

(<http://baeldung.com>)

Compruebe si una cadena es numérica en Java

Última modificación: 13 de mayo de 2018

por baeldung (<http://www.baeldung.com/author/baeldung/>)
(<http://www.baeldung.com/author/baeldung/>)

Java (<http://www.baeldung.com/category/java/>) +



Acabo de anunciar los nuevos módulos de **Spring 5** en **REST With Spring**:



>> COMPRUEBA EL CURSO →

1. Introducción

A menudo, mientras operamos en *Strings*, necesitamos averiguar si una *Cadena* es un número válido o no.

En este tutorial, exploraremos varias formas de determinar si una cadena es numérica, primero usando Java simple, luego expresiones regulares y finalmente usando librerías externas.

Una vez que hayamos terminado de discutir varias implementaciones, utilizaremos puntos de referencia para tener una idea de qué métodos son óptimos.

Comencemos con algunos requisitos previos antes de comenzar el contenido principal.

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

2. Prerrequisito

En la última parte de este artículo, usaremos la biblioteca externa de Apache Commons. Para incluir esta dependencia, agregue las siguientes líneas en *pom.xml*:

```
1 <dependency>
2     <groupId>org.apache.commons</groupId>
3     <artifactId>commons-lang3</artifactId>
4     <version>3.7</version>
5 </dependency>
```

La última versión de esta biblioteca se puede encontrar en Maven Central (<http://search.maven.org/#search%7Cga%7C1%7Cg%3A%22org.apache.commons%22%20AND%20a%3A%22commons-lang3%22>).

3. Java simple

Quizás la manera más fácil y confiable de verificar si una *Cadena* es numérica o no es analizándola usando los métodos integrados de Java:

1. *Integer.parseInt (String)*
2. *Float.parseFloat (String)*
3. *Double.parseDouble (String)*
4. *Long.parseLong (String)*
5. *nuevo BigInteger (String)*

Si estos métodos no arrojan ninguna *NumberFormatException* (<https://docs.oracle.com/javase/7/docs/api/java/lang/NumberFormatException.html>), significa que el análisis fue exitoso y la *cadena* es numérica:

```
1 public static boolean isNumeric(String strNum) {
2     try {
3         double d = Double.parseDouble(strNum);
4     } catch (NumberFormatException | NullPointerException nfe) {
5         return false;
6     }
7     return true;
8 }
```

Veamos este método en acción:

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

```

1  assertThat(isNumeric("22")).isTrue();
2  assertThat(isNumeric("5.05")).isTrue();
3  assertThat(isNumeric("-200")).isTrue();
4  assertThat(isNumeric("10.0d")).isTrue();
5  assertThat(isNumeric(" 22 ")).isTrue();
6
7  assertThat(isNumeric(null)).isFalse();
8  assertThat(isNumeric("")).isFalse();
9  assertThat(isNumeric("abc")).isFalse();

```

En nuestro método *isNumeric()*, solo estamos buscando valores que sean de tipo *Doble*, pero este método también se puede modificar para verificar Números *enteros*, *Flotantes*, *Largos* y grandes mediante el uso de cualquiera de los métodos de análisis que hemos alistado anteriormente.

Estos métodos también se tratan en el artículo de Java String Conversions (<http://www.baeldung.com/java-string-conversions>).

4. Expresiones regulares

Ahora usemos regex `-? \d+ (\.\d+)?` para hacer coincidir las cadenas numéricas que constan del entero positivo o negativo y los flotantes.

Pero no hace falta decir que definitivamente podemos modificar esta expresión regular para identificar y hacer frente a una amplia gama de reglas. Aquí, lo mantendremos simple.

Analicemos esta expresión regular y veamos cómo funciona:

- `-?` - esta parte identifica si el número dado es negativo, el guión `-` busca dash literalmente y el signo de interrogación `?` marca su presencia como una opción
- `\d+` - esto busca uno o más dígitos
- `(\.\d+)?` - esta parte de regex es para identificar los números de flotador. Aquí buscamos uno o más dígitos seguidos de un punto. El signo de interrogación, al final, significa que este grupo completo es opcional.

Las expresiones regulares son un tema muy amplio, y para obtener una breve descripción general, visite este artículo vinculado de Baeldung (<http://www.baeldung.com/regular-expressions-java>).

