JDBC高级使用和数据库事务

课前默写

```
1、写出JDBC的使用步骤
2、写出JDBC的常用的类和方法
3、写出ResultSet的常用方法
```

课程回顾

```
1、JDBC的使用步骤
2、JDBC执行DDL和DML语句
3、JDBC执行DQL语句
4、ResultSet的使用
```

今日内容

```
1、JDBC的批处理
2、JDBC操作二进制
3、数据库事务
4、JDBC实现事务
```

教学目标

```
1、掌握JDBC的批处理
2、熟悉JDBC操作二进制
3、掌握数据库事务和ACID和隔离级别
4、掌握数据库事务的使用
5、掌握JDBC的事务操作
```

第十二章 JDBC批处理操作

批量处理允许您将相关的SQL语句分组到批处理中，并通过对数据库的一次调用提交它们。

当您一次向数据库发送多个SQL语句时，可以减少连接数据库的开销，从而提高性能。

12.1 使用Statement对象进行批处理操作

以下是使用语句对象的批处理的典型步骤序列

- 使用*createStatement（）*方法创建Statement对象。
- 使用*setAutoCommit（）*将auto-commit设置为false 。
- 使用*addBatch（）*方法在创建的语句对象上添加您喜欢的SQL语句到批处理中。
- 在创建的语句对象上使用*executeBatch（）*方法执行所有SQL语句。
- 最后，使用*commit（）*方法提交所有更改。

```
// Create statement object
Statement stmt = conn.createStatement();
```

```
// Set auto-commit to false
conn.setAutoCommit(false);

// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(200,'Zia', 'Ali', 30)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(201,'Raj', 'Kumar', 35)";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create one more SQL statement
String SQL = "UPDATE Employees SET age = 35 " +
             "WHERE id = 100";
// Add above SQL statement in the batch.
stmt.addBatch(SQL);

// Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

12.2 PrepareStatement对象进行批处理

1. 使用占位符创建SQL语句。
2. 使用*prepareStatement（）*方法创建PrepareStatement对象。
3. 使用*setAutoCommit（）*将auto-commit设置为false 。
4. 使用*addBatch（）*方法在创建的语句对象上添加您喜欢的SQL语句到批处理中。
5. 在创建的语句对象上使用*executeBatch（）*方法执行所有SQL语句。
6. 最后，使用*commit（）*方法提交所有更改。

```
// Create SQL statement
String SQL = "INSERT INTO Employees (id, first, last, age) " +
             "VALUES(?, ?, ?, ?)";

// Create PrepareStatement object
PreparedStatemen pstmt = conn.prepareStatement(SQL);

//Set auto-commit to false
conn.setAutoCommit(false);

// Set the variables
pstmt.setInt( 1, 400 );
pstmt.setString( 2, "Pappu" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 33 );
// Add it to the batch
pstmt.addBatch();
```

```
// Set the variables
pstmt.setInt( 1, 401 );
pstmt.setString( 2, "Pawan" );
pstmt.setString( 3, "Singh" );
pstmt.setInt( 4, 31 );
// Add it to the batch
pstmt.addBatch();

//add more batches

//Create an int[] to hold returned values
int[] count = stmt.executeBatch();

//Explicitly commit statements to apply changes
conn.commit();
```

第十三章 JDBC操作二进制

PreparedStatement对象可以使用输入和输出流来提供参数数据。这使您可以将整个文件放入可以保存大值的数据库列，例如CLOB和BLOB数据类型。

有以下方法可用于流式传输数据 -

- **setAsciiStream ()** ：此方法用于提供大的ASCII值。
- **setCharacterStream ()** ：此方法用于提供大型UNICODE值。
- **setBinaryStream ()** ：此方法用于提供较大的二进制值。

setXXXStream（）方法除了参数占位符之外还需要额外的参数，文件大小。

考虑我们要将XML文件XML_Data.xml上传到数据库表中。这是XML文件的内容 -

```
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
```

```
// Import required packages
import java.sql.*;
import java.io.*;
import java.util.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials

   static final String USER = "username";
```

```
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   PreparedStatement pstmt = null;
   Statement stmt = null;
   ResultSet rs = null;
   try{
      // Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
      // Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);

      //Create a Statement object and build table
      stmt = conn.createStatement();
      createXMLTable(stmt);

      //Open a FileInputStream
      File f = new File("XML_Data.xml");
      long fileLength = f.length();
      FileInputStream fis = new FileInputStream(f);

