链滴

Springboot 之 Actuator 服务监控

作者：hjljy
原文链接：https://ld246.com/article/1591168434977
来源网站：链滴

最近在项目当中，启动了多个微服务项目，想着如何监控启动的项目，然后就找到了Actuator这个东！！！总结下自己学习到的知识点！！！

什么是Actuator

Spring Boot Actuator 官方文档

Spring Boot includes a number of additional features to help you monitor and manage your pplication when you push it to production. You can choose to manage and monitor your appl cation by using HTTP endpoints or with JMX. Auditing, health, and metrics gathering can also be automatically applied to your application.

翻译成中文就是：Spring Boot包含许多其他功能，可帮助您在将应用程序投入生产时监控和管理您应用程序。您可以选择使用HTTP端点或JMX管理和监视您的应用程序。审核，运行状况和指标收集可以自动应用于您的应用程序。

简单来说就是：Spring Boot Actuator可以实现对应用的监控以及管理！！！

项目当中如何使用

非常简单，只需要引入相关JAR即可！！！

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
  </dependency>
</dependencies>
```

然后启动项目，在浏览器当中输入: http://127.0.0.1/actuator/ 即可！！！

但是这个时候返回的信息是很少的，只有默认的几个数据！！！如下所示：

{"_links":{
"self":{"href":"http://127.0.0.1:8080/actuator","templated":false},
"health":{"href":"http://127.0.0.1:8080/actuator/health","templated":false},
"health-path":{"href":"http://127.0.0.1:8080/actuator/health/{*path}","templated":true},
"info":{"href":"http://127.0.0.1:8080/actuator/info","templated":false}
}}

然后在浏览器输入对应的地址就可以看到相关的详细信息了！！！

Actuator 端口

在上面actuator路径后面的health,info都可以称为端口，表示actuator针对springboot应用监控的一信息。根据官方文档默认情况下是只有这两个接口可以访问的！！！。但是可以自己进行配置要暴露端口！！！

actuator默认是有很多端口的，如下所示：

端口	描述
auditevents	公开当前应用程序的审核事件信息需要一个AuditEventRepository bean。
beans	显示应用程序中所有Spring Bean的完整表。
caches	公开可用的缓存。
conditions	显示在配置和自动配置类上评估的条以及它们匹配或不匹配的原因。
configprops	显示应用加载的所有的ConfigurationProperties配置信息。
env	显示应用当前运行环境
health	显示应用当前的运行状态
httptrace	显示HTTP跟踪信息（默认情况下，最100个HTTP请求-响应交换）。需要配置一个HttpTraceRepository bean。
info	显示应用程序基础信息
loggers	显示和修改应用程序中日志记录的配置。
metrics	显示当前应用程序的“指标”信息
scheduledtasks	显示应用程序中的计划任务。
mappings	显示所有@RequestMapping路径的理列表。
shutdown	使应用程序正常关闭。默认禁用。
threaddump	执行线程转储文件
heapdump	堆转储文件。
logfile	返回日志文件的内容（如果已设置logging.ile.name或logging.file.path属性）。支持使用HTTP Range标头来检索部分日志文件的内容。
.....	其它详见官方文档

开启和暴露端口

默认情况下上面的端口除了shutdown 以外，都是开启了的。但是由于端口信息里面包含了敏感信息所以默认是通过HTTP访问的话，只暴露了health,info两个基本端口。其它的端口都是没有暴露的。

在配置文件当中开启端口：management.endpoint.{端口名}.enabled=true　这样就开启了。false示不开启。如下所示！！！

```
management.endpoint.shutdown.enabled=true
management.endpoint.info.enabled=true
```

由于默认是开启shutdown 以外所有端口，如果需要只开启某写端口的话,可以快速关闭所有端口进行然后开启某个单独的端口

```
management.endpoints.enabled-by-default=false
management.endpoint.info.enabled=true
```

开启了端口之后，并不能够直接通过HTTP请求访问到，还需要暴露出来　include 表示暴露的端口，eclude表示排除的端口

```
# 暴露所有的端口，除了env,beans
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude=env,beans
```

注意事项：在yml当中 * 具有特殊意义，需要双引号包裹起来！！！

注意事项：端口信息会存在缓存，可以通过设置端口的cache.time-to-live属性来设置缓存

```
#将health端点的缓存的生存时间设置为10秒
management.endpoint.health.cache.time-to-live=10s
```

端口加密访问

由于端口返回的信息或多或少会有一些敏感的信息，所以要求严格的话，是需要对端口访问进行加密理！！！官方推荐是配合Spring Security很容易实现，个人觉得随便加上一个路由拦截器拦截一下就事了，毕竟测试环境没必要，生产环境早都已经层层加密了，估计端口都只会内部访问，本地访问等一系列的限制。

端口的操作基本都是查询，但是还是有一些是比较特别或者有意思的端口

health端口

默认情况下health端口返回的信息是很简单的，可以通过配置返回详细的信息

```
#向所有用户展示详细信息
management.endpoint.health.show-details=always
```

health端口主要返回应用的一些情况，比如:应用状态，数据库状态，磁盘空间信息等等，具体指标见方文档

health端口也支持自定义检查项：

