实验二 Spring 基础编程——登录用户的购物车

一、基础实验——基于 Spring 框架的用户登录模块

（一）实验目的

1、掌握 Spring 环境搭建的基本方法，能在 JAVASE 应用中使用 Spring，并能在 Eclipse 中开发 Spring 应用；
2、初步理解 Spring 的核心机制：控制反转 IoC（Inversion of Control）与依赖注入 DI（Dependency Injection）；
3、理解 Spring 配置文件的作用，掌握 bean 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 为企业应用的开发提供了一个轻量级的解决方案；
2、Spring5 框架共包括 5 大模块，每个模块用于提供不同的解决方案：
（1）核心容器（Core Containe）：核心容器提供了 Spring 框架的基本功能，是其它模块建立的基础，有 spring-core、spring-beans、spring-context、spring-context-support 和 spring-expression（Expression Language、SpEL）组成，其中 spring-beans 和 spring-core 是 spring 框架的核心模块；
（2）AOP 和设备支持（AOP）：提供了面向切面编程（AOP）的实现，由 spring-aop、spring-aspects 和 spring-instrument 3 个子模块组成；
（3）数据访问与集成（Data Access/Integration）：提供了集成 JDBC 的封装包、常用 ORM 框架的封装包等，由 spring-jdbc、spring-orm、spring-oxm、spring-jms 和 spring-tx 组成；
（4）Web 模块：提供了 Web 开发以及集成 Web 框架的封装包，提供了 MVC 框架等，由 spring-websocket、spring-webmvc、spring-web 和 spring-webflux 组成；
（5）消息（Messaging）模块：spring-messaging 是从 Spring4.0 开始新加入的一个模块，主要职责是为 Spring 框架集成一些基础的报文传送应用。
3、在传统的程序设计过程中，当某个 Java 实例（调用者）需要调用另一个 Java 实例（被调用者）时，通常由调用者来创建被调用者的实例，而在控制反转模式下，创建被调用者的工作不再由调用者来完成，两者之间的依赖关系由

Spring 管理，使得两者解耦；

4、在 Spring 中创建被调用者的工作由 Spring 容器来完成，然后将被调用者实例注入调用者，因此也被称为依赖注入；

5、Spring 推荐面向接口编程，这样可以更好地让规范和实现分离，从而提供更好的解耦。

（三）实验内容及步骤

1、登录 https://repo.spring.io/ui/native/release/org/springframework/spring/站点，下载 Spring 框架的依赖 JAR 包（如：spring-5.2.9.RELEASE-dist）；

2、在 Eclipse 中新建 Java 工程 spring-prj1，并添加 common-logging-1.2.jar 和 Spring 的 4 个基础 JAR 包到工程中，如下图所示；

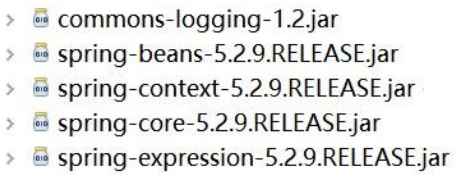

图 2-1 Spring 的 4 个基础包

3、在 spring-prj1 中新建 cn.edu.zjut.bean 包，并在其中创建 UserBean.java，用于记录登录用户信息，代码如下；

```
package cn.edu.zjut.bean;

public class UserBean {
    private String username="";
    private String password="";

    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

4、在 spring-prj1 中新建 cn.edu.zjut.dao 包，并在其中创建 IUserDAO 接口定义数

据持久层的操作，以及实现类 UserDAO 实现数据持久层的操作，代码如下：

```
package cn.edu.zjut.dao;
import cn.edu.zjut.bean.UserBean;

public interface IUserDAO {
    public void search(UserBean user);
}
```

```
package cn.edu.zjut.dao;
import cn.edu.zjut.bean.userBean;

public class UserDAO implements IUserDAO{
    publ ic UserDAO(){
        System.out.println("create UserDAO.");
    }
    public void search(UserBean user) {
        System.out.println("execute --search()-- method.");
    }
}
```

5、在 spring-prj1 中创建 Spring 配置文件 applicationContext.xml，并在其中配置 UserDAO 实例，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.xsd">

    <bean id="userDAO" class="cn.edu.zjut.dao.UserDAO" />

</beans>
```

