JSP程序设计教程

第8章 JSP数据库应用开发

第8章 JSP数据库应用开发

- 8.1 数据库管理系统 √
- 8.2 JDBC概述 √
- 8.3 JDBC中的常用接口 √
- 8.4 连接数据库 √
- 8.5 典型JSP数据库连接 √
- 8.6 数据库操作技术 √
- 8.7 连接池技术 √

人民邮电出版社
POSTS & TELECOM PRESS

课件制作人：王国辉

8.1 数据库管理系统

JSP可以访问并操作很多种数据库管理系统，如SQL Server、MySQL、Oracle、Access、DB2、Sybase和PostgreSQL数据库等，下面介绍几种常用的数据库管理系统。

8.1.1 SQL Server 2000数据库 √

8.1.2 MySQL数据库 √

8.1.3 Oracle数据库 √

8.1.4 Access数据库 √

8.1.1 SQL Server 2000数据库

SQL Server的全称是Microsoft SQL Server，是由美国微软公司制作并发布的一种性能优越的关系型数据库管理系统（Relational Database Management System，RDBMS），具有强大的数据库创建、开发、设计和管理功能。目前，SQL Server数据库在中小型项目上应用非常广泛。

1. 安装SQL Server 2000
2. 安装SQL Server 2000 SP4补丁
3. 创建数据库
4. 创建数据表

8.1.2 MySQL数据库

MySQL是目前最为流行的开放源码的数据库，是完全网络化的跨平台的关系型数据库系统，它是由MySQL AB公司开发、发布并支持的。任何人都能从Internet下载MySQL软件，而无需支付任何费用，并且“开放源码”意味着任何人都可以使用和修改该软件，如果愿意，用户也可以研究源码并进行恰当的修改，以满足自己的需求，不过需要注意的是，这种“自由”是有范围的。

8.1.3 Oracle数据库

ORACLE系统是由以RDBMS为核心的一批软件产品组成，可在多种硬件平台上运行，例如微机、工作站、小型机、中型机和大型机等，并且支持多种操作系统，用户的ORACLE应用可以方便地从一种计算机配置移至另一种计算机配置上。ORACLE的分布式结构可将数据和应用驻留在多台计算机上，并且相互间的通信是透明的。ORACLE支持大数据库、多用户的高性能的事务处理，数据库的大小甚至可以上千兆。支持大量用户同时在同一数据上执行各种数据应用，并使数据争用最小，保证数据一致性。Oracle数据库系统维护具有很高的性能，甚至每天可24h连续工作，正常的系统操作（非计算机系统故障）不会中断数据库的使用。

8.1.4 Access数据库

Access数据库管理系统是Microsoft Office系统软件中的一个重要组成部分，它是一个关系型桌面数据库管理系统，可以用来建立中、小型的数据库应用系统，应用非常广泛。同时，由于Access数据库操作简单、使用方便等特点，许多小型的Web应用程序也采用Access数据库。

8.2 JDBC概述

JDBC是用于执行SQL语句的API类包，由一组用Java语言编写的类和接口组成。JDBC提供了一种标准的应用程序设计接口，通过它可以访问各类关系数据库。下面将对JDBC技术进行详细介绍。

8.2.1 JDBC技术介绍 √

8.2.2 JDBC驱动程序 √

8.2.1 JDBC技术介绍

JDBC的全称为Java DataBase Connectivity，是一套面向对象的应用程序接口（API），制定了统一的访问各类关系数据库的标准接口，为各个数据库厂商提供了标准接口的实现。通过JDBC技术，开发人员可以用纯Java语言和标准的SQL语句编写完整的数据库应用程序，并且真正地实现了软件的跨平台性。在JDBC技术问世之前，各家数据库厂商执行各自的一套API，使得开发人员访问数据库非常困难，特别是在更换数据库时，需要修改大量代码，十分不方便。JDBC的发布获得了巨大的成功，很快就成为了Java访问数据库的标准，并且获得了几乎所有数据库厂商的支持。

8.2.1 JDBC技术介绍

JDBC是一种底层API，在访问数据库时需要在业务逻辑中直接嵌入SQL语句。由于SQL语句是面向关系的，依赖于关系模型，所以JDBC传承了简单直接的优点，特别是对于小型应用程序十分方便。需要注意的是，JDBC不能直接访问数据库，必须依赖于数据库厂商提供的JDBC驱动程序，通常情况下使用JDBC完成以下操作：

