品优购电商系统开发

第 15 章

单点登录解决方案-CAS

传智播客.黑马程序员

课程目标

目标 1：搭建单点登录服务端，开发单点登录客户端

目标 2：实现 CAS 认证数据源设置

目标 3：更换 CAS 登录页面

目标 4：掌握 CAS 与 SpringSecurity 集成

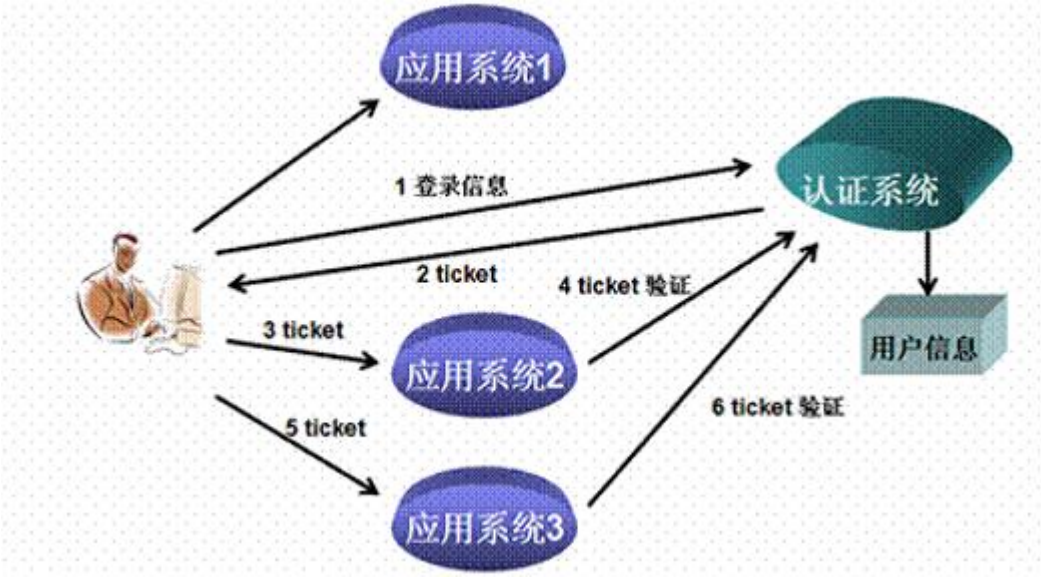
目标 5：完成用户中心单点登录功能

1.开源单点登录系统 CAS 入门

1.1 什么是单点登录

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。

我们目前的系统存在诸多子系统，而这些子系统是分别部署在不同的服务器中，那么使用传统方式的 session 是无法解决的，我们需要使用相关的单点登录技术来解决。



1.2 什么是 CAS

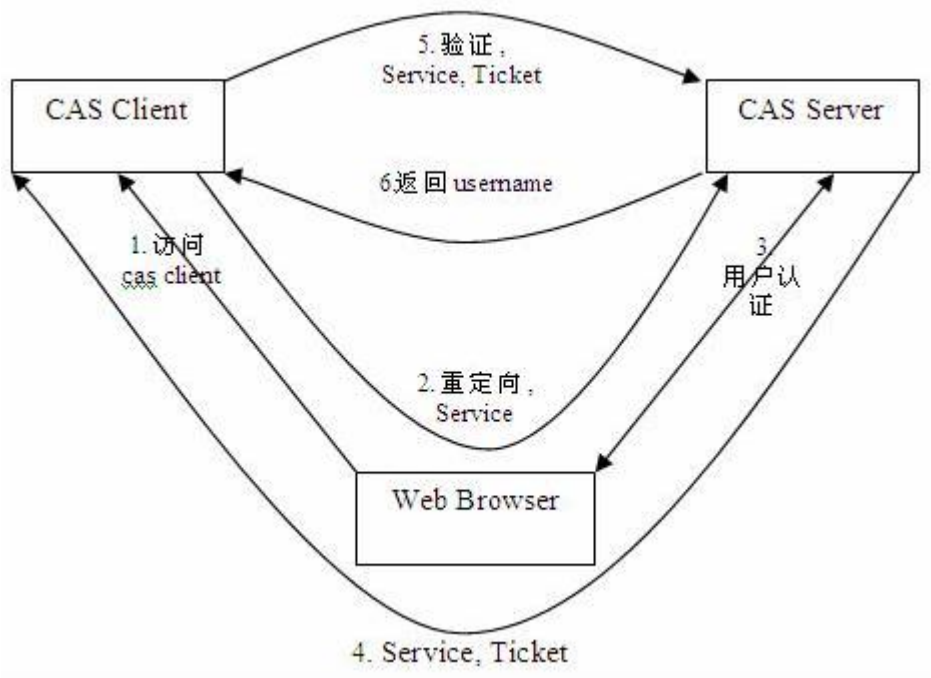
CAS 是 Yale 大学发起的一个开源项目，旨在为 Web 应用系统提供一种可靠的单点登录方法，CAS 在 2004 年 12 月正式成为 JA-SIG 的一个项目。CAS 具有以下特点：

