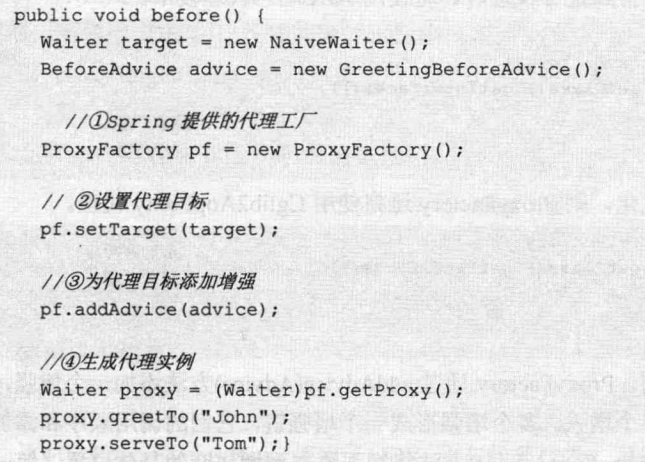
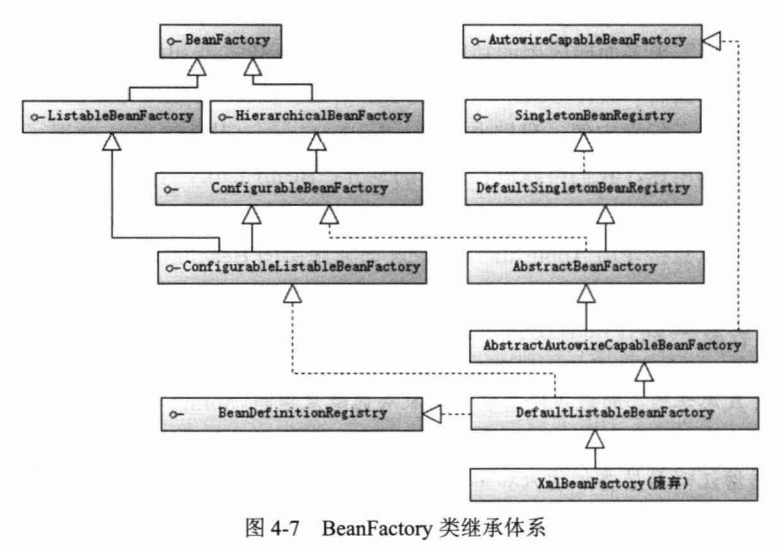
1. AOP
2. 连接点(Joinpoint)：边界，spring只支持方法的边界，执行点(方法)+方位(调用前后、异常)
3. 切点(Pointcut)：筛选目标连接点(实际是执行点)的条件
4. 增强(Advice)：织入到目标连接点上的逻辑
5. AOP不止动态代理这种实现方式，只是Spring是这样实现。另外还有在编译、类加载的时候实现AOP的方式
6. Spring提供了一个代理工厂类ProxyFactory，而具体使用JDK还是CGLIB实现动态代理可以通过相关的set方法来确定，默认是JDK动态代理。通过工厂类可以创建目标对象的代理，手动实现aop的功能(ProxyFactory源码的注释说明了目的：Factory for AOP proxies for programmatic use, rather than via declarative setup in a bean factory.)。



