

# 手写mybatis-3

---

## 2.5 执行结果处理

---

### 2.5.1 执行结果处理要干的是什么事

pst.executeUpdate()的返回结果是int，影响的行数。

pst.executeQuery()的返回结果是ResultSet。

在得到语句执行的结果后，要转为方法的返回结果进行返回。这就是执行结果处理要干的事  
根据方法的返回值类型来进行相应的处理。

这里我们根据语句执行结果的不同，分开处理：

```
@Override
public Object invoke(Object proxy, Method method, Object[] args) throws
    Throwable {
    // TODO 这里需要完成哪些事？
    // 1、获得方法对应的SQL语句
    String id = this.mapper.getName() + "." + method.getName();
    MappedStatement ms = this.configuration.getMappedStatement(id);
    // 2、解析SQL参数与方法参数的对应关系，得到真正的SQL与语句参数值
    RealSqlAndParamValues rsp = ms.getRealSqlAndParamValues(args);
    // 3、获得数据库连接
    Connection conn =
this.configuration.getDataSource().getConnection();
    // 4、创建语句对象。
    PreparedStatement pst = conn.prepareStatement(rsp.getSql());
    // 5、设置语句参数
    int i = 1;
    for (ParamValue p : rsp.getParamValues()) {
        TypeHandler th = p.getTypeHandler()
        th.setParameter(pst, i++, p.getValue());
    }
}
```

```

// 6、执行语句并处理结果
switch (ms.getSqlCommandType()) {
    case INSERT:
    case UPDATE:
    case DELETE:
        int rows = pst.executeUpdate();
        return handleUpdateReturn(rows, ms, method);
    case SELECT:
        ResultSet rs = pst.executeQuery();
        return handleResultSetReturn(rs, ms, method);
}
return null;
}

private Object handleUpdateReturn(int rows, MappedStatement ms, Method
method) {
    // TODO Auto-generated method stub
    return null;
}

private Object handleResultSetReturn(ResultSet rs, MappedStatement ms,
Method method) {
    // TODO Auto-generated method stub
    return null;
}

```

## 2.5.2 pst.executeUpdate()的返回结果处理。

pst.executeUpdate()的返回结果是int

方法的返回值可以是什么？

void、int、long、其他的不可以！

```

private Object handleUpdateReturn(int rows, MappedStatement ms, Method
method) {

    Class<?> returnType = method.getReturnType();
    if (returnType == void.TYPE) {
        return null;
    } else if (returnType == int.class || returnType == Integer.class)
    {
        return rows;
    } else if (returnType == long.class || returnType == Long.class) {
        return (long) rows;
    }

    throw new IllegalArgumentException("update类方法的返回值只能是 :
void/int/Integer/long/Long");
}

```

## 2.5.3 pst.executeQuery()的返回结果处理

pst.executeQuery()的返回结果是ResultSet

方法的返回值可以是什么？

可以是void、单个值、集合。

单个值可以是什么类型的值？

任意值、 ( map)

```

@Select("select count(1) from t_user where sex = #{sex}")
int query(String sex);

@Select("select id,name,sex,age,address from t_user where id = #{id}")
User queryUser(String id);

@Select("select id,name,sex,age,address from t_user where id = #{id}")
Map queryUser1(String id);

```

集合可以是什么类型？

List、Set、数组、Vector

```

@Select("select id,name,sex,age,address from t_user where sex = #{sex}
order by #{orderColumn}")
List<User> query(String sex, String orderColumn);

@Select("select id,name,sex,age,address from t_user where sex = #{sex}
order by #{orderColumn}")
List<Map> query1(String sex, String orderColumn);

```

集合的元素可以是什么类型的？

任意类型的，集合只是单个值多做几遍。

结果集中的列如何与结果、结果的属性对应？

根据结果集列名与属性名对应

如果属性名与列名不一样呢？

则需用户显式说明映射规则。

需要考虑DBCType --- JavaType的处理吗？

无论结果是什么类型的，在这里我们都是要完成一件事：从查询结果中获得数据返回，只是返回类型不同，有不同的获取数据的方式。

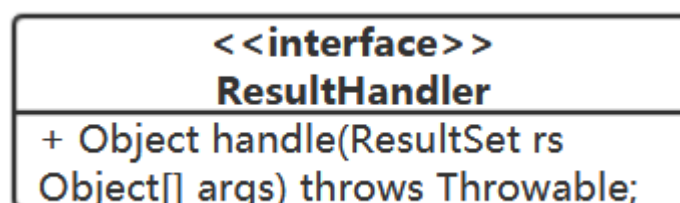
请思考：如何让下面这个方法的代码的写好后不再改变？

```

private Object handleResultSetReturn(ResultSet rs, MappedStatement ms,
Object[] args) {
    // TODO Auto-generated method stub
    return null;
}

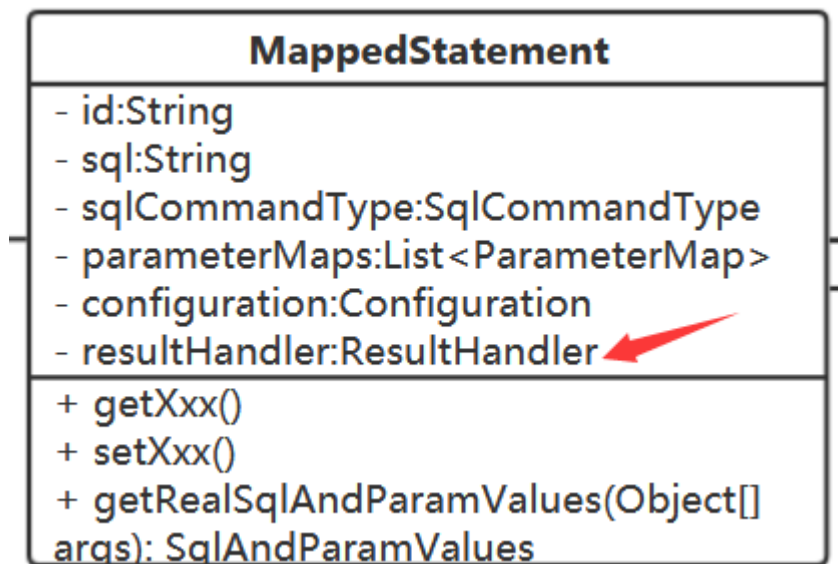
```

我需在此做个抽象，应用策略模式，不同的处理实现这个抽象接口。



那么在handleResultSetReturn()方法中我们从哪得到ResultHandler呢？

从MappedStatement 中获取，每个语句对象（查询类型的）中都持有它对应的结果处理器。  
在解析准备MappedStatement对象时根据方法的返回值类型选定对应的ResultHandler。



