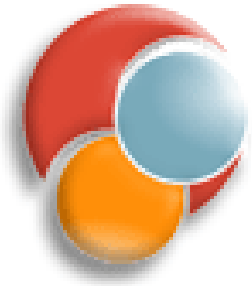


Spring

Sesión 6: Spring security

Indice

- **Introducción**
- Configuración mínima para aplicaciones web
- Autentificación contra la base de datos
- Configuración manual de aplicaciones web
  - Autentificación con formulario
  - Recordar datos del usuario
  - Logout
- Seguridad en la ejecución de código

Spring Security

- Un subproyecto bajo el paraguas de Spring
  - Antes se llamaba “Acegi Security”, lo cual implica que todo el código que antes estaba en **org.acegisecurity** ahora está en **org.springframework.security**
- Versión actual: 2.0
  - En versiones anteriores, para configurar la seguridad había que definir bastantes beans en el XML
  - En la 2.0 se incorporan etiquetas en el XML que generan automáticamente los beans necesarios. **La sintaxis para los casos comunes es mucho más concisa**

¿Qué se puede hacer?

- **Autentificación**: multitud de proveedores de autentificación implementados
  - DAO (datos almacenados en una BD)
  - LDAP
  - X509
  - Single sign on (CAS, OpenID,...)
- **Control de acceso**: una vez autentificados, controlar que tenemos permiso para acceder a un recurso
  - Aplicaciones web: por URL
  - Por método, como en EJB
- **Seguridad en el canal** (HTTPS, ...)

¿Y la seguridad declarativa “estándar” JavaEE?

- Como habéis visto, no es tan estándar. Cada servidor de aplicaciones implementa ciertas cosas a su manera
  - Sobre todo dónde se almacenan y cómo se obtienen las credenciales de los usuarios
  - Nosotros tendremos código totalmente portable, siempre que incluyamos los JAR de Spring
- Spring security es más potente y flexible que la seguridad declarativa estándar

Indice

- Introducción
- **Configuración mínima para aplicaciones web**
- Autentificación contra la base de datos
- Configuración manual de aplicaciones web
  - Autentificación con formulario
  - Recordar datos del usuario
  - Logout
- Seguridad en la ejecución de código

Configuración en el web.xml

- Se usan de manera intensiva **filtros** de servlets. A nosotros nos basta con definir uno que hace de puente entre Spring y los servlets
  - El nombre springSecurityFilterChain debe ser “tal cual”

```
<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>
```

Configuración mínima para aplic. web

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/security
        http://www.springframework.org/schema/security/spring-security-2.0.xsd">
```

Control de acceso

```
  <security:http auto-config="true">
     <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN"/>
     <security:intercept-url pattern="/**" access="ROLE_USER,ROLE_ADMIN"/>
  </security:http>
```

Autentificación

```
  <security:authentication-provider >
     <security:user-service>
        <security:user name="spring" password="spring" authorities="ROLE_USER" />
        <security:user name="admin" password="admin" authorities="ROLE_ADMIN" />
     </security:user-service>
  </security:authentication-provider>
</beans>
```

Etiqueta <http/>

- Seguridad para aplicaciones web
- con **auto-config=true** proporciona automáticamente una serie de servicios
  - Autentificación por formulario autogenerado por Spring
  - Autentificación BASIC
  - “remember me”
  - Logout (borrado de sesión automático)

Etiqueta <intercept-url/>

- Recurso a proteger. Paths al estilo de ant.
- Al comprobar acceso a un recurso, se aplica el primer “intercept-url” que encaja. Por tanto, debemos colocar primero las más específicas

```
<security:http auto-config="true">
    <security:intercept-url pattern="/admin/**"
                            access="ROLE_ADMIN"/>
    <security:intercept-url pattern="/**"
                            access="ROLE_USER,ROLE_ADMIN"/>
</security:http>
```

Autentificación

- **Proveedor de autentificación** (<authentication-provider/>): nos concede o no una “authority” (un rol) en función de nuestro “principal” (login) y “credenciales” (normalmente password)
- <user-service/>: donde se almacenan las credenciales
  - Meterlas en el XML directamente es útil para pruebas
  - Cuidado,los *authority* **deben** comenzar por ROLE_

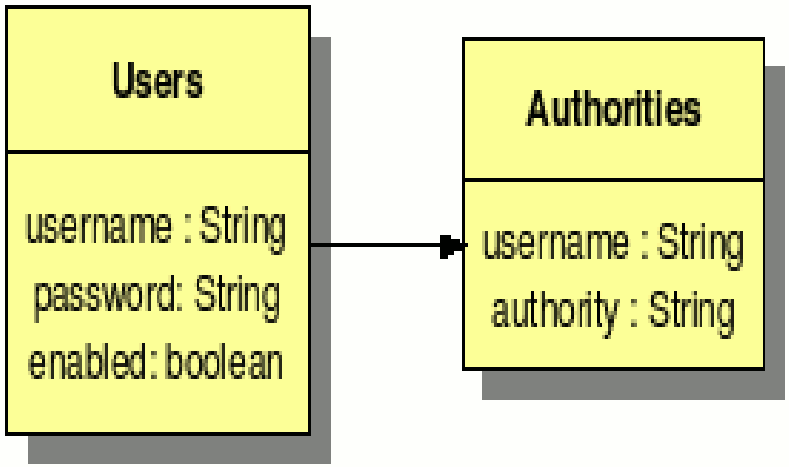
```
<security:authentication-provider >
    <security:user-service>
        <security:user name="spring" password="spring"
                       authorities="ROLE_USER" />
        <security:user name="admin" password="admin"
                       authorities="ROLE_ADMIN" />
    </security:user-service>
</security:authentication-provider>
```

Indice

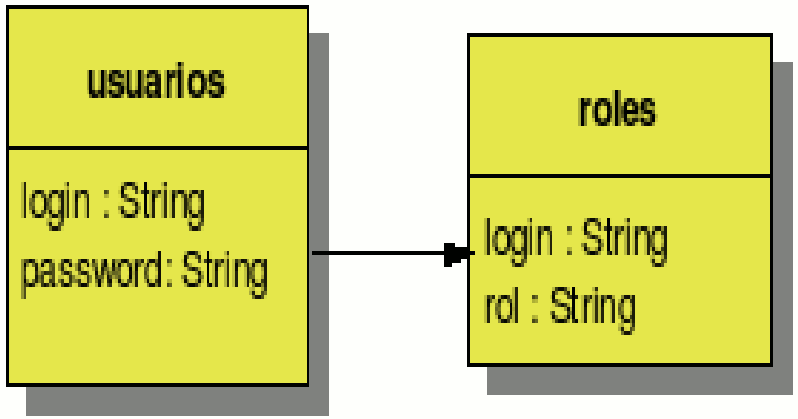
- Introducción
- Configuración mínima para aplicaciones web
- **Autentificación contra la base de datos**
- Configuración manual de aplicaciones web
  - Autentificación con formulario
  - Recordar datos del usuario
  - Logout
- Seguridad en la ejecución de código

Autentificación contra la BD

- La clase **JdbcDaoImpl** de Spring realiza el trabajo, no hace falta implementar una propia
- Pero supone que la BD tiene un determinado schema que habrá que adaptar al nuestro



Supuesto por JdbcDaoImpl



Nuestra BD

Adaptación de JdbcDaoImpl

- La clave está en el SQL que se usa para
  - **Obtener el password** de un usuario

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

  - **Obtener los authorities**

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

- Simplemente debemos cambiar el SQL por otro que funcione con nuestra BD y devuelva los campos que espera JdbcDaoImpl

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

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

Uso de JdbcDaoImpl

- Prop. “usersByUserNameQuery”: SQL para password
- Prop. “authoritiesByUserNameQuery”: SQL para roles

