第 11 章　核心数据库连接包JDBC

数据库连接是动态网站建立要解决的最基本问题之一。在 J2SE（Java 标准运行平台，一般基于个人 PC 机）和 J2EE（Java 企业运行平台，一般基于企业系统）中已经集成了 Java 数据库连接（JDBC）方案。JDBC 是 Java 应用程序（包括 Application 和 Web 应用）连接 DBMS（数据库管理系统）的标准方式，它与 DBMS 无关，可以连接遗留和新建企业数据。

本章要点包括以下内容：

- JDBC 体系结构
- JDBC 驱动器及其类型
- JDBC 驱动器配置
- JDBC 的数据库连接
- JDBC 的 Statement 语句以及 PreparedStatement 准备语句
- JDBC 的结果集 ResultSet
- 基本 SQL 操作语句

11.1　JDBC体系结构

Java 数据库连接（JDBC）体系结构是从 Java 应用程序连接 DBMS 的标准方式。JDBC 既是 Java 编程人员需要使用的 API，也是实现数据库连接的服务提供商的接口模型。

- 作为 API，JDBC 提供了 Java 应用程序与各种不同数据库交互的标准接口。开发者使用这些 JDBC 接口进行各类数据库操作；
- 作为服务提供者的接口模型，JDBC 提供了数据库厂家和第三方中间件厂家实现数据库连接的标准方式，这是为了相互兼容性。类似于产品的规格，例如车的轮胎，所有厂家生成的同型号轮胎规格是一样。

JDBC 利用现有 SQL 标准，可以和 ODBC 之类其他数据库连接标准相互桥接。为了达到这些面向标准化的目标（更大的兼容性），JDBC 使用的接口简单，强类型、支持高性能实现。

现在 JDBC 已经被集成到 JDK 的 java.sql 包中。为了让读者能够更加清晰的了解 JDBC，下面用图描述一下 JDBC 的体系结构，如图 11.1 所示。

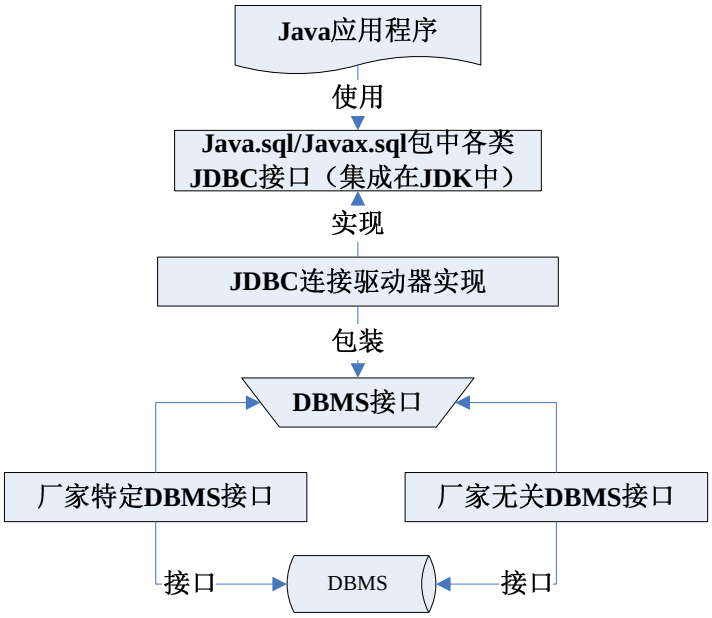

图 11.1　JDBC 体系结构

由图 11.1 可看出，Java 应用程序（这里包括 Application 和 Web 应用）可以使用集成在 JDK 的 java.sql 和 javax.sql 包中 JDBC 接口来连接和操作数据库。另外由图可知，这些包中的核心 JDBC 接口还需要 JDBC 驱动器厂家来实现（即对定义的接口类实现方法），上一章下载的 mysql-connector-java-3.1.12.zip 文件就是针对 MySQL 数据库的一个驱动包。

JDBC 驱动器实现对一个或者多个 DBMS 接口提供基于 Java 的包装器。许多 JDBC 驱动器实现只连接件一种类型的数据库，但有些中间件厂家驱动器实现可以连接不同类型的数据库。

11.2　JDBC驱动器及其类型

上一小节介绍的 JDBC 体系结构中描述的 JDBC 驱动器实现可以交互的两类 DBMS 接口。这些 DBMS 接口可以按照接口开发性分为厂家特定与厂家无关 DBMS 接口。JDBC 驱动器主要有四种不同类型，下面逐一介绍。

11.2.1　JDBC-ODBC桥驱动

第一类 JDBC－ODBC 桥驱动器使 Java 应用程序可以进行 JDBC 调用，这时一般不需要额外下载 JDBC 驱动器，只需要配置数据源。如图 11.2 描述了这一类 JDBC 驱动器的典型配置。

