Expression-Based Access Control in Spring Security

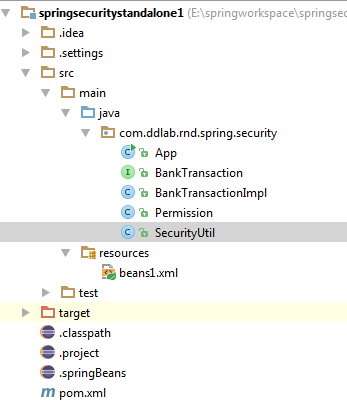
The base class for expression root objects is SecurityExpressionRoot. This provides some common expressions which are available in both web and method security.

**Common built-in expressions**

| **Expression** | **Description** |
| --- | --- |
| hasRole([role]) | Returns true if the current principal has the specified role. By default if the supplied role does not start with 'ROLE\_' it will be added. This can be customized by modifying the defaultRolePrefix onDefaultWebSecurityExpressionHandler. |
| hasAnyRole([role1,role2]) | Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings). By default if the supplied role does not start with 'ROLE\_' it will be added. This can be customized by modifying the defaultRolePrefix onDefaultWebSecurityExpressionHandler. |
| hasAuthority([authority]) | Returns true if the current principal has the specified authority. |
| hasAnyAuthority([authority1,authority2]) | Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings) |
| principal | Allows direct access to the principal object representing the current user |
| authentication | Allows direct access to the current Authentication object obtained from the SecurityContext |
| permitAll | Always evaluates to true |
| denyAll | Always evaluates to false |
| isAnonymous() | Returns true if the current principal is an anonymous user |
| isRememberMe() | Returns true if the current principal is a remember-me user |
| isAuthenticated() | Returns true if the user is not anonymous |
| isFullyAuthenticated() | Returns true if the user is not an anonymous or a remember-me user |
| hasPermission(Object target, Object permission) | Returns true if the user has access to the provided target for the given permission. For example, hasPermission(domainObject, 'read') |
| hasPermission(Object targetId, String targetType, Object permission) | Returns true if the user has access to the provided target for the given permission. For example,hasPermission(1, 'com.example.domain.Message', 'read') |

Standalone example is given below.

# **Project Structure**



# **Maven (pom.xml)**

<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>springsecuritystandalone1</**groupId**>  
 <**artifactId**>springsecuritystandalone1</**artifactId**>  
 <**version**>0.0.1-SNAPSHOT</**version**>  
 <**packaging**>jar</**packaging**>  
  
 <**name**>springsecuritystandalone1</**name**>  
 <**url**>http://maven.apache.org</**url**>  
  
 <**properties**>  
 <**project.build.sourceEncoding**>UTF-8</**project.build.sourceEncoding**>  
 <**spring.version**>3.2.8.RELEASE</**spring.version**>  
 <**spring.security.version**>3.2.3.RELEASE</**spring.security.version**>  
 </**properties**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>junit</**groupId**>  
 <**artifactId**>junit</**artifactId**>  
 <**version**>4.0</**version**>  
 <**scope**>test</**scope**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-core</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-web</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-jdbc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework</**groupId**>  
 <**artifactId**>spring-webmvc</**artifactId**>  
 <**version**>${spring.version}</**version**>  
 </**dependency**>  
  
 *<!-- Spring Security -->* <**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-web</**artifactId**>  
 <**version**>${spring.security.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-config</**artifactId**>  
 <**version**>${spring.security.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>org.springframework.security</**groupId**>  
 <**artifactId**>spring-security-taglibs</**artifactId**>  
 <**version**>${spring.security.version}</**version**>  
 </**dependency**>  
  
 <**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.30</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

# **Spring IOC Configuration(beans1.xml)**

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xmlns:c="http://www.springframework.org/schema/c"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:security="http://www.springframework.org/schema/security"  
 xmlns:util="http://www.springframework.org/schema/util"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd  
 http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd  
 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd  
 http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security-3.2.xsd  
 http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd"**>  
  
 <**security:global-method-security secured-annotations="enabled" pre-post-annotations="enabled"**/>  
  
 <**bean id="bankTxn" class="com.ddlab.rnd.spring.security.BankTransactionImpl"**/>  
</**beans**>

# **Java Code**

## **BankTransaction.java**

**package** com.ddlab.rnd.spring.security;  
**import** org.springframework.security.access.annotation.Secured;  
**import** org.springframework.security.access.prepost.PreAuthorize;  
**public interface** BankTransaction {  
   