6、在 spring-prj1 中新建 cn.edu.zjut.app 包，并在其中创建测试类 SpringEnvTest，调用 UserDAO 实例的 search()方法，代码如下：

```
package cn.edu.zjut.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import cn.edu.zjut.bean.UserBean;
import cn.edu.zjut.dao.IUserDAO;

public class SpringEnvTest {
    public static void main(String[] args) {
```

```
        //创建 Spring 容器
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        UserBean loginUser = new UserBean();
        loginUser.setUsername("SPRING");
        loginUser.setPassword("SPRING");
        //获取 UserDAO 实例
        IUserDAO userDao = (IUserDAO) ctx.getBean("userDAO");
        userDao.search(loginUser);
    }
}
```

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

8、在 spring-prj1 中新建 cn.edu.zjut.service 包，并在其中创建 IUserService 接口定义注册逻辑，以及实现类 UserService 实现注册逻辑，代码如下：

```
package cn.edu.zjut.service;
import cn.edu.zjut.bean.UserBean;

public interface IUserService {
    public void login(UserBean user);
}
```

```
package cn.edu.zjut.service;
import cn.edu.zjut.bean.UserBean;
import cn.edu.zjut.dao.IUserDAO;

public class UserService implements IUserService {
    private IUserDAO userDAO = null;

    public UserService(){
        System.out.println("create UserService.");
    }
    public void setUserDAO(IUserDAO userDAO) {
        System.out.println("--setUserDAO--");
        this.userDAO = userDAO;
    }
    public void login(UserBean user) {
        System.out.println("execute --login()-- method.");
        userDAO.search(user);
    }
}
```

9、修改 Spring 配置文件 applicationContext.xml，在其中增加 UserService 实例的配置，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.xsd">

    <bean id="userDAO" class="cn.edu.zjut.dao.UserDAO" />

    <bean id="userService" class="cn.edu.zjut.service.UserService">
        <property name="userDAO" ref="userDAO" />
    </bean>

</beans>
```

10、修改测试类 SpringEnvTest，调用 UserService 实例的 login()方法，代码如下：

```
package cn.edu.zjut.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import cn.edu.zjut.bean.UserBean;
import cn.edu.zjut.service.IUserService;

public class SpringEnvTest {
    public static void main(String[] args) {
        //创建 Spring 容器
        ApplicationContext ctx = new ClassPathXmlApplicationContext(
                "applicationContext.xml");
        UserBean loginUser = new UserBean();
        loginUser.setUsername("SPRING");
        loginUser.setPassword("SPRING");
        //获取 UserService 实例
        IUserService userService =
                    (IUserService) ctx.getBean("userService");
        userService.login(loginUser);
    }
}
```

11、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）根据实验过程，观察运行后的控制台输出，查找相关资料，总结 Spring 容器管理 Bean 组件的过程（如：何时加载、何时调用 Bean 实例中的方法等）以及进行依赖注入的过程，并记录下来；
（3）根据实验步骤 5、9，查找相关资料，总结 Spring 配置文件中 bean 元素及其属性、子元素的作用，并记录下来；
（4）根据实验步骤 6、10，查找相关资料，总结控制反转模式下两个 Java 实例的依赖关系与传统的程序设计过程体现出的依赖关系由什么区别，控制反转的优点是什么，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。

二、提高实验——基于 Spring IoC 的登录用户购物车

（一）实验目的

1、进一步熟悉 Spring 基础环境搭建的方法，以及在 Eclipse 中开发 Spring 应用的主要步骤；
2、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）；
3、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 cn.edu.zjut.bean 包中创建 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
}
```