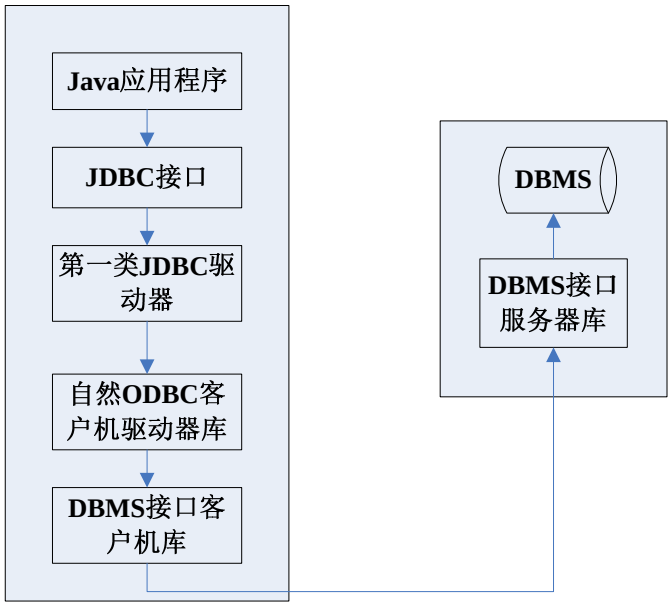

图 11.2　第一类 JDBC－ODBC 桥驱动器

一般 JDK 中已经包括了 JDBC－ODBC 桥接的驱动接口，而 ODBC 早就成为各类数据库通用的驱动方式，所以开发者只需要配置相应的 ODBC 数据源即可，而无须再安装另类驱动程序包。并且这类驱动方法，需要客户端安装数据库客户机库，才能连接到相应的数据库服务器上进行数据库操作。

