实验九 **Spring** 的核心机制：控制反转（**IoC**）——登录用户

的购物车

一、基础实验——**Spring** 容器中的依赖注入

# （一）实验目的

1. 进一步掌握 Spring 环境搭建的基本方法，能熟练地在 JAVASE 应用中使用

Spring，并能熟练地在 Eclipse 中开发 Spring 应用；

1. 深入理解 Spring 的核心机制：控制反转 IoC（Inversion of Control）与依赖注入 DI（Dependency Injection）；
2. 深入理解 Spring 配置文件的作用，掌握配置文件中各主要元素及其属性的作用和基本配置方法。

# （二）基本知识与原理

1. Spring 中主要有两种注入方法：设置注入和构造器注入；
2. 设置注入是指 IoC 容器使用属性的 setter 方法来注入被依赖的实例，这种注入方式简单、直观，因而在 Spring 的依赖注入里大量使用；
3. 使用设置注入的配置文件，在 bean 元素下用 property 元素指定属性名，其中 property元素的name 属性值必须与对应的setter方法名对应，进而调用setter 方法注入具体值；
4. 构造器注入是指 IoC 容器通过调用带参的构造方法注入所依赖的属性，这种方式在构造实例时已经为其完成了依赖关系的初始化；
5. 使用构造器注入的配置文件，在 bean 元素下用 constructor-arg 元素表示构造方法的参数，其中 constructor-arg 元素的 index 属性表示构造方法中参数的索引值；
6. 无论是设置注入和构造器注入，都要为方法指定具体的参数值为属性赋值，参数值有不同的类型，可以分为三种情况：基本数据类型和 String 类型、其他 bean 类型、null 值；
7. 当类的属性是集合类型时，也可以使用 IoC 进行注入，常用的集合类型有 List、

Set、Map 以及 Properties，相应地配置文件可以使用<list>、<set>、<map>和 <props>元素进行配置。

# （三）实验内容及步骤

1. 在 Eclipse 中新建 Java 工程 spring-prj2，并添加 common-logging-1.2.jar 和 Spring 的 4 个基础 JAR 包到工程中（可参考实验八 SSH 整合的基础实验中的实验步骤二）；
2. 在 spring-prj2 中新建 cn.edu.zjut.bean 包，并在其中创建 IItem 接口（代码略）及其实现类 Item（代码片段如下）；

|  |
| --- |
| package cn.edu.zjut.bean; public class Item implements IItem{  private String itemID; private String title; private String description; private double cost;  **public Item(String itemID, String title,**  **String description, double cost) {**  **this.itemID = itemID; this.title = title; this.description = description; this.cost = cost;**  **System.out.println("create Item."); }**  //省略getters/setters方法 } |

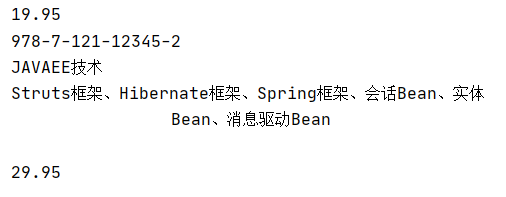
1. 在 spring-prj2 中创建 Spring 配置文件 applicationContext.xml，并在其中使用构造器注入的方式配置 Item 实例，具体代码如下：

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:p="http://www.springframework.org/schema/p"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-4.0.xsd">  <bean id="item1" class="cn.edu.zjut.bean.Item">  <constructor-arg index="0" type="java.lang.String"> <value>978-7-121-12345-1</value>  </constructor-arg> |
| <constructor-arg index="1" type="java.lang.String">  <value>JAVAEE技术实验指导教程</value>  </constructor-arg>  <constructor-arg index="2" type="java.lang.String">  <value>WEB程序设计知识回顾、轻量级JAVAEE应用框架、企业级EJB组  件编程技术、JAVAEE综合应用开发</value>  </constructor-arg>  <constructor-arg index="3" type="double"> <value>19.95</value>  </constructor-arg>  </bean>  <bean id="item2" class="cn.edu.zjut.bean.Item">  <constructor-arg index="0" type="java.lang.String"> <value>978-7-121-12345-2</value>  </constructor-arg>  <constructor-arg index="1" type="java.lang.String">  <value>JAVAEE技术</value>  </constructor-arg>  <constructor-arg index="2" type="java.lang.String">  <value>Struts框架、Hibernate框架、Spring框架、会话Bean、实体  Bean、消息驱动Bean</value>  </constructor-arg>  <constructor-arg index="3" type="double"> <value>29.95</value>  </constructor-arg>  </bean>  </beans> |

