03-Feature-K8S ecosystem awareness

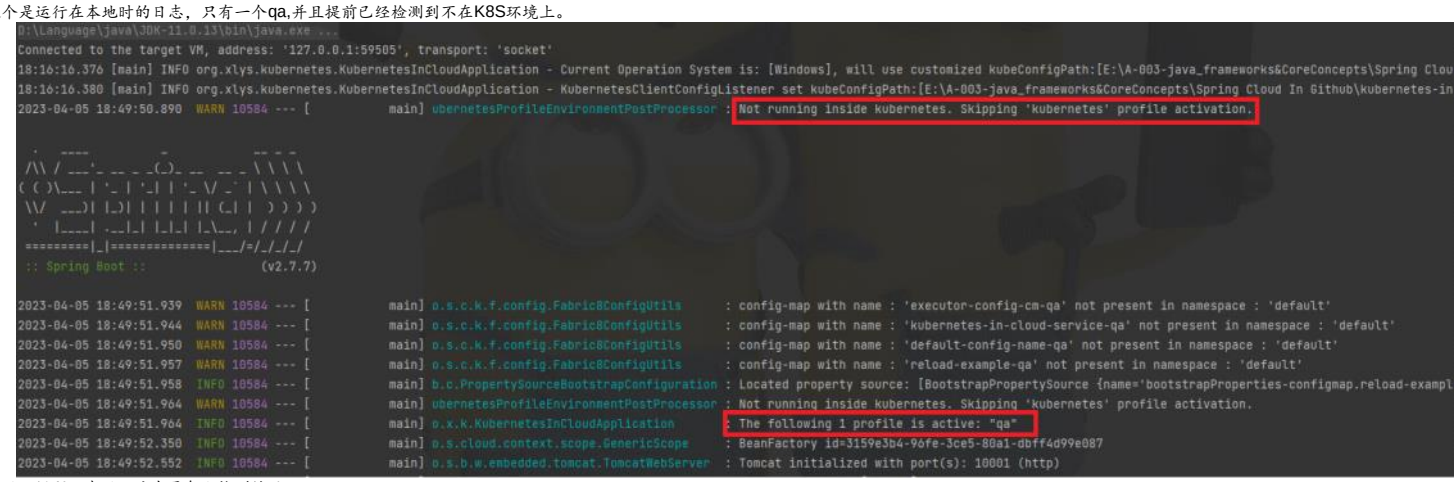
Monday, 3 April 2023 7:51 PM

01-我们的一个Spring应用程序是如何检测自己是否运行在Kubernetes平台的呢？

我认为其实最主要的就是一个profile来区别吧。【前提是我们的配置是根据profile来加载的，如果先加载profile信息，才有可能不漏掉配置文件信息，基于这个前提！后续可以研究一下SpringBoot加载配置的一套流程】

这个是运行在真实kubernetes平台时的日志，里面有kubernetes这个profile。

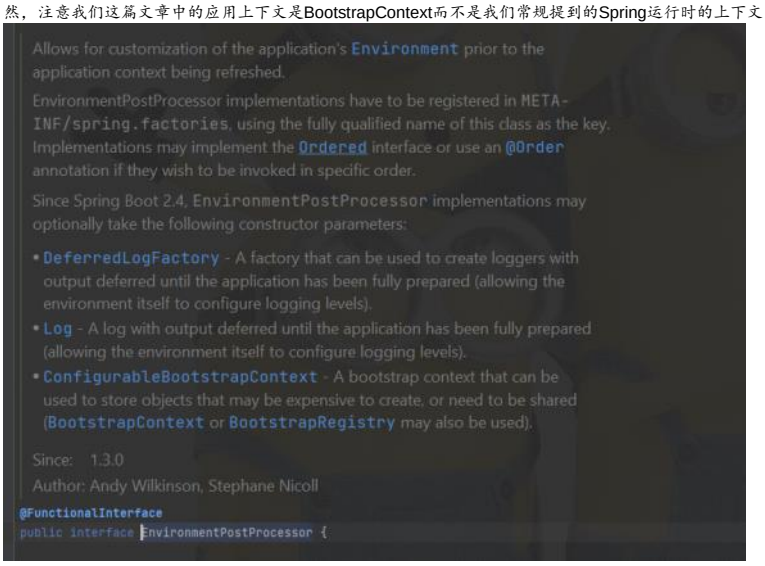
这个是运行在本地时的日志，只有一个qa,并且提前已经检测到不在K8S环境上。



好了，问题就来了，这个是怎么检测的呢？

其实主要基于一个后置处理器Fabric8ProfileEnvironmentPostProcessor以及它的父类AbstractKubernetesProfileEnvironmentPostProcessor提供的功能来实现。

