Spring JdbcTemplate方法详解

摘要: Spring JdbcTemplate方法详解

JdbcTemplate主要提供以下五类方法：

- **execute方法：**可以用于执行任何SQL语句，一般用于执行DDL语句；
- **update方法及batchUpdate方法：**update方法用于执行新增、修改、删除等语句；batchUpdate方法用于执行批处理相关语句；
- **query方法及queryForXXX方法：**用于执行查询相关语句；
- **call方法：**用于执行存储过程、函数相关语句。

JdbcTemplate类支持的回调类：

- **预编译语句及存储过程创建回调：**用于根据JdbcTemplate提供的连接创建相应的语句；

PreparedStatementCreator：通过回调获取JdbcTemplate提供的Connection，由用户使用该Conncetion创建相关的PreparedStatement；

CallableStatementCreator：通过回调获取JdbcTemplate提供的Connection，由用户使用该Conncetion创建相关的CallableStatement；

- **预编译语句设值回调：**用于给预编译语句相应参数设值；

PreparedStatementSetter：通过回调获取JdbcTemplate提供的PreparedStatement，由用户来对相应的预编译语句相应参数设值；

BatchPreparedStatementSetter：；类似于PreparedStatementSetter，但用于批处理，需要指定批处理大小；

- **自定义功能回调：**提供给用户一个扩展点，用户可以在指定类型的扩展点执行任何数量需要的操作；

ConnectionCallback：通过回调获取JdbcTemplate提供的Connection，用户可在该Connection执行任何数量的操作；

StatementCallback：通过回调获取JdbcTemplate提供的Statement，用户可以在该Statement执行任何数量的操作；

PreparedStatementCallback：通过回调获取JdbcTemplate提供的PreparedStatement，用户可以在该PreparedStatement执行任何数量的操作；

CallableStatementCallback：通过回调获取JdbcTemplate提供的CallableStatement，用户可以在该CallableStatement执行任何数量的操作；

- **结果集处理回调**：通过回调处理ResultSet或将ResultSet转换为需要的形式；

RowMapper：用于将结果集每行数据转换为需要的类型，用户需实现方法mapRow(ResultSet rs, int rowNum)来完成将每行数据转换为相应的类型。

RowCallbackHandler：用于处理ResultSet的每一行结果，用户需实现方法processRow(ResultSet rs)来完成处理，在该回调方法中无需执行rs.next()，该操作由JdbcTemplate来执行，用户只需按行获取数据然后处理即可。

ResultSetExtractor：用于结果集数据提取，用户需实现方法extractData(ResultSet rs)来处理结果集，用户必须处理整个结果集；

接下来让我们看下具体示例吧，在示例中不可能介绍到JdbcTemplate全部方法及回调类的使用方法，我们只介绍代表性的，其余的使用都是类似的；

1）预编译语句及存储过程创建回调、自定义功能回调使用：

java代码：

Java代码

```
public void testPpreparedStatement1() {

  int count = jdbcTemplate.execute(new PreparedStatementCreator() {

     @Override

     public PreparedStatement createPreparedStatement(Connection conn)

         throws SQLException {

         return conn.prepareStatement("select count(*) from test");

     }}, new PreparedStatementCallback<Integer>() {

     @Override

     public Integer doInPreparedStatement(PreparedStatement pstmt)

         throws SQLException, DataAccessException {
```

```
        pstmt.execute();

        ResultSet rs = pstmt.getResultSet();

        rs.next();

        return rs.getInt(1);

    }});

   Assert.assertEquals(0, count);

}
```

首先使用PreparedStatementCreator创建一个预编译语句，其次由JdbcTemplate通过PreparedStatementCallback回调传回，由用户决定如何执行该PreparedStatement。此处我们使用的是execute方法。

2）预编译语句设值回调使用：

java代码：

Java代码

```
public void testPreparedStatement2() {

  String insertSql = "insert into test(name) values (?)";

  int count = jdbcTemplate.update(insertSql, new PreparedStatementSetter() {

      @Override

      public void setValues(PreparedStatement pstmt) throws SQLException {

          pstmt.setObject(1, "name4");

  }});

  Assert.assertEquals(1, count);

  String deleteSql = "delete from test where name=?";

  count = jdbcTemplate.update(deleteSql, new Object[] {"name4"});
```

```
  Assert.assertEquals(1, count);

}
```

