内容提要

- JDBC
- JDBC API
- JDBC API的体系结构
- JDBC API的任务
- JDBC API中的重要接口和类
- JDBC程序开发步骤
- 通过JDBC访问数据库示例

JDBC（Java DataBase Connectivity）

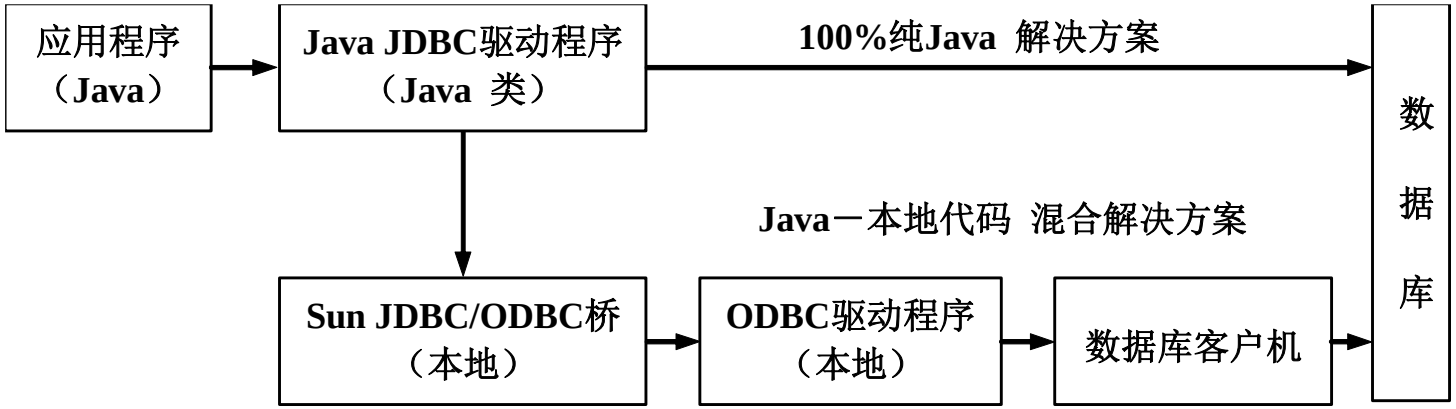
- 是用于执行SQL语句的Java应用程序接口，由一组用Java语言编写的类与接口组成，是一种底层API
- 使开发人员可以用纯Java语言编写完整的数据库应用程序
- 用JDBC写的程序能够自动地将SQL语句传送给几乎任何一种数据库管理系统（DBMS）

JDBC（Java DataBase Connectivity）

- 是一种规范，它让各数据库厂商为Java程序员提供标准的数据库访问类和接口，这样就使得独立于DBMS的Java应用开发工具和产品成为可能
- 隔离了Java与不同数据库之间的对话，使得程序员只须写一遍程序就可让它在任何数据库管理系统平台上运行
- 使用已有的SQL标准，并支持其它数据库连接标准，如与ODBC之间的桥接（遗憾的是，JDK1.8及未来版本不再包含JDBC-ODBC桥！！！）

JDBC（Java DataBase Connectivity）

- Java程序通过JDBC访问数据库的不同方案



JDBC（Java DataBase Connectivity）

- ODBC（Open Database Connectivity）
  - 由微软公司提出，用于在DBMS中存取数据
  - 是一套用C语言实现的访问数据库的API
  - 通过ODBC API，应用程序可以存取保存在多种不同DBMS中的数据，而不论每个DBMS使用了何种数据存储格式和编程接口
  - 对于没有提供JDBC驱动的数据库，从Java程序调用本地的C程序访问数据库会带来一系列安全性、完整性、健壮性等方面的问题，因而通过JDBC-ODBC桥来访问没有提供JDBC接口的数据库是一个常用的方案

JDBC（Java DataBase Connectivity）

- ODBC的结构
  - 应用程序(Application)：本身不直接与数据库打交道，主要负责处理并调用ODBC函数，发送对数据库的SQL请求及取得结果
  - 驱动器管理器(ODBC manager)：为应用程序装载数据库驱动器
  - 数据库驱动器(ODBC Drivers)：实现ODBC的函数调用，提供对特定数据源的SQL请求。
  - 数据源(Data Sources,数据库)：由用户想要存取的数据以及与它相关的操作系统、DBMS和用于访问DBMS的网络平台组成。
- 通过ODBC访问数据库的模式

  你的程序<—> ODBC管理器<—> ODBC驱动程序 <—> 数据库

JDBC（Java DataBase Connectivity）

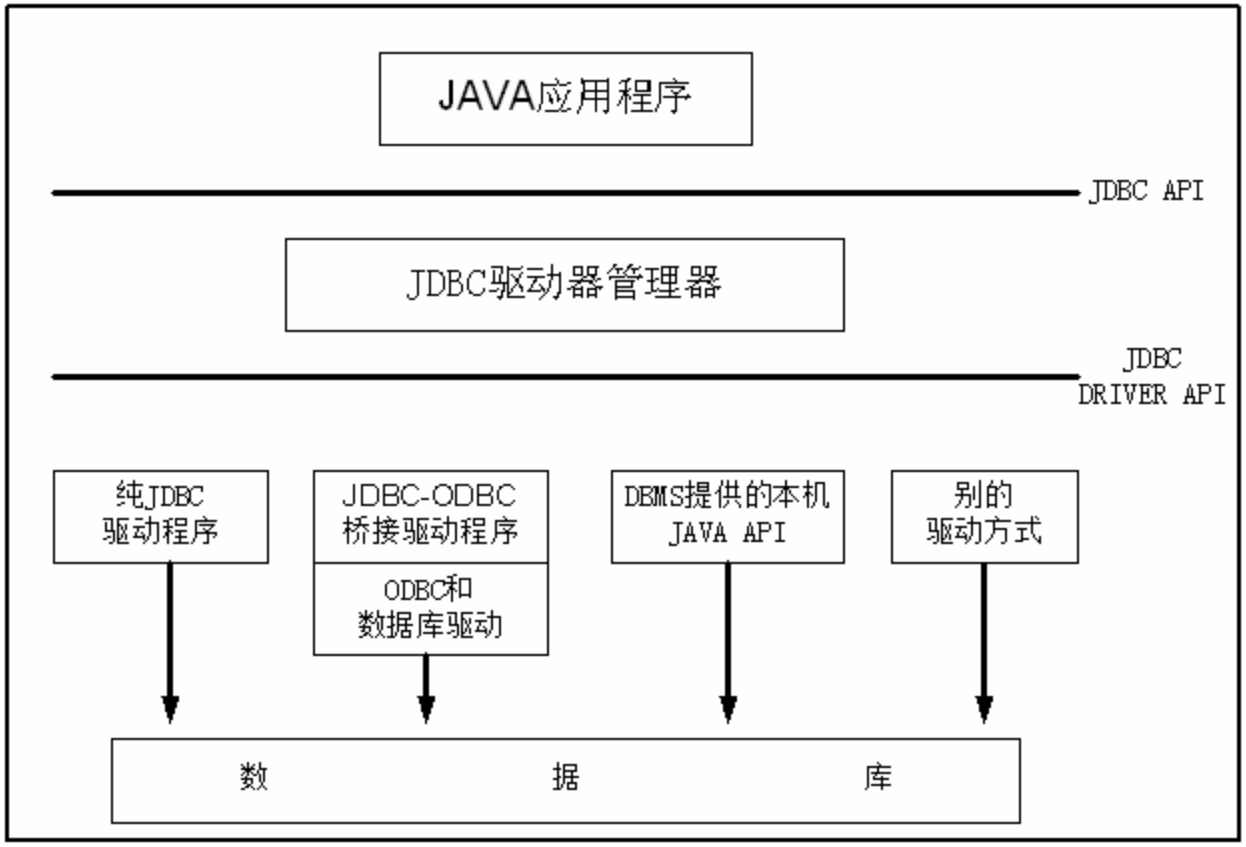
- ODBC的不足
  - 是一个C语言实现的API，并不适合在Java中直接使用。从Java程序调用本地的C程序会带来一系列类似安全性、完整性、健壮性的缺点
  - 完全精确地实现从C代码ODBC到Java API写的ODBC的翻译也并不令人满意。比如，Java没有指针，而ODBC中大量地使用了指针，包括极易出错的无类型指针“void＊”。
  - ODBC并不容易学习，它将简单特性和复杂特性混杂在一起，甚至对非常简单的查询都有复杂的选项。而JDBC刚好相反，它保持了简单事物的简单性，但又允许复杂的特性

