

从0到1解析Mybatis源码(一)

核心流程、字段映射、数据转换

讲师: 岑凯伦

个人介绍

- 上海海事大学14届网络工程本科
- 上海海事大学16届计算机专业硕士
- 阿里巴巴、顺丰实习一年
- 美团点评后端Java工程师
- 技术、写作、运动爱好者
- 邮箱: kailuncen@163.com
- 网站: <http://kailuncen.me/about>



公众号: KailunTalk

目录

- JDBC时代,我们如何开发
- Mybatis时代的开发流程
- Mybatis框架简介
- Mybatis核心流程
- Mybatis核心类
- Mybatis字段映射应用及源码
- Mybatis数据转换应用及源码

存在的问题

1. SQL和代码混合在一起。
2. 需要自己处理调用数据库相关的事情
3. 需要自己处理字段映射
4. 需要自己处理数据类型转换
5. 容错率低。

JDBC时代我们是怎么开发的?

```
public class CityPO {  
    Integer id;  
  
    Long cityId;  
  
    String cityName;  
}
```

1. 编写PO
2. 编写连接数据库的工具类
3. 调用数据库
4. 遍历Result,通过反射,获取对应的值

```
public class Main {  
  
    public static void main(String[] args) throws Exception {  
        Connection connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/demo?useSSL=false", "root", "123456");  
  
        PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM SU_City limit 10");  
  
        List<CityPO> cityPOS = convertRS(preparedStatement.executeQuery(), CityPO.class);  
        System.out.println(JSON.toJSONString(cityPOS));  
    }  
  
    public static <T> List<T> convertRS(ResultSet rs, Class<T> cls) throws Exception {...}  
}
```

Mybatis时代的开发流程

```
public class CityPO {  
    Integer id;  
  
    Long cityId;  
  
    String cityName;  
}
```

1. 编写PO
2. 编写SQL映射文件
3. 编写Mapper接口

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >  
<mapper namespace="po.CityMapper">  
  
    <resultMap id="cityPOMap" type="po.CityPO">  
        <result column="id" property="id"/>  
        <result column="city_id" property="cityId"/>  
        <result column="city_name" property="cityName"/>  
    </resultMap>  
  
    <select id="queryCityPOs" resultType="po.CityPO">  
        SELECT * FROM SU_City limit 10;  
    </select>  
  
</mapper>
```

```
public interface CityMapper {  
    public List<CityPO> queryCityPOs();  
}
```

Mybatis时代的开发流程

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <settings>
    <setting name="localCacheScope" value="STATEMENT"/>
    <setting name="cacheEnabled" value="false" />
    <setting name="defaultStatementTimeout" value="1" />
    <!-- 开启驼峰, 开启后, 只要数据库字段和对象属性名字母相同, 无论中间加多少下划线都可以识别 -->
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>
  <!--数据库配置 -->
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC">
      </transactionManager>
      <dataSource type="POOLED">
        <property name="driver" value="${db.driver}" />
        <property name="url" value="${db.url}" />
        <property name="username" value="${db.username}" />
        <property name="password" value="${db.password}" />
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="mappers/CityMapper.xml" />
  </mappers>
</configuration>
```

4. 编写Mybatis配置文件

Mybatis时代的开发流程

5. 初始化配置,愉快的使用吧

```
public static void main(String[] args) throws Exception {
    SqlSessionFactory sqlSessionFactory = buildSqlSessionFactory();

    SqlSession sqlSession = sqlSessionFactory.openSession(b: true);
    CityMapper cityMapper = sqlSession.getMapper(CityMapper.class);

    System.out.println(JSON.toJSONString(cityMapper.queryCityPOs()));
}

private static SqlSessionFactory buildSqlSessionFactory() throws Exception {
    ResourceBundle bundle = ResourceBundle.getBundle("jdbc");
    Properties properties = new Properties();
    properties.setProperty("db.url", bundle.getString(key: "db.url"));
    properties.setProperty("db.username", bundle.getString(key: "db.username"));
    properties.setProperty("db.password", bundle.getString(key: "db.password"));
    properties.setProperty("db.driver", bundle.getString(key: "db.driver"));
    return new SqlSessionFactoryBuilder().build(Resources.getResourceAsReader("mybatis-config.xml"), properties);
}
```


