

spring源码

Spring核心 IOC实现

准备工作

监听器

1 创建事件多播器

initApplicationEventMulticaster()

new SimpleApplicationEventMulticaster(beanFactory);

- beanFactory.registerSingleton(APPLICATION_EVENT_MULTICASTER_BEAN_NAME, this.applicationEventMulticaster);

ApplicationListener接口的监听器解析过程

refresh->registerListeners

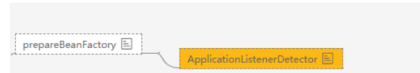
- 将监听器的名字添加到多播器中: applicationListeners

- 这里思考一个问题:

- 为什么registerListeners注册了一遍监听器在BeanPostProcessor中又添加一次呢?

- 答案: 为了懒加载的漏网之鱼!

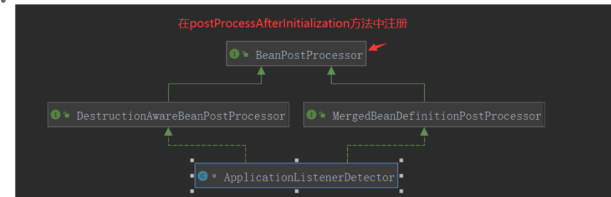
- 在prepareBeanFactory中注册了BeanPostProcessor



在初始化后的BeanPostProcessor中调用

ApplicationListenerDetector

- 将监听器添加到多播器中: applicationListeners

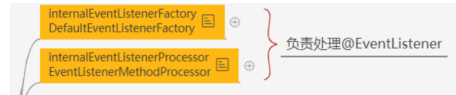


add

- AbstractApplicationContext#applicationListeners

2 @EventListener注解的解析过程

- 在创世Bean定义中注册了2个关于解析@EventListener



调用

- EventListenerMethodProcessor.afterSingletonsInstantiated
处理@EventListener

add

- ApplicationListenerMethodAdapter
AbstractApplicationContext#applicationListeners

4 publishEvent

SimpleApplicationEventMulticaster

- #multicastEvent()

接口

- ApplicationListener.onApplicationEvent

- onApplicationEvent

- 直接调用

注解

- ApplicationListener.onApplicationEvent

- ApplicationListenerMethodAdapter
onApplicationEvent

- 通过反射调用

解析BeanDefinition

- invokeBeanFactoryPostProcess

- 'AnnotationConfigApplicationContext'容器会在该方法中解析'BeanDefinition' - 不同的'spring'容器解析'BeanDefinition'的地方都不一

注册BeanPostProcessor

refresh()

- registerBeanPostProcessors

- 实际上有5个主要的后置处理器,就是通过这个方法去进行注册的 然后每个后置处理器都有他的作用。

- registerBeanPostProcessors

- AbstractBeanFactory.addBeanPostProcessor

- invokeBeanFactoryPostProcessors

- ImportAwareBeanPostProcessor

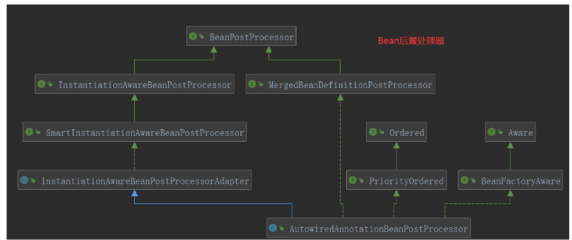
- 执行ConfigurationClassPostProcessor的,会对full的配置类使用cglib代理,且会在这放一个ImportAware

