**MyBatis**

1.配置maven依赖

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>x.x.x</version>

</dependency>

**从 XML 中构建 SqlSessionFactory**

每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 实例为中心的。SqlSessionFactory 的实例可以通过 SqlSessionFactoryBuilder 获得。而 SqlSessionFactoryBuilder 则可以从 XML 配置文件或一个预先定制的 Configuration 的实例构建出 SqlSessionFactory 的实例。

从 XML 文件中构建 SqlSessionFactory 的实例非常简单，建议使用类路径下的资源文件进行配置。但是也可以使用任意的输入流(InputStream)实例，包括字符串形式的文件路径或者 file:// 的 URL 形式的文件路径来配置。MyBatis 包含一个名叫 Resources 的工具类，它包含一些实用方法，可使从 classpath 或其他位置加载资源文件更加容易。

String resource = "org/mybatis/example/mybatis-config.xml";

InputStream inputStream = Resources.getResourceAsStream(resource);

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

**xml配置文件**

XML 配置文件（configuration XML）中包含了对 MyBatis 系统的核心设置，包含获取数据库连接实例的数据源（DataSource）和决定事务作用域和控制方式的事务管理器（TransactionManager）。XML 配置文件的详细内容后面再探讨，这里先给出一个简单的示例:

<?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>

<environments default="development">

<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>

<mappers>

<mapper resource="org/mybatis/example/BlogMapper.xml"/>

</mappers>

</configuration>

要注意 XML 头部的声明，用来验证 XML 文档正确性。environment 元素体中包含了事务管理和连接池的配置。mappers 元素则是包含一组 mapper 映射器（这些 mapper 的 XML 文件包含了 SQL 代码和映射定义信息）。

**从 SqlSessionFactory 中获取 SqlSession**

**XML配置**

配置结构:

\*configuration 配置

\*properties 属性

\*settings 设置

\*typeAliases 类型别名

\*typeHandlers 类型处理器

\*objectFactory 对象工厂

\*plugins 插件

\*environments 环境

\*environment 环境变量

\*transactionManager 事务管理器

\*dataSource 数据源

\*databaseIdProvider 数据库厂商标识

\*mappers 映射器

1. **properties**

在properies中配置的 k/v会在整个配置文件中全局替代,引用的时候用${username}类似的语法

<properties resource=" config.properties">

<property name="username" value="dev\_user"/>

<property name="password" value="F2Fa3!33TYyg"/>

</properties>

resource属性 另外引入了config.properties配置文件,配置文件中以k=v格式写入一些配置,然后在xml中进而引用.

