开箱即用！看看人家的微服务权限解决方案，那叫一个优雅！

原创 梦想de星空 macrozheng 2021-08-24 10:32

收录于合集
#Spring Cloud学习教程
26个

> 记得之前写过一篇文章微服务权限终极解决方案，**Spring Cloud Gateway + Oauth2** 实现统一认证和鉴权！，提供了Spring Cloud中的权限解决方案，其实一开始整合的时候我一直玩不转，又是查资料又是看源码，最终才成功了。最近尝试了下Sa-Token提供的微服务权限解决方案，用起来感觉很优雅，推荐给大家！

前置知识

> 我们将采用Nacos作为注册中心，Gateway作为网关，使用Sa-Token提供的微服务权限解决方案，此方案是基于之前的解决方案改造的，对这些技术不了解的朋友可以看下下面的文章。

- **Spring Cloud Gateway：新一代API网关服务**
- **Spring Cloud Alibaba：Nacos 作为注册中心和配置中心使用**
- 微服务权限终极解决方案，**Spring Cloud Gateway + Oauth2** 实现统一认证和鉴权！
- **Sa-Token**使用教程

应用架构

> 还是和之前方案差不多的思路，认证服务负责登录处理，网关负责登录认证和权限认证，其他API服务负责处理自己的业务逻辑。为了能在多个服务中共享Sa-Token的Session，所有服务都需要集成Sa-Token和Redis。

- micro-sa-token-common：通用工具包，其他服务公用的用户类 `UserDTO` 和通用返回结果类 `CommonResult` 被抽取到了这里。
- micro-sa-token-gateway：网关服务，负责请求转发、登录认证和权限认证。

- micro-sa-token-auth：认证服务，仅包含一个登录接口，调用Sa-Token的API实现。
- micro-sa-token-api：受保护的API服务，用户通过网关鉴权通过后可以访问该服务。

方案实现

> 接下来实现下这套解决方案，依次搭建网关服务、认证服务和API服务。

micro-sa-token-gateway

> 我们首先来搭建下网关服务，它将负责整个微服务的登录认证和权限认证。

- 除了通用的Gateway依赖，我们还需要在 `pom.xml` 中添加如下依赖，包括Sa-Token的Reactor响应式依赖，整合Redis实现分布式Session的依赖以及我们的 `micro-sa-token-common` 依赖；

```
<dependencies>
    <!-- Sa-Token 权限认证（Reactor响应式集成） -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-reactor-spring-boot-starter</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- Sa-Token 整合 Redis (使用jackson序列化方式) -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-dao-redis-jackson</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- 提供Redis连接池 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-pool2</artifactId>
    </dependency>
    <!-- micro-sa-token通用依赖 -->
    <dependency>
        <groupId>com.macro.cloud</groupId>
        <artifactId>micro-sa-token-common</artifactId>
```

```
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

- 接下来修改配置文件 `application.yml` ，添加Redis配置和Sa-Token的配置，如果你看过之前那篇**Sa-Token使用教程**的话，基本就知道这些配置的作用了；

```
spring:
  redis:
    database: 0
    port: 6379
    host: localhost
    password:

# Sa-Token配置
sa-token:
  # token名称 (同时也是cookie名称)
  token-name: Authorization
  # token有效期，单位秒，-1代表永不过期
  timeout: 2592000
  # token临时有效期 (指定时间内无操作就视为token过期)，单位秒
  activity-timeout: -1
  # 是否允许同一账号并发登录 (为false时新登录挤掉旧登录)
  is-concurrent: true
  # 在多人登录同一账号时，是否共用一个token (为false时每次登录新建一个token)
  is-share: false
  # token风格
  token-style: uuid
  # 是否输出操作日志
  is-log: false
  # 是否从cookie中读取token
  is-read-cookie: false
  # 是否从head中读取token
  is-read-head: true
```

- 添加Sa-Token的配置类 `SaTokenConfig` ，注入一个过滤器用于登录认证和权限认证，在 `setAuth` 方法中添加路由规则，在 `setError` 方法中添加鉴权失败的回调处理；

```
@Configuration
```

```
public class SaTokenConfig {
    /**
     * 注册Sa-Token全局过滤器
     */
    @Bean
    public SaReactorFilter getSaReactorFilter() {
        return new SaReactorFilter()
                // 拦截地址
                .addInclude("/**")
                // 开放地址
                .addExclude("/favicon.ico")
                // 鉴权方法：每次访问进入
                .setAuth(r -> {
                    // 登录认证：除登录接口都需要认证
                    SaRouter.match("/**", "/auth/user/login", StpUtil::checkLogin);
                    // 权限认证：不同接口访问权限不同
                    SaRouter.match("/api/test/hello", () -> StpUtil.checkPermission("api:test:hel
                    SaRouter.match("/api/user/info", () -> StpUtil.checkPermission("api:user:info
                })
                // setAuth方法异常处理
                .setError(e -> {
                    // 设置错误返回格式为JSON
                    ServerWebExchange exchange = SaReactorSyncHolder.getContent();
                    exchange.getResponse().getHeaders().set("Content-Type", "application/json; ch
                    return SaResult.error(e.getMessage());
                });
    }
}
```

- 扩展下Sa-Token提供的 `StpInterface` 接口，用于获取用户的权限，我们在用户登录以后会把用户信息存到Session中去，权限信息也会在里面，所以权限码只要从Session中获取即可。

