# MyBatis简介

1.MyBatis开源免费框架,原名叫iBatis, 2010年google code, 2013年迁移到github

2.作用:数据库访问层框架

2.1底层对JDBC的封装

2.2MyBatis优点之一:

3.1使用MyBatis时不需要编写实现类,只需要写需要执行的sql命令

3.新建以mapper结尾的包,在包下新建:实体类名+Mapping.xml

3.1文件作用:编写需要执行的SQL命令

3.2把xml文件理解成实现类

3.3xml文件内容(看具体代码)

3.4测试结果

## 4.环境搭建详解

4.1全局配置文件中的内容

4.1.1<transactionMaanger/>type 属性值可取值

a.JDBC:事务管理使用JDBC原生事务管理方式

b.MANAGED:把事务管理转交给其他容器.原生JDBC事务setAutoCommit(false)

4.1.2<datasource>type属性

a.POOLED使用数据库连接池

b.UNPOOLED不使用数据库连接池

c.JNDI:java命名目录接口技术(安卓开发用的较多)

## 5.三种查询方式

a.selectList()返回值List<resultType属性控制>

适用于:查询结果都需要遍历的需求

b.selectOne():返回值Object

适用于:返回值结果结果只是变量或只有一行

c.selectMap:返回值Map<key,resultType属性控制>

适用于需要在查询结果中通过某列的值取到这行数据的需求

## 6.setting标签

6.1在mybatis全局配置文件中通过<settings>标签控制mybatis全局开关

6.2在mybatis.xml中开启log4j

6.2.1必须保证有log4j.jar

6.2.2在src下有log4j.properties

```
<settings>
  <setting name="logImpl" value="LOG4J"/>
```

</settings>

## 7.parameterType属性

7.1在xxxMapper.xml中<seelct><delete>等标签的parameterType可以控制参数类型

7.2在Mapper.xml中可以通过#{}获取参数

7.2.1parameterType控制参数类型

7.2.2#{}获取参数内容

7.2.2.1使用索引,从0开始#{}表示第一个参数

7.2.2.2也可以使用#{param1}表示第一个参数

7.2.2.3如果只有一个参数(基本数据类型或String),mybatis对#{}里面内容没有

要求,只要

7.2.2.4如果参数是对象#{属性名}

7.2.2.5如果参数是map 写出#{key}

```
<select id="selById" resultType="com.bjsxt.pojo.People" parameterType="int">
   select * from PEOPLE where id=#{0}
</select>
<select id="selById" resultType="com.bjsxt.pojo.People"
parameterType="com.bjsxt.pojo.People">
   select * from PEOPLE where id=${id}
</select>
<select id="selById" resultType="com.bjsxt.pojo.People" parameterType="map">
   select * from PEOPLE where id=#{id} and name=#{name}
</select>
```

7.3SqlSession的selectlist()和selectOne()的第二个参数和selectMap()的第三个参数都表示

方法的参数

```
People p=session.selectOne("com.bjsxt.mapper.PeopleMapper.selById",1);
```

7.4#{}和${}的区别

7.4.1#{}获取参数的内容支持 索引获取,param1获取指定位置参数,并且SQL语句中使用?处理

7.4.2${}字符串拼接,不使用?, 默认找${内容}内容的set,get方法,写一个数字就是一个数字

## 8.实现分页

8,1?不允许在关键字前后进行数学运算,需要在代码中计算完成后传到xml中

```
 select * from (select ROWNUM r,e.* from people e where ROWNUM < #{pageEnd}
) where r>#{pageStart} and r <#{pageEnd}
```

## 9.typeAliases别名

9.1系统内置别名:把类型全小写

9.2给某个类起别名

9.3给某个包起别名

```
<typeAliases>
   <typeAlias type="com.bjsxt.pojo.People"  alias="peo"/>
   <package name="com.IntefaceBinding.pojo"/>
</typeAliases>
```

## 10.MyBatis实现新增

10.1功能:从应用程序角度出发,软件具有哪些功能

10.2业务:完成功能时的逻辑,对象Service中一个方法

10.3事务:从数据库角度出发,完成业务时需要执行的SQL集合,统称为一个事务

10.4在mybatis中默认是关闭了JDBC的自动提交功能

10.5每一个SqlSession默认都是不自动提交事务

10.6session.commit提交事务 事务回滚

10.7openSession(true);自动提交setAutoCommit(true)

10.8mybatis底层都是对JDBC的封装

10.8.1JDBC中的executeUpdate()执行增删改查, 返回值为int

10.8.2mabitis

10.9在openSession()时Mybatis会自动创建SqlSession时同时创建一个Transaction(事务对象)