2、修改 Spring 配置文件 applicationContext.xml，使用构造器注入的方式配置 Item 实例，代码片段如下：

```
<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>
```

3、在 cn.edu.zjut.app 包中创建测试类 SpringBeanTest，对构造器注入进行测试，代码如下：

```
package cn.edu.zjut.app;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import cn.edu.zjut.bean.IItem;

public class SpringBeanTest {
    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());
    }
}
```

4、运行测试类 SpringBeanTest，观察控制台的输出，并记录运行结果；

5、在 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() * item.getCost());
    }
    //省略 getters/setters 方法
}
```

6、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>
```

7、修改测试类 SpringBeanTest，对构造器注入进行测试，代码片段如下：

```
public class SpringBeanTest {
    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());
    }
}
```

8、运行测试类 SpringBeanTest，观察控制台的输出，并记录运行结果；

9、在 cn.edu.zjut.bean 包中创建购物车接口 IShoppingCart（代码略）及其实现类 ShoppingCart.java，其中包含集合类型（List）属性，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;
    }
}
```

10、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>
```

11、修改测试类 SpringBeanTest，对集合类型属性配置进行测试，代码略；

12、运行测试类 SpringBeanTest，观察控制台的输出，并记录运行结果；

13、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；

14、在 cn.edu.zjut.bean 包中创建用户模型类 UserBean（参考实验一），其中包含用户名、密码以及 ShoppingCart 等属性，代码略；

15、修改测试类 SpringBeanTest，调用 UserBean 实例并访问其购物车，循环输出购物车中的商品信息，代码略；

16、运行测试类 SpringBeanTest，观察控制台的输出，并记录运行结果；
17、修改 cn.edu.zjut.service 包中的业务逻辑类 UserService，在其中添加查看购物车的相关方法 void checkshoppingcart(UserBean user)，在该方法中访问用户的购物车，并循环输出购物车中的商品信息，代码略；
18、修改测试类 SpringEnvTest，调用 UserService 实例的 checkshoppingcart()方法，代码略；
19、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；
（3）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；
（4）根据实验步骤 9-13，查找相关资料，总结集合类型属性的配置方法，并记录下来；
（5）观察实验步骤 19 的控制台输出信息，思考该运行结果的产生原因，并记录下来；
（6）碰到的问题及解决方案或思考；
（7）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——基于 SpingAOP 的登录用户登录权限验证

（一）实验目的

1、学习理解 AOP（Aspect Orient Programming）即面向切面编程的基本概念，重点掌握切面、增强处理、切点的概念；
2、掌握使用@AspectJ 实现 Spring AOP 的基本步骤和配置方法，会使用基于 Annotation 的注解方式或基于 XML 配置文件的方式来定义切入点和增强处理。

（二）基本知识与原理

1、AOP（Aspect Orient Programming），即面向切面编程，作为面向对象编程的一种补充，用于处理系统中分布于各个模块中共同关注的服务问题，如事物管理、安全检查、缓存、对象池管理等；

2、面向切面编程的相关术语有：

（1）切面（Aspect）：业务流程运行的某个特定步骤，也就是应用运行过程的关注点，关注点可能横切多个对象，因此也被称为横切关注点；

（2）连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用或异常的抛出，Spring AOP 中，连接点总是方法的调用；

（3）增强处理（Advice）：是切面的具体实现，在前面的某个特定的连接点上执行动作，Spring 中执行的动作往往就是调用某类的具体方法，如实现保存日志功能的类就是通知；

（4）切入点（Pointcut）：可以插入增强处理的连接点，即某连接点将被添加增强处理，则该连接点也就变成了切入点；

3、在@AspectJ 方式下，Spring 有两种途径来定义切入点和增强处理：

（1）基于 Annotation 的注解方式：使用@Aspect、@Pointcut 等 Annotation 来标注切入点和增强处理；

4、基于 XML 配置文件的方式：使用 Sping 配置文件来定义切入点和增强处理。

（三）实验内容及步骤

1、在工程 spring-prj1 中添加 Spring 框架中与 AOP 相关的 JAR 包（spring-aop-5.2.9.RELEASE.jar、spring-aspects-5.2.9.RELEASE.jar）；

