

# 猿灯塔，做程序员的引导者

www.vuandenata.com

## 1. spring 启动refresh():

```
1 public void refresh() throws BeansException, IllegalStateException {
2     synchronized (this.startupShutdownMonitor) {
3         // Prepare this context for refreshing.
4         prepareRefresh();
5
6         // Tell the subclass to refresh the internal bean factory.
7         //主要是创建beanFactory, 同时加载配置文件.xml中的beanDefinition
8         //通过String[] configLocations = getConfigLocations()获取资源路径, 然后加载beanDefinition
9         ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();
10
11         // Prepare the bean factory for use in this context.
12         // 给beanFactory注册一些标准组件, 如ClassLoader, BeanPostProcess
13         prepareBeanFactory(beanFactory);
14
15         try {
16             // Allows post-processing of the bean factory in context subclasses.
17             //提供给子类实现一些postProcess的注册, 如AbstractRefreshableWebApplicationContext;
18             //postProcess, 真对web进行生命周期管理的Scope, 通过registerResolvableDependency()为
19             postProcessBeanFactory(beanFactory);
20
21             // Invoke factory processors registered as beans in the context.
22             //调用所有BeanFactoryProcessor的postProcessBeanFactory()方法
23             invokeBeanFactoryPostProcessors(beanFactory);
24
25             // Register bean processors that intercept bean creation.
26             //注册BeanPostProcessor, BeanPostProcessor作用是用于拦截Bean的创建
27             registerBeanPostProcessors(beanFactory);
28
29             // Initialize message source for this context.
30             //初始化消息Bean
31             initMessageSource();
32
33             // Initialize event multicaster for this context.
34             //初始化上下文的事件多播组件, ApplicationEvent触发时由multicaster通知给ApplicationLi
35             initApplicationEventMulticaster();
36
37             // Initialize other special beans in specific context subclasses.
38             //ApplicationContext初始化一些特殊的bean
39             onRefresh();
40
41             // Check for listener beans and register them.
42             //注册事件监听器, 事件监听Bean统一注册到multicaster里头, ApplicationEvent事件触发后会
43             registerListeners();
44
45             // Instantiate all remaining (non-lazy-init) singletons.
46             //非延迟加载的单例Bean实例化
47             finishBeanFactoryInitialization(beanFactory);
48     }
```

# 猿灯塔，做程序员的引导者

www.vuandenata.com

```
49         // Last step: publish corresponding event.
50         finishRefresh();
51     } catch (BeansException ex) {
52         if (logger.isWarnEnabled()) {
53             logger.warn("Exception encountered during context initialization - " +
54                 "cancelling refresh attempt: " + ex);
55         }
56
57         // Destroy already created singletons to avoid dangling resources.
58         destroyBeans();
59
60         // Reset 'active' flag.
61         cancelRefresh(ex);
62
63         // Propagate exception to caller.
64         throw ex;
65     } finally {
66         // Reset common introspection caches in Spring's core, since we
67         // might not ever need metadata for singleton beans anymore...
68         resetCommonCaches();
69     }
70 }
71 }
```

## 2. 声明式事务源码：

- a. @EnableTransactionManagement：利用TransactionManagementConfigurationSelector给容器中导入两个组件：
  - i. AutoProxyRegistrar；
  - ii. ProxyTransactionManagementConfiguration。
- b. AutoProxyRegistrar：给容器注入InfrastructureAdvisorAutoProxyCreator组件。利用后置处理器在对象创建以后，包装对象，返回一个代理对象（增强器），代理对象执行目标方法时执行拦截器链。
- c. ProxyTransactionManagementConfiguration：给容器注册增强器。
  - i. 设置事务注解信息：AnnotationTransactionAttributeSource解析事务注解；
  - ii. 设置事务拦截器：TransactionInterceptor（是一个MethodInterceptor），保存了事务属性信息和事务管理器。在目标方法执行的时候，执行拦截器链

```
1 protected Object invokeWithinTransaction(Method method, Class<?> targetClass, final Invoca
2     throws Throwable {
3
4     // If the transaction attribute is null, the method is non-transactional.
5     final TransactionAttribute txAttr = getTransactionAttributeSource().getTransactionAttri
6     final PlatformTransactionManager tm = determineTransactionManager(txAttr);
7     final String joinpointIdentification = methodIdentification(method, targetClass, txAttr
8
9     if (txAttr == null || !(tm instanceof CallbackPreferringPlatformTransactionManager)) {
10         // Standard transaction demarcation with getTransaction and commit/rollback co
11         TransactionInfo txInfo = createTransactionIfNecessary(tm, txAttr, joinpointIden
12         Object retVal = null;
13         try {
14             // This is an around advice: Invoke the next interceptor in the chain.
15             // This will normally result in a target object being invoked.
16             retVal = invocation.proceedWithInvocation();
```

# 猿灯塔，做程序员的引导者

www.vuandentata.com

```
17     }
18     catch (Throwable ex) {
19         // target invocation exception
20         completeTransactionAfterThrowing(txInfo, ex);
21         throw ex;
22     }
23     finally {
24         cleanupTransactionInfo(txInfo);
25     }
26     commitTransactionAfterReturning(txInfo);
27     return retVal;
28 }
29 }
```

### 3. newrelic源码：

javaagent、asm操控字节码，类加载时在指定类前后增加拦截器。

openresty + lua开发高性能web服务。转发至kafka