# 分布式服务认证(oauth2.0)

# 技术架构

springBoot+springSecurity+springCloud+oauth2.0+jwt

# 环境搭建

## 搭建父工程distributed-security

### 编写pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.2.3.RELEASE</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.liu.distributedSecurity</groupId>  <artifactId>distributed-security</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>distributed-security</name>  <description>分布式服务认证</description>   <properties>  <java.version>1.8</java.version>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  </properties>   <dependencyManagement>  <dependencies>  <!--springCloud的依赖-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-dependencies</artifactId>  <version>Greenwich.RELEASE</version>  <type>pom</type>  <scope>import</scope>  </dependency>  <!--servlet-api的依赖-->  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>javax.servlet-api</artifactId>  <version>4.0.1</version>  <scope>provided</scope>  </dependency>  <!--jwt依赖-->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  <version>1.0.10.RELEASE</version>  </dependency>  <!--oauth2的依赖-->  <dependency>  <groupId>org.springframework.security.oauth</groupId>  <artifactId>spring-security-oauth2</artifactId>  <version>2.3.6.RELEASE</version>  </dependency>  <dependency>  <groupId>org.springframework.security.oauth.boot</groupId>  <artifactId>spring-security-oauth2-autoconfigure</artifactId>  <version>2.1.3.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency>   <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  <exclusions>  <exclusion>  <groupId>org.junit.vintage</groupId>  <artifactId>junit-vintage-engine</artifactId>  </exclusion>  </exclusions>  </dependency>  </dependencies>  </dependencyManagement>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

