Spring整合Mybatis原理

作者：图灵学院周瑜

在介绍Spring整合Mybatis原理之前，我们得先来稍微介绍Mybatis的工作原理。

Mybatis的基本工作原理

在Mybatis中，我们可以使用一个接口去定义要执行sql，简化代码如下：

定义一个接口，@Select表示要执行查询sql语句。

```
public interface UserMapper {
  @Select("select * from user where id = #{id}")
  User selectById(Integer id);
}
```

以下为执行sql代码：

```
InputStream inputStream = Resources.getResourceAsStream("mybatis.xml");
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
SqlSession sqlSession = sqlSessionFactory.openSession();

```

```
// 以下使我们需要关注的重点
UserMapper mapper = sqlSession.getMapper(UserMapper.class);
Integer id = 1;
User user = mapper.selectById(id);
```

Mybatis的目的是：**使得程序员能够以调用方法的方式执行某个指定的sql，将执行sql的底层逻辑进行了封装。**

这里重点思考以下mapper这个对象，当调用SqlSession的getMapper方法时，会对传入的接口生成一个**代理对象**，而程序要真正用到的就是这个代理对象，在调用代理对象的方法时，Mybatis会取出该方法所对应的sql语句，然后利用JDBC去执行sql语句，最终得到结果。

分析需要解决的问题

Spring和Mybatis时，我们重点要关注的就是这个代理对象。因为整合的目的就是：**把某个Mapper的代理对象作为一个bean放入Spring容器中，使得能够像使用一个普通bean一样去使用这个代理对象，比如能被@Autowire自动注入。**

比如当Spring和Mybatis整合之后，我们就可以使用如下的代码来使用Mybatis中的代理对象了：

```
@Component
public class UserService {
    @Autowired
    private UserMapper userMapper;

    public User getUserById(Integer id) {
        return userMapper.selectById(id);
    }
}
```

UserService中的userMapper属性就会被自动注入为Mybatis中的代理对象。如果你基于一个已经完成整合的项目去调试即可发现，userMapper的类型为：org.apache.ibatis.binding.MapperProxy@41a0aa7d。证明确实是Mybatis中的代理对象。

好，那么现在我们要解决的问题的就是：**如何能够把Mybatis的代理对象作为一个bean放入Spring容器中?**

要解决这个，我们需要对Spring的bean生成过程有一个了解。

Spring中Bean的产生过程

Spring启动过程中，大致会经过如下步骤去生成bean

1. 扫描指定的包路径下的class文件
2. 根据class信息生成对应的BeanDefinition
3. 在此处，程序员可以利用某些机制去修改BeanDefinition
4. 根据BeanDefinition生成bean实例
5. 把生成的bean实例放入Spring容器中

假设有一个A类，假设有如下代码：

一个A类：

```
@Component
public class A {
}
```

一个B类，不存在@Component注解

```
public class B {
}
```

执行如下代码：

```
AnnotationConfigApplicationContext context = new AnnotationConfigA
pplicationContext(AppConfig.class);
System.out.println(context.getBean("a"));
```

输出结果为：com.luban.util.A@6acdbdf5

A类对应的bean对象类型仍然为A类。但是这个结论是不确定的，我们可以利用BeanFactory后置处理器来修改BeanDefinition，我们添加一个BeanFactory后置处理器：

```
@Component
public class LubanBeanFactoryPostProcessor implements BeanFactoryP
ostProcessor {
    @Override
```

```
    public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException {
        BeanDefinition beanDefinition = beanFactory.getBeanDefinition("a");
        beanDefinition.setBeanClassName(B.class.getName());
    }
}
```

这样就会导致，原本的A类对应的BeanDefiniton被修改了，被修改成了B类，那么后续正常生成的bean对象的类型就是B类。此时，调用如下代码会报错：

```
context.getBean(A.class);
```

但是调用如下代码不会报错，尽管B类上没有@Component注解：

```
context.getBean(B.class);
```

并且，下面代码返回的结果是：com.luban.util.B@4b1c1ea0

```
AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
System.out.println(context.getBean("a"));
```

之所以讲这个问题，是想说明一个问题：**在Spring中，bean对象跟class没有直接关系，跟BeanDefinition才有直接关系。**

那么回到我们要解决的问题：**如何能够把Mybatis的代理对象作为一个bean放入Spring容器中?**

在Spring中，**如果你想生成一个bean，那么得先生成一个BeanDefinition**，就像你想new一个对象实例，得先有一个class。

解决问题

继续回到我们的问题，我们现在想自己生成一个bean，那么得先生成一个BeanDefinition，只要有了BeanDefinition，通过在BeanDefinition中设置**bean对象的类型**，然后把BeanDefinition添加给Spring，Spring就会根据BeanDefinition自动帮我们生成一个类型对应的bean对象。