JDBC API

- 是一组由Java语言编写的类和接口，分布在java.sql和javax.sql两个包中
  - *java.sql* 为核心包，其包含于J2SE中
  - *javax.sql* 包扩展了JDBC API的功能，成为了J2EE的一个基本组成部分
- 可分为两个层次
  - 面向底层的JDBC Driver API
    主要是针对数据库厂商开发数据库底层驱动程序使用
  - 面向程序员的JDBC API

JDBC API的体系结构

- Java应用程序通过JDBC API和底层的JDBC Driver API打交道



JDBC API的任务

- 面向程序员的JDBC API可以完成以下主要任务
  - 首先建立和数据源的连接
  - 然后向其传送查询和修改等SQL命令
  - 最后处理数据源返回的SQL执行的结果

JDBC API中的重要接口和类

名称	说明
java.sql.DriverManager 类	处理数据库JDBC驱动的调入并且对产生新的数据库连接提供支持
javax.sql.DataSource 接口	在JDBC 2.0 API中被推荐使用代替DriverManager实现和数据库的连接
java.sql.Connection 接口	代表对特定数据库的连接
java.sql.Statement 接口	代表一个特定的容器，容纳并执行一条SQL语句，返回它所生成结果的对象
java.sql.ResultSet 接口	控制执行查询语句得到的结果集。 ResultSet 对象具有指向其当前数据行的游标。最初，游标被置于第一行之前。next 方法将光标移动到下一行；该方法在 ResultSet 对象没有下一行时返回 false。 默认的 ResultSet 对象不可更新，仅有一个向前移动的游标。可以生成可滚动和/或可更新的 ResultSet 对象。

JDBC程序开发步骤

- 一个基本的JDBC程序开发包含如下步骤
  - 设置环境，引入相应的JDBC类
  - 选择合适的JDBC驱动程序并加载
  - 创建一个Connection对象
  - 创建一个Statement对象
  - 用该Statement对象进行查询等操作
  - 从返回的ResultSet对象中获取相应的数据
  - 关闭Connection

JDBC程序开发步骤：设置环境

- 在本机上安装有关数据库软件
- 下载相应数据库驱动程序并安装
- 在Java程序中引入相应的类和包。任何使用JDBC的源程序都需要引入java.sql包，如必要的时候还需要装载相应的JDBC-ODBC驱动程序的包

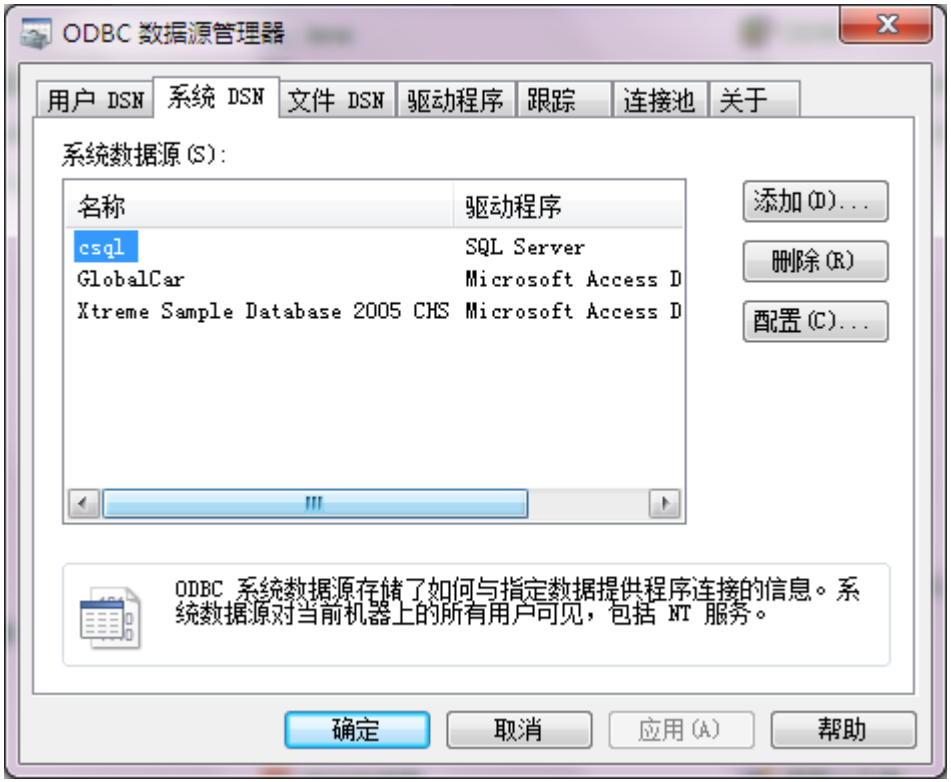
```
import java.sql.*;
import sun.jdbc.odbc.JdbcOdbcDriver;
```

JDBC程序开发步骤：设置环境

- 如果通过JDBC-ODBC桥访问数据库
  - 安装JDK的同时就自动安装了安装JDBC-ODBC桥驱动程序（JDK1.8及未来版本不再包含JDBC-ODBC桥！！！）
  - 安装 DBMS，并建立一个数据库
  - 在ODBC数据源管理器中注册数据源

