第 7 章

JDBC数据库访问

本章内容

- 7.1 JDBC技术概述
- 7.2 传统的数据库连接方法
- 7.3 JDBC API介绍
- 7.4 预处理语句
- 7.5 连接池与数据源
- 7.6 DAO设计模式

本章内容

- 许多Web应用程序都需要访问数据库。在Java应用程序中是通过JDBC访问数据库的，JDBC是Sun公司开发的数据库访问API。
- 本章首先介绍了传统的数据库连接方法，然后介绍了常用的JDBC API，接下来介绍了使用数据源连接数据库的方法，最后讨论了DAO设计模式。

7.1 JDBC 技术概述

- JDBC是Java程序访问数据库的标准，它是由一组Java语言编写的类和接口组成，这些类和接口称为JDBC API，它为Java程序提供一种通用的数据访问接口。
- 使用JDBC API可以访问任何的数据源，从关系数据库到电子表格甚至平面文件，它使开发人员可以用纯Java语言编写完整的数据库应用程序。
- JDBC的基本功能包括：建立与数据库的连接；发送SQL语句；处理数据库操作结果。

7.1.1数据库访问的两层和三层模型

- 两层模型（图7-1）即客户机/数据库服务器结构，也就是通常所说的C/S结构。
- 在两层模型中，Java应用程序通过JDBC API直接和数据源交互。用户的SQL命令被传送给数据库或其他数据源，SQL语句的执行结果返回给用户。数据源可以位于网络的其他机器上。这被称为是客户服务器配置，用户机器为客户，存放数据源的机器为服务器。网络可以是企业内部网，也可以是Internet。

7.1.1数据库访问的两层和三层模型

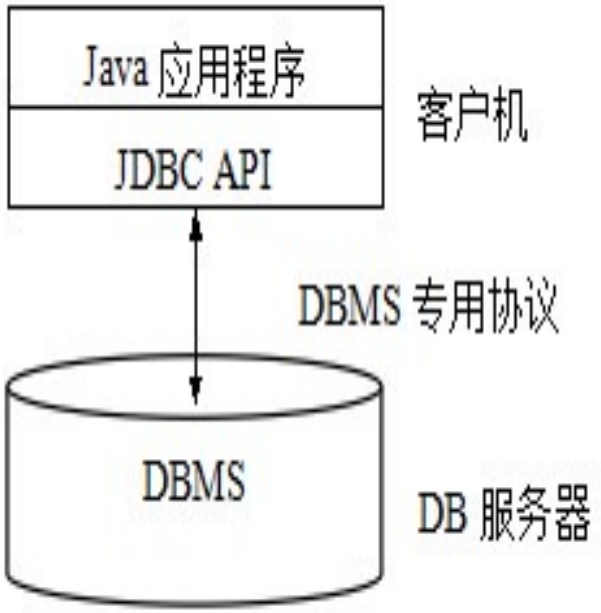


图 7-1　数据库访问的两层模型

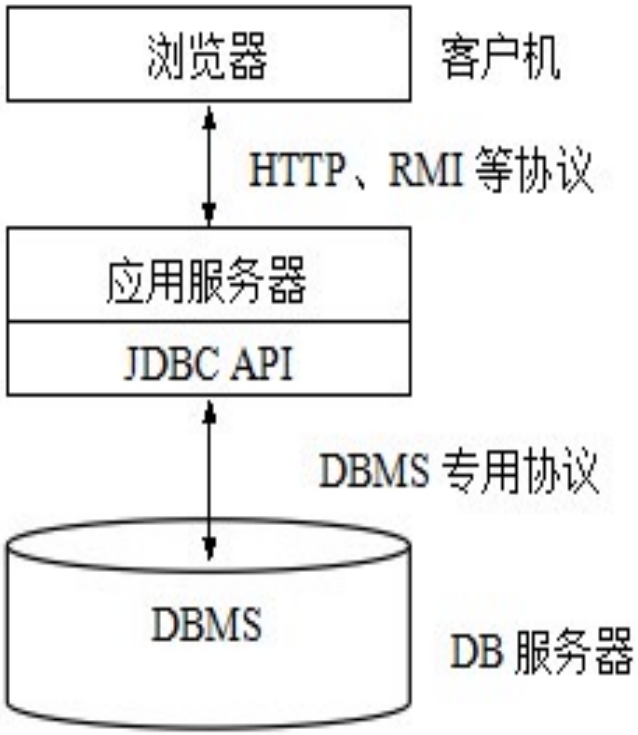


图 7-2　数据库访问的三层模型

7.1.1数据库访问的两层和三层模型

- 三层模型（图7-2）是指客户机/应用服务器/数据库服务器结构，也就是通常所说的B/S结构。在三层模型中，客户机通过Java小程序或浏览器发出SQL请求，该请求首先传送到应用服务器，应用服务器再通过JDBC与数据库服务器进行连接，由数据库服务器处理SQL语句，然后将结果返回给应用服务器，再由应用服务器将结果发送给客户机。这里应用服务器一般是Web服务器，它是一个“中间层”。

