15-应用程序监控管理

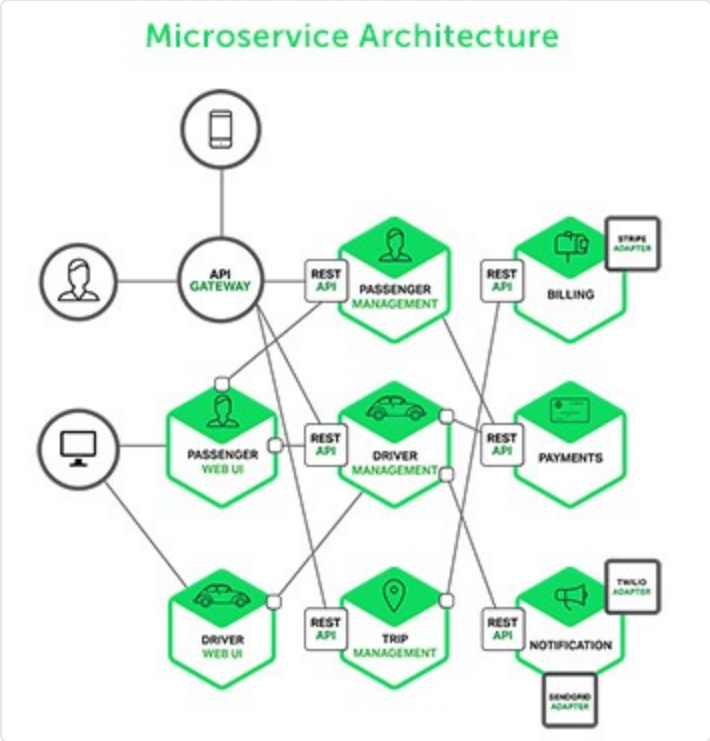
1.Actuator应用监控快速入门

一、 大纲

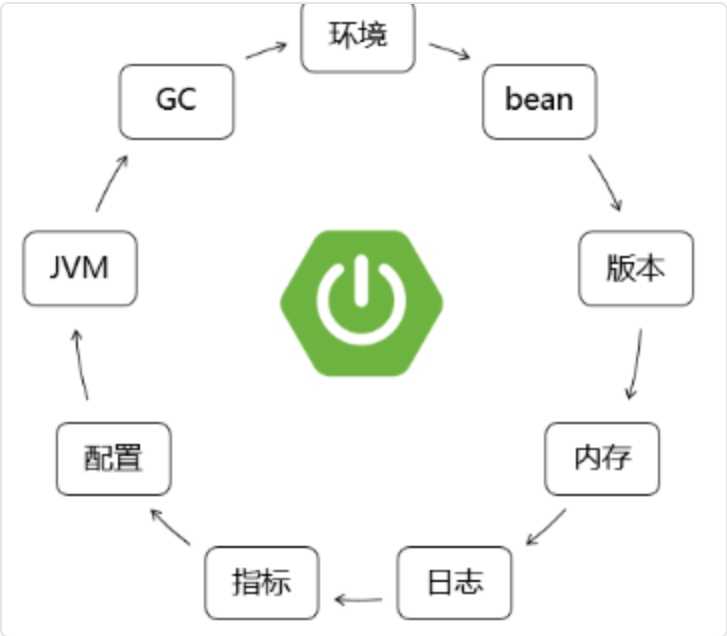
- Spring Boot Actuator简介
- Actuator开启与配置
- 常用监控端点说明

二、Spring Boot Actuator简介

Spring Boot作为构建微服务节点的方案，一定要提供全面而且细致的监控指标，使微服务更易于管理！微服务不同于单体应用，微服务的每个服务节点都单独部署，独立运行，大型的微服务项目甚至有成百上千个服务节点。这就为我们进行系统监控与运维提出了挑战。为了应对这个挑战，其中最重要的工作之一就是：微服务节点能够合理的暴露服务的相关监控指标，用以对服务进行健康检查、监控管理，从而进行合理的流量规划与安排系统运维工作！



Spring Boot Actuator 模块提供了生产级别的功能，比如健康检查，审计，指标收集，HTTP 跟踪等，帮助我们监控和管理Spring Boot 应用、Bean加载情况、环境变量、日志信息、线程信息，JVM 堆信息等。这个模块是一个采集应用内部信息暴露给外部的模块，上述的功能都可以通过HTTP 和 JMX 访问。



