mybatis_day01

一、简介

1.概述

MyBatis 本是apache的一个开源项目iBatis, 2010年这个项目由apache software foundation 迁移到了google code，并且改名为MyBatis 。2013年11月迁移到Github

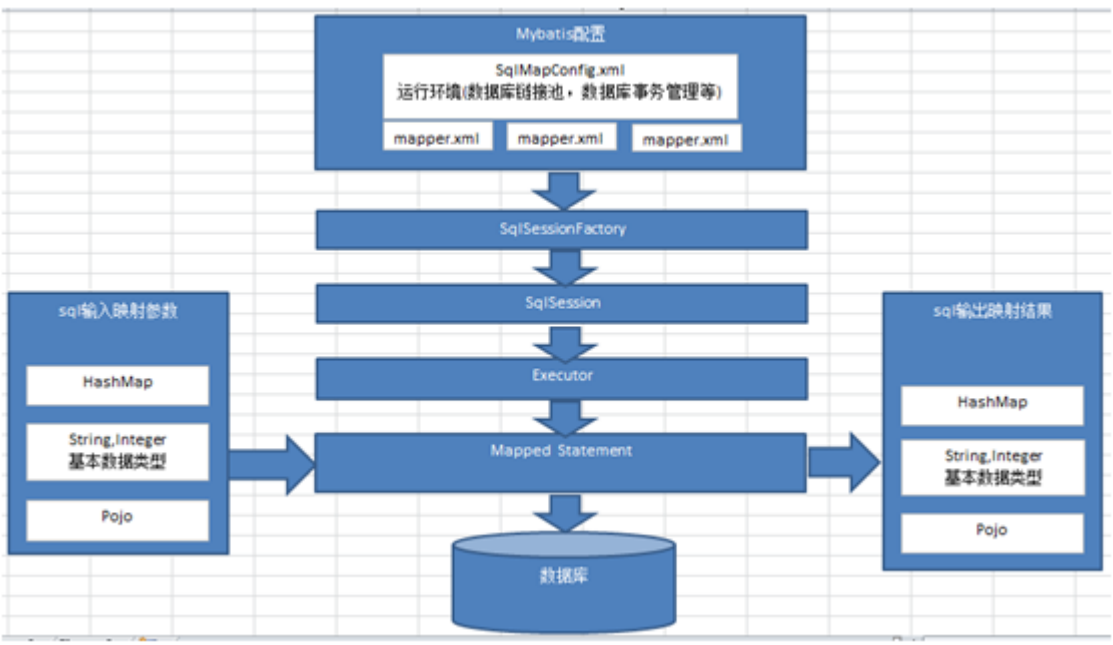
MyBatis 是支持普通 SQL查询，存储过程和高级映射的优秀持久层框架。ORM框架(Hibernate也是orm框架)，MyBatis 消除了几乎所有的JDBC代码和参数的手工设置以及结果集的检索。

MyBatis 使用简单的 XML或注解用于配置和原始映射，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；

2.作用

```
-MyBatis 是一个ORM框架 对象关系映射（Object Relational Mapping），是一个半自动化的ORM框架，
(Hibernate也是全自动化的orm框架)：
-易于学习，几乎消除了所有的JDBC代码；
-原生sql存在Xml文件中，便于管理；
-解除sql与程序代码的耦合；
-支持对象关系映射；
-支持编写动态sql，比如一些条件查询；
-对JDBC进行封装。
```

3.架构



1、 mybatis配置
SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。
mapper.xml文件即sql映射文件，配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。

2、 通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂

3、 由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。

4、 mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。

5、 Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。

6、 Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。

7、 Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。

4.下载使用

> mybaits的代码由github.com管理，

地址：https://github.com/mybatis/mybatis-3/releases

二、入门程序

基本步骤:
1.创建项目导入相关的jar包（核心包，依赖包，数据库驱动包）
2.创建mybatis核心的xml配置文件
3.单个的sqlmapper的xml文件，必须被核心文件加载
4.创建pojo的java对象和sql语句
5.测试程序