JDBC程序开发步骤：设置环境

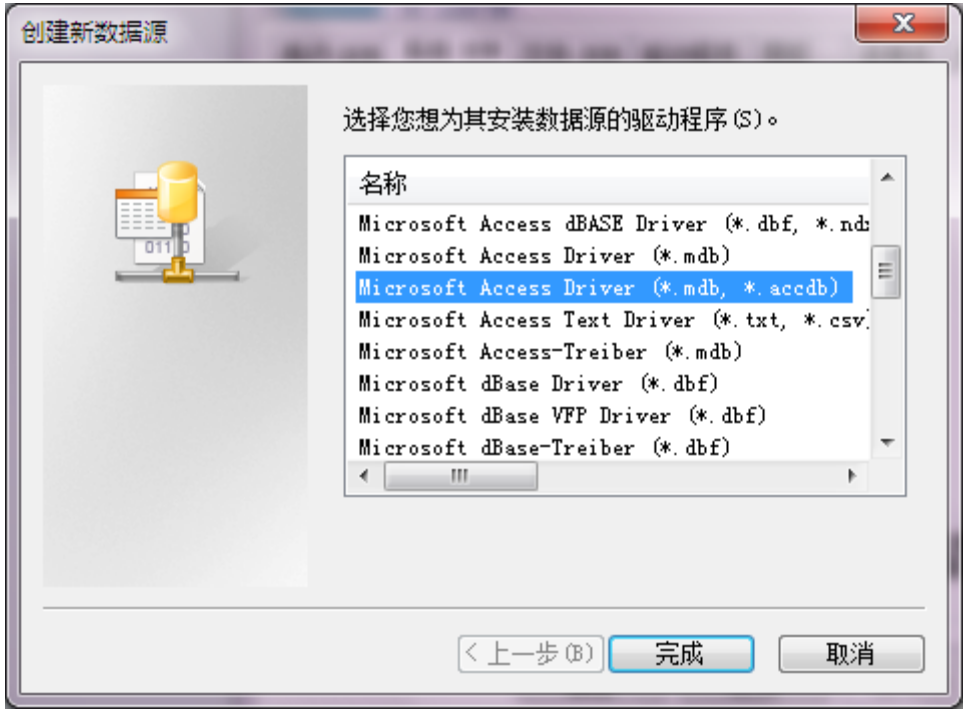
- 示例：在ODBC数据源管理器中注册一个Access数据源



控制面板-->ODBC-->系统DSN-->添加

JDBC程序开发步骤：设置环境

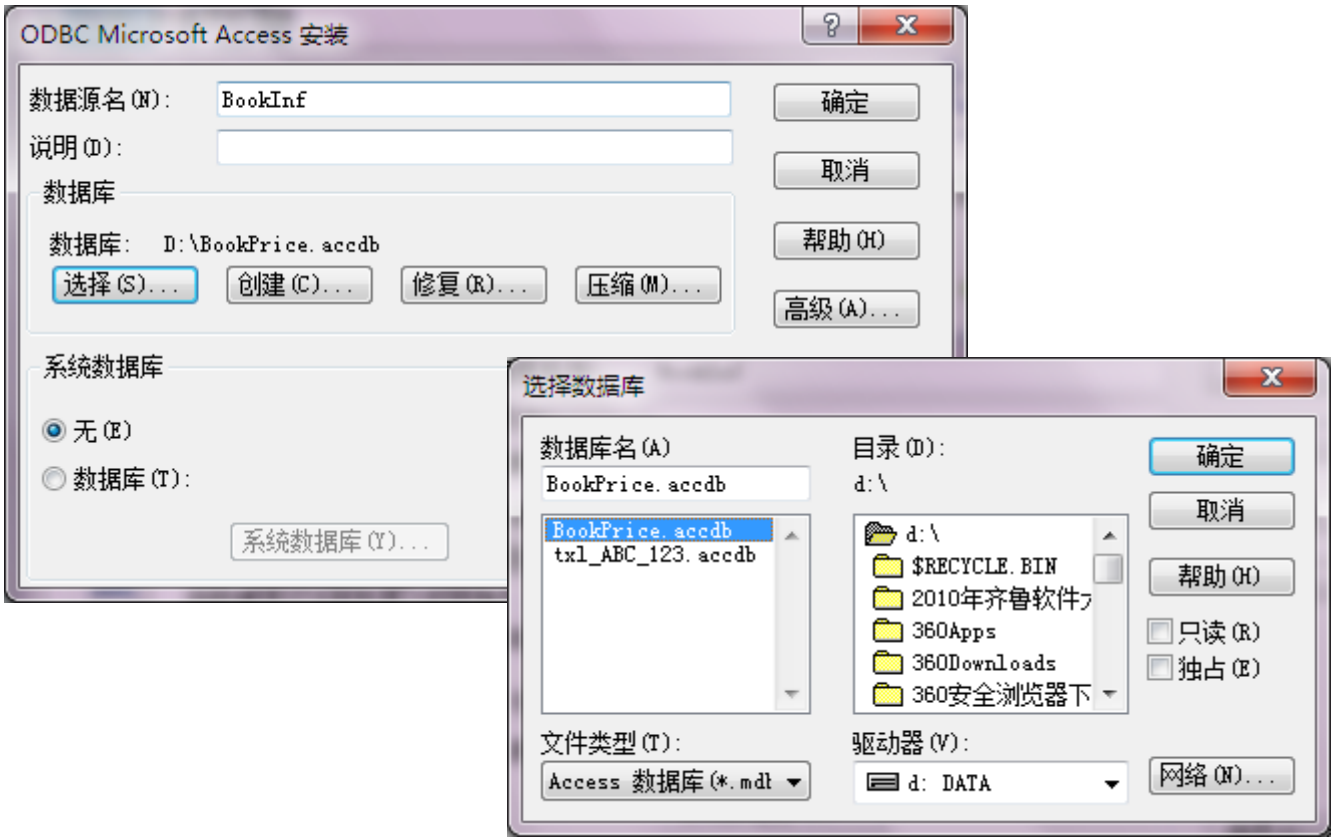
- 示例：在ODBC数据源管理器中注册一个Access数据源



选择“Microsoft AccessDriver (*.mdb, *.accdb)”选项，单击“完成”按钮

JDBC程序开发步骤：设置环境

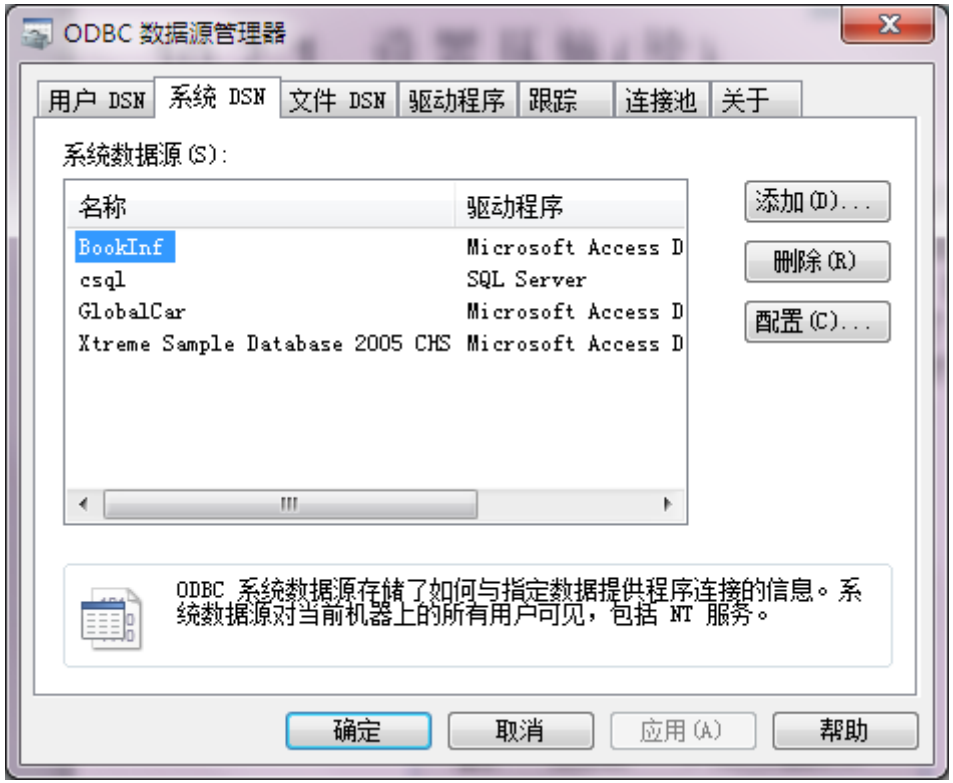
- 示例：在ODBC数据源管理器中注册一个Access数据源



输入数据源名"BookInf"-->选择数据库"d:\BookPrice.accdb"

