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

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?

```java
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

```java
public class Main {

    public static void main(String[] args) throws Exception {
        Connection connection = DriverManager.getConnection( url: "jdbc:mysql://127.0.0.1:3306/demo?useSSL=false", user: "root", password: "123456");

        PreparedStatement preparedStatement = connection.prepareStatement( sql: "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

```java
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
<?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>
```

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

Mybatis时代的开发流程

```xml
<?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resultset。

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)
```

```java
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Execturor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

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

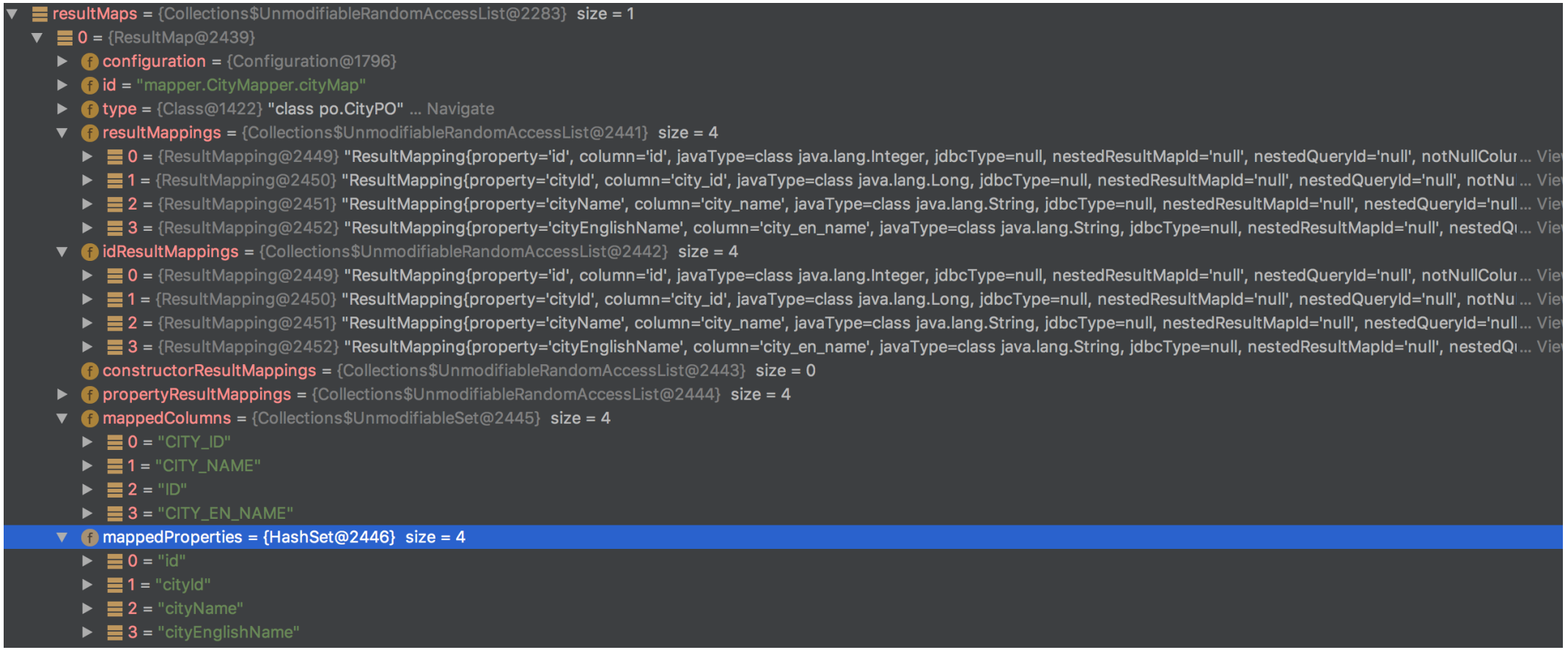
```
public ResultSetWrapper(ResultSet rs, Configuration configuration) throws SQLException {
  super();
  this.typeHandlerRegistry = configuration.getTypeHandlerRegistry();
  this.resultSet = rs;
  final ResultSetMetaData metaData = rs.getMetaData();
  final int columnCount = metaData.getColumnCount();
  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 <code>AS</code>
 * clause.  If a SQL <code>AS</code> is not specified, the value returned from
 * <code>getColumnLabel</code> will be the same as the value returned by the
 * <code>getColumnName</code> 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