1.步骤一

创建项目，并导入相关的jar包

mybaits核心包；

数据库启动包;

```
        <!-- 添加mybatis核心配置文件 -->
        <dependency>
```

```
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.2.7</version>
        </dependency>


        <!-- 添加数据库驱动包 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.39</version>
        </dependency>

         <!-- junit单元测试 -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

2.步骤二

在resources目录下创建mybaits核心配置文件。

SqlMapConfig.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>
    <!-- 和spring整合后 environments配置将废除 -->
    <environments default="development">
        <environment id="development">
            <!-- 使用jdbc事务管理 或者JTA事务管理 -->
            <transactionManager type="JDBC" />
            <!-- 数据库连接池 第三方组件：c3p0 -->
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://localhost:3306/bj1801" />
                <property name="username" value="root" />
                <property name="password" value="root" />
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="UserMapper.xml" />
    </mappers>
</configuration>
```

注：重点关注<dataSource type="POOLED">的type属性、其有三种取值：

- POOLED：使用Mybatis自带的数据库连接池来管理数据库连接
- UNPOOLED：不使用任何数据库连接池来管理数据库连接
- JNDI：jndi形式使用数据库连接、主要用于项目正常使用的时候

3.步骤三

创建对应的mapper文件，用来存放对应的sql语句

a.创建Users.xml文件

```
<?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="test">
 <!--sql语句 -->
</mapper>
```

namespace ：命名空间(可以理解为java中package包，用来区分对象)，用于隔离sql语句

在Users.xml中添加：半自动化，自己主要负责的是具体的sql的编写，添加到mapper中

b.编写对应的sql语句

```
<?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="test">
    <!--sql语句 -->
    <!--sql语句 -->
    <!-- 查询所有的用户 -->
    <select id="findUsers" resultType="com.hzit.bean.Users">
        select * from users
    </select>
    <!-- 根据id获取用户信息 -->
    <select id="findUserById" parameterType="int" resultType="com.hzit.bean.Users">
        select
        * from users where userid = #{userid}
    </select>
    <!-- 自定义条件查询用户列表 -->
    <select id="findUserByUsername" parameterType="java.lang.String"
        resultType="com.hzit.bean.Users">
        select * from users where userName like '%${value}%'
    </select>
</mapper>
```

parameterType：定义输入(参数类型)到sql中的映射类型，

#{id}表示使用preparedstatement设置占位符号并将输入变量id传到sql。

resultType：定义结果(返回值)映射类型。

c.将Users.xml添加SqlMapConfig.xml

mybatis框架需要加载映射文件，将Users.xml添加在SqlMapConfig.xml，如下：

```
<mappers>
    <mapper resource="UserMapper.xml" />
</mappers>
```

该节点放在<environments>节点后面，和该节点是同辈节点。

4.步骤四

a.sql语句

```
CREATE TABLE `users` (
  `userId` int(11) NOT NULL AUTO_INCREMENT,
  `userName` varchar(255) DEFAULT NULL,
  `userPwd` varchar(255) DEFAULT NULL,
  `userAge` int(11) DEFAULT NULL,
  `userSex` varchar(255) DEFAULT NULL,
  `userAddr` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`userId`)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8;
```

b.实体类

```
public class Users {
    private Integer userId;// ID数据库自增
    private String userName;
    private String userPwd;
    private Integer userAge;
    private String userSex;
    private String addr;

    //get and set
}
```

5.步骤五

测试

```
@Test
```

```
    public void test1() throws Exception {
        // 会话工厂
        SqlSessionFactory sqlSessionFactory = null;
        // 配置文件
        String resource = "SqlMapConfig.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        // 使用SqlSessionFactoryBuilder从xml配置文件中创建SqlSessionFactory
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        SqlSession sqlSession = null;
        try {
            // 创建数据库会话实例sqlSession
            sqlSession = sqlSessionFactory.openSession();
            // 查询单个记录，根据用户id查询用户信息
            Users user = sqlSession.selectOne("test.findUserById", 3);
            // 输出用户信息
            System.out.println(user);
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (sqlSession != null) {
                sqlSession.close();
            }
        }
    }
```

输出结果:

```
Users [userId=3, userName=小花, userPwd=123, userAge=12, userSex=m, addr=null]
```

