Plantillas de cadena de texto

This translation is incomplete. Please help translate this article from English

Resumen

Las plantillas de cadena de texto (template strings) son literales de texto que habilitan el uso de expresiones incrustadas. Es posible utilizar cadenas de texto de más de una línea, y funcionalidades de interpolación de cadenas de texto con ellas.

Sintaxis

```
`cadena de texto`
```

```
`línea 1 de la cadena de texto
 línea 2 de la cadena de texto`

`cadena de texto ${expresión} texto`

tag `cadena de texto ${expresión} texto`
```

Descripción

Las plantillas de cadena de texto se delimitan con el caracter de comillas o tildes invertidas (` `) (grave accent) , en lugar de las comillas simples o dobles. Las plantillas de cadena de texto pueden contener marcadores, identificados por el signo de dólar, y envueltos en llaves (`${expresión}`). Las expresiones contenidas en los marcadores, junto con el texto entre ellas, son enviados como argumentos a una función. La función por defecto simplemente concatena las partes para formar una única cadena de texto. Si hay una expresión antes de la plantilla de cadena de texto (i.e. `tag`), llamamos a esta plantilla de cadena de texto "plantilla de cadena de texto con etiqueta". En este caso, la expresión de etiqueta (típicamente una función) es llamada a partir de la cadena resultante de procesar la plantilla de cadena de texto, que luego puede ser manipulada antes de ser devuelta. En caso de querer escapar una comilla o tilde invertida en una plantilla de cadena de texto, pon un backslash \ antes de la comilla o tilde invertida.

```
1 `\`` === '`' // --> verdadero
```

Cadenas de más de una línea

Los caracteres de fin de línea encontrados son parte de la plantilla de cadena de texto. En el caso de cadenas de texto normales, esta es la sintaxis necesaria para obtener cadenas de más de una línea:

```
//ES5
console.log("línea 1 de cadena de texto\n\
línea 2 de cadena de texto");
// "línea 1 de cadena de texto
// línea 2 de cadena de texto"

//ES6
// Para obtener el mismo efecto con cadenas de texto multilínea, con ES6 es posible escribir:
console.log(`línea 1 de la cadena de texto
línea 2 de la cadena de texto`);
// "línea 1 de la cadena de texto
// línea 2 de la cadena de texto"
```

Interpolación de expresiones

Para combinar expresiones dentro de cadenas de texto normales, se usa la siguiente sintaxis:

```
1 var a = 5;
2 var b = 10;
```

```
console.log("Quince es " + (a + b) + " y\nno " + (2 * a + b) + ".");
// "Quince es 15 y
// no 20."
```

Ahora, con las plantillas de cadena de texto, tenemos una sintaxis que habilita la misma funcionalidad, con un código más agradable a la vista y fácil de leer:

```
var a = 5;
var b = 10;
console.log(`Quince es ${a + b} y\nno ${2 * a + b}.`);
// "Quince es 15 y
// no 20."
```

Anidamiento de plantillas

En ciertos momentos, anidar una plantilla es la forma más fácil y quizás más legible de tener cadenas configurables. Dentro de una plantilla con tildes invertidas, es sencillo permitir tildes invertidas interiores simplemente usándolos dentro de un marcador de posición `${ }` dentro de la plantilla. Por ejemplo, si la condición a es verdadera: entonces devuelva este literal con plantilla.

En ES5:

```
var classes = 'header'
classes += (isLargeScreen() ?
   '' : item.isCollapsed ?
     ' icon-expander' : ' icon-collapser');
```

En ES6 con plantillas de cadena de texto y sin anidamiento:

```
const classes = `header ${ isLargeScreen() ? '' :
    (item.isCollapsed ? 'icon-expander' : 'icon-collapser') }`;
```

En ES5 con plantillas de cadena de texto anidadas:

```
const classes = `header ${ isLargeScreen() ? '' :
 `icon-${item.isCollapsed ? 'expander' : 'collapser'}` }`;
```