1. 在 spring-prj2 中新建 cn.edu.zjut.app 包，并在其中创建测试类 SpringEnvTest，对构造器注入进行测试，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.app; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext; import cn.edu.zjut.dao.ICustomerDAO;  public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IItem item1 = (IItem) ctx.getBean("item1");  System.out.println(item1.getItemID());  System.out.println(item1.getTitle()); |
| System.out.println(item1.getDescription()); System.out.println(item1.getCost());  IItem item2 = (IItem) ctx.getBean("item2");  System.out.println(item2.getItemID());  System.out.println(item2.getTitle());  System.out.println(item2.getDescription());  System.out.println(item2.getCost()); }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 在 spring-prj2 的 cn.edu.zjut.bean 包中创建 IItemOrder 接口（代码略）及其实现类 ItemOrder（代码片段如下）；

|  |
| --- |
| package cn.edu.zjut.bean;  public class ItemOrder implements IItemOrder { private IItem item; private int numItems;  public void incrementNumItems() { setNumItems(getNumItems() + 1);  } public void cancelOrder() { setNumItems(0);  } public double getTotalCost() { return (getNumItems() \* getUnitCost()); }  //省略getters/setters方法 } |

1. 修改 Spring 配置文件 applicationContext.xml，在其中使用设置注入的方式增加 ItemOrder 实例的配置，代码片段如下：

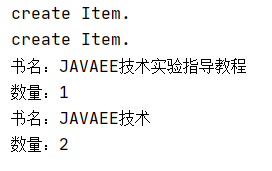
|  |
| --- |
| <bean id="itemorder1" class="cn.edu.zjut.bean.ItemOrder">  <property name="numItems"> <value>1</value> </property>  <property name="item"> <ref bean="item1"/> </property>  </bean>  <bean id="itemorder2" class="cn.edu.zjut.bean.ItemOrder">  <property name="numItems"> <value>2</value> </property>  <property name="item"> <ref bean="item2"/> </property>  </bean> |

1. 修改测试类 SpringEnvTest，对构造器注入进行测试，代码片段如下：

public class SpringEnvTest {

|  |
| --- |
| public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IItemOrder itemorder1 = (IItemOrder) ctx.getBean("itemorder1");  System.out.println("书名："+itemorder1.getItem().getTitle());  System.out.println("数量：" + itemorder1.getNumItems());  IItemOrder itemorder2 = (IItemOrder) ctx.getBean("itemorder2");  System.out.println("书名："+itemorder2.getItem().getTitle());  System.out.println("数量：" + itemorder2.getNumItems());  }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 在 spring-prj2 的 cn.edu.zjut.bean 包中创建购物车类 ShoppingCart.java，其中包含集合类型（List）属性，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.bean; import java.util.\*;  public class ShoppingCart { **private List itemsOrdered;**  public List getItemsOrdered() { return (itemsOrdered);  } public void setItemsOrdered(List itemsOrdered) { this.itemsOrdered = itemsOrdered; }  } |

1. 修改 Spring 配置文件 applicationContext.xml，在其中增加 ShoppingCart 实例的配置，并使用 list 元素对 List 类型属性进行配置，代码片段如下：

|  |
| --- |
| <bean id="shoppingcart" class="cn.edu.zjut.bean.ShoppingCart"> <property name="itemsOrdered">  **<list>**  **<ref bean="itemorder1"/>**  **<ref bean="itemorder2"/>**  **</list>**  </property>  </bean> |

1. 修改测试类 SpringEnvTest，对集合类型属性配置进行测试，代码略；
2. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；

代码和结果



1. 对其它三种集合类型（Set、Map、Properties）进行配置和测试，并记录运行结果。

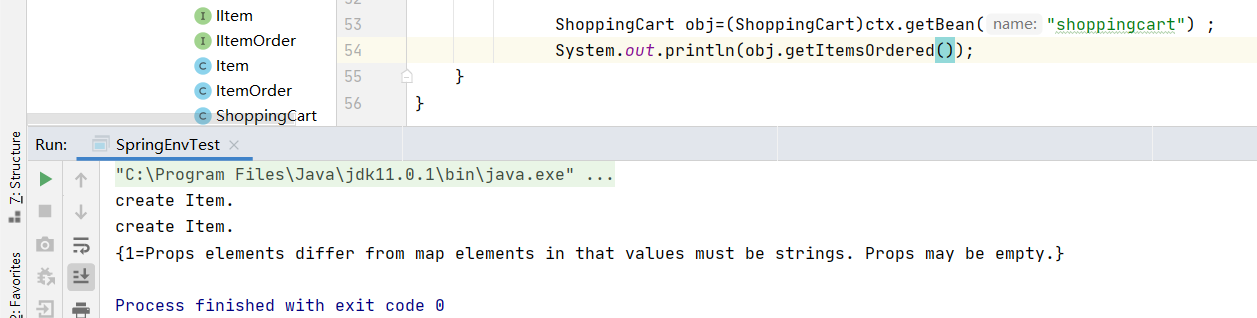
改用Set的结果



改用Map的结果



改用Properties的结果



# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 根据实验过程，比较并总结设置注入与构造器注入各自的优点及适用的场景，并记录下来；