7.1.2 JDBC 驱动程序

- Java应用程序访问数据库的一般过程如图7-3所示。应用程序通过JDBC驱动程序管理器加载相应的驱动程序，通过驱动程序与具体的数据库连接，然后访问数据库。

7.1.2 JDBC 驱动程序

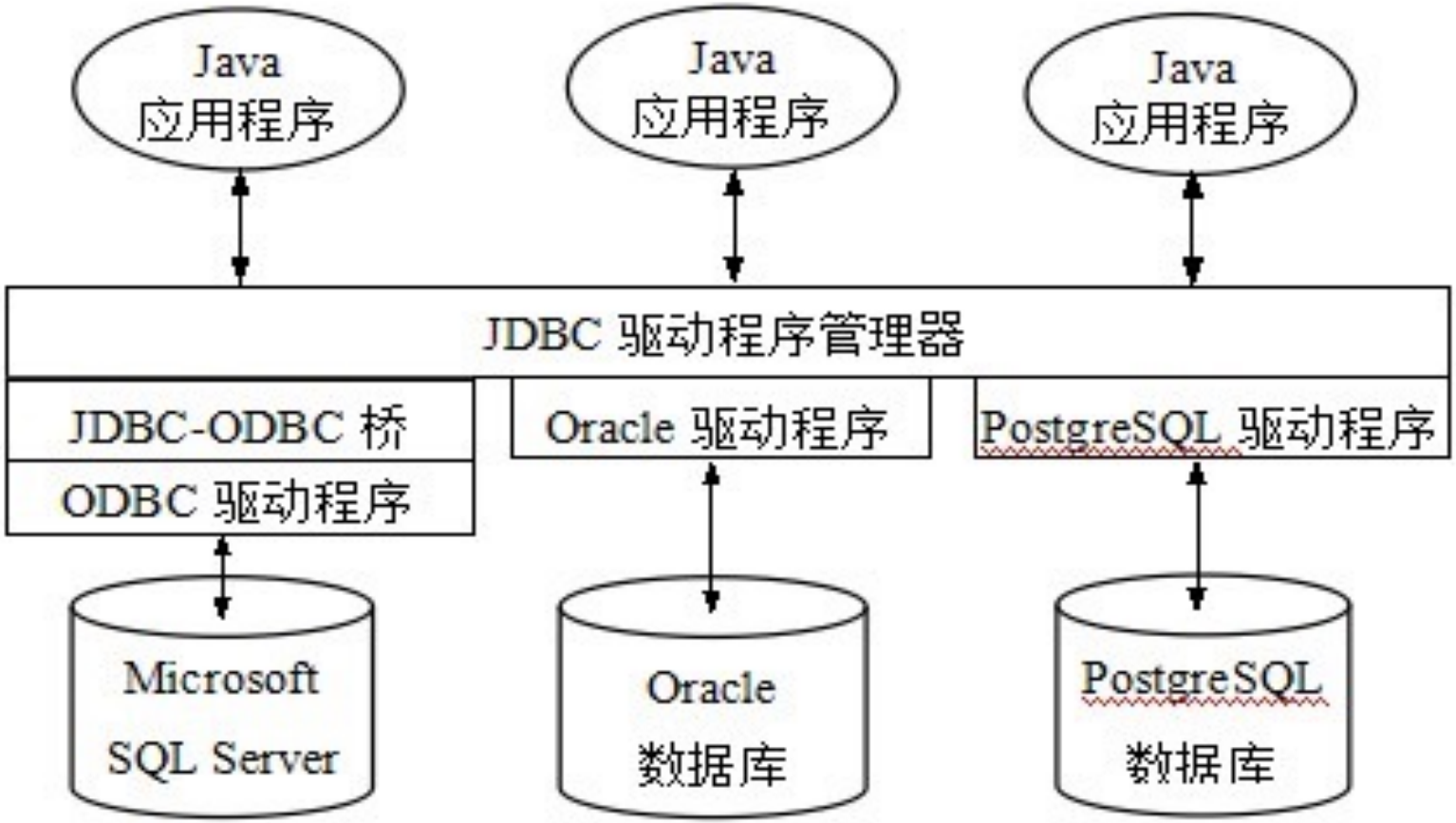


图 7-3　Java 程序访问数据库的过程

7.1.2 JDBC 驱动程序

- 目前有多种类型的数据库，每种数据库都定义了一套API，这些API一般是用C/C++语言实现的，因此需要有在程序收到JDBC请求后，将其转换成适合于数据库系统的方法调用。把完成这类转换工作的程序叫做数据库驱动程序。
- 在Java程序中可以使用的数据库驱动程序主要有4种类型，常用的有下面两种：

  1. JDBC-ODBC桥驱动程序。
  2. 专为某种数据库而编写的驱动程序。

1. JDBC-ODBC 桥驱动程序

- JDBC-ODBC桥驱动程序是为了与Microsoft的ODBC 连接而设计的。
- ODBC（Open DataBase Connectivity）称为开放数据库连接，它是Windows系统与各种数据库进行通信的软件。通过该驱动程序与ODBC进行通信，就可以与各种数据库系统进行通信了。
- 但是不推荐使用这种方法与数据库连接，只是在不能获得数据库专用的JDBC驱动程序或在开发阶段使用这种方法。

2. 专为某种数据库而编写的驱动程序

- 由于ODBC具有一定的缺陷，因此许多数据库厂商专门开发了针对JDBC的驱动程序，这类驱动程序大多是用纯Java语言编写的，因此推荐使用数据库厂商为JDBC开发的专门的驱动程序。

7.1.3　安装JDBC驱动程序

- 使用JDBC-ODBC桥驱动程序连接数据库，不需要安装驱动程序，因为在Java API中已经包含了该驱动程序。
- 使用专用驱动程序连接数据库，必须安装驱动程序。不同的数据库系统提供了不同的JDBC驱动程序，可以到相关网站下载。

7.1.3　安装JDBC驱动程序

- 例如，如果使用PostgreSQL数据库，可以到http://jdbc.postgresql.org/下载，下载后是一个打包文件（如postgresql-9.2-1000.jdbc4.jar）。在开发Web应用程序中，需要将驱动程序打包文件复制到Tomcat安装目录的lib目录中或Web应用程序的WEB-INF\lib目录中。

7.2 传统的数据库连接方法

- JDBC API是在java.sql包和javax.sql包中定义的，其中包括JDBC API用到的所有类和接口。下面是主要的类和接口:

  DriverManager类

  Driver接口

  Connection接口

  Statement接口

  PreparedStatement接口

  ResultSet接口

  CallableStatement接口

  SQLException类

7.2.1　加载驱动程序

- 要使应用程序能够访问数据库，首先必须加载驱动程序。驱动程序是实现了Driver接口的类，它一般由数据库厂商提供。
- 加载JDBC驱动程序最常用的方法是使用Class类的forName()静态方法，该方法的声明格式为：

  public static Class<?> forName(String className)throws ClassNotFoundException
- 参数className为一字符串表示的完整的驱动程序类的名称。如果找不到驱动程序将抛出ClassNotFoundException异常。该方法返回一个Class类的对象。

7.2.1　加载驱动程序

- 对于不同的数据库，驱动程序的类名是不同的。如果使用JDBC-ODBC桥驱动程序连接数据库，则使用JDK自带的驱动程序，名称为“sun.jdbc.odbc.JdbcOdbcDriver”，要加载该驱动程序，可使用下面的语句：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

7.2.1　加载驱动程序

- 如果要加载数据库厂商提供的专门的驱动程序，应该给出专门的驱动程序名。例如，要加载PostgreSQL数据库驱动程序应使用下列语句：

```
Class.forName("org.postgresql.Driver");
```

- 这里，org.postgresql.Driver为PostgreSQL的驱动程序类名。

7.2.2　建立连接对象

- 驱动程序加载成功后应使用DriverManager类的getConnection()建立数据库连接对象。
- DriverManager类维护一个注册的Driver类的列表。

1.DriverManager类

- 建立数据库连接的方法是调用DriverManager类的静态方法getConnection()，该方法的声明格式为：

```
public static Connection
                getConnection(String dburl)
public static Connection getConnection
        (String dburl, String   user,String
password)
```

1.DriverManager类

- 这里字符串参数dburl表示JDBC URL，user表示数据库用户名，password表示口令。调用该方法，DriverManager类试图从注册的驱动程序中选择一个合适的驱动程序，然后建立到给定的JDBC URL的连接。如果不能建立连接将抛出SQLException异常。

2. JDBC URL

- JDBC URL与一般的URL不同，它用来标识数据源，这样驱动程序就可以与它建立一个连接。
- 下面是JDBC URL的标准语法，它包括三个部分，中间用冒号分隔：

  jdbc:<subprotocol>:<subname>

- 其中，jdbc表示协议，JDBC URL的协议总是jdbc。subprotocol表示子协议。subname为子名称，它表示数据库标识符，该部分内容随数据库驱动程序的不同而不同。

2. JDBC URL

- 如果通过JDBC-ODBC桥驱动程序连接数据库，URL的形式为：

  jdbc:odbc:DataSource

- 上面三个部分组成一个整体字符串就是JDBC URL，例如：

  String dburl = "jdbc:odbc:sampleDS" ;

2. JDBC URL

- 如果使用数据库厂商提供的专门的驱动程序连接数据库，JDBC URL可能复杂一些。
- 例如，要连接PostgreSQL数据库，它的JDBC URL为：

  jdbc:postgresql://localhost:5432/dbname
- 这里，localhost表示主机名或IP地址，5432为数据库服务器的端口号，dbname为数据库名。

2. JDBC URL

- 下面代码说明了如何以paipaistore用户连接到PostgreSQL数据库。这里的数据库名为paipaistore、用户名为paipaistore、口令为paipaistore：

```
String dburl =
"jdbc:postgresql://localhost:5432/paipaistore";
Connection conn =
    DriverManager.getConnection(
            dburl, "paipaistore", "paipaistore");
```

常用的数据库JDBC 连接代码

表 7-1　常用数据库的 JDBC 连接代码

数据库	连接代码
Oracle	Class.forName("oracle.jdbc.driver.OracleDriver"); Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@dbServerIP:1521:ORCL",user,password);
PostgreSQL	Class.forName("org.postgresql.Driver"); Connection conn = DriverManager.getConnection("jdbc:postgresql://dbServerIP/dbName", user, password);
MySQL	Class.forName("com.mysql.jdbc.Driver"); Connection conn = DriverManager.getConnection("jdbc:mysql://dbServerIP:3306/dbName?user=userName&password=password");
ODBC	Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); Connection conn = DriverManager.getConnection("jdbc:odbc:DSNName", user, password);
SQL Server	Class.forName("com.micrsoft.jdbc.sqlserver.SQLServerDriver"); Connection conn = DriverManager.getConnection("jdbc:microsoft:sqlserver://dbServerIP:1433;databaseName=master", user, password);

7.2.3 创建语句对象

- 通过Connection对象，可创建语句对象。对于不同的语句对象，可以使用Connection接口的不同方法创建。
- 例如，要创建一个简单的Statement对象可以使用createStatement()，创建PreparedStatement对象应该使用prepareStatement()，创建CallableStatement对象应该使用prepareCall()。

- 下面的代码将创建一个简单的Statement对象。

```
Statement stmt = conn.createStatement();
```

7.2.4 获得SQL语句的执行结果

- 执行SQL语句使用Statement对象的方法。对于查询语句，调用executeQuery(Stringsql)返回ResultSet。ResultSet对象保存查询的结果集，再调用ResultSet的方法可以对查询结果的每行进行处理。

```
String sql = "SELECT * FROM products" ;
ResultSet rst = stmt.executeQuery(sql) ;
while(rst.next()){
        out.print(rst.getString(1)+"\t") ;    }
```

7.2.4 获得SQL语句的执行结果

- 对于DDL语句如CREATE、ALTER、DROP和DML语句如INSERT、UPDATE、DELETE等须使用语句对象的executeUpdate(String sql)。该方法返回值为整数，用来指示被影响的行数。

7.2.5 关闭建立的对象

- 在 Connection 接口、Statement 接口和 ResultSet接口中都定义了close()。当这些对象使用完毕后应使用close()关闭。
- 如果使用Java 7的try-with-resources语句，则可以自动关闭这些对象。

7.2.6 简单的应用示例

- 本示例程序可以根据用户输入的商品号从数据库中查询该商品信息，或者查询所有商品信息。
- 本应用的设计遵循了MVC设计模式，其中视图有queryProduct.jsp、displayProduct.jsp、displayAllProduct.jsp和error.jsp几个页面，Product类实现模型，QueryProductServlet类实现控制器。

