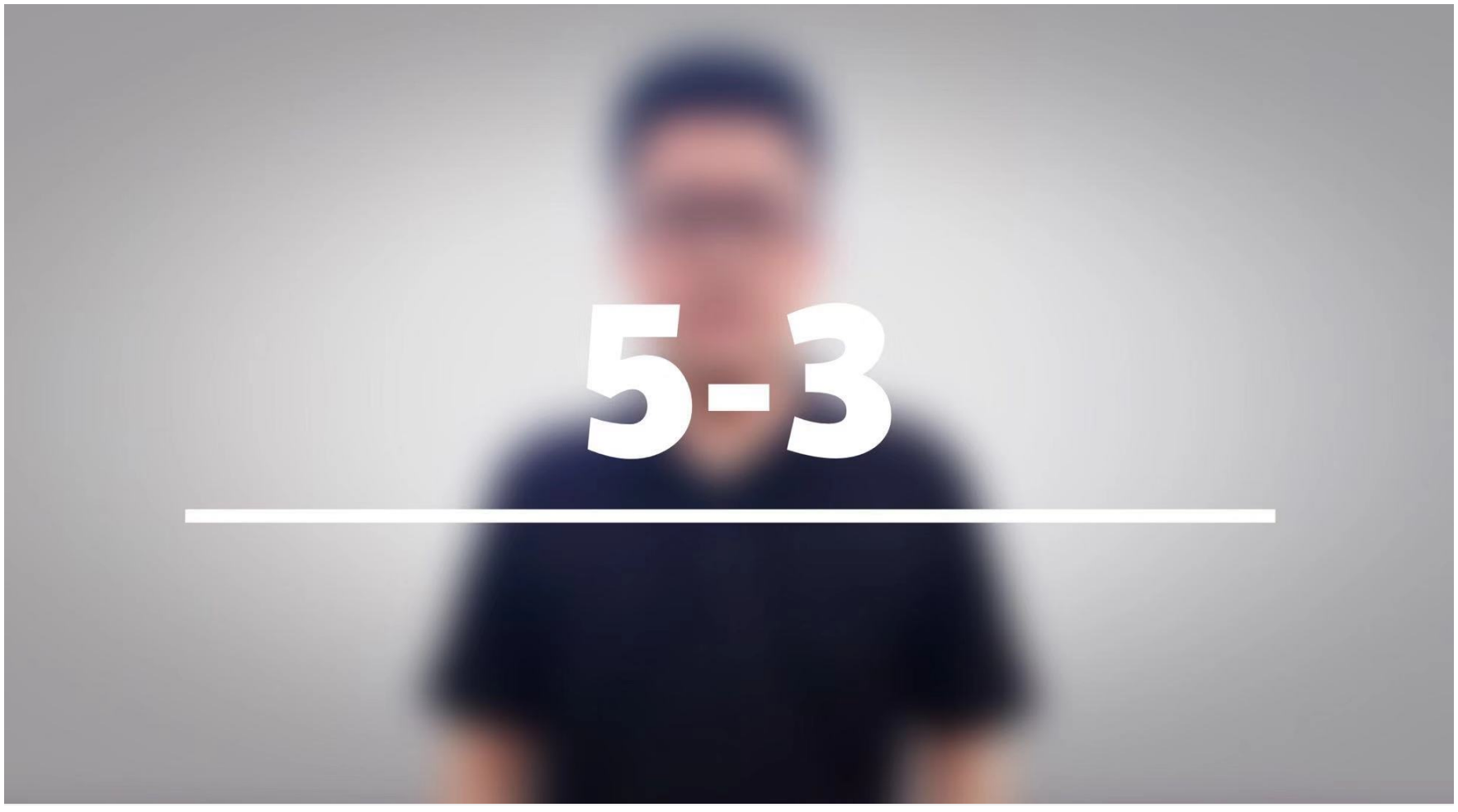


第 5-3 课： Spring Boot Admin 的使用

Spring Boot Actuator 提供了对单个 Spring Boot 应用的监控，信息包含应用状态、内存、线程、堆栈等，比较全面的监控了 Spring Boot 应用的整个生命周期。

