实验五：Spring 的 bean

一、 **实验目的：** Spring 是当前主流的 Java Web 开发框架，它是为了解决企业应用开发的复杂性问题而产生的。对于一个 Java 开发者来说，掌握 Spring 框架的使用，已是其必备的技能之一

二、 **预习要求：**

1、 了解 Spring 的概念和优点

2、 理解 Spring 中的 IoC 和 DI 思想

3、 掌握 ApplicationContext 容器的使用

4、 掌握属性 setter 方法注入的实现

5、 了解 Bean 的常用属性及其子元素

6、 掌握实例化 Bean 的三种方式

7、 熟悉 Bean 的作用域和生命周期

8、 掌握 Bean 的三种装配方式

三、实验内容：

使用 Spring 的 IOC 完成保存用户的操作以及实现 DI 依赖注入。

动手练习 Spring 框架的下载、Spring 容器创建的方式以及入门程序的编写，练习依赖注入实现案例的编写，加深理解，巩固本章的学习内容。

1、Spring 的核心容器

2、Spring 的入门程序

3、依赖注入的概念

4、依赖注入的实现方式

5、 bean 的作用域

6、 基于 XML 的装配

7、 基于 Annotation 的装配

8、 自动装配

四、 实验方法和步骤：

IOC：Inversion of Control 控制反转. 指的是对象的创建权反转(交给)给 Spring.

作用是实现了程序的解耦合.

DI :Dependency Injection 依赖注入.需要有 IOC 的环境,Spring 创建这个类的过程中,Spring 将类的依赖的属性设置进去.

Spring 的 Bean 的属性注入:

【构造方法的方式注入属性】

```
<!-- 第一种：构造方法的方式 -->
<bean id="car" class="com.haust.pojo.Car">
    <constructor-arg name="name" value="保时捷"/>
    <constructor-arg name="price" value="1000000"/>
</bean>
```

【set 方法的方式注入属性】

```
<!-- 第二种：set 方法的方式 -->
<bean id="car2" class="com.haust.pojo.Car">
    <property name="name" value="奇瑞 QQ"/>
    <property name="price" value="40000"/>
</bean>
```

Spring 的属性注入：对象类型的注入:

```
<!-- 注入对象类型的属性 -->
<bean id="person" class="com.haust.pojo.Person">
    <property name="name" value="张三"/>
    <!-- ref 属性：引用另一个 bean 的 id 或 name -->
    <property name="car" ref="car2"/>
</bean>
```

SpEL 的方式的属性注入:Spring3.x 版本后提供的方式.

```
SpEL: Spring Expression Language.
语法:#{ SpEL }

    <!-- SpEL 的注入的方式 -->
    <bean id="car3" class=" com.haust.pojo.Car">
```

```
        <property name="name" value="#{'奔驰'}"/>
        <property name="price" value="#{800000}"/>
    </bean>

    <bean id="person2" class="com.haust.pojo.Person">
    <property name="name" value="#{'李四'}"/>
    <property name="car" value="#{car3}"/>
    </bean>

    <bean id="carInfo" class=" com.haust.pojo.CarInfo"></bean>

    引用了另一个类的属性
    <bean id="car4" class=" com.haust.pojo.Car">
<!--        <property name="name" value="#{'奔驰'}"/> -->
        <property name="name" value="#{carInfo.carName}"/>
        <property name="price" value="#{carInfo.calculatePrice()}"/>
    </bean>
```

注入复杂类型:

```
<!-- Spring 的复杂类型的注入===================== -->
    <bean id="collectionBean" class=" com.haust.pojo.CollectionBean">
        <!-- 数组类型的属性 -->
        <property name="arrs">
            <list>
                <value>张三</value>
                <value>李四</value>
                <value>王五</value>
            </list>
        </property>

        <!-- 注入 List 集合的数据 -->
        <property name="list">
            <list>
                <value>芙蓉</value>
                <value>如花</value>
                <value>凤姐</value>
            </list>
        </property>

        <!-- 注入 Map 集合 -->
        <property name="map">
            <map>
                <entry key="aaa" value="111"/>
```

```
                <entry key="bbb" value="222"/>
                <entry key="ccc" value="333"/>
            </map>
        </property>

        <!-- Properties 的注入 -->
        <property name="properties">
            <props>
                <prop key="username">root</prop>
                <prop key="password">123</prop>
            </props>
        </property>
    </bean>
```

