https://www.cnblogs.com/redcool/p/6413461.html

[**Spring BeanFactory与FactoryBean的区别及其各自的详细介绍于用法**](https://www.cnblogs.com/redcool/p/6413461.html)

**Spring BeanFactory与FactoryBean的区别及其各自的详细介绍于用法**

**1. BeanFactory**

　　BeanFactory，以Factory结尾，表示它是一个工厂类(接口)，用于管理Bean的一个工厂。在Spring中，BeanFactory是IOC容器的核心接口，它的职责包括：实例化、定位、配置应用程序中的对象及建立这些对象间的依赖。

　　Spring为我们提供了许多易用的BeanFactory实现，XmlBeanFactory就是常用的一个，该实现将以XML方式描述组成应用的对象及对象间的依赖关系。XmlBeanFactory类将持有此XML配置元数据，并用它来构建一个完全可配置的系统或应用。

　　实例化容器

1 Resource resource = new FileSystemResource("beans.xml");

2 BeanFactory factory = new XmlBeanFactory(resource);

1 ClassPathResource resource = new ClassPathResource("beans.xml");

2 BeanFactory factory = new XmlBeanFactory(resource);

1 ApplicationContext context = new ClassPathXmlApplicationContext(new String[] {"applicationContext.xml", "applicationContext-part2.xml"});

3 BeanFactory factory = (BeanFactory) context;

 基本就是这些了，接着使用getBean(String beanName)方法就可以取得bean的实例；BeanFactory提供的方法及其简单，仅提供了六种方法供客户调用：

* boolean containsBean(String beanName) 判断工厂中是否包含给定名称的bean定义，若有则返回true
* Object getBean(String) 返回给定名称注册的bean实例。根据bean的配置情况，如果是singleton模式将返回一个共享实例，否则将返回一个新建的实例，如果没有找到指定bean,该方法可能会抛出异常
* Object getBean(String, Class) 返回以给定名称注册的bean实例，并转换为给定class类型
* Class getType(String name) 返回给定名称的bean的Class,如果没有找到指定的bean实例，则排除NoSuchBeanDefinitionException异常
* boolean isSingleton(String) 判断给定名称的bean定义是否为单例模式
* String[] getAliases(String name) 返回给定bean名称的所有别名

**2. FactoryBean**

　　以Bean结尾，表示它是一个Bean，不同于普通Bean的是：它是实现了FactoryBean<T>接口的Bean，根据该Bean的ID从BeanFactory中获取的实际上是FactoryBean的getObject()返回的对象，而不是FactoryBean本身，如果要获取FactoryBean对象，请在id前面加一个&符号来获取。

　　例如自己实现一个FactoryBean，功能：用来代理一个对象，对该对象的所有方法做一个拦截，在调用前后都输出一行LOG，模仿ProxyFactoryBean的功能。

[复制代码](javascript:void(0);)

1 /\*\*

2 \* my factory bean<p>

3 \* 代理一个类，拦截该类的所有方法，在方法的调用前后进行日志的输出

4 \* @author daniel.zhao

5 \*

6 \*/

7 public class MyFactoryBean implements FactoryBean<Object>, InitializingBean, DisposableBean {

8

9 private static final Logger logger = LoggerFactory.getLogger(MyFactoryBean.class);

10

11 private String interfaceName;

12

13 private Object target;

14

15 private Object proxyObj;

16

17 @Override

18 public void destroy() throws Exception {

19 logger.debug("destroy......");

20 }

21

22 @Override

23 public void afterPropertiesSet() throws Exception {

24 proxyObj = Proxy.newProxyInstance(

25 this.getClass().getClassLoader(),

26 new Class[] { Class.forName(interfaceName) },

27 new InvocationHandler() {

28

29 @Override

30 public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

31 logger.debug("invoke method......" + method.getName());

32 logger.debug("invoke method before......" + System.currentTimeMillis());

33 Object result = method.invoke(target, args);

34 logger.debug("invoke method after......" + System.currentTimeMillis());

35 return result;

36 }

37

38 });

39 logger.debug("afterPropertiesSet......");

40 }

41

42 @Override

43 public Object getObject() throws Exception {

44 logger.debug("getObject......");

45 return proxyObj;

46 }

47

48 @Override

49 public Class<?> getObjectType() {

50 return proxyObj == null ? Object.class : proxyObj.getClass();

51 }

52

53 @Override

54 public boolean isSingleton() {

55 return true;

56 }

57

58 public String getInterfaceName() {

59 return interfaceName;

60 }

61

62 public void setInterfaceName(String interfaceName) {

63 this.interfaceName = interfaceName;

64 }

65

66 public Object getTarget() {

67 return target;

68 }

69

70 public void setTarget(Object target) {

71 this.target = target;

72 }

73

74 public Object getProxyObj() {

75 return proxyObj;

76 }

77

78 public void setProxyObj(Object proxyObj) {

79 this.proxyObj = proxyObj;

80 }

81

82 }

[复制代码](javascript:void(0);)

XML-Bean配置如下

1 <bean id="fbHelloWorldService" class="com.ebao.xxx.MyFactoryBean">

2 <property name="interfaceName" value="com.ebao.xxx.HelloWorldService" />

3 <property name="target" ref="helloWorldService" />

4 </bean>

Junit Test class

[复制代码](javascript:void(0);)

1 @RunWith(JUnit4ClassRunner.class)

2 @ContextConfiguration(classes = { MyFactoryBeanConfig.class })

3 public class MyFactoryBeanTest {

4

5 @Autowired

6 private ApplicationContext context;

7

8 /\*\*

9 \* 测试验证FactoryBean原理，代理一个servcie在调用其方法的前后，打印日志亦可作其他处理

10 \* 从ApplicationContext中获取自定义的FactoryBean

11 \* context.getBean(String beanName) ---> 最终获取到的Object是FactoryBean.getObejct(),

12 \* 使用Proxy.newInstance生成service的代理类

13 \*/

14 @Test

15 public void testFactoryBean() {

16 HelloWorldService helloWorldService = (HelloWorldService) context.getBean("fbHelloWorldService");

17 helloWorldService.getBeanName();

18 helloWorldService.sayHello();

19 }

20 }

[复制代码](javascript:void(0);)

其实FactoryBean这栋特点，可以实现很多有用的功能，实现大家多多评论多多补充，一起探讨。

分类: [Spring IOC](https://www.cnblogs.com/redcool/category/934282.html)

标签: [BeanFactory](https://www.cnblogs.com/redcool/tag/BeanFactory/), [FactoryBean](https://www.cnblogs.com/redcool/tag/FactoryBean/)