相比而言设置注入具有以下优点：

                   1）与传统的JavaBean的写法更相似，程序开发人员更容易理解、接受。通过setter方法设定依赖关系显得更加直观、自然。

                   2)对于复杂的依赖关系，如果采用构造注入，会导致构造器过于臃肿，难以阅读。Spring在创建Bean实例时，需要同时实例化其依赖的全部实例，因而导致性能下降。而是用设置注入可以避免这些问题。

                   3）尤其在某些属性可选的情况下，多参数的构造器更加笨重。

某些情况下，构造注入的优势：

                   1）构造注入可以再构造器中决定依赖关系的注入顺序，有限依赖的优先注入。例如，组件中其它依赖关系的注入，常常需要依赖于Datasource的注入。采用构造注入，可以在代码中清晰地决定注入顺序。

                   2）对于依赖关系无需变化的Bean，构造注入更加有用。因为没有setter方法，所有的依赖关系全部在构造器内设定。因此，无需担心后续代码对依赖关系的破坏。

                   3）依赖关系只能在构造器中设定，则只有组建的创建者才能改变组建的依赖关系。队组建的调用者而言，组件内部的依赖关系完全透明，更符合高内聚的原则。

1. 根据实验过程，查找相关资料，总结在设置注入或构造器注入方式下，配置文件中相应的配置方法，以及所使用的相关元素及其属性的作用，并记录下来；

先在主板类生成变量的set和get方法，再去配置文件中配置bean，property是类中的属性，value为设置的值。

构造器注入分为索引注入和类型注入。在配置文件中，不再用<property>标签，而是用<constructor-arg>标签代替，其中索引序号和类型顺序不能乱：

1. 根据实验步骤 10-14，查找相关资料，总结集合类型属性的配置方法，并记录下来；

集合分为Set、Map、Properties、list，配置文件中property的类型要和对应的类中的成员变量的类型相同

1. 碰到的问题及解决方案或思考；
2. 实验收获及总结。

2、上交程序源代码，代码中应有相关注释。

二、提高实验——**Spring** 容器中的 **Bean**

# （一）实验目的

1. 理解 Spring IoC 容器中 bean 的五个作用域所起到的作用，以及使用不同作用域的 bean 之间的区别；
2. 掌握 Spring IoC 容器创建 bean 实例的三种方式；
3. 掌握 Spring 在 bean 的依赖关系注入之后执行特定行为的方法，会通过使用 init-method 属性的方式以及让 Bean 类实现 InitializingBean 接口的方式，添加 bean 的依赖关系注入之后的行为；
4. 掌握 Spring 在 bean 销毁之前执行特定行为的方法，会通过使用 destroy-method 属性的方式以及让 Bean 类实现 DisposableBean 接口的方式，添加 bean 销毁之前的行为；
5. 理解 Spring IoC 容器对 bean 生命周期的管理。

# （二）基本知识与原理

1. Spring 框架的 IoC 容器不仅可以向 bean 注入不同的依赖属性，还可以指定其作用域用以告诉容器如何生成该 bean，bean 有五种作用域，分别是 singleton、 prototype、request、session 和 global session；
2. Spring IoC 容器创建 bean 实例通常有三种方法：
3. 调用构造器创建 bean 实例；
4. 调用静态工厂方法创建 bean 实例；
5. 调用实例工厂方法创建 bean 实例；
6. Spring 可以管理 singleton 作用域 bean 的生命周期，并允许在 bean 的依赖关系注入之后以及 bean 销毁之前执行特定的行为；
7. 管理 bean 的依赖关系注入之后的行为可以有两种方式：
8. 使用 init-method 属性指定某个方法应在 bean 全部依赖关系设置结束后自动执行；
9. 让 Bean 类实现 InitializingBean 接口，并使用该接口提供的 afterPropertiesSet 方法；

5、管理 bean 销毁之前的行为也可以有两种方式：

1. 使用 destroy-method 属性指定某个方法应在 bean 销毁之前自动执行；
2. 让 Bean 类实现 DisposableBean 接口，并使用该接口提供的 destroy 方法。

