目录

[1 OpenID & OAuth2 & SAML 2](#_Toc56001198)

[1.1 OpenID 2](#_Toc56001199)

[1.2 OAuth2 2](#_Toc56001200)

[1.3 SAML 2](#_Toc56001201)

[1.4 三者对比 3](#_Toc56001202)

[2 OIDC 3](#_Toc56001203)

[2.1 场景图 4](#_Toc56001204)

[2.2 OIDC 核心概念 4](#_Toc56001205)

[2.3 OIDC 主要术语 4](#_Toc56001206)

[2.4 OIDC 工作流程 5](#_Toc56001207)

[3 领域 6](#_Toc56001208)

[3.1 添加领域 6](#_Toc56001209)

[3.2 Keycloak领域配置，拿到密钥 6](#_Toc56001210)

[4 客户端管理 7](#_Toc56001211)

[4.1 添加客户端 7](#_Toc56001212)

[4.2 客户端密钥 7](#_Toc56001213)

[5 用户管理 8](#_Toc56001214)

[5.1 添加用户 8](#_Toc56001215)

[5.2 用户授权角色 8](#_Toc56001216)

[6 用户登陆 9](#_Toc56001217)

[6.1 密码登陆 9](#_Toc56001218)

[6.2 授权码登录 9](#_Toc56001219)

[7 API接口授权角色 10](#_Toc56001220)

[8 角色管理 10](#_Toc56001221)

[9 客户端授权 11](#_Toc56001222)

[9.1 配置客户端 11](#_Toc56001223)

[9.2 客户端模板client-templates 12](#_Toc56001224)

[9.3 客户端必须要包含的角色 13](#_Toc56001225)

[9.4 配置资源服务器接口 14](#_Toc56001226)

[9.5 客户端添加全局角色 14](#_Toc56001227)

[9.6 客户端配置角色 16](#_Toc56001228)

[9.7 配置客户端公用Scope 17](#_Toc56001229)

[9.8 开放作用域(scope) 19](#_Toc56001230)

# OpenID & OAuth2 & SAML

## OpenID

OpenID是一种**认证**标准，互联网上有很多账户都是支持OpenID比如谷歌、雅虎、PayPal等等。

用户要使用OpenID就必须先在OpenID身份服务器（Identity Provider， IDP）获得OpenID 账号（比如Google账户）。用户可以使用OpenID账户来登录任何一个接受OpenID认证的服务应用（the relying party，RP，依赖方）。OpenID协议标准就是提供一个框架用来IDP和RP之间通信。

本质而言，用户的OpenID是一个为用户个人所拥有的特殊URL(比如 alice2016.openid.com)，所以有些网站甚至会提供选项让用户自己去填写OpenID。

## OAuth2

准确来讲，OAuth2是一个授权的标准协议。也许会令人困惑，OAuth2是OpenID-Connect的基础，但是OpenID-Connect是认证协议（在OpenID-Connect中，ID-Token也被当做是一种资源）。

让我们回到OAuth2，OAuth2提供了一种代理访问机制，也就是说一个应用（可以被称为客户端）可以代替用户到资源服务器上获得属于用户的资源或是进行符合用户权限的操作 ，而用户不用将自己的用户名和口令等身份凭据分享给客户端。OAuth2是通过IDP给第三方应用颁发令牌（Token）来实现以上功能的，第三方应用通过使用令牌向资源服务换取对应的资源。

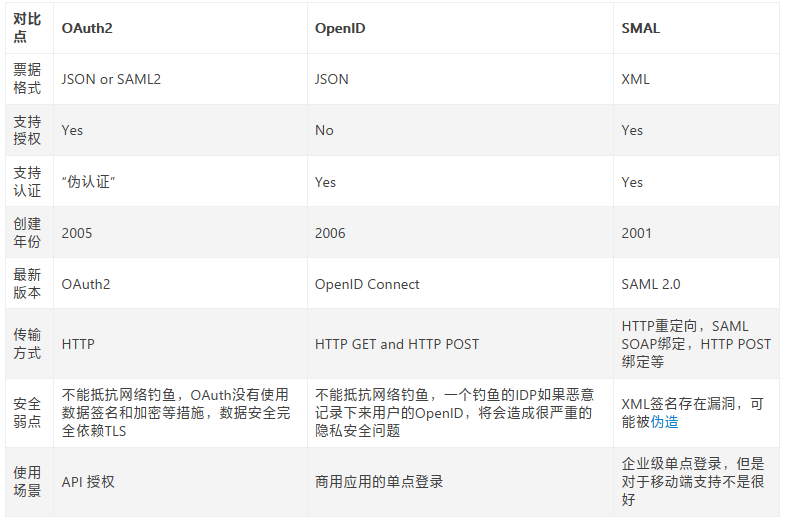
在Twitter的OAuth指导手册中说OAuth2是一种认证协议，实际上，这是基于授权的“伪认证”。

## SAML

SAML协议是三者中时间最长的协议，最初版本制定于2001年，并于2005年修改。作为一种安全性断言标记语言，SAML协议既可以用于认证也用于授权。

所谓的安全性断言，就是关于认证、授权以及用户属性（比如用用户的有效或者住址等信息）的声明集合，在SAML中，这些断言以XML的格式传输。

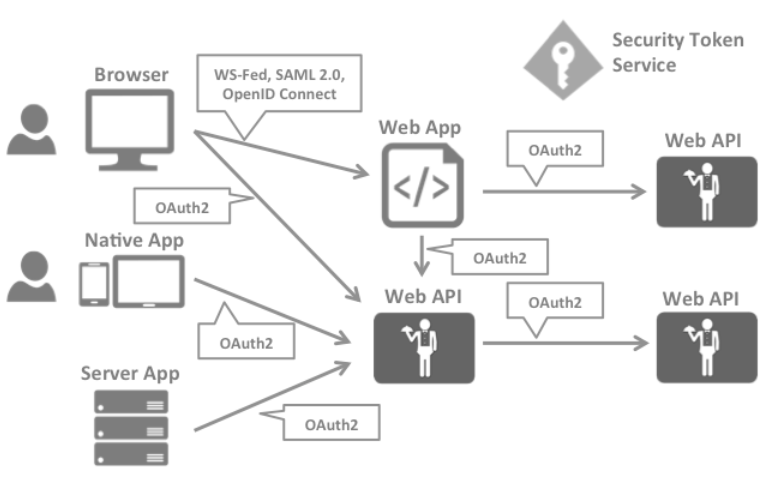
## 三者对比



# OIDC

OIDC是OpenID Connect的简称，OIDC=(Identity, Authentication) + OAuth 2.0。它在OAuth2上构建了一个身份层，是一个基于OAuth2协议的身份认证标准协议。我们都知道OAuth2是一个授权协议，它无法提供完善的身份认证功能（关于这一点请参考[[认证授权] 3.基于OAuth2的认证（译）](http://www.cnblogs.com/linianhui/p/authentication-based-on-oauth2.html)），OIDC使用OAuth2的授权服务器来为第三方客户端提供用户的身份认证，并把对应的身份认证信息传递给客户端，且可以适用于各种类型的客户端（比如服务端应用，移动APP，JS应用），且完全兼容OAuth2，也就是说你搭建了一个OIDC的服务后，也可以当作一个OAuth2的服务来用。