【1】开源的企业级单点登录解决方案。

【2】CAS Server 为需要独立部署的 Web 应用。

【3】CAS Client 支持非常多的客户端(这里指单点登录系统中的各个 Web 应用)，包括 Java, .Net, PHP, Perl, Apache, uPortal, Ruby 等。

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。下图是 CAS 最基本的协议过程：



SSO 单点登录访问流程主要有以下步骤：

1. 访问服务：SSO 客户端发送请求访问应用系统提供的服务资源。

2. 定向认证：SSO 客户端会重定向用户请求到 SSO 服务器。

3. 用户认证：用户身份认证。

4. 发放票据：SSO 服务器会产生一个随机的 Service Ticket。

5. 验证票据：SSO 服务器验证票据 Service Ticket 的合法性，验证通过后，允许客户端访问服务。

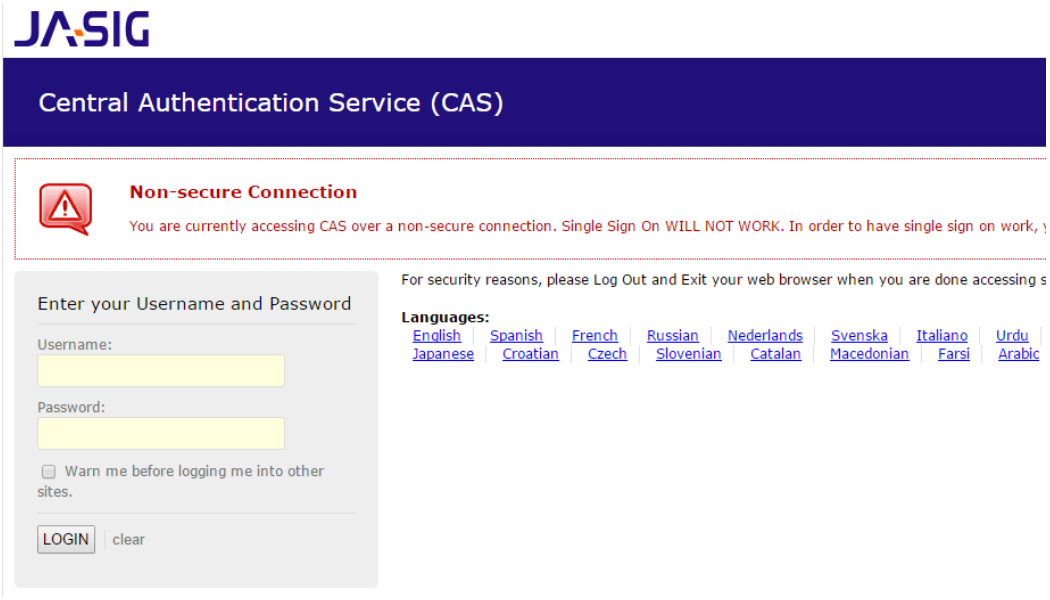
6. 传输用户信息：SSO 服务器验证票据通过后，传输用户认证结果信息给客户端。