Mybatis核心概念

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。

Mybatis核心类

- Configuration : Mybatis框架运行过程中需要的信息的集合地。
- MappedStatement: Mybatis框架抽象代表Sql语句的类。
- SqlSessionFactory : 根据Configuration中信息创建SqlSession的工厂类
- SqlSession : Mybatis 框架向用户提供的操作数据库的API,持有了数据库连接。
- Executor: Mybatis框架内部负责完成操作数据库的类。
- StatementHandler : 操作JDBC的statement操作。
- ResultSetHandler: 操作JDBC的结果集。

Mybatis核心流程

JDBC一次操作数据库的流程:

Connection -> Statement(Sql) -> ResultSet

Mybatis一次操作数据库的流程:

Sqlsession -> Executor(MappedStatement) -> StatementHandler -> ResultSetHandler

Mybatis字段映射应用

```
mysql> mysql> desc SU_City;
```

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
city_id	int(11)	NO	UNI	NULL	
city_name	varchar(20)	NO			
city_en_name	varchar(20)	NO			
city_py_name	varchar(50)	NO			
create_time	datetime	NO		CURRENT_TIMESTAMP	
updatetime	datetime	NO	MUL	CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP

7 rows in set (0.01 sec)

```
public class CityPO {  
    Integer id;  
  
    Long cityId;  
  
    String cityName;
```



Mybatis字段映射方案

1. 驼峰式命名, 比如数据库列名 user_id, 对象字段名userId。
2. 使用AS。比如 select user_id as userId, 对应对象字段名userId。
3. 使用resultMap, 如下。

```
<resultMap id="cityPOMap" type="po.CityPO">  
  <result column="id" property="id"/>  
  <result column="city_id" property="cityId"/>  
  <result column="city_name" property="cityName"/>  
</resultMap>
```

Mybatis字段映射方案

1. 驼峰式命名, 比如数据库列名 user_id,对象字段名userId。
2. 使用AS。 比如 select user_id as userId, 对应对象字段名userId。
3. 使用resultMap, 如下。

```
<resultMap id="cityPOMap" type="po.CityPO">  
  <result column="id" property="id"/>  
  <result column="city_id" property="cityId"/>  
  <result column="city_name"property="cityName"/>  
</resultMap>
```

Mybatis驼峰式命名

```
<setting name="mapUnderscoreToCamelCase" value="true" />
```

Mybatis对数据库的操作都是在Executor的子类中完成，Select的操作在于doQuery()方法中。

```
@Override
public <E> List<E> doQuery(MappedStatement ms, Object parameter, RowBounds rowBounds, ResultHandler resultHandler, BoundSql boundSql) throws SQLException {
    Statement stmt = null;
    try {
        Configuration configuration = ms.getConfiguration();
        StatementHandler handler = configuration.newStatementHandler(wrapper, ms, parameter, rowBounds, resultHandler, boundSql);
        stmt = prepareStatement(handler, ms.getStatementLog());
        return handler.<E>query(stmt, resultHandler);
    } finally {
        closeStatement(stmt);
    }
}
```

切换进入IDEA讲解。

Mybatis中使用AS

ResultSetWrapper完成了具体使用哪个名字作为数据库列名的工作。

```
public ResultSetWrapper(ResultSet rs, Configuration configuration) throws SQLException {
    super();
    this.typeHandlerRegistry = configuration.getTypeHandlerRegistry();
    this.resultSet = rs;
    final ResultSetMetaData metaData = rs.getMetaData();
    final int columnCount = metaData.getColumnCount();
    for (int i = 1; i <= columnCount; i++) {
        columnNames.add(configuration.isUseColumnLabel() ? metaData.getColumnLabel(i) : metaData.getColumnName(i));
        jdbcTypes.add(JdbcType.forCode(metaData.getColumnType(i)));
        classNames.add(metaData.getColumnClassName(i));
    }
}
```


