mybatis开发dao的方法

1. SqlSession使用范围

1. SqlSessionFactoryBuilder

- 通过SqlSessionFactoryBuilder创建会话工厂。
- 将SqlSessionFactoryBuilder当成一个工具类使用即可，不需要使用单例管理SqlSessionFactoryBuilder。
- 在创建SqlSessionFactory的时候，只需要new一次SqlSessionFactoryBuilder即可。

2. SqlSessionFactory

- 通过SqlSessionFactory创建SqlSession，使用单例模式管理SqlSessionFactory（工厂一旦创建，一直使用一个实例）。
- mybatis和spring整合之后，使用单例模式管理SqlSessionFactory。

3. SqlSession

- SqlSession是一个面向用户（程序员）的接口。
- SqlSession中提供了很多操作数据库的方法。如selectOne、selectList。
- SqlSession是线程不安全的，在SqlSession的实现类中，除了有接口中的方法（操作数据库的方法），还有数据域属性。
- SqlSession最佳应用场合在方法体内，定义成局部变量使用。

2. 原始dao开发方法（程序员需要写dao接口和dao实现类）

1. 思路

- 需要向dao实现类中注入SqlSessionFactory，在方法体内通过SqlSessionFactory创建SqlSession。

2. dao接口

```
package dao;

import com.swxc.mybatis.po.User;


public interface UserDao {

    // 根据id查询用户信息
    public User findUserById(int id) throws Exception;

    // 添加用户信息
    public void insertUser(User user) throws Exception;

    // 删除用户信息
    public void deleteUser(int id) throws Exception;

}
```

3. dao接口实现类

```
package dao;

import com.swxc.mybatis.po.User;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
```

```
public class UserDaoImpl implements UserDao {

    // 需要向dao实现类中注入SqlSessionFactory
    // 这里通过构造函数注入
    private SqlSessionFactory sqlSessionFactory;
    public UserDaoImpl(SqlSessionFactory sqlSessionFactory) {
        this.sqlSessionFactory = sqlSessionFactory;
    }

    @Override
    public User findUserById(int id) throws Exception {
        SqlSession sqlSession = sqlSessionFactory.openSession();
        User user = sqlSession.selectOne("test.findUserById", id);
        sqlSession.close();
        return user;
    }

    @Override
    public void insertUser(User user) throws Exception {
        SqlSession sqlSession = sqlSessionFactory.openSession();
        sqlSession.insert("test.insertUser", user);
        sqlSession.commit();
        sqlSession.close();
    }

    @Override
    public void deleteUser(int id) throws Exception {
        SqlSession sqlSession = sqlSessionFactory.openSession();
        sqlSession.delete("test.deleteUser", id);
        sqlSession.commit();
        sqlSession.close();
    }
}
```

4. 测试类

```
package test;

import com.swxc.mybatis.po.User;
import dao.UserDao;
import dao.UserDaoImpl;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Before;
import org.junit.Test;

import java.io.InputStream;

/**
 * Description
 * Author shadowolf
 * Date 2017/7/21. 14:33
 * Version 1.0
 */
public class UserDaoImplTest {

    private SqlSessionFactory sqlSessionFactory;

    @Before
    public void setUp() throws Exception {
        String resource = "SqlMapConfig.xml";
        InputStream inputStream  = Resources.getResourceAsStream(resource);
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    }
```

```
    @Test
    public void testFindUserById() throws Exception {
        // 创建UserDao对象
        UserDao userDao = new UserDaoImpl(sqlSessionFactory);

        // 调用UserDao的方法
        User user = userDao.findUserById(1);

        System.out.println(user);
    }

}
```

5. 原始dao的开发问题

- dao接口实现类方法中存在大量的模板方法，设想能将这些代码提取出来，大大减轻程序员的工作量。
- 调用SqlSession方法时，将statement的id硬编码了。
- 调用SqlSession方法时传入的变量，由于SqlSession方法使用泛型，即使变量类型传入错误，在编译阶段也不报错，不利于程序员开发。

3. mapper代理方法（程序员只需要写mapper接口（相当于dao接口））

1. 思路

- 程序员还需要编写mapper.xml映射文件。
- 程序员编写mapper接口需要遵循一些开发规范，mybatis可以自动生成mapper接口实现类代理对象。
- 开发规范：
  - 在mapper.xml中，namespace等于mapper接口的地址。

```
<!-- namespace命名空间，作用就是对sql进行分类化管理，理解sql隔离
 注意：使用mapper代理方法开发，namespace有特殊重要作用，namespace等于mapper接口的地址
 -->
<mapper namespace="com.swxc.mybatis.mapper.UserMapper">
```

  - mapper.java接口中的方法名和mapper.xml中statement的id一致。
  - mapper.java接口中方法输入参数的类型和mapper.xml中statement的parameterType一致。
  - mapper.java接口中返回值的类型和mapper.xml中statement的resultType一致。

```
<select id="findUserById" parameterType="int" resultType="com.swxc.mybatis.po.User">
    SELECT * FROM USER WHERE id = #{id}
</select>
```

```
// 根据id查询用户信息
public User findUserById(int id) throws Exception;
```

2. mapper.java

```
// 根据id查询用户信息
public User findUserById(int id) throws Exception;
```

3. mapper.xml

```
<select id="findUserById" parameterType="int" resultType="com.swxc.mybatis.po.User">
    SELECT * FROM USER WHERE id = #{id}
</select>
```

4. 在SqlMapConfig.xml中加载mapper.xml

```
<mapper resource="mapper/UserMapper.xml" />
```

5. 测试

```
package test;

import com.swxc.mybatis.mapper.UserMapper;
import com.swxc.mybatis.po.User;
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 org.junit.Before;
import org.junit.Test;

import javax.annotation.Resource;
import java.io.InputStream;


public class UserMapperTest {

    private SqlSessionFactory sqlSessionFactory;

    @Before
    public void setUp() throws Exception {
        // 创建SqlSessionFactory
        String resource = "SqlMapConfig.xml";
        InputStream inputStream = Resources.getResourceAsStream(resource);
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
    }

    @Test
    public void testFindUserById() throws Exception {

        SqlSession sqlSession = sqlSessionFactory.openSession();

        // 创建UserMapper对象，mybatis自动生成mapper代理对象
        UserMapper userMapper = sqlSession.getMapper(UserMapper.class);

        // 调用UserMapper方法
        User user = userMapper.findUserById(1);

        // 释放资源
        sqlSession.close();

        System.out.println(user);

    }

}
```

6. 一些问题的总结

- 代理对象内部调用selectOne或selectList
  - 如果mapper方法返回单个pojo对象（非集合对象），代理对象内部通过selectOne查询数据库。
  - 如果mapper方法返回集合对象，代理对象内部通过selectOne查询数据库。
- mapper接口方法参数只能有一个是否影响系统开发
  - 系统框架中，dao层的代码是被业务层公用的。
  - 即使mapper接口只有一个参数，可以使用包装类型的pojo满足不同的业务方法的需求。
  - 注意：持久层中方法的参数可以使用包装类型、map...，service方法中建议不要使用包装类型（不利于业务层的可扩展性）。