springboot(十九)：使用Spring Boot Actuator监控应用

 2018/02/06

微服务的特点决定了功能模块的部署是分布式的，大部分功能模块都是运行在不同的机器上，彼此通过服务调用进行交互，前后台的业务流会经过很多个微服务的处理和传递，出现了异常如何快速定位是哪个环节出现了问题？

在这种框架下，微服务的监控显得尤为重要。本文主要结合Spring Boot Actuator，跟大家一起分享微服务Spring Boot Actuator的常见用法，方便我们在日常中对我们的微服务进行监控治理。

**Actuator监控**

Spring Boot使用“习惯优于配置的理念”，采用包扫描和自动化配置的机制来加载依赖jar中的Spring bean,不需要任何Xml配置，就可以实现Spring的所有配置。虽然这样做能让我们的代码变得非常简洁，但是整个应用的实例创建和依赖关系等信息都被离散到了各个配置类的注解上，这使得我们分析整个应用中资源和实例的各种关系变得非常的困难。

Actuator是Spring Boot提供的对应用系统的自省和监控的集成功能，可以查看应用配置的详细信息，例如自动化配置信息、创建的Spring beans以及一些环境属性等。

Actuator监控只需要添加以下依赖就可以完成

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

</dependencies>

为了保证actuator暴露的监控接口的安全性，需要添加安全控制的依赖spring-boot-start-security依赖，访问应用监控端点时，都需要输入验证信息。Security依赖，可以选择不加，不进行安全管理，但不建议这么做。

**Actuator 的 REST 接口**

Actuator监控分成两类：原生端点和用户自定义端点；自定义端点主要是指扩展性，用户可以根据自己的实际应用，定义一些比较关心的指标，在运行期进行监控。

原生端点是在应用程序里提供众多 Web 接口，通过它们了解应用程序运行时的内部状况。原生端点又可以分成三类：

* 应用配置类：可以查看应用在运行期的静态信息：例如自动配置信息、加载的springbean信息、yml文件配置信息、环境信息、请求映射信息；
* 度量指标类：主要是运行期的动态信息，例如堆栈、请求连、一些健康指标、metrics信息等；
* 操作控制类：主要是指shutdown,用户可以发送一个请求将应用的监控功能关闭。

Actuator 提供了 13 个接口，具体如下表所示。

| **HTTP 方法** | **路径** | **描述** |
| --- | --- | --- |
| GET | /autoconfig | 提供了一份自动配置报告，记录哪些自动配置条件通过了，哪些没通过 |
| GET | /configprops | 描述配置属性(包含默认值)如何注入Bean |
| GET | /beans | 描述应用程序上下文里全部的Bean，以及它们的关系 |
| GET | /dump | 获取线程活动的快照 |
| GET | /env | 获取全部环境属性 |
| GET | /env/{name} | 根据名称获取特定的环境属性值 |
| GET | /health | 报告应用程序的健康指标，这些值由HealthIndicator的实现类提供 |
| GET | /info | 获取应用程序的定制信息，这些信息由info打头的属性提供 |
| GET | /mappings | 描述全部的URI路径，以及它们和控制器(包含Actuator端点)的映射关系 |
| GET | /metrics | 报告各种应用程序度量信息，比如内存用量和HTTP请求计数 |
| GET | /metrics/{name} | 报告指定名称的应用程序度量值 |
| POST | /shutdown | 关闭应用程序，要求endpoints.shutdown.enabled设置为true |
| GET | /trace | 提供基本的HTTP请求跟踪信息(时间戳、HTTP头等) |

**快速上手**

**相关配置**

**项目依赖**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

</dependencies>

**配置文件**

server:

port: 8080

management:

security:

enabled: false #关掉安全认证

port: 8088 #管理端口调整成8088

context-path: /monitor #actuator的访问路径

endpoints:

shutdown:

enabled: true

info:

app:

name: spring-boot-actuator

version: 1.0.0

* management.security.enabled=false默认有一部分信息需要安全验证之后才可以查看，如果去掉这些安全认证，直接设置management.security.enabled=false
* management.context-path=/monitor 代表启用单独的url地址来监控Spring Boot应用，为了安全一般都启用独立的端口来访问后端的监控信息
* endpoints.shutdown.enabled=true 启用接口关闭Spring Boot