例 1 通过数据访问层和业务逻辑层类测试 IoC 和 DI。

（一）创建 web 项目，引入 jar 包或依赖：

（二）引入相关配置文件：

applicationContext.xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd"

>

</beans>

（三）编写相关的类：

public interface UserDao {

public void say();

```
}
public class UserDaoImpl implements UserDao {
   @Override
   public void say() {
       // TODO Auto-generated method stub
       System.out.println("userDao say hello world!");
   }
}
public interface UserService {
   public void say();
}
public class UserServiceImpl implements UserService {

   private UserDao userdao;
   public void setUserdao(UserDao userdao) {
       this.userdao = userdao;
   }
   @Override
   public void say() {
       // TODO Auto-generated method stub
       userdao.say();
       System.out.println("UserService say hello world!");
   }

}
```

（四）完成配置：

```
<bean id="userDao" class="com.haust.dao.UserDaoImpl"></bean>
       <bean id="userservice"
class="com.haust.service.UserServiceImpl">
```

```
                    <property name="userdao" ref="userDao" />
            </bean>
```

（五）编写测试程序：

```
public class TestIoC {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ApplicationContext context=new
ClassPathXmlApplicationContext("applicationContext.xml");
        UserDao userdao= (UserDao) context.getBean("userDao");
        userdao.say();
    }
}
public class TestDI {
    @Test
    public void testDI() {
        // TODO Auto-generated method stub
        ApplicationContext applicationContext=new
ClassPathXmlApplicationContext("applicationContext.xml");
        UserService userService= (UserService)
applicationContext.getBean("userservice");
        userService.say();
    }
}
```

例 2：完成**开发一个打印机**

（一）创建 web 项目，引入 jar 包：

（二）引入相关配置文件：

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

（三）编写相关的类：

```
public interface Ink {
    public String getColor();
}

public class ColorInkImp implements Ink {
    public String getColor() {
        return "彩色打印机墨盒";
    }
}

public class GreyInkImp implements Ink {
    public String getColor() {
        return "黑白打印机墨盒";
    }
```

```
}

public interface Paper {
    public void putChar(String str);
}

public class A4Paper implements Paper {
    private int rows;//24
    private int cols;//12
    public int getRows() {
        return rows;
    }
    public void setRows(int rows) {
        this.rows = rows;
    }
    public int getCols() {
        return cols;
    }
    public void setCols(int cols) {
        this.cols = cols;
    }

    public void putChar(String str) {
        System.out.println(this.rows+";"+this.cols);
        System.out.println("这是在 A4 纸上输出的内容");
        System.out.println(str);
    }
}

public class B5Paper implements Paper {
```

```
    private int rows;
    private int cols;
    public void putChar(String str) {
        System.out.println(this.rows+";"+this.cols);
        System.out.println("这是在 B5 纸上输出的内容");
        System.out.println(str);
    }
    public int getRows() {
        return rows;
    }
    public void setRows(int rows) {
        this.rows = rows;
    }
    public int getCols() {
        return cols;
    }
    public void setCols(int cols) {
        this.cols = cols;
    }
}

public class Printer {
    private Ink ink;
    private Paper paper;
    public Ink getInk() {
        return ink;
    }
    public void setInk(Ink ink) {
        this.ink = ink;
    }
```

```
    public Paper getPaper() {
        return paper;
    }
    public void setPaper(Paper paper) {
        this.paper = paper;
    }
    public void print(String str)
    {
        System.out.println(ink.getColor());
        paper.putChar(str);
    }
}
```

