<http://www.cnblogs.com/notDog/p/5264666.html>

<http://blog.csdn.net/zmx729618/article/details/51496937>

本教程只是对cas的最简单的使用。

# 介绍

## SSO和CAS

SSO：单点登录（Single Sign On），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。

CAS：耶鲁大学开发的单点登录（Single Sign On）系统称为CAS（Central Authentication Server），他是一个开源的、相对比较简单易用的SSO解决方案。

SSL(Secure Sockets Layer 安全套接层),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。TLS与SSL在传输层对网络连接进行加密。

## 使用场合

目前公司有相互关联的几个独立项目，需要使用单点登录进行整合，实现一站登录，全站皆可以访问。选用CAS开源项目。

## 项目组成

cas-server4.0.0 项目从GitHub中clone到本地后，发现项目结构如下。

|  |  |
| --- | --- |
| 项目名称 | 功能描述 |
| cas-management-webapp | 这是一个管理service的web项目，没有管理用户的功能 |
| cas-server-core | cas核心/基础子项目 |
| cas-server-core-api | core项目的辅助项目，为core提供的功能提供对外服务的api |
| cas-server-documentation | 文档 |
| cas-server-extension-clearpass | 在用户登录时，cas server端保存用户的密码，然后当[cas](http://blog.csdn.net/matthewei6/article/details/50704677" \t "http://blog.csdn.net/matthewei6/article/details/_blank)client  需要时候通过适当的方式从server端获取。 |
| cas-server-integration-ehcache | ticket存储在ehcache中，EhCacheTicketRegistry |
| cas-server-integration-hazelcast | Hazelcast: Java分布式内存网格框架 |
| cas-server-integration-memcached | ticket存储在memcached中，MemCacheTicketRegistry |
| cas-server-integration-mongo | 注册的service存储在mongoDB中，MongoServiceRegistryDao |
| cas-server-support-jdbc | 用户信息存储在数据库中；service也存储在数据库中，   JpaServiceRegistryDaoImpl， 可以和cas-management-webapp配置 |
| cas-server-support-ldap | 用户信息存储在ldap协议的存储中， |
| cas-server-support-oauth | 支持oauth协议授权码模式 |
| cas-server-support-openid | 支持openid协议 |
| cas-server-support-radius | RADIUS 是一种用于在需要认证其链接的网络访问服务器（NAS）和共享认证服务器之间进行认证、授权和记帐信息的文档协议。 |
| cas-server-support-[rest](http://blog.csdn.net/matthewei6/article/details/50709212" \t "http://blog.csdn.net/matthewei6/article/details/_blank) | 可以支持c/s模式，如android app |
| cas-server-support-saml | saml协议相关 |
| cas-server-support-spnego | CAS采用Spnego认证方式通过Kerberos与AD域整合 |
| cas-server-webapp | 主项目 |
| cas-server-webapp-support | 主项目webapp的辅助项目 |
| cas-management-webapp | 这是一个管理service的web项目，没有管理用户的功能 |

LDAP：轻量目录访问协议，英文全称是Lightweight Directory Access Protocol，一般都简称为LDAP。它是基于X.500标准的，但是简单多了并且可以根据需要定制。与X.500不同，LDAP支持TCP/IP，这对访问Internet是必须的。

JAAS：Java Authentication Authorization Service，Java验证和授权API，提供了灵活和可伸缩的机制来保证客户端或服务器端的Java程序。

# 简单入门

## 开发环境准备

生成SSL证书，配置一个tomcat，修改hosts配置增加“127.0.0.1 [www.zhangxue.com”](http://www.zhangxue.com\”)

CAS-Server下载地址：https://www.apereo.org/projects/cas/download-cas

CAS-Client下载地址：http://developer.jasig.org/cas-clients/

CAS官方教程:

https://wiki.jasig.org/display/CASUM/CAS+on+Windows+Quick+Setup+Guide

版本：

CAS Server版本：cas-server-4.0.0

CAS Client版本：cas-client-3.3.3

注意这两个都是源码。解压cas-server-4.0.0-release.zip

可以看到压缩包中有许多项目，我们只需要选用两个项目

cas-server-core：主要业务功能实现，最终打包成jar被cas-server-webapp所使用

cas-server-webapp：为CAS-server主项目，集中了各种配置文件

其他都是一些扩展项目，特定情况下可以选用。

modules下存放了打的好各种jar、wa包，如果你想要直接使用，无需修改源码，可以直接把这两个项目部署到Tomcat下面，直接启动。

## 创建客户端

创建两个项目：cas\_client1和cas\_client2。

并将cas-client/modules目录下所有的jar包导入到项目中。

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>20171018\_cas\_client1</display-name>