## 场景图



## OIDC 核心概念

OAuth2提供了Access Token来解决授权第三方客户端访问受保护资源的问题；OIDC在这个基础上提供了ID Token来解决第三方客户端标识用户身份认证的问题。OIDC的核心在于在OAuth2的授权流程中，一并提供用户的身份认证信息（ID Token）给到第三方客户端，ID Token使用JWT格式来包装，得益于JWT（[JSON Web Token](http://www.cnblogs.com/linianhui/p/oauth2-extensions-protocol-and-json-web-token.html#auto_id_6)）的自包含性，紧凑性以及防篡改机制，使得ID Token可以安全的传递给第三方客户端程序并且容易被验证。此外还提供了UserInfo的接口，用户获取用户的更完整的信息。

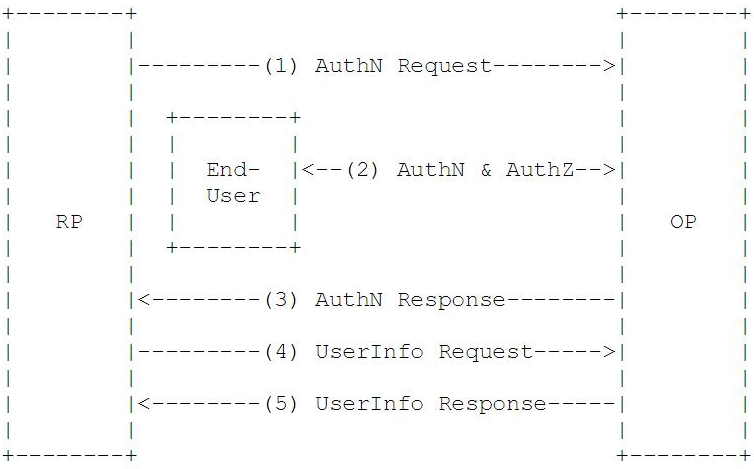
## OIDC 主要术语

1. EU：End User：一个人类用户。
2. RP：Relying Party ,用来代指OAuth2中的受信任的客户端，身份认证和授权信息的消费方；
3. OP：OpenID Provider，有能力提供EU认证的服务（比如OAuth2中的授权服务），用来为RP提供EU的身份认证信息；
4. ID Token：JWT格式的数据，包含EU身份认证的信息。
5. UserInfo Endpoint：用户信息接口（受OAuth2保护），当RP使用Access Token访问时，返回授权用户的信息，此接口必须使用HTTPS。

## OIDC 工作流程

从抽象的角度来看，OIDC的流程由以下5个步骤构成：

1. RP发送一个认证请求给OP；
2. OP对EU进行身份认证，然后提供授权；
3. OP把ID Token和Access Token（需要的话）返回给RP；
4. RP使用Access Token发送一个请求UserInfo EndPoint；
5. UserInfo EndPoint返回EU的Claims。

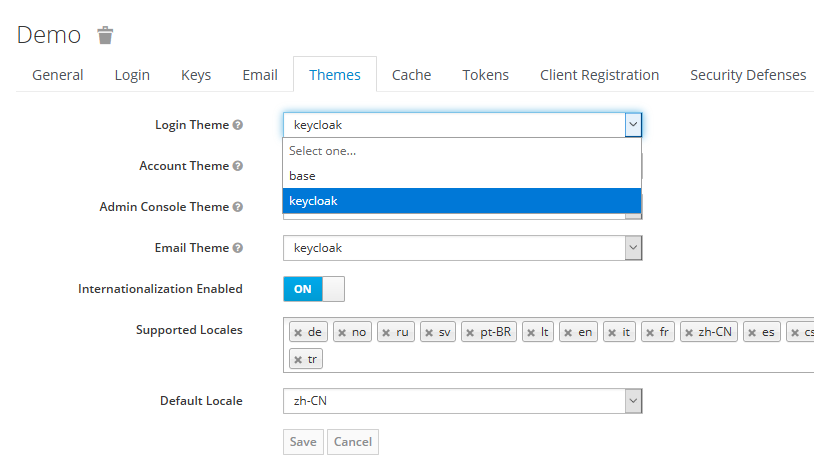


# 登陆统一

对于用户的登陆，keycloak是统一控制的，如果希望做个性化的处理，需要自己去开发css文件，生成对应的皮肤，然后在helm配置中去指定。

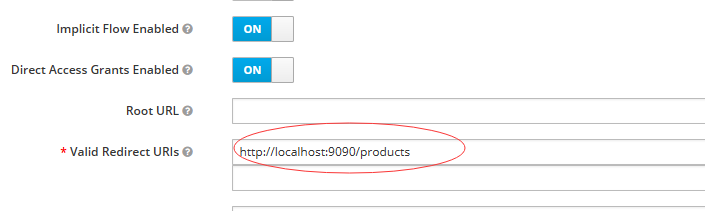
## 自定义皮肤

对于多个客户端来说，只要对接了keycloak，这们都会跳到统一的keycloak登陆页去认证。



## 登录重定向

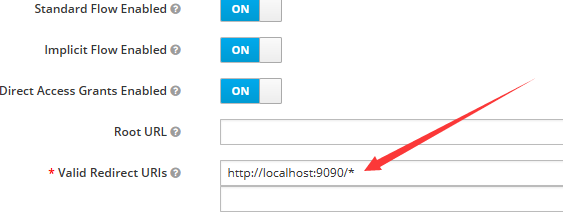
在客户端配置时，需要指定客户端的重定向地址，当登录完成后，会重定向回来，这个配置我们可以是通配符或者是指定的地址，如下：



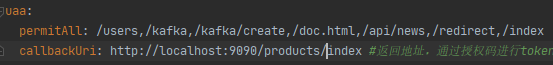
当你设置时指定地址后，你无法在客户端进行重写，如果你的重定向地址与keycloak配置的不同时，将出现下面错误：



