Spring

Sesión 7: Seguridad con Spring Security

Indice

- Configuración básica
- Autentificación contra una base de datos
- Personalización de la seguridad web
- Seguridad de la capa de negocio

Spring Security

- Subproyecto de Spring (no es parte del core framework)
- ¿Por qué esto si ya la seguridad está implementada en JavaEE estándar?
  - Ciertos mecanismos están estandarizados, pero otros dependen del servidor de aplicaciones, básicamente el enlace entre la aplicación y las “fuentes de autorización” (BD, LDAP, certificados)
  - Una aplicación con Spring Security es más portable que una estándar (!)

Configuración mínima para aplicación web

- Añadir las dependencias de Spring Security
- Modificar el web.xml para que Spring Security intercepte las peticiones a la aplicación y pueda controlar el acceso
- Crear un fichero de configuración XML para la seguridad
- Indicar en el web.xml dónde está este archivo

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
  <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>3.1.3.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>3.1.3.RELEASE</version>
</dependency>
```

Configuración mínima para aplicación web (II)

- Spring Security debe interceptar las peticiones para poder controlar el acceso
  - Se usan filtros de servlets

```
<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>
```

- Decirle a Spring dónde está el fichero de configuración de seguridad
  - Es un fichero de configuración de beans “estándar”

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
    /WEB-INF/spring/root-context.xml
    /WEB-INF/spring/security-context.xml
    </param-value>
</context-param>
<listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

Configuración mínima para aplicación web (III)

- Crear el fichero de configuración de seguridad
  - Es un fichero de configuración de beans típico de Spring, pero básicamente contendrá etiquetas de seguridad
  - También podríamos meter las etiquetas en el fichero de configuración "principal" de la aplicación

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-security.xsd">
  <http auto-config="true">
    <intercept-url pattern="/**" access="ROLE_USER" />
  </http>
  <authentication-manager alias="authenticationManager">
    <authentication-provider>
      <user-service>
        <user authorities="ROLE_USER" name="guest" password="guest" />
      </user-service>
    </authentication-provider>
  </authentication-manager>
</beans:beans>
```

Autentificación contra una base de datos

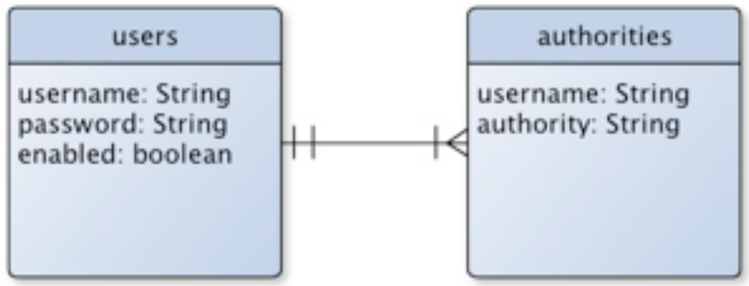
- Cambiar el *authentication manager*
- Esta es una versión inicial, solo sirve si se usa el esquema de BD por defecto (ahora veremos cuál es)

```
<authentication-manager alias="authenticationManager">
    <authentication-provider>
      <jdbc-user-service data-source-ref="miDataSource"/>
    </authentication-provider>
</authentication-manager>

<!-- datasource, como siempre que nos queremos conectar a una BD en Spring -->
<jee:jndi-lookup id="miDataSource" jndi-name="jdbc/securityDS" resource-ref="true"/>
```

Cómo obtiene spring usuarios/roles

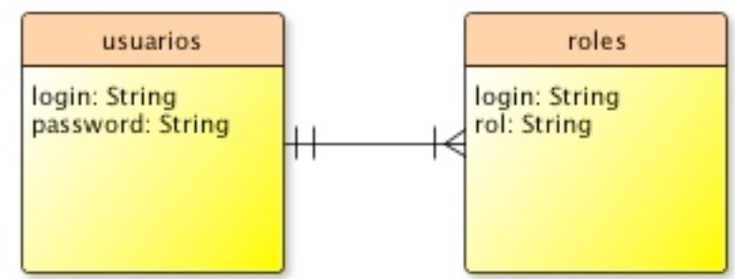
- Se supone un esquema de BD “por defecto” y se hacen 2 *queries* SQL, una para comprobar el password y otra para obtener los roles




```
SELECT username, password, enabled
  FROM users
  WHERE username = ?
```

```
SELECT username, authority
  FROM authorities
  WHERE username = ?
```

