**【Oauth2.0】**

**【主要内容】**

1. Oauth2.0简介
2. Oauth2.0授权思路
3. 基于redis的授权服务器搭建
4. 四种授权方式
5. Oauth2.0整合数据库
6. 基于Jwt的授权服务
7. Jwt的登出问题

**【学习目标】**

|  |  |
| --- | --- |
| 知识点 | 要求 |
| Oauth2.0简介 | 了解 |
| Oauth2.0授权思路 | 掌握 |
| 基于redis的授权服务器搭建 | 掌握 |
| 四种授权方式 | 掌握 |
| Oauth2.0整合数据库 | 掌握 |
| 基于Jwt的授权服务 | 掌握 |
| Jwt的登出问题 | 掌握 |

# Oauth2.0简介

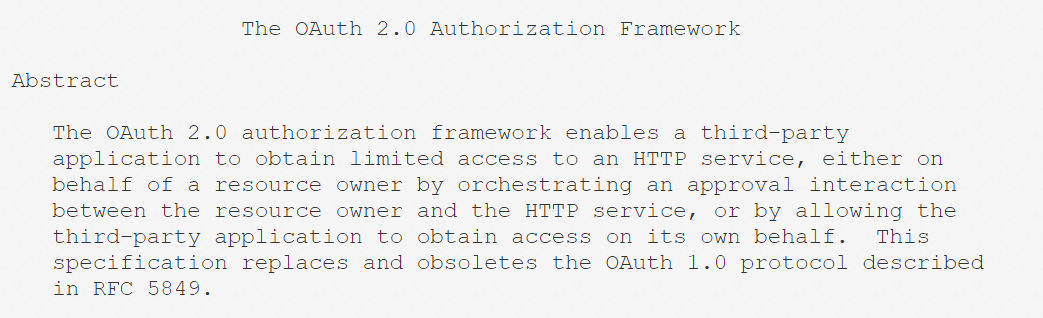
先放三个链接：

<https://github.com/spring-projects/spring-security/wiki/OAuth-2.0-Migration-Guide>

<http://www.ruanyifeng.com/blog/2014/05/oauth_2_0.html>

<http://www.rfcreader.com/#rfc6749>

Oauth2.0官方解释：



**Oauth2.0：关于授权（authorization）的开放网络标准，使得第三方应用有权力去访问Http服务！**

# Oauth2.0的定位

解决的就是**用户和资源之间的关系**问题！

谁能访问那些资源？回顾RBAC模型：基于角色的用户权限控制

5 个表：

User:

User\_role

Role:

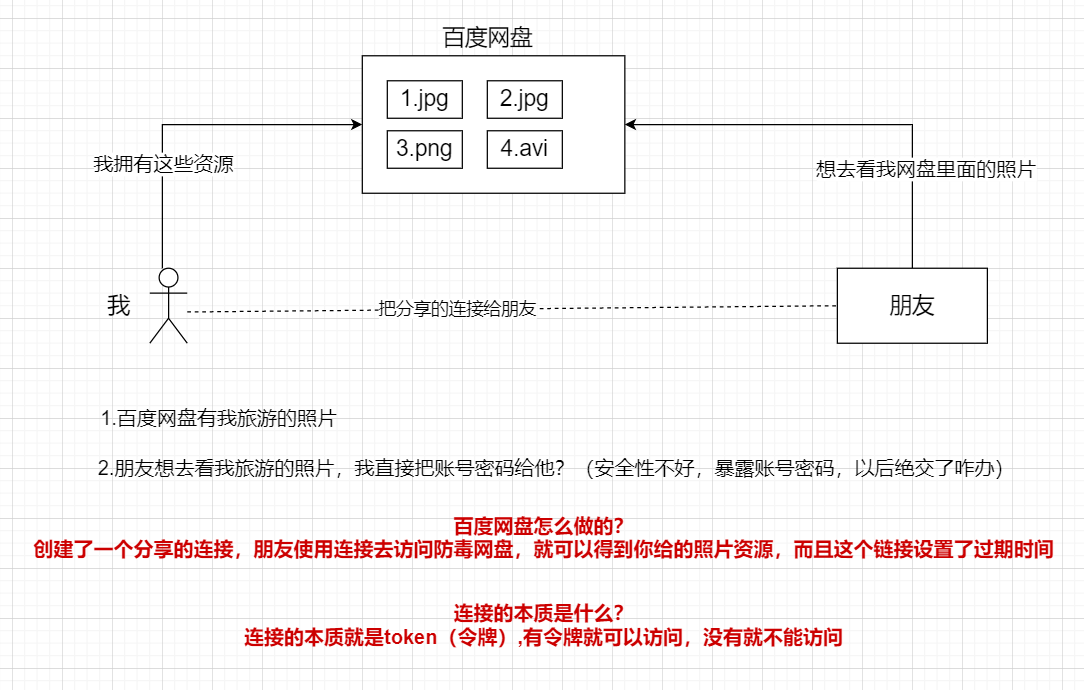
Role\_permission

Permission

RBAC的本质：就是使用角色来打包资源，然后把角色给用户

OAuth2.0 也实现了RBAC 模式里面的打包模式！

# 百度云的例子



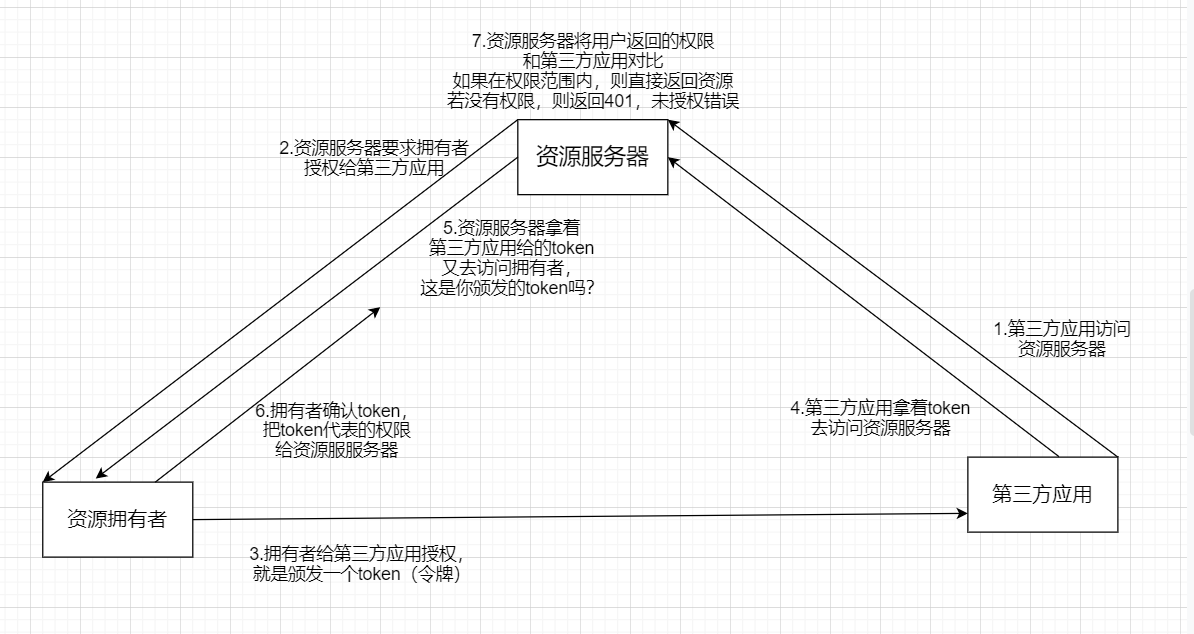
## 名词解释

我：资源的拥有者

百度网盘：资源服务器，里面放的资源

朋友：第三方应用

# Oauth2.0的授权思路



1. 第三方应用访问资源服务器
2. 资源服务器要求资源拥有者授权给第三方应用
3. 资源拥有者给第三方应用授权，就是颁发一个token（令牌）
4. 第三方应用拿着token又去访问资源服务器
5. 资源服务器拿着第三方应用给的token又去访问资源拥有者，这是你颁发的token吗？
6. 资源拥有者确认token，把token代表的权限给资源服服务器
7. 资源服务器将用户返回的权限和第三方应用对比，如果在权限范围内，则直接返回资源若没有权限，则返回401，未授权错误

**总结：OAuth2.0的思路在于**

**我在第三方应用和资源服务器之间，架一个授权服务器，授权服务和资源的拥有者交互，作用有2个：**

**1 颁发token(给第三方应用)**

**2 资源服务器通过token访问授权服务，授权服务能返回权限信息 （校验token）**

## Oauth2.0怎么和微服务整合

我们的好多服务，都需要权限才能访问。

我们把**url 看成资源。这些资源放在资源服务器里面 资源服务器（具体的应用 user-service）**

**资源是url地址**

由授权服务给第三方应用授权，第三方服务得到Token ，然后每次访问都带上Token

**第三方应用：网站的访问者。**

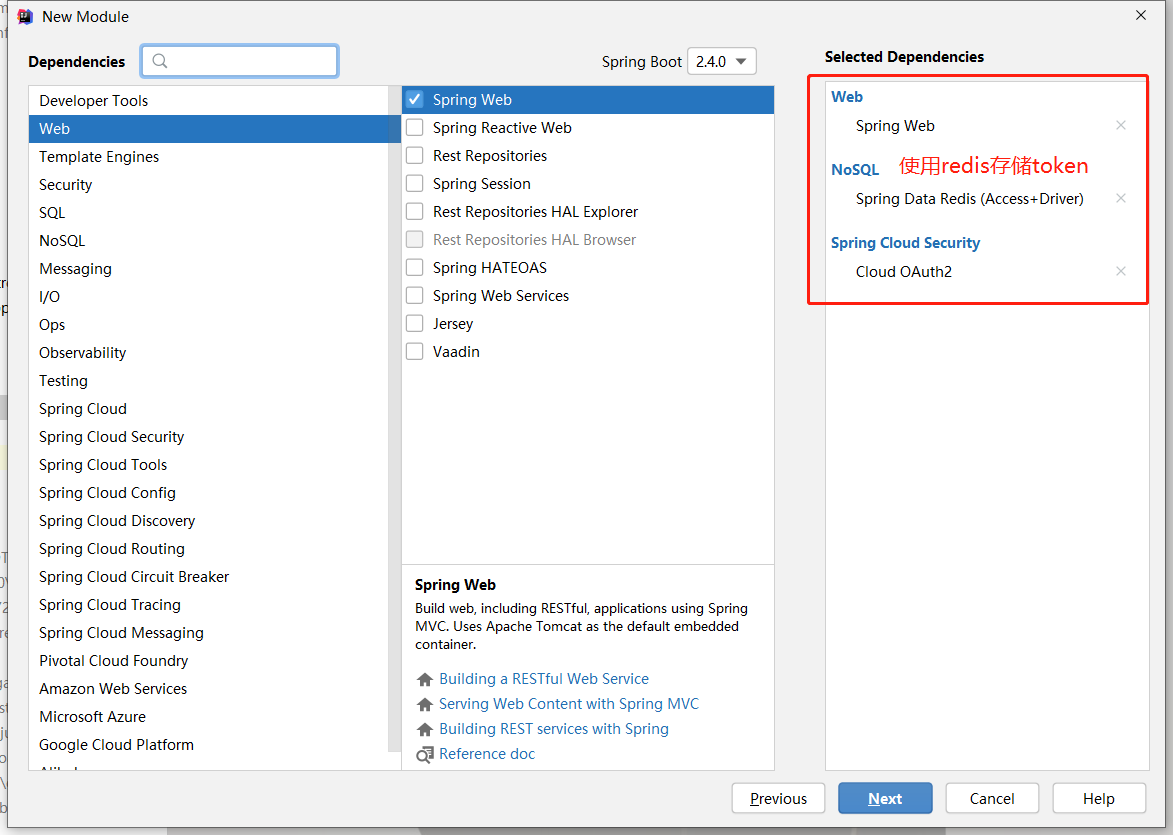
**微服务就是资源的服务器**

**授权服务需要搭建**

**资源的拥有者就是数据库里面的用户名和密码**

# 搭建授权服务器

## 新建项目，选择依赖



## 修改启动类

|  |
| --- |
| @SpringBootApplication @EnableAuthorizationServer //开启授权服务器 public class AuthServerApplication {   public static void main(String[] args) {  SpringApplication.*run*(AuthServerApplication.class, args);  }  } |

## 修改yml配置文件

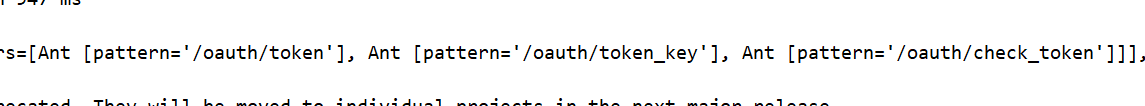
|  |
| --- |
| server:  port: 9999 spring:  application:  name: auth-server  redis: #redis的配置 用于存放我们颁发的token  host: 192.168.226.128  port: 6379  database: 0 |

## 创建配置类AuthorizationConfig

|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.data.redis.connection.RedisConnectionFactory; import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer; import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter; import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer; import org.springframework.security.oauth2.provider.token.TokenStore; import org.springframework.security.oauth2.provider.token.store.redis.RedisTokenStore;  @Configuration //标记为我们自己的配置类 public class AuthorizationConfig extends AuthorizationServerConfigurerAdapter {    //注入redis的连接工厂  @Autowired  private RedisConnectionFactory redisConnectionFactory;    /\*\*  \* 描述: 创建一个redis的tokenStore存放颁发的token  \*/  @Bean  public TokenStore tokenStore() {  return new RedisTokenStore(redisConnectionFactory);  }    /\*\*  \* 描述: 第三方应用的配置，只有配置了第三方应用，才能访问授权服务器  \*  \* @param clients:  \* @return void  \*/  @Override  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  //创建一个基本的第三方应用，第三方应用少的时候，就放在内存中，多的就使用jdbc  clients.inMemory()  .withClient("web") //创建一个web第三方  .secret("web-secret") //密码  .scopes("web-scopes") //作用域  .authorizedGrantTypes("authorization\_code") //授权方式一共有四种，这里是验证码授权，还有密码授权，静默授权，客户端授权  .redirectUris("https://www.baidu.com") //访问成功后的跳转地址  .accessTokenValiditySeconds(7200); //token的过期时间给2小时    }   /\*\*  \* 描述: token的存储方式 我们先存在redis里面  \*  \* @param endpoints:  \* @return void  \*/  @Override  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  //使用redis做token的存储  endpoints.tokenStore(tokenStore());    } } |

## 启动查看控制台

打印了很多暴露的接口出来



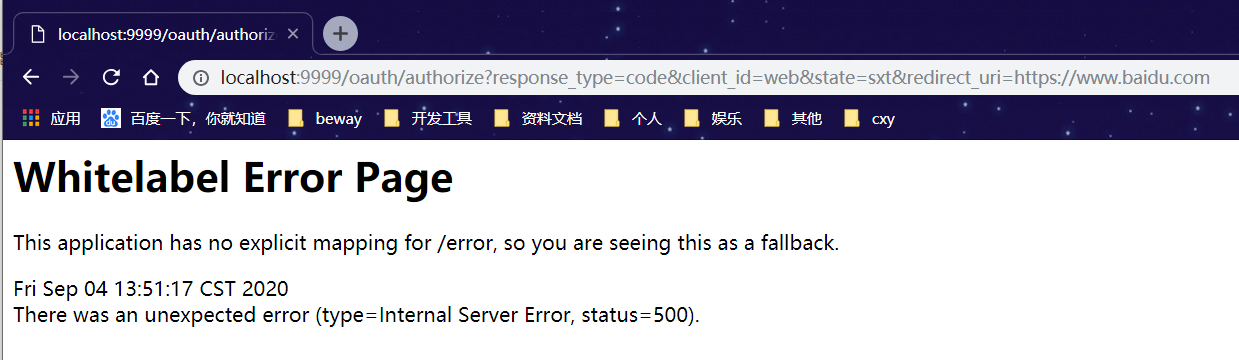
## 怎么获取token

先看请求规范：**发GET请求**

|  |
| --- |
| [http://localhost:9999/oauth/authorize?response\_type=code&client\_id=web&state=sxt&redirect\_uri=https://www.baidu.com](http://localhost:9999/oauth/authorize?response_type=code&client_id=web&state=sxt&redirect_uri=https:/www.baidu.com) |

