



性能优化专题

SSO 单点登录

主讲人：ROBERT: 2831742582

目录 /CONTENTS

01

概念原理

02

核心实现

03

CAS架设

04

应用场景

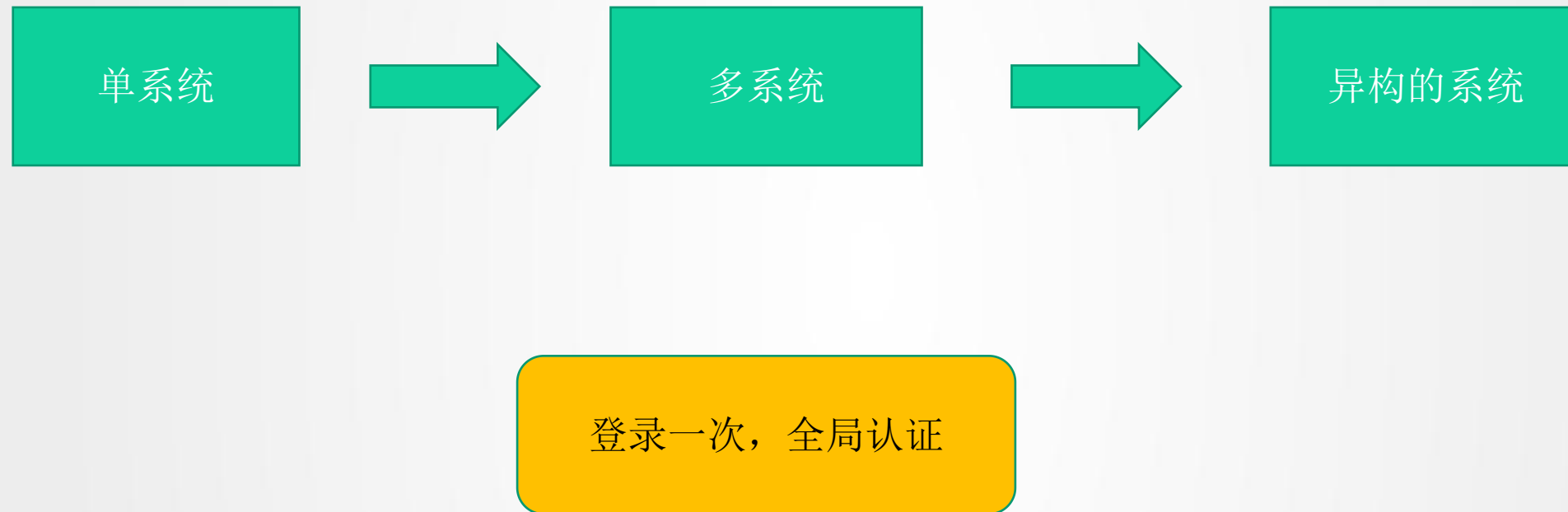
01

基本概念

BASIC CONCEPTS



什么是SSO

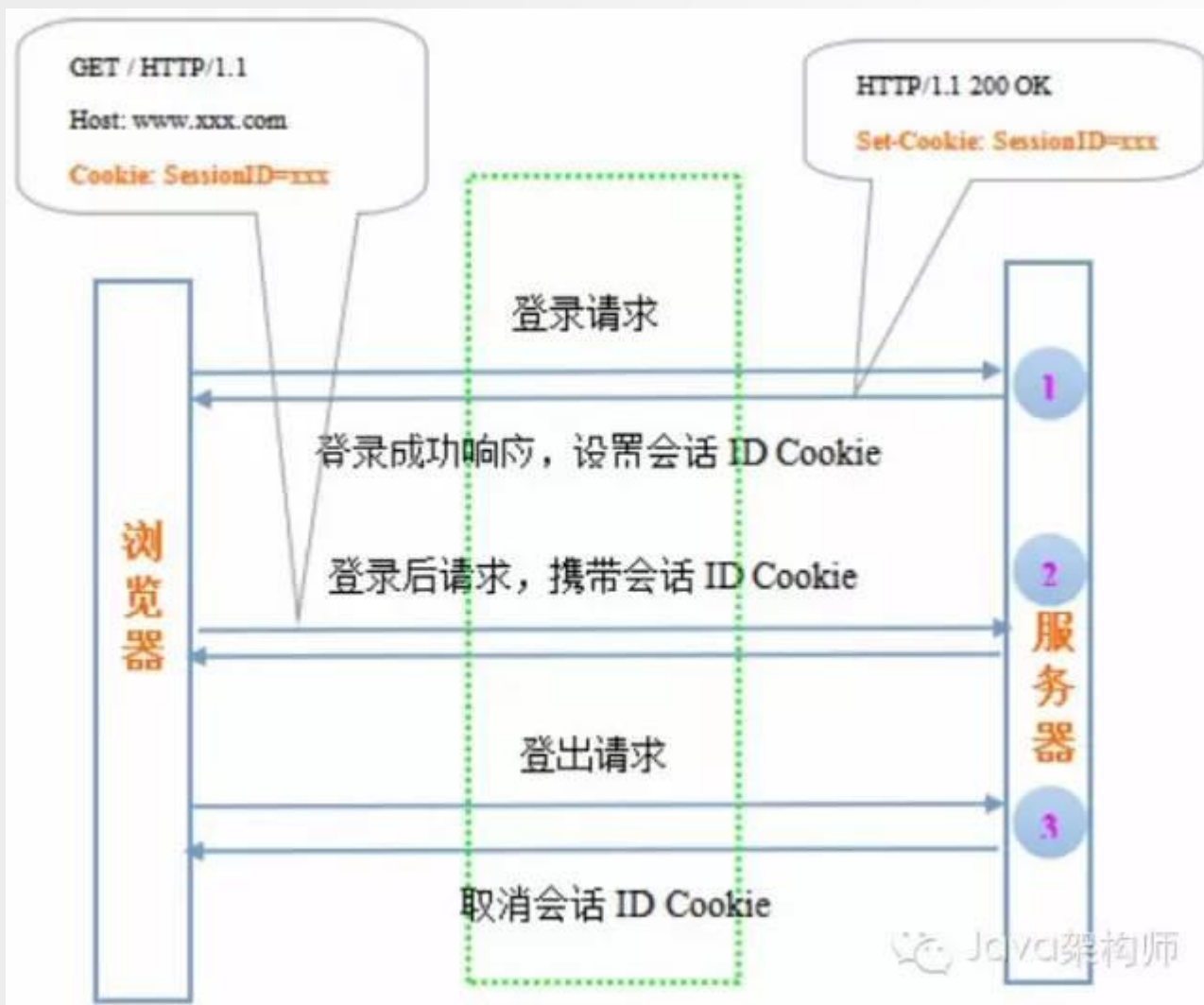


大中型Web应用基本都是多系统组成的应用群，SSO是必须面对的基本问题。

