(/)

Validar números de teléfono con Java Regex

Última modificación: 10 de mayo de 2020

por baeldung (https://www.baeldung.com/author/baeldung/)
(https://www.baeldung.com/author/baeldung/)

Java (https://www.baeldung.com/category/java/) +

Regex (https://www.baeldung.com/tag/regex/)

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

1. Información general

A veces, necesitamos validar el texto para asegurarnos de que su contenido cumpla con algún formato. En este tutorial rápido, veremos cómo validar diferentes formatos de números de teléfono usando expresiones regulares (https://www.baeldung.com/regular-expressions-java) .

2. Expresiones regulares para validar números de teléfono

2.1. Número de diez dígitos

Comencemos con una expresión simple que **verificará si el número tiene diez dígitos y nada más** :

```
@Test
public void whenMatchesTenDigitsNumber_thenCorrect() {
    Pattern pattern = Pattern.compile("^\\d{10}$");
    Matcher matcher = pattern.matcher("2055550125");
    assertTrue(matcher.matches());
}
```

Esta expresión permitirá números como *2055550125* .

2.2. Número con espacios en blanco, puntos o guiones

En el segundo ejemplo, veamos cómo podemos **permitir espacios en blanco, puntos o guiones opcionales (-)** entre los números:

```
@Test
public void whenMatchesTenDigitsNumberWhitespacesDotHyphen_thenCorrect() {
    Pattern pattern = Pattern.compile("^(\\d{3}[- .]?){2}\\d{4}$");
    Matcher matcher = pattern.matcher("202 555 0125");
    assertTrue(matcher.matches());
}
```

Para lograr este objetivo adicional (espacios en blanco o guiones opcionales), simplemente hemos agregado los caracteres:

- [-.]?

Este patrón permitirá números como *2055550125* , *202 555 0125* , *202.555.0125* y *202-555-0125* .

2.3. Número con paréntesis

A continuación, agreguemos la posibilidad de tener la **primera parte de nuestro teléfono entre paréntesis** :

```
@Test
public void whenMatchesTenDigitsNumberParenthesis_thenCorrect() {
    Pattern pattern = Pattern.compile"^((\\(\\d{3}\\))|\\d{3})[- .]?\\d{3}[- .]?\\d{4}$");
    Matcher matcher = pattern.matcher("(202) 555-0125");
    assertTrue(matcher.matches());
}
```

Para permitir el paréntesis opcional en el número, hemos agregado los siguientes caracteres a nuestra expresión regular:

- (\\ (\\ d {3} \\)) | \\ d {3})

Esta expresión permitirá números como *(202) 5550125* , *(202) 555-0125* o *(202) -555-0125* . Además, esta expresión también permitirá los números de teléfono cubiertos en el ejemplo anterior.

2.4. Número con prefijo internacional

Finalmente, veamos cómo **permitir un prefijo internacional al comienzo de un número de teléfono** :

```
@Test
public void whenMatchesTenDigitsNumberPrefix_thenCorrect() {
  Pattern pattern = Pattern.compile("^(\\+\\d{1,3}( )?)?((\\(\\d{3}\\))|\\d{3})[- .]?\\d{3}[- .]?\\d{4}$
  Matcher matcher = pattern.matcher("+111 (202) 555-0125");

  assertTrue(matcher.matches());
}
```

Para permitir el prefijo en nuestro número, hemos agregado al comienzo de nuestro patrón los caracteres:

- (\\ + \\ d {1,3} ()?)?

Esta expresión permitirá que los números de teléfono incluyan prefijos internacionales, teniendo en cuenta que los prefijos internacionales son normalmente números con un máximo de tres dígitos.

3. Aplicación de múltiples expresiones regulares

Como hemos visto, un número de teléfono válido puede adoptar varios formatos diferentes. Por lo tanto, es posible que deseemos verificar si nuestra *Cadena* cumple con alguno de estos formatos.

En la última sección, comenzamos con una expresión simple y agregamos más complejidad para lograr el objetivo de cubrir más de un formato. Sin embargo, a veces no es posible usar solo una expresión. En esta sección, veremos **cómo unir múltiples expresiones regulares en una sola** .

Si no podemos crear una expresión regular común que pueda validar todos los casos posibles que queremos cubrir, podemos definir diferentes expresiones para cada uno de los casos y luego usarlos todos juntos concatenando con un símbolo de barra (|).

Veamos un ejemplo donde usamos las siguientes expresiones:

- La expresión utilizada en la última sección:
  - ^ (\\ + \\ d {1,3} ()?)? ((\\ (\\ d {3} \\)) | \\ d {3}) [-.]? \\ d { 3} [-.]? \\ d {4} $
- Expresión regular para permitir números como *+111 123 456 789:*
  - ^ (\\ + \\ d {1,3} ()?)? (\\ d {3} []?) {2} \\ d {3} $
- Patrón para permitir números como *+111 123 45 67 89:*
  - ^ (\\ + \\ d {1,3} ()?)? (\\ d {3} []?) (\\ d {2} []?) {2} \\ d {2} $

```
@Test
public void whenMatchesPhoneNumber_thenCorrect() {
    String patterns
      = "^(\\+\\d{1,3}( )?)?((\\(\\d{3}\\))|\\d{3})[- .]?\\d{3}[- .]?\\d{4}$"
      + "|^(\\+\\d{1,3}( )?)?(\\d{3}[ ]?){2}\\d{3}$"
      + "|^(\\+\\d{1,3}( )?)?(\\d{3}[ ]?)(\\d{2}[ ]?){2}\\d{2}$";

    String[] validPhoneNumbers
      = {"2055550125","202 555 0125", "(202) 555-0125", "+111 (202) 555-0125", "636 856 789", "+111 636

    Pattern pattern = Pattern.compile(patterns);
    for(String phoneNumber : validPhoneNumbers) {
        Matcher matcher = pattern.matcher(phoneNumber);
        assertTrue(matcher.matches());
    }
}
```

Como podemos ver en el ejemplo anterior, al usar el símbolo de tubería, podemos usar las tres expresiones de una vez, lo que nos permite cubrir más casos que con una sola expresión regular.

4. Conclusión

En este artículo, hemos visto cómo verificar si una *Cadena* contiene un número de teléfono válido utilizando diferentes expresiones regulares. También hemos aprendido a usar múltiples expresiones regulares al mismo tiempo.

Como siempre, el código fuente completo del artículo está disponible en GitHub (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-regex) .

2 Deja una respuesta

Únete a la discusión...

1 1 0 0 2

el más nuevo más antiguo más votado

Chas B

Invitado

Para los números de teléfono, puede que usar el libphonenumber de Google sea mejor que regex: https://github.com/google/libphonenumber (https://github.com/google/libphonenumber)

+ 0 0 − Respuesta

Hace 7 días

(https://www.baeldung.com/author/loredana-crusoveanu/)

Invitado

Loredana Crusoveanu (https://www.baeldung.com/author/loredana-crusoveanu/)

Hola Chas,
gracias por los comentarios. En este artículo, nos centramos solo en Java Regex.
Sin embargo, libphonenumber parece ser bastante interesante. Podríamos escribir un artículo dedicado en el futuro.
Salud

+ 0 0 − Respuesta

Hace 6 días

CATEGORIAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)

DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)

JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)

SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)

PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)

JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)

HTTP DEL LADO DEL CLIENTE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)

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)
TRABAJOS (/TAG/ACTIVE-JOB/)
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)