而如果你设置为通配符时，你是可以在客户端程序里去重写它的。



我们可以在程序的配置里重写这个地址，只要符合通配符规则即可。

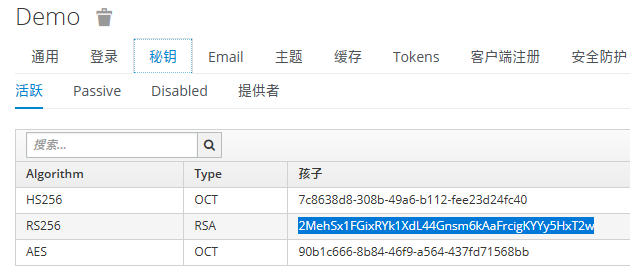


# 领域

## 添加领域



## Keycloak领域配置，拿到密钥

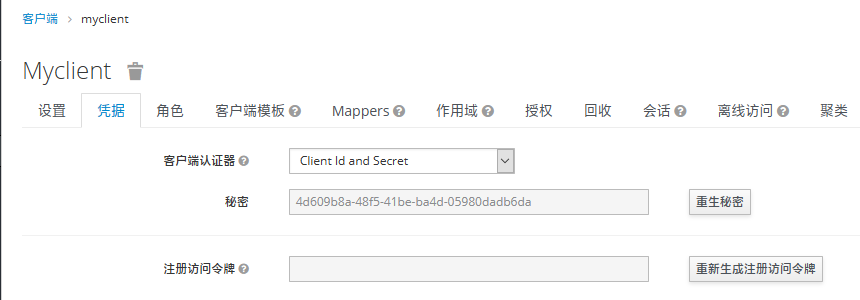


# 客户端管理

## 添加客户端



## 客户端密钥



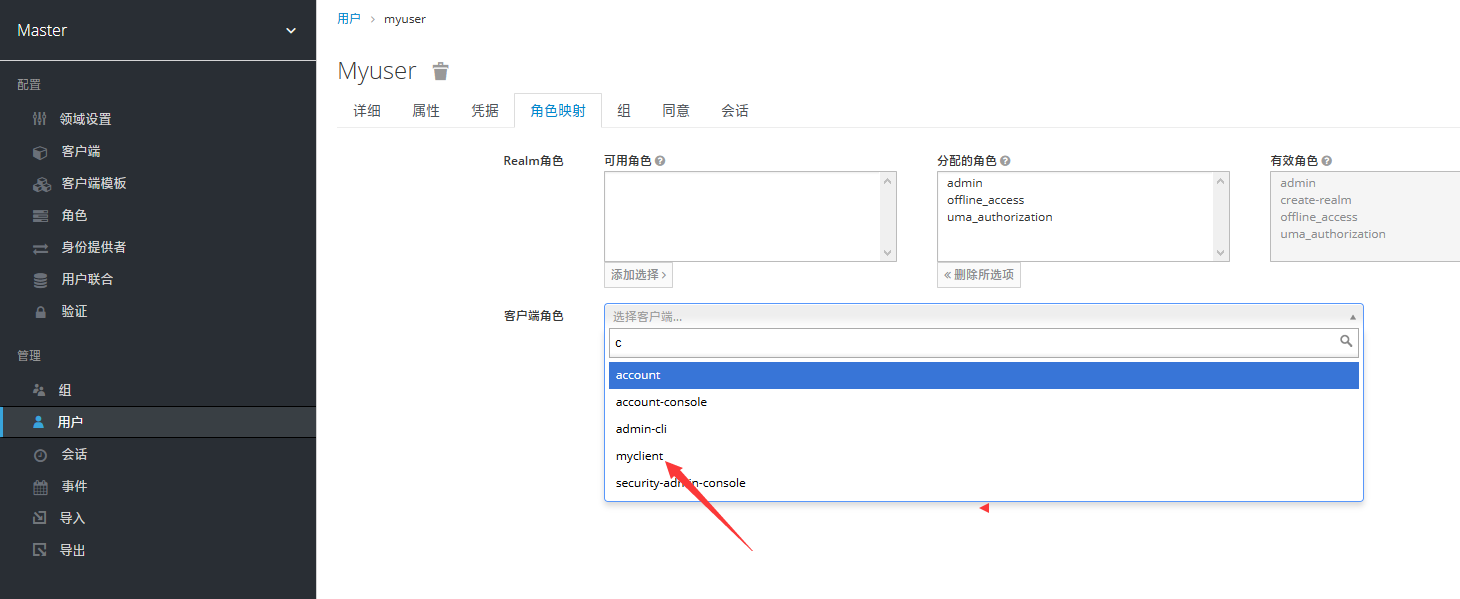
# 用户管理

## 添加用户



## 用户授权角色

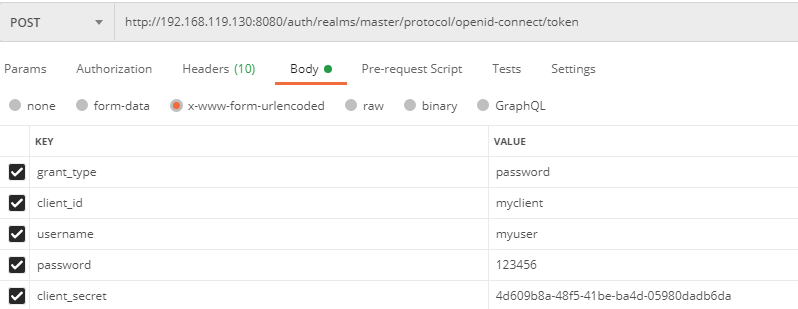
可以使用领域里的角色，或者使用客户端里建立的角色



# 用户登陆

## 密码登陆

<http://192.168.119.130:8080/auth/realms/master/protocol/openid-connect/token>



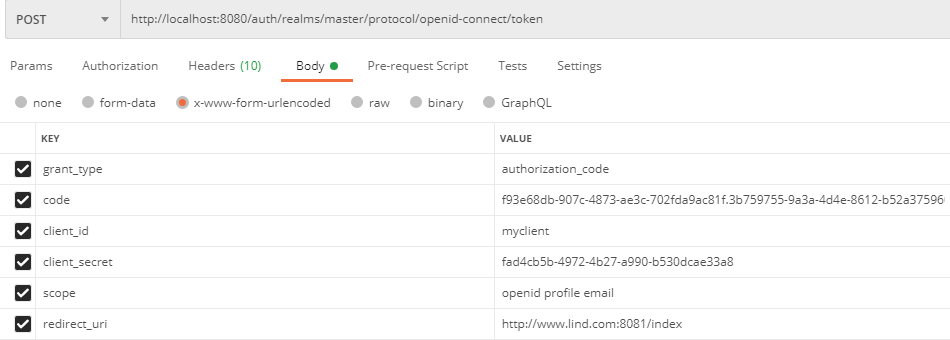
## 授权码登录

拿到code

<http://localhost:8080/auth/realms/master/protocol/openid-connect/auth?client_id=myclient&scope=openid%20profile%20email&redirect_uri=http://www.lind.com:8081/index&response_type=code>

