

Blog sobre Java EE

Estás aquí: [Inicio](#)/[Spring](#)/[Spring Avanzado](#)/Uso de Spring Properties y encriptación

Uso de Spring Properties y encriptación

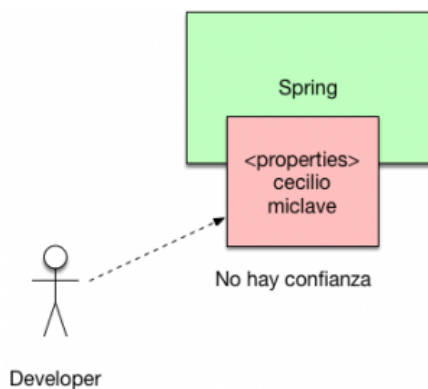
9 mayo, 2018 por [Cecilio Álvarez Caules](#) — 2 comentarios

Prueba ya Google AdWords - Llega a más clientes

Empieza a anunciarte hoy y te regalamos 75€ al invertir tus primeros 25€ [adwords.google.com](#)



El uso de **Spring Properties** es muy común cuando trabajamos con **Spring Framework**. Sin embargo hay situaciones en las cuales el manejo de propiedades **puede tener implicaciones no deseadas**. Uno de los casos más habituales es cuando en un fichero de propiedades se almacenan datos “**delicados**” como usuarios y contraseñas de una base de datos.



Spring y Confianza

No siempre la empresa tiene una confianza absoluta en los desarrolladores que contrata ya que en muchas casuísticas hay una cadena de subcontratas fuerte entre el desarrollador y el cliente final. ¿Cómo podemos abordar esto?. Vamos a ver un ejemplo muy sencillo apoyándonos en Spring Framework y usando un hola mundo de Spring security. Para ello el primer paso es saber cuales son las dependencias de maven que vamos a utilizar:

```
1 <dependencies>
2
3 <!-- https://mvnrepository.com/artifact/org.springframework/spring-
4 <dependency>
5     <groupId>org.springframework</groupId>
6     <artifactId>spring-core</artifactId>
7     <version>4.3.13.RELEASE</version>
8 </dependency>
9
10 <!-- https://mvnrepository.com/artifact/org.springframework/spring-
```

Este sitio web utiliza cookies para que usted tenga la mejor experiencia de usuario. Si continúa navegando está dando su consentimiento para la aceptación de las mencionadas cookies y la aceptación de nuestra [política de cookies](#), pinche el enlace para mayor información.

ACEPTAR

plugin cookies

```

21         <version>4.3.13.RELEASE</version>
22     </dependency>
23
24
25     <!-- https://mvnrepository.com/artifact/org.springframework/spring-
26     <dependency>
27         <groupId>org.springframework</groupId>
28         <artifactId>spring-webmvc</artifactId>
29         <version>4.3.13.RELEASE</version>
30     </dependency>
31
32     <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet
33     <dependency>
34         <groupId>javax.servlet</groupId>
35         <artifactId>javax.servlet-api</artifactId>
36         <version>3.1.0</version>
37         <scope>provided</scope>
38     </dependency>
39
40     <dependency>
41         <groupId>jstl</groupId>
42         <artifactId>jstl</artifactId>
43         <version>1.2</version>
44     </dependency>
45     <dependency>
46         <groupId>com.github.ulisesbocchio</groupId>
47         <artifactId>jasypt-spring-boot</artifactId>
48         <version>1.18</version>
49     </dependency>
50
51
52     <!-- https://mvnrepository.com/artifact/org.springframework.security
53     <dependency>
54         <groupId>org.springframework.security</groupId>
55         <artifactId>spring-security-web</artifactId>
56         <version>4.2.2.RELEASE</version>
57     </dependency>
58     <!-- https://mvnrepository.com/artifact/org.springframework.security
59     <dependency>
60         <groupId>org.springframework.security</groupId>
61         <artifactId>spring-security-config</artifactId>
62         <version>4.2.3.RELEASE</version>
63     </dependency>
64 </dependencies>

