**春天的杰作Spring源码分析**

主讲：Cat 老师

北京动力节点教育科技有限公司  
2020 • 北京

动力节点•版权所有•禁止传播

## Spring整合MyBatis涉及三个项目

**Spring Framework：** <https://github.com/spring-projects/spring-framework>

Spring provides everything required beyond the Java programming language for creating enterprise applications for a wide range of scenarios and architectures.

Spring提供了Java编程语言以外的所有所需内容，可用于为各种场景和体系结构创建企业应用程序；

**MyBatis：** <https://github.com/mybatis/mybatis-3>

MyBatis SQL Mapper Framework for Java

MyBatis SQL Mapper Java框架；

**MyBatis-Spring：** <https://github.com/mybatis/spring>

MyBatis Spring Adapter

**注意：**Spring整合MyBatis，如何整合的问题不是Spring框架提供的，而是MyBatis框架自己提供的，MyBatis作者写了一个项目：mybatis-spring，用来整合spring，这是由于历史原因造成的；

Spring容器两个核心步骤：

1. --> 根据配置（xml、注解）创建BeanDefinition（beanName、beanClass）
2. --> 根据BeanDefinition实例化具体的bean对象；

注：如果中间有扩展点，会调用扩展点代码的实现；

# Spring与MyBatis整合核心源码分析

1. 什么是**Bean**PostProcessor，什么是**BeanFactory**PostProcessor？

BeanPostProcessor在Spring容器初始化bean之前和初始化bean之后，干预bean的初始化行为；

BeanFactoryPostProcessor是spring容器注册BeanDefinition时，用来干预BeanDefinition的行为；

1. 什么是FactoryBean，什么是BeanFactory？

FactoryBean是工厂bean，获取对象时候或调getObject（）方法返回对象；

BeanFactory是spring容器；

1. Mapper接口是怎么扫描出来的？

通过MapperScannerConfigurer扫描mapper，该类是一个BeanFactoryPostProcessor，允许注册BeanDefinition；

1. Mapper接口所对应的具体类是什么？

刚扫描的mapper包得到的beanDefinition

里面的beanClass = GoodsMapper.class

然后替换成了class org.mybatis.spring.mapper.MapperFactoryBean

（偷天换日）

OrderMapper.class 都会替换成 MapperFactoryBean.class

1. @Mapper注解和@MapperScan注解有什么联系和区别？

@Mapper注解是一个标记注解，其实没什么用，要结合@MapperScan注解一起才有作用，比如：@MapperScan(value = "com.bjpowernode.mapper", annotationClass = Mapper.class)

有@Mapper注解标记的接口，可以被扫描出来，没有标记的接口就不能扫描出来，当然如果不指定annotationClass属性，则默认是扫描包下的所有接口；

1. Mapper接口是怎么实例化的？

采用MapperFactoryBean代表Mapper接口来创建一个对象；

1. Service中注入的Mapper接口是什么对象？

MapperFactoryBean实现了FactoryBean，当往service中注入Mapper时会从容器中会调用getObject()方法；

@Override  
**public** T getObject() **throws** Exception {  
 **return** getSqlSession().getMapper(**this**.**mapperInterface**);  
}

// jdk动态代理

**protected** T newInstance(MapperProxy<T> mapperProxy) {  
 **return** (T) Proxy.*newProxyInstance*(**mapperInterface**.getClassLoader(),   
**new** Class[] { **mapperInterface** },

mapperProxy);  
}

Mapper接口-->MapperFactoryBean实现类-->getObject()方法-->mybatis底层的动态代理产生代理对象；