04-Feature-Health & Info indicator

Monday, 3 April 2023 7:51 PM

01-基础使用之SpringBoot actuator

这个参考：https://www.jianshu.com/p/14a14b2d011c

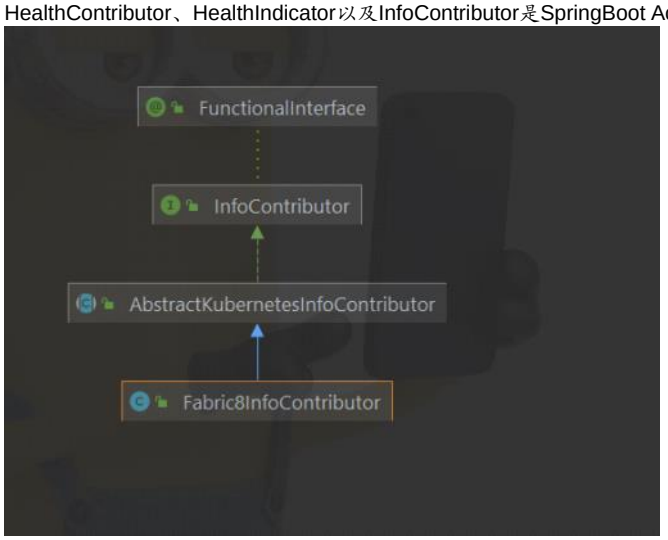
02-Kubernetes是如何利用HealthContributor来实现将pod信息加入到actuator的health端点呢？

首先我们是强依赖于SpringBoot Actuator模块的。因为其实不管是这个health还是info都是actuator来搭台，各个技术栈、技术点的实现来唱戏。大家把自己的一些信息整合收集好后交给actuator的端点来以供查询、展现。所以我们首先就是看actuator有没有，没这个平台，谁都唱不了戏！

```
Author: wind57
@Configuration(proxyBeanMethods = false)
@ConditionalOnClass(HealthIndicator.class)
@ConditionalOnKubernetesEnabled
public class Fabric8ActuatorConfiguration {

    @Bean
    @ConditionalOnEnabledHealthIndicator("kubernetes")
    public Fabric8HealthIndicator kubernetesHealthIndicator(PodUtils<Pod> podUtils) {
        return new Fabric8HealthIndicator(podUtils);
    }

    @Bean
    @ConditionalOnEnabledInfoContributor("kubernetes")
    public Fabric8InfoContributor kubernetesInfoContributor(PodUtils<Pod> podUtils) {
        return new Fabric8InfoContributor(podUtils);
    }

}
```

在满足条件后，由Fabric8ActuatorConfiguration把对应的实现注入Spring容器。也就是交由actuator来管理。这里有两点需要提一下：

1. HealthContributor、HealthIndicator以及InfoContributor是SpringBoot Actuator提供的接口抽象，各个技术点都是基于此实现。



2. Fabric8InfoContributor、Fabric8HealthIndicator都是基于Fabric8PodUtils来实现具体的原始数据获取。由Fabric8PodUtils来和kubernetes集群交互获取当前pod信息并组装好返回。

```
public class Fabric8InfoContributor extends AbstractKubernetesInfoContributor {

    private final PodUtils<Pod> utils;

    public Fabric8InfoContributor(PodUtils<Pod> utils) { this.utils = utils; }
```

```java
public class Fabric8InfoContributor extends AbstractKubernetesInfoContributor {

    private final PodUtils<Pod> utils;

    public Fabric8InfoContributor(PodUtils<Pod> utils) { this.utils = utils; }

    @Override
    public Map<String, Object> getDetails() {
        Pod current = this.utils.currentPod().get();

        if (current != null) {
            Map<String, Object> details = CollectionUtils.newHashMap( expectedSize: 7);
            details.put(INSIDE, true);

            ObjectMeta metadata = current.getMetadata();
            details.put(NAMESPACE, metadata.getNamespace());
            details.put(POD_NAME, metadata.getName());

            PodStatus status = current.getStatus();
            details.put(POD_IP, status.getPodIP());
            details.put(HOST_IP, status.getHostIP());

            PodSpec spec = current.getSpec();
            details.put(SERVICE_ACCOUNT, spec.getServiceAccountName());
            details.put(NODE_NAME, spec.getNodeName());

            return details;
        }
        return Collections.singletonMap(INSIDE, false);
    }
```

03-InfoContirbutor如何被Kubernetes利用起来的？

参考上面的health端点的使用。

04-思考上面actuator的实现与标记接口作用

refer to： https://www.baeldung.com/java-marker-interfaces

标记接口就是做特殊标记作用，里面没有任何实现内容。通常作为一个判断条件。目前我见到的用法有反射获取这个类是否存在，或者用instanceof关键字作为判断条件。

经典的一个实现就是spring-cloud-starter-bootstrap里面这个Marker接口，用于标记bootstrap是否可用。这里的构造方法是私有的！！！

```java
package org.springframework.cloud.bootstrap.marker;

/**
 * A marker class, so that, if present, spring cloud bootstrap is enable similar to how
 * 'spring.cloud.bootstrap.enabled=true' works.
 */
public abstract class Marker {

    private Marker() {
    }

}
```