```
/**
 * 自定义权限验证接口扩展
 */
@Component
public class StpInterfaceImpl implements StpInterface {

    @Override
    public List<String> getPermissionList(Object loginId, String loginType) {
```

```
        // 返回此 loginId 拥有的权限码列表
        UserDTO userDTO = (UserDTO) StpUtil.getSession().get("userInfo");
        return userDTO.getPermissionList();
    }

    @Override
    public List<String> getRoleList(Object loginId, String loginType) {
        // 返回此 loginId 拥有的角色码列表
        return null;
    }

}
```

micro-sa-token-auth

> 接下来我们来搭建下认证服务，只要集成Sa-Token并实现登录接口即可，非常简单。

- 首先在 `pom.xml` 中添加相关依赖，包括Sa-Token的SpringBoot依赖、整合Redis实现分布式Session的依赖以及我们的 `micro-sa-token-common` 依赖；

```
<dependencies>
    <!-- Sa-Token 权限认证 -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-spring-boot-starter</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- Sa-Token 整合 Redis （使用jackson序列化方式） -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-dao-redis-jackson</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- 提供Redis连接池 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-pool2</artifactId>
    </dependency>
    <!-- micro-sa-token通用依赖 -->
```

```
    <dependency>
        <groupId>com.macro.cloud</groupId>
        <artifactId>micro-sa-token-common</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

- 接下来修改配置文件 `application.yml` ，照抄之前网关的配置即可；

```
spring:
  redis:
    database: 0
    port: 6379
    host: localhost
    password:

# Sa-Token配置
sa-token:
  # token名称 (同时也是cookie名称)
  token-name: Authorization
  # token有效期，单位秒，-1代表永不过期
  timeout: 2592000
  # token临时有效期 (指定时间内无操作就视为token过期)，单位秒
  activity-timeout: -1
  # 是否允许同一账号并发登录 (为false时新登录挤掉旧登录)
  is-concurrent: true
  # 在多人登录同一账号时，是否共用一个token (为false时每次登录新建一个token)
  is-share: false
  # token风格
  token-style: uuid
  # 是否输出操作日志
  is-log: false
  # 是否从cookie中读取token
  is-read-cookie: false
  # 是否从head中读取token
  is-read-head: true
```

- 在 `UserController` 中定义好登录接口，登录成功后返回Token，具体实现在 `UserServiceImpl` 类中；

```
/**
 * 自定义Oauth2获取令牌接口
 * Created by macro on 2020/7/17.
 */
@RestController
@RequestMapping("/user")
public class UserController {
    @Autowired
    private UserServiceImpl userService;

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public CommonResult login(@RequestParam String username, @RequestParam String password) {
        SaTokenInfo saTokenInfo = userService.login(username, password);
        if (saTokenInfo == null) {
            return CommonResult.validateFailed("用户名或密码错误");
        }
        Map<String, String> tokenMap = new HashMap<>();
        tokenMap.put("token", saTokenInfo.getTokenValue());
        tokenMap.put("tokenHead", saTokenInfo.getTokenName());
        return CommonResult.success(tokenMap);
    }
}
```

- 在 `UserServiceImpl` 中添加登录的具体逻辑，首先验证密码，密码校验成功后，通知下Sa-Token登录的用户ID，然后把用户信息直接存储到Session中去；

```
/**
 * 用户管理业务类
 * Created by macro on 2020/6/19.
 */
@Service
public class UserServiceImpl{

    private List<UserDTO> userList;

    public SaTokenInfo login(String username, String password) {
        SaTokenInfo saTokenInfo = null;
        UserDTO userDTO = loadUserByUsername(username);
        if (userDTO == null) {
            return null;
        }
        if (!SaSecureUtil.md5(password).equals(userDTO.getPassword())) {
```

```
            return null;
        }
        // 密码校验成功后登录，一行代码实现登录
        StpUtil.login(userDTO.getId());
        // 将用户信息存储到Session中
        StpUtil.getSession().set("userInfo",userDTO);
        // 获取当前登录用户Token信息
        saTokenInfo = StpUtil.getTokenInfo();
        return saTokenInfo;
    }
}
```