Cookie有作用域限制，顶级域名Cookie不能共享，故登录会话不能共享。

直接改造各子系统共享Session，通用性灵活性不强，或对原系统入侵性大，不是解决SSO根本办法。

单Web系统登录机理



认证(Authentication)操作，就是证明这个浏览器请求用户是合法系统用户，一般情况就是验证用户名和密码。

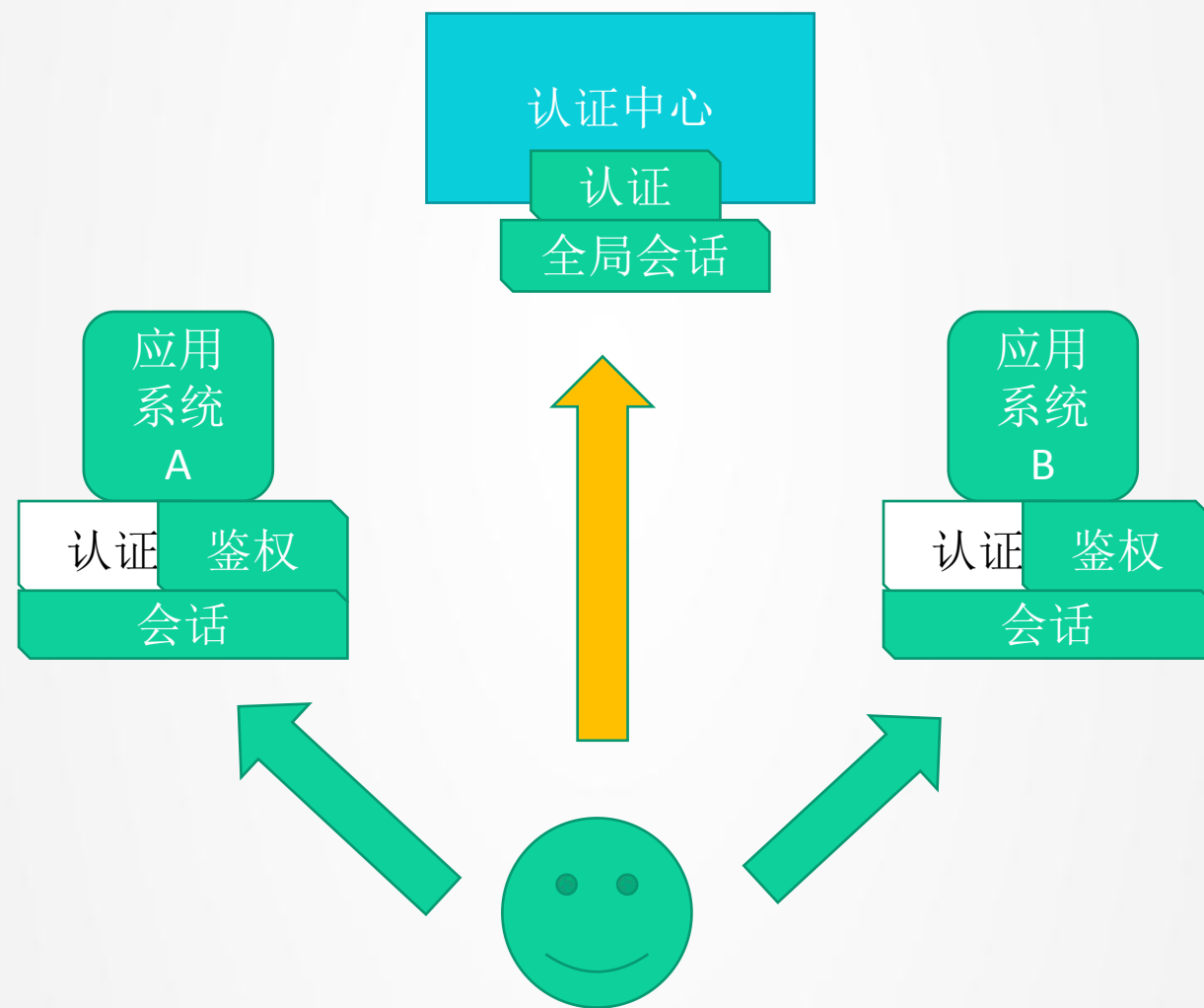
授权(Authorization)，就是根据该用户在此系统中的权限定义，绑定正确的权限信息，为用户后续正确使用系统功能提供安全保障。

