

单点登陆

单点登陆(Single Sign On), 简称SSO, 是目前比较流行的企业业务整合的解决方案之一, SSO定义是在多个应用系统中, 用户只需要登陆一次就可以访问所有相互信任的应用系统.

简单点说就是在一个多系统共存的环境下, 用户在一处登陆后, 就不用在其他系统中登陆, 也就是用户的一次登陆就能得到其他所有系统的信任. 单点登陆在大型网站里使用得非常频繁, 例如像阿里巴巴这样的网站, 在网站的背后是成百上千的子系统, 用户一次操作或交易可能涉及到几十个子系统的协作, 如果每个子系统都需要用户认证, 就非常麻烦了, 实现单点登陆说到底就是要解决如何产生和存储那个信任, 再就是其他系统如何验证这个信任的有效性, 因此要点也就以下两个:

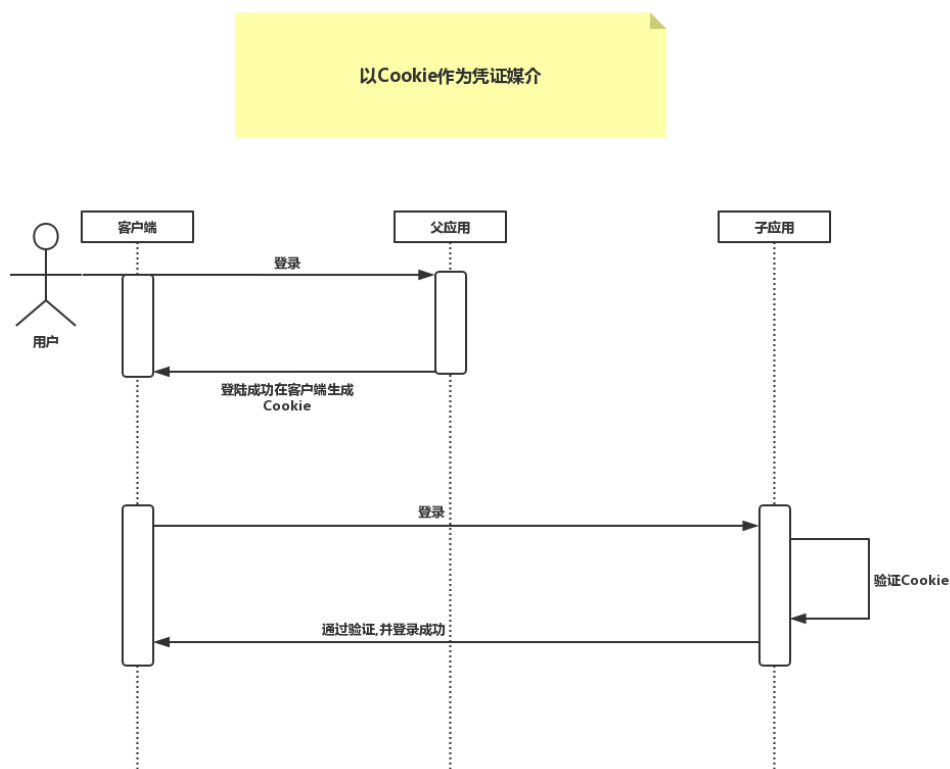
- 存储信任
- 验证信任

如果一个系统做到了开头所讲的效果, 也就算单点登陆, 单点登陆有不同的实现方式, 本文就罗列其中一些实现方式.

以Cookie作为凭证媒介

最简单的单点登陆实现方式, 是使用cookie作为媒介, 存放用户凭证.

用户登陆父应用之后, 应用返回一个加密的cookie, 当用户访问子应用的时候, 携带上这个cookie, 授权应用解密cookie并进行校验, 校验通过则登陆当前用户.



不难发现以上方式把信任存储在客户端的Cookie中, 这种方式:

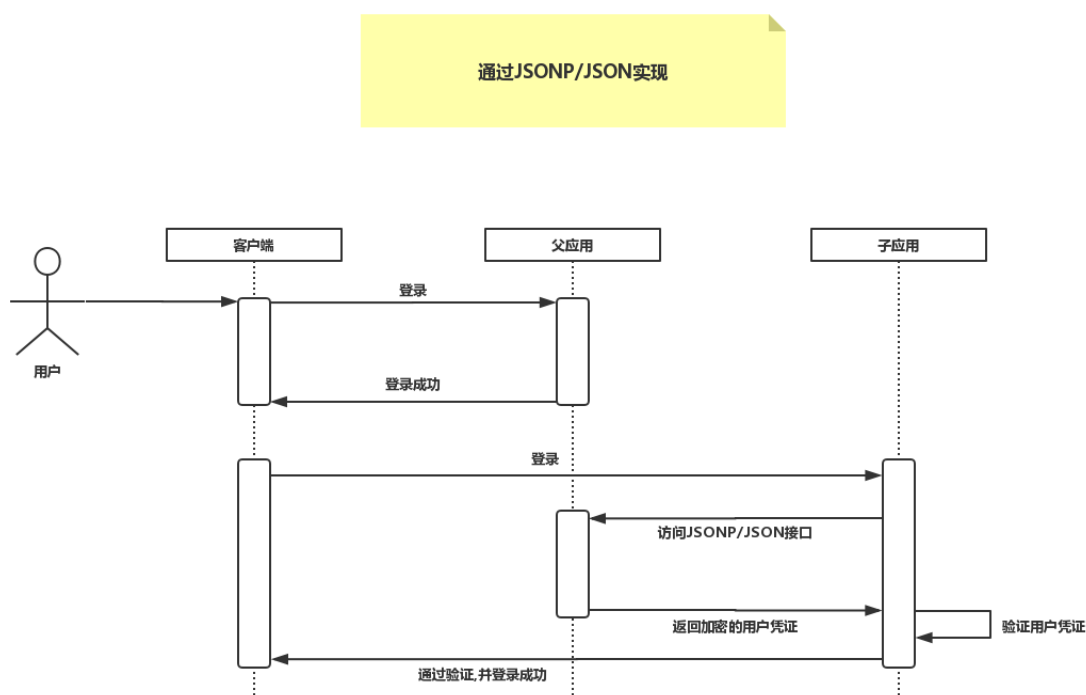
- Cookie不安全
- 不能跨域实现免登

对于第一个问题, 通过加密 Cookie 可以保证安全性, 当然这是在源代码不泄露的前提下. 如果 Cookie 的加密算法泄露, 攻击者通过伪造 Cookie 则可以伪造特定用户身份, 这是很危险的. 对于第二个问题, 更是硬伤.

通过JSONP/JSON实现

对于跨域问题, 可以使用JSONP实现.

用户在父应用中登陆后, 跟Session匹配的Cookie会存到客户端中, 当用户需要登陆子应用的时候, 授权应用访问父应用提供的JSONP接口, 并在请求中带上父应用域名下的Cookie, 父应用接收到请求, 验证用户的登陆状态, 返回加密的信息, 子应用通过解析返回来的加密信息来验证用户, 如果通过验证则登陆用户.