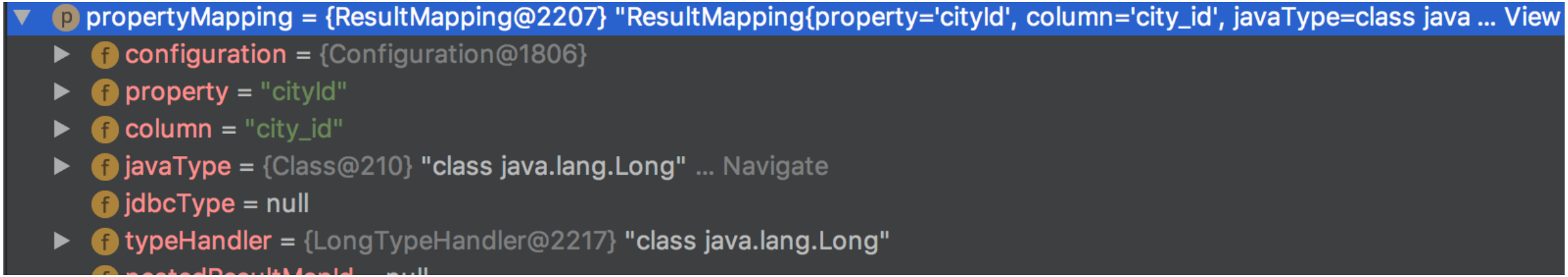
List<ResultMap> resultMaps = mappedStatement.getResultMaps();



切换画面至IDEA

Mybatis使用ResultMap

进入propertyMapping阶段



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

```java
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

```java
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

```java
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

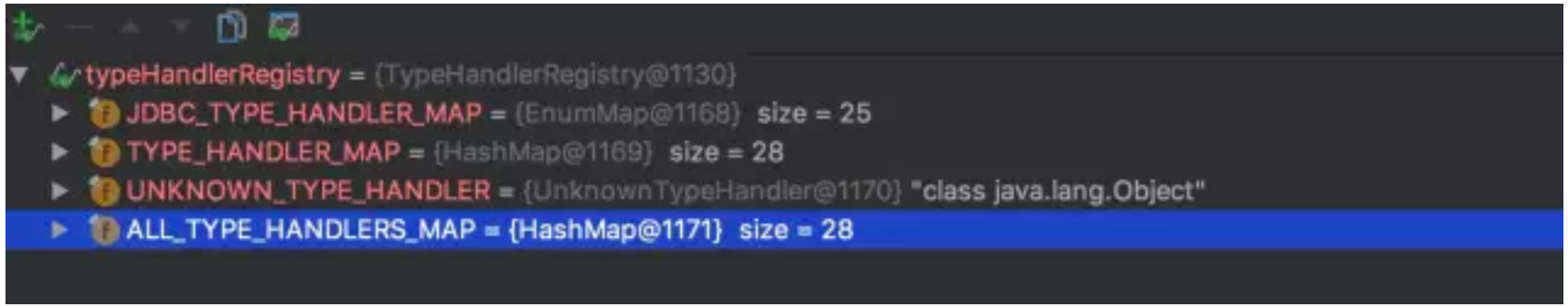
```java
/**
 * 统一处理 Mybatis返回的BigDecimal的尾数范围
 *
 * @author cenkailun
 * @Date 17/5/31
 * @Time 下午2:13
 */
@MappedJdbcTypes(JdbcType.NUMERIC)
public class SubBigDecimalTypeHandler extends BigDecimalTypeHandler {
    @Override
    public BigDecimal getNullableResult(ResultSet rs, String columnName) throws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

