|  |
| --- |
| Auth认证中心 |

|  |
| --- |
| 陈军  2017/12/9 |

目录

[1 技术选型 2](#_Toc501399596)

[1.1 硬件环境 2](#_Toc501399597)

[1.2 软件环境 2](#_Toc501399598)

[1.3 项目构建和管理工具 2](#_Toc501399599)

[1.4 后端技术 2](#_Toc501399600)

[2 什么是OAuth2.0 3](#_Toc501399601)

[2.1 OAut2.0简介 3](#_Toc501399602)

[2.2 运行流程 3](#_Toc501399603)

[2.3 可支持的用户的授权模式 3](#_Toc501399604)

[3 为什么选择Password模式 4](#_Toc501399605)

[3.1 什么是密码模式 4](#_Toc501399606)

[3.2 选择密码模式的原因 4](#_Toc501399607)

[4 数据模型 5](#_Toc501399608)

[5 Q&A 5](#_Toc501399609)

[5.1 为什么选择Token 5](#_Toc501399610)

[5.2 认证服务中心验证流程 6](#_Toc501399611)

# 技术选型

## 硬件环境

整体硬件环境以普通PC来组建、64-bit(x64) 处理器、1GB以上系统内存

## 软件环境

1. 建议Linux/Windows环境
2. 数据库ORACL
3. Redis
4. JDK1.7以上版本
5. Tomcat

## 项目构建和管理工具

1. Maven

Maven是一个项目管理工具，主要用于项目构建，依赖管理，项目信息管理。

1. 开发工具

Eclipse

1. 代码管理工具

GIT

## 后端技术

Spring Boot、Spring MVC、Spring JPA（hibernate）

OAuth2.0

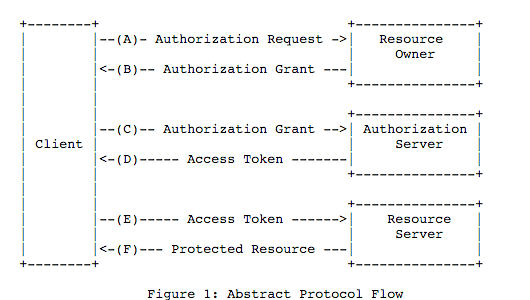
Redis、Java、Log4j、JUnit Test

# 什么是OAuth2.0

## OAut2.0简介

OAuth是一个关于授权（authorization）的开放网络标准，目前的版本是2.0版。它是一种让用户允许第三方应用在某一个网站上获取自己相关信息的协议，它能在不告诉第三方应用密码的情况下，使得第三方应用可以获得相关信息，并且对第三方应用可以使用的权限进行一定的划分控制。

## 运行流程



* + 1. 第三方应用请求用户，希望得到授权
    2. 用户给予授权
    3. 第三方应用通过授权向认证服务器获取token
    4. 认证服务器确认第三方授权无误，授予token
    5. 第三方应用通过token访问资源服务器
    6. 资源服务器确认token无误向第三方应用开发资源

## 可支持的用户的授权模式

OAuth 2.0定义了四种授权方式：

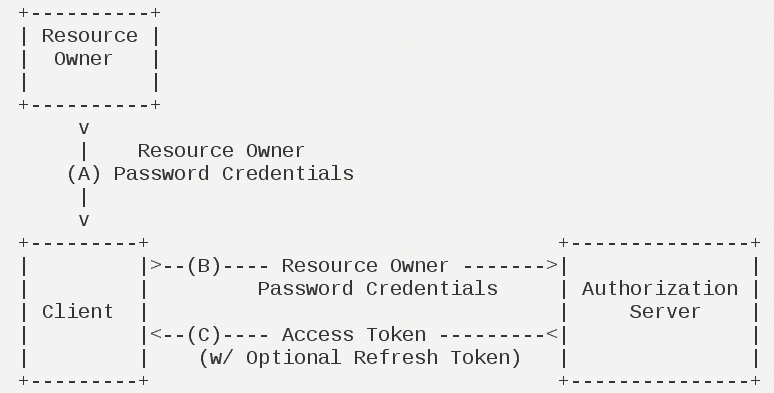
1. 授权码模式（authorization code）
2. 隐式模式（implicit）
3. 密码模式（resource owner password credentials）
4. 客户端模式（client credentials）

# 为什么选择Password模式

## 什么是密码模式

密码模式（Resource Owner Password Credentials Grant）中，用户向客户端提供自己的用户名和密码。客户端使用这些信息，向认证服务器索要授权。

