公共安全平台SDK详细设计

fort SDK

2016年5月

1、引言

1.1、编写目的

公共安全平台SDK详细设计是设计的第二个阶段，这个阶段的主要任务是在公共安全平台SDK详细设计概要设计书基础上，对概要设计中产生的功能模块进行过程描述，设计功能模块的内部细节，包括算法和详细数据结构，为编写源代码提供必要的说明。

1.2、项目背景

@see 公共安全平台详细设计。

为了便于新应用的开发，本SDK封装了公共安全平台的接口，实现了SecurityClient、SecurityHttpFilter、WebSocketSTOMPClient

1.3、参考资料

Spring WebSocket Support、SSO

2、总体设计

2.1、需求描述

达到权限控制的基础功能，使用SDK实现登录、注册、登出、用户信息修改、URL级别的权限控制、导航栏权限控制、不同一级域单点登录。同时，系统最大限度地实现易安装，易维护性，易操作性，运行稳定，安全可靠。

2.2、软件架构

项目启动时，通过SecurityClient获得全部SecurityResource（不包括SecurityUser)放到Security Resource Cache中，并启动WebSocketSTOMPClient和fort建立连接，订阅资源更新消息。

2.3、规范

2.3.1 JDK版本

由于SDK是在开发者的应用内运行的，为了更好的兼容，使用JDK1.7版本开发。

3、程序描述

3.1、Security Resource Cache

安全资源缓存

字段名	类型	描述
resourceEntities	Map<Long, SecurityResourceEntity>	安全资源实体缓存
navs	Map<Long, SecurityNav>	安全导航栏缓存
authorities	Map<Long, SecurityAuthority>	安全权限缓存
roles	Map<Long, SecurityRole>	安全角色缓存