7.2.6 简单的应用示例

- 该应用需要访问数据库表products中的数据，该表的定义如下。

```
CREATE TABLE products (
  prod_id character(3) NOT
NULL,
  pname character varying(30)
      NOT NULL,
  price numeric(8,2),
  stock integer,
CONSTRAINT product_pkey PRIMARY
KEY (prod_id)
)
```

7.2.6　简单的应用示例

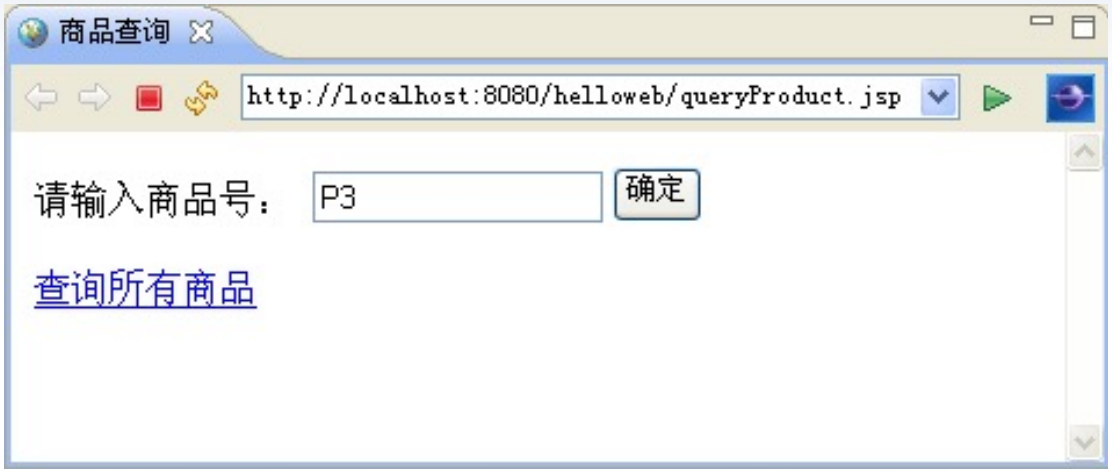
- 根据表的定义，设计下面的JavaBeans存放商品信息，这里表的字段对应Product类的成员变量。

  程序7.1　Product.java
- 下面是queryProduct.jsp页面代码。

  程序7.2　queryProduct.jsp

7.2.6 简单的应用示例

- 该页面运行结果如图7-4所示。



- 下面的Servlet连接数据库，当用户在文本框中输入商品号，单击“确定”按钮，将执行doPost()，当用户单击“查询所有商品”链接时，将执行doGet()。
- 程序7.3 QueryProductServlet.java

7.2.6　简单的应用示例

- 下面的JSP页面displayProduct.jsp和displayAllProduct.jsp分别显示查询一件商品和所有商品信息。

  程序7.4　displayProduct.jsp

- 当单击图7-4中“查询所有商品”链接时，控制将转到displayAllProduct.jsp页面，如下所示

  程序7.5　displayAllProduct.jsp

7.2.6　简单的应用示例

- 当查询的商品不存在时，显示下面的页面。

  程序7.6　error.jsp

- 在图7-4的页面中输入商品号，单击“确定”按钮，则显示如图7-5所示页面。在图7-4中单击“查询所有商品”链接，则显示如图7-6所示页面。

图 7-5　显示指定商品

图 7-6　显示所有商品

7.3 JDBC API介绍

- JDBC API是Java语言的标准API，目前的最新版本是JDBC 4.0。在JDK 7.0中，它是通过两个包提供的：
  - java.sql包
  - javax.sql包

7.3 JDBC API介绍

- java.sql包提供了基本的数据库编程的类和接口，如驱动程序管理类DriverManager、创建数据库连接Connection接口、执行SQL语句以及处理查询结果的类和接口等。
- javax.sql包主要提供了服务器端访问和处理数据源的类和接口，如DataSource、RowSet、RowSetMetaData、PooledConnection接口等，它们可以实现数据源管理、行集管理以及连接池管理等。

7.3.1 Connection接口

- 调用DriverManager类的静态方法getConnection()或数据源（DataSource）对象的getConnection()都可以得到连接（Connection）对象，得到连接对象后就可以调用它的createStatement()创建SQL语句（Statement）对象以及在连接对象上完成各种操作，下面是Connection接口创建Statement对象的方法。

7.3.1 Connection接口

- public Statement createStatement()：创建一个Statement对象。如果这个Statement对象用于查询，那么调用它的executeQuery()返回的ResultSet是一个不可滚动、不可更新的ResultSet。

7.3.1 Connection接口

- public Statement createStatement(int resultType, int concurrency)：创建一个Statement对象。如果这个Statement对象用于查询，那么这两个参数决定executeQuery()返回的ResultSet是否是一个可滚动、可更新的ResultSet。

7.3.1 Connection接口

- public Statement createStatement(int resultType, int concurrency, int holdability)：创建一个Statement对象。如果这个Statement对象用于查询，那么前两个参数决定executeQuery()返回的ResultSet是否是一个可滚动、可更新的ResultSet，第三个参数决定可保持性（holdability）。

7.3.2 Statement接口

- 一旦创建了Statement对象，就可以用它来向数据库发送SQL语句，实现对数据库的查询和更新操作等。

1. 执行查询语句

- 可以使用Statement接口的下列方法向数据库发送SQL查询语句。

  public ResultSet executeQuery(String sql)

- 该方法用来执行SQL查询语句。参数sql为用字符串表示的SQL查询语句。查询结果以ResultSet对象返回，一般称为结果集对象。在ResultSet对象上可以逐行逐列地读取数据。

2. 执行非查询语句

- 可以使用Statement接口的下列方法向数据库发送非SQL查询语句。

  public int executeUpdate(String sql)