如果属性在不只一个地方进行了配置，那么 MyBatis 将按照下面的顺序来加载：

1在 properties 元素体内指定的属性首先被读取。

2然后根据 properties 元素中的 resource 属性读取类路径下属性文件或根据 url 属性指定的路径读取属性文件，并覆盖已读取的同名属性。

3最后读取作为方法参数传递的属性，并覆盖已读取的同名属性。

因此，通过方法参数传递的属性具有最高优先级，resource/url 属性中指定的配置文件次之，最低优先级的是 properties 属性中指定的属性。

从MyBatis 3.4.2开始，你可以为占位符指定一个默认值。例如：

<property name="username" value="${username:ut\_user}"/> 这种方式

当然, 这个特性默认是关闭的。如果你想为占位符指定一个默认值， 你应该添加一个指定的属性来开启这个特性。例如：

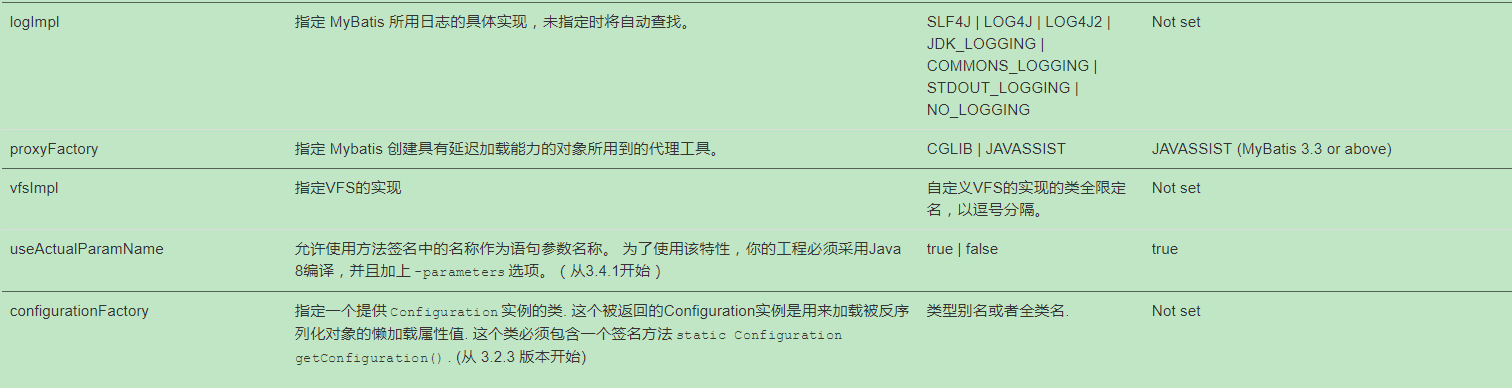
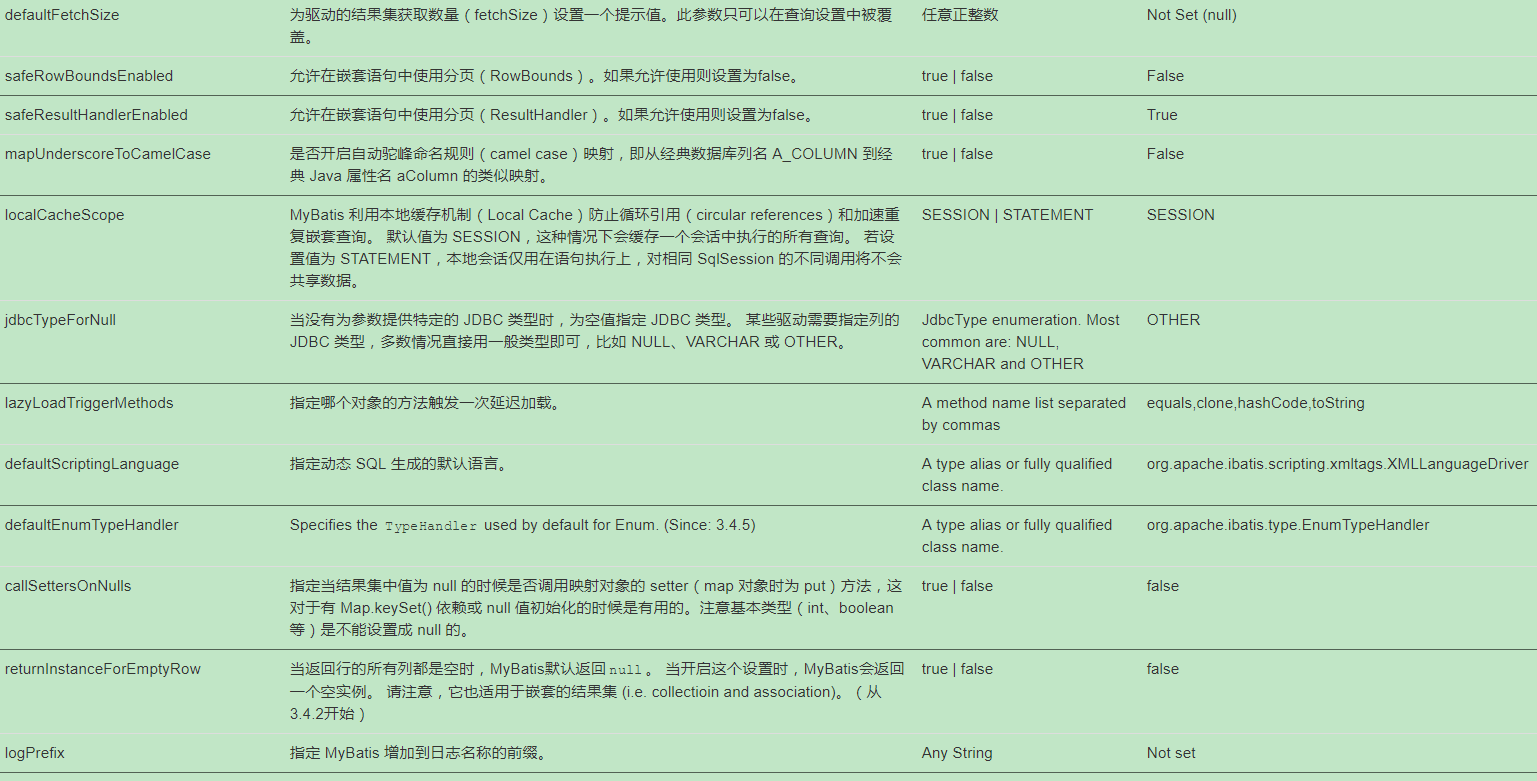
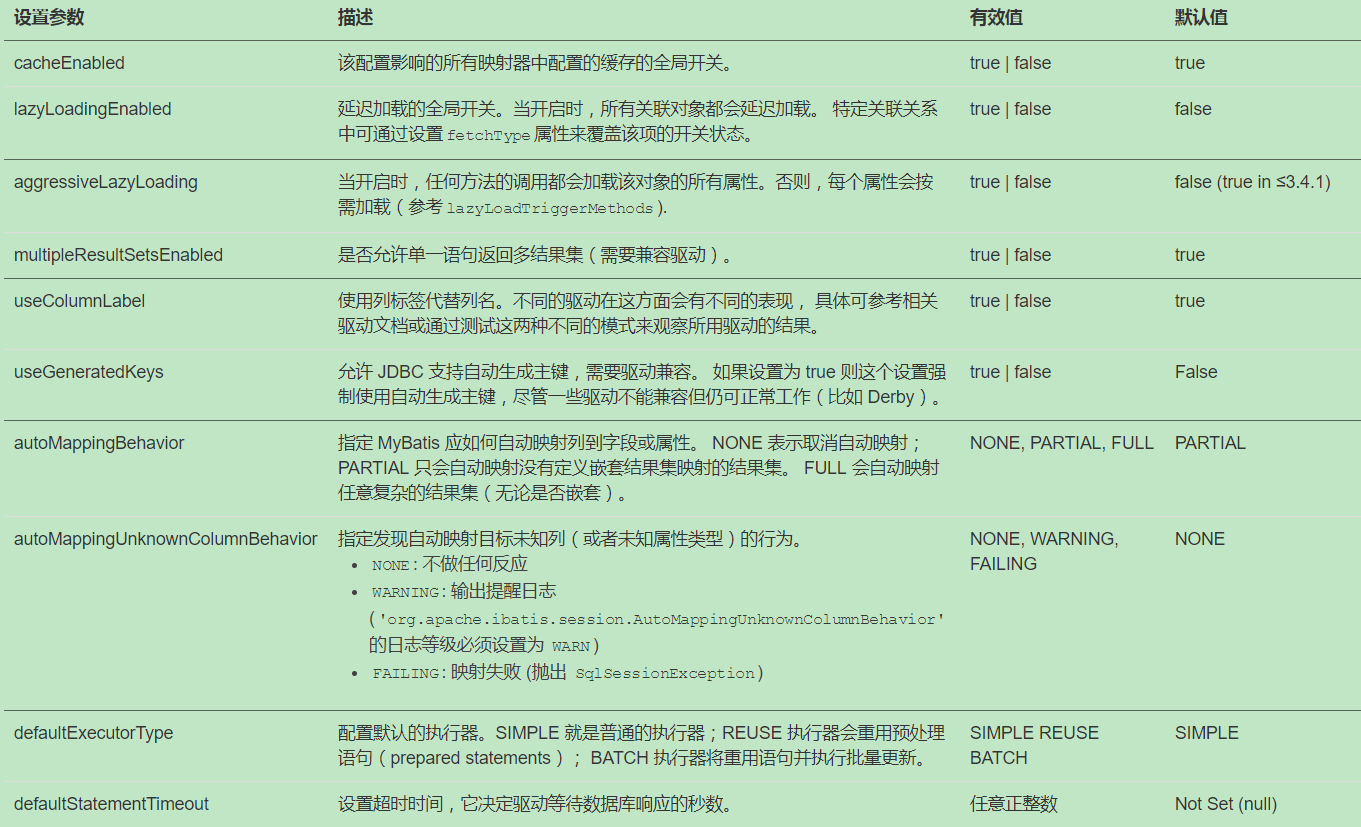
<properties resource="org/mybatis/example/config.properties">

