MyBatis 的核心配置文件 SqlMapConfig.xml

一.核心配置文件 SqlMapConfig.xml

有的叫 SqlMapConfig.xml, 也有的叫 mybatis-config.xml, 这里习惯用 SqlMapConfig.xml . 其中，这个配置文件主要配置的内容依次是: 注意，顺序是不能颠倒的。 可见约束文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- 引入约束 -->
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <properties/> <!--配置属性信息--->

    <setting/>      <!--设置-mybatis 运行参数-->

    <typeAliases /><!--类型命名 别名-->

    <typeHandlers/><!--类型处理器--->

    <objectFactory/><!--对象工厂-->

    <plugins/>          <!--插件--->

    <environments>  <!--配置环境-->

        <environment><!--环境变量-->

            <transactionManager/><!--事务管理器-->

                <dataSource /><!--数据源-->

        </environment>

    </environments>
```

```
    <databaseIdProvider/><!--数据库厂商标识-->

    <mappers/><!--映射器-->
</configuration>
```

```
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
```

这里，只讲一些常见的使用方式，如<properties>属性，<setting> <typeAliases> <environments> <mappers> 其余的暂时不讲。

二. properties 属性

可以引用配置文件，或者提前设置属性，来进行引用属性。 常见的就是关于数据库的配置。

以前的写法:

```
<!-- 开发环境  development -->
        <environments default="development">
                <environment id="development">
                        <!-- 事务管理 -->
                        <transactionManager type="JDBC"></transactionManager>
                        <!-- 数据源 ，为 pooled 连接池 -->
                        <dataSource type="pooled">
                                <!-- 直接硬编码在 dataSource 资源里面 -->
                                <property name="driver"
value="com.mysql.jdbc.Driver"/>
                                <property name="url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf8"/>
                                <property name="username" value="root"/>
                                <property name="password" value="abc123"/>
                        </dataSource>
                </environment>
        </environments>
```

```
13
14
15
```

二.一 子元素设置

```
<!-- 子元素设置 -->
        <properties>
                <!-- 先设置一些属性，为了避免 username 和 password 重复，前面用 jdbc. 前缀
-->
                <property name="jdbc.driver" value="com.mysql.jdbc.Driver"/>
                <property name="jdbc.url"
value="jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf8"/>
                <property name="jdbc.username" value="root"/>
                <property name="jdbc.password" value="abc123"/>
        </properties>
```

```
1
2
3
4
5
6
7
8
```

然后在 datasource 数据库资源中配置：

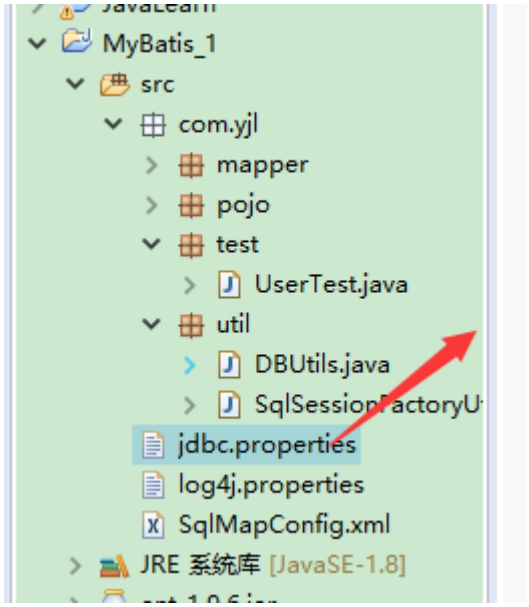
```
<!-- 数据源 ， 为 pooled 连接池 -->
        <dataSource type="pooled">
                                <!-- 直接硬编码在 dataSource 资源里面 -->
                                <property name="driver" value="${jdbc.driver}"/>
                                <property name="url" value="${jdbc.url}"/>
                                <property name="username"
value="${jdbc.username}"/>
```

