链滴

SpringCloud Alibaba 微服务实战十七 - JWT 认证

作者：jianzh5

原文链接：https://ld246.com/article/1594256410116

来源网站：链滴

概述

在 OAuth2 体系中认证通过后返回的令牌信息分为两大类：**不透明令牌（opaque tokens）** 和 **透令牌（not opaque tokens）。**

不透明令牌 就是一种无可读性的令牌，一般来说就是一段普通的 UUID 字符串。使用不透明令牌会低系统性能和可用性，并且增加延迟，因为资源服务不知道这个令牌是什么，代表谁，需要调用认证务器获取用户信息接口，如下就是我们在资源服务器中的配置，需要指明认证服务器的接口地址。

```
security:
  oauth2:
    resource:
      user-info-uri: http://localhost:5000/user/current/get
      id: account-service
```

透明令牌的典型代表就是 JWT 了，用户信息保存在 JWT 字符串中，资源服务器自己可以解析令牌不需要去认证服务器校验令牌。

之前的章节中我们是使用了不透明令牌access_token，但考虑到在微服务体系中这种中心化的授权服会成为瓶颈，本章我们就使用jwt来替换之前的access_token，专（zhuang）业（bi）点就叫去中心。

jwt 是什么

Json web token (JWT), 是为了在网络应用环境间传递声明而执行的一种基于JSON的开放标准（RFC 519)。该token被设计为紧凑且安全的，特别适用于分布式站点的单点登录（SSO）场景。JWT的声一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源也可以增加一些额外的其它业务逻辑所必须的声明信息，该token也可直接被用于认证，也可被加密。

简单点说就是一种固定格式的字符串，通常是加密的；

它由三部分组成，头部、载荷与签名，这三个部分都是json格式。

- Header 头部：JSON方式描述JWT基本信息，如类型和签名算法。使用Base64编码为字符串
- Payload 载荷: JSON方式描述JWT信息，除了标准定义的，还可以添加自定义的信息。同样使用Bas64编码为字符串。

1. iss: 签发者
2. sub: 用户
3. aud: 接收方
4. exp(expires): unix时间戳描述的过期时间

5. iat(issued at): unix时间戳描述的签发时间

● Signature 签名: 将前两个字符串用 . 连接后，使用头部定义的加密算法，利用密钥进行签名，并将名信息附在最后。

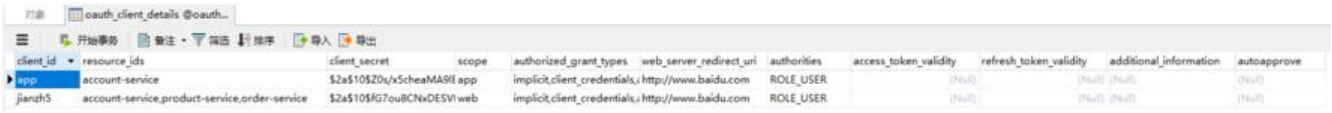
JWT可以使用对称的加密密钥，但更安全的是使用非对称的密钥，本篇文章使用的是对称加密。

代码修改

数据库

原来使用access_token的时候我们建立了7张oauth2相关的数据表

使用jwt的话只需要在数据库存储一下client信息即可，所以我们只需要保留数据表oauth_client_details。



其他数据表已不再需要，大家可以删除。

认证服务 AuthorizationServerConfig

1. 修改 AuthorizationServerConfig 中TokenStore的相关配置

```
@Bean
public TokenStore tokenStore() {
    //return new JdbcTokenStore(dataSource);
        return new JwtTokenStore(jwtTokenEnhancer());
}


/**
 * JwtAccessTokenConverter
 * TokenEnhancer的子类，帮助程序在JWT编码的令牌值和OAuth身份验证信息之间进行转换。
 */
@Bean
public JwtAccessTokenConverter jwtTokenEnhancer(){
        JwtAccessTokenConverter converter = new JwtAccessTokenConverter();
        // 设置对称签名
        converter.setSigningKey("javadaily");
        return converter;
}
```

