1. 文件配置
2. 修改pom.xml

增加以下内容：

<!-- Import Spring Security to do Access Control,test on 180822 -->

<dependency>

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

<artifactId>spring-security-web</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-security-config</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-security-acl</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-security-core</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-security-taglibs</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

<dependency>

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

<artifactId>spring-security-ldap</artifactId>

<version>3.2.9.RELEASE</version>

</dependency>

现在最新的SpringSecurity版本为5.0.7。但是由于我们在ALM Web中使用的Spring和SpringMVC版本为3.2.13，所以使用与其匹配的SpringSecurity版本为3.2.9

1. 增加integrity-secutiry.xml



1. 修改Web.xml

修改如下内容：

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:integrity-servlet.xml,classpath:integrity-security.xml,classpath:integrity-context.xml</param-value>

</context-param>

增加如下内容：

<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>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:integrity-security.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

这部分修改使integrity-security.xml可以正常加载

删除如下内容：

<listener>

<listener-class>ext.tcl.listener.LoginListener</listener-class>

</listener>

<!-- Ext ajax 请求session超时过滤器 -->

<filter>

<filter-name>sessionFilter</filter-name>

<filter-class>ext.tcl.filter.SessionFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>sessionFilter</filter-name>

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

</filter-mapping>

<!-- 在/的路径下，去springMVC中去找路径 -->

<servlet>

<servlet-name>integrity</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:integrity-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

1. 修改拦截规则

在integrity-servlet.xml中移除以下内容：

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<!-- 登陆认证拦截器 -->

<mvc:interceptor>

<!-- /\*\*表示所有url包括子url路径 -->

<mvc:mapping path=*"/\*\*"*/>

<bean class=*"ext.tcl.interceptor.LoginInterceptor"*>

<property name=*"allowUrls"*>

<list>

<!-- 如果请求中包含以下路径，则不进行拦截 -->

<value>/login</value>

<value>/js</value>

<value>/extjs</value>

<value>/css</value>

<value>/images</value>

</list>

</property>

</bean>

</mvc:interceptor>

</mvc:interceptors>

这部分修改移除了SpringMVC使用的拦截器规则，SpringMVC的拦截器和SpringSecurity的Filter应该有冲突，如果不去掉这个注册的话，SpringSecurity的登录信息无法传到服务器端。

增加以下内容：

<context:component-scan base-package=*"ext.tcl.security"* />

后面可以考虑修改代码结构就不用增加这个了

1. Java代码修改

增加以下文件：

\ext\tcl\action\UserController.java

\ext\tcl\security\LoginAuthenticationProvider.java

\ext\tcl\security\LoginUser.java

\ext\tcl\security\LoginUserDetailsImpl.java

\ext\tcl\security\LoginUserDetailsService.java

\ext\tcl\security\LoginUserDetailsServiceImpl.java

\ext\tcl\security\TCLAuthenticationSuccessHandler.java

1. JSP代码修改

修改Login.Jsp

<form method=*"post"* action=*"*${pageContext.request.contextPath }*/loginForm"*>

<input id=*"username"* name=*"username"* placeholder=*"User Name"* required=*""* type=*"text"*>

<hr class=*"hr15"*>

<input id=*"password"* name=*"password"* placeholder=*"Password"* required=*""* type=*"password"*>