## 搭建common子工程

用于存放公共的工具类，实体类，service及mapper

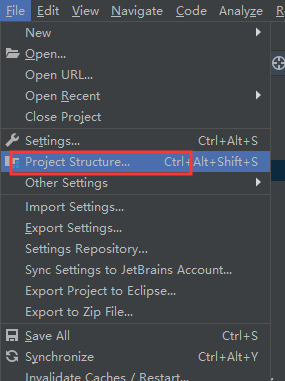
### 创建common子工程

### 编写公共的工具类，实体类，service及mapper

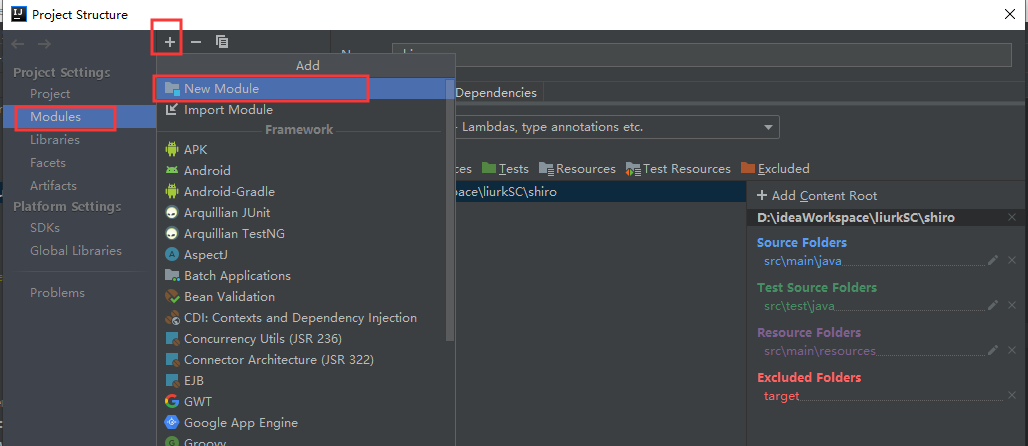
## 搭建uaa授权服务子工程

### 2.3.1 创建uaa授权服务子工程

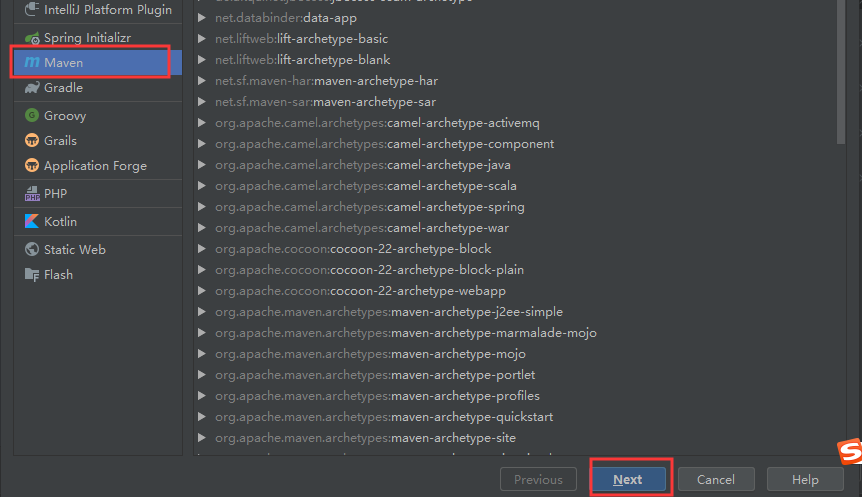
file --> Project Structure