这一类数据库驱动的缺点就是不能提供非常好的性能，一般不适合实际系统的应用，而仅仅使用在开始项目开发过程中。

在 Windows 操作系统中可以通过数据源来配置 ODBC 数据库连接。操作步骤如下所示：

（1）打开“控制面板”|“管理工具”|“数据源”，弹出如图 11.3 所示的“ODBC 数据源管理器”对话框。

（2）单击“添加”按钮，弹出“创建新数据源”对话框，如图 11.4 所示。

图 11.3　添加数据库

图 11.4　选择数据库

（3）在其中选择所需要的数据库类型，这里以 Sql Server 为例。然后单击“完成”按钮，弹出如图 11.5 所示的对话框。

（4）给选择的数据库起一个名称，并进行适当的描述。然后在下拉框中选择数据库的服务器地址，

本处选择当地（本机）安装的数据库，也可以通过 IP 地址指定一个远程的数据库。单击“下一步”弹出图 11.6 所示的对话框。

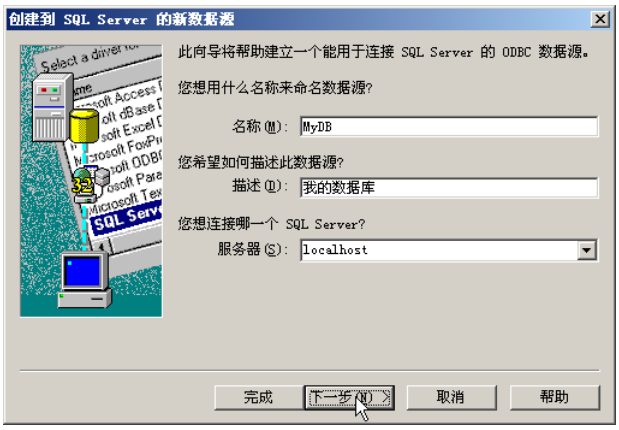

图 11.5　数据库取名操作

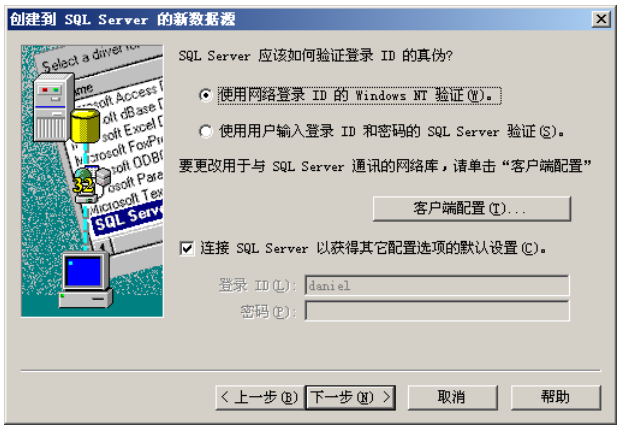

图 11.6　验证方式选择

（5）选择数据库登录验证方式，此处使用了 Window NT 验证方法，所以如图 11.6 配置即可。然后单击“下一步”按钮，弹出如图 11.7 所示的对话框。

（6）选择相应的数据库，本书选择 tempDB 数据库，然后单击“下一步”按钮，弹出图 11.8 所示的对话框。

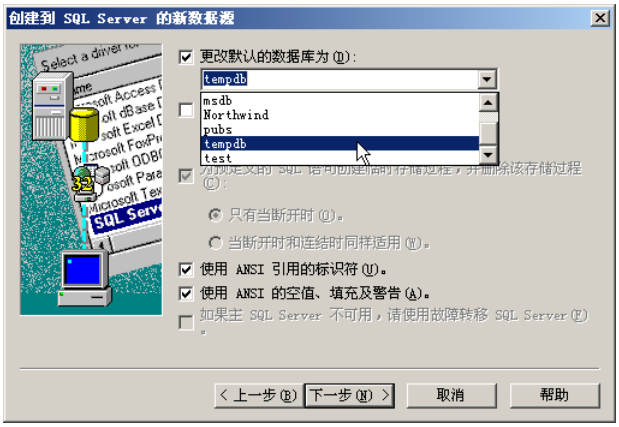

图 11.7　数据库选择

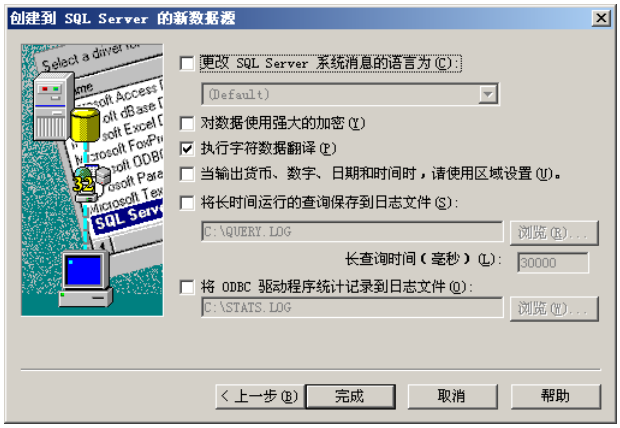

图 11.8　验证方式选择

（7）从弹出的对话框中，配置系统信息的所使用的语言，这里选择默认或者简体中文。配置完成之后，单击“完成”按钮完成数据库的配置，弹出 11.9 图所示的对话框。

（8）从弹出的对话框中可以看到所有的配置信息，确认正确时，即可单击“确定”按钮完成配置。如果单击“测试数据源”可以查看数据库是否能够正确连接。测试正确的界面如图 11.10 所示。

图 11.9　所有配置信息的确定

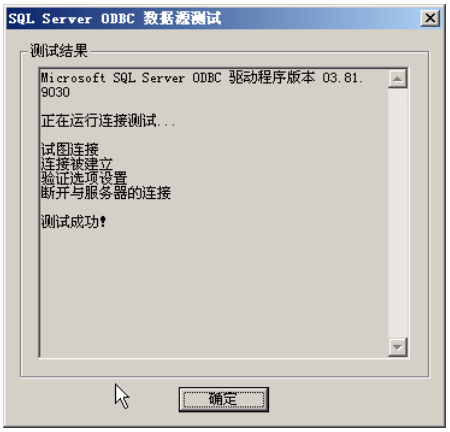

图 11.10　测试成功

11.2.2　自然API部分支持Java技术驱动器

这类驱动器可以直接映射特定数据库厂家的 DBMS 自然库接口进行调用。这样，JDBC 驱动器被设计成调用 Oracle、Sybase 或者其他数据库管理系统所提供的自然客户机库。大部分数据库厂家的数据库中都带有 JDBC 第二类驱动器。这类驱动器要比 JDBC－ODBC 桥提供更好的操作性能，因为它们越过了 ODBC 中间层。

这类驱动器和第一类驱动器一样都需要在客户端安装相应的 DBNS 接口客户机库，通过客户机库再连接特定的 DBMS。该类驱动器原理如图 11.11 所示。

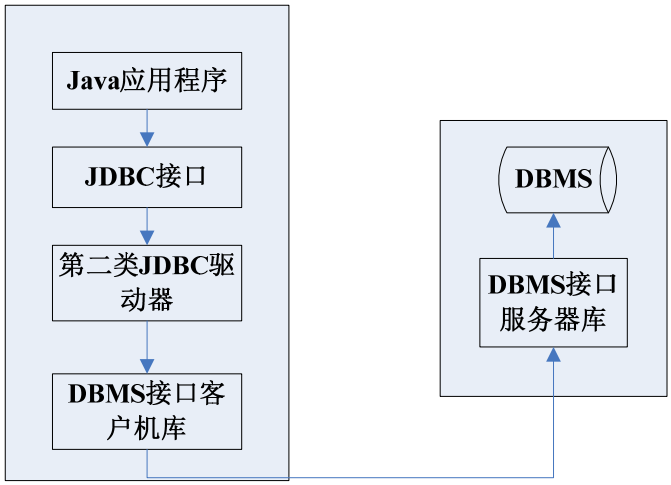

图 11.12　第二类数据库驱动器原理

和第一类驱动器相对比，可以看出该驱动直接跳过了 ODBC 中间层，所以能够更好地提供数据库操作性能。但是和第一类驱动器原理一样，需要在数据库客户机上安装 DBMS 接口客户机库。

该类驱动器的一般配置步骤如下：

（1）按照 DBMS 厂家说明对要连接的 DBMS 安装自然 DBMS 接口客户机库。

（2）在 Java 客户机应用程序中装入第二类 JDBC 驱动器类，该方法会在下面的小节中重点介绍。

11.2.3　网络协议完全支持Java技术驱动器

该类驱动可以使客户机 Java 应用程序进行 JDBC 调用，它首先映射到 DBMS 厂家无关的网络协议中间件，然后再由这些网上中间件映射到特定的 DBMS 厂家的接口。具体调用过程可见图 11.13 所示。

