Atitit mybatis的扩展使用sql udf,js java等语言

目录

[1.1. 默认，mybatis使用xml，sql等语言来书写业务流程 1](#_Toc27681)

[2. 使用java扩展函数 1](#_Toc9195)

[2.1. 1.TypeHandler概念  1](#_Toc6096)

[2.2. 自定义类型处理器 2](#_Toc7882)

[2.3. 配置注册自定义处理器（mybatis.cfg.xml） 2](#_Toc1168)

[2.4. 使用 3](#_Toc15118)

[3. 默认TypeHandle}实现45个 4](#_Toc6228)

[3.1. StringTypeHandler 6](#_Toc5555)

## 默认，mybatis使用xml，sql等语言来书写业务流程

# 使用java扩展函数

## 1.TypeHandler概念

TypeHandler,类型转换器,在mybatis中用于实现java类型和JDBC类型的相互转换.mybatis使用prepareStatement来进行参数设置的时候,需要通过typeHandler将传入的java参数设置成合适的jdbc类型参数,这个过程实际上是通过调用PrepareStatement不同的set方法实现的;在获取结果返回之后,也需要将返回的结果转换成我们需要的java类型,这时候是通过调用ResultSet对象不同类型的get方法时间的;所以不同类型的typeHandler其实就是调用PrepareStatement和ResultSet的不同方法来进行类型的转换,有些时候会在调用PrepareStatement和ResultSet的相关方法之前,可以对传入的参数进行一定的处理.

---------------------

使用场景：mybatis在预处理语句（PreparedStatement）中设置一个参数时，或者从结果集（ResultSet）中取出一个值时，都会用到TypeHandler。它的作用就是将java类型（javaType）转化为jdbc类型（jdbcType），或者将jdbc类型（jdbcType）转化为java类型（javaType）。

## 自定义类型处理器

* 实现TypeHandler接口或者继承BaseTypehandler  
  TypeHandler是一个接口，它定义了如下四个方法，实现类必须去实现，方法如下：

void setParameter(PreparedStatement var1, int var2, T var3,JdbcType var4) throws SQLException;

T getResult(ResultSet var1, String var2) throws SQLException;

T getResult(ResultSet var1, int var2) throws SQLException;

T getResult(CallableStatement var1, int var2) throws SQLException;

}

setParameter：通过preparedStatement对象设置参数，将T类型的数据存入数据库。

getResult：通过列名或者下标来获取结果数据，也可以通过CallableStatement获取数据。

## 配置注册自定义处理器（mybatis.cfg.xml）

<!--自定义类型处理器-->

<typeHandlers>

<typeHandler handler="com.mdd.mybatis.typehandle.MyTypeHandle"></typeHandler>

</typeHandlers>

最后，可以让 MyBatis 为你查找类型处理器：

<!-- mybatis-config.xml --><typeHandlers>

<package name="org.mybatis.example"/></typeHandlers>

注意在使用自动检索（autodiscovery）功能的时候，只能通过注解方式来指定 JDBC 的类型。

## 使用

不是使用的常见的双括号函数形式，而是模板语法指定处理器

#{name,typeHandler=com.mdd.mybatis.typehandle.MyTypeHandle},

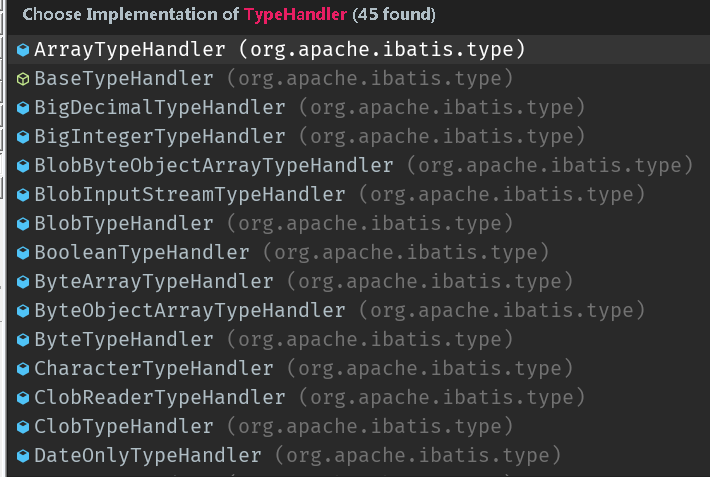
但注册完成之后也仍然不能起作用,因为还需要标识那些参数和返回的结果是需要使用这个TypeHandler进行处理的;具体来说,在插入数据和对返回结果进行处理的时候,可以对参数配置javaType和jdbcType或直接配置typeHandler属性来进行标识

下面是插入数据时标识用指定TypeHandler进行处理

---------------------

要注意 MyBatis 不会窥探数据库元信息来决定使用哪种类型，所以你必须在参数和结果映射中指明那是 VARCHAR 类型的字段， 以使其能够绑定到正确的类型处理器上