1. <http://localhost:9999/oauth> 是规定的context-path
2. authorize 是oauth2.0的默认的api前缀，你改不了
3. response\_type 是授权类型，code是授权码授权
4. client\_id 是第三方应用id，就是我们刚才定义的web
5. state 是状态值，维护请求和回调之间的状态的不透明值，防止跨站点请求伪造
6. redirect\_uri 请求成功后的重定向地址，要是公网的地址，必须**https**

## 访问试试

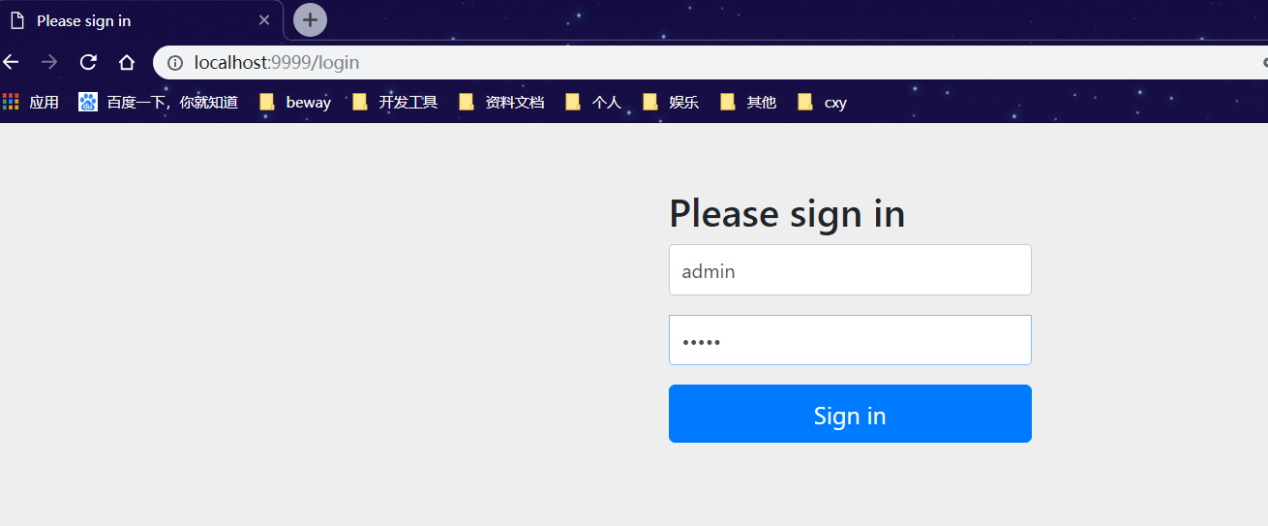


发现报了500，这是因为我们没有登录，获取不了token，我们搞个用户先登录一下

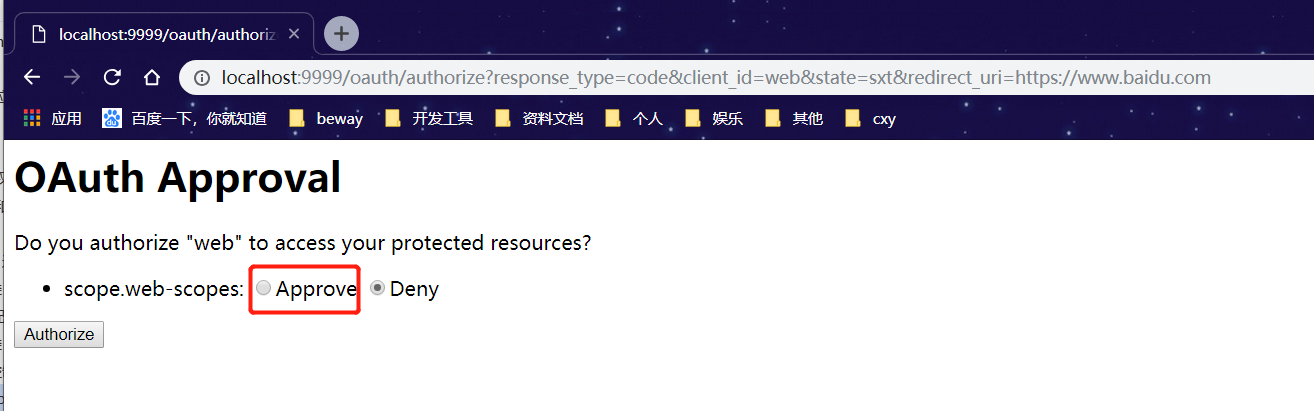
## 创建WebSecurityConfig（security的知识点）

|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder; import org.springframework.security.config.annotation.web.builders.HttpSecurity; import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter; import org.springframework.security.core.authority.SimpleGrantedAuthority; import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import org.springframework.security.crypto.password.PasswordEncoder;  import java.util.HashSet;  @Configuration public class WebSecurityConfig extends WebSecurityConfigurerAdapter {    @Autowired  private PasswordEncoder passwordEncoder;   @Override  protected void configure(HttpSecurity http) throws Exception {  //设置所有的请求都要在登陆后才能访问  http.authorizeRequests().anyRequest().authenticated();  //给一个登录页面  http.formLogin();  }   /\*\*  \* 描述: 在内存中创建一个用户来登录  \*  \* @param auth:  \* @return void  \*/  @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  User user = new User("admin", passwordEncoder.encode("admin"), new HashSet<SimpleGrantedAuthority>());  auth.inMemoryAuthentication().withUser(user);  }    /\*\*  \* 描述: 加密方式  \*  \* @param :  \* @return org.springframework.security.crypto.password.PasswordEncoder  \*/  @Bean  public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder();  } } |

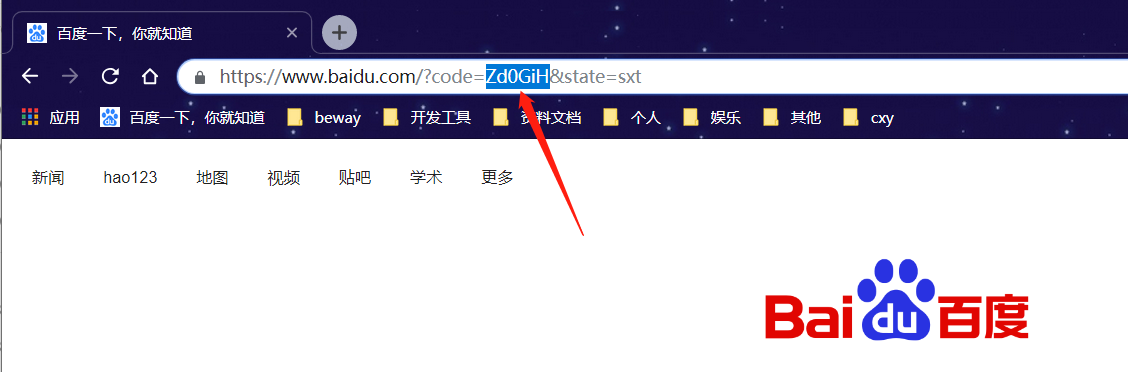
## 再启动访问试试



登陆后，访问我们刚才的连接



点击Approve，然后点击授权



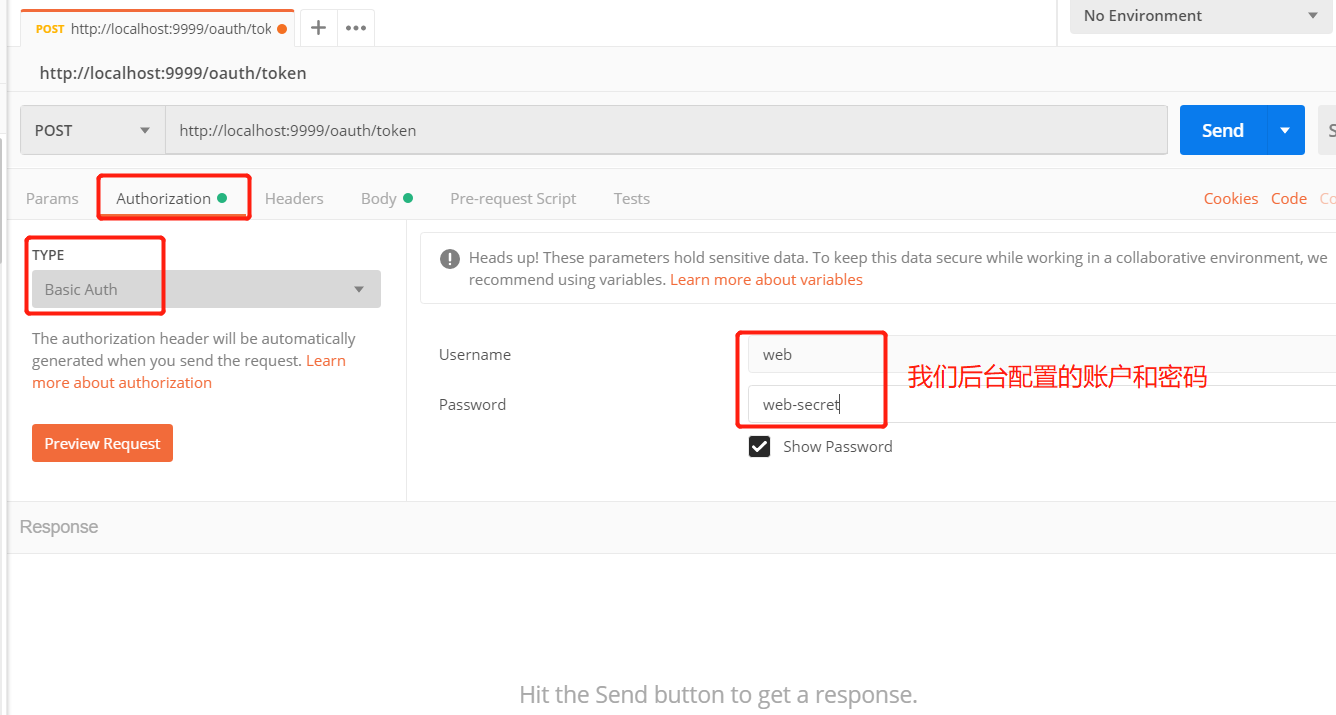
发现跳到了我们配置的重定向的地址，然后我们可以获取到code

## 使用code获取token

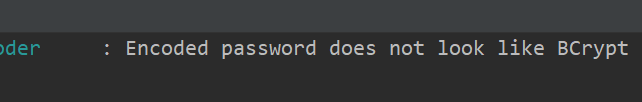
这里我们使用postman发一个**post请求**

[http://localhost:9999/oauth/token?grant\_type=authorization\_code&code=8GirYx&redirect\_uri=https://www.baidu.com](http://localhost:8080/oauth/token?grant_type=authorization_code&code=8GirYx&redirect_uri=https:/www.baidu.com)





发送试试看，后台报错了，我们刚才定义的第三方应用web的密码没加密

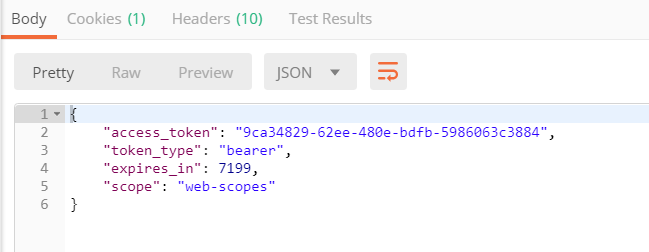


## 修改AuthorizationConfig



## 重启测试，记得重新获取code

返回access\_token就代表成功了



## 上面就是授权码授权，总结一下流程



# 静默授权

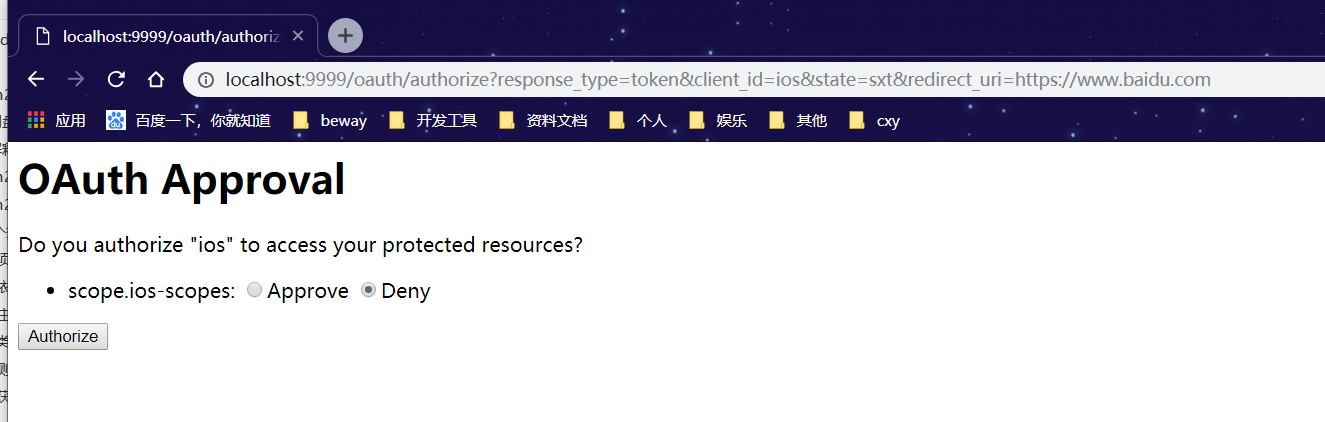
**不获取code，直接拿到token**

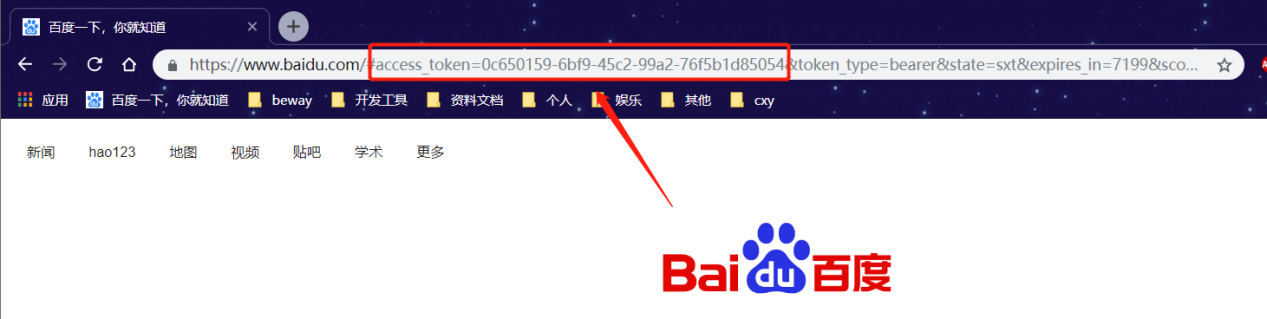
## 修改AuthorizationConfig类

|  |
| --- |
| /\*\*  \* 描述: 第三方应用的配置，只有配置了第三方应用，才能访问授权服务器  \*  \* @param clients:  \* @return void  \*/ @Override public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  //创建一个基本的第三方应用，第三方应用少的时候，就放在内存中，多的就使用jdbc  clients.inMemory()  .withClient("web") //创建一个web第三方  .secret(passwordEncoder.encode("web-secret")) //密码  .scopes("web-scopes") //作用域  .authorizedGrantTypes("authorization\_code") //授权方式一共有四种，这里是验证码授权，还有密码授权，静默授权，客户端授权  .redirectUris("https://www.baidu.com") //访问成功后的跳转地址  .accessTokenValiditySeconds(7200) //token的过期时间给2小时  **.and()  .withClient("ios")  .secret(passwordEncoder.encode("ios-secret"))  .scopes("ios-scopes")  .authorizedGrantTypes("implicit")//静默授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200);**   } |

## 访问测试

|  |
| --- |
| [http://localhost:9999/oauth/authorize?response\_type=token&client\_id=ios&state=sxt&redirect\_uri=https://www.baidu.com](http://localhost:9999/oauth/authorize?response_type=token&client_id=ios&state=sxt&redirect_uri=https:/www.baidu.com) |