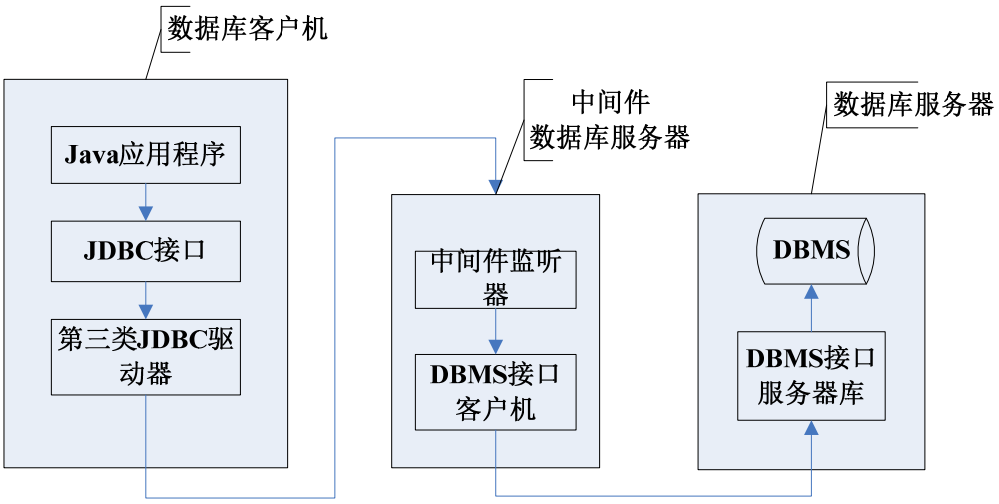

图 11.12　第三类 JDBC 驱动器

其中厂家无关中间件监听器是由其他中间件厂家来实现的，可以支持连接不同后端数据库类型，大大加强数据库连接的兼容性。这样，Java 客户机应用程序不再需要信赖客户端安装的自然库，可以比较灵活地和不同后端数据库进行通信。因此，特定 DBMS 厂家的接口配置放在中间件数据库服务器本地。尽管中间件监听器厂家要选择与中间件监听器通信的网络协议，但许多厂家实现相当开放的方案，可以在 Internet 与 Intranet 环境中使用。

该类驱动器的配置过程如下：

（1）在中间件服务器上按 DBMS 厂家说明对要连接的 DBMS 安装自然 DBMS 接口客户机库，但是这时客户机应用程序可以完全独立出来，更多不需要考虑后端数据库的类型。

（2）按中间件厂家说明安装中间件远程网络监听器的中间件组件库。

（3）将中间件远程网络监听器配置成使用一个或几个自然安装的 DBMS 接口，用第 1 类、第 2 类或者第 4 类 JDBC 驱动器连接数据库厂家的远程网络监听器。

（4）在 Java 客户机应用程序装入中间厂家的第 3 类 JDBC 驱动器类。

11.2.4　自然协议完全支持Java技术驱动器

该类驱动器的典型配置如图 11.13 所示。

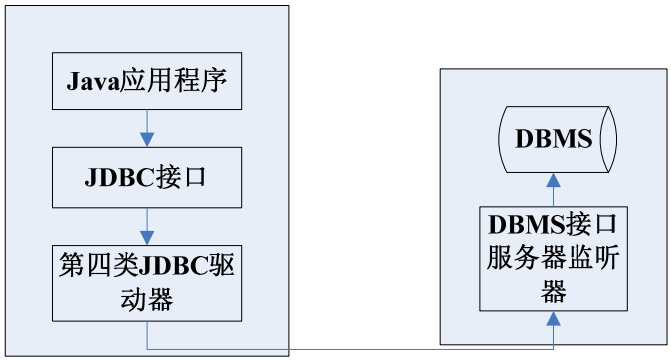

图 11.13　自然协议完全支持 Java 技术驱动器

该类驱动器实现网络直接访问厂家的数据库。和第三类驱动器相类似，第四类驱动器中客户机 Java 应用程序也不必要依赖于客户端安装的自然库。但是，由于这些驱动器是厂家特定的，即该类驱动器只对应一种 DBMS，因此这类驱动器方案在客户端的配置并不灵活，客户机只能与这个厂家的 DBMS 通信。

该类驱动器的配置过程如下：将要连接的 DBMS 对应的特定驱动器安装在客户机 Java 应用程序中即可，下一小节将具体介绍 JDBC 的驱动器配置。

11.3　JDBC驱动器配置

了解 JDBC 驱动器及其类型后，一定要知道如何在客户机 Java 应用程序中配置 JDBC 驱动器。Java.sql.DriverManager 类是 JDBC API 用户直接装入与管理 JDBC 驱动器的主接口。DriverManager 可以通过设置系统属性 jdbc.drivers 隐式装入驱动器类，也可以用 Class.forName()方法来显式装入驱动器类。另外，通过 DriverManager 还可以生成新的数据库连接，与所装入驱动器的池相关联。

11.3.1　隐式装入驱动器类

要隐式装入驱动器类，只要设置系统属性 jdbc.drivers，指定用冒号分开的驱动器类名。注意，指定的驱动器类名必须是在 CLASSPATH 中。如下装入两个驱动器类的方法：

```
java －D jdbc.drivers=sun.jdbc.odbc.JdbcOdbcDriver: com.assuredtech.jdbc.JdbcDriver MyProgram
```

代码说明：后面 MyProgram 指定所要使用的 Web 模块。隐式方式载入驱动器在实际开发过程中不经常使用，读者只要了解即可。

11.3.2　显式装入驱动器类

要显式装入驱动器类，需要在操作数据库的 Java 类中使用 Class.forName()方法，基本操作如下：

```
Class.forName(“com.assuredtech.jdbc.JdbcDriver”);
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
```

代码说明：以上代码依次装载了两个驱动器类。一个对应用程序进行 JDBC 驱动器配置的灵活模式是按 java.util.Properties 对象可读格式在文件中提供一组 JDBC 驱动器名，然后读取并装入这些驱动器名，对每个驱动器调用 Class.forName()方法。

