Web应用开发
之
JDBC数据库访问

本节内容

- JDBC API
- 数据源

三、JDBC API介绍

- JDBC API是Java语言的标准API，目前的最新版本是JDBC 4。
- JDBC API是通过两个包提供的：
  - java.sql包
  - javax.sql包

三、 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。

三、 JDBC API介绍

- JDBC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

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

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。
- 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。

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）,在事务commit 或rollback 后，ResultSet是否仍然可用还是关闭。

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。

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。

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，
- 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)
```

- 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。

1. 检索字段值

➢ 返回其他数据类型的方法:

- 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。图片
- public Clob getClob(int columnIndex)：返回指定列的Clob对象值。文章

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。

2. 数据类型转换

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

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。

①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。

②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。

③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

- 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

- 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

- 例：在products表中修改一件商品的名称，见UpdateProductServlet.java。

```
String sql = "SELECT prod_id,pname
                    FROM products
                   WHERE prod_id ='10001'";
Statement stmt =
  dbconn.createStatement(ResultSet.TYPE_FORWARD_ONLY,Resul
  tSet.CONCUR_UPDATABLE);
ResultSet rset = stmt.executeQuery(sql);
rset.next();
rset.updateString(2, "华为p40 pro手机"); // 修改当前行的字段值
rset.updateRow();     // 更新当前行
```

3. 5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，在多次执行同一个语句时效率较低

- 使用PreparedStatement对象，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。

3.5.1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。

(1)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。

3.5.1创建PreparedStatement对象

(2)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。

注：第一个参数是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。

3.5.2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。
- 为占位符设置整数值和字符串值的方法：

➢ 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(2, "华为p30 pro手机");
pstmt.setDouble(3, 3499.00);
pstmt.setInt(4, 30);
```

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。
- 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。
- 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()：

```
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();
```

- 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。

示例：添加商品addproduct.jsp

```
<%@ page contentType="text/html; charset=UTF-8"
            pageEncoding="UTF-8"%>
<html>
<head><title>商品添加查询</title></head>
<body>
<form action = "addproduct1.do" method="post">
 请输入商品信息：<br>
 商品号码：<input type = "text" name="prod_id" size="30"><br><br>
 商品名称：<input type = "text" name="pname" size="30"><br><br>
 商品价格：<input type = "text" name="price" size="30"><br><br>
 商品库存：<input type = "text" name="stock" size="30">
<br><input type = "submit" value = "添加商品">
</form>
</body>
</html>
```

示例：添加商品AddProductServlet1

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    Product product = new Product();
    request.setCharacterEncoding("utf-8");
    String message = null;
    try {
        String sql = "INSERT INTO products VALUES (?, ?, ?, ?) ";
        PreparedStatement pstmt = dbconn.prepareStatement(sql);
        pstmt.setString(1, request.getParameter("prod_id"));
        pstmt.setDouble(2, Double.parseDouble(request.getParameter("price")));
        pstmt.setString(3, request.getParameter("pname"));
        pstmt.setInt(4, Integer.parseInt(request.getParameter("stock")));
        int n = pstmt.executeUpdate();
        product.setProd_id(request.getParameter("prod_id"));
        product.setPname(request.getParameter("pname"));
        product.setPrice(Double.parseDouble(request.getParameter("price")));
        product.setStock(Integer.parseInt(request.getParameter("stock")));
        if (n>0) {
           message = "<li>添加商品成功！</li>";
        } else {
           message = "<li>添加商品失败！</li>";
        }
        } catch (SQLException e) {
           e.printStackTrace();
           message = "<li>添加商品异常！</li>";
    }
    request.setAttribute("result", message);
    request.setAttribute("product", product);
    RequestDispatcher rd = getServletContext().getRequestDispatcher("/addproductresult.jsp");
    rd.forward(request, response);
}
```

四、数据源

- 管理Web应用程序访问数据库的方法。

（1）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。

（2）为了提高数据库访问效率，从JDBC 2.0开始提供了一种连接池和数据源的技术访问数据库。

4.1 数据源介绍

- 数据源（DataSource）是目前Web应用开发中获取数据库连接的首选方法。

（1）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。

（2）使用数据源技术，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。

4.1 数据源

- 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。
- 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。
- 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。

4.1 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。

4.2 配置数据源

- 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
- 配置了连接PostgreSQL数据库的数据源，内容如程序context.xml

1. 配置局部数据源

```
<?xml version="1.0" encoding="utf-8"?>
<Context reloadable = "true">
    <Resource
        name="jdbc/sampleDS"
        type="javax.sql.DataSource"
        driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/postgres"
        username="postgres"
        password="postgres"
        maxActive="4"
        maxIdle="2"
        maxWait="5000"
    />
</Context>
```

1. 配置局部数据源

- name：数据源名，这里是jdbc/sampleDS。
- type：指定该资源的类型，这里为DataSource类型。
- 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。
- 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。

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。

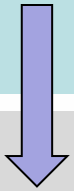
2. 在应用程序中使用数据源

- 使用数据源获得数据库连接对象修改QueryProductServlet.java的数据库连接程序

- 通过InitialContext类创建一个上下文对象context
- 通过它的lookup()查找数据源对象
- 通过数据源对象从连接池中返回一个数据库连接对象

修改QueryProductServlet.java

```
Connection dbconn = null;
public void init() {
    String driver = "org.postgresql.Driver";
    String dburl = "jdbc:postgresql://127.0.0.1:5432/postgres";
    String username = "postgres";
    String password = "postgres";
    try {
        Class.forName(driver);
        dbconn = DriverManager.getConnection(dburl, username, password);
    } catch (ClassNotFoundException e1) {
        System.out.println(e1);
    } catch (SQLException e2) { }
}
```



```
Connection dbconn = null;
DataSource dataSource;
public void init() {
    try {
       Context context = new InitialContext();
       dataSource =(DataSource)context.lookup("java:comp/env/jdbc/sampleDS");
       dbconn=dataSource.getConnection();
    }catch(NamingException ne){
        System.out.println("Exception:"+ne.toString());
    }catch (SQLException se) {
        System.out.println("Exception:"+se.toString());
    }
}
```

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"
      driverClassName="org.postgresql.Driver"
      url="jdbc:postgresql://localhost:5432/postgres"
      username="postgres"
      password="postgres"
      maxActive="4"
      maxIdle="2"
      maxWait="5000"
/>
```

- 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

- 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。

小　结

- 从JDBC 2.0开始提供了通过数据源建立连接对象的机制。
- 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。