

08、指标监控

1、SpringBoot Actuator

1、简介

未来每一个微服务在云上部署以后，我们都需要对其进行监控、追踪、审计、控制等。SpringBoot就抽取了Actuator场景，使得我们每个微服务快速引用即可获得生产级别的应用监控、审计等功能。

XML | 复制代码

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-actuator</artifactId>
4 </dependency>
```

▼ org.springframework.boot:spring-boot-starter-actuator:2.4.0
 org.springframework.boot:spring-boot-starter:2.4.0 (omitted for duplicate)
 > org.springframework.boot:spring-boot-actuator-autoconfigure:2.4.0
 > io.micrometer:micrometer-core:1.6.1

atguigu.com 尚硅谷

2、1.x与2.x的不同

Spring Boot Actuator 1.x

- 支持SpringMVC
- 基于继承方式进行扩展
- 层级Metrics配置
- 自定义Metrics收集
- 默认较少的安全策略

Spring Boot Actuator 2.x

- 支持SpringMVC、JAX-RS以及Webflux
- 注解驱动进行扩展
- 层级&名称空间Metrics
- 底层使用MicroMeter，强大、便捷
- 默认丰富的安全策略

3、如何使用

- 引入场景
- 访问 <http://localhost:8080/actuator/> <http://localhost:8080/actuator/> **
- 暴露所有监控信息为HTTP

YAML | 复制代码

```
1 management:
2   endpoints:
3     enabled-by-default: true #暴露所有端点信息
4     web:
5       exposure:
6         include: '*' #以web方式暴露
```

- 测试

<http://localhost:8080/actuator/beans> <http://localhost:8080/actuator/beans>
<http://localhost:8080/actuator/configprops> <http://localhost:8080/actuator/configprops>
<http://localhost:8080/actuator/metrics> <http://localhost:8080/actuator/metrics>
<http://localhost:8080/actuator/metrics/jvm.gc.pause>
<http://localhost:8080/actuator/metrics/jvm.gc.pause>
<http://localhost:8080/actuator/> <http://localhost:8080/actuator/metrics>
endpointName/detailPath
.....

4、可视化

<https://github.com/codecentric/spring-boot-admin> <https://github.com/codecentric/spring-boot-admin>

2、Actuator Endpoint

1、最常使用的端点

ID	描述
<code>auditevents</code>	暴露当前应用程序的审核事件信息。需要一个 <code>AuditEventRepository</code> 组件。
<code>beans</code>	显示应用程序中所有 Spring Bean 的完整列表。
<code>caches</code>	暴露可用的缓存。
<code>conditions</code>	显示自动配置的所有条件信息，包括匹配或不匹配的原因。
<code>configprops</code>	显示所有 <code>@ConfigurationProperties</code> 。
<code>env</code>	暴露 Spring 的属性 <code>ConfigurableEnvironment</code>
<code>flyway</code>	显示已应用的所有 Flyway 数据库迁移。 需要一个或多个 <code>Flyway</code> 组件。
<code>health</code>	显示应用程序运行状况信息。
<code>httptrace</code>	显示 HTTP 跟踪信息（默认情况下，最近 100 个 HTTP 请求） 需要一个 <code>HttpTraceRepository</code> 组件。
<code>info</code>	显示应用程序信息。
<code>integrationgraph</code>	显示 Spring <code>IntegrationGraph</code> 。需要依赖 <code>spring-integration-core</code> 。
<code>loggers</code>	显示和修改应用程序中日志的配置。
<code>liquibase</code>	显示已应用的所有 Liquibase 数据库迁移。需要一个或多个 <code>Liquibase</code> 组件。
<code>metrics</code>	显示当前应用程序的“指标”信息。
<code>mappings</code>	显示所有 <code>@RequestMapping</code> 路径列表。
<code>scheduledtasks</code>	显示应用程序中的计划任务。
<code>sessions</code>	允许从 Spring Session 支持的会话存储中检索和删除用户会话。 使用 Spring Session 的基于 Servlet 的 Web 应用程序。
<code>shutdown</code>	使应用程序正常关闭。默认禁用。
<code>startup</code>	显示由 <code>ApplicationStartup</code> 收集的启动步骤数据。需要 <code>SpringApplication</code> 进行配置 <code>BufferingApplicationStartup</code> 。
<code>threaddump</code>	执行线程转储。

如果您的应用程序是Web应用程序（Spring MVC, Spring WebFlux或Jersey），则可以使用以下附加端点：

ID	描述
heapdump	返回 hprof 堆转储文件。
jolokia	通过HTTP暴露JMX bean（需要引入Jolokia，不适用于WebFlux）。需要引入依赖 jolokia-core。
logfile	返回日志文件的内容（如果已设置 logging.file.name 或 logging.file.path 属性）。支持使用HTTP Range 标头来部分日志文件的内容。
prometheus	以Prometheus服务器可以抓取的格式公开指标。需要依赖 micrometer-registry-prometheus。

最常用的Endpoint

- Health: 监控状况
- Metrics: 运行时指标
- Loggers: 日志记录

2、Health Endpoint

健康检查端点，我们一般用于在云平台，平台会定时的检查应用的健康状况，我们就需要Health Endpoint可以为平台返回当前应用的一系列组件健康状况的集合。

重要的几点：

- health endpoint返回的结果，应该是一系列健康检查后的一个汇总报告
- 很多的健康检查默认已经自动配置好了，比如：数据库、redis等
- 可以很容易的添加自定义的健康检查机制

