一般来说，Web 应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中 的合法主体，也就是说用户能否访问该系统。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个 文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

对 于上面提到的两种应用情景，Spring Security 框架都有很好的支持。在用户认证方面，Spring Security 框架支持主流的认证方式，包括 HTTP 基本认证、HTTP 表单验证、HTTP 摘要认证、OpenID 和 LDAP 等。在用户授权方面，Spring Security 提供了基于角色的访问控制和访问控制列表（Access Control List，ACL），可以对应用中的领域对象进行细粒度的控制。

若Spring Security 整合 CAS单点登录 ，使用CAS进行认证和获取授权信息，使用Spring Security验证权限，则可以很好的把公共的认证和授权与具体应用剥离开来，同时简化应用的配置。本文就Spring Security 与 CAS的整合进行说明。

一般来说，Web 应用的安全性包括用户认证（Authentication）和用户授权（Authorization）两个部分。用户认证指的是验证某个用户是否为系统中 的合法主体，也就是说用户能否访问该系统。用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同用户所具有的权限是不同的。比如对一个 文件来说，有的用户只能进行读取，而有的用户可以进行修改。一般来说，系统会为不同的用户分配不同的角色，而每个角色则对应一系列的权限。

对 于上面提到的两种应用情景，Spring Security 框架都有很好的支持。在用户认证方面，Spring Security 框架支持主流的认证方式，包括 HTTP 基本认证、HTTP 表单验证、HTTP 摘要认证、OpenID 和 LDAP 等。在用户授权方面，Spring Security 提供了基于角色的访问控制和访问控制列表（Access Control List，ACL），可以对应用中的领域对象进行细粒度的控制。

若Spring Security 整合 CAS单点登录 ，使用CAS进行认证和获取授权信息，使用Spring Security验证权限，则可以很好的把公共的认证和授权与具体应用剥离开来，同时简化应用的配置。本文就Spring Security 与 CAS的整合进行说明。

一 、 基本需求  
1. jdk 5.0  
2. tomcat 6  
3. Spring 3.0.5.RELEASE  
4. Spring Security 3.1.0.RELEASE  
5. CAS cas-server-3.4.7， cas-client-3.2.0   
6. 使用http协议进行传输  
7. 通过jdbc进行用户验证，需要通过casserver提供除登录用户名以外的附加信息（用于Spring Security 进行验证权限）

二、搭建CAS Server  
1. 把从 <http://www.jasig.org/cas/download> 上下载cas解压找到 cas-server-3.4.7-releasecas-server-3.4.7modulescas-server-webapp-3.4.7.war , 解压cas-server-webapp-3.4.7.war ，部署在至tomcat上端口为 8080的server上，如部署路径为 <http://localhost:8080/cas> 。为了达到需求目的，我们主要需要对 /WEB-INF/deployerConfigContext.xml 文件进行修改。

2. 使用jdbc数据源进行用户认证，需要修改deployerConfigContext.xml 的authenticationHandlers方式  
<property name="authenticationHandlers">  
<list>  
<!--  
| This is the authentication handler that authenticates services by means of callback via SSL, thereby validating  
| a server side SSL certificate.  
+-->  
<bean class="org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticationHandler"  
p:httpClient-ref="httpClient" />  
<!--  
| This is the authentication handler declaration that every CAS deployer will need to change before deploying CAS   
| into production. The default SimpleTestUsernamePasswordAuthenticationHandler authenticates UsernamePasswordCredentials  
| where the username equals the password. You will need to replace this with an AuthenticationHandler that implements your  
| local authentication strategy. You might accomplish this by coding a new such handler and declaring  
| edu.someschool.its.cas.MySpecialHandler here, or you might use one of the handlers provided in the adaptors modules.  
+-->  
//注释掉，否则只要用户名和密码一致的话都可以得到认证   
<!-- bean class="org.jasig.cas.authentication.handler.support.SimpleTestUsernamePasswordAuthenticationHandler" / -->  
  
//数据库认证方式  
<!--DATABASE -- >  
<bean class="org.jasig.cas.adaptors.jdbc.QueryDatabaseAuthenticationHandler">  
<property name="dataSource" ref="dataSource" />  
<property name="sql" value="select password from t\_admin\_user where login\_name=?" />  
</bean>  
  
</list>  
<property name="authenticationHandlers">

<!-- DATABASE 增加数据源配置 -->  
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">  
<property name="driverClassName"><value>com.mysql.jdbc.Driver</value></property>  
<property name="url"><value>jdbc:mysql:///cas?useUnicode=true&amp;characterEncoding=utf-8</value></property>  
<property name="username"><value>root</value></property>  
</bean>