```

Spring Properties Configuración

El siguiente paso es configurar Spring a través del uso de anotaciones para que pueda arrancar, para ello vamos a usar un initializer y dos ficheros de configuración:

```

1  package com.arquitecturajava.init;
2
3  import javax.servlet.ServletContext;
4  import javax.servlet.ServletException;
5  import javax.servlet.ServletRegistration;
6
7  import org.springframework.web.WebApplicationInitializer;
8  import org.springframework.web.context.support.AnnotationConfigWebApplicati
9  import org.springframework.web.filter.DelegatingFilterProxy;
10 import org.springframework.web.servlet.DispatcherServlet;
11
12 import com.arquitecturajava.config.ConfiguracionSpring;
13

```

```

24 ServletRegistration.Dynamic servlet = contenedor.addServlet("dispat
25 servlet.setLoadOnStartup(1);
26 servlet.addMapping("/");
27
28 contenedor.addFilter("springSecurityFilterChain", new DelegatingFil
29     .addMappingForUrlPatterns(null, false, "/*");
30 }
31
32 }

```

Acabamos de inicializar el despachador de Spring y el filtro de seguridad, el siguiente paso es dar de alta los beans propios de Spring apoyandonos en anotaciones.

```

1 package com.arquitecturajava.config;
2
3 import org.springframework.context.annotation.Bean;
4 import org.springframework.context.annotation.ComponentScan;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.context.annotation.Import;
7 import org.springframework.web.servlet.ViewResolver;
8 import org.springframework.web.servlet.config.annotation.EnableWebMvc;
9 import org.springframework.web.servlet.view.InternalResourceViewResolver;
10 import org.springframework.web.servlet.view.JstlView;
11
12 @Configuration
13 @ComponentScan("com.arquitecturajava.*")
14 @EnableWebMvc
15 @Import(ConfiguracionSeguridad.class)
16 public class ConfiguracionSpring {
17
18     @Bean
19     public ViewResolver internalResourceViewResolver() {
20         InternalResourceViewResolver bean = new InternalResourceViewResolve
21         bean.setViewClass(JstlView.class);
22         bean.setPrefix("/WEB-INF/vistas/");
23         bean.setSuffix(".jsp");
24         return bean;
25     }
26
27 }

```

```

1 package com.arquitecturajava.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Configuration;
5 import org.springframework.security.config.annotation.authentication.buide
6 import org.springframework.security.config.annotation.web.builders.HttpSecu
7 import org.springframework.security.config.annotation.web.configuration.Ena
8 import org.springframework.security.config.annotation.web.configuration.Web
9
10 @Configuration
11 @EnableWebSecurity
12 public class ConfiguracionSeguridad extends WebSecurityConfigurerAdapter{
13
14     @Override
15     protected void configure(HttpSecurity http) throws Exception {
16
17         http.authorizeRequests().antMatchers("/**").hasRole("BASICO").and()
18         super.configure(http);
19     }
20
21     @Autowired
22     public void configureGlobal(AuthenticationManagerBuilder auth) thro

```

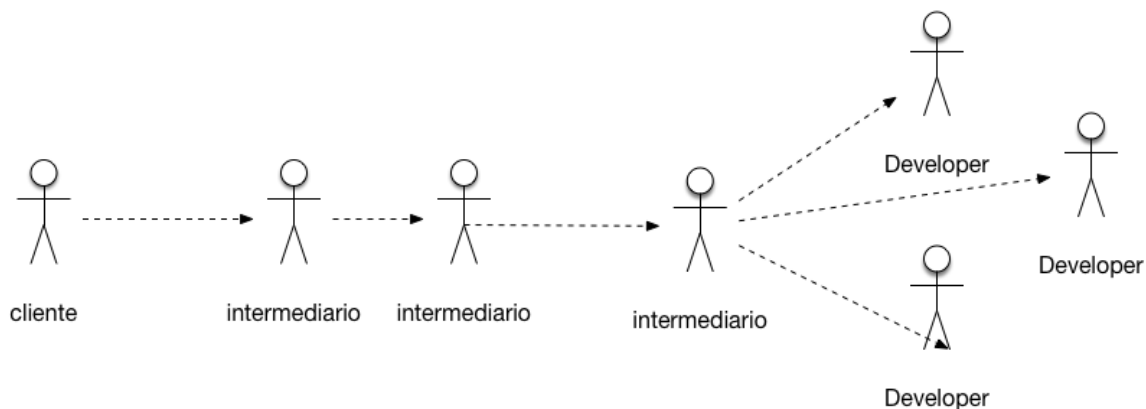
Por último un controlador para hacer pruebas:

```

1  package com.arquitecturajava.web;
2
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  @Controller
6  public class HolaController {
7
8      @RequestMapping("/hola")
9      public String hola() {
10
11          return "hola";
12      }
13  }

```

En principio la aplicación es perfectamente válida sin embargo tenemos un problema. El usuario y la clave (“cecilio”, “miclave”) **son fácilmente visibles por parte de un desarrollador**, los podemos externalizar a propiedades pero estaremos en la misma situación no conocemos al desarrollador.