通过JdbcTemplate的int update(String sql, PreparedStatementSetter pss)执行预编译sql，其中sql参数为“insert into test(name) values (?) ”，该sql有一个占位符需要在执行前设值，PreparedStatementSetter实现就是为了设值，使用setValues(PreparedStatement pstmt)回调方法设值相应的占位符位置的值。JdbcTemplate也提供一种更简单的方式“update(String sql, Object... args)”来实现设值，所以只要当使用该种方式不满足需求时才应使用PreparedStatementSetter。

3）结果集处理回调：

java代码：

Java代码

```
public void testResultSet1() {

  jdbcTemplate.update("insert into test(name) values('name5')");

  String listSql = "select * from test";

  List result = jdbcTemplate.query(listSql, new RowMapper<Map>() {

      @Override

      public Map mapRow(ResultSet rs, int rowNum) throws SQLException {

          Map row = new HashMap();

          row.put(rs.getInt("id"), rs.getString("name"));

          return row;

  }});

  Assert.assertEquals(1, result.size());

  jdbcTemplate.update("delete from test where name='name5'");

}
```

RowMapper接口提供mapRow(ResultSet rs, int rowNum)方法将结果集的每一行转换为一个Map，当然可以转换为其他类，如表的对象画形式。

java代码：

Java代码

```
public void testResultSet2() {
  jdbcTemplate.update("insert into test(name) values('name5')");
  String listSql = "select * from test";
  final List result = new ArrayList();
  jdbcTemplate.query(listSql, new RowCallbackHandler() {
      @Override
      public void processRow(ResultSet rs) throws SQLException {
          Map row = new HashMap();
          row.put(rs.getInt("id"), rs.getString("name"));
          result.add(row);
  }});
  Assert.assertEquals(1, result.size());
  jdbcTemplate.update("delete from test where name='name5'");
}
```

RowCallbackHandler接口也提供方法processRow(ResultSet rs)，能将结果集的行转换为需要的形式。

java代码：

Java代码

```
public void testResultSet3() {

  jdbcTemplate.update("insert into test(name) values('name5')");

  String listSql = "select * from test";

  List result = jdbcTemplate.query(listSql, new ResultSetExtractor<List>() {

      @Override

      public List extractData(ResultSet rs)

     throws SQLException, DataAccessException {

          List result = new ArrayList();

          while(rs.next()) {

              Map row = new HashMap();

              row.put(rs.getInt("id"), rs.getString("name"));

              result.add(row);

           }

           return result;

  }});

  Assert.assertEquals(0, result.size());

  jdbcTemplate.update("delete from test where name='name5'");

}
```

ResultSetExtractor使用回调方法extractData(ResultSet rs)提供给用户整个结果集，让用户决定如何处理该结果集。

当然JdbcTemplate提供更简单的queryForXXX方法，来简化开发：

java代码：

Java代码

```
//1.查询一行数据并返回int型结果
```

```
jdbcTemplate.queryForInt("select count(*) from test");

//2. 查询一行数据并将该行数据转换为Map返回

jdbcTemplate.queryForMap("select * from test where name='name5'");

//3.查询一行任何类型的数据，最后一个参数指定返回结果类型

jdbcTemplate.queryForObject("select count(*) from test", Integer.class);

//4.查询一批数据，默认将每行数据转换为Map

jdbcTemplate.queryForList("select * from test");

//5.只查询一列数据列表，列类型是String类型，列名字是name

jdbcTemplate.queryForList("

select name from test where name=?", new Object[]{"name5"}, String.class);

//6.查询一批数据，返回为SqlRowSet，类似于ResultSet，但不再绑定到连接上

SqlRowSet rs = jdbcTemplate.queryForRowSet("select * from test");
```