在handleResultSetReturn方法中只需调用ms中的ResultHandler：

```
private Object handleResultSetReturn(ResultSet rs, MappedStatement ms,
Object[] args) throws Throwable {
    return ms.getResultHandler().handle(rs, args);
}
```

可以来看具体的放回值情况了！

### 2.5.3.1 方法返回单个值

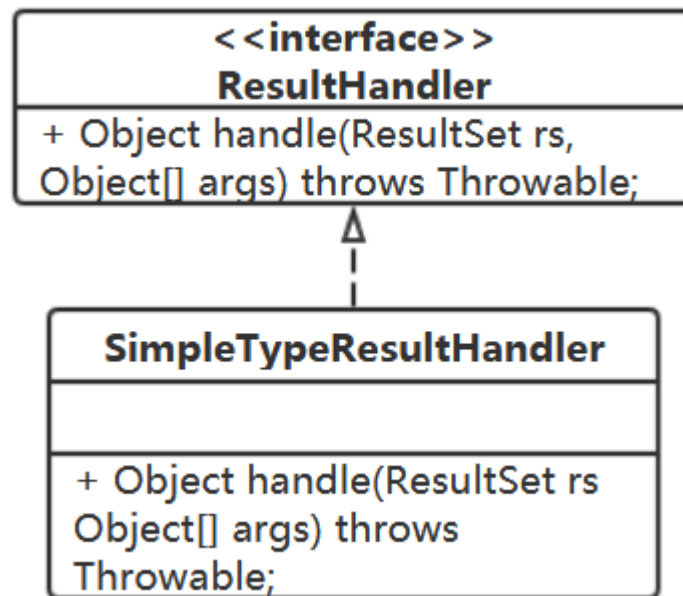
```
@Select("select count(1) from t_user where sex = #{sex}")
int query(String sex);

@Select("select id,name,sex,age,address from t_user where id = #{id}")
User queryUser(String id);

@Select("select id,name,sex,age,address from t_user where id = #{id}")
Map queryUser1(String id);
```

#### 1、基本数据类型、String 如何处理？

针对这种情况，提供对应的ResultHandler实现：



SimpleTypeResultHandler中需要定义什么属性？

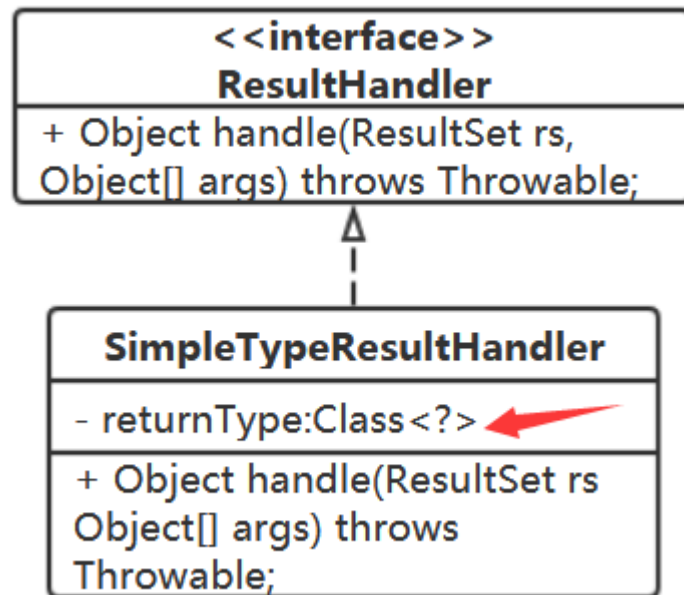
handle方法中的逻辑该是怎样的？

```
public Object handle(ResultSet rs, Object[] args) throws Throwable {  
    //从rs中取对应值  
    return rs.getXXX(000);  
}
```

问题1：该调用rs的哪个get方法？

得根据返回值来，返回值类型从哪来？

从SimpleTypeResultHandler中取，在创建MappedStatement时，根据反射获得的返回值类型给入到SimpleTypeResultHandler中。



SimpleTypeResultHandler的handle方法中的代码逻辑如下：

```
public Object handle(ResultSet rs, Object[] args) throws Throwable {
    if (returnType == short.class || returnType == Short.class) {
        return rs.getShort(xxx);
    } else if (returnType == int.class || returnType == Integer.class) {
        return rs.getInt(xxx);
    } else if (returnType == long.class || returnType == Long.class) {
        return rs.getLong(xxx);
    }
    ...
    return null;
}
```