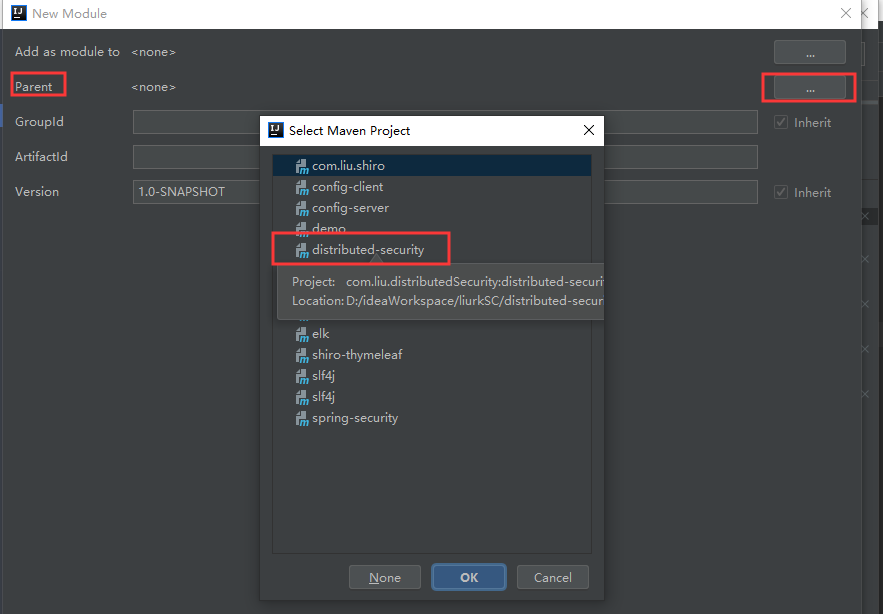
选择Modules，点击+号，选择New Module



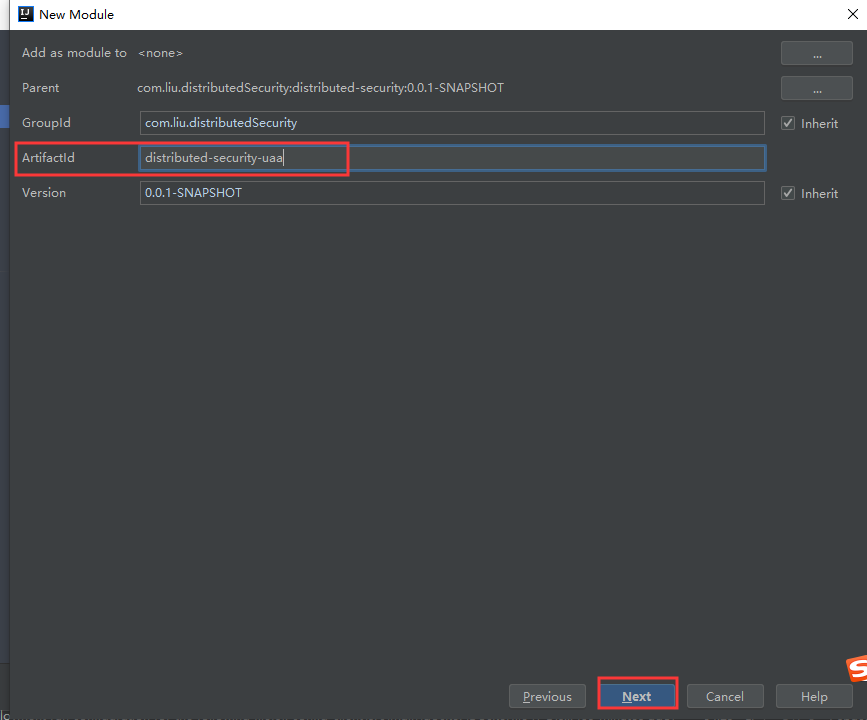
选择maven，点击next



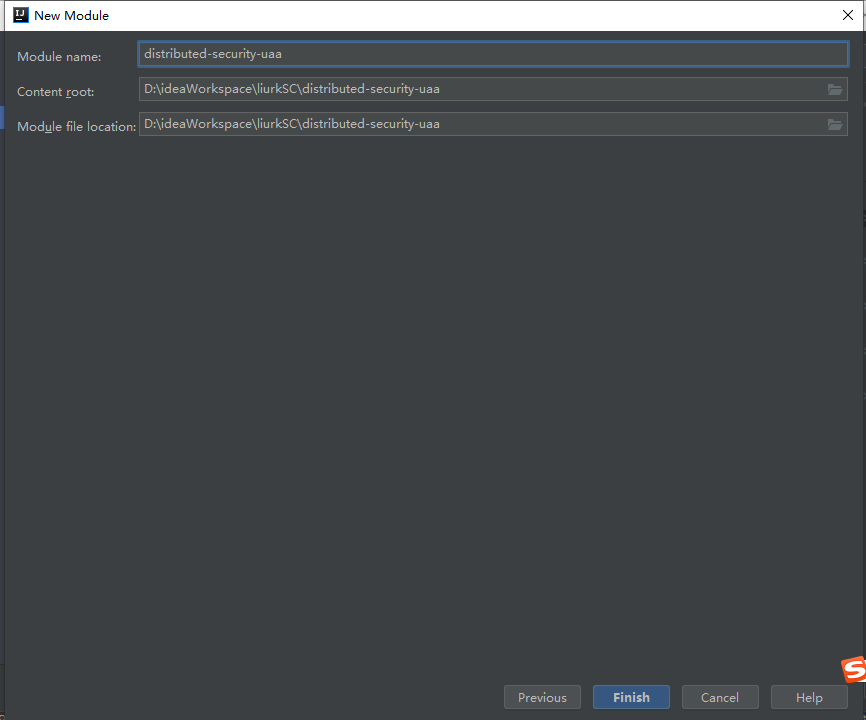
parent选择我们的父工程，点击ok



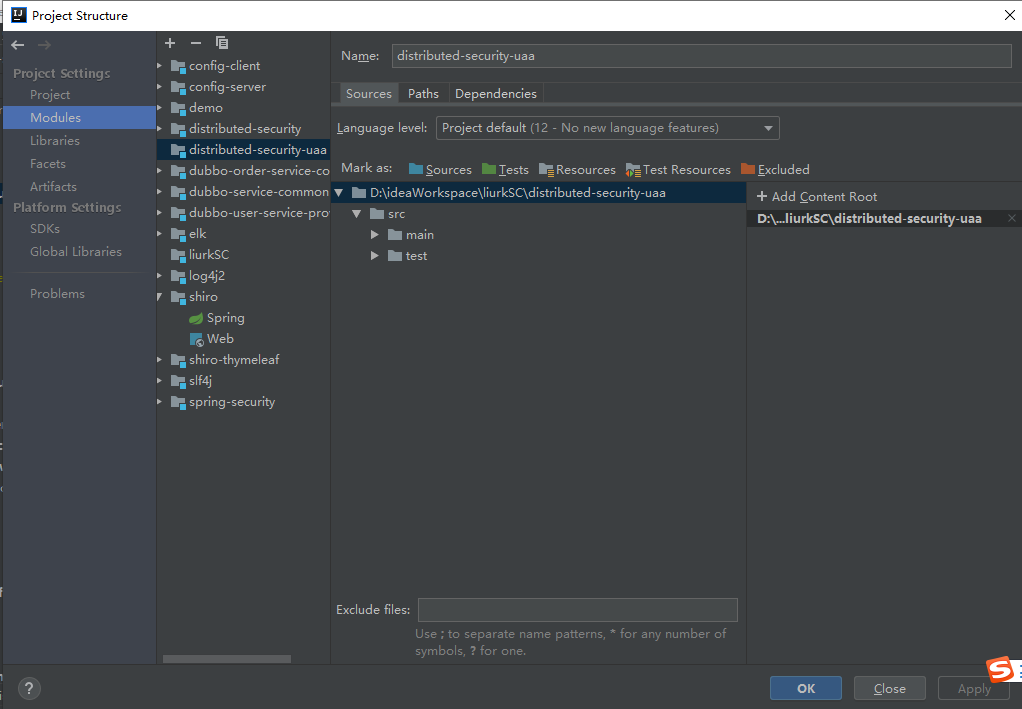
输入Artifactld(distributed-security-uaa)，点击Next