Por ahora, creemos un método usando la expresión regular anterior:

```

1  public static boolean isNumeric(String strNum) {
2      return strNum.matches("-?\\d+(\\.\\d+)?");
3  }

```

Veamos ahora algunas afirmaciones para el método anterior:

```

1  assertThat(isNumeric("22")).isTrue();
2  assertThat(isNumeric("5.05")).isTrue();
3  assertThat(isNumeric("-200")).isTrue();
4
5  assertThat(isNumeric("abc")).isFalse();

```

5. Apache Commons

En esta sección, discutiremos varios métodos disponibles en la biblioteca de Apache Commons.

5.1. *NumberUtils.isCreatable (String)*

NumberUtils (<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/math/NumberUtils.html>) de Apache Commons proporciona un método estático *NumberUtils.isCreatable (String)* (<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/math/NumberUtils.html#isCreatable-java.lang.String->) que comprueba si un número válido de Java es o no.

Este método acepta:

1. Números hexadecimales que comienzan con 0x o 0X
2. Números Octal comenzando con un 0 líder
3. Notación científica (por ejemplo, 1.05e-10)
4. Números marcados con un calificador de tipo (por ejemplo, 1L o 2.2d)

Si la cadena suministrada es *nula* o está *vacía / en blanco*, entonces no se considera como un número y este método devolverá *falso* en ese caso.

Vamos a ejecutar algunas pruebas con este método:

```

1  assertThat(NumberUtils.isCreatable("22")).isTrue();
2  assertThat(NumberUtils.isCreatable("5.05")).isTrue();
3  assertThat(NumberUtils.isCreatable("-200")).isTrue();
4  assertThat(NumberUtils.isCreatable("10.0d")).isTrue();
5  assertThat(NumberUtils.isCreatable("1000L")).isTrue();
6  assertThat(NumberUtils.isCreatable("0xFF")).isTrue();
7  assertThat(NumberUtils.isCreatable("07")).isTrue();
8  assertThat(NumberUtils.isCreatable("2.99e+8")).isTrue();
9
10 assertThat(NumberUtils.isCreatable(null)).isFalse();
11 assertThat(NumberUtils.isCreatable("")).isFalse();
12 assertThat(NumberUtils.isCreatable("abc")).isFalse();
13 assertThat(NumberUtils.isCreatable(" 22 ")).isFalse();
14 assertThat(NumberUtils.isCreatable("09")).isFalse();

```

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Observe cómo estamos obteniendo afirmaciones *verdaderas* para números hexadecimales, números octales y notaciones científicas en las líneas 6, 7 y 8, respectivamente.

También en la línea 14, la cadena "09" devuelve *falso* porque el "0" precedente indica que este es un número octal y "09" no es un número octal válido.

Por cada entrada que devuelve *verdadero* con este método, podemos usar *NumberUtils.createNumber (String)*

(<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/math/NumberUtils.html#createNumber-java.lang.String->) que nos dará el número válido.

5.2. *NumberUtils.isParsable (String)*

El método *NumberUtils.isParsable (String)*

(<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/math/NumberUtils.html#isParsable-java.lang.String->) verifica si la *cadena* dada es procesable o no.

Los números procesables son aquellos que se analizan satisfactoriamente mediante cualquiera de los métodos de análisis, como *Integer.parseInt (String)*, *Long.parseLong (String)*, *Float.parseFloat (String)* o *Double.parseDouble (String)*.

A diferencia de *NumberUtils.isCreatable ()*, este método no aceptará números hexadecimales, anotaciones científicas o cadenas que terminen con cualquier calificador de tipo, es decir, 'f', 'F', 'd', 'D', 'l' o 'L'.

Veamos algunas afirmaciones:

```

1  assertTrue(NumberUtils.isParsable("22")).isTrue();
2  assertTrue(NumberUtils.isParsable("-23")).isTrue();
3  assertTrue(NumberUtils.isParsable("2.2")).isTrue();
4  assertTrue(NumberUtils.isParsable("09")).isTrue();
5
6  assertTrue(NumberUtils.isParsable(null)).isTrue();
7  assertTrue(NumberUtils.isParsable("")).isFalse();
8  assertTrue(NumberUtils.isParsable("6.2f")).isFalse();
9  assertTrue(NumberUtils.isParsable("9.8d")).isFalse();
10 assertTrue(NumberUtils.isParsable("22L")).isFalse();
11 assertTrue(NumberUtils.isParsable("0xFF")).isFalse();
12 assertTrue(NumberUtils.isParsable("2.99e+8")).isFalse();

```

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

En la línea 4, a diferencia de *NumberUtils.isCreatable ()*, el número que comienza con la cadena "0" no se considera como un número octal, sino como un número decimal normal y, por lo tanto, devuelve verdadero.