常见参数
#{}表示一个占位符号，通过#{}可以实现preparedStatement向占位符中设置值。
    自动进行java类型和jdbc类型转换；
    #{}可以有效防止sql注入。
    #{}可以接收简单类型值或pojo属性值。
    如果parameterType传输单个简单类型值，#{}括号中可以是value或其它名称。

${}表示拼接sql串，通过${}可以将parameterType 传入的内容拼接在sql中且不进行jdbc类型转换，
    ${}可以接收简单类型值或pojo属性值，
    如果parameterType传输单个简单类型值，${}括号中只能是value。

parameterType：指定输入参数类型，mybatis通过ognl从输入对象中获取参数值拼接在sql中。
resultType：指定输出结果类型，mybatis将sql查询结果的一行记录数据映射为resultType指定类型的对象。

selectOne:查询一条记录，如果使用selectOne查询多条记录则抛出异常：
org.apache.ibatis.exceptions.TooManyResultsException: Expected one result (or null) to be returned by selectOne(), but found: 3
    at org.apache.ibatis.session.defaults.DefaultSqlSession.selectOne(DefaultSqlSession.java:70)

 selectList:可以查询一条或多条记录。

三、操作数据

1.为了方便演示使用junit进行测试。首先第一步需要导入junit相对应的jar包。上面已经导入

2.mybaits所有的操作都需要得到SqlSession对象。

```
    SqlSession sqlSession = null; //全局变量
    @Before
    public void init() throws Exception {
        // 会话工厂
        SqlSessionFactory sqlSessionFactory = null;
        // 配置文件
        String resource = "SqlMapConfig.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        // 使用SqlSessionFactoryBuilder从xml配置文件中创建SqlSessionFactory
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        // 创建数据库会话实例sqlSession
        sqlSession = sqlSessionFactory.openSession();
    }
```

@Before注解，表示每次@Test 之前都会调用该方法，这么表示每次都会去创建 sqlSession，所有后面可以直接使用

1.删除数据

a.在UserMapper.xml中添加sql语句：

```
<!-- 删除用户 -->
<delete id="deleteUsersById" parameterType="int">
    delete from Users where userid = #{userid}
</delete>
```

b.测试类

```
//删除
@Test
public void test01() {
    //test.deleteUsersById 调用对应sql   13:实参
    int row = sqlSession.delete("test.deleteUsersById", 13);
    System.out.println(row);
    sqlSession.commit();// 增删改需要提交事务
}
```

2.修改数据

a.sql语句

```
<!-- 更改用户 -->
    <update id="updateUser" parameterType="com.hzit.bean.Users">
        update Users set
        userName=#{userName},userPwd=#{userPwd},userAge=#{userAge},userSex=#
{userSex},userAddr=#{addr}
        where
        userId=#{userId}
    </update>
```

b.测试语句

```
// 修改
    @Test
    public void test02() {

        // 可以去获取一个对象 也可以new一个新对象
        Users users = sqlSession.selectOne("test.findUserById", 4);
        System.out.println("修改之前:" + users);

        // 修改
        users.setUserName("赵子龙他哥");
        sqlSession.update("test.updateUser", users);
        sqlSession.commit();

        Users newUser = sqlSession.selectOne("test.findUserById", 4);
        System.out.println("修改之后:" + newUser);

    }
```

c.结果

```
修改之前:Users [userId=4, userName=赵子龙, userPwd=123, userAge=88, userSex=m, addr=null]
修改之后:Users [userId=4, userName=赵子龙他哥, userPwd=123, userAge=88, userSex=m, addr=null]
```

3.添加数据

3.1普通添加(不考虑主键)

a.sql语句

```
<!-- 1. 添加普通对象，不考虑id自动增长: -->
    <insert id="addUser" parameterType="com.hzit.bean.Users">
        insert into Users
        values(#{userId},#{userName},#{userPwd},#{userAge},#{userSex},#{addr})
    </insert>
```

b.测试

```
// 添加1
    @Test
    public void test03() {

        Users users = new Users();
        users.setUserId(1001);
        users.setUserName("来耍");
        users.setUserPwd("123");
        users.setUserSex("m");
        users.setUserAge(20);
        users.setAddr("广东深圳");

        int row = sqlSession.insert("test.addUser", users);
        System.out.println(row);
        sqlSession.commit();
    }
```

