Expresiones regulares

Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en cadenas.

En JavaScript, las expresiones regulares también son objetos. Estos patrones se utilizan con los métodos `exec()` y `test()` de `RegExp`, y con los métodos `match()`, `matchAll()`, `replace()`, `replaceAll()`, `search()` y `split()` de `String`.

Crear una expresión regular

Se puede construir una expresión regular de estas dos formas:

- Usando una expresión regular literal, que consiste en un patrón encerrado entre barras, como sigue:

```
let re = /ab+c/;
```

  Las expresiones regulares literales proporcionan la compilación de la expresión regular cuando se carga el script. Si la expresión regular permanece constante, su uso puede mejorar el rendimiento.

- O llamando a la función constructora del objeto RegExp, de la siguiente manera:

```
let re = new RegExp('ab+c');
```

  El uso de la función constructora proporciona una compilación en tiempo de ejecución de la expresión regular. Usa la función constructora cuando sepas que el patrón de la expresión regular cambiará, o no conoces el patrón y lo obtienes de otra fuente, como la entrada del usuario.

Escribir un patrón de expresión regular

Un patrón de expresión regular se compone de caracteres simples, como `/abc/`, o una combinación de caracteres simples y especiales, como `/ab*c/` o `/Capítulo (\d)\.\d*/`.

Usar patrones simples

Los patrones simples se construyen con caracteres para los que deseas encontrar una coincidencia directa.
Por ejemplo, el patrón `/abc/` coincide con combinaciones de caracteres en cadenas solo cuando ocurre la secuencia exacta `"abc"` (todos los caracteres juntos y en ese orden). Tal coincidencia tendría éxito en las cadenas `"Hola, ¿conoces tu abc?"` y `"Los últimos diseños de aviones evolucionaron a partir de slabcraft"`. En ambos casos, la coincidencia es con la subcadena `"abc"`. No hay ninguna coincidencia en la cadena `"Grab crab"` porque aunque contiene la subcadena `"ab c"`, no contiene la subcadena `"abc"` exacta.

Usar caracteres especiales

Cuando la búsqueda de una coincidencia requiere algo más que una coincidencia exacta, como por ejemplo buscar una o más 'b', o encontrar espacios en blanco, puedes incluir caracteres especiales en el patrón.

Por ejemplo, para hacer coincidir una sola "a" seguida de cero o más "b"s seguidas de "c", usarías el patrón /ab*c/: el * después de "b" significa "0 o más apariciones del elemento anterior". En la cadena "cbbabbbbcdebc", este patrón coincidirá con la subcadena "abbbbc".

El documento “Sintaxis de expresiones regulares” proporciona listas de los diferentes caracteres especiales que encajan en cada categoría, junto con descripciones y ejemplos.

Aserciones
: Las aserciones incluyen límites, que indican el comienzo y el final de líneas y palabras, y otros patrones que indican de alguna manera que el reconocimiento es posible (incluidas las expresiones anticipadas, inversas y condicionales).

Clases de caracteres
: Distingue diferentes tipos de caracteres. Por ejemplo, distinguir entre letras y dígitos.

Grupos y rangos
: Indica grupos y rangos de caracteres de expresión.

Cuantificadores
: Indica el número de caracteres o expresiones que deben coincidir.

Escapes de propiedades Unicode
: Distinguir según las propiedades de los caracteres Unicode, por ejemplo, letras mayúsculas y minúsculas, símbolos matemáticos y de puntuación.

En la siguiente tabla se pueden ver todos los caracteres especiales que se pueden usar en expresiones regulares:

Caracteres	Tipo de caracteres especiales
\, ., \cX, \d, \D, \f, \n, \r, \s, \S, \t, \v, \w, \W, \0, \xhh, \uhhhh, \ uhhhhh, [\b]	Clases de caracteres
^, $, x(?=y), x(?!y), (?<=y)x, (?<!y)x, \b, \B	Aserciones
(x), (?:x), (?<Name>x), x|y, [xyz], [^xyz], *Number*	Grupos y rangos
*, +, ?, x{*n*}, x{*n*,}, x{*n*,*m*}	Cuantificadores
\p{*UnicodeProperty*}, \P{*UnicodeProperty*}	Escapes de propiedades Unicode

Escapando

Si necesitas usar literalmente cualquiera de los caracteres especiales (en realidad buscando un "*", por ejemplo), lo debes escapar colocando una barra invertida delante de él. Por ejemplo, para buscar "a" seguido de "*" seguido de "b", usarías /a*b/ — la barra invertida "escapa" de "*", volviéndola literal en lugar de especial.