      //Create PreparedStatement and stream data
      String SQL = "INSERT INTO XML_Data VALUES (?,?)";
      pstmt = conn.prepareStatement(SQL);
      pstmt.setInt(1,100);
      pstmt.setAsciiStream(2,fis,(int)fileLength);
      pstmt.execute();

      //Close input stream
      fis.close();

      // Do a query to get the row
      SQL = "SELECT Data FROM XML_Data WHERE id=100";
      rs = stmt.executeQuery (SQL);
      // Get the first row
      if (rs.next ()){
         //Retrieve data from input stream
         InputStream xmlInputStream = rs.getAsciiStream (1);
         int c;
         ByteArrayOutputStream bos = new ByteArrayOutputStream();
         while (( c = xmlInputStream.read ()) != -1)
            bos.write(c);
         //Print results
         System.out.println(bos.toString());
      }
      // Clean-up environment
      rs.close();
      stmt.close();
      pstmt.close();
      conn.close();

   }catch(SQLException se){
```

```
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(pstmt!=null)
            pstmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main

public static void createXMLTable(Statement stmt)
   throws SQLException{
   System.out.println("Creating XML_Data table..." );
   //Create SQL Statement
   String streamingDataSql = "CREATE TABLE XML_Data " +
                             "(id INTEGER, Data LONG)";
   //Drop table first if it exists.
   try{
      stmt.executeUpdate("DROP TABLE XML_Data");
   }catch(SQLException se){
   }// do nothing
   //Build table.
   stmt.executeUpdate(streamingDataSql);
}//end createXMLTable
}//end JDBCExample
```

第十四章 数据库事务

> 一组要么同时执行成功，要么同时执行失败的SQL语句。是数据库操作的一个执行单元。

14.1 数据库事务讲解

事务开始于

- 连接到数据库上，并执行一条DML语句insert、update或delete
- 前一个事务结束后，又输入了另一条DML语句

事务结束于

- 执行commit或rollback语句。
- 执行一条DDL语句，例如create table语句，在这种情况下，会自动执行commit语句。
- 执行一条DDL语句，例如grant语句，在这种情况下，会自动执行commit。
- 断开与数据库的连接
- 执行了一条DML语句，该语句却失败了，在这种情况中，会为这个无效的DML语句执行rollback语句。

14.2 事务的四大特点（ACID）

- **actomicity(原子性)**

  表示一个事务内的所有操作是一个整体，要么全部成功，要么全部失败

- **consistency(一致性)**

  表示一个事务内有一个操作失败时，所有的更改过的数据都必须回滚到修改前状态

- **isolation(隔离性)**

事务查看数据时数据所处的状态，要么是另一并发事务修改它之前的状态，要么是另一事务修改它之后的状态，事务不会查看中间状态的数据。

- **durability(持久性)**

  持久性事务完成之后，它对于系统的影响是永久性的。

14.3 事务隔离级别

SQL标准定义了4类隔离级别，包括了一些具体规则，用来限定事务内外的哪些改变是可见的，哪些是不可见的。低级别的隔离级一般支持更高的并发处理，并拥有更低的系统开销。

Read Uncommitted（读取未提交内容)

在该隔离级别，所有事务都可以看到其他未提交事务的执行结果。本隔离级别很少用于实际应用，因为它的性能也不比其他级别好多少。读取未提交的数据，也被称之为脏读（Dirty Read）。

Read Committed（读取提交内容）

这是大多数数据库系统的默认隔离级别（但不是MySQL默认的）。它满足了隔离的简单定义：一个事务只能看见已经提交事务所做的改变。这种隔离级别 也支持所谓的不可重复读（Nonrepeatable Read），因为同一事务的其他实例在该实例处理其间可能会有新的commit，所以同一select可能返回不同结果。

Repeatable Read可重读

这是MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行。不过理论上，这会导致另一个棘手的问题：幻读（Phantom Read）。简单的说，幻读指当用户读取某一范围的数据行时，另一个事务又在该范围内插入了新行，当用户再读取该范围的数据行时，会发现有新的“幻影” 行。InnoDB和Falcon存储引擎通过多版本并发控制（MVCC，Multiversion Concurrency Control）机制解决了该问题。

Serializable 可串行化 这是最高的隔离级别，它通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。简言之，它是在每个读的数据行上加上共享锁。在这个级别，可能导致大量的超时现象和锁竞争。

这四种隔离级别采取不同的锁类型来实现，若读取的是同一个数据的话，就容易发生问题。

例如：

脏读(Drity Read)：某个事务已更新一份数据，另一个事务在此时读取了同一份数据，由于某些原因，前一个RollBack了操作，则后一个事务所读取的数据就会是不正确的。

不可重复读(Non-repeatableread):在一个事务的两次查询之中数据不一致，这可能是两次查询过程中间插入了一个事务更新的原有的数据。

幻读(Phantom Read):在一个事务的两次查询中数据笔数不一致，例如有一个事务查询了几列(Row)数据，而另一个事务却在此时插入了新的几列数据，先前的事务在接下来的查询中，就会发现有几列数据是它先前所没有的

14.4 Java中事务应用

如果JDBC连接处于*自动提交*模式，默认情况下，则每个SQL语句在完成后都会提交到数据库。

事务使您能够控制是否和何时更改应用于数据库。它将单个SQL语句或一组SQL语句视为一个逻辑单元，如果任何语句失败，则整个事务将失败。

要启用手动事务支持，而不是JDBC驱动程序默认使用的*自动提交*模式，请使用Connection对象的**setAutoCommit（）**方法。如果将boolean false传递给setAutoCommit（），则关闭自动提交。我们可以传递一个布尔值true来重新打开它。

14.4.1 事务的提交和回滚

完成更改后，我们要提交更改，然后在连接对象上调用**commit（）**方法，如下所示：