注意：有时在实际开发过程中，还会使用 JNDI 来访问已经设置好的数据源对象，此小节所介绍的方法不适合该方法。至于这方面的介绍将在以后章节中重点讲解。

11.4　JDBC连接

在上一小节介绍完了如何配置 JDBC 驱动后，下面将要介绍在程序中对 JDBC 的操作。首先需要从 JDBC 驱动器中取得数据库连接的句柄。数据库连接表示通信会话，打开通信信道和在数据库客户机与数据库服务器之间执行 SQL 语句。java.sql.Connection 接口表示这个数据库连接。数据库客户机与数据库之间的会话终止时，通过 Connection.close()方法来关闭数据库连接。

11.4.1　数据库URL

数据库 URL（统一资源定位器）表示完全限定数据库连接名，标识要连接的数据库和数据库驱动器。要完全描述连接名，就要知道数据库驱动器类型、数据库类型、数据库连接名类型和一些连接实例信息等，例如用户名、密码与数据库实例的 IP 地址。数据库 URL 表示的一般形式如下：

```
Jdbc:subprotocol:subname
```

其中 jdbc 是所有数据库 URL 中使用到的数据库驱动器类型关键字；subprotocol 表示要连接的数据库类型；subname 提供这个数据库类型建立所需的其他信息。

特定数据库实例使用的 subprotocol 与 subname 应在 JDBC 驱动器厂家的文档中描述。

11.4.2　建立连接

要用 DriverManager 建立连接对象实例，就要调用 DriverManager 的一个 getConnection()方法，用数据库 URL 作为参数。在调用这个方法时，DriverManager 首先从装入的驱动器池（这里可能使用 Class.forName()装载了多个数据库驱动器）中找到能接收这个数据库 URL 的驱动器，然后让驱动器连接具有相应数据库 URL 的数据库。然后向调用 DriverManager.getConnection()方法的对象返回一个 Connection 对象。

其中 DriverManager 类中有三个 getConnection()方法：

- getConnection(String url)连接指定数据库 URL。
- getConnection(String url,String user,String password)连接指定数据库 URL、数据库用户名和数据库用户密码。
- getConnection(String url,java.util.Properties info)连接指定数据库 URL 和一组作为连接变元的属性。有时通过 Properties 对象传入 user 与 password 属性。

作为选项，也可以调用 DriverManager.setLoginTimeout()方法，指定驱动器连接数据库时等待的超时秒数。下面通过一个实例来演示 getConnection()方法的使用，详细代码如下：

```
Package com.db;
import java.sql.*;

public class ConnectDB {
    //获取数据库连接 Connection 对象
    public static Connection getCon()
    {
```

```
        Connection con = null;                                          //定义一个数据库连接 Connection
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");              //装载数据库驱动
            String url = "jdbc:odbc:qqNews";                            //定义数据库 URL
            String userID = "sa";                                       //数据库登录用户名
            String Password = "12345";                                  //数据库登录密码
            con = DriverManager.getConnection(url,userID,Password);     //DriverManager 生成数据库连接
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        return con;
    }
}
```

代码说明：以上创建的 ConnectDB 类中只包含一个 getCon()方法。“import java.sql.*”语句是将 java.sql 包中的所有类导入，也可以指定某一个类导入，例如“import java.sql.DriverManager”，本类需要使用到 java.sql 包中的 DriverManager 和 Connection 类。另外，本类连接数据库所使用的驱动器是 JDBC－ODBC 桥。getCon()方法返回一个 Connection 对象实例。

11.5　JDBC语句

获取 Connection 对象后，就可以进行数据库查询操作了。使用 Connection 对象可以生成 statement 实例。statement 接口中包含了很多的数据库基本操作语句。

11.5.1　Statement语句

这一章将要介绍 statement 接口中的大部分常用方法。使用 Connection 对象中的 createStatement()方法来生成 statement 对象的语句如下：

```
Statement statement = connection.createStatement();
```

这样就获得了 Statement 对象实例，然后就可以调用 Statement 类中的方法来进行数据库操作，下面列出了执行实际 SQL 命令的三个方法：

（1）ResultSet executeQuery(String sql)可以执行 SQL 查询并获取到 ResultSet 对象。例如：

```
ResultSet rst = statement.executeQuery("select * from users");
```

代码说明：其中 statement 对象实例是由 Connection 对象中的 createStatement()方法获取到。“select * from users”语句是查询出数据库表 users 中的所有信息。调用 Statement 对象中的 executeQuery()方法返回一个 ResultSet 数据集对象。

（2）int executeUpdate(String sql)可以执行 SQL 插入、删除以及更新操作，如果是更新会返回更新行数。例如：

```
int nValue = statement.executeUpdate("insert into users values('1010','jack','23','男')");
```

（3）boolean execute(String sql)是一个最为一般的执行方法，可以执行 SQL DDL 命令，然后取得一个布尔值，表示是否返回 ResultSet。例如：

```
Boolean returnValue ＝ satatement.execute(“select * from users”);
```

11.5.2　Statement实例介绍

下面通过实例来演示 statement 对象中执行语句的使用，该实例将从用户表中返回所有用户信息，并打印在页面中。代码如下：

```
package com.DB;
import java.sql.*;

public class ConnectDB {
	//获取数据库连接 Connection 对象
	public static Connection getCon()
	{
		Connection con = null;                                          //定义一个数据库连接 Connection
		try
		{
			Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");              //装载数据库驱动
			String url = "jdbc:odbc:qqNews";                            //定义数据库 URL
			String userID = "sa";                                       //数据库登录用户名
			String Password = "12345";                                  //数据库登录密码
			con = DriverManager.getConnection(url,userID,Password);     //DriverManager 生成数据库连接
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
		return con;
	}
	/**
	 * @主调用方法
	 */
	public static void main(String[] args) {

		Statement stmt = null;                  //定义 Statement 对象
		ResultSet   rst = null;                 //定义 ResultSet 对象
		ConnectDB cdb= new ConnectDB();         //初始化一个 ConnectDB（本身）对象实例
		try
		{
		 stmt = (cdb.getCon()).createStatement();//取得 Statement 实例
		 rst = stmt.executeQuery("select * from userinfo");             //取得数据集
		 while(rst.next())                                              //循环输出 ResultSet 集中数据
		 {
			System.out.print("ID:"+rst.getString(1)+" ");
			System.out.println("Name:"+rst.getString(2)+"; ");
		 }
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
	}
```

```
}
```