这种方式把令牌直接传给前端，是很不安全的。因此，只能用于一些安全要求不高的场景，并且令牌的有效期必须非常短，通常就是会话期间（session）有效，浏览器关掉，令牌就失效了

# 密码授权（重要）

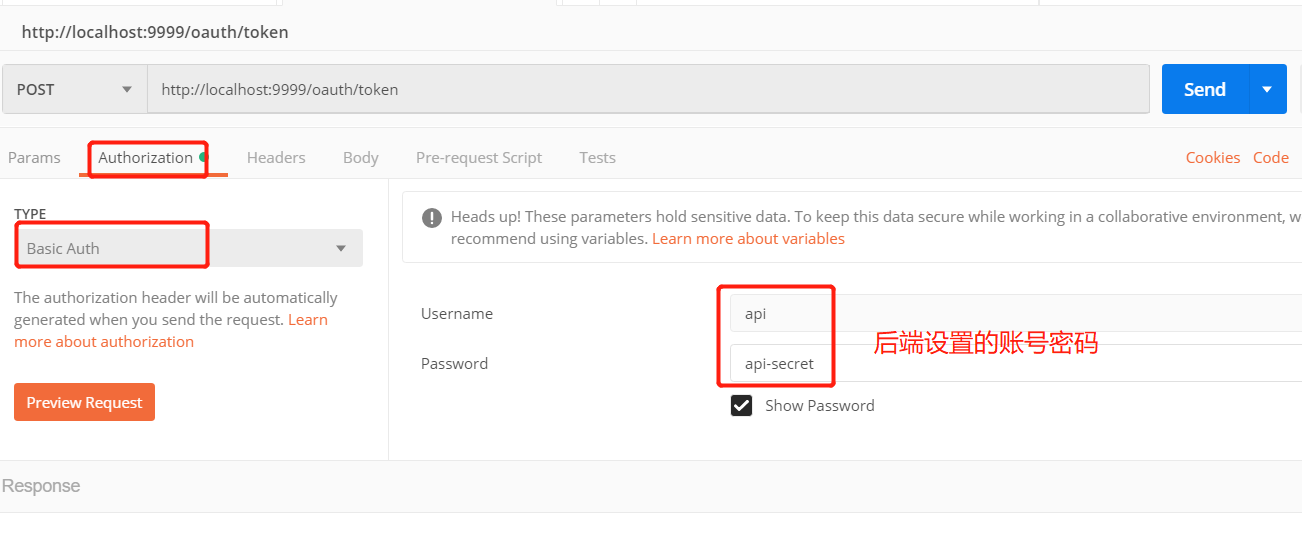
## 密码授权必须要有认证管理器，修改WebSecurityConfig

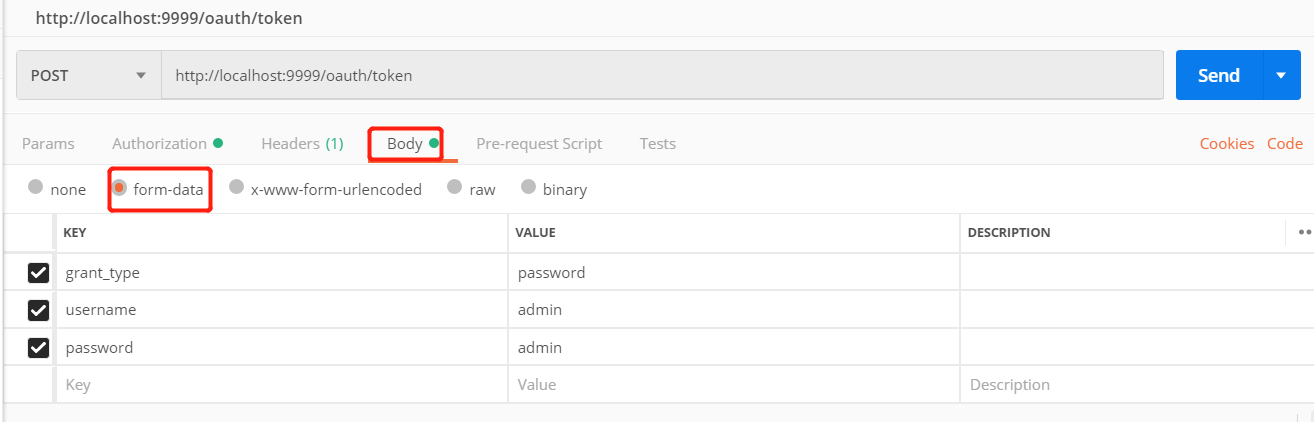
|  |
| --- |
| /\*\*  \* 密码授权 需要这个认证管理器  \*  \* @return  \* @throws Exception  \*/ @Override @Bean public AuthenticationManager authenticationManagerBean() throws Exception {  return super.authenticationManagerBean(); } |

## 修改AuthorizationConfig类

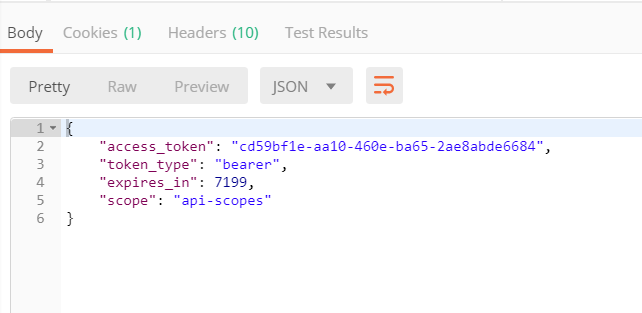
|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.data.redis.connection.RedisConnectionFactory; import org.springframework.security.authentication.AuthenticationManager; import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer; import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter; import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer; import org.springframework.security.oauth2.provider.token.TokenStore; import org.springframework.security.oauth2.provider.token.store.redis.RedisTokenStore;  @Configuration //标记为我们自己的配置类 public class AuthorizationConfig extends AuthorizationServerConfigurerAdapter {    //注入redis的连接工厂  @Autowired  private RedisConnectionFactory redisConnectionFactory;   /\*\*  \* 描述: 创建一个redis的tokenStore存放颁发的token  \*/  @Bean  public TokenStore tokenStore() {  return new RedisTokenStore(redisConnectionFactory);  }   @Autowired  @Lazy  private PasswordEncoder passwordEncoder;   **/\*\*  \* 描述: 注入认证管理器  \*  \* @return  \*/  @Autowired  private AuthenticationManager authenticationManager;**   /\*\*  \* 描述: 第三方应用的配置，只有配置了第三方应用，才能访问授权服务器  \*  \* @param clients:  \* @return void  \*/  @Override  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  //创建一个基本的第三方应用，第三方应用少的时候，就放在内存中，多的就使用jdbc  clients.inMemory()  .withClient("web") //创建一个web第三方  .secret(passwordEncoder.encode("web-secret")) //密码  .scopes("web-scopes") //作用域  .authorizedGrantTypes("authorization\_code") //授权方式一共有四种，这里是验证码授权，还有密码授权，静默授权，客户端授权  .redirectUris("https://www.baidu.com") //访问成功后的跳转地址  .accessTokenValiditySeconds(7200)  .and()  .withClient("ios")  .secret(passwordEncoder.encode("ios-secret"))  .scopes("ios-scopes")  .authorizedGrantTypes("implicit")//静默授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200)  **.and()  .withClient("api")  .secret(passwordEncoder.encode("api-secret"))  .scopes("api-scopes")  .authorizedGrantTypes("password") //密码授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200); //token的过期时间给2小时**  super.configure(clients);  }   /\*\*  \* 描述: token的存储方式 我们先存在redis里面  \*  \* @param endpoints:  \* @return void  \*/  @Override  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  //使用redis做token的存储,密码授权必须要认证管理器   **endpoints.tokenStore(tokenStore()).authenticationManager(authenticationManager);**  super.configure(endpoints);  } } |

## 测试获取token





token已经拿到了

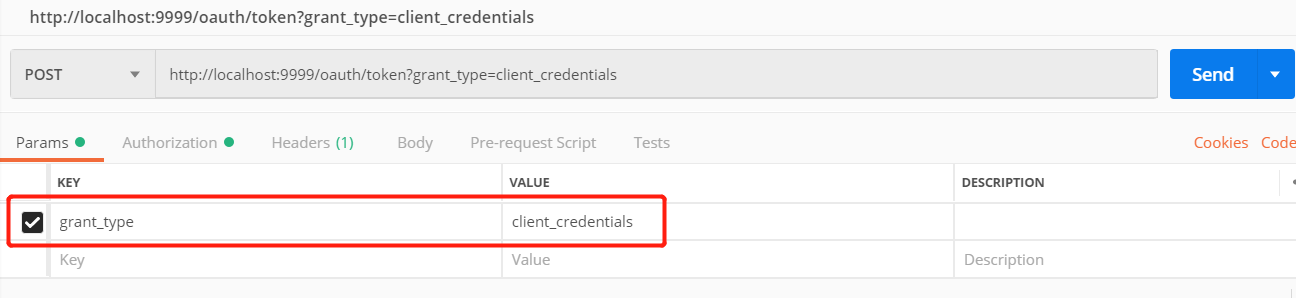


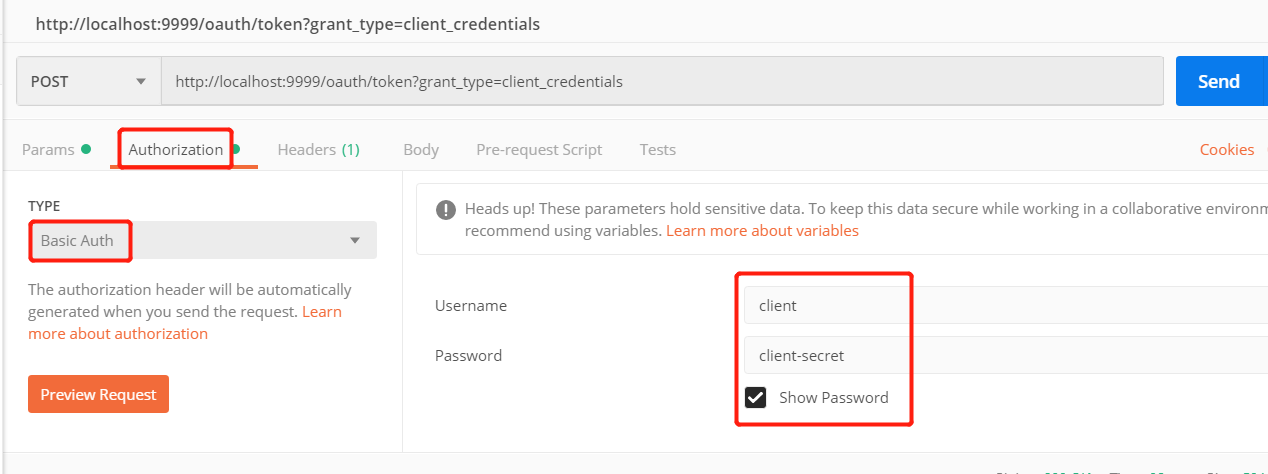
# 客户端授权

## 修改AuthorizationConfig类

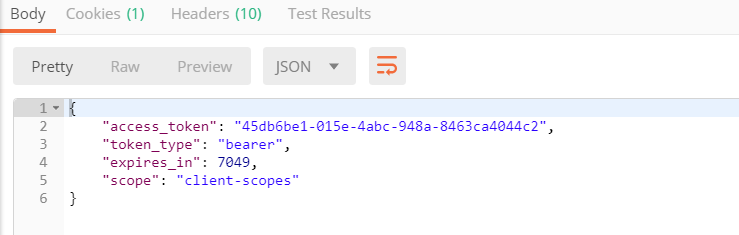
|  |
| --- |
| /\*\*  \* 描述: 第三方应用的配置，只有配置了第三方应用，才能访问授权服务器  \*  \* @param clients:  \* @return void  \*/ @Override public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  //创建一个基本的第三方应用，第三方应用少的时候，就放在内存中，多的就使用jdbc  clients.inMemory()  .withClient("web") //创建一个web第三方  .secret(passwordEncoder.encode("web-secret")) //密码  .scopes("web-scopes") //作用域  .authorizedGrantTypes("authorization\_code") //授权方式一共有四种，这里是验证码授权，还有密码授权，静默授权，客户端授权  .redirectUris("https://www.baidu.com") //访问成功后的跳转地址  .accessTokenValiditySeconds(7200)  .and()  .withClient("ios")  .secret(passwordEncoder.encode("ios-secret"))  .scopes("ios-scopes")  .authorizedGrantTypes("implicit")//静默授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200)  .and()  .withClient("api")  .secret(passwordEncoder.encode("api-secret"))  .scopes("api-scopes")  .authorizedGrantTypes("password") //密码授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200)   **.and()  .withClient("client")  .secret(passwordEncoder.encode("client-secret"))  .scopes("client-scopes")  .authorizedGrantTypes("client\_credentials") //客户端授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200); //token的过期时间给2小时**   } |

