

(/)

Spring Security - Roles y privilegios

Última modificación: 6 de noviembre de 2018

por Eugen Paraschiv (<https://www.baeldung.com/author/eugen/>)
(<https://www.baeldung.com/author/eugen/>)

Seguridad de primavera (<https://www.baeldung.com/category/spring/spring-security/>)

Autorización (<https://www.baeldung.com/tag/authorization/>)

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (</ls-course-start>)

1. Información general

Este artículo continúa la serie Registro con Spring Security con un vistazo a cómo implementar correctamente **Roles y Privilegios**.

Otras lecturas:

Introducción a las expresiones de seguridad de Spring
(<https://www.baeldung.com/spring-security-expressions>)

Guía simple y práctica de Spring Security Expressions.

Leer más (<https://www.baeldung.com/spring-security-expressions>) →

Introducción a la seguridad del método Spring
(<https://www.baeldung.com/spring-security-method-security>)

Una guía de seguridad a nivel de método utilizando el marco de Spring Security.

Leer más (<https://www.baeldung.com/spring-security-method-security>) →

Spring Security - Redireccionar a la URL anterior después de iniciar sesión
(<https://www.baeldung.com/spring-security-redirect-login>)

Un breve ejemplo de redirección después de iniciar sesión en Spring Security.

Leer más (<https://www.baeldung.com/spring-security-redirect-login>) →

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa](/privacy-policy/) (</privacy-policy/>)

2. Rol y privilegio del usuario

Primero, comencemos con nuestras entidades. Tenemos tres entidades principales:

- el *usuario*
- el *Rol*: representa los roles de alto nivel del usuario en el sistema; cada rol tendrá un conjunto de privilegios de bajo nivel
- el *Privilegio* - representa una de bajo nivel, privilegio granular / autoridad en el sistema de

Aquí está **el usuario**:

```

1  @Entity
2  public class User {
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.AUTO)
6      private Long id;
7
8      private String firstName;
9      private String lastName;
10     private String email;
11     private String password;
12     private boolean enabled;
13     private boolean tokenExpired;
14
15     @ManyToMany
16     @JoinTable(
17         name = "users_roles",
18         joinColumns = @JoinColumn(
19             name = "user_id", referencedColumnName = "id"),
20         inverseJoinColumns = @JoinColumn(
21             name = "role_id", referencedColumnName = "id"))
22     private Collection<Role> roles;
23 }

```

Como puede ver, el usuario contiene los roles, pero también algunos detalles adicionales que son necesarios para un mecanismo de registro adecuado.

Siguiente: aquí está **el papel**:

```

1  @Entity
2  public class Role {
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.AUTO)
6      private Long id;
7
8      private String name;
9      @ManyToMany(mappedBy = "roles")
10     private Collection<User> users;
11
12     @ManyToMany
13     @JoinTable(
14         name = "roles_privileges",
15         joinColumns = @JoinColumn(
16             name = "role_id", referencedColumnName = "id"),
17         inverseJoinColumns = @JoinColumn(
18             name = "privilege_id", referencedColumnName = "id"))
19     private Collection<Privilege> privileges;
20 }

```

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

Ok

Y finalmente **el privilegio** :

```
1  @Entity
2  public class Privilege {
3
4      @Id
5      @GeneratedValue(strategy = GenerationType.AUTO)
6      private Long id;
7
8      private String name;
9
10     @ManyToMany(mappedBy = "privileges")
11     private Collection<Role> roles;
12 }
```

Como puede ver, estamos considerando tanto el rol de usuario <-> como las relaciones de privilegio de rol <-> **bidireccional de muchos a muchos** .

3. Configurar privilegios y roles

A continuación, concentrémonos en hacer una configuración temprana de los privilegios y roles en el sistema.