05-思考一下这个注解的工作原理：@ConditionalOnEnabledHealthIndicator("kubernetes")

- 背景知识介绍：

  这个注解都是一个模子。直接在我们的类上根据条件使用@ConditionalOnXxx(…)。

  其实这个注解也是一个条件注解，利用了条件注解@Conditional的继承性。

  然后在这个@Conditional(…)里面添加我们自定义实现了的条件类【直接或者间接实现Condition接口】，多个的话就是&关系。在具体的方法match里面实现匹配逻辑。

  顺便提一下，感觉很常用的就是SpringBootCondition这个抽象类，实现了Condition接口，大部分直接继承它来用。同样的条件匹配结果Spring也抽象成了ConditionOutcome。

- 上菜：

  ConfigurationClassParser中有个属性configurationClasses，有个方法就是解析这些个配置类processConfigurationClass，这便是入口。

  通常是一套根据配置的信息，转换成系统属性，然后在做匹配判断的时候来获取，从而得出结果。主要方法是getMatchOutcome()。

```java
@Override
public ConditionOutcome getMatchOutcome(ConditionContext context, AnnotatedTypeMetadata metadata) {  context: ConditionEval
    AnnotationAttributes annotationAttributes = AnnotationAttributes  annotationAttributes:  size = 1
            .fromMap(metadata.getAnnotationAttributes(this.annotationType.getName()));  metadata: "org.springframework.boot
    String endpointName = annotationAttributes.getString( attributeName: "value");  endpointName: "diskspace"  annotationAttr
    ConditionOutcome outcome = getEndpointOutcome(context, endpointName);  endpointName: "diskspace"  context: Condition
    if (outcome != null) {
        return outcome;
    }
    return getDefaultOutcome(context, annotationAttributes);
}

protected ConditionOutcome getEndpointOutcome(ConditionContext context, String endpointName) {
    Environment environment = context.getEnvironment();
    String enabledProperty = this.prefix + endpointName + ".enabled";
    if (environment.containsProperty(enabledProperty)) {
        boolean match = environment.getProperty(enabledProperty, Boolean.class, defaultValue: true);
        return new ConditionOutcome(match, ConditionMessage.forCondition(this.annotationType)
                .because( reason: this.prefix + endpointName + ".enabled is " + match));
    }
    return null;
}
```

如果上面一套没有得到结果，那么再利用默认配置去尝试匹配一次。返回最终结果。

```java
protected ConditionOutcome getDefaultOutcome(ConditionContext context, AnnotationAttributes annotationAttributes) {
    return getDefaultEndpointsOutcome(context);
}
```

Return the default outcome that should be used.

Deprecated since 2.6.0 for removal in 3.0.0 in favor of `getDefaultOutcome (ConditionContext, AnnotationAttributes)`

Params: context – the condition context

Returns: the default outcome

```java
@Deprecated
protected ConditionOutcome getDefaultEndpointsOutcome(ConditionContext context) {
    boolean match = Boolean
            .parseBoolean(context.getEnvironment().getProperty( key: this.prefix + "defaults.enabled",  defaultValue: "true"));
    return new ConditionOutcome(match, ConditionMessage.forCondition(this.annotationType)
            .because( reason: this.prefix + "defaults.enabled is considered " + match));
}

}
```

05-基于此拓展，我们如何自定义端点？如何添加端点信息内容呢？

refer to: https://blog.51cto.com/u_12633149/3700485

```java
@Configuration
@WebEndpoint(id = "selfMonitor")
public class SelfMonitorEndPointConfig {

    @ReadOperation
    public Map<String, Object> getSelfMonitorInfo(PodUtils utils) {
        Object pod = utils.currentPod().get();
        Boolean insideKubernetes = utils.isInsideKubernetes();
        int cores = Runtime.getRuntime().availableProcessors();
        long freeMemory = Runtime.getRuntime().freeMemory();
        return new HashMap<>() {{
            put("Core Number", cores);
            put("Free Memory", freeMemory);
            put("Inside K8S", insideKubernetes);
            put("Pod", pod);
        }};
    }
}
```

给已有端点Health添加信息内容：

```java
@Component("SEHII")
public class SystemExtraHealthInfoIndicator extends AbstractHealthIndicator {
    private static final boolean DEFAULT_HEALTH_CHECK_RESULT = true;

    @Override
    protected void doHealthCheck(Health.Builder builder) throws Exception {
        LinkedHashMap<String, Object> map = new LinkedHashMap<>();
        if (!isNullOrEmpty(System.getenv(SYSTEM_ROOT)) && System.getenv(SYSTEM_ROOT).contains(WINDOWS_PLATFORM_KEY_WORD) && DEFAULT_HEALTH_CHE
            builder.up();
            map.put("System Author", "xlys");
            map.put("System Created At", "2023");
            map.put("System Running OS", WINDOWS_PLATFORM_KEY_WORD);
        }
        builder.withDetails(map);
    }

    @Override
    public Health getHealth(boolean includeDetails) {
        return super.getHealth(includeDetails);
    }
}
```