单点登录框架

1. CAS

解决问题：多个系统只需登录一次，无需重复登录

原理：授权服务器，被授权客户端

1、授权服务器（一个）保存了全局的一份session，客户端（多个）各自保存自己的session

2、客户端登录时判断自己的session是否已登录，若未登录，则（告诉浏览器）重定向到授权服务器（参数带上自己的地址，用于回调）

3、授权服务器判断全局的session是否已登录，若未登录则定向到登录页面，提示用户登录，登录成功后，授权服务器重定向到客户端（参数带上ticket【一个凭证号】）

4、客户端收到ticket后，请求服务器获取用户信息

5、服务器同意客户端授权后，服务端保存用户信息至全局session，客户端将用户保存至本地session

默认不支持http请求， 仅支持https

缺点：

cas单点登录技术适用于传统应用的场景比较多， 官方示例也是以javaWeb为准， 对微服务化应用，前后端分离应用，

支持性较差

目前新版CAS也支持OAuth2.0 方式

2. OAuth2.0

解决问题：第三方系统访问主系统资源，用户无需将在主系统的账号告知第三方，只需通过主系统的授权，第三方就可使用主系统的资源（

如：APP1需使用微信支付，微信支付会提示用户是否授权，用户授权后，APP1就可使用微信支付功能了）

原理：主系统，授权系统（给主系统授权用的，也可以跟主系统是同一个系统），第三方系统

1、第三方系统需要使用主系统的资源，第三方重定向到授权系统

2、根据不同的授权方式，授权系统提示用户授权

3、用户授权后，授权系统返回一个授权凭证（accessToken）给第三方系统【accessToken是有有效期的】

4、第三方使用accessToken访问主系统资源【accessToken失效后，第三方需重新请求授权系统，以获取新的accessToken】

OAUTH2中的角色：

- Resource Server: 被授权访问的资源

- Authotization Server：OAUTH2认证授权中心
- Resource owner : 资源拥有者
- Client：使用API的客户端(如Android 、IOS、web app)

OAuth 2.0定义了四种授权方式。

  - 密码模式（resource owner password credentials）
  - 授权码模式（authorization code）
  - 简化模式（implicit）
  - 客户端模式（client credentials）

密码模式（resource owner password credentials）**(**为遗留系统设计**)(**支持refresh token**)

  - 这种模式是最不推荐的，因为client可能存了用户密码
  - 这种模式主要用来做遗留项目升级为oauth2的适配方案
  - 当然如果client是自家的应用，也是可以
  - 支持refresh token

授权码模式（authorization code）(正宗方式)(支持refresh token)

  - 这种模式算是正宗的oauth2的授权模式
  - 设计了auth code，通过这个code再获取token
  - 支持refresh token

简化模式（implicit）(为web浏览器应用设计)(不支持refresh token)

  - 这种模式比授权码模式少了code环节，回调url直接携带token
  - 这种模式的使用场景是基于浏览器的应用
  - 这种模式基于安全性考虑，建议把token时效设置短一些
  - 不支持refresh token

客户端模式（client credentials）(为后台api服务消费者设计)(不支持refresh token)

  - 这种模式直接根据client的id和密钥即可获取token，无需用户参与
  - 这种模式比较合适消费api的后端服务，比如拉取一组用户信息等
  - 不支持refresh token，主要是没有必要

> refresh token的初衷主要是为了用户体验不想用户重复输入账号密码来换取新token，因而设计了refresh token用于换取新token

> 这种模式由于没有用户参与，而且也不需要用户账号密码，仅仅根据自己的id和密钥就可以换取新token，因而没必要refresh token

安全控制框架

1. spring-security

spring-security 是spring家族的安全控制框架， 功能非常完善。 Spring Security是能够为J2EE项目提供综合性的安全访问控制解决方案的安全框架。

它依赖于Servlet过滤器。这些过滤器拦截进入请求，并且在应用程序处理该请求之前进行某些安全处理。

Spring Security对用户请求的拦截过程如下:

2. shiro

Apache Shiro 是一个强大而灵活的开源安全框架，它干净利落地处理身份认证，授权，企业会话管理和加密。

以下是你可以用 Apache Shiro 所做的事情：

1. 验证用户来核实他们的身份
2. 对用户执行访问控制，
    1. 判断用户是否被分配了一个确定的安全角色
    2. 判断用户是否被允许做某事

1. 在任何环境下使用 Session API，即使没有 Web 或 EJB 容器。
2. 在身份验证，访问控制期间或在会话的生命周期，对事件作出反应。
3. 聚集一个或多个用户安全数据的数据源，并作为一个单一的复合用户“视图”。
4. 启用单点登录（SSO）功能。 内置了jasig-cas
5. 为没有关联到登录的用户启用"Remember Me"服务

市面上一些主流的技术搭配

spring-security + oauth2

spring-security + cas 功能较弱，对前后端分离的项目支持不是很好

shiro + cas

比较

features	spring-security + oauth2	shiro + cas
依赖	jdk jwt redis	jdk jwt redis
自定义权限	支持，用户登录后将用户的权限列表写入认证服务器	支持， 用户登录后将用户的权限列表写入客户
共享session	支持	支持
前后端分离	支持，参数始终携带access_token	支持不够友好，需要改造CAS服务端
缺点	功能较重，学习成本较高，无法短时间内了解的比较深文档比较复杂	需要集成shiro 和 cas， 项目框架较重，cas官方提供的示例是仅支持javaWeb， 对前后端分离的项目支持不够友好，需要对CAS服务端进行改造， 复杂性较高。
文档	文档完善	文档完善
优点	功能完善，针对权限控制这一块提供了比较完善的解决方案集成该框架较简单，开发周期较短spring家族产品，和spring-cloud系列的技术集成较简单，更加成熟可以搭建稳定的认证服务器。	文档清晰，较简单功能完善，对权限，用户认证这一块提供的功能非常丰富集成shiro较简单，开发周期很短。
活跃度	活跃	活跃
Starts	5.4k	8.2k