但是这样监控也有一些问题：第一，所有的监控都需要调用固定的接口来查看，如果全面查看应用状态需要调用很多接口，并且接口返回的 JSON 信息不方便运营人员理解；第二，如果 Spring Boot 应用集群非常大，每个应用都需要调用不同的接口来查看监控信息，操作非常繁琐低效。在这样的背景下，就诞生了另外一个开源软件：**Spring Boot Admin**。

什么是 Spring Boot Admin

Spring Boot Admin 是一个管理和监控 Spring Boot 应用程序的开源软件，每个应用都认为是一个客户端，通过 HTTP 或者使用 Eureka 注册到 admin server 中进行展示，Spring Boot Admin UI 部分使用 Vue.js 将数据展示在前端。

Spring Boot Admin 是一个针对 Spring Boot 的 Actuator 接口进行 UI 美化封装的监控工具，它可以在列表中浏览所有被监控 spring-boot 项目的基本信息、详细的 Health 信息、内存信息、JVM 信息、垃圾回收信息、各种配置信息（比如数据源、缓存列表和命中率）等，还可以直接修改 logger 的 level。

值得注意的是 Spring Boot Admin 并不是 Spring Boot 官方出品的开源软件，但是其软件质量和使用广泛度都非常的高，并且 Spring Boot Admin 会及时随着 Spring Boot 的更新而更新，当 Spring Boot 推出 2.X 版本时 Spring Boot Admin 也及时进行了更新。

Spring Boot Admin 2.x 不仅是跟着支持了 Spring Boot 2.x，还在 1.x 的基础上进行了大量的更新和优化:

- 重新规划了项目依赖包，让项目中更方便的集成 Spring Boot Admin
- 1.x 前端使用了 Angular.js，2.x 使用 Vue 对界面进行了重写，界面美观度提升幅度非常高
- 提供了支持 Spring Cloud 的组件
- 其他更新，具体参考：Changes with 2.x (http://codecentric.github.io/spring-boot-admin/current/#_changes_with_2_x)

Spring Boot Admin 分为服务端和客户端，服务端其实就是一个监控后台用来汇总展示所有的监控信息，客户端就是我们的应用，使用时需要先启动服务端，在启动客户端的时候打开 Actuator 的接口，并指向服务端的地址，这样服务端会定时读取相关信息以达到监控的目的。

接下来演示如何使用 Spring Boot Admin 对 Spring Boot 应用进行监控。

监控单体应用

先给大家展示如何使用 Spring Boot Admin 监控单个 Spring Boot 应用。

Admin Server 端

项目依赖

```
<dependency>
  <groupId>de.codecentric</groupId>
  <artifactId>spring-boot-admin-starter-server</artifactId>
  <version>2.1.0</version>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

2.x 下只需要添加此一个包即可，其他组件会自动依赖添加。

配置文件

```
server.port=8000
```

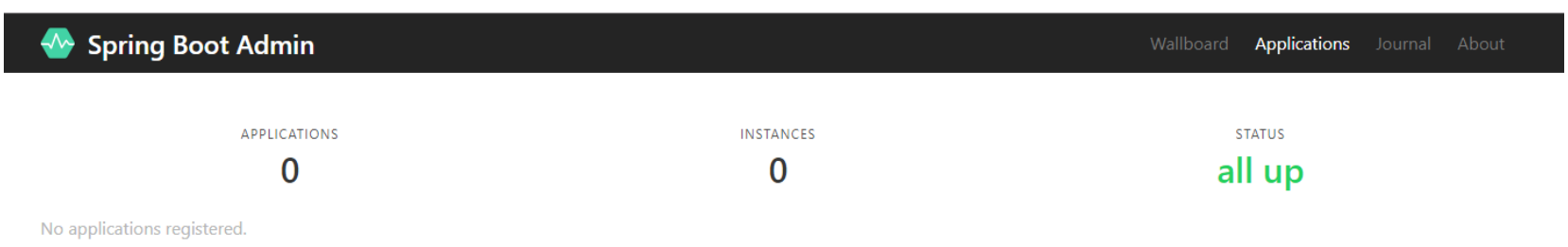
服务端设置端口为：8000。

启动类

