



JavaEE 实验二实验报告

姓　　名　　 Mcrivers

班　　级　　 软件工程2102班

学　　号

提交日期　　 2023.10.29

目录

[一、基础实验——基于 Spring 框架的用户登录模块 1](#_Toc149503130)

[1. 运行结果截图 1](#_Toc149503131)

[2. Spring容器管理 Bean 组件的过程、进行依赖注入的过程 1](#_Toc149503132)

[3. Spring 配置文件中 bean 元素及其属性、子元素的作用 1](#_Toc149503133)

[4. 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的区别、控制反转的优点 2](#_Toc149503134)

[5. 碰到的问题及解决方案或思考 2](#_Toc149503135)

[6. 实验收获及总结 3](#_Toc149503136)

[二、提高实验——基于 Spring IoC 的登录用户购物车 3](#_Toc149503137)

[1. 运行结果截图 3](#_Toc149503138)

[2. 设值注入和构造器注入各自的有点和适用场景 6](#_Toc149503139)

[3.设值注入和构造器注入方式下配置文件中的配置方法及相关元素及其属性的作用 7](#_Toc149503140)

[4.总结集合类型属性的配置方法 7](#_Toc149503141)

[5.思考步骤19所产生的运行结果的产生原因 8](#_Toc149503142)

[6. 碰到的问题及解决方案或思考 9](#_Toc149503143)

[7.实验收获及总结 9](#_Toc149503144)

[三、扩展实验——基于 SpingAOP 的登录用户登录权限验证 9](#_Toc149503145)

[1. 运行结果截图 9](#_Toc149503146)

[2. 整理 Spring AOP 中的基本概念 10](#_Toc149503147)

[3. 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 11](#_Toc149503148)

[4. @Pointcut 注解中切入点表达式的含义，思考切入点表达式应该怎么写，并记录下来 12](#_Toc149503149)

[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 13](#_Toc149503150)

[6. 碰到的问题及解决方案或思考 14](#_Toc149503151)

[7.实验收获及总结 14](#_Toc149503152)

# 一、基础实验——基于 Spring 框架的用户登录模块

## 1. 运行结果截图

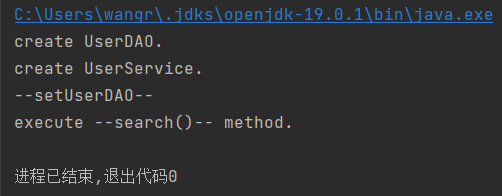


图1.1.1 运行结果1

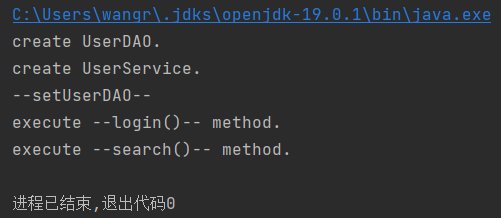


图1.1.2 运行结果2

## 2. Spring容器管理 Bean 组件的过程、进行依赖注入的过程

1. 加载xml配置文件，xml文件中声明了哪些类需要交给 Spring 管理；

2. 根据配置信息实例化 Bean 对象；

3. 根据xml配置文件中<init-method>元素指定的方法初始化 Bean ；

4. 根据xml配置文件中的`<property>` 元素来进行设值注或构造函数注入；

5. 其他的组件或者类可以通过 Spring 容器来获取这些 Bean 实例，然后调用 Bean 中的方法或者访问属性。

## 3. Spring 配置文件中 bean 元素及其属性、子元素的作用

**1. <bean> 元素的属性**

① id（或name）：用于指定Bean的唯一标识符。

② class：指定Bean的类全名，用于告诉Spring容器要实例化哪个类。

③ scope：定义Bean的作用域，包括 singleton（单例，默认值）、prototype（原型）、request（每个HTTP请求一个实例）、session（每个HTTP Session一个实例）等。

④ lazy-init：指定是否延迟初始化Bean，默认为false，即在容器启动时就初始化Bean。

⑤ init-method：指定Bean初始化时调用的方法。

⑥ destroy-method：指定Bean销毁时调用的方法。

⑦ autowire：指定自动装配的模式，可以是 no（默认，不自动装配）、byName（根据属性名自动装配）、byType（根据属性类型自动装配）等。

⑧ autowire-candidate：在自动装配时，该属性为false表示该Bean不参与自动装配。

⑨ dependency-check：指定Spring容器在实例化Bean时是否检查其依赖，默认为none，表示不检查。

**2. <bean> 元素的子元素**

① <property>：用于设置Bean的属性，包括name属性（指定属性名）和value属性（指定属性值），或者ref属性（指定对其他Bean的引用）。

② <constructor-arg>：用于设置构造函数参数，可以包含type属性（指定参数类型）和value属性（指定参数值），或者ref属性（指定对其他Bean的引用）。

③ <qualifier>：用于指定自动装配时的限定符，用于标识特定的Bean。

④ <lookup-method>：用于指定返回某个Bean的方法。

## 4. 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的区别、控制反转的优点

1. 在传统的程序设计过程中，一个对象直接依赖于另一个对象的具体实现。当需要改变依赖关系时，需要修改源代码，这样的设计是紧耦合的；在控制反转模式下，控制权被转移给了容器，对象的创建、组装和管理都由IoC容器来完成，对象不再负责依赖查找和创建，而是由IoC容器负责，对象之间的依赖关系通过配置文件、注解或者其他方式进行配置。

2. 松耦合、灵活性好、重用性高、集中管理。

## 5. 碰到的问题及解决方案或思考

基础实验中没有碰到问题。

## 6. 实验收获及总结

在本次实验中，我学到了如何搭建Spring环境，能够在JavaSE应用中使用Spring，并且学会了在IDEA中开发Spring应用。同时，我初步理解了Spring的核心机制：控制反转IoC与依赖注入DI。

# 二、提高实验——基于 Spring IoC 的登录用户购物车

## 1. 运行结果截图

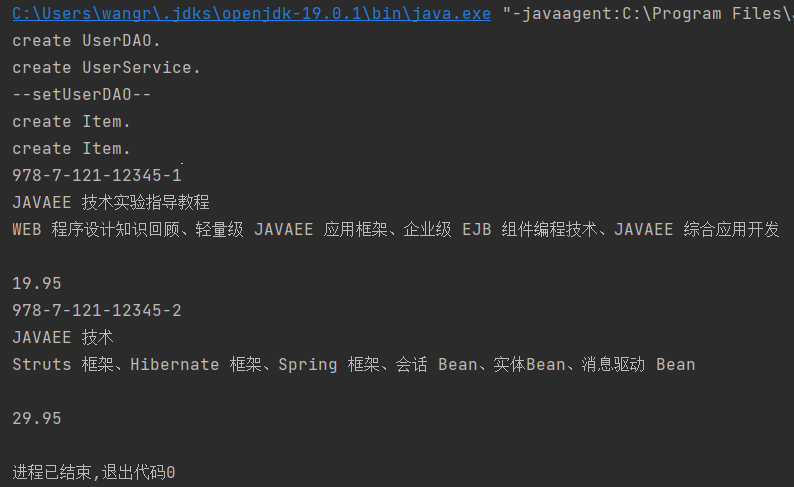


图2.1.1 运行结果1

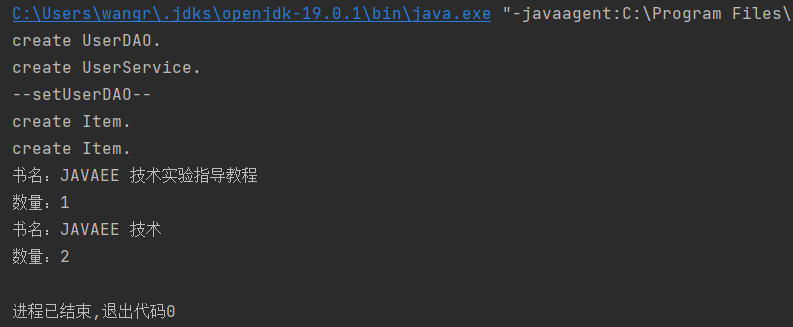


图2.1.2 运行结果2

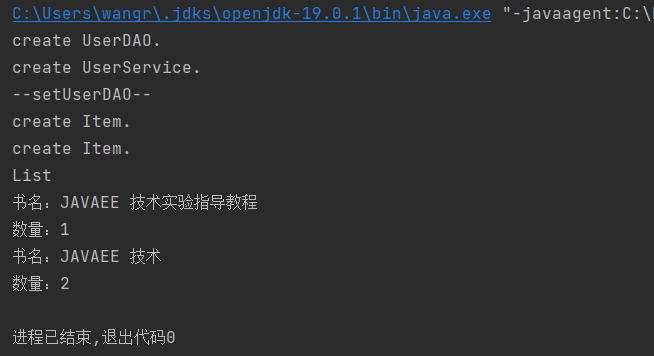


图 2.1.3 运行结果3（List）

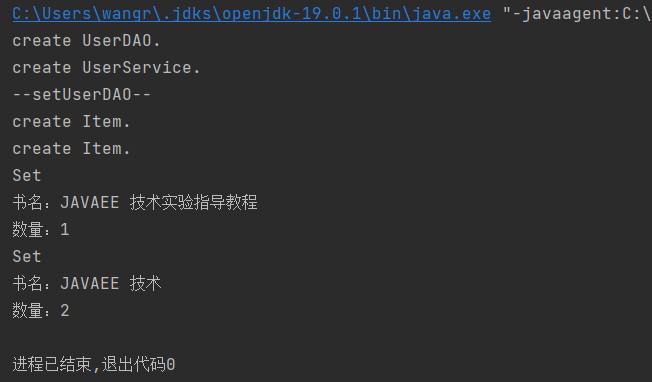


图 2.1.4 运行结果3（Set）

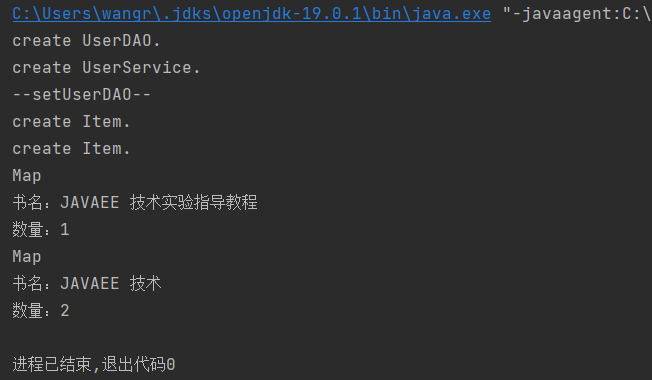


图 2.1.4 运行结果4（Map）



图 2.1.5 运行结果5（Properties）

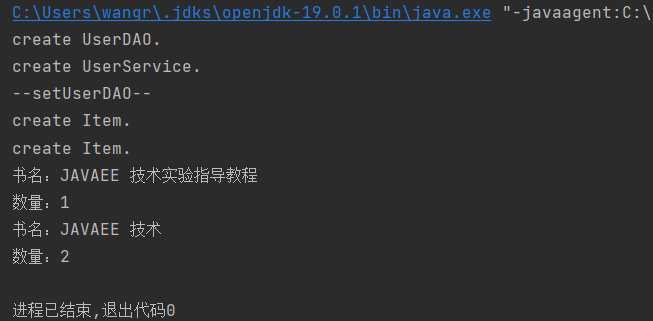


图 2.1.6 运行结果6（调用UserBean实例并访问其购物车）

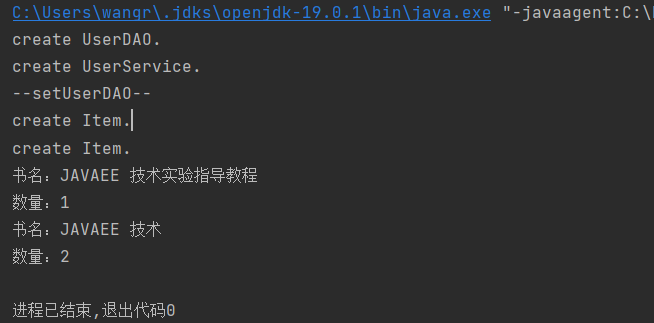


图 2.1.7运行结果7（添加checkshoppingcart(UserBean user)）

## 2. 设值注入和构造器注入各自的有点和适用场景

**1.设值注入**

优点：

①可选度高，可以选择需要注入的属性，不必全部传递所有属性

②容易修改，当需要修改时只需要修改对应的setter方法 ③简洁度高，当依赖关系复杂时，依然可以保持相对简洁

适用场景：

①可选依赖：当一个类的某些依赖是可选的时

②配置的可修改性：想通过配置文件来改变依赖时

**2.构造器注入**

优点：

①不可变性：构造器注入可以确保依赖项在对象创建后不可变

②一致性和稳定性

③依赖项必需性：可以确保在创建对象时，所有必需的依赖项都被传入

适用场景：

①必需依赖：当一个类的某些依赖是必需的时

②不可变性需求：当希望创建不可变的对象时

## 3.设值注入和构造器注入方式下配置文件中的配置方法及相关元素及其属性的作用

**1.设值注入**

①使用<bean>元素定义Bean

② <property>元素的属性：

name：指定要注入的属性名称。

value：指定要注入的值（字面值）

ref：指定一个引用类型参数，引用另一个Bean

**2.构造器注入**

①使用<bean>元素定义Bean

②<constructor-arg>元素的属性：

index：指定参数的索引位置（从0开始），用于指定构造函数的参数位置。

value：指定字面值参数的值。

ref：指定一个引用类型参数，引用另一个Bean

## 4.总结集合类型属性的配置方法

①List集合属性的配置：

Copy code

<property name="listProperty">

<list>

<value>Item 1</value>

<value>Item 2</value>

</list>

</property>

② Set集合属性的配置：

Copy code

<property name="setProperty">

<set>

<value>Item A</value>

<value>Item B</value>

</set>

</property>

③Map集合属性的配置：

Copy code

<property name="mapProperty">

<map>

<entry key="Key 1" value="Value 1"/>

<entry key="Key 2" value-ref="anotherBean"/>

</map>

</property>

④Properties集合属性的配置：

Copy code

<property name="propertiesProperty">

<props>

<prop key="Key 1">Value 1</prop>

<prop key="Key 2">Value 2</prop>

</props>

</property>

## 5.思考步骤19所产生的运行结果的产生原因

UserBean user = (UserBean) ctx.getBean("userBean");

当UserBean被实例化时，其引用的shoppingcart也被初始化；shoppingcart被初始化时，其引用的itemorder1和itemorder2也被初始化；itemorder1和itemorder2被初始化时，item1和item2被初始化，并被构造器注入书本信息。

UserService userservice =(UserService)ctx.getBean("userService");

UserService被实例化。

IShoppingCart shoppingCart = userservice.checkshoppingcart(user);

userservice.checkshoppingcart(user)函数调用了user. getShoppingCart()，返回一个IShoppingCart变量，并将其赋值给shoppingcart。

List<IItemOrder> itemsOrdered = shoppingCart.getItemsOrdered();  
for (IItemOrder itemOrder : itemsOrdered)  
{  
 System.*out*.println("书名：" + itemOrder.getItem().getTitle());  
 System.*out*.println("数量：" + itemOrder.getNumItems());  
}

打印书本信息。

## 6. 碰到的问题及解决方案或思考

问题：

在执行代码

IItemOrder itemorder1 = shoppingcart.getItemsOrdered().get(0)时，提示类型不匹配

解决方案：

进行强制类型转换

IItemOrder itemorder1 = (IItemOrder) shoppingcart.getItemsOrdered().get(0)

## 7.实验收获及总结

在本次实验中，我加深了对Spring配置文件的认识，掌握了配置文件中各主要元素及其属性的作用和基本配置方法。同时，我深刻理解了Spring中主要有两种依赖注入的方法：设置注入和构造器注入，还学会了如何处理集合类型的属性，包括List、Set、Map以及Properties。

# 三、扩展实验——基于 SpingAOP 的登录用户登录权限验证

## 1. 运行结果截图

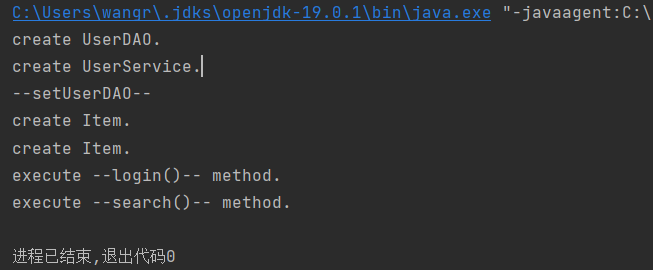


图3.1.1 运行结果1

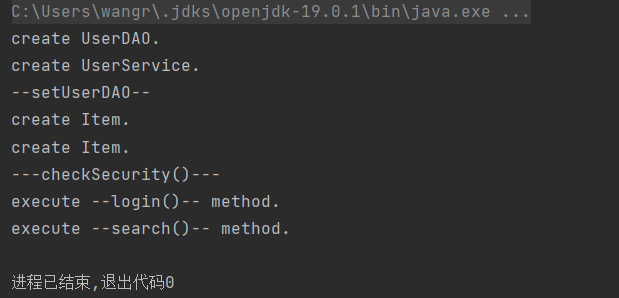


图3.1.2 运行结果2

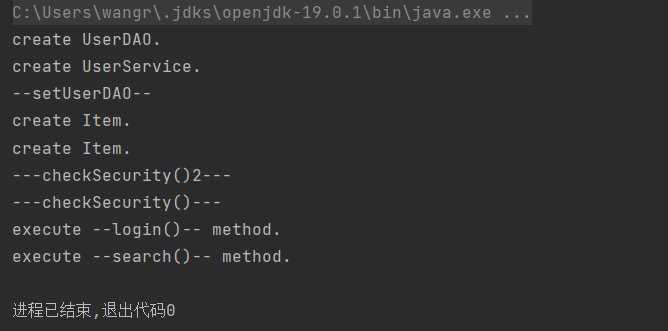


图3.1.3 运行结果3

## 2. 整理 Spring AOP 中的基本概念

1. 切面（Aspect）：

切面是一个包含通知（Advice）和切点（Pointcut）的模块。通知定义了在何时、何地执行切面的逻辑，而切点则定义了何处应该执行这些通知。切面实际上是一个类，在其中包含了通知和切点。

2. 通知（Advice）：

通知是在切面的特定连接点（Join Point）上执行的代码。在Spring AOP中，有以下几种类型的通知：

前置通知（Before Advice）：在目标方法执行前执行。

后置通知（After Advice）：在目标方法执行后执行，无论方法是否抛出异常。

返回通知（After Returning Advice）：在目标方法成功执行后执行。

异常通知（After Throwing Advice）：在目标方法抛出异常时执行。

环绕通知（Around Advice）：在目标方法前后执行，可以控制目标方法的执行逻辑。

3. 切点（Pointcut）：

切点是指在应用程序中定义的某个特定的连接点集合，通知会在这些连接点上执行。切点定义了通知被触发的条件，例如哪些方法调用、哪些类等。

4. 连接点（Join Point）：

连接点是在应用程序执行过程中能够触发通知的点。在Spring中，连接点通常表示方法的调用，但也可以是其他的事件，如字段访问或异常抛出等。

5. 增强处理（Weaving）：

增强处理是指将切面的通知织入到目标对象的过程。这可以在编译期、类加载期、运行期等不同阶段进行。Spring AOP主要使用的是运行期织入，即在应用程序运行期间将通知织入到目标对象的方法调用中。

6. 引入（Introduction）：

引入允许向现有的类添加新方法或字段，而不需要修改被引入的类。引入是AOP的另一个功能，它允许我们向现有的Java对象添加新的接口及其实现。

## 3. 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

基本步骤：Spring AOP的基本步骤包括定义切面（Aspect）、定义切点（Pointcut）、定义通知（Advice）和将切面织入到应用程序中。

**使用基于 Annotation 的方式定义：**

①定义切面

@Aspect  
public class MyAspect {  
 // 定义切点  
 @Pointcut("execution(\* com.example.service.\*.\*(..))")  
 private void serviceMethods() {}  
  
 // 定义通知  
 @Before("serviceMethods()")  
 public void beforeServiceMethods(JoinPoint joinPoint) {  
 // 前置通知的逻辑  
 }  
}

②在Spring配置类中启用AspectJ自动代理

@Configuration  
@EnableAspectJAutoProxy  
public class AppConfig {  
 // 配置其他的Bean  
}

**使用基于 XML 配置文件的方式定义：**

①定义切面和切点

<bean id="myAspect" class="com.example.aspect.MyAspect"/>  
  
<aop:config>  
 <aop:aspect id="myAspect" ref="myAspect">  
 <aop:pointcut id="serviceMethods" expression="execution(\* com.example.service.\*.\*(..))"/>  
 <aop:before method="beforeServiceMethods" pointcut-ref="serviceMethods"/>  
 </aop:aspect>  
</aop:config>

②配置AspectJ自动代理

<aop:aspectj-autoproxy/>

## 4. @Pointcut 注解中切入点表达式的含义，思考切入点表达式应该怎么写，并记录下来

execution: 表示这是一个方法执行的连接点。

\*: 表示匹配任意的返回类型。

\*.\*.login\*: 表示匹配任意包下任意类的以 "login" 开头的方法名。

(..): 表示匹配任意数量、任意类型的方法参数。

@Pointcut("execution(\* cn.edu.zjut.service..\*.\*(..)) && (execution(\* \*.add\*(..)) || execution(\* \*.delete\*(..)) || execution(\* \*.update\*(..)))")

private void serviceMethods() {}

## 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

**优点和作用：**

模块化开发： AOP允许将横切关注点（例如日志记录、性能统计、安全控制等）从业务逻辑中分离出来，使得系统各模块职责更加清晰，易于维护和扩展。

代码复用： 切面提供了一种将横切关注点模块化的方式，可以被多个模块共享，从而实现代码的复用，减少了重复代码。

降低耦合性： AOP使得业务逻辑和横切关注点分离，降低了各个模块之间的耦合性，提高了系统的灵活性和可维护性。

集中管理： AOP将横切关注点集中管理，可以统一修改、扩展和管理这些关注点，提高了系统的可维护性。

提高系统可用性和可靠性： 通过AOP可以很方便地添加日志、监控、事务管理等功能，提高了系统的可用性和可靠性。

提高开发效率： 通过AOP，开发人员可以专注于业务逻辑的编写，而无需关心横切关注点，提高了开发效率。

**适用场景：**

日志记录： 记录方法的入参、出参、方法执行时间等信息。

性能统计： 统计方法的执行时间，用于性能分析和优化。

安全控制： 检查用户权限，进行安全认证。

事务管理： 控制事务的开始、提交、回滚等操作。

异常处理： 统一处理方法抛出的异常。

缓存管理： 控制缓存的生命周期和刷新策略。

权限控制： 根据用户权限限制方法的访问。

分布式系统中的远程调用： 在分布式系统中，AOP可以用于处理远程服务的调用、超时重试等。

## 6. 碰到的问题及解决方案或思考

该实验中基本没有碰到问题。

## 7.实验收获及总结

本次实验中，我深入学习了AOP的基本概念和原理。我深入理解了AOP的概念、掌握了@AspectJ注解方式并了解了XML配置方式。