所以，现在我们要解决两个问题：

1. **Mybatis的代理对象的类型是什么？因为我们要设置给BeanDefinition**
2. **我们怎么把BeanDefinition添加给Spring容器？**

注意：上文中我们使用的BeanFactory后置处理器，他只能修改BeanDefinition，并不能新增一个BeanDefinition。我们应该使用Import技术来添加一个BeanDefinition。后文再详细介绍如果使用Import技术来添加一个BeanDefinition，可以先看一下伪代码实现思路。

假设：我们有一个UserMapper接口，他的代理对象的类型为UserMapperProxy。
那么我们的思路就是这样的，伪代码如下：

```
BeanDefinitoin bd = new BeanDefinitoin();
bd.setBeanClassName(UserMapperProxy.class.getName());
SpringContainer.addBd(bd);
```

但是，这里有一个严重的问题，就是上文中的UserMapperProxy是我们假设的，他表示一个代理类的类型，然而Mybatis中的代理对象是利用的JDK的动态代理技术实现的，也就是代理对象的代理类是动态生成的，我们根本无法确定代理对象的代理类到底是什么。

所以回到我们的问题：**Mybatis的代理对象的类型是什么？**

本来可以有两个答案：

1. 代理对象对应的代理类
2. 代理对象对应的接口

那么答案1就相当于没有了，因为是代理类是动态生成的，那么我们来看答案2：**代理对象对应的接口**

如果我们采用答案2，那么我们的思路就是：

```
BeanDefinition bd = new BeanDefinitoin();
// 注意这里，设置的是UserMapper
bd.setBeanClassName(UserMapper.class.getName());
SpringContainer.addBd(bd);
```

但是，实际上给BeanDefinition对应的类型设置为一个接口是**行不通**的，因为Spring没有办法根据这个BeanDefinition去new出对应类型的实例，接口是没法直接new出实例的。

那么现在问题来了，我要解决的问题：**Mybatis的代理对象的类型是什么?**
两个答案都被我们否定了，所以这个问题是无解的，所以我们不能再沿着这个思路去思考了，只能回到最开始的问题：**如何能够把Mybatis的代理对象作为一个bean放入Spring容器中?**

总结上面的推理：**我们想通过设置BeanDefinition的class类型，然后由Spring自动的帮助我们去生成对应的bean，但是这条路是行不通的。**

终极解决方案

那么我们还有没有其他办法，可以去生成bean呢？并且**生成bean的逻辑不能由Spring来帮我们做**了，得由我们自己来做。

FactoryBean

有，那就是Spring中的FactoryBean。我们可以利用FactoryBean去自定义我们要生成的bean对象，比如：

```
@Component
public class LubanFactoryBean implements FactoryBean {
    @Override
    public Object getObject() throws Exception {
        Object proxyInstance = Proxy.newProxyInstance(LubanFactoryBean.class.getClassLoader(), new Class[]{UserMapper.class}, new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                if (Object.class.equals(method.getDeclaringClass())) {
                    return method.invoke(this, args);
                } else {
                    // 执行代理逻辑
                    return null;
                }
            }
        });

        return proxyInstance;
    }

```

```
    @Override
    public Class<?> getObjectType() {
        return UserMapper.class;
    }
}
```

我们定义了一个LubanFactoryBean，它实现了FactoryBean，getObject方法就是用来自定义生成bean对象逻辑的。

执行如下代码：

```
public class Test {
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
        System.out.println("lubanFactoryBean: " + context.getBean("lubanFactoryBean"));
        System.out.println("&lubanFactoryBean: " + context.getBean("&lubanFactoryBean"));
        System.out.println("lubanFactoryBean-class: " + context.getBean("lubanFactoryBean").getClass());
    }
}
```

将打印：
lubanFactoryBean: com.luban.util.LubanFactoryBean$1@4d41cee
&lubanFactoryBean: com.luban.util.LubanFactoryBean@3712b94
lubanFactoryBean-class: class com.sun.proxy.$Proxy20

从结果我们可以看到，从Spring容器中拿名字为"lubanFactoryBean"的bean对象，就是我们所自定义的jdk动态代理所生成的代理对象。

所以，我们可以通过FactoryBean来向Spring容器中添加一个自定义的bean对象。上文中所定义的LubanFactoryBean对应的就是UserMapper，表示我们定义了一个LubanFactoryBean，相当于把UserMapper对应的代理对象作为一个bean放入到了容器中。

但是作为程序员，我们不可能每定义了一个Mapper，还得去定义一个LubanFactoryBean，这是很麻烦的事情，我们改造一下LubanFactoryBean，让他变得更通用，比如：

```java
@Component
public class LubanFactoryBean implements FactoryBean {

    // 注意这里
    private Class mapperInterface;
    public LubanFactoryBean(Class mapperInterface) {
        this.mapperInterface = mapperInterface;
    }

    @Override
    public Object getObject() throws Exception {
        Object proxyInstance = Proxy.newProxyInstance(LubanFactoryBean.class.getClassLoader(), new Class[]{mapperInterface}, new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

                if (Object.class.equals(method.getDeclaringClass())) {
                    return method.invoke(this, args);
                } else {
                    // 执行代理逻辑
                    return null;
                }
            }
        });

        return proxyInstance;
    }

    @Override
    public Class<?> getObjectType() {
        return mapperInterface;
    }
}
```