Podemos usar este método como reemplazo de lo que hicimos en la sección 3, donde estamos tratando de analizar un número y verificar si hay un error.

5.3. *StringUtils.isNumeric(CharSequence)*

El método *StringUtils.isNumeric(CharSequence)*

(<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html#isNumeric-java.lang.CharSequence->) verifica estrictamente los dígitos Unicode. Esto significa:

1. Cualquier dígito de cualquier idioma que sea un dígito Unicode es aceptable
2. Como un punto decimal no se considera un dígito Unicode, no es válido
3. Los signos principales (ya sean positivos o negativos) tampoco son aceptables

Veamos ahora este método en acción:

```

1  assertTrue(StringUtils.isNumeric("123")).isTrue();
2  assertTrue(StringUtils.isNumeric("१२३")).isTrue();
3  assertTrue(StringUtils.isNumeric("१२३")).isTrue();
4
5  assertFalse(StringUtils.isNumeric(null)).isFalse();
6  assertFalse(StringUtils.isNumeric("")).isFalse();
7  assertFalse(StringUtils.isNumeric(" ")).isFalse();
8  assertFalse(StringUtils.isNumeric("12 3")).isFalse();
9  assertFalse(StringUtils.isNumeric("ab2c")).isFalse();
10 assertFalse(StringUtils.isNumeric("12.3")).isFalse();
11 assertFalse(StringUtils.isNumeric("-123")).isFalse();

```

Tenga en cuenta que los parámetros de entrada en las líneas 2 y 3 representan los números 123 en árabe y Devanagari, respectivamente. Dado que son dígitos Unicode válidos, este método devuelve *verdadero* en ellos.

5.4. *StringUtils.isNumericSpace(CharSequence)*

El *StringUtils.isNumericSpace(CharSequence)*

(<https://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html#isNumericSpace-java.lang.CharSequence->) chequea estrictamente para dígitos y / o espacio Unicode. Es lo mismo que *StringUtils.isNumeric()* con la única diferencia de que también acepta espacios, no solo espacios iniciales y finales, sino también si están entre números:

¿Cuál de estos es el más cercano a su trabajo / Sección actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

```

1  assertThat(StringUtils.isNumericSpace("123")).isTrue();
2  assertThat(StringUtils.isNumericSpace("\u0023")).isTrue();
3  assertThat(StringUtils.isNumericSpace("")).isTrue();
4  assertThat(StringUtils.isNumericSpace(" ")).isTrue();
5  assertThat(StringUtils.isNumericSpace("12 3")).isTrue();
6
7  assertThat(StringUtils.isNumericSpace(null)).isFalse();
8  assertThat(StringUtils.isNumericSpace("ab2c")).isFalse();
9  assertThat(StringUtils.isNumericSpace("12.3")).isFalse();
10 assertThat(StringUtils.isNumericSpace("-123")).isFalse();

```

6. Puntos de referencia

Antes de concluir este artículo, repasemos rápidamente los resultados de referencia que nos ayudarán a analizar cuáles de los métodos mencionados anteriormente son los enfoques óptimos:

1	Benchmark	Mode	Cnt	Score	Err
2	Benchmarking.usingCoreJava	avgt	20	152.061 ±	24.3
3	Benchmarking.usingRegularExpressions	avgt	20	1299.258 ±	175.6
4	Benchmarking.usingNumberUtils_isCreatable	avgt	20	63.811 ±	5.6
5	Benchmarking.usingNumberUtils_isParsable	avgt	20	58.706 ±	5.2
6	Benchmarking.usingStringUtils_isNumeric	avgt	20	35.599 ±	8.4
7	Benchmarking.usingStringUtils_isNumericSpace	avgt	20	37.010 ±	4.3

Como podemos ver, la operación más costosa es con expresiones regulares, seguida por una solución básica basada en Java. Todas las demás operaciones que utilizan la biblioteca de Apache Commons son en general iguales.

7. Conclusión

En este artículo, exploramos diferentes formas de verificar si una cadena es numérica o no. Analizamos ambas soluciones: métodos de la biblioteca y bibliotecas externas.

Como siempre, la implementación de todos los ejemplos y fragmentos de código dados anteriormente, incluido el código utilizado para realizar los puntos de referencia, se puede encontrar en GitHub (<https://github.com/eugenp/tutorials/tree/master/core-java>).