JDBC程序开发步骤：设置环境

- 示例：在ODBC数据源管理器中注册一个Access数据源



依次确定返回到“ODBC 数据源管理器”对话框，这时可以发现“系统DSN”选项中增添了一个新确定的数据源“BookInf”。单击“确定”按钮，就完成了数据源的注册

JDBC程序开发步骤：装载驱动程序

- 用Class.forName方法显式装载驱动程序，如：
  - Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  - Class.forName("com.hxtt.sql.access.AccessDriver");
  - 以完整的Java类名字符串(可以从驱动程序的说明文档中得到)为参数，装载此类，并返回一个Class对象描述此类
  - 执行上述代码时将自动创建一个驱动器类的实例，并自动调用驱动器管理器DriverManager类中的RegisterDriver方法来注册它
  - 如果驱动器程序不存在，该方法就会抛出ClassNotFoundException异常，因此需要捕获它：

```
try{  Class.forName("驱动程序类名称");  }
catch(ClassNotFoundException e){System.out.println(e.getMessage);}
```

JDBC程序开发步骤：建立连接

- 调用DriverManager.getConnection()方法来建立与数据库的连接。例如：
  - String url = "jdbc:odbc:BookInf";
  - String url = "jdbc:Access:/D:/BookPrice.accdb";
  - Connection con = DriverManager.getConnection(url, "", "");
  - 将返回与指定数据库建立的连接
  - 该方法有三个字符串参数
    - 第1个是JDBC URL，格式为 jdbc:子协议:子名称
      - Jdbc表示协议，JDBC URL 中的协议总是jdbc；
      - 子协议是驱动程序名称；
      - 子名称是数据库的名称，若数据库位于远程服务器上，则还应该包括网络地址，//主机名或IP地址:端口/数据库名
    - 第2个是访问数据库所需的用户名
    - 第3个是用户密码

JDBC程序开发步骤：建立连接

- 调用DriverManager.getConnection()方法来建立与数据库的连接。例如：
  - String url = "jdbc:odbc:BookInf";
  - String url = "jdbc:Access:/D:/BookPrice.accdb";
  - Connection con = DriverManager.getConnection(url, "", "");
  - Connection是一个接口，表示与指定数据库的连接
  - DriverManager类位于JDBC的管理层，作用于用户和驱动程序之间。它负责跟踪在一个系统中所有可用的JDBC驱动程序，并在数据库和相应驱动程序之间建立连接

JDBC程序开发步骤：操作数据库

- 建立好到数据库的连接后，就可以进行对数据库的操作了，一般包括如下三个步骤
  - 使用Connection对象创建Statement对象
  - 使用Statement对象执行SQL命令
  - 从上一步骤返回的ResultSet对象中提取执行结果

操作数据库：创建Statement对象

- Connection接口有3个方法可用来创建向数据库发送SQL语句的对象
  - createStatement方法：创建向数据库发送SQL语句的Statement对象，用于简单的SQL语句。例如：

```
Statement stmt = con.createStatement();
Statement stmt = con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE)
```

  - prepareStatement方法：创建向数据库发送SQL语句的PreparedStatement对象，用于执行带有一个或多个参数的SQL语句。在SQL语句执行前，参数应被赋值

```
sql = "INSERT INTO BookInf(BookName, Price) VALUES(?,?)";
PreparedStatement pstmt = con.prepareStatement(sql);
// 指定SQL语句中的参数值
pstmt.setString(1, "解析几何2"); //第1个参数值
pstmt.setDouble(2, 39.5); // 第2个参数值
```

  - prepareCall方法：创建向数据库发送SQL语句的CallableStatement对象，用于调用数据库中的存储过程

操作数据库：使用Statement对象执行SQL语句

- Statement 接口提供了三种执行 SQL 语句的方法，使用哪一个方法由 SQL 语句所产生的内容决定
  - **executeQuery方法**：用于执行产生单个结果集的语句，例如 SELECT 语句

```
ResultSet rs = stmt.executeQuery("Select * From BookInf");
```

  - **executeUpdate方法**：用于执行 INSERT、UPDATE 或 DELETE 语句，以及 CREATE TABLE

```
stmt.executeUpdate("DELETE FROM BookInf WHERE BookName ='高等数学'");
```

    - 返回一个整数，表示受影响的行数（即更新计数），比如修改了几行、删除了几行等。对于 CREATE TABLE 等语句，因不涉及到行的操作，所以executeUpdate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。例如执行某个存储过程，这时有可能出现多个结果的情况

操作数据库：使用PreparedStatement对象执行SQL语句

- SQL 语句可以被预编译并存储在 PreparedStatement 对象中，然后可以使用此对象多次高效地执行该语句。
- 注：用于设置 IN 参数值的设置方法（setShort、setString 等等）必须指定与输入参数的已定义 SQL 类型兼容的类型。例如，如果 IN 参数具有 SQL 类型 INTEGER，那么应该使用 setInt 方法。
- 如果需要任意参数类型转换，使用 setObject 方法时应该将目标 SQL 类型作为其参数。
- 在以下设置参数的示例中，con 表示一个活动连接：

```
sql="UPDATE BookInf SET price = ? WHERE BookName = ?"
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt. setDouble(1, 33.00)
pstmt. setString(2, "大学英语")
```

操作数据库：使用PreparedStatement对象执行SQL语句

- PreparedStatement 接口提供了三种执行参数化SQL语句的方法
  - executeUpdate方法：用于执行 INSERT、UPDATE 或 DELETE 语句，以及 CREATE TABLE

    ```
    pstmt.executeUpdate();
    ```

    - 返回一个整数，表示受影响的行数（即更新计数），比如修改了几行、删除了几行等。对于 CREATE TABLE 等语句，因不涉及到行的操作，所以executeUpdate的返回值总为零
  - executeQuery方法：用于执行产生单个结果集的参数化语句，例如 SELECT 语句

    ```
    ResultSet rs = pstmt.executeQuery();
    ```

  - 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。例如执行某个存储过程，这时有可能出现多个结果的情况

操作数据库：提取SQL语句执行结果

- 查询结果作为结果集（ResultSet）对象返回后，我们可以从ResultSet对象中提取结果
  - 使用next方法
    - ResultSet对象中含有检索出来的行，其中有一个指示器，指向当前可操作的行，初始状态下指示器是指向第一行之前
    - next方法的功能是将指示器下移一行，所以第一次调用next方法时便将指示器指向第一行，以后每一次对next的成功调用都会将指示器移向下一行