# （三）实验内容及步骤

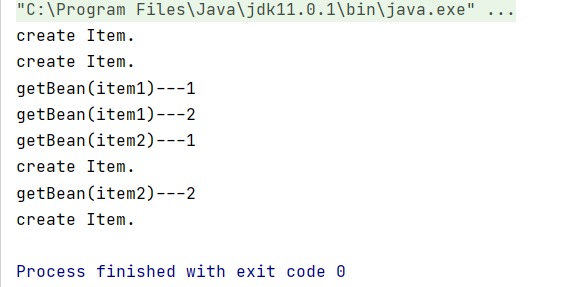
1. 修改 Spring 配置文件 applicationContext.xml，为 Item 实例的配置添加作用域属性（scope），代码片段如下：

|  |
| --- |
| <bean id="item1" class="cn.edu.zjut.bean.Item" scope="singleton">  <constructor-arg index="0" type="java.lang.String"> <value>978-7-121-12345-1</value>  </constructor-arg>  ……  </bean>  <bean id="item2" class="cn.edu.zjut.bean.Item" scope="prototype">  <constructor-arg index="0" type="java.lang.String"> <value>978-7-121-12345-2</value>  </constructor-arg>  ……  </bean> |

1. 修改测试类 SpringEnvTest，对 bean 的作用域进行测试，代码片段如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml"); |
| System.out.println("getBean(item1)---1");  IItem item11 = (IItem) ctx.getBean("item1"); System.out.println("getBean(item1)---2");  IItem item12 = (IItem) ctx.getBean("item1"); System.out.println("getBean(item2)---1");  IItem item21 = (IItem) ctx.getBean("item2"); System.out.println("getBean(item2)---2");  IItem item22 = (IItem) ctx.getBean("item2"); }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



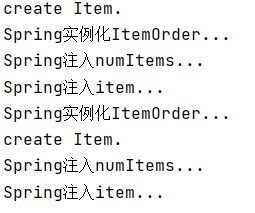
1. 修改 cn.edu.zjut.bean 包中的 ItemOrder.java，在其构造方法及 setters 方法中添加相应的输出，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.bean;  public class ItemOrder implements IItemOrder, **InitializingBean** { private IItem item; private int numItems;  public ItemOrder() {  System.out.println("Spring实例化ItemOrder...");  }  public void setItem(IItem item) {  System.out.println("Spring注入item..."); this.item = item;  }  public void setNumItems(int n) {  System.out.println("Spring注入numItems..."); this.numItems = n;  }  ……  } |

1. 修改测试类 SpringEnvTest，对 bean 的初始化方法进行测试，代码片段如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IItemOrder itemorder1 = (IItemOrder)ctx.getBean("itemorder1");  }  } |

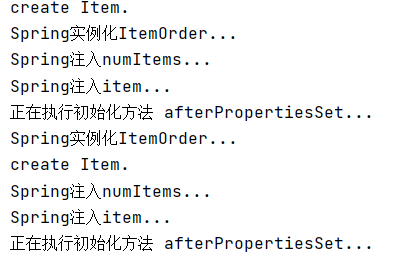
1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 修改 cn.edu.zjut.bean 包中的 ItemOrder.java，通过实现 InitializingBean 接口的方式，添加 bean 的依赖关系注入之后的行为，代码片段如下：

|  |
| --- |
| package cn.edu.zjut.bean;  public class ItemOrder implements IItemOrder, **InitializingBean** { **public void afterPropertiesSet() throws Exception {**  **System.out.println("**正在执行初始化方法 **afterPropertiesSet..."); }**  ……  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 修改 cn.edu.zjut.bean 包中的 ItemOrder.java，通过自定义方法，添加 bean 的依赖关系注入之后的行为，代码片段如下：

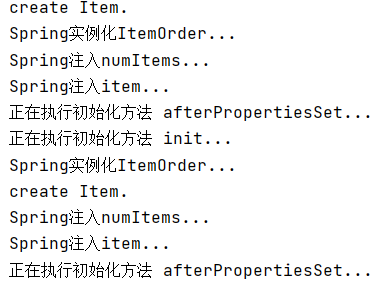
|  |
| --- |
| package cn.edu.zjut.bean;  public class ItemOrder implements IItemOrder, InitializingBean { **public void init() {**  **System.out.println("**正在执行初始化方法 **init..."); }**  ……  } |