- 该方法执行由字符串sql指定的SQL语句，该语句可以是INSERT、DELETE、UPDATE语句或者无返回值的SQL语句，如SQL DDL语句CREATE TABLE。返回值是更新的行数，如果语句没有返回则返回值为0。

2. 执行非查询语句

- public boolean execute(String sql)：执行可能有多个结果集的SQL语句，sql为任何的SQL语句。如果语句执行的第一个结果为ResultSet对象，该方法返回true，否则返回false。
- public int[] executeBatch()：用于在一个操作中发送多条SQL语句。

3. 释放Statement

- 与Connection对象一样，Statement对象使用完毕应该用close()将其关闭，释放其占用的资源。
- 但这并不是说在执行了一条SQL语句后就立即释放这个Statement对象，可以用同一个Statement对象执行多个SQL语句。

7.3.3 ResultSet接口

- ResultSet对象表示SELECT语句查询得到的记录集合，结果集一般是一个记录表，其中包含多个记录行和列标题，记录行从1开始，一个Statement对象一个时刻只能打开一个ResultSet对象。
- 如果需要对结果集的每行进行处理，需要移动结果集的游标。所谓游标（cursor）是结果集的一个标志或指针。对新产生的ResultSet对象，游标指向第一行的前面，可以调用ResultSet的next()，使游标定位到下一条记录。

7.3.3 ResultSet接口

- next()的格式如下。

public boolean next() throws SQLException

- 将游标从当前位置向下移动一行。第一次调用next()将使第一行成为当前行，以后调用游标依次向后移动。如果该方法返回true，说明新行是有效的行，若返回false，说明已无记录。

1. 检索字段值

- ResultSet接口提供了检索行的字段值的方法，由于结果集列的数据类型不同，所以应该使用不同的getXxx()获得列值，例如若列值为字符型数据，可以使用下列方法检索列值：

```
String getString (int columnIndex)

String getString (String columnName)
```

1. 检索字段值

- 返回结果集中当前行指定的列号或列名的列值，结果作为字符串返回。columnIndex为列在结果行中的序号，序号从1开始，columnName为结果行中的列名。

- 下面列出了返回其他数据类型的方法，这些方法都可以使用这两种形式的参数：

1. 检索字段值

- public boolean getBoolean(int columnIndex)：返回指定列的boolean值。
- public Date getDate(int columnIndex)：返回指定列的Date对象值。
- public Object getObject(int columnIndex)：返回指定列的Object对象值。
- public Blob getBlob(int columnIndex)：返回指定列的Blob对象值。
- public Clob getClob(int columnIndex)：返回指定列的Clob对象值。

2. 数据类型转换

- 在ResultSet对象中的数据为从数据库中查询出的数据，调用ResultSet对象的getXxx()方法返回的是Java语言的数据类型，因此这里就有数据类型转换的问题。
- 实际上调用getXxx()方法就是把SQL数据类型转换为Java语言数据类型，表7-2列出了SQL数据类型与Java数据类型的转换。

2. 数据类型转换

表 7-2 SQL 数据类型与 Java 数据类型之间的对应关系

SQL 数据类型	Java 数据类型	SQL 数据类型	Java 数据类型
CHAR	String	DOUBLE	double
VARCHAR	String	NUMERIC	java.math.BigDecimal
BIT	boolean	DECIMAL	java.math.BigDecimal
TINYINT	byte	DATE	java.sql.Date
SMALLINT	short	TIME	java.sql.Time
INTEGER	int	TIMESTAMP	java.sql.Timestamp
REAL	float	CLOB	Clob
FLOAT	double	BLOB	Blob
BIGINT	long	STRUCT	Struct

7.3.4　可滚动与可更新的ResultSet

- 可滚动的ResultSet是指在结果集对象上可以前后移动指针访问结果集中的记录。
- 可更新的ResultSet是指不但可以访问结果集中的记录，还可以通过结果集对象更新数据库。

1. 可滚动的ResultSet

- 要使用可滚动的ResultSet对象，必须使用Connection对象的带参数的createStatement()创建的Statement，然后该对象的结果集才是可滚动的，该方法的格式为：

```
public Statement createStatement(
        int resultType, int   concurrency)
```

- 如果这个Statement对象用于查询，那么这两个参数决定executeQuery()返回的ResultSet是否是一个可滚动、可更新的ResultSet。

1. 可滚动的ResultSet

- 参数resultType的取值应为ResultSet接口中定义的下面常量。

  ResultSet.TYPE_SCROLL_SENSITIVE

  ResultSet.TYPE_SCROLL_INSENSITIVE

  ResultSet.TYPE_FORWARD_ONLY

- 前两个常量用于创建可滚动的ResultSet。使用TYPE_SCROLL_SENSITIVE常量，当数据库发生改变时，这些变化对结果集可见。使用TYPE_SCROLL_INSENSITIVE常量，当数据库发生改变时，这些变化对结果集不可见。使用TYPE_FORWARD_ONLY常量创建不可滚动的结果集。

1. 可滚动的ResultSet

- 对于可滚动的结果集，ResultSet接口提供了下面的移动结果集游标的方法。
- public boolean previous() throws SQLException：游标向前移动一行，如果存在合法的行返回true，否则返回false。
- public boolean first() throws SQLException：移动游标使其指向第一行。
- public boolean last() throws SQLException：移动游标使其指向最后一行。
- public boolean absolute(int rows) throws SQLException：移动游标使其指向指定的行。

1. 可滚动的ResultSet

- public boolean relative(int rows) throws SQLException：以当前行为基准相对移动游标的指针，rows为向后或向前移动的行数。rows若为正值是向前移动，若为负值是向后移动。
- public boolean isFirst() throws SQLException：返回游标是否指向第一行。
- public boolean isLast() throws SQLException：返回游标是否指向最后一行。
- public int getRow()：返回游标所在当前行的行号。