## 测试访问

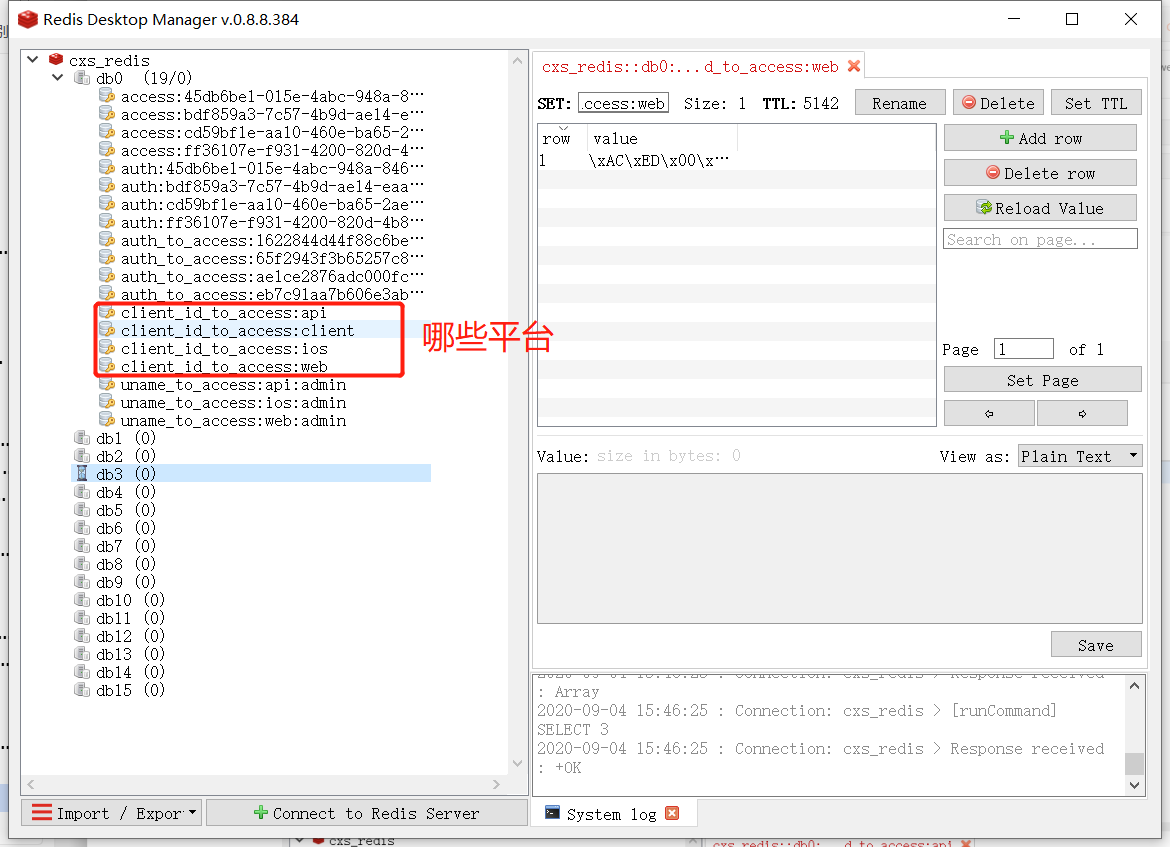




token就拿到了



## 查看redis存了哪些数据



# 自定义用户名和密码测试访问和权限

因为Oauth2.0是整合了SpringSecurity，所以我们可以直接使用Security的知识

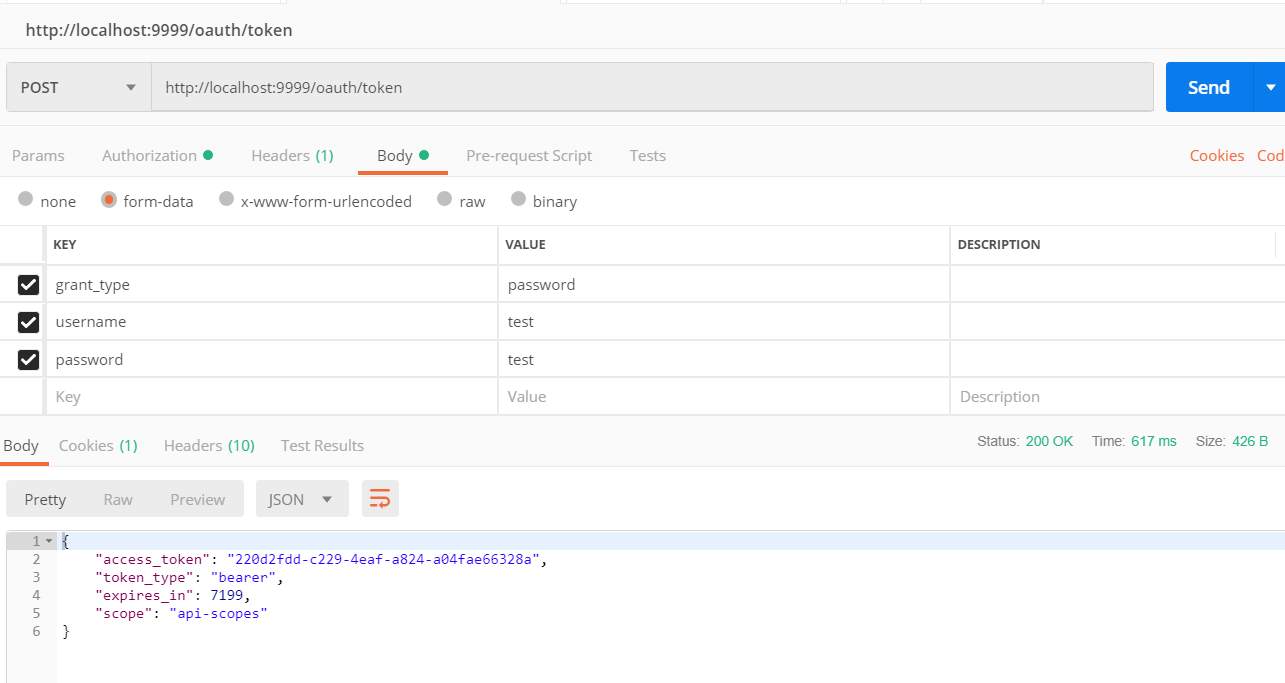
## 创建UserDetailServiceImpl

|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.security.core.authority.SimpleGrantedAuthority; import org.springframework.security.core.userdetails.User; import org.springframework.security.core.userdetails.UserDetails; import org.springframework.security.core.userdetails.UserDetailsService; import org.springframework.security.core.userdetails.UsernameNotFoundException; import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.stereotype.Service;  import java.util.Arrays; import java.util.HashMap; import java.util.List; import java.util.Map;  @Service public class UserDetailServiceImpl implements UserDetailsService {    @Autowired  private PasswordEncoder passwordEncoder;    @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  Map<String, User> users = new HashMap<>(4);  //模拟数据库创建两个用户  List<SimpleGrantedAuthority> roles = Arrays.*asList*(new SimpleGrantedAuthority("admin"));  User admin = new User("admin", passwordEncoder.encode("admin"), roles);  users.put("admin", admin);  List<SimpleGrantedAuthority> roles2 = Arrays.*asList*(new SimpleGrantedAuthority("test"));  User test = new User("test", passwordEncoder.encode("test"), roles2);  users.put("test", test);  return users.get(username);  } } |

## 修改WebSecurityConfig

|  |
| --- |
| @Autowired  private UserDetailsServiceImpl userDetailsService;   /\*\*  \* 配置认证的  \*  \* @param auth  \* @throws Exception  \*/  @Override  protected void configure(AuthenticationManagerBuilder auth) throws Exception {  // 走自己的登录授权方法  auth.userDetailsService(userDetailsService);  } |

## 启动测试获取token

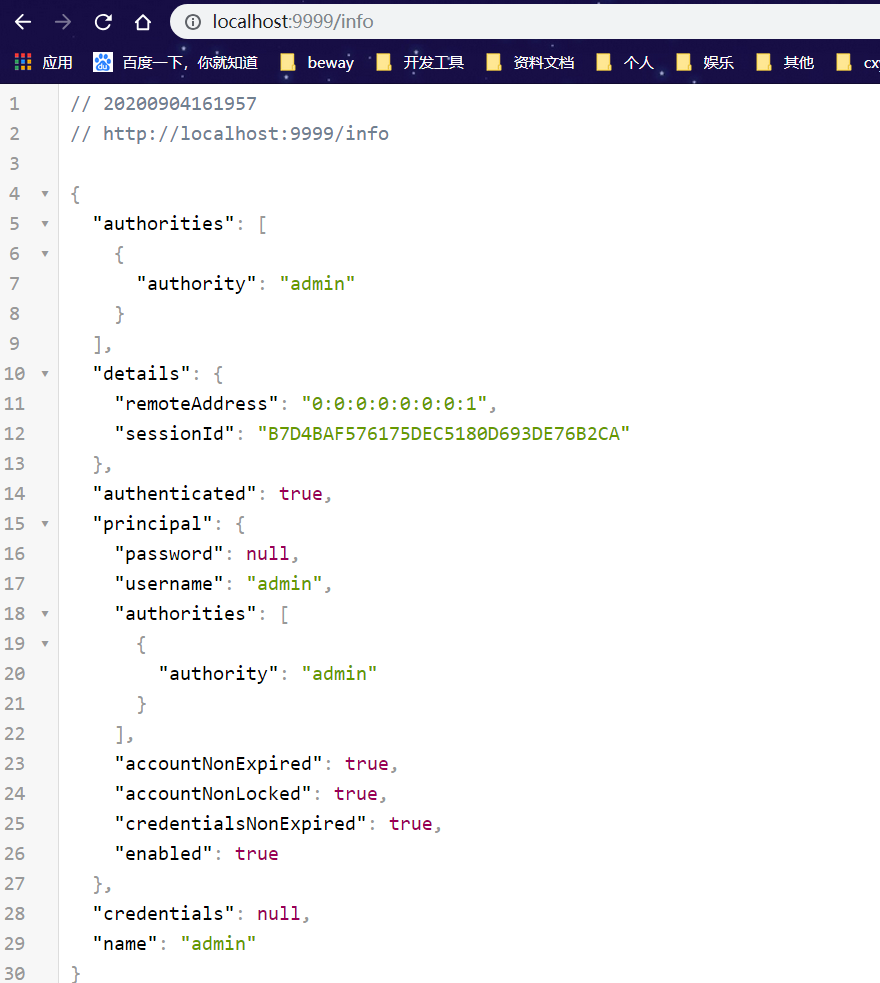


## 如何获取当前用户呢？

### 创建TestController

|  |
| --- |
| @RestController public class TestController {    /\*\*  \* 描述: 获取当前用户//必须登录才能访问  \*  \* @param principal:  \* @return java.lang.Object  \*/  @GetMapping("info")  public Object info(Principal principal) {  return principal;  }  } |

### 登陆后可以访问



### 将授权服务器变成资源服务器再试试访问

修改启动类

|  |
| --- |
| @SpringBootApplication @EnableAuthorizationServer //开启授权服务器 **@EnableResourceServer //开启资源服务器 下面所有的接口访问都需要带token** public class AuthServerApplication {   public static void main(String[] args) {  SpringApplication.*run*(AuthServerApplication.class, args);  }  } |

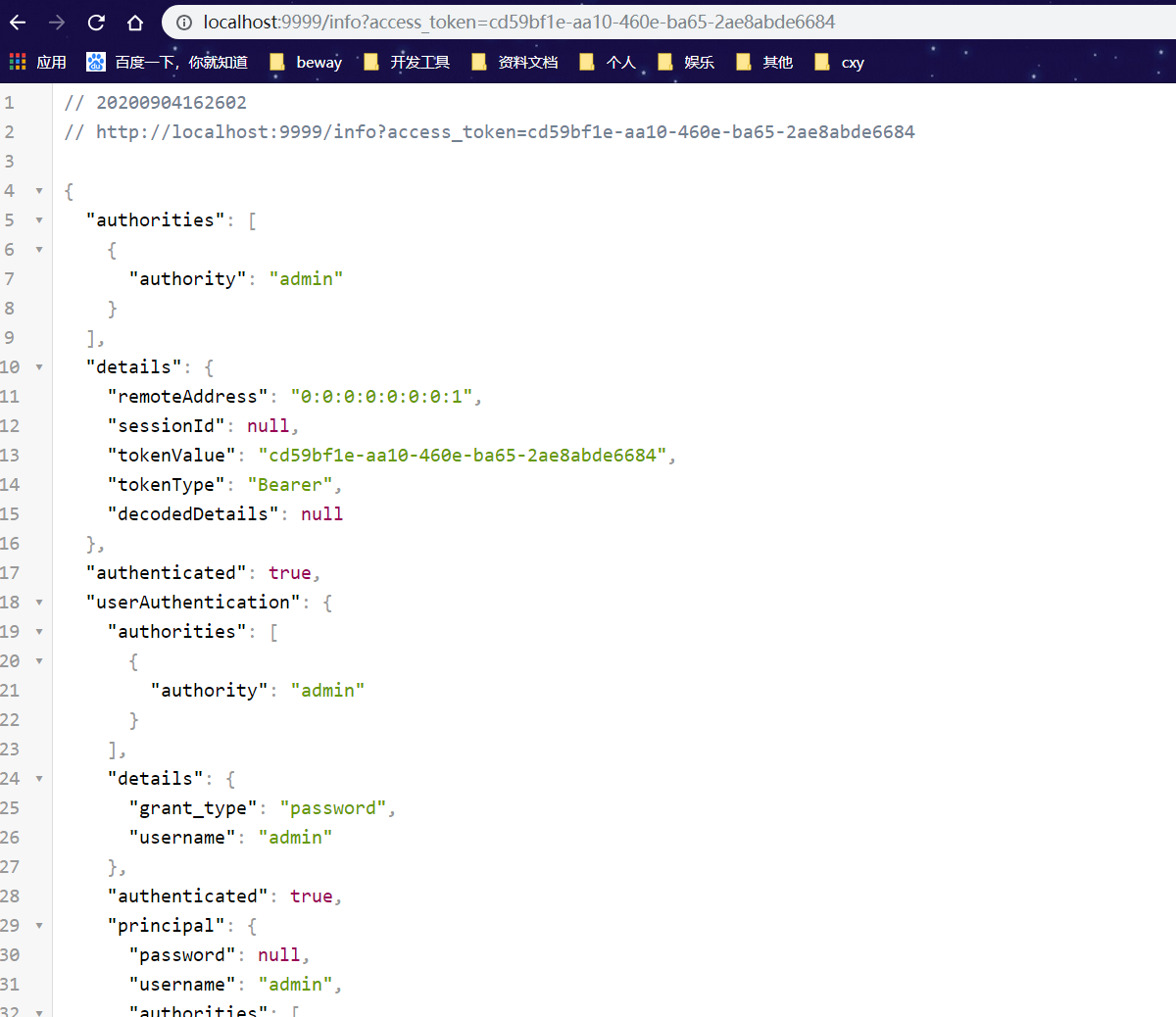
测试访问http://localhost:9999/info 发现并没有让我们登录，而是直接报错了



带上token访问试试

先获取token再发请求

http://localhost:9999/info?access\_token=cd59bf1e-aa10-460e-ba65-2ae8abde6684



## 测试一下权限

### 开启角色拦截

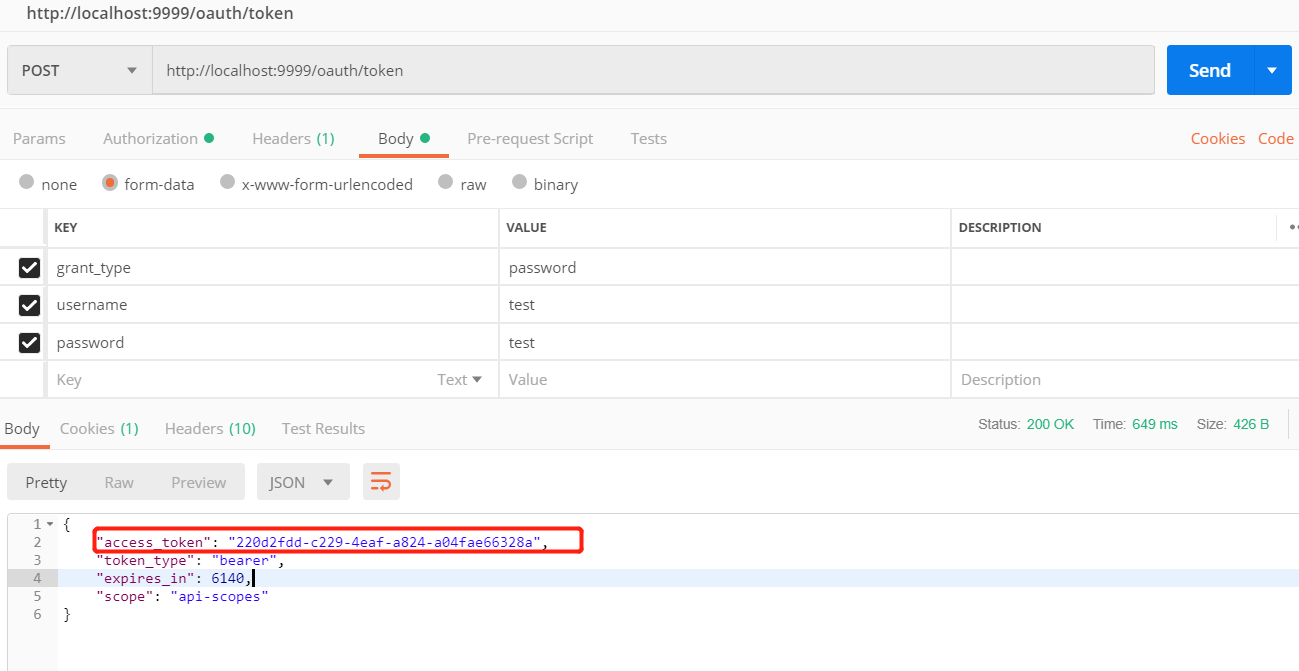
|  |
| --- |
| @SpringBootApplication @EnableAuthorizationServer //开启授权服务器 @EnableResourceServer //开启资源服务器 下面所有的接口访问都需要带token @EnableGlobalMethodSecurity(prePostEnabled = true) //security里面的注解，所有的方法级别的访问需要验证权限 public class AuthServerApplication {   public static void main(String[] args) {  SpringApplication.*run*(AuthServerApplication.class, args);  }  } |

### 修改TestController

|  |
| --- |
| /\*\*  \* 描述: 只有admin权限的才可以访问  \*  \* @param :  \* @return java.lang.String  \*/ @GetMapping("admin") @PreAuthorize("hasAuthority('admin')") public String admin() {  return "admin"; }  /\*\*  \* 描述: 只有test权限的才可以访问  \*  \* @param :  \* @return java.lang.String  \*/ @GetMapping("test") @PreAuthorize("hasAuthority('test')") public String test() {  return "test"; } |

### 访问测试

先获取test用户的token



然后去访问admin接口，就拒绝访问



去访问test的接口，则可以访问



# 集成数据库

## 创建数据库和执行sql，准备实体类



## 添加依赖

|  |
| --- |
| <!-- 集成数据库--> <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.2.0</version> </dependency> <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.21</version> </dependency> |