Actuator 也可以和一些外部的应用监控系统整合（Prometheus, Graphite, DataDog, Influx, Wavefront, New Relic等）。这些监控系统提供了出色的仪表板，图形，分析和警报，可帮助你通过一个统一友好的界面，监视和管理你的应用程序。

三、Actuator开启与配置

> 新建 boot-actuator模块

3.1.开启监控

在Spring Boot 项目中开启Actuator非常简单，只需要引入如下的maven坐标即可。

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

3.2.默认开放访问的监控端点

Spring Boot Actuator启用之后，HTTP协议下默认开放了两个端点的访问：

- /actuator/health用以监控应用状态。返回值是应用状态信息，包含四种状态DOWN(应用不正常), OUT_OF_SERVICE(服务不可用),UP(状态正常), UNKNOWN(状态未知)。如果服务状态正常，我们访问http:/host:port/actuator/health得到如下响应信息：

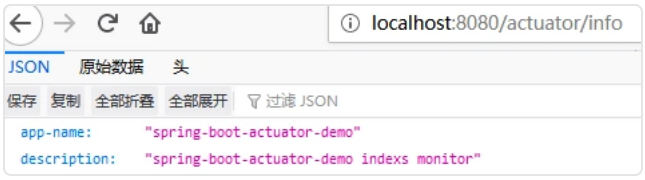
```java
{
    "status" : "UP"
}
```

从上面的响应结果看，该监控端点的监控信息非常有限，如果我们想让展示信息更加丰富的话，可以做如下配置。

```properties
management.endpoint.health.show-details=always
```

- /actuator/info 用来响应应用相关信息，默认为空。可以根据我们自己的需要，向服务调用者暴露相关信息。如下所示,配置属性可以随意起名，但都要挂在info下面：

```properties
info.app-name=spring-boot-actuator-demo
info.description=spring-boot-actuator-demo indexs monitor
```



3.3.开放端点配置（exposure）

如果我们希望开放更多的监控端点给服务调用者，需要配置：**开放部分监控端点**，端点名称用逗号分隔。

```properties
## 开放访问的服务端点
management.endpoints.web.exposure.include=beans,env
## 不暴露对外开放的服务端点
management.endpoints.web.exposure.exclude=mappings
```

开放所有监控端点:

```properties
management.endpoints.web.exposure.include=*
```

星号在YAML配置文件中中有特殊的含义，所以在YAML配置文件使用星号一定要加引号，如下所示：

```yaml
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

3.4.开启端点配置（enabled）

- 针对actuator提供的服务端点，开启启用(enabled)不等于开放访问(include)。
- 绝大部分的监控端点是默认开启的（下图中的Yes），少部分监控端点默认是不开启的，比如：shutdown。
- 对于默认不启用的监控服务端点，一定要先开启(enabled)，开启的配置方法如下：