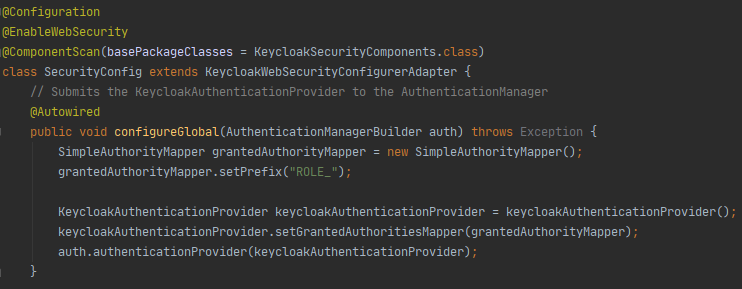
拿到access\_token

<http://localhost:8080/auth/realms/master/protocol/openid-connect/token>

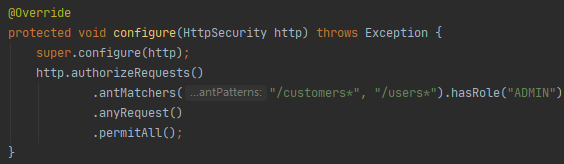


# API接口授权角色

重写SimpleAuthorityMapper将spring角色前缀ROLE\_这种过滤



对指定api添加权限

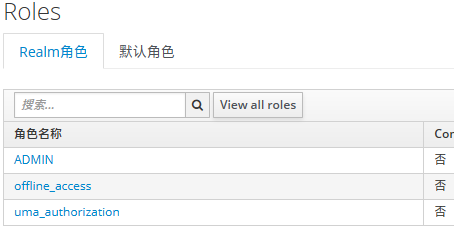


# 角色管理

角色可以被应用在客户端上，也可以应用到具体的用户上；不过有一点要注意，无论是客户端角色还是用户角色，它们都是独立的，用户不会继承客户端的设置。

## Keycloack里添加角色

在keyclock中建立ADMIN角色并分给用户





重新登陆获取token，使用新token调用接口，接口生效！

## 客户端token里的角色

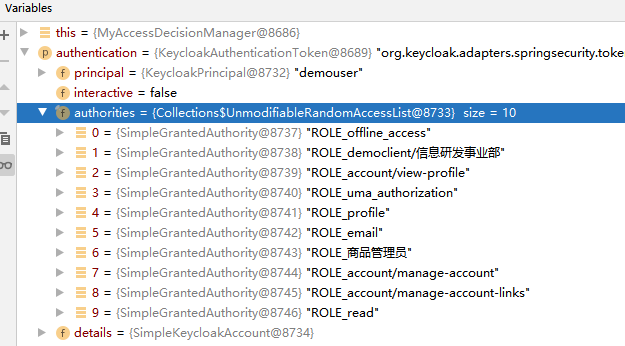
角色会被包含到token字符串里，通过jwt解析可以看到如图：



## Java程序解析token

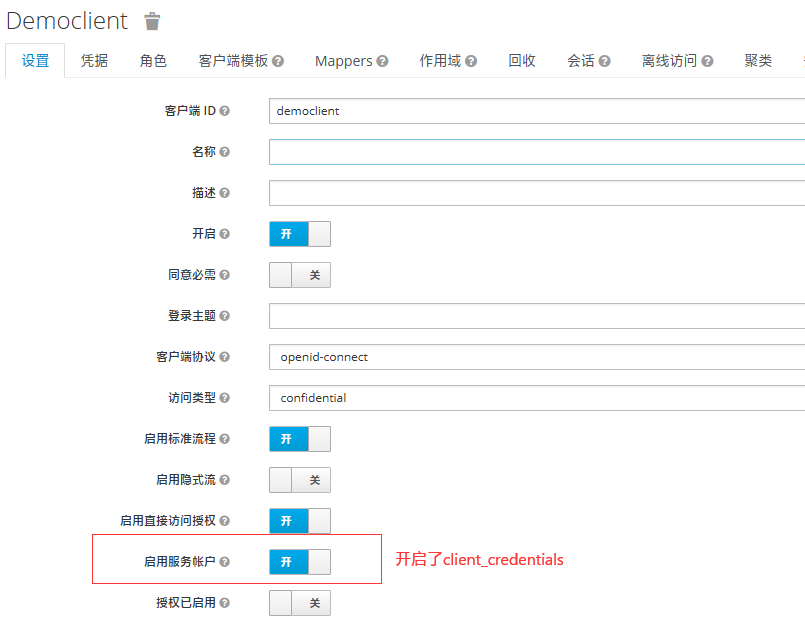
Springboot集成了keycloak之后，它会帮助我们对token进行解析，同时会被token传到接口AccessDecisionManager里，在方法decide时有参数 Authentication authentication，我们解析authentication可以看到具体的角色的集合.

