**Java调用SQL Server的存储过程详解**

**1使用不带参数的存储过程**

使用 JDBC 驱动程序调用不带参数的存储过程时，必须使用 call SQL 转义序列。不带参数的 call 转义序列的语法如下所示：

1. {call **procedure**-**name**}

作为实例，在 SQL Server 2005 AdventureWorks 示例数据库中创建以下存储过程：

1. **CREATE** **PROCEDURE** GetContactFormalNames
2. **AS**
3. **BEGIN**
4. **SELECT** **TOP** 10 Title + ' ' + FirstName + ' ' + LastName **AS** FormalName
5. **FROM** Person.Contact
6. **END**

此存储过程返回单个结果集，其中包含一列数据(由 Person.Contact 表中前十个联系人的称呼、名称和姓氏组成)。

在下面的实例中，将向此函数传递 AdventureWorks 示例数据库的打开连接，然后使用 executeQuery 方法调用 GetContactFormalNames 存储过程。

1. **public** **static** **void** executeSprocNoParams(Connection con) ...{
2. **try** ...{
3. Statement stmt = con.createStatement();
4. ResultSet rs = stmt.executeQuery("{call dbo.GetContactFormalNames}");
6. **while** (rs.next()) ...{
7. System.out.println(rs.getString("FormalName"));
8. }
9. rs.close();
10. stmt.close();
11. }
12. **catch** (Exception e) ...{
13. e.printStackTrace();
14. }
15. }

**2使用带有输入参数的存储过程**

使用 JDBC 驱动程序调用带参数的存储过程时，必须结合 SQLServerConnection 类的 prepareCall 方法使用 call SQL 转义序列。带有 IN 参数的 call 转义序列的语法如下所示：

1. {call **procedure**-**name**[([parameter][,[parameter]]...)]}

构造 call 转义序列时，请使用 ?(问号)字符来指定 IN 参数。此字符充当要传递给该存储过程的参数值的占位符。可以使用 SQLServerPreparedStatement 类的 setter 方法之一为参数指定值。可使用的 setter 方法由 IN 参数的数据类型决定。

向 setter 方法传递值时，不仅需要指定要在参数中使用的实际值，还必须指定参数在存储过程中的序数位置。例如，如果存储过程包含单个 IN 参数，则其序数值为 1。如果存储过程包含两个参数，则第一个序数值为 1，第二个序数值为 2。

作为如何调用包含 IN 参数的存储过程的实例，使用 SQL Server 2005 AdventureWorks 示例数据库中的 uspGetEmployeeManagers 存储过程。此存储过程接受名为 EmployeeID 的单个输入参数(它是一个整数值)，然后基于指定的 EmployeeID 返回雇员及其经理的递归列表。下面是调用此存储过程的 Java 代码：

1. **public** **static** **void** executeSprocInParams(Connection con) ...{
2. **try** ...{
3. PreparedStatement pstmt = con.prepareStatement("{call dbo.uspGetEmployeeManagers(?)}");
4. pstmt.setInt(1, 50);
5. ResultSet rs = pstmt.executeQuery();
6. **while** (rs.next()) ...{
7. System.out.println("EMPLOYEE:");
8. System.out.println(rs.getString("LastName") + ", " + rs.getString("FirstName"));
9. System.out.println("MANAGER:");
10. System.out.println(rs.getString("ManagerLastName") + ", " + rs.getString("ManagerFirstName"));
11. System.out.println();
12. }
13. rs.close();
14. pstmt.close();
15. }
16. **catch** (Exception e) ...{
17. e.printStackTrace();
18. }
19. }

**3使用带有输出参数的存储过程**

使用 JDBC 驱动程序调用此类存储过程时，必须结合 SQLServerConnection 类的 prepareCall 方法使用 call SQL 转义序列。带有 OUT 参数的 call 转义序列的语法如下所示：

1. {call **procedure**-**name**[([parameter][,[parameter]]...)]}

构造 call 转义序列时，请使用 ?(问号)字符来指定 OUT 参数。此字符充当要从该存储过程返回的参数值的占位符。要为 OUT 参数指定值，必须在运行存储过程前使用 SQLServerCallableStatement 类的 registerOutParameter 方法指定各参数的数据类型。

使用 registerOutParameter 方法为 OUT 参数指定的值必须是 java.sql.Types 所包含的 JDBC 数据类型之一，而它又被映射成本地 SQL Server 数据类型之一。有关 JDBC 和 SQL Server 数据类型的详细信息，请参阅了解 JDBC 驱动程序数据类型。

当您对于 OUT 参数向 registerOutParameter 方法传递一个值时，不仅必须指定要用于此参数的数据类型，而且必须在存储过程中指定此参数的序号位置或此参数的名称。例如，如果存储过程包含单个 OUT 参数，则其序数值为 1;如果存储过程包含两个参数，则第一个序数值为 1，第二个序数值为 2。

作为实例，在 SQL Server 2005 AdventureWorks 示例数据库中创建以下存储过程： 根据指定的整数 IN 参数 (employeeID)，该存储过程也返回单个整数 OUT 参数 (managerID)。根据 HumanResources.Employee 表中包含的 EmployeeID，OUT 参数中返回的值为 ManagerID。

在下面的实例中，将向此函数传递 AdventureWorks 示例数据库的打开连接，然后使用 execute 方法调用 GetImmediateManager 存储过程：

