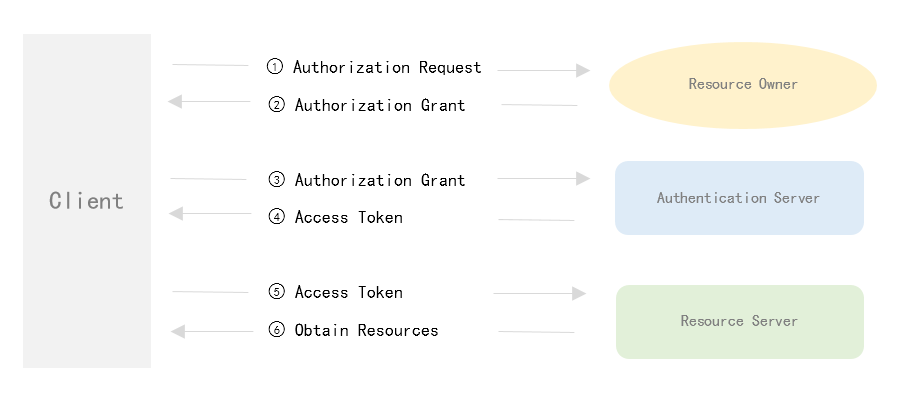
# Spring Security Oauth2 授权认证

# Oauth 简单介绍

OAuth 是一个关于授权（authorization）的开放网络标准，在全世界得到广泛应用，目前的版本是2.0版。

OAuth 是一个开放标准，允许用户让第三方应用访问该用户在某一网站上存储的私密的资源（如照片，视频，联系人列表）， 而不需要将用户名和密码提供给第三方应用。OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。 每一个令牌授权一个特定的网站在特定的时段内访问特定的资源。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息。 更多OAuth2请参考理解[OAuth 2.0](http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html)



* client：第三方应用（即App或向外提供接口）
* Resource Owner:资源所有者（即用户）
* Authentication Server:授权认证服务（发配Access Token）
* Resource Server：资源服务器（存储用户资源信息等资源）

# 搭建授权服务器

# 核心依赖

|  |
| --- |
| <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.19.RELEASE</**version**> </**parent**>  <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-security</**artifactId**>  </**dependency**>  <**dependency**>  <**groupId**>org.springframework.security.oauth</**groupId**>  <**artifactId**>spring-security-oauth2</**artifactId**>  </**dependency**>  *<!-- 将token存储在redis中 -->* <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-data-redis</**artifactId**>  </**dependency**>  </**dependencies**> |

# 配置认证服务器

|  |
| --- |
| */\*\*  \** ***@author*** *fengxuechao  \** ***@date*** *2019/3/26  \*/* @Configuration @EnableAuthorizationServer **public class** AuthServerConfig **extends** AuthorizationServerConfigurerAdapter {   **private static final** String ***DEMO\_RESOURCE\_ID*** = **"\*"**;   @Autowired  AuthenticationManager **authenticationManager**;   @Autowired  **private** DataSource **dataSource**;   @Autowired  **private** RedisConnectionFactory **redisConnectionFactory**;   */\*\*  \* 声明TokenStore实现  \*  \** ***@return*** *\*/* @Bean  **public** TokenStore tokenStore() { *// return new JdbcTokenStore(dataSource);* **return new** RedisTokenStore(**redisConnectionFactory**);  }   */\*\*  \* 声明 ClientDetails实现  \*  \** ***@return*** *\*/* @Bean  **public** JdbcClientDetailsService clientDetailsService() {  **return new** JdbcClientDetailsService(**dataSource**);  }   */\*\*  \* 第三方用户客户端详情  \* Grant Type代表当前授权的类型：  \* <p>  \* authorization\_code：传统的授权码模式<br>  \* implicit：隐式授权模式<br>  \* password：资源所有者（即用户）密码模式<br>  \* client\_credentials：客户端凭据（客户端ID以及Key）模式<br>  \* refresh\_token：获取access token时附带的用于刷新新的token模式  \* </p>  \*  \** ***@param clients*** *\** ***@throws*** *Exception  \*/* @Override  **public void** configure(ClientDetailsServiceConfigurer clients) **throws** Exception {  String[] redirectUris = **new** String[]{  **"https://www.baidu.com"**,  **"http://localhost:8081/product/1"**, **"http://127.0.0.1:8081/product/1"**,  **"http://localhost:8083/login"**, **"http://127.0.0.1:8083/login"** };  clients.jdbc(**dataSource**)  .withClient(**"client\_1"**)  .secret(**"123456"**)  .resourceIds(***DEMO\_RESOURCE\_ID***)  .redirectUris(redirectUris) *// 回调地址* .accessTokenValiditySeconds(1200) *// access\_token 有效时间* .refreshTokenValiditySeconds(50000) *// refresh\_token 有效时间* .authorizedGrantTypes(**"client\_credentials"**, **"refresh\_token"**, **"password"**, **"authorization\_code"**)  .scopes(**"all"**)  .authorities(**"client"**)  .autoApprove(**true**) *// 授权码模式:自动授权* .and()  .withClient(**"client\_2"**).secret(**"123456"**) *// 可信任的客户端凭据* .resourceIds(***DEMO\_RESOURCE\_ID***) *// 客户端能访问的资源ID* .authorizedGrantTypes(**"client\_credentials"**)  .scopes(**"read"**) *// 表示权限范围，可选项，用户授权页面时进行选择* .authorities(**"client"**) *// 授予客户端的权限* .and().build();  }   @Override  **public void** configure(AuthorizationServerEndpointsConfigurer endpoints) **throws** Exception {  *// redis保存token  // endpoints.tokenStore(new RedisTokenStore(redisConnectionFactory))  // JDBC 保存 token* endpoints.tokenStore(tokenStore());  endpoints.setClientDetailsService(clientDetailsService());  endpoints.authenticationManager(**authenticationManager**);   }   @Override  **public void** configure(AuthorizationServerSecurityConfigurer oauthServer) **throws** Exception {  *//允许表单认证* oauthServer.allowFormAuthenticationForClients();  oauthServer.tokenKeyAccess(**"permitAll()"**);  oauthServer.checkTokenAccess(**"isAuthenticated()"**);  } } |