问题2：该取结果集中的哪一列？

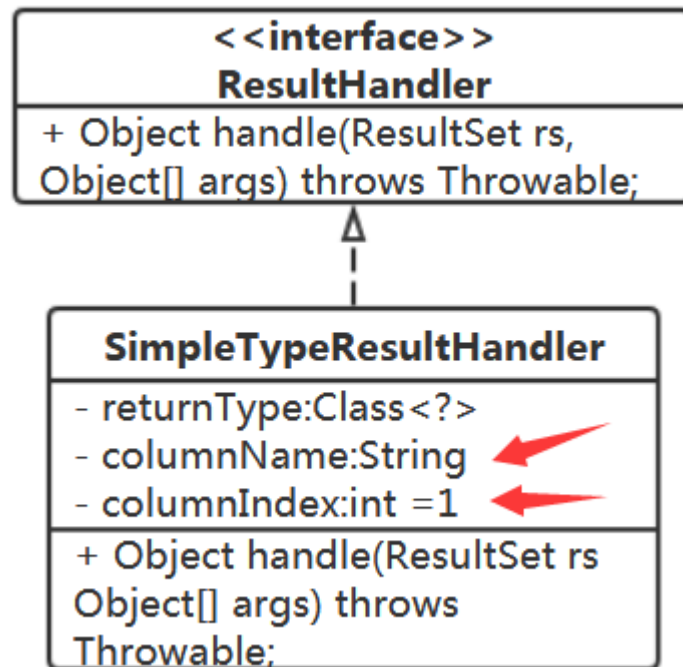
如果结果集中只有一列：那就取第1列。

如果结果集中是有多列呢？

问题：结果集中应不应该有多列？

两种方案：

- 1、该返回值情况下不允许结果集多列。
- 2、不限制，用户指定列名，不指定则取第一列。



问题3：这么多if else 合适吗？

不合适，咋办？

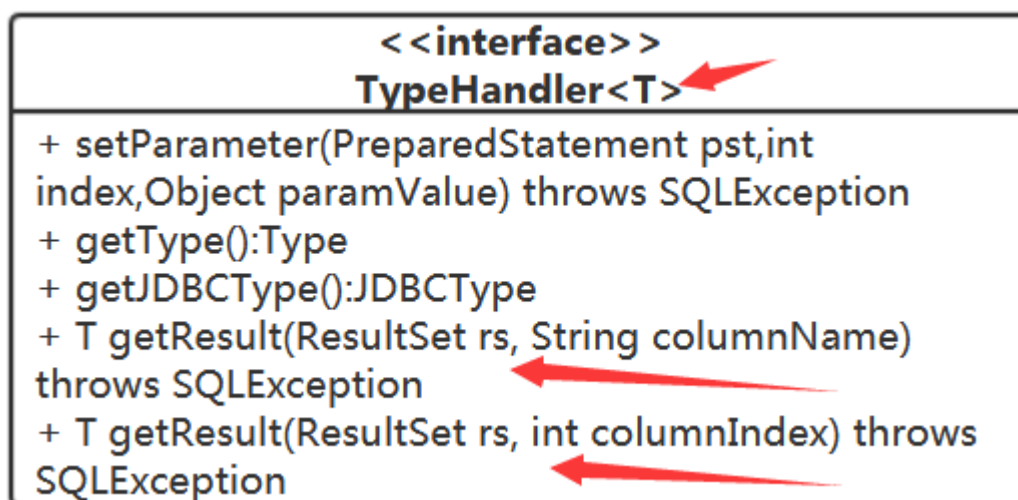
策略模式

该定义怎样的策略？

这是要做什么事情？

从结果集中获取值，跟pst.setXXX一样。

可不可以>TypeHandler中加方法？





```
public interface TypeHandler<T> {

    Type getType();

    JDBCType getJDBCType();

    void setParameter(PreparedStatement pst, int index, Object
paramValue) throws SQLException;

    T getResult(ResultSet rs, String columnName) throws SQLException;

    T getResult(ResultSet rs, int columnIndex) throws SQLException;

}
```

```
public class StringTypeHandler implements TypeHandler<String> {

    @Override
    public Type getType() {
        return String.class;
    }

    @Override
    public JDBCType getJDBCType() {
        return JDBCType.VARCHAR;
    }

    @Override
    public void setParameter(PreparedStatement pst, int index, Object
paramValue) throws SQLException {
        pst.setString(index, (String) paramValue);
    }

    @Override
    public String getResult(ResultSet rs, String columnName) throws
SQLException {
        return rs.getString(columnName);
    }

    @Override
```

```

    public String getResult(ResultSet rs, int columnIndex) throws
SQLException {
        return rs.getString(columnIndex);
    }
}

```

一样的，在启动解析阶段完成结果的TypeHandler选定。

TypeHandlerRegistry
- typeHandlerMap:Map<Type,Map<JDBCType,TypeHandler>>
+ registerTypeHandler(TypeHandler th)

根据返回值类型，从TypeHandlerRegistry中取，要取，还得有JDBCType，用户可以指定，也可不指定，不指定则使用默认的该类型的TypeHandler。

默认TypeHandler如何注册，修改registerTypeHandler方法的定义：

```

registerTypeHandler(TypeHandler th, boolean default){

    Map<JDBCType,TypeHandler> cmap = typeHandlerMap.get(th.getType());
    if(cmap == null){
        cmap = new HashMap<JDBCType,TypeHandler>();
        typeHandlerMap.put(th.getType(),cmap);
    }
    cmap.put(th.getJDBCType(),th);

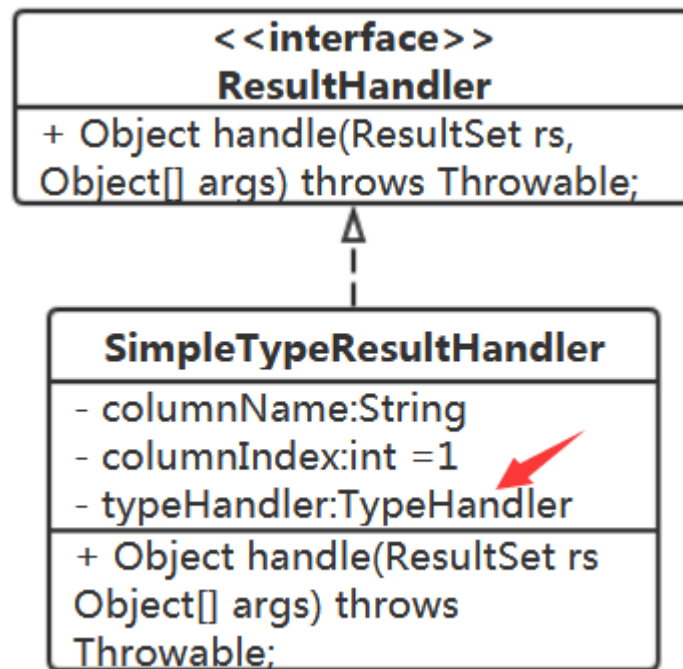
    if(default){
        cmap.put(null,th);
    }
}

```

很好，那就可以在SimpleTypeResultHandler中持有对应的TypeHandler。

问：在SimpleTypeResultHandler中还有必要持有Class<?> returnType吗？

不需要，在TypeHandler中有了。



SimpleTypeResultHandler 的handle方法代码就简单了：

```
public Object handle(ResultSet rs, Object[] args) throws Throwable {
    if (StringUtils.isEmpty(columnName)) {
        return typeHandler.getResult(rs, columnName);
    } else {
        return typeHandler.getResult(rs, columnIndex);
    }
}
```

## 2 对象类型返回结果的处理

```
@Select("select id,name,sex,age,address from t_user where id = #{id}")
User queryUser(String id);
```

分析：

1、要完成的事情是什么？

创建对象

从结果集中取数据给到对象

问题：

## 1、如何创建对象？

反射调用构造方法。

构造方法有多种情况：

### 1 未显式定义构造方法

```
public class User {  
    private String id;  
    private String name;  
    private String sex;  
    ...  
    public String getId() {  
        return id;  
    }  
    public void setId(String id) {  
        this.id = id;  
    }  
    ...  
}
```

这种情况不需要考虑什么，直接创建对象！

### 2 显式定义了一个构造方法

```
public class User {  
    private String id;  
    private String name;  
    private String sex;  
    ...  
  
    public User(String id, String name, String sex) {  
        super();  
        this.id = id;  
        this.name = name;  
        this.sex = sex;  
    }  
  
    public String getId() {  
        return id;  
    }  
}
```

```

    }
    public void setId(String id) {
        this.id = id;
    }
    ...
}

```

此种情况下，要创建对象，则需要对应的构造参数值。

问题1：构造参数值从哪来？

ResultSet

问题2：怎么知道该从ResultSet中取哪个列的值，取什么类型的值？

得定义构造参数与ResultSet中列的对应规则，下面的规则是否可以？

- 1、优先采用指定列名的方式：用参数名称当列名、或用户为参数指定列名（参数名与列名不一致时、取不到参数名时）；
- 2、如不能取得参数名，则按参数顺序来取对应顺序的列。