3.2、Security Client

安全客户端，封装了fort的http接口。

3.3、Security Context

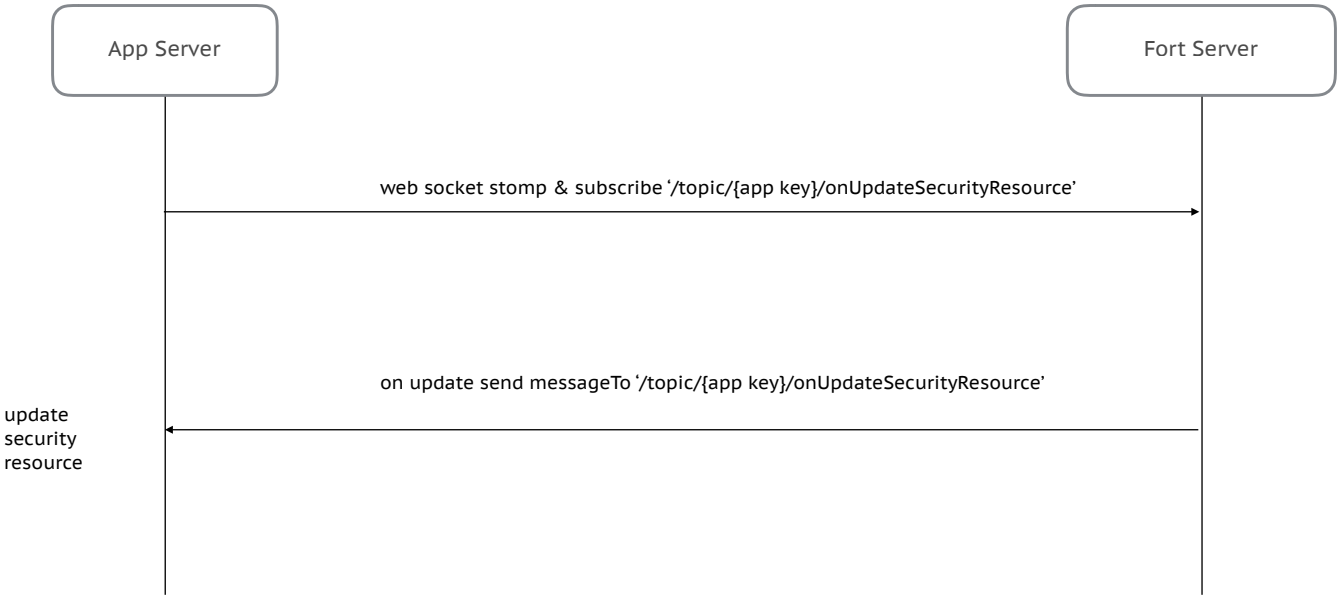
安全上下文，可以从上下文中获得当前登录人的用户名、用户令牌、权限列表、等。

3.4、Security Http Filter

安全Http过滤器，拦截用户请求，判断用户是否有权访问此资源。
可配置哪些资源不需要过滤，可配置403视图地址。

3.5、WebSocket STOMP Client

stomp client，订阅资源更新消息。



消息通过String JSONArray方式发送，每条消息的数据结构如下：

字段名	类型	描述
option	String	选项(RESTful风格)，新增：POST，更新：PUT，删除：DELETE
resourceClass	String	更新的资源类名

字段名	类型	描述
data	Object	数据

4、SSO单点登录

4.1、同域

- http://www.mydomain.com/site1
- http://www.mydomain.com/site2

这两个站点共享同样的主机地址(同样的域mydomain.com和子域www)，且两个站点都被配置成了对用户验证和授权都使用表单验证。假设你已经登录过了站点www.mydomain.com/site1，如前所述，你的浏览器现在对于站点www.mydomain.com/site1已经有了表单验证的cookie。

现在你随意访问以www.mydomain.com/site1开头的URL，表单验证的cookie都将被包含在请求被发送。为什么？是因为此cookie本来就属于该站点吗？对的，但不是完全正确。事实上，是因为请求的URL：www.mydomain.com/site1和http://www.mydomain.com/拥有同样的域名和子域名。

那么在你登录了www.mydomain.com/site1后，如果你点击www.mydomain.com/site2下的URL，表单验证的cookie也将被包含在请求中发送，这同样是因为www.mydomain.com/site2与站点http://www.mydomain.com/拥有同样的域名和子域名，尽管它是不一样的应用站点(site2)。显然，在拥有一样主机地址不一样的应用站点名之间是可以共享表单验证cookie的，这样就实现了一处登录处处都已经登录的功能(也就是单点登录)。

4.2、同域，不同子域

- http://site1.mydomain.com/
- http://site2.mydomain.com/

这两个站点共享同样的域(同样的二级域名mydomain.com)，但拥有不一样的三级域名(不一样的子域site1和site2)。
默认情况下浏览器仅仅发送主机地址一样(相同的域和子域)的站点的cookie。因此站点site1.mydomain.com不能获取到站点site2.mydomain.com下的cookie(因为他们没有相同的主机地址，它们的子域不同)，尽管你为这两个站点配置了相同的machineKey，一个站点还是不能获取另一个站点下的cookie。
除了你为所有的站点配置了一样的machineKey，你还需要为验证cookie定义相同的域以使得浏览器在同样的域名下能够发送任何请求。

你需要像下面这样配置表单验证cookie：

<forms name="name" loginUrl="URL" defaultUrl="URL" domain="mydomain.com"/>

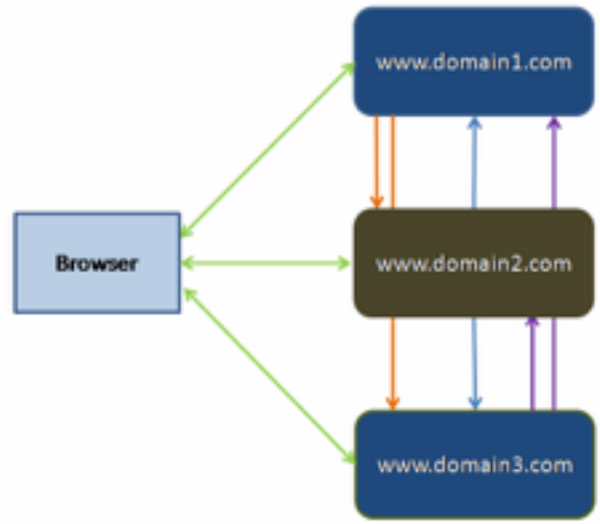
4.3、跨域

- http://www.domain1.com/
- http://www.domain2.com/
- http://www.domain3.com/

为了实现在这些站之间实现SSO，当用户在任意一个站登录时，我们需要为所有的站点设置验证cookie。

如果用户1登录进http://www.domain1.com/，那么在给站点1response前会在response中加入验证的cookie，但当我们需要同时能够登录进http://www.domain2.com/和http://www.domain3.com/时，我们需要同时在同样的客户端浏览器上为站点2和站点3设置验证cookie。因此，在response返回到浏览器前，站点1不得不定向到站点2和站点3去设置验证cookie。