```
@EnableAdminServer
@SpringBootApplication
public class AdminServerApplication {

  public static void main(String[] args) {
    SpringApplication.run(AdminServerApplication.class, args);
  }
}
```

完成上面三步之后，启动服务端，访问网址 http://localhost:8000 (http://local - host:8000) 可以看到以下界面：



因为刚启动没有应用，因此显示：No applications registered.，同时根据上图也可以看出 applications 页面会展示项目的应用数、实例数和状态三个信息。

接下来我们构建一个客户端，并注册到服务端。

Admin Client 端

项目依赖

```
<dependencies>
  <dependency>
    <groupId>de.codecentric</groupId>
    <artifactId>spring-boot-admin-starter-client</artifactId>
    <version>2.1.0</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
</dependencies>
```

添加 spring-boot-starter-web 是为了使应用处于启动状态，spring-boot-admin-starter-client 会自动添加 Actuator 相关依赖。

配置文件

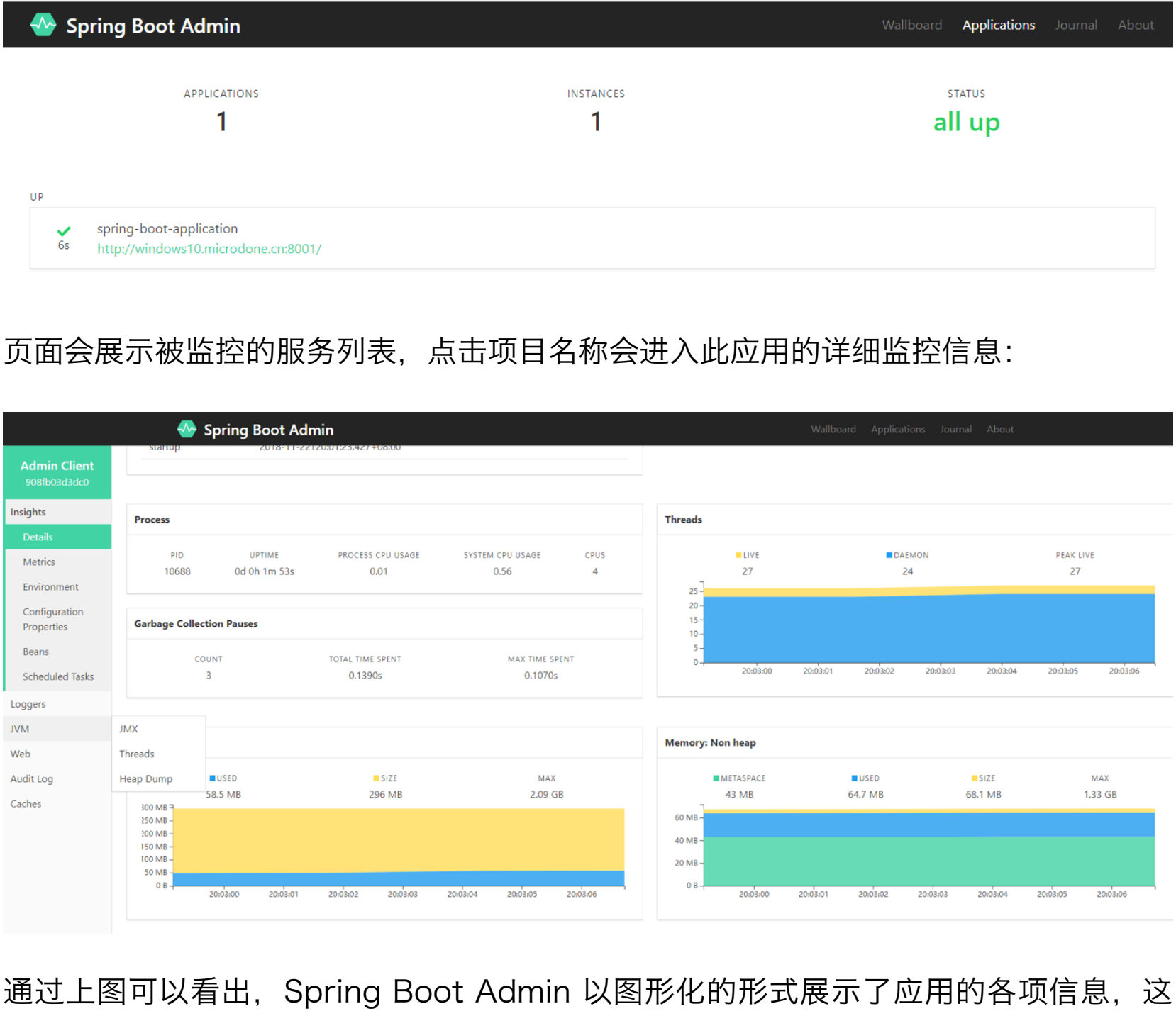
```
server.port=8001
spring.application.name=Admin Client
spring.boot.admin.client.url=http://localhost:8000
management.endpoints.web.exposure.include=*