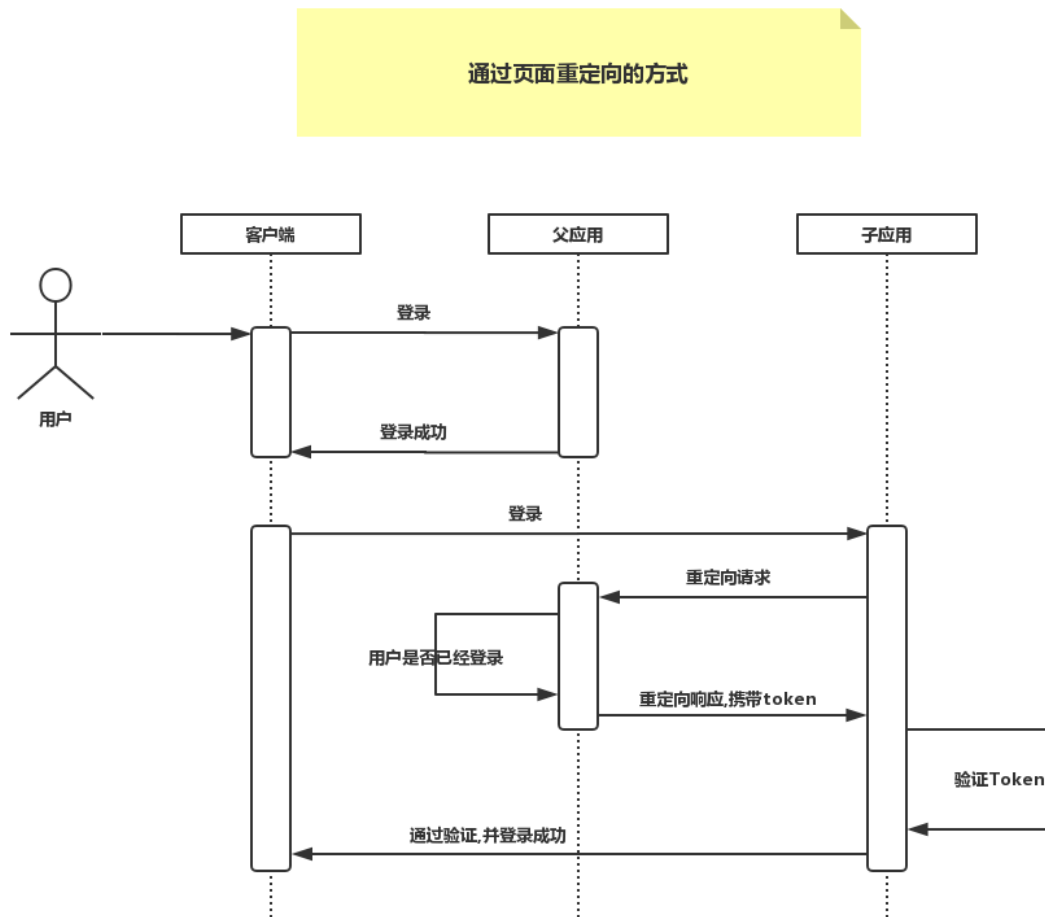


这种方式虽然能解决跨域问题, 但是安全性其实跟把信任存储到Cookie是差不多的. 如果一旦加密算法泄露了, 攻击者可以在本地建立一个实现了登录接口的假冒父应用, 通过绑定Host来把子应用发起的请求指向本地的假冒父应用, 并作出回应.

因为攻击者完全可以按照加密算法来伪造响应请求, 子应用接收到这个响应之后一样可以通过验证, 并且登录特定用户.

通过页面重定向的方式

最后一种介绍的方式, 是通过父应用和子应用来回重定向中进行通信, 实现信息的安全传递. 父应用提供一个GET方式的登录接口, 用户通过子应用重定向连接的方式访问这个接口, 如果用户还没有登录,则返回一个登录页面, 用户输入账号密码进行登录, 如果用户已经登录了,则生成加密的 token,并且重定向到子应用提供的验证token的接口, 通过解密和校验之后,子应用登录当前用户.



这种方式较前面两种方式,解决了上面两种方式暴露出的安全性问题和跨域问题, 但是并没有前面两种方式方便. 安全与方便, 本来就是一对(a pair of)矛盾.

使用独立登录系统

一般说来,大型应用会把授权的逻辑与用户信息的相关逻辑独立成一个应用, 称为用户中心. 用户中心不处理业务逻辑, 只是处理用户信息的管理以及授权给第三方应用, 第三方应用需要登录的时候, 则把用户的登录请求转发给用户中心进行处理, 用户处理完毕返回凭证, 第三方应用验证凭证, 通过后就登录用户.

[微平台文章-原来单点登录这么简单.](#)

节选上文:

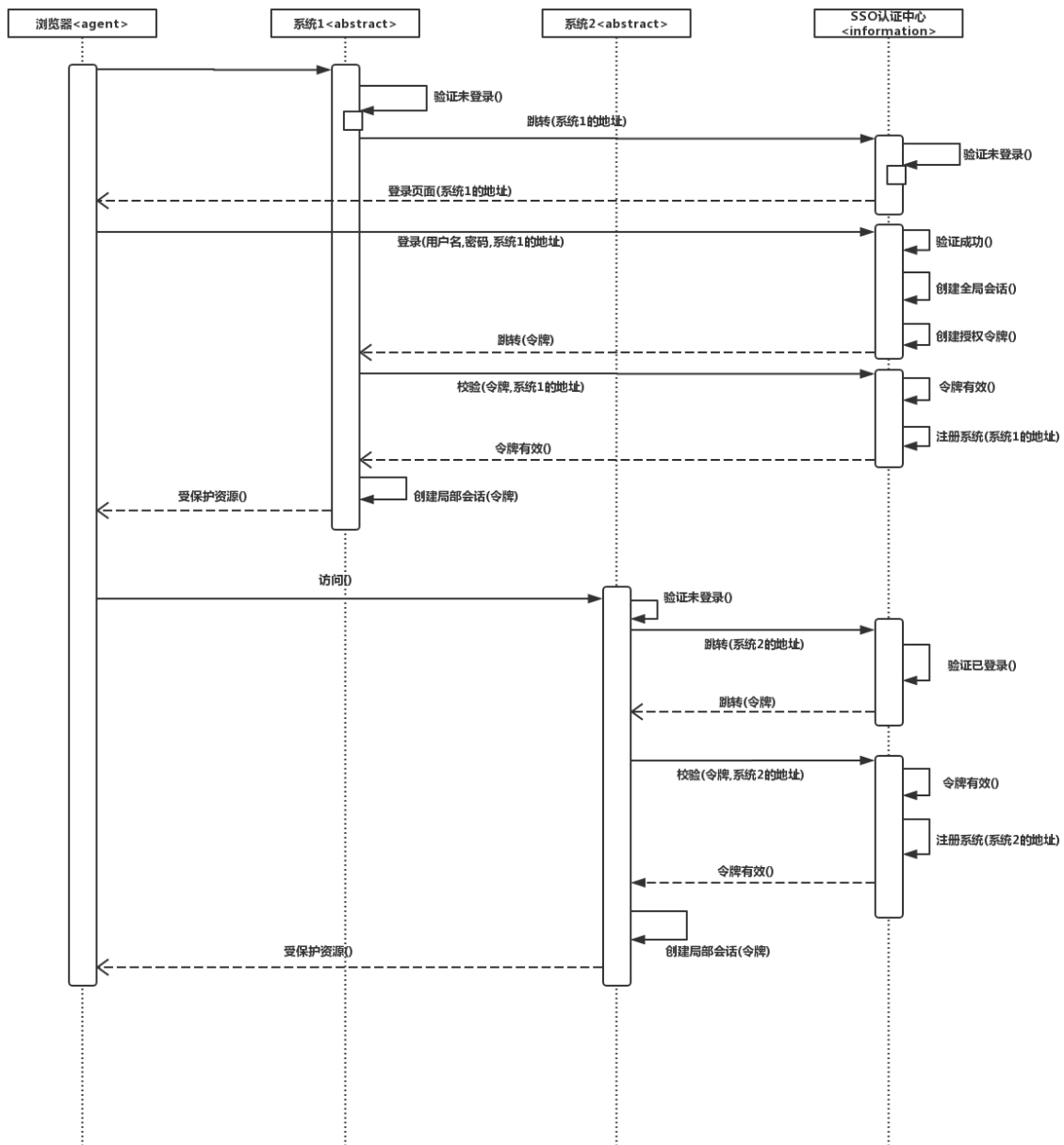
单系统登录解决方案的核心是cookie,cookie携带会话id在浏览器与服务器之间维护会话状态. 但cookie是有限制的, 这个限制就是cookie的域 (通常对应网站的域名), 浏览器发送http请求会自动携带与该域匹配的cookie,而不是所有cookie

