Rays's Note

of

Mybatis

目 录

[1. MyBatis概述 1](#_Toc17468693)

[1.1. 简介 1](#_Toc17468694)

[1.2. Hello World 1](#_Toc17468695)

[1.2.1. xml配置 2](#_Toc17468696)

[1.2.2. 代码配置 3](#_Toc17468697)

[2. 配置 4](#_Toc17468698)

[2.1. XML文件 4](#_Toc17468699)

[2.2. 代码配置 4](#_Toc17468700)

[3. 环境 5](#_Toc17468701)

[3.1. 数据源 datasource 5](#_Toc17468702)

[3.2. 事务管理器 transactionManager 6](#_Toc17468703)

[3.3. 事务隔离 6](#_Toc17468704)

[4. 映射器 7](#_Toc17468705)

[4.1. 映射器配置和加载 7](#_Toc17468706)

[4.2. insert、update和delete 7](#_Toc17468707)

[4.3. 查询映射 8](#_Toc17468708)

[4.3.1. 查询语句 select 8](#_Toc17468709)

[4.3.2. 一对一映射 8](#_Toc17468710)

[4.3.2.1. 二次查询 8](#_Toc17468711)

[4.3.2.2. 关联查询 9](#_Toc17468712)

[4.3.3. 一对多映射 10](#_Toc17468713)

[4.3.3.1. 二次查询 10](#_Toc17468714)

[4.3.3.2. 关联查询 11](#_Toc17468715)

[5. 缓存 12](#_Toc17468716)

# 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和数据交互的的核心类是SqlSessionFactory，这个类可以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。SqlSessionFactory可以从xml构建，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中指定的映射文件配置如下:

**<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,并且指定了该语句的对应Dog类。然后在初始化模块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"));

然后业务模块调用语句查询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);

}

这里直接在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**);

映射也可以代码中配置，这里创建DogMapper接口，添加如下配置：

**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。

# 配置

mybatis位于程序和数据库之间，orm和db操作是其核心功能。配置文件则定义具体的工作细节，如工作的数据库，数据库中的表和java对象的关系，查询是否需要缓存等等。数据库操作的核心类是SqlSessionFactory，我们编写的各种配置被mybatis读取之后都工作在后台，而SqlSessionFactory是程序使用mybatis的入口。

## XML文件

配置文件就是一个xml文件，其结构如下：

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（环境配置）

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）

文件保存为xxx.xml这样的文件，在程序中，我们可以这样使用配置文件来创建SqlSessionFactory：

SqlSessionFactory sqlSessionFactory = **new** SqlSessionFactoryBuilder()

                    .build(Resources.getResourceAsStream(

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

## 代码配置

代码配置不需要使用xml文件，直接在代码中使用各种配置类在构建mybatis，代码配置的方式如下：

// 创建数据源

DataSource ds = **new** PooledDataSource(...);

// 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);

SqlSessionFactory sqlSessionFactory

        = **new** SqlSessionFactoryBuilder().build(configuration);

# 环境

境就是配置数据源和事务管理器，这些配置直接用于SqlSessionFactory来生成SqlSession。mybatis可以配置多个数据源，但是一个SqlSessionFactory只能使用一种环境的配置。如果需要同时使用多个环境，那么需要在程序中创建多个SqlSessionFactory。一个环境对应的SqlSessionFactory对象应该贯穿应用程序的整个生命周期，这种对象只需要被创建一次然后长久存在。

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>**

**<environment** id="xx"**>**...**</environment>**

**</environments>**

代码中使用需要使用某种环境，可以指定具体的环境ID来构建SqlSessionFactory，不指定则会使用默认的环境:

SqlSessionFactory factory =

SqlSessionFactoryBuilder.build(reader, devID, properties)

## 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：简单的数据源配置，每次请求都会打开和关闭新的连接。

POOLED：配置一个数据源连接池，除了数据源本身的属性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：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来配置一些特殊的数据源。

## 事务管理器 transactionManager

MyBatis中有两种事务管理器，JDBC和MANAGED。

JDBC：利用从数据源得到的连接来管理事务。

MANAGED：从不提交和回滚连接，而是由容器来管理事务。

## 事务隔离

SqlSession是数据库会话的抽象，根据配置的不同数据源和事务管理器，SqlSession的行为会有所差异。当然也要考虑数据库对于标准JDBC的适配度。

使用SqlSessionFactory的openSession方法可以开启数据库会话，方法也可以传入一个TransactionIsolationLevel的枚举参数，来定义此会话的事务隔离级别，和JDBC标准一致，mybatis定义了四种级别。

* READ\_UNCOMMITTED：允许脏读、不可重复读和幻读。

SqlSession sess1 = openSession(READ\_UNCOMMITTED);

SqlSession sess2 = openSession(READ\_UNCOMMITTED);

sess1.insert("Question.insert", newQuestion(id, 1L));

q2 = sess2.selectOne("Question.selectOne", id); // 有记录

sess2的事务能够查询倒sess1未提交的记录。

* READ\_COMMITTED：避免脏读，允许发生幻读和不可重复读。

SqlSession sess1 = Config.openSession(READ\_COMMITTED);