```
                        <property name="password"
value="${jdbc.password}"/>
</dataSource>
```

调用测试方法 findAllTest 可以正常的查询。

二.二 配置文件

在 src 源文件下创建 jdbc.properties 配置文件，里面设置属性，进行相应的引用。




```
jdbc.driver=com.mysql.jdbc.Driver
```

```
jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf8
jdbc.username=root
jdbc.password=abc123
```

SqlMapConfig.xml 中引入配置文件

```
        <!--配置文件引入  url 为网络资源，  只能引入一个文件-->
        <properties resource="jdbc.properties"></properties>
```

相应的数据源配置为:

```
<!-- 数据源 ，为 pooled 连接池 -->
                        <dataSource type="pooled">
                                <!-- 直接硬编码在 dataSource 资源里面 -->
                                <property name="driver" value="${jdbc.driver}"/>
                                <property name="url" value="${jdbc.url}"/>
                                <property name="username"
value="${jdbc.username}"/>
                                <property name="password"
value="${jdbc.password}"/>
                        </dataSource>
```

- 6
- 7
- 8

二.三 程序参数传递

如数据库开发时， 数据库的用户名和密码，生产数据库的用户名和密码应该对开发者是保密的，运维人员需要对其进行相应的加密，所以配置文件中的信息通常是加密后的信息。

代码形式为:

```
/**
         * 单例模式 获取实例
         * @author 两个蝴蝶飞
         * @return
         */
        public static SqlSessionFactory getInstance(){
                synchronized(SqlSessionFactoryUtils.class){
                        if(sqlSessionFactory==null){
                                InputStream input=null;
                                // Mybatis 核心配置文件名
                                String resource_name="SqlMapConfig.xml";

                                //关于属性文件
                                InputStream proStream=null;
                                Reader propReader=null;
                                Properties properties=null;
                                try {

        input=Resources.getResourceAsStream(resource_name);

        proStream=Resources.getResourceAsStream("jdbc.properties");
                                        propReader=new
InputStreamReader(proStream);
                                        properties=new Properties();
                                        properties.load(propReader);
                                        //重新编码用户名和密码 ,开发者定义好的的 decode
encode() 的加密和解密规则。
```

```
        properties.setProperty("username",decode(properties.getProperty("jdbc.usern
ame")));

        properties.setProperty("password",decode(properties.getProperty("jdbc.passw
ord")));
                                        sqlSessionFactory=new
SqlSessionFactoryBuilder().build(input,properties);
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                }
                return sqlSessionFactory;
        }
```

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
```

二.四 优先级

程序参数传递的优先级最高， 配置文件 resource/url 的次之，properties 属性中指定的属性优先级最低。

建议:

1. 最好不要混用，只用一种方式即可。
2. 最好选用 属性文件的方式进行配置。

三. setting 配置

setting 配置属性，会改变 mybatis 的运行时的行为，即使不配置 setting,程序也会正常的运行。

setting 配置的各项参数有:

设置参数	描　述	有效值	默认值
cacheEnabled	该配置影响所有映射器中配置的缓存全局开关	true、false	true
lazyLoadingEnabled	延迟加载的全局开关。当它开启时，所有关联对象都会延迟加载。特定关联关系中可通过设置fetchType属性来覆盖该项的开关状态	true、false	false
aggressiveLazyLoading	当启用时，对任意延迟属性的调用会使带有延迟加载属性的对象完整加载；反之，每种属性将会按需加载	true、false	true
multipleResultSetsEnabled	是否允许单一语句返回多结果集（需要兼容驱动）	true、false	true

