Internet Computing Workshop

Part 3 (continuation):
Spring security

Announcements
Thursday 4th March 2010 at 16:00 UG40,
School of Computer Science Domain Driven Design using Naked Objects Dan Haywood
Haywood Associates Ltd
Assessment for Spring

Recap
- Why Spring?
  - Description of the problem
- Overall view
- Inspecting a sample of an Spring-Hibernate Application → List of Questions!
- Detailed study of Spring:
  - Dependency Injection
  - Beans and their wiring
  - Aspect oriented Programming
  - Interaction with db
  - Transactions
  - Security

Spring security

Contents
What is Acegi?
an overview
Sample code
Authentication Manager
Controlling Access

“Spring Security” also known as Acegi
What is Acegi?
a Spring-based declarative security framework for handling authentication and authorisation, at both the web request level and at the method invocation level
You can declare “who” invokes “what”!
Acegi uses dependency injection and AoP
Why is it called Acegi?

An overview of Acegi

Security Interceptor: intercept access to resources to enforce security AND applies security rules.
You don’t directly use Security Interceptor as it delegates the task to one of these:
- Authentication Manger
- Access Decision Manager
- Run-as Manager
- After-invocation Manager
An overview of Acegi

**Authentication Manager** determines who you are:
- principal (e.g., a username/email)
- Credentials (e.g., password).

**Access Decision Manager** decide if you are authorized to access a resource (after passing through Authentication Manager).

After passing through the previous two managers, **Run-as Manager** is used to manage your authentication and access while accessing the resources.

For example: you may be allowed to view (read access) but not modify the contents (write access).

**After-invocation Manager** enforces security after the access.

For example altering the returned values so that the user is only able to access certain properties of the returned object.

---

Extension of the previous example to include security:

- **ehcache.xml** -
- **log4j.properties** -
- **Shop-spring-hib.xml** //setting for security
  - shop/
    - dao/ RoleDAO added
    - model/ Role...
    - usecases/ Modified
    - Main.java Modified

**AuthenticationManager** requires implementing a single method `authenticate()`.

If successful returns an `org.acegisecurity.Authentication` object.

If authentication fails an `AuthenticationException` will be thrown.

We often use **ProviderManager**, an implementation of `AuthenticationManager`.

**ProviderManager** has a list of authentication providers through `providers` property—out of many, we use two (see code in previous slide):
- DaoAuthenticationProvider:
- AnonymousAuthenticationProvider:
  authenticates a user as an anonymous user.
DaoAuthenticationProvider

1. retrieves user info such as username and password from a database
2. DaoAuthenticationProvider performs authentication by comparing the username/password retrieved from the database with the principal and credentials passed in an Authentication object from the authentication manager

Outcome: Authentication object or AuthenticationException.

so you must create DAO interfaces and their implementations

1. Observe DAO objects for Role
2. Look at lines 214-222
3. Look at line 212
Next we look at them in details.

encrypted passwords

Suppose that we want to store the password encrypted
We (application) don’t want to know the users passwd, so user-provided password must also be encrypted.
There are various options:
• encoding.PlaintextPasswordEncoder
No encryption (default)
• encoding.ShaPasswordEncoder
Does Secure Hash Algorithm (SHA) on the password
Similarly encoding.Md5PasswordEncoder

What is a salt?
Two salt sources available for encryption are:
• SystemWideSaltSource
Provides the same salt for all users
• ReflectionSaltSource
Encoded each users data using a different salt value (better security)
Also to enable caching UserCache property must be declared.

Controlling access

After authentication Access Decision Manager decides if a user has required privileges to access secured resources.
(org.acegisecurity .AccessDecisionManager)
Behind the scene a voting mechanism is used:
AccessDecisionManager uses an AccessDecisionVoter to decide, on the basis of user’s authorities and configuration required by the resource (Role assignments) to give access or to deny or to abstain vote.
See lines 275-291 for sample of Conf's

Strategies for tallying the votes:
1. AffirmativeBased
Allows access if at least one voter votes to grant access
2. ConsensusBased
Allows access if a consensus of voters vote to grant access
3. UnanimousBased
Allows access if all voters vote to grant access
In the example AffirmativeBased (Line 246)
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_

See Lines 242-250

Controlling access

RoleVoter decides on its vote by comparing all configuration attributes which are prefixed with Role_ 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_.

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

Finally, Controlling access for the weblayer is done through a set of Filters, we will study this in Wicket.