Plantillas de cadena de texto con postprocesador

Una forma más avanzada de plantillas de cadenas de texto son aquellas que contienen una función de postprocesado. Con ellas es posible modificar la salida de las plantillas, usando una función. El primer argumento contiene un array con las cadenas de texto de la plantilla ("Hola" y "mundo" en el ejemplo). El segundo y subsiguientes argumentos con los valores procesados (ya *cocinados)* de las expresiones de la plantilla (en este caso "15" y "50"). Finalmente, la función devuelve la cadena de texto manipulada. El nombre "tag" de la función no es nada especial, se puede usar cualquier nombre de función en su lugar.

```
1  var persona = 'Mike';
2  var edad = 28;
3
4  function myTag(strings, expPersona, expEdad)
5  {
```

```
  var str0 = strings[0]; // "Ese "
  var str1 = strings[1]; // " es un "

  // Hay tecnicamente un String después
  // la expresión final (en nuestro ejemplo)
  // pero es vacia (""), asi que se ignora.
  // var str2 = strings[2];

  var strEdad;
  if (expEdad > 99)
  {
    strEdad = 'viejo';
  }
  else
  {
    strEdad = 'joven';
  }

  // Podemos incluso retornar un string usando una plantilla de cadena de texto
  return `${str0}${expPersona}${str1}${strEdad}`;
}

var salida = myTag`Ese ${ persona } es un ${ edad }`;

console.log(salida);
// Ese Mike es un joven
```

Cadenas en crudo (*raw*)

La propiedad especial `raw`, disponible en el primer argumento de las plantillas de cadenas de texto postprocesadas, nos permite acceder a las cadenas de texto tal como fueron ingresadas, sin procesar secuencias de escape

```
function tag(strings, ...values) {
  console.log(strings.raw[0]);
  // "linea 1 de cadena de texto \\n línea 2 de cadena de texto"
}

tag`línea 1 de cadena de texto \n línea 2 de cadena de texto`;
```

Adicionalmente, el método `String.raw()` permite crear cadenas de texto en crudo tal como serían generadas por la función por defecto de plantilla, concatenando sus partes.

```
var str = String.raw`¡Hola\n${2+3}!`;
// "¡Hola\n5!"

str.length;
// 9

str.split('').join(',');
// "¡,H,o,l,a,\,n,5,!"
```

Seguridad

Las plantillas de cadenas de texto **NO DEBEN** ser construidas por usuarios no confiables, porque tienen acceso a variables y funciones.

```
`${console.warn("this es",this)}`; // "this es" Window

let a = 10;
console.warn(`${a+=20}`); // "30"
console.warn(a); // 30
```

Especificaciones

Especificación	Status	Comentarios
ECMAScript 2015 (6th Edition, ECMA-262) La definición de 'Template Literals' en esta especificación. ECMAScript 2015 (6th Edition, ECMA-262) La definición de 'Tagged Templates' en esta especificación.	ST Standard	Definición inicial.

Compatibilidad de navegadores

> **We're converting our compatibility data into a machine-readable JSON format.** This compatibility table still uses the old format, because we haven't yet converted the data it contains. **Find out how you can help!**

Escritorio Móvil

Funcionalidad	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Soporte básico	41	34 (34)	Sin soporte	29	9.1

Funcionalidad	Android	Chrome for Android	Firefox Mobile (Gecko)	IE Mobile	Opera Mobile	Safari Mobile
Soporte básico	67	71	63.0 (63)	Sin soporte	46	9

Ver también

- `String`
- `String.raw()`
- Lexical grammar
- Template-like strings in ES3 compatible syntax

Última actualización: 23 may. 2019, por los colaboradores de MDN

Temas relacionados

JavaScript

Tutorials:

- Complete beginners
- JavaScript Guide
- Intermediate
- Advanced

References:

- Built-in objects
- Expressions & operators
- Statements & declarations
- Functions
- Classes
- Errors
- Misc

  JavaScript technologies overview

  Lexical grammar