Mybatis中使用AS

```
/**
 * Gets the designated column's suggested title for use in printouts and
 * displays. The suggested title is usually specified by the SQL AS
 * clause. If a SQL AS is not specified, the value returned from
 * getColumnLabel will be the same as the value returned by the
 * getColumnName method.
 *
 * @param column the first column is 1, the second is 2, ...
 * @return the suggested column title
 * @exception SQLException if a database access error occurs
 */
String getColumnLabel(int column) throws SQLException;
```

Mybatis使用ResultMap

```
List<ResultMap> resultMap = mappedStatement.getResultMaps();
```

```
▼ resultMap = {Collections$UnmodifiableRandomAccessList@2283} size = 1
  ▼ 0 = {ResultMap@2439}
    ▶ configuration = {Configuration@1796}
    ▶ id = "mapper.CityMapper.cityMap"
    ▶ type = {Class@1422} "class po.CityPO" ... Navigate
    ▼ resultMappings = {Collections$UnmodifiableRandomAccessList@2441} size = 4
      ▶ 0 = {ResultMapping@2449} "ResultMapping{property='id', column='id', javaType=class java.lang.Integer, jdbcType=null, nestedResultMapId='null', nestedQueryId='null', notNullColor... Vie
      ▶ 1 = {ResultMapping@2450} "ResultMapping{property='cityId', column='city_id', javaType=class java.lang.Long, jdbcType=null, nestedResultMapId='null', nestedQueryId='null', notNu ... Vie
      ▶ 2 = {ResultMapping@2451} "ResultMapping{property='cityName', column='city_name', javaType=class java.lang.String, jdbcType=null, nestedResultMapId='null', nestedQueryId='null... Vie
      ▶ 3 = {ResultMapping@2452} "ResultMapping{property='cityEnglishName', column='city_en_name', javaType=class java.lang.String, jdbcType=null, nestedResultMapId='null', nestedQi... Vie
    ▼ idResultMappings = {Collections$UnmodifiableRandomAccessList@2442} size = 4
      ▶ 0 = {ResultMapping@2449} "ResultMapping{property='id', column='id', javaType=class java.lang.Integer, jdbcType=null, nestedResultMapId='null', nestedQueryId='null', notNullColor... Vie
      ▶ 1 = {ResultMapping@2450} "ResultMapping{property='cityId', column='city_id', javaType=class java.lang.Long, jdbcType=null, nestedResultMapId='null', nestedQueryId='null', notNu ... Vie
      ▶ 2 = {ResultMapping@2451} "ResultMapping{property='cityName', column='city_name', javaType=class java.lang.String, jdbcType=null, nestedResultMapId='null', nestedQueryId='null... Vie
      ▶ 3 = {ResultMapping@2452} "ResultMapping{property='cityEnglishName', column='city_en_name', javaType=class java.lang.String, jdbcType=null, nestedResultMapId='null', nestedQi... Vie
    ▶ constructorResultMappings = {Collections$UnmodifiableRandomAccessList@2443} size = 0
    ▶ propertyResultMappings = {Collections$UnmodifiableRandomAccessList@2444} size = 4
    ▼ mappedColumns = {Collections$UnmodifiableSet@2445} size = 4
      ▶ 0 = "CITY_ID"
      ▶ 1 = "CITY_NAME"
      ▶ 2 = "ID"
      ▶ 3 = "CITY_EN_NAME"
    ▼ mappedProperties = {HashSet@2446} size = 4
      ▶ 0 = "id"
      ▶ 1 = "cityId"
      ▶ 2 = "cityName"
      ▶ 3 = "cityEnglishName"
```