## 修改application.yml配置文件

|  |
| --- |
| server:  port: 9999 spring:  application:  name: auth-server  datasource:  username: root  password: root  url: jdbc:mysql://localhost:3306/oauth-study?serverTimezone=UTC  driver-class-name: com.mysql.cj.jdbc.Driver  redis:  host: 192.168.127.128  port: 6380  password: cxs1013??   database: 1 mybatis-plus:  configuration:  log-impl: org.apache.ibatis.logging.stdout.StdOutImpl |

## 修改Member类

|  |
| --- |
| package com.sxt.domain;  import com.baomidou.mybatisplus.annotation.IdType; import com.baomidou.mybatisplus.annotation.TableField; import com.baomidou.mybatisplus.annotation.TableId; import com.baomidou.mybatisplus.annotation.TableName; import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor; import org.springframework.security.core.GrantedAuthority; import org.springframework.security.core.userdetails.UserDetails;  import java.io.Serializable; import java.util.Collection; import java.util.Collections;  /\*\*  \* @Author: 武汉尚学堂  \*/ @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "`member`") public class Member **implements Serializable, UserDetails** {   @TableId(value = "memberid", type = IdType.*AUTO*)  private Integer memberid;   @TableField(value = "membername")  private String membername;   @TableField(value = "memberpwd")  private String memberpwd;   @TableField(value = "memberaddress")  private String memberaddress;   /\*\*  \* 这个方法？？  \* 权限  \* 因为前台用户不涉及权限  \*  \* @return  \*/  **@Override  public Collection<? extends GrantedAuthority> getAuthorities() {  return Collections.*EMPTY\_LIST*;  }**   /\*\*  \* 密码  \*  \* @return  \*/  @Override  public String getPassword() {  return this.memberpwd;  }   /\*\*  \* 用户名SecurityContextHolder.getContext().getAuthorication().getprinciple().xxx  \*  \* @return  \*/  @Override  public String getUsername() {  return String.*valueOf*(memberid);  }   @Override  public boolean isAccountNonExpired() {  return true;  }   @Override  public boolean isAccountNonLocked() {  return true;  }   @Override  public boolean isCredentialsNonExpired() {  return true;  }   @Override  public boolean isEnabled() {  return true;  }  } |

## 修改SysUser类

|  |
| --- |
| package com.sxt.domain;  import com.baomidou.mybatisplus.annotation.IdType; import com.baomidou.mybatisplus.annotation.TableField; import com.baomidou.mybatisplus.annotation.TableId; import com.baomidou.mybatisplus.annotation.TableName; import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor; import org.springframework.security.core.GrantedAuthority; import org.springframework.security.core.authority.SimpleGrantedAuthority; import org.springframework.security.core.userdetails.UserDetails; import org.springframework.util.CollectionUtils;  import java.io.Serializable; import java.util.ArrayList; import java.util.Collection; import java.util.Collections; import java.util.List;  /\*\*  \* @Author: 武汉尚学堂  \*/ @Data @AllArgsConstructor @NoArgsConstructor @TableName(value = "sys\_user") public class SysUser **implements Serializable, UserDetails** {  @TableId(value = "userid", type = IdType.*AUTO*)  private Integer userid;   @TableField(value = "username")  private String username;   @TableField(value = "userpwd")  private String userpwd;   @TableField(value = "sex")  private String sex;   @TableField(value = "address")  private String address;   /\*\*  \* 给一个权限  \*/  @TableField(exist = false)  private List<String> authorities = Collections.*emptyList*();   /\*\*  \* 权限  \*  \* @return  \*/  **@Override  public Collection<? extends GrantedAuthority> getAuthorities() {  if (CollectionUtils.*isEmpty*(authorities)) {  return Collections.*emptyList*();  }  //有权限  ArrayList<SimpleGrantedAuthority> simpleGrantedAuthorities = new ArrayList<>();  authorities.forEach(authoritie -> {  simpleGrantedAuthorities.add(new SimpleGrantedAuthority(authoritie));  });  return simpleGrantedAuthorities;  }**   /\*\*  \* 用户名  \*  \* @return  \*/  public String getUsername() {  return String.*valueOf*(userid);  }   /\*\*  \* 密码  \*  \* @return  \*/  @Override  public String getPassword() {  return this.userpwd;  }   @Override  public boolean isAccountNonExpired() {  return true;  }   @Override  public boolean isAccountNonLocked() {  return true;  }   @Override  public boolean isCredentialsNonExpired() {  return true;  }   @Override  public boolean isEnabled() {  return true;  } } |

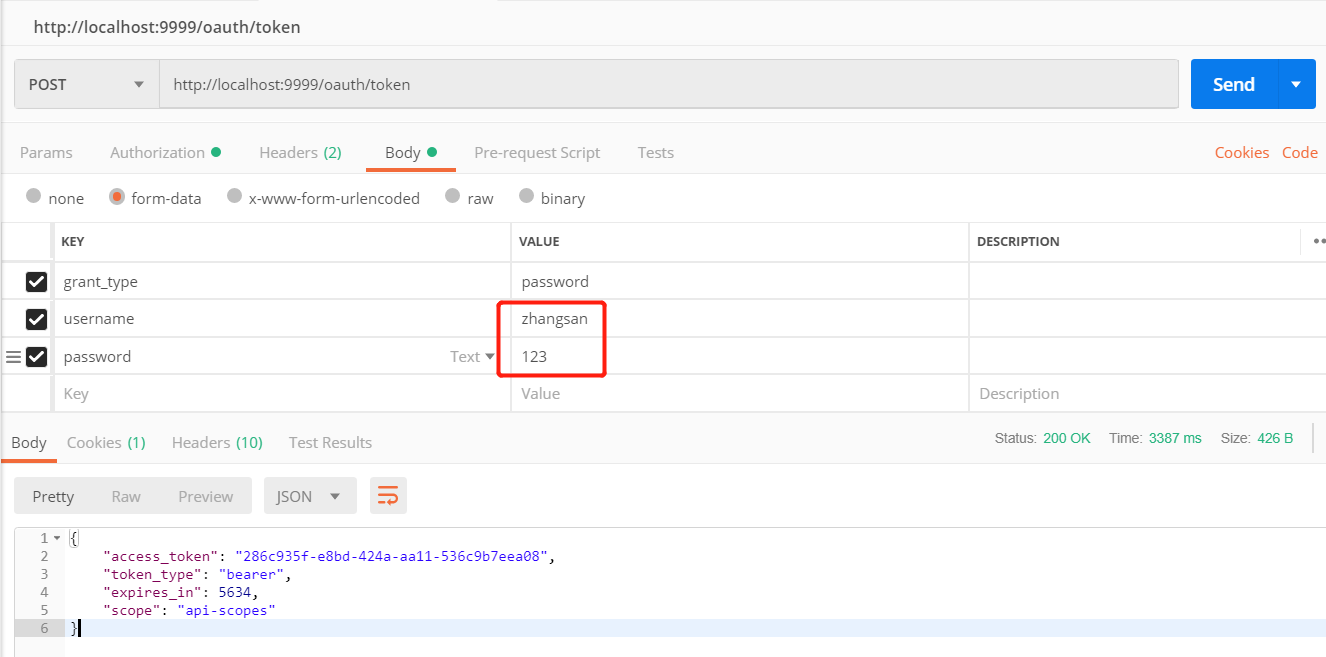
## 修改UserDetailServiceImpl

|  |
| --- |
| @Service public class UserDetailServiceImpl implements UserDetailsService {   public static final String *LOGIN\_TYPE* = "login\_type";   public static final String *SYS\_USER* = "sys\_user";   public static final String *MEMBER* = "member";   @Autowired  private MemberService memberService;   @Autowired  private SysUserService sysUserService;    @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  //从请求头中得到用户的类型，后台管理系统和会员都用这一套登录  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = requestAttributes.getRequest();  String loginType = request.getHeader(*LOGIN\_TYPE*);  if (StringUtils.*isEmpty*(loginType)) {  //如果为空，则直接给成会员的  loginType = *MEMBER*;  }  switch (loginType) {  case *SYS\_USER*:  //如果是系统用户，就查询系统表  SysUser sysUser = sysUserService.getOne(new LambdaQueryWrapper<SysUser>()  .eq(SysUser::getUsername, username)  );  if (!ObjectUtils.*isEmpty*(sysUser)) {  //如果查到了 查询用户的权限  List<String> permissions = getSysUserRoles(sysUser.getUserid());  if (!ObjectUtils.*isEmpty*(permissions)) {  //如果权限列表不为空，就放到sysUser里面  sysUser.setPermissions(permissions);  return sysUser;  }  }  break;  case *MEMBER*:  //如果是会员，就直接查询有没有用户  Member member = memberService.getOne(new LambdaQueryWrapper<Member>()  .eq(Member::getMembername, username)  );  return member;  default:  }  return null;  }   /\*\*  \* 描述: 查询系统用户权限的方法  \*  \* @param userid:  \* @return java.util.List<java.lang.String>  \*/  private List<String> getSysUserRoles(Integer userid) {  return sysUserService.selectPermissionByUserId(userid);  } } |

## 注意查询的sql

|  |
| --- |
| @Select(" select percode from sys\_permission t1 join sys\_role\_permission t2 on(t1.perid=t2.perid) join sys\_user\_role t3 on(t2.roleid=t3.roleid) where t3.userid = #{userid} ") |

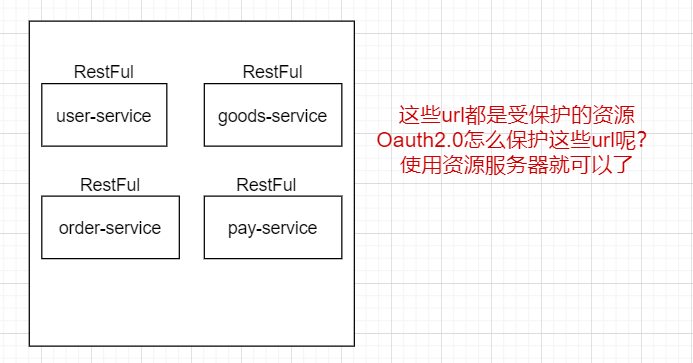
## 测试获取token



认证服务的本质在于给第三方应用颁发token

第三方应用使用token就能访问受保护的资源

# 受保护的资源



## 修改provider-order-service

### 添加依赖

|  |
| --- |
| <!-- 资源服务器也需要oauth2的依赖--> <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId> </dependency> |

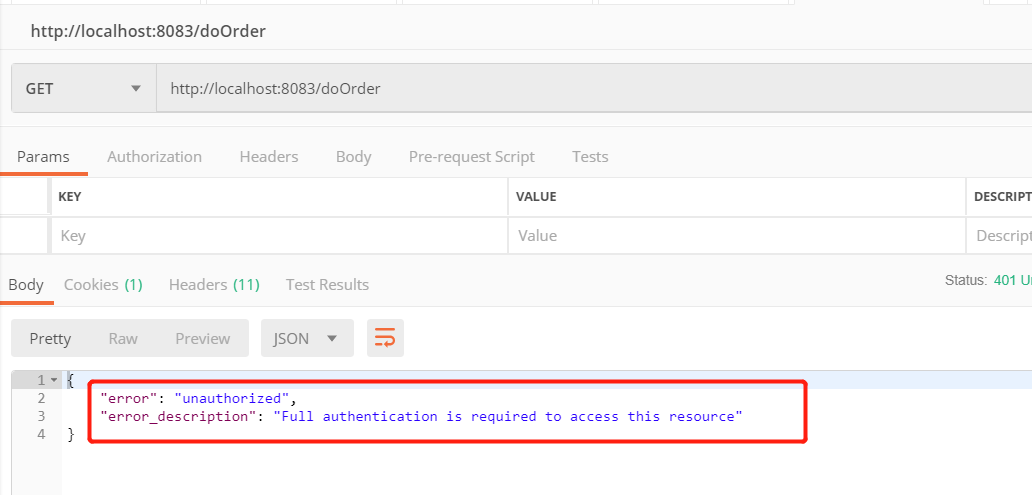
### 修改启动类

|  |
| --- |
| @SpringBootApplication @EnableResourceServer // 开启资源服务器 public class OrderServiceApplication {   public static void main(String[] args) {  SpringApplication.*run*(OrderServiceApplication.class, args);  }  } |

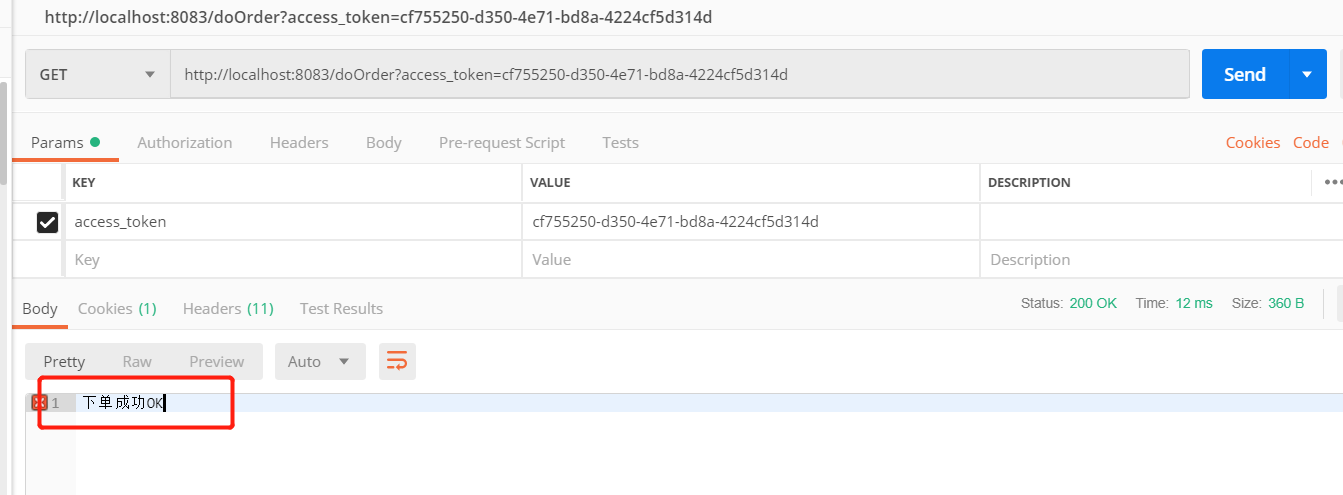
### 修改yml配置文件

|  |
| --- |
| security:  oauth2:  resource: #第三方访问资源服务器，但是资源服务器无法校验token，只有auth-server可以校验token，这里填写资源服务器换取用户的地址  user-info-uri: http://localhost:9999/info |

### 测试访问



加上token访问



### 权限和角色的拦截

**添加一个ResourceConfig类**

|  |
| --- |
| @Configuration @EnableResourceServer //开启资源服务器 @EnableGlobalMethodSecurity(prePostEnabled = true) //开启方法级别的验证 public class ResourceConfig extends ResourceServerConfigurerAdapter {  } |

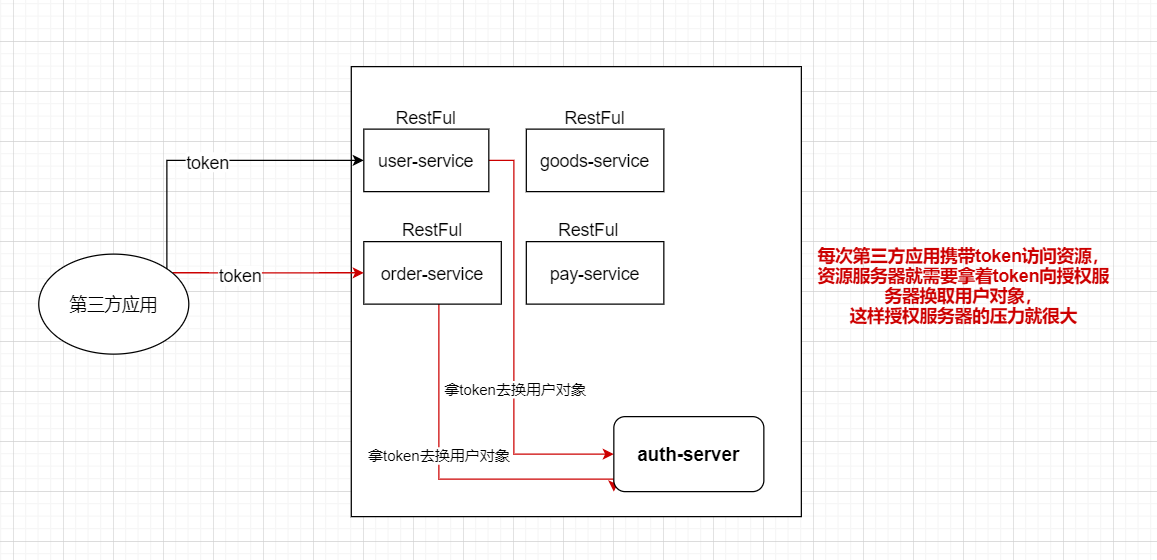
**修改OrderController**

|  |
| --- |
| @RestController public class OrderController {   /\*\*  \* 描述: 需要admin的权限才能下单成功  \*  \* @param :   \* @return java.lang.String  \*/  @RequestMapping("doOrder")  @PreAuthorize("hasAuthority('admin')")  public String doOrder(){  return "下单成功OK";  } } |