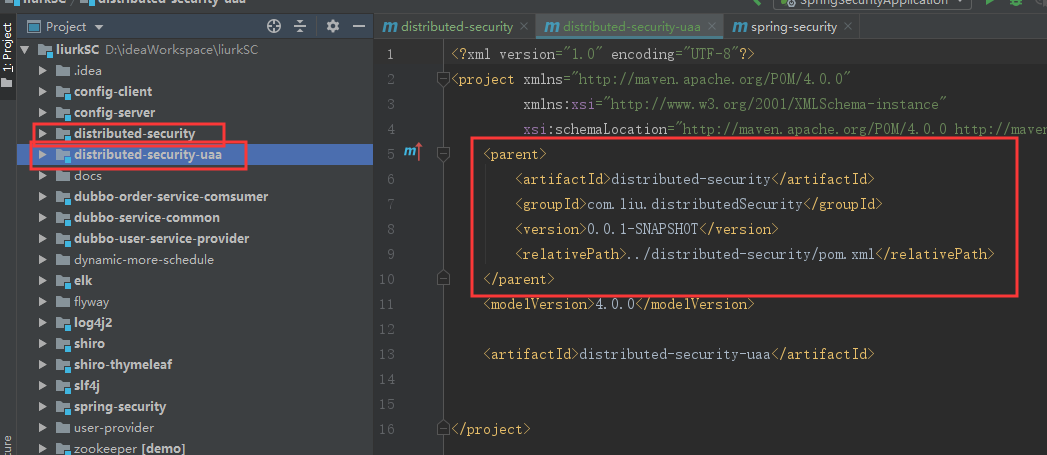
修改module name，点击finish



点击apply，再点击ok



至此，父子工程创建完毕



### 2.3.2 编写pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>distributed-security</artifactId>  <groupId>com.liu.distributedSecurity</groupId>  <version>0.0.1-SNAPSHOT</version>  <relativePath>../distributed-security/pom.xml</relativePath>  </parent>  <modelVersion>4.0.0</modelVersion>   <artifactId>distributed-security-uaa</artifactId>   <!--子工程里面的依赖不用写版本，因为都依赖父工程里面的版本-->  <dependencies>  <!--依赖common模块-->  <dependency>  <groupId>com.liu.distributedSecurity</groupId>  <artifactId>distributed-security-common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!--eureka-client服务注册中心客户端-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-netflix-eureka-client</artifactId>  </dependency>  <!--hystrix熔断-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-netflix-hystrix</artifactId>  </dependency>  <!--ribbon负载均衡-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency>  <!---->  <dependency>  <groupId>com.netflix.hystrix</groupId>  <artifactId>hystrix-javanica</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.retry</groupId>  <artifactId>spring-retry</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-commons</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-security</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  </dependency>  <!---->  <!--<dependency>  <groupId>javax.interceptor</groupId>  <artifactId>javax.interceptor.api</artifactId>  </dependency>-->  <!--mysql数据库-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <!--mybatis-plus插件-->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.1.0</version>  </dependency>  <!--数据库连接池-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.6</version>  </dependency>  </dependencies>  </project> |

