企业级开发第三章

小書匠

目录

三 MyBatis的基本应用

1 教学内容

1. MyBatis的工作原理和入门程序;
2. MyBatis的核心配置。

2学习要求

1. 理解MyBatis的工作原理;
2. 熟练掌握MyBatis入门程序;
3. 熟练掌握MyBatis的配置。

3 mybatis简介

3.1问题分析

3.1.1 直接用jdbc

大量没有技术含量的体力活,例如我们做插入时:

```
stmt = conn.prepareStatement("insert into t_user values(default ,?,?,?,?,?)");
            stmt.setString(1, user.getName());
            stmt.setString(2, user.getPwd());
            stmt.setInt(3, user.getAge());
            stmt.setDate(4, new Date(user.getBirthday().getTime()));
            stmt.setString(5, user.getPhoto());
```

我们需要完成映射:

Java	关系数据库SQL语句
类User	表t_user
属性name/pwd/age/birthday/id等	列 name/pwd/age/birthday/id
属性类型	列类型

企业实战中列多达50列, 这样存粹是体力活;同时存在大量重复代码

3.1.2用JdbcTemplate

```
this.getJdbcTemplate().update("insert into tb_user values(default ,?,?,?,?,?)",
        user.getName(), user.getPwd(),
        user.getAge(), new Date(user.getBirthday().getTime()),
        user.getPhoto());
```

虽然减少了重复代码, 但是仍然还得继续完成映射:

3.1.3总结

不管用jdbc或者JdbcTemplate,我们都花费了大量精力完成映射且是没有技术含量的体力活, 企业实战中, 我们都借助于框架来减少这种映射代码, 提高开发效率, Java中用的比较多是Hibernate和Mybatis,在国内轻量级MyBatis用比较多。

3.2 简介

MyBatis 是一款优秀的持久层框架, 它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO(Plain Old Java Objects, 普通老式 Java 对象)为数据库中的记录。

3.3 MyBatis安装和配置

新建一个Maven。

3.3.1 添加依赖

```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.23</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.6</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
```

3.3.2新建mybatis配置文件

MyBatis也要连接上数据库, 因而需要配置连接上数据库的四个信息。对Mybatis做一定配置。

```
<?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="com.mysql.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/nyist"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
            </dataSource>
        </environment>
    </environments>
</configuration>
```

3.3.3新建MyBatisUtil工具类

替代JdbcUtil,只不过这里是获取SqlSession而不是Connect

```
package com.guodexian.mybatis.util;

import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import java.io.IOException;
import java.io.InputStream;

/**
 * @author 南阳德刚教育<br>
 * 2020/9/5 16:08<br>
 * 说明:
 */
public class MyBatisUtil {
    private static  SqlSessionFactory sqlSessionFactory = null;
    static {//工程只需要一个即可
        String resource = "mybatis-config.xml";
        InputStream inputStream = null;
        try {
            inputStream = Resources.getResourceAsStream(resource);
            sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static SqlSession  getSqlSession(){
        return  sqlSessionFactory.openSession();
    }
}
```

3.4编写Mapper文件

3.4.1 文件

为了减少Java代码, 让MyBatis帮我完成映射代码, 我们需要告诉映射关系, 我们写在xml文件中;如果我们列名和属性名完全一致, MyBatis会自动完成映射查询时:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper
        "http://mybatis.org/dtd/mybatis-3-mapper.
```

```
<!--避免id值重复问题，使用namespace限定-->
<mapper namespace="com.guodexian.mybatis.dao.UserMapper">
    <select id="save" parameterType="com.guodexian.mybatis.entity.User">
        insert into t_user values(default ,#{name},#{pwd},#{birthday},#{age})
    </select>
    <update id="update" parameterType="com.guodexian.mybatis.entity.User">
        update  t_user set name=#{name},pwd=#{pwd},age=#{age},birthday=#{birthday} where id=#{id}
    </update>
    <delete id="delete" parameterType="int">
        delete from t_user where id=#{id}
    </delete>
</mapper>
```

3.4.2 配置

因为MyBatisUtil读取配置文件, 我们需要把Mapper配置进去:

```
    <mappers>
        <mapper resource="com/guodexian/mybatis/dao/UserMapper.xml"></mapper>
    </mappers>
```

3.5 新建Dao

3.5.0 新建实体类

```
1  package com.guodexian.mybatis.entity;
2
3  import java.util.Date;
4
5  public class User {
6      private Integer id;
7      private String name;
8      private String pwd;
9      private Date birthday;
10     private int age;
11
12     public Integer getId() {
13         return id;
14     }
15
16     public void setId(Integer id) {
17         this.id = id;
18     }
19
20     public String getName() {
21         return name;
22     }
23
24     public void setName(String name) {
25         this.name = name;
26     }
27
28     public String getPwd() {
29         return pwd;
30     }
31
32     public void setPwd(String pwd) {
33         this.pwd = pwd;
```

```
    }

    public Date getBirthday() {
        return birthday;
    }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", pwd='" + pwd + '\'' +
                ", birthday=" + birthday +
                ", age=" + age +
                '}';
    }
}
```

3.5.1修改UserDao接口

```
package com.guodexian.javaweb.dao;

import com.guodexian.javaweb.model.User;
public interface UserDao {
    /**
     * 保存用户
     * @param user
     * @return
     */
    int save(User user);

    /**
     * 更新用户
     * @param user
     * @return
     */
    int update(User user);

    /**
     * 根据id删除
     * @param id
     * @return
     */
    int delete(int id);

}
```

3.5.2 UserDaoMyBatisImpl

```
package com.guodexian.mybatis.dao.impl;

import com.guodexian.mybatis.dao.UserDao;
import com.guodexian.mybatis.entity.User;
import com.guodexian.mybatis.util.MyBatisUtil;
import org.apache.ibatis.session.SqlSession;

public class UserDaoMyBatisImpl implements UserDao {
    @Override
    public int save(User user) {
        SqlSession sqlSession=null;
        int ret=0;
        try {
            sqlSession= MyBatisUtil.getSqlSession();
            ret=sqlSession.insert("com.guodexian.mybatis.dao.UserMapper.save",user);
            sqlSession.commit();
        }catch (Exception e){
            if (sqlSession!=null){
                sqlSession.rollback();
            }
            e.printStackTrace();
        }finally {
            if (sqlSession!=null){
                sqlSession.close();
            }
        }
        return ret;
    }

    @Override
    public int update(User user) {
        SqlSession sqlSession=null;
        int ret=0;
        try {
            sqlSession=MyBatisUtil.getSqlSession();
            ret=sqlSession.update("com.guodexian.mybatis.dao.UserMapper.update",user);
            sqlSession.commit();
        }catch (Exception e){
            if (sqlSession!=null){
                sqlSession.rollback();
            }
            e.printStackTrace();
        }finally {
            if (sqlSession!=null){
                sqlSession.close();
            }
        }
        return ret;
    }

    @Override
    public int delete(int id) {
        SqlSession sqlSession=null;
        int ret=0;
        try {
            sqlSession=MyBatisUtil.getSqlSession();
            ret=sqlSession.delete("com.guodexian.mybatis.dao.UserMapper.delete",id);
            sqlSession.commit();
        }catch (Exception e){
            if (sqlSession!=null){
                sqlSession.rollback();
            }
            e.printStackTrace();
        }finally {
            if (sqlSession!=null){
```

```
                sqlSession.close();
            }
        }
        return ret;
    }
}
```

3.5.3 修改测试类

在test/java下新建测试类：

```
package com.guodexian.mybatis.test;

import com.guodexian.mybatis.dao.UserDao;
import com.guodexian.mybatis.dao.impl.UserDaoMyBatisImpl;
import com.guodexian.mybatis.entity.User;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import java.util.Date;

public class UserDaoTest {
    private UserDao userDao;

    @Before
    public void init() {
        userDao = new UserDaoMyBatisImpl();
    }

