狂神说MyBatis04：使用注解开发

秦疆 狂神说 2020-04-10

使用注解开发

上集回顾：狂神说MyBatis03：ResultMap及分页

> 面向接口编程

- 大家之前都学过面向对象编程，也学习过接口，但在真正的开发中，很多时候我们会选择面向接口编程
- **根本原因： 解耦，可拓展，提高复用，分层开发中，上层不用管具体的实现，大家都遵守共同的标准，使得开发变得容易，规范性更好**
- 在一个面向对象的系统中，系统的各种功能是由许许多多的不同对象协作完成的。在这种情况下，各个对象内部是如何实现自己的,对系统设计人员来讲就不那么重要了；
- 而各个对象之间的协作关系则成为系统设计的关键。小到不同类之间的通信，大到各模块之间的交互，在系统设计之初都是要着重考虑的，这也是系统设计的主要工作内容。面向接口编程就是指按照这种思想来编程。

关于接口的理解

- 接口从更深层次的理解，应是定义（规范，约束）与实现（名实分离的原则）的分离。
- 接口的本身反映了系统设计人员对系统的抽象理解。
- 接口应有两类：
  - 第一类是对一个个体的抽象，它可对应为一个抽象体(abstract class)；
  - 第二类是对一个个体某一方面的抽象，即形成一个抽象面（interface）；
- 一个体有可能有多个抽象面。抽象体与抽象面是有区别的。

三个面向区别

- 面向对象是指，我们考虑问题时，以对象为单位，考虑它的属性及方法.
- 面向过程是指，我们考虑问题时，以一个具体的流程（事务过程）为单位，考虑它的实现.
- 接口设计与非接口设计是针对复用技术而言的，与面向对象（过程）不是一个问题.更多的体现就是对系统整体的架构

> 利用注解开发

- **mybatis最初配置信息是基于 XML ,映射语句(SQL)也是定义在 XML 中的。而到MyBatis 3提供了新的基于注解的配置。不幸的是，Java 注解的的表达力和灵活性十分有限。最强大的 MyBatis 映射并不能用注解来构建**
- sql 类型主要分成 :
  - @select ()
  - @update ()
  - @Insert ()
  - @delete ()

注意：利用注解开发就不需要mapper.xml映射文件了 .

1、我们在我们的接口中添加注解

```
//查询全部用户
@Select("select id,name,pwd password from user")
public List<User> getAllUser();
```

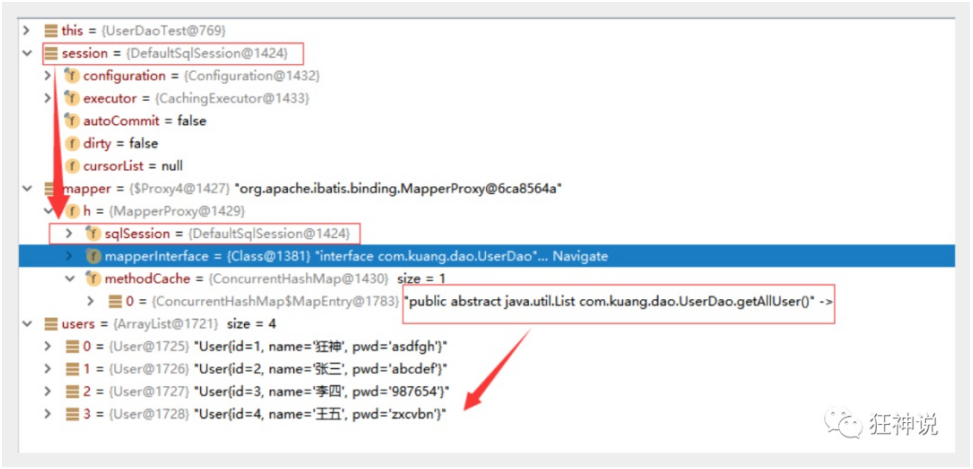
2、在mybatis的核心配置文件中注入

```
<!--使用class绑定接口-->
<mappers>
   <mapper class="com.kuang.mapper.UserMapper"/>
</mappers>
```