## 如何后端获取token的值

|  |
| --- |
| @GetMapping("tokenInfo") public String tokenInfo() {  //如何拿到token  //1.从request中获取  ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder.*getRequestAttributes*();  HttpServletRequest request = requestAttributes.getRequest();  String authorization = request.getHeader("Authorization");  //token  System.*out*.println(authorization);  String token = authorization.replaceAll("bearer ", "");  //2.从SecurityContextHolder中获取  Object principal = SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  System.*out*.println(principal);  Authentication authentication = SecurityContextHolder.*getContext*().getAuthentication();  //转成我们需要的  OAuth2AuthenticationDetails details = (OAuth2AuthenticationDetails) authentication.getDetails();  String tokenValue = details.getTokenValue();  System.*out*.println(tokenValue);  return tokenValue; } |

## 思考问题



授权服务器的压力太大了？为什么资源服务器需要访问授权服务器？

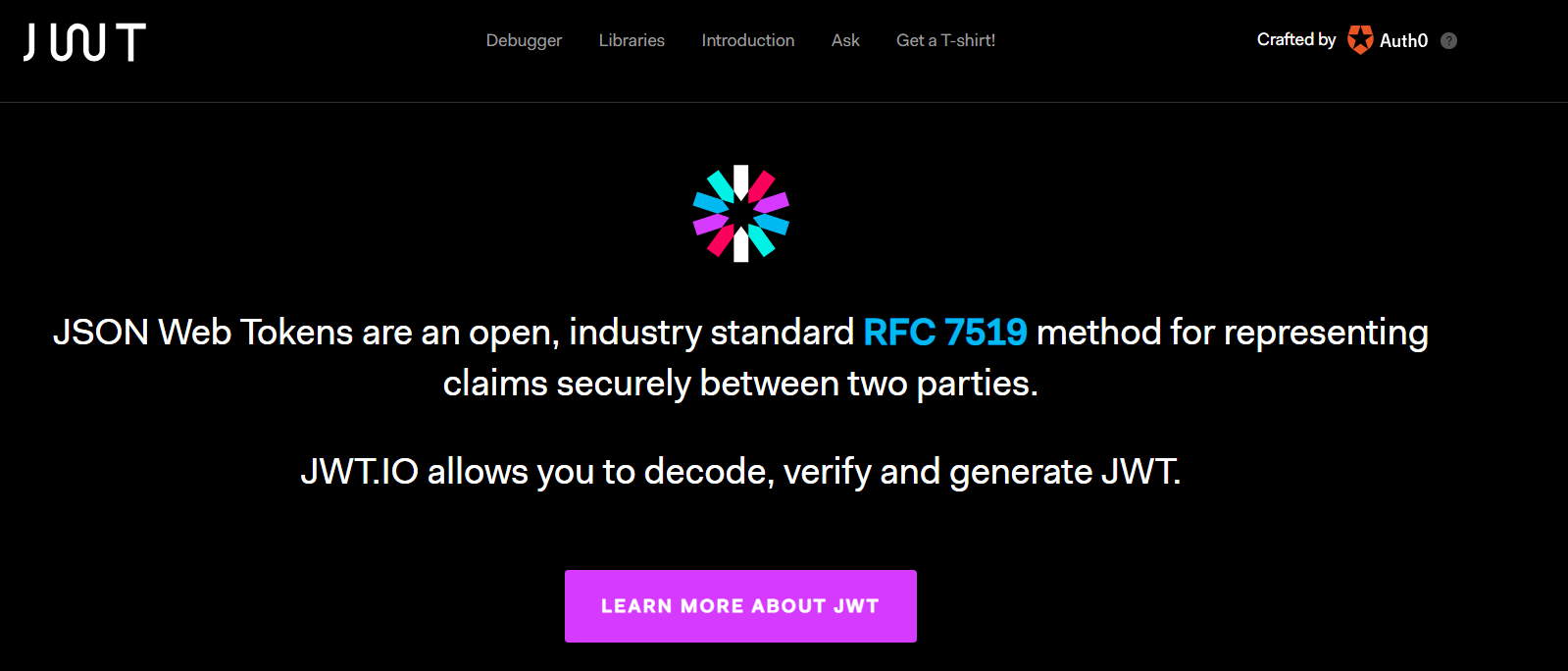
原因：**因为资源服务器里面无法校验token，资源服务器里面也无法根据token 获取到当前用户的权限**

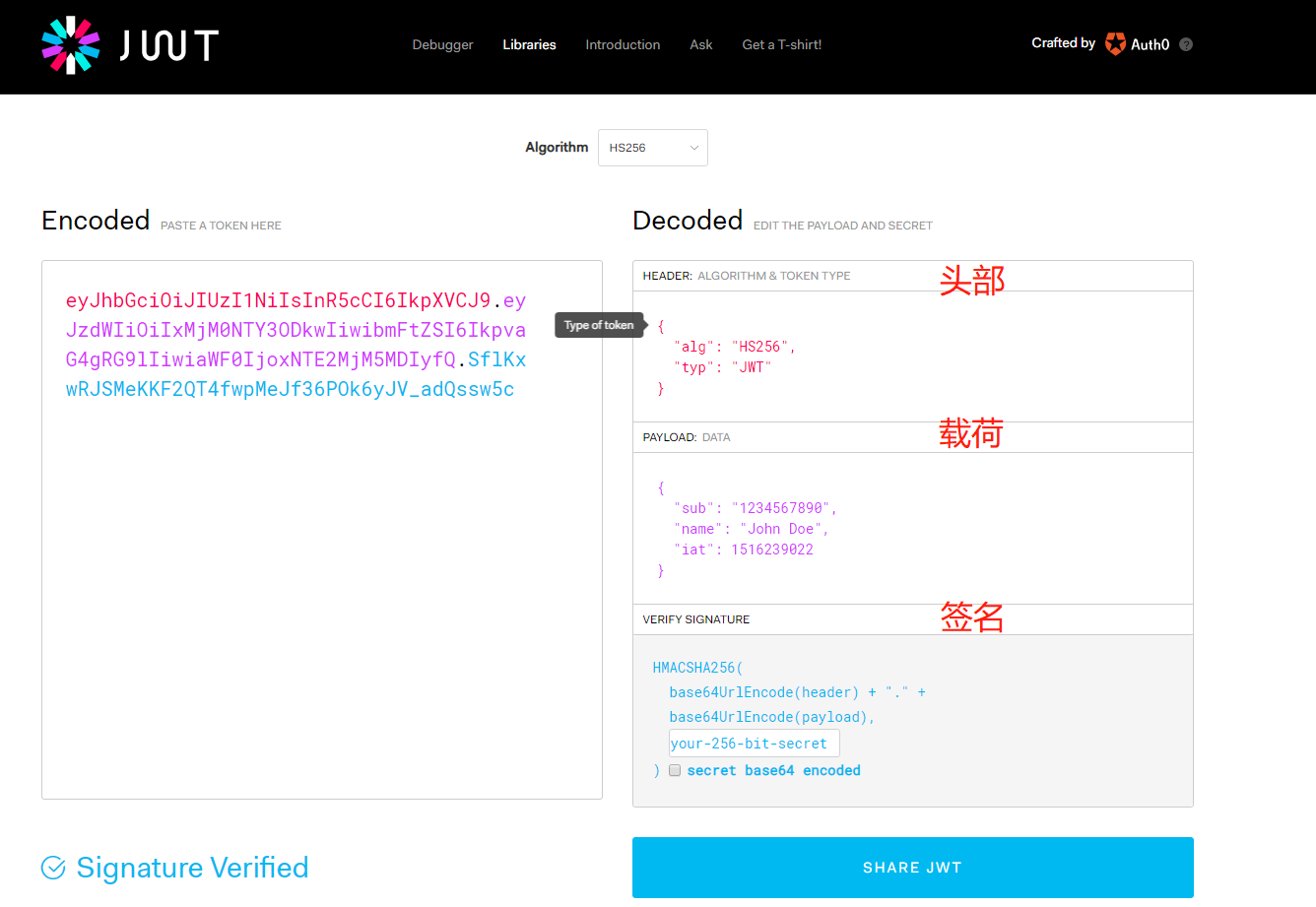
JWT:JSON WEB TOKEN

既然资源服务器需要拿着token访问授权服务得到用户和权限，我现在直接把当前的用户对象放在token 里面，让每个资源服务器可以直接解析这个token，拿到用户对象