```properties
# shutdown是服务端点名称，可以替换
management.endpoint.shutdown.enabled=true
```

三、常用监控端点说明

Spring Boot Actuator监控端点的分类

- 静态配置类：主要是一些静态配置信息，比如： Spring Bean 加载信息、yml 或properties配置信息、环境变量信息、请求接口关系映射信息等；
- 动态指标类：主要用于展现程序运行期状态，例如内存堆栈信息、请求链信息、健康指标信息等；
- 操作控制类：主要是shutdown功能，用户可以远程发送HTTP请求，从而关闭监控功能。

ID(监控端点名称)	描述	服务是否默认启用
auditevents	应用程序的审计事件相关信息	Yes
beans	应用中所有Spring Beans的完整列表	Yes
conditions	（configuration and auto-configuration classes）的状态及它们被应用或未被应用的原因	Yes
configprops	@ConfigurationProperties的集合列表	Yes
env	Spring的 ConfigurableEnvironment的属性	Yes
flyway	flyway 数据库迁移路径，如果有的话	Yes
liquibase	Liquibase数据库迁移路径，如果有的话	Yes
metrics	应用的metrics指标信息	Yes
mappings	所有@RequestMapping路径的集合列表	Yes
scheduledtasks	应用程序中的计划任务	Yes
sessions	允许从Spring会话支持的会话存储中检索和删除(retrieval and deletion)用户会话。使用Spring Session对反应性Web应用程序的支持时不可用。	Yes
shutdown	允许应用以优雅的方式关闭（默认情况下不启用）	No
threaddump	线程名、线程ID、线程的状态、是否等待锁资源、线程堆栈等信息	Yes
httptrace显示	HTTP跟踪信息（默认显示最后100个HTTP请求 – 响应交换）	Yes

如果使用web应用(Spring MVC, Spring WebFlux, 或者 Jersey)，还可以使用以下端点：

ID (监控端点名称)	描述	默认启用
heapdump	返回一个GZip压缩的hprof堆dump文件	Yes
jolokia	通过HTTP暴露JMX beans（当Jolokia在类路径上时，WebFlux不可用）	Yes
logfile	返回日志文件内容（如果设置了logging.file或logging.path属性的话），支持使用HTTP **Range**头接收日志文件内容的部分信息	Yes
prometheus	以可以被Prometheus服务器抓取的格式显示metrics信息	Yes

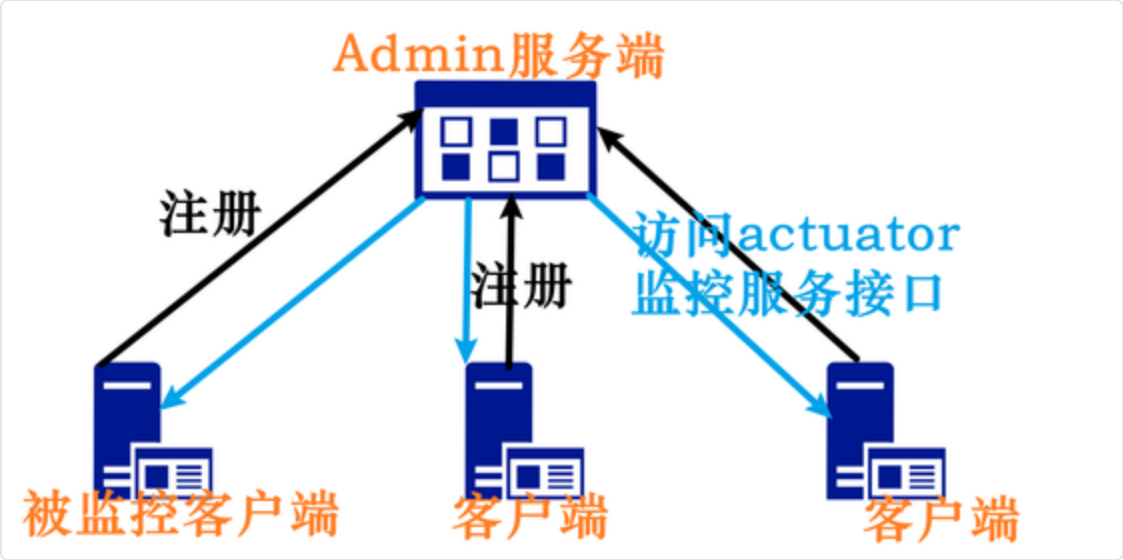
2.SpringBootAdmin界面化监控

在之前的内容中，我们学习了如何使用 Spring Boot Actuator对单个Spring Boot应用进行监控。通过浏览器访问url可以各个监控应用端点，可以知道应用的运行状态信息。但是这种方式返回的数据都是JSON的格式，各种指标需要单独访问查看。

我们迫切希望有一种方式：可以通过方便地通过鼠标点击和友好的UI界面的展现方式查看应用的各种运行指标。Spring Boot Admin应运而生！

一、Spring Boot Admin 介绍

- Spring Boot Admin是一个针对Spring Boot Actuator的JSON数据响应结果进行UI美化封装的监控工具
- 通过Spring Boot Admin，可以在可视化页面中浏览所有被监控的spring-boot项目的Actuator运行时信息，甚至还可以直接修改logger的level。



- Spring Boot Admin包括客户端和服务端两个部分，一个服务端可以展示多个客户端的监控结果：
  - 客户端：**即需要监控的应用服务**，需集成spring-boot-admin-starter-client,通过HTTP协议注册到Spring Boot Admin服务端,从而进行集中展示。（也可以结合Spring Cloud服务注册中心）
  - 服务端：访问客户端的Actuator运行时数据，并使用UI界面进行展示。**是一个独立的Spring Boot应用**,需集成spring-boot-admin-starter-server

二、创建SpringBoot Admin服务端

新建一个模块：boot-admin ，maven坐标引入

XML | 复制代码