3.通过casserver提供除登录用户名以外的附加信息（用于Spring Security 进行验证权限），修改/WEB-INF/deployerConfigContext.xml  
3.1 修改credentialsToPrincipalResolvers  
<property name="credentialsToPrincipalResolvers">  
<list>  
<bean class="org.jasig.cas.authentication.principal.UsernamePasswordCredentialsToPrincipalResolver" >   
<property name="attributeRepository" ref="attributeRepository" /> //增加此属性，为认证过的用户的Principal添加属性  
</bean>  
<bean class="org.jasig.cas.authentication.principal.HttpBasedServiceCredentialsToPrincipalResolver" />  
</list>  
</property>  
  
3.2 修改该文件中默认的 attributeRepositorybean配置  
<!-- 使用SingleRowJdbcPersonAttributeDao 获取更多用户的信息 -->  
<bean id="attributeRepository" class="org.jasig.services.persondir.support.jdbc.SingleRowJdbcPersonAttributeDao">  
<constructor-arg index="0" ref="dataSource"/>  
<constructor-arg index="1" value="select role\_name,group\_name from role where login\_name = ?"/>   
<!--这里的key需写username,value对应数据库用户名字段 -->  
<property name="queryAttributeMapping">  
<map>  
<entry key="username" value="login\_name"/>  
</map>  
</property>  
<!--key对应数据库字段，value对应客户端获取参数 -->  
<property name="resultAttributeMapping">  
<map>  
<entry key="role\_name" value="authorities"/> //这个从数据库中获取的角色，用于在应用中security的权限验证  
</map>  
</property>  
</bean>

3.3 修改该文件中最默认的serviceRegistryDao中的属性全部注释掉  
这 个bean中的RegisteredServiceImpl的ignoreAttributes属性将决定是否添加attributes属性内容，默认为 false:不添加，只有去掉这个配置，cas server才会将获取的用户的附加属性添加到认证用的Principal的attributes中去。  
<bean id="serviceRegistryDao" class="org.jasig.cas.services.InMemoryServiceRegistryDaoImpl"></bean>

3.4 若采用CAS的 Cas20ServiceTicketValidator 认证，则需要修改WEB-INFviewjspprotocol2.0casServiceValidationSuccess.jsp文件，才能把获取的属性传递至客户端  
<%@ page session="false" %>  
<%@ taglib prefix="c" uri="<http://java.sun.com/jsp/jstl/core>" %>  
<%@ taglib uri="<http://java.sun.com/jsp/jstl/functions>" prefix="fn" %>  
<cas:serviceResponse xmlns:cas='http://www.yale.edu/tp/cas'>  
<cas:authenticationSuccess>  
<cas:user>${fn:escapeXml(assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.id)}</cas:user>  
  
<c:if test="${not empty pgtIou}">  
<cas:proxyGrantingTicket>${pgtIou}</cas:proxyGrantingTicket>  
</c:if>  
<c:if test="${fn:length(assertion.chainedAuthentications) > 1}">  
<cas:proxies>  
<c:forEach var="proxy" items="${assertion.chainedAuthentications}" varStatus="loopStatus" begin="0" end="${fn:length(assertion.chainedAuthentications)-2}" step="1">  
<cas:proxy>${fn:escapeXml(proxy.principal.id)}</cas:proxy>  
</c:forEach>  
</cas:proxies>  
</c:if>  
<!-- 增加如下内容 -->  
<c:if test="${fn:length(assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.attributes)> 0}">  
<cas:attributes>  
<c:forEach   
var="attr"  
items="${assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.attributes}"  
varStatus="loopStatus"   
begin="0"  
end="${fn:length(assertion.chainedAuthentications[fn:length(assertion.chainedAuthentications)-1].principal.attributes)-1}"  
step="1">  
<cas:${fn:escapeXml(attr.key)}>${fn:escapeXml(attr.value)}</cas:${fn:escapeXml(attr.key)}>  
</c:forEach>  
</cas:attributes>  
</c:if>  
</cas:authenticationSuccess>  
</cas:serviceResponse>

至此，CAS Server 搭建完毕。

三、搭建CAS Client （即Spring Security）应用

1. CAS Client 下需要把spring-security-cas-3.1.0.M2.jar等Spring Security 相关的jar引入，把cas-client-core-3.2.0.jar引入，用于从cas server上获取相关认证与授权信息。

2. CAS Client应用的 web.xml ， 增加如下  
<!-- spring 配置文件-->  
<context-param>  
<param-name>contextConfigLocation</param-name>  
<param-value>classpath:applicationContext-security-ns.xml</param-value>  
</context-param>