# 引入JWT（Oauth2.0）

官网：<https://jwt.io/>





## 名词解释

JWT由三部分组成（头部，载荷，签名）

头部：放jwt的算法，由于是jwt的标准，我们无法修改

载荷：用户自定义的数据，这个用户可以自己修改

签名：安全策略，jwt的安全性，由签名保证

Jwt 就是一个字符串而已，怎么校验该字符串是否为我们自己颁发的字符串

Jwt 会使用签名校验我颁发的token 是否为我自己构造的token

## 生成自己JWT的token(在auth-server里)

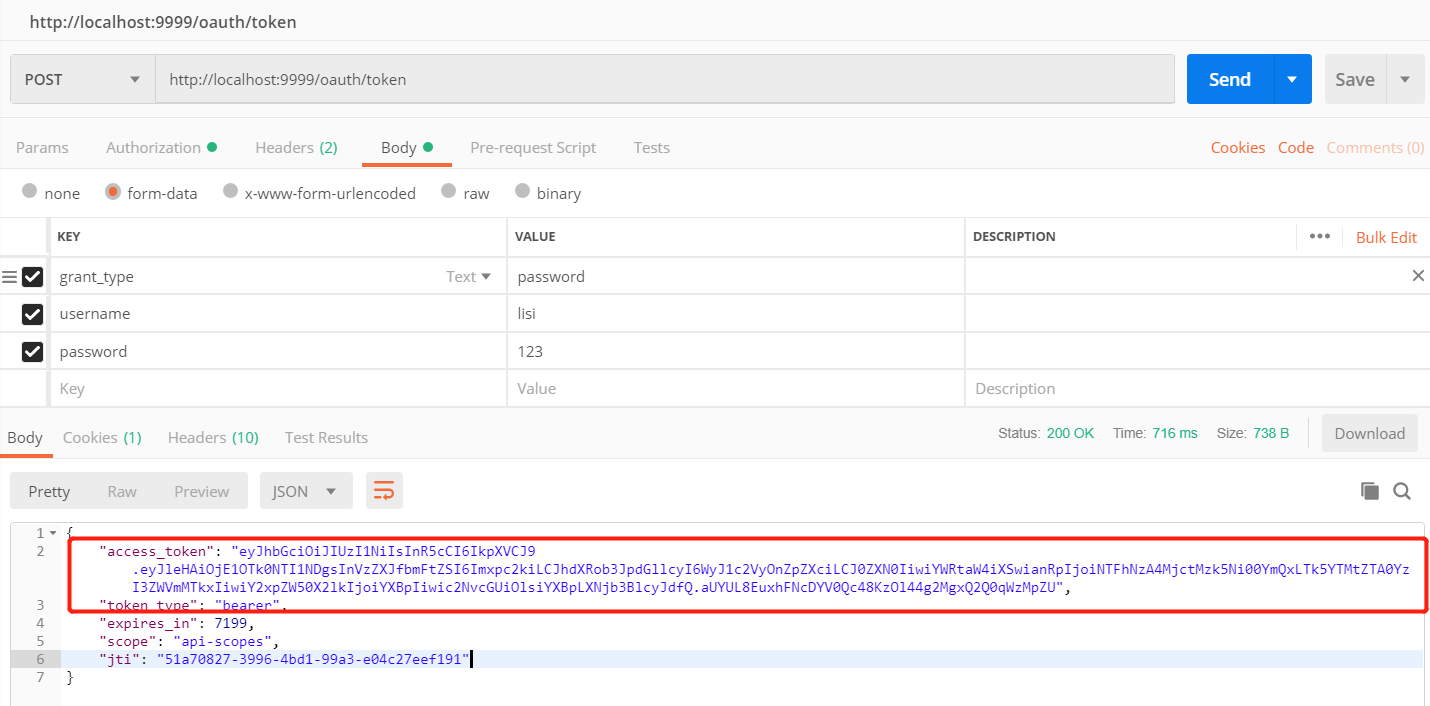
**由授权服务器生成token**

思考：该token 是否需要存储在redis 里面

### 修改authorizationConfig类（使用jwt）

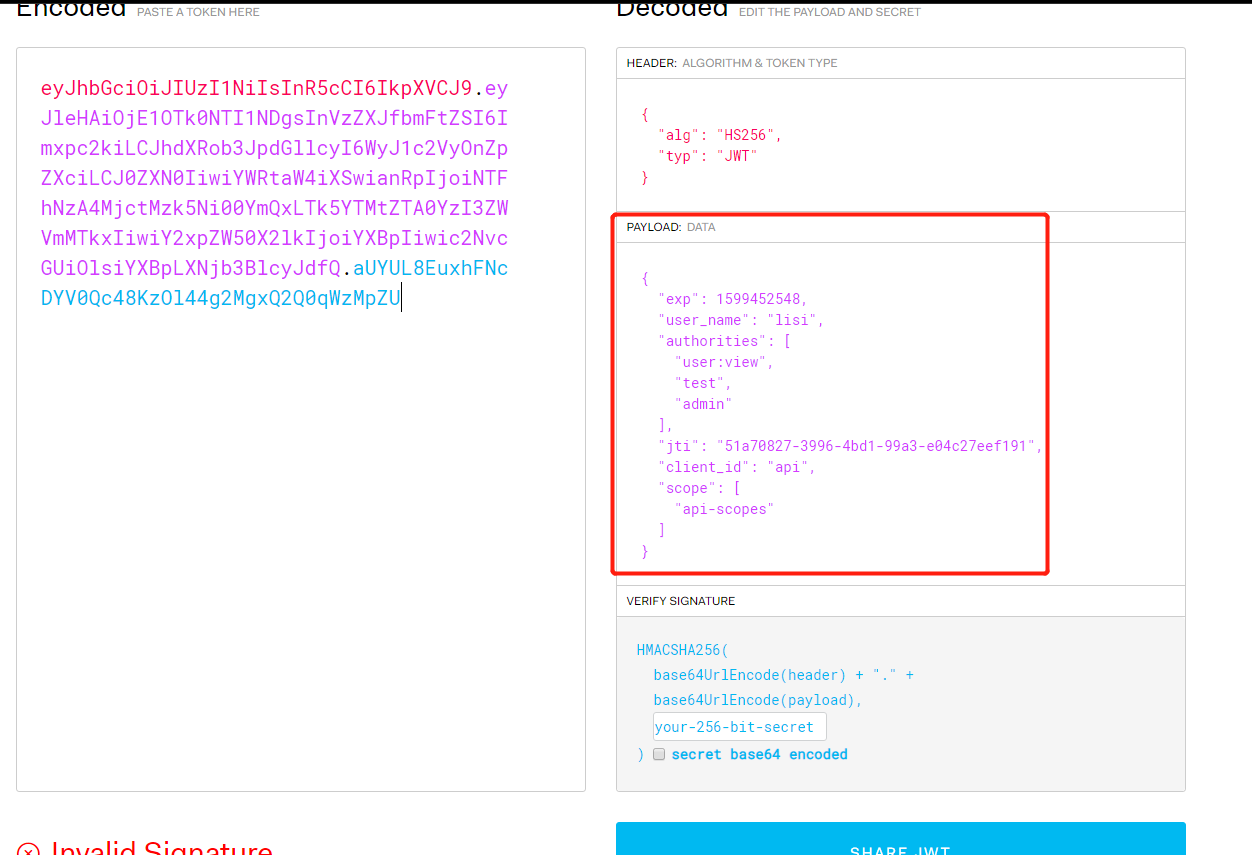
|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.security.authentication.AuthenticationManager; import org.springframework.security.crypto.password.PasswordEncoder; import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer; import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter; import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer; import org.springframework.security.oauth2.provider.token.TokenStore; import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter; import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;  @Configuration //标记为我们自己的配置类 public class AuthorizationConfig extends AuthorizationServerConfigurerAdapter {    **/\*\*  \* 描述: 注入jwt的token转换  \*  \* @param null:  \* @return  \*/  @Autowired  private JwtAccessTokenConverter jwtAccessTokenConverter;   /\*\*  \* 使用jwt存储token  \*  \* @return  \*/  @Bean  public TokenStore tokenStore() {  return new JwtTokenStore(jwtAccessTokenConverter);  }   /\*\*  \* 描述: 创建jwt的token转换器，签名先试用对称加密  \*  \* @param :  \* @return org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter  \*/  @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  jwtAccessTokenConverter.setSigningKey("sxt-oauth2.0");  return jwtAccessTokenConverter;  }**   @Autowired  private PasswordEncoder passwordEncoder;   /\*\*  \* 描述: 注入认证管理器  \*  \* @param null:  \* @return  \*/  @Autowired  private AuthenticationManager authenticationManager;   /\*\*  \* 描述: 第三方应用的配置，只有配置了第三方应用，才能访问授权服务器  \*  \* @param clients:  \* @return void  \*/  @Override  public void configure(ClientDetailsServiceConfigurer clients) throws Exception {  //创建一个基本的第三方应用，第三方应用少的时候，就放在内存中，多的就使用jdbc  clients.inMemory()  .withClient("web") //创建一个web第三方  .secret(passwordEncoder.encode("web-secret")) //密码  .scopes("web-scopes") //作用域  .authorizedGrantTypes("authorization\_code") //授权方式一共有四种，这里是验证码授权，还有密码授权，静默授权，客户端授权  .redirectUris("https://www.baidu.com") //访问成功后的跳转地址  .accessTokenValiditySeconds(7200)  .and()  .withClient("ios")  .secret(passwordEncoder.encode("ios-secret"))  .scopes("ios-scopes")  .authorizedGrantTypes("implicit")//静默授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200)  .and()  .withClient("api")  .secret(passwordEncoder.encode("api-secret"))  .scopes("api-scopes")  .authorizedGrantTypes("password") //密码授权  .redirectUris("https://www.baidu.com")  .accessTokenValiditySeconds(7200) //token的过期时间给2小时  .and()  .withClient("client")  .secret(passwordEncoder.encode("client-secret"))  .scopes("client-scopes")  .authorizedGrantTypes("client\_credentials") //客户端授权  .accessTokenValiditySeconds(7200);    }   @Autowired  private UserDetailServiceImpl userDetailService;   /\*\*  \* 描述: token的存储方式 我们先存在redis里面  \*  \* @param endpoints:  \* @return void  \*/  @Override  public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {  //使用jwt做token存储  endpoints.tokenStore(tokenStore())  .userDetailsService(userDetailService)  .authenticationManager(authenticationManager)   **.accessTokenConverter(jwtAccessTokenConverter)**  ;    } } |

### 访问生成一个token看看



### 放在JWT官网解析看一下

JWT 里面不能放私密的信息，因为客户端可以对他解密，头部和载荷，可以解密



## 对provider-order-service资源服务器改造

### 修改ResourceConfig类（使用jwt）

之前：我们资源服务器需要发请求访问授权服务器获取用户(使用token 换取user 对象)

现在：因为token 自带用户对象，我们只要验证jwt的正确性，以及从jwt 里面获取到权限信息就可以了

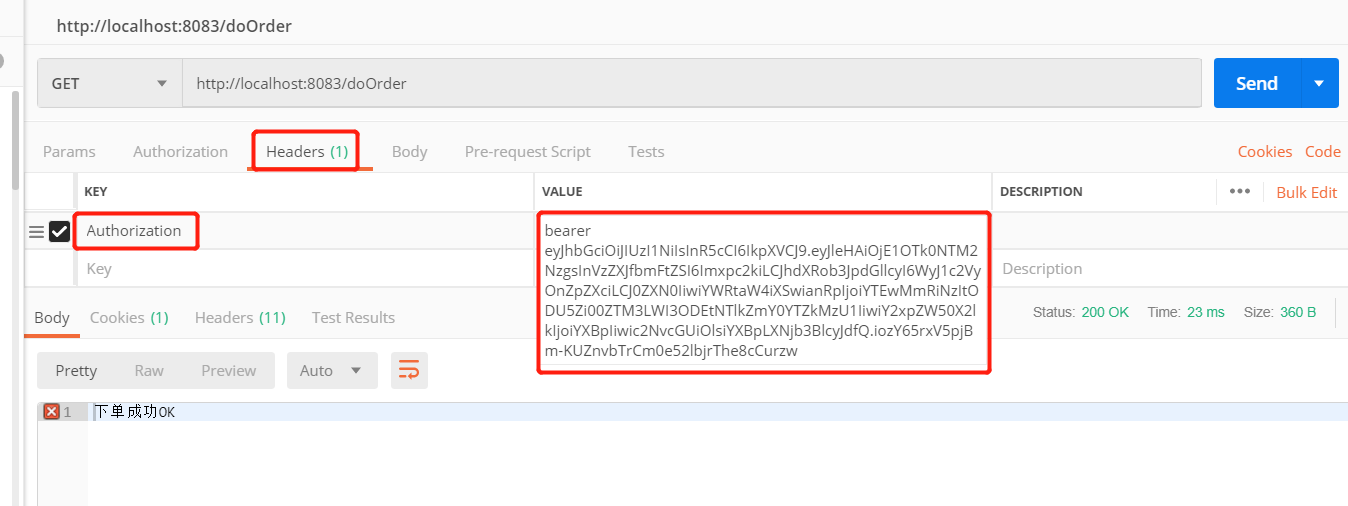
|  |
| --- |
| import org.springframework.beans.factory.annotation.Autowired; import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration; import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity; import org.springframework.security.config.annotation.web.builders.HttpSecurity; import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter; import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer; import org.springframework.security.oauth2.provider.token.TokenStore; import org.springframework.security.oauth2.provider.token.store.JwtAccessTokenConverter; import org.springframework.security.oauth2.provider.token.store.JwtTokenStore;  @Configuration @EnableGlobalMethodSecurity(prePostEnabled = true) //开启方法级别的拦截验证 public class ResourceConfig extends ResourceServerConfigurerAdapter {   @Autowired  @Qualifier("jwtTokenStore")  private TokenStore tokenStore;    /\*\*  \* 创建jwt转换器 对称加密  \*  \* @return  \*/  @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  jwtAccessTokenConverter.setSigningKey("sxt-oauth2.0");  return jwtAccessTokenConverter;  }   /\*\*  \* 创建jwtTokenStore token存储  \*  \* @param jwtAccessTokenConverter  \* @return  \*/  @Bean("jwtTokenStore")  public TokenStore jwtTokenStore(JwtAccessTokenConverter jwtAccessTokenConverter) {  return new JwtTokenStore(jwtAccessTokenConverter);  }   /\*\*  \* 描述: 解析token的位置  \*  \* @param resources:  \* @return void  \*/  @Override  public void configure(ResourceServerSecurityConfigurer resources) throws Exception {  resources.tokenStore(tokenStore);    }  } |

### 修改yml配置文件



### 测试访问

在请求头里面添加一个key：Authorization，value是bearer开头，记住bearer后面有一个空格，再加上jwt



## JWT的安全性问题

Jwt的安全验证靠签名，只有签名，我就可以颁发token

签名用来生成token时做加密，**如果签名泄露了，别人也可以颁发token**

使用签名加密解密，我们把它称为**对称加密**

使用公钥和私钥，它的安全性将很高

在支付宝，我们有个公钥，有个私钥

ssh 也是公钥，也有私钥

非对称加密：

**私钥产生一个token ，公钥能验证该token**，现在公钥泄露了，对我们没有影响，因为公钥无法生成token ，只有验证token

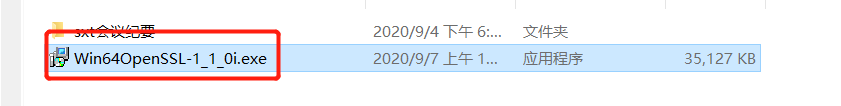
在jwt 里面，怎么使用公钥和私钥的非对称加密？

在java 里面，怎么产生公钥和私钥

## 在java中生成公钥和秘钥

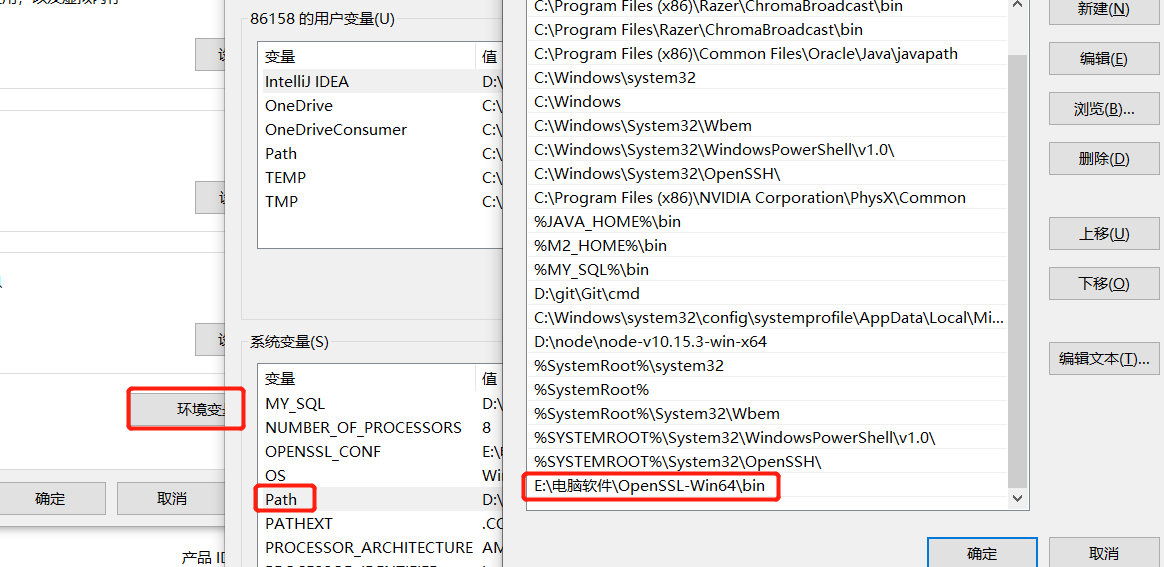
### 安装ssl软件

<https://oomake.com/download/openssl>



### 配置环境变量

因为我们要使用keytool命令



### 生成一个私钥

|  |
| --- |
| keytool -genkeypair -alias cxs-jwt -validity 3650 -keyalg RSA -dname "CN=jwt,OU=jtw,O=jwt,L=zurich,S=zurich,C=CH" -keypass cxs123 -keystore cxs-jwt.jks -storepass cxs123 |



### 得到一个公钥

|  |
| --- |
| keytool -list -rfc --keystore cxs-jwt.jks | openssl x509 -inform pem -pubkey |





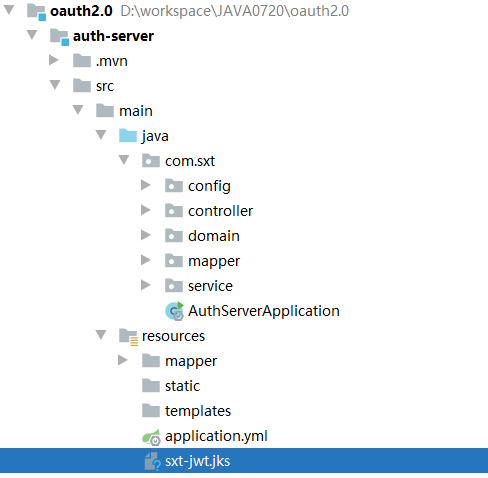
复制出来处理

|  |
| --- |
| -----BEGIN PUBLIC KEY-----  MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA5EqP4HTpezvtILLRMHpp  VZQs8Ti2nxDLhZV1wVeNUUPmVjbODZiS4P01iDIX7SBhyvKmhHGKc7BUWNCUGGtF  n0zQIk6Gb60ajIOO3tyigFT1owlN8Ctra8bFBvqUgl6FZ4VezGORT8CuINmBOwRq  fUU8QFouIGghCMd4hkhnV4BzmXbrp5RPvUL3tLl3r0HVstiNyv0fWdIjLqt4SV+X  FF7MFLA3zVfsbm6SL7mTVygRqlBA19jL0iYI+vZGW3o7v5zZKMq+Ysn0APh6Q5ZA  CmLKqEFf5NQGagUKftct0WvjdKY+WFS+hcCW/rQB5YvzkiujV3g2E3cmICCraT0q  BQIDAQAB  -----END PUBLIC KEY----- |