设置参数	描　述	有效值	默认值
useColumnLabel	使用列标签代替列名。不同的驱动在这方面会有不同的表现，具体可参考相关驱动文档或通过测试这两种不同的模式来观察所用驱动的结果	true、false	true
useGeneratedKeys	允许 JDBC 支持自动生成主键，需要驱动兼容。如果设置为 true，则这个设置强制使用自动生成主键，尽管一些驱动不能兼容但仍可正常工作（比如Derby）	true、false	false
autoMappingBehavior	指定 MyBatis 应如何自动映射列到字段或属性。 NONE 表示取消自动映射； PARTIAL 只会自动映射没有定义嵌套结果集映射的结果集； FULL 会自动映射任意复杂的结果集（无论是否嵌套）	NONE、PARTIAL、FULL	PARTIAL
defaultExecutorType	配置默认的执行器。 SIMPLE 是普通的执行器； REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行批量更新	SIMPLE、REUSE、BATCH	SIMPLE
defaultStatementTimeout	设置超时时间，它决定驱动等待数据库响应的秒数。当没有设置的时候它取的就是驱动默认的时间	Any positive integer	Not set (null)
safeRowBoundsEnabled	允许在嵌套语句中使用分页（RowBounds）	true、false	False

mapUnderscoreToCamelCase	是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A_COLUMN 到经典 Java 属性名 aColumn 的类似映射	true、false	false
localCacheScope	MyBatis 利用本地缓存机制（Local Cache）防止循环引用（circular references）和加速重复嵌套查询。默认值为 SESSION，这种情况下会缓存一个会话中执行的所有查询。若设置值为 STATEMENT，本地会话仅用在语句执行上，对相同 SqlSession 的不同调用将不会共享数据	SESSION、STATEMENT	SESSION
jdbcTypeForNull	当没有为参数提供特定的 JDBC 类型时，为空值指定 JDBC 类型。某些驱动需要指定列的 JDBC 类型，多数情况直接用一般类型即可，比如 NULL、VARCHAR、OTHER	JdbcType 枚举，最常见的是 NULL、VARCHAR 和 OTHER	OTHER

设置参数	描　　述	有效值	默认值
lazyLoadTriggerMethods	指定对象的方法触发一次延迟加载	如果是一个方法列表，我们则用逗号将它们隔开	equals,clone,hashCode,toString
defaultScriptingLanguage	指定动态 SQL 生成的默认语言	你可以配置类的别名或者类的全限定名	org.apache.ibatis.scripting.xmltags.XMLDynamicLanguageDriver
callSettersOnNulls	指定当结果集中值为 null 的时候是否调用映射对象的 setter（map 对象时为 put）方法，这对于有 Map.keySet()依赖或 null 值初始化的时候是有用的。注意基本类型（int、boolean 等）是不能设置成 null 的	true、false	false
logPrefix	指定 MyBatis 增加到日志名称的前缀	任何字符串	没有设置
logImpl	指定 MyBatis 所用日志的具体实现，未指定时将自动查找	SLF4J、LOG4J、LOG4J2、JDK_LOGGING、COMMONS_LOGGING、STDOUT_LOGGING、NO_LOGGING	没有设置
proxyFactory	指定 MyBatis 创建具有延迟加载能力的对象所用到的代理工具	CGLIB、JAVASSIST	版本 3.3.0（含）以上 JAVASSIST，否则 CGLIB