1. 修改 Spring 配置文件 applicationContext.xml，通过使用 init-method 属性的方

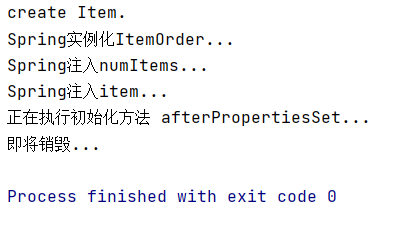
式，为 ItemOrder 实例添加 bean 的依赖关系注入之后的行为，代码片段如下：

|  |
| --- |
| <bean id="itemorder1" class="cn.edu.zjut.bean.ItemOrder" **init-method="init"**>  <property name="numItems"> <value>1</value> </property>  <property name="item"> <ref bean="item1"/> </property>  </bean> |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



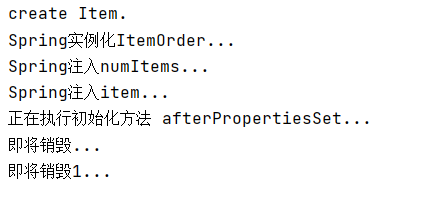
1. 修改 cn.edu.zjut.bean 包中的 ItemOrder.java，通过实现 DisposableBean 接口的方式，添加 bean 销毁之前的行为（代码略）；
2. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 修改 cn.edu.zjut.bean 包中的 ItemOrder.java，通过自定义方法，添加 bean 销毁之前的行为（代码略）；并修改 Spring 配置文件 applicationContext.xml，通过使用 destroy-method 属性的方式，为 ItemOrder 实例添加 bean 销毁之前

的行为（代码略）；

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 在工程spring-prj2的cn.edu.zjut.bean包中创建工厂类ItemOrderFactory.java，并在类中提供静态工厂方法返回 ItemOrder 实例，具体代码如下：

package cn.edu.zjut.bean;

|  |
| --- |
| public class ItemOrderFactory { public static ItemOrder createItemOrder() {  System.out.println("调用静态工厂方法创建bean..."); return new ItemOrder();  }  } |

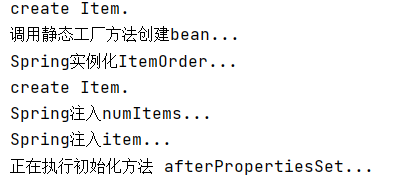
1. 修改 Spring 配置文件 applicationContext.xml，使用 factory-method 属性调用静态工厂方法，创建 ItemOrder 实例，代码片段如下：

|  |
| --- |
| <bean id="itemorder2" **class="cn.edu.zjut.bean.ItemOrderFactory" factory-method="createItemOrder"**>  <property name="numItems"> <value>1</value> </property>  <property name="item"> <ref bean="item1"/> </property>  </bean> |

1. 修改测试类 SpringEnvTest，对静态工厂方法创建 bean 的方法进行测试，代码片段如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  IItemOrder itemorder2 = (IItemOrder)ctx.getBean("itemorder2");  }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 修改工厂类 ItemOrderFactory.java，将其中的静态工厂方法修改为实例工厂方法，具体代码如下：

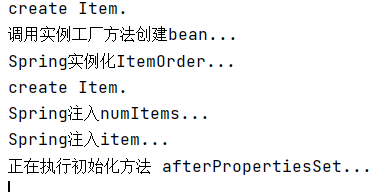
|  |
| --- |
| package cn.edu.zjut.bean;  public class ItemOrderFactory { public ItemOrder createItemOrder() {  System.out.println("调用实例工厂方法创建bean..."); return new ItemOrder();  }  } |

1. 修改 Spring 配置文件 applicationContext.xml，使用 factory-method 属性和 factory-bean 属性调用实例工厂方法，创建 ItemOrder 实例，代码片段如下：

|  |
| --- |
| **<bean id="itemorderFactory" class="cn.edu.zjut.bean.ItemOrderFactory"/>**  <bean id="itemorder2" class="cn.edu.zjut.bean.ItemOrder" **factory-method="createItemOrder" factory-bean="itemorderFactory"**>  <property name="numItems"> <value>1</value> </property>  <property name="item"> <ref bean="item1"/> </property> |

</bean>

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果。



# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 根据实验步骤 1-3，查找相关资料，总结 Spring IoC 容器中 bean 的五个作用域（singleton、prototype、request、session、global session），以及使用不同作用域的 bean 之间的区别，并记录下来；