```xml
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-server</artifactId>
</dependency>
```

在项目启动类上面加上@EnableAdminServer注解

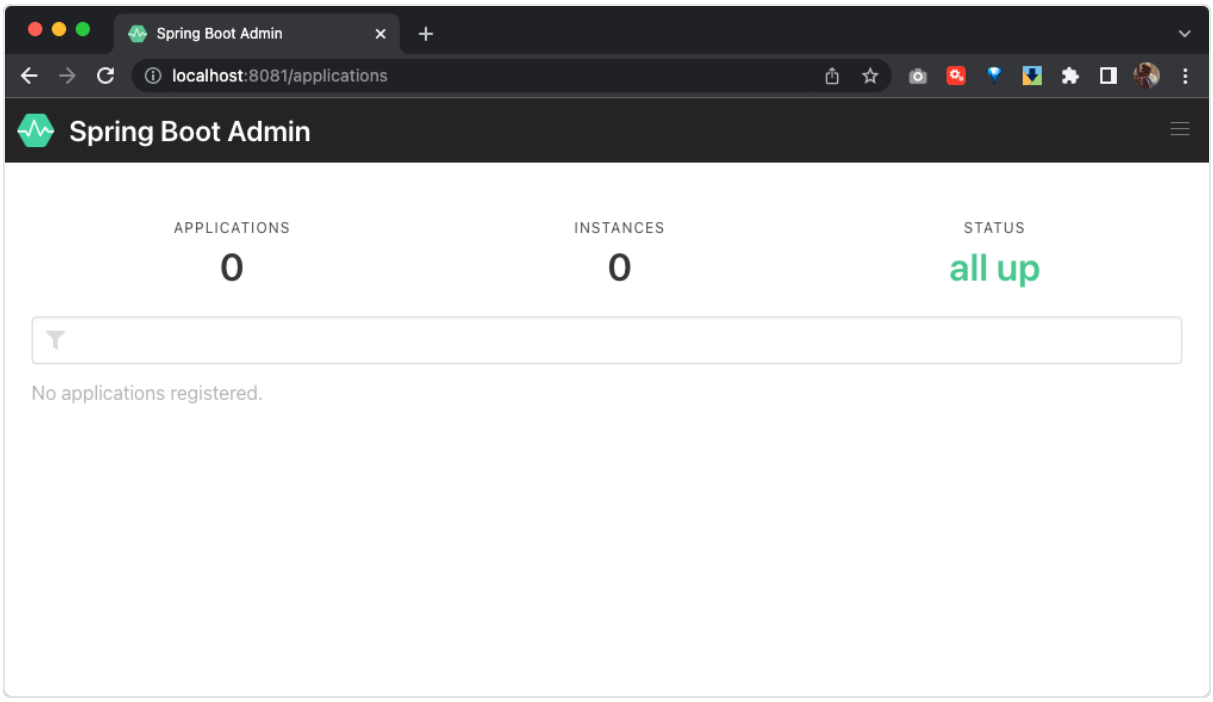
```java
@EnableAdminServer
@SpringBootApplication
public class BootActuatorAdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(BootLaunchActuatorAdminApplication.class, args);
    }
}
```

指定访问端口

```yaml
server:
  port: 8081

```

访问 http://localhost:8081/ ，因为目前没有任何的客户端应用注册上来，所以界面上没有相关的展示信息。



三、集成SpringBoot Admin客户端

在我们之前的项目boot-actuator里面引入下面的依赖，作为被SpringBoot Admin监控的客户端存在。

