黑客派

Spring Ioc&Aop

作者：yiranblade

原文链接：https://hacpai.com/article/1489824695768

来源网站：黑客派

<h2 id="引言">引言</h2>
控制反转，面向切面都是程序设计当中比较成熟和有用的概念，它对于高内聚低耦合，添加新功等都是非常成熟的设计方案，在 Java 中尤其以 Spring 框架对于其实现较为成熟，故在此探讨下其原。
<h2 id="Spring-Aop">Spring Aop</h2>

<b> Aop 代理其实是由 AOP 框架动态生成的一个对象，即代理对象。AOP 代理包含了目标对象的全方法，但 AOP 代理中的方法与目标对象的方法存在差异：AOP 方法在特定切入点添加了增强处理，回调了目标对象的方法。
 AOP 代理所包含的方法与目标对象的方法示意图

 首先对目标对象以及拦截器进行正确的配，以便 aopProxy 代理对象的产出，主要是通过配置 proxyfactorybean 的属性来完成或者使用 proxfactory 来实现，proxyfactorybean 使用 getObject 作为入口方法，首先对通知链进行初始化，通链封装了一系列的拦截器，拦截器通过从配置文件中读取，然后为代理对象的生成做准备。
 SprigAop 通过 aopProxyFactory 作为 aopProxy 代理对象的生产工厂，由它来负责产生相应的 aopProx 代理对象，默认使用 defaultAopFactory 来生产对象，它决定了 aopProxy 代理对象的生产策略，(jk 的 proxy 或者 gclib),aopProxy 对象的产生最后委托给 jdkdynamicAOPproxy 或者 cglib2proxy 厂来完成。
 aopproxy 代理对象拿到后，在代理的接口方法被调用的时候，并不是直接运行目对象的调用方法，而是根据 proxy 机制，改变原有的目标对象方法调用的运行轨迹。首先会触发对方的调用进行拦截，这些拦截对目标调用的功能增强提供了工作空间，拦截过程在 JDK 的 proxy 代理象中，是通过 invoke 方法完成，而 gclib 是由设置好的 callback 方法完成的。
 proxyfactorybean 的回调中，首先会根据配置来对拦截器是否与当前调用的方法相匹配进行判断，如果当前调用的法与配置的拦截器相匹配，那么相应的拦截器就会发挥作用，这个过程是一个遍历过程，会遍历在 proxy 代理对象中设置的拦截器链的所有拦截器。经过这个过程后，代理对象中定义好的拦截器链中的拦器会被逐一调用，直到整个拦截器的调用完成。在拦截器调用完成之后，然后对目标对象进行方法调。
<h2 id="Spring-Ioc">Spring IoC</h2>
Spring 容器的原理，其实就是通过解析 XML 文件，或取到用户配置的 bean，然后通过反射将些 bean 挨个放到集合中，然后对外提供一个 getBean()方法，以便我们获得这些 bean。
 首先须谈一谈 Beandefinition,Beandefinition 为管理 bean 之间的依赖关系提供了帮助只要遵循 Spring 的定义规则来提供 bean 定义信息，我们可以使用各种形式的 bean 定义信息，比较常用的是使用 XM 文件格式。在初始化 IoC 容器 的过程中，首先要定位到 bean 的定义信息，Spring 使用 resource 口来统一了 bean 的定义信息，而定位则通过 resourceLoader 来完成，如果使用上下文，applicatio context 由于本身是 defaultresourceLoader 的子类， 为用户提供了定位功能，如果是使用 beanfactory 作为 IoC 容器的话，客户需要为 beanfactory 制定响应的 resource 来完成 bean 信息的定位。<r> 容器初始化，如果使用上下文的话，需要一个对上下文进行初始化的过程，完成初始化以后，才使用 IoC 容器
 这个过程是通过构造函数的中调用 refresh 方法实现，refresh 方法相当于就是器的初始化函数，在初始化的
 过程中，主要是要对 beandefinition 信息进行载入和注册工作，是要在 IoC 容器当建立一 beandefinition 定义的数据映像，Spring 把载入功能从 IoC 容器中分离了来，通过 beandefinitionReader 完成 bean 定义信息的读取，解析和 IoC 容器内部 beandefinition 的建立，在 defaultListablebeanfactory 中 beandefintion 被维护在 hashmap 中，IoC 容器的 bea 管理和操作就是通过 beandefinition 来完成。
 在容器初始化完成之后，但初始化只是在 IoC 器内部建立了 beandefinition，具体的依赖关系还没有注入，
 容器在用户第一次向 IoC 容器请 bean 时，IoC 容器对相关的 bean 依赖进行注入，如果需要提前注入，可以通过
 lazy-init 属进行预初始化（也是上下文初始化的一部分，起到提前完成依赖注入的控制作用），在依赖注入
 完成之后，IoC 就会保持这些具备依赖关系的 bean 供用户直接使用，通过 getbean 来获得 bean,</>
<h3 id="依赖注入主要通过createbeanInstance和populatebean这两个方法-">依赖注入主要通过 ceatebeanInstance 和 populatebean 这两个方法，</h3>
<h3 id="createbeanInstance-主要是根据beandefintion来生成">createbeanInstance 主要是根据 beandefintion 来生成</h3>
bean 所包含的对象，可以通过工厂方法、构造函数或者 autowire 进行实例化。构造函数有两

实例化策略，
 一是通过 beanUtils，利用 jvm 的反射功能，二是利用 gclib 来生成。
 在始化 bean 对象之后，就需要把依赖关系设置好，主要是对 bean 对象的属性的处理过程。
 这过程是在 populatebean 中委托 beandefinitionResolver 对 beandefinition 进行解析，通过 beanwapper 的 setProertyValues 方法然后注入到 property 中。
 在 bean 的创建和对象依赖注入中需要通过依据 beandefinition 中的信息来递归的完成依赖注入，一个是
 在上下文体系当中查找要的 bean 和创建 bean 的递归调用。另一个就是在依赖注入的的时候，递归的调用
 容器的 getean 方法，得到当前 Bean 的依赖 bean，同时触发了对依赖 bean 的创建和注入。在对 bean 的属性br> 进行依赖注入的时候，解析的过程也是递归的过程。</p>