Spring Cloud Gateway 实现网关 Token 鉴权

1.添加依赖:

在 pom.xml 文件中添加 Spring Cloud Gateway 的依赖:

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

2.配置路由和过滤器:

在 application.yml 文件中添加路由配置和自定义过滤器的配置:

```
spring:
  cloud:
    gateway:
      routes:
      id: auth-service
        uri: http://auth-service
        predicates:
        Path=/auth/**
      id: other-service
        uri: http://other-service
        predicates:
        Path=/other/**
      globalFilters:
      name: TokenAuthFilter
        args:
          tokenHeader: Authorization
          excludedUrls:
```

```
        - "/auth/login"
        - "/auth/register"
```

上述配置中,我们定义了两个路由:一个是 auth-service,另一个是 other-service。同时我们配置了一个全局过滤器 TokenAuthFilter,它负责对请求进行 Token 鉴权。

3.实现 TokenAuthFilter:

创建自定义的 TokenAuthFilter 类,实现 GatewayFilter 接口:

```
@Component
public class TokenAuthFilter implements GatewayFilter, Ordered {

    private final String tokenHeader;
    private final List<String> excludedUrls;

    public
TokenAuthFilter(@Value("${spring.cloud.gateway.globalFilters.TokenAuthFilter.tokenHeader}") String tokenHeader,
@Value("${spring.cloud.gateway.globalFilters.TokenAuthFilter.excludedUrls}") List<String> excludedUrls) {
        this.tokenHeader = tokenHeader;
        this.excludedUrls = excludedUrls;
    }

    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
GatewayFilterChain chain) {

        String path = exchange.getRequest().getURI().getPath();
        // 检查是否在白名单中
        if (excludedUrls.contains(path)) {
            return chain.filter(exchange);
        }

        // 获取请求头中的 Token
        String token =
exchange.getRequest().getHeaders().getFirst(tokenHeader);
```

```
        // 验证 Token
        if (isValidToken(token)) {
            return chain.filter(exchange);
        } else {
exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
        return exchange.getResponse().setComplete();
        }

    }

    private boolean isValidToken(String token) {
        // 实现 Token 验证逻辑
        // 可以调用认证服务进行 Token 验证
        return token != null && token.equals("valid_token");
    }

    @Override
    public int getOrder() {
        return 10;
    }

}
```

在上述实现中,我们首先检查当前请求的路径是否在白名单中,如果在则直接放行。否则,我们从请求头中获取 Token,并调用 isValidToken 方法进行 Token 的验证。如果 Token 验证通过,则继续执行后续的过滤器链;否则,返回 401 Unauthorized 错误。

4.添加错误处理:

为了统一处理 401 Unauthorized 错误,我们可以添加一个全局异常处理器:

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(WebFilterException.class)
    public ResponseEntity<ErrorResponse>
handleWebFilterException(WebFilterException ex) {

        ErrorResponse errorResponse = new
```

```
ErrorResponse(HttpStatus.UNAUTHORIZED.value(), ”Unauthorized”);

        return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(errorResponse)
;

    }

    // 其他异常处理

}
```

在上述异常处理器中,我们捕获 WebFilterException 异常,并返回一个标准的 JSON 格式错误响应。

5.测试和部署:

完成以上配置后,我们可以启动 Spring Cloud Gateway 服务,并在客户端请求微服务接口时验证 Token 鉴权和白名单规则是否生效。如果一切正常,就可以将 Spring Cloud Gateway 服务部署到生产环境中了。

总的来说,使用 Spring Cloud Gateway 实现 Token 鉴权的核心步骤包括:配置路由和过滤器、实现自定义的 TokenAuthFilter、添加全局异常处理。通过这种方式,我们可以在微服务架构中统一管理 Token 鉴权逻辑,提高系统的安全性和可维护性。

原文链接: https://juejin.cn/post/7375351908896342035