De manera similar, si estás escribiendo un literal de expresión regular y necesitas buscar una barra inclinada ("/"), la debes escapar (de lo contrario, esta termina el patrón). Por ejemplo, para buscar la cadena "/ejemplo/" seguida de uno o más caracteres alfabéticos, usarías /\/ejemplo\/[a-z]+/i: las barras invertidas antes de cada barra, las hace literales.

Para hacer coincidir una barra invertida literal, debes escapar de la barra invertida. Por ejemplo, para encontrar la cadena "C:\" donde "C" puede ser cualquier letra, usarías /[A-Z]:\\/ — la primera barra invertida escapa a la que sigue, por lo que la expresión busca una sola barra invertida literal.

Si usas el constructor RegExp con un literal de cadena, recuerda que la barra invertida es un escape en los literales de cadena, por lo que para usarlo en la expresión regular, debes escapar en el nivel del literal de cadena. /a*b/ y new RegExp("a*b") crean la misma expresión, que busca "a" seguida de un "*" literal seguido de "b".

Si las cadenas de escape aún no forman parte de tu patrón, puedes agregarlas usando String.replace:

```
function escapeRegExp(string) {
  return string.replace(/[.*+\-?^${}()|[\]\\]/g,'\\$&');    // $& significa toda la cadena coincidente
}
```

La "g" después de la expresión regular es una opción o indicador que realiza una búsqueda global, buscando en toda la cadena y devolviendo todas las coincidencias.

Usando paréntesis

Los paréntesis alrededor de cualquier parte del patrón de expresión regular hacen que se recuerde esa parte de la subcadena coincidente. Una vez reconocida, la subcadena se puede recuperar para otro uso.

Usar expresiones regulares en JavaScript

Métodos que usan expresiones regulares

Método	Descripción
exec()	Ejecuta una búsqueda por una coincidencia en una cadena. Devuelve un arreglo de información o null en una discrepancia.
test()	Prueba una coincidencia en una cadena. Devuelve true o false.
match()	Devuelve un arreglo que contiene todas las coincidencias, incluidos los grupos de captura, o null si no se encuentra ninguna coincidencia.
matchAll()	Devuelve un iterador que contiene todas las coincidencias, incluidos los grupos de captura.
search()	Prueba una coincidencia en una cadena. Devuelve el índice de la coincidencia, o -1 si la búsqueda falla.
replace()	Ejecuta una búsqueda por una coincidencia en una cadena y reemplaza la subcadena coincidente con una subcadena de reemplazo.
replaceAll()	Ejecuta una búsqueda de todas las coincidencias en una cadena y reemplaza las subcadenas coincidentes con una subcadena de reemplazo.
split()	Utiliza una expresión regular o una cadena fija para dividir una cadena en un arreglo de subcadenas.

Cuando desees saber si un patrón se encuentra en una cadena, utiliza los métodos test() o search(); para obtener más información (pero una ejecución más lenta) utiliza los métodos exec() o match(). Si usas exec() o match() y si la búsqueda tiene éxito, estos métodos devuelven un arreglo y actualizan las propiedades del objeto expresión regular asociado y también del objeto de expresión regular predefinido, el objeto RegExp. Si la búsqueda falla, el método exec() devuelve null (que coacciona a false).

En el siguiente ejemplo, el script utiliza el método exec() para encontrar una coincidencia en una cadena.

```
var myRe = /d(b+)d/g;
var myArray = myRe.exec('cdbbdbsbz');
```

Si no necesitas acceder a las propiedades de la expresión regular, una forma alternativa de crear myArray es con este script:

```
var myArray = /d(b+)d/g.exec('cdbbdbsbz');
   // similar a "cdbbdbsbz" .match(/d(b+)d/g); sin embargo,
   // "cdbbdbsbz" .match (/d(b+)d/g) genera Array ["dbbd"], mientras
   // /d(b+)d/g.exec('cdbbdbsbz ') produce Array ['dbbd', 'bb', index: 1, input: 'cdbbdbsbz' ].
```

Si deseas construir la expresión regular a partir de una cadena, otra alternativa más es este script:

```
var myRe = new RegExp('d(b+)d', 'g');
var myArray = myRe.exec('cdbbdbsbz');
```

Con estos scripts, la búsqueda se realiza correctamente, devuelve el arreglo y actualiza las propiedades que se muestran en la siguiente tabla.

Resultado de la ejecución de expresiones regulares.