Vincularemos esto al inicio de la aplicación y usaremos un *ApplicationListener* en *ContextRefreshedEvent* para cargar nuestros datos iniciales al iniciar el servidor:

```

1  @Component
2  public class InitialDataLoader implements
3      ApplicationListener<ContextRefreshedEvent> {
4
5      boolean alreadySetup = false;
6
7      @Autowired
8      private UserRepository userRepository;
9
10     @Autowired
11     private RoleRepository roleRepository;
12
13     @Autowired
14     private PrivilegeRepository privilegeRepository;
15
16     @Autowired
17     private PasswordEncoder passwordEncoder;
18
19     @Override
20     @Transactional
21     public void onApplicationEvent(ContextRefreshedEvent event) {
22
23         if (alreadySetup)
24             return;
25         Privilege readPrivilege
26             = createPrivilegeIfNotFound("READ_PRIVILEGE");
27         Privilege writePrivilege
28             = createPrivilegeIfNotFound("WRITE_PRIVILEGE");
29
30         List<Privilege> adminPrivileges = Arrays.asList(
31             readPrivilege, writePrivilege);
32         createRoleIfNotFound("ROLE_ADMIN", adminPrivileges);
33         createRoleIfNotFound("ROLE_USER", Arrays.asList(readPrivilege));
34
35         Role adminRole = roleRepository.findByName("ROLE_ADMIN");
36         User user = new User();
37         user.setFirstName("Test");
38         user.setLastName("Test");
39         user.setPassword(passwordEncoder.encode("test"));
40         user.setEmail("test@test.com");
41         user.setRoles(Arrays.asList(adminRole));
42         user.setEnabled(true);
43         userRepository.save(user);
44
45         alreadySetup = true;
46     }
47
48     @Transactional
49     private Privilege createPrivilegeIfNotFound(String name) {
50
51         Privilege privilege = privilegeRepository.findByName(name);
52         if (privilege == null) {
53             privilege = new Privilege(name);
54             privilegeRepository.save(privilege);
55         }
56         return privilege;
57     }
58
59     @Transactional
60     private Role createRoleIfNotFound(
61         String name, Collection<Privilege> privileges) {
62
63         Role role = roleRepository.findByName(name);
64         if (role == null) {
65             role = new Role(name);
66             role.setPrivileges(privileges);
67             roleRepository.save(role);
68         }
69         return role;
70     }
71 }

```

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy/\)](/privacy-policy/)

Entonces, ¿qué sucede durante este código de configuración simple? Nada complicado:

- estamos creando los privilegios
- estamos creando los roles y les asignamos los privilegios
- estamos creando un usuario y le asignamos un rol

Tenga en cuenta cómo estamos usando un indicador de **configuración previa** para **determinar si la configuración debe ejecutarse o no**. Esto se debe simplemente a que, según la cantidad de contextos que haya configurado en su aplicación, *ContextRefreshedEvent* puede *activarse* varias veces. Y solo queremos que la configuración se ejecute una vez.

Dos notas rápidas aquí: **primero, sobre terminología**. Estamos usando los términos *Privilegio - Rol* aquí, pero en primavera, estos son ligeramente diferentes. En primavera, nuestro Privilegio se conoce como Rol, y también como una autoridad (otorgada), lo cual es un poco confuso. No es un problema para la implementación, por supuesto, pero definitivamente vale la pena señalar.

Segundo: estos roles de primavera (nuestros privilegios) necesitan un prefijo; de manera predeterminada, ese prefijo es "ROLE", pero se puede cambiar. No estamos usando ese prefijo aquí, solo para simplificar las cosas, pero tenga en cuenta que si no lo está cambiando explícitamente, será necesario.

4. *UserDetailsService* personalizado

Ahora, veamos el proceso de autenticación.