操作数据库：提取SQL语句执行结果

- 使用getXXX方法

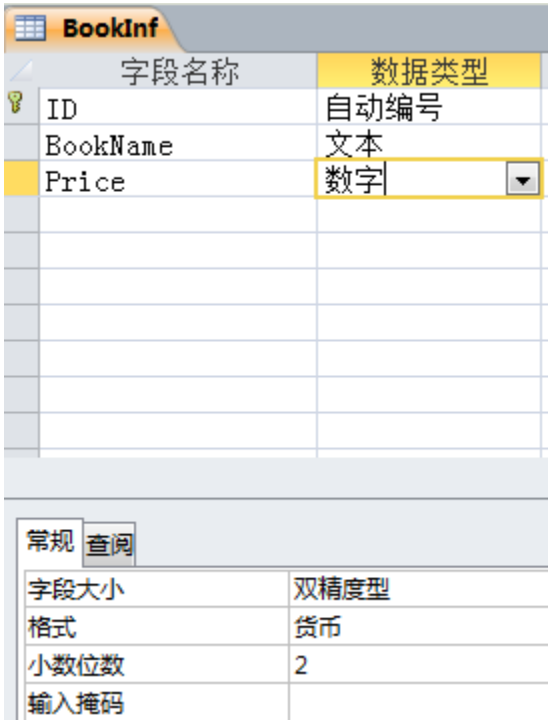
- 使用相应类型的getXXX方法可以从当前行指定列中提取不同类型的数据。例如，提取VARCHAR类型数据时就要用getString方法，而提取FLOAT类型数据的方法是getFloat
- 例如：

```
String nm= rs.getString("BookName");
float prc=rs.getFloat("Price");
```

通过JDBC访问数据库示例

【例11-1】通过JDBC访问Access2010数据库BookPrice，进行查询、添加、修改、删除等操作。

数据库中的BookInf表的结构和数据如下图所示：



BookInf

ID	BookName	Price
1	数学分析	¥34.50
2	高等代数	¥29.00
3	大学物理	¥20.00
4	数值计算方法	¥26.15
5	偏微分方程	¥27.00
*	(新建)	¥0.00

- JDK1.8及未来版本不再包含JDBC-ODBC桥
- 本例使用Access-JDBC30驱动(见Access_JDBC30.jar)，请通过项目属性->Java Build Path添加该外部jar包

HXTT MS Access JDBC Drivers：http://www.hxtt.com/access.html

```
import java.sql.*;
public class DBAccessTest {
   public static void main(String[] args) {
        String nm;
        float prc;
        String sql;
        Connection con; // 数据库连接
        Statement stmt; // Statement
        ResultSet rs; // 查询结果集
        // 加载数据库JDBC驱动
        try{
            Class.forName("com.hxtt.sql.access.AccessDriver");
        }
        catch(ClassNotFoundException e) {
            System.out.println(e.getMessage());
            return;
        }
```

```
try{
        String url = "jdbc:Access:/D:/BookPrice.accdb";
        con = DriverManager.getConnection(url, "", "");  // 连接数据库
        // 创建Statement对象
        // stmt = con.createStatement();
        stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                   ResultSet. CONCUR_UPDATABLE);
        sql = "SELECT BookName, Price FROM BookInf";
        rs = stmt.executeQuery(sql);    // 执行SQL查询
        // 显示所有记录
        System.out.println("书名\t\t\t价格(RMB)");
        while (rs.next()){
            nm= rs.getString("BookName");     prc=rs.getFloat("Price");
            String s = String.format("%1$s\t\t\t%2$.2f",nm,prc);
            System.out.println(s);
        }
        rs.close();  // 关闭结果集
```

```
// 添加记录
sql="INSERT INTO BookInf(BookName,Price)
        Values('心理学','26.35')";
stmt.executeUpdate(sql);

// 修改记录
sql="UPDATE BookInf set Price='25.84'
        WHERE BookName='大学物理'";
stmt.executeUpdate(sql);

// 删除记录
sql="DELETE FROM BookInf
        WHERE BookName='数学分析'";
stmt.executeUpdate(sql);
```

```
            // 再次显示所有记录
            System.out.println("增删改后的记录：");
            System.out.println("书名\t\t\t价格(RMB)");
            sql = "SELECT BookName, Price FROM BookInf";
            rs = stmt.executeQuery(sql);
            while (rs.next()){
                nm= rs.getString("BookName");      prc=rs.getFloat("Price");
                String s = String.format("%1$s\t\t\t%2$.2f",nm,prc);
                System.out.println(s);
             }
             rs.close(); // 关闭结果集
             // 关闭数据库资源
             stmt.close();  // 关闭Statement
             con.close();  // 关闭连接
        }
       catch(Exception e){ System.out.println(e.getMessage());  }
    }
}
```

- 运行结果

书名	价格
数学分析	RMB34.50
高等代数	RMB29.00
大学物理	RMB20.00
数值计算方法	RMB26.15
偏微分方程	RMB27.00

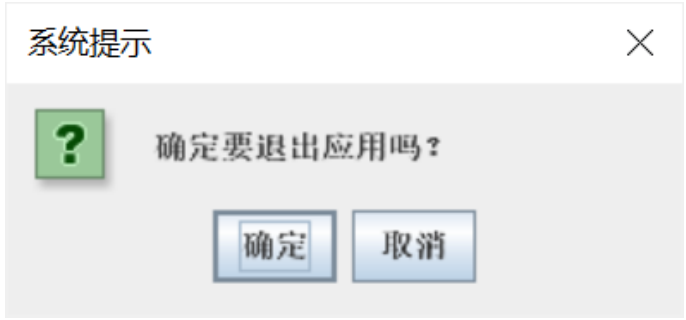
增删改后的记录:

书名	价格
心理学	RMB26.35
高等代数	RMB29.00
大学物理	RMB25.84
数值计算方法	RMB26.15
偏微分方程	RMB27.00

- 结果说明
  - 可见，的确在数据表中添加了一条记录，修改了一条记录，删除了一条记录

通过JDBC访问数据库示例

【例11-2】实现一个GUI的数据库应用程序



```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
// 窗体类
class SimpleDBMSFrm extends JFrame
                                    implements ActionListener{
   Connection con;
   Statement stmt;
   ResultSet rs;
   JButton btque,btupd,btinc,btdel,btvie,btext;
   JLabel lb1,lb2;
   JTextField tfld1,tfld2;
   JTextArea ta;
   String nm;
   float prc;
   String str1,str2;
```

```
SimpleDBMSFrm(){ // 构造方法
    super("简单数据管理管理系统");
    this.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

    JPanel topPanel=new JPanel();
    topPanel.setLayout(new GridLayout(2, 3));
    btvie=new JButton("显  示");   btque=new JButton("查  询");
    btupd=new JButton("修  改");   btinc=new JButton("添  加");
    btdel=new JButton("删  除");   btext=new JButton("退  出");
    topPanel.add(btvie);                topPanel.add(btque);
    topPanel.add(btupd);               topPanel.add(btinc);
    topPanel.add(btdel);                topPanel.add(btext);

    JPanel leftPanel=new JPanel();
    leftPanel.setLayout(new GridLayout(2, 1));
    lb1=new JLabel(" 书名：");   lb2=new JLabel(" 价格：");
    leftPanel.add(lb1);                 leftPanel.add(lb2);
```

```
JPanel centerPanel=new JPanel();
centerPanel.setLayout(new GridLayout(2, 1));
tfld1=new JTextField(15);  tfld2=new JTextField(15);
centerPanel.add(tfld1);      centerPanel.add(tfld2);

ta=new JTextArea(10,30);
JScrollPane sp=new JScrollPane(ta);

this.add(topPanel,"North");
this.add(leftPanel, "West");
this.add(centerPanel, "Center");
this.add(sp, "South");
this.pack();
this.setResizable(false); // 不许改变窗体大小
```

```
// 添加事件监听器
btvie.addActionListener(this);    btque.addActionListener(this);
 btupd.addActionListener(this);  btinc.addActionListener(this);
 btdel.addActionListener(this);    btext.addActionListener(this);

 this.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent e)
     { clearMemoryCloseWidowAndExit(e.getWindow()); }
 });
 // 加载数据库驱动程序、建立连接、创建Statement对象
 try{
            Class.forName("com.hxtt.sql.access.AccessDriver");
            String url = "jdbc:Access:/D:/BookPrice.accdb";
            con = DriverManager.getConnection(url, "", "");
            stmt = con.createStatement();
 }
 catch(Exception e){ System.out.println(e.getMessage()); }
}
```

```
// 关闭窗体的确认及数据库资源的释放
void clearMemoryCloseWidowAndExit(Window wnd){
        //参数：wnd--要关闭的窗体
        int option =JOptionPane.showConfirmDialog(wnd,
                        "确定要退出应用吗？","系统提示",
                        JOptionPane.OK_CANCEL_OPTION,
                        JOptionPane.QUESTION_MESSAGE);
        if(option!=JOptionPane.OK_OPTION) return;
        wnd.dispose(); // 销毁window
        try{
              stmt.close( ); //关闭Statement对象
              con.close( );  //关闭数据库链接
        }
        catch(Exception e){System.out.println(e.toString());}
        System.exit(0);
}
```

```
public void actionPerformed(ActionEvent e) {// 按钮动作事件处理程序
    if(e.getSource( )==btvie){  // 显示
        try{
            String sql = "SELECT BookName, Price FROM BookInf";
            rs = stmt.executeQuery(sql);
            ta.setText("书名\t\t\t价格(RMB)");
            while (rs.next( )){
                nm= rs.getString("BookName");
                prc=rs.getFloat("Price");
               String s = String.format("%1$s\t\t\t%2$.2f",nm,prc);
                ta.append(s+"\n");
            }
           rs.close( );
        }catch(Exception e1){ System.out.println(e1.getMessage()); }
    }
```

```
else if(e.getSource( )==btque){  // 查询
    try{
        int rec=0;
        String sql = "SELECT BookName, Price FROM BookInf";
        rs = stmt.executeQuery(sql);
        while (rs.next( )){
            nm= rs.getString("BookName");
            prc=rs.getFloat("Price");
            if(nm.equals(tfld1.getText( ).trim( ))){
                tfld2.setText(String.format("RMB%1$.2f",prc));
                rec=1;
                break;
             }
        }
        if(rec==0)  tfld2.setText("数据库中没有这本书");
        rs.close( );
    } catch(Exception e2){ }
}
```

```
else if(e.getSource( )==btupd){  // 修改
    try{  str1=tfld1.getText( ).trim( );      str2=tfld2.getText( ).trim( );
          String strUpd = "UPDATE BookInf SET Price = '" + str2+ "'
                                  WHERE BookName = '" + str1 + "'";
          stmt.executeUpdate(strUpd);
     } catch(Exception e3){System.out.println(e3.toString()); }
} else if(e.getSource( )==btinc){ // 添加
    try{  str1=tfld1.getText( ).trim( );      str2=tfld2.getText( ).trim( );
          String strInc = "INSERT INTO BookInf(BookName,Price)
                                  Values('"+str1+"','"+str2+"')";
          stmt.executeUpdate(strInc);
          JOptionPane.showMessageDialog(this, "添加完成！ ",
                  "好消息", JOptionPane.INFORMATION_MESSAGE);
    }catch(Exception e4){
          System.out.println(e4.toString());
          JOptionPane.showMessageDialog(this, "添加失败！ ",
                          "坏消息", JOptionPane.ERROR_MESSAGE);
    }
}
```

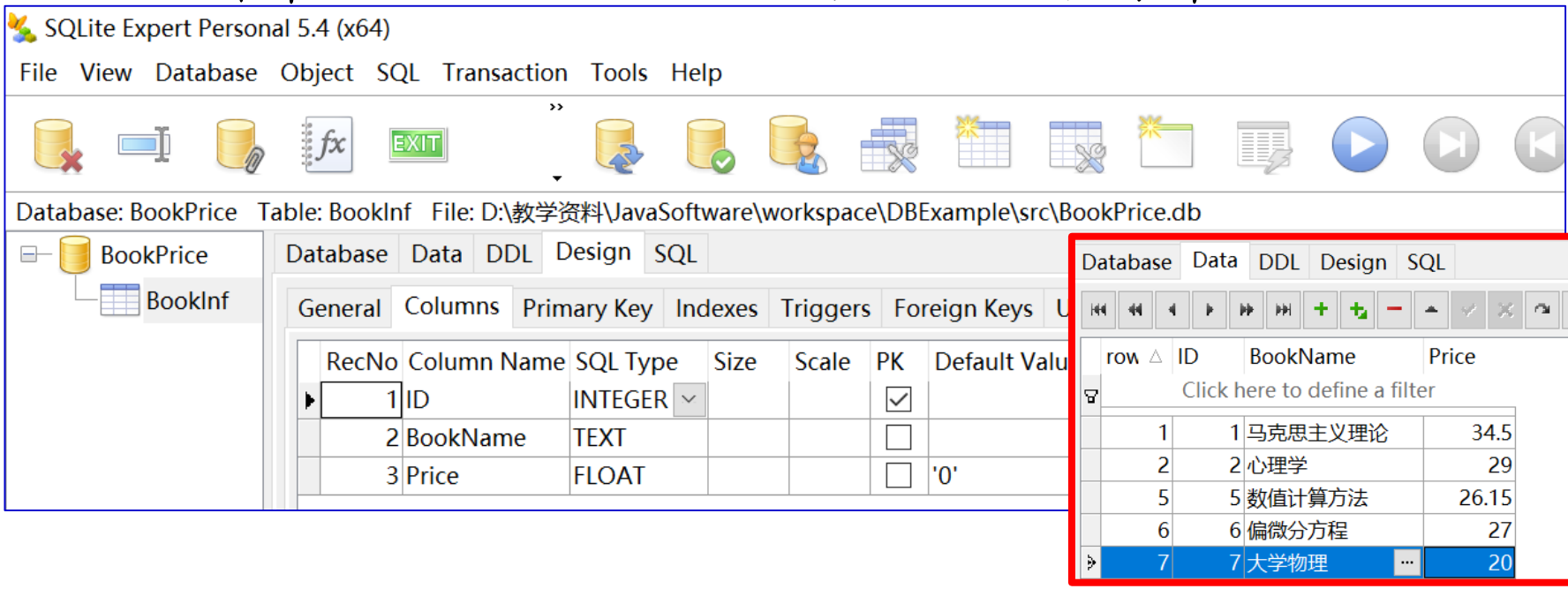
```
else if(e.getSource( )==btdel){  // 删除
    str1=tfld1.getText( ).trim( );
    if(!str1.isEmpty()) {
        int option =JOptionPane.showConfirmDialog(this,
                    "确定要删除该记录吗？","系统提示",
                     JOptionPane.YES_NO_OPTION,
                     JOptionPane.QUESTION_MESSAGE);
        if(option==JOptionPane.YES_OPTION){
            try{  String strDel = "DELETE FROM BookInf
                                  WHERE BookName ='" + str1 + "'";
                  stmt.executeUpdate(strDel);
            } catch(Exception e5){
                  System.out.println(e5.toString());
                  JOptionPane.showMessageDialog(this,
                              "删除失败！","坏消息",
                              JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}
```

```
            else if(e.getSource( )==btext){  // 退出
                clearMemoryCloseWidowAndExit(this);
            }
    }  //  public void actionPerformed(ActionEvent e)

}  // end of class SimpleDBMSFrm

// 主类
public class SimpleDBMS {
    public static void main(String[] args) {
            SimpleDBMSFrm frm=new SimpleDBMSFrm();
            frm.setVisible(true);
    }
}
```