之前我们是将access_token存入数据库，使用jwt后不再需要存入数据库，所以我们需要修改存储方。

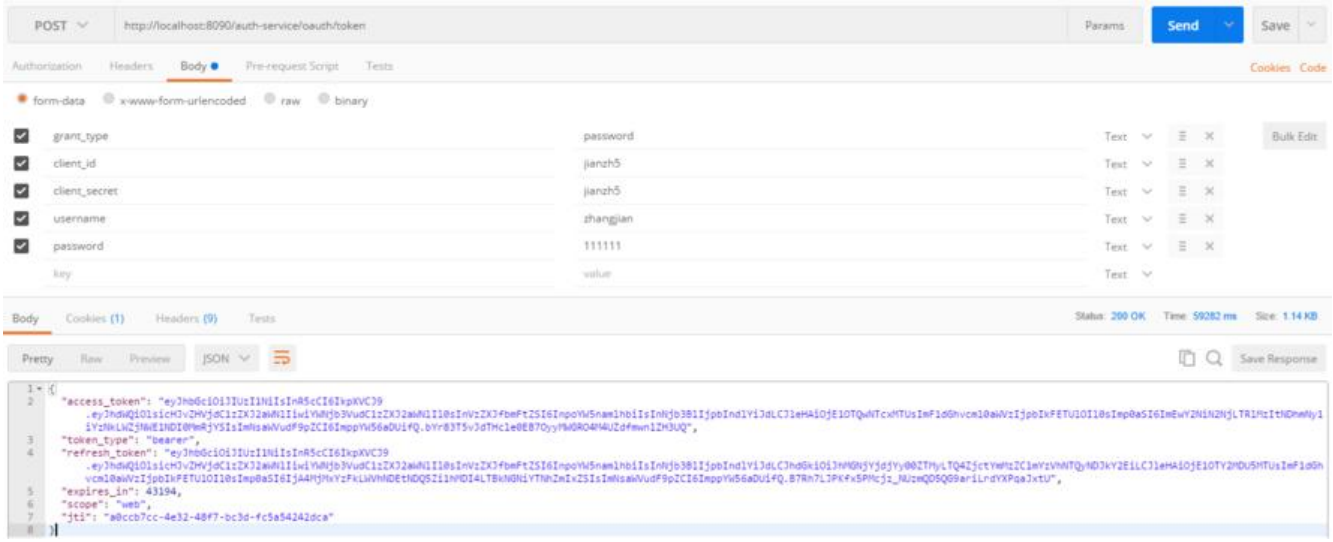
jwt需要使用加密算法对信息签名，这里我们先使用 **对称秘钥** （javadaily）来签署我们的令牌，对称钥当然这也以为着资源服务器也需要使用相同的秘钥。

2. 修改 configure(AuthorizationServerEndpointsConfigurer endpoints) 方法，配置jwt

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
        //如果需要使用refresh_token模式则需要注入userDetailService
        endpoints.authenticationManager(this.authenticationManager)
                        .userDetailsService(userDetailService)
                        .tokenStore(tokenStore())
                        .accessTokenConverter(jwtTokenEnhancer());
}
```

这里主要是注入accessTokenConverter，即上面配置的token转换器。

经过上面的配置，认证服务器已经可以帮我们生成jwt token了，这里我们先使用Postman调用一下看看生成的jwt token。



从上图看出已经正常生成jwt token，我们可以将生成的jwt token拿到https://jwt.io/网站上进行解。

如果大家对生成jwt token的逻辑不是很了解，可以在 DefaultTokenServices#createAccessToken(Auth2Authentication authentication) 和 JwtAccessTokenConverter#enhance(OAuth2AccessToken accessToken, OAuth2Authentication authentication)上打个断点，观察代码执行的效果。

3. 删除认证服务器提供给资源服务器获取用户信息的接口

```
/**
 * 获取授权的用户信息
 * @param principal 当前用户
 * @return 授权信息
 */@GetMapping("current/get")
public Principal user(Principal principal){
        return principal;
}
```

用了透明令牌jwt token后资源服务器可以直接解析验证token，不再需要调用认证服务器接口，所以

处可以直接删除。

4. 修改jwt token有效期（可选）

jwt token的默认有效期为12小时，refresh token的有效期为30天，如果要修改默认时间可以注入 DefaultTokenServices 并修改有效时间。

```
@Primary
@Bean
public DefaultTokenServices tokenServices(){
        DefaultTokenServices tokenServices = new DefaultTokenServices();
        tokenServices.setTokenEnhancer(jwtTokenEnhancer());
        tokenServices.setTokenStore(tokenStore());
        tokenServices.setSupportRefreshToken(true);
        //设置token有效期，默认12小时，此处修改为6小时   21600
        tokenServices.setAccessTokenValiditySeconds(60 * 60 * 6);
        //设置refresh_token的有效期，默认30天，此处修改为7天
        tokenServices.setRefreshTokenValiditySeconds(60 * 60 * 24 * 7);
        return tokenServices;
}
```

然后在 configure() 方法中添加tokenServices

```
@Override
public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {
                //如果需要使用refresh_token模式则需要注入userDetailService
        endpoints.authenticationManager(this.authenticationManager)
                        .userDetailsService(userDetailService)
            //注入自定义的tokenservice，如果不使用自定义的tokenService那么就需要将tokenServc
里的配置移到这里
                        .tokenServices(tokenServices());
}
```

资源服务器 ResourceServerConfig

1. 删除资源服务器中配置认证服务器的接口属性　user-info-uri