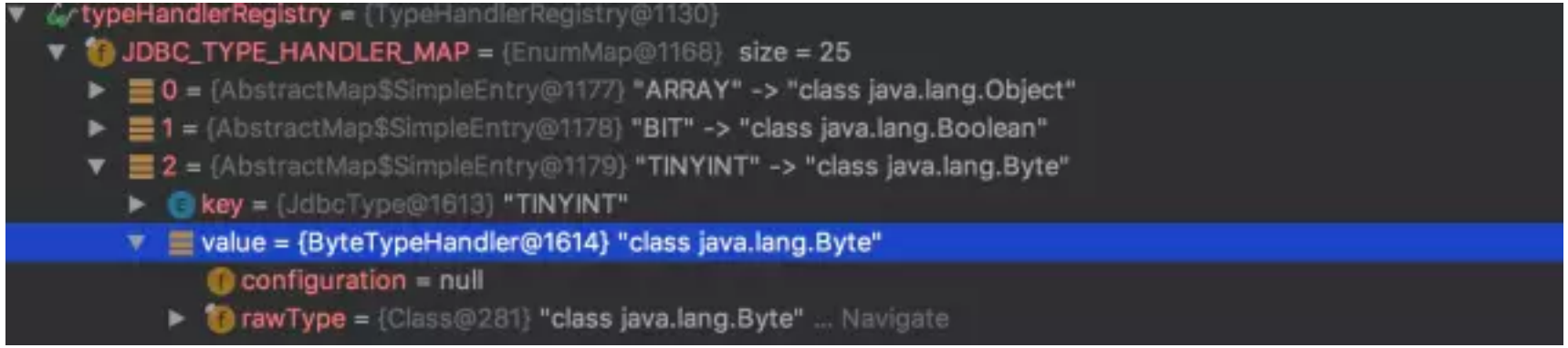
Mybatis TypeHandler

TypeHandlerRegistry



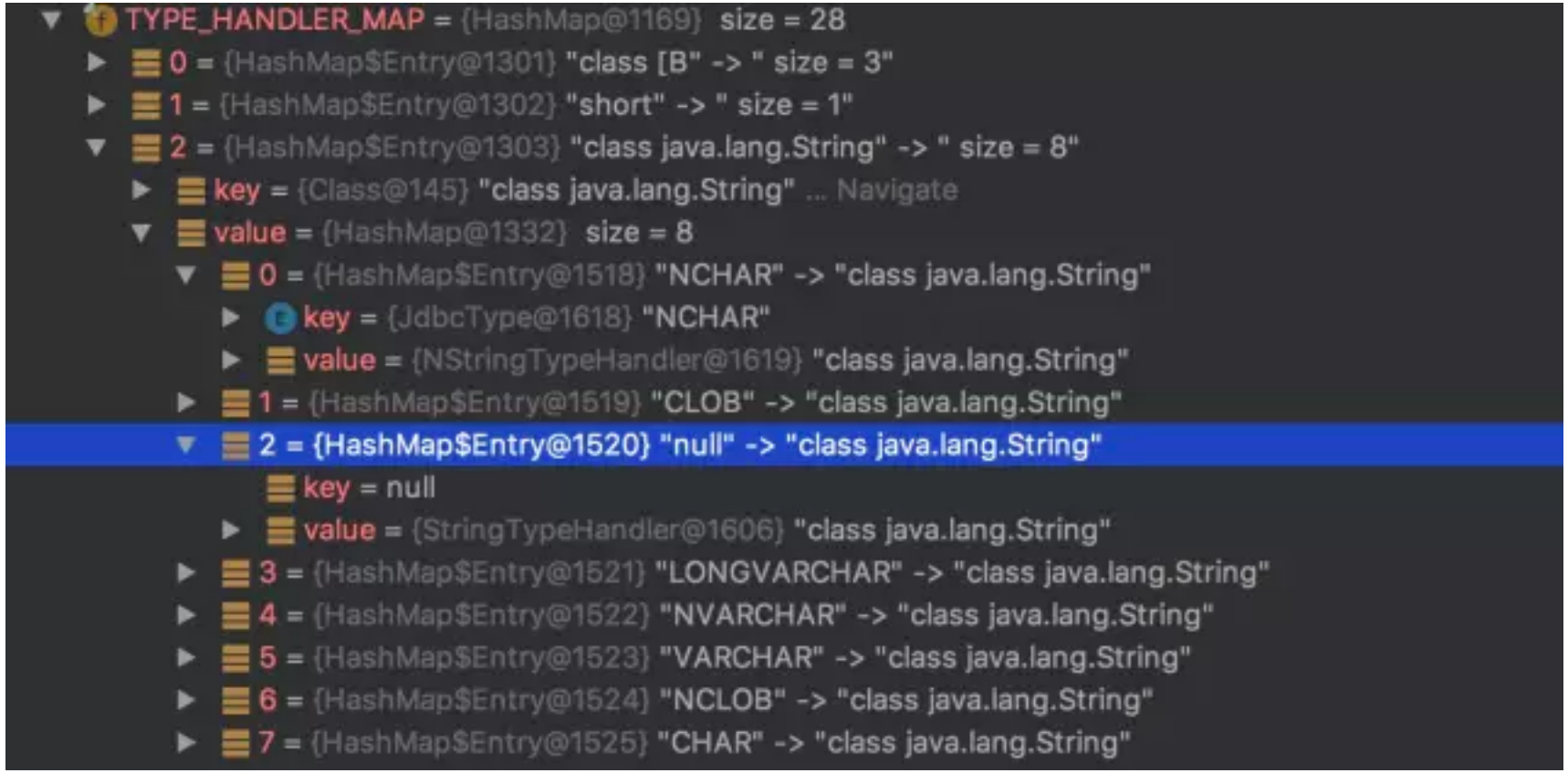
Mybatis TypeHandler