```xml
<dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
</dependency>
```

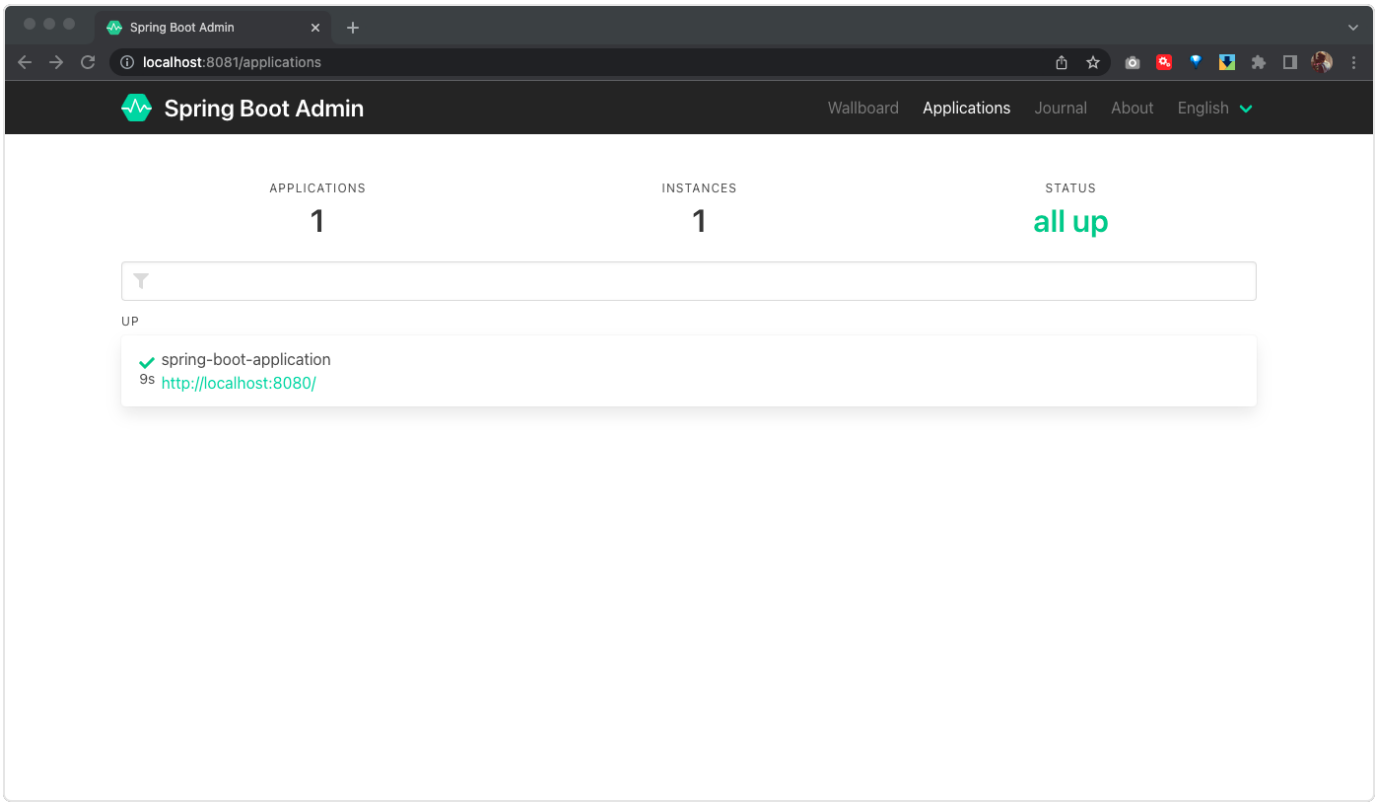
进行application.yml配置

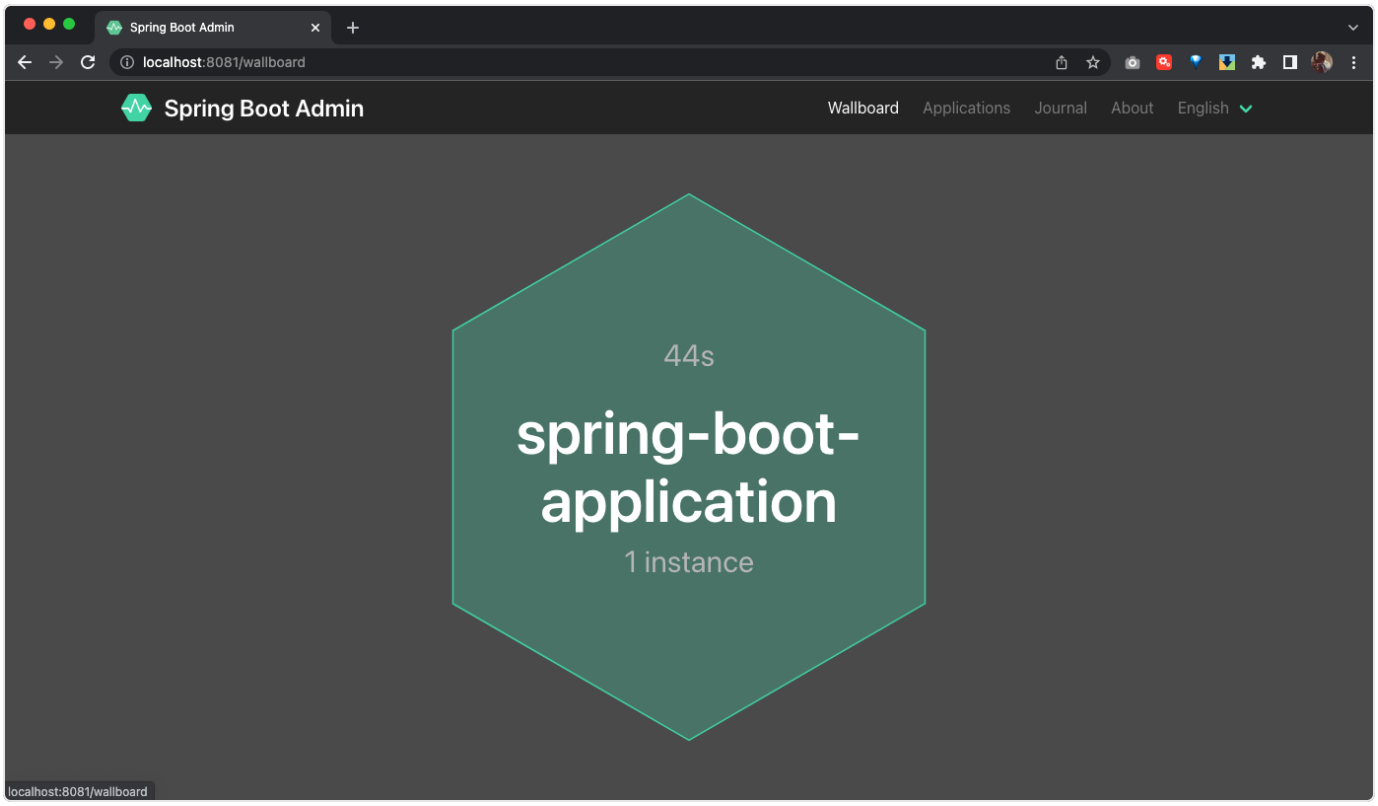
```yaml
spring:
  boot:
    admin:
      client:
        url: http://localhost:8081
management:
  endpoint:
    health:
      show-details: always
  endpoints:
    web:
      exposure:
        include: '*'
```

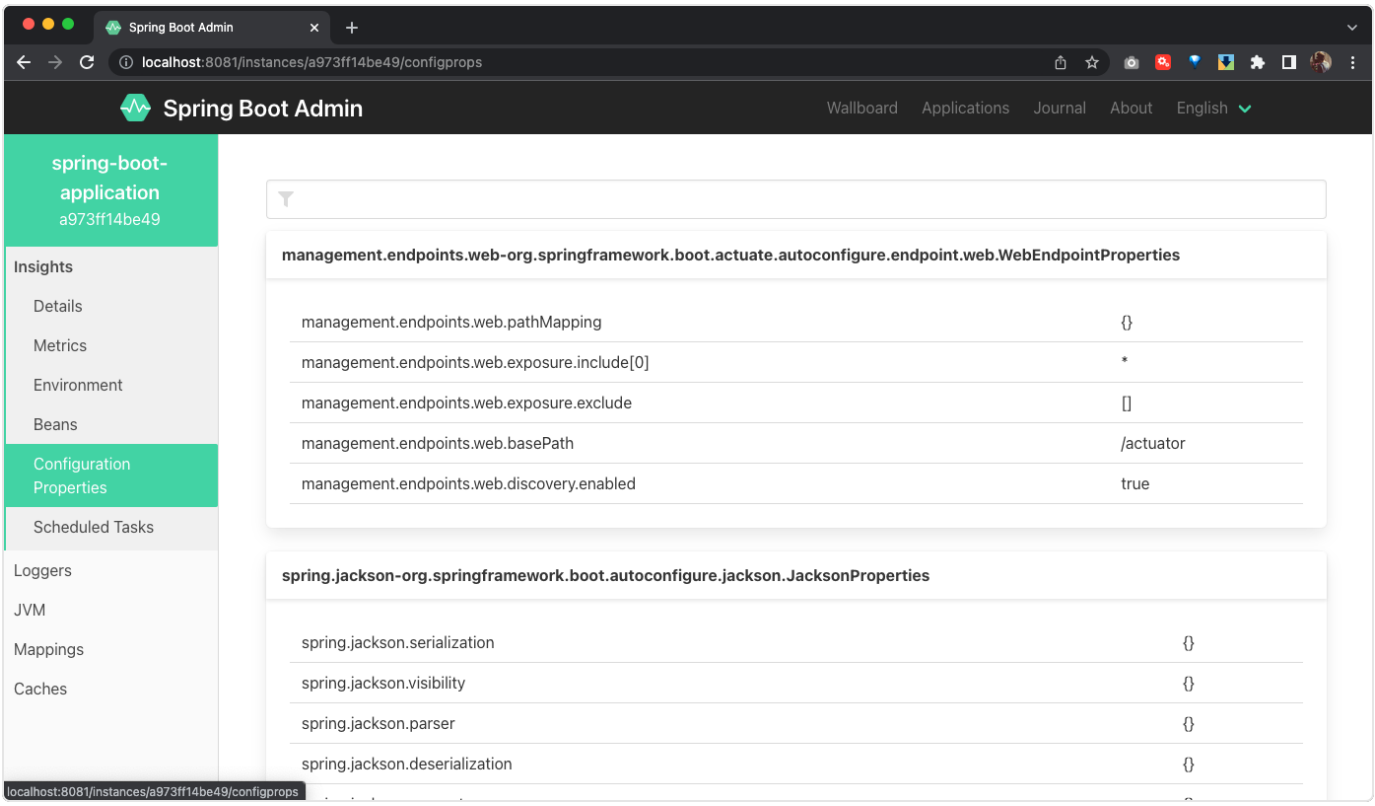
- spring.boot.admin.client.url体现的是服务端的访问地址，也就是监控注册的地址。
- 当客户端注册到spring boot admin服务端之后，admin服务端就会访问客户端应用的"/actuator"访问端点信息，因为我们为boot-launch配置了用户密码的访问权限(上一节)，所以需要将用户名密码告知服务端，它才能正确的获取"/actuator"访问端点信息。这就是metadata.user配置段的作用