配置完成之后，启动项目就可以继续验证各个监控功能了。

**命令详解**

**autoconfig**

Spring Boot的自动配置功能非常便利，但有时候也意味着出问题比较难找出具体的原因。使用 autoconfig 可以在应用运行时查看代码了某个配置在什么条件下生效，或者某个自动配置为什么没有生效。

启动示例项目，访问：http://localhost:8088/monitor/autoconfig返回部分信息如下：

{

"positiveMatches": {

"DevToolsDataSourceAutoConfiguration": {

"notMatched": [

{

"condition": "DevToolsDataSourceAutoConfiguration.DevToolsDataSourceCondition",

"message": "DevTools DataSource Condition did not find a single DataSource bean"

}

],

"matched": [ ]

},

"RemoteDevToolsAutoConfiguration": {

"notMatched": [

{

"condition": "OnPropertyCondition",

"message": "@ConditionalOnProperty (spring.devtools.remote.secret) did not find property 'secret'"

}

],

"matched": [

{

"condition": "OnClassCondition",

"message": "@ConditionalOnClass found required classes 'javax.servlet.Filter', 'org.springframework.http.server.ServerHttpRequest'; @ConditionalOnMissingClass did not find unwanted class"

}

]

}

}

}

**configprops**

查看配置文件中设置的属性内容，以及一些配置属性的默认值。

启动示例项目，访问：http://localhost:8088/monitor/configprops返回部分信息如下：

{

...

"environmentEndpoint": {

"prefix": "endpoints.env",

"properties": {

"id": "env",

"sensitive": **true**,

"enabled": **true**

}

},

"spring.http.multipart-org.springframework.boot.autoconfigure.web.MultipartProperties": {

"prefix": "spring.http.multipart",

"properties": {

"maxRequestSize": "10MB",

"fileSizeThreshold": "0",

"location": **null**,

"maxFileSize": "1MB",

"enabled": **true**,

"resolveLazily": **false**

}

},

"infoEndpoint": {

"prefix": "endpoints.info",

"properties": {

"id": "info",

"sensitive": **false**,

"enabled": **true**

}

}

...

}

**beans**

根据示例就可以看出，展示了bean的别名、类型、是否单例、类的地址、依赖等信息。

启动示例项目，访问：http://localhost:8088/monitor/beans返回部分信息如下：

[

{

"context": "application:8080:management",

"parent": "application:8080",

"beans": [

{

"bean": "embeddedServletContainerFactory",

"aliases": [

],

"scope": "singleton",

"type": "org.springframework.boot.context.embedded.tomcat.TomcatEmbeddedServletContainerFactory",

"resource": "null",

"dependencies": [

]

},

{

"bean": "endpointWebMvcChildContextConfiguration",

"aliases": [

],

"scope": "singleton",

"type": "org.springframework.boot.actuate.autoconfigure.EndpointWebMvcChildContextConfiguration$$EnhancerBySpringCGLIB$$a4a10f9d",

"resource": "null",

"dependencies": [

]

}

}

]

**dump**

/dump 接口会生成当前线程活动的快照。这个功能非常好，方便我们在日常定位问题的时候查看线程的情况。 主要展示了线程名、线程ID、线程的状态、是否等待锁资源等信息。

启动示例项目，访问：http://localhost:8088/monitor/dump返回部分信息如下：

[

{

"threadName": "http-nio-8088-exec-6",

"threadId": 49,

"blockedTime": -1,

"blockedCount": 0,

"waitedTime": -1,

"waitedCount": 2,

"lockName": "java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject@1630a501",

"lockOwnerId": -1,

"lockOwnerName": **null**,

"inNative": **false**,

"suspended": **false**,

"threadState": "WAITING",

"stackTrace": [

{

"methodName": "park",

"fileName": "Unsafe.java",

"lineNumber": -2,

"className": "sun.misc.Unsafe",

"nativeMethod": **true**

},

{

"methodName": "park",

"fileName": "LockSupport.java",

"lineNumber": 175,

"className": "java.util.concurrent.locks.LockSupport",

"nativeMethod": **false**

},

{

"methodName": "await",

"fileName": "AbstractQueuedSynchronizer.java",

"lineNumber": 2039,

"className": "java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject",

"nativeMethod": **false**

},

...