```

- spring.boot.admin.client.url 配置 Admin Server 的地址
- management.endpoints.web.exposure.include=* 打开客户端 Actuator 的监控

启动类

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

配置完成之后，启动 Client 端，Admin 服务端会自动检查到客户端的变化，并展示其应用:

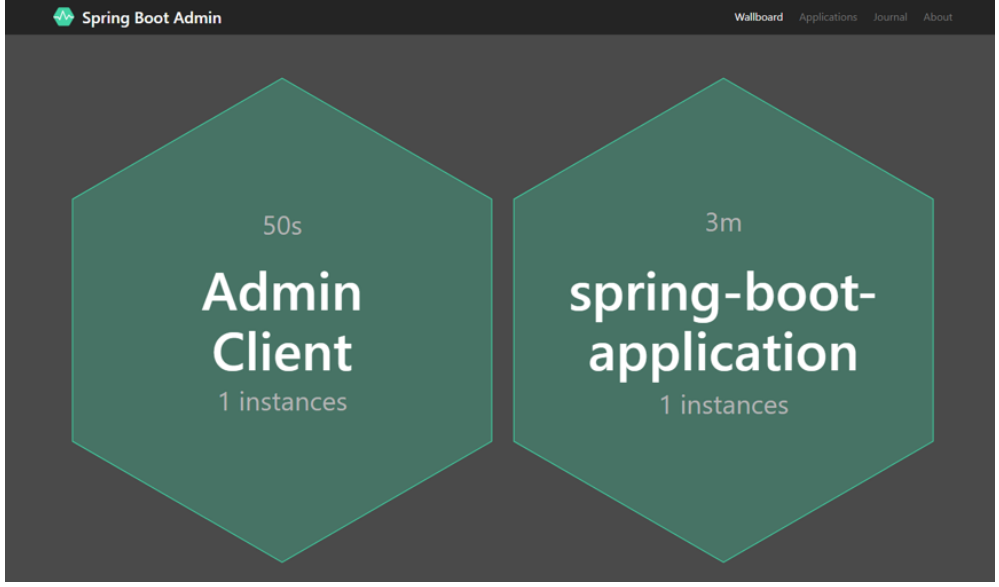


页面会展示被监控的服务列表，点击项目名称会进入此应用的详细监控信息：

通过上图可以看出，Spring Boot Admin 以图形化的形式展示了应用的各项信息，这些信息大多都来自于 Spring Boot Actuator 提供的接口。利用图形化的形式很容易看到应用的各项参数变化，甚至有些页面还可以进行一些操作，比如改变打印日志的级别等。

点击 journal 页面可以看到应用状态变化的历史过程：

点击第一个菜单 wallboard 可以以更形象的方式查看应用数量启动的时间等。



监控微服务

如果我们使用的是单个 Spring Boot 应用，就需要在每一个被监控的应用中配置 Admin Server 的地址信息；如果应用都注册在 Eureka 中就不需要再对每个应用进行配置，Spring Boot Admin 会自动从注册中心抓取应用的相关信息。

如果使用了 Spring Cloud 的服务发现功能，就不需要再单独添加 Admin Client 客户端，仅仅需要 Spring Boot Server，其他内容会自动进行配置。

接下来以 Eureka 作为服务发现的示例来进行演示，实际上也可以使用 Consul 或者 Zookeeper。

（1）服务端和客户端添加 spring-cloud-starter-eureka 到包依赖中

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</arti-
factId>
</dependency>
```