<!--========================单点登录开始======================== -->

<listener>

<listener-class>

org.jasig.cas.client.session.SingleSignOutHttpSessionListener

</listener-class>

</listener>

<!--用于单点退出，该过滤器用于实现单点登出功能，可选配置 -->

<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>

<!--该过滤器用于实现单点登入功能，可选配置。 -->

<filter>

<filter-name>CAS Filter</filter-name>

<filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>

<init-param>

<param-name>casServerLoginUrl</param-name>

<param-value>http://www.zhangxue.com:8080/cas-server-webapp-4.0.0/login</param-value>

</init-param>

<init-param>

<param-name>serverName</param-name>

<param-value>http://www.zhangxue.com:8018</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CAS Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!--该过滤器负责对Ticket的校验工作，必须启用它 -->

<filter>

<filter-name>CAS Validation Filter</filter-name>

<filter-class>

org.jasig.cas.client.validation.Cas20ProxyReceivingTicketValidationFilter

</filter-class>

<init-param>

<param-name>casServerUrlPrefix</param-name>

<param-value>http://www.zhangxue.com:8080/cas-server-webapp-4.0.0</param-value>

</init-param>

<init-param>

<param-name>serverName</param-name>

<param-value>http://www.zhangxue.com:8018</param-value>

</init-param>

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CAS Validation Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 该过滤器负责实现HttpServletRequest请求的包裹， 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。 -->

<filter>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<filter-class>

org.jasig.cas.client.util.HttpServletRequestWrapperFilter

</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 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。 比如AssertionHolder.getAssertion().getPrincipal().getName()。 -->

<filter>

<filter-name>CAS Assertion Thread Local Filter</filter-name>

<filter-class>org.jasig.cas.client.util.AssertionThreadLocalFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS Assertion Thread Local Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- ========================单点登录结束 ======================== -->

</web-app>

特别需要注意的是红色背景的部分。写的是casServerLoginUrl写的是登录的cas登录用到url。casServerUrlPrefix写的是cas服务的url。而serverName写的是需要单点登录的服务url。这个位置为了避免出现问题，推荐使用域名：端口，不要添加项目名称，当然项目的contextPath最好修改为/。方便访问。

cas\_client2项目的端口为8028。

## 运行项目

cas\_client1项目的端口为8018。，cas\_client2项目的端口为8028。

分别运行在不同的tomcat中。通过浏览器访问

http://www.zhangxue.com:8018或http://www.zhangxue.com:8028，都会跳转到下面的页面上。输入密码密码即可登录成功，登陆后可正常访问client1和client2项目。

（4.0默认账号密码在cas-server-webapp-4.0.0\WEB-INF\deployerConfigContext.xml中）。



# 配置http

CAS 默认认证方式使用的是HTTPS协议，一般对安全性不高的话建议取消改成HTTP方式。因为，开启的话会经常提示证书过期、需要用户确认等，客户觉得CAS证书的提示太烦了，当前有需要的可以开启HTTPS。所以直接修改CAS-SERVER服务端的配置文件即可。

取消HTTPS协议，需要修改3个配置文件。

## deployerConfigContext.xml

<!--Required for proxy ticket mechanism. -->

<beanid="proxyAuthenticationHandler"

class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"

p:httpClient-ref="httpClient"/>

增加参数p:requireSecure="false"，是否需要安全验证，即HTTPS，false为不采用。修改后为：

<beanid="proxyAuthenticationHandler"

class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"

p:httpClient-ref="httpClient"

p:requireSecure="false" />

## spring-configuration\ticketGrantingTicketCookieGenerator.xml

<beanid="ticketGrantingTicketCookieGenerator"class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"

p:cookieSecure="true"

p:cookieMaxAge="-1"

p:cookieName="CASTGC"

p:cookiePath="/cas" />

修改 p:cookieSecure="true" 为p:cookieSecure="false"，即不开启https验证。

## spring-configuration\ warnCookieGenerator.xml

<beanid="warnCookieGenerator"class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"

p:cookieSecure="true"

p:cookieMaxAge="-1"

p:cookieName="CASPRIVACY"

p:cookiePath="/cas" />

修改 p:cookieSecure="true" 为p:cookieSecure="false"，即不开启https验证。

## 默认账号密码（可选）

CAS4的默认用户配置在 deployerConfigContext.xml配置文件中，用户名密码为casuser/Mellon。

## 补充说明

单点登录（Single Sign On , 简称 SSO ）是目前比较流行的服务于企业业务整合的解决方案之一， SSO 使得在多个应用系统中，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。