<property name="org.apache.ibatis.parsing.PropertyParser.enable-default-value" value="true"/> <!-- Enable this feature -->

</properties>

1. **settings**

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。下表描述了设置中各项的意图、默认值等。



一个完整的<settings>

<settings>

<setting name="cacheEnabled" value="true"/>

<setting name="lazyLoadingEnabled" value="true"/>

<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="mapUnderscoreToCamelCase" value="false"/>

<setting name="localCacheScope" value="SESSION"/>

<setting name="jdbcTypeForNull" value="OTHER"/>

<setting name="lazyLoadTriggerMethods" value="equals,clone,hashCode,toString"/>

</settings>

1. **配置环境（environments）**

尽管可以配置多个环境，每个 SqlSessionFactory 实例只能选择其一, 所以，如果你想连接两个数据库，就需要创建两个 SqlSessionFactory 实例，每个数据库对应一个。而如果是三个数据库，就需要三个实例，依此类推，记起来很简单。

1. **映射器（mappers）**

我们现在就要定义 SQL 映射语句了。但是首先我们需要告诉 MyBatis 到哪里去找到这些语句

<!-- Using classpath relative resources -->

<mappers>

<mapper resource="org/mybatis/builder/AuthorMapper.xml"/>

<mapper resource="org/mybatis/builder/BlogMapper.xml"/>

<mapper resource="org/mybatis/builder/PostMapper.xml"/>

</mappers>

<!-- Using url fully qualified paths -->

<mappers>

<mapper url="file:///var/mappers/AuthorMapper.xml"/>

<mapper url="file:///var/mappers/BlogMapper.xml"/>

<mapper url="file:///var/mappers/PostMapper.xml"/>

</mappers>

<!-- Using mapper interface classes -->

<mappers>

<mapper class="org.mybatis.builder.AuthorMapper"/>

<mapper class="org.mybatis.builder.BlogMapper"/>

<mapper class="org.mybatis.builder.PostMapper"/>

</mappers>

<!-- Register all interfaces in a package as mappers -->

<mappers>

<package name="org.mybatis.builder"/>

</mappers>

**Mapper XML 文件**

**select**

<select id="selectPerson" parameterType="int" resultType="hashmap">

SELECT \* FROM PERSON WHERE ID = #{id}

</select>

这个语句被称作 selectPerson，接受一个 int（或 Integer）类型的参数，并返回一个 HashMap 类型的对象，其中的键是列名，值便是结果行中的对应值。

#{id}

这就告诉 MyBatis 创建一个预处理语句参数 是preparedStatement语句的封装版本

<select

id="selectPerson"

parameterType="int"

parameterMap="deprecated"

**resultType="hashmap" //返回的类型不仅可以是基本类型 int 等,还可以是 map,或者一个符合规范的javaBean对象 mybatis会自动帮你映射!!!!!**

resultMap="personResultMap"

flushCache="false"

