Rays's Note

of

Mybatis

目 录

[1. MyBatis概述 2](#_Toc16604976)

[1.1. 简介 2](#_Toc16604977)

[1.2. Hello World 2](#_Toc16604978)

[1.2.1. xml配置 3](#_Toc16604979)

[1.2.2. 代码配置 4](#_Toc16604980)

[2. XML配置 5](#_Toc16604981)

[2.1. 属性 properties 5](#_Toc16604982)

[2.2. 设置 settings 5](#_Toc16604983)

[2.3. 类型别名 typeAliases 5](#_Toc16604984)

[2.4. 类型处理器 typeHandlers 5](#_Toc16604985)

[2.5. 对象工厂 objectFactory 6](#_Toc16604986)

[2.6. 插件 6](#_Toc16604987)

[2.7. 环境配置 7](#_Toc16604988)

[2.7.1. 事务管理器transactionManager 8](#_Toc16604989)

[2.7.2. 数据源 datasource 8](#_Toc16604990)

[2.8. 数据库厂商标识 databaseIdProvider 8](#_Toc16604991)

[2.9. 映射器 8](#_Toc16604992)

[3. XML映射文件 10](#_Toc16604993)

[3.1. select 10](#_Toc16604994)

[3.2. insert，update和delete 10](#_Toc16604995)

[3.2.1. 主键生成 12](#_Toc16604996)

[3.2.2. 多行插入 12](#_Toc16604997)

[3.2.3. sql片段 13](#_Toc16604998)

[3.3. 参数 13](#_Toc16604999)

[3.4. 结果映射 14](#_Toc16605000)

[3.4.1. 关联映射 association 14](#_Toc16605001)

[3.4.2. 集合 collection 16](#_Toc16605002)

[3.4.3. 鉴别器 17](#_Toc16605003)

[3.5. 自动映射 17](#_Toc16605004)

[4. 缓存 19](#_Toc16605005)

# MyBatis概述

## 简介

MyBatis是一个持久层框架，他使用映射文件来完成ORM映射。MyBatis的全景图如下：



## Hello World

mybatis就只有mybatis-xx.jar一个包，只要将需要的版本的包置于类路径即可，使用maven可以在POM中添加如下信息：

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

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

</dependency>

现在准备一个一张数据表Dog，然后使用mybatise实现一个符合ORM查询方法。

|  |  |
| --- | --- |
| Dog | |
| age | number |
| name | varchar |

和物理库的表对应，我们需要创建一个类Dog

class Dog {

int age;

String name;

// get set method

}

### xml配置

mybatis的核心是SqlSessionFactory，这个类的作用是获取数据库连接对象SqlSession。SqlSession用于执行数据库的增删改查语句，同时处理语句和java对象之间映射关系。为了构建SqlSessionFactory，需要创建配置文件，在mybatis-config.xml配置如下：

<configuration>

<properties resource="datasource.properties">

</properties>

<environments default="dev">

<environment id="dev">

<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="com/ray/xml/DogMapper.xml"/>

</mappers>

</configuration>

environment中配置了数据源的url，用户名，密码等信息。mapper中指定了一个映射文件:

<mapper namespace="com.ray.xml.DogMapper">

<select id="queryDog" resultType="com.ray.Dog">

select \* from dog where id = #{id}

</select>

</mapper>

DogMapper.xml中定义了查询的语句,并且指定了改语句的结果集对应Dog类。代码中我们首先需要根据配置文件初始化SqlSessionFactory:

// 使用配置文件创建 SqlSessionFactory

SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder()

.build(Resources.getResourceAsStream(

"com/ray/xml/mybatis-config.xml"));

然后调用语句查询dog表：

// 获取 session 使用 session操作数据库

try (SqlSession sess = sqlSessionFactory.openSession()) {

// 调用mapper查询

Dog d = sess.selectOne("com.ray.xml.DogMapper.queryDog", 1);

}

直接在main函数中执行上述语句，就能够查询出dog表，并且会自动的将数据字段映射到Dog对象的属性。

### 代码配置

如果不使用xml，我们可以在代码中配置MyBatis的相关配置：

Properties prop = loadProperties("datasource.properties");

// 创建数据源

DataSource ds = new PooledDataSource(

prop.getProperty("driver"),

prop.getProperty("url"),

prop.getProperty("username"),

prop.getProperty("password"));