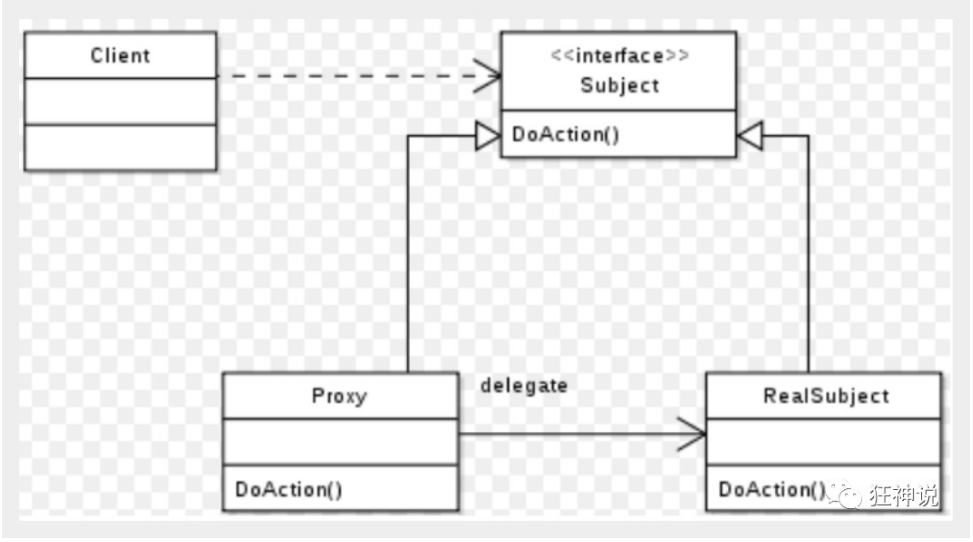
3、我们去进行测试

```
@Test
public void testGetAllUser() {
   SqlSession session = MybatisUtils.getSession();
   //本质上利用了jvm的动态代理机制
   UserMapper mapper = session.getMapper(UserMapper.class);

   List<User> users = mapper.getAllUser();
   for (User user : users){
       System.out.println(user);
  }

   session.close();
}
```