SqlSession sess2 = Config.openSession(READ\_COMMITTED);

q1 = sess1.selectOne("Question.selectOne", id);  // 有记录

sess2.delete("Question.deleteOne", id);

sess2.commit();

sess1.clearCache();

q2 = sess1.selectOne("Question.selectOne", id);   // 无记录

从sess1的角度来看，两次查询完全一致，但是查询结果不一致。

* REPEATABLE\_READ：避免脏读和不可重复读，可能发生幻读。
* SERIALIZABLE：避免脏读、不可重复读和幻读。

附录：

脏读：事务读取其他事务的修改但未提交数据。

不可重复读：事务对一行数据的两次查询，结果不一致。select操作未获得锁时会发生。

幻读：事务的两次相同查询，结果不一致。幻读是不可重读特殊情况，查询没有获取范围锁，就可能发生。

# 映射器

mybatis的核心功能就是映射，也就是ORM解决方案。映射器是映射的定义，在将java实体保存到数据库，或者将数据库查询出结果保存到实体时。

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。

## 映射器配置和加载

映射的具体配置在映射器之中，可以使用xml文件来配置，就是Mapper.xml，也可以使用注解来配置。两种方式效果一致，注解和xml配置的映射器，可以任意被xml配置和代码配置加载：

<mappers>

    // 加入xml配置的映射文件

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

    // 加入注解配置的映射文件

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

</mappers>

代码配置MyBatis的加载方式如下：

Configuration configuration = **new** Configuration(environment);

// 加入xml配置的映射文件

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

// 加入注解配置的映射文件

configuration.addMapper(com.ray.entity.QuestionMapper.**class**);

## 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

## 查询映射

### 查询语句 select

select语句属于读操作，语句会从数据库得到结果集。在映射器中需要配置查询的语句，查询的参数和映射关系。一个简短的查询语句如下：

**<select** id="selectSingle" resultType="com.ray.Author"**>**

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

**</select>**

配置中加载此映射器后，代码中这样来使用：

// 调用xml语句

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

这里只配置了语句和结果类型， mybatis会在后台创建结果映射，这个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。除了映射select语句还有一些管理层面的配置，如缓存等。对于复杂的映射，可以创建单独的结果映射来指定更多的映射关系。

### 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select属性需要二次查询的语句，column指定了二次查询的参数， property和javaType指定二次查询结果对应的实体。

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注解的one指定需要二次查询的语句，column指定二次查询的参数，property属性指定二次查询结果需要映射到的实体。

#### 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。

#### 二次查询

一对多的二次查询配置如下：

**<select** id="selectByQ" resultType="com.ray.entity.Reply" **>**

    SELECT \* FROM REPLY WHERE QUESTION\_ID = #{q\_id}

**</select>**

**<resultMap** id="questionResult" type="com.ray.entity.Question"**>**

**<association** column="author\_id"

               property="author" javaType="com.ray.entity.Author"

               select="Author.selectOne" fetchType="lazy" **/>**

**<collection** column="id" property="replys"

               select="Reply.selectByQ" fetchType="lazy"**/>**

**</resultMap>**

使用resultMap的collection的select属性指定需要二次查询的语句，colnmu指定二次查询的参数，property和javaType指定二次查询结果对应的实体。这个是association的配置是完全一致的。

使用注解的配置方式如下：

**public** **interface** QuestionMapper {

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

    @Results({

        @Result(column="create\_date", property="createDate"),

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

                one=@One(fetchType=FetchType.LAZY,

                select="com.ray.anno.mapper.AuthorMapper.selectOne")),

        @Result(property="replys", column="id",

                many=@Many(fetchType=FetchType.LAZY,

                        select="\*.ReplyMapper.selectByQ")

        )

    })

**public** Question selectOne(**long** id);

}

使用result注解的many指定需要二次查询的语句，column指定二次查询的参数，property属性指定二次查询结果需要映射到的实体列表。

#### 关联查询

关联查询的配置如下，基本和assocition一致：

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

    <!-- 其他配置 -->

**<collection** property="replys" ofType="\*.Reply"**>**

**<id** property="id" column="REPLY\_ID"**/>**

**<result** property="content" column="REPLY\_CONTENT"**/>**

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

**</collection>**

**</resultMap>**

# 缓存

缓存用于减少和数据库的交互频率，默认情况下mybatis只会创建会话级别的缓存，即同一个会话多次查询的语句一致时，只会访问一次数据库。要启用全局的二级缓存，可以在映射器使用<cache>元素。

<cache

eviction="FIFO"

flushInterval="60000"

size="512"

readOnly="true"/>

cache有多个属性可以定制缓存。flushInterval设定缓存的刷新间隔，单位是毫秒，默认无。size设定缓存的大小，默认1024。eviction可以设定缓存的淘汰算法，可选的淘汰策略有：LRU，FIFO，SOFT，WEAK，默认是LRU。

缓存的工作机制是：所有select语句的结果会被缓存，所有insert、update和delte语句会刷新缓存。缓存的作用域是一个映射文件中的语句。如果想在多个mapper文件中共享缓存可以使用<cache-ref>应用其他文件中配置的缓存。

# 其他配置

配置文件中还包含其他的一些配置，他们不是mybatis的核心，但是也有一些重要的功能。

## 属性

用于配置mybatis配置上下文的属性，这些属性可以来自属性文件，也可以直接配置单独的属性，也可以在代码中直接设置。

在xml中使用properties元素来配置属性：

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

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

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

</properties>

配置的这些属性可以用于其他配置项来替换属性占位符${propName}。

## 设置

设置是mybatis的一些可选项，用于定制mybatis 的行为。使用<settings>元素来配置设置，列举一些设置可以帮助了解设置的大致目的：

* cacheEnabled：全局的二级缓存开关，默认true
* lazyLoadingEnabled：全局的延迟加载开关，具体映射的配置会覆盖这个设置，默认false。
* autoMappingBehavior：自动映射的行为，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）。默认是PARTIAL。
* defaultStatementTimeout：设置语句的默认超时时间

除此之外还是众多的其他设置，而且每个版本的设置也会有所不同。

## 类型别名

类型别名的目的是将比较长的类名限定配置成比较短的标识符，配置后的标识符可以用于需要使用原本的类名限定的地方。使用<typeAliases>可以配置类型别名。