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



Mybatis TypeHandler

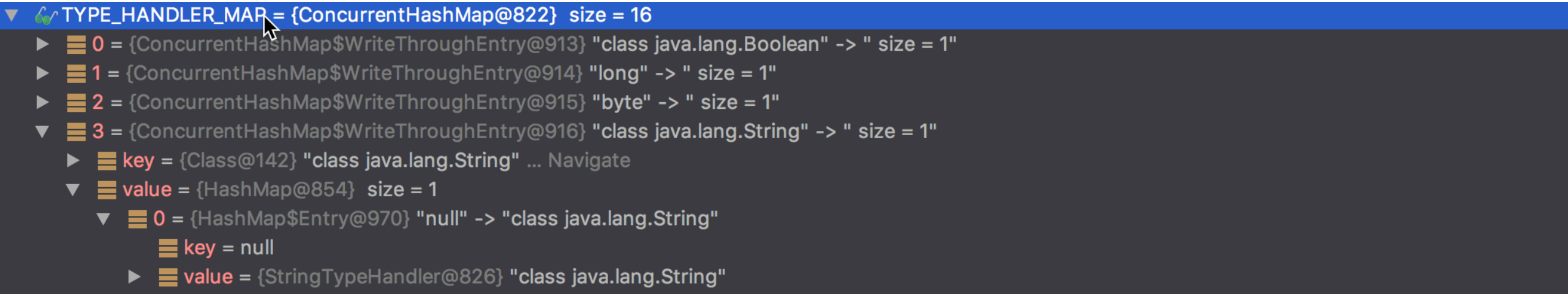
TYPE_HANDLER_MAP中记录的是Java类型和对应的所有JdbcType以及其对应TypeHandler的映射关系关系。