#### singleton：单例模式，在整个Spring IoC容器中，使用singleton定义的Bean将只有一个实例。

#### prototype：原型模式，每次通过容器的getBean方法获取prototype定义的Bean时，都将产生一个新的Bean实例

#### request：对于每次HTTP请求，使用request定义的Bean都将产生一个新实例，即每次HTTP请求将会产生不同的Bean实例。只有在Web应用中使用Spring时，该作用域才有效

#### session：对于每次HTTP Session，使用session定义的Bean豆浆产生一个新实例。同样只有在Web应用中使用Spring时，该作用域才有效

#### global session：每个全局的HTTP Session，使用session定义的Bean都将产生一个新实例。典型情况下，仅在使用portlet context的时候有效。同样只有在Web应用中使用Spring时，该作用域才有效

1. 根据实验步骤 4-6，查找相关资料，总结 Spring IoC 容器管理的 bean 的生命周期，并记录下来；

* 通过构造器或工厂方法创建 Bean 实例
* 为 Bean 的属性设置值和对其他 Bean 的引用
* 调用 Bean 的初始化方法
* 当容器关闭时, 调用 Bean 的销毁方法
* 在 Bean 的声明里设置 init-method 和 destroy-method 属性, 为 Bean 指定初始化和销毁方法.

1. 根据实验步骤 7-15，查找相关资料，总结 Spring 如何管理在 bean 的依赖关系注入之后以及 bean 销毁之前执行特定的行为，总结 Spring 执行 Bean 实例化、依赖关系注入、初始化方法和析构方法的顺序，并记录下来；

一种是实现InitializingBean和DisposableBean接口，一种是在bean用initmethod和destroymethod设置自定义的方法执行特定行为。

顺序是bean实例化最先，然后初始化对象，之后依赖注入，最后析构

1. 根据实验步骤 4-6、16-22，查找相关资料，总结 Spring IoC 容器创建 bean 实例的三种方式，及相应的配置过程，并记录下来；

**利用无参构造函数 + setter方法注入值构造Bean对象实例**

其本质是 SpringContext 利用无参的构造函数创建一个对象，然后利用setter方法赋值，所以需要在pojo中提供无参数构造函数，并定义对应字段的setter方法。如果无参构造函数不存在，Spring上下文创建对象的时候便会报错

setter方法注入值时，xml配置bean时使用的 <property name="属性名" value="值" ></property>标签对属性进行赋值

Spring容器通过该方法为name属性注入参数。

### ****利用有参构造函数直接注入值构造Bean对象实例****

当利用有参构造注入值构造时，pojo中不需要setter方法；

在构造注入值时，xml配置bean时通过使用<constructor-arg name="属性名" value="值"></constructor-arg>标签的name属性和value属性指定了：构造方法赋值

**静态工厂方法创建bean对象的核心是： class + factory-method**

首先是封装类对象，如果需要对属性注入值，则需要定义有参构造方法或者生产setter方法；

其次是定义静态工厂方法。使用静态工厂方法创建Bean实例需要为<bean />元素指定出id外如下属性：

class：指定静态工厂的实现类( 即包名.类名 )【相当于指定工厂的地址】

factory-method：指定由静态工厂的哪个方法创建该Bean实例（方法名）【指定由工厂的哪个车间创建Bean】

如果静态工厂方法需要参数，则使用<constructor-arg />元素传入

## ****调用实例工厂方法创建Bean对象****

调用实例工厂方法创建bean对象相比较静态工厂方法创建bean对象，就是将静态方法中的**factory**实现类先实例化一个factorBean对象，然后调用factorBean对象来创建bean对象；调用实例化工厂需要为<bean />指以下两个属性

factory-bean ：该属性指定工厂Bean的id

factory-method：该属性指定实例工厂的工厂方法。

如果方法需要参数，则使用<constructor-arg />元素传入

1. 碰到的问题及解决方案或思考；
2. 实验收获及总结。

2、上交程序源代码，代码中应有相关注释。

三、扩展实验——深入 **Spring** 容器

# （一）实验目的

1. 理解 Spring 中 BeanFactory 的作用，掌握使用 XmlBeanFactory 创建 BeanFactory 实 例 的 方 法 ， 掌 握 使 用 FileSystemXmlApplicationContext 或

ClassPathXmlApplicationContext 创建 ApplicationContext 实例的方法；

1. 理解 Spring 中 ApplicationContext 对国际化的支持，并掌握国际化的基本步骤；
2. 理解 Spring 的事件机制，掌握 Spring 完成事件监听及处理的基本步骤。

# （二）基本知识与原理

1. Spring 容器最基本的接口是 BeanFactory，BeanFactory 负责配置、创建、管理 bean，包括管理 bean 与 bean 之间的依赖关系，BeanFactory 有一个实现类： org.springframework.beans.factory.xml.XmlBeanFactory；
2. ApplicationContext 是 BeanFactory 的子接口，对于大部分 JavaEE 应用而言，使用它作为 Spring 容器更方便， ApplicationContext 的常用实现类是