如果需要应用到项目中，还需要许多个性化的订制，比如登陆页的美化、通过数据库认证、服务端与客户端用户信息的交互等。

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：

l 登录界面：casLoginView.jsp

l 登录成功：casGenericSuccess.jsp

l 登出界面：casLogoutView.

# 自定义用户验证

CAS自带的用户验证逻辑太过简单，如何像正常网站一样，通过验证DB中的用户数据，来验证用户以及密码的合法性呢。

## 方案1：CAS默认的JDBC扩展方案

CAS自带了两种简单的通过JDBC方式验证用户的处理器。

1.QueryDatabaseAuthenticationHandler

2.SearchModeSearchDatabaseAuthenticationHandler

这两个处理类位于cas-server-support-jdbc这个扩展工程下。

### 第一步：改写用户验证处理器

打开deployerConfigContext.xml，找到这个处理器

<bean class="org.jasig.cas.authentication.handler.support.SimpleTestUsernamePasswordAuthenticationHandler" />

<bean id="SearchModeSearchDatabaseAuthenticationHandler"

class="org.jasig.cas.adaptors.jdbc.SearchModeSearchDatabaseAuthenticationHandler"

abstract="false" singleton="true" lazy-init="default" autowire="default" dependency-check="default">

<property name="tableUsers">

<value>users</value>

</property>

<property name="fieldUser">

<value>**user\_id**</value>

</property>

<property name="fieldPassword">

<value>**password**</value>

</property>

<property name="dataSource" ref="dataSource" />

</bean>

或者

<bean class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler">

<property name="dataSource" ref="dataSource" />

<property name="sql" value="select password from **some\_table** where lower(**username**) = lower(**?**)" /></bean>

把对应的用户表名、用户名、密码替换成为 你实际数据库中的 用户表名、用户名、密码对应的字段名称。

### 第二步：加载驱动（oracle、mysql任选一种）

依然在这个文件中deployerConfigContext.xml,在最后一个</bean>标签之前，加上下面配置即可（oracle、mysql任选其一）。

<!-- Oracle connector --><bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName">

<value>oracle.jdbc.driver.OracleDriver</value>

</property>

<property name="url">

<value>jdbc:oracle:thin:@database-server-name:1521:SID</value>

</property>

<property name="username">

<value>admusr</value>

</property>

<property name="password">

<value>admpwd</value>

</property></bean>

<!-- MySQL connector --><bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">

<property name="driverClassName">

<value>com.mysql.jdbc.Driver</value>

</property>

<property name="url">

<value>jdbc:mysql://database-server-name:3306/db-name</value>

</property>

<property name="username">

<value>admusr</value>

</property>

<property name="password">

<value>admpwd</value>

</property></bean>

### 第三步：导入必要的jar包

cas-server-webapp的pom.xml。

<dependency>

<groupId>org.jasig.cas</groupId>

<artifactId>cas-server-support-jdbc</artifactId>

<version>${project.version}</version>

</dependency>

<dependency>

<groupId>commons-dbcp</groupId>

<artifactId>commons-dbcp</artifactId>

<version>1.2.1</version>

<scope>runtime</scope>

</dependency>

<!-- oracle 、 mysql 任选其一 start-->

<dependency>

<groupId>oracle</groupId>

<artifactId>ojdbc14</artifactId>

<version>10.2.0.4</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.5</version>

<scope>provided</scope>

</dependency>

重新启动项目，在登录页验证即可。

## 套用原有项目的用户登录逻辑

背景：

1.因为是要整合多个项目的登录功能，换句话说，就是原有的项目其实分别都有已经实现的用户登录逻辑页面、登录逻辑，并且是经过了大量测试，程序逻辑比较健壮。

2.原有的项目并不是通过JDBC方式来访问数据库的，为了保持代码风格统一，最好与其他项目保持一致，或者直接套用原有的验证逻辑。（比如：我们项目使用的是Mybatis方式）

