

Anuncios

AUTOMATTIC

We're hiring PHP developers  
anywhere in the world. Join us!

APPLY



REPORT THIS AD

**devs4j**

EL MEJOR SITIO WEB SOBRE PROGRAMACIÓN EN ESPAÑOL.

ESPAÑOL

# Spring framework 5 : Leer información de archivos .properties

BY RAIDENTRANCE ON FEBRERO 11, 2019 · ( DEJA UN COMENTARIO )

1 Vote

Una tarea común al hacer aplicaciones utilizando spring es leer configuraciones de archivos de tipo .properties, en este ejemplo tomaremos como base el post Spring framework 5 : Uso de @Autowire para listas de objetos (<https://devs4j.com/?p=7598>) y lo modificaremos para leer el valor de las figuras de un archivo properties en lugar de inyectarlo directamente.

## Paso 1 Creación del archivo properties

El primer paso será crear el archivo .properties que contendrá los valores de radio, ancho, largo y lado de las figuras que construiremos:

```
/src/main/resources/areas.properties
```

```
circle.radius = 10.0  
rectangle.width = 10.0  
rectangle.height = 5.0  
square.side=10.0
```

## Paso 2 Carga de propiedades a spring

Una vez que se creó el archivo de propiedades el siguiente paso será cargar esas propiedades a spring para esto crearemos la siguiente clase de configuración:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.support.PropertySourcesPlaceholderConfigurer;

/**
 * @author raidentrance
 *
 */
@Configuration
@PropertySource("classpath:areas.properties")
public class FigurePropertyConfiguration {

    @Bean
    public PropertySourcesPlaceholderConfigurer
        return new PropertySourcesPlaceholderConfigurer();
}
```

Como se puede ver en la anotación `@PropertySource` se define el archivo `.properties` del cuál se cargaran las propiedades.

## Paso 3 Modificar los beans para utilizar las propiedades

Una vez que se cargaron las propiedades a spring el siguiente paso será modificar nuestras clases, en el ejemplo anterior se utilizó `@Value("10.0")` lo cual inyectaba el valor de 10 a la referencia, ahora en lugar de hacer eso haremos un `@Value("${circle.radius:0}")` lo cual tomará el valor de la propiedad `circle.radius` y lo inyectará en nuestra variable, veamos como queda el código:

```
/**
 * @author raidentrance
 *
 */
public abstract class Figure {

    public abstract double getArea();
}
```

```
}
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * @author raidentrance
 *
 */
@Component
public class Circle extends Figure {

    private double radius;

    private static final Logger log = LoggerFactory.getLogger(Circle.class);

    public Circle(@Value("${circle.radius:0}") double radius) {
        this.radius = radius;
    }

    @Override
    public double getArea() {
        log.info("Calculating the are of a circle with radius: " + radius);
        return Math.pow(Math.PI * radius, 2);
    }

}
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * @author raidentrance
 *
 */
```

```
*/
@Component
public class Rectangle extends Figure {

    private double width;

    private double height;

    private static final Logger log = LoggerFactory.getLogger(Rectangle.class);

    public Rectangle(@Value("${rectangle.width:0}") double width, @Value("${rectangle.height:0}") double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double getArea() {
        log.info("Calculating the area of a rectangle with width: " + width + " and height: " + height);
        return width * height;
    }

}
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

/**
 * @author raidentrance
 */
@Component
public class Square extends Figure {

    private double side;

    private static final Logger log = LoggerFactory.getLogger(Square.class);

    public Square(@Value("${square.side:0}") double side) {
        this.side = side;
    }

}
```

```
    }

    @Override
    public double getArea() {
        log.info("Calculating the are of a s
        return Math.pow(side, 2);
    }
}
```

La siguiente sintaxis **@Value("\${square.side:0}")** significa toma el valor de la propiedad square.side, en caso de que no exista se asignará el valor 0.

El uso de propiedades es muy común debido a que es posible cambiar las configuraciones del código sin tener que re compilar el código.

Para estar al pendiente sobre nuestro contenido nuevo síguenos en nuestras redes sociales <https://www.facebook.com/devs4j/>

(<https://www.facebook.com/devs4j/>) y <https://twitter.com/devs4j> (<https://twitter.com/devs4j>).

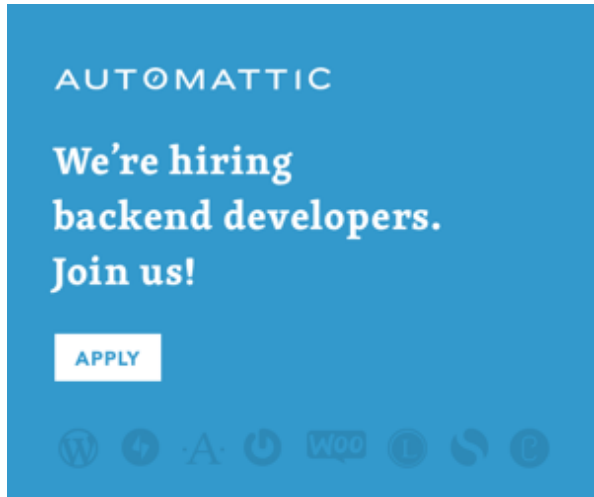
*Autor: Alejandro Agapito Bautista*

*Twitter: @raidentrance*

(<https://geeksjavamexico.wordpress.com/mentions/raidentrance/>)

*Contacto:raidentrance@gmail.com*

Anuncios



AUTOMATTIC

**We're hiring  
backend developers.  
Join us!**

APPLY

WordPress, PHP, JavaScript, Node.js, WooCommerce, and other technologies.

REPORT THIS AD



**Monetize your  
WordPress blog!**

WordAds

LEARN MORE

REPORT THIS AD



## Publicado por raidentrance

Soy @raidentrance en Twitter y en Github, soy egresado de la Facultad de Ingeniería de la UNAM, cuento con 8 certificaciones en diferentes áreas del desarrollo de software, me gustan las cervezas y soy Geek. Ver todas las entradas de raidentrance (<https://devs4j.com/author/raidentrance/>)