    @Test
    public void testSave() {
        User user = new User();
        user.setPwd("888888");
        user.setName("wangwu");
        user.setAge(23);
        user.setBirthday(new Date(1990, 1, 1));
        int ret = userDao.save(user);
        System.out.println(ret);
        Assert.assertEquals(1, ret);
    }

    @Test
    public void testUpdate() {
        User user = new User();
        user.setId(3);
        user.setPwd("999999");
        user.setName("wangwu");
        user.setAge(23);
        user.setBirthday(new Date(1990 - 1900, 1, 1));
        int ret = userDao.update(user);
        System.out.println(ret);
        Assert.assertEquals(1, ret);
    }

    @Test
    public void testDel() {
        int ret = userDao.delete(1);
        System.out.println(ret);
        Assert.assertEquals(1, ret);
    }
```

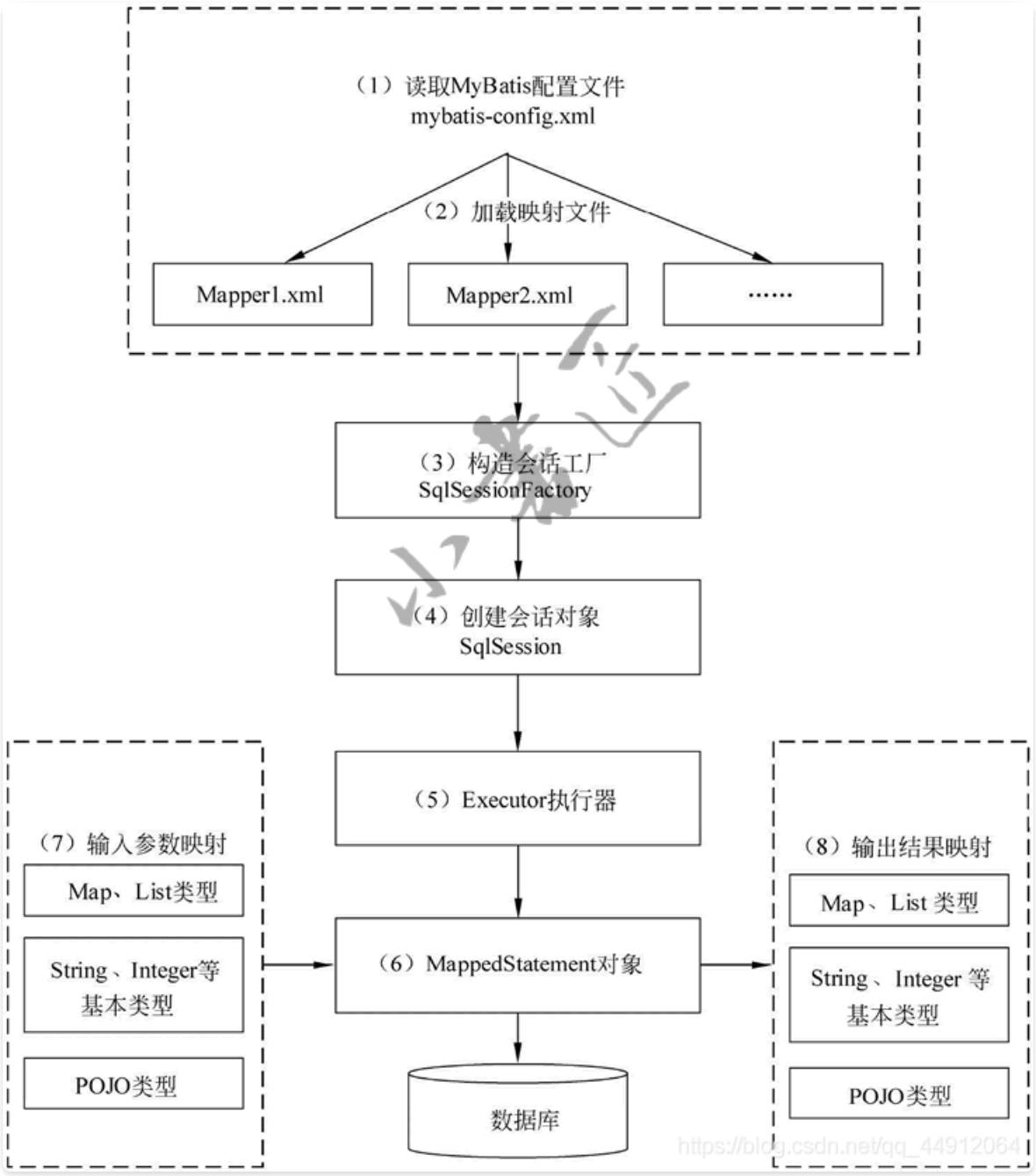
```
}
```

3.5.4 测试

安装测试类方法顺序, 依次执行, 看看结果

3.6 mybatis工作原理

3.6.1 原理解释



mybatis工作原理

上面中流程就是MyBatis内部核心流程, 每一步流程的详细说明如下文所述:

1. 读取MyBatis的配置文件：mybatis-config.xml为MyBatis的全局配置文件，用于配置数据库连接信息。
2. 加载映射文件：映射文件即SQL映射文件，该文件中配置了操作数据库的SQL语句，需要在MyBatis配置文件mybatis-config.xml中加载。mybatis-config.xml 文件可以加载多个映射文件，每个文件对应数据库中的一张表。
3. 构造会话工厂：通过MyBatis的环境配置信息构建会话工厂SqlSessionFactory。
4. 创建会话对象：由会话工厂创建SqlSession对象，该对象中包含了执行SQL语句的所有方法。
5. Executor执行器：MyBatis底层定义了一个Executor接口来操作数据库，它将根据SqlSession传递的参数动态地生成需要执行的SQL语句，同时负责查询缓存的维护。
6. MappedStatement对象：在Executor接口的执行方法中有一个MappedStatement类型的参数，该参数是对映射信息的封装，用于存储要映射的SQL语句的id、参数等信息。
7. 输入参数映射：输入参数类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输入参数映射过程类似于JDBC对preparedStatement对象设置参数的过程。
8. 输出结果映射：输出结果类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输出结果映射过程类似于JDBC对结果集的解析过程。

MyBatis的基本工作原理就是：**先封装SQL，接着调用JDBC操作数据库，最后把数据库返回的表结果封装成Java类**。

3.6.2 启动日志验证

首先添加日志依赖：