# 默认TypeHandle}实现45个



public TypeHandlerRegistry() {

register(Boolean.class, new BooleanTypeHandler());

register(boolean.class, new BooleanTypeHandler());

register(JdbcType.BOOLEAN, new BooleanTypeHandler());

register(JdbcType.BIT, new BooleanTypeHandler());

register(Byte.class, new ByteTypeHandler());

register(byte.class, new ByteTypeHandler());

register(JdbcType.TINYINT, new ByteTypeHandler());

register(Short.class, new ShortTypeHandler());

register(short.class, new ShortTypeHandler());

register(JdbcType.SMALLINT, new ShortTypeHandler());

register(Integer.class, new IntegerTypeHandler());

register(int.class, new IntegerTypeHandler());

register(JdbcType.INTEGER, new IntegerTypeHandler());

register(Long.class, new LongTypeHandler());

register(long.class, new LongTypeHandler());

register(Float.class, new FloatTypeHandler());

register(float.class, new FloatTypeHandler());

register(JdbcType.FLOAT, new FloatTypeHandler());

register(Double.class, new DoubleTypeHandler());

register(double.class, new DoubleTypeHandler());

register(JdbcType.DOUBLE, new DoubleTypeHandler());

register(Reader.class, new ClobReaderTypeHandler());

register(String.class, new StringTypeHandler());

register(String.class, JdbcType.CHAR, new StringTypeHandler());

register(String.class, JdbcType.CLOB, new ClobTypeHandler());

register(String.class, JdbcType.VARCHAR, new StringTypeHandler());

register(String.class, JdbcType.LONGVARCHAR, new ClobTypeHandler());

register(String.class, JdbcType.NVARCHAR, new NStringTypeHandler());

register(String.class, JdbcType.NCHAR, new NStringTypeHandler());

register(String.class, JdbcType.NCLOB, new NClobTypeHandler());

register(JdbcType.CHAR, new StringTypeHandler());

register(JdbcType.VARCHAR, new StringTypeHandler());

register(JdbcType.CLOB, new ClobTypeHandler());

register(JdbcType.LONGVARCHAR, new ClobTypeHandler());

register(JdbcType.NVARCHAR, new NStringTypeHandler());

register(JdbcType.NCHAR, new NStringTypeHandler());

register(JdbcType.NCLOB, new NClobTypeHandler());

register(Object.class, JdbcType.ARRAY, new ArrayTypeHandler());

register(JdbcType.ARRAY, new ArrayTypeHandler());

register(BigInteger.class, new BigIntegerTypeHandler());

register(JdbcType.BIGINT, new LongTypeHandler());

register(BigDecimal.class, new BigDecimalTypeHandler());

register(JdbcType.REAL, new BigDecimalTypeHandler());

register(JdbcType.DECIMAL, new BigDecimalTypeHandler());

register(JdbcType.NUMERIC, new BigDecimalTypeHandler());

register(InputStream.class, new BlobInputStreamTypeHandler());

register(Byte[].class, new ByteObjectArrayTypeHandler());

register(Byte[].class, JdbcType.BLOB, new BlobByteObjectArrayTypeHandler());

register(Byte[].class, JdbcType.LONGVARBINARY, new BlobByteObjectArrayTypeHandler());

register(byte[].class, new ByteArrayTypeHandler());

register(byte[].class, JdbcType.BLOB, new BlobTypeHandler());

register(byte[].class, JdbcType.LONGVARBINARY, new BlobTypeHandler());

register(JdbcType.LONGVARBINARY, new BlobTypeHandler());

register(JdbcType.BLOB, new BlobTypeHandler());

register(Object.class, UNKNOWN\_TYPE\_HANDLER);

register(Object.class, JdbcType.OTHER, UNKNOWN\_TYPE\_HANDLER);

register(JdbcType.OTHER, UNKNOWN\_TYPE\_HANDLER);

register(Date.class, new DateTypeHandler());

register(Date.class, JdbcType.DATE, new DateOnlyTypeHandler());

register(Date.class, JdbcType.TIME, new TimeOnlyTypeHandler());

register(JdbcType.TIMESTAMP, new DateTypeHandler());

register(JdbcType.DATE, new DateOnlyTypeHandler());

register(JdbcType.TIME, new TimeOnlyTypeHandler());

register(java.sql.Date.class, new SqlDateTypeHandler());

register(java.sql.Time.class, new SqlTimeTypeHandler());

register(java.sql.Timestamp.class, new SqlTimestampTypeHandler());

---------------------

## StringTypeHandler

*public class* StringTypeHandler *extends* BaseTypeHandler<String> {  
 *public* StringTypeHandler() {  
 }  
  
 *public void* setNonNullParameter(PreparedStatement ps, *int* i, String parameter, JdbcType jdbcType) *throws* SQLException {  
 ps.setString(i, parameter);  
 }