首先我们要认识一个SpringBoot提供的接口：EnvironmentPostProcessor，主要用来在应用上下文刷新之前对environment做自定义配置。【当然，注意我们这篇文章中的应用上下文是BootstrapContext而不是我们常规提到的Spring运行时的上下文】



Application启动之后，我们知道在environment准备完毕的时候会发布一个ApplicationEnvironmentPreparedEvent事件。对应的监听器会监听此事件并执行对应的操作。SpringBoot中就有这么一个监听器EnvironmentPostProcessorApplicationListener。

```java
@Override
public List<EnvironmentPostProcessor> getEnvironmentPostProcessors(DeferredLogFactory logFactory,
        ConfigurableBootstrapContext bootstrapContext) {
    Instantiator<EnvironmentPostProcessor> instantiator = new Instantiator<>(EnvironmentPostProcessor.class,
            (parameters) -> {
                parameters.add(DeferredLogFactory.class, logFactory);
                parameters.add(Log.class, logFactory::getLog);
                parameters.add(ConfigurableBootstrapContext.class, bootstrapContext);
                parameters.add(BootstrapContext.class, bootstrapContext);
                parameters.add(BootstrapRegistry.class, bootstrapContext);
            });
    return (this.classes != null) ? instantiator.instantiateTypes(this.classes)
            : instantiator.instantiate(this.classLoader, this.classNames);
}

}
```

。

CloudPlatformDetectedUtils.java | EnvironmentPostProcessorApplicationListener.java | EnvironmentPostProcessor.java | Abstra

```java
    }

    @Override
    public boolean supportsEventType(Class<? extends ApplicationEvent> eventType) {
        return ApplicationEnvironmentPreparedEvent.class.isAssignableFrom(eventType)
                || ApplicationPreparedEvent.class.isAssignableFrom(eventType)
                || ApplicationFailedEvent.class.isAssignableFrom(eventType);
    }

    @Override
    public void onApplicationEvent(ApplicationEvent event) {
        if (event instanceof ApplicationEnvironmentPreparedEvent) {
            onApplicationEnvironmentPreparedEvent((ApplicationEnvironmentPreparedEvent) event);
        }
        if (event instanceof ApplicationPreparedEvent) {
            onApplicationPreparedEvent();
        }
        if (event instanceof ApplicationFailedEvent) {
            onApplicationFailedEvent();
        }
    }

    private void onApplicationEnvironmentPreparedEvent(ApplicationEnvironmentPreparedEvent event) {
        ConfigurableEnvironment environment = event.getEnvironment();
        SpringApplication application = event.getSpringApplication();
        for (EnvironmentPostProcessor postProcessor : getEnvironmentPostProcessors(application.getResourceL
                event.getBootstrapContext())) {
            postProcessor.postProcessEnvironment(environment, application);
        }
    }
```

Choose Implementation of **EnvironmentPostProcessor.postProcessEnvironment(ConfigurableEnvironment, SpringApplication)** (14 found)