3） 存储过程及函数回调：

首先修改JdbcTemplateTest的setUp方法，修改后如下所示：

java代码：

Java代码 ☆

```


@Before

public void setUp() {

    String createTableSql = "create memory table test" +

    "(id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY, " +

    "name varchar(100))";

    jdbcTemplate.update(createTableSql);
```

```
    String createHsqldbFunctionSql =
      "CREATE FUNCTION FUNCTION_TEST(str CHAR(100)) " +
      "returns INT begin atomic return length(str);end";
    jdbcTemplate.update(createHsqldbFunctionSql);
    String createHsqldbProcedureSql =
      "CREATE PROCEDURE PROCEDURE_TEST" +
      "(INOUT inOutName VARCHAR(100), OUT outId INT) " +
      "MODIFIES SQL DATA " +
      "BEGIN ATOMIC " +
      "  insert into test(name) values (inOutName); " +
      "  SET outId = IDENTITY(); " +
      "  SET inOutName = 'Hello,' + inOutName; " +
    "END";
    jdbcTemplate.execute(createHsqldbProcedureSql);
}
```

其中CREATE FUNCTION FUNCTION_TEST用于创建自定义函数，CREATE PROCEDURE PROCEDURE_TEST用于创建存储过程，注意这些创建语句是数据库相关的，本示例中的语句只适用于HSQLDB数据库。

其次修改JdbcTemplateTest的tearDown方法，修改后如下所示：

java代码：

Java代码

```
public void tearDown() {
```

```
    jdbcTemplate.execute("DROP FUNCTION FUNCTION_TEST");

    jdbcTemplate.execute("DROP PROCEDURE PROCEDURE_TEST");

    String dropTableSql = "drop table test";

    jdbcTemplate.execute(dropTableSql);

}
```

其中drop语句用于删除创建的存储过程、自定义函数及数据库表。

接下来看一下hsqldb如何调用自定义函数：

java代码：

Java代码

```

public void testCallableStatementCreator1() {

    final String callFunctionSql = "{call FUNCTION_TEST(?)}";

    List<SqlParameter> params = new ArrayList<SqlParameter>();

    params.add(new SqlParameter(Types.VARCHAR));

    params.add(new SqlReturnResultSet("result",

       new ResultSetExtractor<Integer>() {

           @Override

           public Integer extractData(ResultSet rs) throws SQLException,

               DataAccessException {

               while(rs.next()) {

                   return rs.getInt(1);

               }

              return 0;

       }));

```

```
    Map<String, Object> outValues = jdbcTemplate.call(

       new CallableStatementCreator() {

            @Override

            public CallableStatement createCallableStatement(Connection conn) throws SQLException
{

              CallableStatement cstmt = conn.prepareCall(callFunctionSql);

              cstmt.setString(1, "test");

              return cstmt;

    }}, params);

    Assert.assertEquals(4, outValues.get("result"));

}
```

- **{call FUNCTION_TEST(?)}**：定义自定义函数的sql语句，注意hsqldb {?= call …}和{call …}含义是一样的，而比如mysql中两种含义是不一样的；
- params：用于描述自定义函数占位符参数或命名参数类型；SqlParameter用于描述IN类型参数、SqlOutParameter用于描述OUT类型参数、SqlInOutParameter用于描述INOUT类型参数、SqlReturnResultSet用于描述调用存储过程或自定义函数返回的ResultSet类型数据，其中SqlReturnResultSet需要提供结果集处理回调用于将结果集转换为相应的形式，hsqldb自定义函数返回值是ResultSet类型。
- **CallableStatementCreator**：提供Connection对象用于创建CallableStatement对象
- outValues：调用call方法将返回类型为Map<String, Object>对象；
- **outValues.get("result")**：获取结果，即通过SqlReturnResultSet对象转换过的数据；其中SqlOutParameter、SqlInOutParameter、SqlReturnResultSet指定的name用于从call执行后返回的Map中获取相应的结果，即name是Map的键。