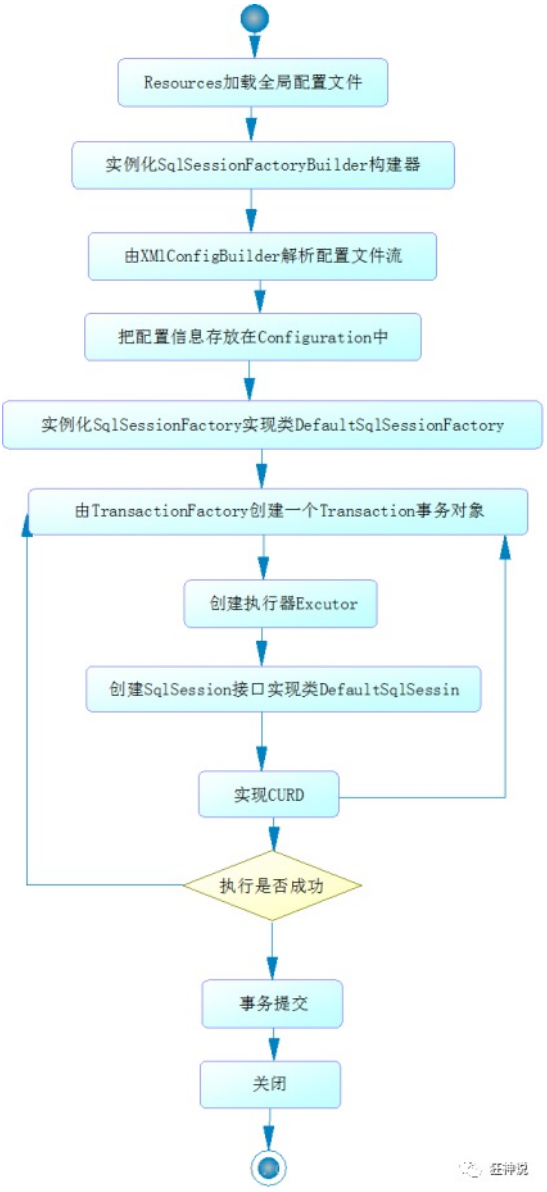
4、利用Debug查看本质



5、本质上利用了jvm的动态代理机制



6、Mybatis详细的执行流程



> 注解增删改

改造MybatisUtils工具类的getSession( ) 方法，重载实现。

```
//获取SqlSession连接
public static SqlSession getSession(){
    return getSession(true); //事务自动提交
}
```

```
  public static SqlSession getSession(boolean flag){
      return sqlSessionFactory.openSession(flag);
  }
```

【注意】确保实体类和数据库字段对应

查询：

1、编写接口方法注解

```
//根据id查询用户
@Select("select * from user where id = #{id}")
User selectUserById(@Param("id") int id);
```

2、测试

```
@Test
public void testSelectUserById() {
   SqlSession session = MybatisUtils.getSession();
   UserMapper mapper = session.getMapper(UserMapper.class);

   User user = mapper.selectUserById(1);
   System.out.println(user);

   session.close();
}
```

新增：

1、编写接口方法注解

```
//添加一个用户
@Insert("insert into user (id,name,pwd) values (#{id},#{name},#{pwd})")
int addUser(User user);
```

2、测试

```
@Test
public void testAddUser() {
   SqlSession session = MybatisUtils.getSession();
   UserMapper mapper = session.getMapper(UserMapper.class);

   User user = new User(6, "秦疆", "123456");
   mapper.addUser(user);

   session.close();
}
```

修改：

1、编写接口方法注解

```
//修改一个用户
```

```
@Update("update user set name=#{name},pwd=#{pwd} where id = #{id}")
int updateUser(User user);
```

2、测试

```
@Test
public void testUpdateUser() {
    SqlSession session = MybatisUtils.getSession();
    UserMapper mapper = session.getMapper(UserMapper.class);

    User user = new User(6, "秦疆", "zxcvbn");
    mapper.updateUser(user);

    session.close();
}
```

删除：

1、编写接口方法注解

```
//根据id删除用
@Delete("delete from user where id = #{id}")
int deleteUser(@Param("id")int id);
```

2、测试

```
@Test
public void testDeleteUser() {
    SqlSession session = MybatisUtils.getSession();
    UserMapper mapper = session.getMapper(UserMapper.class);

    mapper.deleteUser(6);

    session.close();
}
```

【注意点：增删改一定记得对事务的处理】

> 关于@Param

@Param注解用于给方法参数起一个名字。以下是总结的使用原则：

- 在方法只接受一个参数的情况下，可以不使用@Param。
- 在方法接受多个参数的情况下，建议一定要使用@Param注解给参数命名。
- 如果参数是 JavaBean ， 则不能使用@Param。
- 不使用@Param注解时，参数只能有一个，并且是Javabean。

> #与$的区别

- #{} 的作用主要是替换预编译语句(PrepareStatement)中的占位符？【推荐使用】

```
INSERT INTO user (name) VALUES (#{name});
INSERT INTO user (name) VALUES (?);
```

- ${} 的作用是直接进行字符串替换

```
INSERT INTO user (name) VALUES ('${name}');
INSERT INTO user (name) VALUES ('kuangshen');
```

> 使用注解和配置文件协同开发，才是MyBatis的最佳实践！

使用注解开发可以提高我们的开发效率，可以合理使用哦！

end

视频同步更新，这次一定！

“赠人玫瑰，手有余香”

狂神说 的赞赏码

狂神说