Vamos a ver cómo recuperar al usuario dentro de nuestro *UserDetailsService* personalizado, y cómo asignar el conjunto correcto de autoridades de los roles y privilegios que el usuario ha asignado:

```
1 @Service("userDetailsService")
2 @Transactional
3 public class MyUserDetailsService implements UserDetailsService {
4
5     @Autowired
6     private UserRepository userRepository;
7
8     @Autowired
9     private IUserService service;
10
11     @Autowired
12     private MessageSource messages;
13
14     @Autowired
15     private RoleRepository roleRepository;
16
17     @Override
18     public UserDetails loadUserByUsername(String email)
19         throws UsernameNotFoundException {
20
21         User user = userRepository.findByEmail(email);
22         if (user == null) {
23             return new org.springframework.security.core.userdetails.User(
24                 " ", " ", true, true, true, true,
25                 getAuthorities(Arrays.asList(
26                     roleRepository.findByName("ROLE_USER"))));
27         }
28
29         return new org.springframework.security.core.userdetails.User(
30             user.getEmail(), user.getPassword(), user.isEnabled(), true, true,
31             true, getAuthorities(user.getRoles()));
32     }
33
34     private Collection<? extends GrantedAuthority> getAuthorities(
35         Collection<Role> roles) {
36
37         return getGrantedAuthorities(getPrivileges(roles));
38     }
39
40     private List<String> getPrivileges(Collection<Role> roles) {
41
42         List<String> privileges = new ArrayList<>();
43         List<Privilege> collection = new ArrayList<>();
44         for (Role role : roles) {
45             collection.addAll(role.getPrivileges());
46         }
47         for (Privilege item : collection) {
48             privileges.add(item.getName());
49         }
50         return privileges;
51     }
52
53     private List<GrantedAuthority> getGrantedAuthorities(List<String> privileges) {
54         List<GrantedAuthority> authorities = new ArrayList<>();
55         for (String privilege : privileges) {
56             authorities.add(new SimpleGrantedAuthority(privilege));
57         }
58         return authorities;
59     }
60 }
```

Lo interesante a seguir aquí es cómo se asignan los privilegios (y roles) a las entidades de GrantedAuthority.

Esta asignación hace que toda la configuración de seguridad sea **altamente flexible y potente**: puede mezclar y combinar roles y privilegios tan granulares como sea necesario, y al final, se asignarán correctamente a las autoridades y volverán al marco.

5. Registro de *usuario*

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy/\)](#)

Ok

Finalmente, echemos un vistazo al registro de un nuevo usuario.

Hemos visto cómo se realiza la configuración para crear el usuario y se le asignan roles (y privilegios); ahora veamos cómo debe hacerse esto durante el registro de un nuevo usuario:

```

1  @Override
2  public User registerNewUserAccount(UserDto accountDto) throws EmailExistsException {
3
4      if (emailExist(accountDto.getEmail())) {
5          throw new EmailExistsException
6              ("There is an account with that email adress: " + accountDto.getEmail());
7      }
8      User user = new User();
9
10     user.setFirstName(accountDto.getFirstName());
11     user.setLastName(accountDto.getLastName());
12     user.setPassword(passwordEncoder.encode(accountDto.getPassword()));
13     user.setEmail(accountDto.getEmail());
14
15     user.setRoles(Arrays.asList(roleRepository.findByName("ROLE_USER")));
16     return repository.save(user);
17 }

```

En esta implementación simple, suponemos que se está registrando un usuario estándar, por lo que se le *asigna el rol ROLE_USER*.

Por supuesto, una lógica más compleja se puede implementar fácilmente de la misma manera, ya sea al tener múltiples métodos de registro codificados o al permitir que el cliente envíe el tipo de usuario que se está registrando.

6. Conclusión

En este tutorial, ilustramos cómo implementar Roles y Privilegios con JPA, para un sistema respaldado por Spring Security.

La **implementación completa** de este tutorial de Registro con Spring Security se puede encontrar en el proyecto GitHub (<https://github.com/eugenp/spring-security-registration>); este es un proyecto basado en Maven, por lo que debería ser fácil de importar y ejecutar tal como está.