- 这里有一点需要提醒下，Sa-Token的Session并不是我们平时理解的HttpSession，而是它自己实现的类似Session的机制。

micro-sa-token-api

> 接下来我们来搭建一个受保护的API服务，实现获取登录用户信息的接口和需要特殊权限才能访问的测试接口。

- 首先在 `pom.xml` 中添加相关依赖，和上面的 `micro-sa-token-auth` 一样；

```
<dependencies>
    <!-- Sa-Token 权限认证 -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-spring-boot-starter</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- Sa-Token 整合 Redis （使用jackson序列化方式） -->
    <dependency>
        <groupId>cn.dev33</groupId>
        <artifactId>sa-token-dao-redis-jackson</artifactId>
        <version>1.24.0</version>
    </dependency>
    <!-- 提供Redis连接池 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-pool2</artifactId>
    </dependency>
```

```
    </dependency>
    <!-- micro-sa-token通用依赖 -->
    <dependency>
        <groupId>com.macro.cloud</groupId>
        <artifactId>micro-sa-token-common</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

- 接下来修改配置文件 `application.yml` ，照抄之前网关的配置即可；

```
spring:
  redis:
    database: 0
    port: 6379
    host: localhost
    password:

# Sa-Token配置
sa-token:
  # token名称 (同时也是cookie名称)
  token-name: Authorization
  # token有效期，单位秒，-1代表永不过期
  timeout: 2592000
  # token临时有效期 (指定时间内无操作就视为token过期)，单位秒
  activity-timeout: -1
  # 是否允许同一账号并发登录 (为false时新登录挤掉旧登录)
  is-concurrent: true
  # 在多人登录同一账号时，是否共用一个token (为false时每次登录新建一个token)
  is-share: false
  # token风格
  token-style: uuid
  # 是否输出操作日志
  is-log: false
  # 是否从cookie中读取token
  is-read-cookie: false
  # 是否从head中读取token
  is-read-head: true
```

- 添加获取用户信息的接口，由于使用了Redis实现分布式Session，直接从Session中获取即可，是不是非常简单！

```
/**
 * 获取登录用户信息接口
 * Created by macro on 2020/6/19.
 */
@RestController
@RequestMapping("/user")
public class UserController{

    @GetMapping("/info")
    public CommonResult<UserDTO> userInfo() {
        UserDTO userDTO = (UserDTO) StpUtil.getSession().get("userInfo");
        return CommonResult.success(userDTO);
    }

}
```

- 添加需要 `api:test:hello` 权限访问的测试接口，预置的 `admin` 用户拥有该权限，而 `macro` 用户是没有的。

```
/**
 * 测试接口
 * Created by macro on 2020/6/19.
 */
@RestController
@RequestMapping("/test")
public class TestController {

    @GetMapping("/hello")
    public CommonResult hello() {
        return CommonResult.success("Hello World.");
    }

}
```

功能演示

> 三个服务搭建完成后，我们用Postman来演示下微服务的认证授权功能。

- 首先启动Nacos和Redis服务，然后再启动 `micro-sa-token-gateway` 、 `micro-sa-token-auth` 和 `micro-sa-token-api` 服务，启动顺序无所谓；

- 直接通过网关访问登录接口获取Token，访问地址：http://localhost:9201/auth/user/login

- 通过网关访问API服务， 不带`Token` 调用获取用户信息的接口，无法正常访问，访问地址：http://localhost:9201/api/user/info

- 通过网关访问API服务， `带Token` 调用获取用户信息的接口，可以正常访问；

- 通过网关访问API服务，使用 `macro` 用户访问需 `api:test:hello` 权限的测试接口，无法正常访问，访问地址：http://localhost:9201/api/test/hello

- 登录切换为 `admin` 用户，该用户具有 `api:test:hello` 权限；

- 通过网关访问API服务，使用 `admin` 用户访问测试接口，可以正常访问。

总结

对比之前使用Spring Security的微服务权限解决方案，Sa-Token的解决方案更简单、更优雅。使用Security我们需要定义鉴权管理器、分别处理未认证和未授权的情况、还要自己定义认证和资源服务器配置，使用非常繁琐。而使用Sa-Token，只要在网关上配置过滤器实现认证和授权，然后调用API实现登录及权限分配即可。具体区别可以参考下图。

参考资料

官方文档：http://sa-token.dev33.cn/

项目源码地址

https://github.com/macrozheng/springcloud-learning/tree/master/micro-sa-token

微信**8.0**将好友放开到了一万，小伙伴可以加我大号了，先到先得，再满就真没了

扫描下方二维码即可加我微信啦， **2021**，抱团取暖，一起牛逼。

推荐阅读

- 在国企里面当程序猿是一种怎样的体验**?**
- 再见**Spring Security**！推荐一款功能强大的权限认证框架，用起来够优雅！
- **Lombok**！代码简洁神器还是代码“亚健康”元凶？
- 为何程序员们没事总爱戴个耳机，看完恍然大悟**...**
- **woc,**又一个大佬辞职了**.....**
- 基于**SpringBoot**的文件在线预览神器，可支持**99%**常用文件的在线预览！
- **40K+Star**！**Mall**电商实战项目开源回忆录！
- **mall-swarm** 微服务电商项目发布重大更新，打造**Spring Cloud**最佳实践！

macrozheng

专注Java技术分享，解析优质开源项目。涵盖SpringBoot、SpringCloud、Docker、K8S...

240篇原创内容

公众号

阅读原文

喜欢此内容的人还喜欢

项目中到底该不该用Lombok?

macrozheng