

nuncios

AUTOMATTIC

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

APPLY



REPORT THIS AD

**devs4j**

EL MEJOR SITIO PARA SABER SOBRE PROGRAMACIÓN EN ESPAÑOL.

ESPAÑOL

# Spring framework 5 : Spring Expression Language SpEL

BY RAIDENTRANCE ON FEBRERO 12, 2019 · ( DEJA UN COMENTARIO )

1 Vote

Spring expression language (SpEL) es un lenguaje de expresiones que permite realizar operaciones sobre la información en tiempo de ejecución, en el post anterior se utilizó para leer información de un archivo .properties, en este post hablaremos más sobre expression language en detalle.

Los operadores disponibles en SpEL son los siguientes:

Operadores aritmeticos	+, -, *, /, %, ^, div, mod
Relacionales	<, >, ==, !=, <=, >=, lt, gt, eq, ne, le, ge
Lógicos	and, or, not, &&,   , !
Condicionales	?:
Expresiones regulares	Matchers

A continuación mostraremos como utilizarlos en una aplicación de spring.

## SpelExpressionParser

Antes de iniciar, el primer paso será aprender a utilizar la clase

**SpelExpressionParser** la cual nos permite evaluar expresiones sin necesidad de iniciar el contexto de Spring framework, a continuación se muestra como crearlo:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.expression.Expression;
import org.springframework.expression.ExpressionParser;
import org.springframework.expression.spel.standard.SpelExpressionParser;

/**
 * @author raidentrance
 *
 */
public class SpelExpressionApplication {

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

    public static void main(String[] args) {
        ExpressionParser expressionParser = new SpelExpressionParser();
        Expression expression = expressionParser.parseExpression("1 + 1");
        log.info("String expression {}", expression.getValue());
    }
}
```

Analizando el código anterior podemos ver lo siguiente :

- **SpelExpressionParser** nos permite evaluar expresiones de spring SpEL
- El método Expression **parseExpression**(String expression) recibe como parámetro una expresión de spring y devuelve un objeto de tipo **Expression**.
- La clase **Expression** contiene un método llamado Object **getValue()** el cuál devuelve el resultado de la expresión.

## Ejemplos básicos de expresiones

Una vez que entendimos como evaluar expresiones SpEL, el siguiente paso será ver algunos ejemplos:

### Uso de Strings

```
public static void main(String[] args) {  
    ExpressionParser expressionParser = new SpelExpressionParser();  
    Expression expression = expressionParser.parseExpression("Hi from devs4j");  
    log.info("Concat expression {}", expression.getValue());  
    expression = expressionParser.parseExpression("toUpperCase");  
    log.info("To upper case expression {}", expression.getValue());  
    expression = expressionParser.parseExpression("length");  
    log.info("Get length expression {}", expression.getValue());  
}
```

La expresión anterior concatena los Strings “Hi” y “from devs4j” al ejecutarlo el resultado será:

```
11:05:36.022 [main] INFO com.devs4j.spel.SpelExpressionParser - Hi from devs4j  
11:05:36.026 [main] INFO com.devs4j.spel.SpelExpressionParser - To upper case  
11:05:36.026 [main] INFO com.devs4j.spel.SpelExpressionParser - 16
```

## Operadores aritméticos

```
private static final Logger log = LoggerFactory.getLogger(SpelExpressionParser.class);  
  
public static void main(String[] args) {  
    ExpressionParser expressionParser = new SpelExpressionParser();  
    Expression expression = expressionParser.parseExpression("1 + 2");  
    log.info("Arithmetic expression {}", expression.getValue());  
  
    expression = expressionParser.parseExpression("1 * 2");  
    log.info("Arithmetic expression {}", expression.getValue());  
  
    expression = expressionParser.parseExpression("1 - 2");  
    log.info("Arithmetic expression {}", expression.getValue());  
  
    expression = expressionParser.parseExpression("1 / 2");  
    log.info("Arithmetic expression {}", expression.getValue());  
  
    expression = expressionParser.parseExpression("1 % 2");  
    log.info("Arithmetic expression {}", expression.getValue());  
}
```