c.结果

1001	来福	123	20	m	广东深圳

3.2使用数据库自增主键

a.sql语句

```
    <!-- 2. 添加普通对象，使用mysql自动增长: -->
    <insert id="addUserByKey" parameterType="com.hzit.bean.Users">
        insert into
        Users(userName,userPwd,userAge,userSex,userAddr)
        values(#{userName},#{userPwd},#{userAge},#{userSex},#{addr})
    </insert>
```

b.测试

```
@Test
    public void test04() {
        Users users = new Users();
        users.setUserName("旺财");
        users.setUserPwd("123");
        users.setUserSex("m");
        users.setUserAge(20);
        users.setAddr("广东深圳");

        int row = sqlSession.insert("test.addUserByKey", users);
        System.out.println(row);
        sqlSession.commit();
    }
```

3.3使用程序自增主键

a.sql语句

```
<!-- 3.通过程序生成key,支持数据库自增，也支持Oracle序列 -->
    <insert id="addUserByGenKey" parameterType="com.hzit.bean.Users">
        <selectKey keyProperty="userId" resultType="int" order="BEFORE">
            SELECT
            LAST_INSERT_ID()
        </selectKey>

        insert into
        Users(userId,userName,userPwd,userAge,userSex,userAddr)
        values(#{userId},#{userName},#{userPwd},#{userAge},#{userSex},#{addr})
    </insert>
```

b.测试

```
@Test
    public void test05() {
        Users users = new Users();
        users.setUserName("赵大锤");
        users.setUserPwd("123");
        users.setUserSex("m");
        users.setUserAge(20);
        users.setAddr("广东深圳");

        int row = sqlSession.insert("test.addUserByGenKey", users);
        System.out.println(row);
        sqlSession.commit();
    }
```

3.4相关参数

> SelectKey在Mybatis中为了解决Insert数据时不支持主键自动生成的问题，他可以随意的设置生成主键的方式

属性	描述
keyProperty	selectKey 语句结果应该被设置的目标属性
resultType	结果的类型。MyBatis 通常可以算出来,但是写上也没有. MyBatis 允许任何简单类型用作主键的类型,包括字符串
statementType	和前面的相 同,MyBatis 支持 STATEMENT ,PREPARED 和 CALLABLE 语句的映射类型,分别代表 PreparedStatement 和CallableStatement 类型
SelectKey	需要注意order属性，像Mysql一类支持自动增长类型的数据库中， order需要设置为after才会取到正确的值。 像Oracle这样取序列的情况，需要设置为before，否则会报错 >

四、dao层Mapper动态代理开发

以上是通过手动的去调用对应的方法，例如selectOne(),selectList()等等。这种方式需要自己去实现代码。

Mapper接口开发方法只需要程序员编写Mapper接口（相当于Dao接口），由Mybatis框架根据接口定义创建接口的动态代理对象

Mapper接口开发需要遵循以下规范：

1、 Mapper.xml文件中的namespace与mapper接口的类路径相同。
2、 Mapper接口方法名和Mapper.xml中定义的每个statement的id相同
3、 Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同
4、 Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的resultType的类型相同

4.1定义dao层接口

```java
public interface UsersMapp {

    // 通过Id查找
    public Users finUserByid(int uId) throws Exception;

    // 查询所有users
    public List<Users> findUserList() throws Exception;

    // 通过名称模糊查询 users
    public List<Users> findUserByUsername(String uName) throws Exception;

    // 添加
    public int addUsers(Users users) throws Exception;

}
```

4.2定义对应sql语句

```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="com.hzit.mapper.UsersMapp">

    <!-- 通过指定ID查找对应数据 -->
    <select id="finUserByid" parameterType="int" resultType="com.hzit.bean.Users">
        select
        uId,uName,uPwd,phone uPhone,address
        from users where uId=#{uId}
    </select>

    <!-- 查询所有的数据 -->
    <select id="findUserList" resultType="com.hzit.bean.Users">
        select * from users
    </select>

    <!-- 自定义条件查询用户列表 -->
    <select id="findUserByUsername" parameterType="java.lang.String"
        resultType="com.hzit.bean.Users">
        select * from users where uName like '%${value}%'
    </select>

    <!-- 添加 使用mysql数据库自己维护主键 useGeneratedKeys:true 表示开启自动生成 keyProperty:主键属性值 -->
    <insert id="addUsers" parameterType="com.hzit.bean.Users" useGeneratedKeys="true"
        keyProperty="uId">
        insert into users(uName,uPwd,phone,address)
        values(#{uName},#{uPwd},#{uPhone},#{address})
    </insert>
</mapper>
```