Objeto	Propiedad o índice	Descripción	En este ejemplo
myArray		La cadena coincidente y todas las subcadenas recordadas.	['dbbd', 'bb', index: 1, input: 'cdbbdbsbz']
	índice	El índice basado en 0 de la coincidencia en la cadena de entrada.	1
	entrada	La cadena original.	'cdbbdbsbz'
	[0]	Los últimos caracteres encontrados.	'dbbd'
myRe	lastIndex	El índice en el que comenzará la siguiente búsqueda. (Esta propiedad se establece solo si la expresión regular usa la opción g).	5
	fuente	El texto del patrón. Actualizado en el momento en que se crea la expresión regular, no se ejecuta.	'd(b+)d'

Como se muestra en la segunda forma de este ejemplo, puedes usar una expresión regular creada con un iniciador de objeto sin asignarla a una variable. Sin embargo, si lo hace, cada aparición es una nueva expresión regular. Por este motivo, si utilizas esta forma sin asignarla a una variable, no podrás acceder posteriormente a las propiedades de esa expresión regular. Por ejemplo, supongamos que tienes este script:

```
let myRe = /d(b+)d/g;
const myArray = myRe.exec('cdbbdbsbz');
console.log('El valor de lastIndex es ' + myRe.lastIndex);
// "El valor de lastIndex es 5"
```

Sin embargo, si tienes este script:

```
const myArray = /d(b+)d/g.exec('cdbbdbsbz');
console.log('El valor de lastIndex es ' + /d(b+)d/g.lastIndex);
// "El valor de lastIndex es 0"
```

Las apariciones de /d(b+)d/g en las dos declaraciones son objetos de expresión regular diferentes y, por lo tanto, tienen valores diferentes para su propiedad lastIndex. Si necesitas acceder a las propiedades de una expresión regular creada con un iniciador de objeto, primero debes asignarla a una variable.

Búsqueda avanzada con banderas

Las expresiones regulares tienen seis indicadores opcionales que permiten funciones como la búsqueda global y que no distinga entre mayúsculas y minúsculas. Estos indicadores se pueden usar por separado o juntos en cualquier orden y se incluyen como parte de la expresión regular.

Indicadores de expresión regular

Bandera	Descripción	Propiedad
g	Búsqueda global.	RegExp.prototype.global
i	Búsqueda que no distingue entre mayúsculas y minúsculas.	RegExp.prototype.ignoreCase
m	Búsqueda multilínea.	RegExp.prototype.multiline
s	Permite que el . coincida con caracteres de nueva línea.	RegExp.prototype.dotAll
u	"unicode"; tratar un patrón como una secuencia de puntos de código Unicode.	RegExp.prototype.unicode
y	Realiza una búsqueda "pegajosa" que coincida a partir de la posición actual en la cadena de destino.	RegExp.prototype.sticky

Para incluir una bandera con la expresión regular, usa esta sintaxis:

```
let re = /patrón/banderas;
```

o

```
let re = new RegExp('patrón', 'banderas');
```

Ten en cuenta que las banderas son parte integral de una expresión regular. No se pueden agregar ni eliminar más tarde.

Por ejemplo, `re = /\w+\s/g` crea una expresión regular que busca uno o más caracteres seguidos de un espacio y busca esta combinación en toda la cadena.

```
let re = /\w+\s/g;
let str = 'fee fi fo fum';
myArray = str.match(re);
log(myArray);
```

Podrías reemplazar la línea:

```
let re = /\w+\s/g;
```

con:

```
let re = new RegExp('\\w+\\s', 'g');
```

y obtener el mismo resultado.

El comportamiento asociado con el indicador g es diferente cuando se usa el método .exec(). Los roles de "clase" y "argumento" se invierten: En el caso de .match(), la clase cadena (o tipo de datos) posee el método y la expresión regular es solo un argumento, mientras que en el caso de .exec(), es la expresión regular la que posee el método, siendo la cadena el argumento. Compara esto str.match(re) con re.exec(str). El indicador g se usa con el método .exec() para obtener una progresión iterativa.

```
const xArray;
while (xArray = re.exec(str))
    console.log(xArray);
// produce:
// ["fee ", index: 0, input: "fee fi fo fum"]
// ["fi ", index: 4, input: "fee fi fo fum"]
// ["fo ", index: 7, input: "fee fi fo fum"]
```

La bandera m se utiliza para especificar que una cadena de entrada de varias líneas se debe tratar como varias líneas. Si se usa el indicador m, ^ y $ coinciden al principio o al final de cualquier línea dentro de la cadena de entrada en lugar del inicio o el final de toda la cadena.

Ejemplos

Hay varios ejemplos disponibles en las páginas de referencia para exec(), test(), match(), matchAll(), search(), replace(), split()