1. Spring另外还提供了一个代理工厂bean：ProxyFactoryBean，以便于以spring的方式创建proxy对象：



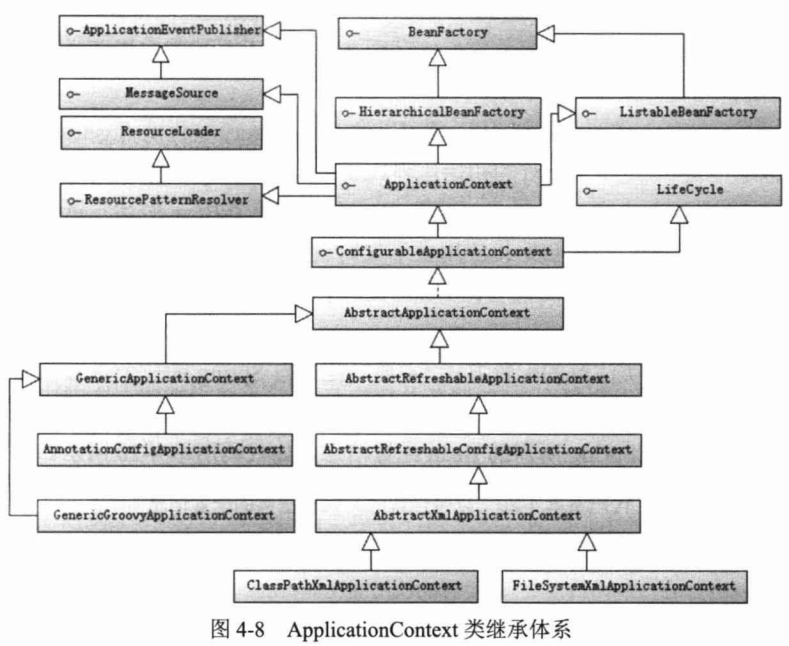
1. ProxyFactory通过getProxy()方法获取proxy，ProxyFactoryBean通过getObject()获取proxy，并且最终都是通过ProxyCreatorSupport. createAopProxy() 🡪 DefaultAopProxyFactory. createAopProxy()的方式最终获取到符合要求的proxy。
2. Spring提供的Pointcut接口，包含ClassFilter和MethodMatcher两个属性，用于定位某些类的某些方法，即目标执行点。
3. IOC
4. Spring容器/IOC容器包括面向Spring底层的BeanFactory和面向开发的ApplicationContext，一般称BeanFactory为IOC容器，而ApplicationContext为应用上下文或者spring容器。
5. 在BeanFactory的继承体系中，spring提供了多个不同功能的接口，遵循了设计模式的原则：接口隔离。这样实现类可以根据需求只实现部分接口，避免臃肿的接口设计导致实现类不得不实现不需要的方法。getBean(String beanName)是BeanFactory的主要方法。

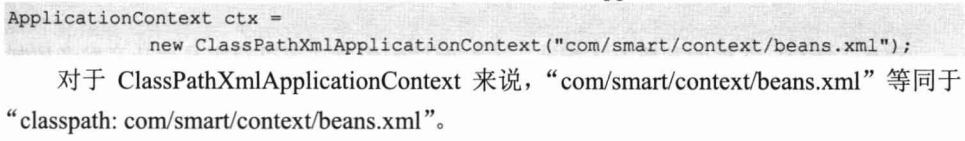


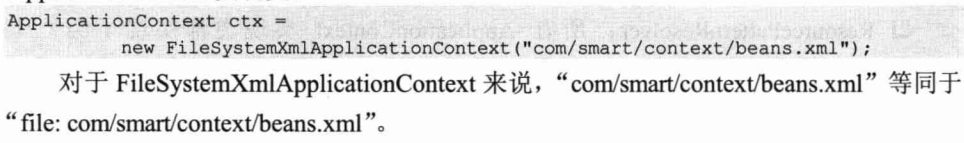
1. 另外，在BeanFactory的继承体系中，spring还提供了一些常用接口的默认实现类，例如DefaultListableBeanFactory就是在这个继承体系中一个集大成者的默认实现类，通过使用XmlBeanDefinitionReader、DefaultListableBeanFactory，可以通过编码的方式启动spring容器：



