# 应用监控简介

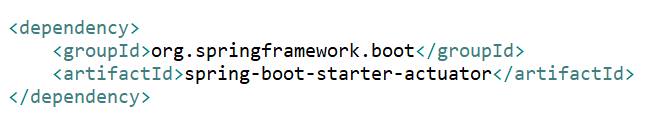
在一个Spring Boot项目运行时，开发者需要对Spring Boot项目进行实时监控，获取项目的运行情况，在项目出错时能够实现自动报警等。Spring Boot提供了actuator来帮助开发者获取应用程序的实时运行数据。开发者可以选择使用HTTP端点或JMX来管理和监控应用程序，获取应用程序的运行数据，包括健康状况、应用信息、内存使用情况等。

## 端点配置

## 引入端点

开发者可以使用执行器（actuator）中的端点（EndPoints）对应于进行监控或者与应用进行交互。

1. 加入spring-boot-starter-actuator依赖，如下：



加入执行器依赖后，Spring Boot会默认包含许多端点，如下：

|  |  |  |
| --- | --- | --- |
| **端点** | **描述** | **是否默认已开启** |
| auditevents | 展示当前应用程序的审计事件信息 | YES |
| beans | 展示所有Spring Beans信息 | YES |
| conditions | 展示一个自动配置类的使用报告，该报告展示所有自动配置类以及它们被使用未被使用的原因 | YES |
| configprops | 展示所有@ConfigurationProperties的列表 | YES |
| env | 展示系统运行环境信息 | YES |
| flyway | 展示数据库迁移路径 | YES |
| health | 展示应用程序的健康信息 | YES |
| httptrace | 展示trace信息（默认为最新的100条HTTP请求） | YES |
| info | 展示应用的定制信息，这些定制信息以info开头 | YES |
| loggers | 展示并修改应用的日志配置 | YES |
| liquibase | 展示任何Liquibase数据库迁移路径 | YES |
| metrics | 展示应用程序度量信息 | YES |
| mappings | 展示所有@RequestMapping路径的集合列表 | YES |
| scheduledtasks | 展示应用程序所有的定时任务 | YES |
| shutdown | 远程关闭应用接口 | NO |
| sessions | 展示并操作Spring Session会话 | YES |
| threaddump | 展示线程活动的快照 | YES |

如果应用程序是一个Web应用，则还有如下端点：

|  |  |  |
| --- | --- | --- |
| **端点** | **描述** | **默认是否已开启** |
| heapdump | 返回一个GZip压缩的hprof堆转储文件 | YES |
| jolokia | 展示通过HTTP暴露的JMX beans | YES |
| logfile | 返回日志文件内容 | YES |
| prometheus | 张氏一个可以被Prometheus服务器抓取的metrices数据 | YES |

## 开启端点

1. 引入端点后，大部分的端点都是默认开启的，只有shutdown端点未开启，如果需要开启，可以在application.properties中配置，如下：



1. 如果开发者不想开启这么多端点，那么可以关闭所有的端点，然后开启需要开启的端点，如下：



第3行表示关闭所有的端点；

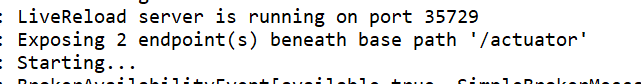
第4行表示开启info端点；

## 暴露端点

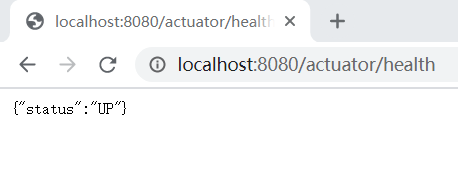
1. 一个端点只有暴露了才能够被访问。端点的默认暴露情况如下：

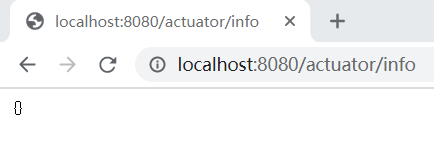
|  |  |  |
| --- | --- | --- |
| **端点** | **JMX（ Java管理扩展）** | **Web** |
| autitevents | YES | NO |
| beans | YES | NO |
| conditions | YES | NO |
| configprops | YES | NO |
| env | YES | NO |
| flyway | YES | NO |
| health | YES | YES |
| httptrace | YES | NO |
| info | YES | YES |
| loggers | YES | NO |
| liquibase | YES | NO |
| metrics | YES | NO |
| mappings | YES | NO |
| scheduledtasks | YES | NO |
| shutdown | YES | NO |
| sessions | YES | NO |
| threaddump | YES | NO |
| heapdump | N/A | NO |
| jolokia | N/A | NO |
| logfile | N/A | NO |
| prometheus | N/A | NO |