（1）同数据库建立连接；

（2）向数据库发送SQL语句；

（3）处理从数据库返回的结果。

8.2.1 JDBC技术介绍

JDBC具有下列优点：

（1）JDBC与ODBC十分相似，便于软件开发人员理解；

（2）JDBC使软件开发人员从复杂的驱动程序编写工作中解脱出来，可以完全专著与业务逻辑的开发；

（3）JDBC支持多种关系型数据库，大大增加了软件的可移植性；

（4）JDBC API是面向对象的，软件开发人员可以将常用的方法进行二次封装，从而提高代码的重用性。

8.2.1 JDBC技术介绍

与此同时，JDBC也具有下列缺点：

（1）通过JDBC访问数据库时速度将受到一定影响；

（2）虽然JDBC API是面向对象的，但通过JDBC访问数据库依然是面向关系的；

（3）JDBC提供了对不同厂家的产品的支持，这将对数据源带来影响。

8.2.2 JDBC驱动程序

JDBC驱动程序是用于解决应用程序与数据库通信的问题，它可以分为JDBC-ODBC Bridge、JDBC-Native API Bridge、JDBC-middleware和Pure JDBC Driver4种，下面分别进行介绍。

1. JDBC-ODBC Bridge

JDBC-ODBC Bridge是通过本地的ODBC Driver连接到RDBMS上。这种连接方式必须将ODBC二进制代码（许多情况下还包括数据库客户机代码）加载到使用该驱动程序的每个客户机上，因此，这种类型的驱动程序最适合于企业网，或者是利用Java编写的3层结构的应用程序服务器代码。

8.2.2 JDBC驱动程序

2. JDBC-Native API Bridge

JDBC-Native API Bridge驱动通过调用本地的native程序实现数据库连接，这种类型的驱动程序把客户机API上的JDBC调用转换为Oracle、Sybase、Informix、DB2或其他DBMS的调用。需要注意的是，和JDBC-ODBC Bridge驱动程序一样，这种类型的驱动程序要求将某些二进制代码加载到每台客户机上。

3. JDBC-middleware

8.2.2 JDBC驱动程序

JDBC-middleware驱动是一种完全利用Java编写的JDBC驱动，这种驱动程序将JDBC转换为与DBMS无关的网络协议，然后将这种协议通过网络服务器转换为DBMS协议，这种网络服务器中间件能够将纯Java客户机连接到多种不同的数据库上，使用的具体协议取决于提供者。通常情况下，这是最为灵活的JDBC驱动程序，有可能所有这种解决方案的提供者都提供适合于Intranet用的产品。为了使这些产品也支持Internet访问，它们必须处理Web所提出的安全性、通过防火墙的访问等方面的额外要求。几家提供者正将JDBC驱动程序加到他们现有的数据库中间件产品中。

8.2.2 JDBC驱动程序

4. Pure JDBC Driver

Pure JDBC Driver驱动是一种完全利用Java编写的JDBC驱动，这种类型的驱动程序将JDBC调用直接转换为DBMS所使用的网络协议。这将允许从客户机机器上直接调用DBMS服务器，是Intranet访问的一个很实用的解决方法。由于许多这样的协议都是专用的，因此数据库提供者自己将是主要来源，有几家提供者已在着手做这件事了。

8.3 JDBC中的常用接口

JDBC提供了许多接口和类，通过这些接口和类，可以实现与数据库的通信，本节将详细介绍一些常用的JDBC接口和类。

8.3.1 驱动程序接口Driver √

8.3.2 驱动程序管理器DriverManager √

8.3.3 数据库连接接口Connection √

8.3.4 执行SQL语句接口Statement √

8.3.5 执行动态SQL语句接口PreparedStatement √

8.3.6 执行存储过程接口CallableStatement √

8.3.7 访问结果集接口ResultSet √

8.3.1 驱动程序接口Driver

每种数据库的驱动程序都应该提供一个实现java.sql.Driver接口的类，简称Driver类，在加载Driver类时，应该创建自己的实例并向java.sql.DriverManager类注册该实例。

通常情况下通过java.lang.Class类的静态方法forName(String className)，加载要连接数据库的Driver类，该方法的入口参数为要加载Driver类的完整包名。成功加载后，会将Driver类的实例注册到DriverManager类中，如果加载失败，将抛出ClassNotFoundException异常，即未找到指定Driver类的异常。

8.3.2 驱动程序管理器 DriverManager

java.sql.DriverManager类负责管理JDBC驱动程序的基本服务，是JDBC的管理层，作用于用户和驱动程序之间，负责跟踪可用的驱动程序，并在数据库和驱动程序之间建立连接。另外，DriverManager类也处理诸如驱动程序登录时间限制及登录和跟踪消息的显示等工作。成功加载Driver类并在DriverManager类中注册后，DriverManager类即可用来建立数据库连接。

当调用DriverManager类的getConnection()方法请求建立数据库连接时，DriverManager类将试图定位一个适当的Driver类，并检查定位到的Driver类是否可以建立连接。如果可以，则建立连接并返回，如果不可以，则抛出SQLException异常。DriverManager类提供的常用方法如下表所示。

8.3.2 驱动程序管理器 DriverManager

方 法 名 称	功 能 描 述
getConnection(String url, String user, String password)	为静态方法，用来获得数据库连接，有3个入口参数，依次为要连接数据库的URL、用户名和密码，返回值类型为java.sql.Connection
setLoginTimeout(int seconds)	为静态方法，用来设置每次等待建立数据库连接的最长时间
setLogWriter(java.io.PrintWriter out)	为静态方法，用来设置日志的输出对象
println(String message)	为静态方法，用来输出指定消息到当前的JDBC日志流