## 改造我们的项目，使用非对称加密（auth-server）

使用私钥生成token：在auth-server 授权服务器里面

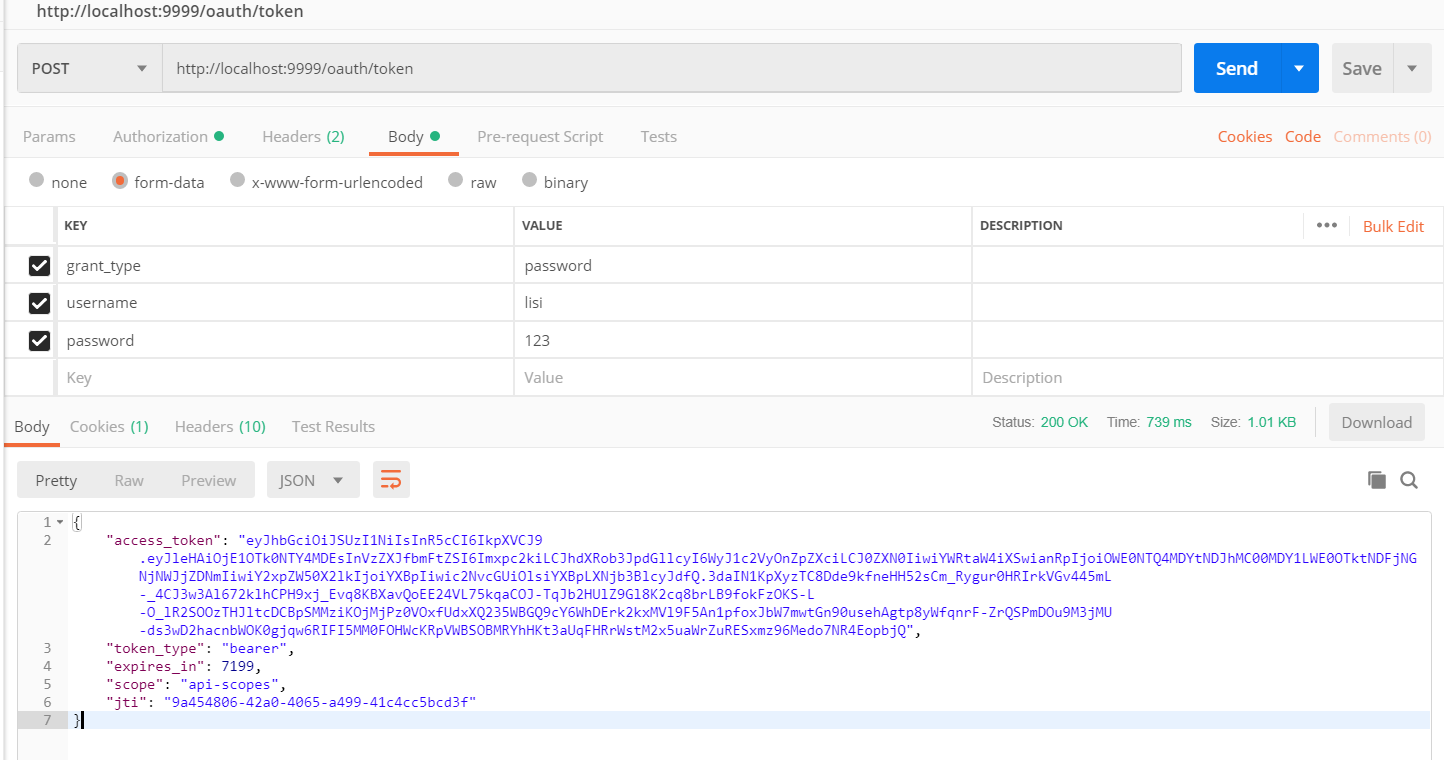
### 复制私钥到auth-server 的resourece 文件夹里面



### 修改AuthorizationConfig，使用非对称加密

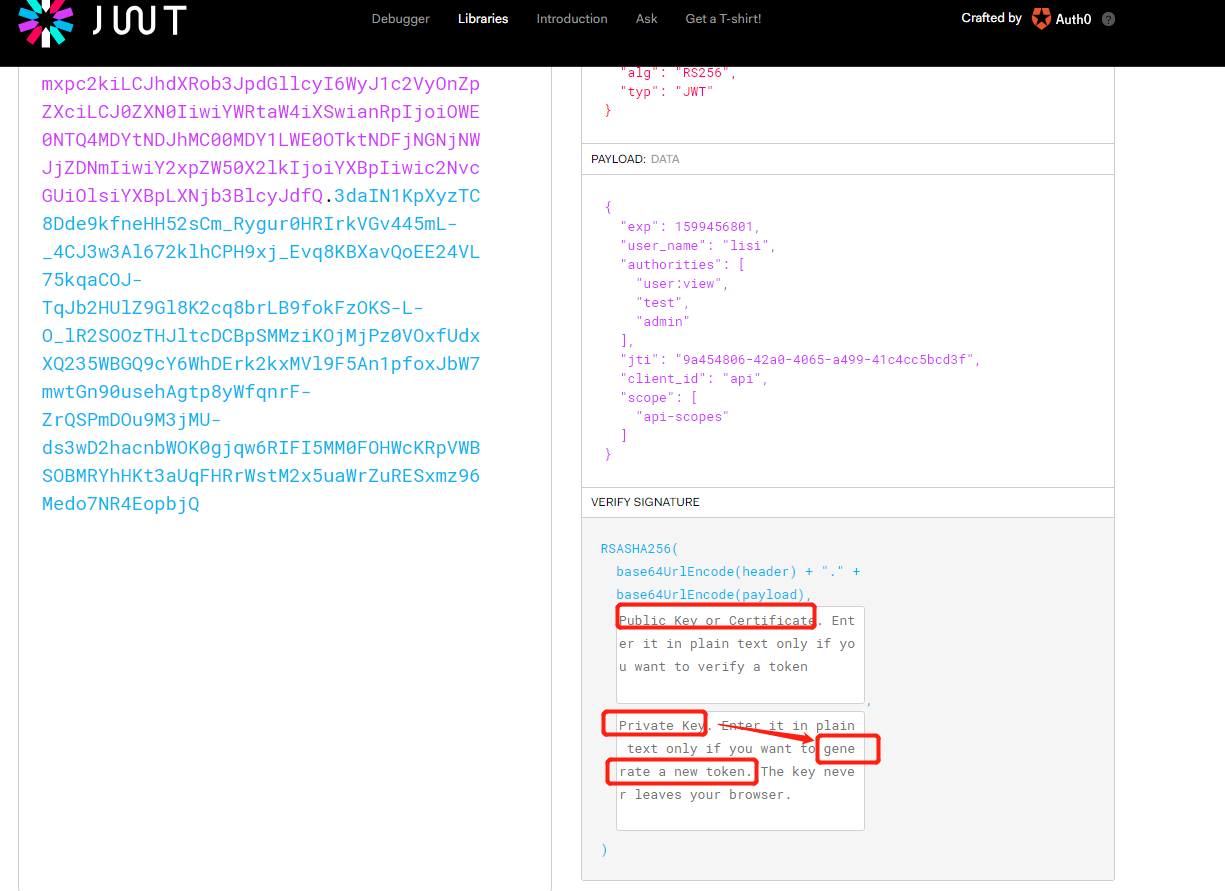
|  |
| --- |
| @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  //使用非对称加密，只要私钥不泄密，别人无法生成token，公钥只能验证token  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  //将文件加载进来  ClassPathResource resource = new ClassPathResource("cxs-jwt.jks");  //得到keyStore  KeyStoreKeyFactory keyFactory = new KeyStoreKeyFactory(resource, "cxs123".toCharArray());  //往转换器里设置私钥  KeyPair keyPair = keyFactory.getKeyPair("cxs-jwt");  jwtAccessTokenConverter.setKeyPair(keyPair); // jwtAccessTokenConverter.setSigningKey("sxt-oauth2.0");  return jwtAccessTokenConverter;  } |

### 再生成一个token看看



### 放在JWT官网看看

签名部分提示，公钥用来验证token，私钥才能生成新的token



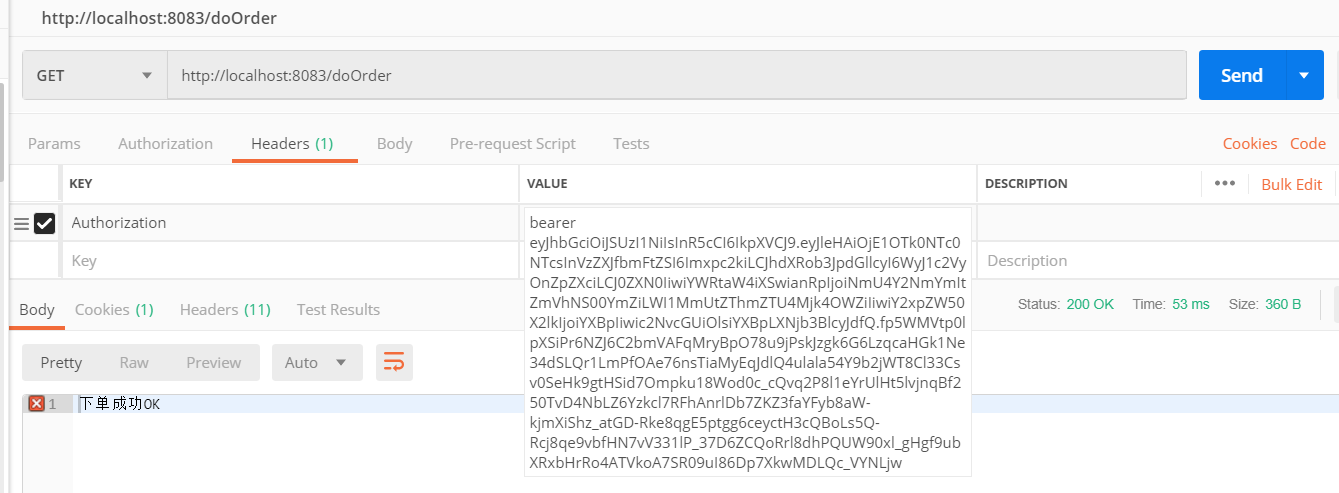
## 改造provider-order-service（非对称加密）

### 复制公钥到order-service 的resourece 文件夹里面

### 修改ResourceConfig，使用非对称加密

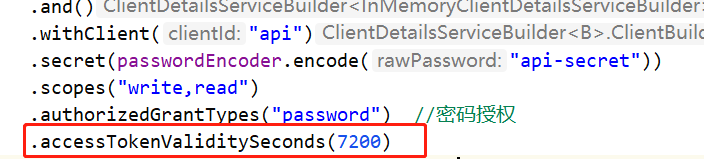
|  |
| --- |
| @Bean  public JwtAccessTokenConverter jwtAccessTokenConverter() {  JwtAccessTokenConverter jwtAccessTokenConverter = new JwtAccessTokenConverter();  //读到公钥  ClassPathResource resource = new ClassPathResource("jwt-publicKey.txt");  try {  String publicKey = FileUtil.*readString*(resource.getFile(), Charset.*defaultCharset*());  //设置验证  jwtAccessTokenConverter.setVerifierKey(publicKey);  } catch (IOException e) {  e.printStackTrace();  } // jwtAccessTokenConverter.setSigningKey("sxt-oauth2.0");  return jwtAccessTokenConverter;  } |

### 测试访问



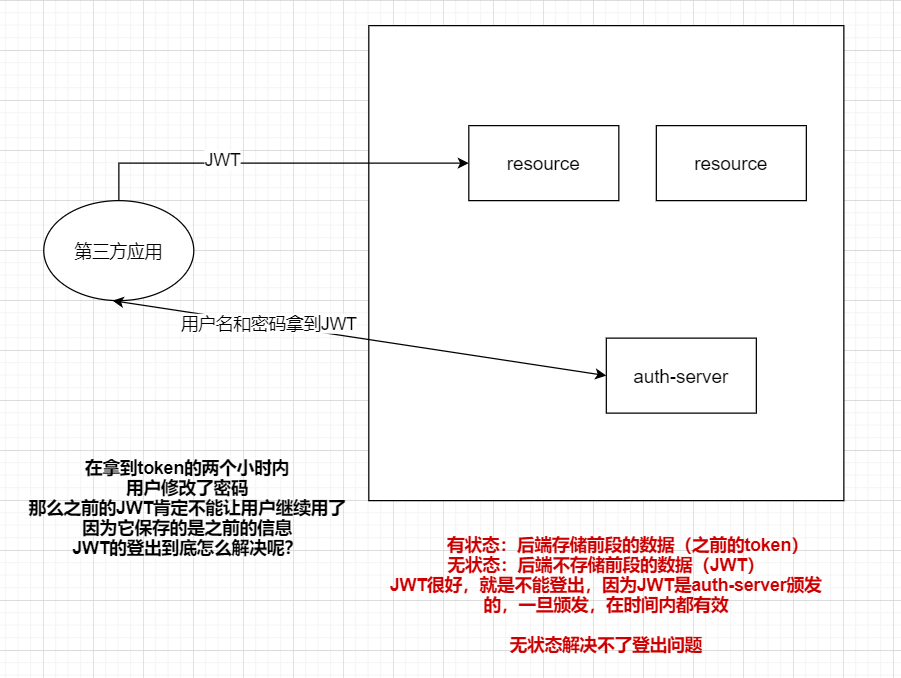
# JWT的登出问题

场景：修改了用户名或密码，我怎么不让它使用之前的jwt访问了？

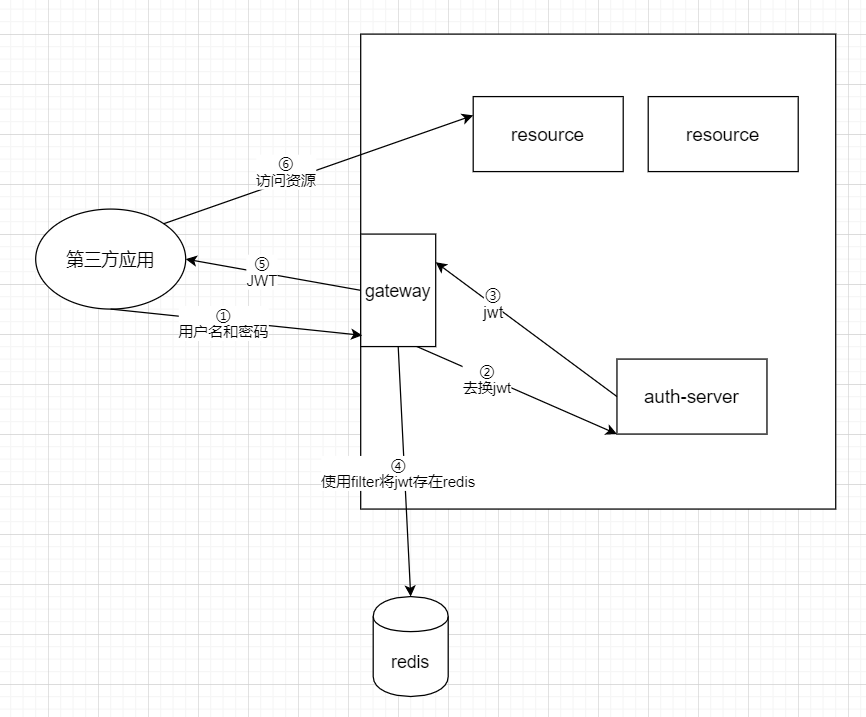


在2 个小时之内，该token 有效。

在2 个小时之内，我把用户名和密码修改了，怎么让该token 失效！



## 解决思路



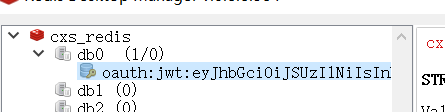
在用户登录以后，获取一个JWT，我们在网关这里层将JWT存在redis里面，当用户登出以后，我们直接删掉redis里面的JWT，就可以了

### 在网关中写一个GatewayConfig类

请求登录的时候，拿到token，然后放在redis里面

|  |
| --- |
| @Configuration public class GatewayConfig {   @Autowired  private StringRedisTemplate redisTemplate;   public static final String *OAUTH\_PREFIX* = "oauth:jwt:";    @Bean  public RouteLocator routes(RouteLocatorBuilder builder) {  return builder.routes()  .route("auth-server-router", r -> r.path("/oauth/\*\*")  .filters(f -> f.modifyResponseBody(String.class, String.class,  (exchange, s) -> {  String path = exchange.getRequest().getURI().getPath();  //s就是相应的值，我们解析后存入redis  if ("/oauth/token".equals(path)) {  JSONObject jsonObject = JSONUtil.*parseObj*(s);  String access\_token = jsonObject.getStr("access\_token");  Long expires\_in = jsonObject.getLong("expires\_in");  redisTemplate.opsForValue().set(*OAUTH\_PREFIX* + access\_token, "", Duration.*ofSeconds*(expires\_in));  }  return Mono.*just*(s);  })).uri("lb://auth-server"))  .build();  } } |

### 请求登录后查看redis



### 在网关中写一个检查token的类

每次用户请求都先走网关，有token，并且redis里面也有，就放行，没有token就报401

|  |
| --- |
| @Component public class JwtCheckFilter implements GlobalFilter, Ordered {   @Autowired  private StringRedisTemplate redisTemplate;   //放行的路径  public static final List<String> *ALLOW\_PATH* = Arrays.*asList*("/oauth/token");   public static final String *AUTHORIZATION* = "Authorization";    /\*\*  \* 描述: 检验请求是否携带token  \*  \* @param exchange:  \* @param chain:  \* @return reactor.core.publisher.Mono<java.lang.Void>  \*/  @SneakyThrows  @Override  public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {  ServerHttpRequest request = exchange.getRequest();  String path = request.getURI().getPath();  if (*ALLOW\_PATH*.contains(path)) {  return chain.filter(exchange);  }  //验证有没有token  List<String> list = request.getHeaders().get(*AUTHORIZATION*);  if (!ObjectUtils.*isEmpty*(list)) {  String authorization = list.get(0);  String token = authorization.replaceAll("bearer ", "");  if (!StringUtils.*isEmpty*(token) && redisTemplate.hasKey("oauth:jwt:" + token)) {  return chain.filter(exchange);  }  }  ServerHttpResponse response = exchange.getResponse();  //如果没有token或者redis里面没有token，就返回401  Map<String, Object> map = new HashMap<>();  map.put("code", HttpStatus.*UNAUTHORIZED*.value());  map.put("msg", "非法访问");  response.getHeaders().add("content-Type", "application/json;charset=UTF-8");  ObjectMapper objectMapper = new ObjectMapper();  byte[] bytes = objectMapper.writeValueAsBytes(map);  DataBuffer buffer = response.bufferFactory().wrap(bytes);  return response.writeWith(Mono.*just*(buffer));  }   @Override  public int getOrder() {  return 0;  } } |

问题：搭建OAuth2.0标准的授权服务器+REST资源服务器，并提供用户注册服务和REST资源访问接口。这个怎么去实现？

怎么实现在token有效期内，client可以在user第二次连接时候，避免再次授权