```
conn.commit( );
```

否则，要使用连接名为conn的数据库回滚更新，请使用以下代码 -

```
conn.rollback( );
```

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   String SQL = "INSERT INTO Employees  " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees  " +
                "VALUES (107, 22, 'Sita', 'Singh')";
   stmt.executeUpdate(SQL);
   // If there is no error.
   conn.commit();
}catch(SQLException se){
   // If there is any error.
   conn.rollback();
}
```

14.4.2 使用 Savepoints

新的JDBC 3.0 Savepoint接口为您提供了额外的事务控制。

设置保存点时，可以在事务中定义逻辑回滚点。如果通过保存点发生错误，则可以使用回滚方法来撤消所有更改或仅保存在保存点之后所做的更改。

Connection对象有两种新的方法来帮助您管理保存点 -

- **setSavepoint（String savepointName）**：定义新的保存点。它还返回一个Savepoint对象。
- **releaseSavepoint（Savepoint savepointName）**：删除保存点。请注意，它需要一个Savepoint对象作为参数。此对象通常是由setSavepoint（）方法生成的保存点。

```
try{
   //Assume a valid connection object conn
   conn.setAutoCommit(false);
   Statement stmt = conn.createStatement();

   //set a Savepoint
   Savepoint savepoint1 = conn.setSavepoint("Savepoint1");
   String SQL = "INSERT INTO Employees " +
                "VALUES (106, 20, 'Rita', 'Tez')";
   stmt.executeUpdate(SQL);
   //Submit a malformed SQL statement that breaks
   String SQL = "INSERTED IN Employees " +
                "VALUES (107, 22, 'Sita', 'Tez')";
   stmt.executeUpdate(SQL);
   // If there is no error, commit the changes.
   conn.commit();

}catch(SQLException se){
   // If there is any error.
   conn.rollback(savepoint1);
}
```

1、要取消掉JDBC的自动提交：void setAutoCommit(boolean autoCommit)

2、执行各个SQL语句，加入到批处理之中

3、如果所有语句执行成功，则提交事务 commit()；如果出现了错误，则回滚：rollback()

```
try {
      connection.setAutoCommit(false);
      add(connection);
//          int i = 1/0;
            sub(connection);
            System.out.println("===============");
            connection.commit();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            System.out.println("---------------");
            try {
                connection.rollback();
            } catch (SQLException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();

            }
```

```
    }
```

介绍:以上就是java代码利用jdbc操作数据库的最简单版本,数据库事务通常要借助补捉异常语句!

作业题

```
1、实现一个学生管理系统，要求实现正删改查，基于控制台实现即可
要求可以对学生信息进行添加、修改、删除、查询的功能
```

面试题

```
1、什么是JDBC，在什么时候会用到它?
2、execute，executeQuery，executeUpdate的区别是什么?
3、JDBC的PreparedStatement是什么?
4、相对于Statement，PreparedStatement的优点是什么?
```