8.3.3 数据库连接接口 Connection

java.sql.Connection接口负责与特定数据库的连接，在连接的上下文中可以执行SQL语句并返回结果，还可以通过getMetaData()方法获得由数据库提供的相关信息，例如数据表、存储过程和连接功能等信息。Connection接口提供的常用方法如下表所示。

8.3.3 数据库连接接口 Connection

方法名称	功能描述
createStatement()	创建并返回一个Statement实例，通常在执行无参数的SQL语句时创建该实例
prepareStatement()	创建并返回一个PreparedStatement实例，通常在执行包含参数的SQL语句时创建该实例，并对SQL语句进行了预编译处理
prepareCall()	创建并返回一个CallableStatement实例，通常在调用数据库存储过程时创建该实例
setAutoCommit()	设置当前Connection实例的自动提交模式，默认为true，即自动将更改同步到数据库中，如果设为false，需要通过执行commit()或rollback()方法手动将更改同步到数据库中
getAutoCommit()	查看当前的Connection实例是否处于自动提交模式，如果是则返回true，否则返回false
setSavepoint()	在当前事务中创建并返回一个Savepoint实例，前提条件是当前的Connection实例不能处于自动提交模式，否则将抛出异常
releaseSavepoint()	从当前事务中移除指定的Savepoint实例
setReadOnly()	设置当前Connection实例的读取模式，默认为非只读模式，不能在事务当中执行该操作，否则将抛出异常，有一个boolean型的入口参数，设为true则表示开启只读模式，设为false则表示关闭只读模式
isReadOnly()	查看当前的Connection实例是否为只读模式，如果是则返回true，否则返回false
isClosed()	查看当前的Connection实例是否被关闭，如果被关闭则返回true，否则返回false
commit()	将从上一次提交或回滚以来进行的所有更改同步到数据库，并释放Connection实例当前拥有的所有数据库锁定
rollback()	取消当前事务中的所有更改，并释放当前Connection实例拥有的所有数据库锁定；该方法只能在非自动提交模式下使用，如果在自动提交模式下执行该方法，将抛出异常；有一个入口参数为Savepoint实例的重载方法，用来取消Savepoint实例之后的所有更改，并释放对应的数据库锁定
close()	立即释放Connection实例占用的数据库和JDBC资源，即关闭数据库连接

8.3.4 执行SQL语句接口Statement

java.sql.Statement接口用来执行静态的SQL语句，并返回执行结果。例如，对于insert、update和delete语句，调用executeUpdate(String sql)方法，而select语句则调用executeQuery(String sql)方法，并返回一个永远不能为null的ResultSet实例。Statement接口提供的常用方法如下表所示。

8.3.4 执行SQL语句接口Statement

方法名称	功能描述
executeQuery(String sql)	执行指定的静态SELECT语句，并返回一个永远不能为null的ResultSet实例
executeUpdate(String sql)	执行指定的静态INSERT、UPDATE或DELETE语句，并返回一个int型数值，为同步更新记录的条数
clearBatch()	清除位于Batch中的所有SQL语句，如果驱动程序不支持批量处理将抛出异常
addBatch(String sql)	将指定的SQL命令添加到Batch中，String型入口参数通常为静态的INSERT或UPDATE语句，如果驱动程序不支持批量处理将抛出异常
executeBatch()	执行Batch中的所有SQL语句，如果全部执行成功，则返回由更新计数组成的数组，数组元素的排序与SQL语句的添加顺序对应数组元素有以下几种情况：①大于或等于零的数，说明SQL语句执行成功，为影响数据库中行数的更新计数；②–2，说明SQL语句执行成功，但未得到受影响的行数；③–3，说明SQL语句执行失败，仅当执行失败后继续执行后面的SQL语句时出现。如果驱动程序不支持批量，或者未能成功执行Batch中的SQL语句之一，将抛出异常
close()	立即释放Statement实例占用的数据库和JDBC资源，即关闭Statement实例

8.3.5 执行动态SQL语句接口PreparedStatement

java.sql.PreparedStatement接口继承于Statement接口，是Statement接口的扩展，用来执行动态的sql语句，即包含参数的SQL语句。通过PreparedStatement实例执行的动态SQL语句，将被预编译并保存到PreparedStatement实例中，从而可以反复并且高效地执行该SQL语句。

需要注意的是，在通过setXxx()方法为SQL语句中的参数赋值时，必须通过与输入参数的已定义SQL类型兼容的方法，也可以通过setObject()方法设置各种类型的输入参数。PreparedStatement的使用方法如下：

8.3.5 执行动态SQL语句接口 PreparedStatement

```
PreparedStatement ps = connection
            .prepareStatement("select * from table_name where id>? and
(name=? or name=?)");
ps.setInt(1, 1);
ps.setString(2, "wgh");
ps.setObject(3, "sk");
ResultSet rs = ps.executeQuery();
```