同时setAutoCommit设置为false

## 11.MyBatis接口绑定方案及多参数传递

11.1作用:实现创建一个接口后mapper.xml由mybatis生成接口的实现类,通过调用接口对象就可以

获取mapper.xml

11.2后面nybatis和spring整合时使用的是这个方案

11.3实现步骤

11.3.1创建一个接口

11.3.1.1接口包名和接口名与mapper.xml中<mapper>namespace相同

11.3.1.2接口中方法名和mapper.xml标签的id属性相同

11.3.2在mybatis.xml中使用<package>进行扫描接口和mapper.xml

## 12.动态SQL

12.1根据不同的条件需要执行不同的SQL命令,称为动态SQL

12.2MyBatis

# 13ThreadLocal

13.1线程容器,给线程绑定一个Object内容,后只要线程不变,可以随时取出.改变线程无法取出.

## 14.缓存

14.1应用程序和数据库交互的过程是一个相对耗时的过程

14.2缓存存在的意义:让应用程序减少对数据库的访问,提升程序运行效率

14.3MyBatis中默认sqlSession缓存开启

14.3.1同一个SqlSession对象调用同一个<select>,只有第一次是访问数据库的.第一次后

把查询结果缓存到SqlSessino缓存区(内存)中

14.3.2缓存的是statement对象(一个<select>对应一个对象,并且一个session一个空间)

14.3.3缓存流程

1.先到缓存区找,是否存在statement

2.返回结果

3.如果没有缓存statement对象,去数据库获取数据

4.数据库返回查询结果

5.把查询结果放到缓存区

14.3.4SqlSessionFactory缓存(二级缓存)

14.3.1.有效范围:同一个factory内哪个SqlSession都可以获取

14.3.2.什么时候可以使用二级缓存

a.当数据频繁被使用,很少被修改

14.3.3.使用二级缓存

a.在mapper.xml中添加

b.如果不写readOnly="true"需要把实体类序列化

```
<cache readOnly="true"></cache>
```

c.当SqlSession对象close()或commit()时会把SqlSessin缓存的数据刷到

SqlSessionFactory缓存区中

## 15.MyBatis实现多表查询

15.1MyBatis实现多表查询方式

15.1.1业务装配.对两个表编写单表查询语句,在业务(Service)把查询的两个结果进行关联

15.1.2使用Auto Mapping特性,在实现两表联合查询时通过别名完成映射

15.1.3使用MyBatis的<resultMap>标签进行实现.

15.2多表查询时,类中包含另一个类的对象的分类

15.2.1单个对象

15.2.2集合对象

## 16.<resultMap>标签

16.1<resultMap>标签写在mapper.xml中,由程序员控制SQL查询结果与实体类的映射关系

16.1.1默认MyBatis使用Auto Mapping特性

16.2使用<resultMap>标签时,<select>标签不写resultType属性,而是使用resultMap属性引用

<resultMap>标签

```
<mapper>
  <resultMap id="mymap" type="Teacher">
    <!--主键使用id标签配置映射关系-->
    <id column="id" property="id1"/>
    <!--其他列使用result标签配置映射关系-->
    <result column="name" property="name1"/>
  </resultMap>
  <select id="selAll" resultMap="mymap">
    select * from TEACHER
  </select>
</mapper>
```

16.3使用resultmap实现关联单个对象(N+1方式)

16.3.1N+1查询方式,先查询出某个表的全部信息,根据这个表的信息,再查询另一个表的信息

16.4与业务装配的区别:

16.4.1在service里面写的代码,有mybatis来完成装配

16.5实现步骤:

16.5.1在studnet实现类中包含一个Teacher对象

16.5.2在TeacherMapper中提供一个查询

16.5.3在StudentMapper中

16.5.3.1<association>装配一个对象时使用

16.5.3.2property:对象在类中的属性名

16.5.3.3select:通过哪个查询查询出这个对象的信息

16.5.3.4column:把当前表的哪个列的值作为参数传递给另一个查询

16.5.3.4大前提使用N+1方式时,如果列名和属性名相同可以不配置,使用Auto

mapping特性.但是mybatis只会给列装配一次

```
<association property="teacher"
select="com.resultmap.mapper.TeacherMapper.selById" column="tid">
</association>
```

## 16.4. 使用 resultMap 实现关联单个对象(联合查询方式)

16.4.1 只需要编写一个 SQL,在 StudentMapper 中添加下面效果

16.4.2 <association/>只要专配一个对象就用这个标签

16.4.3 此时把<association/>小的<resultMap>看待

16.4.4 javaType 属性:<association/>专配完后返回一个什么类型的对象.取值是一个类(或类的

别名

```
<association property="teacher" javaType="Teacher" >
    <id column="tid" property="id"/>
    <result column="tname" property="name"/>
</association>
```

## 16.5 N+1 方式和联合查询方式对比

16.5.1 N+1:需求不确定时.

16.5.2 联合查询:需求中确定查询时两个表一定都查询.