既然这样,为什么不将web应用群中所有子系统的域名统一在一个顶级域名下,例如“*.baidu.com”,然后将它们的cookie域设置为“baidu.com”,这种做法理论上是可以的,甚至早期很多多系统登录就是采用这种同域名共享cookie的方式.

因此,我们需要一种全新的登录方式来实现多系统应用群的登录,这就是单点登录。

什么是单点登录？单点登录全称Single Sign On (以下简称SSO),是在多系统应用群中登录一个系统,便可在其他所有系统中得到授权而无需再次登录, 包括单点登录与单点注销两部分.

单点登录



1. 用户访问系统1的受保护资源, 系统1发现用户未登录, 跳转至sso认证中心, 并将自己的地址作为参数;
2. sso认证中心发现用户未登录, 将用户引导至登录页面;
3. 用户输入用户名密码提交登录申请;
4. sso认证中心校验用户信息, 创建用户与sso认证中心之间的会话,称为全局会话, 同时创建授权令牌;
5. sso认证中心带着令牌跳转回最初的请求地址(系统1);
6. 系统1拿到令牌, 去sso认证中心校验令牌是否有效;
7. sso认证中心校验令牌,返回有效,注册系统1;
8. 系统1使用该令牌创建与用户的会话, 称为局部会话, 返回受保护资源;
9. 用户访问系统2的受保护资源;