可以看出，在Web应用中，只有health和info端点是默认暴露的，因此启动项目后，默认只有这个端点可以访问，从打印台也可以观察出。如下：



此时，项目启动完成后，可以通过/autcuator/info和/autcuator/health访问端点查看应用信息，如下：





1. 开发者可以在application.properties自定义要暴露哪些端点，，例如要暴露mappings端点和metrics端点，配置如下：



3）如果要暴露全部端点，配置如下：



虽然暴露了所有的端点，但在项目启动时观察打印台，只有部分端点被暴露了，如下：



这是因为还有部分端点需要相关依赖才能使用，对于已暴露的端口，可以直接通过“/actuator/端点名”进行访问。

## 端点保护

暴露端点后，该端点就可以被任何人所访问，但是有的端点包含敏感信息，此时最好能将这些端点保护起来。可以使用Spring Security或Shiro来对端点进行保护，当访问端点路径时，必须输入用户和密码才可以访问。

## 端点缓存

对于一些不带参数的端点请求（访问端点时不需要传入参数）会自动进行缓存，开发者可以在application.properties配置文件配置缓存时间，如下：



表示beans端点的缓存时间为100秒；如果需要修改其他端点的缓存时间，则将beans修改为其他端点名称即可；

如果端点被保护了，则端点不会被缓存，所以也不需要设置缓冲时间；

## 端点路径映射

默认情况下，所有的端点都是暴露在 “/actuator”路径下的，以health端点为例，health端点的访问路径是“/actuator/health”，如果开发者需要对端点的访问路径进行映射，配置如下：



第一行表示将访问前缀“/actuator”修改为“/”；

第二行表示修改health端点的访问路径为“/healthcheck”；

经过以上配置后，health端点的访问路径被配置成了“/healthcheck”

## 端点CORS支持

所以的端点默认都没有都没有开启跨域，开发者可以通过如下配置快速开启CORS支持，进而实现跨域。



以上配置表示允许端点处理来自<http://localhost:8081>地址的请求，允许的请求方式为GET和POST。

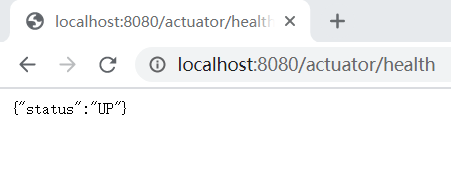
（关于CORS，可以参考第四章第六节）

# 常见端点

## health端点

### 健康信息

开发者可以通过health端点来获取应用的运行数据，进而确保应用程序的健康，提早解决，较少损失。但在默认情况下，health端点只能获取到status信息，如下：



UP（up）表示应用在线；

这是因为detail信息默认是不显示的，details属性取值如下：

（1）never：表示不显示details信息（默认值）

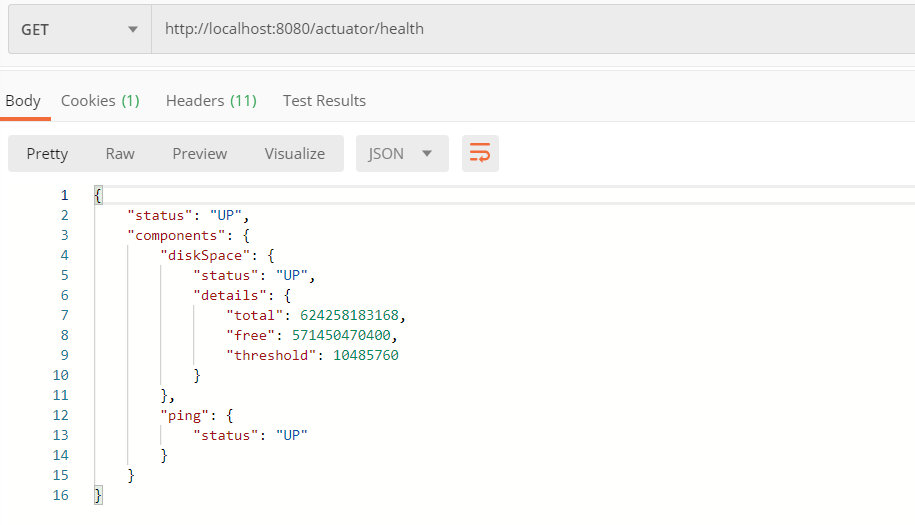
（2）always：表示将detail信息展示给所有人；

（3）when-authorized：表示将detail信息展示给认证用户，即登录后才可以查看；还可以通过management.endpoint.health.roles属性配置要求的角色，如果不配置，则默认所有登录的角色都可以访问，如果配置了，例如management.endpoint.health.roles=ADMIN表示认证的用户必须具有ADMIN角色才能查看details信息

