Controlling access

AccessDecisionVoter provides 3 possible outcomes for accessing the secure resources:
- **ACCESS_GRANTED**
- **ACCESS_DENIED**
- **ACCESS_ABSTAIN**

RoleVoter is an implementation of AccessDecisionManager provided by Spring.

Secured resource must have a configuration attribute whose name starts with `Role_` prefixed with all of the authorities granted to authenticated user.

If RoleVoter finds a match, it will vote ACCESS_GRANTED, otherwise, will vote ACCESS_DENIED.

RoleVoter will abstain from voting when the authorities required for access are not prefixed with `Role_`.

If all the voters abstain, by default access decision manager denies access.

You can override this by changing the `allowIfAllAbstain` property on the access decision manager to `true`.

Securing method invocations

As expected, AOP is used to enforce secure access to methods. See line 258.

MethodSecurityInterceptor determines if the user has been authenticated and if it can call the method.

If positive outcome, the method call will be invoked.
If not:
- **AuthenticationException**: user cannot be authenticated.
- **AccessDeniedException**: user hasn’t been granted authority to make the call.
Preferences

You have seen:

Preferences preferences =
Preferences.userNodeForPackage(Main.class);
What is a preference? java.util.prefs.
A method of storing and retrieving user
preferences that persist across application
invocations.
API: the preferences are "remembered" from
one run of an application to the next

... automatically maintains separate preference
lists for multiple users,
transparently handles storing the preferences
information
How? Uses registry and hidden files to "node in
a hierarchical collection of preference data"
Obtain Preferences using static method
userNodeForPackage()
one parameter class object

Preferences

Preferences preferences =
Preferences.userNodeForPackage(Main.class);

or

Preferences.userRoot().node(this.getClassName()).get("principal");

Stored variable can be retrieved next time by
get, getBoolean, getInt,...

String username = preferences.get("principal", "")

Preferences objects hold data as key/value:
To store a preferences item invoke put():
preferences.put("principal", ");
Why preferences are used in the FMC code?
To simulate the servlet session
What is that?

Authentication

In your sample code u see (what is the story?):
Authentication token = null;
boolean userAuthenticated = false;
...

token = new
UsernamePasswordAuthenticationToken(username, password);
userAuthenticated =
fmcService.authenticate(token);

Authentication an interface that contains:
• identity of the principal
• its credentials and
• GrantedAuthority (an array of them).
GrantedAuthority has various implementations,
we use GrantedAuthorityImpl that assign a
string that represents the authority of the
principal (for example "Role_anonymous")
Authentication

The are a number of authentication providers for processing for example DaoAuthenticationProvider (for hibernate) provides AuthenticationDao. Which we wire in our bean (see below)

```
You go further to see, what does it say?
if (!userAuthenticated)
{
    token = new AnonymousAuthenticationToken(
        new GrantedAuthority[]{
            new GrantedAuthorityImpl("Role_anonymous")
        }
    );
```

Storing authentication object

You see the code
Authentication auth = SecurityContextHolder.getContext().getAuthentication();

What is the above? accessing the populated Authentication object that applies to the current principal. Access is via an implementation of ContextHolder (here SecurityContextHolder)...

Storing authentication object

You see the code
SecureContext provides a mutator and accessor for the Authentication object

ContextHolder is set with a correct Authentication for a necessary duration (for example HTTP servlet request)

What about our example?

Preferences, as long as they are kept.

Wiring and assigning role to methods

Within MethodSecurityInterceptor wire:
1. authenticationManager
2. accessDecisionManager and
3. objectDefinitionSource with the values of the form

FMC.usecases.FMCService.createDB=Role_anonymous
FMC.usecases.FMCService.getCustomerByEmail=Role_user,Role_admin

See FMCSpring.xml bean with id="FMCServiceSecurity"
This wiring lets you authenticate

1. Use authenticationManager by passing token:Authentication
2. Set authentication on the context

```
public boolean authenticate(Authentication token) {
    Authentication auth = null;
    ...
    auth = authenticationManager.authenticate(token);
```

// if everything ok
if (auth != null)
{
    SecurityContextHolder.getContext().setAuthentication(auth);
}

daoAuthenticationProvider

Makes use of four beans (see the class diag.):

```
<bean id="daoAuthenticationProvider" ..
<property name="userDetailsService"
    ref="customerDAO"/>
<property name="userCache"
    ref="userCache"/>
<property name="passwordEncoder"
    ref="passwordEncoder"/>
<property name="saltSource">
```

daoAuthenticationProvider

For example in password encoder

```
<bean id="passwordEncoder"
    class="org.acegisecurity.provide rs.encoding.ShaPasswordEncoder"/>
```

How to decide to allow access?

Acegi intercepts access to an object or a webpage, but what if there are multiple factors involved in decision?

Example:
Patient: Patient
AccessPatientsDetails(r:Role,e:emergencylevel):Boolean
Complex logic: different principals, password… level of emergency

Voting

User can implement their own AccessDecisionManager
to control all aspects of authorisation.
But, spring provides a number of implementations:
What does this mean?

```xml
<bean id="accessDecisionManager" class="org.acegisecurity.vote.AffirmativeBased">
  <property name="allowIfAllAbstainDecisions" value="false"/>
  <property name="decisionVoters">
    <list>
      <ref local="roleVoter"/>
    </list>
  </property>
</bean>
```

Deciding on votes

three AccessDecisionManager:
1. ConsensusBased:
   grant or deny access based on the consensus of non-abstain votes.
2. AffirmativeBased
   grant access if one or more ACCESS_GRANTED were received (i.e. there was at least one grant vote).
3. UnanimousBased
   grants if provider expects unanimous ACCESS_GRANTED ignoring abstains deny access if there is any ACCESS_DENIED vote).

Logout: cleaning job 😊

```java
private static void logout(FMCService fmcservice, Preferences preferences){
  preferences.put("principal", "");
  preferences.put("password", "");
  Authentication tok = new AnonymousAuthenticationToken("anonymousKey","anonymous", new
  GrantedAuthority[]{new GrantedAuthorityImpl("Role_anonymous")});
```

A few words on web access

carried out by setting filters (implementation of javax.servlet.Filter)
We specify patterns of access and the role that can access Create an XML access file in WEB-INF and add pattern and role
```xml
<http auto-config="true">
  <intercept-url pattern="/" access="ROLE_USER"/>
</http>
```
Implementation by adding filters to intercept Request Response Picture