```
保存 复制 全部折叠 全部展开 | ⌂ 过滤
status: "UP"
▼ components:
  ▶ db: ...
  ▶ diskSpace: ...
  ▶ ping: {atguigu.com 尚硅谷}
```

3、Metrics Endpoint

提供详细的、层级的、空间指标信息，这些信息可以被pull（主动推送）或者push（被动获取）方式得到；

- 通过Metrics对接多种监控系统
- 简化核心Metrics开发
- 添加自定义Metrics或者扩展已有Metrics

```

▼ names:
  0: "http.server.requests"
  1: "jvm.buffer.count"
  2: "jvm.buffer.memory.used"
  3: "jvm.buffer.total.capacity"
  4: "jvm.classes.loaded"
  5: "jvm.classes.unloaded"
  6: "jvm.gc.live.data.size"
  7: "jvm.gc.max.data.size"
  8: "jvm.gc.memory.allocated"
  9: "jvm.gc.memory.promoted"
  10: "jvm.gc.pause"
  11: "jvm.memory.committed"
  12: "jvm.memory.max"
  13: "jvm.memory.used"
  14: "jvm.threads.daemon"
  15: "jvm.threads.live"
  16: "jvm.threads.peak"
  17: "jvm.threads.states"
  18: "logback.events"
  19: "process.cpu.usage"
  20: "process.start.time"
  21: "process.uptime"
  22: "system.cpu.count"
  23: "system.cpu.usage"

```

4、管理Endpoints

1、开启与禁用Endpoints

- 默认所有的Endpoint除过shutdown都是开启的。
 - 需要开启或者禁用某个Endpoint。配置模式为 `management.endpoint.`
- `<endpointName>.enabled = true`

YAML | 复制代码

```

1 management:
2   endpoint:
3     beans:
4       enabled: true

```

- 或者禁用所有的Endpoint然后手动开启指定的Endpoint

[YAML](#) | [复制代码](#)

```
1 management:
2   endpoints:
3     enabled-by-default: false
4   endpoint:
5     beans:
6       enabled: true
7     health:
8       enabled: true
```

2、暴露Endpoints

支持的暴露方式

- HTTP: 默认只暴露**health**和**info** Endpoint
- JMX: 默认暴露所有Endpoint
- 除过**health**和**info**, 剩下的Endpoint都应该进行保护访问。如果引入**SpringSecurity**, 则会默认配置安全访问规则

ID	JMX	Web
auditevents	Yes	No
beans	Yes	No
caches	Yes	No
conditions	Yes	No
configprops	Yes	No
env	Yes	No
flyway	Yes	No
health	Yes	Yes
heapdump	N/A	No
httptrace	Yes	No
info	Yes	Yes
integrationgraph	Yes	No
jolokia	N/A	No
logfile	N/A	No
loggers	Yes	No
liquibase	Yes	No
metrics	Yes	No
mappings	Yes	No
prometheus	N/A	No
scheduledtasks	Yes	No
sessions	Yes	No
shutdown	Yes	No
startup	Yes	No
threaddump	Yes	No

3、定制 Endpoint

1、定制 Health 信息

Java | 复制代码