如果开启了details，将其值设置为always，如下：



第一行表示暴露所有端点；访问“/actuator/health”路径，显示如下：

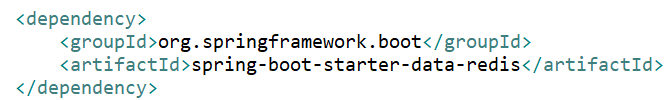


### 健康指示器

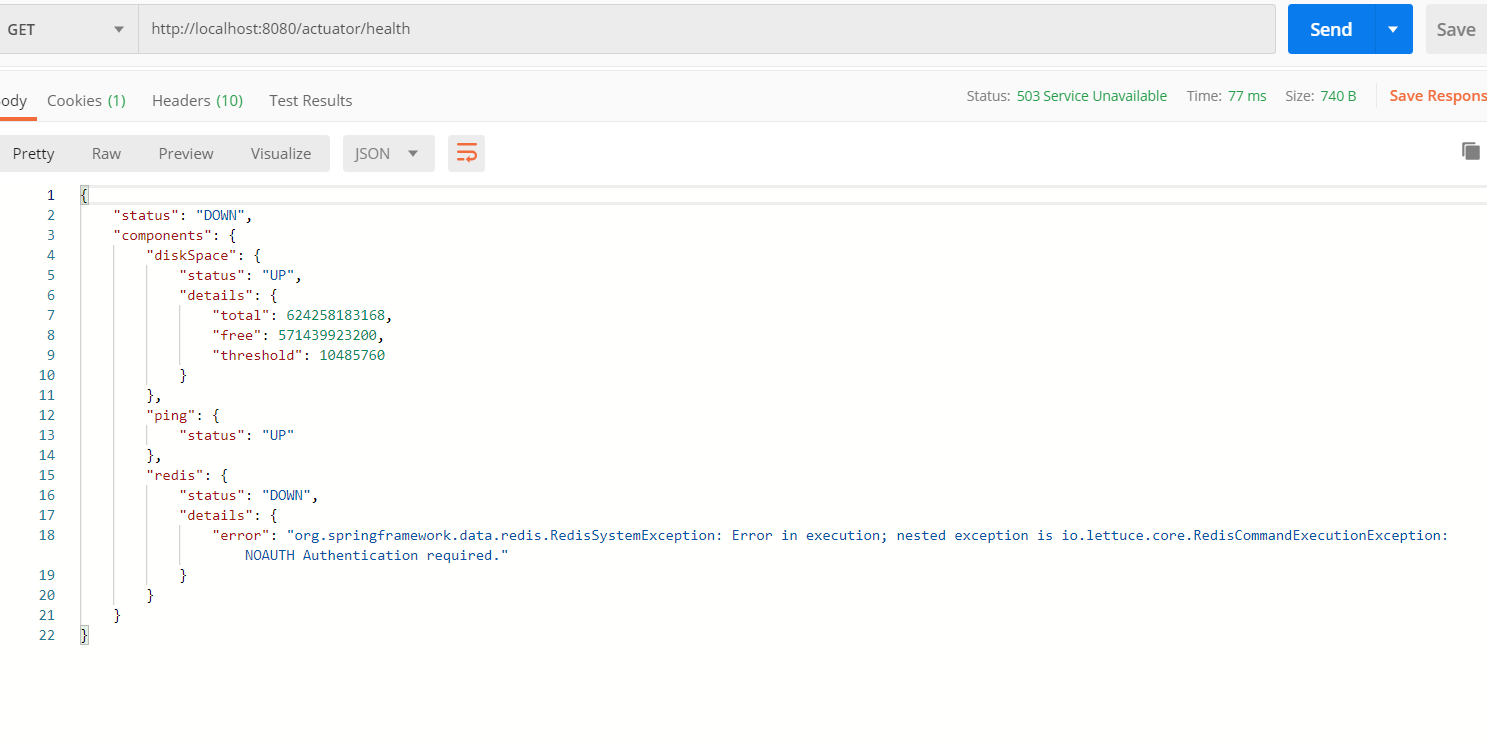
在访问health端点，其显示的健康信息是与所添加的依赖有关的。Spring Boot会根据依赖的添加情况来自动配置一些HelathIndicators，如下：

|  |  |
| --- | --- |
| **名字** | **描述** |
| CassandraHealthIndicator | 检查Cassandra数据库状况 |
| DiskSpaceHealthIndicator | 低磁盘空间检查 |
| DataSourceHealthIndicator | 检查是否可以从DataSource获取一个Connection |
| ElasticsearchHealthIndicator | 检查Elasticsearch集群状况 |
| InfluxDbHealthIndicator | 检查InfluxDB状况 |
| JmsHealthIndicator | 检查JMS消息代理状况 |
| MailHealthIndicator | 检查邮件服务器状况 |
| MongoHealthIndicator | 检查MongDB服务器状况 |
| Neo4jHealthIndicator | 检查Neo4j服务器状况 |
| RabbitHealthIndicator | 检查Rabbit服务器状况 |
| RedisHealthIndication | 检查Redis服务器状况 |
| SolrHealthIndication | 检查Solr服务器状况 |

那么项目中存在相关的依赖，那么列表中对应的HealthIndicators将会被自动配置，然后显示在health端点信息中。例如，在项目中添加了Redis依赖，如下：



那么访问health端点会显示的健康信息中会包括Redis的相关信息，如下：



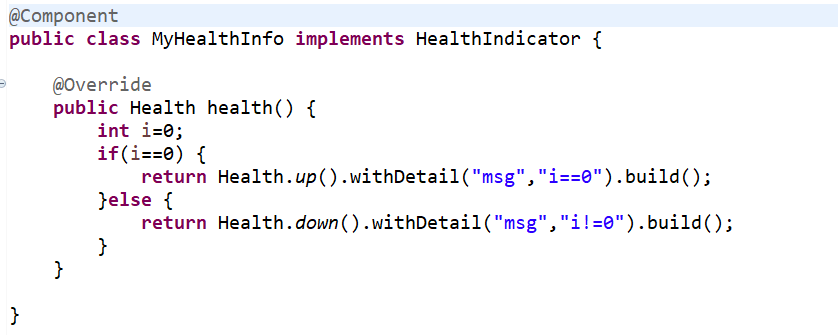
如果开发者不需要监控这么多HealthIndicators，则可以通过如下配置关闭所有的HealthIndicators自动化配置。