FileSystemXmlApplicationContext 和 ClassPathXmlApplicationContext；

1. ApplicationContext 接口继承 MessageSource 接口，用以实现国际化功能；
2. Spring 支持事件机制，通过 ApplicationEvent 类和 ApplicationListener 接口，可以实现 ApplicationContext 的事件处理。

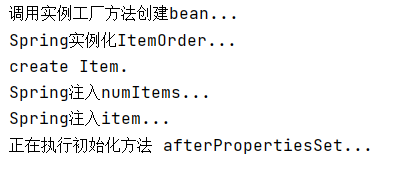
# （三）实验内容及步骤

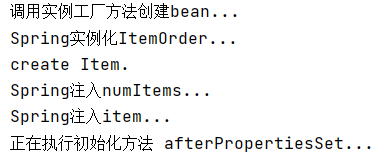
1. 修改测试类 SpringEnvTest，分别使用 XmlBeanFactory 创建 BeanFactory 实例，或使用 FileSystemXmlApplicationContext、ClassPathXmlApplicationContext 创建

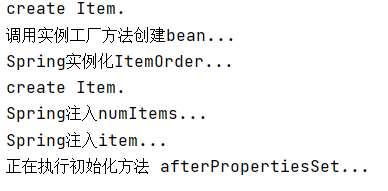
ApplicationContext 实例，代码片段如下：

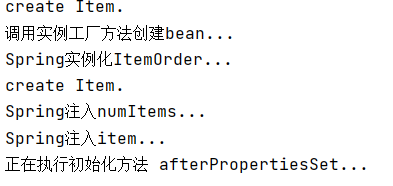
|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) { **FileSystemResource isr = new**  **FileSystemResource("src/applicationContext.xml");**  **XmlBeanFactory factory = new XmlBeanFactory(isr);**  IItemOrder itemorder3 = (IItemOrder) factory.getBean("itemorder3"); }  } |
| public class SpringEnvTest { public static void main(String[] args) { **ClassPathResource res = new**  **ClassPathResource("applicationContext.xml");**  **XmlBeanFactory factory = new XmlBeanFactory(res);**  ……  }  } |
| public class SpringEnvTest { public static void main(String[] args) {  **ApplicationContext ctx = new ClassPathXmlApplicationContext(**  **"applicationContext.xml");**  IItemOrder itemorder3 = (IItemOrder)ctx.getBean("itemorder3");  }  } |
| public class SpringEnvTest { public static void main(String[] args) {  **ApplicationContext ctx = new FileSystemXmlApplicationContext( "src/applicationContext.xml");**  ……  }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；









1. 在 spring-prj2 中新建 cn.edu.zjut.local 包，并在其中创建简体中文的资源文件： message\_zh\_CN.properties，在其中输入“HelloWorld={0}，现在是{1}”，并对其进行转码，最终得到：

HelloWorld={0}\uFF0C\u73B0\u5728\u662F{1}

1. 在 cn.edu.zjut.local 包中创建美式英语的资源文件：message\_en\_US.properties，具体代码如下：

HelloWorld={0},now is{1}

1. 修改 Spring 配置文件 applicationContext.xml，增加名为 messageSource 的 bean 实例以完成国际化的配置，代码片段如下：

|  |
| --- |
| <bean id="messageSource" class=  "org.springframework.context.support.ResourceBundleMessageSource"> <property name="basenames">  <list>  <value>local/message</value>  <!-- 如果有多个资源文件，全部列在此处 -->  </list>  </property>  </bean> |

1. 修改测试类 SpringEnvTest，对 ApplicationContext 对国际化的支持进行测试，代码片段如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  Object[] objects = new Object[]{"HelloWorld",new Date()};  String message= ctx.getMessage("HelloWorld",objects,Locale.CHINA);  // String message=  // ctx.getMessage("HelloWorld",objects,Locale.US);  System.out.println(message); }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果； 8、修改测试类 SpringEnvTest，将其中的“Locale.CHINA”替换成“Locale.US”，运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果；



1. 添加 spring-aop-4.0.0.RELEASE.jar 到工程 spring-prj2 中；
2. 在 spring-prj2 中新建 cn.edu.zjut.event 包，并在其中创建 Spring 容器事件类，并继承 ApplicationEvent 类，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.event; import org.springframework.context.ApplicationEvent;  public class EmailEvent **extends ApplicationEvent**{ private String address; private String text;  public EmailEvent(Object source){ super(source);  } public EmailEvent(Object source, String address, String text){ super(source); this.setAddress(address); this.setText(text);  }  //省略getters/setters方法 } |