代码说明：“ConnectDB cdb= new ConnectDB();”代码初始化 ConnectDB 实例。然后调用“cdb.getCon()”返回一个数据库连接 Connection 对象，再调用 Connection 对象中的 createStatement()方法来返回一个 Statement 实例。执行 statement 对象中的 executeQuery()方法来返回一个数据集 ResultSet。最后，依次循环输出数据集 ResultSet 中的所有数据信息。

11.6　准备语句PreparedStatement

上一小节介绍的 Statement 对象可以表示一个 SQL 语句，要在每次执行时编译，但也可以使用准备语句，就是预编译的 SQL 语句，即 java.sql.PreparedStatement 接口标识。

11.6.1　PreparedStatement语句

PreparedStatement 要比普通的 Statement 对象使用更加灵活，它可以用参数化 SQL 语句生成。使用 PreparedStatement 生成的 SQL 命令中使用问号来预留一个参数位，表示一个输入变量。在执行 preparedStatement 生成的语句之前可以动态地设置这些预留的参数。显然这样的方式要比 Statement 更加灵活。PreparedStatement 对象是通过 Connection 对象的 preparedStatement()方法来生成的，具体语句如下：

```
PreparedStatement pstm = connection.preparedStatement(“select * from users where userId = ?”);
```

代码说明：以上代码通过 PreparedStatement 对象来生成一个 SQL 命令，其中使用问号来预留一个参数位。在执行该 SQL 语句之前，是可以通过调用 PreparedStatement 对象中的 setXxx()方法（Xxx 根据参数的类型变化）来设置该预留的参数值。

通过 setXxx()方法设置完参数后，同样可以调用 PreparedStatement 对象中的 executeQuery()、executeUpdate()或者 execute()方法来执行 SQL 语句。

11.6.1　PreparedStatement实例介绍

下面通过修改上一小节实例的主调用函数来演示 PreparedStatement 对象的使用方法，具体修改代码如下：

```
public static void main(String[] args) {

    PreparedStatement pstmt = null;          //定义 Statement 对象
    ResultSet   rst = null;                  //定义 ResultSet 对象
    ConnectDB cdb= new ConnectDB();          //初始化一个 ConnectDB（本身）对象实例
    try
    {
      pstmt = (cdb.getCon()).preparedStatement("insert into userinfo values(?,?,?,?)");
      pstmt.setInt(1,1002);
      pstmt.setString(2,'tom');
      pstmt.setInt(3,24);
      pstmt.setString(4,'男');
      rst = pstmt.executeUpdate();           //取得数据集
    }
```

```
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }
```

代码说明：首先通过 Connection 中的 preparedStatement()方法来生成由 PreparedStatement 对象定义的 SQL 语句。然后通过 PreparedStatement 对象中的赋值语句来给预留的参数设值。最后，调用 executeUpdate()方法来执行 SQL 语句。

11.7　结果集ResultSet

执行查询语句就是为了能够取得数据结果，在 JDBC 中的 java.sql.ResultSet 接口包装的对象就是用来表示查询的 0 个或者多个结果。ResultSet 中每个结果表示一个数据库行信息，并且可以跨一个或者多个表。ResultSet 接口中包含大量方法，基本都是简单的类型安全 getXxx()与 updateXxx()方法。下面将对常用的方法作介绍。

11.7.1　操纵结果集

Statement 接口及其子接口（例如 PreparedStatement 接口）执行 executeQuery()方法时返回 ResultSet 对象。ResultSet 对象还可以在 Statement 对象执行 execute()方法时返回 Statement 对象，但不是直接从方法调用返回，而是用 Statement 对象的 getResultSet()方法取得所返回 ResultSet 对象的句柄。如果没有 ResultSet 对象或 SQL 更新命令返回更新数，则 getResultSet 方法返回 null 值。

由于 execute()语句执行的 SQL 语句有时产生多个 ResultSet 对象，因此可以调用 Statement 接口中的 getMoreResults()方法，转入下一个 ResultSet，返回一个真布尔值，表示返回的是 ResultSet。调用 getMoreResults()方法时，关闭从 getResultSet()返回的任何现有结果集。如果是进行的更新当前结果，则 execute()方法可能产生多个更新，getMoreResults()方法返回假值。这样，如果 getResultSet()方法返回 null 而 getMoreResults()返回假值，则可以用 getUpdateCount()方法返回当前结果中的更新数。