1. **public** **static** **void** executeStoredProcedure(Connection con) ...{
2. **try** ...{
3. CallableStatement cstmt = con.prepareCall("{call dbo.GetImmediateManager(?, ?)}");
4. cstmt.setInt(1, 5);
5. cstmt.registerOutParameter(2, java.sql.Types.INTEGER);
6. cstmt.execute();
7. System.out.println("MANAGER ID: " + cstmt.getInt(2));
8. }
9. **catch** (Exception e) ...{
10. e.printStackTrace();
11. }
12. }

本示例使用序号位置来标识参数。或者，也可以使用参数的名称(而非其序号位置)来标识此参数。下面的代码示例修改了上一个示例，以说明如何在 Java 应用程序中使用命名参数。请注意，这些参数名称对应于存储过程的定义中的参数名称：

CREATE PROCEDURE GetImmediateManager

1. @employeeID **INT**,
2. @managerID **INT** **OUTPUT**
3. **AS**
4. **BEGIN**
5. **SELECT** @managerID = ManagerID
6. **FROM** HumanResources.Employee
7. **WHERE** EmployeeID = @employeeID
8. **END**

存储过程可能返回更新计数和多个结果集。Microsoft SQL Server 2005 JDBC Driver 遵循 JDBC 3.0 规范，此规范规定在检索 OUT 参数之前应检索多个结果集和更新计数。也就是说，应用程序应先检索所有 ResultSet 对象和更新计数，然后使用 CallableStatement.getter 方法检索 OUT 参数。否则，当检索 OUT 参数时，尚未检索的 ResultSet 对象和更新计数将丢失。

**4 使用带有返回状态的存储过程**

使用 JDBC 驱动程序调用这种存储过程时，必须结合 SQLServerConnection 类的 prepareCall 方法使用 call SQL 转义序列。返回状态参数的 call 转义序列的语法如下所示：

1. {[?=]call **procedure**-**name**[([parameter][,[parameter]]...)]}

构造 call 转义序列时，请使用 ?(问号)字符来指定返回状态参数。此字符充当要从该存储过程返回的参数值的占位符。要为返回状态参数指定值，必须在执行存储过程前使用 SQLServerCallableStatement 类的 registerOutParameter 方法指定参数的数据类型。

此外，向 registerOutParameter 方法传递返回状态参数值时，不仅需要指定要使用的参数的数据类型，还必须指定参数在存储过程中的序数位置。对于返回状态参数，其序数位置始终为 1，这是因为它始终是调用存储过程时的第一个参数。尽管 SQLServerCallableStatement 类支持使用参数的名称来指示特定参数，但您只能对返回状态参数使用参数的序号位置编号。

作为实例，在 SQL Server 2005 AdventureWorks 示例数据库中创建以下存储过程：

1. **CREATE** **PROCEDURE** CheckContactCity
2. (@cityName **CHAR**(50))
3. **AS**
4. **BEGIN**
5. IF ((**SELECT** COUNT(\*)
6. **FROM** Person.Address
7. **WHERE** City = @cityName) > 1)
8. **RETURN** 1
9. **ELSE**
10. **RETURN** 0
11. **END**

该存储过程返回状态值 1 或 0，这取决于是否能在表 Person.Address 中找到 cityName 参数指定的城市。

在下面的实例中，将向此函数传递 AdventureWorks 示例数据库的打开连接，然后使用 execute 方法调用 CheckContactCity 存储过程：

1. **public** **static** void executeStoredProcedure(**Connection** con) ...{
2. try ...{
3. CallableStatement cstmt = con.prepareCall("{? = call dbo.CheckContactCity(?)}");
4. cstmt.registerOutParameter(1, java.sql.Types.**INTEGER**);
5. cstmt.setString(2, "Atlanta");
6. cstmt.**execute**();
7. System.**out**.println("RETURN STATUS: " + cstmt.getInt(1));
8. }
9. cstmt.**close**();
10. catch (Exception e) ...{
11. e.printStackTrace();
12. }
13. }

**5 使用带有更新计数的存储过程**

使用 SQLServerCallableStatement 类构建对存储过程的调用之后，可以使用 execute 或 executeUpdate 方法中的任意一个来调用此存储过程。executeUpdate 方法将返回一个 int 值，该值包含受此存储过程影响的行数，但 execute 方法不返回此值。如果使用 execute 方法，并且希望获得受影响的行数计数，则可以在运行存储过程后调用 getUpdateCount 方法。

作为实例，在 SQL Server 2005 AdventureWorks 示例数据库中创建以下表和存储过程：

1. **CREATE** **TABLE** TestTable
2. (Col1 **int** IDENTITY,
3. Col2 **varchar**(50),
4. Col3 **int**);
6. **CREATE** **PROCEDURE** UpdateTestTable
7. @Col2 **varchar**(50),
8. @Col3 **int**
9. **AS**
10. **BEGIN**
11. **UPDATE** TestTable
12. **SET** Col2 = @Col2, Col3 = @Col3
13. **END**;

在下面的实例中，将向此函数传递 AdventureWorks 示例数据库的打开连接，并使用 execute 方法调用 UpdateTestTable 存储过程，然后使用 getUpdateCount 方法返回受存储过程影响的行计数。

1. **public** **static** void executeUpdateStoredProcedure(**Connection** con) ...{
2. try ...{
3. CallableStatement cstmt = con.prepareCall("{call dbo.UpdateTestTable(?, ?)}");
4. cstmt.setString(1, "A");
5. cstmt.setInt(2, 100);
6. cstmt.**execute**();
7. **int** count = cstmt.getUpdateCount();
8. cstmt.**close**();
10. System.**out**.println("ROWS AFFECTED: " + count);
11. }
12. catch (Exception e) ...{
13. e.printStackTrace();