```
security:
    oauth2:
        resource:
            id: account-service
```

2. 注入TokenStore 和 JwtAccessTokenConverter

```
@Bean
public TokenStore tokenStore() {
        return new JwtTokenStore(jwtTokenEnhancer());
}

@Bean
public JwtAccessTokenConverter jwtTokenEnhancer(){
        JwtAccessTokenConverter jwtTokenEnhancer = new JwtAccessTokenConverter();
        jwtTokenEnhancer.setSigningKey("javadaily");
        return jwtTokenEnhancer;
```

```
}
```

注意：对称加密算法需要跟认证服务器秘钥保持一致，当然这里可以提取到配置文件中。

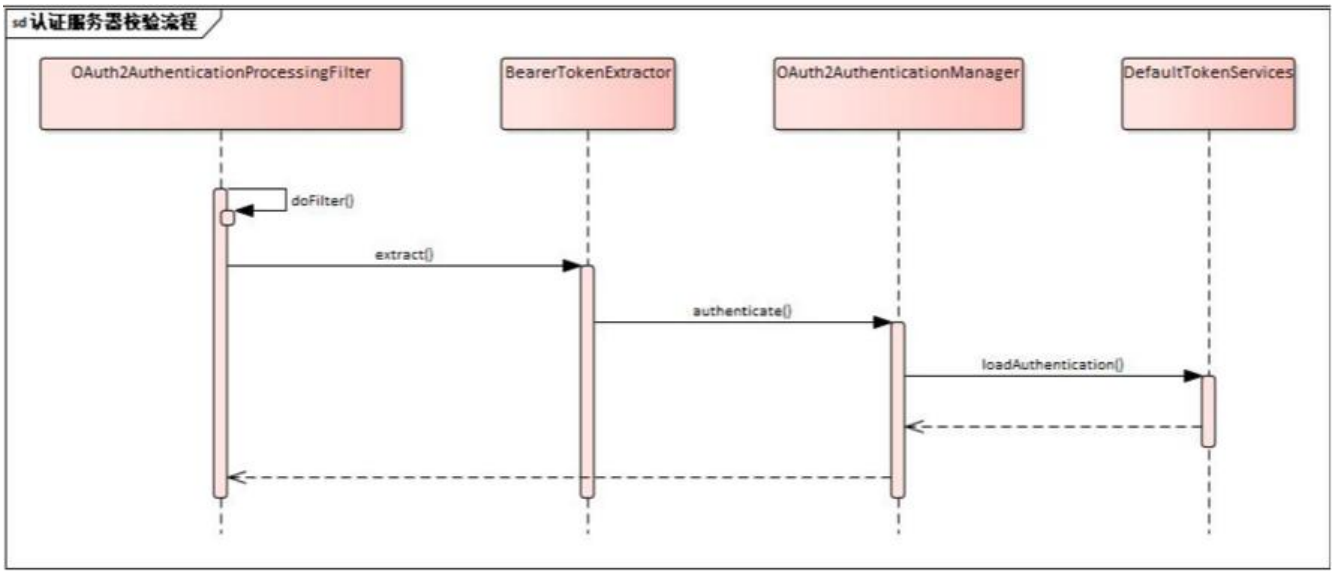
3. 添加 configure(ResourceServerSecurityConfigurer resources) 方法，加入token相关配置

```
@Override
public void configure(ResourceServerSecurityConfigurer resources) throws Exception {
        resources.resourceId(resourceId)
                        .tokenStore(tokenStore());
}
```

思考题：资源服务器使用jwt后从哪校验token呢?

给应用添加 @EnableResourceServer 注解后会给Spring Security的FilterChan添加一个 OAuth2AuthenticationProcessingFilter，OAuth2AuthenticationProcessingFilter 会使用 OAuth2AuthenticationManager 来验证token。

校验逻辑主体代码执行顺序如下：



建议大家在 OAuth2AuthenticationProcessingFilter#doFilter() 处打个断点体会一下校验过程。

网关 SecurityConfig

1. 创建 ReactiveJwtAuthenticationManager 从tokenStore加载 OAuth2AccessToken

由于原来的access_token是存储在数据库中，所以我们编写了 ReactiveJdbcAuthenticationManager 来从数据库获取access_token，现在使用jwt我们也需要定义一个jwt的相关类 ReactiveJwtAuthenticationManager，代码跟 ReactiveJdbcAuthenticationManager 一样，这里就不再贴出。

2. 注入TokenStore 和 JwtAccessTokenConverter

```
@Bean
public TokenStore tokenStore() {
        return new JwtTokenStore(jwtTokenEnhancer());
}