注意：客户端的角色是以客户端名称为前缀的，与角色之间有/进行分隔



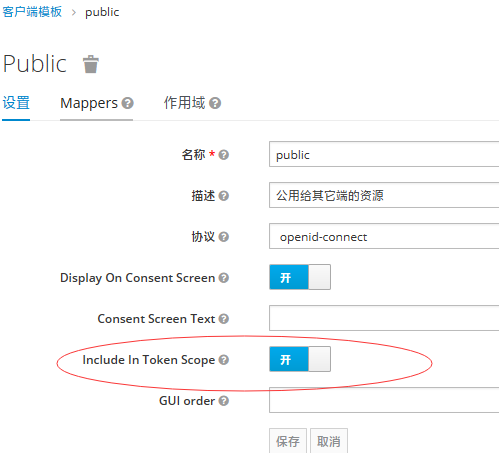
# 客户端授权

## 配置客户端

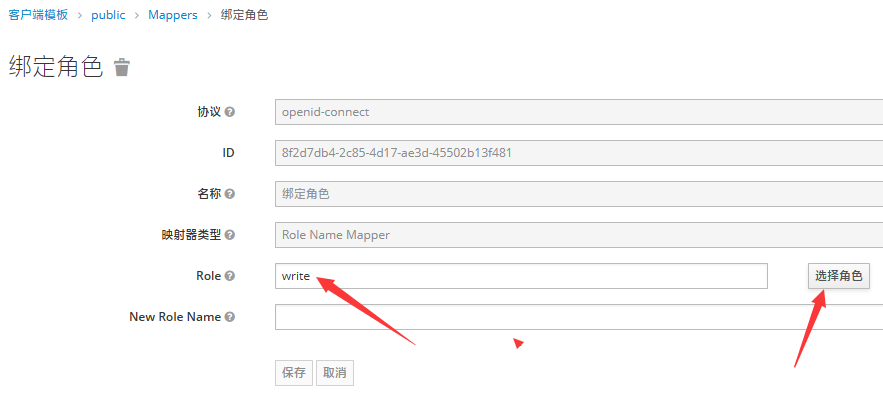


## 客户端模板client-Scopes

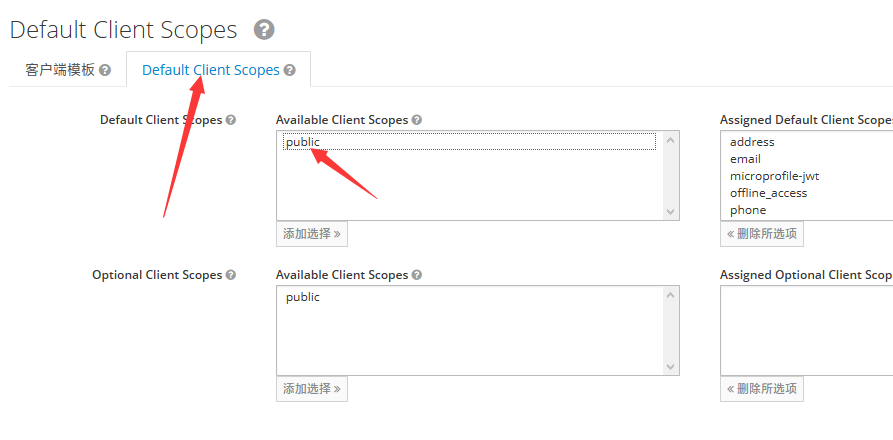
我们可以通过客户端模版(Client Scopes)功能来维护scope，当你希望你的scope包含在客户端token里，你可以这样设置：