4.3测试代码

```java
public static void main(String[] args) throws Exception {
        // 会话工厂
        SqlSessionFactory sqlSessionFactory = null;
        // 配置文件
        String resource = "SqlMapConfig.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        SqlSession sqlSession = null;
        sqlSession = sqlSessionFactory.openSession(true);
        // 获取 mapper代理对象
        UsersMapp mapper = sqlSession.getMapper(UsersMapp.class);

        System.out.println(">>>获取所有信息");
        List<Users> userList = mapper.findUserList();
        System.out.println(userList);
```

```
        System.out.println(">>>通过ID获取信息");
        Users users = mapper.finUserByid(105);
        System.out.println(users);

        System.out.println(">>>通过uName模糊查询");
        List<Users> findUserByUsername = mapper.findUserByUsername("zhang");
        System.out.println(findUserByUsername);

        System.out.println(">>>添加数据,ID自动增长");
        Users users2 = new Users();
        users2.setuName("hello");
        users2.setuPwd("8888");
        users2.setuPhone("1881123");
        users2.setAddress("china");
        int addUsers = mapper.addUsers(users2);
        System.out.println(addUsers);

    }
```

以上代码必须符号以上四种规则，否则会报错。

五、log4j日志文件

Mybatis使用log4j作为默认的日志。可以在项目中添加log4j信息

a. 第一步导入相应的log4j需要的jar包信息

```
<!-- log4j日志文件 -->
       <dependency>
           <groupId>log4j</groupId>
           <artifactId>log4j</artifactId>
           <version>1.2.17</version>
       </dependency>
```

b. 第二步，创建log4j.properties配置文件，

注意：名称不能改变，必须log4j.properties

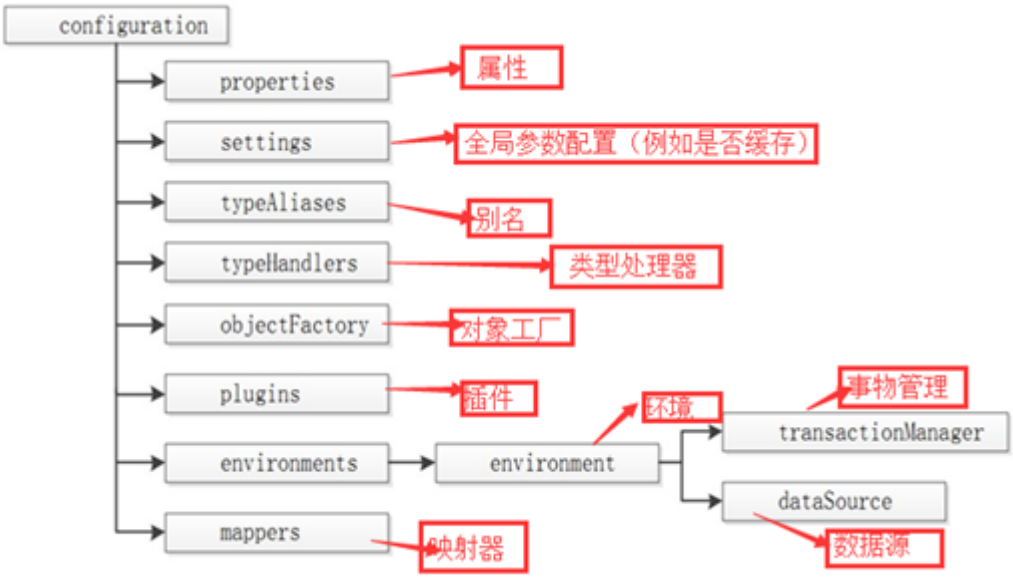
必须放在resource目录下面

log4j.properties配置文件信息：

```
# Global logging configuration 开发时候建议使用 debug
log4j.rootLogger=DEBUG, stdout
# Console output...
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

六、配置文件详解

1.SqlMapConfig.xml配置文件



1.1properties（属性）

SqlMapConfig.xml可以引用java属性文件中的配置信息如下,在resource下定义db.properties文件.

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/bj1801?useUnicode=true&characterEncoding=utf8
jdbc.username=root
jdbc.password=root
```