### 2.3.3 编写UaaApplication启动类和application.yml配置文件

## 2.4 搭建order服务资源子工程

### 2.4.1 创建order服务资源子工程

### 2.4.2 编写pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <parent>  <artifactId>distributed-security</artifactId>  <groupId>com.liu.distributedSecurity</groupId>  <version>0.0.1-SNAPSHOT</version>  <relativePath>../distributed-security/pom.xml</relativePath>  </parent>  <modelVersion>4.0.0</modelVersion>   <artifactId>distributed-security-order</artifactId>   <!--子工程里面的依赖不用写版本，因为都依赖父工程里面的版本-->  <dependencies>  <!--依赖common模块-->  <dependency>  <groupId>com.liu.distributedSecurity</groupId>  <artifactId>distributed-security-common</artifactId>  <version>0.0.1-SNAPSHOT</version>  </dependency>  <!--eureka-client服务注册中心客户端-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-netflix-eureka-client</artifactId>  </dependency>  <!--hystrix熔断-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-netflix-hystrix</artifactId>  </dependency>  <!--ribbon负载均衡-->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId>  </dependency>  <!---->  <dependency>  <groupId>com.netflix.hystrix</groupId>  <artifactId>hystrix-javanica</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.retry</groupId>  <artifactId>spring-retry</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-freemarker</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.data</groupId>  <artifactId>spring-data-commons</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-security</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-oauth2</artifactId>  </dependency>  <!---->  <dependency>  <groupId>org.springframework.security</groupId>  <artifactId>spring-security-jwt</artifactId>  </dependency>  <!---->  <!--<dependency>  <groupId>javax.interceptor</groupId>  <artifactId>javax.interceptor.api</artifactId>  </dependency>-->  <!--mysql数据库-->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <!--mybatis-plus插件-->  <dependency>  <groupId>com.baomidou</groupId>  <artifactId>mybatis-plus-boot-starter</artifactId>  <version>3.1.0</version>  </dependency>  <!--数据库连接池-->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>druid</artifactId>  <version>1.1.6</version>  </dependency>  </dependencies>  </project> |

### 2.4.3 编写UaaApplication启动类和application.yml配置文件

# 3. 配置授权服务

## 3.1 授权服务配置类AuthorizationServer

com.liu.distributedSecurity.uaa.config.AuthorizationServer

### 3.1.1配置客户端详情服务

ClientDetailsServiceConfiguer能够使用内存或者jdbc来实现客户端详情服务(ClientDetailsService)，ClientDetailsService负责查找ClientDetails

ClientDetails属性：

clientId：用来标识客户端的id(必须)

secret：客户端安全码(需要值得信任的客户端)

scope：用来限制客户端的访问范围，如果为空的话，那么客户端拥 有全部的访问权限

authorizedGrantTypes：客户端可以使用的授权类型(5种)，默认为空

authorities：客户端可以使用的权限(基于spring security authorities)

### 3.1.2 配置令牌服务(管理令牌)

AuthorizationServerTokenServices接口定义了一些操作使得你可以对令牌进行一些必要的管理，令牌可以被用来加载身份信息，里面包含了这个令牌的相关权限。

自己可以创建AuthorizationServerTokenServices这个接口的实现，则需要继承DefaultTokenServices这个类，里面包含了一些有用的实现，你可以使用它来修改令牌的格式和令牌的存储。默认的，当它尝试创建一个令牌的时候，是使用随机值来进行填充的，除了持久化令牌是委托一个TokenStore接口来实现以外，这个类几乎帮你做了所有事情。并且TokenStore这个接口有一个默认的实现InMemoryTokenStore，所有的令牌被保存在了内存中。

InMemoryTokenStore：默认，保存在内存中。它可以工作在单服务器上(即访问并发量压力不大的情况下，并且它在失败的时候不会备份)。因为不会被保存在磁盘中，所以更易于调试。

JdbcTokenStore：这是一个基于jdbc实现的版本，令牌会被保存进关系型数据库，使用这个版本的实现时，你可以在不同的服务器之间共享令牌信息。需要添加spring-jdbc这个依赖。