{

"methodName": "getTask",

"fileName": "ThreadPoolExecutor.java",

"lineNumber": 1067,

"className": "java.util.concurrent.ThreadPoolExecutor",

"nativeMethod": **false**

},

{

"methodName": "runWorker",

"fileName": "ThreadPoolExecutor.java",

"lineNumber": 1127,

"className": "java.util.concurrent.ThreadPoolExecutor",

"nativeMethod": **false**

},

{

"methodName": "run",

"fileName": "ThreadPoolExecutor.java",

"lineNumber": 617,

"className": "java.util.concurrent.ThreadPoolExecutor$Worker",

"nativeMethod": **false**

},

{

"methodName": "run",

"fileName": "TaskThread.java",

"lineNumber": 61,

"className": "org.apache.tomcat.util.threads.TaskThread$WrappingRunnable",

"nativeMethod": **false**

},

{

"methodName": "run",

"fileName": "Thread.java",

"lineNumber": 745,

"className": "java.lang.Thread",

"nativeMethod": **false**

}

],

"lockedMonitors": [

],

"lockedSynchronizers": [

],

"lockInfo": {

"className": "java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject",

"identityHashCode": 372286721

}

}

...

]

**env**

展示了系统环境变量的配置信息，包括使用的环境变量、JVM 属性、命令行参数、项目使用的jar包等信息。和configprops不同的是，configprops关注于配置信息，env关注运行环境信息。

启动示例项目，访问：http://localhost:8088/monitor/env返回部分信息如下：

{

"profiles": [

],

"server.ports": {

"local.management.port": 8088,

"local.server.port": 8080

},

"servletContextInitParams": {

},

"systemProperties": {

"com.sun.management.jmxremote.authenticate": "false",

"java.runtime.name": "Java(TM) SE Runtime Environment",

"spring.output.ansi.enabled": "always",

"sun.boot.library.path": "C:\\Program Files\\Java\\jdk1.8.0\_101\\jre\\bin",

"java.vm.version": "25.101-b13",

"java.vm.vendor": "Oracle Corporation",

"java.vendor.url": "http://java.oracle.com/",

"java.rmi.server.randomIDs": "true",

"path.separator": ";",

"java.vm.name": "Java HotSpot(TM) 64-Bit Server VM",

"file.encoding.pkg": "sun.io",

"user.country": "CN",

"user.script": "",

"sun.java.launcher": "SUN\_STANDARD",

"sun.os.patch.level": "",

"PID": "5268",

"com.sun.management.jmxremote.port": "60093",

"java.vm.specification.name": "Java Virtual Machine Spe

为了避免敏感信息暴露到 /env 里，所有名为password、secret、key(或者名字中最后一段是这些)的属性在 /env 里都会加上“\*”。举个例子，如果有一个属性名字是database.password，那么它在/env中的显示效果是这样的：

"database.password":"\*\*\*\*\*\*"

**/env/{name}**用法

就是env的扩展 可以获取指定配置信息，比如：http://localhost:8088/monitor/env/java.vm.version,返回：{"java.vm.version":"25.101-b13"}

**health**

可以看到 HealthEndPoint 给我们提供默认的监控结果，包含 磁盘检测和数据库检测

启动示例项目，访问：http://localhost:8088/monitor/health返回部分信息，下面的JSON响应是由状态、磁盘空间和db。描述了应用程序的整体健康状态,UP 表明应用程序是健康的。磁盘空间描述总磁盘空间,剩余的磁盘空间和最小阈值。application.properties阈值是可配置的

{

"status": "UP",

"diskSpace": {

"status": "UP",

"total": 209715195904,

"free": 183253909504,

"threshold": 10485760

}

"db": {

"status": "UP",

"database": "MySQL",

"hello": 1

}

}

其实看 Spring Boot-actuator 源码，你会发现 HealthEndPoint 提供的信息不仅限于此，org.springframework.boot.actuate.health 包下 你会发现 ElasticsearchHealthIndicator、RedisHealthIndicator、RabbitHealthIndicator 等