Mybatis TypeHandler

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

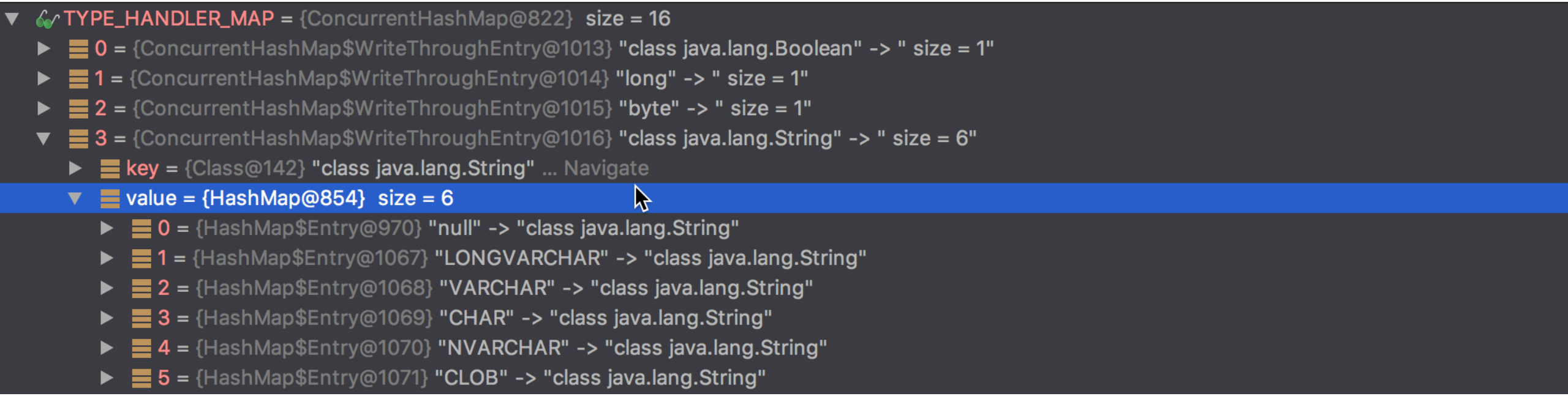
切换进入IDEA



Mybatis TypeHandler

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

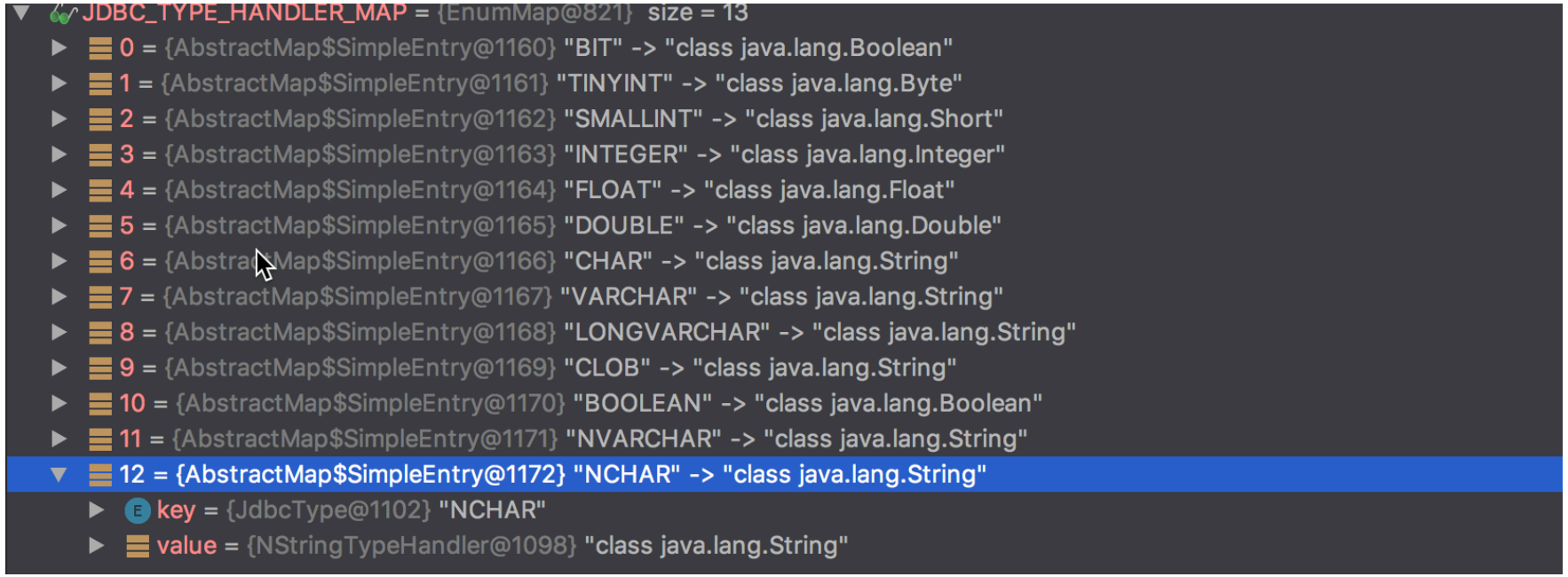
切换进入IDEA



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



Mybatis TypeHandler

自定义的TypeHandler是如何被注册进去的。