JwtTokenStore：JSON Web Token(JWT)，它可以把令牌相关的数据进行编码(因此对于后端服务来说，它不需要存储)，缺点就是撤销一个已经授权的令牌将会非常困难，所以它通常用来处理一个生命周期较短的令牌以及撤销刷新令牌(refresh\_token)；另外一个缺点就是这个令牌占用的空间比较大，如果你加入了比较多的用户凭证信息。JwtTokenStore不会保存任何数据，但是它在转换令牌值以及授权信息方面与DefaultTokenServices扮演的角色是一样的

### 3.1.3 配置令牌的访问端点

AuthorizationServerEndpointsConfigurer这个对象的实例可以完成令牌服务以及令牌endpoint的配置

配置授权类型(Grant Types)

1. authenticationManager：认证管理器，当你选择了资源所有者密码(password)授权类型的时候，必须设置这个属性注入一个AuthenticationManager对象
2. userDetailsService：说明有一个自己的UserDetailsService这个接口的实现，或者你可以把这个东西设置到全局域上面(例如GlobalAuthenticationManagerConfiguer这个配置对象)，当你设置了这个之后，那么”refresh\_token”即刷新令牌授权类型模式的流程中就会包含一个检查，用来确保这个账号是否仍然后效，假如禁用了此账户
3. AuthorizationCodeServices：这个类型是用来设置授权码服务的，主要用于"authorization\_code"授权码类型模式
4. ImplicitGrantService：设置隐式授权模式
5. TokenGranter：实现TokenGranter接口，自己掌握授权，一般是上面的4种模式满足不了需求才考虑

### 3.1.4 配置令牌端点的安全约束

## 3.2 web安全配置WebSecurityConfig

com.liu.distributedSecurity.uaa.config.WebSecurityConfig

|  |
| --- |
| 1579486853(1) |

# 测试授权服务，获取token

## 4.1 授权码模式(最安全)

### 4.1.1 介绍

|  |
| --- |
| 1579505015(1) |

### 4.1.2 测试

#### 4.1.2.1 启动项目并申请授权码

访问如下地址会跳转到授权页并点击授权(approve)，response\_type=code说明是授权码模式，dsuaa是服务的context-path，后面的路径是token的访问端点

<http://localhost:8081/dsuaa/oauth/authorize?client_id=c1&response_type=code&scope=all&redirect_uri=http://www.baidu.com>

|  |
| --- |
| 1579504083(1) |

点击Authorize按钮，会重定向到redirect\_uri地址并带上授权码code

|  |
| --- |
| 1579504123(1) |

资源拥有者打开客户端，客户端要求资源拥有者给予权限，它将被浏览器重定向到授权服务器，重定向时会附加客户端信息

参数列表如下：

1. client\_id：客户端准入标识，需要与配置文件里面withClient的值一致

|  |
| --- |
| 1579502950(1) |

1. response\_type：授权码模式固定为code
2. Scope：客户端权限，需要与配置文件里面scope的值一致
3. redirect\_url：跳转的url，需要与配置文件里面redirectUris的值一致。当授权码申请成功后会跳转到此地址，并在后面带上code参数(授权码)

#### 4.1.2.2 客户端拿着授权码从授权服务器访问access\_token

<http://localhost:8081/dsuaa/oauth/token>

|  |
| --- |
| 1579504714(1) |

## 4.2 简化模式

### 4.2.1 介绍

|  |
| --- |
| 1579505185(1) |

### 4.2.2 测试

（1）资源拥有者打开客户端，客户端要求资源拥有者给予权限，它将被浏览器重定向到授权服务器，重定向时会附件客户端身份信息，response\_code=token说明是简化模式

[http://localhost:8081/dsuaa/oauth/authorize?client\_id=c1&response\_type=token&scope=all&redirect\_uri=http://www.baidu.com](http://localhost:8081/dsuaa/oauth/authorize?client_id=c1&response_type=code&scope=all&redirect_uri=http://www.baidu.com)

