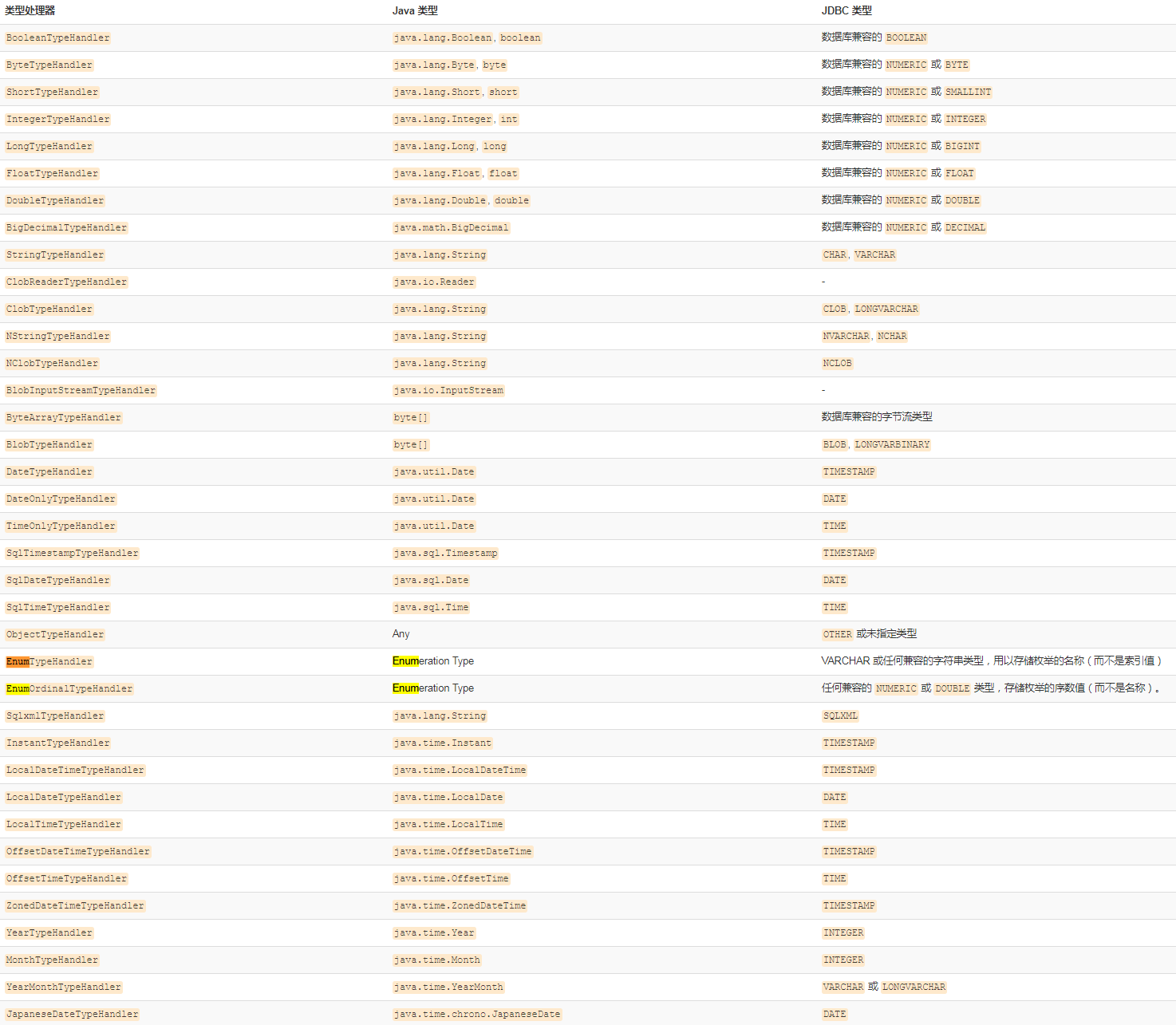
# 1.概述MyBatis类型处理器TypeHandler

无论是 MyBatis 在预处理语句（PreparedStatement）中设置一个参数时，还是从结果集中取出一个值时， 都会用类型处理器将获取的值以合适的方式转换成 Java 类型。下表描述了一些默认的类型处理器。