<hr class=*"hr15"*>

<input type=*"submit"* value=*"登录"* name=*"submit"*>

</form>

这部分的作用是将用户登录的表单和Security中的登陆配置相匹配

1. 配置的对应关系

1） Filter的注册

该功能最重要的是integrity-security中的设置以下为针对integrity-security.xml内容的分析：

<security:http pattern=*"/\*\*/\*.jpg"* security=*"none"*/>

<security:http pattern=*"/\*\*/\*.png"* security=*"none"*/>

<security:http pattern=*"/\*\*/\*.gif"* security=*"none"*/>

<security:http pattern=*"/\*\*/\*.ico"* security=*"none"*/>

<security:http pattern=*"/\*\*/\*.css"* security=*"none"*/>

<security:http pattern=*"/\*\*/\*.js"* security=*"none"*/>

以上内容为不需要经过SpringSecurity过滤的地址，我们把静态地址和.js文件设置为不需要过滤

2） Login相关页面的注册

<security:http pattern=*"/loginPage"* security=*"none"* />

设置登陆页面不需要过滤

<security:http auto-config=*"true"* use-expressions=*"true"*>

<security:intercept-url pattern=*"/\*\*"* access=*"hasRole('everyone')"*/>

<security:form-login

login-page=*"/loginPage"*

password-parameter=*"password"*

username-parameter=*"username"*

authentication-failure-url=*"/loginPage?error=error"*

authentication-success-handler-ref=*"authSuccess"*

login-processing-url=*"/loginForm"*/>

<security:logout invalidate-session=*"true"* logout-success-url=*"/index"* logout-url=*"/logout"*/>

</security:http>

login-page=*"/loginPage"* 这句用来设置SpringSecurity使用的登陆页面

password-parameter=*"password"*

username-parameter=*"username"*

在登陆Form中对应的username,password使用的提交参数名

authentication-failure-url=*"/loginPage?error=error"*这句用来说明登录失败后的跳转页面

authentication-success-handler-ref=*"authSuccess"* 这句用来说明使用的登陆Handler

login-processing-url=*"/loginForm"*

这句话来说明提交到Login中使用的表单名，所以在整个项目中应该不能重名（猜测，未验证）

<security:logout invalidate-session=*"true"* logout-success-url=*"/index"* logout-url=*"/logout"*/>

这句表明登陆成功后的跳转页面和注销的页面地址

3）登录页面的配置

登录的loginForm表单内容如下：

<form method=*"post"* action=*"*${pageContext.request.contextPath }*/loginForm"*>

<input id=*"username"* name=*"username"* placeholder=*"UserName"* required=*""* type=*"text"*>

<hr class=*"hr15"*>

<input id=*"password"* name=*"password"* placeholder=*"Password"* required=*""* type=*"password"*>

<hr class=*"hr15"*>

<input type=*"submit"* value=*"登录"* name=*"submit"*>

</form>

其中需要注意以下几点：

两个Input必须使用name=””,必须标注name信息，只是用了id则匹配不上。Name中的数据必须和前面integrity-security.xml中username-parameter和password-parameter的值对应，否则提交不成功。

action=*"*${pageContext.request.contextPath }*/loginForm"*

action需要加全路径，按照网上某些例子，action可以只添加/loginForm但是经过测试发现如果使用/loginForm后提交地址会变为：

<http://localhost:8880/loginForm>

缺少了中间项目名的信息，导致提交不成功。

3）登陆使用的定制账号验证过程如下

SpringSecurity本身继承了LDAP的验证过程，可以通过配置使用LDAP进行验证，但在配置过程中没有成功，所以修改了Login的验证过程，同样在integrity-security.xml中添加如下内容：

<b:bean id=*"loginUserDetailService"* class=*"ext.tcl.security.LoginUserDetailsServiceImpl"*></b:bean>

<b:bean id=*"loginAuthenticationProvider"* class=*"ext.tcl.security.LoginAuthenticationProvider"*>

<b:property name=*"userDetailsService"* ref=*"loginUserDetailService"*></b:property>

</b:bean>

<security:authentication-manager alias=*"tclAuthenticationManager"*>

<security:authentication-provider ref=*"loginAuthenticationProvider"*>

</security:authentication-provider>

</security:authentication-manager>

增加了关于*loginAuthenticationProvider*的注册信息指向*ext.tcl.security.LoginAuthenticationProvider*

LoginAuthenticationProvider这个类继承了AbstractUserDetailsAuthenticationProvider，并且通过使用类里面的userDetailsService进行LDAP验证。在userDetailsService的实现中使用自己的LDAP验证规则。

4）handler的注册

<b:bean id=*"authSuccess"* class=*"ext.tcl.security.TCLAuthenticationSuccessHandler"*></b:bean>

这个handler的代码配置如下：

**public** **class** TCLAuthenticationSuccessHandler **implements** AuthenticationSuccessHandler {

**private** **static** Log *logger* = LogFactory.*getLog*(TCLAuthenticationSuccessHandler.**class**);

@Autowired

**private** IPermissionService permissionService;

**private** RedirectStrategy redirectStrategy = **new** DefaultRedirectStrategy();

@Override

**public** **void** onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication auth) **throws** ServletException, IOException {