（2）启动类添加注解

```
@Configuration
@EnableAutoConfiguration
@EnableDiscoveryClient
@EnableAdminServer
public class SpringBootAdminApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootAdminApplication.-
class, args);
    }

    @Configuration
    public static class SecurityPermitAllConfig extends WebSecuri-
tyConfigurerAdapter {
        @Override
        protected void configure(HttpSecurity http) throws Excep-
tion {
            http.authorizeRequests().anyRequest().permitAll()
                .and().csrf().disable();
        }
    }
}
```

使用类 SecurityPermitAllConfig 关闭了安全验证。

（3）在客户端中配置服务发现的地址

```
eureka:
  instance:
    leaseRenewalIntervalInSeconds: 10
    health-check-url-path: /actuator/health
    metadata-map:
      startup: ${random.int}    #needed to trigger info and end-
point update after restart
  client:
    registryFetchIntervalSeconds: 5
    serviceUrl:
      defaultZone: ${EUREKA_SERVICE_URL:http://localhost:8761}/eu-
reka/

management:
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
      show-details: ALWAYS
```

Spring Cloud 提供了示例代码可以参考这里：spring-boot-admin-sample-eureka (https://github.com/codecentric/spring-boot-admin/tree/master/spring-boot-admin-samples/spring-boot-admin-sample-eureka/)。

重启启动服务端和客户端之后，访问服务端的相关地址就可以看到监控页面了。

安全控制

Spring Boot Admin 后台有很多的敏感信息和操作，如果公司不做权限控制可能会影响到公司系统的安全性。Spring Boot Admin 也考虑到了这个因素，可以利用前面的 Spring Security 做安全访问控制，在 spring-boot-admin-server 上进行改造。

（1）添加 Spring Boot Security 依赖包

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

（2）添加安全访问控制

和前面的 Security 配置一样，给项目添加访问控制。

```
@Configuration
public  class SecuritySecureConfig extends WebSecurityConfigurer-
Adapter {
    private final String adminContextPath;

    public SecuritySecureConfig(AdminServerProperties adminServer-
Properties) {
        this.adminContextPath = adminServerProperties.getContext-
Path();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // @formatter:off
        SavedRequestAwareAuthenticationSuccessHandler successHan-
dler = new SavedRequestAwareAuthenticationSuccessHandler();
        successHandler.setTargetUrlParameter("redirectTo");
        successHandler.setDefaultTargetUrl(adminContextPath + "/");

        http.authorizeRequests()
            .antMatchers(adminContextPath + "/assets/**").permit-
All()
            .antMatchers(adminContextPath + "/login").permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin().loginPage(adminContextPath + "/login").suc-
cessHandler(successHandler).and()
        .logout().logoutUrl(adminContextPath + "/logout").and()
        .httpBasic().and()
        .csrf()
            .csrfTokenRepository(CookieCsrfTokenRepository.with-
HttpOnlyFalse())
            .ignoringAntMatchers(
                adminContextPath + "/instances",
                adminContextPath + "/actuator/**"
            );
    }
}
```

