装配Bean 基于XML

1.实例化方式

- 3种bean实例化方式：默认构造，静态工厂，实例工厂
- 默认构造
- 静态工厂

我操，我静态工厂用不了~算了不纠结这个了，继续过下去
**********搞定了，是版本问题，在maven上用了高版本的jar包，能够通过测试了

首先创建接口UserService

MyBeanFactory.java | TestStaticFactory.java | UserService.java | UserServiceImpl.java | beans.xml

```
1  package com.mccken.c_inject.b_static_factory;
2
3  public interface UserService {
4      public void addUser();
5  }
6
```

再创建接口实现类UserServiceImpl

MyBeanFactory.java | TestStaticFactory.java | UserService.java | UserServiceImpl.java | beans.xml

```
1  package com.mccken.c_inject.b_static_factory;
2
3  public class UserServiceImpl implements UserService {
4
5      public void addUser() {
6          System.out.println("**********inject.b_static_factory");
7      }
8
9  }
10
```

下面创建静态工厂，实现一个静态方法~返回实现类

MyBeanFactory.java | TestStaticFactory.java | UserService.java | UserServiceImpl.java | beans.xml

```
1  package com.mccken.c_inject.b_static_factory;
2
3
4  public class MyBeanFactory {
5
6      /*
7       * 创建实例
8       */
9      public static UserServiceImpl createService() {
10         return new UserServiceImpl();
11     }
12
13 }
14
```

其中设置<bean>为：
<bean id="userServiceId" class="com.mccken.c_inject.b_static_factory.MyBeanFactory"

factory-method="createService"></bean>

```
<!--
    将静态工厂创建的实例交与spring
        class确定静态工厂全限定类名
        factory-method确定静态方法类名
-->
<bean id="userServiceId" class="com.mccken.c_inject.b_static_factory.MyBeanFactory"
 factory-method="createService"></bean>
```

class为静态工厂全限定类名

factory-method确定静态工厂中的静态方法类名

然后照旧写test类，测试静态工厂模式创建实例

```
@Test
public void demo02(){
    //静态工厂
    String xmlPath = "com/mccken/c_inject/b_static_factory/beans.xml";
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
    UserService userService = applicationContext.getBean("userServiceId",UserService.class);
    userService.addUser();

}
```

此处在applicationContext.getBean()中，添加了UserService.class，就不用强转了

- 实例工厂

必须先有工厂的实例对象，通过实例对象创建对象，提供的所有方法都是非静态的

- <bean>

```
<!-- 创建工厂实例 -->
<bean id="myBeanFactoryId" class="com.mccken.c_inject.c_factory.MyBeanFactory"></bean>
<!-- 获得userservice
    *factory-bean 确定工厂实例
    *factory-method确定普通方法
-->
<bean id="userServiceId" factory-bean="myBeanFactoryId" factory-method="createService"></bean>
```

- 实例工厂，相当于把静态工厂的静态方法static删掉就行了

```
package com.mccken.c_inject.c_factory;
/*
 * 实例工厂
 */

public class MyBeanFactory {

    /*
     * 创建实例
     */
    public UserServiceImpl createService() {
        return new UserServiceImpl();
    }

}
```

- test几乎没有改变

```java
@Test
public void demo02(){
    //实例工厂
    String xmlPath = "com/mccken/c_inject/c_factory/beans.xml";
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
    UserService userService = applicationContext.getBean("userServiceId",UserService.class);
    userService.addUser();

}
```

****

****

.2.bean种类

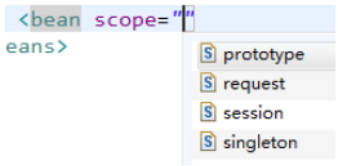
- 普通bean：<bean id=" " calss="A" >,spring直接创建A实例，并返回
- FactoryBean:是一个特殊的bean，具有工厂生成对象能力，只能生成特定的对象

- 普通 bean：之前操作的都是普通 bean。<bean id="" class="A"> ， spring 直接创建 A 实例，并返回
- FactoryBean：是一个特殊的 bean，具有工厂生成对象能力，只能生成特定的对象。
  bean 必须使用 FactoryBean 接口，此接口提供方法 getObject() 用于获得特定 bean。
  <bean id="" class="FB"> 先创建 FB 实例，使用调用 getObject()方法，并返回方法的返回值
  FB fb = new FB();
  return fb.getObject();
- BeanFactory 和 FactoryBean 对比？
  BeanFactory：工厂，用于生成任意 bean。
  FactoryBean：特殊 bean，用于生成另一个特定的 bean。例如：ProxyFactoryBean ，此工厂 bean 用于生产代理。<bean id="" class="....ProxyFactoryBean"> 获得代理对象实例。AOP 使用

.3.作用域

- 作用域：用于确定spring创建bean实例个数

Bean的作用域



类别	说明
singleton	在Spring IoC容器中仅存在一个Bean实例，Bean以单例方式存在，默认值
prototype	每次从容器中调用Bean时，都返回一个新的实例，即每次调用getBean()时，相当于执行new XxxBean()
request	每次HTTP请求都会创建一个新的Bean，该作用域仅适用于WebApplicationContext环境
session	同一个HTTP Session 共享一个Bean，不同Session使用不同Bean，仅适用于WebApplicationContext 环境
globalSession	一般用于Portlet应用环境，该作用域仅适用于WebApplicationContext 环境

- 取值

  singleton 单例，默认值
  prototype 多例，没执行一次getBean将获得一个实例。例如:struts整合spring，配置action多例

  - 配置信息

<bean id="" class="" scope="">

结果

-

```
<bean id="userServiceId" class="com.mccken.d_scope.UserServiceImpl" scope="prototype"></bean>
```

- 单例模式
  - bean默认

```
<bean id="userServiceId" class="com.mccken.d_scope.UserServiceImpl"></bean>
```

  - Test中创建两个不一样的实例，再查看得到的两个对象

```java
@Test
public void demo02(){
    //单例模式
    String xmlPath = "com/mccken/d_scope/beans.xml";
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext(xmlPath);
    UserService userService = applicationContext.getBean("userServiceId",UserService.class);
    UserService userService2 = applicationContext.getBean("userServiceId",UserService.class);

    System.out.println(userService);
    System.out.println(userService2);

}
```

- 两个实例得到的对象为同一对象~

```
四月 16, 2017 5:34:19 下午 org.springframework.context
信息: Refreshing org.springframework.context.support
四月 16, 2017 5:34:19 下午 org.springframework.beans.fa
信息: Loading XML bean definitions from class path r
com.mccken.d_scope.UserServiceImpl@9838eb
com.mccken.d_scope.UserServiceImpl@9838eb
```

- 多例模式

在单例模式的基础上，添加个scope="prototype",

```
<bean id="userServiceId" class="com.mccken.d_scope.UserServiceImpl" scope="prototype"></bean>
```

由此得到的结果为两个不同的对象

```
四月 16, 2017 5:39:18 下午 org.springframework.context
信息: Refreshing org.springframework.context.support
四月 16, 2017 5:39:18 下午 org.springframework.beans.fa
信息: Loading XML bean definitions from class path r
com.mccken.d_scope.UserServiceImpl@1ef45e3
com.mccken.d_scope.UserServiceImpl@18067a5
```