3.CAS扩展的JDBC验证方式还是过于简单，对于复杂的验证逻辑不便于扩展。

### 第一步：自定义一个用户验证处理器

可以在原有的cas-server-support-jdbc工程下，进行开发，也可以自己创建一个项目，进行进行开发。需要注意的是，必须继承AbstractUsernamePasswordAuthenticationHandler

public final class **MyUserAuthenticationHandler** extends **AbstractUsernamePasswordAuthenticationHandler** {

/\*\*

\* 这个API是原有项目验证用户相关处理的API，直接导入相关jar包就可以使用

\*/

@Autowired

private UserApi userApi;

@Override

protected final boolean authenticateUsernamePasswordInternal

( final UsernamePasswordCredentials credentials) throws **MyAuthenticationException** {

String username = getPrincipalNameTransformer().transform(credentials.getUsername());

String password = credentials.getPassword();

try {

userApi.login(username, password);

}

catch (Exception e) {

throw new **MyAuthenticationException**(e);

}

return true;

}

}

### 第二步：自定义异常处理器

因为CAS原有的异常处理逻辑是，只能通过errorcode就去messages\_zh\_CN.properties寻找已经中定义好的异常消息，如果找不到直接返回errorcode到页面，并不支持直接抛出文字形式的异常消息。且，还需要为每种异常都定义专属的异常类，例如：

BadCredentialsAuthenticationException.java

BadPasswordAuthenticationException.java

BadUsernameOrPasswordAuthenticationException.java

而本异常类可以直接接受Api抛出的所有异常，或者也可以自定义文字形式的异常消息。如：new Exception("这是异常消息")。同样也支持原有的根据errorCode读取异常消息的方式。

public final class **MyAuthenticationException** extends AuthenticationException {

/\*\* Serializable ID. \*/

private static final long serialVersionUID = 3906648604830611762L;

/\*\*

\* 本构造方法可以返回任何自定义的异常消息到前台

\* <p>例子:TongXiangAuthenticationException(new Exception("自定义的异常消息"))</p>

\*/

public MyAuthenticationException(final Throwable throwable) {

super(throwable.getMessage());

}

/\*\*

\* 本构造方法只能返回在messages\_zh\_CN.properties中已经定义好的异常消息

\*/

public MyAuthenticationException(final String code){

super(code);

}

# 易出现的问题

## 客户端启动报错，报错信息如下

严重: Exception starting filter CAS Single Sign Out Filter

java.lang.IllegalArgumentException: casServerUrlPrefix cannot be null.

at org.jasig.cas.client.util.CommonUtils.assertNotNull(CommonUtils.java:80)

at org.jasig.cas.client.session.SingleSignOutHandler.init(SingleSignOutHandler.java:142)

at org.jasig.cas.client.session.SingleSignOutFilter.init(SingleSignOutFilter.java:58)

at org.apache.catalina.core.ApplicationFilterConfig.initFilter(ApplicationFilterConfig.java:279)

at org.apache.catalina.core.ApplicationFilterConfig.getFilter(ApplicationFilterConfig.java:260)

at org.apache.catalina.core.ApplicationFilterConfig.<init>(ApplicationFilterConfig.java:105)

at org.apache.catalina.core.StandardContext.filterStart(StandardContext.java:4830)

at org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5510)

at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:150)

at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1575)

at org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1565)

at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.java:303)

at java.util.concurrent.FutureTask.run(FutureTask.java:138)

at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)

at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)

at java.lang.Thread.run(Thread.java:662)

再确定web.xml文件casServerUrlPrefix配置无误之后，很可能的原因是cas client core版本不对。

## 登录后500、403等异常

当在cas server登录成功后，准备往客户端跳转，突然发现浏览器空白，且浏览器输入地址栏出现类似http://localhost:8008/j1-center/?ticket=ST-1-51PDs4kgv5XpOW7Vab91-cas01.

example.org路径的时候，很有可能是标红的路径配置错误。

当客户端页面出现：org.jasig.cas.client.validation.TicketValidationException: No principal was found in the response from the CAS server.页面报500的时候，很有可能出现的原因还是在cas client web.xml中cas server路径配置错误。

这些问题可能都是web.xml中的问题，查询这里的[红色标记的部分](#_创建客户端)