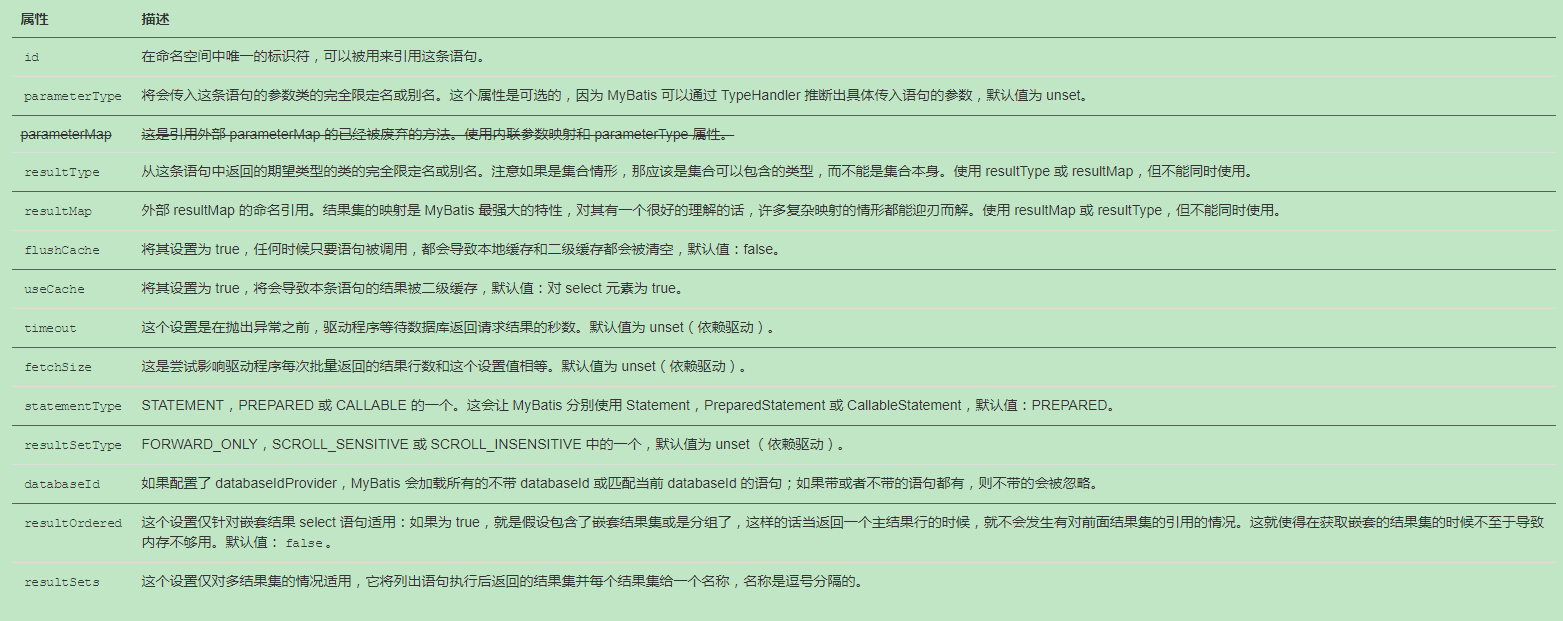
useCache="true"

timeout="10000"

fetchSize="256"

statementType="PREPARED"

resultSetType="FORWARD\_ONLY">



**insert, update 和 delete**

<insert

id="insertAuthor"

parameterType="domain.blog.Author"

flushCache="true"

statementType="PREPARED"

keyProperty=""

keyColumn=""

useGeneratedKeys=""

timeout="20">

<update

id="updateAuthor"

parameterType="domain.blog.Author"

flushCache="true"

statementType="PREPARED"

timeout="20">

<delete

id="deleteAuthor"

parameterType="domain.blog.Author"

flushCache="true"

statementType="PREPARED"

timeout="20">



例子:

<insert id="insertAuthor">

insert into Author (id,username,password,email,bio)

values (#{id},#{username},#{password},#{email},#{bio})

</insert>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<update id="updateAuthor">

update Author set

username = #{username},

password = #{password},

email = #{email},

bio = #{bio}

where id = #{id}

</update>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<delete id="deleteAuthor">

delete from Author where id = #{id}

</delete>

**sql**

这个元素可以被用来定义可重用的 SQL 代码段，可以包含在其他语句中。

<sql id="userColumns"> ${tablename}.id,${ tablename }.name </sql>

这个 SQL 片段可以被包含在其他语句中，例如：

<select id="selectUsers" resultType="map">

select

<include refid="userColumns"><property name=" tablename " value="t1"/></include>,

<include refid="userColumns"><property name=" tablename " value="t2"/></include>

from some\_table t1

cross join some\_table t2

</select>

上面这个条sql就等价于

select

t1.id,t1.name,t2.id,t2.name

from some\_table t1

cross join some\_table t2

属性值可以用于包含的refid属性或者包含的字句里面的属性值，例如：

<sql id="sometable">

${prefix}Table

</sql>

<sql id="someinclude">

from

<include refid="${include\_target}"/>

</sql>

<select id="select" resultType="map">

select

field1, field2, field3

<include refid="someinclude">

<property name="prefix" value="Some"/>

<property name="include\_target" value="sometable"/>

</include>

</select>

更多请见 : **http://www.mybatis.org/mybatis-3/zh/sqlmap-xml.html**