Rays's

Personal Study Note

of

Mybatis

目 录

[1. MyBatis概述 1](#_Toc15599089)

[1.1. 简介 1](#_Toc15599090)

[1.2. Hello World 1](#_Toc15599091)

[1.2.1. 基于xml文件的Hello World 1](#_Toc15599092)

[1.2.2. 基于代码配置的Hello World 3](#_Toc15599093)

[2. XML配置 4](#_Toc15599094)

# MyBatis概述

## 简介

MyBatis 是一个持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 可以使用简单的 XML 或注解来配置和映射原生类型、接口和 Java 的 POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

## Hello World

mybatis的相关jar包就只有mybatis-xx.jar，只要将需要的版本的包置于工程的类路径即可，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>

现在准备一个数据库，和一张表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

}

### 基于xml文件的Hello World

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。

首先需要配置数据源，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Dog类和dog表之间的映射。接下来需要编写DogMapper.xml

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

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

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

</select>

</mapper>

DogMapper中定义了Dog查询的语句，同时定义了一个占位符ID。然后在代码中可以调用这个语句来查询dog表：

// 使用配置文件创建 SqlSessionFactory

SqlSessionFactory sqlSessionFactory =

new SqlSessionFactoryBuilder().build(Resources.getResourceAsStream(

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

// 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);

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

}

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。

### 基于代码配置的Hello World

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。

# XML配置

配置文件决定mybatis的行为，配置文件结构如下：

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）

## 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提供数据库类型和java类型之间的转换。如int和number之间互相转换，String和varchar和blob等之间的转换。可以自定义类型处理器来处理一些非标准的数据库类型和java类型。要实现自定义类型处理器可以实现TypeHanler接口或者继承BaseTypehandler类。

@MappedJdbcTypes(JdbcType.VARCHAR)

public class RaysTypeHandler extends BaseTypeHandler<String> {

@Override

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

ps.setString(i, parameter);

}

@Override

public String getNullableResult(ResultSet rs, String columnName) {

return rs.getString(columnName);

}

@Override

public String getNullableResult(ResultSet rs, int columnIndex) {

return rs.getString(columnIndex);

}

@Override

public String getNullableResult(CallableStatement cs, int columnIndex) {

return cs.getString(columnIndex);

}

}

然后在配置文件中配置typeHandler

<typeHandlers>

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

</typeHandlers>