<!-- spring security filter -->  
<filter>  
<filter-name>springSecurityFilterChain</filter-name>  
<filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>  
</filter>

<filter-mapping>  
<filter-name>springSecurityFilterChain</filter-name>  
<url-pattern>/\*</url-pattern>  
</filter-mapping>

<!-- spring 默认侦听器 -->  
<listener>  
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>

3. spring security 文件的配置 applicationContext-security-ns.xml

<?xml version="1.0"?>  
<beans xmlns="<http://www.springframework.org/schema/beans>"   
xmlns:security="<http://www.springframework.org/schema/security>"   
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-3.0.xsd>   
<http://www.springframework.org/schema/security><http://www.springframework.org/schema/security/spring-security-3.1.xsd>">

<!--  
Enable security, let the casAuthenticationEntryPoint handle all intercepted urls.  
The CAS\_FILTER needs to be in the right position within the filter chain.  
-->   
<security:http entry-point-ref="casAuthenticationEntryPoint" auto-config="true">  
<security:intercept-url pattern="/\*\*" access="ROLE\_USER"></security:intercept-url>  
<security:custom-filter position="CAS\_FILTER" ref="casAuthenticationFilter"></security:custom-filter>  
</security:http>

<!--  
Required for the casProcessingFilter, so define it explicitly set and  
specify an Id Even though the authenticationManager is created by  
default when namespace based config is used.  
-->  
<security:authentication-manager alias="authenticationManager">  
<security:authentication-provider ref="casAuthenticationProvider"></security:authentication-provider>  
</security:authentication-manager>

<!--  
This section is used to configure CAS. The service is the  
actual redirect that will be triggered after the CAS login sequence.  
-->  
<bean id="serviceProperties" class="org.springframework.security.cas.ServiceProperties">  
//http://localhost:8088/SpringSecurity 具体应用  
// j\_spring\_cas\_security\_check spring的虚拟URL，此标志标识使用 CAS authentication upon return from CAS SSO login.  
<property name="service" value="[http://localhost:8088/SpringSecurity/j\_spring\_cas\_security\_check"></property](http://localhost:8088/SpringSecurity/j_spring_cas_security_check)>  
<property name="sendRenew" value="false"></property>  
</bean>

<!--  
The CAS filter handles the redirect from the CAS server and starts the ticket validation.  
-->  
<bean id="casAuthenticationFilter" class="org.springframework.security.cas.web.CasAuthenticationFilter">  
<property name="authenticationManager" ref="authenticationManager"></property>  
</bean>

<!--  
The entryPoint intercepts all the CAS authentication requests.  
It redirects to the CAS loginUrl for the CAS login page.  
-->  
<bean id="casAuthenticationEntryPoint" class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">  
<property name="loginUrl" value="[http://localhost:8080/cas/login"></property](http://localhost:8080/cas/login)> //SSO登录地址  
<property name="serviceProperties" ref="serviceProperties"></property>  
</bean>

<!--  
Handles the CAS ticket processing.  
-->  
<bean id="casAuthenticationProvider" class="org.springframework.security.cas.authentication.CasAuthenticationProvider">  
<property name="authenticationUserDetailsService" ref="authenticationUserDetailsService"/>  
<property name="serviceProperties" ref="serviceProperties"></property>  
<property name="ticketValidator">  
<bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">  
<constructor-arg index="0" value="<http://localhost:8080/cas>" /> //SSO验证地址  
</bean>  
</property>  
<property name="key" value="cas"></property>  
  
</bean>  
<!-- authorities对应 CAS server的 登录属性， 在此设置到spirng security中，用于spring security的验证 -->  
<bean id="authenticationUserDetailsService" class="org.springframework.security.cas.userdetails.GrantedAuthorityFromAssertionAttributesUserDetailsService">  
<constructor-arg>  
<array>  
<value>authorities</value>  
</array>  
</constructor-arg>  
</bean>  
</beans>

至此，CAS客户端搭建完毕。

四、总结

通过上述的配置，则具体应用在使用的时候，用户认证和授权则无需过问，只需在应用中配置相关的角色访问权限即可。即，只需对下面的红色部分进行修改，即可以完成应用的认证和授权工作。大大简化了应用和认证与授权的剥离工作

<security:http entry-point-ref="casAuthenticationEntryPoint" auto-config="true">  
<security:intercept-url pattern="/\*\*" access="ROLE\_USER"></security:intercept-url>   
<security:custom-filter position="CAS\_FILTER" ref="casAuthenticationFilter"></security:custom-filter>  
</security:http>

五、扩展

若在同一SSO下有多个应用，同一户在不同应用下有不同的角色，则考虑扩展获取用户权限的环节；资源和角色在数据库中进行配置等等。