改造LubanFactoryBean之后，LubanFactoryBean变得灵活了，可以在构造LubanFactoryBean时，通过构造传入不同的Mapper接口。

实际上LubanFactoryBean也是一个Bean，我们也可以通过生成一个BeanDefinition来生成一个LubanFactoryBean，并给构造方法的参数设置不同的值，比如伪代码如下：

```
BeanDefinition bd = new BeanDefinitoin();
// 注意一：设置的是LubanFactoryBean
bd.setBeanClassName(LubanFactoryBean.class.getName());
// 注意二：表示当前BeanDefinition在生成bean对象时，会通过调用LubanFactoryBean的构造方法来生成，并传入UserMapper
bd.getConstructorArgumentValues().addGenericArgumentValue(UserMapper.class.getName())
SpringContainer.addBd(bd);
```

特别说一下注意二，表示表示当前BeanDefinition在生成bean对象时，会通过调用LubanFactoryBean的构造方法来生成，并传入UserMapper的Class对象。那么在生成LubanFactoryBean时就会生成一个UserMapper接口对应的代理对象作为bean了。

到此为止，其实就完成了我们要解决的问题：**把Mybatis中的代理对象作为一个bean放入Spring容器中**。只是我们这里是用简单的JDK代理对象模拟的Mybatis中的代理对象，如果有时间，我们完全可以调用Mybatis中提供的方法区生成一个代理对象。这里就不花时间去介绍了。

Import

到这里，我们还有一个事情没有做，就是怎么真正的定义一个BeanDefinition，并把它**添加**到Spring中，上文说到我们要利用Import技术，比如可以这么实现：

定义如下类：

```
public class LubanImportBeanDefinitionRegistrar implements ImportBeanDefinitionRegistrar {

    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry registry) {
        BeanDefinitionBuilder builder = BeanDefinitionBuilder.genericBeanDefinition();
        AbstractBeanDefinition beanDefinition = builder.getBeanDefinition();
```

```
        beanDefinition.setBeanClass(LubanFactoryBean.class);
        beanDefinition.getConstructorArgumentValues().addGenericArgumentValue(UserMapper.class);
        // 添加beanDefinition
        registry.registerBeanDefinition("luban"+UserMapper.class.getSimpleName(), beanDefinition);
    }
}
```

并且在AppConfig上添加@Import注解：

```
@Import(LubanImportBeanDefinitionRegistrar.class)
public class AppConfig {
```

这样在启动Spring时就会新增一个BeanDefinition，该BeanDefinition会生成一个LubanFactoryBean对象，并且在生成LubanFactoryBean对象时会传入UserMapper.class对象，通过LubanFactoryBean内部的逻辑，相当于会自动生产一个UserMapper接口的代理对象作为一个bean。

总结

总结一下，通过我们的分析，我们要整合Spring和Mybatis，需要我们做的事情如下：

1. 定义一个LubanFactoryBean
2. 定义一个LubanImportBeanDefinitionRegistrar
3. 在AppConfig上添加一个注解@Import(LubanImportBeanDefinitionRegistrar.class)

优化

这样就可以基本完成整合的需求了，当然还有两个点是可以优化的

第一，单独再定义一个@LubanScan的注解，如下：

```
@Retention(RetentionPolicy.RUNTIME)
@Import(LubanImportBeanDefinitionRegistrar.class)
public @interface LubanScan {
}
```

这样在AppConfig上直接使用@LubanScan即可

第二，在LubanImportBeanDefinitionRegistrar中，我们可以去扫描Mapper，在LubanImportBeanDefinitionRegistrar我们可以通过AnnotationMetadata获取到对应的@LubanScan注解，所以我们可以在@LubanScan上设置一个value，用来指定待扫描的包路径。然后在LubanImportBeanDefinitionRegistrar中获取所设置的包路径，然后扫描该路径下的所有Mapper，生成BeanDefinition，放入Spring容器中。

所以，到此为止，Spring整合Mybatis的核心原理就结束了，再次总结一下：

1. 定义一个LubanFactoryBean，用来将Mybatis的代理对象生成一个bean对象
2. 定义一个LubanImportBeanDefinitionRegistrar，用来生成不同Mapper对象的LubanFactoryBean
3. 定义一个@LubanScan，用来在启动Spring时执行LubanImportBeanDefinitionRegistrar的逻辑，并指定包路径

以上这个三个要素分别对象org.mybatis.spring中的：

1. MapperFactoryBean
2. MapperScannerRegistrar
3. @MapperScan