



Risus

昵称: Risus
园龄: 6年9个月
粉丝: 0
关注: 2
+加关注

博客园 首页 新随笔 联系 管理 订阅 XML 随笔- 15 文章- 1 评论- 0 阅读- 3839

Spring容器加载过程

1. PrepareRefresh() 刷新前的预处理

- initPropertySources()初始化一些属性设置
- getEnvironment().validateRequiredProperties(); 检验属性的合法等
- earlyApplicationEvents = new LinkedHashSet<ApplicationEvent>; 保存容器中的一些早期时间

2. obtainFreshBeanFactory(); 获取BeanFactory

- refreshBeanFactory(); 刷新【创建】BeanFactory
- getBeanFactory(); 返回刚才GenericApplicationContext创建的BeanFactory对象
- 将创建的BeanFactory对象【DefaultListableBeanFactory】返回

3. prepareBeanFactory(beanFactory); BeanFactory的预备准备工作 (BeanFactory进行一些设置)

- 设置BeanFactory的类加载器、支持表达式解析器
- 添加部分BeanPostProcessor【ApplicationContextAwareProcessor】
- 设置忽略的自动装配的接口 EnvironmentAware、EmbeddedValueResolverAware。。。
- 注册可以解析的自动装配，我们能直接在任何组件中自动注入： BeanFactory、ResourceLoader、 ApplicationEventPublisher、ApplicationContext
- 添加BeanPostProcessor,【ApplicationListenerDetector】
- 添加编译时的AspectJ支持
-
- 给BeanFactory中注册一些能用的组件： environment【ConfigurableEnvironment】、SystemProperties【Map<String, Object>】、 systemEnvironment【Map<String, Object>】

4. postProcessBeanFactory(beanFactory); BeanFactory准备工作完成后进行的后置处理工作：

- 子类通过这个方法在BeanFactory创建并预准备完成后做的进一步设置

=====以上是BeanFactoty的创建和预准备工作
=====

5. invokeBeanFactoryPostProcessor(beanFactory); 执行BeanFactoryPostProcessor

BeanFactoryPostProcessor: BeanFactory的后置处理器。在BeanFactory标准初始化之后执行的。

两个接口， BeanFactoryPostProcessor、 BeanDefinitionRegistryPostProcessor

1) 执行BeanFactoryPostProcessor的方法：

先执行BeanDefinitionRegistryPostProcessor--

- a. 获取所有的BeanDefinitionRegistryPostProcessor
- b. 先执行实现了PriorityOrdered优先级接口的BeanDefinitionRegistryPostProcessor
执行方法postProcessor.postProcessBeanDefinitionRegistry(registry)
- c. 再执行实现了Ordered顺序接口的BeanDefinitionRegistryPostProcessor

| | | | | | | |
|-------------|----|----|----|----|----|----|
| < 2024年8月 > | | | | | | |
| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
| 28 | 29 | 30 | 31 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类

Java(11)
Redis(2)
Spring(1)

随笔档案

2020年9月(3)
2020年5月(12)