**info**

info就是我们自己配置在配置文件中以Info开头的配置信息，比如我们在示例项目中的配置是：

info:

app:

name: spring-boot-actuator

version: 1.0.0

启动示例项目，访问：http://localhost:8088/monitor/info返回部分信息如下：

{

"app": {

"name": "spring-boot-actuator",

"version": "1.0.0"

}

}

**mappings**

描述全部的URI路径，以及它们和控制器的映射关系

启动示例项目，访问：http://localhost:8088/monitor/mappings返回部分信息如下：

{

"/\*\*/favicon.ico": {

"bean": "faviconHandlerMapping"

},

"{[/hello]}": {

"bean": "requestMappingHandlerMapping",

"method": "public java.lang.String com.neo.controller.HelloController.index()"

},

"{[/error]}": {

"bean": "requestMappingHandlerMapping",

"method": "public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)"

}

}

**metrics**

最重要的监控内容之一，主要监控了JVM内容使用、GC情况、类加载信息等。

启动示例项目，访问：http://localhost:8088/monitor/metrics返回部分信息如下：

{

"mem": 337132,

"mem.free": 183380,

"processors": 4,

"instance.uptime": 254552,

"uptime": 259702,

"systemload.average": -1.0,

"heap.committed": 292864,

"heap.init": 129024,

"heap.used": 109483,

"heap": 1827840,

"nonheap.committed": 45248,

"nonheap.init": 2496,

"nonheap.used": 44269,

"nonheap": 0,

"threads.peak": 63,

"threads.daemon": 43,

"threads.totalStarted": 83,

"threads": 46,

"classes": 6357,

"classes.loaded": 6357,

"classes.unloaded": 0,

"gc.ps\_scavenge.count": 8,

"gc.ps\_scavenge.time": 99,

"gc.ps\_marksweep.count": 1,

"gc.ps\_marksweep.time": 43,

"httpsessions.max": -1,

"httpsessions.active": 0

}

对 /metrics接口提供的信息进行简单分类如下表：

| **分类** | **前缀** | **报告内容** |
| --- | --- | --- |
| 垃圾收集器 | gc.\* | 已经发生过的垃圾收集次数，以及垃圾收集所耗费的时间，适用于标记-清理垃圾收集器和并行垃圾收集器(数据源自java.lang.management. GarbageCollectorMXBean) |
| 内存 | mem.\* | 分配给应用程序的内存数量和空闲的内存数量(数据源自java.lang. Runtime) |
| 堆 | heap.\* | 当前内存用量(数据源自java.lang.management.MemoryUsage) |
| 类加载器 | classes.\* | JVM类加载器加载与卸载的类的数量(数据源自java.lang. management.ClassLoadingMXBean) |
| 系统 | processors、instance.uptime、uptime、systemload.average | 系统信息，例如处理器数量(数据源自java.lang.Runtime)、运行时间(数据源自java.lang.management.RuntimeMXBean)、平均负载(数据源自java.lang.management.OperatingSystemMXBean) |
| 线程池 | thread.\* | 线程、守护线程的数量，以及JVM启动后的线程数量峰值(数据源自 java.lang .management.ThreadMXBean) |
| 数据源 | datasource.\* | 数据源连接的数量(源自数据源的元数据，仅当Spring应用程序上下文里存在 DataSource Bean 的时候才会有这个信息) |
| Tomcat 会话 | httpsessions.\* | Tomcat的活跃会话数和最大会话数(数据源自嵌入式Tomcat的Bean，仅在使用嵌入式Tomcat服务器运行应用程序时才有这个信息) |
| HTTP | counter.status.*、gauge.response.* | 多种应用程序服务HTTP请求的度量值与计数器 |