PreparedStatement接口提供的常用方法如下表所示。

8.3.5 执行动态SQL语句接口PreparedStatement

方 法 名 称	功 能 描 述
executeQuery()	执行前面包含参数的动态SELECT语句，并返回一个永远不能为null的ResultSet实例
executeUpdate()	执行前面包含参数的动态INSERT、UPDATE或DELETE语句，并返回一个int型数值，为同步更新记录的条数
clearParameters()	清除当前所有参数的值
setXxx()	为指定参数设置Xxx型值
close()	立即释放Statement实例占用的数据库和JDBC资源，即关闭Statement实例

8.3.6 执行存储过程接口CallableStatement

java.sql.CallableStatement接口继承于PreparedStatement接口，是PreparedStatement接口的扩展，用来执行SQL的存储过程。

JDBC API定义了一套存储过程SQL转义语法，该语法允许对所有RDBMS通过标准方式调用存储过程。该语法定义了两种形式，分别是包含结果参数和不包含结果参数。如果使用结果参数，则必须将其注册为OUT型参数，参数是根据定义位置按顺序引用的，第一个参数的索引为1。

为参数赋值的方法使用从PreparedStatement中继承来的setXxx()方法。在执行存储过程之前，必须注册所有OUT参数的类型；它们的值是在执行后通过getXxx()方法检索的。

CallableStatement可以返回一个或多个ResultSet实例。处理多个ResultSet对象的方法是从Statement中继承来的。

8.3.7 访问结果集接口ResultSet

java.sql.ResultSet接口类似于一个数据表，通过该接口的实例可以获得检索结果集，以及对应数据表的相关信息，例如列名和类型等，ResultSet实例通过执行查询数据库的语句生成。

ResultSet实例具有指向其当前数据行的指针。最初，指针指向第一行记录的前方，通过next()方法可以将指针移动到下一行，因为该方法在没有下一行时将返回false，所以可以通过while循环来迭代ResultSet结果集。在默认情况下ResultSet对象不可以更新，只有一个可以向前移动的指针，因此，只能迭代它一次，并且只能按从第一行到最后一行的顺序进行。如果需要，可以生成可滚动和可更新的ResultSet对象。

8.3.7 访问结果集接口ResultSet

ResultSet接口提供了从当前行检索不同类型列值的getXxx()方法，均有两个重载方法，可以通过列的索引编号或列的名称检索，通过列的索引编号较为高效，列的索引编号从1开始。对于不同的getXxx()方法，JDBC驱动程序尝试将基础数据转换为与getXxx()方法相应的Java类型，并返回适当的Java类型的值。

在JDBC 2.0 API（JDK 1.2）之后，为该接口添加了一组更新方法updateXxx()，均有两个重载方法，可以通过列的索引编号或列的名称指定列，用来更新当前行的指定列，或者初始化要插入行的指定列，但是该方法并未将操作同步到数据库，需要执行updateRow()或insertRow()方法完成同步操作。

ResultSet接口提供的常用方法如下表所示。

8.3.7 访问结果集接口 ResultSet

方法名称	功能描述
first()	移动指针到第一行；如果结果集为空则返回false，否则返回true；如果结果集类型为TYPE_FORWARD_ONLY将抛出异常
last()	移动指针到最后一行；如果结果集为空则返回false，否则返回true；如果结果集类型为TYPE_FORWARD_ONLY将抛出异常
previous()	移动指针到上一行；如果存在上一行则返回true，否则返回false；如果结果集类型为TYPE_FORWARD_ONLY将抛出异常
next()	移动指针到下一行；指针最初位于第一行之前，第一次调用该方法将移动到第一行；如果存在下一行则返回true，否则返回false
beforeFirst()	移动指针到ResultSet实例的开头，即第一行之前；如果结果集类型为TYPE_FORWARD_ONLY将抛出异常
afterLast()	移动指针到ResultSet实例的末尾，即最后一行之后；如果结果集类型为TYPE_FORWARD_ ONLY将抛出异常
absolute()	移动指针到指定行；有一个int型入口参数，正数表示从前向后编号，负数表示从后向前编号，编号均从1开始；如果存在指定行则返回true，否则返回false；如果结果集类型为TYPE_ FORWARD_ONLY将抛出异常
relative()	移动指针到相对于当前行的指定行；有一个int型入口参数，正数表示向后移动，负数表示向前移动，视当前行为0；如果存在指定行则返回true，否则返回false；如果结果集类型为TYPE_FORWARD_ONLY将抛出异常
getRow()	查看当前行的索引编号；索引编号从1开始，如果位于有效记录行上则返回一个int型索引编号，否则返回0
findColumn()	查看指定列名的索引编号；该方法有一个String型入口参数，为要查看列的名称，如果包含指定列，则返回int型索引编号，否则将抛出异常
isBeforeFirst()	查看指针是否位于ResultSet实例的开头，即第一行之前，如果是则返回true，否则返回false
isAfterLast()	查看指针是否位于ResultSet实例的末尾，即最后一行之后，如果是则返回true，否则返回false
isFirst()	查看指针是否位于ResultSet实例的第一行，如果是则返回true，否则返回false
isLast()	查看指针是否位于ResultSet实例的最后一行，如果是则返回true，否则返回false
close()	立即释放ResultSet实例占用的数据库和JDBC资源，当关闭所属的Statement实例时也将执行此操作