// 事务管理器

TransactionFactory tsf = new JdbcTransactionFactory();

// 创建配置环境

Environment environment = new Environment("dev", tsf, ds);

// 创建配置对象，并添加映射器

Configuration configuration = new Configuration(environment);

configuration.addMapper(DogMapper.class);

我们在代码中创建的配置对象和xml中的配置是等效的，配置文件使用到了DogMapper，这是一个接口对应DogMapper.xml的配置：

interface DogMapper {

@Select("select \* from dog where id = #{id}")

public Dog selectDog(int i);

}

Select 注解指定的Dog类和Dog表的关联的查询语句。然后我们在程序中可以使用配置对象来创建SqlSessionFactory，进而创建Session来操作数据库。

SqlSessionFactory sqlSessionFactory =

new SqlSessionFactoryBuilder().build(configuration);

try (SqlSession session = sqlSessionFactory.openSession()) {

DogMapper mapper = session.getMapper(DogMapper.class);

Dog d = mapper.selectDog(1);

System.out.println(d.getName() + ", " + d.getAge());

}

这里没有使用语句来操作而是通过创建一个DogMapper的代理对象来完成ORM映射。这样基于xml和代码的基本配置就整理完毕，但是关于mybatis还有很多需要学习。

# 映射

mybatis的核心功能就是映射，也就是ORM解决方案，这种映射定义了程序将java对象保存到数据库时，或者从数据库查询出结果时，java对象的属性和数据库表字段之间的对应关系。

映射关系可以使用xml文件来配置，就是Mapper.xml，也可以创建一个Mapper配置类，使用注解来配置。

如果用xml来配置MyBatis，需要在配置文件中加入如下的配置:

<mappers>

// 加入xml配置的映射文件

<mapper resource="com/ray/entity/DogMapper.xml"/>

// 加入注解配置的映射文件

<mapper class="com.ray.entity.DogMapper">

</mappers>

如果用代码来配置MyBatis，可以如下来配置：

Configuration configuration = new Configuration(environment);

// 加入xml配置的映射文件

configuration.addMapper("com/ray/entity/DogMapper.xml");

// 加入注解配置的映射文件

configuration.addMapper(com.ray.entity.DogMapper.class);

## 配置语句

mybatis操作数据库需要自行编写sql语句，然后将语句配置到映射文件里。增删改都可以归类于写操作，查则属于读操作。因此语句配置可以分为两大类：读操作的select与写操作的insert、update、delete。

### insert、update和delete

如要插入author表，可以编写语句如下

<insert id="insertAuthor" parameterType="com.ray.Author">

insert into author (id, username, realname, create\_date)

values (#{id}, #{username}, #{realname}, #{createDate})

</insert>

id是这条语句的标识，parameterType表示参数类型。如果是在Mapper类中可以配置如下：

interface AuthorMapper {

@insert("insert into author ..")

insertAuthor(Author author)

}

将mapper的xml文件或者Mapper类加入配置，在程序中可以通过SqlSession来调用语句：

// 调用xml语句

session.update("pkg.insertAuthor", author);

// 调用Mapper类的语句

session.getMapper(AuthorMapper.class).save(author);

除了id和parameter之外还有很多可选的属性可以配置语句的具体属性，具体的参见官方文档，这里不再介绍。语句中的#{}类似

### select语句

select语句属于读操作，语句会从数据库得到结果集，因此除了sql，我们还需要配置如果将结果集映射到java属性。mybatis的映射结果有两种：结果集映射、自动映射。

select语句的配置和其他的语句类似，如果要查询author表可以编写语句如下：

<select id="selectSingle" resultTyp="com.ray.Author">

selelct \* from author where id = #{id}

</select>

同样的如果是使用Mapper类，配置如下：

@Select("select \* from author where id = #{id}")

public Author selectSingle(int id);

方法的返回值对应xml配置的resultType，select语句的使用方式如下：

// 调用xml语句

session.select("pkg.selectSingle", id);

// 调用Mapper类的语句

session.getMapper(AuthorMapper.class).selectSingle(id)

这样的简单的select语句mybatis也会创建默认的结果映射，默认的自动映射会在结果集中寻找和java属性名称一样的字段（忽略大小写），然后将值设置到java属性中。除了常规的映射配置之外，select也有许多其他方面的配置，这里不一一介绍。