2. 可更新的ResultSet

- 在JDBC 2.0之前，ResultSet对象只可用于查询数据、向前移动游标和读取每列数据值。为了更新数据，需要通过Statement对象执行另外的SQL语句。
- JDBC 2.0提供了直接通过ResultSet对象更新数据库表中数据的能力。要实现该功能，应该创建一个可更新的ResultSet对象。

2. 可更新的ResultSet

- 首先在使用Connection的
  createStatement(int resultType,
  int concurrency)
- 创建Statement对象时，通过指定第二个参数的值决定是否创建可更新的结果集，该参数也使用ResultSet接口中定义的常量，如下所示。
  ResultSet.CONCUR_READ_ONLY
  ResultSet.CONCUR_UPDATABLE
- 使用第一个常量创建一个只读的ResultSet对象，不能通过它更新表。使用第二个常量则创建一个可更新的ResultSet对象。

2. 可更新的ResultSet

- 得到可更新的ResultSet对象后，就可以调用适当的updateXxx()更新指定的列值。对于每种数据类型，ResultSet都定义了相应的updateXxx()，例如：
- public void updateInt(int columnIndex, int x)：用指定的整数x的值更新当前行指定的列的值，其中columnIndex为列的序号。
- public void updateInt(String columnName, int x)：用指定的整数x的值更新当前行指定的列的值，其中columnName为列名

2. 可更新的ResultSet

- public void updateString(int columnIndex, String x)：用指定的字符串x的值更新当前行指定的列的值，其中columnIndex为列的序号。
- public void updateString(String columnName, String x)：用指定的字符串x的值更新当前行指定的列的值，其中columnName为列名。

2. 可更新的ResultSet

- 每个updateXxx()都有两个重载的版本，一个是第一个参数是int类型的，用来指定更新的列号，另一个是第一个参数是String类型的，指定更新的列名。第二个参数的类型与要更新的列的类型一致。有关其他方法请参考API文档。

2. 可更新的ResultSet

- 下面是通过可更新的ResultSet对象实现对表的插入、删除和修改。
- void moveToInsertRow() throws SQLException：将游标移到插入行。要插入一行数据首先应该使用该方法将游标移到插入行。插入行是与可更新结果集相关的特殊行。它实际上是一个在将行插入到结果集之前构建的新行的缓冲区。当游标处于插入行时，调用updateXxx()用相应的数据修改每列的值。完成新行数据修改之后，调用insertRow()将新行插入结果集。。

2. 可更新的ResultSet

- void insertRow() throws SQLException：插入一行数据。在调用该方法之前，插入行所有的列都必须给定一个值。调用insertRow()之后，这个ResultSet仍然位于插入行。这时，可以插入另外一行数据，或者移回到刚才ResultSet记住的位置（当前行位置）。
- void moveToCurrentRow() throws SQLException：返回到当前行。也可以在调用insertRow()之前通过调用moveToCurrentRow()取消插入。

2. 可更新的ResultSet

- void deleteRow() throws SQLException：从结果集中删除当前行，同时从数据库中将该行删除。
- void updateRow() throws SQLException：执行该方法，将用当前行的新内容更新结果集，同时更新数据库。当使用updateXxx()更新了需要更新的所有列之后，调用该方法把更新写入表中。在调用updateRow()之前，如果决定不想更新这行数据，可以调用cancelRowUpdate()取消更新。

2. 可更新的ResultSet

- 下面代码说明了如何在products表中修改一件商品的名称。

```
String sql = "SELECT prod_id,pname
                    FROM products
                   WHERE prod_id ='P2'";
ResultSet rset = stmt.executeQuery(sql);
rset.next();
// 修改当前行的字段值
rset.updateString(2,"iPhone 5 手机");
rset.updateRow();     // 更新当前行
```

7.4 预处理语句

- Statement对象在每次执行SQL语句时都将语句传给数据库，这样在多次执行同一个语句时效率较低，这时可以使用PreparedStatement对象。如果数据库支持预编译，它可以将SQL语句传给数据库作预编译，以后每次执行这个SQL语句时，速度就可以提高很多。
- PreparedStatement接口继承了Statement接口，因此它可以使用Statement接口中定义的方法。PreparedStatement对象还可以创建带参数的SQL语句，在SQL语句中指出接收哪些参数，然后进行预编译。

7.4.1 创建PreparedStatement对象

- 创建PreparedStatement对象与创建Statement对象类似，唯一不同的是需要给创建的PreparedStatement对象传递一个SQL命令，即需要将执行的SQL命令传递给其构造方法而不是execute()。用Connection的下列方法创建PreparedStatement对象。
- PreparedStatement prepareStatement(String sql)：使用给定的SQL命令创建一个PreparedStatement对象，在该对象上返回的ResultSet是只能向前滚动的结果集。

7.4.1 创建PreparedStatement对象

PreparedStatement prepareStatement(
String sql，int resultType, int concurrency)

- 使用给定的SQL命令创建一个PreparedStatement对象，在该对象上返回的ResultSet可以通过resultType和concurrency参数指定是否可滚动、是否可更新。

7.4.1　创建PreparedStatement对象

- 这些方法的第一个参数是SQL字符串。这些SQL字符串可以包含一些参数，这些参数在SQL中使用问号（?）作为占位符，在SQL语句执行时将用实际数据替换。

7.4.2 使用PreparedStatement对象

- PreparedStatement对象通常用来执行动态SQL语句，此时需要在SQL语句通过问号指定参数，每个问号为一个参数。例如：

```
String sql = "SELECT * FROM products
                    WHERE prod_id = ?";
String sql = "INSERT INTO products VALUES
                    (?, ?, ?, ?) ";
PreparedStatement pstmt =
                    conn.prepareStatement(sql);
```