Spring y Criptografía

Esto en algunas estructuras de empresa genera problemas **y se necesita un paso de seguridad adicional**. Para ello vamos a apoyarnos en **jasypt** una librería criptográfica que se puede integrar de forma muy sencilla con spring y afectar a cualquier cadena que almacene el framework. Para ello **lo que utilizaremos es una clave maestra que se encarga de encriptar el resto de datos delicados**. Para poder realizar esta operación necesitaremos modificar el fichero de seguridad de Spring.

```

1  package com.arquitecturajava.config;
2
3  import java.io.IOException;
4  import java.util.Properties;
5

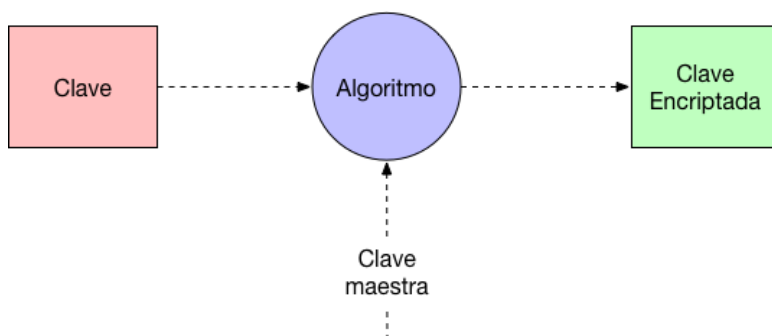
```

```

16 @Configuration
17 @EnableWebSecurity
18 public class ConfiguracionSeguridad extends WebSecurityConfigurerAdapter{
19
20
21     //lo asigno con un menos -d a nivel de properties
22     @Value("${clave_criptografica}")
23     private String clave;
24     @Override
25     protected void configure(HttpSecurity http) throws Exception {
26
27         http.authorizeRequests().antMatchers("/**").hasRole("BASICO").and()
28         super.configure(http);
29     }
30
31     @Autowired
32     public void configureGlobal(AuthenticationManagerBuilder auth) thro
33         auth
34             .inMemoryAuthentication()
35             .withUser(getUsuarioSegura()).password(getPasswordSegur
36
37     }
38
39
40     private String getPasswordSegura() throws IOException {
41         StandardPBEStrngEncryptor encryptor = new StandardPBEStrngEnc
42         Properties props = new EncryptableProperties(encryptor);
43
44         encryptor.setPassword(clave);
45         props.load(this.getClass().getClassLoader().getResourceAsStream
46
47         return props.getProperty("clave");
48     }
49
50     private String getUsuarioSegura() throws IOException {
51         StandardPBEStrngEncryptor encryptor = new StandardPBEStrngEnc
52         Properties props = new EncryptableProperties(encryptor);
53
54         encryptor.setPassword(clave);
55         props.load(this.getClass().getClassLoader().getResourceAsStream
56         return props.getProperty("usuario");
57     }
58
59 }

```

Como se puede observar hemos añadido un paso intermedio en el que se definen los métodos para leer una clave encriptada a través del fichero de propiedades. **¿Cómo funciona esto?** . Bueno necesitamos un algoritmo cryptográfico que se encargue de encriptar los datos utilizando una clave maestra.



```

Dcatalina.base="/Users/miusuario/GobiernoMayo/.metadata/.plugins/org.eclipse.wst.server.core/tmp0"
-Dcatalina.home="/Users/miusuario/Desktop/apache-tomcat-8.5.24" -
Dwtp.deploy="/Users/miusuario/GobiernoMayo/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps"
-Djava.endorsed.dirs="/Users/miusuario/Desktop/apache-tomcat-8.5.24/endorsed"
-Dclave_criptografica=clavemaestra

```

Generando las claves

El siguiente paso es encriptar el usuario y la clave utilizando nuestra "clavemaestra". Para ello abrimos un terminal y ejecutamos sobre el jar de jasypt.

```

java -cp jasypt-1.9.2.jar org.jasypt.intf.cli.JasyptPBESStringEncryptionCLI input="cecilio"
password="clavemaestra"