AbstractKubernetesProfileEnvironmentPostProcessor (org.springframework.cloud.kubernetes.commons.profile)	Gradle: org.springframework.cloud:spring-cloud-kubernetes-comm
CachedRandomPropertySourceEnvironmentPostProcessor (org.springframework.cloud.util.random)	Gradle: org.springframework.cloud:spr
CloudFoundryVcapEnvironmentPostProcessor (org.springframework.boot.cloud)	Gradle: org.springfr
ConfigDataEnvironmentPostProcessor (org.springframework.boot.context.config)	Gradle: org.springfr
ConfigDataMissingEnvironmentPostProcessor (org.springframework.cloud.commons)	Gradle: org.springframework.cloud:spring-c
ConfigFileApplicationListener (org.springframework.boot.context.config)	Gradle: org.springfr
DebugAgentEnvironmentPostProcessor (org.springframework.boot.reactor)	Gradle: org.springfr
DecryptEnvironmentPostProcessor (org.springframework.cloud.bootstrap.encrypt)	Gradle: org.springframework.cloud:spr
HostInfoEnvironmentPostProcessor (org.springframework.cloud.client)	Gradle: org.springframework.cloud:spring-c
IntegrationPropertiesEnvironmentPostProcessor (org.springframework.boot.autoconfigure.integration)	Gradle: org.springframework.boot:spring-boot-au
RandomValuePropertySourceEnvironmentPostProcessor (org.springframework.boot.env)	Gradle: org.springfr
SpringApplicationJsonEnvironmentPostProcessor (org.springframework.boot.env)	Gradle: org.springfr
SpringBootTestRandomPortEnvironmentPostProcessor (org.springframework.boot.test.web)	Gradle: org.springframework.
SystemEnvironmentPropertySourceEnvironmentPostProcessor (org.springframework.boot.env)	Gradle: org.springfr

那么到了AbstractKubernetesProfileEnvironmentPostProcessor执行时，就是来解决两个问题：namespace和profile。
逻辑如下：

```java
@Override
public void postProcessEnvironment(ConfigurableEnvironment environment, SpringApplication application) {

    application.addInitializers(ctx -> LOG.replayTo(AbstractKubernetesProfileEnvironmentPostProcessor.class));

    boolean kubernetesEnabled = environment.getProperty( key: "spring.cloud.kubernetes.enabled", Boolean.class,  defaultValue: true);
    if (!kubernetesEnabled) {
        return;
    }
    addNamespaceFromServiceAccountFile(environment);
    addKubernetesProfileIfMissing(environment);
}
```

- NameSpace：拿到一个serviceAccountNamespacePath文件路径【这个文件路径首先去根据spring.cloud.kubernetes.client.serviceAccountNamespacePath拿，没有的话默认是去拿/var/run/secrets/kubernetes.io/serviceaccount/namespace（这个是K8S集群存储时提供的的默认路径）】，尝试解析这个文件并获得namespace信息。如果最终拿到了namespace信息，封装成一个叫做KUBERNETES_NAMESPACE_PROPERTY_SOURCE的PropertySource。由后续组件去拿对应配置。

```
private void addNamespaceFromServiceAccountFile(ConfigurableEnvironment environment) {
    String serviceAccountNamespacePathString = environment.getProperty(NAMESPACE_PATH_PROPERTY,
            SERVICE_ACCOUNT_NAMESPACE_PATH);
    String namespace = KubernetesNamespaceProvider
            .getNamespaceFromServiceAccountFile(serviceAccountNamespacePathString);
    if (StringUtils.hasText(namespace)) {
        environment.getPropertySources().addLast(new MapPropertySource(PROPERTY_SOURCE_NAME,
                Collections.singletonMap(NAMESPACE_PROPERTY, namespace)));
    }

}
```

- Profile：使用Fabric8ProfileEnvironmentPostProcessor#isInsideKubernetes(Environment)的方法去检查是否运行在K8S平台，如果是就把"kubernetes"这个profile加到激活的profile列表中。没有就算了。至于上面这个isInsideKubernetes主要考的是两个，一个系统环境变量，另一个是PodUtils检测。这里提供以下大致代码逻辑。

```
public class Fabric8ProfileEnvironmentPostProcessor extends AbstractKubernetesProfileEnvironmentPostProcessor {

    @Override
    protected boolean isInsideKubernetes(Environment environment) {
        try (DefaultKubernetesClient client = new DefaultKubernetesClient()) {
            Fabric8PodUtils podUtils = new Fabric8PodUtils(client);
            return environment.containsProperty(Fabric8PodUtils.KUBERNETES_SERVICE_HOST)
                    || podUtils.isInsideKubernetes();
        }
    }

}
```