问题3：用户如何来指定列名？

注解、xml配置

```

public User(@Arg(column="id")String id, @Arg(column="xname")String
name, @Arg(column="sex")String sex) {
    super();
    this.id = id;
    this.name = name;
    this.sex = sex;
}

```

```

@Documented
@Retention(RUNTIME)
@Target(PARAMETER)
public @interface Arg {

    String name() default "";

    String column() default "";

    Class<?> javaType() default void.class;
}

```

```

    jdbcType jdbcType() default jdbcType.UNDEFINED;

    Class<? extends TypeHandler> typeHandler() default
    undefinedTypeHandler.class;

}

```

```

<resultMap id="User" type="com.study.mike.mybatis.sample.model.User">
    <constructor>
        <arg name="" column="" jdbcType="" javaType=""
typeHandler=""/>
    </constructor>
</resultMap>

```

mybatis-mapper.dtd 中增加如下定义

```

<!ELEMENT resultMap (constructor?)>
<!--ATTLIST resultMap
id CDATA #REQUIRED
type CDATA #REQUIRED
-->

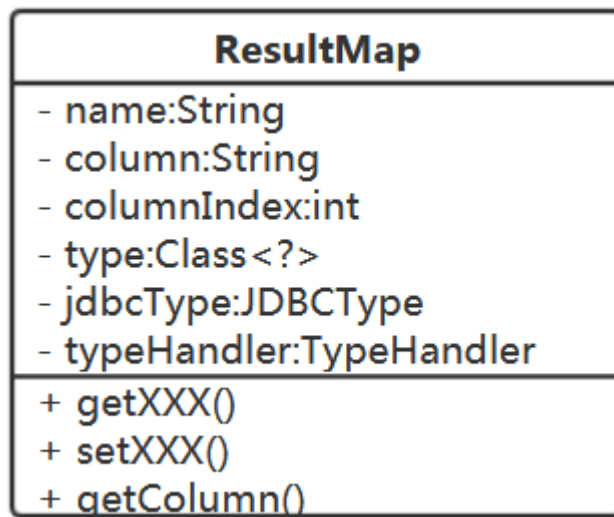
<!ELEMENT constructor (arg*)>

<!--ELEMENT arg EMPTY>
<!--ATTLIST arg
javaType CDATA #IMPLIED
column CDATA #IMPLIED
jdbcType CDATA #IMPLIED
typeHandler CDATA #IMPLIED
name CDATA #IMPLIED
-->

```

问题4：这些映射信息得到后如何表示、存储？

定义一个结果映射实体：ResultMap

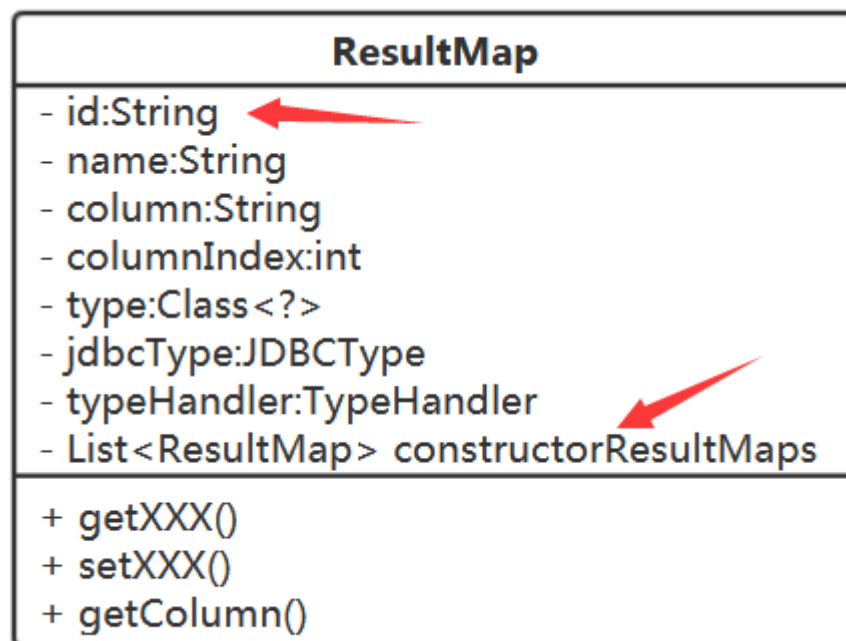


注意，在创建ResultMap时，当用户没有指定TypeHandler或是UndefinedTypeHandler时，要根据type、jdbcType取对应的typeHandler，没有则为null;

问题5、ResultMap 元素怎么表示？

ResultMap类定义本身就是表示一种java类型与JDBCType类型的映射，基本数据类型与复合类型（类）都是java类型。

扩充一下ResultMap即可：



注意：这里有个使用规则需要注意一下：

如果ResultMap中有TypeHandler，则该结果直接通过调用TypeHandler来获得。没有TypeHandler时则看有constructorResultMaps没，有则根据此取结果集中的值来调用对应的构造方法创建对象。

### 3 定义了多个构造方法，怎么办？

```
public class User {  
    private String id;  
    private String name;  
    private String sex;  
    ...  
  
    public User(String id, String name, String sex) {  
        super();  
        this.id = id;  
        this.name = name;  
        this.sex = sex;  
    }  
    public User(String id, String name, String sex, int age) {  
        super();  
        this.id = id;  
        this.name = name;  
        this.sex = sex;  
        this.age = age;  
    }  
    public String getId() {  
        return id;  
    }  
    public void setId(String id) {  
        this.id = id;  
    }  
    ...  
}
```

用户指定构造方法，没有指定时则用默认构造方法（没有则报错）。

用户怎么指定：

注解：



```

@MapConstructor
public User(@Arg("id")String id, @Arg("xname")String name,
@Arg("sex")String sex) {
    super();
    this.id = id;
    this.name = name;
    this.sex = sex;
}

```

```

/**
 * 标识选用的构造方法
 */
@Documented
@Retention(RUNTIME)
@Target(CONSTRUCTOR)
public @interface MapConstructor {

}

```

xml：根据constructor元素中 arg元素的数量、javaType来确定构造函数。注意arg有顺序规则、必须指定构造方法的全部参数。

```

<resultMap id="User" type="com.study.mike.mybatis.sample.model.User">
    <constructor>
        <arg column="id" javaType="String"/>
        <arg column="name" javaType="String"/>
        <arg column="sex" javaType="String"/>
    </constructor>
</resultMap>