切换画面至IDEA

Mybatis使用ResultMap

进入propertyMapping阶段

```
▼ p propertyMapping = {ResultMapping@2207} "ResultMapping{property='cityId', column='city_id', javaType=class java ... View
  ▶ f configuration = {Configuration@1806}
  ▶ f property = "cityId"
  ▶ f column = "city_id"
  ▶ f javaType = {Class@210} "class java.lang.Long" ... Navigate
  ▶ f jdbcType = null
  ▶ f typeHandler = {LongTypeHandler@2217} "class java.lang.Long"
  ▶ f nestedResultMap = null
```

切换画面至IDEA

Mybatis 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。

类型处理器	Java 类型	JDBC 类型
BooleanTypeHandler	java.lang.Boolean, boolean	数据库兼容的 BOOLEAN
ByteTypeHandler	java.lang.Byte, byte	数据库兼容的 NUMERIC 或 BYTE
ShortTypeHandler	java.lang.Short, short	数据库兼容的 NUMERIC 或 SHORT INTEGER
IntegerTypeHandler	java.lang.Integer, int	数据库兼容的 NUMERIC 或 INTEGER
LongTypeHandler	java.lang.Long, long	数据库兼容的 NUMERIC 或 LONG INTEGER
FloatTypeHandler	java.lang.Float, float	数据库兼容的 NUMERIC 或 FLOAT
DoubleTypeHandler	java.lang.Double, double	数据库兼容的 NUMERIC 或 DOUBLE
BigDecimalTypeHandler	java.math.BigDecimal	数据库兼容的 NUMERIC 或 DECIMAL
StringTypeHandler	java.lang.String	CHAR, VARCHAR
ClobReaderTypeHandler	java.io.Reader	-
ClobTypeHandler	java.lang.String	CLOB, LONGVARCHAR
NStringTypeHandler	java.lang.String	NVARCHAR, NCHAR
NClobTypeHandler	java.lang.String	NCLOB
BlobInputStreamTypeHandler	java.io.InputStream	-
ByteArrayTypeHandler	byte[]	数据库兼容的字节流类型

Mybatis TypeHandler

```
public class BigDecimalTypeHandler extends BaseTypeHandler<BigDecimal> {

    @Override
    public void setNonNullParameter(PreparedStatement ps, int i, BigDecimal parameter, JdbcType jdbcType)
        throws SQLException {
        ps.setBigDecimal(i, parameter);
    }

    @Override
    public BigDecimal getNullableResult(ResultSet rs, String columnName)
        throws SQLException {
        return rs.getBigDecimal(columnName);
    }

    @Override
    public BigDecimal getNullableResult(ResultSet rs, int columnIndex)
        throws SQLException {
        return rs.getBigDecimal(columnIndex);
    }

    @Override
    public BigDecimal getNullableResult(CallableStatement cs, int columnIndex)
        throws SQLException {
        return cs.getBigDecimal(columnIndex);
    }
}
```

Mybatis TypeHandler

```
public class BigDecimalTypeHandler extends BaseTypeHandler<BigDecimal> {  
  
    @Override  
    public void setNonNullParameter(PreparedStatement ps, int i, BigDecimal parameter, JdbcType jdbcType)  
        throws SQLException {  
        ps.setBigDecimal(i, parameter);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(ResultSet rs, String columnName)  
        throws SQLException {  
        return rs.getBigDecimal(columnName);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(ResultSet rs, int columnIndex)  
        throws SQLException {  
        return rs.getBigDecimal(columnIndex);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(CallableStatement cs, int columnIndex)  
        throws SQLException {  
        return cs.getBigDecimal(columnIndex);  
    }  
}
```

