[**CAS 4.0 单点登录教程**](http://alexwangjie.iteye.com/blog/2117552)

**博客分类：**

* [SSO](http://alexwangjie.iteye.com/category/323278)

[CAS](http://www.iteye.com/blogs/tag/CAS)[SSO](http://www.iteye.com/blogs/tag/SSO)

**1.概述**   
单点登录（Single Sign On），简称为 SSO，是目前比较流行的企业业务整合的解决方案之一。SSO的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。   
耶鲁大学（yale）开发的单点登录（Single Sign On）系统称为CAS（Central Authentication Service）被设计成一个独立的Web应用程序(cas.war)。   
CAS在2004年12月成为Jasig项目，所以也叫JA-SIG CAS。   
本文中服务器版本基于4.0.0版本，对应的客户端版本为3.2.1；   
官网：<http://jasig.github.io/cas/4.0.0/index.html>   
  
**2. CAS 原理和协议**   
从结构上看，CAS 包含两个部分： CAS Server 和 CAS Client。CAS Server 需要独立部署，主要负责对用户的认证工作；CAS Client 负责处理对客户端受保护资源的访问请求，需要登录时，重定向到 CAS Server。图1 是 CAS 最基本的协议过程：   
图 1. CAS 基础协议   
  
  
   
  
CAS Client 与受保护的客户端应用部署在一起，以 Filter 方式保护受保护的资源。对于访问受保护资源的每个 Web 请求，CAS Client 会分析该请求的 Http 请求中是否包含 Service Ticket，如果没有，则说明当前用户尚未登录，于是将请求重定向到指定好的 CAS Server 登录地址，并传递 Service （也就是要访问的目的资源地址），以便登录成功过后转回该地址。用户在第 3 步中输入认证信息，如果登录成功，CAS Server 随机产生一个相当长度、唯一、不可伪造的 Service Ticket，并缓存以待将来验证，之后系统自动重定向到 Service 所在地址，并为客户端浏览器设置一个 Ticket Granted Cookie（TGC），CAS Client 在拿到 Service 和新产生的 Ticket 过后，在第 5，6 步中与 CAS Server 进行身份合适，以确保 Service Ticket 的合法性。   
在该协议中，所有与 CAS 的交互均采用 SSL 协议，确保，ST 和 TGC 的安全性。协议工作过程中会有 2 次重定向的过程，但是 CAS Client 与 CAS Server 之间进行 Ticket 验证的过程对于用户是透明的。   
另外，CAS 协议中还提供了 Proxy （代理）模式，以适应更加高级、复杂的应用场景，具体介绍可以参考 CAS 官方网站上的相关文档。   
  
**3.环境**   
.Tomcat 7.x 本文中安装目录为: D:\apache-tomcat-7.0.54   
.JDK 7 本文中安装目录为: D:\Java\jdk1.7.0\_60   
.CAS Server版本: cas-server-4.0.0   
.CAS Client版本: cas-client-3.2.1   
  
**4.配置证书**   
证书是单点登录认证系统中很重要的一把钥匙，客户端于服务器的交互安全靠的就是证书；本教程由于是演示所以就自己用JDK自带的keytool工具生成证书；如果以后真正在产品环境中使用肯定要去证书提供商去购买，证书认证一般都是由VeriSign认证，中文官方网站：[http://www.verisign.com/cn/。](http://www.verisign.com/cn/%E3%80%82)   
也可以申请免费的StartSSL CA证书: StartSSL(公司名：StartCom)也是一家CA机构，它的根证书很久之前就被一些具有开源背景的浏览器支持(Firefox浏览器、谷歌Chrome浏览器、苹果Safari浏览器等)。 申请地址：http://www.startssl.com 申请方法参考：<http://www.linuxidc.com/Linux/2011-11/47478.htm>   
  