另外，为了让Spring Boot Admin 展示的内容更加丰富，我们将所有的服务端点都开放出来。如果不开放，spring boot admin只能获取健康检查"/health"和“/info”两个默认开放访问的基础信息，也就失去了使用Spring Boot Admin的意义。

再次访问 http://localhost:8081/ ，这次展示的监控信息内容就很多了：


Spring Boot Admin
localhost:8081/applications
Spring Boot Admin
Wallboard
Applications
Journal
About
English
APPLICATIONS
1
INSTANCES
1
STATUS
all up
UP
spring-boot-application
9s
http://localhost:8080/


Spring Boot Admin
localhost:8081/wallboard
Spring Boot Admin
Wallboard
Applications
Journal
About
English
44s
spring-boot-application
1 instance
localhost:8081/wallboard


Spring Boot Admin
localhost:8081/instances/a973ff14be49/configprops
Spring Boot Admin
Wallboard
Applications
Journal
About
English
spring-boot-application
a973ff14be49
Insights
Details
Metrics
Environment
Beans
Configuration Properties
Scheduled Tasks
Loggers
JVM
Mappings
Caches
management.endpoints.web-org.springframework.boot.actuate.autoconfigure.endpoint.web.WebEndpointProperties
management.endpoints.web.pathMapping {}
management.endpoints.web.exposure.include[0] *
management.endpoints.web.exposure.exclude []
management.endpoints.web.basePath /actuator
management.endpoints.web.discovery.enabled true
spring.jackson-org.springframework.boot.autoconfigure.jackson.JacksonProperties
spring.jackson.serialization {}
spring.jackson.visibility {}
spring.jackson.parser {}
spring.jackson.deserialization {}
localhost:8081/instances/a973ff14be49/configprops