2、在站点 http://www.eclipse.org/aspectj/downloads.php 下载 aspectj9（如 aspectj-1.9.1.jar），并添加其中的 aspectj.weaver.jar 到工程 spring-prj1 中；

3、在站点 http://sourceforge.net/projects/aopalliance/下载 aopalliance.jar，并将其添加到工程 spring-prj1 中；

4、在 cn.edu.zjut.app 包中创建测试类 SpringAOPTest，调用 UserService 实例的 login()方法（代码略，参考基础实验步骤 10），运行并观察控制台的输出；

5、新建 cn.edu.zjut.aspect 包，并在其中创建 SecurityHandler.java，用于实现权限检查，并用基于 Annotation 的注解方式定义切面，其中用@Aspect 修饰的类是切面类，用@Pointcut 定义切点，用@Before 定义 Before 增强处理，代码如下：

```
package cn.edu.zjut.aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
```

```
@Aspect
public class SecurityHandler {
     /** 定义 Pointcut,Pointcut 的名称是 modify,
      * 此方法不能有返回值和参数，该方法只是一个标识*/
     @Pointcut("execution(* login*(..)) ")
     private void search(){};

     /** 定义 Advice，标识在那个切入点何处织入此方法 */
     @Before("search()")
     private void checkSecurity() {
            System.out.println("---checkSecurity()---");  }
}
```

6、修改 Spring 配置文件 applicationContext.xml，在头文件中添加“xmlns：aop”的命名申明，并在“xsi：schemaLocation”中指定 aop 配置的 schema 的地址，同时增加对 SecurityHandler 实例的配置，并启动注解配置 AOP 支持，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"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- 启动使用注解配置 AOP 支持 -->
    <aop:aspectj-autoproxy />

    <bean id="securityHandler"
          class="cn.edu.zjut.aspect.SecurityHandler" />
    ……
</beans>
```

7、运行测试类 SpringAOPTest，观察控制台的输出，并记录运行结果；

8、在 cn.edu.zjut.aspect 包中创建 SecurityHandler2.java，代码如下：

```
package cn.edu.zjut.aspect;

public class SecurityHandler2 {
     private void checkSecurity() {
            System.out.println("---checkSecurity()2---");  }
}
```

9、修改 Spring 配置文件 applicationContext.xml，增加对 SecurityHandler2 实例的配置，并使用 XML 配置文件的方式定义切面，代码片段如下：

```
<bean id="securityHandler2"
      class="cn.edu.zjut.aspect.SecurityHandler2" />
<!-- 配置文件的方式 -->
<aop:config>
    <aop:aspect id="security" ref="securityHandler2">
        <aop:pointcut id="find"
               expression="execution(* *.login*(..))" />
        <aop:before method="checkSecurity" pointcut-ref="find" />
    </aop:aspect>
</aop:config>
```

10、运行测试类 SpringAOPTest，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）根据实验过程，查找相关资料，整理 Spring AOP 中的基本概念（如切面、增强处理、切点等），并记录下来；

（3）根据实验过程，总结 Spring AOP 的基本步骤，以及使用基于 Annotation 的注解方式或基于 XML 配置文件的方式来定义切入点和增强处理的基本方法，并记录下来；

（4）根据实验步骤 5 或 9，查找 AspectJ 切入点表达式的相关资料，记录其中 @Pointcut 注解中切入点表达式的含义；若切入点是 cn.edu.zjut.service 包下所有实现类中的增删改方法，思考切入点表达式应该怎么写，并记录下来；

（5）对比实验步骤 4 与 7，观察运行后的控制台输出以及相应的程序代码，总结 Spring AOP 优点或作用、适用场景，并记录下来；

（6）碰到的问题及解决方案或思考；

（7）实验收获及总结。

2、上交程序源代码，代码中应有相关注释。