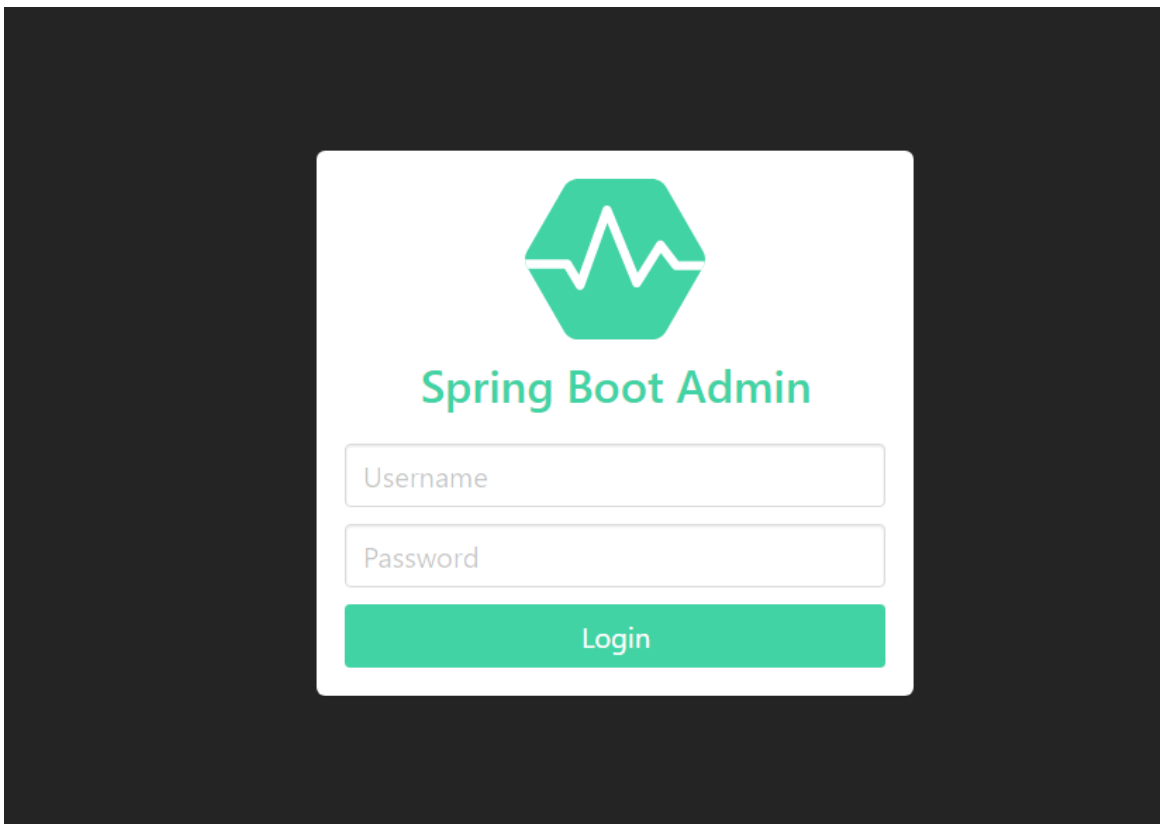
- antMatchers(adminContextPath + "/assets/**").permitAll() 所有静态内容不做安全验证

- anyRequest().authenticated() 其他请求均需要验证
- formLogin() 配置登录
- logout() 配置登出
- httpBasic() 支持 HTTP，引导 Spring Boot Admin 客户端注册
- csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse()) 打开跨站点请求保护 Cookies
- adminContextPath + "/instances" 取消跨站点请求保护 "/instances"，方便 Admin 客户端注册
- adminContextPath + "/actuator/** " 取消跨站点请求保护 "/actuator/**"，可以让 Admin 监控到 Actuator 的相关接口

然后可以给 security 设置一个用户名和密码:

```
spring.security.user.name=admin
spring.security.user.password=admin
```

配置完成之后重启 Admin Server 端，访问网址 http://localhost:8000 (http://lo - calhost:8000) 就会发现需要一个登录的用户名和密码了。



使用刚才设置的用户名和密码登录之后，发现注册到 Server 端的服务数成为 0 了，这是客户端统一也需要配置帐户名和密码信息。

```
spring.security.user.name=admin
spring.security.user.password=admin
```

配置完成之后重新启动，在服务端就又可以查看监控到的应用了。

其他方式

如果 Actuator 的端口被使用 HTTP 认证保护，那么 Spring Boot Admin Server 访问的时候需要凭证信息，这时候可以使用 metadata 的方式对账户和密码进行配置。

直接使用客户端注册的方式：

```
spring.boot.admin.client:
    url: http://localhost:8080
    instance:
      metadata:
        user.name: ${spring.security.user.name}
        user.password: ${spring.security.user.password}
```

使用 Eureka 进行注册的方式：

```
eureka:
  instance:
    metadata-map:
      user.name: ${spring.security.user.name}
      user.password: ${spring.security.user.password}
```

Eureka 中的 metadataMap 是专门用来存放一些自定义的数据，当注册中心或者其他服务需要此服务的某些配置时可以在 metadataMap 里取。实际上，每个 instance 都有各自的 metadataMap，map 中存放着需要用到的属性。例如，上面配置中的 eureka.instance.metadata-map.user.name，当这个服务成功注册到 Eureka 上，Spring Boot Admin 就会拿到这个 instance，进而拿到 metadataMap 里的属性，然后放入请求头，向此服务发送请求，访问此服务的 Actuator 开放的端点。