```
private void parseConfiguration(XNode root) {
  try {
    Properties settings = settingsAsPropertiess(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);
  }
}
```

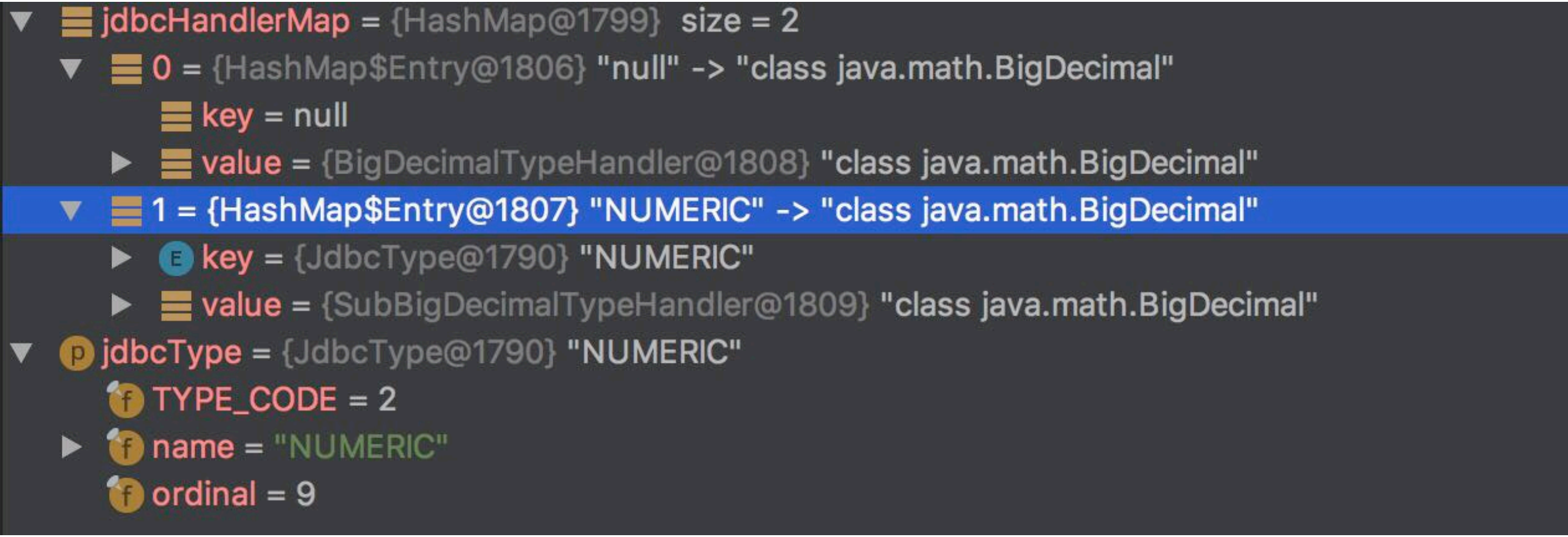
Mybatis TypeHandler

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;
}
```

切入IDEA

Mybatis TypeHandler



Q&A

公众号: KailunTalk