```

## 2、该给对象哪些属性值？

创建出对象后，可以从结果集中取值来填充对象的属性。

问题1：该给哪些属性赋值？

可以有两种规则：

- 1、用户指定要给哪些属性赋值。
- 2、自动映射赋值：取列的值赋给同名的属性。

两者可以一起使用。

那么这里就涉及两个事情：

1、用户如何指定？

注解方式：我们给定义一个注解 @Result

```
public class User {  
    @Result  
    private String id;  
    @Result(column="xname")  
    private String name;  
    ...  
}
```

```
@Documented  
@Retention(RUNTIME)  
@Target({ TYPE, FIELD })  
public @interface Result {  
    String column() default "";  
    Class<?> javaType() default void.class;  
    JdbcType jdbcType() default JdbcType.UNDEFINED;  
    Class<? extends TypeHandler> typeHandler() default  
    UndefinedTypeHandler.class;  
}
```

xml方式：

```
<resultMap id="User" type="com.study.mike.mybatis.sample.model.User">  
    <constructor>  
        <arg column="id" javaType="String"/>  
        <arg column="name" javaType="String"/>  
        <arg column="sex" javaType="String"/>  
    </constructor>  
    <result property="age" column="age" />  
</resultMap>
```

mybatis-mapper.dtd

```
<!ELEMENT resultMap (constructor?,result*)>  
<!ATTLIST resultMap
```

```

id CDATA #REQUIRED
type CDATA #REQUIRED
>

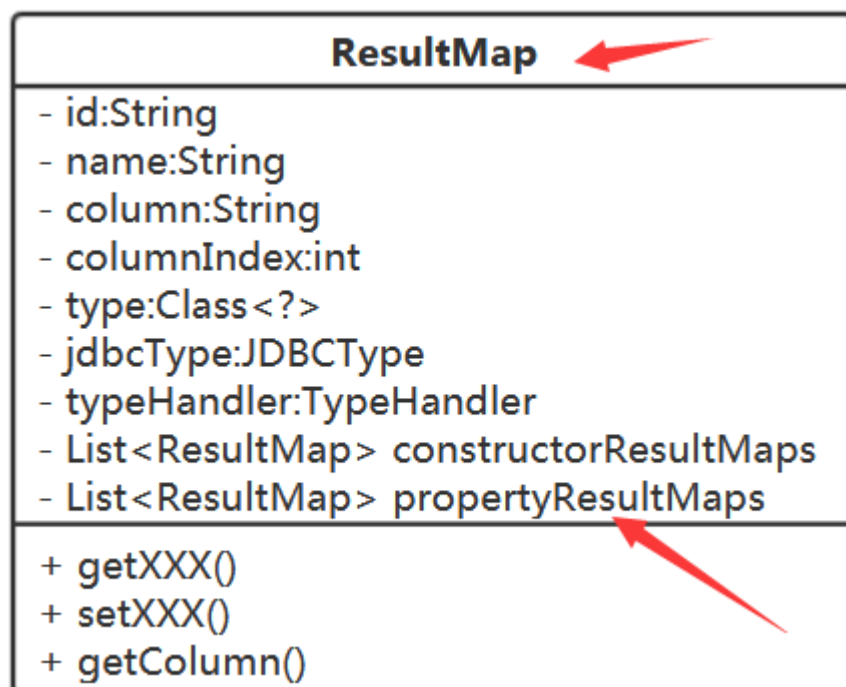
<!ELEMENT constructor (arg*)>

<!ELEMENT arg EMPTY>
<!ATTLIST arg
javaType CDATA #IMPLIED
column CDATA #IMPLIED
jdbcType CDATA #IMPLIED
typeHandler CDATA #IMPLIED
name CDATA #IMPLIED
>

<!ELEMENT result EMPTY>
<!ATTLIST result
property CDATA #IMPLIED
javaType CDATA #IMPLIED
column CDATA #IMPLIED
jdbcType CDATA #IMPLIED
typeHandler CDATA #IMPLIED
>

```

问题：这些信息如何表示、存储？



## 2、是否自动映射如何指定？

增加一个属性即可：


```
<resultMap id="User" type="com.study.mike.mybatis.sample.model.User"
autoMapping="true">
    <result property="age" column="age" />
</resultMap>
```

```
<!ELEMENT resultMap (constructor?,result*)>
<!ATTLIST resultMap
id CDATA #REQUIRED
type CDATA #REQUIRED
autoMapping (true|false) #IMPLIED
>
```


注解方式：

```
/**
标识类对象要进行自动映射
*/
@Documented
@Retention(RUNTIME)
@Target({ TYPE, FIELD })
public @interface AutoMapping {
}
```

```
@AutoMapping
public class User {
    @Result
    private String id;
    @Result(column="xname")
    private String name;
    ...
}
```

ResultMap
<ul style="list-style-type: none"><li>- id:String</li><li>- name:String</li><li>- column:String</li><li>- columnIndex:int</li><li>- type:Class&lt;?&gt;</li><li>- jdbcType:JDBCType</li><li>- typeHandler:TypeHandler</li><li>- List&lt;ResultMap&gt; constructorResultMaps</li><li>- List&lt;ResultMap&gt; propertyResultMaps</li><li>- autoMapping:boolean </li></ul>
<ul style="list-style-type: none"><li>+ getXXX()</li><li>+ setXXX()</li><li>+ getColumn()</li></ul>

为方便统一开启自动映射，我们可以在Configuration中设计一个全局配置参数，具体的可以覆盖全局的。

Configuration
<ul style="list-style-type: none"> <li>- mappedStatements:Map&lt;String,MappedStatement&gt;</li> <li>- xmlMapperBuilder:XMLMapperBuilder</li> <li>- mapperAnnotationBuilder:MapperAnnotationBuilder</li> <li>- dataSource:DataSource</li> <li>- mappers:Set&lt;Class&lt;?&gt;&gt;</li> <li>- typeHandlerRegistry:TypeHandlerRegistry</li> <li>- autoMapping:boolean </li> </ul>
<ul style="list-style-type: none"> <li>+ addMappedStatement(MappedStatement ms)</li> <li>+ getMappedStatement(String id) : MappedStatement</li> <li>+ hasMappedStatement(String id) : boolean</li> <li>+ addXmlMapper(InputStream inputStream,String resource)</li> <li>+ addMapper(Class&lt;?&gt; type)</li> <li>+ addMappers(String packageName)</li> <li>+ addMappers(String packageName,Class&lt;?&gt; superType)</li> <li>+ addMappersWithAnnotation(String packageName,Class&lt;?&gt; annotation)</li> <li>+ addMappers(String packageName,Class&lt;?&gt; superType,Class&lt;?&gt; annotation)</li> <li>+ setDataSource(DataSource dataSource)</li> <li>+ getDataSource():DataSource</li> <li>+ getMappers():Set&lt;Class&lt;?&gt;&gt;</li> <li>+ getTypeHandlerRegistry():TypeHandlerRegistry</li> </ul>

在哪可配置它？

在mybatis-config.xml中增加一个配置项即可。

```

<configuration>
  <settings>
    <setting name="autoMappingBehavior" value="PARTIAL"/>
  </settings>
</configuration>

