1、properties标签

```
<!--
        mybatis可以使用properties标签来引入外部properties文件内容
            1、resource：引入类路径下文件
            2、url：引入网络路径或者磁盘路径下的文件
     -->
    <properties resource="jdbc.properties"></properties>
```

2、settings标签

```
<!--
        settings包含很多重要的设置项
            setting用来设置每一个设置项
                name：要设置的设置项名称
                value：要设置的setting值
     -->
     <settings>
     <!-- 以开启驼峰命名为示例 -->
        <setting name="mapUnderscoreToCamelCase" value="true"/>
     </settings>
```

settings设置

- 这是 MyBatis 中极为重要的调整设置，它们会改变 MyBatis 的运行时行为。

设置参数	描述	有效值	默认值
cacheEnabled	该配置影响的所有映射器中配置的**缓存**的全局开关。	true | false	TRUE
lazyLoadingEnabled	延迟加载的全局开关。**当开启时，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
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
defaultStatementTimeout	**设置超时时间**，它决定驱动等待数据库响应的秒数。	Any positive integer	Not Set (null)
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

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

3、typeAliases起别名

单独为某个类起别名

```
<!-- 设置起别名 -->
    <!--
        typeAliases：为某个java类起别名
            type：是要起别名类的全类名，默认是类名的小写
            alias：指定新的别名
     -->
    <typeAliases>
        <typeAlias type="daiwei.test.pojo.User" alias="myUser"/>
    </typeAliases>
```

为某个包批量起别名

```
<!-- 设置起别名 -->
    <typeAliases>
        <package name="daiwei.test.pojo"/>
    </typeAliases>
    <!--
```

```
        为某个包下的类批量起别名
            name：  为name的包下的类及其自包起一个默认别名（该类名小写）
     -->
```

通过注解的方式来起别名

```
@Alias("myUser")
public class User {



    private int id;
    private String name;
....
```

一般情况下不推荐起别名，都是直接使用全类名，这样在类比较度偶的情况下方便查看类型。

4、environment 运行环境

在environment里面可以配置多个数据源，根据实际使用需要通过default选择指定的数据源。

```
<environments default="development">
    <!-- environment这里可以配置多个环境，如开发的环境、测试的环境
            id为其的唯一标识符，default为当前指定的数据环境；
            environment里面配置必须要有俩个标签
                transactionManager：事务管理器
                    type：事务管理器类型
JDBC(jdbcTransactionFactory)|MANAGED(managedTransactionFactory)
                        自定义事务管理器：实现transactionFactory接口，type指定为
全类名。
                dataSource:数据源
                    type：数据源类型 UNPOOLED(unpooledDataSourceFactory)
                                    |POOLED(pooledDataSourceFactory)
                                    |JNDI(jndiDataSourceFactory)
                        自定义数据源：  实现DataSourceFactory接口，type指定为实现
类全类名。
    -->
```

```
    <!-- transaction  -->
        <environment id="test">
            <transactionManager type="JDBC"></transactionManager>
            <dataSource type="POOLED">
                <property name="driver" value="${dirver}"/>
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
            </dataSource>
        </environment>
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${driver}" />
                <property name="url" value="${url}" />
                <property name="username" value="${username}" />
                <property name="password" value="${password}" />
            </dataSource>
        </environment>
    </environments>
```

5、datbaseIdProvider

mybatis支持多数据库，在不同的数据库，也是发送不同的sql文件，databaseProvider的具体配置方法（在mybatis-config.xml中）

```

<!-- databaseIdProvider ： 支持多数据库厂商
        type="DB_VENDOR"(vendorDatabaseIdProvider)
            作用就是得到数据库厂商标识（驱动getDatabaseProductName())
            mybatis就能根据不同厂商的标识来执行不同的sql
            MySQL，Oracle，  SQL Server，XXX

    -->

    <databaseIdProvider type="DB_VENDOR">
    <!-- 为每个不同的数据库厂商取别名 -->
        <property name="MySQl" value="mysql"/>
        <property name="Oracle" value="oracle"/>
        <property name="SQL Server" value="sqlserver"/>
    </databaseIdProvider>

```

在mapper.xml中，在具体的sql上配置databaseId（不带标识的默认为加载当前环境下的sql语句，和带了当前环境的databaseId的sql语句）

```
<update id="updateUserById" parameterType="daiwei.test.pojo.User"
databaseId="mysql"><!-- databaseId 数据厂商别名，  -->
        UPDATE `user`
        SET `name` = #{name},
         `age` = #{age},
         `sex` = #{sex}
        WHERE
            `id` = #{id}
    </update>
```

6、mappers标签

```
<!-- mapper标签：注册一个sql映射
            resource：引用类路径（bin目录)下的sql映射文件。
            url：引用网络路径或者磁盘下面的sql映射文件。
            class：1、引用注册接口（需要在同目录下有同名的sql映射xml文件）
                   2、可以基于注解,没有映射文件的方式
                    @Select("select * from user where id = #{id}")
                    public User getUserById(Integer id);
            推荐：
                比较重要的复杂的dao接口使用sql映射文件来写
                不重要，简单的接口可以使用注解的方式快速开发
        package标签：批量注册。统一注册某个包下的的dao接口和其同名sql文件
            注：这个package使用时候要把接口和同名SQL映射文件放在一个目录下
     -->
    <mappers>
        <mapper resource="userMapper.xml" />
        <!-- <mapper class="daiwei.test.myInterface.UserMapper"/> -->
        <!-- <package name="daiwei.test.myInterface"/> -->
    </mappers>
```