## 结果映射

结果映射即通过显示的配置，告诉mybatis数据库字段和java属性的对应关系。在xml中可以使用select元素的resultMap属性来引用结果映射，在注解中可以使用@ResultMap()注解来引用配置在xml中的结果映射。

现在有一个问答网站的模型，一个Question可以有多个Reply和一个Author，一个Author可以有多个Question和多个Reply。针对模型来讨论Mybatis的映射。

### 一对一映射

一对一映射是指数据表的一行记录会关联到另一张表的一行记录，在java中表现为一个实体持有另一个实体。比如一个Question持有一个Author。在mybatis中有两种方式来处理这种映射关系。

#### 二次查询

二次查询就是先查询Question，然后根据查询到Question再生成语句查询Author。这种方式表面上看起来增加了数据库的访问，但是我们可以通过配置延迟加载来延迟对Author的表的查询，某些场景下对Author表的查询可能都不会发生。具体配置如下：

<resultMap id="questionResult" type="\*.Question">

<association property="author" javaType="\*.Author"

column="author\_id" select="selectAuthor"/>

</resultMap>

<select id="selectQuestion" resultMap="questionResult">

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

</select>

<!-- AuthorMapper.xml -->

<select id="selectAuthor" resultType="\*.Author">

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

</select>

使用resultMap的association的column和select属性指定了结果字段会映射到另一个表的记录和关联的查询语句，property和javaType属性额外的记录对应实体的java属性和属性的java类型。

使用注解配置的方式如下：

public interface QuestionMapper {

@Select("select \* from question where id = #{id}")

@Results({

@Result(property="author", column="author\_id",

one=@One(fetchType=FetchType.LAZY,

select="\*.AuthorMapper.selectOne"))

})

public Question selectOne(long id);

}

使用result注解表明author属性和author\_id字段的关系，并且指定了二次查询的语句。fetchType属性指定为延迟加载。

#### 关联查询

关联查询是在搜索时使用关联查询语句，同时搜索出Question和Author表的字段，然后使用映射将对应的字段映射到对应的属性。

<select id="selectOneU" resultMap="questionResultU">

SELECT Q.ID AS QUESTION\_ID,

Q.TITLE AS TITLE,

Q.CONTENT AS CONTENT,

Q.CREATE\_DATE AS QUESTION\_CREATE\_DATE,

A.ID AS AUTHOR\_ID,

A.USERNAME AS USERNAME,

A.REALNAME AS REALNAME,

A.CREATE\_DATE AS AUTHOR\_CREATE\_DATE

FROM QUESTION Q

LEFT JOIN AUTHOR A ON A.ID = Q.AUTHOR\_ID

WHERE Q.ID = #{id}

</select>

<resultMap id="questionResultU" type="\*.Question" autoMapping="true">

<id property="id" column="QUESTION\_ID"/>

<result property="createDate" column="QUESTION\_CREATE\_DATE"/>

<association property="author" javaType="\*.Author"

autoMapping="true">

<id property="id" column="AUTHOR\_ID" />

<result property="createDate" column="AUTHOR\_CREATE\_DATE"/>

</association>

</resultMap>

association这里表示映射关联查询,其中没有指定二次查询的语句，只是配置了java属性，然后再元素体内配置了一些字段映射。association也可以使用resultMap属性来再次引用其他的结果映射定义。autoMapping属性指定开启自动映射，其会将字段映射到同名的java属性，不过如果java属性之间有重名可能会映射错乱。

目前联合查询的映射不支持注解方式配置。

### 一对多映射

一对多映射表示数据表的一行记录对应其他表的多行记录，在java中表现为一个实体持有其他实体的列表。比如一个Question持有一个Reply的列表。一对多的映射和一对一的映射的差距不大，也有两种方式。

#### 一对多二次查询

一对多的二次查询配置如下：

# XML映射文件

## 结果映射

### 集合 collection

集合映射用来处理一对多关系（或者多对多），即一个数据一行记录关联到另一张表的多行记录的情况，比如一个博客可能有多条评论。同一对一的关系类似，集合的映射也有不同方式来处理，而且集合的配置方式和关联也及其类似：