注意：从 3.4.5 开始，MyBatis 默认支持 JSR-310（日期和时间 API） 。



你可以重写类型处理器或创建你自己的类型处理器来处理不支持的或非标准的类型。 具体做法为：实现 org.apache.ibatis.type.TypeHandler 接口， 或继承一个很便利的类 org.apache.ibatis.type.BaseTypeHandler， 然后可以选择性地将它映射到一个 JDBC 类型。

比如：

*\*\*  
 \** ***@Author*** *了个  
 \** ***@date*** *2020/1/8 13:43  
 \*/*//指定被关联的 JDBC 类型方式1  
@MappedJdbcTypes(JdbcType.*VARCHAR*)  
//@MappedTypes(String.class)  
  
//通过类型处理器的泛型，MyBatis 可以得知该类型处理器处理的 Java 类型  
public class ExampleTypeHandler extends BaseTypeHandler<String> {  
  
 @Override  
 public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType jdbcType) throws SQLException {  
 ps.setString(i, parameter);  
 }  
  
 @Override  
 public String getNullableResult(ResultSet rs, String columnName) throws SQLException {  
 return rs.getString(columnName)+"-->test1";  
 }  
  
 @Override  
 public String getNullableResult(ResultSet rs, int columnIndex) throws SQLException {  
 return rs.getString(columnIndex)+"-->test2";  
 }  
  
 @Override  
 public String getNullableResult(CallableStatement cs, int columnIndex) throws SQLException {  
 return cs.getString(columnIndex)+"-->test3";  
 }  
}

Mybatis配置文件中的

<typeHandlers>  
 <!--这里这里的javaType和jdbcType的配置 非常重要，要特别注意-->  
 <!--<typeHandler handler="extend.use.typeHandlers.MyTypeHandler1" javaType="String" jdbcType="VARCHAR"/>-->  
 <typeHandler handler="extend.use.typeHandlers.MyTypeHandler1"/>  
 <!--指定被关联的 JDBC 类型方式2-->  
 <typeHandler handler="extend.use.typeHandlers.ExampleTypeHandler" **jdbcType="VARCHAR"**/>  
</typeHandlers>

使用上述的类型处理器将会覆盖已经存在的处理 Java 的 String 类型属性和 VARCHAR 参数及结果的类型处理器。 要注意 MyBatis 不会通过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。这是因为 MyBatis 直到语句被执行时才清楚数据类型。

通过类型处理器的泛型，MyBatis 可以得知该类型处理器处理的 Java 类型，不过这种行为可以通过两种方法改变：

* 在类型处理器的配置元素（typeHandler 元素）上增加一个 javaType 属性（比如：javaType="String"）；
* 在类型处理器的类上（TypeHandler class）增加一个 @MappedTypes 注解来指定与其关联的 Java 类型列表。 如果在 javaType 属性中也同时指定，则注解方式将被忽略。

可以通过两种方式来指定被关联的 JDBC 类型：

* 在类型处理器的配置元素上增加一个 jdbcType 属性（比如：jdbcType="VARCHAR"）；
* 在类型处理器的类上增加一个 @MappedJdbcTypes 注解来指定与其关联的 JDBC 类型列表。 如果在 jdbcType 属性中也同时指定，则注解方式将被忽略。

当在 ResultMap 中决定使用哪种类型处理器时，此时 Java 类型是已知的（从结果类型中获得），但是 JDBC 类型是未知的。 因此 Mybatis 使用 javaType=[Java 类型], jdbcType=null 的组合来选择一个类型处理器。 这意味着使用 @MappedJdbcTypes 注解可以*限制*类型处理器的范围，同时除非显式的设置，否则类型处理器在 ResultMap 中将是无效的。 如果希望在 ResultMap 中使用类型处理器，那么设置 @MappedJdbcTypes 注解的 includeNullJdbcType=true 即可。 然而从 Mybatis 3.4.0 开始，如果**只有一个**注册的类型处理器来处理 Java 类型，那么它将是 ResultMap 使用 Java 类型时的默认值（即使没有 includeNullJdbcType=true）。

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>

注意在使用自动发现功能的时候，只能通过注解方式来指定 JDBC 的类型。

你可以创建一个能够处理多个类的泛型类型处理器。为了使用泛型类型处理器， 需要增加一个接受该类的 class 作为参数的构造器，这样在构造一个类型处理器的时候 MyBatis 就会传入一个具体的类。