下面的流程图详细描述了思路：



4.3.1、此单点登录模型的缺点

这个模型在两个站点上还是能运行的很好的。从一个站点登录或注销，此SSO模型下的站点都将遵从请求-重定向-返回的流程。当用户登录任一页面时，因为已经存储了所有站点的验证cookie，那么就不需要再执行上面的那个循环的流程了。

但是当站点超过两个时，问题就变得复杂了，当登录站点1时，程序将重定向到站点2和站点3进行验证cookie的设置，最后站点3在跳转到站点1，服务器返回用户请求的页面。这使得每个站点的登录和注销的过程变得复杂并花费较高的代价。如何超过3个站点呢？如果这样去设计20+站点的单点登录呢？这个模型将完全不能胜任了。

并且此模型需要每个站点都具备用户验证逻辑，因为需要来请求此站点并设置其验证cookie。因此此模型丢失了一般意义上的单点登录的概念，我们需要一个更好一点的模型去实现单点登录的功能。

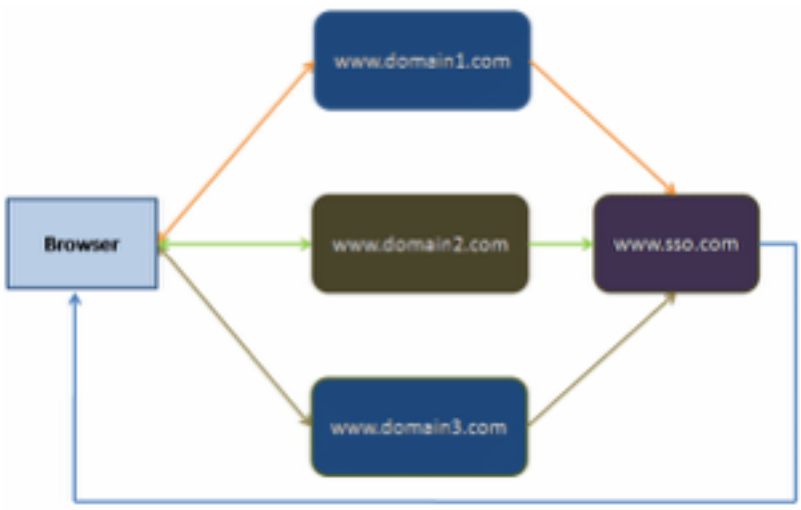
4.3.2、更好的跨域单点登录架构

前面提到的架构中，设置移除cookie都需要跳转到N-1个站点去完成。每个站点还需要知道N-1个站点复杂的登录注销逻辑，

如果我们为所有的站点只去维护一份验证cookie呢？使用一个独立的站点去完成验证用户并设置验证cookie的工作呢？这个想法好像不错。要使用单点登录，那么就需要用户的数据是统一的，这样的话就可以通过一个站点提供web或者WCF服务来完成验证和授权的功能。这样就省去了冗余的用户验证逻辑，现在最重要的是这个独立的站点如何在SSO架构中起作用。

在这个架构模型中，浏览器不存储任何其他站点的验证cookie，只存那个独立站点的验证cookie，我们就给它起名叫http://www.sso.com/。在此架构中，对每一个站点的请求都将被直接跳转到http://www.sso.com/，由于检查验证cookie是否存在。如果cookie存在，如果存在，返回请求的页面，如果不存在，那么就跳转到对应的登录页面。

大致流程图如下：



刚开始，浏览器没有任何http://www.sso.com/站点下的验证cookie。请求站点1和站点2任何需要验证的页面(需要内部的跳转到sso站点检查验证cookie是否存在)。用户登录后，sso站点的验证cookie存储在本地（重要的是用户令牌仅仅用户用户登录会话时）。

现在请求站点1或者站点2都跳转到sso站点，浏览器发送sso站点的验证cookie并检查用户令牌，验证后再跳转到原始请求的URL，原始站点检查用户令牌正确后返回用户请求的页面。