通常使用 Statement 或者 PreparedStatement 对象获取到 ResultSet 对象的句柄之后，需要从每一行取得各个列数据，然后对下一行继续操作，直到数据全部取完。其中使用 ResultSet 对象中的 next()方法来操作行的滚动，而使用 getXxx()方法来获取每列的数据。next()方法一般返回一个布尔类型值，如果为真则把将游标移到下一行，否则结束操作。

getXxx()方法有两种形式。一种形式取列索引 int，另一种形式则取列名为参数。每个 getXxx()方法返回一个类型值，具体对应 getXxx()方法中请求的实际 Xxx 类型。例如下面一段代码：

```
String firstName = resultSet.getString(2);
String firstName = resultSet.getString(‘FIRST_NAME”);
```

代码说明：上面两段代码是等价的，getString(2)表示获取该行的第二列数据；而下一个语句是通过列名 FIRST_NAME 来获取该列数据的。

11.7.2　ResultSet实例介绍

下面还是通过修改上面小节的实例来演示 ResultSet 接口的操作方法，具体代码如下：

```
    public static void main(String[] args) {
```

```
        Statement stmt = null;                    //定义 Statement 对象
        ResultSet   rst = null;                   //定义 ResultSet 对象
        ConnectDB cdb= new ConnectDB();           //初始化一个 ConnectDB（本身）对象实例
        try
        {
         stmt = (cdb.getCon()).createStatement(); //取得 Statement 实例
         rst = stmt.executeQuery("select * from users");                    //取得数据集
         while(rst.next())                                                  //循环输出 ResultSet 集中数据
         {
               System.out.print("用户编号："+rst.getString(1)+" ");
               System.out.println("用户姓名："+rst.getString("Name")+"; ");
               System.out.println("用户年龄："+rst.getString(3)+"; ");
               System.out.println("用户性别："+rst.getString("Sex")+"; ");
         }
        }
        catch(Exception e)
        {
              e.printStackTrace();
        }
    }
```

代码说明：代码中使用 ResultSet 对象的 next()方法来进行行的滚动操作，通过 getXxx()方法的两种形式来读取行的列数据。

11.8　SQL基本语句

SQL 语句是一种标准化的结构查询语言，各种不同关系的数据库，SQL 语句大同小异。上面小节中已经多多少少涉及到了 SQL 语句的编写。这一小节将对基本的 SQL 语句作重点介绍。

11.8.1　SQL查询语句

在 SQL 语句中，最为常见的就是 select 查询语句，其基本的语法如下：

```
select selectList [into new_table]
from tableName
[where search_condition ]
[group by group_by_expression ]
[having search2_condition ]
[order by order_expression [asc | desc] ]
```

其中 selectList 是要查询的内容，例如字符名称的列表；[into new_table]是要将查询出的结果存储到另一张新表中；tableName 为需要查询的数据库表；search_condition 用来制定查询条件；group_by_expression 是用来指定查询分组的条件；search2_condition 为指定分组后组内的查询条件；order_expression [asc | desc]用来指定排序的方式，以及按照什么字段排序。

例如最为简单的 SQL 查询语句：

```
Select userId, name, description from user;
```

上面的查询语句是用来从 user 数据库表中所有用户的信息，其中只返回 userId，name 与 description

这三个字段信息。

如果需要查询出所有字段的信息，则上面语句改为：

```
Select * from user where name='zzb';
```

11.8.1.1　使用 where 子句进行条件查询

有时，用户在查询数据库时，往往不需要了解全部信息，而只需要其中一部分满足条件的信息，这时就需要使用到 where 字句。Where 字句中一般由表达式以及逻辑联结词 And、or、not、between 等组成。

例如需要从员工表中查询出介于 2000 元和 3000 元之间的员工姓名：

```
Select e_name from employee where e_wage between 2000 and 3000;
```

或者

```
Select e_name from employee where e_wage > 2000 and e_wage < 3000;
```

11.8.1.2　使用 order 子句对结果进行排序

当用户要对查询结果进行排序时，这时可以在 select 语句中加入 order by 字句。在 order by 字句中可以使用一个或者多个排序条件，其优先级是从左到右。

例如查询工作级别为‘2‘的员工姓名，查询结果按工资排序：

```
select e_name from employee where job_levle='2' order by e_wage;
```

该 SQL 默认按照降序排序，所以结果将按照工资的由高到低进行排序。

11.8.1.3　使用 group 子句对查询结构分组

当用户要对查询结果进行分组时，就需要在 select 语句中加入 group 字句。

例如需要查询工作级别为‘2’的员工姓名，查询结果按照部门分组：

```
select e_name, dept_id from employee
where job_level = '2'
group by dept_id,e_name
```

该语句的查询结果将按照部门 id 和名称进行分组显示。

11.8.1.4　使用 having 子句对分组进行条件查询

Having 字句使用来选择特殊组，它将组的一些属性与常数值进行比较，如果一个组满足 having 子句中的逻辑表达式，它就可以包含在查询结果中。

例如查询有多个员工的工资不低于 6000 的部门编号：

```
Select dept_id,count(*) from employee
Where e_wage >= 6000
Group by dept_id
Having count(*) > 1
```

查询结果将按照部门号进行分组显示，并且需要满足该组中的员工数超过 1 名。该 SQL 语句中还使用到了 count()统计函数，用来计算所有信息条数。

11.8.1.5　连接查询

连接查询一般用来对多个表中的数据进行提取、组合。简单说，如果一个查询需要对多个表进行操作，那么就可以称该查询为连接查询。

例如需要查询订货表中的所有产品的编号以及名称：

```
Select distinct orders.p_id, p_name
From oders, products
Where orders.p_id = product.p_id and orders.num > 1000;
```

