JDBC基础教程之CallableStatement PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/253/2021_2022_JDBC_E5_9F_BA_E7_A1_80_c97_253050.htm 概述 CallableStatement 对象为所有的 DBMS 提供了一种以标准形式调用已储存过程的方法。已储存过程储存在数据库中。对已储存过程的调用是CallableStatement对象所含的内容。这种调用是用一种换码语法来写的，有两种形式：一种形式带结果参，另一种形式不带结果参数。结果参数是一种输出 (OUT) 参数，是已储存过程的返回值。两种形式都可带有数量可变的输入（IN 参数）、输出（OUT 参数）或输入和输出（INOUT 参数）的参数。问号将用作参数的占位符。 在 JDBC 中调用已储存过程的语法如下所示。注意，方括号表示其间的内容是可选项；方括号本身并不是语法的组成部份。{call 过程名[(?, ?, ...)]} 返回结果参数的过程的语法为：{? = call 过程名[(?, ?, ...)]} 不带参数的已储存过程的语法类似：{call 过程名} 通常，创建 CallableStatement 对象的人应当知道所用的 DBMS 是支持已储存过程的，并且知道这些过程都是些什么。然而，如果需要检查，多种DatabaseMetaData 方法都可以提供这样的信息。例如，如果 DBMS 支持已储存过程的调用，则supportsStoredProcedures 方法将返回 true，而getProcedures 方法将返回对已储存过程的描述。CallableStatement 继承 Statement 的方法（它们用于处理一般的 SQL 语句），还继承了 PreparedStatement 的方法（它们用于处理 IN 参）。CallableStatement 中定义的所有方法都用于处理 OUT 参数或 INOUT 参数的输出部分：注册 OUT 参数的 JDBC 类型（一

般 SQL 类型）、从这些参数中检索结果，或者检查所返回的值是否为 JDBC NULL。 1、创建 CallableStatement 对象 CallableStatement 对象是用 Connection 方法 prepareCall 创建的。下例创建 CallableStatement 的实例，其中含有对已储存过程 getTestData 调用。该过程有两个变量，但不含结果参数：CallableStatement cstmt = con.prepareCall("{call getTestData(?, ?)}"). 其中?占位符为IN、OUT还是INOUT参数，取决于已储存过程getTestData。 2、IN和OUT参数 将IN参数传给 CallableStatement 对象是通过 setXXX 方法完成的。该方法继承自 PreparedStatement。所传入参数的类型决定了所用的setXXX 方法（例如，用 setFloat 来传入 float 值等）。 如果已储存过程返回 OUT 参数，则在执行 CallableStatement 对象以前必须先注册每个 OUT 参数的 JDBC 类型（这是必需的，因为某些 DBMS 要求 JDBC 类型）。注册 JDBC 类型是用 registerOutParameter 方法来完成的。语句执行完后，CallableStatement 的 getXXX 方法将取回参数值。正确的 getXXX 方法是为各参数所注册的 JDBC 类型所对应的 Java 类型。换言之， registerOutParameter 使用的是 JDBC 类型（因此它与数据库返回的 JDBC 类型匹配），而 getXXX 将之转换为 Java 类型。 作为示例，下述代码先注册 OUT 参数，执行由 cstmt 所调用的已储存过程，然后检索在 OUT 参数中返回的值。方法 getByte 从第一个 OUT 参数中取出一个 Java 字节，而 getBigDecimal 从第二个 OUT 参数中取出一个 BigDecimal 对象（小数点后面带三位数）：CallableStatement cstmt = con.prepareCall("{call getTestData(?, ?)}").cstmt.registerOutParameter(1,

Java.sql.Types.TINYINT).cstmt.registerOutParameter(2, Java.sql.Types.DECIMAL, 3).cstmt.executeQuery().byte x = cstmt.getByte(1).Java.math.BigDecimal n = cstmt.getBigDecimal(2, 3). CallableStatement 与 ResultSet 不同，它不提供用增量方式检索大 OUT 值的特殊机制。 3、INOUT参数 既支持输入又接受输出的参数（INOUT 参数）除了调用 registerOutParameter 方法外，还要求调用适当的 setXXX 方法（该方法是从 PreparedStatement 继承来的）。setXXX 方法将参数值设置为输入参数，而 registerOutParameter 方法将它的 JDBC 类型注册为输出参数。setXXX 方法提供一个 Java 值，而驱动程序先把这个值转换为 JDBC 值，然后将它送到数据库中。这种 IN 值的 JDBC 类型和提供给 registerOutParameter 方法的 JDBC 类型应该相同。然后，要检索输出值，就要用对应的 getXXX 方法。例如，Java 类型为byte 的参数应该使用方法 setByte 来赋输入值。应该给registerOutParameter 提供类型为 TINYINT 的 JDBC 类型，同时应使用 getByte 来检索输出值。 下例假设有一个已储存过程 reviseTotal，其唯一参数是 INOUT 参数。方法setByte 把此参数设为 25，驱动程序将把它作为 JDBC TINYINT 类型送到数据库中。接着，registerOutParameter 将该参数注册为 JDBC TINYINT。执行完该已储存过程后，将返回一个新的 JDBC TINYINT 值。方法 getByte 将把这个新值作为 Java byte 类型检索。CallableStatement cstmt = con.prepareCall("{call reviseTotal(?)}").cstmt.setByte(1, 25).cstmt.registerOutParameter(1, Java.sql.Types.TINYINT).cstmt.executeUpdate().byte x = cstmt.getByte(1). 4、先检索结果，再检索 OUT 参数 由于某些

DBMS 的限制，为了实现最大的可移植性，建议先检索由执行CallableStatement 对象所产生的结果，然后再用 CallableStatement.getXXX 方法来检索 OUT 参数。如果 CallableStatement 对象返回多个 ResultSet 对象（通过调用 execute 方法），在检索 OUT 参数前应先检索所有的结果。这种情况下，为确保对所有的结果都进行了访问，必须对 Statement 方法 getResultSet、getUpdateCount 和getMoreResults 进行调用，直到不再有结果为止。 检索完所有的结果后，就可用 CallableStatement.getXXX 方法来检索 OUT 参数中的值。
5、检索作为OUT参数的NULL值 返回到 OUT 参数中的值可能会是JDBC NULL。当出现这种情形时，将对 JDBC NULL 值进行转换以使 getXXX 方法所返回的值为 null、0 或 false，这取决于getXXX 方法类型。对于 ResultSet 对象，要知道0或false是否源于JDBCNULL的唯一方法，是用方法wasNull进行检测。如果 getXXX 方法读取的最后一个值是 JDBC NULL，则该方法返回 true，否则返回 flase。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com