```
/**
 * @author 海加尔金鹰
```

```
 */
@Component
public class MyHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        Boolean success = check();
        if (success) {
            return Health.down().withDetail("Error Code", "自定义健康检查失败").build();
        }
        return Health.up().withDetail("Error Code", "自定义健康检查成功").build();
    }

    private Boolean check() {
        //这里处理自定义的健康检查详情
        double random = Math.random();
        if(random*1000<500){
            System.out.println(random*1000);
            return true;
        }
        return false;
    }
}
```

这样在浏览器输入：http://127.0.0.1/actuator/health 就可以看到自定义检查项是否成功，并且每都会重新检查！！！

备注信息：返回的自定义检查项是在my里面，这是因为MyHealthIndicator 是my开头的原因。

info端口

info端口主要展示我们在配置文件当中自己配置的信息：例如在properties当中配置，在url当中就会回这些数据

```
info.name=hjljy
info.version=v1.0.0
info.des=this is a demo
```

shutdown端口

这个端口，个人觉得算是比较鸡肋的端口，比较主要作用是关闭应用，谁会去关闭线上的应用啊？就要发版也不会这么关闭

首先需要先开启端口，然后暴露出来

```
management.endpoint.shutdown.enabled=true
management.endpoints.web.exposure.include=*
```

接下来通过POST请求就可以关闭应用了。

```
POST http://localhost:8080/actuator/shutdown

HTTP/1.1 200
```

```
Content-Type: application/vnd.spring-boot.actuator.v3+json
Transfer-Encoding: chunked
Date: Mon, 01 Jun 2020 14:16:29 GMT
Keep-Alive: timeout=60
Connection: keep-alive

{
  "message": "Shutting down, bye..."
}
```

loggers端口

loggers端口可以显示和修改应用程序中日志记录的配置，如果线上环境只记录了error级别的错误，进行排查的时候可以将级别设置为info,debug级别等等！！！

也是通过POST请求进行的设置：（如果是get请求，表示查看日志级别）

```
POST http://localhost:8080/actuator/loggers/root
Content-Type: application/json

{
   "configuredLevel": "info"
}
```

httptrace端口

这个端口需要额外进行配置才能返回信息，如果没有配置是不会有这个端口的。端口主要作用是返回近的100个请求信息？不过没有记录请求和响应参数，只有请求地址，响应结果。只需要将下面的代放到配置类里面注入即可，或者直接放在springboot启动类里面

```
    @Bean
    public HttpTraceRepository httpTraceRepository() {
        return new InMemoryHttpTraceRepository();
    }
```

其它端口

其余端口基本都是查询，直接在浏览器输入地址，就可以看到返回的信息。非常的方便！！！，缺点返回的数据是JSON格式，需要自己处理下！！！

自定义属性配置

主要是针对Actuator默认的配置，是可以自定义进行处理设置的。

自定义管理路径

如果觉得Actuator默认的actuator路径不喜欢，可以通过配置将actuator换成其它的路径

```
management.endpoints.web.base-path=/show
```

这时所有的端口路径就从 actuator/{name} 变成了show/{name},例如：actuator/info---->show/in

o

如果觉得某个端口路径名字过长，不好记，也可以自定义某个端口的路径

management.endpoints.web.path-mapping.scheduledtasks=/task

这时scheduledtasks端口路径就从 actuator/scheduledtasks---->actuator/task
如果上面两个一起用，就变成了show/task

自定义管理port和限制

默认情况下,应用的port就是actuator的端口，但是也可以自己设置

```
# 这里就将actuator的端口修改为了8090，访问时的端口就变成了8090了。
management.server.port=8090
# 如果设置为-1就表示不暴露端口，所有的请求都会变成404
management.server.port=-1
# 如果设置为127.0.0.1就表示只能通过本地访问，无法通过IP访问了。
management.server.address=127.0.0.1
```

自定义管理端口

如果觉得上面的端口信息还不够充分，可以自定义需要获取，统计的信息端口。

```
/**
 * @author 海加尔金鹰
 */
@Endpoint(id = "hjljy") //指定端口为hjljy
@Component
public class HjljyEndpoint {

    private static String NAME = "这是测试数据";

    //表示通过GET请求获取
    @ReadOperation
    public String getName() {
        return NAME;
    }
    //表示通过POST请求获取
    @WriteOperation
    public String setName(String name) {
        NAME = name;
        return NAME;
    }
    //表示通过DETELE请求获取
    @DeleteOperation
    public String initName() {
        NAME = "删除数据";
        return NAME;
    }
}
```

测试结果如下：

1 发送GET请求：

GET http://localhost:8090/actuator/hjljy

返回数据：这是测试数据

2 发送POS请求：

```
POST http://localhost:8090/actuator/hjljy
Content-Type: application/json

{
  "name": "碎银几两"
}
```

返回数据：碎银几两

3 发送DETELE请求

DELETE http://localhost:8090/actuator/hjljy

返回数据：删除数据

总结

1 感觉很惊艳，当然也可能是因为我没怎么接触线上环境，或者没怎么关注运维技巧，反正目前觉得个东西对于开发来说非常友好吧，利于BUG的分析，线上的监控啊什么的。

2 就是全部返回JSON有点难受。不过据说有个Springboot Admin可以进行可视化分析。