8.4 连接数据库

在对数据库进行操作时，首先需要连接数据库，在JSP中连接数据库大致可以分加载JDBC驱动程序、创建Connection对象的实例、执行SQL语句、获得查询结果和关闭连接等5个步骤，下面进行详细介绍。

8.4.1 加载JDBC驱动程序 √

8.4.2 创建数据库连接 √

8.4.3 执行SQL语句 √

8.4.4 获得查询结果 √

8.4.5 关闭连接 √

8.4.1 加载JDBC驱动程序

在连接数据库之前，首先要加载要连接数据库的驱动到JVM（Java虚拟机），通过java.lang.Class类的静态方法forName(String className)实现，例如加载SQL Server2000驱动程序的代码如下：

```
try {
        Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
} catch (ClassNotFoundException e) {
        System.out.println("加载数据库驱动时抛出异常，内容如下：");
        e.printStackTrace();
}
```

8.4.1 加载JDBC驱动程序

成功加载后，会将加载的驱动类注册给DriverManager类，如果加载失败，将抛出ClassNotFoundException异常，即未找到指定的驱动类，所以需要在加载数据库驱动类时捕捉可能抛出的异常。

技巧：通常将负责加载驱动的代码放在static块中，这样做的好处是只有static块所在的类第一次被加载时才加载数据库驱动，避免重复加载驱动程序，浪费计算机资源。

人民邮电出版社
POSTS & TELECOM PRESS

8.4.2 创建数据库连接

java.sql.DriverManager（驱动程序管理器）类是JDBC的管理层，负责建立和管理数据库连接。通过DriverManager类的静态方法getConnection(String url, String user, String password)可以建立数据库连接，3个入口参数依次为要连接数据库的路径、用户名和密码，该方法的返回值类型为java.sql.Connection，典型代码如下：

```
Connection conn = DriverManager.getConnection(
"jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=db_database08", "sa", "");
```

在上面的代码中，连接的是本地的SQL Server数据库，数据库名称为db_database08，登录用户为sa，密码为空。

人民邮电出版社
POSTS & TELECOM PRESS

8.4.3 执行SQL语句

建立数据库连接（Connection）的目的是与数据库进行通信，实现方式为执行SQL语句，但是通过Connection实例并不能执行SQL语句，还需要通过Connection实例创建Statement实例，Statement实例又分为以下3种类型。

（1）Statement实例：该类型的实例只能用来执行静态的SQL语句；

（2）PreparedStatement实例：该类型的实例增加了执行动态SQL语句的功能；

（3）CallableStatement对象：该类型的实例增加了执行数据库存储过程的功能。

其中Statement是最基础的，PreparedStatement继承了Statement，并做了相应的扩展，而CallableStatement继承了PreparedStatement，又做了相应的扩展，从而保证在基本功能的基础上，各自又增加了一些独特的功能。

8.4.4 获得查询结果

通过Statement接口的executeUpdate()或executeQuery()方法，可以执行SQL语句，同时将返回执行结果。如果执行的是executeUpdate()方法，将返回一个int型数值，代表影响数据库记录的条数，即插入、修改或删除记录的条数；如果执行的是executeQuery()方法，将返回一个ResultSet型的结果集，其中不仅包含所有满足查询条件的记录，还包含相应数据表的相关信息，例如，列的名称、类型和列的数量等。

8.4.5 关闭连接

在建立Connection，Statement和ResultSet实例时，均需占用一定的数据库和JDBC资源，所以每次访问数据库结束后，应该及时销毁这些实例，释放它们占用的所有资源，方法是通过各个实例的close()方法，并且在关闭时建议按照以下的顺序：

```
resultSet.close();
statement.close();
connection.close();
```

8.4.5 关闭连接

采用上面的顺序关闭的原因在于Connection是一个接口，close()方法的实现方式可能多种多样。如果是通过DriverManager类的getConnection()方法得到的Connection实例，在调用close()方法关闭Connection实例时会同时关闭Statement实例和ResultSet实例。但是通常情况下需要采用数据库连接池，在调用通过连接池得到的Connection实例的close()方法时，Connection实例可能并没有被释放，而是被放回到了连接池中，又被其他连接调用，在这种情况下如果不手动关闭Statement实例和ResultSet实例，它们在Connection中可能会越来越多，虽然JVM的垃圾回收机制会定时清理缓存，但是如果清理得不及时，当数据库连接达到一定数量时，将严重影响数据库和计算机的运行速度，甚至导致软件或系统瘫痪。