建立会话，Session机制或自己基于Cookie开发的类似功能，建立起本次会话。

登录成功后，服务器需进行登录**状态判断**，识别操作是否是本次登录用户的操作。

登出时，服务端**取消会话**，本次登录用户会话结束。下次请求时，系统即判断是非登录用户。

单Web应用登录，主要涉及到认证、授权、会话建立、取消会话等几个关键环节。



关键问题

一、登录信息传递问题

由于认证中心系统与应用系统分开，需要传递是否登录及相关登录信息，以便后续操作。

二、登录状态判断问题

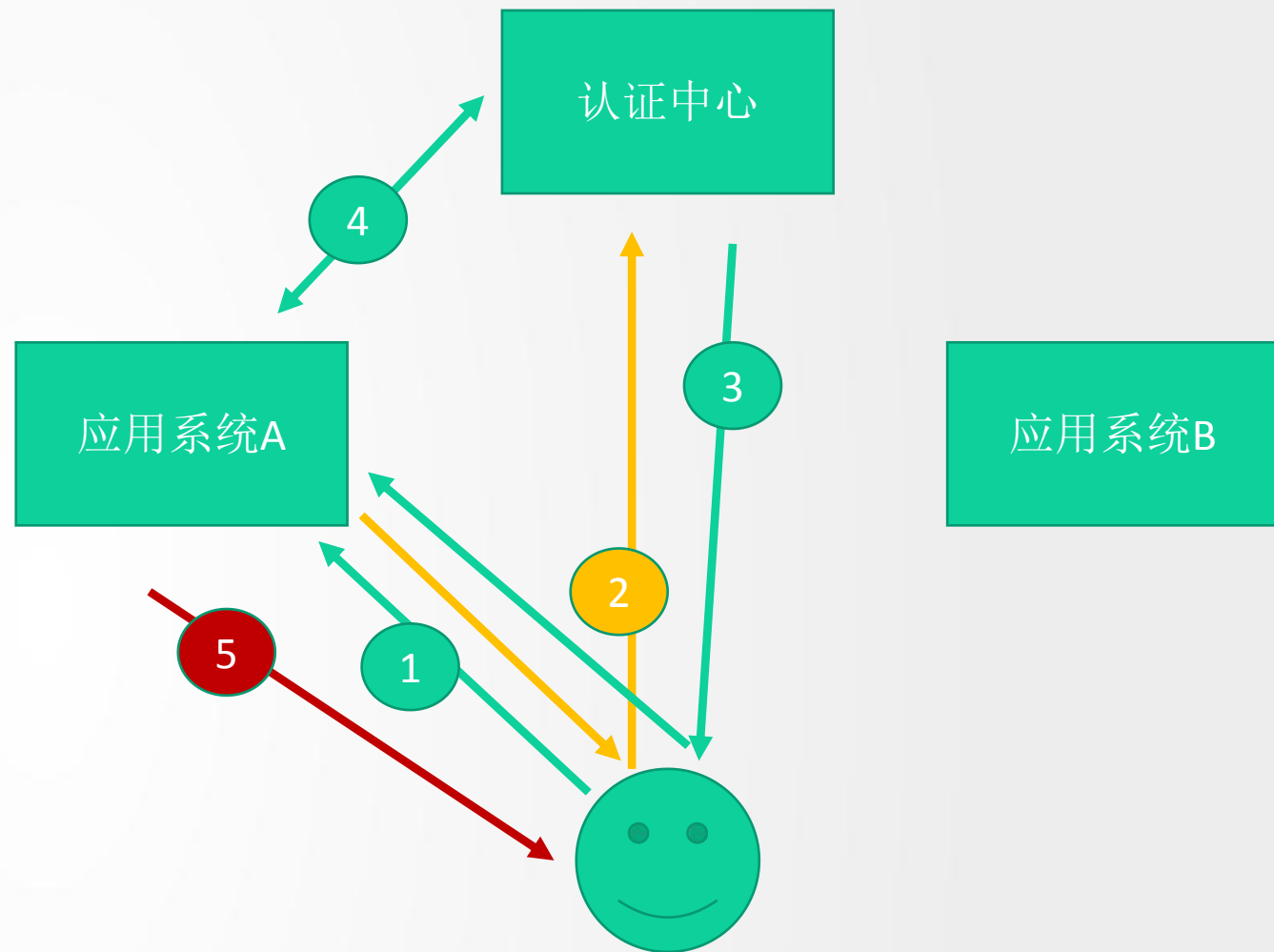
同样，当用户已在其他地方登录，访问该应用时。应用系统需要了解登录状态。

三、登出问题

认证中心注销后，所有应用系统同时注销。

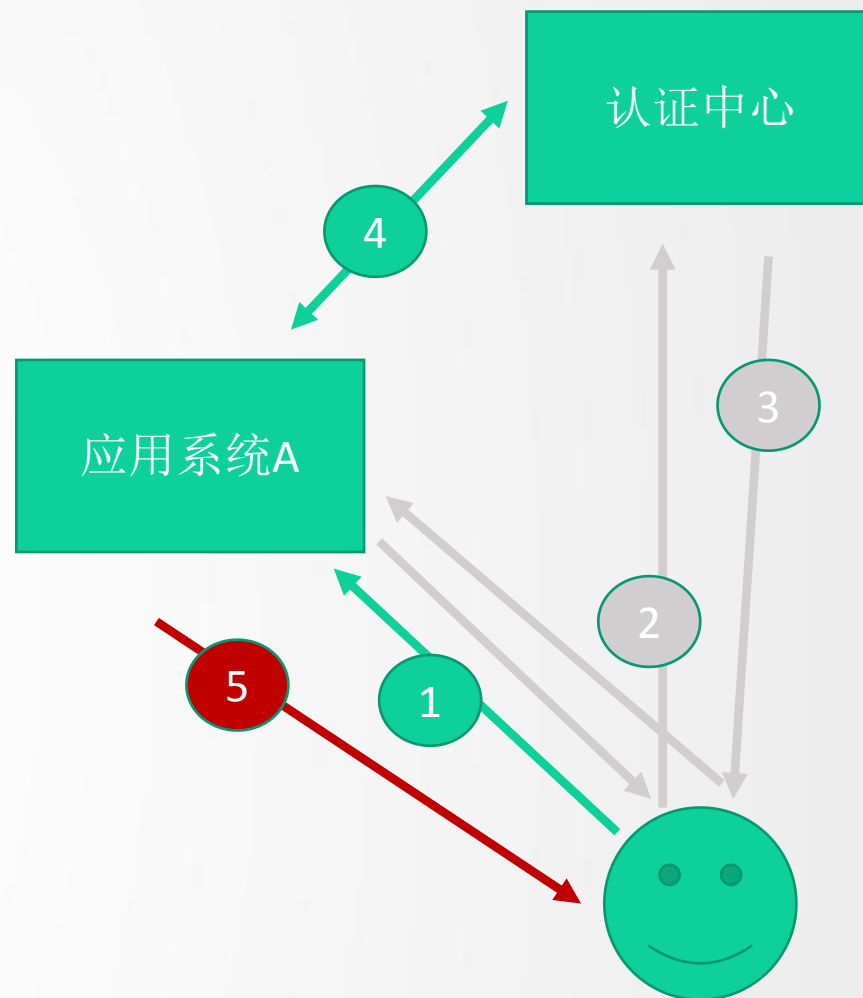
一、登录信息传递问题

1. 用户浏览器访问系统A需登录受限资源。
2. 系统A发现该请求需要登录，将请求重定向到认证中心，进行登录。
3. 认证中心呈现登录页面，用户登录，登录成功后，认证中心重定向请求到系统A，并附上认证通过令牌。
4. 系统A与认证中心通信，验证令牌有效，证明用户已登录。
5. 系统A将受限资源返给用户。