阅读排行榜

1. Spring容器加载过程(1448)
2. Redis-分布式数据库CAP原理(569)
3. Phaser(346)
4. Java基础之线程4-线程同步(169)
5. Java基础之线程6-wait和sleep区别(145)

- 执行方法postProcessor.postProcessBeanDefinitionRegistry(registry)
- d. 最后执行没有实现任何优先级或者是顺序接口的BeanDefinitionRegistryPostProcessor
- 再执行
- a. 获取所有的BeanFactoryPostProcessor
- b. 先执行实现了PriorityOrdered优先级接口的BeanFactoryPostProcessor
- 执行方法postProcessor.postProcessBeanFactory(registry)
- c. 再执行实现了Ordered顺序接口的BeanFactoryPostProcessor
- 执行方法postProcessor.postProcessBeanFactory(registry)
- d. 最后执行没有实现任何优先级或者是顺序接口的BeanFactoryPostProcessor
- 执行方法postProcessor.postProcessBeanFactory(registry)
6. registerBeanPostProcessor(beanFactory); 注册BeanPostProcessor (Bean的后置处理器)
- 不同接口类型的BeanPostProcessor: 在Bean创建前后的执行时机是不一样的
- BeanPostProcessor、
- DestructionAwareBeanPostProcessor、
- InstantiationAwareBeanPostProcessor、
- SmartInstantiationAwareBeanPostProcessor、
- MergedBeanDefinitionPostProcessor
- 1) 获取所有的BeanPostProcessor; 后置处理器都默认可以通过PriorityOrdered、Ordered接口来指定优先级
- 2) 先注册PriorityOrdered优先级接口的BeanPostProcessor
- 把每一个BeanPostProcessor添加到BeanFactory中
- beanFactory.addBeanPostProcessor(postProcessor)
- 3) 再注册Ordered优先级接口的BeanPostProcessor
- 4) 然后再注册没有任何优先级接口的BeanPostProcessor
- 5) 最终注册MergedBeanDefinitionPostProcessor
- 6) 注册一个ApplicationListenerDetector: 再Bean创建完成后检查是否是ApplicationListener, 如果是则执行
- applicationContext.addApplicationListener((ApplicationListener<?>) bean)
7. initMessageSource(); 初始化MessageSource组件 (做国际化功能; 消息绑定, 消息解析)
- 1) 获取BeanFactory
- 2) 看容器中是否有id为messageSource, 类型是MessageSource的组件
- 如果有就赋值给messageSource, 如果没有就自己创建一个DelegatingMessageSource;
- MessageSource: 取出国际化配置文件中某个key的值; 能按照区域信息获取;
- 3) 把创建好的messageSource注册到容器中, 以后获取国际化配置文件的时候, 可以自动注入MessageSource, 然后可以再调用它的getMessage方法
- beanFactory.registerSingleton(MESSAGE_SOURCE_BEAN_NAME, this.messageSource)
8. initApplicationEventMulticaster(); 初始化事件派发器;
- 1) 获取BeanFactory
- 2) 从BeanFactory中获取applicationEventMulticaster的ApplicationEventMulticaster;
- 3) 如果上一步没有配置, 那就会自己创建一个SimpleApplicationEventMulticaster, 然后将创建的ApplicationEventMulticaster组件添加到BeanFactory中, 以后其他组件可以直接注入
9. onRefresh(); 留给子容器(子类)
- 1) 子类重写这个方法, 在容器刷新的时候可以自定义逻辑
10. registerListeners(); 将项目中所有ApplicationListener注册进容器中
- 1) 从容器中拿到所有的ApplicationListener
- 2) 将每个监听器添加到事件派发器中
- getApplicationEventMulticaster().addApplicationListenerBean(listenerBeanName)
- 3) 派发之前步骤产生的事件;
11. finishBeanFactoryInitialization(beanFactory); 初始化所有剩下的单实例Bean

- 1) beanFactory.preInstantiateSingletons(); 初始化所有剩下的单实例Bean
 - a. 获取容器中的所有bean, 依次进行初始化和创建对象
 - b. 获取BeanDefinition
 - c. BeanDefinition不是懒加载的,
 - a) 判断是不是FactoryBean; 是否是实现FactoryBean接口的Bean
 - b) 如果不是FactoryBean; 使用getBean(beanName)创建对象
 0. getBean(beanName) -> ioc.getBean();
 1. doGetBean(name, null, null, false)
 2. 先获取缓存中保存的单实例Bean。如果能获取到, 说明这个Bean之前被创建过 (所有创建过的单实例Bean都会被缓存起来)
 - 从singletonObjects中获取
 3. 缓存中获取不到, 开始Bean的创建对象流程;
 4. 标记当前Bean已经被创建
 5. 获取Bean的定义信息
 6. 获取当前Bean依赖的其它Bean; 如果有, 还是按照getBean() 把依赖的Bean先创建出来
 7. 启动单实例Bean的创建流程
 - 1) createBean(beanName, mbd, args);
 - 2) Object bean = resolveBeforeInstantiation(beanName, mbdToUse);
 - 让BeanPostProcessor先拦截返回代理对象;
 - InstantiationAwareBeanPostProcessor提前执行
 - 先触发: postProcessBeforeInstantiation();
 - 如果有返回值; 再触发postProcessAfterInstantiation()
 - 3) 如果前面的InstantiationAwareBeanPostProcessor没有返回代理对象; 调用 4)
 - 4) Object beanInstance = doCreateBean(beanName, mbdToUse, args) 创建Bean
 - 1) 创建Bean实例, createBeanInstance(beanName, mbd, args)
 - 利用工厂方法或者对象的构造器创建出Bean实例
 - 2) applyMergedBeanDefinitionPostProcessors(mbd, beanType, beanName)
 - 调用MergedBeanDefinitionPostProcessor的
 - postProcessMergedBeanDefinition(mbd, beanType, beanName)
 - 3) 给Bean属性赋值, 调用populateBean(beanName, mbd, instanceWrapper)
 - 赋值之前:
 - a. 拿到InstantiationAwareBeanPostProcessor后置处理器
 - 执行postProcessAfterInstantiation()
 - b. 拿到InstantiationAwareBeanPostProcessor后置处理器
 - 执行postProcessPropertyValues():
 - c. 应用Bean属性的值: 为属性利用setter方法进行赋值 (反射)
 - applyPropertyValues(beanName, mbd, bw, pvs)
 - 4) 初始化Bean; initializeBean(beanName, exposedObject, mbd);
 - a. 【执行Aware接口方法】 invokeAwareMethods(beanName, bean);执行xxxAware接口的方法
 - BeanNameAware\BeanClassLoaderAware\BeanFactoryAware
 - b) 【执行后置处理器初始化之前】
 - applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName)
 - BeanPostProcessor.postProcessBeforeInitialization();
 - c) 【执行初始化方法】 invokeInitMethods(beanName, wrappedBean, mbd);
 - a. 是否是InitializingBean接口的实现: 执行接口规定的初始化
 - b. 是否自定义初始化方法
 - d) 【执行后置处理器初始化之后】 applyBeanPostProcessorsAfterInitialization
 - BeanPostProcessor.postProcessAfterInitialization();
 - e) 注册Bean的销毁方法