UserDetails userDetails = (UserDetails) SecurityContextHolder.*getContext*()

.getAuthentication()

.getPrincipal();

String username = userDetails.getUsername();

setOriginPermission(request, username);

redirectStrategy.sendRedirect(request, response, "/index");

}

**private** **void** setOriginPermission(HttpServletRequest request, String username) {

HttpSession session = request.getSession();

**try** {

**if** (session.getAttribute("permissions") == **null**) {

// String user = (String) session.getAttribute("userName");

session.setAttribute("userName",username);

List<TypePermissionVO> pms = permissionService.permissions(username);

permissionService.initialUserPermission(username);

*logger*.info("pms "+pms+" pmsJSON "+JsonUtil.*list2json*(pms));

session.setAttribute("permissions", pms);

session.setAttribute("pmsJSON", JsonUtil.*list2json*(pms));

}

} **catch** (Exception e) {

*logger*.warn("Load Permissions Exception: " + e.getMessage());

}

}

}

SpringSecurity中使用UserDetails userDetails = (UserDetails) SecurityContextHolder.*getContext*()

.getAuthentication()

.getPrincipal();

String username = userDetails.getUsername();

获取用户的信息，这里是账号是标准操作。

**private** RedirectStrategy redirectStrategy = **new** DefaultRedirectStrategy();

这里的redirectStrategy是跳转的标准操作。

网上有例子使用

request.getRequestDispatcher("/login").forward(request, response);

进行登录成功后的跳转，经测试发现并不能成功跳转，在登录和成功之后的操作结束后，会跳转至loginForm并且提示出405的错误。但是实际上loginForm在之前已经成功提交。从网上搜也并没有看到类似的错误或者解决方法。

1. 权限配置

在LoginUserDetailsServiceImpl中代码如下：

/\*\*

\* 进行登录验证

\*/

**public** UserDetails loadUserByUsername(String username, String password) **throws** UsernameNotFoundException {

**boolean** result = **this**.validate(username, password);

**if** (!result) {

**return** **null**;

}

Collection<GrantedAuthority> authorities = **new** ArrayList<GrantedAuthority>();

**try** {

List<String> groups = typeUtil.groups(username);

**if** (groups.isEmpty()) {

} **else** {

List<GroupVO> groupRoles = groupService.findGroups(groups);

**for** (GroupVO groupVO : groupRoles) {

List<RoleVO> gr = groupVO.getRoles();

**if** (gr != **null**) {

**for** (RoleVO roleVO : gr) {

authorities.add(**new** SimpleGrantedAuthority(roleVO.getName()));

}

}

}

}

**boolean** isLatam = csiService.isLatamUsers(username);

**if**(isLatam){

authorities.add(**new** SimpleGrantedAuthority("LATAM"));

}

} **catch** (APIException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

LoginUserDetailsImpl user = **new** LoginUserDetailsImpl(username, password, authorities);

**return** user;

}

/\*\*

\* 在此处验证

\*

\* **@param** username

\* **@param** password

\* **@return**

\*/

**private** **boolean** validate(String username, String password) {

/\*\*

\* 此处应该在数据库获取用户信息进行验证

\*/

**boolean** valide = **false**;

**try** {

String userDN = ldapService.getDnForUser(username);

valide = ldapService.authenticate(userDN, password);

} **catch** (RuntimeException e1) {

valide = **false**;

}

**return** valide;

}

使用validate方法调用原有的LDAP帐号验证程序来确定帐号密码是否正确。

如果用户输入帐号密码正确，则使用typeUtil.groups(username);来获取用户所述的组。

在获取组信息之后使用groupService.findGroups(groups);获取这些组映射的角色信息。

由于CSI的权限配置是通过user Item的DC信息进行确认的，所以在确认角色之后通过使用csiService.isLatamUsers(username);来获取用户是否是Latam用户。这里在service中增加了一个使用Boolean作为返回值的接口。