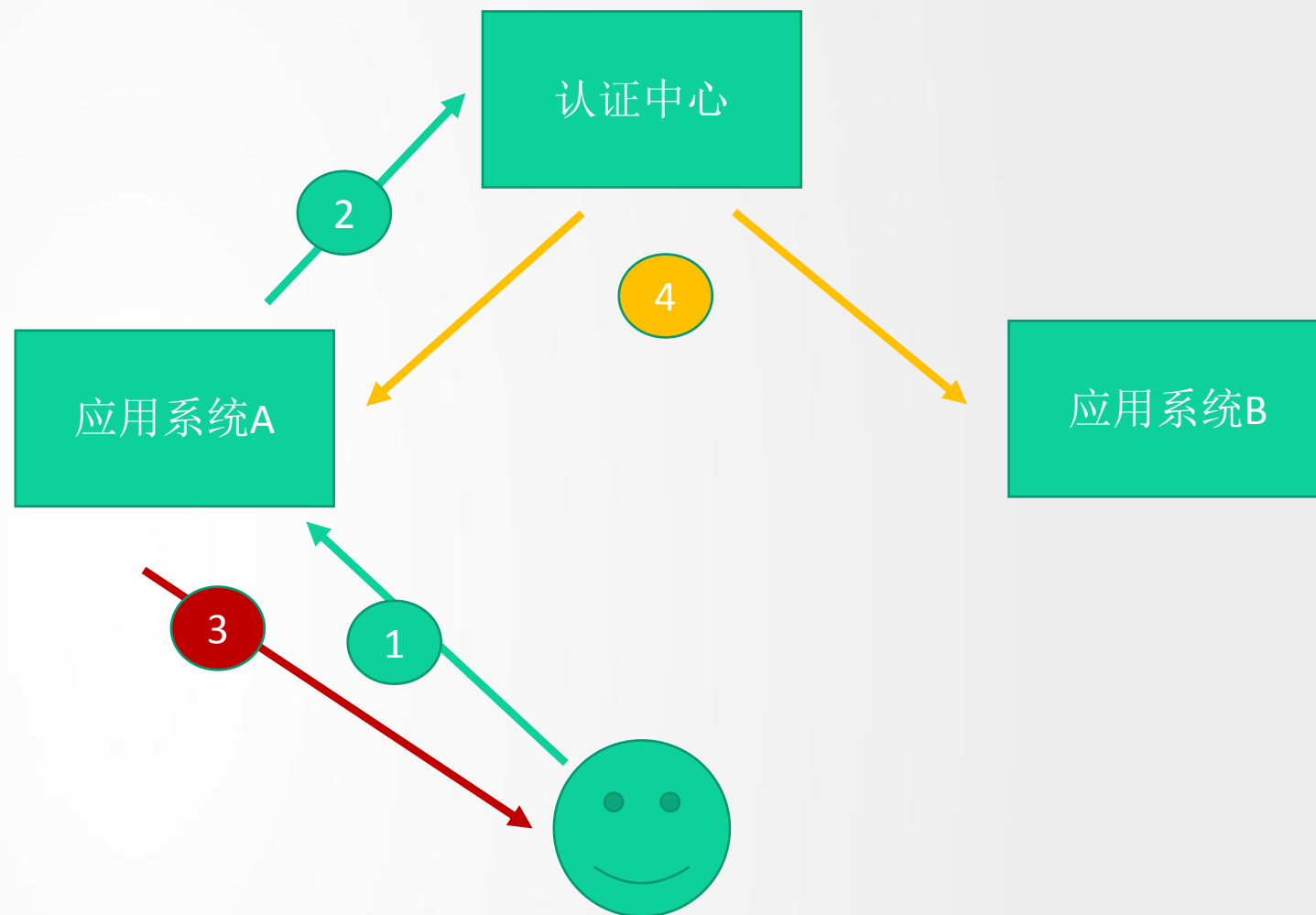
二、登录状态判断问题

1. 浏览器访问另一应用B需登录受限资源。
2. 系统B发现该请求需要登录，将请求重定向到认证中心，进行登录。
3. 认证中心发现已经登录，即重定向请求响应到系统B，附上认证令牌。
4. 系统B与认证中心通信，验证令牌有效,证明用户已登录。
5. 系统B将受限资源返回给客户端。



三、登出问题

1. 客户端向应用A发送登出Logout请求。
2. 应用A取消本地会话，同时通知认证中心，用户已登出。
3. 应用A返回客户端登出请求。
4. 认证中心通知所有用户登录访问的应用，用户已登出。



用户到认证中心登录后，用户和认证中心之间建立起了会话，我们把这个会话称为**全局会话**。当用户后续访问系统应用时，我们不可能每次应用请求都到认证中心去判定是否登录，这样效率非常低下，这也是单Web应用不需要考虑的。

我们可以在系统应用和用户浏览器之间建立起局部会话，局部会话保持了客户端与该系统应用的登录状态，局部会话依附于全局会话存在，全局会话消失，局部会话必须消失。

用户访问应用时，首先判断**局部会话**是否存在，如存在，即认为是登录状态，无需再到认证中心去判断。如**不存在**，就重定向到认证中心判断全局会话是否存在，如存在，按1提到的方式通知该应用，该应用与客户端就建立起它们之间局部会话，下次请求该应用，就不去认证中心验证了。



02

核心实现

CORE

四大接口

1、面向用户的登录接口

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

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

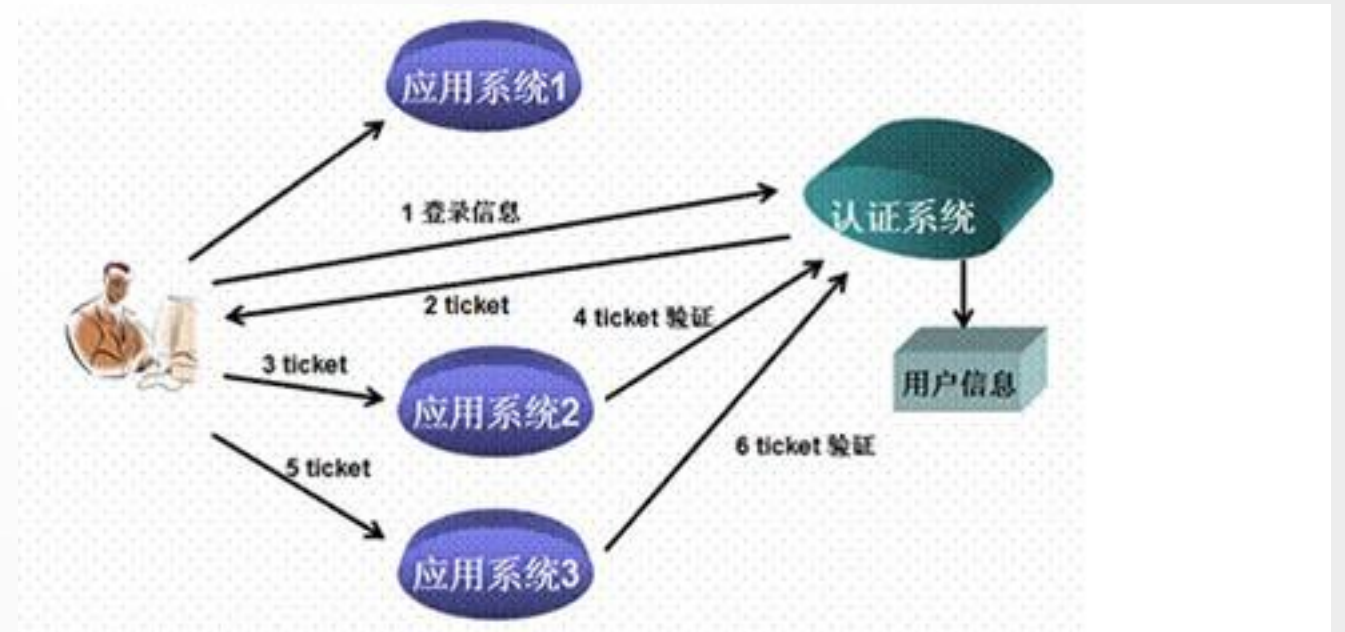
2、面向应用系统令牌校验接口

```
@RequestMapping("/verify")
@ResponseBody
public String verify(String token, HttpServletRequest
    req) {
    this.verify(token);
    return "ok";
}
```