1. 浏览器出现向授权服务器授权页面，用户点击授权

|  |
| --- |
| 1579505626(1) |

1. 授权服务器将授权码、令牌(access\_token)以hash的形式存放在重定向uri的fargment中发送给浏览器

注意：fargment主要是用来标识URI所标识资源里的某个资源，在URL的末尾通过(#)作为fargment的开头，其中#并不属于fargment的值。如https://domain/index#L18这个URL中L18就是fargment的值。js通过响应浏览器地址栏变化的方式就能获取到fargment

一般来说，简化模式用于没有服务器端的第三方单页面应用，因为没有服务器端就无法接收授权码

|  |
| --- |
| 1579506117(1) |

## 4.3 密码模式

### 4.3.1 介绍

|  |
| --- |
| 1579506739(1) |

### 4.3.2 测试

（1）资源拥有者将用户名、密码发送给客户端

（2）客户端直接拿着资源拥有者的用户名、密码向授权服务器申请令牌，授权服务器将令牌(access\_token)发送给client

<http://localhost:8081/dsuaa/oauth/authorize?client_id=c1&client_secret=secret&grant_type=password&username=admin&password=123456>

|  |
| --- |
| 1579507348(1) |

这种模式简单，但是直接将用户敏感信息泄露给客户端

## 4.4 客户端模式

### 4.4.1 介绍

|  |
| --- |
| 1579507433(1) |

### 4.4.2 测试

（1）客户端向授权服务器发送自己的身份信息，并请求令牌(access\_token)

（2）确认客户端身份无误后，将令牌发送给client

[http://localhost:8081/dsuaa/oauth/authorize?client\_id=c1&client\_secret=secret&grant\_type=client\_credentials](http://localhost:8081/dsuaa/oauth/authorize?client_id=c1&client_secret=secret&grant_type=password&username=admin&password=123456)

|  |
| --- |
| 1579507734(1) |

# 测试资源服务(distributed-security-order)

## 编写测试服务配置文件

com.liu.distributedSecurity.order.config.OrderServerConfig

|  |
| --- |
| 1579657264(1) |

## 编写测试服务

|  |
| --- |
| 1579656568(1) |

## 校验令牌

1. 获取令牌，任意一种方式

|  |
| --- |
| 1579657523(1) |

1. 检验令牌

<http://localhost:8081/dsuaa/oauth/check_token>

|  |
| --- |
| 1579657632(1) |
| 使用用户名密码方式获取的token，在校验令牌时会展示拥有的权限  1579659601(1) |

## 测试服务

### 4.4.1 不带令牌

|  |
| --- |
| 1579658471(1) |

### 4.4.2 带令牌请求服务

按照oauth2.0协议要求，请求资源需要携带token，如下：

在headers里面，token的参数名为：Authorization，值为：Bearer + token值

1. 如果没有权限@PreAuthorize("hasAuthority('sys:user:update')")控制权限

注意：如果方法的权限校验没有控制住，需要在资源服务模块添加WebSecurityConfig配置文件，就如同授权服务的WebSecurityConfig。

但是如果在资源服务模块OrderServerConfig里面添加了httpSecurity的配置的话是不需要单独添加WebSecurityConfig配置文件的

|  |
| --- |
| 1579658939(1) |

1. 如果令牌错误，会报无效的令牌

|  |
| --- |
| 1579659030(1) |

1. 携带令牌且令牌正确

注意：这里不能用客户端模式获取token，否则会报不允许访问(没有权限，因为资源方法上面@PreAuthorize("hasAuthority('sys:user:update')")有权限控制)，

这里测试使用的是用用户名和密码的方式获取

|  |
| --- |
| 1579659097(1)  控制台会看到执行了此方法  1579659152(1) |

# JWT(JSON Web Token)

## jwt介绍

通过上边的测试我们发现，当资源服务和授权服务不在一起时资源服务使用RemotTokenServices远程请求授权服务验证token，如果访问量较大将会影响系统的性能

解决方式：

令牌采用JWT格式即可解决上边的问题，用户认证通过会得到一个JWT令牌，JWT令牌中已经包括了用户的相关信息，客户端只需要携带JWT令牌访问资源服务，资源服务根据事先约定的算法自行完成令牌的校验，无需每次都请求认证服务完成授权