（四）完成配置：

```
<beans>
    <bean id="colorInk" class="com.aptech.jb.ink.ColorInk" />
    <bean id="greyInk" class="com.aptech.jb.ink.GreyInk" />
    <bean id="a4Paper" class="com.aptech.jb.paper.TextPaper">
        <property name="charPerLine" value="10" />
        <property name="linePerPage" value="8" />
    </bean>
    <bean id="b5Paper" class="com.aptech.jb.paper.TextPaper">
        <property name="charPerLine" value="6" />
        <property name="linePerPage" value="5" />
    </bean>
    <bean id="printer" class="com.aptech.jb.Printer">
        <property name="ink" ref="colorInk"/>
        <property name="paper" ref="b5Paper"/>
    </bean>
</beans>
```

（五）编写测试程序：

```
public class TestDI {
    public static void main(String[] args) {
    ApplicationContext context =
    new ClassPathXmlApplicationContext(
        "applicationContext.xml");
    // 由 Spring 创建 printer 对象，并根据配置文件注入
    // 依赖的组件，完成组装
```

```
    Printer printer = (Printer)context.getBean("printer");
    printer.print("几位轻量级容器的作者曾骄傲地对我...");
}
```

例 3：采用 IOC 方式组建军队

（一）创建 web 项目，引入 jar 包：

（二）引入相关配置文件：

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
</beans>
```

（三）编写相关的类：

```
public interface Assaultable {
    public void attack();
}

public interface Mobile {
    public void move();
}
```

```
public abstract class Weapon implements Assaultable, Mobile {
	public void move() {
	}
	public void attack() {
	}
}

public class Tank extends Weapon {
	@Override
	public void move() {
		System.out.println("坦克在移动");
	}
	@Override
	public void attack() {
		System.out.println("坦克在攻击");
	}
}

public class Flighter extends Weapon {
	@Override
	public void move() {
		System.out.println("飞机在移动");
	}
	@Override
	public void attack() {
		System.out.println("飞机在攻击");
	}
}

import java.util.HashSet;
```

```
import java.util.Iterator;
import java.util.Set;
public class Army {
	Set<Weapon> set=new HashSet<Weapon>();
	public Set<Weapon> getSet() {
		return set;
	}
	public void setSet(Set<Weapon> set) {
		this.set = set;
	}
	public void attackall()
	{
		Iterator<Weapon> it=set.iterator();
		while(it.hasNext())
		{
			Weapon weapon=it.next();
			weapon.attack();
			weapon.move();
		}
	}
}
```

（四）完成配置：

```
<beans>
<bean id=" weapon1" class="com.hkd.ioc. Flighter "></bean>
<bean id=" weapon2" class="com.hkd.ioc. Tank "></bean>
  <bean id=" army " class="com.hkd.ioc. Army ">
     <property name=" set " >
       <set>
          <ref bean=" weapon1"/>
```

```
            <ref bean=" weapon2"/>
        </set>
      </property name >
  </bean>
</beans>
```

（五）编写测试程序：

```
public class TestIoC {
    public static void main(String[] args) {
     ApplicationContext context =
     new ClassPathXmlApplicationContext(
          "applicationContext.xml");
    // 由 Spring 创建 army 对象，并根据配置文件注入
    // 依赖的组件，完成组装
    Army army = (Army)context.getBean("army ");
    army. attackall();
}
```

思考题：

（一）Spring 框架提供的两种核心容器:BeanFactory 和 ApplicationContext，试通过给两者创建方式并说明二者的区别。

（二） ApplicationContext 接口的实现类：ClassPathXmlApplicationContext 和 FileSystemXmlApplicationContext 两者之间的区别。

五、 实验作业要求：

（一） 实验目的：

（二） 实验内容：

（三） 实验结果：可以是运行结果截图或其他形式的结果展示

（四） 问题及解决：实验中遇到的问题及解决方法。

（五） 回答思考题提出的问题，