1.嵌套查询

现在Blog类中有一个属性List<Post> posts;我们可以编写映射如下：

<resultMap id="blogResult" type="Blog">

<collection property="posts" javaType="ArrayList" column="id"

ofType="Post" select="selectPostsForBlog"/>

</resultMap>

<select id="selectBlog" resultMap="blogResult">

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

</select>

<select id="selectPostsForBlog" resultType="Post">

SELECT \* FROM POST WHERE BLOG\_ID = #{id}

</select>

ofType指定了集合内的元素类型，collection内部的配置元素则指定了集合类型的映射关系。select属性指定了使用的查询语句，然后查询语句种中嵌套定义了二次查询。

2.关联查询

集合关联的查询和普通关联一致，可以编写语句：

<select id="selectBlog" resultMap="blogResult">

select

B.id as blog\_id,

B.title as blog\_title

P.id as post\_id,

P.body as post\_body

from Blog B

left outer join Post P on B.id = P.blog\_id

where B.id = #{id}

</select>

然后编写结果映射如下：

<resultMap id="blogResult" type="Blog">

<id property="id" column="blog\_id" />

<result property="title" column="blog\_title"/>

<collection property="posts" ofType="Post" resultMap="blogPostResult" columnPrefix="post\_"/>

</resultMap>

<resultMap id="blogPostResult" type="Post">

<id property="id" column="id"/>

<result property="subject" column="subject"/>

<result property="body" column="body"/>

</resultMap>

### 鉴别器

<discriminator javaType="int" column="draft">

<case value="1" resultType="DraftPost"/>

</discriminator>

鉴别器类似于java语言中的switch case 语句，可以根据不同值，使用不同的配置。

## 自动映射

MyBatis可以自动处理映射，其会在结果集中配对和属性名一致的列名，忽略大小写。开启mapUnderscoreToCamelCase配置可以自动在驼峰命名分隔和下划线风格之间互相映射。

自动映射有三个等级

NONE - 禁用自动映射

PARTIAL - 对非嵌套的属性进行映射

FULL - 对所有属性自动映射

默认的级别是PARTIAL。

# XML配置

配置文件决定mybatis的行为，下面将介绍MyBatis的一些配置。

## 属性 properties

mybatis的属性可以从属性文件定义也可以在配置文件中直接定义：

<properties resource="config/datasource.properties">

<property name="username" value="root"/>

<property name="password" value="root"/>

</properties>

包含的属性可以在配置文件中用来替换占位符 value = "${username}"，如果多个地方都定义了属性，那么先读取的属性会被后读取的覆盖。属性的加载顺序如下：

1.properties 元素体内指定的属性

2.resource 属性或根据 url 属性中定义的属性文件中的属性

3.作为方法参数传递的属性

属性可以支持默认值的设置，使用配置${username:root}可以使用默认值功能。这些功能是可以配置的：

<property name="...enable-default-value" value="true"/> <!-- 启用默认值特性 -->

<property name="...default-value-separator" value="?:"/> <!-- 修改默认值分隔符 -->

## 设置 settings

设置中定义mybatis运行时的一些行为，具体的配置和行为可以参考文档。

## 类型别名 typeAliases

类型别名是为java类型设置一个短的名字，目的是减少限定名的冗余。例如一个类 com.ray.persistence.bean.Dog，我们可以为其配置别名：

<typeAliases>

<typeAlias alias="Dog" type="com.ray.persistence.bean.Dog"/>

</typeAliases>

这样在任意地方配置时，如果需要使用com.ray…Dog 可以直接使用Dog。

## 类型处理器 typeHandlers

类型处理器的目的是告诉mybatis数据的一种类型和java的类型之间如果互相转换，mybatis自己自然定义了一些列的处理器。但是对于一些非标准的数据库类型或特殊需求可能需要其他的一些转换，可能需要自定义类型处理器来处理。要实现自定义类型处理器可以实现TypeHanler接口或者继承BaseTypehandler类。如：

@MappedJdbcTypes(JdbcType.VARCHAR)

public class RaysTypeHandler extends BaseTypeHandler<String> {

public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType jdbcType) {

ps.setString(i, parameter);

}

public String getNullableResult(ResultSet rs, String columnName) {

return rs.getString(columnName);

}

public String getNullableResult(ResultSet rs, int columnIndex) {

return rs.getString(columnIndex);

}

public String getNullableResult(CallableStatement cs, int idx) {

return cs.getString(idx);