16.5.3 N+1 名称由来

16.5.3.1 举例:学生中有 3 条数据

16.5.3.2 需求:查询所有学生信息级授课老师信息

16.5.3.3 需要执行的 SQL 命令

16.5.3.4查询全部学生信息:select * from 学生

16.5.3.5执行 3 遍 select * from 老师 where id=学生的外键

16.5.3.6使用多条 SQl 命令查询两表数据时,如果希望把需要的数据都查询出来,需要执行

N+1 条 SQl 才能把所有数据库查询出来.

**16.5.3.7缺点:**

效率低

**16.5.3.8优点:**

a.如果有的时候不需要查询学生是同时查询老师.只需要执行一个 select * from student;

16.5.3.9适用场景: 有的时候需要查询学生同时查询老师,有的时候只需要查询学生.

16.5.3.10如果解决 N+1 查询带来的效率低的问题

16.5.3.10.1默认带的前提: 每次都是两个都查询.

16.5.3.10.2使用两表联合查询.

### 16.4使用<resultMap>查询关联集合对象(N+1)

```
<collection property="list" ofType="com.resultmap2.pojo.Student"
select="com.resultmap2.mapper.StudentMapper.selByTid" column="id">
</collection>
```

### 16.5使用<resultMap>查询关联集合对象(联合查询)

```
<collection property="list" ofType="com.resultmap2.pojo.Student">
   <id column="sid" property="id"/>
   <result column="sname" property="name"/>
   <result column="age" property="age"/>
   <result column="tid" property="tid"/>
</-collection>
```

## 17使用AutoMapping结合别名实现多表查询

17.1只能使用多表联合查询方式

17.2查询出的列明和属性名相同

在 SQL 是关键字符,两侧添加反单引号

```
select t.id `teacher.id`,t.name `teacher.name`,s.id id,s.name name,age,tid from
student s LEFT JOIN teacher t on t.id=s.tid
```

## 18.MyBatis注解

18.1注解:为了简化配置文件

18.2MyBatis的注解简化mapper.xml文件

18.2.1如果涉及动态sql依然使用mapper.xml

18.3mapper.xml和注解可以共存

18.4使用注解时mybatis.xml中<mapper>使用

18.4.1<package/>

18.4.2<mapper class="">

```
@Select("select * from teacher")
@Insert("insert into TEACHER values(#{id},#{name})")
@Update("update TEACHER set name=#{name} where id=#{id}")
@Delete("delete from TEACHER where id=#{0}")
@Results(value={
        @Result(id=true,property ="id",column="id"),
        @Result(property = "name" ,column = "name"),
        @Result(property = "list" ,column = "id",many
=@Many(select="com.Annotation.mapper.StudentMapper.selById"))
         })
@Select("select * from TEACHER")
```

## 19.运行原理

1. 运行过程中涉及到的类

1.1 Resources MyBatis 中 IO 流的工具类

1.1.1 加载配置文件

1.2 SqlSessionFactoryBuilder() 构建器

1.2.1 作用:创建 SqlSessionFactory 接口的实现类

1.3 XMLConfigBuilder MyBatis 全局配置文件内容构建器类

1.3.1 作用负责读取流内容并转换为 JAVA 代码.

1.4 Configuration 封装了全局配置文件所有配置信息.

1.4.1 全局配置文件内容存放在 Configuration 中

1.5 DefaultSqlSessionFactory 是SqlSessionFactory接口的实现类

1.6 Transaction 事务类

16.1 每一个 SqlSession 会带有一个 Transaction 对象.

1.7 TransactionFactory 事务工厂

1.7.1 负责生产 Transaction

1.8 Executor MyBatis 执行器1.8.1 作用:负责执行 SQL 命令

1.8.2 相当于 JDBC 中 statement 对象(或 PreparedStatement或 CallableStatement)