```
<jee:jndi-lookup id="miDataSource" jndi-name="jdbc/MiDataSource"
                                resource-ref="true"/>
<security:authentication-provider user-service-ref="miJDBCUserDetails"/>

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

Indice

- Introducción
- Configuración mínima para aplicaciones web
- Autentificación contra la base de datos
- **Configuración manual de aplicaciones web**
  - Autentificación con formulario
  - Recordar datos del usuario
  - Logout
- Seguridad en la ejecución de código

Autentificación con formulario

```
<security:http auto-config="true">
   <security:intercept-url pattern="/index.jsp" filters="none"/>
   <security:intercept-url pattern="/**"
                        access="ROLE_REGISTRADO, ROLE_ADMIN"/>
    <security:form-login login-page="/index.jsp"
                     default-target-url="/principal.jsp" />
</security:http>
```

- El formulario estará en “login-page” y al hacer el login saltaremos automáticamente a “default-target-url”
- Debemos desproteger la página que contiene el formulario (filters=”none”)

Formulario de autentificación

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

- Los campos y el ”action” deben tener ciertos nombres,como en la seguridad estándar.
- A diferencia de la seguridad estándar, no causa problemas el ir directamente a la página de login

Recordar los datos de login

- Mecanismo activado por defecto con auto-config=true
  - Pero al hacer login hay que especificar que queremos usarlo
- Se implementa con una cookie. Únicamente necesitamos un campo del formulario para decir si queremos crearla o no

```
<form action="j_spring_security_check">
   login: <input type="text" name="j_username"/> <br/>
   password: <input type="text" name="j_password"/> <br/>
   <input type="checkbox"
            name="_spring_security_remember_me"/>
             Recordar mi usuario y password <br/>
   <input type="submit" value="Entrar"/>
</form>
```

Logout

- Borra la sesión de manera automática y salta a la página de “salida” (por defecto, “/”)
- Para dispararlo, hay que llamar a la URL “**/j_spring_security_logout**”, aunque es configurable
  - logout-url: URL que dispara el logout
  - logout-success-url: a dónde se salta tras el logout

```
<security:http auto-config="true">
  ...
  <security:logout logout-url="/logout.jsp"
                   logout-success-url="/adios.jsp"/>
</security:http>
```

Indice

- Introducción
- Configuración mínima para aplicaciones web
- Autentificación contra la base de datos
- Configuración manual de aplicaciones web
  - Autentificación con formulario
  - Recordar datos del usuario
  - Logout
- **Seguridad en la ejecución de código**

Seguridad en ejecución de código

- Comprobar la “authority” antes de ejecutar cierto/s método/s. Se controla con la etiqueta <global-method-security/>
- Formas de configurarlo
  - **Con AOP en el XML**: podemos cambiar la seguridad sin tocar para nada el código
  - **Con anotaciones en el fuente:** los permisos están dentro del propio fuente, más claro para ciertas situaciones
- Independientemente de cómo lo configuremos, Spring lo implementa internamente mediante AOP

Seguridad de código con AOP

- Se coloca un pointcut con sintaxis AspectJ en el XML, junto con los “authorities” que pueden “traspasarlo”

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

Seguridad de código con anotaciones

- Aunque Spring tiene una anotación propia, @Secured, soporta las de JSR-250, es decir, las de EJB 3.0 (@RolesAllowed,...)
  - Debemos habilitar el soporte de anotaciones JSR250 en el XML. Si quisiéramos habilitar también @Secured, pondríamos también secured-annotations="enabled"

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

```
@RolesAllowed("ROLE_ADMIN")
public void eliminarUsuario {
    ...
}
```

¿Preguntas...?