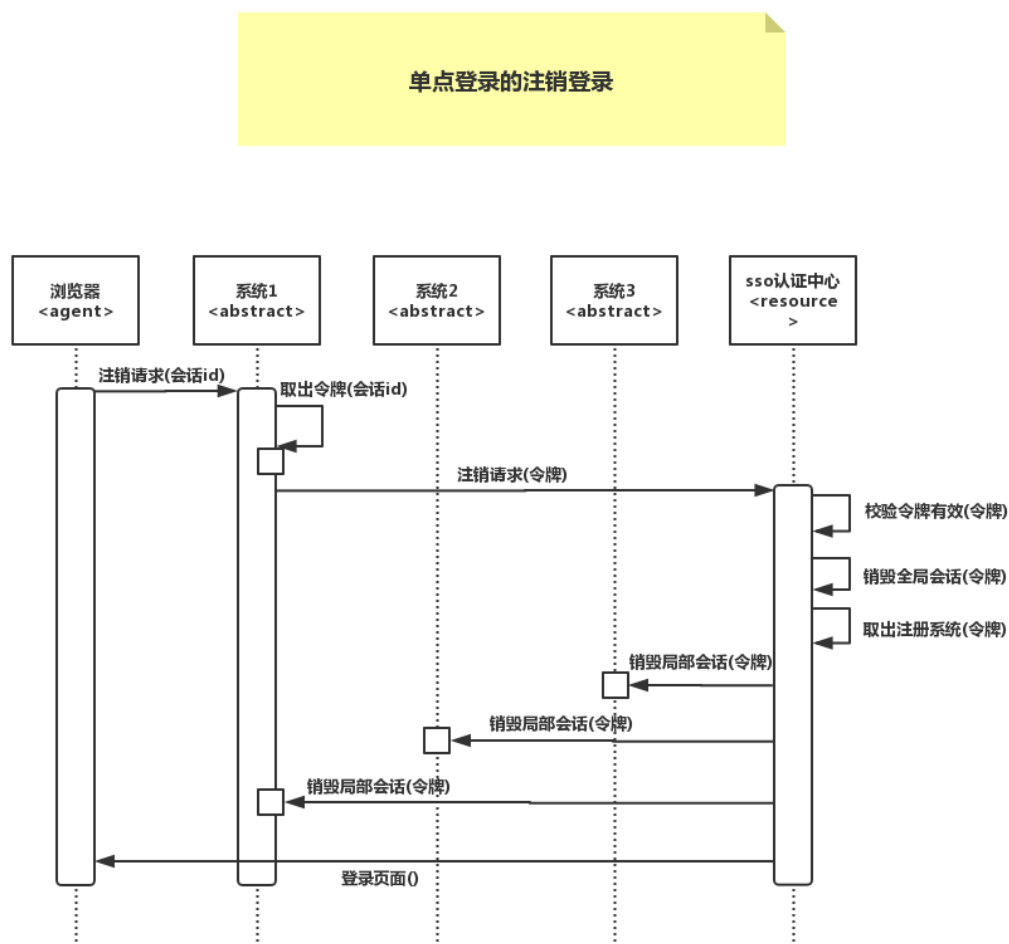
10. 系统2发现用户未登录, 跳转至sso认证中心,并将自己的地址作为参数;
11. sso认证中心发现用户已登录,跳转回系统2的地址,并附上令牌;
12. 系统2拿到令牌,去sso认证中心校验令牌是否有效;
13. sso认证中心校验令牌,返回有效,注册系统2;
14. 系统2使用该令牌创建与用户的局部会话,返回受保护资源.

用户登录成功之后,会与sso认证中心及各个子系统建立会话, 用户与sso认证中心建立的会话称为全局会话, 用户与各个子系统建立的会话称为局部会话, 局部会话建立之后, 用户访问子系统受保护资源将不再通过sso认证中心, 全局会话与局部会话有如下约束关系.

1. 局部会话存在,全局会话一定存在.
2. 全局会话存在,局部会话不一定存在;
3. 全局会话销毁, 局部会话必须销毁.

你可以通过博客园、百度、csdn、淘宝等网站的登录过程加深对单点登录的理解, 注意观察登录过程中的跳转url与参数

单点登录自然也要单点注销, 在一个子系统中注销, 所有子系统的会话都将被销毁, 用下面的图来说明:



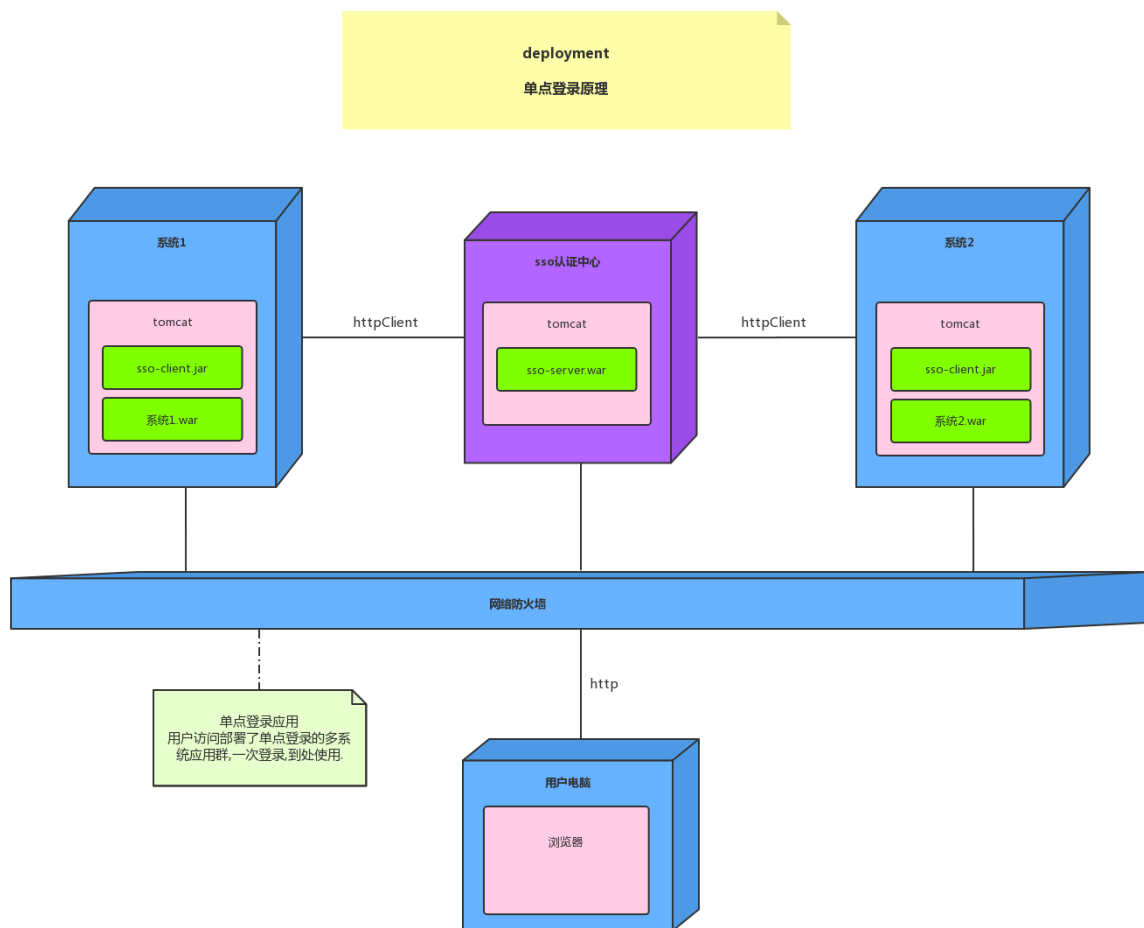
sso认证中心一直监听全局会话的状态, 一旦全局会话销毁, 监听器将通知所有注册系统执行注销操作

下面对上图简要说明:

1. 用户向系统1发起注销请求;
2. 系统1根据用户与系统1建立的会话id拿到令牌,向sso认证中心发起注销请求;
3. sso认证中心校验令牌有效,销毁全局会话,同时取出所有用此令牌注册的系统地址;
4. sso认证中心向所有注册系统发起注销请求;
5. 各注册系统接收sso认证中心的注销请求,销毁局部会话;
6. sso认证中心引导用户至登录页面.

部署

单点登录涉及sso认证中心与众子系统, 子系统与sso认证中心需要通信以交换令牌, 校验令牌及发起注销请求, 因而子系统必须集成sso的客户端, sso认证中心则是sso服务端, 整个单点登录过程实质是sso客户端与服务端通信的过程, 用下图描述:



sso认证中心与sso客户端通信方式有多种, 这里以简单好用的httpClient为例, web service, rpc, restful api 都可以.

实现

只是简要介绍下基于java的实现过程, 不提供完整源码, 明白了原理, 我相信你们可以自己实现. sso采用客户端/服务端架构, 我们先看sso-client与sso-server要实现的功能 (下面: sso认证中心=sso-server).

sso-client:

1. 拦截子系统未登录用户请求,跳转至sso认证中心;
2. 接收并存储sso认证中心发送的令牌;
3. 与sso-server通信,校验令牌的有效性;
4. 建立局部会话;
5. 拦截用户注销请求, 向sso认证中心发送注销请求;
6. 接收sso认证中心发送的注销请求, 销毁局部会话.

sso-server

1. 验证用户的登录信息;
2. 创建全局会话;
3. 创建授权令牌;
4. 与sso-client通信发送令牌;
5. 校验sso-client令牌有效性;
6. 系统注册;
7. 接收sso-client注销请求,注销所有会话.

接下来,我们按照原理一步步实现sso吧!

1. sso-client拦截未登录请求

java拦截请求的方式有servlet, filter, listener三种方式, 我们采用 filter . 在sso-client中新建LoginFilter.java类并实现Filter接口, 在doFilter()方法中假如对未登录用户的拦截.