### 自定义HealthInfo

health端点除了收集HealthIndicators健康信息外，开发者可以自定义HealthInfo，只需要实现HealthIndicator接口覆写health方法即可。

示例：

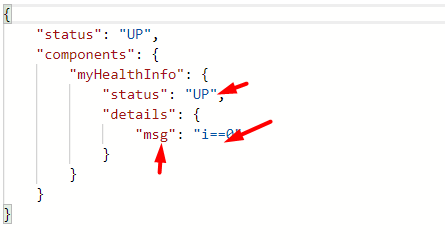


这里以i==0为假设，在实际开发中往往检查某项功能是否正常作为判断条件，如网络连接正常；

up和down方法分别对应两种常见的响应状态，即“UP”和“DOWN”；

withDetail方法用于将信息添加到health端点中，信息名为msg，信息内容为i==0；

如果此时访问health端点，如下：

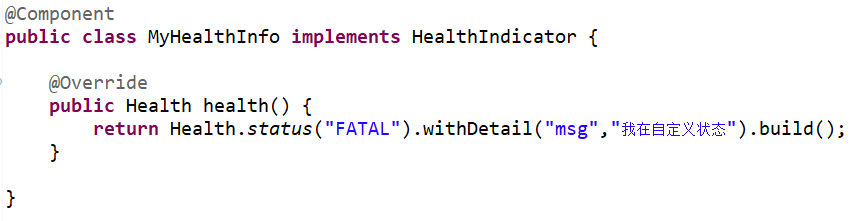


### 自定义状态

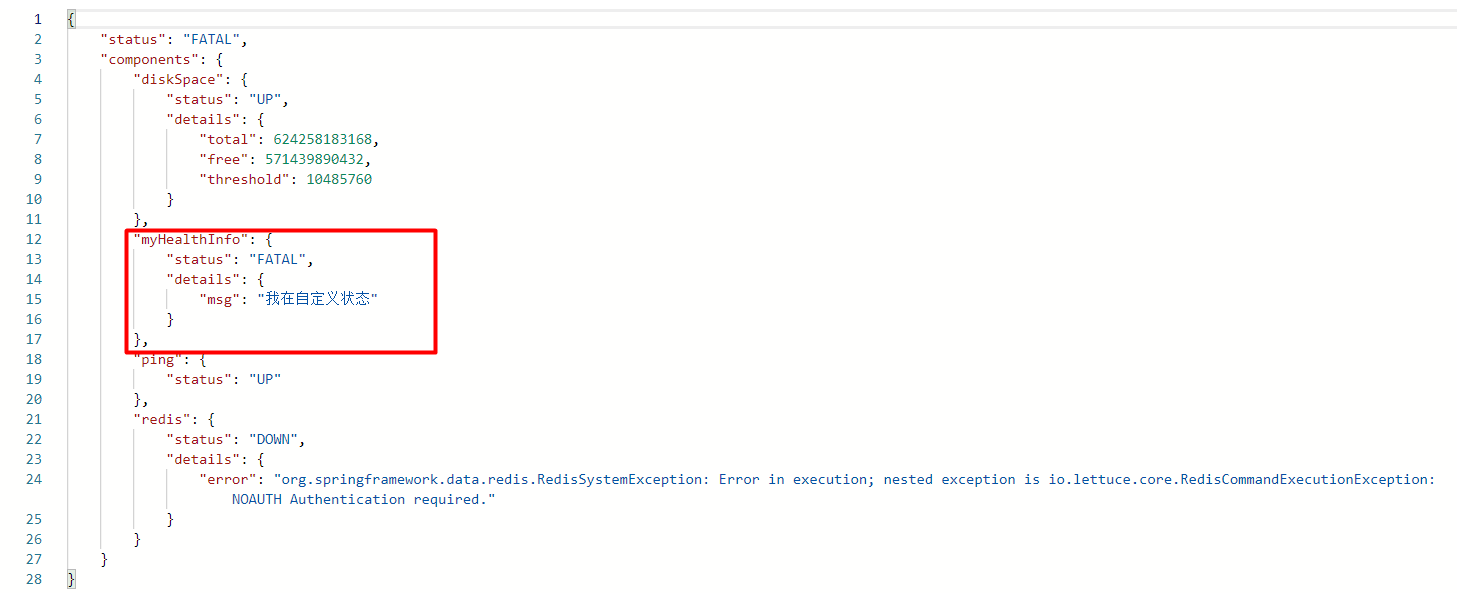
默认的响应状态一共有4种，定义在OrderedHealthAggregator类中，分别是DOWN、OUT\_OF\_SERVICE、UP、UNKNOWN。如果开发者想增加响应状态，可以自定义类继承自HealthAggregator或者在application.properties中通过management.health.status.order属性进行配置。如下：