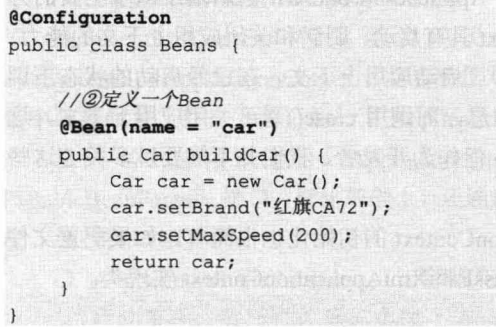
1. ApplicationContext通过继承BeanFactory的两个子接口，即ListableBeanFactory和HierarchicalBeanFactory，从而继承了BeanFactory。整个ApplicationContext的继承体系也是通过多个不同功能的接口实现复杂功能的整合，同样遵循了接口隔离的设计原则。Spring根据获取配置信息方式的不同，提供了两个重要的实现类，可以方便的启动容器：FileSystemXlmApplicationContext和ClassPathXmlApplicationContext，启动容器的方式比BeanFactory简单很多。

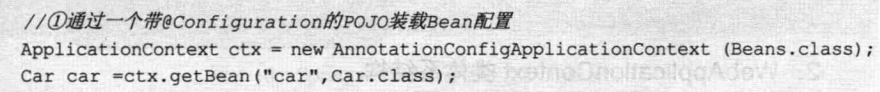




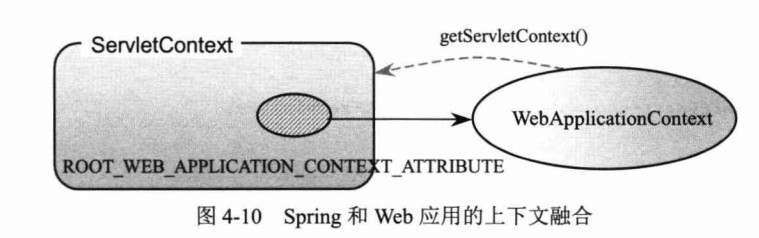


1. Spring获取bean配置信息，除了通过读取配置文件并解析配置信息，还可以通过Java编码的方式，借助@Configuration和@Bean这两个注解，可以为spring容器提供配置信息。这种方式提供配置信息，可以更直观的看到bean的创建过程。Spring专门设计了AnnotationConfigApplicationContext来获取通过注解方式设置的bean配置，创建相应的bean，使用方式也很简单，如下所示：





1. 在ApplicationContext的继承体系中，设计了针对web应用的WebApplicationcontext的继承分支，它从web根目录的目录中装载配置信息来完成初始化工作。WebApplicationContext是ServletContext的一个属性，以便于web应用环境可以访问spring容器，并且WebApplicationContext所需的配置文件参数也是作为ServletContext的属性放置在servlet容器中，这也是为什么在web.xml的配置中，需要通过配置监听器ContextLoaderListener来监听ServletContext的启动，然后启动spring容器applicationContext。通过这样方式，spring容器和servlet容器可以互相访问，如图：