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
    HttpServletRequest req = (HttpServletRequest) request;
    HttpServletResponse res = (HttpServletResponse) response;
    HttpSession session = req.getSession();

    if(session.getAttribute("isLogin")){
        chain.doFilter(request, response);
        return;
    }

    //跳转至sso认证中心
    res.sendRedirect("sso-server-url-with-system-url");
}
```

2. sso-server拦截未登录请求

拦截从sso-client跳转至sso认证中心的未登录请求,跳转至登录页面,这个过程与sso-client完全一样.

3. sso-server验证用户登录信息

用户在登录页面输入用户名密码, 请求登录, sso认证中心校验用户信息, 校验成功, 将会话状态标记

为“已登录”

```
@RequestMapping("/login")
public String login(String username,String password,HttpServletRequest req) {
    this.checkLoginInfo(username,password);
    req.getSession().setAttribute("isLogin",true);
    return "success";
}
```

4. sso-server创建授权令牌

授权令牌是一串随机字符, 以什么样的方式生成都没有关系, 只要不重复, 不宜伪造即可, 下面是一个例子:

```
String token = UUID.randomUUID().toString();
```

5. sso-client取得令牌并校验

sso认证中心登录后, 跳转回子系统并附上令牌, 子系统(sso-client)取得令牌, 然后去sso认证中心校验, 在LoginFilter.java的doFilter()中添加几行:

```
//请求附带token参数
String token = req.getParameter("token");
if(token != null){
    //去sso认证中心校验token
    boolean verifyResult = this.verify("sso-server-verify-url",token);
    if(!verifyResult) {
        res.sendRedirect("sso-server-url");
        return;
    }
    chain.doFilter(request,response);
}
```

verify() 方法使用httpClient实现,这里仅简略介绍, httpClient详细使用方法请参考官方文档:

```
HttpPost httpPost = new HttpPost("sso-server-verify-url-with-token");
HttpResponse httpResponse = httpClient.execute(httpPost);
```

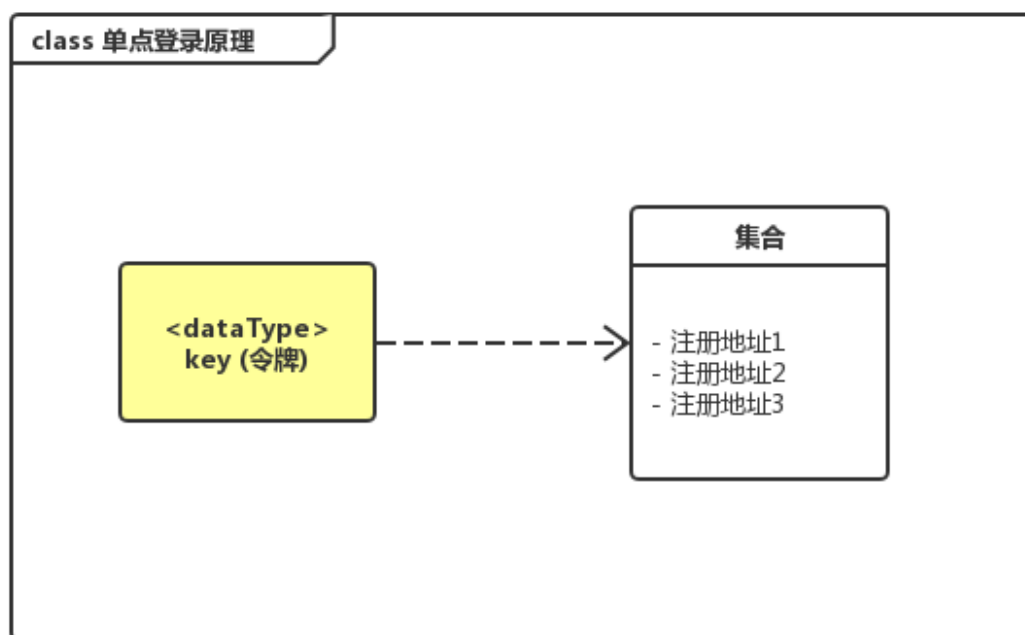
6. sso-server接收并处理校验令牌请求

用户在sso认证中心登录成功后, sso-server创建授权令牌并存储该令牌, 所以, sso-server对令牌的校验就是去查找这个令牌是否存在以及是否过期, 令牌校验成功后sso-server将发送校验请求的系统注册到sso认证中心. (就是存储起来的意思)

令牌与注册系统地址通常存储在key-value数据库(如redis)中, redis可以为key设置有效时间也就是令牌的有效期. redis运行在内存中, 速度非常快, 正好 sso-server 不需要持久化任何数据.

令牌与注册系统地址可以用下图描述的结构存储在redis中, 可能你会问,为什么要存储这些系统的地址? 如果不存储, 注销的时候就麻烦了, 用户向sso认证中心提交注销请求, sso认证中心注销全局会话, 但不知道哪些系统用此全局会话建立了自己的局部会话, 也不知道要向哪些子系统发送注销请求注销局部会话:

class 单点登录原理



7. sso-client校验令牌成功创建局部会话

令牌校验成功后, sso-client将当前局部会话标记为“已登录”, 修改LoginFilter.java , 添加几行:

```
if (verifyResult) {  
    session.setAttribute("isLogin", true);  
}
```

sso-client还需将当前会话id与令牌绑定, 表示这个会话的登录状态与令牌相关, 此关系可以用java的hashmap保存, 保存的数据用来处理sso认证中心发来的注销请求

8. 注销过程

用户向子系统发送带有“logout”参数的请求(注销请求), sso-client拦截器拦截该请求, 向sso认证中心发起注销请求:

```
String logout = req.getParameter("logout");
if(logout != null){
    this.ssoServer.logout(token);
}
```

sso认证中心也用同样的方式识别出sso-client的请求是注销请求(带有"logout"参数),sso认证中心注销全局会话.