此时，开发者如果想自定义一个状态为FALAT，配置完成后，就直接在health方法中返回自定义的响应状态了，修改MyHealth的health方法，如下：



访问health端点，如下：



成功自定义状态。

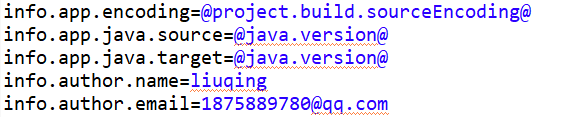
## info端点

info端点开启并暴露后默认通过“/actuator/info”路径访问，它包含的信息主要分为三大类：自定义信息、Git信息、项目构建信息。

### 自定义信息

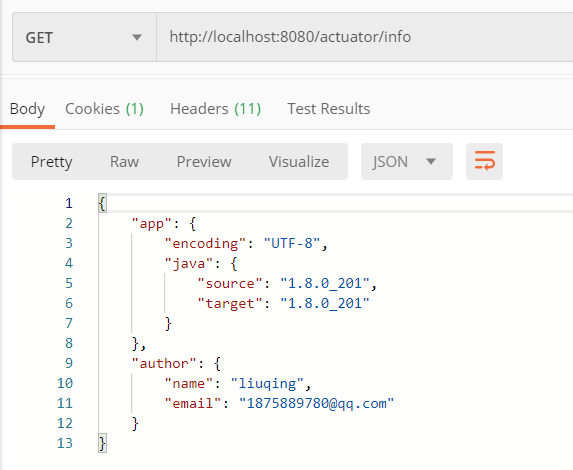
自定义信息可以在application.properties中添加，也可以通过Java代码添加。

1）在application.properties中添加，如下：



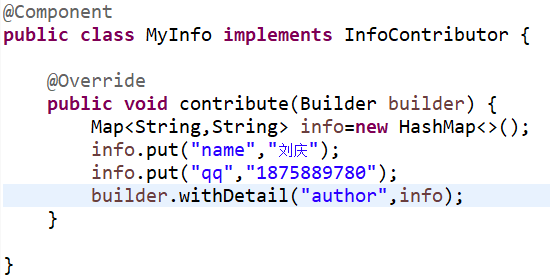
“@...@”表示引用Maven中的定义；

启动项目，访问info端点，如下：

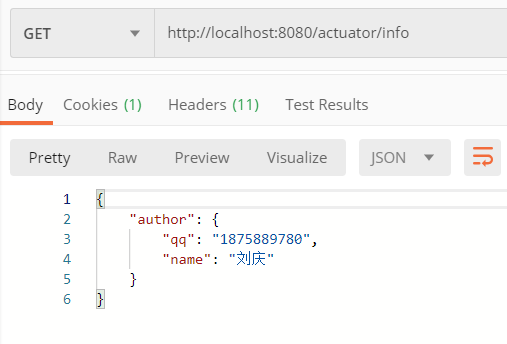


1. 通过Java代码自定义信息，只需要自定义类继承自InfoContributor，然后实现

contribute方法即可，如下



启动项目，访问info端点，如下：



### Git信息

Git是指Git提交信息，当classpath下存在一个git.properties文件时，Spring Boot会自动配置一个GitProperties Bean；开发者可通过Git插件自定生成Git提交信息，然后将这些信息展示在info端点中。

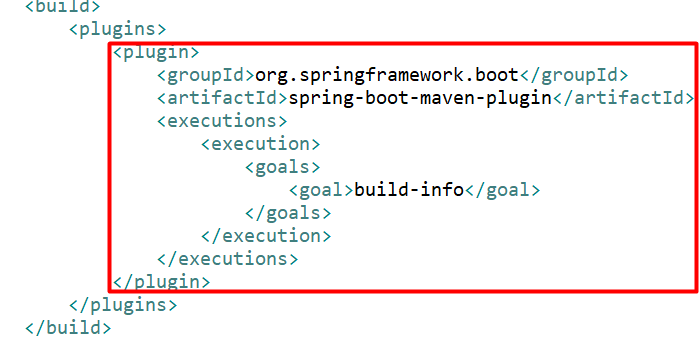
这些信息包括分支、提交的版本号、提交人、提交时间、提交用户、提交备注等信息。

### 项目构建信息

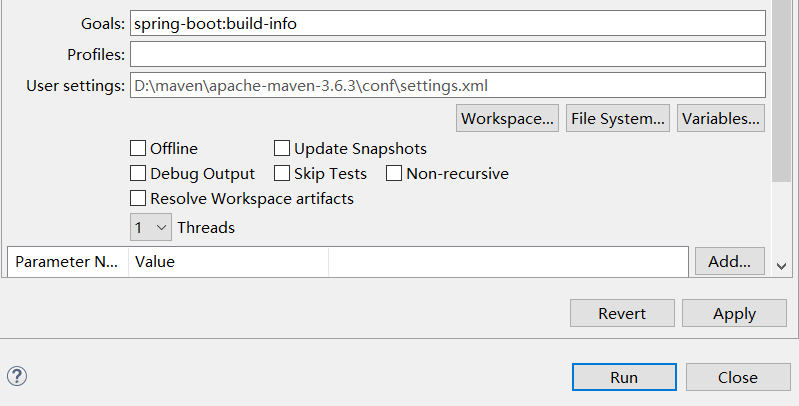
如果在classpath下存在META-INF/build-info.properties，Spring Boot将自动构建B uilderProperties Bean，然后info端点会发布build-info.properties文件中信息。