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

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

- 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。



四大接口

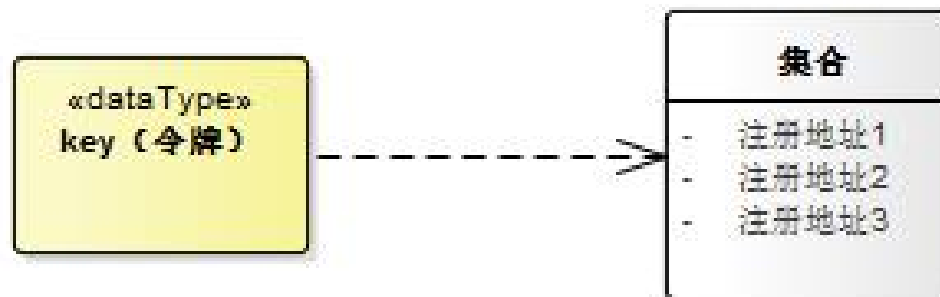
3、面向应用系统用户验证接口

```
Verify(String token) {  
    TokenInfo token = getByToken(token);  
    return !token.isExpired();  
}
```

4、面向应用系统注销接口

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

class 单点登录原理



两个功能

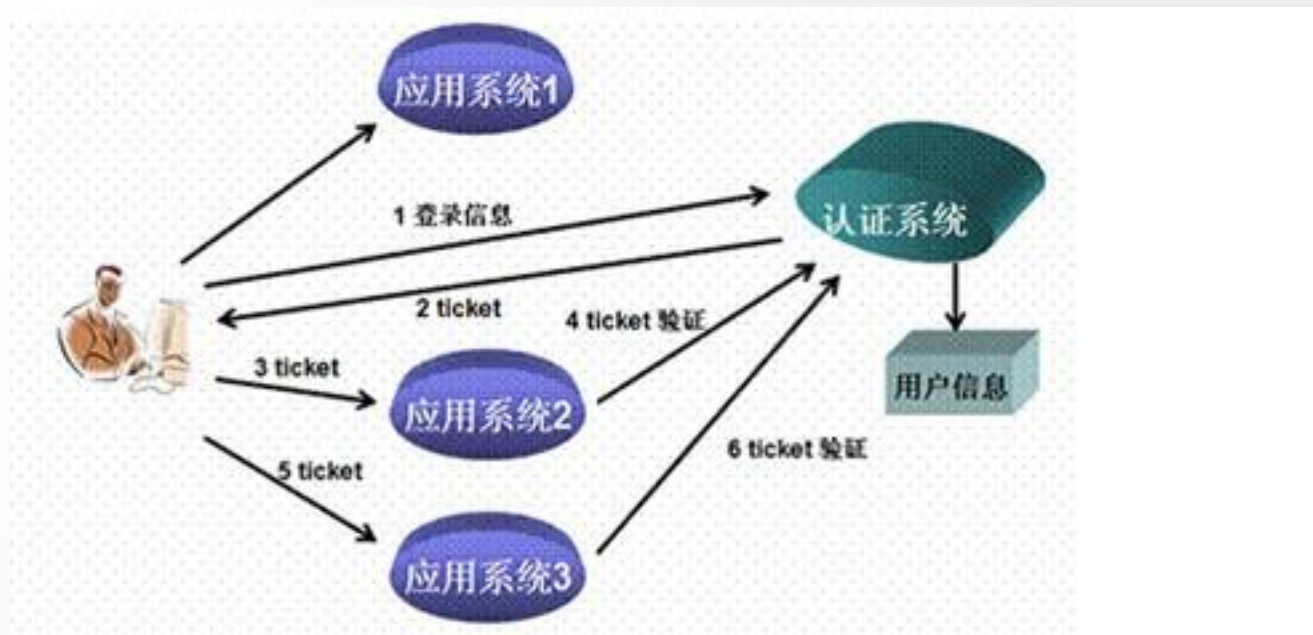
1、用户验证过滤器

```
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-login-url");  
}
```