### 什么是JWT

JWT(JSON Web Token)是一个开放的行业标准，它定义了一种简介的、自包含的协议格式，用于在通信双方传递json对象，传递的对象经过数字签名可以被验证和信任。JWT可以使用HMAC算法或使用RSA的公钥/私钥对来签名，防止被篡改。

### JWT令牌优缺点

优点：

1. JWT基于json，方便解析
2. 可以在令牌中自定义内容，容易扩展
3. 可以通过非对称加密算法及数字签名技术，JWT防止篡改，安全性高
4. 资源服务使用JWT可不依赖认证服务即可完成授权

缺点：

1. JWT令牌较长，占存储空间比较大

### JWT令牌结构

JWT令牌由三部分(header、)组成，每部分中间用(.)分割，eg：xxx.xxx.xxx

1. Header

头部包括令牌的类型(即JWT)及使用的hash算法(eg:HMAC、SHA256、RSA)

|  |
| --- |
| {     **"alg"**:**"SHA256"**,     **"type"**:**"JWT"** } |

将header的内容使用Base64Url编码，得到的字符串就是JWT令牌的第一部分

1. Payload

内容部分也是一个json对象，它是存放有效信息的地方，它可以存放JWT提供的现成字段

eg：iss(签发者)、exp(过期时间戳)、sub(面向的用户)，也可以自定义字段

此部分不建议存放敏感信息，因为此部分可以解码还原原始内容

将payload的内容使用Base64Url编码，得到的字符串就是JWT令牌的第二部分

|  |
| --- |
| {     **"sub"**:**"123456"**,     **"name"**:**"456"**,     **"admin"**:**"true"** } |

1. Signature

第三部分是签名，此部分用于防止JWT内容被篡改

这个部分使用Base64Url将前两部分进行编码，编码后使用(.)连接组成字符串，最后使用header中声明的签名算法进行签名

|  |
| --- |
| HMAXSHA256(  Base64UrlEncoder(header) + “.” +  Base64UrlEncoder(payload),  secret  ) |

# 将uaa服务改造成JWT令牌服务

## 修改uaa服务的配置文件

### 将令牌存储策略从基于内存的改为基于JWT

修改com.liu.distributedSecurity.uaa.config.TokenConfig类

|  |
| --- |
| 1579672504(1) |

### 6.1.2 定义令牌服务

修改com.liu.distributedSecurity.uaa.config.AuthorizationServer类

|  |
| --- |
| 1579672921(1) |

## 生成JWT令牌

|  |
| --- |
| 1579673123(1) |

## 校验令牌

### 6.3.1 通过/oauth/check\_token校验令牌

|  |
| --- |
| 1579673188(1) |

# 资源服务校验令牌

## 修改资源服务配置文件

### 将uaa授权服务的TokenConfig配置文件拷贝到order资源服务

|  |
| --- |
| 1579673714(1) |

### 修改order资源服务的OrderServerConfig配置文件

|  |
| --- |
| 1579673683(1) |

## 资源服务端校验令牌

1. 使用以前的令牌，发现不能用

|  |
| --- |
| 1579673881(1) |

1. 使用JWT令牌，可以正常访问，控制台会看到执行了此方法

|  |
| --- |
| 1579673950(1) |
| 1579673972(1) |

# 完善环境配置

将客户端信息和授权码(给予授权码模式)存放到数据库中。

因为目前在内存中存储，如果是分布式的部署到多台服务器，基于内存的方式不行

## 创建表，初始化数据

## 配置授权服务

### 修改AuthorizationServer类

com.liu.distributedSecurity.uaa.config.AuthorizationServer

（1）ClientDetailsService(客户端信息)数据从数据库读取

|  |
| --- |
| 1579678870(1) |

1. AuthorizationCodeServices(授权码)存到数据库

|  |
| --- |
| 1579676241(1) |

## 测试获取令牌，客户信息从数据库获取

## 测试授权码是否存到数据库

# springSecurity实现系统分布式授权

## 需求分析

|  |
| --- |
| 1580788530(1)   1. Uaa认证服务负责认证授权 2. 所有请求经过网关到达微服务 3. 网关负责鉴权客户端以及转发请求 |