以上定义了文件内容，可以直接在配置文件中进行引用。

SqlMapConfig.xml引用如下：

```
<propertiesresource="db.properties"/>
<environmentsdefault="development">
    <environmentid="development">
        <transactionManagertype="JDBC"/>
        <dataSourcetype="POOLED">
            <propertyname="driver"value="${jdbc.driver}"/>
            <propertyname="url"value="${jdbc.url}"/>
            <propertyname="username"value="${jdbc.username}"/>
            <propertyname="password"value="${jdbc.password}"/>
        </dataSource>
    </environment>
</environments>
```

1.2settings（配置）

mybatis全局配置参数，全局参数将会影响mybatis的运行行为。

Setting(设置)	Description（描述）	Valid Values(验证值组)	Default(默认值)
cacheEnabled	在全局范围内启用或禁用缓存配置任何映射器在此配置下。	true | false	TRUE
lazyLoadingEnabled	在全局范围内启用或禁用延迟加载。禁用时，所有协会将热加载。	true | false	TRUE
aggressiveLazyLoading	启用时，有延迟加载属性的对象将被完全加载后调用懒惰的任何属性。否则，每一个属性是按需加载。	true | false	TRUE
multipleResultSetsEnabled	允许或不允许从一个单独的语句（需要兼容的驱动程序）要返回多个结果集。	true | false	TRUE
useColumnLabel	使用列标签，而不是列名。在这方面，不同的驱动有不同的行为。参考驱动文档或测试两种方法来决定你的驱动程序的行为如何。	true | false	TRUE
useGeneratedKeys	允许JDBC支持生成的密钥。兼容的驱动程序是必需的。此设置强制生成的键被使用，如果设置为true，一些驱动会不兼容性，但仍然可以工作。	true | false	FALSE
autoMappingBehavior	指定MyBatis的应如何自动映射列到字段/属性。NONE自动映射。PARTIAL只会自动映射结果没有嵌套结果映射定义里面。 FULL会自动映射的结果映射任何复杂的（包含嵌套或其他）。	NONE, PARTIAL, FULL	PARTIAL
defaultExecutorType	配置默认执行人。SIMPLE执行人确实没有什么特别的。 REUSE执行器重用准备好的语句。 BATCH执行器重用语句和批处理更新。	SIMPLE REUSE BATCH	SIMPLE
defaultStatementTimeout	设置驱动程序等待一个数据库响应的秒数。	Any positive integer	Not Set (null)
safeRowBoundsEnabled	允许使用嵌套的语句RowBounds。	true | false	FALSE
mapUnderscoreToCamelCase	从经典的数据库列名A_COLUMN启用自动映射到骆驼标识的经典的Java属性名aColumn。	true | false	FALSE
localCacheScope	MyBatis的使用本地缓存，以防止循环引用，并加快反复嵌套查询。默认情况下（SESSION）会话期间执行的所有查询缓存。如果localCacheScope=STATMENT本地会话将被用于语句的执行，只是没有将数据共享之间的两个不同的调用相同的SqlSession。	SESSION | STATEMENT	SESSION
dbcTypeForNull	指定为空值时，没有特定的JDBC类型的参数的JDBC类型。有些驱动需要指定列的JDBC类型，但其他像NULL，VARCHAR或OTHER的工作与通用值。	JdbcType enumeration. Most common are: NULL, VARCHAR and OTHER	OTHER

lazyLoadTriggerMethods	指定触发延迟加载的对象的方法。	A method name list separated by commas	equals, clone, hash Code, toString
defaultScriptingLanguage	指定所使用的语言默认为动态SQL生成。	A type alias or fully qualified class name.	org.apache.ibatis.scripting.xmltags.XMLDynamicLanguageDriver
callSettersOnNulls	指定如果setter方法或地图的put方法时，将调用检索到的值是null。它是有用的，当你依靠Map.keySet（）或null初始化。注意原语（如整型，布尔等）不会被设置为null。	true | false	FALSE
logPrefix	指定的前缀字串，MyBatis将会增加记录器的名称。	Any String	Not set
logImpl	指定MyBatis的日志实现使用。如果此设置是不存在的记录的实施将自动查找。	SLF4J | LOG4J | LOG4J2 | JDK_LOGGING | COMMONS_LOGGING | STDOUT_LOGGING |	Not set
proxyFactory	指定代理工具，MyBatis将会使用创建懒加载能力的对象。	CGLIB | JAVASSIST	