```

### 3、对象中包含对象该如何映射及处理

对象中包对象是个问题，先把问题搞清楚，看下面的语句示例：

```

<!-- Very Complex Statement -->
<select id="selectBlogDetails" resultMap="detailedBlogResultMap">
  select
    B.id as blog_id,

```

```

        B.title as blog_title,
        B.author_id as blog_author_id,
        A.id as author_id,
        A.username as author_username,
        A.password as author_password,
        A.email as author_email,
        A.bio as author_bio,
        A.favourite_section as author_favourite_section,
        P.id as post_id,
        P.blog_id as post_blog_id,
        P.author_id as post_author_id,
        P.created_on as post_created_on,
        P.section as post_section,
        P.subject as post_subject,
        P.draft as draft,
        P.body as post_body
    from Blog B
        left outer join Author A on B.author_id = A.id
        left outer join Post P on B.id = P.blog_id
    where B.id = #{id}
</select>

```

再看类

```

public class Blog {
    private String id;
    private String title;
    private Author author;
    private List<Post> posts;
    ....
}

public class Author {
    private String id;
    private String username;
    ...
}

public class Post {
    private String id;
    private String subject;
    ...
}

```

```
}
```

这就是对象中包含对象，要从查询结果中得到Blog，Blog的Author posts数据。

这就是ORM中的关系映射问题。

结果的映射是简单的，因为就是指定里面的属性取哪个列的值。

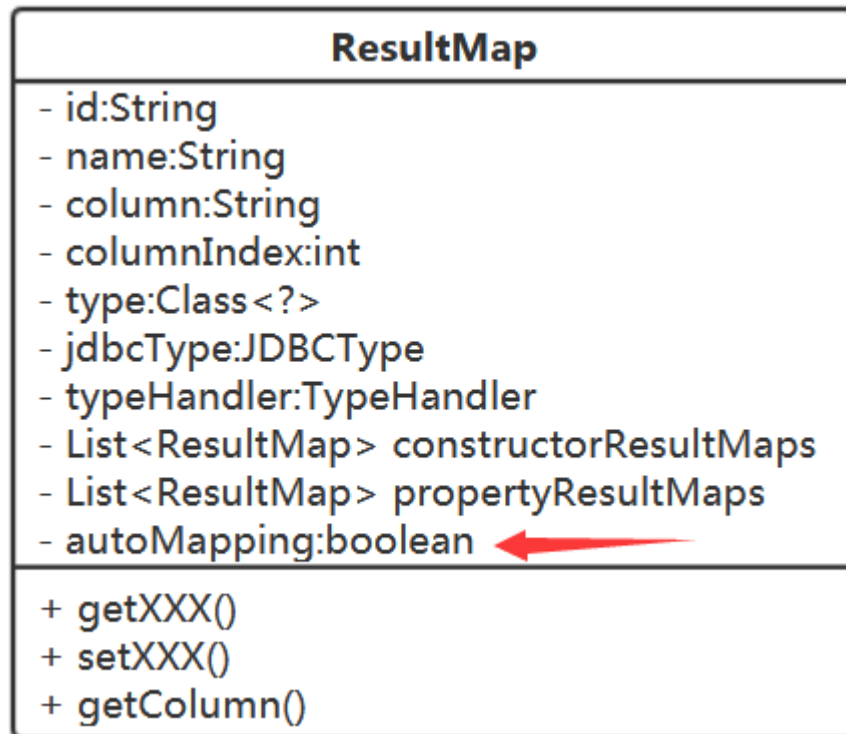
```
public class Blog {
    @Result(column="blog_id")
    private String id;
    @Result(column="blog_title")
    private String title;
    @Result
    private Author author;
    @Result
    private List<Post> posts;
    ....
}

public class Author {
    @Result(column="author_id")
    private String id;
    @Result(column="author_username")
    private String username;
    ...
}

public class Post {
    @Result(column="post_id")
    private String id;
    @Result(column="post_subject")
    private String subject;
    ...
}
```

我们的ResultMap类也是支持的：





但是从结果集中取值来填装对象则是复杂的！

请先看查询的结果示例：

blog_id	blog_title	author_id	author_username	post_id	post_subject
b01	成为架构师	a888	mike	p00000001	集群架构原理
b01	成为架构师	a888	mike	p00000002	缓存架构
b01	成为架构师	a888	mike	p00000003	安全架构
b02	成为理发大师	a666	tony	p10000011	精洗之道
b02	成为理发大师	a666	tony	p10000012	精剪之道
b02	成为理发大师	a666	tony	p10000013	吹发绝技

```
while(rs.next()){
```

```
}
```

复杂点：不是一行一个Blog对象，处理行时要判断该行的blog是否已取过了。

问题核心点在哪？

当我操作一行，如何判断该行的Blog已经取过没？

这就要求要知道区分Blog的唯一标识、区分Author的唯一标识。怎么知道？

用户得告诉我们他们的id属性是哪个，对应的列是哪个。

让用户怎么来指定id属性呢？

注解方式：在@Arg、@Result注解中增加id指定项。

```
@Documented
@Retention(RUNTIME)
@Target(PARAMETER)
public @interface Arg {
    boolean id() default false;
    String name() default "";
    String column() default "";
    Class<?> javaType() default void.class;
    JdbcType jdbcType() default JdbcType.UNDEFINED;
    Class<? extends TypeHandler> typeHandler() default
    UndefinedTypeHandler.class;
}
```

```
@Documented
@Retention(RUNTIME)
@Target({ TYPE, FIELD })
public @interface Result {
    boolean id() default false;
    String column() default "";
    Class<?> javaType() default void.class;
    JdbcType jdbcType() default JdbcType.UNDEFINED;
    Class<? extends TypeHandler> typeHandler() default
    UndefinedTypeHandler.class;
}
```

xml方式增加：增加argId、id元素


```

<resultMap id="detailedBlogResultMap" type="Blog">
  <constructor>
    <idArg column="blog_id" javaType="int"/>
  </constructor>
  ....
</resultMap>

<resultMap id="AuthorMap" type="Author">
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
  <result property="password" column="author_password"/>
</resultMap>

```

在ResultMap中增加ID信息

ResultMap
<ul style="list-style-type: none"> <li>- id:String</li> <li>- name:String</li> <li>- column:String</li> <li>- columnIndex:int</li> <li>- type:Class&lt;?&gt;</li> <li>- jdbcType:JDBCType</li> <li>- typeHandler:TypeHandler</li> <li>- List&lt;ResultMap&gt; constructorResultMaps</li> <li>- List&lt;ResultMap&gt; propertyResultMaps</li> <li>- List&lt;ResultMap&gt; idResultMaps </li> <li>- autoMapping:boolean</li> </ul>
<ul style="list-style-type: none"> <li>+ getXXX()</li> <li>+ setXXX()</li> </ul>