**解释说明：**

* 请注意，这里的一些度量值，比如数据源和Tomcat会话，仅在应用程序中运行特定组件时才有数据。你还可以注册自己的度量信息。
* HTTP的计数器和度量值需要做一点说明。counter.status 后的值是HTTP状态码，随后是所请求的路径。举个例子，counter.status.200.metrics 表明/metrics端点返回 200(OK) 状态码的次数。
* HTTP的度量信息在结构上也差不多，却在报告另一类信息。它们全部以gauge.response 开头,，表明这是HTTP响应的度量信息。前缀后是对应的路径。度量值是以毫秒为单位的时间，反映了最近处理该路径请求的耗时。
* 这里还有几个特殊的值需要注意。root路径指向的是根路径或/。star-star代表了那些Spring 认为是静态资源的路径，包括图片、JavaScript和样式表，其中还包含了那些找不到的资源。这就是为什么你经常会看到 counter.status.404.star-star，这是返回了HTTP 404 (NOT FOUND) 状态的请求数。
* /metrics接口会返回所有的可用度量值，但你也可能只对某个值感兴趣。要获取单个值，请求时可以在URL后加上对应的键名。例如，要查看空闲内存大小,可以向/metrics/mem.free发一 个GET请求。例如访问：http://localhost:8088/monitor/metrics/mem.free，返回：{"mem.free":178123}。

**shutdown**

开启接口优雅关闭Spring Boot应用，要使用这个功能首先需要在配置文件中开启：

endpoints:

shutdown:

enabled: true

配置完成之后，启动示例项目，访问：http://localhost:8088/monitor/shutdown返回部分信息如下：

{

"message": "Shutting down, bye..."

}

此时你会发现应用已经被关闭。

**trace**

/trace 接口能报告所有Web请求的详细信息，包括请求方法、路径、时间戳以及请求和响应的头信息，记录每一次请求的详细信息。

启动示例项目，先访问一次：http://localhost:8080/hello，再到浏览器执行：http://localhost:8088/monitor/trace查看返回信息：

[

{

"timestamp": 1516780334777,

"info": {

"method": "GET",

"path": "/hello",

"headers": {

"request": {

"host": "localhost:8080",

"connection": "keep-alive",

"cache-control": "max-age=0",

"user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/63.0.3239.84 Safari/537.36",

"upgrade-insecure-requests": "1",

"accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,\*/\*;q=0.8",

"accept-encoding": "gzip, deflate, br",

"accept-language": "zh-CN,zh;q=0.9",

"cookie": "UM\_distinctid=16053ba344f1cd-0dc220c44cc94-b7a103e-13c680-16053ba3450751; Hm\_lvt\_0fb30c642c5f6453f17d881f529a1141=1513076406,1514961720,1515649377; CNZZDATA1260945749=232252692-1513233181-%7C1516085149; Hm\_lvt\_6d8e8bb59814010152d98507a18ad229=1515247964,1515296008,1515672972,1516086283"

},

"response": {

"X-Application-Context": "application:8080",

"Content-Type": "text/html;charset=UTF-8",

"Content-Length": "11",

"Date": "Wed, 24 Jan 2018 07:52:14 GMT",

"status": "200"

}

},

"timeTaken": "4"

}

}

]

上述信息展示了，/hello请求的详细信息。

**其它配置**

**敏感信息访问限制**

根据上面表格，鉴权为false的，表示不敏感，可以随意访问，否则就是做了一些保护，不能随意访问。

endpoints.mappings.sensitive=false

这样需要对每一个都设置，比较麻烦。敏感方法默认是需要用户拥有ACTUATOR角色，因此，也可以设置关闭安全限制：

management.security.enabled=false

或者配合Spring Security做细粒度控制。

**启用和禁用接口**

虽然Actuator的接口都很有用，但你不一定需要全部这些接口。默认情况下，所有接口(除 了/shutdown)都启用。比如要禁用 /metrics 接口，则可以设置如下：

endpoints.metrics.enabled = false

如果你只想打开一两个接口，那就先禁用全部接口，然后启用那几个你要的，这样更方便。

endpoints.enabled = **false**

endpoints.metrics.enabled = **true**

[**示例代码-github**](https://github.com/ityouknow/spring-boot-examples)

[**示例代码-码云**](https://gitee.com/ityouknow/spring-boot-examples)