1.3 CAS 服务端部署

Cas 服务端其实就是一个 war 包。

在资源\cas\source\cas-server-4.0.0-release\cas-server-4.0.0\modules 目录下

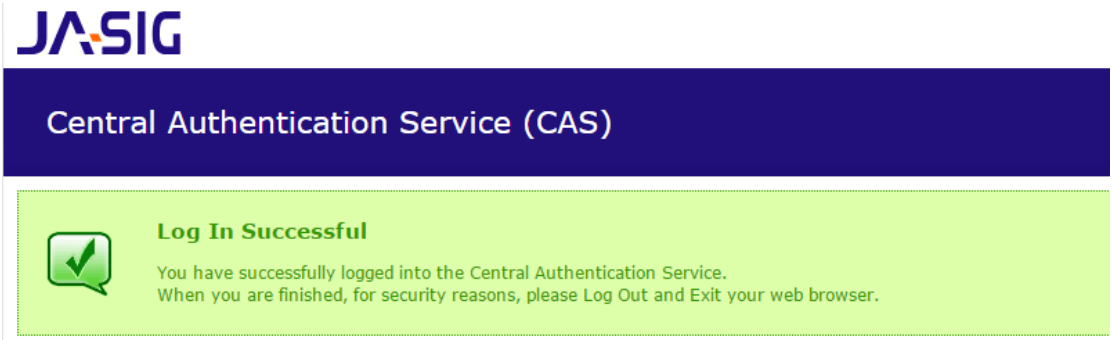
cas-server-webapp-4.0.0.war 将其改名为 cas.war 放入 tomcat 目录下的 webapps 下。启动 tomcat 自动解压 war 包。浏览器输入 http://localhost:8080/cas/login ，可看到登录页面



不要嫌弃这个页面丑，我们后期可以再提升它的颜值。暂时把注意力放在功能实现上。

这里有个固定的用户名和密码 casuser /Mellon

登录成功后会跳到登录成功的提示页面



1.4 CAS 服务端配置

1.4.1 端口修改

如果我们不希望用 8080 端口访问 CAS，可以修改端口

（1）修改 TOMCAT 的端口

打开 tomcat 目录 conf\server.xml 找到下面的配置

```
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

将端口 8080，改为 9100

（2）修改 CAS 配置文件

修改 cas 的 WEB-INF/cas.properties

```
server.name=http://localhost:9100
```

1.4.2 去除 https 认证

CAS 默认使用的是 HTTPS 协议，如果使用 HTTPS 协议需要 SSL 安全证书（需向特定的机构申请和购买）。如果对安全要求不高或是在开发测试阶段，可使用 HTTP 协议。我们这里讲解通过修改配置，让 CAS 使用 HTTP 协议。

（1）修改 cas 的 WEB-INF/deployerConfigContext.xml

找到下面的配置

```
<bean
class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"
p:httpClient-ref="httpClient"/>
```

这里需要增加参数 p:requireSecure="false"，requireSecure 属性意思为是否需要安全验证，即 HTTPS，false 为不采用

（2）修改 cas 的/WEB-INF/spring-configuration/ticketGrantingTicketCookieGenerator.xml

找到下面配置

```
<bean id="ticketGrantingTicketCookieGenerator"
class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"
    p:cookieSecure="true"
    p:cookieMaxAge="-1"
    p:cookieName="CASTGC"
    p:cookiePath="/cas" />
```

参数 p:cookieSecure="true"，同理为 HTTPS 验证相关，TRUE 为采用 HTTPS 验证，FALSE 为不采用 https 验证。

参数 p:cookieMaxAge="-1"，是 COOKIE 的最大生命周期，-1 为无生命周期，即只在当前打开的窗口有效，关闭或重新打开其它窗口，仍会要求验证。可以根据需要修改为大于 0 的数字，比如 3600 等，意思是在 3600 秒内，打开任意窗口，都不需要验证。

我们这里将 cookieSecure 改为 false， cookieMaxAge 改为 3600

（3）修改 cas 的 WEB-INF/spring-configuration/warnCookieGenerator.xml

找到下面配置

```
<bean id="warnCookieGenerator"
class="org.jasig.cas.web.support.CookieRetrievingCookieGenerator"
p:cookieSecure="true"
p:cookieMaxAge="-1"
p:cookieName="CASPRIVACY"
p:cookiePath="/cas" />
```

我们这里将 cookieSecure 改为 false， cookieMaxAge 改为 3600

1.5 CAS 客户端入门小 Demo

1.5.1 客户端工程 1 搭建

（1）搭建工程引入依赖

创建 Maven 工程 （war）casclient_demo1 引入 cas 客户端依赖并制定 tomcat 运行端口为 9001