@Bean
```

```
public JwtAccessTokenConverter jwtTokenEnhancer(){
        JwtAccessTokenConverter jwtTokenEnhancer = new JwtAccessTokenConverter();
        jwtTokenEnhancer.setSigningKey("javadaily");
        return jwtTokenEnhancer;
}
```

注意：对称加密算法需要跟认证服务器秘钥保持一致，当然这里可以提取到配置文件中。

3. 修改 SecurityConfig#SecurityWebFilterChain() 方法，替换 ReactiveJdbcAuthenticationManager

```
ReactiveAuthenticationManager tokenAuthenticationManager
= new ReactiveJwtAuthenticationManager(tokenStore());
```

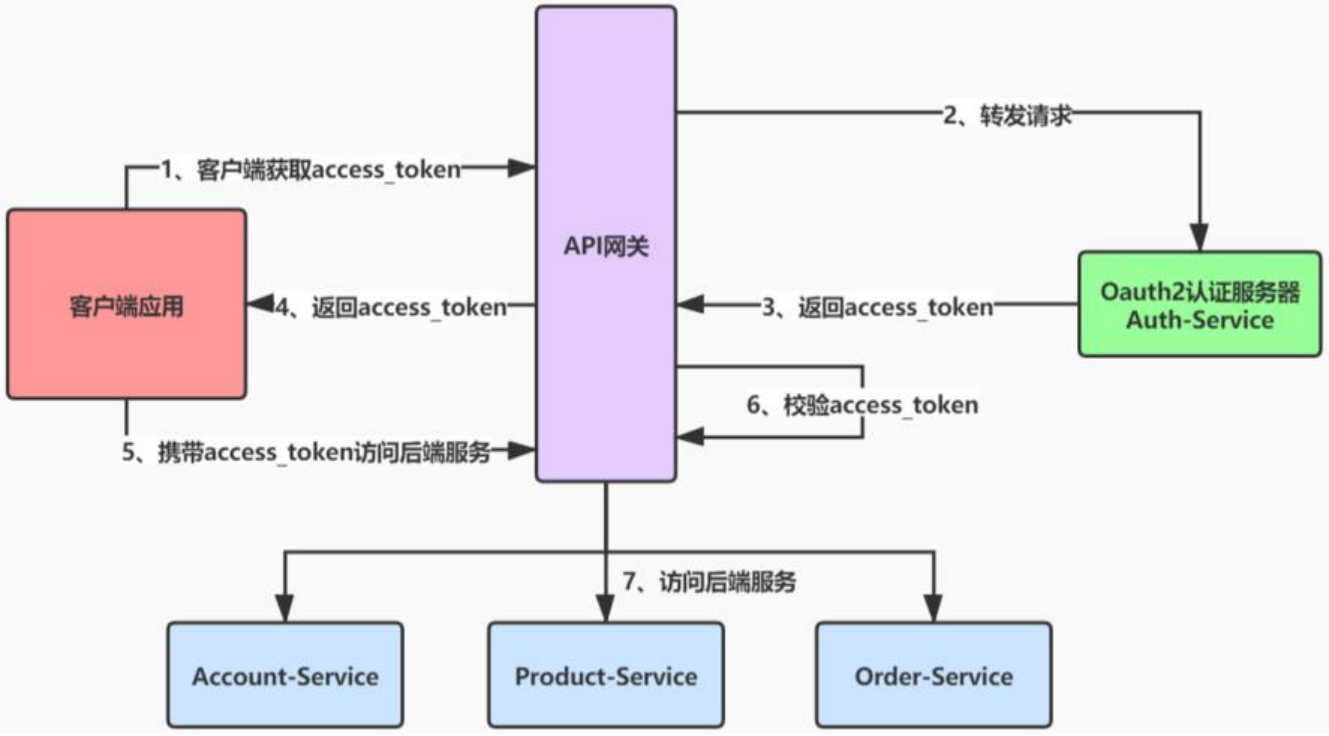
只需要将tokenStore传入构造器即可。

测试

大家自行测试。

小结

使用jwt token 和 access_token 最大的区别就是资源服务器不再需要去认证服务器校验token，提升系统整体性能，使用jwt后项目的流程架构如下：



本系列文章目前是第19篇，如果大家对之前的文章感兴趣可以移步至 http://javadaily.cn/tags/SprinCloud 查看

使用了jwt我们不仅要看到jwt的优点，也要看到它的缺点，这样我们才能根据实际场景自由选择，下是jwt最大的两个缺点：

- jwt是一次性的，一旦token被签发，那么在到期时间之前都是有效的，无法废弃。如果你中途修改

用户权限需要更新信息那就只能重新签发一个jwt，但是旧的jwt还是可以正常使用，使用旧的jwt拿到信息也就是过时的。

- jwt 包含了认证信息，一旦泄露，任何人都可以获得该令牌的所有权限。为了防止盗用，jwt的有效间不应该设置过长。