```
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.14.1</version>
        </dependency>
```

然后在resources目录下添加文件log4j2.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration status="TRACE" monitorInterval="30">
    <!--先定义所有的appender-->
    <appenders>
        <!--输出控制台的配置-->
        <console name="Console" target="SYSTEM_OUT">
            <!--输出日志的格式-->
            <PatternLayout pattern="[%d{HH:mm:ss:SSS}] [%p] - %l - %m%n"/>
        </console>
    </appenders>
    <!--然后定义logger，只有定义了logger并引入的appender，appender才会生效-->
    <loggers>
        <!--过滤掉spring和mybatis的一些无用的DEBUG信息-->
        <logger name="org.mybatis" level="INFO"/>
        <root level="all">
            <!--输出到控制台-->
            <appender-ref ref="Console"/>
        </root>
    </loggers>
</configuration>
```

然后运行testSave测试方法, 看日志:

```
.java:137) - Opening JDBC Connection
rce.java:434) - Created connection 30699728.
saction.java:101) - Setting autocommit to false on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1d470d0]
=>  Preparing: insert into t_user values(default ,?,?,?,?)
=> Parameters: wangwu(String), 888888(String), 3890-02-01 00:00:00.0(Timestamp), 23(Integer)
==    Updates: 1
) - Committing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1d470d0]
on.java:123) - Resetting autocommit to true on JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1d470d0]
) - Closing JDBC Connection [com.mysql.cj.jdbc.ConnectionImpl@1d470d0]
urce.java:391) - Returned connection 30699728 to pool.
```

Mybatis执行流程