- BeanPostProcessorChecker

- 它的作用就是在postProcessAfterInitialization里检查作用在此Bean上的BeanPostProcessor够不够数,如果不够数

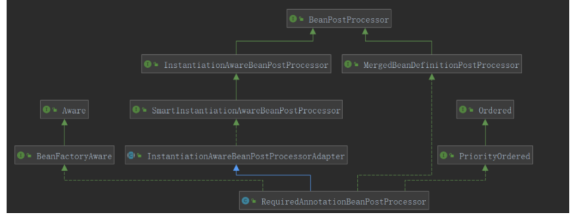
- AutowiredAnnotationBeanPostProcessor

- 解析@Autowired



RequiredAnnotationBeanPostProcessor

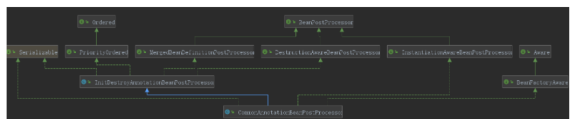
解析@Required



CommonAnnotationBeanPostProcessor

它负责解析@Resource、@WebServiceRef、@EJB三个注解。

类图



方法

```
CommonAnnotationBeanPostProcessor
static class initializer try (@SuppressWarnings...
CommonAnnotationBeanPostProcessor()
autowireResource(BeanFactory, LookupElement, String): Object
buildLazyResourceProxy(LookupElement, String): Object
buildResourceMetadata(Class<?>): InjectionMetadata
findResourceMetadata(String, Class<?>, PropertyValues): InjectionMetadata
getResource(LookupElement, String): Object
ignoreResourceType(String): void
postProcessAfterInstantiation(Object, String): boolean !InstantiationAwareBeanPostProc
postProcessBeforeInstantiation(Class<?>, String): Object !InstantiationAwareBeanPostPr
postProcessMergedBeanDefinition(RootBeanDefinition, Class<?>, String): void !InitDesti
postProcessPropertyValues(PropertyValues, PropertyDescriptor[], Object, String): Prope
setAlwaysUseIndiLookup(boolean): void
setBeanFactory(BeanFactory): void
setFallbackToDefaultTypeMatch(boolean): void
setIndiFactory(BeanFactory): void
setResourceFactory(BeanFactory): void
alwaysUseIndiLookup: boolean = false
beanFactory: BeanFactory
ejbRefClass: Class<? extends Annotation>
embeddedValueResolver: StringValueResolver
```

bean的后置处理器@处调用

Bean创建过程

refresh()

finishBeanFactoryInitialization()

为容器填充相关对象,实例化bean (实例化剩余的bean) 实现了后置处理器的bean均不在此处进行注册

preInstantiateSingletons

进行bean定义的合并, 判断是否是懒加载, 抽象类, 是否是单例 !bd.isAbstract() && !bd.isSingleton() && !bd.isLazyInit(); 如

getBean

doGetBean

实例化

属性注入

循环依赖

初始化

Spring AOP实现

@EnableAspectJAutoProxy

AspectJAutoProxyRegistrar

#registerBeanDefinitions

该类实现了 @Import的 ImportBeanDefinitionRegistrar接口 主要是注册了一个AnnotationAwareAspectJAutoProxyCreator的BeanDefinit

AopConfigUtils

#registerAspectJAnnotationAutoProxyCreatorIfNecessary

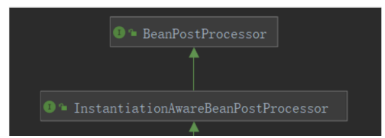
AopConfigUtils

#registerOrEscalateApcAsRequired

AnnotationAwareAspectJAutoProxyCreator

继承了AbstractAutoProxyCreator

关系结构图

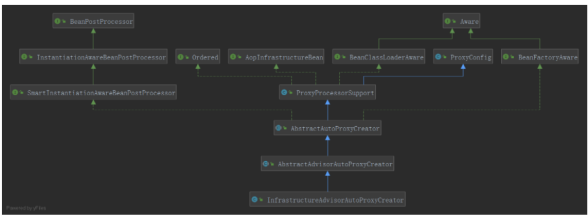


- TransactionManagementConfigurationSelector
- 使用@Import注解 为我们容器中导入组件--->TransactionManagementConfigurationSelector

- AutoProxyRegistrar
 - internalAutoProxyCreator
 - InfrastructureAdvisorAutoProxyCreator

beanName:

关系结构图

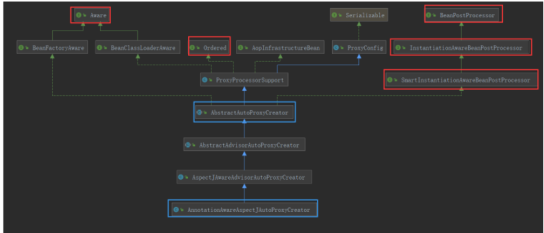


- AbstractAutoProxyCreator
 - AOP实现最-核心类
 - InstantiationAwareBeanPostProcessor
 - 属性赋值 Populate 调用的BeanPostProcess
 - postProcessBeforeInstantiation
 - 将切面、切点、通知都加载出来放入到缓存
 - SmartInstantiationAwareBeanPostProcessor
 - 实例化后循环体就会调用的
 - getEarlyBeanReference
 - 循环引用下 调用创建动态代理
 - BeanPostProcess
 - postProcessAfterInitialization
 - 创建代理

- ProxyTransactionManagementConfiguration
 - 它是个配置类
 - beanName:internalTransactionAdvisor
 - beanClass:BeanFactoryTransactionAttributeSourceAdvisor
 - 直接配置了 advisor.不用像aop那样去循环所有的bean去找advis
 - TransactionAttributeSource
 - TransactionInterceptor
 - 直接指定拦截器, 也不用转换了

- 注册BeanPostProcessors
 - refresh()
 - registerBeanPostProcessors
 - 实际上有5个主要的后置处理器, 就是通过这个方法去进行注册的 然后每个后置处理器都有他的作用。
 - registerBeanPostProcessors
 - AbstractBeanFactory.addBeanPostProcessor

开启aop时会注册: 1.'@EnableAspectJAutoProxy' 处理aop的主要逻辑



关键方法

```
public class AbstractAutoProxyCreator {
    private boolean isAdvisorPreFiltered() {
        return true;
    }
    private boolean buildAdvisors(String[] beanNames, Object[] beans) {
        // ...
    }
    private boolean createProxy(Class beanClass, String beanName, TargetSource targetSource) {
        // ...
    }
    private boolean determineCandidateConstructors(Class beanClass, String[] constructorNames, SmartInstantiationAwareBeanPostProcessor smartInstantiationAwareBeanPostProcessor) {
        // ...
    }
    private boolean getAdvisorAndAdvisorOfBean(Class beanClass, String[] constructorNames, TargetSource targetSource) {
        // ...
    }
    private boolean getBeanFactory() {
        // ...
    }
    private boolean getCacheKey(Class beanClass, String[] constructorNames, TargetSource targetSource) {
        // ...
    }
    private boolean getEarlyBeanReference(Object bean, String beanName, SmartInstantiationAwareBeanPostProcessor smartInstantiationAwareBeanPostProcessor) {
        // ...
    }
    private boolean isProxyClass(Class beanClass) {
        // ...
    }
    private boolean postProcessAfterInitialization(Object bean, String beanName, boolean instantiationAwareBeanPostProcessor) {
        // ...
    }
    private boolean postProcessBeforeInitialization(Object bean, String beanName, Object beanPostProcessor) {
        // ...
    }
    private boolean postProcessBeforeInstantiation(Class beanClass, String beanName, Object beanPostProcessor) {
        // ...
    }
    private boolean postProcessPropertyValues(PropertyValues propertyValues, PropertyDescriptor[] propertyDescriptors, Object bean, String beanName, PropertyValues instantiationAwareBeanPostProcessor) {
        // ...
    }
    private boolean resolveInterceptorNames() {
        // ...
    }
    private boolean setAdvisorAdapterRegistry(AdvisorAdapterRegistry advisorAdapterRegistry) {
        // ...
    }
    private boolean setApplyCommonInterceptorOfExt(boolean applyCommonInterceptorOfExt) {
        // ...
    }
    private boolean setBeanFactory(BeanFactory beanFactory) {
        // ...
    }
    private boolean setCustomTargetSourceCreators(TargetSourceCreator[] targetSourceCreators) {
        // ...
    }
    private boolean setFrozen(boolean frozen) {
        // ...
    }
    private boolean setInterceptorNames(String[] interceptorNames) {
        // ...
    }
}
```

bean的后置处理器9处调用

- 解析切面 (缓存通知)
 - 在真正创建doCreateBean之前调用
 - createBean
 - resolveBeforeInstantiation
 - AbstractAutoProxyCreator
 - postProcessBeforeInstantiation
 - 事务切面解析
- 创建代理
 - 在初始化后调用
 - doCreateBean
 - initializeBean

-