

### 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。

```

:: Spring Boot :: (v2.7.7)

2023-04-05 00:14:12.340 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'executor-config-cm-kubernetes' not present in namespace : 'default'
2023-04-05 00:14:12.351 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'executor-config-cm-qa' not present in namespace : 'default'
2023-04-05 00:14:12.351 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'kubernetes-in-cloud-service-kubernetes' not present in namespace : 'default'
2023-04-05 00:14:13.655 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'kubernetes-in-cloud-service-qa' not present in namespace : 'default'
2023-04-05 00:14:13.752 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'default-config-name-kubernetes' not present in namespace : 'default'
2023-04-05 00:14:13.752 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'default-config-name-qa' not present in namespace : 'default'
2023-04-05 00:14:13.856 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'reload-example-kubernetes' not present in namespace : 'default'
2023-04-05 00:14:13.854 WARN 1 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'reload-example-qa' not present in namespace : 'default'
2023-04-05 00:14:14.053 INFO 1 --- [main] b.c.PropertySourceBootstrapConfiguration : Located property source: [BootstrapPropertySource {name='bootstrapProperties-configmap.reload-example-config-name.default'}, BootstrapPropertySource {name='bootstrapProperties-configmap.kubernetes-in-cloud-service.default'}, BootstrapPropertySource {name='bootstrapProperties-configmap.default'}]
2023-04-05 00:14:14.355 INFO 1 --- [main] o.x.k.kubernetes.InCloudApplication : the following 2 profiles are active: "kubernetes", "qa"
2023-04-05 00:14:51.853 INFO 1 --- [main] o.s.c.cloud.context.scope.GenericScope : BeanFactory id=1085741b-f46d-3768-9962-ed934a47036e
```

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。

```

D:\Language\java\jdk-11.0.13\bin\java.exe ...
Connected to the target VM, address: '127.0.0.1:59505', transport: 'socket'
18:16:16.376 [main] INFO org.xlys.kubernetes.KubernetesInCloudApplication - Current Operation System is: [Windows], will use customized kubeConfigPath:[E:\A-003-java_frameworks&CoreConcepts\Spring Cloud In Github\kubernetes-in-cloud-service\src\main\resources\kubeconfig.yaml]
18:16:16.380 [main] INFO org.xlys.kubernetes.KubernetesInCloudApplication - KubernetesClientConfigListener set kubeConfigPath:[E:\A-003-java_frameworks&CoreConcepts\Spring Cloud In Github\kubernetes-in-cloud-service\src\main\resources\kubeconfig.yaml]
2023-04-05 18:49:50.890 WARN 10584 --- [main] ubernetesProfileEnvironmentPostProcessor : Not running inside kubernetes. Skipping 'kubernetes' profile activation.

:: Spring Boot :: (v2.7.7)

2023-04-05 18:49:51.939 WARN 10584 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'executor-config-cm-qa' not present in namespace : 'default'
2023-04-05 18:49:51.944 WARN 10584 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'kubernetes-in-cloud-service-qa' not present in namespace : 'default'
2023-04-05 18:49:51.950 WARN 10584 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'default-config-name-qa' not present in namespace : 'default'
2023-04-05 18:49:51.957 WARN 10584 --- [main] o.s.c.k.f.config.Fabric8ConfigUtils : config-map with name : 'reload-example-qa' not present in namespace : 'default'
2023-04-05 18:49:51.958 INFO 10584 --- [main] b.c.PropertySourceBootstrapConfiguration : Located property source: [BootstrapPropertySource {name='bootstrapProperties-configmap.reload-example-config-name.default'}, BootstrapPropertySource {name='bootstrapProperties-configmap.kubernetes-in-cloud-service.default'}, BootstrapPropertySource {name='bootstrapProperties-configmap.default'}]
2023-04-05 18:49:51.964 WARN 10584 --- [main] ubernetesProfileEnvironmentPostProcessor : Not running inside kubernetes. Skipping 'kubernetes' profile activation.
2023-04-05 18:49:51.964 INFO 10584 --- [main] o.x.k.kubernetes.InCloudApplication : The following 1 profile is active: "qa"
2023-04-05 18:49:52.350 INFO 10584 --- [main] o.s.c.cloud.context.scope.GenericScope : BeanFactory id=3159e3b4-9afe-3ce5-80a1-dbf4d99e087
2023-04-05 18:49:52.552 INFO 10584 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 10001 (http)
```

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】

```

Allows for customization of the application's Environment prior to the application context being refreshed.

EnvironmentPostProcessor implementations have to be registered in META-INF/spring.factories, using the fully qualified name of this class as the key. Implementations may implement the Ordered interface or use an @Order annotation if they wish to be invoked in specific order.

Since Spring Boot 2.4, EnvironmentPostProcessor implementations may optionally take the following constructor parameters:

• DeferredLogFactory - A factory that can be used to create loggers with output deferred until the application has been fully prepared (allowing the environment itself to configure logging levels).
• Log - A log with output deferred until the application has been fully prepared (allowing the environment itself to configure logging levels).
• ConfigurableBootstrapContext - A bootstrap context that can be used to store objects that may be expensive to create, or need to be shared (BootstrapContext or BootstrapRegistry may also be used).

Since: 1.3.0
Author: Andy Wilkinson, Stephane Nicoll

@FunctionalInterface
public interface EnvironmentPostProcessor {
```

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。

```

@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);
}
}

```

```

@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.getResourceLoader(),
        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-commons
CachedRandomPropertySourceEnvironmentPostProcessor	(org.springframework.cloud.util.random)	Gradle: org.springframework.cloud:spring-cloud-util
CloudFoundryVcapEnvironmentPostProcessor	(org.springframework.boot.cloud)	Gradle: org.springframework.boot:spring-boot-cloud
ConfigDataEnvironmentPostProcessor	(org.springframework.boot.context.config)	Gradle: org.springframework.boot:spring-boot-context-configuration
ConfigDataMissingEnvironmentPostProcessor	(org.springframework.cloud.commons)	Gradle: org.springframework.cloud:spring-cloud-commons
ConfigFileApplicationListener	(org.springframework.boot.context.config)	Gradle: org.springframework.boot:spring-boot-context-configuration
DebugAgentEnvironmentPostProcessor	(org.springframework.boot.reactor)	Gradle: org.springframework.boot:spring-boot-reactor
DecryptEnvironmentPostProcessor	(org.springframework.cloud.bootstrap.encrypt)	Gradle: org.springframework.cloud:spring-cloud-bootstrap-encrypt
HostInfoEnvironmentPostProcessor	(org.springframework.cloud.client)	Gradle: org.springframework.cloud:spring-cloud-client
IntegrationPropertiesEnvironmentPostProcessor	(org.springframework.boot.autoconfigure.integration)	Gradle: org.springframework.boot:spring-boot-autoconfigure-integration
RandomValuePropertySourceEnvironmentPostProcessor	(org.springframework.boot.env)	Gradle: org.springframework.boot:spring-boot-environment
SpringApplicationJsonEnvironmentPostProcessor	(org.springframework.boot.env)	Gradle: org.springframework.boot:spring-boot-environment
SpringBootTestRandomPortEnvironmentPostProcessor	(org.springframework.boot.test.web)	Gradle: org.springframework.boot:spring-boot-test-web
SystemEnvironmentPropertySourceEnvironmentPostProcessor	(org.springframework.boot.env)	Gradle: org.springframework.boot:spring-boot-environment

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

```

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

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

    boolean kubernetesEnabled = environment.getProperty("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();
        }
    }
}
```