Acabo de anunciar el nuevo curso *Learn Spring*, centrado en los fundamentos de Spring 5 y Spring Boot 2:

>> VER EL CURSO (/ls-course-end)

Únet
de lo
129 €

▲ el más nuevo ▲ más antiguo ▲ más votado



jirkapinkas



¿Por qué no en lugar de implementar el método de creación ApplicationListener anotado con la anotación PostConstruct? Como el bean es singleton, funcionará de la misma manera, ¿verdad?

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

+ 0 -

Ok

🕒 Hace 4 años ▲



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)



Claro, @PostConstruct (o tal vez un InitializingBean) también funcionaría. La razón principal por la que asistí al evento fue que quería saltar una vez que todo el contexto se inicia, no solo ese bean en particular. Sin embargo, no es una gran diferencia, ya que solo estoy escribiendo algunos datos, así que sí, eso también estaría bien. Saludos, Eugen.

+ 0 -

Hace 4 años



Huésped

Arusland



El artículo estaría completo si se le mostrara un ejemplo de uso de privilegios.

+ 0 -

Hace 4 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)



Hola Arusland, gracias por los comentarios. Los privilegios se usan realmente, si observa MyUserDetailsService (getPrivileges), ahí es donde los privilegios se convierten en autoridades simples. Spring Security luego usa estas autoridades. Eso es todo lo que hacen los privilegios realmente: son autoridades para ser utilizadas. Espero que aclare las cosas. Saludos, Eugen.

+ 0 -

Hace 4 años



Huésped

Arusland



OK gracias)

+ 0 -

Hace 4 años



Huésped

aelix



Artículo maravilloso. Debe mencionar que la anotación @Secured no funcionará porque sus privilegios no tienen el prefijo ROLE_. El RoleVoter los ignorará. Por lo tanto, debe usar explícitamente @PreAuthorize ("hasAuthority ('READ_PRIVILEGE_')") en su lugar, o hacer lo que hice e incluir los nombres de los roles en la lista de autoridades otorgadas en getPrivileges () para que pueda mezclar / combinar ambos.

+ 0 -

hace 3 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)



Hola Aelix, ese es un buen punto. Por lo general, cambio ese prefijo predeterminado, pero no se mencionó en el artículo, así que lo arreglé con algunas notas adicionales sobre el mecanismo. Gracias por mencionarlo. Saludos, Eugen.

+ -1 -

hace 3 años



Huésped

Vineet Tyagi



En Spring Security 4 no necesita tener el prefijo ROLE_.

+ 0 -

hace 3 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)



Ese es un buen punto: hice una excavación rápida de JIRA y ese es definitivamente el caso. Sin embargo, parece que después del cambio, hay algunos errores (pequeños) si falta el prefijo, que se están solucionando con nuevas versiones. Entonces, aún puede valer la pena mantener el prefijo por ahora. Saludos y gracias por el útil comentario. Eugen

+ 0 -

hace 3 años

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

Ok



Huésped

Michael Tabak

¿No es esto básicamente porque a partir de Spring Security 3.1 (creo que esa es la versión) hasRole() antepone "ROLE_" a lo que sea que pase si "ROLE_" aún no está presente? ¿Hay algún otro cambio de primavera que afecte esto?

+ 0 -

hace 3 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)

No debería haber otras cosas que se vean afectadas. Pero, cuando lo estaba investigando hace un par de meses, noté algunas JIRA relacionadas con este cambio, por lo que está claro que hubo algún impacto después de que el prefijo "ROLE_" no se requiriera. Es por eso que sugerí seguir usándolo por un tiempo, hasta que estemos seguros de que no hay otro problema asociado.

+ 0 -

hace 3 años



Huésped

AaronA

¿A dónde se ha ido el código?

+ 0 -

hace 3 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)

Hola Aaron: lo cambié a un nuevo repositorio y actualicé la mayoría de los enlaces, pero me perdí este. Debería estar bien ahora, gracias por hacérmelo saber. Saludos, Eugen.

