链滴

Spring Boot Admin 排坑指南

作者：liumapp

原文链接：https://ld246.com/article/1564813581585

来源网站：链滴

Spring Boot Admin 1.x其简陋的页面让人不忍直视，但更新到2.x系列后，像脱胎换骨一般好用

这篇博客记录我个人在使用Spring Boot Admin过程中遇到过的坑，每个坑位都会附上详细的填坑办法

环境参数:

- Spring Boot 2.x
- Spring Boot Admin 2.x
- JDK1.8+
- CentOS

服务直接注册失败

常见的注册失败问题可以分为以下两种

- Spring Boot Admin服务端与客户端不在同一台服务器上
- 提示安全校验不通过

第一种问题的解决办法:

必须在客户端配置boot.admin.client.instance.service-url属性，让Spring Boot Admin服务端可以过网络获取客户端的数据（否则默认会通过主机名去获取）

```
boot:
  admin:
    client:
      url: ${your spring boot admin url}
      username: ${your spring boot admin username}
      password: ${your spring boot admin password}
      instance:
        prefer-ip: true
        service-url: ${your spring boot client url}
```

第二种问题的解决办法：

首先，安全检验问题，其实就是现在服务端配置账号密码，然后客户端在注册的时候提供账号密码进登录来完成校验

这个过程的实现，作为Spring全家桶项目，推荐使用Spring Security来解决，所以如果出现校验失败那多半是Spring Security的配置出现问题

接下来介绍如何分别配置服务端与客户端来处理这个问题

服务端配置

通过maven加载Spring Security依赖

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

设置服务端的用户名和密码（客户端来注册时使用此账号密码进行登录）

```
spring:
  security:
    user:
      name: liumapp
      password: superliumapp
```

编写Spring Security配置类

```
import de.codecentric.boot.admin.server.config.AdminServerProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
rerAdapter;
import org.springframework.security.web.authentication.SavedRequestAwareAuthenticationS
ccessHandler;
import org.springframework.security.web.csrf.CookieCsrfTokenRepository;

/**
 * file SecuritySecureConfig.java
 * author liumapp
 * github https://github.com/liumapp
 * email liumapp.com@gmail.com
 * homepage http://www.liumapp.com
 * date 2018/11/29
 */
@Configuration
public class SecuritySecureConfig extends WebSecurityConfigurerAdapter {
    private final String adminContextPath;

    public SecuritySecureConfig(AdminServerProperties adminServerProperties) {
        this.adminContextPath = adminServerProperties.getContextPath();
    }

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

        http.authorizeRequests()
                .antMatchers(adminContextPath + "/assets/**").permitAll()
                .antMatchers(adminContextPath + "/login").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin().loginPage(adminContextPath + "/login").successHandler(successHandl
r).and()
                .logout().logoutUrl(adminContextPath + "/logout").and()
                .httpBasic().and()
                .csrf()
                .csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFalse())
```

原文链接：Spring Boot Admin 排坑指南

```
                .ignoringAntMatchers(
                        adminContextPath + "/instances",
                        adminContextPath + "/actuator/**"
                );
        // @formatter:on
    }
}
```

上面这段代码，需要大家注意的就一个AdminServerProperties类，通过浏览它的部分源代码:

```
@ConfigurationProperties("spring.boot.admin")
public class AdminServerProperties {
    /**
     * The context-path prefixes the path where the Admin Servers statics assets and api should
be
     * served. Relative to the Dispatcher-Servlet.
     */
    private String contextPath = "";

    /**
     * The metadata keys which should be sanitized when serializing to json
     */
    private String[] metadataKeysToSanitize = new String[]{".*password$", ".*secret$", ".*key$",
.*$token$", ".*credentials.*", ".*vcap_services$"};

    /**
     * For Spring Boot 2.x applications the endpoints should be discovered automatically using
he actuator links.
     * For Spring Boot 1.x applications SBA probes for the specified endpoints using an OPTIO
S request.
     * If the path differs from the id you can specify this as id:path (e.g. health:ping).
     */
    private String[] probedEndpoints = {"health", "env", "metrics", "httptrace:trace", "httptrace",
"threaddump:dump", "threaddump", "jolokia", "info", "logfile", "refresh", "flyway", "liquibase",
heapdump", "loggers", "auditevents", "mappings", "scheduledtasks", "configprops", "caches",
beans"};

    //以下省略...

}
```

可以发现AdminServerProperties定义了Spring Boot Admin的配置属性，登录自然也是其中之一，以我们在编写Spring Security配置类的时候，务必要引入AdminServerProperties

到这里，Spring Boot Admin服务端对于Spring Security的配置便结束了，接下来让我们开始客户端Security配置

客户端配置

首先对于客户端，我们除了Spring Boot Admin Client依赖外，还需要额外引入 Spring Security依赖:

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

```
    <version>2.0.2</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

在此基础上通过编写客户端application.yml配置文件来设置账号密码

```
spring:
  boot:
    admin:
      client:
        url: ${your sba server url}
        username: ${your sba username}
        password: ${your sba password}
        instance:
          service-base-url: ${your client url}
```

接下来对Client端的Spring Security做配置，允许Server端读取actuator暴露的数据

添加一个配置类:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfig
rerAdapter;

@Configuration
public class SecurityPermitAllConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests().anyRequest().permitAll()
                .and().csrf().disable();
    }
}
```

到此，因为安全验证而不能注册成功的问题便可以解决

注册成功但无法显示日志

这个问题产生原因有两种

- 客户端日志没有以文件形式存储下来
- 客户端容器化部署后，日志文件没有映射到宿主机磁盘上

针对第一种情况，解决办法比较简单，将系统产生的日志以文件形式保存即可:

```
logging:
  file: ./log/client.log
  pattern:
    file: "%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} %clr(%5p) %clr(${PID}){magenta} %clr(---)
faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan} %clr(:){faint} %m%n%wEx"
```

第二种情况较为复杂，首先要分清除是用什么工具来部署容器的，但一般而言直接通过文件映射即可

这里以docker为例，在docker内通过设置volumes来映射日志文件

```
volumes:
  - ./log:/client/log/
```

注册成功但信息显示不全

偶尔也会遇到这种情况：Spring Boot Admin客户端注册服务端是成功的，但是统计页面显示的数据少（可能只有日志这一栏）

造成这种问题的原因在于：我们没有开放客户端的actuator接口地址给服务端访问

那么解决办法也很简单，允许服务端访问actuator即可

首先我们需要确保项目有actuator依赖(一般来说，spring-boot-admin-starter-client本身就包含这依赖，所以不需要额外引入)：

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

然后打开actuator的端口，在client端的配置文件中增加以下内容：

```
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

同时考虑到client与server域名存在不一样的情况，顺便把跨域也解决掉，增加跨域配置类：

```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

/**
 * @author liumapp
 * @file CorsConfig.java
 * @email liumapp.com@gmail.com
 * @homepage http://www.liumapp.com
 * @date 2018/8/11
 */
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowCredentials(true)
                .allowedHeaders("*")
                .allowedOrigins("*")
```

```
                .allowedMethods("*");
    }
}
```

问题即可解决