 @PreAuthorize (**"hasRole('ROLE\_MANAGER')"**)  
 **public void** approvePersonalLoan(String actNo);  
  
 @PreAuthorize(**"hasRole('ROLE\_MANAGER') or hasRole('ROLE\_ADMIN')"**)  
 **public void** approveHouseBuildingLoan( **int** amount);  
   
 @Secured(**"ROLE\_USER"**)  
 **public void** openAccount(String userName);  
   
 @PreAuthorize (**"hasRole('ROLE\_ADMIN')"**)  
 **public void** resetNetBankingPassword(String userName);  
   
 @PreAuthorize (**"hasAnyRole('ROLE\_MANAGER,ROLE\_TELLER')"**)  
 **public void** checkAccountStatus(String actNo) ;  
  
 @PreAuthorize(**"#permission.message == 'write' and hasRole('ROLE\_ADMIN')"**)  
 **public void** editAccount(String actNo, Permission permission);  
  
 @PreAuthorize(**" hasAuthority('ADMIN') "**)  
 **public void** disburseLoan();  
  
 @PreAuthorize(**"isFullyAuthenticated()"**)  
 **public void** applyCreditCard();  
}

## **BankTransactionImpl.java**

**package** com.ddlab.rnd.spring.security;  
  
**public class** BankTransactionImpl **implements** BankTransaction {  
  
 **public void** approvePersonalLoan(String actNo) {  
 System.***out***.println(**"The personal loan has been approved for the account no : "**+actNo);  
 }  
  
 **public void** openAccount(String userName) {  
 System.***out***.println(**"Hi "**+userName+**" , your account will be activate within 24 hours"**);  
   
 }  
  
 **public void** resetNetBankingPassword(String userName) {  
 System.***out***.println(**"Hi "**+userName+**", Administrator has rest your internet banking password "** + **"and an email has been sent you"**);  
 }  
  
 **public void** checkAccountStatus(String actNo) {  
 System.***out***.println(**"You available balance is 2000 INR"**);  
 }  
  
 **public void** approveHouseBuildingLoan(**int** amount) {  
 System.***out***.println(**"House building loan approved ..."**);  
 }  
  
 **public void** editAccount(String actNo , Permission permission) {  
 System.***out***.println(**"Your account has been edited and modified ..."**);  
 }  
  
 **public void** disburseLoan() {  
 System.***out***.println(**"Loan amount disbursed ..."**);  
 }  
  
 **public void** applyCreditCard() {  
 System.***out***.println(**"you have applied for credit card ..."**);  
 }  
}

## **Permission.java**

**package** com.ddlab.rnd.spring.security;  
  
**public class** Permission {  
  
 **private** String **message**;  
  
 **public** Permission(String message) {  
 **this**.**message** = message;  
 }  
  
 **public** String getMessage() {  
 **return message**;  
 }  
}

## **SecurityUtil.java**

**package** com.ddlab.rnd.spring.security;  
  
**import** java.util.List;  
**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
**import** org.springframework.security.core.Authentication;  
**import** org.springframework.security.core.authority.SimpleGrantedAuthority;  
**import** org.springframework.security.core.context.SecurityContextHolder;  
**import** org.springframework.security.core.userdetails.User;  
  
**public class** SecurityUtil {  
  
 **public static void** loginAs(String userName, String password, String roleType) {  
 *setSecurity*(userName, password, roleType);  
 }  
  
 **private static void** setSecurity(String userName, String password, String roleType) {  
 List<SimpleGrantedAuthority> auths = **new** java.util.ArrayList<SimpleGrantedAuthority>();  
 auths.add(**new** SimpleGrantedAuthority(roleType));  
 User user = **new** User(userName, password, auths);  
 Authentication authToken = **new** UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword(), user.getAuthorities());  
 SecurityContextHolder.*getContext*().setAuthentication(authToken);  
 }  
  
 **public static void** removeUser() {  
 SecurityContextHolder.*getContext*().setAuthentication(**null**);  
 }  
  
}

## **App.java**

**package** com.ddlab.rnd.spring.security;  
**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.support.ClassPathXmlApplicationContext;  
  
**public class** App {  
 *//Only Manager or Admin can approve loan* **public static void** approveLoan(BankTransaction bankTxn) {  
 SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_MANAGER"**);  
*// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_ADMIN");//It will not work* bankTxn.approvePersonalLoan(**"12345"**);  
 }  
  
 *//Only Manager or Admin can approve house building loan* **public static void** approveHouseLoan(BankTransaction bankTxn) {  
*// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_ADMIN");* SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_MANAGER"**);  
 bankTxn.approveHouseBuildingLoan(1234);  
 }  
  