- Si tenemos otro esquema distinto, será cuestión de “montar” 2 queries que devuelvan los resultados con el formato que espera Spring




```
SELECT login as username, password, true as
enabled
  FROM usuarios
  WHERE login=?
```

```
SELECT login as username, rol as authority
   FROM roles
   WHERE login=?
```

En el fichero de configuración...

```
<authentication-manager alias="authenticationManager">
  <authentication-provider user-service-ref="miUserServiceJDBC" />
</authentication-manager>

<beans:bean id="miUserServiceJDBC"
  class="org.springframework.security.core.userdetails.jdbc.JdbcDaoImpl">
  <beans:property name="dataSource" ref="miDataSource"/>
  <beans:property name="usersByUsernameQuery"
                  value="SELECT login as username, password, true as enabled
                         FROM usuarios WHERE login=?"/>
  <beans:property name="authoritiesByUsernameQuery"
                  value="SELECT login as username, rol as authority
                         FROM roles WHERE login=?"/>
</beans:bean>

<jee:jndi-lookup id="miDataSource" jndi-name="jdbc/securityDS" resource-ref="true"/>
```

Personalización de la seguridad

- Con la configuración anterior, Spring nos da un formulario de login auto-generado
- Usar nuestro propio formulario de login
- Fichero de configuración
  - ¡Acordarse de desproteger el acceso al formulario!

```
<http pattern="/login.html" security="none"/>

<http>
  <intercept-url pattern="/**" access="ROLE_REGISTRADO, ROLE_ADMIN" />
  <form-login login-page="/login.html" default-target-url="/main.html" />
</http>
```

- HTML del formulario, muy similar al estándar

```
<form action="j_spring_security_check" method="post">
    Usuario: <input type="text" name="j_username"/> <br/>
    Contraseña: <input type="password" name="j_password"/> <br/>
    <input type="submit" value="Entrar"/>
</form>
```

  - En el estándar no se puede ir directamente a la URL del formulario, lo que nos obliga al “truco” de ir a una URL protegida para forzar el salto al formulario. Aquí no hace falta eso

Otras ”mejoras”

- Logout

```
<http>
    ...
    <logout logout-url="/logout" logout-success-url="/adios.jsp"/>
</http>
```

```
<!--borrará la sesión y saltará a “adios.jsp” -->
<a href="logout">Logout</a>
```

- Remember-me: no tenemos que hacer login cada vez que accedamos, se guarda en el navegador
  - No se guarda login/password, sino un *token* de acceso (*hash* md5 a partir de login,password,clave de la aplicacion, fecha expiración)

```
<http>
     ...
     <remember-me key="claveDeLaAplicacion"/>
</http>
```

```
<!-- añadir este campo al formulario de login -->
<input type="checkbox" name="_spring_security_remember_me"/>
          Recordarme en este ordenador
```

Autentificación basic vs. formulatio

- BASIC es apropiado para clientes REST/Acceso remoto

```
<!-- servicios REST sin estado con autentificación Basic -->
<http pattern="/restful/**" create-session="stateless">
  <intercept-url pattern='/**' access='ROLE_REMOTE' />
  <http-basic />
</http>

<!-- Desproteger la página de login-->
<http pattern="/login.html" security="none"/>

<!-- Clientes web con autentificación basada en formulario -->
<http>
  <intercept-url pattern='/**' access='ROLE_USER' />
  <form-login login-page='/login.html' default-target-url="/home.html"/>
  <logout />
</http>
```

Seguridad de “grano fino” en los JSP

- Dependencia de una librería adicional

```
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-taglibs</artifactId>
	<version>3.1.3.RELEASE</version>
</dependency>
```

- Tomamos una URL “de referencia”. Si el usuario actual puede acceder a ella, el contenido del tag se mostrará

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
...
<sec:authorize url="/admin/eliminar">
  <a href="/admin/eliminar">Eliminar</a>
</sec:authorize>
```

Seguridad de “grano fino” en los JSP

- También se pueden usar expresiones SpEL
  - hasRole(rol)
  - hasAnyRole(rol1,rol2,...)
  - isFullyAuthenticated() no ha entrado con el remember-me
  - hasIpAddress(dir)

```
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
...
<sec:authorize access="hasRole('ROLE_ADMIN') and hasIpAddress('127.0.0.1')">
  <p>Esto solo lo debería ver un admin conectado localmente</p>
</sec:authorize>
```

- Hay que activar soporte de SpEL para seguridad
  - Activarlo es fácil (use-expressions="true"), el “problema” es que entonces lo debemos usar en todos los sitios, incluyendo el propio fichero de configuración

```
<http use-expressions="true">
    <!-- access antes era simplemente “ROLE_USER, ROLE_ADMIN” -->
    <intercept-url pattern="/**" access="hasAnyRole('ROLE_USER', 'ROLE_ADMIN')"/>
</http>
```

Seguridad de la capa de negocio

- Podemos proteger directamente cualquier método de cualquier clase
  - Apropiado para clientes remotos y paranoicos de la seguridad

- Dos tipos de soporte. Cuidado, hay que activarlo con global-method-security para que funcionen las anotaciones
  - Estándar (JSR250) - `@Secured`

```
<global-method-security jsr250-annotations="enabled"/>
```

  - Propio de Spring, más potente

```
<global-method-security pre-post-annotations="enabled"/>
```

- Podemos activar los dos simultáneamente si lo deseamos (poner los dos atributos)

Ejemplos de anotaciones Spring

- `@PreAuthorize`, `@PostAuthorize`
  - La primera es la más común, comprueba la condición SpEL antes de ejecutar el método
  - La segunda comprueba el valor de retorno del método, si no cumple la condición se genera una excepción
- Con # podemos referenciar los parámetros
- `principal` es el usuario autentificado

```
public interface IMensajeriaBO {
    ...
    @PreAuthorize("hasRole('ROLE_USER') and #u.credito>0")
    public void enviarMensaje(Usuario u, Mensaje m);
    ...
}
```

```
public interface IUsuarioBO {
    @PreAuthorize("#u.login == principal.username and hasRole('ROLE_USER')")
    public void cambiarPassword(Usuario u, String nuevoPassword);
}
```

Filtrar colecciones

- `@PostFilter`: para métodos que devuelvan colecciones
  - una vez llamado el método, y justo antes de retornar, elimina los elementos de la colección que no cumplan la condición SpEL especificada
  - `filterObject` representa el elemento actual de la colección (el que se está procesando y decidiendo si eliminar o no)

```
public interface IUsuarioBO {
...
<!-- si el usuario actual tiene rol ROLE_ROOT, podrá ver todos los usuarios,
     si tiene el rol ROLE_ADMIN y el usuario a devolver no es admin también lo
     podrá ver -->
<!-- si un usuario no tiene ninguno de los dos roles, no podrá ver nada -->
   @PostFilter("hasRole('ROLE_ROOT') or (hasRole('ROLE_ADMIN')
                  and !filterObject.isAdmin())")
   public List<Usuario> getUsuarios();
...
}
```

Seguridad en el XML de configuración

- Podemos especificar una condición que abarque múltiples métodos, en lugar de estar anotándolos uno a uno
- Se está usando AOP - Programación Orientada a Aspectos, consultar apéndice de los apuntes
- Ejemplos:
  - Todos los métodos del paquete es.ua.jtech.negocio cuyo nombre comience por listar
  - Todos los métodos de la clase UsuariosBO
  - Todos los métodos que devuelvan un objeto de tipo Usuario
  - Todos los métodos cuyo nombre comience por “set”, tengan un solo parámetro y devuelvan void (*juraría que esto en Java tiene un nombre...*)

```
<global-method-security>
    <protect-pointcut
                    expression="execution(* eliminar*(..))"
                    access="ROLE_ADMIN"/>
</global-method-security>
```

¿Preguntas...?