上面的这些配置不需要全部都配置，只需要配置常用的一些属性即可。

常用的开发中，需要配置的属性:

```
<settings>
                <!-- 设置配置文件 -->
```

```
        <!-- 开启二级缓存 -->
        <setting name="cacheEnabled" value="true"/>
        <!-- 控制懒加载的 -->
        <setting name="lazyLoadingEnabled" value="true"/>
        <setting name="aggressiveLazyLoading" value="false"/>
        <setting name="multipleResultSetsEnabled" value="true"/>
        <setting name="useColumnLabel" value="true"/>
        <setting name="useGeneratedKeys" value="false"/>
        <setting name="autoMappingBehavior" value="PARTIAL"/>
        <setting name="autoMappingUnknownColumnBehavior" value="WARNING"/>
        <setting name="defaultExecutorType" value="SIMPLE"/>
        <setting name="defaultStatementTimeout" value="25"/>
        <setting name="defaultFetchSize" value="100"/>
        <setting name="safeRowBoundsEnabled" value="false"/>
        <setting name="localCacheScope" value="SESSION"/>
        <setting name="jdbcTypeForNull" value="OTHER"/>
        <setting name="lazyLoadTriggerMethods"
value="equals,clone,hashCode,toString"/>
        <!-- 设置日志为 log4j -->
        <setting name="logImpl" value="LOG4J"/>
    </settings>
```

```
12
13
14
15
16
17
18
19
20
21
22
```

四.别名 typealiases

实体类的全限定名称过长，需要使用一个短的名称来代替它，可以在 Mybatis 整个上下文生命周期中使用。 可以分为 系统定义别名和自定义别名，其中 别名是不区分大小写的， 别名 User 与别名 user 是同样的意思。 一个 typealiases 的实体是在解析配置文件的时候生成的，长期保存在 Configuration 对象之中，当我们使用它时，再把它拿出来，而不是每一次使用时，都重新生成。

四.一 MyBatis 系统定义别名

别　名	映射的类型	支持数组
_byte	byte	是
_long	long	是
_short	short	是
_int	int	是
_integer	int	是
_double	double	是
_float	float	是
_boolean	boolean	是
string	String	否

别　名	映射的类型	支持数组
byte	Byte	是
long	Long	是
short	Short	是
int	Integer	是
integer	Integer	是
double	Double	是
float	Float	是
boolean	Boolean	是
date	Date	是
decimal	BigDecimal	是
bigdecimal	BigDecimal	是
object	Object	是
map	Map	否
hashmap	HashMap	否
list	List	否
arraylist	ArrayList	否
collection	Collection	否
iterator	Iterator	否
ResultSet	ResultSet	否

支持数组的，只要加个 [] 就可以，如 byte[] 为_byte[] 。

系统定义的别名，在 org.apache.ibatis.type.TypeAliasRegistry 类中进行定义。

```
public TypeAliasRegistry() {
    registerAlias("string", String.class);
    registerAlias("byte", Byte.class);
```

```
registerAlias("long", Long.class);
registerAlias("short", Short.class);
registerAlias("int", Integer.class);
registerAlias("integer", Integer.class);
registerAlias("double", Double.class);
registerAlias("float", Float.class);
registerAlias("boolean", Boolean.class);

registerAlias("byte[]", Byte[].class);
registerAlias("long[]", Long[].class);
registerAlias("short[]", Short[].class);
registerAlias("int[]", Integer[].class);
registerAlias("integer[]", Integer[].class);
registerAlias("double[]", Double[].class);
registerAlias("float[]", Float[].class);
registerAlias("boolean[]", Boolean[].class);

registerAlias("_byte", Byte.TYPE);
registerAlias("_long", Long.TYPE);
registerAlias("_short", Short.TYPE);
registerAlias("_int", Integer.TYPE);
registerAlias("_integer", Integer.TYPE);
registerAlias("_double", Double.TYPE);
registerAlias("_float", Float.TYPE);
registerAlias("_boolean", Boolean.TYPE);

registerAlias("_byte[]", byte[].class);
registerAlias("_long[]", long[].class);
registerAlias("_short[]", short[].class);
registerAlias("_int[]", int[].class);
registerAlias("_integer[]", int[].class);
registerAlias("_double[]", double[].class);
registerAlias("_float[]", float[].class);
registerAlias("_boolean[]", boolean[].class);

registerAlias("date", Date.class);
registerAlias("decimal", BigDecimal.class);
registerAlias("bigdecimal", BigDecimal.class);
registerAlias("biginteger", BigInteger.class);
registerAlias("object", Object.class);

registerAlias("date[]", Date[].class);
```

```
    registerAlias("decimal[]", BigDecimal[].class);
    registerAlias("bigdecimal[]", BigDecimal[].class);
    registerAlias("biginteger[]", BigInteger[].class);
    registerAlias("object[]", Object[].class);

    registerAlias("map", Map.class);
    registerAlias("hashmap", HashMap.class);
    registerAlias("list", List.class);
    registerAlias("arraylist", ArrayList.class);
    registerAlias("collection", Collection.class);
    registerAlias("iterator", Iterator.class);

    registerAlias("ResultSet", ResultSet.class);
  }
```

- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37

- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45
- 46
- 47
- 48
- 49
- 50
- 51
- 52
- 53
- 54
- 55
- 56
- 57
- 58
- 59

上面是 Mybatis 系统已经定义好的，不需要我们重复定义，直接使用即可。

但常常使用开发都自定义的别名。

四.二 自定义别名

如 我们前面所使用的 com.yjl.pojo.User 类，当多次使用时，如 insert ，update 和 select 时，不必要重复性定义，只定义别名即可 user .

```
<select id="getById"  parameterType="int" resultType="user">
                select * from user where id=#{id}
        </select>
```

只需要写成 resultType= “user” 即可，不需要 resultType= “com.yjl.pojo.User”

四.三 XML 形式定义别名

1. 可以单个类定义

```
<!-- 配置别名 -->
        <typeAliases>
                <!--type 指类的全限定名称, alias 为 别名-->
                <typeAlias type="com.yjl.pojo.User" alias="user"/>
        </typeAliases>
```

2.如果类过多的话，那么可以用包。

```
<!-- 配置别名 -->
        <typeAliases>
                <!-- 定义包的形式 ,可以多个-->
                <package name="com.yjl.pojo"/>
                <package name="com.yjl.pojo2"/>
        </typeAliases>
```

四.四 注解定义 @Alias

```
@Alias(value ="user")
public class User {

}
```

这样就可以直接使用 user 别名了。

五. 环境设置 environments

```
<!-- 开发环境  development 环境为开发环境 -->
        <environments default="development">
                <environment id="development">
                        <!-- 事务管理 -->
                        <transactionManager type="JDBC"></transactionManager>
```

```
                    <!-- 数据源 ， 为 pooled 连接池 -->
                    <dataSource type="pooled">
                            <!-- 定义数据库连接属性-->
                            <property name="driver" value="${jdbc.driver}"/>
                            <property name="url" value="${jdbc.url}"/>
                            <property name="username"
value="${jdbc.username}"/>
                            <property name="password"
value="${jdbc.password}"/>
                    </dataSource>
            </environment>
        </environments>
```

环境 default 默认为 development 开发环境。 其中, id 为默认的开发环境。

transactionManager 事务管理的类型有三种:

1. jdbc 使用 jdbc 的形式管理事务，在独立编码，不引用其他数据库框架时使用
2. managed 采用容器方式管理事务，在 JNDI 中常常使用。
3. 自定义 使用者自己定义，常常用于特殊的环境中。

datasource 的类型有四种:

1. pooled 连接池数据库
2. unpooled 非连接池数据库
3. JNDI JNDI 数据源
4. 自定义数据源

六 引入映射文件 mapper

六.一 文件路径方式

```
<mappers>
        <!-- 引入文件资源,可依次写多个-->
        <mapper resource="com/yjl/mapper/UserMapper.xml"/>
</mappers>
```

采用的是 文件路径 / 的方式。

六.二 包名引入 (多个时)

```
<mappers>
        <!-- 包名引入，用. -->
        <package name="com.yjl.mapper"/>
    </mappers>
```

六.三 类注册引入

```
<mappers>
        <!--类注册引入，用 class 属性 -->
        <mapper class="com.yjl.mapper.UserMapper"/>
    </mappers>
```

建议使用 package，可引入包。