8.5 典型JSP数据库连接

8.5.1 SQL Server 2000数据库的连接 √

8.5.2 Access数据库的连接 √

8.5.3 MySQL数据库的连接 √

课件制作人：王国辉

8.5.1 SQL Server 2000数据库的连接

SQL Server 2000数据库的驱动为：

```
String driverClass="com.microsoft.jdbc.sqlserver.SQLServerDriver";
```

连接SQL Server 2000数据库需要用到的包有msbase.jar，mssqlserver.jar和msutil.jar。

连接SQL Server 2000数据库的URL为：

```
String url =
"jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=db_database08";
```

在上面的URL中，“127.0.0.1”为数据库所在机器的IP地址，该IP代表本机，也可替换为“localhost”；“1433”为SQL Server 2000数据库默认的端口号；“db_database08”为数据库名。

8.5.1 SQL Server 2000数据库的连接

【例8-1】 在JSP中连接SQL Server 2000数据库db_databse08

8.5.2 Access数据库的连接

Access数据库的驱动为：

```
String driverClass="sun.jdbc.odbc.JdbcOdbcDriver";
```

连接Access数据库需要通过JDBC-ODBC方式，不需要引入任何包。

连接Access数据库的URL为：

```
String url = "jdbc:odbc:driver={Microsoft Access Driver
(*.mdb)};DBQ=E:/db_database08.mdb";
```

由于在上面的URL中采用的是系统默认的连接Access数据库的驱动Microsoft Access Driver（*.mdb），所以不需要手动配置ODBC驱动；“E:/db_database08.mdb”为Access数据库的绝对存放路径，在实际程序中，可以通过request对象的相关方法获取数据库文件的存放路径。

8.5.2 Access数据库的连接

【例8-2】 在JSP中连接Access数据库db_databse08

8.5.3 MySQL数据库的连接

MySQL数据库的驱动为：

```
String driverClass="com.mysql.jdbc.Driver";
```

连接MySQL数据库需要用到的包为mysql-connector-java-3.0.16-ga-bin.jar。

连接MySQL数据库的URL为：

```
String url="jdbc:mysql://127.0.0.1:3306/db_database08";
```

在上面的URL中，“127.0.0.1”为数据库所在机器的IP地址，该IP代表本机，也可替换为“localhost”；“3306”为MySQL数据库默认的端口号；“db_database08”为要连接的数据库名。

【例8-3】 在JSP中连接MySQL数据库db_databse08

8.6 数据库操作技术

在开发Web应用程序时，经常需要对数据库进行操作，最常用的数据库操作技术，包括向数据库查询、添加、修改或删除数据库中的数据，这些操作即可以通过静态的SQL语句实现，也可以通过动态的SQL语句实现，还可以通过存储过程实现，具体采用的实现方式要根据实际情况而定。

8.6.1 查询操作 √

8.6.2 添加操作 √

8.6.3 修改操作 √

8.6.4 删除操作 √

8.6.1 查询操作

JDBC中提供了两种实现数据查询的方法，一种是通过Statement对象执行静态的SQL语句实现；另一种是通过PreparedStatement对象执行动态的SQL语句实现。由于PreparedStatement类是Statement类的扩展，一个PreparedStatement对象包含一个预编译的SQL语句，该SQL语句可能包含一个或多个参数，这样应用程序可以动态地为其赋值，所以PreparedStatement对象执行的速度比Statement对象快。因此在执行较多的SQL语句时，建议使用PreparedStatement对象。

下面将通过两个实例分别应用这两种方法实现数据查询。

【例8-4】 查询操作示例1

【例8-5】 查询操作示例2

8.6.2 添加操作

同查询操作相同，JDBC中也提供了两种实现数据添加操作的方法，一种是通过Statement对象执行静态的SQL语句实现；另一种是通过PreparedStatement对象执行动态的SQL语句实现。

通过Statement对象和PreparedStatement对象实现数据添加操作的方法同实现查询操作的方法基本相同，所不同的就是执行的SQL语句及执行方法不同，实现数据添加操作时采用的是executeUpdate()方法，而实现数据查询时使用的是executeQuery()方法。实现数据添加操作使用的SQL语句为INSERT语句，其语法格式如下：

8.6.2 添加操作

Insert [INTO] table_name[(column_list)] values(data_values)

上面语法中各参数说明如下表所示。

参　数	描　述
[INTO]	可选项，无特殊含义，可以将它用在INSERT和目标表之前
table_name	要添加记录的数据表名称
column_list	是表中的字段列表，表示向表中哪些字段插入数据；如果是多个字段，字段之间用逗号分隔；不指定column_list，默认向数据表中所有字段插入数据
data_values	要添加的数据列表，各个数据之间使用逗号分隔；数据列表中的个数、数据类型必须和字段列表中的字段个数、数据类型相一致
values	引入要插入的数据值的列表；对于column_list（如果已指定）中或者表中的每个列，都必须有一个数据值；必须用圆括号将值列表括起来；如果VALUES列表中的值与表中的值和表中列的顺序不相同，或者未包含表中所有列的值，那么必须使用column_list明确地指定存储每个传入值的列

