**基本概念**

表值参数（Table-Valued Parameter），是SQLServer的一种自定义类型，在其他数据库中可能也有类似类型。  
它描述了表结构，可用于封装该表的数据。  
在java中，可用于传递某张的表多行数据给数据库（SQLServer）。  
可用于预处理SQL语句（PreparedStatement）的参数，也可用于存储过程的参数。

微软官方资料（https://docs.microsoft.com/en-us/sql/connect/jdbc/using-table-valued-parameters）  
介绍了如何在jdbc中使用表值参数传递数据。  
数据可封装在3种类型中：SQLServerDataTable，ResultSet，ISQLServerDataRecord。  
1.SQLServerDataTable可方便的将一组静态数据（List<XxxxModel>）转换成表值参数数据  
2.ResultSet是数据结果集  
3.ISQLServerDataRecord是一个接口，需有实现类。类似一个“数据源”，可用于特殊用途。假如用于List<XxxxModel>数据，转换较麻烦。

针对不同数据库，jdbc有一个跨数据库的api，在某些框架内（Mybatis）“只能”使用这些通用api  
1.PreparedStatement的类有一个setObject方法，调用存储过程，可用此方法传递表值参数。  
2.PreparedStatement的类还有一个setArray的方法，我不太清楚其用法，今后可多学习。

**Jdbc中的使用**

数据初始化，创建表，表值参数，存储过程

*-- 表***CREATE TABLE** z\_table (  
[id] **int NOT NULL** ,  
[**name**] **varchar**(255) **NOT NULL**)  
*-- 表值参数(Table-Valued Parameter)***CREATE TYPE** dbo.z\_type **AS TABLE** ( id **int**, **name varchar**(255) )  
*-- 存储过程***CREATE** PROC z\_proc(@c z\_type READONLY)  
**as  
INSERT** z\_table **SELECT** *\** **FROM** @c

模型类

**public class** ZTableModel {  
 **private int id**;  
 **private** String **name**;

// … get/set, toString, Constructor

}

表值对象“数据源”，用于第3种使用方式（ISQLServerDataRecord）

**public class** ZTypeRecord **implements** ISQLServerDataRecord {

// SQLServerMetaData getColumnMetaData()描述表值对象列的信息，列名和类型

// int getColumnCount()描述表值对象有多少列

// Object[] getRowData()每行的数据

// boolean next()是否有下一行

}

具体使用

1. 调用sql语句，传递SQLServerDataTable

SQLServerDataTable sourceDataTable = **new** SQLServerDataTable();  
sourceDataTable.addColumnMetadata(**"id"** , Types.***INTEGER***);  
sourceDataTable.addColumnMetadata(**"name"** , Types.***VARCHAR***);  
sourceDataTable.addRow(1, **"a"**);  
sourceDataTable.addRow(2, **"b"**);  
  
SQLServerPreparedStatement pStmt =  
 (SQLServerPreparedStatement) connection.prepareStatement(**"INSERT INTO dbo.z\_table SELECT** *\** **FROM ?"**);  
pStmt.setStructured(1, **"dbo.z\_type"**, sourceDataTable);  
pStmt.execute();

2. 调用存储过程，传递ResultSet

ResultSet rs = connection.createStatement().executeQuery(**"SELECT intColumn, varcharColumn FROM XxxTable"**);  
  
SQLServerCallableStatement pStmt =  
 (SQLServerCallableStatement) connection.prepareCall(**"exec z\_proc ?"**);  
pStmt.setObject(1, rs);  
pStmt.execute();

3. 调用sql语句，传递ISQLServerDataRecord

List<ZTableModel> list = **new** ArrayList<>();  
list.add(**new** ZTableModel(1, **"a"**));  
list.add(**new** ZTableModel(2, **"b"**));  
ISQLServerDataRecord zTypeRecord = **new** ZTypeRecord(list);  
  
SQLServerPreparedStatement pStmt =  
 (SQLServerPreparedStatement) connection.prepareStatement(**"INSERT INTO dbo.z\_table SELECT** *\** **FROM ?"**);  
pStmt.setStructured(1, **"dbo.z\_type"**, zTypeRecord);  
pStmt.execute();

**Mybatis中使用**

因为Mybatis对PreparedStatement采用了jdk动态代理的技术。在代码中无法追踪到SQLServerCallableStatement，仅能追踪到$Proxy134这种类，所以我估计仅能用于存储过程传参。

且Mybatis要新建自定义TypeHandler才能处理特殊类型

* 新建ZTypeTypeHandler继承Mybatis的BaseTypeHandler

ZTypeTypeHandler的setNonNullParameter方法中

// parameter是参数(List<ZTableModel>)

**public void** setNonNullParameter

(PreparedStatement ps, **int** i, Object parameter, JdbcType jdbcType)

**throws** SQLException {

SQLServerDataTable sourceDataTable = **new** SQLServerDataTable();  
sourceDataTable.addColumnMetadata(**"id"** ,java.sql.Types.***INTEGER***);  
sourceDataTable.addColumnMetadata(**"name"** , Types.***VARCHAR***);  
List<ZTableModel> dataList = (List<ZTableModel>)parameter;  
**for** (ZTableModel z : dataList) {  
 sourceDataTable.addRow(z.getId(), z.getName());  
}  
ps.setObject(i, sourceDataTable);

}

* Spring配置文件中指定handler

<**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>

// 具体配置……

<**property name="typeHandlers" ref="zTypeTypeHandler"**/>  
</**bean**>

* Dao中

**void** callTableType(List<ZTableModel> list);

* Mapper中需要显示指定TypeHandler

<**update id="callTableType" parameterType="list" statementType="CALLABLE"**>  
 <![CDATA[  
 {call *z\_proc*(#{list,typeHandler=com.hsnn.basic.common.models.ZTypeTypeHandler})}  
 ]]>  
</**update**>