# 配置资源服务器

|  |
| --- |
| @Order(6) @Configuration @EnableResourceServer **public class** ResourceServerConfiguration **extends** ResourceServerConfigurerAdapter {   **private static final** String ***DEMO\_RESOURCE\_ID*** = **"\*"**;   @Override  **public void** configure(ResourceServerSecurityConfigurer resources) {  resources.resourceId(***DEMO\_RESOURCE\_ID***).stateless(**true**);  }   @Override  **public void** configure(HttpSecurity http) **throws** Exception {  http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***IF\_REQUIRED***)  .and().requestMatchers().anyRequest()  .and().anonymous()  .and().authorizeRequests()  .antMatchers(**"/product/\*\*"**).access(**"#oauth2.hasAnyScope('read', 'all')"**)  *// 配置访问权限控制，必须认证过后才可以访问* .antMatchers(**"/\*\*"**).authenticated();  } } |

# 请求拦截配置

|  |
| --- |
| */\*\*  \* Security 主配置文件  \*  \** ***@author*** *fengxuechao  \** ***@date*** *2019/3/26  \*/* @Order(2) @EnableWebSecurity **public class** SecurityConfig **extends** WebSecurityConfigurerAdapter {   @Autowired  **private** DataSource **dataSource**;   */\*\*  \* 具体的用户权限控制实现类  \* 通过重载，自定义 user-detail 服务。  \*  \** ***@return*** *\*/* @Bean  @Override  **protected** UserDetailsService userDetailsService() {  InMemoryUserDetailsManager manager = **new** InMemoryUserDetailsManager(); *// JdbcUserDetailsManager manager = new JdbcUserDetailsManager(); // manager.setDataSource(dataSource);* manager.createUser(User.*withUsername*(**"user\_1"**).password(**"123456"**).authorities(**"USER"**).build());  manager.createUser(User.*withUsername*(**"admin"**).password(**"admin"**).authorities(**"USER"**).build());  **return** manager;  }   */\*\*  \* 安全策略配置  \* 通过重载，配置如何通过拦截器保护请求  \*  \** ***@param http*** *\** ***@throws*** *Exception  \*/* @Override  **protected void** configure(HttpSecurity http) **throws** Exception {  http  .csrf().disable()  .requestMatchers().antMatchers(**"/oauth/\*\*"**, **"/login/\*\*"**, **"/logout/\*\*"**)  .and().authorizeRequests().antMatchers(**"/oauth/\*"**).authenticated()  .and().formLogin().permitAll();  }   */\*\*  \* 定义不需要过滤的静态资源（等价于HttpSecurity的permitAll）  \* 通过重载，配置Spring Security的Filter链  \*  \** ***@param web*** *\** ***@throws*** *Exception  \*/* @Override  **public void** configure(WebSecurity web) **throws** Exception {  web.ignoring().antMatchers(**"/css/\*\*"**);  } } |

注意：ResourceServerConfiguration 和 SecurityConfiguration上配置的顺序  
SecurityConfiguration 一定要在 ResourceServerConfiguration 之前，因为 spring 实现安全是通过添加过滤器(Filter)来实现的， 基本的安全过滤应该在oauth过滤之前, 所以在 SecurityConfiguration 设置 @Order(2) , 在 ResourceServerConfiguration 上设置 @Order(6)

# 受保护资源

|  |
| --- |
| */\*\*  \** ***@author*** *fengxuechao  \** ***@date*** *2019/3/26  \*/* @RestController **public class** AuthEndpoints {   @GetMapping(**"/product/{id}"**)  **public** String getProduct(@PathVariable String id) {  Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  **return "product id : "** + id;  }   @GetMapping(**"/order/{id}"**)  **public** String getOrder(@PathVariable String id) {  Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  **return "order id : "** + id;  }   @GetMapping(**"/user/me"**)  **public** Principal user(Principal principal) {  **return** principal;  } } |