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

Acabo de anunciar los nuevos módulos de Spring 5 en REST With Spring:

>> VERIFIQUE LAS LECCIONES (/rest-with-spring-course#new-modules)

3 Deja una respuesta



Join the discussion...

2 1 0 ⚡ 🔥



✉ Suscribir ▼

▲ el más nuevo ▲ más antiguo ▲ el más votado



Huésped

JoeHx (<http://hendrixjoseph.github.io/>)



Me preguntaba cómo el uso de try-catch se compararía con el uso de expresiones regulares. Es interesante la gran diferencia entre esos dos y el resto.

+ 0 - Reply

🕒 24 days ago



Guest

John Douglass



You will probably get a good performance boost for regex by reusing a Pattern object, e.g.:

```
private static Pattern pNumeric = Pattern.compile("-?\\d+(\\.\\d+)?");
public static boolean isNumeric(String strNum) {
    return pNumeric.matcher(strNum).matches();
}
```

It was about 3-4x faster for me, but still much slower than the alternatives.

+ 0 - Reply

🕒 7 days ago ▲

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior


Desarrollador principal

Arquitecto

Gerente

8/6/2018

Compruebe si una cadena es numérica en Java | Baeldung



(<http://www.baeldung.com/author/loredana-crusoveanu/>)

Editor

Loredana Crusoveanu (<http://www.baeldung.com/author/loredana-crusoveanu/>) updated this article.

+ 0 -

Reply

🕒

7 days ago

CATEGORIES

- SPRING ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/](http://www.baeldung.com/category/spring/))
- REST ([HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/](http://www.baeldung.com/category/rest/))
- JAVA ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/](http://www.baeldung.com/category/java/))
- SECURITY ([HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/](http://www.baeldung.com/category/security-2/))
- PERSISTENCE ([HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/](http://www.baeldung.com/category/persistence/))
- JACKSON ([HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/](http://www.baeldung.com/category/jackson/))
- HTTPCLIENT ([HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/](http://www.baeldung.com/category/http/))
- KOTLIN ([HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/](http://www.baeldung.com/category/kotlin/))

SERIES

- JAVA "BACK TO BASICS" TUTORIAL ([HTTP://WWW.BAELDUNG.COM/JAVA-TO-BASICS/](http://www.baeldung.com/java-to-basics/))
- JACKSON JSON TUTORIAL ([HTTP://WWW.BAELDUNG.COM/JACKSON/](http://www.baeldung.com/jackson/))
- HTTPCLIENT 4 TUTORIAL ([HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE/](http://www.baeldung.com/httpclient-guide/))
- REST WITH SPRING TUTORIAL ([HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/](http://www.baeldung.com/rest-with-spring-series/))
- SPRING PERSISTENCE TUTORIAL ([HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/](http://www.baeldung.com/persistence-with-spring-series/))
- SECURITY WITH SPRING ([HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING/](http://www.baeldung.com/security-spring/))

ABOUT

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

[ABOUT BAELDUNG \(HTTP://WWW.BAELDUNG.COM/ABOUT/\)](http://www.baeldung.com/about/)

[THE COURSES \(HTTP://COURSES.BAELDUNG.COM\)](http://courses.baeldung.com)

[CONSULTING WORK \(HTTP://WWW.BAELDUNG.COM/CONSULTING\)](http://www.baeldung.com/consulting)

[META BAELDUNG \(HTTP://META.BAELDUNG.COM/\)](http://meta.baeldung.com/)

[EL ARCHIVO COMPLETO \(HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE\)](http://www.baeldung.com/full_archive)

[ESCRIBIR PARA BAELDUNG \(HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES\)](http://www.baeldung.com/contribution-guidelines)

[CONTACTO \(HTTP://WWW.BAELDUNG.COM/CONTACT\)](http://www.baeldung.com/contact)

[INFORMACIÓN DE LA COMPAÑÍA \(HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO\)](http://www.baeldung.com/baeldung-company-info)

[TÉRMINOS DE SERVICIO \(HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE\)](http://www.baeldung.com/terms-of-service)

[POLÍTICA DE PRIVACIDAD \(HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY\)](http://www.baeldung.com/privacy-policy)

[EDITORES \(HTTP://WWW.BAELDUNG.COM/EDITORS\)](http://www.baeldung.com/editors)

[KIT DE MEDIOS \(PDF\) \(HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF\)](https://s3.amazonaws.com/baeldung.com/baeldung+-+media+kit.pdf)

**¿Cuál de estos es el más
cercano a su trabajo /
función actual?**

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente