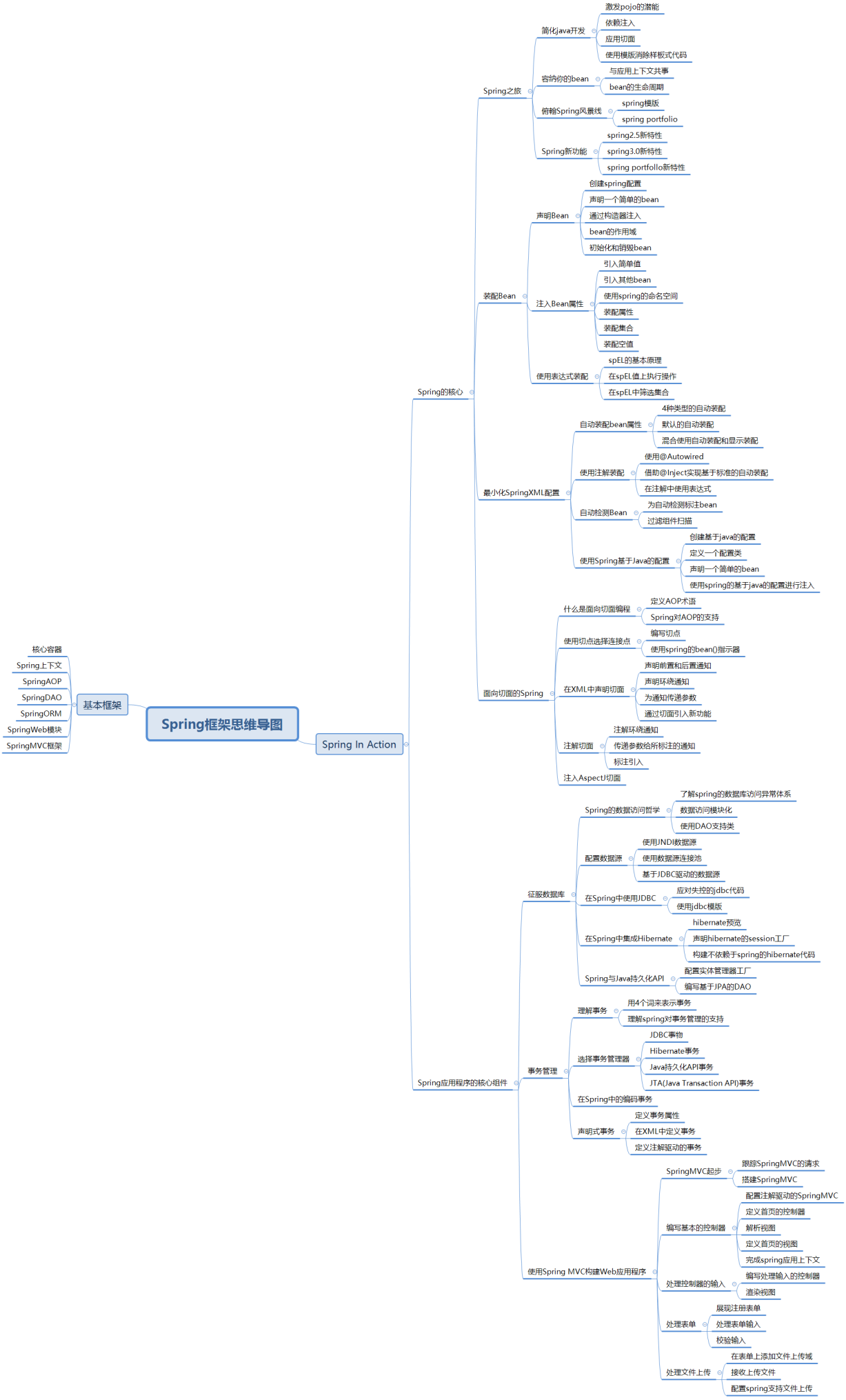
spring总结


Spring框架思维导图
基本框架
核心容器
Spring上下文
SpringAOP
SpringDAO
SpringORM
SpringWeb模块
SpringMVC框架
Spring In Action
Spring的核心
Spring之旅
简化java开发
激发pojo的潜能
依赖注入
应用切面
使用模版消除样板式代码
容纳你的bean
与应用上下文共事
bean的生命周期
俯瞰Spring风景线
spring模版
spring portfolio
Spring新功能
spring2.5新特性
spring3.0新特性
spring portfolio新特性
装配Bean
声明Bean
创建spring配置
声明一个简单的bean
通过构造器注入
bean的作用域
初始化和销毁bean
注入Bean属性
引入简单值
引入其他bean
使用spring的命名空间
装配属性
装配集合
装配空值
使用表达式装配
spEL的基本原理
在spEL值上执行操作
在spEL中筛选集合
最小化SpringXML配置
自动装配bean属性
4种类型的自动装配
默认的自动装配
混合使用自动装配和显示装配
使用注解装配
使用@Autowired
借助@Inject实现基于标准的自动装配
在注解中使用表达式
自动检测Bean
为自动检测标注bean
过滤组件扫描
使用Spring基于Java的配置
创建基于java的配置
定义一个配置类
声明一个简单的bean
使用spring的基于java的配置进行注入
面向切面的Spring
什么是面向切面编程
定义AOP术语
Spring对AOP的支持
使用切点选择连接点
编写切点
使用spring的bean()指示器
在XML中声明切面
声明前置和后置通知
声明环绕通知
为通知传递参数
通过切面引入新功能
注解切面
注解环绕通知
传递参数给所标注的通知
标注引入
注入Aspectj切面
Spring应用程序的核心组件
征服数据库
Spring的数据访问哲学
了解spring的数据库访问异常体系
数据访问模块化
使用DAO支持类
配置数据源
使用JNDI数据源
使用数据源连接池
基于JDBC驱动的数据源
在Spring中使用JDBC
应对失控的jdbc代码
使用jdbc模版
在Spring中集成Hibernate
hibernate预览
声明hibernate的session工厂
构建不依赖于spring的hibernate代码
Spring与Java持久化API
配置实体管理器工厂
编写基于JPA的DAO
事务管理
理解事务
用4个词来表示事务
理解spring对事务管理的支持
选择事务管理器
JDBC事物
Hibernate事务
Java持久化API事务
JTA(Java Transaction API)事务
在Spring中的编码事务
声明式事务
定义事务属性
在XML中定义事务
定义注解驱动的事务
使用Spring MVC构建Web应用程序
SpringMVC起步
跟踪SpringMVC的请求
搭建SpringMVC
编写基本的控制器
配置注解驱动的SpringMVC
定义首页的控制器
解析视图
定义首页的视图
完成spring应用上下文
处理控制器的输入
编写处理输入的控制器
渲染视图
处理表单
展现注册表单
处理表单输入
校验输入
处理文件上传
在表单上添加文件上传域
接收上传文件
配置spring支持文件上传

1、什么是Spring框架，Spring框架主要包含哪些模块

Spring是一个开源框架，Spring是一个轻量级的Java 开发框架。它是为了解决企业应用开发的复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许使用者选择使用哪一个组件，同时为J2EE 应用程序开发提供集成的框架。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。Spring的核心是控制反转（IoC）和面向切面（AOP）。简单来说，Spring是一个分层的full-stack(一站式) 轻量级开源框架。

主要包含的模块：

2、Spring框架的优势

1、Spring通过DI、AOP和消除样板式代码来简化企业级Java开发

2、Spring框架之外还存在一个构建在核心框架之上的庞大生态圈，它将Spring扩展到不同的领域，如Web服务、REST、移动开发以及NoSQL

3、低侵入式设计，代码的污染极低

4、独立于各种应用服务器，基于Spring框架的应用，可以真正实现Write Once,Run Anywhere的承诺

5、Spring的IoC容器降低了业务对象替换的复杂性，提高了组件之间的解耦

6、Spring的AOP允许将一些通用任务如安全、事务、日志等进行集中式处理，从而提供了更好的复用

7、Spring的ORM和DAO提供了与第三方持久层框架的的良好整合，并简化了底层的数据库访问

8、Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可自由选用Spring框架的部分或全部

3、IOC和DI是什么?

控制反转是就是应用本身不负责依赖对象的创建和维护,依赖对象的创建及维护是由外部容器负责的,这样控制权就有应用转移到了外部容器,控制权的转移就是控制反转。

依赖注入是指:在程序运行期间,由外部容器动态地将依赖对象注入到组件中如：一般，通过构造函数注入或者setter注入。

4、描述下Spring IOC容器的初始化过程

- spring核心接口
  - BeanFactory 顶层核心接口，规范子工厂，生产bean
  - BeanDefinition 封装了一切用来生产bean方式
- spring ioc、bean生命周期
- spring 扩展点
  - 就是我们程序员自己根据需要去实现的spring底层的扩展

BeanFactoryPostProcessor bean工厂后置处理器（可以用来修改bean定义的）

9个Bean的后置处理器

1. InstantiationAwareBeanPostProcessor.postProcessBeforeInstantiation
2. SmartInstantiationAwareBeanPostProcessor.determineCandidateConstructors
3. MergedBeanDefinitionPostProcessor.postProcessMergedBeanDefinition
4. SmartInstantiationAwareBeanPostProcessor.getEarlyBeanReference
5. InstantiationAwareBeanPostProcessor.postProcessAfterInstantiation

6. InstantiationAwareBeanPostProcessor.postProcessPropertyValues
7. BeanPostProcessor.postProcessBeforeInitialization
8. BeanPostProcessor.postProcessAfterInitialization
9. DestructionAwareBeanPostProcessor.requiresDestruction

怎么看源码?

1.把spring的项目源码直接下过来编译。

1. new springApplication()
2. 解析xml配置文件路径

(`AbstractApplicationContext#obtainFreshBeanFactory`)

3.创建Bean工厂

4.加载bean定义 到BeanDefinitionMap

5.调用了bean工厂的后置处理器：**invokeBeanFactoryPostProcessors（）**

>org.springframework.context.support.AbstractApplicationContext#finishBeanFactoryInitialization

>org.springframework.beans.factory.support.DefaultListableBeanFactory#preInstantiateSingletons

6. 判断是否符合生产标准（是不是抽象、懒加载、单例)

>org.springframework.beans.factory.support.AbstractBeanFactory#doGetBean

7.真正的生产Bean, 去单例池中获取看是否已经创建，如果已经创建则直接返回， 如果单例池没有就需要重新创建（为了解决循环依赖，将当前Bean加入到正在创建的标识中singletonsCurrentlyInCreation）

>org.springframework.beans.factory.support.AbstractBeanFactory#createBean

8.可以使用Bean的后置处理器直接返回自定义的Bean实例

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#doCreateBean

9. 调用doCreateBean开始真正创建Bean

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#createBeanInstance

10 实例化Bean(通过工厂方法、Supplier、Bean后置处理器:SmartInstantiationAwareBeanPostProcessor.determineCandidateConstructors,BeanDefinition的ConstructorArgumentValues) 默认调用无参构造函数来实例化

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#populateBean

11.注入属性值

>org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory#initializeBean(java.lang.String, java.lang.Object,

org.springframework.beans.factory.support.RootBeanDefinition)

12.初始化Bean 调用Awea 调用@PostConstrut Aop的动态代理也是在初始完进行生成的

13. 最终加入到单例池中

5、BeanFactory 和 FactoryBean的区别?

- **BeanFactory**是个Factory，也就是IOC容器或对象工厂，在Spring中，所有的Bean都是由BeanFactory(也就是IOC容器)来进行管理的，提供了实例化对象和拿对象的功能。

使用场景：

  - 从Ioc容器中获取Bean(byName or byType)
  - 检索Ioc容器中是否包含指定的Bean
  - 判断Bean是否为单例

- **FactoryBean**是个Bean，这个Bean不是简单的Bean，而是一个能生产或者修饰对象生成的工厂Bean,它的实现与设计模式中的工厂模式和修饰器模式类似。
- facotryBean不是在spring上文加载的时候创建的， 在getBean的时候创建的

使用场景

  - ProxyFactoryBean

6、BeanFactory和ApplicationContext的异同

BeanFactory：spring 顶层核心接口，用来生产bean的.

相同：

- Spring提供了两种不同的IOC 容器，一个是BeanFactory，另外一个是ApplicationContext，它们都是Java interface，ApplicationContext继承于BeanFactory(ApplicationContext继承ListableBeanFactory。
- 它们都可以用来配置XML属性，也支持属性的自动注入。
- BeanFactory 和 ApplicationContext 都提供了一种方式，使用getBean("bean name")获取bean。

不同：

- BeanFactory不支持国际化，即i18n，但ApplicationContext提供了对它的支持。
- BeanFactory与ApplicationContext之间的另一个区别是能够将事件发布到注册为监听器的bean。
- BeanFactory 的一个核心实现是XMLBeanFactory 而ApplicationContext 的一个核心实现是ClassPathXmlApplicationContext，Web容器的环境我们使用WebApplicationContext并且增加了getServletContext 方法。
- 如果使用自动注入并使用BeanFactory，则需要使用API注册AutoWiredBeanPostProcessor，如果使用ApplicationContext，则可以使用XML进行配置。
- 简而言之，BeanFactory提供基本的IOC和DI功能，而ApplicationContext提供高级功能，BeanFactory可用于测试和非生产使用，但ApplicationContext是功能更丰富的容器实现，应该优于

BeanFactory

7、Spring Bean 的生命周期?

总结：

（1）实例化Bean：

对于BeanFactory容器，当客户向容器请求一个尚未初始化的bean时，或初始化bean的时候需要注入另一个尚未初始化的依赖时，容器就会调用createBean进行实例化。对于ApplicationContext容器，当容器启动结束后，通过获取BeanDefinition对象中的信息，实例化所有的bean。

（2）设置对象属性（依赖注入）：

实例化后的对象被封装在BeanWrapper对象中，紧接着，Spring根据BeanDefinition中的信息 以及 通过BeanWrapper提供的设置属性的接口完成依赖注入。

（3）处理Aware接口：

接着，Spring会检测该对象是否实现了xxxAware接口，并将相关的xxxAware实例注入给Bean：

①如果这个Bean已经实现了BeanNameAware接口，会调用它实现的setBeanName(String beanId)方法，此处传递的就是Spring配置文件中Bean的id值；

②如果这个Bean已经实现了BeanFactoryAware接口，会调用它实现的setBeanFactory()方法，传递的是Spring工厂自身。

③如果这个Bean已经实现了ApplicationContextAware接口，会调用setApplicationContext(ApplicationContext)方法，传入Spring上下文；

（4）BeanPostProcessor：

如果想对Bean进行一些自定义的处理，那么可以让Bean实现了BeanPostProcessor接口，那将会调用postProcessBeforeInitialization(Object obj, String s)方法。

（5）InitializingBean 与 init-method：

如果Bean在Spring配置文件中配置了 init-method 属性，则会自动调用其配置的初始化方法。

（6）如果这个Bean实现了BeanPostProcessor接口，将会调用postProcessAfterInitialization(Object obj, String s)方法；由于这个方法是在Bean初始化结束时调用的，所以可以被应用于内存或缓存技术；

以上几个步骤完成后，Bean就已经被正确创建了，之后就可以使用这个Bean了。

（7）DisposableBean：

当Bean不再需要时，会经过清理阶段，如果Bean实现了DisposableBean这个接口，会调用其实现的destroy()方法；

（8）destroy-method：

最后，如果这个Bean的Spring配置中配置了destroy-method属性，会自动调用其配置的销毁方法。

8、Spring AOP的实现原理?

Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理。JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK动态代理的核心是InvocationHandler接口和Proxy类。

如果目标类没有实现接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

9、 声明式事务的优缺点：

- **优点**：不需要在业务逻辑代码中编写事务相关代码，只需要在配置文件配置或使用注解（@Transaction），这种方式没有侵入性。
- **缺点**：声明式事务的最细粒度作用于方法上，如果像代码块也有事务需求，只能变通下，将代码块变为方法。

10、Spring 的不同事务传播行为有哪些，干什么用的?

11、Spring 中用到了那些设计模式?

- 代理模式—在AOP中被用的比较多。
- 单例模式—在spring配置文件中定义的bean默认为单例模式。
- 模板方法—用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。
- 工厂模式—BeanFactory用来创建对象的实例。
- 适配器--spring aop
- 装饰器--spring data hashmapper
- 观察者-- spring 事件驱动模型
- 回调--Spring Aware回调接口

12、Spring如何解决循环依赖?

https://blog.csdn.net/qq_36381855/article/details/79752689

13、bean的作用域

（1）singleton：默认，每个容器中只有一个bean的实例，单例的模式由BeanFactory自身来维护。

（2）prototype：为每一个bean请求提供一个实例。

（3）request：为每一个网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收。

（4）session：与request范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效。

14、Spring框架中有哪些不同类型的事件

（1）上下文更新事件（ContextRefreshedEvent）：在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。

（2）上下文开始事件（ContextStartedEvent）：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。

（3）上下文停止事件（ContextStoppedEvent）：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。

（4）上下文关闭事件（ContextClosedEvent）：当ApplicationContext被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。

（5）请求处理事件（RequestHandledEvent）：在Web应用中，当一个http请求（request）结束触发该事件。

15、Spring通知有哪些类型

（1）前置通知（Before advice）：在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

（2）返回后通知（After returning advice）：在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

（3）抛出异常后通知（After throwing advice）：在方法抛出异常退出时执行的通知。

（4）后通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

（5）环绕通知（Around Advice）：包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。 环绕通知是最常用的一种通知类型。

16、Spring的自动装配

在spring中，对象无需自己查找或创建与其关联的其他对象，由容器负责把需要相互协作的对象引用赋予各个对象，使用autowire来配置自动装载模式。

在Spring框架xml配置中共有5种自动装配：

（1）no：默认的方式是不进行自动装配的，通过手工设置ref属性来进行装配bean。

（2）byName：通过bean的名称进行自动装配，如果一个bean的 property 与另一bean 的name 相同，就进行自动装配。

（3）byType：通过参数的数据类型进行自动装配。

（4）constructor：利用构造函数进行装配，并且构造函数的参数通过byType进行装配。

（5）autodetect：自动探测，如果有构造方法，通过 construct的方式自动装配，否则使用 byType的方式自动装配。

基于注解的方式：

使用@Autowired注解来自动装配指定的bean。在使用@Autowired注解之前需要在Spring配置文件进行配置，<context:annotation-config />。在启动spring IoC时，容器自动装载了一个AutowiredAnnotationBeanPostProcessor后置处理器，当容器扫描到@Autowied、@Resource或@Inject时，就会在IoC容器自动查找需要的bean，并装配给该对象的属性。在使用@Autowired时，首先在容器中查询对应类型的bean：

如果查询结果刚好为一个，就将该bean装配给@Autowired指定的数据；

如果查询的结果不止一个，那么@Autowired会根据名称来查找；

如果上述查找的结果为空，那么会抛出异常。解决方法时，使用required=false。

@Autowired可用于：构造函数、成员变量、Setter方法

注：@Autowired和@Resource之间的区别

(1) @Autowired默认是按照类型装配注入的，默认情况下它要求依赖对象必须存在（可以设置它required属性为false）。

(2) @Resource默认是按照名称来装配注入的，只有当找不到与名称匹配的bean才会按照类型来装配注入。