1. 在 spring-prj2 中新建 cn.edu.zjut.listener 包，并在其中创建 Spring 容器事件的监听器类，并实现 ApplicationListener 接口，具体代码如下：

|  |
| --- |
| package cn.edu.zjut.listener; import org.springframework.context.ApplicationEvent; import org.springframework.context.ApplicationListener; import cn.edu.zjut.event.EmailEvent;  public class EmailNotifier **implements ApplicationListener**{ **public void onApplicationEvent(ApplicationEvent evt)**{ if(evt instanceof EmailEvent){  EmailEvent emailEvent = (EmailEvent)evt;  System.out.println("需要发送邮件的接收地址 " + emailEvent.getAddress());  System.out.println("需要发送邮件的邮件正文 " + emailEvent.getText());  } else{  System.out.println("容器本身的事件 " + evt); }  } |

}

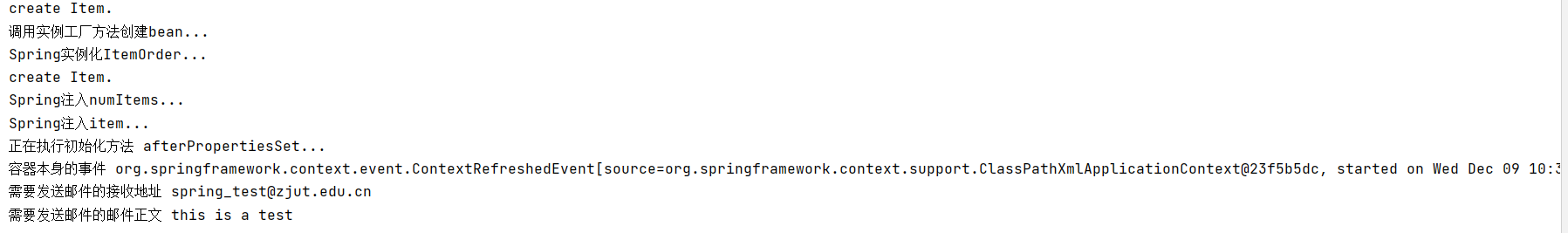
1. 修改 Spring 配置文件 applicationContext.xml，Spring 容器事件注册监听器，代码片段如下：

<bean class="cn.edu.zjut.listener.EmailNotifier"/>

1. 修改测试类 SpringEnvTest，调用 ApplicationContext 的 publishEvent 来触发事件，代码片段如下：

|  |
| --- |
| public class SpringEnvTest { public static void main(String[] args) {  ApplicationContext ctx = new ClassPathXmlApplicationContext(  "applicationContext.xml");  **EmailEvent ele = new EmailEvent("hello",**  **"spring\_test@zjut.edu.cn", "this is a test"); ctx.publishEvent(ele);**  }  } |

1. 运行测试类 SpringEnvTest，观察控制台的输出，并记录运行结果。



# （四）实验要求

1、填写并上交实验报告，报告中应包括：

1. 运行结果截图；
2. 结合实验步骤 1，总结使用 XmlBeanFactory 创建 BeanFactory 实例，或使用

FileSystemXmlApplicationContext 、 ClassPathXmlApplicationContext 创建

ApplicationContext 实例的基本方法，并记录下来；

从BeanFactory中获取bean需要实例化beanfactory,构造函数的参数为配置文件的路径。  
例如加载ClassPath下的配置文件可以用ClassPathResource加载，然后传递给XmlBeanFactory构造函数。

FileSystemXmlApplicationContext

这个方法是从文件绝对路径加载配置文件，例如：

ApplicationContext ctx = new FileSystemXmlApplicationContext( "G:/Test/applicationcontext.xml ");

如果在参数中写的不是绝对路径，那么方法调用的时候也会默认用绝对路径来找

ClassPathXmlApplicationContext

这个方法是从classpath下加载配置文件(适合于相对路径方式加载)，例如：

ApplicationContext ctx = new ClassPathXmlApplicationContext( "/applicationcontext.xml ");

1. 根据实验步骤 3-8，总结 ApplicationContext 支持国际化的基本步骤，并记录下来；

创建国际化资源文件，在配置文件中创建bean管理资源文件。

1. 根据实验步骤 9-14，查找相关资料，总结 Spring 完成事件监听及处理的基本步骤，并记录下来；（5）碰到的问题及解决方案或思考；（6）实验收获及总结。

2、上交程序源代码，代码中应有相关注释。