- SQL命令中的每个占位符都是通过它们的序号被引用，从SQL字符串左边开始，第一个占位符的序号为1，依此类推。

1. 设置占位符

- 创建PreparedStatement对象之后，在执行该SQL语句之前，必须用数据替换每个占位符。可以通过PreparedStatement接口中定义的setXxx()为占位符设置具体的值。
- 例如，下面方法分别为占位符设置整数值和字符串值。
- public void setInt(int parameterIndex,int x)：这里parameterIndex为参数的序号，x为一个整数值。
- public void setString(int parameterIndex, String x)：为占位符设置一个字符串值。

1. 设置占位符

- 每个Java基本类型都有一个对应的setXxx()，此外，还有许多对象类型，如Date和BigDecimal都有相应的setXxx()。
- 对于前面的INSERT语句，可以使用下面的方法设置占位符的值。

```
pstmt.setString(1,"P8");
pstmt.setString(2,"iPhone 5手机");
pstmt.setDouble(3, 1490.00);
pstmt.setInt(4, 5);
```

2. 用复杂数据设置占位符

- 使用PreparedStatement对象还可以在数据库中插入和更新复杂数据。
- 例如，如果向表中插入日期或时间数据，数据库对日期的格式有一定的格式规定，如果不符合格式的要求，数据库不允许插入。因此，一般要查看数据库文档来确定使用什么格式，使用预处理语句就不必这么麻烦。

2. 用复杂数据设置占位符

- 使用预处理语句对象可以对要插入到数据库的数据进行处理。对于日期、时间和时间戳的数据，只要简单地创建相应的java.sql.Date或java.sql.Time对象，然后把它传给预处理语句对象的setDate()或setTime()即可。假设getSqlDate()返回给定日期的Date对象，使用下面语句可以设置日期参数。

```
Date d = getSqlDate("23-Jul-13");
pstmt.setDate(1,d);
```

3. 执行预处理语句

- 设置好PreparedStatement对象的全部参数后，调用它的有关方法执行语句。
- 对查询语句应该调用executeQuery()，如下所示。

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

- 对更新语句，应该调用executeUpdate()，如下所示。

```
int n = pstmt.executeUpdate();
```

3. 执行预处理语句

- 对其他类型的语句，应该调用execute()，如下所示。

```
boolean b = pstmt.execute();
```

- 注意：对预处理语句，必须调用这些方法的无参数版本，如executeQuery()等。如果调用executeQuery(String)、executeUpdate(String)或者execute (String)，将抛出SQLException异常。如果在执行SQL语句之前没有设置好全部参数，也会抛出一个SQLException异常。

7.5 连接池与数据源

- 在设计需要访问数据库的Web应用程序时，需要考虑的一个主要问题是如何管理Web应用程序与数据库的通信。
- 一种方法是为每个HTTP请求创建一个连接对象，Servlet建立数据库连接、执行查询、处理结果集、请求结束关闭连接。建立连接是比较耗费时间的操作，如果在客户每次请求时都要建立连接，这将导致增大请求的响应时间。
- 为了提高数据库访问效率，从JDBC 2.0开始提供了一种更好的方法建立数据库连接对象，即使用连接池和数据源的技术访问数据库。

7.5.1 连接池与数据源介绍

- 数据源（DataSource）的概念是在JDBC 2.0中引入的，是目前Web应用开发中获取数据库连接的首选方法。
- 这种方法是事先建立若干连接对象，将它们存放在数据库连接池（connection pooling）中供数据访问组件共享。
- 使用这种技术，应用程序在启动时只需创建少量的连接对象即可。这样就不需要为每个HTTP请求都创建一个连接对象，这会大大降低请求的响应时间。

7.5.1　连接池与数据源介绍

- 数据源是通过javax.sql.DataSource接口对象实现的，通过它可以获得数据库连接，因此它是对DriverManager工具的一个替代。
- 通常DataSource对象是从连接池中获得连接对象。连接池预定义了一些连接，当应用程序需要连接对象时就从连接池中取出一个，当连接对象使用完毕将其放回连接池，从而可以避免在每次请求连接时都要创建连接对象。

7.5.1　连接池与数据源介绍

- 通过数据源获得数据库连接对象不能直接在应用程序中通过创建一个实例的方法来生成DataSource对象，而是需要采用Java命名与目录接口 （Java Naming and Directory Interface，JNDI）技术来获得DataSource对象的引用。

7.5.1 连接池与数据源介绍

- 可以简单地把JNDI理解为一种将名字和对象绑定的技术，首先为要创建的对象指定一个唯一的名字，然后由对象工厂创建对象，并将该对象与唯一的名字绑定，外部程序可以通过名字来获得某个对象的访问。
- 在javax.naming包中提供了Context接口，该接口提供了将名字和对象绑定，通过名字检索对象的方法。可以通过该接口的一个实现类InitialContext获得上下文对象。

7.5.2 配置数据源

- 下面讨论在Tomcat中如何配置使用DataSource建立数据库连接。
- 在Tomcat中可以配置两种数据源：局部数据源和全局数据源。局部数据源只能被定义数据源的应用程序使用，全局数据源可被所有的应用程序使用。
- 注意：在Tomcat中，不管配置哪种数据源，都要将JDBC驱动程序复制到Tomcat安装目录的lib目录中，并且需要重新启动服务器。

1. 配置局部数据源

- 建立局部数据源非常简单，首先在Web应用程序中建立一个META-INF目录，在其中建立一个context.xml文件，下面代码配置了连接PostgreSQL数据库的数据源，内容如下。
- 程序7.7 context.xml

1. 配置局部数据源

- name：数据源名，这里是jdbc/sampleDS。
- driverClassName：使用的JDBC驱动程序的完整类名。
- url：传递给JDBC驱动程序的数据库URL。
- username：数据库用户名。
- password：数据库用户口令。
- type：指定该资源的类型，这里为DataSource类型。
- maxActive：可同时为连接池分配的活动连接实例的最大数。
- maxIdle：连接池中可空闲的连接的最大数。
- maxWait：在没有可用连接的情况下，连接池在抛出异常前等待的最大毫秒数。