```
    <dependencies>
        <!-- cas -->
        <dependency>
            <groupId>org.jasig.cas.client</groupId>
            <artifactId>cas-client-core</artifactId>
            <version>3.3.3</version>
        </dependency>
        <dependency>
             <groupId>javax.servlet</groupId>
             <artifactId>servlet-api</artifactId>
             <version>2.5</version>
             <scope>provided</scope>
        </dependency>
    </dependencies>
<build>
    <plugins>
        <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-compiler-plugin</artifactId>
             <version>2.3.2</version>
             <configuration>
                 <source>1.7</source>
                 <target>1.7</target>
```

```
            </configuration>
        </plugin>
        <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>9001</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
         </plugin>
      </plugins>
   </build>
```

（2）添加 web.xml

```
<?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"
    version="2.5">
   <!-- 用于单点退出，该过滤器用于实现单点登出功能，可选配置 -->
   <listener>
```

```
<listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</list
ener-class>
    </listener>
    <!-- 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>
    <!-- 该过滤器负责用户的认证工作，必须启用它 -->
    <filter>
        <filter-name>CASFilter</filter-name>
<filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-clas
s>
        <init-param>
            <param-name>casServerLoginUrl</param-name>
            <param-value>http://localhost:9100/cas/login</param-value>
            <!--这里的 server 是服务端的 IP -->
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
```

```
            <param-value>http://localhost:9001</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CASFilter</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-c
lass>
        <init-param>
            <param-name>casServerUrlPrefix</param-name>
            <param-value>http://localhost:9100/cas</param-value>
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
            <param-value>http://localhost:9001</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>
</web-app>
```

（3）编写 index.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"

    pageEncoding="utf-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8">

<title>一品优购</title>

</head>

<body>

欢迎来到一品优购

<%=request.getRemoteUser()%>

</body>

</html>
```

request.getRemoteUser()为获取远程登录名

1.5.2 客户端工程 2 搭建

（1）创建 Maven 工程 （war）casclient_demo2 引入 cas 客户端依赖并制定 tomcat 运行端口为 9002

（2）创建 web.xml，参照 casclient_demo1 ,将 serverName 的值改为 http://localhost:9002，一共两处

（3）创建 index.jsp ,内容显示“欢迎来到二品优购”

1.5.3 单点登录测试

（1）启动 cas 部署的 tomcat

（2）启动客户端工程 1 和客户端工程 2

（3）地址栏输入 http://localhost:9001/ 和 http://localhost:9002/ ，地址均会跳转到 CAS 登录页

（4）输入用户名和密码后，页面跳转回 9002 ，再次访问 9001 也可以打开主页面。

1.5.4 单点退出登录

地址栏输入 http://localhost:9100/cas/logout

即可看到退出后的提示页面

我们可以将这个链接添加到 index.jsp 中

```
<a href="http://localhost:9100/cas/logout">退出登录</a>
```

但我们更希望退出登录后，能自动跳转到某个页面，那如何处理呢？

修改 cas 系统的配置文件 cas-servlet.xml

```
<bean id="logoutAction" class="org.jasig.cas.web.flow.LogoutAction"
        p:servicesManager-ref="servicesManager"
        p:followServiceRedirects="${cas.logout.followServiceRedirects:true}"/>
```

改为 true 后，可以在退出时跳转页面到目标页面，修改 index.jsp 的退出链接

```
<a href="http://localhost:9100/cas/logout?service=http://www.baidu.com">退出登录</a>
```

2.CAS 服务端数据源设置

2.1 需求分析

我们现在让用户名密码从我们的品优购的 user 表里做验证

2.2 配置数据源

（1）修改 cas 服务端中 web-inf 下 deployerConfigContext.xml ，添加如下配置

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
                    p:driverClass="com.mysql.jdbc.Driver"
                    p:jdbcUrl="jdbc:mysql://127.0.0.1:3306/pinyougoudb?characterEncoding=utf8"
                    p:user="root"
                    p:password="123456" />
<bean id="passwordEncoder"
            class="org.jasig.cas.authentication.handler.DefaultPasswordEncoder"
            c:encodingAlgorithm="MD5"
            p:characterEncoding="UTF-8" />
<bean id="dbAuthHandler"
                class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler"
                p:dataSource-ref="dataSource"
                p:sql="select password from tb_user where username = ?"
                p:passwordEncoder-ref="passwordEncoder"/>
```

然后在配置文件开始部分找到如下配置