切换画面进入IDEA

Mybatis TypeHandler

```
public class BigDecimalTypeHandler extends BaseTypeHandler<BigDecimal> {  
  
    @Override  
    public void setNonNullParameter(PreparedStatement ps, int i, BigDecimal parameter, JdbcType jdbcType)  
        throws SQLException {  
        ps.setBigDecimal(i, parameter);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(ResultSet rs, String columnName)  
        throws SQLException {  
        return rs.getBigDecimal(columnName);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(ResultSet rs, int columnIndex)  
        throws SQLException {  
        return rs.getBigDecimal(columnIndex);  
    }  
  
    @Override  
    public BigDecimal getNullableResult(CallableStatement cs, int columnIndex)  
        throws SQLException {  
        return cs.getBigDecimal(columnIndex);  
    }  
}
```

切换画面进入IDEA

Mybatis TypeHandler

场景: 从SqlServer中取数据, 遇到很多列都是Numeric(10,2)类型, 指的是字段是数字型, 长度为10, 小数为两位。Mybatis默认的BigDecimalTypeHandler取到后, 都默认变成4位小数, 不够的补了0。而上层的要求是, 拿到的和数字相关的数据都要2位小数。

1. 在所有给上层赋值的时候, 都人工对BigDecimal的数据做如下操作。

```
setScale(2, BigDecimal.ROUND_HALF_UP)
```

2. 继承BigDecimalTypeHandler, 覆盖原来的取值方法, 对取到的数值做范围限定。

Mybatis TypeHandler

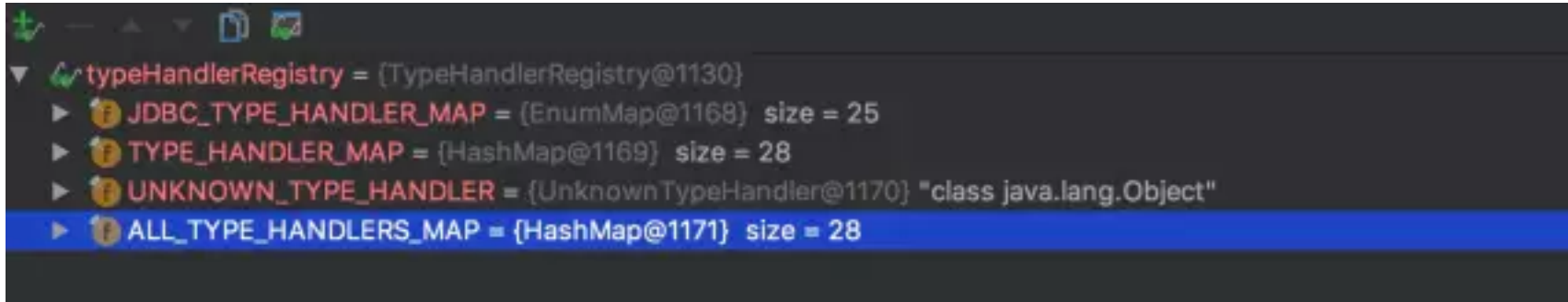
```
/**
 * 统一处理 Mybatis返回的BigDecimal的尾数范围
 *
 * @author cenkailun
 * @Date 17/5/31
 * @Time 下午2:13
 */
@MappedJdbcTypes(JdbcType.NUMERIC)
public class SubBigDecimalTypeHandler extends BigDecimalTypeHandler {
    @Override
    public BigDecimal getNullableResult(ResultSet rs, String columnName) throws SQLException {
        return super.getNullableResult(rs, columnName).setScale(newScale: 2, BigDecimal.ROUND_HALF_UP);
    }

    @Override
    public BigDecimal getNullableResult(ResultSet rs, int columnIndex) throws SQLException {
        return super.getNullableResult(rs, columnIndex).setScale(newScale: 2, BigDecimal.ROUND_HALF_UP);
    }

    @Override
    public BigDecimal getNullableResult(CallableStatement cs, int columnIndex) throws SQLException {
        return super.getNullableResult(cs, columnIndex).setScale(newScale: 2, BigDecimal.ROUND_HALF_UP);
    }
}
```

Mybatis TypeHandler

TypeHandlerRegistry



```
▼ typeHandlerRegistry = {TypeHandlerRegistry@1130}
  ▶ JDBC_TYPE_HANDLER_MAP = {EnumMap@1168} size = 25
  ▶ TYPE_HANDLER_MAP = {HashMap@1169} size = 28
  ▶ UNKNOWN_TYPE_HANDLER = {UnknownTypeHandler@1170} "class java.lang.Object"
  ▶ ALL_TYPE_HANDLERS_MAP = {HashMap@1171} size = 28
```

The screenshot shows a debugger window with a tree view of a `TypeHandlerRegistry` object. The root node is `typeHandlerRegistry = {TypeHandlerRegistry@1130}`. It has four children: `JDBC_TYPE_HANDLER_MAP = {EnumMap@1168} size = 25`, `TYPE_HANDLER_MAP = {HashMap@1169} size = 28`, `UNKNOWN_TYPE_HANDLER = {UnknownTypeHandler@1170} "class java.lang.Object"`, and `ALL_TYPE_HANDLERS_MAP = {HashMap@1171} size = 28`. The last item is highlighted with a blue background.

Mybatis TypeHandler

JDBC_TYPE_HANDLER_MAP中记录的是JdbcType和TypeHandler对应的关系。

```
▼ typeHandlerRegistry = {TypeHandlerRegistry@1130}
  ▼ JDBC_TYPE_HANDLER_MAP = {EnumMap@1168} size = 25
    ▶ 0 = {AbstractMap$SimpleEntry@1177} "ARRAY" -> "class java.lang.Object"
    ▶ 1 = {AbstractMap$SimpleEntry@1178} "BIT" -> "class java.lang.Boolean"
    ▼ 2 = {AbstractMap$SimpleEntry@1179} "TINYINT" -> "class java.lang.Byte"
      ▶ key = {JdbcType@1613} "TINYINT"
      ▼ value = {ByteTypeHandler@1614} "class java.lang.Byte"
        ⓘ configuration = null
        ▶ rawType = {Class@281} "class java.lang.Byte" ... Navigate
```

Mybatis TypeHandler

TYPE_HANDLER_MAP中记录的是Java类型和对应的所有JdbcType以及其对应TypeHandler的映射关系。

```
▼ TYPE_HANDLER_MAP = {HashMap@1169} size = 28
  ▶ 0 = {HashMap$Entry@1301} "class [B" -> " size = 3"
  ▶ 1 = {HashMap$Entry@1302} "short" -> " size = 1"
  ▼ 2 = {HashMap$Entry@1303} "class java.lang.String" -> " size = 8"
    ▶ key = {Class@145} "class java.lang.String" ... Navigate
    ▼ value = {HashMap@1332} size = 8
      ▼ 0 = {HashMap$Entry@1518} "NCHAR" -> "class java.lang.String"
        ▶ key = {JdbcType@1618} "NCHAR"
        ▶ value = {NStringTypeHandler@1619} "class java.lang.String"
      ▶ 1 = {HashMap$Entry@1519} "CLOB" -> "class java.lang.String"
      ▼ 2 = {HashMap$Entry@1520} "null" -> "class java.lang.String"
        ▶ key = null
        ▶ value = {StringTypeHandler@1606} "class java.lang.String"
      ▶ 3 = {HashMap$Entry@1521} "LONGVARCHAR" -> "class java.lang.String"
      ▶ 4 = {HashMap$Entry@1522} "NVARCHAR" -> "class java.lang.String"
      ▶ 5 = {HashMap$Entry@1523} "VARCHAR" -> "class java.lang.String"
      ▶ 6 = {HashMap$Entry@1524} "NCLOB" -> "class java.lang.String"
      ▶ 7 = {HashMap$Entry@1525} "CHAR" -> "class java.lang.String"
```

Mybatis TypeHandler

```
1 register(String.class, new StringTypeHandler());
2 register(String.class, JdbcType.NCHAR, new NStringTypeHandler());
3 register(JdbcType.NCHAR, new NStringTypeHandler());
```