```
1 import org.springframework.boot.actuate.health.Health;
2 import org.springframework.boot.actuate.health.HealthIndicator;
3 import org.springframework.stereotype.Component;
4
5 @Component
6 public class MyHealthIndicator implements HealthIndicator {
7
8     @Override
9     public Health health() {
10         int errorCode = check(); // perform some specific health check
11         if (errorCode != 0) {
12             return Health.down().withDetail("Error Code", errorCode).build();
13         }
14         return Health.up().build();
15     }
16
17 }
18
19 构建Health
20 Health build = Health.down()
21     .withDetail("msg", "error service")
22     .withDetail("code", "500")
23     .withException(new RuntimeException())
24     .build();
```

YAML | 复制代码

```
1 management:
2   health:
3     enabled: true
4     show-details: always #总是显示详细信息。可显示每个模块的状态信息
```

Java | 复制代码

```

1  @Component
2  public class MyComHealthIndicator extends AbstractHealthIndicator {
3
4      /**
5       * 真实的检查方法
6       * @param builder
7       * @throws Exception
8       */
9      @Override
10     protected void doHealthCheck(Health.Builder builder) throws Exception
11         //mongodb。 获取连接进行测试
12         Map<String, Object> map = new HashMap<>();
13         // 检查完成
14         if(1 == 2){
15             //           builder.up(); //健康
16             builder.status(Status.UP);
17             map.put("count", 1);
18             map.put("ms", 100);
19         }else {
20             //           builder.down();
21             builder.status(Status.OUT_OF_SERVICE);
22             map.put("err", "连接超时");
23             map.put("ms", 3000);
24         }
25
26
27         builder.withDetail("code", 100)
28             .withDetails(map);
29
30     }
31 }
```

2、定制info信息

常用两种方式

1、编写配置文件

YAML | 复制代码

```

1  info:
2      appName: boot-admin
3      version: 2.0.1
4      mavenProjectName: @project.artifactId@ #使用@@可以获取maven的pom文件值
5      mavenProjectVersion: @project.version@
```

2、编写InfoContributor

Java | 复制代码

```
1 import java.util.Collections;
2
3 import org.springframework.boot.actuate.info.Info;
4 import org.springframework.boot.actuate.info.InfoContributor;
5 import org.springframework.stereotype.Component;
6
7 @Component
8 public class ExampleInfoContributor implements InfoContributor {
9
10     @Override
11     public void contribute(Info.Builder builder) {
12         builder.withDetail("example",
13             Collections.singletonMap("key", "value"));
14     }
15
16 }
```

<http://localhost:8080/actuator/info> <<http://localhost:8080/actuator/info>> 会输出以上方式返回的所有info信息

3、定制Metrics信息

1、SpringBoot支持自动适配的Metrics

- JVM metrics, report utilization of:
 - Various memory and buffer pools
 - Statistics related to garbage collection
 - Threads utilization
 - Number of classes loaded/unloaded
- CPU metrics
- File descriptor metrics
- Kafka consumer and producer metrics
- Log4j2 metrics: record the number of events logged to Log4j2 at each level
- Logback metrics: record the number of events logged to Logback at each level
- Uptime metrics: report a gauge for uptime and a fixed gauge representing the application's absolute start time
- Tomcat metrics (`server.tomcat.mbeanregistry.enabled` must be set to `true` for all Tomcat metrics to be registered)
- Spring Integration <<https://docs.spring.io/spring-integration/docs/5.4.1/reference/html/system-management.html#micrometer>>

integration> metrics

2、增加定制Metrics

Java | 复制代码

```

1 class MyService{
2     Counter counter;
3     public MyService(MeterRegistry meterRegistry){
4         counter = meterRegistry.counter("myservice.method.running.counter");
5     }
6
7     public void hello() {
8         counter.increment();
9     }
10 }
11
12
13 //也可以使用下面的方式
14 @Bean
15 MeterBinder queueSize(Queue queue) {
16     return (registry) -> Gauge.builder("queueSize", queue::size).register(
17 }
```

4、定制Endpoint

Java | 复制代码

```

1 @Component
2 @Endpoint(id = "container")
3 public class DockerEndpoint {
4
5
6     @ReadOperation
7     public Map getDockerInfo(){
8         return Collections.singletonMap("info","docker started...");
9     }
10
11     @WriteOperation
12     private void restartDocker(){
13         System.out.println("docker restarted....");
14     }
15
16 }
```

场景：开发**ReadinessEndpoint**来管理程序是否就绪，或者**LivenessEndpoint**来管理程序是否存活；

当然，这个也可以直接使用 [<https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-kubernetes-probes>](https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-kubernetes-probes)

更多内容参照：大厂学院