注：因为hsqldb {?= call …}和{call …}含义是一样的，因此调用自定义函数将返回一个包含结果的ResultSet。

最后让我们示例下mysql如何调用自定义函数：

java代码：

Java代码 ☆

```
public void testCallableStatementCreator2() {
    JdbcTemplate mysqlJdbcTemplate = new JdbcTemplate(getMysqlDataSource);
    //2.创建自定义函数
String createFunctionSql =
    "CREATE FUNCTION FUNCTION_TEST(str VARCHAR(100)) " +
     "returns INT return LENGTH(str)";
String dropFunctionSql = "DROP FUNCTION IF EXISTS FUNCTION_TEST";
mysqlJdbcTemplate.update(dropFunctionSql);
mysqlJdbcTemplate.update(createFunctionSql);
//3.准备sql,mysql支持{?= call …}
final String callFunctionSql = "{?= call FUNCTION_TEST(?)}";
//4.定义参数
List<SqlParameter> params = new ArrayList<SqlParameter>();
params.add(new SqlOutParameter("result", Types.INTEGER));
params.add(new SqlParameter("str", Types.VARCHAR));
Map<String, Object> outValues = mysqlJdbcTemplate.call(
new CallableStatementCreator() {
    @Override
    public CallableStatement createCallableStatement(Connection conn) throws SQLException {
      CallableStatement cstmt = conn.prepareCall(callFunctionSql);
      cstmt.registerOutParameter(1, Types.INTEGER);
      cstmt.setString(2, "test");
        return cstmt;
    }}, params);

```

```
   Assert.assertEquals(4, outValues.get("result"));

}

public DataSource getMysqlDataSource() {

    String url = "jdbc:mysql://localhost:3306/test";

    DriverManagerDataSource dataSource =

        new DriverManagerDataSource(url, "root", "");     dataSource.setDriverClassName("com.mysql
.jdbc.Driver");

    return dataSource;

}
```

- **getMysqlDataSource**：首先启动mysql（本书使用5.4.3版本），其次登录mysql创建test数据库（“create database test;”），在进行测试前，请先下载并添加mysql-connector-java-5.1.10.jar到classpath；
- **{?= call FUNCTION_TEST(?)}**：可以使用{?= call …}形式调用自定义函数；
- **params**：无需使用SqlReturnResultSet提取结果集数据，而是使用SqlOutParameter来描述自定义函数返回值；
- **CallableStatementCreator**：同上个例子含义一样；
- **cstmt.registerOutParameter(1, Types.INTEGER)**：将OUT类型参数注册为JDBC类型Types.INTEGER，此处即返回值类型为Types.INTEGER。
- **outValues.get("result")**：获取结果，直接返回Integer类型，比hsqldb简单多了吧。

最后看一下如何如何调用存储过程：

java代码：

Java代码 ☆

```

public void testCallableStatementCreator3() {

```

```
    final String callProcedureSql = "{call PROCEDURE_TEST(?, ?)}";

    List<SqlParameter> params = new ArrayList<SqlParameter>();

    params.add(new SqlInOutParameter("inOutName", Types.VARCHAR));

    params.add(new SqlOutParameter("outId", Types.INTEGER));

    Map<String, Object> outValues = jdbcTemplate.call(

      new CallableStatementCreator() {

        @Override

        public CallableStatement createCallableStatement(Connection conn) throws SQLException {

          CallableStatement cstmt = conn.prepareCall(callProcedureSql);

          cstmt.registerOutParameter(1, Types.VARCHAR);

          cstmt.registerOutParameter(2, Types.INTEGER);

          cstmt.setString(1, "test");

          return cstmt;

    }}, params);

    Assert.assertEquals("Hello,test", outValues.get("inOutName"));

    Assert.assertEquals(0, outValues.get("outId"));

}
```

- **{call PROCEDURE_TEST(?, ?)}**：定义存储过程sql；
- **params**：定义存储过程参数；SqlInOutParameter描述INOUT类型参数、SqlOutParameter描述OUT类型参数；
- CallableStatementCreator：用于创建CallableStatement，并设值及注册OUT参数类型；
- outValues：通过SqlInOutParameter及SqlOutParameter参数定义的name来获取存储过程结果。

JdbcTemplate类还提供了很多便利方法，在此就不一一介绍了，但这些方法是由规律可循的，第一种就是提供回调接口让用户决定做什么，第二种可以认为是便利方法（如queryForXXX），用于那些比较简单的操作。

来源： https://my.oschina.net/u/437232/blog/279530#comment-list