```

el resultado será:

```

----ARGUMENTS-----
input: cecilio
password: clavemaestra

----OUTPUT-----
xEPg2YnZiInlKvnwYkx31g==

```

Ya tenemos encriptado el usuario nos falta encriptar su clave:

```

----ARGUMENTS-----
input: miclave
password: clavemaestra

----OUTPUT-----
A0Atq09c7ADqM6YfI4hNWw==

```

Es momento de ubicar ambos datos encriptados en nuestro fichero de propiedades que spring leerá.

[JAVA SE](#)
[SPRING](#)
[JAVA EE](#)
[JAVASCRIPT](#)
[FRAMEWORKS JS](#)
[ARQUITECTURA](#)
[MIS LIBROS](#)
[MIS CURSOS](#)

```

usuario=ENC(xEPg2YnZiInlKvnwYkx31g==)

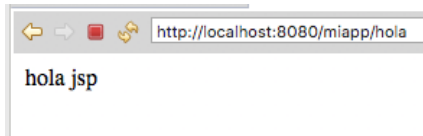
```

Login with Username and Password

User:

Password:

Introducimos “cecilio” de usuario y “miclave” como contraseña y accedemos:



Hemos accedido utilizando Spring Properties y su encriptación.

Otros artículos relacionados

1. [Spring Security JDBC y su configuracion](#)
2. [Spring Security \(I\) configuracion](#)
3. [Spring PropertyPlaceholderConfigurer](#)



PDF



Archivada en: [Spring Avanzado](#)

Etiquetada con: [SpringTips](#)

Leave a Reply

2 Comments on "Uso de Spring Properties y encriptación"



Join the discussion

☒ Subscribe ▼

▲ newest ▲ oldest ▲ most voted



Gabriel Espinel



Este sitio web utiliza cookies para que usted tenga la mejor experiencia de usuario. Si continúa navegando está dando su consentimiento para la aceptación de las mencionadas cookies y la aceptación de nuestra [política de cookies](#), pinche el enlace para mayor información.

[plugin cookies](#)

[ACEPTAR](#)



Cecilio Álvarez Caules



Gracias por el aporte . Es cierto que es mejorable 😊

Author

+ 0 - Reply

🕒 20 hours ago



BUSCAR

Buscar en este sitio ...

Cupón
Descuento
50%
PACKJAVACORE

Mis Cursos de Java Gratuitos

Java Herencia



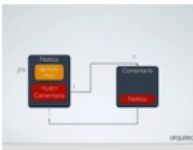
Java JDBC



Servlets



Introduccion JPA



Mis Cursos de Java



Java Web



Pack Java Core



Arquitectura Java Solida con Spring



POPULAR

Arquitecturas REST y sus niveles

Nuevo Curso:Arquitectura Java Sólida con Spring 4.3 y Anotaciones

Angular 5 Hello World y su funcionamiento

Java 9 Modules y el concepto de modularidad

Java 8 Lambda Syntax ,simplificando nuestro código

El concepto de Java Annotations y su funcionamiento

Spring REST Test utilizando Rest Assured

Java 9 Collections y sus novedades

Spring @Qualifier utilizando @Autowired

Java Composite Pattern y recursividad

CONTACTO

contacto@arquitecturajava.com

[REST API, JSON, SOAP y sus curiosidades](#)

[Usando Java Session en aplicaciones web](#)

[Java Iterator vs ForEach](#)

[Introducción a Servicios REST](#)

[¿Cuales son las certificaciones Java?](#)

[Java Override y encapsulación](#)

[Ejemplo de Java Singleton \(Patrones y ClassLoaders\)](#)

[¿Qué es Gradle?](#)

[REST JSON y Java](#)

[Usando el patron factory](#)

[Ejemplo de JPA , Introducción \(I\)](#)

[Uso de Java Generics \(I\)](#)

[¿Qué es un Microservicio?](#)

[Comparando java == vs equals](#)

[Mis Libros](#)

[Arquitecturas REST y sus niveles](#)

[Spring MVC Configuración \(I\)](#)

Copyright © 2018 · [eleven40 Pro Theme](#) en [Genesis Framework](#) · [WordPress](#) · [Acceder](#)