其中 distinct 字段是用来指定返回未重复行信息。

11.8.2　SQL插入语句

数据库中的信息是经常需要更新的，用户需要添加数据，insert 语句提供了该功能。Insert 语句通常有两种形式。一种是插入一条记录；另一种是插入子查询的结果。后者可以一次插入多条记录信息。

11.8.2.1　插入单行

例如插入单行数据到订购商信息表中：

```
Insert firms(firm_id, f_name, f_intro) values(10070001,'SQL','制作数据库软件的公司');
```

11.8.2.2　插入子查询结果

子查询不仅可以嵌套在 select 语句中，用以构造父查询的条件，也可以嵌套在 insert 语句中，用以生成要插入的数据。

例如要对每个部门，求员工总数，并将结果存入到 department_info 表中：

```
Insert into department_info(dept_id, e_num)
Select dept_id, count(*) from employee
Group by dept_id;
```

该 SQL 语句将从 employee 数据库表中的查询信息存储到 department_info 数据库表中，这里需要字段的对应。

11.8.3　SQL删除语句

delete 语句用来从表中删除数据。

11.8.3.1　删除一行记录

例如将编号为‘10031011’的员工信息从数据库表中删除：

```
Delete from employee where emp_id = '10031011';
```

11.8.3.2　删除多行记录

例如需要删除所有的部门信息：

```
delete from department_info
```

该 SQL 执行后，将 department_info 表中的所有信息全部删除。

11.8.4　SQL更新语句

另外，用户还可以使用 update 语句来更新一张表中的一列或者多列数据信息。

11.8.4.1　更新一条记录的值

例如需要将部门编号为“1002“的部门名称改名为“财务部”：

```
Update department set d_name="财务部" where dept_id="1002";
```

11.8.4.2　更新多个记录的值

例如要将所有员工的工资水平上涨 100 元：

```
Update employee set e_wage = e_wage + 100;
```

或者，又例如将工作级别为“3”的所有员工工资上涨 5%：

```
Update employee set e_wage = e_wage*(1+5%) where job_level = '3';
```

11.9　SQL与Java映射

本章的一些实例演示了如何从 ResultSet 对象中取得列数据的 Java 类型。有时数据库厂家可能让你用其类型名编写应用程序，但操纵数据时更移植的方法是按照 JDBC 标准 SQL 与 Java 映射。下面表列出了标准和常用 SQL 与 Java 映射以及标准和常用 Java 与 SQL 映射关系。

表 11.1　SQL与Java类型、Java与SQL类型映射

SQL类型与Java类型映射		Java类型与SQL类型映射	
SQL类型	**Java类型**	**Java类型**	**SQL类型**
CHAR	java.sql.String	String	CHAR,VARCHAR,LONGVARCHAR
VARCHAR	java.sql.String	boolean	BIT,BOOLEAN
LONGVARCHAR	java.sql.String	byte[]	BINARY,VARBINARY,LONGVARBINARY
BIT	boolean	byte	TINAYINT
BOOLEAN	boolean	short	SMALLINT
BINARY	byte[]	int	INTEGER
VARBINARY	byte[]	long	BIGINT
LONGVARBINARY	byte[]	float	REAL
TINYINT	byte	double	FLOAT,DOUBLE
SMALLINT	short	java.math.BigDecimal	NUMERIC,DECIMAL
INTEGER	int	java.sql.Date	DATE
BIGINT	long	java.sql.Time	TIME
REAL	float	java.sql.Timestamp	TIMESTAMP
FLOAT	double		
DOUBLE	double		
NUMERIC	java.lang.math.BigDecimal		
DECIMAL	java.lang.math.BigDecimal		
DATE	jva.sql.Date		
TIME	jva.sql.Time		
TIMESTAMP	jva.sql.Timestamp		

由上表可知，当 Java 中的 String 类型映射 SQL 类型时，通常映射为 VARCHAR，但是 String 也可以映射成 CHAR 或者 LONGVARCHAR，只要 String 的长度低于或者超过某一个驱动器特定长度的极限。同样 byte 数组类型也可以根据指定的特定长度来选择映射为 BINARY、VARBINARY 或是 LONGVARBINARY 类型。

DECIMAL 与 NUMERIC 类型在需要绝对精度的时候使用。这样，货币的数据库值通常用 DECIMAL 与 NUMERIC 类型返回。这些类型也可以用 ResultSet 对象的 getString()方法来读取。

Java.util.Date 类没有对某些数据库时间标志值提供足够的粒度，还包括不适合 SQL 类型把日期与时间分割成不同实体的日期与时间信息。为此，需要使用三个从 java.util.Date 派生出来的类来作为基础 JDBC 类型帮助器，这个类分别为：java.util.Date、java.sql.Time 以及 java.sql.Timestamp 类型。下面一一介绍这三个类：

- java.util.Date 截尾任何非日期相关时间信息，将 java.util.Date 的时、秒、毫秒字段存放成 0。
- java.sql.Time 截取任何日期相关信息，将 java.util.Date 的时、年、月、日存放成“0 日期”（从 1970 年 1 月 1 日开始）。
- java.sql.Timestamp 在 java.util.Date 提供的时间中添加纳秒精度。

11.10　本章小结

本章主要对数据库连接包 JDBC 作了简要的介绍，其中包括驱动器如何装载、如何进行数据库的连接、操作 JDBC 语句的 Statement 和 PreparedStatement 存在的不同点、ResultSet 结果集的使用。通过这些内容的讲解，读者应该学会数据库的基本操作。至于数据源、JNDI 以及连接池和事务处理的概念将在后面章节中陆续介绍。