+ 0 -

hace 3 años



Huésped

Bui DucCanh

Hola, el primer artículo útil gracias.
Quiero crear algunas páginas de administración (administrar roles, permisos) para crear nuevos administradores, moderadores, crear nuevos roles a partir de permisos ... ¿Dónde debo declarar: PAPEL, PERMISO? Un controlador debe administrar por 1 PAPEL con permisos: CREAR, VER, ACTUALIZAR, ELIMINAR, EXPORTAR ...
¡Gracias!
P / s: Soy un principiante de Java, Spring.

+ 0 -

hace 3 años



Huésped

Eugen Paraschiv (<https://www.baeldung.com/>)


Hola Bui,
esa es una pregunta interesante, pero también es de muy alto nivel y hay muchas maneras de diseñar tu dominio. Puede administrar objetos secundarios a través de los padres o puede hacerlo por su cuenta, y hay ventajas y desventajas en ambos enfoques.
Entonces, desafortunadamente, este no es uno en el que obtendrá una respuesta simple, correcta / incorrecta. Pero espero que eso lo coloque en el camino correcto para encontrar esa respuesta para su escenario particular.
Saludos,
Eugen.

+ 0 -

hace 3 años

[Cargar más comentarios](#)

¡Los comentarios están cerrados en este artículo!

 **ezoic** (<https://www.ezoic.com/what-is-ezoic/>)

reportar este anun

CATEGORÍAS

PRIMAVERA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/](https://www.baeldung.com/category/spring/))
DESCANSO ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/](https://www.baeldung.com/category/rest/))
JAVA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/](https://www.baeldung.com/category/java/))
SEGURIDAD ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](https://www.baeldung.com/category/security-2/))
PERSISTENCIA ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](https://www.baeldung.com/category/persistence/))
JACKSON ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/](https://www.baeldung.com/category/json/jackson/))
HTTP DEL LADO DEL CLIENTE ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/](https://www.baeldung.com/category/http/))
KOTLIN ([HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](https://www.baeldung.com/category/kotlin/))

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" ([/JAVA-TUTORIAL](#))
JACKSON JSON TUTORIAL ([/JACKSON](#))
HTTPCLIENT 4 TUTORIAL ([/HTTPCLIENT-GUIDE](#))
RESTO CON SPRING TUTORIAL ([/REST-WITH-SPRING-SERIES](#))
TUTORIAL SPRING PERSISTENCE ([/PERSISTENCE-WITH-SPRING-SERIES](#))
SEGURIDAD CON PRIMAVERA ([/SECURITY-SPRING](#))

ACERCA DE

SOBRE BAELDUNG ([/ABOUT](#))
LOS CURSOS ([HTTPS://COURSES.BAELDUNG.COM](https://courses.baeldung.com))
TRABAJO DE CONSULTORÍA ([/CONSULTING](#))
META BAELDUNG ([HTTP://META.BAELDUNG.COM/](http://meta.baeldung.com/))
EL ARCHIVO COMPLETO ([/FULL_ARCHIVE](#))
ESCRIBIR PARA BAELDUNG ([/CONTRIBUTION-GUIDELINES](#))
EDITORES ([/EDITORS](#))
NUESTROS COMPAÑEROS ([/PARTNERS](#))
ANUNCIE EN BAELDUNG ([/ADVERTISE](#))

TÉRMINOS DE SERVICIO ([/TERMS-OF-SERVICE](#))
POLÍTICA DE PRIVACIDAD ([/PRIVACY-POLICY](#))
INFORMACIÓN DE LA COMPAÑÍA ([/BAELDUNG-COMPANY-INFO](#))
CONTACTO ([/CONTACT](#))

Utilizamos cookies para mejorar su experiencia con el sitio. Para obtener más información, puede leer la [Política de privacidad y cookies completa \(/privacy-policy\)](#)

Ok