2、登出接口

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

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认证中心发出的注销请求，销毁局部会话



03

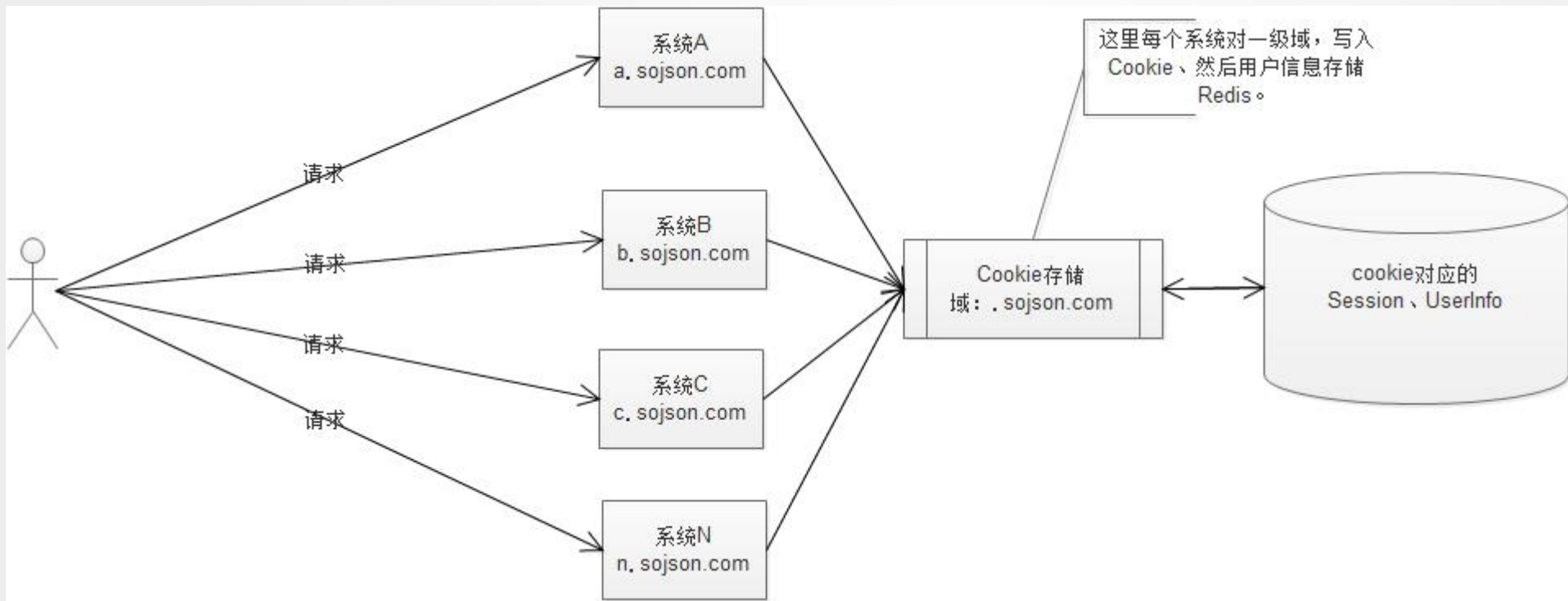
CAS架设

CAS



四种方式

N个系统，但是一级域名 是一致的。



PS: 这个方案比较简单，只要提供公共的 SDK 即可，不需要第三个系统的出现，这个 SDK 的工作需要管理 Cookie 和用户信息。

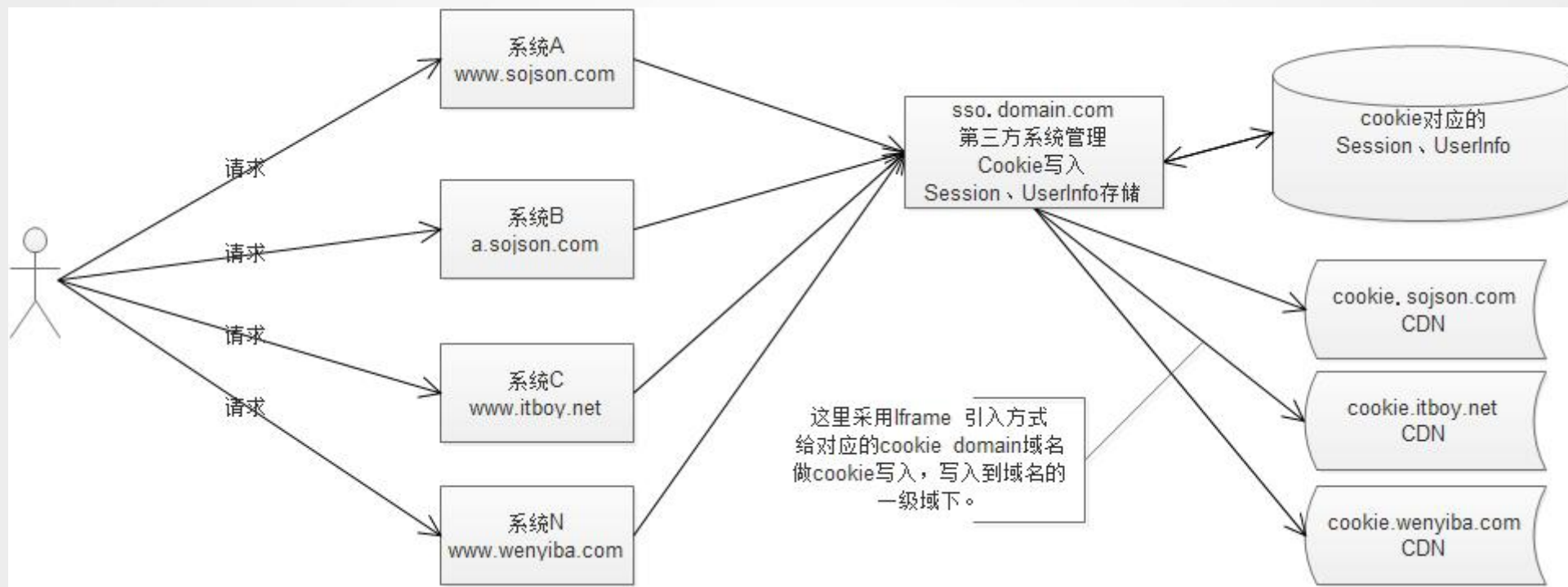
原理: 其实质这里就是利用了 二级域名 写 一级域名 的 Cookie 。

优点: 轻量级、可插拔、效率非常高。

缺点: 局限性限于一级域名是一样的。

四种方式

域名比较乱，有同一个一级域名的（www.xxx.com、a.xxx.com），也有不同域名的



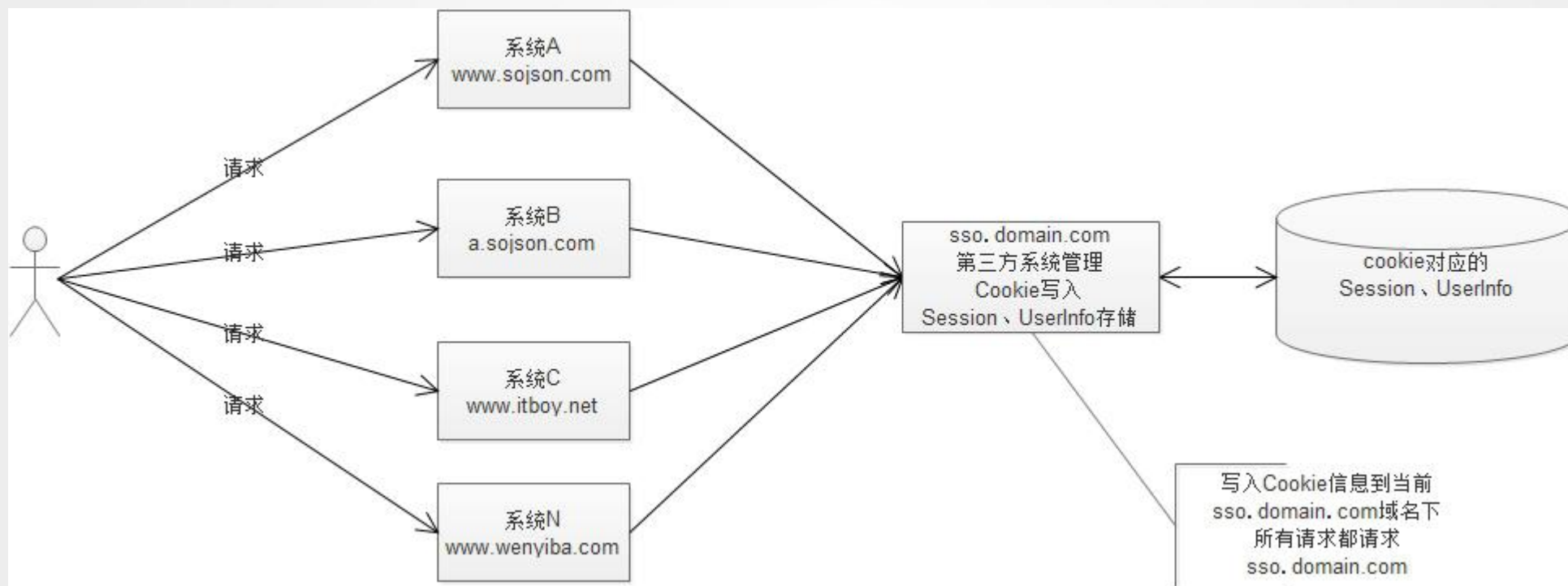
原理：通过SSO 系统（登录、退出）， **Iframe** 引用的方式引入Cookie.domain.com的方式，利用 **Javascript** 操作（写入 / 删除 / 修改） **cookie** ，而这个cookie.domain.com 域名是放入 **CDN** 上，获取用户信息当前系统直接通过 **Redis** （只读）获取。