1. 配置局部数据源

- 通过上面的设置后，不用在Web应用程序的web.xml文件中声明资源的引用就可以直接使用局部数据源。
- 注意：在Tomcat中，如果修改了context.xml文件，应将Tomcat安装目录中的conf/Catalina//localhost/paipaistore.xml文件删除。

2. 在应用程序中使用数据源

- 配置了数据源后，就可以使用javax.naming.Context接口的lookup()查找JNDI数据源，如下面代码可以获得jdbc/sampleDS数据源的引用。

```
Context context = new InitialContext();
DataSource ds = (DataSource)context.
  lookup("java:comp/env/jdbc/sampleDS");
```

2. 在应用程序中使用数据源

- 查找数据源对象的lookup()的参数是数据源名字符串，但要加上“java:comp/env”前缀，它是JNDI命名空间的一部分。
- 得到了DataSource对象的引用后，就可以通过它的getConnection()获得数据库连接对象Connection。

2. 在应用程序中使用数据源

- 对程序7.3的数据库连接程序，如果使用数据源获得数据库连接对象，修改后的程序如下。
- 程序7.8 ProductQueryServlet.java
- 代码首先通过InitialContext类创建一个上下文对象context，然后通过它的lookup()查找数据源对象，最后通过数据源对象从连接池中返回一个数据库连接对象。

3. 配置全局数据源

- 全局数据源可被所有应用程序使用，它是通过<tomcat-install>/conf/server.xml文件的<GlobalNamingResources>元素定义的，定义后就可在任何的应用程序中使用。
- 假设要配置一个名为jdbc/paipaistore的数据源，应该按下列步骤操作。

3. 配置全局数据源

（1）首先在server.xml文件的<GlobalNamingResources>元素内增加下面代码。

```
<Resource
        name="jdbc/paipaistore"
        type="javax.sql.DataSource"
        maxActive="4"
        maxIdle="2"
        username="paipaistore"
        maxWait="5000"
        driverClassName="org.postgresql.Driver"
        password="paipaistore"
        url="jdbc:postgresql://localhost:5432/paipaistore"
/>
```

- 这里的name属性值是指全局数据源名称，其他属性与局部数据源属性含义相同。

3. 配置全局数据源

（2）在Web应用程序中建立一个META-INF目录，在其中建立一个context.xml文件，内容如下。

- 程序7.9 context.xml
- 上述文件中<ResourceLink>元素用来创建到全局JNDI资源的链接。该元素有三个属性。
- global：指定在全局JNDI环境中所定义的全局资源名。
- name：指定数据源名，该名相对于java:comp/env命名空间前缀。
- type：指定该资源的类型的完整类名。

3. 配置全局数据源

- 配置了全局数据源后，需重新启动Tomcat服务器才能生效。使用全局数据源访问数据库与局部数据源相同。
- 说明：有些服务器（包括Tomcat）提供了图形界面的管理工具，使用这些工具配置数据源更方便。

7.6 DAO设计模式

- DAO（Data Access Object）称为数据访问对象。DAO设计模式可以在使用数据库的应用程序中实现业务逻辑和数据访问逻辑分离，从而使应用的维护变得简单。
- 它通过将数据访问实现（通常使用JDBC技术）封装在DAO类中，提高应用程序的灵活性。

7.6 DAO设计模式

- 7.6.1 设计持久对象
- 7.6.2 设计DAO对象
- 7.6.3 使用DAO对象

7.6.1　设计持久对象

- 在分布式Web应用中，经常需要把数据从表示层传输到业务层，或者从业务层传输到表示层。跨层传输数据最好的方法是使用持久对象（Persistent Object）。持久对象只包含数据元素，不包含任何业务逻辑，业务逻辑由业务对象实现。
- 持久对象必须是可序列化的，也就是它的类必须实现java.io.Serializable接口。

7.6.1　设计持久对象

- 下面程序的Customer类就是持久对象。

  程序7.10　Customer.java

- 该持久对象用于在程序中保存应用数据，并可实现对象与关系数据的映射，它实际上是一个可序列化的JavaBeans。

7.6.2　设计DAO对象

- DAO设计模式有许多种变体，这里介绍一种简单的。先定义一个基类BaseDao连接数据库，通过该类可以获得一个连接对象，然后定义CustomerDao类，其中定义了添加客户、查找客户、查找所有客户的方法。

7.6.2　设计DAO对象

- 程序7.11　BaseDao.java
- 下面的CustomerDao类继承了BaseDao类并实现了添加客户、查询客户、查询所有客户的方法。
- 程序7.12　CustomerDao.java
- 该类没有给出修改记录和删除记录的方法，读者可自行补充完整。

7.6.3 使用DAO对象

- 下面的addCustomer.jsp页面通过一个表单提供向数据库中插入的数据。
- 程序7.13 addCustomer.jsp
- 下面的AddCustomerServlet使用了DAO对象和传输对象，通过JDBC API实现将数据插入到数据库中。
- 程序7.14 AddCustomerServlet.java
- 该程序首先从请求对象中获得请求参数并进行编码转换，创建一个Customer对象，然后调用CustomerDao对象的insertCustomer()将客户对象插入数据库中，最后根据该方法执行结果将请求再转发到addCustomer.jsp页面。

7.7 小 结

- Java程序是通过JDBC API访问数据库。JDBC API定义了Java程序访问数据库的接口。访问数据库首先应该建立到数据库的连接。传统的方法是通过DriverManager类的getConnection()建立连接对象。使用这种方法很容易产生性能问题。因此，从JDBC 2.0开始提供了通过数据源建立连接对象的机制。

7.7 小 结

- 通过数据源连接数据库，首先需要建立数据源，然后通过JNDI查找数据源对象，建立连接对象，最后通过JDBC API操作数据库。通过PreparedStatement对象可以创建预处理语句对象，它可以执行动态SQL语句。