Index.jsp修改如下：



Index.jsp中首先增加以下引用来使SpringSecurity生效：

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

如果需要使用角色来控制显示则需要用到以下标签：

<sec:authorize access=*"hasRole('everyone')"*>

</security>

比如CSI和CCSI报表的周围增加了如下内容：

<sec:authorize access=*"hasRole('LATAM')"*>

<li>CCSI Report</li>

<li>CSI Report</li>

</sec:authorize>

并且去掉了后续代码中使用islatam进行标签隐藏的代码。

注意：

els.each(**function**(index,el) {

**var** prtEl = el.parentElement.parentElement;

**var** groupAdmin = NameSpace.Session.permission[NameSpace.Session.permission.length - 1];

$(el).hide();

**if**(prtEl.id == 'box-6' && groupAdmin.relevance) {

$(el).show();

}

**if**(prtEl.id == 'box-4' || prtEl.id == 'box-5') {

$(el).show();

}

});

这部分代码隐藏的标签注释掉，使用Spring Security进行标签显示的控制。

以下代码同样进行注释：

//if($(this).text()=="Reports List" || $(this).text()=="CCSI Report"

// ||$(this).text()=="CSI Report" || $(this).text()=="Admin Group" || $(this).text()=="FR ImportAndReuse"){

// $(".itemId").hide();

//}else{

// $(".itemId").show();

//}

原系统逻辑

1. 使用拦截器拦截请求

拦截器注册在integrity-servlet.xml中

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<!-- 登陆认证拦截器 -->

<mvc:interceptor>

<!-- /\*\*表示所有url包括子url路径 -->

<mvc:mapping path="/\*\*"/>

<bean class="ext.tcl.interceptor.LoginInterceptor">

<property name="allowUrls">

<list>

<!-- 如果请求中包含以下路径，则不进行拦截 -->

<value>/login</value>

<value>/js/jquery</value>

<value>/extjs</value>

<value>/css</value>

<value>/images</value>

</list>

</property>

</bean>

</mvc:interceptor>

</mvc:interceptors>

在这个注册文件中排除了以下地址的拦截：

<value>/login</value>

<value>/js/jquery</value>

<value>/extjs</value>

<value>/css</value>

<value>/images</value>

对应代码ext.tcl.interceptor.LoginInterceptor

该类实现了public class LoginInterceptor implements HandlerInterceptor接口，在preHandle方法中有以下逻辑：

if(request.getRequestURI().indexOf("/login")>=0 || request.getRequestURI().indexOf("/tableau")>=0

|| request.getRequestURI().indexOf("/getToken")>=0){

return true;

}

这部分表示凡是在/login,/tableau,/getToken文件夹或者对应请求都不尽兴拦截

这三个功能中/login对应登陆，/tableau和/getToken对应着和Tableau系统的集成和外放。

if(session.getAttribute("userName")!=null) {

UserSession usersession = (UserSession) session.getAttribute("UserSession");

if(usersession != null && session.getId().equals(usersession.getSessionId())) {

try {

if(session.getAttribute("permissions") == null) {

String user = (String) session.getAttribute("userName");

List<TypePermissionVO> pms = permissionService.permissions(user);

permissionService.initialUserPermission(user);

session.setAttribute("permissions", pms);

session.setAttribute("pmsJSON", JsonUtil.list2json(pms));

}

} catch (Exception e) {

log.warn("Load Permissions Exception: " + e.getMessage());

}

return true;

}

}

这部分根据用户名生成对应的权限信息并保存在Session中。

boolean flag = false;

String requestUrl = request.getRequestURI().replace(request.getContextPath(), "");

if(null != allowUrls && allowUrls.length>=0) {

for(String url : allowUrls) {

if(requestUrl.contains(url)) {

flag = true;

break;

}

}

}

if(flag) {

return true;

}

这部分对应前面配置的XML中的AllowURLS。表示用户如果没有登陆过，则判断请求的是不是不需要验证的内容，如果是则放行。

//用户没有登录挑战到登录页面

request.getRequestDispatcher("/login.jsp").forward(request, response);

这部分是说用户登陆的是需要验证的页面，并且没有登录。则跳转回登陆页面。

1. 使用Filter在登录时进行权限验证
2. 使用js控制标签可见性