```
<bean id="authenticationManager" class="org.jasig.cas.authentication.PolicyBasedAuthenticationManager">
        <constructor-arg>
```

```
            <map>
                <entry key-ref="proxyAuthenticationHandler" value-ref="proxyPrincipalResolver" />
                <entry key-ref="primaryAuthenticationHandler" value-ref="primaryPrincipalResolver" />
            </map>
        </constructor-arg>
        <property name="authenticationPolicy">
            <bean class="org.jasig.cas.authentication.AnyAuthenticationPolicy" />
        </property>
</bean>
```

其中

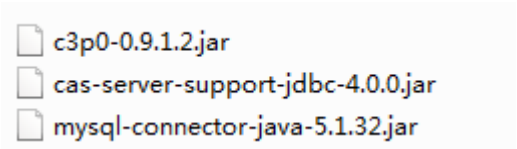
```
<entry key-ref="primaryAuthenticationHandler" value-ref="primaryPrincipalResolver" />
```

一句是使用固定的用户名和密码，我们在下面可以看到这两个 bean ,如果我们使用数据库认证用户名和密码，需要将这句注释掉。

添加下面这一句配置

```
<entry key-ref="dbAuthHandler" value-ref="primaryPrincipalResolver"/>
```

（2）将以下三个 jar 包放入 webapps\cas\WEB-INF\lib 下



（这三个 jar 包在资源\cas\jar 目录下）

用数据库中的用户名和密码进行测试

3.CAS 服务端界面改造

3.1 需求分析

我们现在动手将 CAS 默认的登录页更改为自己的品优购登陆页

3.2 改头换面

3.2.1 拷贝资源

（1）将品优购的登陆页 login.html 拷贝到 cas 系统下 WEB-INF\view\jsp\default\ui 目录下

（2）将 css js 等文件夹拷贝到 cas 目录下

（3） 将原来的 casLoginView.jsp 改名（可以为之后的修改操作做参照），将 login.html 改名为 casLoginView.jsp

3.2.2 修改页面

编辑 casLoginView.jsp 内容

（1）添加指令

```
<%@ page pageEncoding="UTF-8" %>

<%@ page contentType="text/html; charset=UTF-8" %>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
```

（2）修改 form 标签

```
<form:form method="post" id="fm1" commandName="${commandName}" htmlEscape="true"
class="sui-form">

.....
```

```
</form:form>
```

（3）修改用户名框

```
<form:input id="username" tabindex="1"
        accesskey="${userNameAccessKey}" path="username" autocomplete="off" htmlEscape="true"
        placeholder="邮箱/用户名/手机号" class="span2 input-xfat" />
```

（4）修改密码框

```
 <form:password    id="password" tabindex="2" path="password"
          accesskey="${passwordAccessKey}" htmlEscape="true" autocomplete="off"
           placeholder="请输入密码" class="span2 input-xfat"      />
```

（5）修改登陆按钮

```
<input type="hidden" name="lt" value="${loginTicket}" />
<input type="hidden" name="execution" value="${flowExecutionKey}" />
<input type="hidden" name="_eventId" value="submit" />
<input class="sui-btn btn-block btn-xlarge btn-danger" accesskey="l" value="登陆" type="submit" />
```

修改后效果如下：

3.3 错误提示

在表单内加入错误提示框

```
<form:errors path="*" id="msg" cssClass="errors" element="div" htmlEscape="false" />
```

测试：输入错误的用户名和密码，提示是英文。这个提示信息是在 WEB-INF\classes 目录下的 messages.properties 文件中

```
authenticationFailure.AccountNotFoundException=Invalid credentials.

authenticationFailure.FailedLoginException=Invalid credentials.
```

设置国际化为 zn_CN ,修改 cas-servlet.xml

```
<bean id="localeResolver" class="org.springframework.web.servlet.i18n.CookieLocaleResolver" p:defaultLocale="zh_CN" />
```

我们需要将此信息拷贝到 messages_zh_CN.properties 下，并改为中文提示（转码）

```
authenticationFailure.AccountNotFoundException=\u7528\u6237\u4E0D\u5B58\u5728.