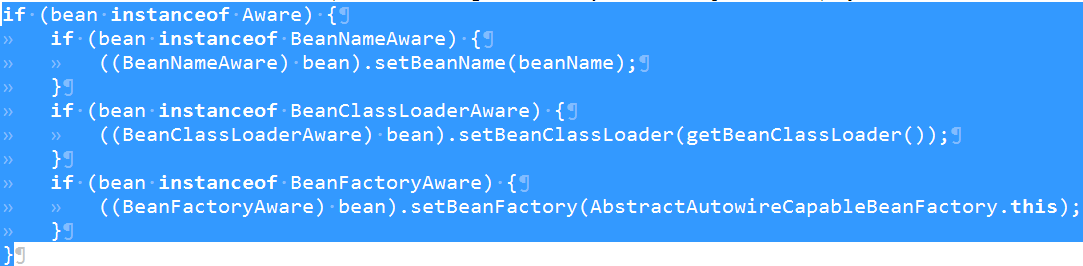


1. 综上所述，启动spring容器的方式有3种：
2. 原始的XmlBeanDefinitionReader、DefaultListableBeanFactory组合方式
3. ApplicationContext系列(FileSystemXmlApplicationContext、ClassPathXmlApplicationContext和AnnotationConfigApplicationContext)
4. WebApplicationContext通过配置监听器ContextLoaderListener
5. BeanFactory是个容器，负责bean的创建和管理，FactoryBean是个工厂bean，因此FactoryBean的实现类也是容器的管理对象. FactoryBean主要设计了一个getObject接口，一般用于创建较复杂的bean，比如ProxyFactoryBean可为容器中的bean创建代理。
6. 一般情况下，在xml中通过<bean>配置一个bean的时候，创建的bean就是class指定的类的bean，而如果这个class是实现了FactoryBean的话，那返回的是这个工厂所创建的bean，而不是工厂本身。
7. 昇山寺
8. 昇山寺
9. Spring生命周期
10. Spring定义了一个bean名称生成策略的接口BeanNameGenerator，该接口只定义了一个方法：

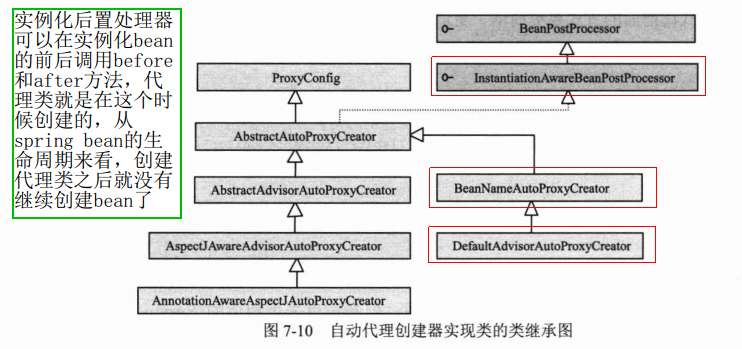
generateBeanName(BeanDefinition definition, BeanDefinitionRegistry registry)，并提供两个实现类，即：

AnnotationBeanNameGenerator和DefaultBeanNameGenerator

1. 在基于xml配置文件的bean定义中，**DefaultBeanNameGenerator**是默认的bean名字生成策略，其策略是：1)如果BeanClassName为空，但是父类名称parent不为空，则当前bean名称为parent$child；2)如果BeanClassName为空但是其factoryBean名称不为空，则当前bean名称为factoryBean$created；3)如果beanClassName不为空，但是要被注册为内部bean，则beanName为beanClassName#对应beandefinition的hash值；4) 如果beanClassName不为空，则beanName为beanClassName#编号(beanClassName#2，编号是为了区分同一个beanClassName的多个bean)
2. 在通过注解和扫描方式获取bean定义中，默认的bean名称生成策略是**AnnotationBeanNameGenerator**，其策略如下：1)如果该注解允许注入名字，且value的值不为空，则直接使用注解中value的值作为beanName；2)如果传入的不是一个注解的beandefinition或者该注解不允许注入名字或者value的值为空，则使用该bean的类名称(不是全限定)作为beanName，其中，第一字母小写，但是，如果开头连续两个字母大写，则直接使用类名称作为beanName(即第一个字母不要小写)
3. 在bean的生命周期中，创建一个bean实例的策略优先级分别是工厂创建 🡪 使用构造器注入的构造器创建 🡪 使用默认构造器创建
4. 在创建bean实例之前，InstantiationAwareBeanPostProcessor这个bean后置处理器有机会执行它的before和after处理方法，借此可以更改bean实例的创建过程，比如可以创建一个bean的代理(Give BeanPostProcessors a chance to return a proxy instead of the target bean instance.)。如果在这个接口中完成了实例的创建，比如创建了代理，那么原来bean的后续创建流程就不会执行，也就是不会创建原来的bean实例。
5. 生命周期中，设置完属性populateBean()之后，会进行bean的初始化initializeBean()，在initializeBean()方法内部，如果bean实现了相关装配接口\*\*\*Aware，则会在真正初始化bean之前执行相关的装配方法，包括setBeanName、setBeanClassLoader、setBeanFactory