```
@RequestMapping("/logout")
public String logout(HttpServletRequest req) {
    HttpSession session = req.getSession();
    if(session != null) {
        session.invalidate(); //触发LogoutListener
    }
    return "redirect:/";
}
```

sso认证中心有一个全局会话的监听器,一旦全局会话注销,将通知所有注册系统注销

```
public class LogoutListener implements HttpSessionListener {
    @Override
    public void sessionCreated(HttpSessionEvent event) {}
    @Override
    public void sessionDestroyed(HttpSessionEvent event) {
        //通过httpClient向所有注册系统发送注销请求
    }
}
```


单点登录的demo我看了很多就是同域名下的,如果不是同域名下的还好做吧?

html5 javascript java 44 次浏览

心里好慌, 不知道现在做的demo到底符不符合整体的业务场景.
也每个人探讨...技术都比较low.

问题解决后请 采纳答案 进行终结; 如果自己找到解决方案, 你可以 自问自答 并采纳。


已关注 1 收藏 评论 邀请回答 编辑 ...

 ukzq RP 11
4 小时前提问

1 个回答

默认排序 时间排序

有帮助, 有参考价值

 1 这应该是浏览器跨域问题吧? 可以尝试着了解下前端跨域怎么解决. 常见的解决方案有jsonp, 服务端设置Access-Control-Allow-Origin, nginx设置反向代理等

 评论 赞赏 编辑 ...
采纳

 flura RP 18
3 小时前回答

Session跨域

所谓Session跨域就是摒弃了系统(Tomcat)提供的Session,而使用自定义的类似Session的机制来保存客户端数据的一种解决方案.

如:通过设置cookie的domain来实现cookie的跨域传递. 在cookie中传递一个自定义的session_id,这个session_id是客户端的唯一标记. 将这个标记作为key,将客户端需要保存的数据作为value,在服务端进行保存(数据库保存或NoSQL保存),这种机制就是Session的跨域解决.

什么是跨域: 客户端请求的时候,请求的服务器,不是同一个IP,端口,域名,主机名的时候,都称为跨域.

什么是域: 在应用模型中, 一个完整的, 有独立访问路径的功能集合称为一个域,如: 百度称为一个应用或系统,百度下有若干的域,

127.0.0.1 localhost 两个地址的访问也叫跨域.

Spring Session共享

spring-session 技术是 spring 提供的用于处理集群会话共享的解决方案, spring-session 技术是将用户session数据保存到三方存储容器中, 如: mysql, redis 等.

spring-session 技术是解决同域名下的多服务器集群session共享问题的. **不能解决跨域session共享问题.**

这样的问题也是我考虑的:

有的同学问我, SSO系统登录后, 跳回原业务系统时, 带了个参数ST, 业务系统还要拿ST再次访问SSO进行验证, 觉得这个步骤有点多余. 他想SSO登录认证通过后, 通过回调地址将用户信息返回给原业务系统, 原业务系统直接设置登录状态, 这样流程简单, 也完成了登录, 不是很好吗?

其实这样问题时很严重的, 如果我在SSO没有登录, 而是直接在浏览器中敲入回调的地址, 并带上伪造的用户信息, 是不是业务系统也认为登录了呢? 这是很可怕的.

- 单点登录(SSO系统)是保障各业务系统的用户资源的安全
- 各个业务系统获得的信息是, 这个用户能不能访问我的资源
- 单点登录, 资源都在各个业务系统这边, 不在SSO那一方, 用户在给SSO服务器提供了用户名和密码后, 作为业务系统并不知道这件事, SSO随便给业务系统一个ST,那么业务系统是不能确定这个ST是用户伪造的还是真实有效的,所以要拿这个ST去SSO服务器再问一下,这个用户给我的ST是否有效,是有效的我才能让这个用户访问.

<https://www.jianshu.com/p/75edcc05acfd> <参考

[SpringBoot 整合Shiro实现动态权限加载更新+Session共享+单点登录](#)

[单点登录系统原理与实现, 图文并茂, 附源码](#)

[SpringCloud之SSO 单点登录\(附源码\)](#)

[简单的 CAS 实现 SSO 单点登录](#)