步骤如下：

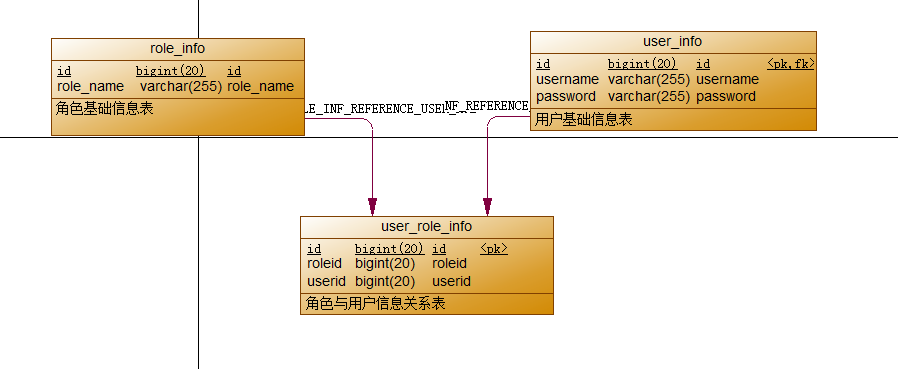
1. 用户向客户端提供用户名和密码。
2. 客户端将用户名和密码发给认证服务器，向后者请求令牌。
3. 认证服务器确认无误后，向客户端提供访问令牌。

## 选择密码模式的原因

1. 客户端可信任度比较高，为同一公司出品。因此满足密码模式的使用标准
2. 开发成本相对较低，无需前端界面
3. 需要其它模式可支持扩展

# 数据模型

数据库存储，通过oracle数据库存储用户基本信息、角色基本信息以及用户和角色的关系表，从而支持一个用户有多个角色，或一个角色可以有多个用户的关系，基础表结构如下图：



# Q&A

## 为什么选择Token

* + - 1. 无状态

Token 以无状态的形式存储在客户端，可扩展性强，由于这种无状态，负载均衡器能够将用户信息从一个服务传到其他服务器上。

如果使用Session，每次认证用户发起请求时，服务器需要去创建一个记录来存储信息。当越来越多的用户发请求时，内存的开销也会不断增加。并且，每次请求都需要用户向已验证的服务器发送验证信息。用户量大时，可能会造成一些拥堵。

* + - 1. 更安全

如果使用Cookie作为认证方式容易被CSRF(跨站请求伪造)攻击。因此，请求中发送token而不再是发送Cookie更为安全。即使在客户端使用cookie存储token，cookie也仅仅是一个存储机制而不是用于认证。不将信息存储在Session中，让我们少了对session操作。

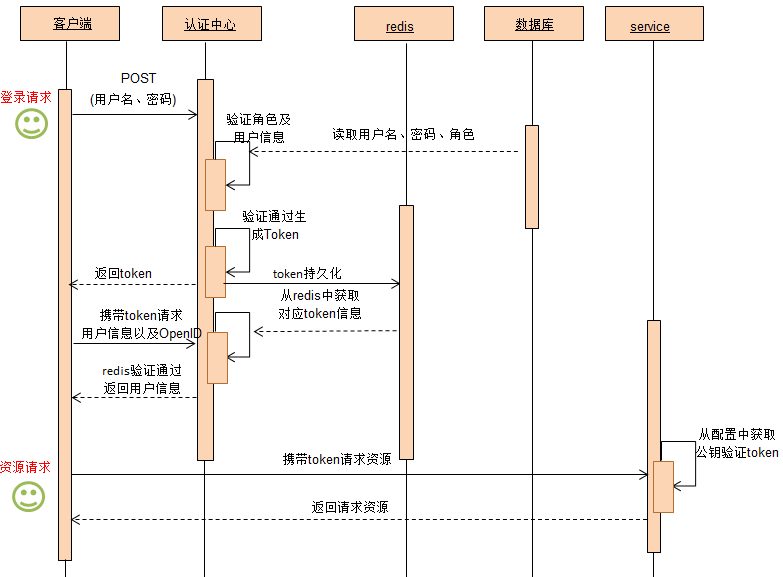
* + - 1. 可扩展性强

Token能够创建与其它程序共享权限的程序。使用token时，可以提供可选的权限给第三方应用程序。当用户想让另一个应用程序访问它们的数据，我们可以通过建立自己的API，生成特定权限的token。

* + - 1. 多平台跨域

对应用程序和服务进行扩展的时候，需要介入各种各样的设备和应用程序。只要用户有一个通过了验证的token，数据和资源就能够在任何域上被请求到。实现了CORS(跨域资源共享)，

## 认证服务中心验证流程



1. 用户提供用户名/密码/角色信息存储至数据库
2. 用户通过用户名/密码在客户端登录，客户端携带用户名/密码向认证中心请求令牌
3. 认证中心验证户名/密码以及角色信息
4. 认证中心验证通过，向客户端返回访问令牌,同时将令牌储存至REDIS服务器
5. 客户端携带Token向Service请求资源
6. Service端从配置中获取公钥验证Token
7. Token验证无误后允许客户端访问Service Controller方法
8. Serice端响应请求的资源到客户端