build-info.properties文件可通过插件生成，如下：

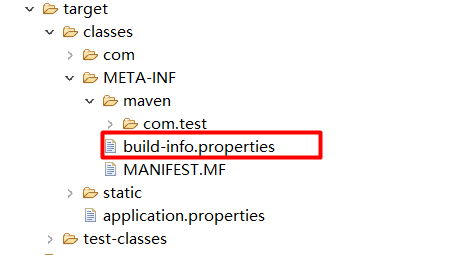
1. 在pom.xml中添加插件，如下：



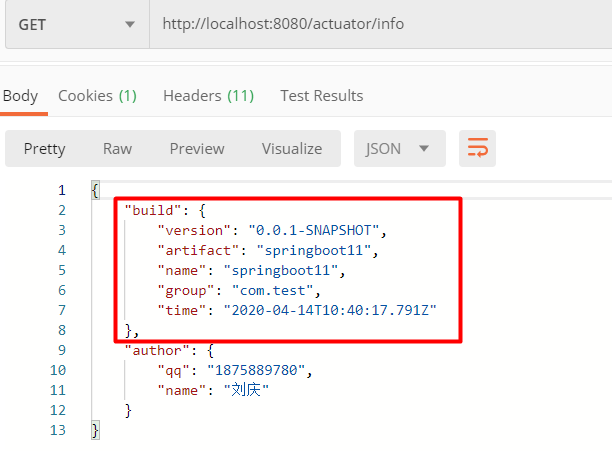
1. 执行Maven命令spring-boot:build-info，如下：



1. 在classpath下会生成build-info.properties文件，如下：



1. 启动项目，访问info端点，会显示build-info.properties文件中的信息，如下：



# 监控信息可视化

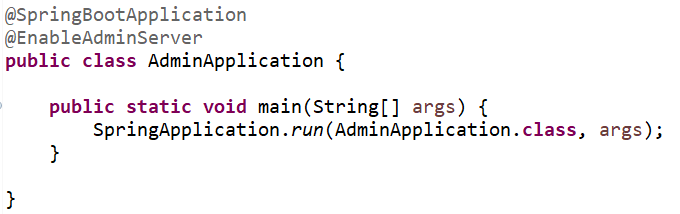
前面介绍了监控端点，但它们返回的都是JSON数据，这样查看起来就不方便。Spring Boot中提供了监控信息管理端，用于实现监控信息的可视化，这样可以方便开发者快速查看系统运行状况，而不用去一个一个地调用接口了。

## 创建可视化后台

1. 创建一个项目，作为后台，添加相关依赖，如下：

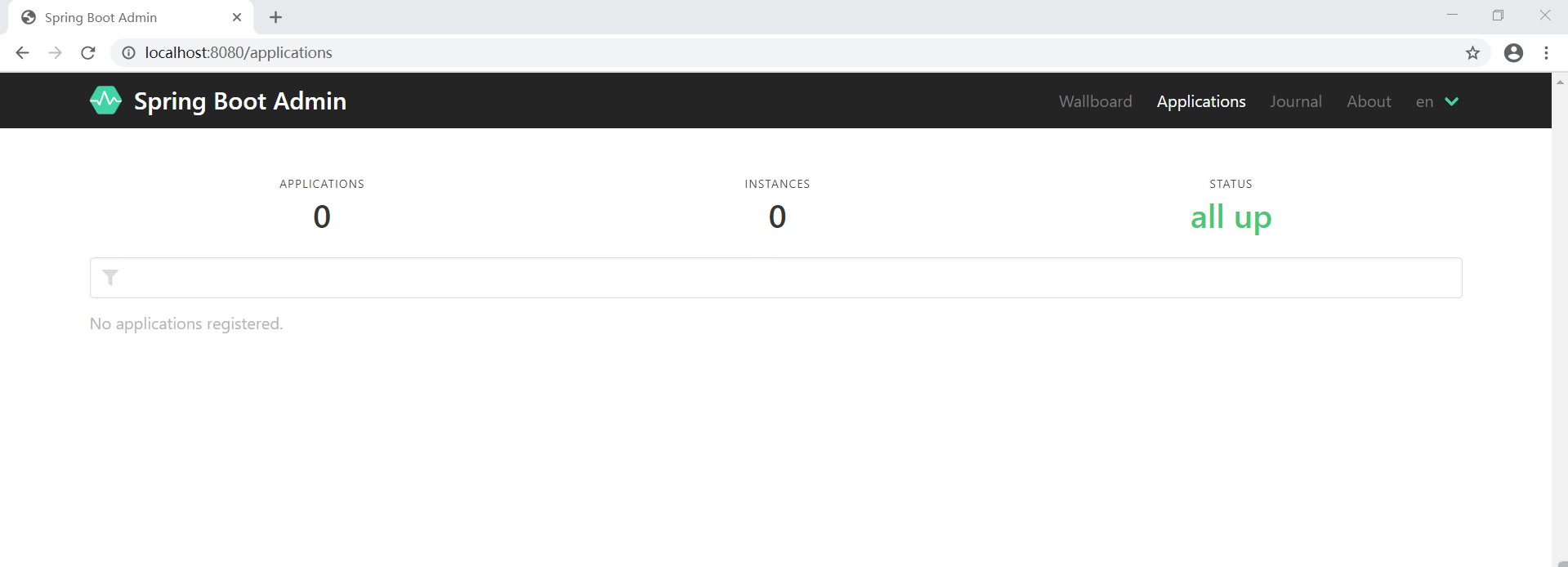


1. 在项目启动类上标注@EnableAdminServer注解，表示启动AdminServer，如下：



如果与上一项目端口号相同，记得修改一下端口号；

1. 启动项目，访问<http://localhost:8080/index.html>，如下：



Admin将通过图表的方式展示监控信息；

## 配置监控应用

一个应用要想被可视化后台监控，通过可视化后台来获取该应用信息，需要经过一些配置。（被监控的应用被也称为Client）

1. 在需要被监控的应用中添加依赖，如下：



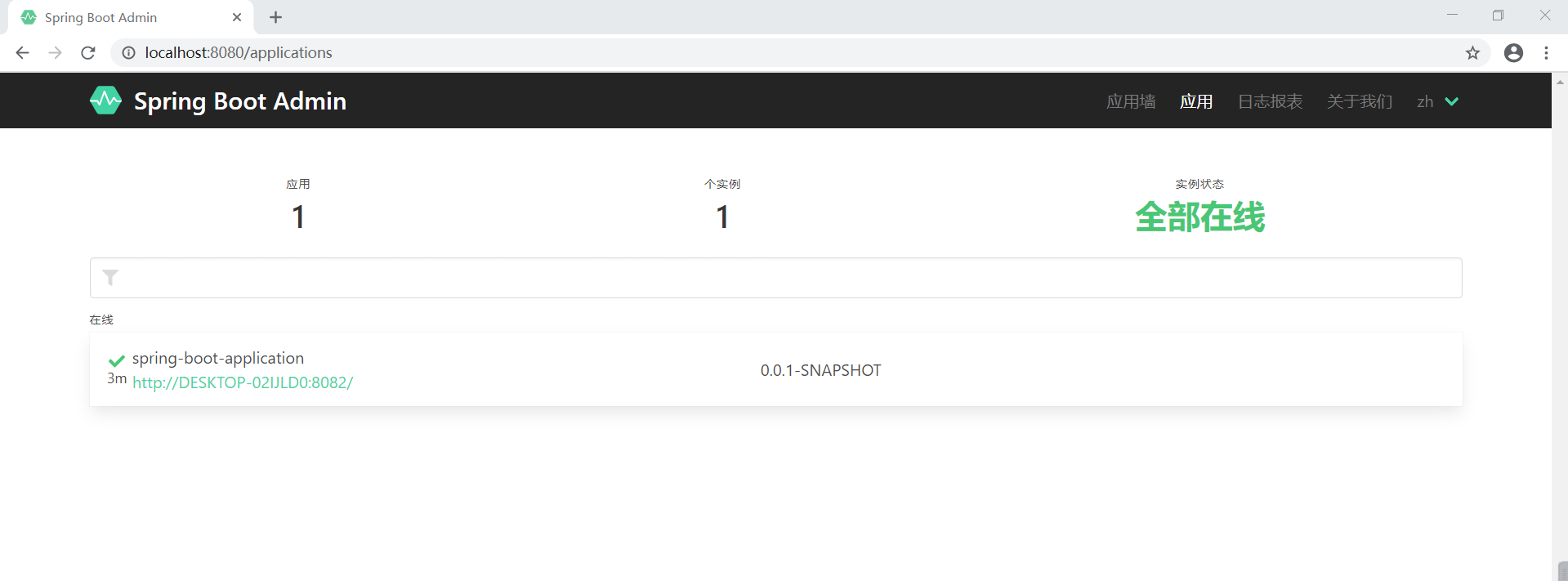
1. 在需要被监控的应用中添加配置，如下：



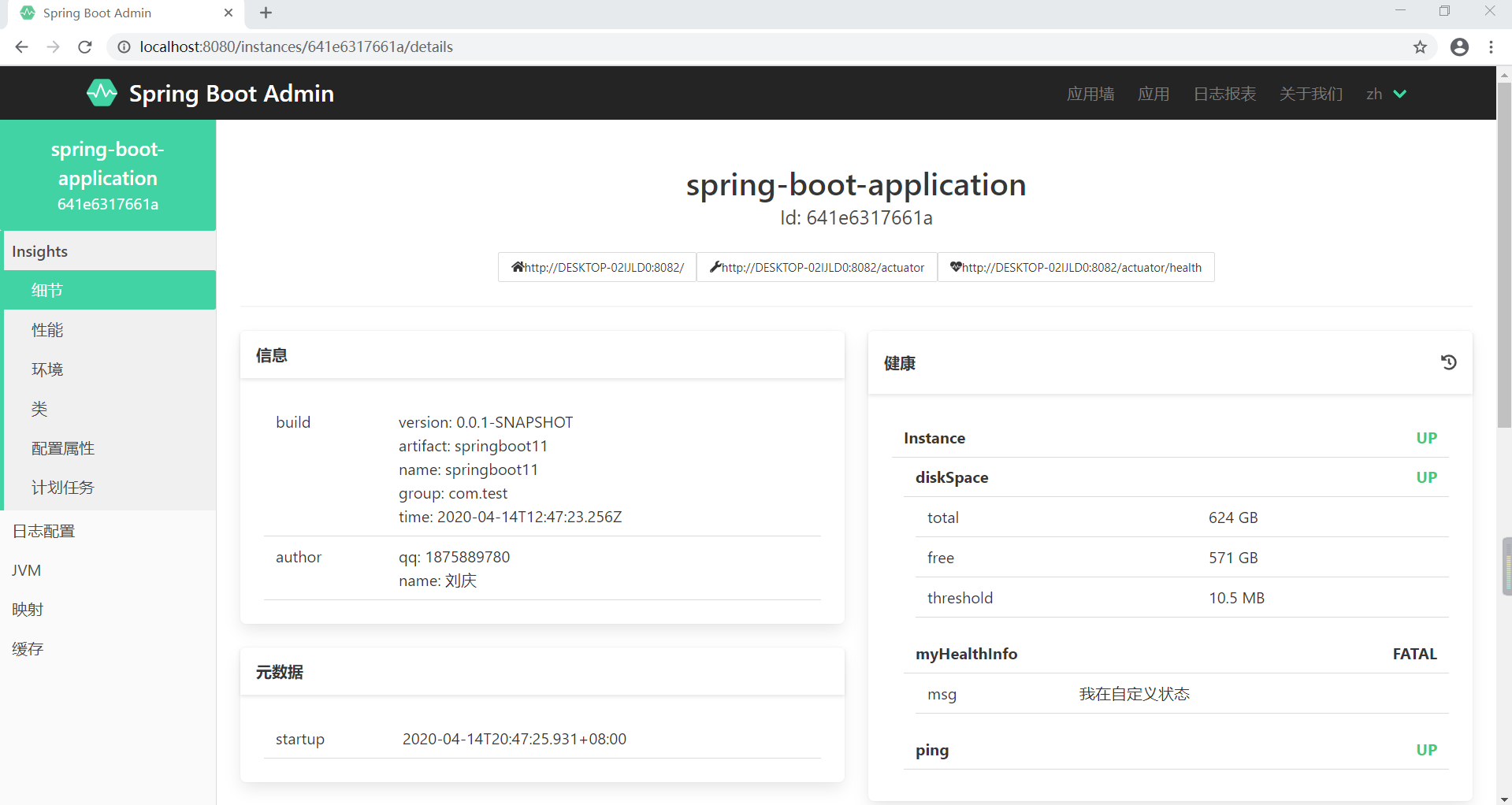
第一行表示修改服务器端口号，修改它是为了不与可视化后台项目的端口号相同

第二行表示可视化后台AdminServer的URL地址；

1. 启动该项目，同时也启动可视化后台，访问localhost:8080/index.html，即访问可视化后台，如下：