通过JDBC访问数据库示例

【例11-3】通过JDBC访问SQLite3数据库BookPrice，进行查询、添加、修改、删除等操作。

数据库中的BookInf表的结构和数据如下图所示：



- 下载管理工具SQLite Expert Personal:
  http://www.sqliteexpert.com/download.html
- 下载sqlite-jdbc包：https://github.com/xerial/sqlite-jdbc/releases

```
import java.sql.*;

public class DBAccessTest {
   public static void main(String[] args) {
         String nm;
         float prc;
         String sql;
         Connection con; // 数据库连接
         Statement stmt; // Statement
         ResultSet rs; // 查询结果集

         // 加载数据库JDBC驱动
         try{ Class.forName("org.sqlite.JDBC");        }
         catch(ClassNotFoundException e)
         {  System.out.println(e.getMessage()); }
```

```
try{
        String url = "jdbc:sqlite:/D:/教学资料/JavaSoftware/workspace
                        /DBExample/src/BookPrice.db";
        con = DriverManager.getConnection(url, "", "");  // 连接数据库
        // 创建Statement对象
        Statement stmt = con.createStatement();
        rs = stmt.executeQuery(sql);    // 执行SQL查询
        // 显示所有记录
        while (rs.next()){
            nm= rs.getString("BookName");
            prc=rs.getFloat("Price");
            String s = String.format("书名：%1$-24s\t价格：RMB%2$.2f",nm,prc);
            System.out.println(s);
        }
        rs.close();  // 关闭结果集
```

通过JDBC访问数据库示例

【例11-4】使用PreStatement对象执行参数化SQL查询：
从Sqlite数据库BookPrice的BookInf表中查询价格在指定范围内的记录

```
String sql;
Connection con; // 数据库连接
PreparedStatement pstmt;
try{  Class.forName("org.sqlite.JDBC"); } // 加载数据库JDBC驱动
catch(ClassNotFoundException e){System.out.println(e.getMessage()); }
try{
  // 连接数据库
   String url = "jdbc:sqlite:/D:/教学资料/JavaSoftware/workspace
                                          /DBExample/src/BookPrice.db";
   con = DriverManager.getConnection(url, "", "");
```

```
    // SQL语句
    sql = "SELECT BookName, Price FROM BookInf
          WHERE Price>=? AND Price<=?"; //?代表SQL语句中的参数变量

    pstmt = con.prepareStatement(sql); // 创建PreparedStatement对象
    // 指定SQL语句中的参数值
    pstmt.setDouble(1, 30); //第1个参数值。
    pstmt.setDouble(2, 40); //第2个参数值。

    ResultSet rs=pstmt.executeQuery(); // 执行sql操作语句
    while(rs.next()){
            String bn=rs.getString("BookName");
            double bp=rs.getDouble("Price");
            System.out.println(bn+"\t"+bp);
    }
    // 关闭数据库资源
    rs.close();
    pstmt.close(); // 关闭Statement
    con.close();  // 关闭连接
} catch(Exception e){ System.out.println(e.getMessage()); }
```

通过JDBC访问数据库示例

【例11-5】使用PreStatement对象执行参数化SQL插入:
将单条记录插入Sqlite数据库BookPrice的BookInf表

```
String sql;
Connection con; // 数据库连接
PreparedStatement pstmt;
try{  Class.forName("org.sqlite.JDBC"); } // 加载数据库JDBC驱动
catch(ClassNotFoundException e){System.out.println(e.getMessage()); }
try{
  // 连接数据库
   String url = "jdbc:sqlite:/D:/教学资料/JavaSoftware/workspace
                                        /DBExample/src/BookPrice.db";
   con = DriverManager.getConnection(url, "", "");
```

```
// SQL语句，?代表SQL语句中的参数变量
sql = "INSERT INTO BookInf(BookName, Price) VALUES(?,?)";
pstmt = con.prepareStatement(sql); // 创建PreparedStatement对象
// 指定SQL语句中的参数值
pstmt.setString(1, "解析几何"); //第1个参数值
pstmt.setDouble(2, 39.5); // 第2个参数值
//在此 PreparedStatement 对象中执行 SQL 语句
int iud=pstmt.executeUpdate();
if(iud>0)
    System.out.println("记录添加成功^_^");
else
    System.out.println("记录添加失败~@~");
// 关闭数据库资源
pstmt.close(); // 关闭Statement
con.close();  // 关闭连接
}catch(Exception e){ System.out.println(e.getMessage());   }
```

通过JDBC访问数据库示例

【例11-5】使用PreStatement对象执行参数化SQL插入:
将多条记录插入Sqlite数据库BookPrice的BookInf表

```
String[] names={"Python入门","油藏模拟"};
double[] prices={24.0,21.5};
sql = "INSERT INTO BookInf(BookName, Price) VALUES(?,?)";
pstmt = con.prepareStatement(sql); // 创建PreparedStatement对象

// 循环利用pstmt添加多条记录
for(int i=0;i<names.length;i++){
        // 指定SQL语句中的参数值
        pstmt.setString(1, names[i]); //第1个参数值
        pstmt.setDouble(2, prices[i]); //第2个参数值
        int iud=pstmt.executeUpdate(); // 执行sql操作语句在此
}
```