切换进入IDEA

```
▼ TYPE_HANDLER_MAP = {ConcurrentHashMap@822} size = 16
  ▶ 0 = {ConcurrentHashMap$WriteThroughEntry@913} "class java.lang.Boolean" -> " size = 1"
  ▶ 1 = {ConcurrentHashMap$WriteThroughEntry@914} "long" -> " size = 1"
  ▶ 2 = {ConcurrentHashMap$WriteThroughEntry@915} "byte" -> " size = 1"
  ▼ 3 = {ConcurrentHashMap$WriteThroughEntry@916} "class java.lang.String" -> " size = 1"
    ▶ key = {Class@142} "class java.lang.String" ... Navigate
    ▼ value = {HashMap@854} size = 1
      ▼ 0 = {HashMap$Entry@970} "null" -> "class java.lang.String"
        key = null
        ▶ value = {StringTypeHandler@826} "class java.lang.String"
```

Mybatis TypeHandler

```
1 register(String.class, new StringTypeHandler());
2 register(String.class, JdbcType.NCHAR, new NStringTypeHandler());
3 register(JdbcType.NCHAR, new NStringTypeHandler());
```

切换进入IDEA

```
▼ TYPE_HANDLER_MAP = {ConcurrentHashMap@822} size = 16
  ▶ 0 = {ConcurrentHashMap$WriteThroughEntry@1013} "class java.lang.Boolean" -> " size = 1"
  ▶ 1 = {ConcurrentHashMap$WriteThroughEntry@1014} "long" -> " size = 1"
  ▶ 2 = {ConcurrentHashMap$WriteThroughEntry@1015} "byte" -> " size = 1"
  ▼ 3 = {ConcurrentHashMap$WriteThroughEntry@1016} "class java.lang.String" -> " size = 6"
    ▶ key = {Class@142} "class java.lang.String" ... Navigate
    ▼ value = {HashMap@854} size = 6
      ▶ 0 = {HashMap$Entry@970} "null" -> "class java.lang.String"
      ▶ 1 = {HashMap$Entry@1067} "LONGVARCHAR" -> "class java.lang.String"
      ▶ 2 = {HashMap$Entry@1068} "VARCHAR" -> "class java.lang.String"
      ▶ 3 = {HashMap$Entry@1069} "CHAR" -> "class java.lang.String"
      ▶ 4 = {HashMap$Entry@1070} "NVARCHAR" -> "class java.lang.String"
      ▶ 5 = {HashMap$Entry@1071} "CLOB" -> "class java.lang.String"
```

Mybatis TypeHandler

```
1 register(String.class, new StringTypeHandler());
2 register(String.class, JdbcType.NCHAR, new NStringTypeHandler());
3 register(JdbcType.NCHAR, new NStringTypeHandler());
```


切换进入IDEA

```
▼ JDBC_TYPE_HANDLER_MAP = {EnumMap@821} size = 13
  ▶ 0 = {AbstractMap$SimpleEntry@1160} "BIT" -> "class java.lang.Boolean"
  ▶ 1 = {AbstractMap$SimpleEntry@1161} "TINYINT" -> "class java.lang.Byte"
  ▶ 2 = {AbstractMap$SimpleEntry@1162} "SMALLINT" -> "class java.lang.Short"
  ▶ 3 = {AbstractMap$SimpleEntry@1163} "INTEGER" -> "class java.lang.Integer"
  ▶ 4 = {AbstractMap$SimpleEntry@1164} "FLOAT" -> "class java.lang.Float"
  ▶ 5 = {AbstractMap$SimpleEntry@1165} "DOUBLE" -> "class java.lang.Double"
  ▶ 6 = {AbstractMap$SimpleEntry@1166} "CHAR" -> "class java.lang.String"
  ▶ 7 = {AbstractMap$SimpleEntry@1167} "VARCHAR" -> "class java.lang.String"
  ▶ 8 = {AbstractMap$SimpleEntry@1168} "LONGVARCHAR" -> "class java.lang.String"
  ▶ 9 = {AbstractMap$SimpleEntry@1169} "CLOB" -> "class java.lang.String"
  ▶ 10 = {AbstractMap$SimpleEntry@1170} "BOOLEAN" -> "class java.lang.Boolean"
  ▶ 11 = {AbstractMap$SimpleEntry@1171} "NVARCHAR" -> "class java.lang.String"
  ▼ 12 = {AbstractMap$SimpleEntry@1172} "NCHAR" -> "class java.lang.String"
    ▶ key = {JdbcType@1102} "NCHAR"
    ▶ value = {NStringTypeHandler@1098} "class java.lang.String"
```