**1> 创建证书**   
首先创建文件夹：D:/cas/keys用于存放相关的证书文件。   
然后打开windows 命令控制台：（开始 --> cmd ->回车）   
切换到JDK安装目录中Key: D:\Java\jdk1.7.0\_60\jre\bin   
输入以下命令：   
keytool -genkey -alias mycas -keyalg RSA -keysize 2048 -keystore d:/cas/keys/mycas.keystore   
  
接下来会有提示 如图2：   
图2. 创建证书：   
  
  
   
此步完成后可在D:/cas/keys文件夹下找到mycas.keystore文件   
  
**2>导出证书**   
输入命令：   
keytool -export -file d:/cas/keys/mycas.crt -alias mycas -keystore d:/cas/keys/mycas.keystore   
此步完成后可在D:/cas/keys文件夹下找到mycas.crt文件   
  
**3>客户端JVM导入证书**   
在客户端的JDK安装目录下输入命令：   
keytool -import -keystore d:\Java\jdk1.7.0\_60\jre\lib\security\cacerts -file d:/cas/keys/mycas.crt -alias mycas  
其中：d:\Java\jdk1.7.0\_60\jre\lib\security\cacerts为客户端JVM的密钥库位置   
如果提示：   
keytool error: java.io.IOException: Keystore was tampered with, or password was incorrect   
那么输入密码：changeit   
这是因为JDK安装后会默认创建一个密钥库，密码为：changeit   
也可以删除d:\Java\jdk1.7.0\_60\jre\lib\security\cacerts 在输入上述命令，如图3   
图3. 导入证书：   
  
  
   
**4>将证书应用到tomcat**   
打开tomcat目录的conf/server.xml文件，8443端处，并设置keystoreFile、keystorePass修改结果如下：

**Java代码  [收藏代码](javascript:void())**

1. <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
2. maxThreads="150" scheme="https" secure="true"
3. clientAuth="false" sslProtocol="TLS"
4. keystoreFile="D:/cas/keys/mycas.keystore"
5. keystorePass="生成KEY的密码"
6. />

参数说明：keystoreFile：在第一步创建的key存放位置 keystorePass：创建证书时的密码。   
打开https://localhost:8443/，可以测试，如图4：   
图4，测试证书   
  
   
点击“继续浏览此网站(不推荐)，现在进入Tomcat目录了吧，如果是那么你又向成功迈进了一步。   
  
**5.CAS Server端配置**   
解压缩cas-server-4.0.0 jar包 ，将modules文件夹下的cas-server-webapp-4.0.0.war拷贝到D:\apache-tomcat-7.0.54\webapps目录下，改名为cas.war（只是为了方便URL输入），并启动Tomcat， 会自动解压得到cas工程文件夹。   
  
好了，现在打开浏览器，输入https://localhost:8443/cas/login ，显示如图5：   
图5 .CAS默认登录页面   
  
  
   
CAS Server 4.0版本默认采用的username/password 在cas\WEB-INF下的deployerConfigContext.xml中是这样的：

**Java代码  [收藏代码](javascript:void())**

1. <bean id="primaryAuthenticationHandler"
2. **class**="org.jasig.cas.authentication.AcceptUsersAuthenticationHandler">
3. <property name="users">
4. <map>
5. <entry key="casuser" value="Mellon"/>
6. </map>
7. </property>
8. </bean>

输入username: casuser password:Mellon 登录成功，如图6：   
图6.CAS 默认登录成功   
  
  
   
你成功了吗？如果没有成功请再检查以上步骤！   
  
**6.CAS Server 用户认证定制**   
上面的初体验仅仅是简单的身份验证，实际应用中肯定是要读取数据库的数据，下面我们来进一步配置CAS服务器怎么读取数据库的信息进行身份验证，即cas-server-support-jdbc。   
  
首先，打开D:\apache-tomcat-7.0.54\webapps\cas\WEB-INF\deployerConfigContext.xml，   
注释掉默认的用户验证,添加JDBC认证，代码如下：