通过JDBC访问数据库示例

【例11-6】使用PreStatement对象执行参数化SQL更新：
更新Sqlite数据库BookPrice的BookInf表中的指定记录

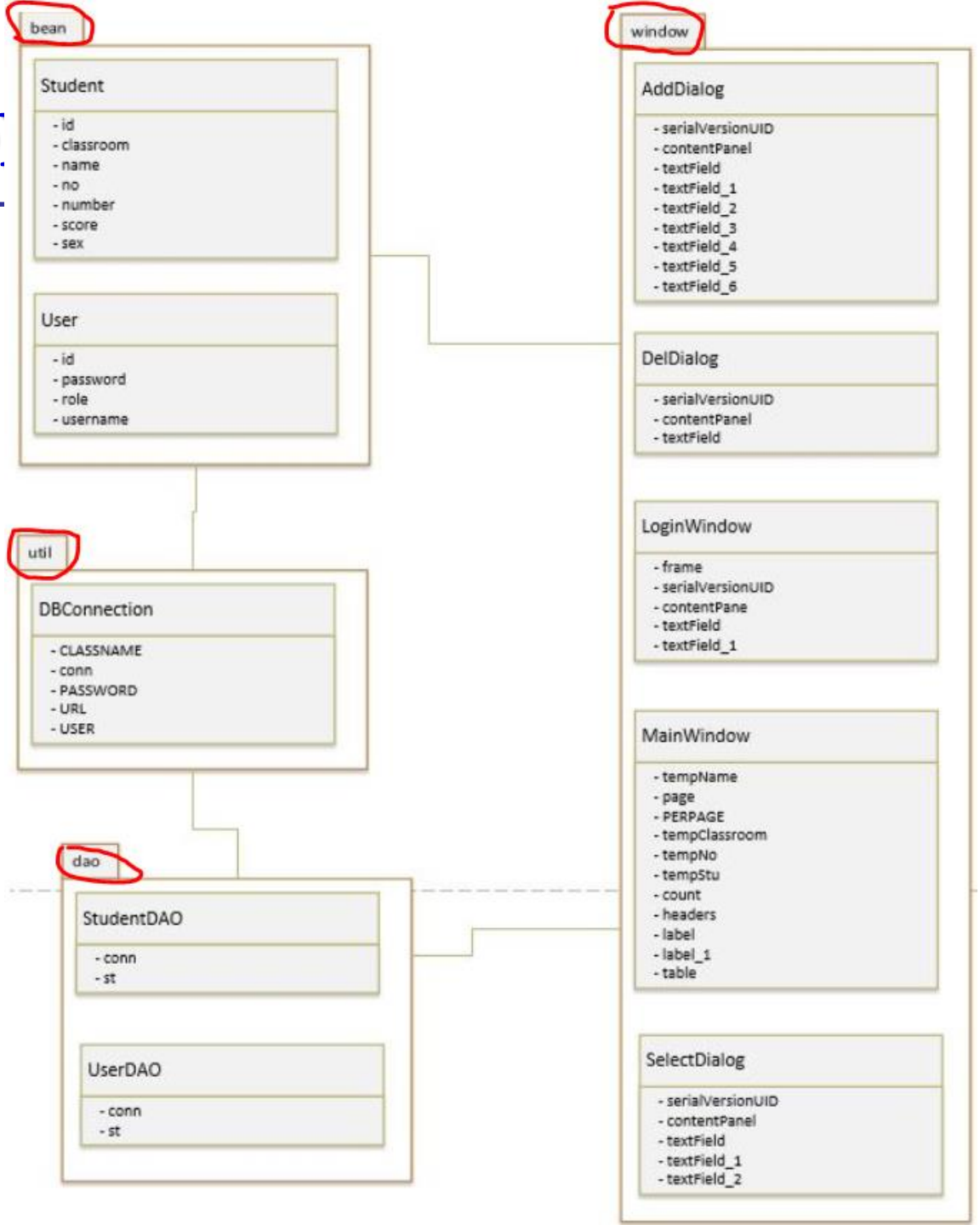
```
sql = "UPDATE BookInf SET Price=Price+10 WHERE Price<?";
pstmt = con.prepareStatement(sql); // 创建PreparedStatement对象
// 指定SQL语句中的参数值
pstmt.setDouble(1, 30); //第1个参数值
int iud=pstmt.executeUpdate(); // 执行sql操作语句
System.out.println("更新了"+iud+"条记录！ ");
sql = "SELECT BookName, Price FROM BookInf";
pstmt=con.prepareStatement(sql); //创建新的PreparedStatement 对象，用于执行查询
ResultSet rs = pstmt.executeQuery();
while(rs.next()){
        String bn=rs.getString("BookName");
        double bp=rs.getDouble("Price");
        System.out.println(bn+"\t"+bp);
}
```

通过JD

【例11-7】学生信息管理系统

项目设计的包：

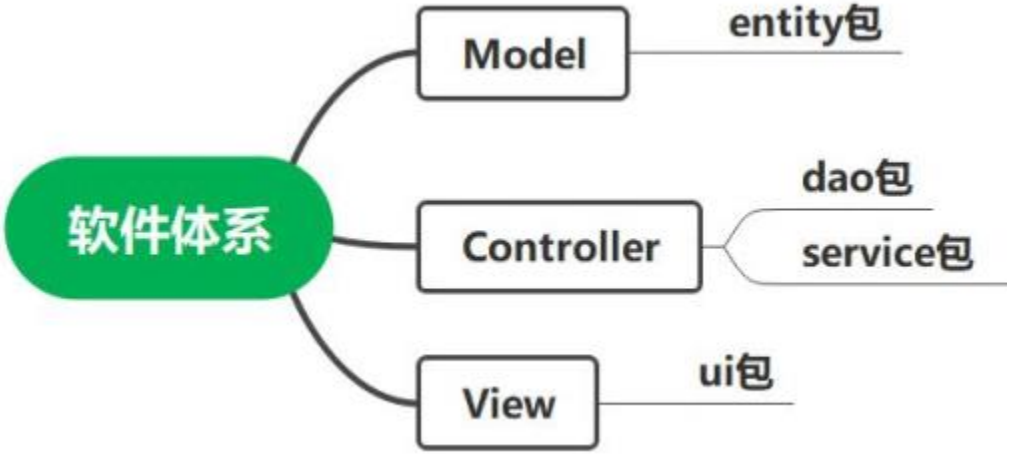
- **bean**包：定义表示学生和用户实体的类
- **util**包：定义了数据库连接类
- **dao**包：定义了各实体对应的数据访问对象类
- **window**包：定义各个窗体类



通过JDBC访问数据库示例

MVC架构

- 模型（Model，对应entity 层）：独立于两个层次，用于封装数据，在两层间传送数据。Model 不引用各层次，但其余两个层都会引用 Model。
- 视图（View，对应UI 层）：展现给用户的界面，用户在使用一个系统时的所见所得。
- 控制器（Controller, 对应 dao 层与 service 层）：针对具体问题的操作，具体到对于某个实体的增删改查、对数据层的操作和对数据业务的逻辑处理。控制器作用于模型和视图上。它控制数据流向模型对象，并在数据变化时更新视图。它使视图与模型分离开。



小 结

1. JDBC API的体系结构及主要任务
2. 通过JDBC访问数据库的方法
   - ① 用Class.forName方法载入具体数据库的JDBC驱动
   - ② 用DriverManager.getConnectio方法创建Connection对象con，以建立与数据库的连接
   - ③ 用con.Statement方法创建SQL语句的执行对象stmt
   - ④ 用stmt.executeQuery方法执行SQL查询的到ResultSet对象rs，用stmt.executeUpdate方法执行添加记录、修改记录、删除记录等操纵数据的SQL语句
   - ⑤ 用rs.next方法移动记录，用rs.getString等方法获取当前记录的某个字段的值
   - ⑥ 用rs.close关闭结果集，用con.close方法关闭数据库连接，用stmt.close方法关闭Statement对象

习 题

a) 在Access或SQLite数据库中创建一个表并输入若干记录，然后编写一个Java控制台程序，输出所有记录并统计记录数。

b) 仿照例11-2编写一个具有GUI的数据库应用程序，数据库采用Access或SQLite，采用JTextArea或Jtable显示数据表。