authenticationFailure.FailedLoginException=\u5BC6\u7801\u9519\u8BEF.
```

第一个是用户名不存在时的错误提示

第二个是密码错误的提示

4. CAS 客户端与 SpringSecurity 集成

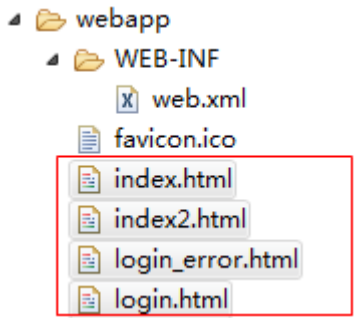
4.1 Spring Security 测试工程搭建

（1）建立 Maven 项目 casclient_demo3 ，引入 spring 依赖和 spring secrity 相关依赖 ，tomcat 端口设置为 9003

（2）建立 web.xml ,添加过滤器等配置

（3）创建配置文件 spring-security.xml

（4）添加 html 页面



以上步骤参照我们第 4 章的 spring-security-demo

4.2 Spring Security 与 CAS 集成

（1）引入依赖

```
<dependency>

        <groupId>org.springframework.security</groupId>

        <artifactId>spring-security-cas</artifactId>

        <version>4.1.0.RELEASE</version>

</dependency>

<dependency>

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

        <artifactId>cas-client-core</artifactId>

        <version>3.3.3</version>

        <exclusions>

            <exclusion>

                <groupId>org.slf4j</groupId>

                <artifactId>log4j-over-slf4j</artifactId>

            </exclusion>

        </exclusions>

</dependency>
```

（2）修改 spring-security.xml

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

<beans:beans xmlns="http://www.springframework.org/schema/security"

    xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd

                        http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">



    <!--   entry-point-ref  入口点引用 -->

    <http use-expressions="false" entry-point-ref="casProcessingFilterEntryPoint">

        <intercept-url pattern="/**" access="ROLE_USER"/>

        <csrf disabled="true"/>

        <!-- custom-filter 为过滤器， position 表示将过滤器放在指定的位置上，before 表示放
在指定位置之前 ，after 表示放在指定的位置之后 -->

        <custom-filter ref="casAuthenticationFilter"  position="CAS_FILTER" />

        <custom-filter ref="requestSingleLogoutFilter" before="LOGOUT_FILTER"/>

        <custom-filter ref="singleLogoutFilter" before="CAS_FILTER"/>

    </http>



    <!-- CAS 入口点 开始 -->

    <beans:bean id="casProcessingFilterEntryPoint"
class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">

        <!-- 单点登录服务器登录 URL -->
```

```
        <beans:property name="loginUrl" value="http://localhost:9100/cas/login"/>
        <beans:property name="serviceProperties" ref="serviceProperties"/>
    </beans:bean>
    <beans:bean id="serviceProperties"
class="org.springframework.security.cas.ServiceProperties">
        <!--service 配置自身工程的根地址+/login/cas   -->
        <beans:property name="service" value="http://localhost:9003/login/cas"/>
    </beans:bean>
    <!-- CAS 入口点 结束 -->


    <!-- 认证过滤器 开始 -->
    <beans:bean id="casAuthenticationFilter"
class="org.springframework.security.cas.web.CasAuthenticationFilter">
        <beans:property name="authenticationManager" ref="authenticationManager"/>
    </beans:bean>
         <!-- 认证管理器 -->
     <authentication-manager alias="authenticationManager">
         <authentication-provider  ref="casAuthenticationProvider">
         </authentication-provider>
     </authentication-manager>
         <!-- 认证提供者 -->
     <beans:bean id="casAuthenticationProvider"
class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
        <beans:property name="authenticationUserDetailsService">
           <beans:bean
```

```
class="org.springframework.security.core.userdetails.UserDetailsByNameServiceWrapper">

                <beans:constructor-arg ref="userDetailsService" />

            </beans:bean>

        </beans:property>

        <beans:property name="serviceProperties" ref="serviceProperties"/>

        <!-- ticketValidator 为票据验证器 -->

        <beans:property name="ticketValidator">

            <beans:bean
class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">

                <beans:constructor-arg index="0" value="http://localhost:9100/cas"/>

            </beans:bean>

        </beans:property>

        <beans:property name="key" value="an_id_for_this_auth_provider_only"/>

    </beans:bean>

          <!-- 认证类 -->

     <beans:bean id="userDetailsService"
class="cn.itcast.demo.service.UserDetailServiceImpl"/>



    <!-- 认证过滤器 结束 -->

    <!-- 单点登出  开始  -->

    <beans:bean id="singleLogoutFilter"
class="org.jasig.cas.client.session.SingleSignOutFilter"/>

    <beans:bean id="requestSingleLogoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFilter">

        <beans:constructor-arg
```

```
value="http://localhost:9100/cas/logout?service=http://www.baidu.com"/>

        <beans:constructor-arg>

            <beans:bean
class="org.springframework.security.web.authentication.logout.SecurityContextLogout
Handler"/>

        </beans:constructor-arg>

        <beans:property name="filterProcessesUrl" value="/logout/cas"/>

    </beans:bean>

    <!-- 单点登出  结束 -->

</beans:beans>
```