为scope添加mappers，即它有一个映射的功能，如绑定用户信息，绑定角色等



最后将public添加到默认的客户模板里即可



对于已有的客户端，需要手动将客户端模板public添加到自己的scope里



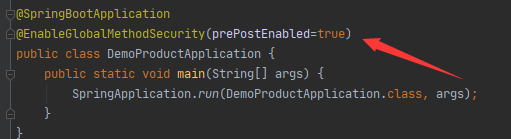
## 客户端必须要包含的角色

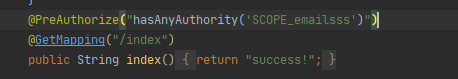
默认的offline\_access是必须要选的，否则无法获取token



## 配置资源服务器接口

开启@PreAuthorize注解



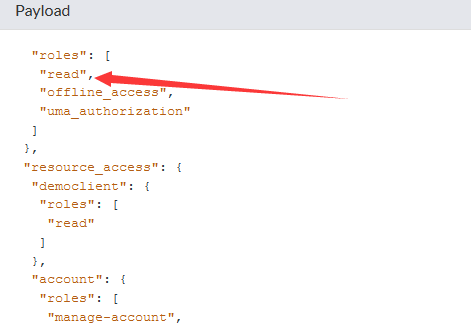
、

为客户端添加角色，与用户无关

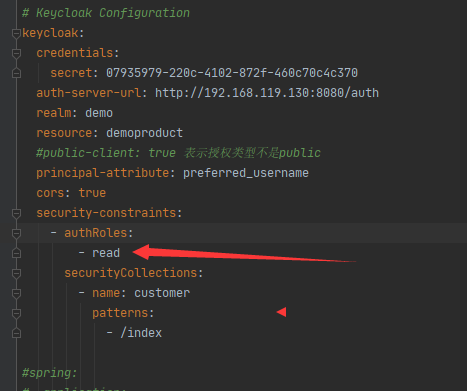
## 客户端添加全局角色

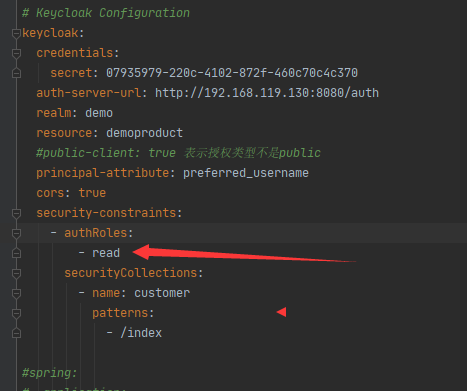


Token里的位置



通过全局的角色赋值给客户端之后，在客户端访问的资源里，可以访问到有这个角色控制的资源。

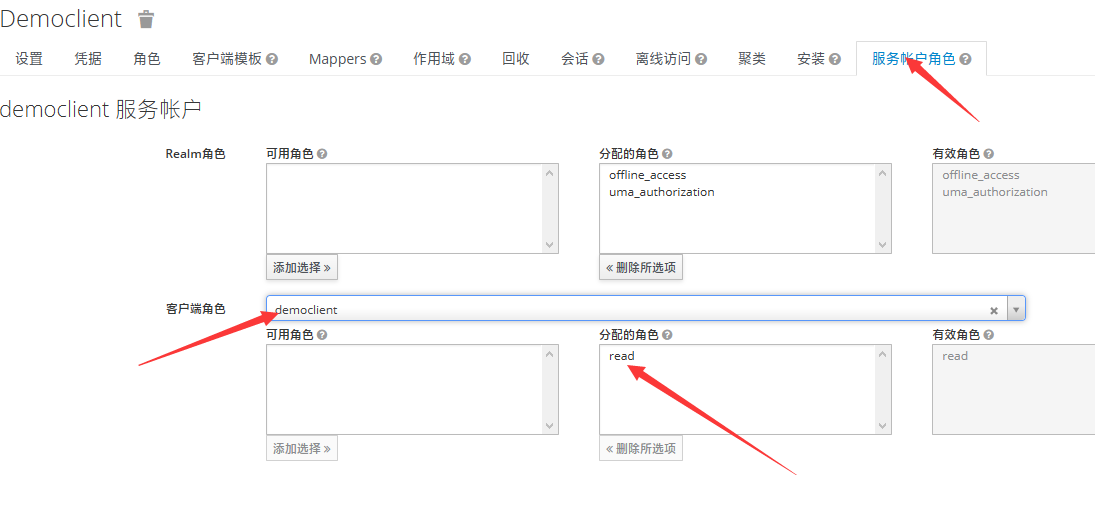




## 客户端配置角色

客户端里的角色标签，这个标签在之后选择时需要先择客户端

如下图，先选择democlient客户端，再选择它的角色read

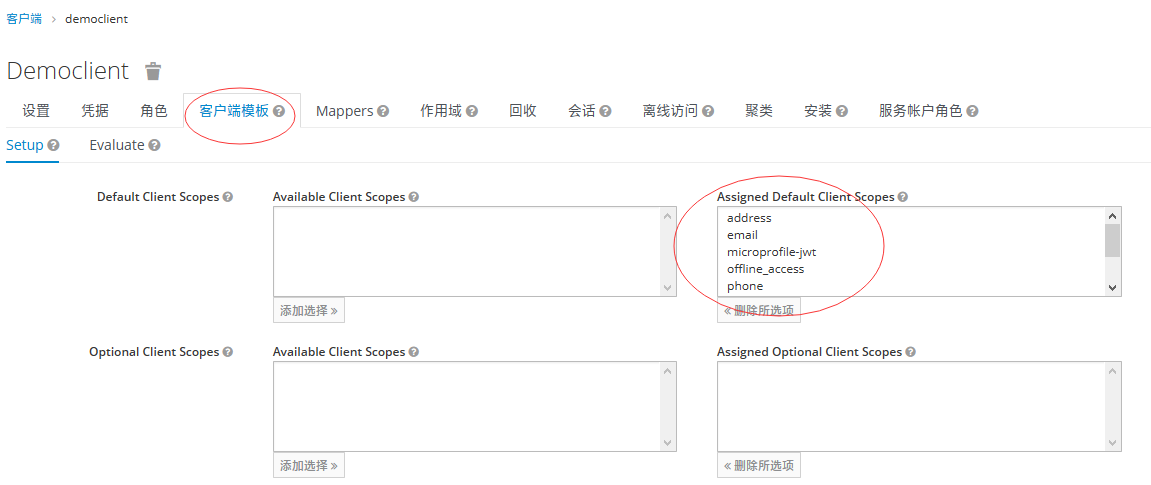


比如，在democlient里选择demoproduct 里的角read，它的Token如下：

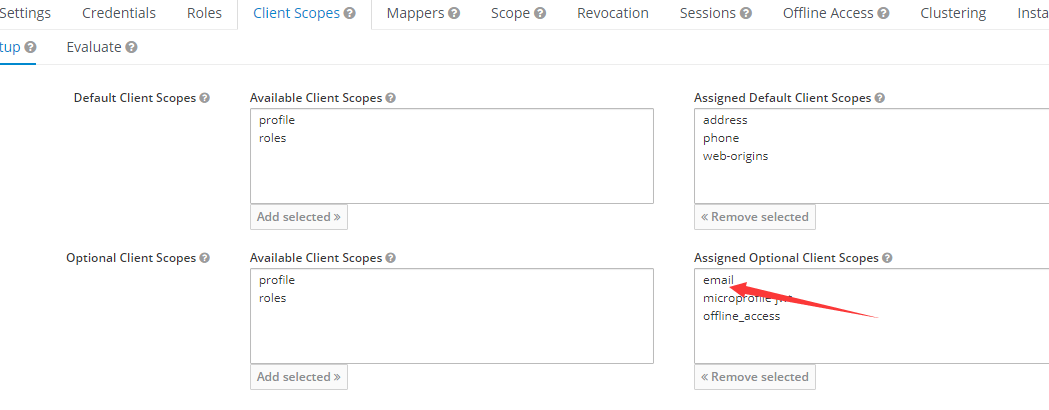


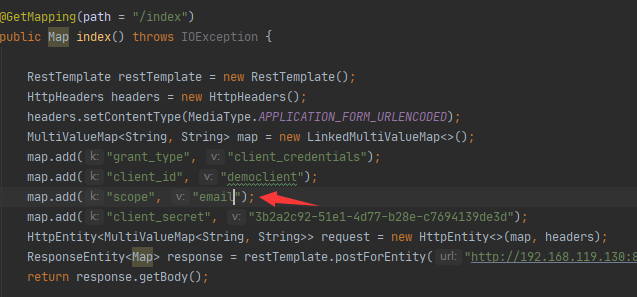
## 配置客户端公用Scope

配置客户端公用Scope，这些scope与客户请求无关，无论客户是否有scope参数 ，在返回的token里会有出现它们

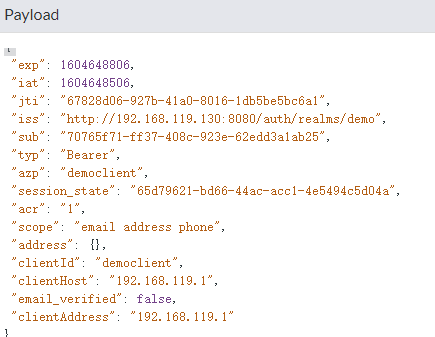


Options client scopes是可选的scope，它与你的客户传的参数scope有关，如果你开放了只开放email,而客户没有传email，在token里也不会出现，只有客户传了 email之后，token里的scope才会出现它。

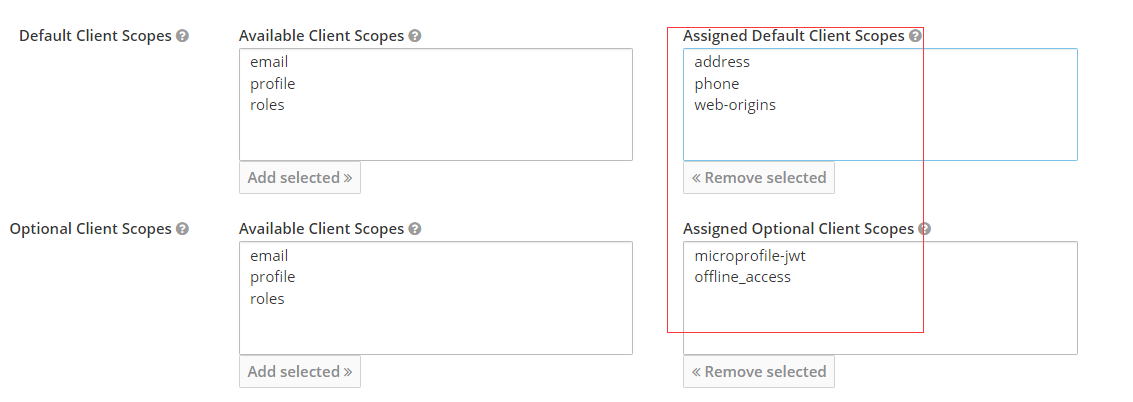


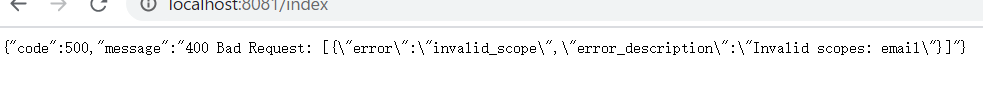


生成的token里才会有email



注意：如果授权中的客户端没有email scope，而用户授权时传了scope email，那就会直接出错。





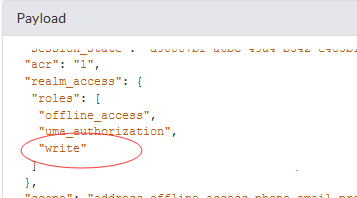
## 开放作用域(scope)