1.3typeAliases（类型别名）

可以设置单个类，或者包的别名。设置完之后可以直接使用别名，代替全路径。在SqlMapConfig.xml中配置：

```
<typeAliases>
    <!--单个别名定义 -->
    <typeAliasalias="Users"type="com.hzit.bean.Users"/>
    <!--批量别名定义，扫描整个包下的类，别名为类名（首字母大写或小写都可以） -->
    <packagename=" com.hzit.bean"/>
</typeAliases>
```

1.4mappers（映射器）

Mapper配置的几种方法：

a.resource方式<mapper resource=" " />

使用相对于类路径的资源

```
如：<mapper resource="sqlmap/User.xml" />
```

b.url方式<mapper url=" " \/>

使用完全限定路径

```
如：<mapper url="file:///D:\workspace_spingmvc\mybatis_01\config\sqlmap\User.xml" />
```

c.class方式<mapper class=" " />

使用mapper接口类路径

```
如：<mapper class="com.hzit.dao.UserMapper"/>
```

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

d.package 方式<package name=""/>

注册指定包下的所有mapper接口

```
<package name="com.hzit.dao"/>
```

注意：此种方法要求mapper接口名称和mapper映射文件名称相同，且放在同一个目录中。

2.xxxMapper.xml配置文件

SQL 映射文件结构：

```
-cache -  配置给定命名空间的缓存。
-cache-ref -  从其他命名空间引用缓存配置。
-resultMap  -  最复杂，也是最有力量的元素，用来描述如何从数据库结果集中来加载对象。（重点）
-sql -  可以重用的 SQL 块，也可以被其他语句引用。
-insert -  映射插入语句
-update -  映射更新语句
-delete -  映射删除语句
-select -  映射查询语-
```

2.1Select中的属性

属性	描述
id	在命名空间中唯一的标识符，可以被用于引用 该语句。
parameterType	将会传入该语句的参数类的完全限定名或别名。
resultType	从该语句中返回的期望类型的类的完全限定名 或别名。
resultMap	命名引用外部的resultMap。
flushCache	将其设置为true，无论语句什么时候被调用， 都会导致缓存被清空。默认值为false
useCache	将其设置为true，将会导致本条语句的结果 被缓存。默认值为true。
timeout	该设置驱动程序等待数据库返回请求结果。
fetchSize	暗示驱动程序每次批量返回的结果行数。 默认不设置（驱动自行处理）
statementType	STATEMENT、PREPARED或CALLABLE的 一种
resultSetType	FORWARD_ONLY|SCROLL_SENSITIVE| SCROLL_INSENSITIVE中的一种。默认不设置 （驱动自行处理）。

2.2 insert、update和delete

属性	描述
id	在命名空间中唯一的标识符，可以被用于引用该语句。
parameterType	将会传入该语句的参数类的完全限定名或别名。
flushCache	将其设置为true，无论语句什么时候被调用，都会 导致缓存被清空。默认值为false。
timeout	该设置驱动程序等待数据库返回请求结果，并抛出 异常时间的最大等待值。默认不设置（驱动自行处理）。
statementType	STATEMENT、PREPARED或CALLABLE的一种。 用于方便MyBatis选择使用Statement、 PreparedStatement或CallableStatement。 默认值为PREPARED
useGeneratedKeys	（仅对insert有用）通知MyBatis使用JDBC的 getGeneratedKeys方法来取出由数据（如MySQL 和SQL Server的数据库管理系统的自动递增字段） 内部生成的主键。默认值为false。
keyProperty	（仅对insert有用）标记一个属性，MyBatis会通过 getGeneratedKeys或insert语句的selectKey子元素设 置其值。默认不设置。