1.8.3 默认的执行器 SimpleExcutor

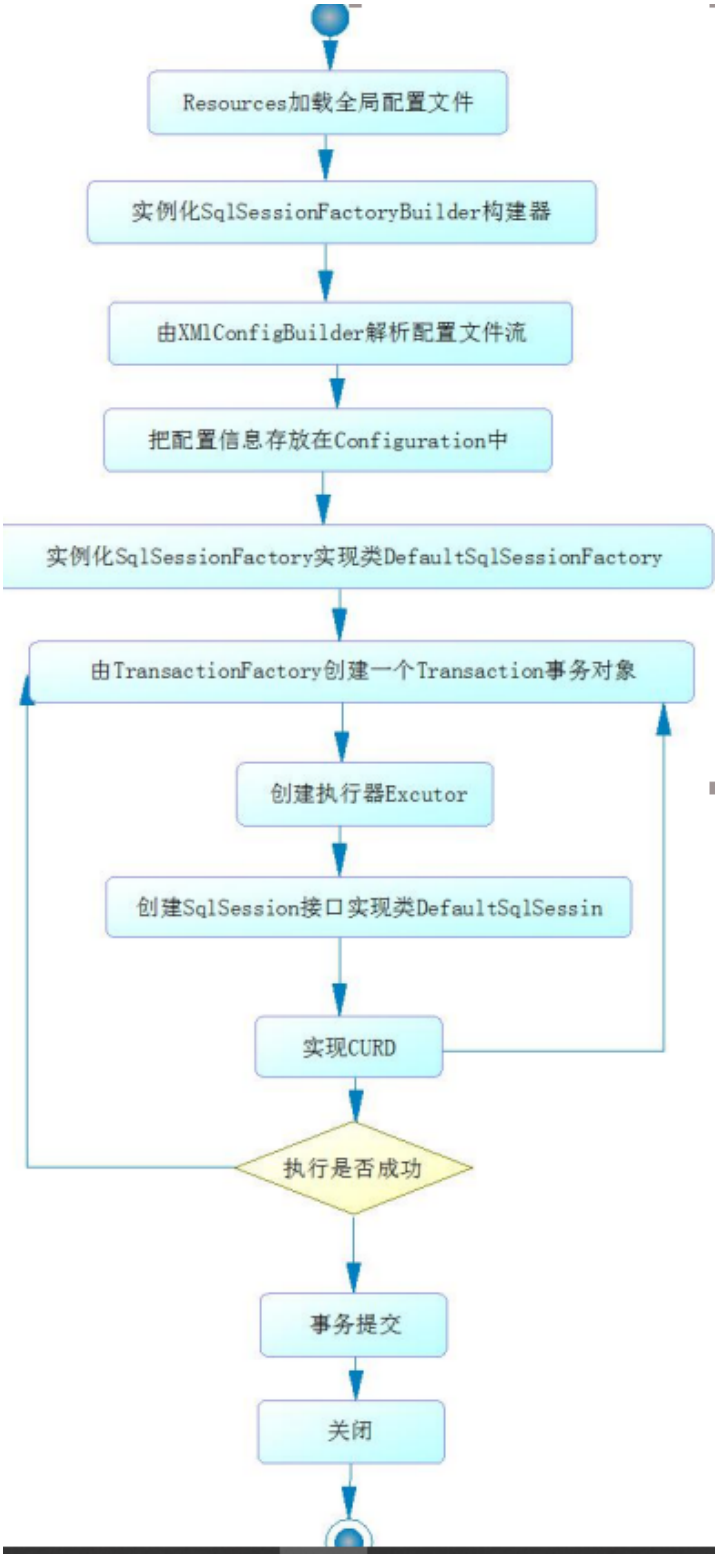
1.8.4 批量操作 BatchExcutor

1.8.5 通过 openSession(参数控制)

1.9 DefaultSqlSession 是 SqlSession 接口的实现类

1.10 ExceptionFactory MyBatis 中异常工厂

**2.** 流程图




**3.文字解释**

在 MyBatis 运行开始时需要先通过 Resources 加载全局配置文件.下面需要实例化 SqlSessionFactoryBuilder 构建器.帮助 SqlSessionFactory 接口实现类 DefaultSqlSessionFactory.

在实例化 DefaultSqlSessionFactory 之前需要先创建 XmlConfigBuilder 解析全局配置文件流,并把解析结果存放在 Configuration 中.之后把 Configuratin 传递给 DefaultSqlSessionFactory.到此 SqlSessionFactory 工

厂创建成功.

由 SqlSessionFactory 工厂创建 SqlSession.

每次创建 SqlSession 时,都需要由 TransactionFactory 创建 Transaction 对象,同时还需要创建 SqlSession 的执行器 Excutor,最后实例化 DefaultSqlSession,传递给 SqlSession 接口.

根据项目需求使用 SqlSession 接口中的 API 完成具体的事务操作.

如果事务执行失败,需要进行 rollback 回滚事务.

如果事务执行成功提交给数据库.关闭 SqlSession

到此就是 MyBatis 的运行原理.(面试官说的.)