# 测试

获取 access\_token

|  |
| --- |
| *# 客户端凭据（客户端ID以及Key）模式* **POST** http://localhost:8081/oauth/token?grant\_type=client\_credentials&scope=all&client\_id=client\_1&client\_secret=123456 |

访问受保护资源

|  |
| --- |
| *# 受保护资源* **GET** http://localhost:8081/product/1?access\_token=4a8d663a-9284-415d-801a-1222023a9cfa |

# 总结

## 令牌的生成和管理

AccessToken的存在意义：

创建AccessToken，并保存，以备后续请求访问都可以认证成功并获取到资源。

如果自定义 AccessToken ,那么就可以附加一些信息，例如：相关权限

## AuthorizationServerTokenServices

AuthorizationServerTokenServices 提供了对AccessToken的相关操作创建、刷新、获取。

|  |
| --- |
| **public interface** AuthorizationServerTokenServices { OAuth2AccessToken createAccessToken(OAuth2Authentication authentication) **throws** AuthenticationException; OAuth2AccessToken refreshAccessToken(String refreshToken, TokenRequest tokenRequest)  **throws** AuthenticationException;   OAuth2AccessToken getAccessToken(OAuth2Authentication authentication);  } |

## DefaultTokenServices

AuthorizationServerTokenServices竟然可以操作AccessToken，那么OAuth2就默认为我们提供了一个默认的DefaultTokenServices。包含了一些有用实现，可以使用它来修改令牌的格式和令牌的存储等，但是生成的token是随机数。

## TokenStore

创建AccessToken完之后，除了发放给第三方，肯定还得保存起来，才可以使用。因此，TokenStore为我们完成这一操作，将令牌（AccessToken）保存或持久化。

TokenStore也有一个默认的实现类InMemoryTokenStore，从名字就知道是通过保存到内存进而实现保存Access Token。 TokenStore的实现有多种类型，可以根据业务需求更改Access Token的保存类型：

* InMemoryTokenStore
* JdbcTokenStore
* JwtTokenStore
* RedisTokenStore

## 自定义错误处理

实际上我们上面说到的端点，其实可以看成Controller，用于返回不同端点的响应内容。

授权服务的错误信息是使用标准的Spring MVC来进行处理的，也就是 @ExceptionHandler 注解的端点方法，我们可以提供一个 WebResponseExceptionTranslator 对象。最好的方式是改变响应的内容而不是直接进行渲染。

假如说在呈现令牌端点的时候发生了异常，那么异常委托了 HttpMessageConverters 对象（它能够被添加到MVC配置中）来进行输出。

假如说在呈现授权端点的时候未通过验证，则会被重定向到 /oauth/error 即错误信息端点中。whitelabel error （即Spring框架提供的一个默认错误页面）错误端点提供了HTML的响应，但是我们大概可能需要实现一个自定义错误页面（例如只是简单的增加一个 @Controller 映射到请求路径上 @RequestMapping("/oauth/error")）。

## **资源服务器**

资源服务器，其实就是存放一些受令牌保护的资源，只有令牌并且有效正确才能获取到资源。 内部是通过Spring OAuth2的Spring Security Authentication filter 的过滤链来进行保护。

我们可以继承ResourceServerConfigurerAdapter，来使用 ResourceServerSecurityConfigurer进行相关配置。

**ResourceServerTokenServices**

ResourceServerTokenServices 是组成授权服务的另一半。

1）.若是资源服务器和授权服务在同一个应用，可以使用DefaultTokenServices

2）.若是分离的。ResourceServerTokenServices必须知道令牌的如何解码。

ResourceServerTokenServices解析令牌的方法：

使用RemoteTokenServices，资源服务器通过HTTP请求来解码令牌。每次都请求授权服务器的端点-/oauth/check\_toke，以此来解码令牌。

同时授权认证服务器需要把/oauth/check\_toke暴露出来。

|  |
| --- |
| */\*\*  \** ***@author*** *fengxuechao  \** ***@date*** *2019/3/26  \*/* @Configuration @EnableAuthorizationServer **public class** AuthServerConfig **extends** AuthorizationServerConfigurerAdapter {    @Override  **public void** configure(AuthorizationServerSecurityConfigurer oauthServer) **throws** Exception {  *//允许表单认证* oauthServer.allowFormAuthenticationForClients();  oauthServer.tokenKeyAccess(**"permitAll()"**);  oauthServer.checkTokenAccess(**"isAuthenticated()"**);  } } |

# 参考资源

* <https://projects.spring.io/spring-security-oauth/docs/oauth2.html>
* <http://www.spring4all.com/article/582>
* <https://www.baeldung.com/security-spring>