1. Spring bean的生命周期描述：整体上，bean会经历创建🡪使用🡪销毁的过程。bean的创建过程比较复杂，包括1)实例化2)设置属性3)初始化。在实例化阶段，如果容器注册了实例化装配后置处理器InstantiationAwareBeanPostProcessor，则会在实例化前后调用该处理器的before和after方法；在设置属性阶段，如果容器注册了实例化装配后置处理器InstantiationAwareBeanPostProcessor，则会在设置属性之前调用postProcessPropertyValues方法检查或者修改属性值；在初始化阶段，如果bean实现了BeanNameAware、BeanClassLoaderAware、BeanFactoryAware这三个装配接口，则会在初始化之前调用这三个接口对应的方法setBeanName、setBeanClassLoader和setBeanFactory，紧接着，如果容器注册了相关的后置处理器BeanPostProcessor，则会调用这些处理器的BeforeInitialization方法来影响bean，然后，如果bean实现了接口InitializingBean，会调用其afterPropertiesSet方法来影响这个bean的状态，最后才会调用开发人员指定的init-method方法，在相关的初始化方法都执行之后，调用后置处理器的AfterInitialization方法。通过以上过程基本实现了bean的创建，对于prototype作用域的bean，在完成创建之后就会交给调用者，容器不再负责相关生命周期；而对于singleton作用域的bean，则会被缓存在容器中，在容器销毁时，如果器注册了DestructionAwareBeanPostProcessor、或者bean实现了DisposableBean接口、或者开发人员指定了destroy-method，则会执行相应的操作。
2. 在影响bean生命周期的过程中，不建议bean实现相关接口，这样bean与spring就绑定，可以通过替代方案实现。
3. 对于BeanFactory，BeanPostProcessor是要手动add到容器中，而对于ApplicationContext，则会被自动注册到容器中。
4. Spring框架中，利用bean生命周期干预bean创建的例子：AOP中自动创建代理，其类继承体系如下图所示。从图中可以看出这些自动代理创建器实现了InstantiationAwareBeanPostProcessor接口，这个接口继承了BeanPostProcessor接口，并提供了两个方法在实例化bean前后会被调用，并且在spring容器中，后置处理器会被自动注册到容器中，然后在创建bean的过程中被调用。一旦在实例化bean阶段调用了实例化后置处理器并创建了代理对象，那么原来bean的创建流程就会在实例化阶段之后被终止，不会创建对应的bean；



1. 昇山寺
2. 数据库
3. 抽象数据访问接口UserDao，实现类可以带上使用技术的名称，例如HibernateUserDao、MyBatisUserDao、JdbcUserDao，个人赞同这种方式
4. 应用
5. 配置web.xml的DispatcherServlet需要设置contextConfigLocation，在ContextLoaderListener也需要设置contextConfigLocation。在servlet容器启动时， ContextLoaderListener会首先启动root容器，在随后加载DispatcherServlet的时候，也会创建只在该servlet内部使用的spring容器，并且设置该容器的parent为root容器。局部spring容器主要是获取和该servlet相关的组件，比如Controller、ViewResovler等，并且可以获取父容器中的bean，反之不行。
6. Spring MVC
7. 配置web.xml的DispatcherServlet需要设置contextConfigLocation，在ContextLoaderListener也需要设置contextConfigLocation。在servlet容器启动时， ContextLoaderListener会首先启动root容器，在随后加载DispatcherServlet的时候，也会创建只在该servlet内部使用的spring容器，并且设置该容器的parent为root容器。局部spring容器主要是获取和该servlet相关的组件，比如Controller、ViewResovler等，并且可以获取父容器中的bean，反之不行。
8. 昇山寺
9. 昇山寺