Mybatis TypeHandler

自定义的TypeHandler是如何被注册进去的。

```
private void parseConfiguration(XNode root) {
    try {
        Properties settings = settingsAsProperties(root.evalNode("settings"));
        //issue #117 read properties first
        propertiesElement(root.evalNode("properties"));
        loadCustomVfs(settings);
        typeAliasesElement(root.evalNode("typeAliases"));
        pluginElement(root.evalNode("plugins"));
        objectFactoryElement(root.evalNode("objectFactory"));
        objectWrapperFactoryElement(root.evalNode("objectWrapperFactory"));
        reflectorFactoryElement(root.evalNode("reflectorFactory"));
        settingsElement(settings);
        // read it after objectFactory and objectWrapperFactory issue #631
        environmentsElement(root.evalNode("environments"));
        databaseIdProviderElement(root.evalNode("databaseIdProvider"));
        typeHandlerElement(root.evalNode("typeHandlers"));
        mapperElement(root.evalNode("mappers"));
    } catch (Exception e) {
        throw new BuilderException("Error parsing SQL Mapper Configuration. Cause: " + e, e);
    }
}
```



Mybatis TypeHandler

Mybatis如何选择合适的TypeHandler

```
private Object getRowValue(ResultSetWrapper rsw, ResultMap resultMap) throws SQLException {  
    final ResultLoaderMap lazyLoader = new ResultLoaderMap();  
    Object resultObject = createResultObject(rsw, resultMap, lazyLoader, columnPrefix: null);  
    if (resultObject != null && !hasTypeHandlerForResultObject(rsw, resultMap.getType())) {  
        final MetaObject metaObject = configuration.newMetaObject(resultObject);  
        boolean foundValues = !resultMap.getConstructorResultMappings().isEmpty();  
        if (shouldApplyAutomaticMappings(resultMap, isNested: false)) {  
            foundValues = applyAutomaticMappings(rsw, resultMap, metaObject, columnPrefix: null) || foundValues;  
        }  
        foundValues = applyPropertyMappings(rsw, resultMap, metaObject, lazyLoader, columnPrefix: null) || foundValues;  
        foundValues = lazyLoader.size() > 0 || foundValues;  
        resultObject = foundValues ? resultObject : null;  
        return resultObject;  
    }  
    return resultObject;  
}
```

1

2

切入IDEA

Mybatis TypeHandler

```
▼ jdbchandlerMap = {HashMap@1799} size = 2
  ▼ 0 = {HashMap$Entry@1806} "null" -> "class java.math.BigDecimal"
    key = null
    value = {BigDecimalTypeHandler@1808} "class java.math.BigDecimal"
  ▼ 1 = {HashMap$Entry@1807} "NUMERIC" -> "class java.math.BigDecimal"
    key = {JdbcType@1790} "NUMERIC"
    value = {SubBigDecimalTypeHandler@1809} "class java.math.BigDecimal"
  ▼ jdbcType = {JdbcType@1790} "NUMERIC"
    TYPE_CODE = 2
    name = "NUMERIC"
    ordinal = 9
```

Q&A



公众号: KailunTalk