1. spring 启动refresh()：

```
public void refresh() throws BeansException, IllegalStateException {
    synchronized (this.startupShutdownMonitor) {
        // Prepare this context for refreshing.
        prepareRefresh();

        // Tell the subclass to refresh the internal bean factory.
        //主要是创建beanFactory，同时加载配置文件.xml中的beanDefinition
        //通过String[] configLocations = getConfigLocations()获取资源路径，然后加载beanDefinit
        ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();

         // Prepare the bean factory for use in this context.
         // 给beanFactory注册一些标准组件，如ClassLoader，BeanPostProcess
         prepareBeanFactory(beanFactory);

         try {
             // Allows post-processing of the bean factory in context subclasses.
             //提供给子类实现一些postProcess的注册，如AbstractRefreshableWebApplicationContext
             //postProcess，真对web进行生命周期管理的Scope，通过registerResolvableDependency()方
             postProcessBeanFactory(beanFactory);

             // Invoke factory processors registered as beans in the context.
             //调用所有BeanFactoryProcessor的postProcessBeanFactory()方法
             invokeBeanFactoryPostProcessors(beanFactory);

             // Register bean processors that intercept bean creation.
             //注册BeanPostProcessor，BeanPostProcessor作用是用于拦截Bean的创建
             registerBeanPostProcessors(beanFactory);

             // Initialize message source for this context.
             //初始化消息Bean
             initMessageSource();

             // Initialize event multicaster for this context.
             //初始化上下文的事件多播组件，ApplicationEvent触发时由multicaster通知给ApplicationLi
             initApplicationEventMulticaster();

             // Initialize other special beans in specific context subclasses.
             //ApplicationContext初始化一些特殊的bean
             onRefresh();

             // Check for listener beans and register them.
             //注册事件监听器，事件监听Bean统一注册到multicaster里头，ApplicationEvent事件触发后会由
             registerListeners();

             // Instantiate all remaining (non-lazy-init) singletons.
             //非延迟加载的单例Bean实例化
             finishBeanFactoryInitialization(beanFactory);

```

```
            // Last step: publish corresponding event.
            finishRefresh();
        }catch (BeansException ex) {
            if (logger.isWarnEnabled()) {
            logger.warn("Exception encountered during context initialization - " +
            "cancelling refresh attempt: " + ex);
            }

            // Destroy already created singletons to avoid dangling resources.
            destroyBeans();

            // Reset 'active' flag.
            cancelRefresh(ex);

            // Propagate exception to caller.
            throw ex;
        }finally {
            // Reset common introspection caches in Spring's core, since we
            // might not ever need metadata for singleton beans anymore...
            resetCommonCaches();
        }
    }
}
```

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

```
protected Object invokeWithinTransaction(Method method, Class<?> targetClass, final Invoca
    throws Throwable {

    // If the transaction attribute is null, the method is non-transactional.
    final TransactionAttribute txAttr = getTransactionAttributeSource().getTransactionAttr
    final PlatformTransactionManager tm = determineTransactionManager(txAttr);
    final String joinpointIdentification = methodIdentification(method, targetClass, txAtt

    if (txAttr == null || !(tm instanceof CallbackPreferringPlatformTransactionManager)) {
            // Standard transaction demarcation with getTransaction and commit/rollback ca
            TransactionInfo txInfo = createTransactionIfNecessary(tm, txAttr, joinpointIde
            Object retVal = null;
            try {
                    // This is an around advice: Invoke the next interceptor in the chain.
                    // This will normally result in a target object being invoked.
                    retVal = invocation.proceedWithInvocation();
```

```
            }
            catch (Throwable ex) {
                    // target invocation exception
                    completeTransactionAfterThrowing(txInfo, ex);
                    throw ex;
            }
            finally {
                    cleanupTransactionInfo(txInfo);
            }
            commitTransactionAfterReturning(txInfo);
            return retVal;
    }
}
```

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