//GenericTypeHandler.javapublic class GenericTypeHandler<E extends MyObject> extends BaseTypeHandler<E> {

private Class<E> type;

public GenericTypeHandler(Class<E> type) {

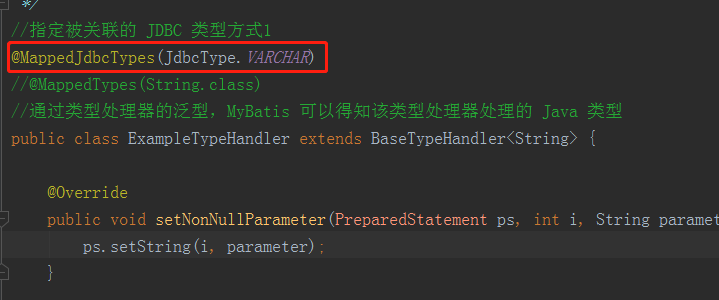
if (type == null) throw new IllegalArgumentException("Type argument cannot be null");

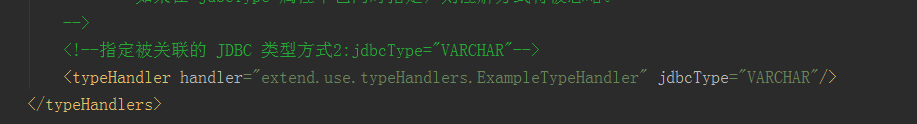
this.type = type;

}

...

**注意：1：：：**

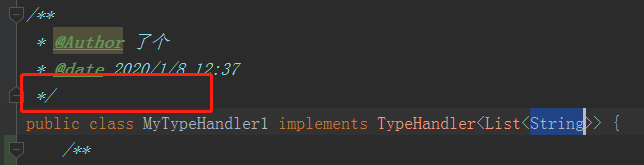




这样配置的话，会对所有返回字段类型为JdbcType.VARCHAR 的都起效果！

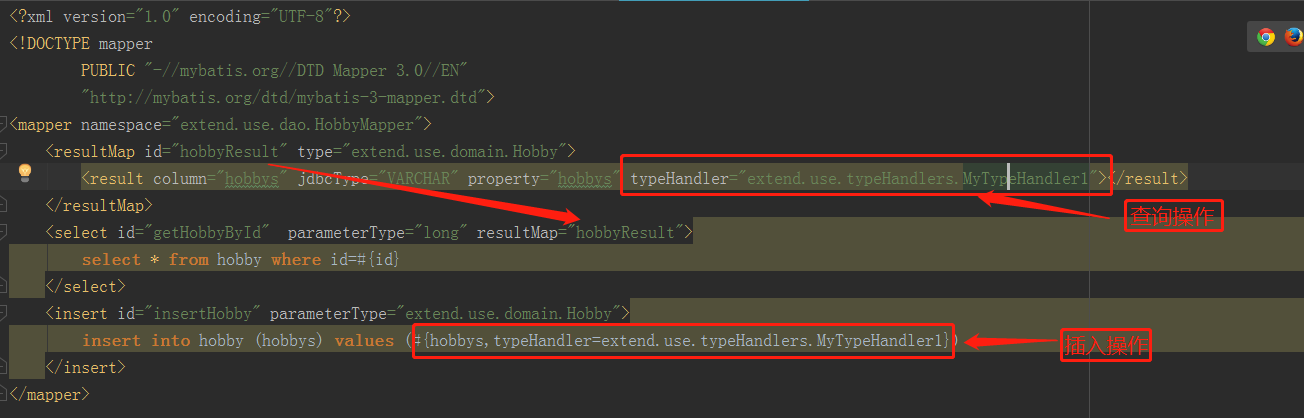
**注意：2：：：**

无果并没有指定 关联的 JDBC 类型（两种方式都没有指定的话）





想上面这样的话，如果想要达到效果，则在使用的使用需要像下面这样：



# 2.demo1

参考：extend.use.typeHandlers

# 3.demo2

参考：extend.use.typeHandlers

# 4.demo3 TODO

你可以创建一个能够处理多个类的泛型类型处理器。为了使用泛型类型处理器， 需要增加一个接受该类的 class 作为参数的构造器，这样在构造一个类型处理器的时候 MyBatis 就会传入一个具体的类。

//GenericTypeHandler.javapublic class GenericTypeHandler<E extends MyObject> extends BaseTypeHandler<E> {

private Class<E> type;

public GenericTypeHandler(Class<E> type) {

if (type == null) throw new IllegalArgumentException("Type argument cannot be null");

this.type = type;

}

...