8.6.2 添加操作

应用Statement对象向数据表tb_user中添加数据的关键代码如下：

```
Statement stmt=conn.createStatement();
int rtn= stmt.executeUpdate("insert into tb_user (name,pwd)
values('hope','111')");
```

利用PreparedStatement对象向数据表tb_user中添加数据的关键代码如下：

```
PreparedStatement pStmt = conn.prepareStatement("insert into
tb_user (name,pwd) values(?,?)");
pStmt.setString(1,"dream");
pStmt.setString(2,"111");
int rtn= pStmt.executeUpdate();
```

8.6.3 修改操作

同添加操作相同，JDBC中也提供了两种实现数据修改操作的方法，一种是通过Statement对象执行静态的SQL语句实现；另一种是通过PreparedStatement对象执行动态的SQL语句实现。

通过Statement对象和PreparedStatement对象实现数据修改操作的方法同实现添加操作的方法基本相同，所不同的就是执行的SQL语句不同，实现数据修改操作使用的SQL语句为UPDATE语句，其语法格式如下：

```
UPDATE table_name
SET <column_name>=<expression>
    [….,<last column_name>=<last expression>]
[WHERE<search_condition>]
```

8.6.3 修改操作

参　数	描　述
table_name	需要更新的数据表名
SET	指定要更新的列或变量名称的列表
column_name	含有要更改数据的列的名称；column_name 必须驻留于 UPDATE 子句中所指定的表或视图中；标识列不能进行更新；如果指定了限定的列名称，限定符必须同 UPDATE 子句中的表或视图的名称相匹配
expression	变量、字面值、表达式或加上括号返回单个值的subSELECT 语句；expression 返回的值将替换column_name中的现有值
WHERE	指定条件来限定所更新的行
<search_condition>	为要更新行指定需满足的条件，搜索条件也可以是连接所基于的条件，对搜索条件中可以包含的谓词数量没有限制

8.6.3 修改操作

应用Statement对象修改数据表tb_user中name字段值为“dream”的记录，关键代码如下：

```
Statement stmt=conn.createStatement();
int rtn= stmt.executeUpdate("update tb_user set
name='hope',pwd='222' where name='dream'");
```

利用PreparedStatement对象修改数据表tb_user中name字段值为“hope”的记录，关键代码如下：

```
PreparedStatement pStmt = conn.prepareStatement("update tb_user
set name=?,pwd=? where name=?");
        pStmt.setString(1,"dream");
        pStmt.setString(2,"111");
        pStmt.setString(3,"hope");
        int rtn= pStmt.executeUpdate();
```

8.6.3 修改操作

说明：在实际应用中，经常是先将要修改的数据查询出来并显示到相应的表单中，然后将表单提交到相应处理页，在处理页中获取要修改的数据，并执行修改操作，完成数据修改。

8.6.4 删除操作

实现数据删除操作也可以通过两种方法实现，一种是通过Statement对象执行静态的SQL语句实现；另一种是通过PreparedStatement对象执行动态的SQL语句实现。通过Statement对象和PreparedStatement对象实现数据删除操作的方法同实现添加操作的方法基本相同，所不同的就是执行的SQL语句不同，实现数据删除操作使用的SQL语句为DELETE语句，其语法格式如下：

```
DELETE FROM <table_name >[WHERE<search condition>]
```

在上面的语法中，table_name用于指定要删除数据的表的名称；<search_condition>用于指定删除数据的限定条件。在搜索条件中对包含的谓词数量没有限制。

8.6.4 删除操作

应用Statement对象从数据表tb_user中删除name字段值为“hope”的数据，关键代码如下：

```
Statement stmt=conn.createStatement();
int rtn= stmt.executeUpdate("delete tb_user where name='hope'");
```

利用PreparedStatement对象从数据表tb_user中删除name字段值为“dream”的数据，关键代码如下：

```
PreparedStatement pStmt = conn.prepareStatement("delete from
tb_user where name=?");
pStmt.setString(1,"dream");
int rtn= pStmt.executeUpdate();
```

人民邮电出版社
POSTS & TELECOM PRESS

8.7 连接池技术

8.7.1 连接池简介√

8.7.2 在Tomcat中配置连接池 √

8.7.3 使用连接池技术访问数据库 √

课件制作人：王国辉

8.7.1 连接池简介

通常情况下，在每次访问数据库之前都要先建立与数据库的连接，这将消耗一定的资源，并延长了访问数据库的时间，如果是访问量相对较低的系统还可以，如果访问量较高，将严重影响系统的性能。为了解决这一问题，引入了连接池的概念。所谓连接池，就是预先建立好一定数量的数据库连接，模拟存放在一个连接池中，由连接池负责对这些数据库连接进行管理。这样，当需要访问数据库时，就可以通过已经建立好的连接访问数据库了，从而免去了每次在访问数据库之前建立数据库连接的开销。