优点：因为是采用压力分化，Cookie.domain.com 部署在**CDN**上，这样的话，对各个系统造成的压力是 **0** ，用第三方系统（**SSO**）维护，权限更大，操作性更强，但又Cookie 信息在当前域名的一级域下，获取简单，大量减少对 **sso** 的访问量。

缺点：如果浏览器安全性过高，Iframe 的方式操作 **Cookie** 将会失败。比如IE浏览器，目前正在攻克IE浏览器。

四种方式

域名比较乱，有同一个一级域名的（www.xxx.com、a.xxx.com），也有不同域名的。



原理：所有的请求（登录、退出、获取用户信息、当前用户状态）都请求sso系统，sso系统维护用户信息，Session，UserInfo。

优点：实现较为简单。

缺点：SSO压力非常大。

四种方式

CAS是中央认证服务Central Authentication Service的简称。最初由耶鲁大学的Shawn Bayern 开发，后由Jasig社区维护，经过十多年发展，目前已成为影响最大、广泛使用的、基于Java实现的、开源SSO解决方案。

2012年，Jasig和另一个有影响的组织Sakai Foundation合并，组成Apereo。Apereo是一个由高等学术教育机构发起组织的联盟，旨在为学术教育机构提供高质量软件，当然很多软件也被大量应用于商业环境，譬如CAS。目前CAS由Apereo社区维护。

CAS的官方网址是： <https://www.apereo.org/projects/cas>

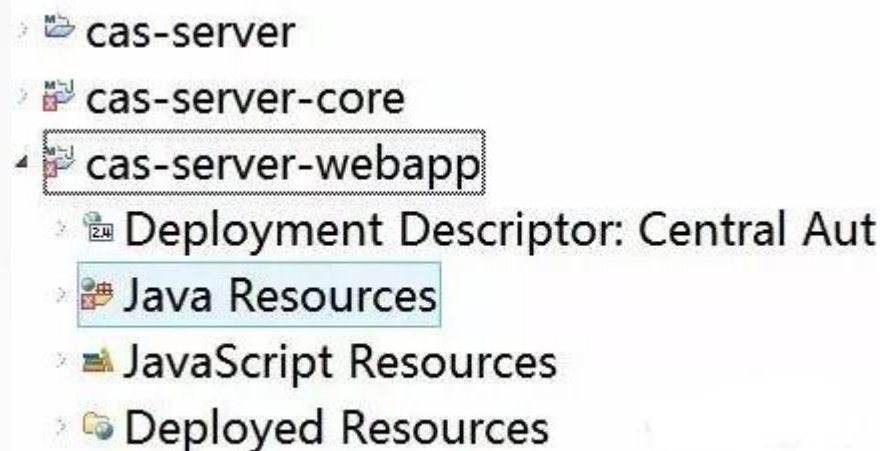
工程代码网址： <https://github.com/Jasig/cas>

客户端下载地址： www.ja-sig.org/downloads/cas-clients/

1、解压到硬盘如e:\cas-4.1.0，只导入三个重要工程：父工程cas-server、核心包工程cas-server-core、运行web包工程cas-server-webapp。

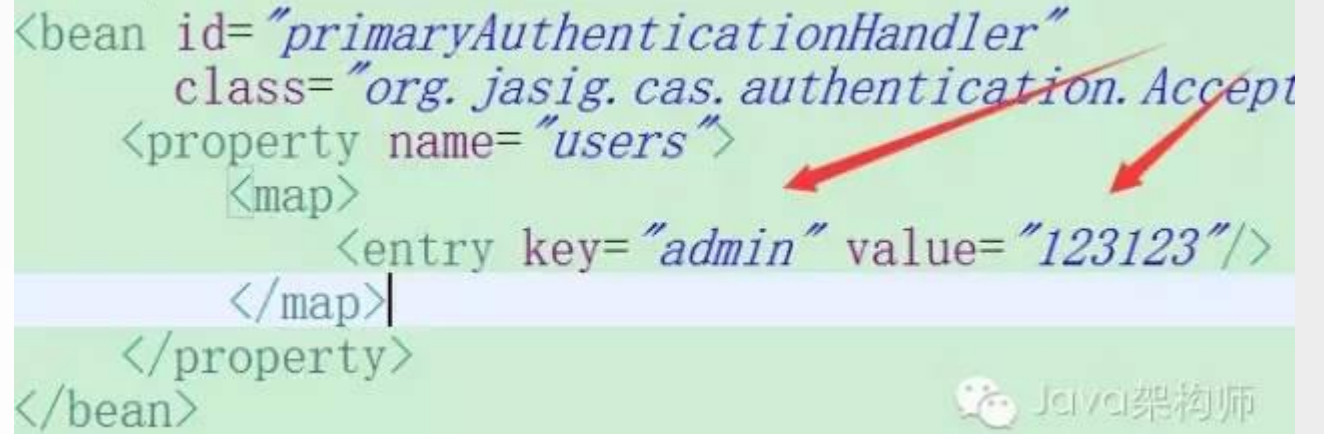
2、此时pom.xml可能提示有错误，按eclipse提示更正信息ignore即可。然后我们利用maven compile、package cas-server-webapp工程。正常情况下会生成cas.war 认证中心安装包。

```
Webapp assembled in [1094 msecs]  
Building war: E:\cas-4.0.4\cas-server-webapp\target\cas.war  
Packaging classes  
Building jar: E:\cas-4.0.4\cas-server-webapp\target\cas-class  
--- maven-site-plugin:3.1.r1174614:attach-descriptor (attach-
```



3、找到WEB-INF\deployerConfigContext.xml,找到primaryAuthenticationHandler段落，修改登录用户名为admin，密码123123 做登录体验用。
(CAS缺省使用的是配置文件方式管理用户名和密码，实际应用中最多的是利用数据库管理，CAS提供相应接口)

```
<bean id="primaryAuthenticationHandler"  
      class="org.jasig.cas.authentication.AcceptingAuthenticationHandler"  
      <property name="users">  
        <map>  
          <entry key="admin" value="123123"/>  
        </map>  
      </property>  
</bean>
```



Java架构师

4、找到WEB-INF\spring-configuration\ticketGrantingTicketCookieGenerator.xml文件，将p:cookieSecure的值改为false。CAS缺省要求使用https协议，我们将此改为false，使其在http协议下也能工作。

5 在本机的host文件中配置一个域名，如www.cas.com,浏览器输入该网址，即可出现cas登录页



- 1.我们在host文件中再配置一个域名,www.ssoclient.com用于表示此应用系统登录网址。
- 2.建立一个java web maven工程，端口号设置为81，将CAS Client包配置到pom.xml中。

```
<dependency>  
<groupId>org.jasig.cas.client</groupId>  
<artifactId>cas-client-core</artifactId>  
<version>3.3.3</version>  
</dependency>
```


首先配置登出listener和filter

```
<listener>
<listener-
class>org.jasig.cas.client.session.SingleSignOutHttpSessionListene
r</listener-class>
</listener>
<filter>
<filter-name>CAS Single Sign Out Filter</filter-name>
<filter-
class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>CAS Single Sign Out Filter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

接着配置认证filter，即受限资源需要先经过此filter.注意这里面要配置认证中心登录网址，以及标识此系统应用的服务名称。

```
<filter>
<filter-name>CAS Authentication Filter</filter-name>
<filter-
class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-
class>
<init-param>
<param-name>casServerLoginUrl</param-name>
<param-value>http://www.cas.com/login</param-value>
</init-param>
<init-param>
<param-name>serverName</param-name>
<param-value>http://www.ssoclient.com:81</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>CAS Authentication Filter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

接下来配置验证票据ticket的filter令牌token:

```
<filter>
<filter-name>CAS Validation Filter</filter-name>
<filter-
class>org.jasig.cas.client.validation.Cas20ProxyReceivingTicketVal
idationFilter</filter-class>
<init-param>
<param-name>casServerUrlPrefix</param-name>
<param-value>http://www.cas.com</param-value>
</init-param>
<init-param>
<param-name>serverName</param-name>
<param-value>http://www.ssoclient.com:81</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>CAS Validation Filter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

最后我们配置一个Filter封装标准的HttpRequest，使得request.getRemoteUser()和request.getUserPrincipal()这两个方法可用。

```
<filter>
<filter-name>CAS HttpServletRequest Wrapper Filter</filter-
name>
<filter-
class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</f
ilter-class>
</filter>
<filter-mapping>
<filter-name>CAS HttpServletRequest Wrapper Filter</filter-
name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

在这样配置下，整个应用系统都是受限资源，我们定义一个一句话的首页JSP：

```
${user}, 你好啊！ <a href="http://www.cas.com/logout">登出  
</a>
```

对应的controller也很简单：

```
@RequestMapping("/index.do")  
public String showIndex(HttpServletRequest request,  
    HttpServletResponse response) {  
    Principal principal = request.getUserPrincipal();  
    request.getSession().setAttribute("user", principal.getName());  
    return "/index";  
}
```


04

应用场景

APPLY

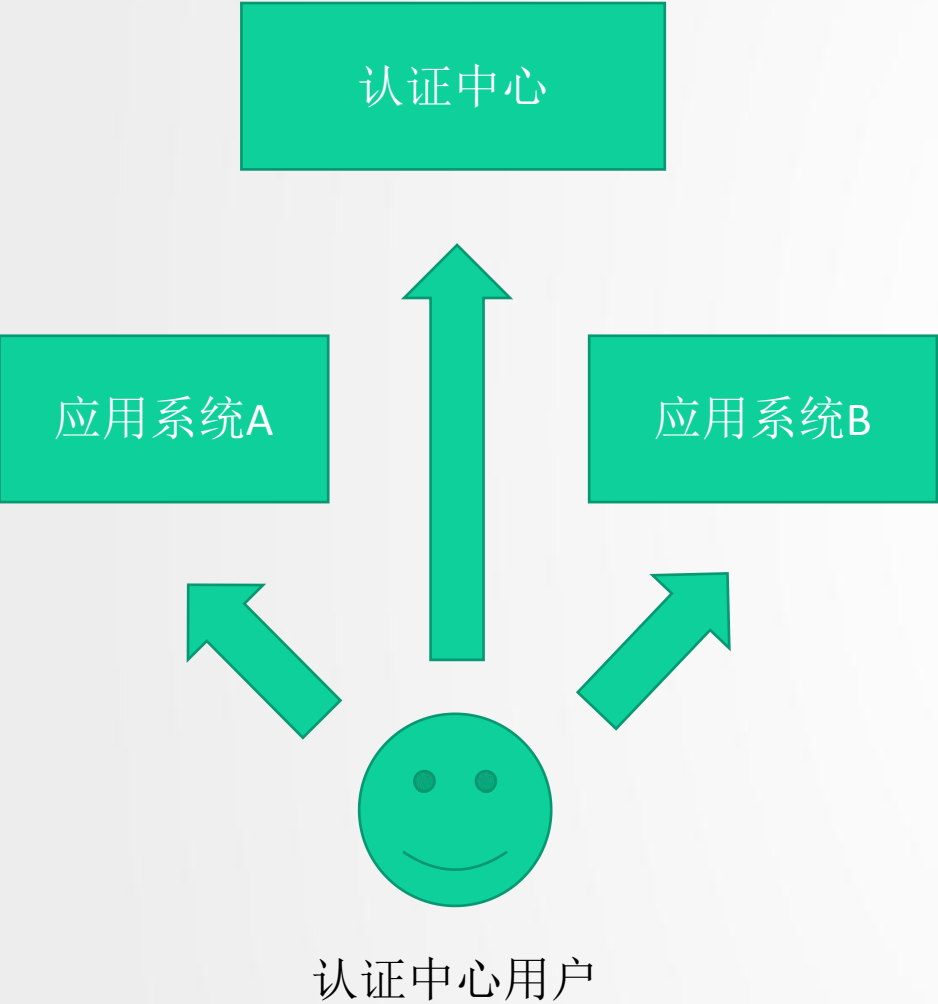


多个认证中心的处理

应用系统包含JAVA, PHP等多语言, 异构系统的认证

应用系统移动端的处理

采用公认认证标准及系统, 如CAS



只需要在B系统内增加一个单点注销，及用户直接登录页面。

过滤器中添加本地验证过滤器


```
<b:bean id="casAuthenticationManager" class="org.springframework.security.authentication.ProviderManager">
  <b:constructor-arg>
    <b:list>
      <b:bean id="casAuthenticationProvider" class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
        <b:property name="authenticationUserDetailsService" ref="casAuthenticationUserDetailsService" />
        <b:property name="serviceProperties" ref="serviceProperties" />
        <b:property name="ticketValidator">
          <b:bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">
            <b:constructor-arg index="0" value="${cas.url}" />
          </b:bean>
        </b:property>
        <b:property name="key" value="an_id_for_this_auth_provider_only" />
      </b:bean>
      <b:bean id="localAuthenticationProvider" ref="localAuthenticationProvider">
        <b:property name="userDetailsService" ref="userDetailsService" />
      </b:bean>
    </b:list>
  </b:constructor-arg>
</b:bean>
```