5) 将创建的Bean添加到缓存中 - singletonObjects [Map对象]

IOC容器就是这些Map; 很多的Map里保存了单实例Bean, 环境信息、、、

完成以后; 再来检查所有Bean是否是SmartInitializingSingleton接口的实现类,

如未定, 就执行afterSingletonsInstantiated();

12. finishRefresh(); 完成BeanFactory的初始化创建工作; IOC容器就创建完成;

1) initLifecycleProcessor(); 初始化和生命周期相关的后置处理器; LifecycleProcessor

默认从容器中找是否有lifecycleProcessor的组件【LifecycleProcessor】; 如果没有, 创建/使用默认的生命周期组件 new DefaultLifecycleProcessor(); 再加入到容器中;

写一个 LifecycleProcessor的实现类, 可以在BeanFactory的下面两个方法刷新和关闭前后进行拦截调用

onRefresh()

onClose()

2) getLifecycleProcessor().onRefresh();

拿到前面定义的生命周期处理器 (BeanFactory) ; 回调.onRefresh();

3) publishEvent(new ContextRefreshedEvent(this)); 发布容器刷新完成时间;

4) liveBeansView.registerApplicationContext();

=====总结=====

1) spring容器在启动的时候, 先回保存所有注册进来的Bean的定义信息

a. xml注册bean: <bean>

b. 注解注册Bean: @Service、@Repository、@Component、@Bean、xxx

2) Spring容器会在合适的时机创建这些Bean

a.用到这个bean的时候, 利用getBean创建Bean, 创建好以后保存在容器中。

b. 统一创建剩下的所有bean的时候: finishBeanFactoryInitialization();

3) 后置处理器:

a. 每一个bean创建完成, 都会使用各种后置处理器处理, 来增强bean的功能;

例如: AutoWiredAnnotationBeanPostProcessor: 处理自动注入功能

AnnotationAwareAspectJAutoProxyCreator: 来做AOP功能


4) 事件驱动模型:

ApplicationListener: 事件监听

ApplicationEventMulticaster: 事件派发

分类: Spring





Risus

粉丝 - 0 关注 - 2

+加关注

0

0

升级成为会员

» 下一篇: [Redis-分布式数据库CAP原理](#)

posted @ 2020-05-06 19:02 Risus 阅读(1448) 评论(0) 编辑 收藏 举报

会员力量, 点亮园子希望

[刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页

- 【推荐】轻量又高性能的 SSH 工具 IShell: AI 加持, 快人一步
- 【推荐】100%开源! 大型工业跨平台软件C++源码提供, 建模, 组态!
- 【推荐】天翼云2核4G云服务器加购同规格数据库, 3个月仅售21.7元
- 【推荐】2024阿里云超值优品季, 精心为您准备的上云首选必备产品
- 【推荐】会员力量, 点亮园子希望, 期待您升级成为博客园VIP会员



编辑推荐：

- 如何做好团队开发中的 CodeReview（代码评审）？
- 可以调用 Null 的实例方法吗？
- 一文搞懂应用架构的3个核心概念
- 深入理解单元测试：技巧与最佳实践
- 数据裂变，数据库高可用架构设计实践

阅读排行：

- 寻访中国100家.NET中大型企业 —— 第二站：苏州行
- 网易云音乐故障 2 小时，这次到底谁背锅？（今天记得领补偿）
- 仅花一天时间，开发者重制 32 年前经典 Mac 应用！
- 方法的三种调用形式
- .NET 9 优化，抢先体验 C# 13 新特性

Copyright © 2024 Risus
Powered by .NET 8.0 on Kubernetes