问题：要体现出一对一，一对多关系吗？我们会在哪里需要知道这个关系？

看一个mybatis中的复杂xml ResultMap示例：

```

<!-- 超复杂的 Result Map -->
<resultMap id="detailedBlogResultMap" type="Blog">
  <constructor>
    <idArg column="blog_id" javaType="int"/>

```

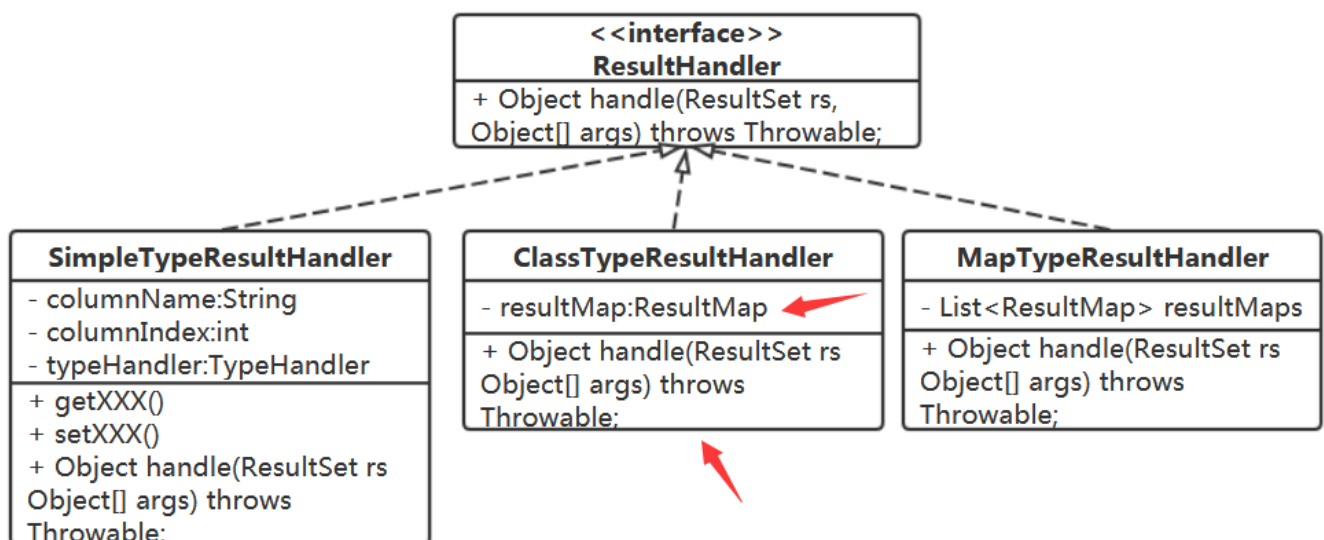
```

</constructor>
<result property="title" column="blog_title"/>
<association property="author" javaType="Author">
  <id property="id" column="author_id"/>
  <result property="username" column="author_username"/>
  <result property="password" column="author_password"/>
  <result property="email" column="author_email"/>
  <result property="bio" column="author_bio"/>
  <result property="favouriteSection"
column="author_favourite_section"/>
</association>
<collection property="posts" ofType="Post">
  <id property="id" column="post_id"/>
  <result property="subject" column="post_subject"/>
  <association property="author" javaType="Author"/>
  <collection property="comments" ofType="Comment">
    <id property="id" column="comment_id"/>
  </collection>
</collection>
</resultMap>

```

知道唯一标识了，要判断前面是否取过了，则还需要有个上下文持有取到的对象，并能根据id列值取到对应的对象。

为对象类型返回结果定义一个ResultHandler实现ClassTypeResultHandler：



### 3 Map

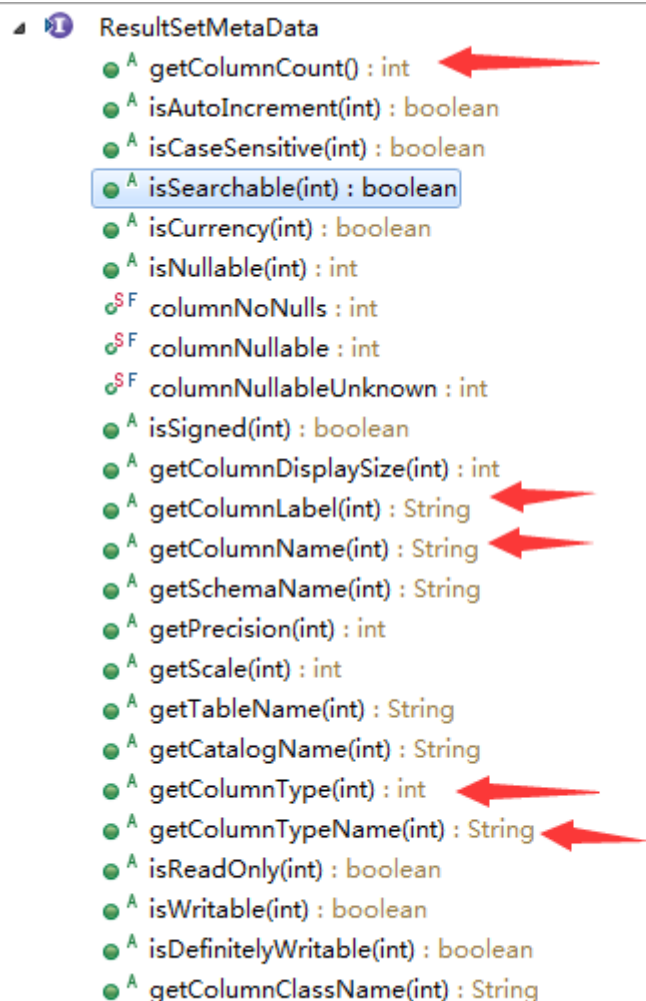
```
@Select("select id,name,sex,age,address from t_user where id = #{id}")
Map queryUser1(String id);
```

不能在解析阶段获得ResultMap

当执行完第一次查询就可以确定下来

TypeHandlerRegistry
- typeHandlerMap:Map<Type,Map<JDBCType,TypeHandler>>
+ registerTypeHandler(TypeHandler th)

我们从结果集中能得到的是JDBCType



问题：

1、key 用什么？

用列名

2、取成什么java类型的值？

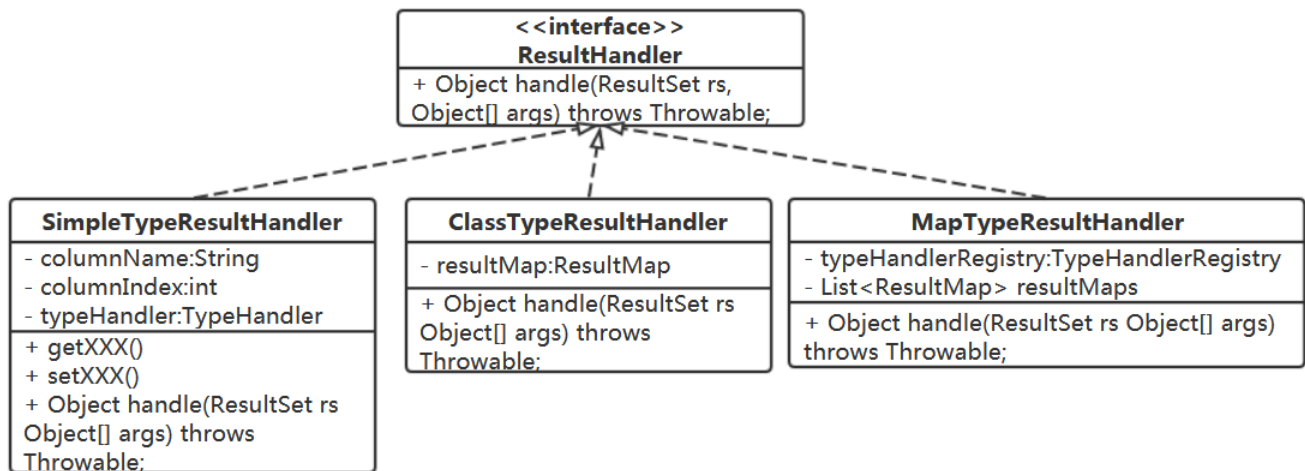
JDBCType中根据整型类型值获得对应的JDBCType

```
/**
 * Returns the {@code JDBCType} that corresponds to the specified
 * {@code Types} value
 * @param type {@code Types} value
 * @return The {@code JDBCType} constant
 * @throws IllegalArgumentException if this enum type has no
constant with
 * the specified {@code Types} value
 * @see Types
 */
public static JDBCType valueOf(int type) {
    for( JDBCType sqlType : JDBCType.class.getEnumConstants()) {
        if(type == sqlType.type)
            return sqlType;
    }
    throw new IllegalArgumentException("Type:" + type + " is not a
valid "
        + "Types.java value.");
}
```

TypeHandler ---> javaType

在TypeHandlerRegistry中定义一个JDBCType类型对应的默认的类型Handler集合，来完成取java值放入到Map中

TypeHandlerRegistry	
- typeHandlerMap:Map<Type,Map<JDBCType,TypeHandler>>	
- jdbcTypeHandlerMap:Map<JDBCType,TypeHandler>	←
+ registerTypeHandler(TypeHandler th)	



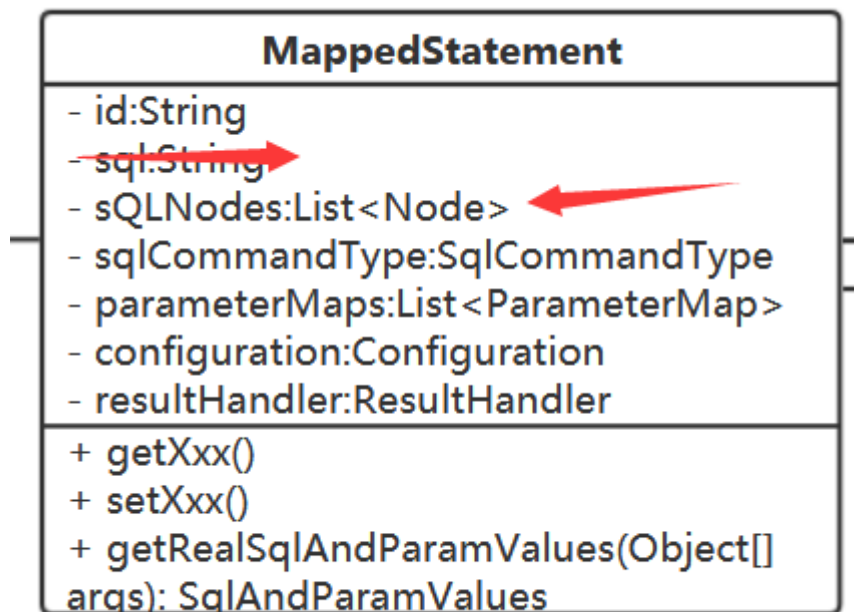
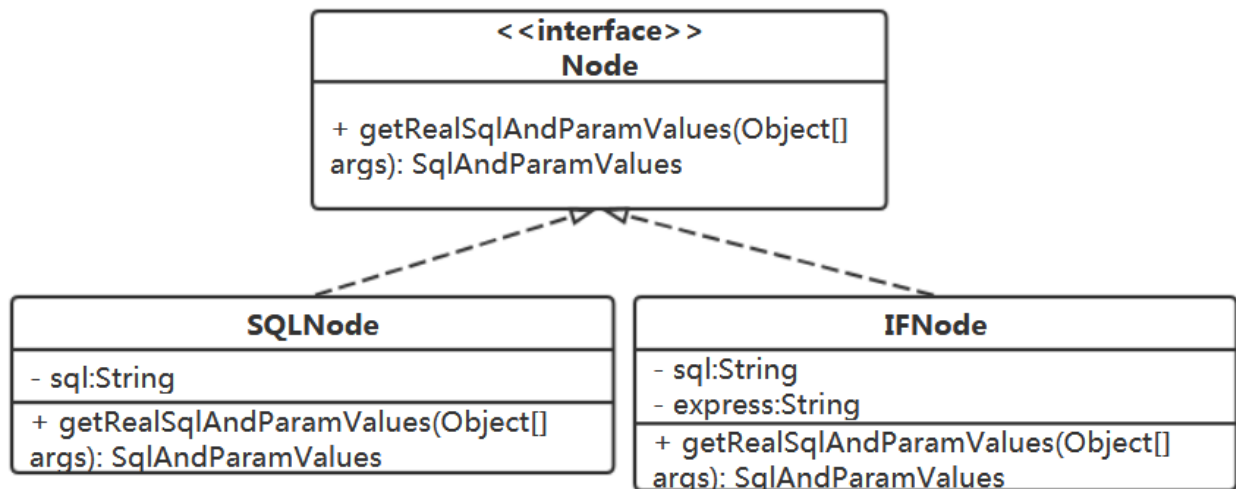
第一次处理结果时，要把这个ResultMaps填充好，后需查询结果的处理就是直接使用resultMaps

### 2.5.3.2 方法返回集合

返回集合就是单个的重复

## 2.6 动态SQL

---



## 2.7 不同的语句类型

Statement

PreparedStatement

CallableStatement