据类型，如果使用 setByte 方法，则 JDBC 驱动程序会将 JDBC 的  TINYINT 发送给数据库。上述这种方法是可行的，因为许多数据库可从一种相关的类型  转换为另一种类型，并且通常 TINYINT 可用于 SMALLINT 适用的任何地方。通常，对  于要适用于尽可能多的数据库的应用程序，则使用与数据库所需的确切的 JDBC 数据类  型相应的 Java 数据类型。比如，所需的 JDBC 类型是 SMALLINT，则使用 setShort 代替  setByte 将使应用程序的可移植性更好。  **【学生】**思考、讨论。 | **教师通过访问记录（一）展示，让学生了解访问记录（一）的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了访问记录（一），学会利用“宝库”，做一个知识“大富翁”。成为一个对社会、对国家有用的人。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述参数中数据类型的一致性。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**展示访问记录（二）  **二、查询记录**  查询操作的最大特点是可能返回相当多的合乎要求的记录，这些记录需要用一种特殊的对象接收并为后续操作做好准备，这就是 ResultSet 对象。  ResultSet 类是结果集，用于保存 SQL 语句执行结果中的所有行，并且通过一套getXXX 方法对这些行中不同列数据进行访问。同时使用的 next 方法可用于移动到ResultSet 中的下一行，使下一行成为当前行。  **（一）行和游标**  ResultSet 对指向其当前数据行的游标进行维护。通过调用 next 方法，每调用一次next 方法，游标向下移动一行。最初它位于所有行之前，因此第一次调用 next 将把游标置于第一行上，使它成为当前行。随着每次调用 next 方法，将按照从上至下的顺序获取ResultSet 中的所有行。在 ResultSet 对象或其父类 Statement 对象关闭之前，游标一直保持有效。  **（二）列**  getXXX 方法提供了获取当前行中某列值的途径。在每一行内，可按任何次序获取列值。但为了保证可移植性，应该从左至右获取列值，并且一次性地读取列值。列名或列号可用于标识要从中获取数据的列。例如：如果 ResultSet 对象 rs 的第二列名为“title”，并将值存储为字符串，则下列两行代码中的任意一行都可以获取存储在该列中的值：  String s = rs.getString("title");  String s = rs.getString(2);  注意：列是从左至右编号的，并且从列 1 开始。同时，用作 getXXX 方法的输入的列名不区分大小写。  **（三）数据类型和转换**  对于 getXXX 方法，JDBC 驱动程序试图将基本数据类型转换成指定的 Java 类型，然后返回适合的 Java 值。例如：如果方法为 getString，而数据库中数据类型为 VARCHAR，则 JDBC 驱动程序将把 VARCHAR 转换成 Java 的 String。getString 的返回值将为 String 对象。  **（四）流的使用**  ResultSet 可以获取任意大的 LONGVARBINARY 或 LONGVARCHAR 数据。方法getByte 和 getString 将数据返回为较大的数据块（最大为 Statement.getMaxFieldSize 的返回值）。但是，以固定块获取较多的数据可能会更方便、安全，可以通过返回 ResultSet 类的 java.io.inputStream 流来完成。从该流中可分块读取数据。  注意：必须立即访问这些数据，因为在下一次对 ResultSet 调用 getXXX 时，它们将自动关闭。JDBC 具有三个获取流的方法，分别具有不同的返回值：getBinaryStream 返回只提供数据库原字节而不进行任何转换的流；getAsciiStream 返回提供单字节 ASCII 字符的流；getUnicodeStream 返回提供双字节 Unicode 字符的流。  **（五）NULL 结果值**  要确定给定结果值是否是 JDBC 的 NULL，必须先读取该列，然后使用 ResultSet.wasNull( ) 方法检查该次读取是否返回 NULL。当使用 getXXX 方法读取时，方法 wasNull 将返回不同的 NULL 值。Java 的 null 值，对应于返回 Java 对象的 getXXX 方法（如 getString、getBigDecimal、getBytes、getDate、getTime、getTimestamp、getAsciiStream、getUnicodeStream、getBinaryStream、getObject 等）； 零 值， 对 应 于 getByte、getShort、getInt、getLong、getFloat、getDouble 等方法；false 值，对应于 getBoolean 方法。同样，大家可以结合任务下面的相应代码，熟悉对 ResultSet 对象的基本操作。  **任务实施**  /\* 源程序文件名：Ex0902.java\*/  import java.sql.\*;  public class Ex0902  {  public static void main(String args[])  {  String driver = "sun.jdbc.odbc.JdbcOdbcDriver";  String url = "jdbc:odbc:book\_shop";  String user = "";  String password = "";  try  {  Class.forName(driver);  }  catch(Exception e)  {  System.out.println(" 无法加载驱动程序！ "+driver);  e.printStackTrace();  }  String no[] = {"7040139037","7040153260"};  String com[] = {" 北京大学出版社 "," 人民大学出版社 "};  try  {  Connection con = DriverManager.getConnection (url,user,password);  PreparedStatement psmt = con.prepareStatement ("UPDATE bookinfo SET  publishcom = ? WHERE no = ?");  for(int i = 0; i<no.length; i++)  {  psmt.setString(1,com[i]);  psmt.setString(2,no[i]);  psmt.executeUpdate();  }  psmt.close();  Statement smt = con.createStatement();  ResultSet rst = smt.executeQuery("SELECT \* FROM bookinfo");  System.out.println(" 修改后 bookinfo 表中的记录：");  System.out.println(" 书号 \t\t 书名 \t\t\t 出版社 ");  while(rst.next())  {  System.out.println(rst.getString(1)+"\t"+rst.getString(2)+"\t"+rst.  getString(5));  **【学生】**思考、讨论。 | **教师通过访问记录（二）展示，让学生了解访问记录（二）的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了访问记录（一），了解查询操作的最大特点是可能返回相当多的合乎要求的记录。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述NULL 结果值。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**展示思考与练习  **一、简答题**  （1）JDBC 驱动程序有几类？各自的特点是什么？  （2）Statement 类提供的三种执行 SQL 语句的方法分别适用于何种情况？  （3）使用 PreparedStatement 类的 setXXX 方法和 ResultSet 类的 getXXX 方法时，如何保证数据类型的一致性？  **二、选择题**  （1）Java 连接数据库时采用的方法为（　　）。  A. ODBC B. JDBC C. ACCESS D. SQL  （2）JDBC 的作用不包括（　　）。  A. 与一个数据库建立连接  B. 向数据库发送 SQL 语句  C. 处理数据库返回结果  D. 创建数据库  （3）JDBC 应用程序接口不包括（　　）。  A. DriverManger B. Connection  C. Exception D. Statement  **实训任务**  （1）编写程序：删除 bookinfo 表中所有出版社为中国矿业出版社的记录。  （2）编写程序：输出 bookinfo 表中所有价格高于 20 的书籍的书名、作者、单价和出版社。  **【学生】**思考、讨论。 | **教师通过思考与练习展示，让学生了解思考与练习的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了思考与练习，让学生可以巩固学习知识，增强实践能力。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **使用 JDBC 访问数据库大致可分为几步？** | 通过课后练习，使学生巩固所学新知识 |
| **教学反思** | 要尽量多留一些时间给学生练习。计算机教学最后要达到的目的就是让学生能熟练的进行上机操作，所以在教学中要多留出一些时间给学生进行上机操作练习，即所谓的“百讲不如一练”。 | |