**Java代码  [收藏代码](javascript:void())**

1. <!--<bean id="primaryAuthenticationHandler"
2. **class**="org.jasig.cas.authentication.AcceptUsersAuthenticationHandler">
3. <property name="users">
4. <map>
5. <entry key="casuser" value="huiquan"/>
6. </map>
7. </property>
8. </bean> -->
10. <bean id="dataSource"
11. **class**="com.mchange.v2.c3p0.ComboPooledDataSource"
12. p:driverClass="com.mysql.jdbc.Driver"p:jdbcUrl="jdbc:mysql://192.168.0.10:3306/college?useUnicode=true&amp;characterEncoding=UTF-8&amp;zeroDateTimeBehavior=convertToNull"
13. p:user="root"
14. p:password="root" />
16. <!-- 密码加密方式-->
17. <bean id="passwordEncoder"
18. **class**="org.jasig.cas.authentication.handler.DefaultPasswordEncoder"
19. c:encodingAlgorithm="SHA1"
20. p:characterEncoding="UTF-8" />
22. <bean id="dbAuthHandler"
23. **class**="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler"
24. p:dataSource-ref="dataSource"
25. p:sql="select password from college.pub\_party where partyno=? and partytypecode='0000006' and data\_status='1'" />
26. <!-- p:passwordEncoder-ref="passwordEncoder" --><!-- 暂时不使用密码加密 -->

根据上述配置，我们需要在D:\apache-tomcat-7.0.54\webapps\cas\WEB-INF\lib中加入c3p0-0.9.1.2.jar包、mysql-connector-java-5.1.21.jar包和cas-server-support-jdbc-4.0.0.jar包。   
再修改D:\apache-tomcat-7.0.54\webapps\cas\WEB-INF\deployerConfigContext.xml中authentication manager的配置：

**Java代码  [收藏代码](javascript:void())**

1. <bean id="authenticationManager" **class**="org.jasig.cas.authentication.PolicyBasedAuthenticationManager">
2. <constructor-arg>
3. <map>
5. <entry key-ref="proxyAuthenticationHandler" value-ref="proxyPrincipalResolver" />
6. <!--<entry key-ref="primaryAuthenticationHandler" value-ref="primaryPrincipalResolver" /> -->
7. <entry key-ref="dbAuthHandler" value-ref="primaryPrincipalResolver"/>
8. </map>
9. </constructor-arg>
10. <property name="authenticationPolicy">
11. <bean **class**="org.jasig.cas.authentication.AnyAuthenticationPolicy" />
12. </property>
13. </bean>

再次运行Tomcat，访问https://localhost:8443/cas/login就能进行数据库验证了。   
  
**7.CAS Client 配置**   
本文中客户端工程部署在同一个Tomcat中，所以CAS Server和Client的域名都为localhost。   
在Client工程WEB-INF/lib下添加cas-client-core-3.2.1.jar包。   
  