 *//Only user can open account* **public static void** openAccount(BankTransaction bankTxn) {  
 SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_USER"**);  
 bankTxn.openAccount(**"Deb"**);  
 }

*//Only MANAGER or TELLER can check the status* **public static void** checkAccountStatus(BankTransaction bankTxn) {  
*// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_TELLER");* SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_MANAGER"**);  
 bankTxn.checkAccountStatus(**"11111"**);  
 }  
  
 *//Only ADMIN can reset password* **public static void** resetPassword(BankTransaction bankTxn) {  
 SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_ADMIN"**);  
 bankTxn.resetNetBankingPassword(**"Piku"**);  
 }  
  
 *//Only admin who has write permission can edit the account* **public static void** editAccount(BankTransaction bankTxn) {  
 SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_ADMIN"**);  
 bankTxn.editAccount(**"12345"**, **new** Permission(**"write"**));  
 }  
  
 *//Person having ADMIN authority can disburse loan* **public static void** disburseLoan(BankTransaction bankTxn) {  
 SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ADMIN"**);  
 bankTxn.disburseLoan();  
 }  
  
 *//Anybody properly authenticated can apply for credit card  
 //If we remove the user and invoke the method, it will throw exception* **public static void** applyCreditCard(BankTransaction bankTxn) {  
*// SecurityUtil.loginAs("Deb", "pqrs", "ADMIN");  
// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_MANAGER");  
// SecurityUtil.removeUser();//user removed, the following will throw exception  
// bankTxn.applyCreditCard();* SecurityUtil.*loginAs*(**"Deb"**, **"pqrs"**, **"ROLE\_MANAGER"**);  
 bankTxn.applyCreditCard();  
 }  
  
 **public static void** main(String[] args) {  
 ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext(**"beans1.xml"**);  
 BankTransaction bankTxn = (BankTransaction) applicationContext.getBean(**"bankTxn"**);  
 *approveLoan*(bankTxn);  
 *approveHouseLoan*(bankTxn);  
 *openAccount*(bankTxn);  
 *checkAccountStatus*(bankTxn);  
 *resetPassword*(bankTxn);  
 *editAccount*(bankTxn);  
 *disburseLoan*(bankTxn);  
 *applyCreditCard*(bankTxn);  
 }  
}

**Difference between @Secured and @PreAuthorize in spring security 3?**

<http://stackoverflow.com/questions/3785706/whats-the-difference-between-secured-and-preauthorize-in-spring-security-3>

Simply, @PreAuthorize is newer than @Secured.

So I say it is better to use @PreAuthorize as it is "expression-based" and you can use expressions like hasRole, hasAnyRole, permitAll, etc.

If you wanted to do something like access the method only if the user has Role1 and Role2 the you would have to use @PreAuthorize

@PreAuthorize("hasRole('ROLE\_role1') and hasRole('ROLE\_role2')")

Using

@Secured({"role1", "role2"}) is treated as an OR

@PreAuthorize allows you to get a more fine-grained control on the rules to secure o method. You can use SpEL expression inside of it.

Securing a method with @Secured gives you the same result as @PreAuthorize but the @Secured is limited and you don't get as much options to tweak the rules (a gross simplification it's that the rules are "static").

@PreAuthorizeis a newer version, so you should always go with @PreAuthorize, which is indeed better for the reasons mentioned [here](http://stackoverflow.com/questions/3785706/whats-the-difference-between-secured-and-preauthorize-in-spring-security-3).

And in fact

@Secured("ROLE\_ADMIN") is identical to @PreAuthorize("hasRole('ROLE\_ADMIN')")

In addition, @PreAuthorize syntax is more readable.

e.g. @Secured({"ROLE\_USER", "ROLE\_ADMIN"") is treated as ROLE\_USER or ROLE\_ADMIN, which is something weird and confusing.

On the other side with @PreAuthorize you define explicitly or, and expressions, which is obviously convenient and more readable.

So go with @PreAuthorize.

@Secured and @RolesAllowed are the same the only difference is @RolesAllowed is a standard annotation (i.e. not only spring security) whereas @Secured is spring security only.  
  
@PreAuthorize is different in a way that it is more powerful then the other 2. It allows for SpEL expression for a more fine-grained control. Which to use well the simplest thing that could possible work, if you don't need expression etc. go with the standard annotations to limit the dependency on spring classes. But that all is IMHO of course.  
  
I suggest a read of the reference guide because all is explained in there.