客户端有realm的角色，在授权后，会在access\_token中体现出来，而这些角色开放出多少，可以由作用域scope来控制，如下图所示，虽然客户端有read,write两个角色，但在作用域（scope）里只开放write，所以在生成的token里，只有write角色。

注意：客户端自己的角色它是全部包含的,而其它客户端的角色也可以在这里进行配置，即虽然你在【service account roles】这里菜单里配置了角色，也是可以通过【scope】进行管理的，当你开启了【Full Scope Allowed】选项之后，那么在【service account roles】里配置的角色，将全部被添加到token里返回；但是如果你在【service account roles】没有配置客户端角色，即使在【scope】里选择了客户端角色 ，它也不会返回的，所以从某种意义上来说，【service account roles】是返回客户端角色的前提，而【scope】主要用来进行控制。



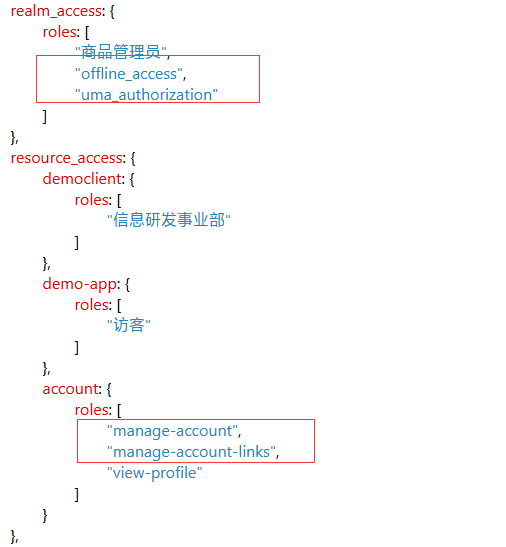
生成的token的解析



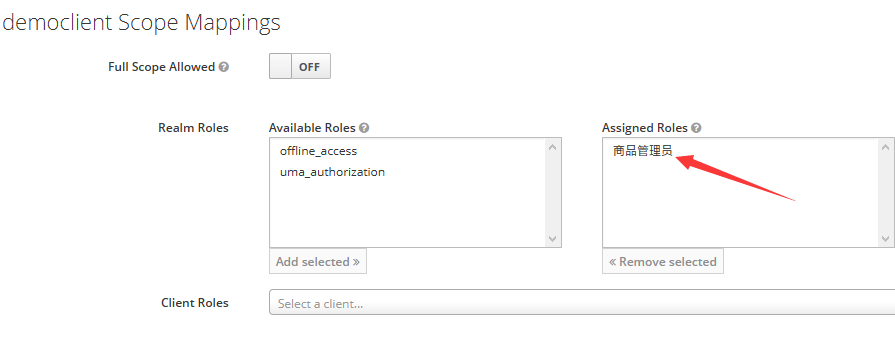
当然，如果你希望开放所有已定义的角色，把作用域开放打开即可



当你把范围【scope】全部打开后，在token里将出现系统默认的角色



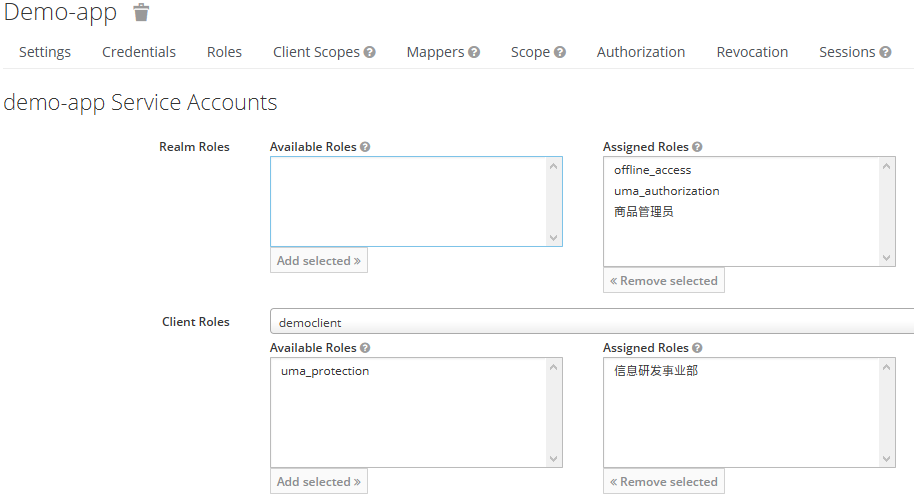
当把全部范围关半之后，在token里只会有你选中的角色





## 客户端授权角色

每个客户端（client）都有默认的角色uma\_protection，及realm里的角色"offline\_access","uma\_authorization”等【通过9.8节可以关系默认的角色】，如果你需要添加自定义的角色，那么需要在“客户端配置->service account roles”菜单里进行配置，如图，将全局角色“商品管理员”及democlient客户端的角色“信息研发事业部”都添加到了客户端角色 里，而在客户端授权时，它将被添加到token里返回。



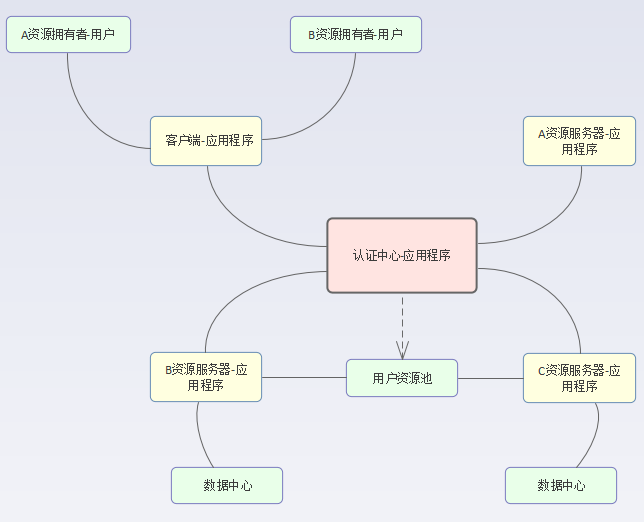
返回的token,【aud】表示授权的客户端，【realm\_access】表示当前领域级的角色，【resource\_access】是各个授权客户端的角色。