邮件告警

Spring Boot Admin 将微服务中所有应用信息在后台进行了展示，非常方便我们对微服务整体的监控和治理。但是我们的运营人员也不可能一天 24 小时盯着监控后台，因此如果服务有异常的时候，有对应的邮件告警就太好了，其实 Spring Boot Admin 也给出了支持。

我们对上面的示例项目 spring-boot-admin-server 进行改造。

添加依赖

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

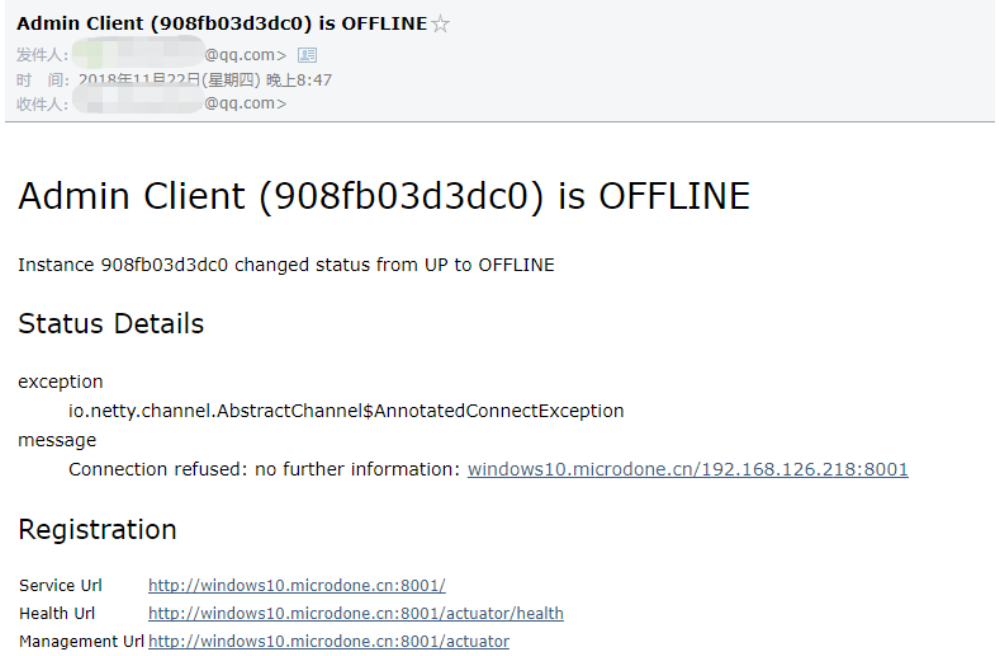
增加了邮件发送的 starter 包。

配置文件

```
spring.mail.host=smtp.qq.com
spring.mail.username=xxx@qq.com
spring.mail.password=xxx
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.smtp.starttls.required=true
spring.boot.admin.notify.mail.from=yyyy@qq.com
spring.boot.admin.notify.mail.to=zzz@qq.com
```

在配置文件中添加邮件发送相关信息：邮件的发送者、接受者、协议、移动授权码等。

配置完成后，重新启动项目 spring-boot-admin-server，这样 Admin Server 就具备了邮件告警的功能，默认情况下 Admin Server 对 Eureka 中的服务上下线都进行了监控，当服务上下线的时候我们就会收到如下邮件：



当然这只是最基本的邮件监控，在实际的使用过程中，需要根据情况对邮件告警内容进行自定义，比如监控堆内存的使用情况，当到达一定比例的时候进行告警等。

总结

Spring Boot Admin 解决了我们对大规模 Spring Boot 应用监控的需求，Spring Boot Admin 充分利用了 Actuator 开放的相关接口，采用优秀的图形界面将这些信息进行了展示，方便我们更加直观的查看集群中应用的状态。Spring Boot Admin 不仅可以监控单个 Spring Boot 应用，也可以结合 Spring Cloud 监控注册到服务中心的所有应用状态，再结合报警系统的使用就可以随时感知到应用的状态变化。在实际工作中 Spring Boot Admin 是我们在后期运营中频繁用到的一个组件，应该作为重点关注。

> 点击这里下载源码 (https://github.com/ityouknow/spring-boot-leaning/tree/gitbook_column2.0)。

(/gitchat/column/5b86228ce15aa17d68b5b55a/topic/5c25965e4fcd483b0265b15c)(/gitcha