修改web.xml如下：

**Java代码  [收藏代码](javascript:void())**

1. <!-- ======================== 单点登录/登出 ======================== -->
3. <!-- 该过滤器用于实现单点登出功能，可选配置。 -->
4. <filter>
5. <filter-name>CAS Single Sign Out Filter</filter-name>
6. <filter-**class**>org.jasig.cas.client.session.SingleSignOutFilter</filter-**class**>
7. </filter>
9. <!-- 该过滤器负责用户的认证工作，必须启用它 -->
10. <filter>
11. <filter-name>CAS Authentication Filter</filter-name>
12. <filter-**class**>org.jasig.cas.client.authentication.AuthenticationFilter</filter-**class**>
13. <init-param>
14. <param-name>casServerLoginUrl</param-name>
15. <param-value>https://localhost:8443/cas/login</param-value>
16. </init-param>
17. <init-param>
18. <param-name>serverName</param-name>
19. <param-value>http://localhost:8080</param-value>
20. </init-param>
21. </filter>
23. <!-- 该过滤器负责对Ticket的校验工作，必须启用它 -->
24. <filter>
25. <filter-name>CAS Validation Filter</filter-name>
26. <filter-**class**>org.jasig.cas.client.validation.Cas10TicketValidationFilter</filter-**class**>
27. <init-param>
28. <param-name>casServerUrlPrefix</param-name>
29. <param-value>https://localhost:8443/cas</param-value>
30. </init-param>
31. <init-param>
32. <param-name>serverName</param-name>
33. <param-value>http://localhost:8080</param-value>
34. </init-param>
35. <init-param>
36. <param-name>redirectAfterValidation</param-name>
37. <param-value>**true**</param-value>
38. </init-param>
39. </filter>
41. <!-- 该过滤器负责实现HttpServletRequest请求的包裹，
42. 比如允许开发者通过HttpServletRequest的getRemoteUser()方法获得SSO登录用户的登录名，可选配置。 -->
43. <filter>
44. <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
45. <filter-**class**>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-**class**>
46. </filter>
48. <!-- 该过滤器使得开发者可以通过org.jasig.cas.client.util.AssertionHolder来获取用户的登录名。
49. 比如AssertionHolder.getAssertion().getPrincipal().getName()。-->
50. <filter>
51. <filter-name>CAS Assertion Thread Local Filter</filter-name>
52. <filter-**class**>org.jasig.cas.client.util.AssertionThreadLocalFilter</filter-**class**>
53. </filter>
55. <filter-mapping>
56. <filter-name>CAS Single Sign Out Filter</filter-name>
57. <url-pattern>/\*</url-pattern>
58. </filter-mapping>
59. <filter-mapping>
60. <filter-name>CAS Authentication Filter</filter-name>
61. <url-pattern>/\*</url-pattern>
62. </filter-mapping>
63. <filter-mapping>
64. <filter-name>CAS Validation Filter</filter-name>
65. <url-pattern>/\*</url-pattern>
66. </filter-mapping>
67. <filter-mapping>
68. <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
69. <url-pattern>/\*</url-pattern>
70. </filter-mapping>
71. <filter-mapping>
72. <filter-name>CAS Assertion Thread Local Filter</filter-name>
73. <url-pattern>/\*</url-pattern>
74. </filter-mapping>
76. <listener>
77. <listener-**class**>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-**class**>
78. </listener>
79. <!-- ======================== 单点登录/登出结束 ======================== -->

现在运行Client工程，首次访问任一页面就会跳转到https://localhost:8443/cas/login进行认证。同时，把你的退出链接设置为：https://sso.wsria.com/cas/logout 即可实现单点推出。   
  
**8、美化CAS服务器界面**   
CAS服务端(cas-server)的界面只能在测试的时候用一下，真正系统上线肯定需要定制开发自己的页面，就像网易和CSDN的统一认证平台一样，所有子系统的认证都通过此平台来转接，大家可以根据他们的页面自己定制出适合所属应用或者公司的界面；简单介绍一下吧，复制 cas\WEB-INF\view\jsp\default\ui的一些JSP文件，每一个文件的用途文件名已经区分了，自己修改了替换一下就可以了。 例如：   
登录界面：casLoginView.jsp   
登录成功：casGenericSuccess.jsp   
登出界面：casLogoutView.jsp   
  
**9、更新Client端页面跳转配置**   
完成前面的步骤，并不能得到我们理想的效果。例如，我们的目标是：   
Client端index页面 -> CAS Server端login页面 -> 认证成功 -> Client端index页面，并显示欢迎信息。   
但由于Client端有自己的登录验证拦截机制，效果却是这样的：   
Client端index页面 -> CAS Server端login页面->认证成功 -> Client端login页面。   
所以，我需要修改Client端的认证拦截/跳转页面。   
前面的步骤以后，我们在Client端的HttpServletRequest中已经可以通过getRemoteUser()方法得到用户名了。   
String username = request.getRemoteUser();   
再据此修正控制器login页面跳转或拦截器就行了。   
  
**结束语**   
本文只是简单的介绍了CAS的搭建和JDBC认证定制。CAS还提供了多种认证方式：如LADP、legacy、trusted、x.509等等，更多深入的扩展则需要重写AuthenticationHandler的实现等   
在此列出一些比较好的相关博文以供参考：