8.7.1 连接池简介

连接池还解决了数据库连接数量限制的问题。由于数据库能够承受的连接数量是有限的，当达到一定程度时，数据库的性能就会下降，甚至崩溃，而池化管理机制，通过有效地使用和调度这些连接池中的连接，则解决了这个问题（在这里我们不讨论连接池对连接数量限制的问题）。

数据库连接池的具体实施办法是：

（1）预先创建一定数量的连接，存放在连接池中；

（2）当程序请求一个连接时，连接池是为该请求分配一个空闲连接，而不是去重新建立一个连接；当程序使用完连接后，该连接将重新回到连接池中，而不是直接将连接释放；

（3）当连接池中的空闲连接数量低于下限时，连接池将根据管理机制追加创建一定数量的连接；当空闲连接数量高于上限时，连接池将释放一定数量的连接。

8.7.1 连接池简介

在每次用完Connection后，要及时调用Connection对象的close()或dispose()方法显式关闭连接，以便连接可以及时返回到连接池中，非显式关闭的连接可能不会添加或返回到池中。

连接池具有下列优点：

（1）创建一个新的数据库连接所耗费的时间主要取决于网络的速度以及应用程序和数据库服务器的（网络）距离，而且这个过程通常是一个很耗时的过程，而采用数据库连接池后，数据库连接请求则可以直接通过连接池满足，而不需要为该请求重新连接、认证到数据库服务器，从而节省了时间；

8.7.1 连接池简介

（2）提高了数据库连接的重复使用率；
（3）解决了数据库对连接数量的限制。

与此同时，连接池具有下列缺点：

（1）连接池中可能存在多个与数据库保持连接但未被使用的连接，在一定程度上浪费了资源；
（2）要求开发人员和使用者准确估算系统需要提供的最大数据库连接的数量。

8.7.2 在Tomcat中配置连接池

在通过连接池技术访问数据库时，首先需要在Tomcat下配置数据库连接池，下面以SQL Server 2000为例介绍在Tomcat 6.0下配置数据库连接池的方法。

（1）将SQL Server数据库的3个JDBC驱动包msbase.jar、mssqlserver.jar和msutil.jar复制到Tomcat安装路径下的common\lib文件夹中。

（2）配置数据源。在配置数据源时，可以将其配置到Tomcat安装目录下的conf\server.xml文件中，也可以将其配置到Web工程目录下的META-INF\context.xml文件中，建议采用后者，因为这样配置的数据源更有针对性，配置数据源的具体代码如下：

8.7.2 在Tomcat中配置连接池

```
<Context>
        <Resource name="TestJNDI" type="javax.sql.DataSource"
auth="Container"
        driverClassName="com.microsoft.jdbc.sqlserver.SQLServerDriver
"
        url="jdbc:microsoft:sqlserver://127.0.0.1:1433;DatabaseName=db_
db_database08"
                username="sa" password="" maxActive="4" maxIdle="2"
maxWait="6000" />
</Context>
```

在配置数据源时需要配置的<Resource>元素的属性及其说明如下表所示。

8.7.2 在Tomcat中配置连接池

属性名称	说　明
name	设置数据源的JNDI名
type	设置数据源的类型
auth	设置数据源的管理者，有两个可选值Container和Application，Container表示由容器来创建和管理数据源，Application表示由Web应用来创建和管理数据源
driverClassName	设置连接数据库的JDBC驱动程序
url	设置连接数据库的路径
username	设置连接数据库的用户名
password	设置连接数据库的密码
maxActive	设置连接池中处于活动状态的数据库连接的最大数目，0表示不受限制
maxIdle	设置连接池中处于空闲状态的数据库连接的最大数目，0表示不受限制
maxWait	设置当连接池中没有处于空闲状态的连接时，请求数据库连接的请求的最长等待时间（单位为ms），如果超出该时间将抛出异常，−1表示无限期等待

8.7.3 使用连接池技术访问数据库

JDBC2.0提供了javax.sql.DataSource接口，负责与数据库建立连接，在应用时不需要编写连接数据库代码，可以直接从数据源中获得数据库连接。在DataSource中预先建立了多个数据库连接，这些数据库连接保存在数据库连接池中，当程序访问数据库时，只需从连接池中取出空闲的连接，访问结束后，再将连接归还给连接池。

DataSource对象由容器（例如Tomcat）提供，不能通过创建实例的方法来获得DataSource对象，需要利用Java的JNDI（Java Nameing and Directory Interface，Java命名和目录接口）来获得DataSource对象的引用。

8.7.3 使用连接池技术访问数据库

JNDI是一个应用程序设计的API，为开发人员提供了查询和访问各种命名和目录服务的通用的、统一的接口，类似JDBC，都是构建在抽象层上的。JNDI提供了一种统一的方式，可以用在网络上查找和访问JDBC服务中，通过指定一个资源名称，可以返回数据库连接建立所必须的信息。

【例8-6】 应用连接池技术访问数据库db_database08，并显示数据表tb_user中的全部数据