（3）创建 UserDetailsServiceImpl

```
/**

 * 认证类

 */

public class UserDetailServiceImpl implements UserDetailsService {

    @Override

    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {

        //构建角色集合

        List<GrantedAuthority> authorities=new ArrayList();

        authorities.add(new SimpleGrantedAuthority("ROLE_USER"));

        return new User(username, ""  , authorities);

    }

}
```

这个类的主要作用是在登陆后得到用户名，可以根据用户名查询角色或执行一些逻辑。

4.3 获取登录名

我们在处理后端逻辑需要获得登录名，那么如何获取单点登录的用户名呢？其实和我们之前获得用户名的方式是完全相同的，我们下面来做个测试。

（1）web.xml 添加 springmvc

```
<servlet>
    <servlet-name>springmvc</servlet-name>

    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class
>
    <!-- 指定加载的配置文件 ，通过参数 contextConfigLocation 加载-->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springmvc.xml</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>springmvc</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

（2）创建 springmvc.xml

```
    <context:component-scan base-package="cn.itcast.demo" />
    <mvc:annotation-driven />
```

（3）创建 UserController

```
@RestController

public class UserController {

    @RequestMapping("/findLoginUser")

    public void  findLoginUser(){

        String name =
SecurityContextHolder.getContext().getAuthentication().getName();

        System.out.println(name);

    }

}
```

地址栏输入 http://localhost:9003/findLoginUser.do 即可在控制台看到输出的登录名。

4.4 退出登录

修改 spring-security.xml

```
<beans:bean id="requestSingleLogoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFilter">

       <beans:constructor-arg
value="http://localhost:9100/cas/logout?service=http://localhost:9003/index2.html"/
>

       <beans:constructor-arg>

           <beans:bean
class="org.springframework.security.web.authentication.logout.SecurityContextLogout
Handler"/>

       </beans:constructor-arg>

       <beans:property name="filterProcessesUrl" value="/logout/cas"/>

</beans:bean>
```

在页面上添加链接

```
<a href="/logout/cas">退出登录</a>
```

创建 index2.html,将 index2.html 设置为可匿名访问

```
<http pattern="/index2.html" security="none"></http>
```

5.品优购用户中心

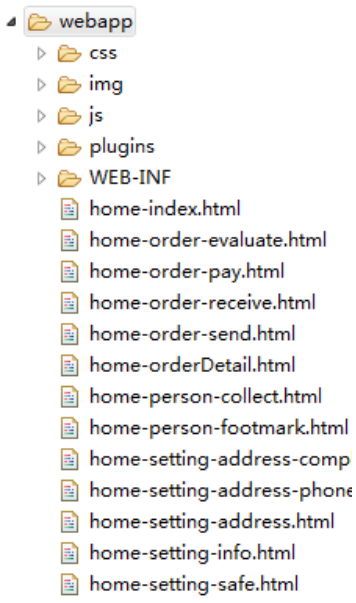
5.1 需求分析

用户中心实现单点登录。

5.2 代码实现

5.2.1 用户中心实现单点登录

（1）将用户中心相关的页面（home-开头的）拷贝至 pinnyougou-user-web



（2）pom.xml 引入 springSecurity、cas 客户端和 springSecurity Cas 整合包依赖（参照 casclient_demo3）。

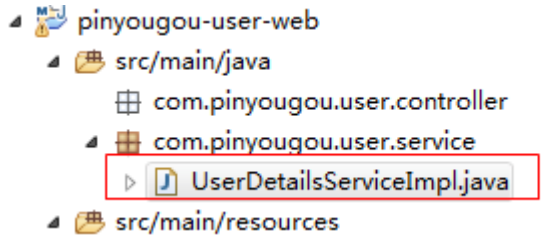
（3）web.xml 添加 spring-security 过滤器（参照参照 casclient_demo3）设置首页为 home-index.html

```
<welcome-file-list>

    <welcome-file>home-index.html</welcome-file>

</welcome-file-list>
```

（4）构建 UserDetailsServiceImpl.java （参照 casclient_demo3）



（5）添加 spring-security.xml（参照 casclient_demo3），并做以下修改

配置匿名访问资源

```
<!-- 匿名访问资源 -->

<http pattern="/css/**" security="none"></http>

<http pattern="/js/**" security="none"></http>

<http pattern="/image/**" security="none"></http>

<http pattern="/plugins/**" security="none"></http>

<http pattern="/register.html" security="none"></http>

<http pattern="/user/add.do" security="none"></http>

<http pattern="/user/sendCode.do" security="none"></http>
```

设置服务地址属性

```
<beans:bean id="serviceProperties"
          class="org.springframework.security.cas.ServiceProperties">

    <beans:property name="service" value="http://localhost:9106/login/cas"/>

</beans:bean>
```

设置认证类

```
<beans:bean

id="userDetailsService" class="com.pinyougou.user.service.UserDetailServiceImpl"/>
```

5.2.2 页面显示用户名

（1）pinyougou-user-web 创建 LoginController.java

```
@RestController

@RequestMapping("/login")

public class LoginController {

    @RequestMapping("/name")

    public Map showName(){

        String name =
SecurityContextHolder.getContext().getAuthentication().getName();//得到登陆人账号

        Map map=new HashMap<>();

        map.put("loginName", name);

        return map;

    }

}
```

（2）创建 loginService.js

```
//服务层

app.service('loginService',function($http){

    //读取列表数据绑定到表单中

    this.showName=function(){

        return $http.get('../login/name.do');

    }
```

```
});
```

（3）创建 indexController.js

```
//首页控制器
app.controller('indexController',function($scope,loginService){
    $scope.showName=function(){
        loginService.showName().success(
            function(response){
                $scope.loginName=response.loginName;
            }
        );
    }
});
```

（5）修改 home-index.html　引入 js

```
    <script type="text/javascript" src="plugins/angularjs/angular.min.js"></script>
    <script type="text/javascript" src="js/base.js"></script>
    <script type="text/javascript" src="js/service/loginService.js"></script>
    <script type="text/javascript" src="js/controller/indexController.js"></script>
```

指令，调用方法查询登陆名

```
<body ng-app="pinyougou" ng-controller="indexController" ng-init="showName()">
```

显示用户名

```
 <span class="name">{{loginName}}</span>
```

5.2.3 退出登录

设置退出登录后的跳转地址

```
<beans:bean id="requestSingleLogoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFilter">

       <beans:constructor-arg
value="http://localhost:9100/cas/logout?service=http://localhost:9103"/>

      ........

</beans:bean>
```

退出登录后，跳转到网站首页

```
<span class="safe"> <a href="/logout/cas">退出登录 </a></span>
```

附录 A. Spring Security 内置过滤器表

别名	**Filter** 类
CHANNEL_FILTER	ChannelProcessingFilter
SECURITY_CONTEXT_FILTER	SecurityContextPersistenceFilter
CONCURRENT_SESSION_FILTER	ConcurrentSessionFilter
LOGOUT_FILTER	LogoutFilter
X509_FILTER	X509AuthenticationFilter
PRE_AUTH_FILTER	AstractPreAuthenticatedProcessingFilter 的子类
CAS_FILTER	CasAuthenticationFilter
FORM_LOGIN_FILTER	UsernamePasswordAuthenticationFilter
BASIC_AUTH_FILTER	BasicAuthenticationFilter
SERVLET_API_SUPPORT_FILTER	SecurityContextHolderAwareRequestFilter
JAAS_API_SUPPORT_FILTER	JaasApiIntegrationFilter
REMEMBER_ME_FILTER	RememberMeAuthenticationFilter
ANONYMOUS_FILTER	AnonymousAuthenticationFilter
SESSION_MANAGEMENT_FILTER	SessionManagementFilter
EXCEPTION_TRANSLATION_FILTER	ExceptionTranslationFilter
FILTER_SECURITY_INTERCEPTOR	FilterSecurityInterceptor
SWITCH_USER_FILTER	SwitchUserFilter