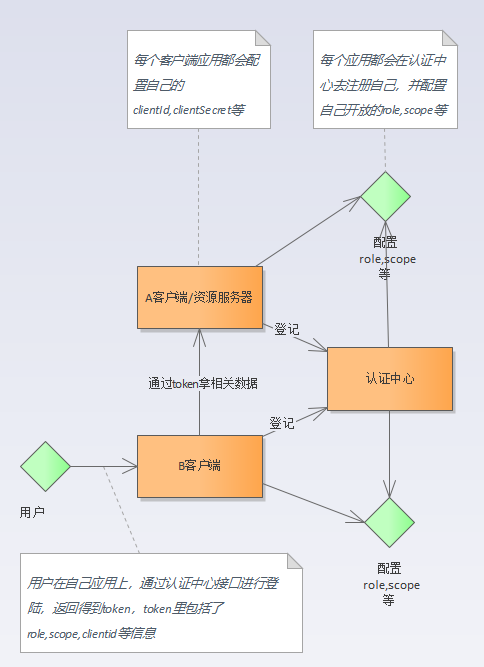
# 设想

## Keycloak与实现项目关系



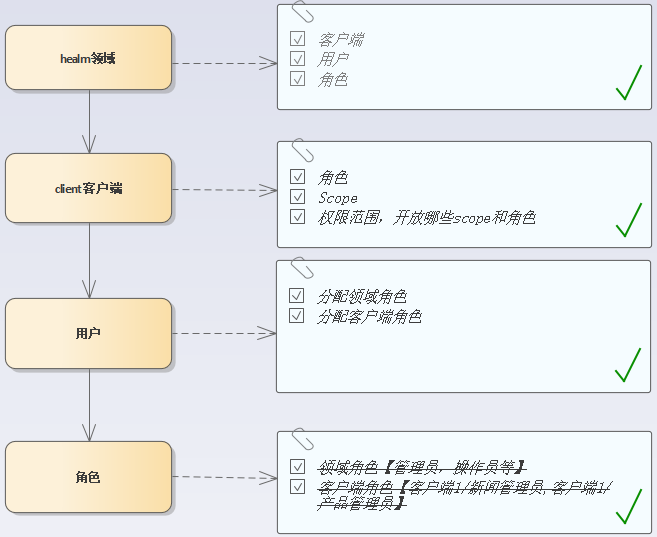
## 客户端和资源

客户端也是一个资源服务器，它们都需要先到认证中心去注册，然后得到clientId,clientSecret等唯一性信息。



## Role & Authority

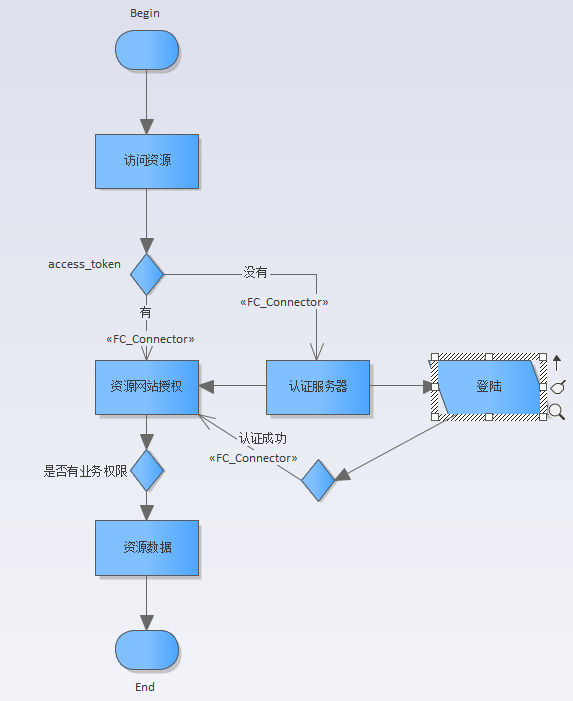
它们是用户的属性，一般被设计成用户与角色的关联，角色与权限的关联。



## scope

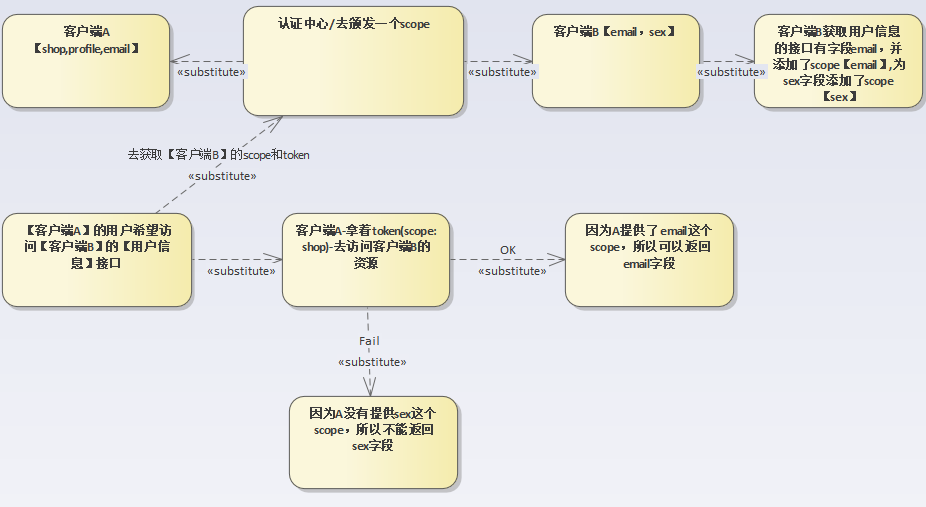
它是client客户端的属性，需要在认证中心去配置，对于资源服务器来说，可以在资源实体上同时配置对应的scope值，然后通过用户包含的scope来决定是不返回这些字段。

## 认证与获取资源的过程



## Scope的户法

Scope可以直接应用在客户端上，可以与用户没有关系，也可以与用户有关，返回用户相关的数据。

* Scope控制访问的具体数据字段
* Scope控制接口集合

## Role 的用法

Role是角色，与用户有着直接关系，需要在用户里去绑定role， role在keycloack里可以有realm【英 /relm/ 】和 client范围的，可以为用户分别进行绑定，但要注意，client级别的role在spring被解析后，它是以client/role的形式返回的。