```
        expression = expressionParser.parseExpression(expression);
        log.info("Arithmetic expression {}", expression);

        expression = expressionParser.parseExpression(expression);
        log.info("Arithmetic expression {}", expression);

    }
}
```

La salida del código anterior sería:

```
11:17:42.464 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.468 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.469 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.469 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.470 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.470 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
11:17:42.470 [main] INFO com.devs4j.spel.SpelExpressionParser - Arithmetic expression {}
```

## Operadores relacionales

```
public static void main(String[] args) {
    ExpressionParser expressionParser = new SpelExpressionParser();
    Expression expression = expressionParser.parseExpression("1 < 2");
    log.info("Relational expression {}", expression);

    expression = expressionParser.parseExpression("1 <= 2");
    log.info("Relational expression {}", expression);

    expression = expressionParser.parseExpression("1 > 2");
    log.info("Relational expression {}", expression);

    expression = expressionParser.parseExpression("1 >= 2");
    log.info("Relational expression {}", expression);

    }
}
```

Salida del código anterior:

```
11:19:59.557 [main] INFO com.devs4j.spel.SpelExpres  
11:19:59.561 [main] INFO com.devs4j.spel.SpelExpres  
11:19:59.562 [main] INFO com.devs4j.spel.SpelExpres  
11:19:59.562 [main] INFO com.devs4j.spel.SpelExpres
```

## Operadores lógicos

```
public static void main(String[] args) {  
    ExpressionParser expressionParser = new Spel  
    Expression expression = expressionParser.par  
    log.info("Relational expression {}", expres  
  
    expression = expressionParser.parseExpressio  
    log.info("Relational expression {}", expres  
  
    expression = expressionParser.parseExpressio  
    log.info("Relational expression {}", expres  
}
```

Salida:

```
11:22:29.569 [main] INFO com.devs4j.spel.SpelExpres  
11:22:29.573 [main] INFO com.devs4j.spel.SpelExpres  
11:22:29.573 [main] INFO com.devs4j.spel.SpelExpres
```

## Operadores condicionales

```
private static final Logger log = LoggerFactory.getI  
  
public static void main(String[] args) {  
    ExpressionParser expressionParser = new Spel  
    Expression expression = expressionParser.par  
    log.info("Conditional expression {}", expres  
}
```

Salida:

```
11:24:44.866 [main] INFO com.devs4j.spel.SpelExpres
```

## Uso de expresiones regulares en SpEL

Veamos el siguiente modelo:

```
class User {  
    private String username;  
    private String password;  
  
    public User(String username, String password)  
    {  
        this.username = username;  
        this.password = password;  
    }  
  
    // Getters y setters  
}
```

```
public static void main(String[] args) {  
  
    User user = new User("raidentrance", "devs4j");  
  
    ExpressionParser expressionParser = new SpelExpressionParser();  
    Expression expression = expressionParser.parseExpression("username");  
    log.info("Regex expression {}", expression.getLiteralValue());  
  
    expression = expressionParser.parseExpression("password");  
    log.info("Regex expression {}", expression.getLiteralValue());  
}
```

Salida:

```
11:37:49.944 [main] INFO com.devs4j.spel.SpelExpres  
11:37:49.947 [main] INFO com.devs4j.spel.SpelExpres
```

Para estar al pendiente sobre nuestro contenido nuevo síguenos en nuestras redes sociales


<https://twitter.com/devs4j> (<https://twitter.com/devs4j>).

Contacto:raidentrance@gmail.com

# AUTOMATTIC

```
<?php find_developers( [  
    'language' => PHP,  
    'specialty' => SCALING,  
    'location' => ANYWHERE,  
] );
```

APPLY

[REPORT THIS AD](#)

AUTOMATTIC

We're hiring  
backend developers.  
Join us!

APPLY

WordPress logo, Automattic logo, A logo, Power button icon, WooCommerce logo, L logo, S logo, P logo

REPORT THIS AD

7/9



## 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/>)