}

}

然后在配置文件中配置typeHandler

<typeHandlers>

<typeHandler handler="com.ray.example.RaysTypeHandler"/>

</typeHandlers>

处理器类上的MappedJdbcTypes告诉mybatis这个处理器用于处理何种数据库类型，而其泛型则表示这个处理器处理的java类型。这些也可以在配置文件的typeHandler中配置，也可以通过注解来配置。

## 对象工厂 objectFactory

MyBatis在从结果集中创建实体对象时，会使用一个对象工厂ObjectFactory来初始化对象。默认的对象工厂通过类的构造器或者指定的初始化方法来初始化对象。可以继承DefaultObjectFactory来定义自己的对象工厂。

## 插件

MyBatis允许在一些方法执行前拦截方法调用，这种拦截可以借助插件来完成。允许拦截的方法包括：

Executor(update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)

ParameterHandler(getParameterObject, setParameters)

ResultSetHandler(handleResultSets, handleOutputParameters)

StatementHandler(prepare, parameterize, batch, update, query)

使用插件的方式是实现Interceptor接口，然后加入配置中。

@Intercepts({@Signature(

type = Executor.class,

method = "query",

args = { MappedStatement.class, Object.class,

RowBounds.class, ResultHandler.class}) })

public class RaysInterceptor implements Interceptor {

@Override

public Object intercept(Invocation invocation) throws Throwable {

String name = invocation.getMethod().getName();

System.out.println("befor interceptor " + name);

return invocation.proceed();

}

}

## 环境配置

如果存在多个环境，比如生产、开发和测试的环境是不同的，那么可以在配置文件中配置多套数据源，在加载配置的时候传入的参数可以选择使用的环境。尽管可以配置多个环境，但是同一个SqlSessionFactory只能使用一种环境的配置，假如程序中需要同时使用多个数据库，那么只能配置多个SqlSessionFactory。配置环境的方式如下：

<environments default="development">

<environment id="development">

<transactionManager type="JDBC">

<property name="..." value="..."/>

</transactionManager>

<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>

然后在代码中使用需要使用某种环境，可以这样来构建SqlSessionFactory:

SqlSessionFactory factory =

SqlSessionFactoryBuilder.build(reader, dev, properties)

其中环境和属性的参数是可选的，如果环境参数缺失则使用默认环境。

### 事务管理器transactionManager

MyBatis中有两种事务管理器，JDBC和MANAGED。

JDBC事务管理器直接使用JDBC的提交和回滚设置。

MANAGED事务管理器从不提交和回滚连接，而是由容器来管理事务。

### 数据源 datasource

datasource元素用来配置JDBC数据源，type属性定义数据源类型，有三种内置的数据库类型UNPOOLED，POOLED，JNDI。

UNPOOLED数据源就是简单得数据源配置，每次请求都会打开和关闭新的连接。

POOLED数据源其实就是配置一个数据源连接池，这样在配置的时候除了数据按本身的属性，如用户名、密码等还要配置连接池的一些属性，如连接池的大小、连接生存时间、连接超时等等。

JNDI数据源用于EJB服务，这里不做介绍。

可以通过实现接口DataSourceFactory或者继承实现了该接口的相应的一些类如UnpooledDataSourceFactory来为一些特殊的数据库添加数据源支持。

## 数据库厂商标识 databaseIdProvider

厂商标识配置的目的是适配不同的数据库之间的方言差异，我们在配置查询语言时可以指定语句的databaseId，这个ID和数据库的元信息中的名称一致（DatabaseMetaData#getDatabaseProductName() ），那么MyBatis就能实现对不同数据库的适配。一般来说，数据库的元信息都比较长，我们可以通过厂商标识配置将长名称配置为段名称，这样在语句中的databaseId就不至于太长。如：

<databaseIdProvider type="DB\_VENDOR">

<property name="SQL Server" value="sqlserver"/>

</databaseIdProvider>

# 缓存

MyBatis的缓存在默认情况只启用本地缓存，即只缓存同一个会话的结果。要启用全局的二级缓存，可以在配置文件中加入配置<cache/>。默认的二级缓存配置效果如下：

所有select语句的结果会被缓存

所有insert、update和delete会刷新缓存

缓存淘汰算法为LFU

缓存不会定时刷新

缓存会保存列表或对象的1024个引用

缓存是线程安全的

缓存只作用于cache标签所在的映射文件中的语句。

这些配置都可以被缓存标签的属性锁修改:

<cache

eviction="FIFO"

flushInterval="60000"

size="512"

readOnly="true"/>