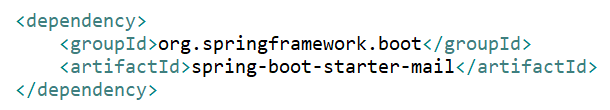
可以看到，应用被后台成功的管理，你可以在这里查看该被监控应用的更多信息，例如：



# 邮箱报警

虽然使用AdminServer可以实现监控信息的可视化，但是运维工程师不可能一天24小时盯着屏幕查看各个应用的运行状况。如果应用运行出现问题时能够主动通知运维工程师，这样就会方便很多。对此，我们可以使用邮件报警。

1. 在后台管理项目中，添加邮件发送依赖，如下：

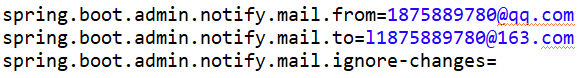


1. 在applicatio.properties中配置邮件发送需要的基本信息，如下：



以上配置如果不理解，请参考第十三章第一节；

1. 配置邮件报警，如下：



在这里设置了邮件发送人、收件人以及忽略掉的事件；

在默认情况下，当被监控应用的状态变为UNKNOWN或者UP时不会发送报警邮件，这里的配置表示被监控应用的任何变化都会发送报警邮件；

1. 启动管理后台项目，之后启动被监管的项目，可以发现，当监管的项目成功上线后，会给邮箱发送一份邮件，当被监管的项目被关闭，则也会发送一封报警邮件给邮箱。