Chaîne de caractères

L' `String` objet est utilisé pour représenter et manipuler une séquence de caractères.

La description

Les chaînes sont utiles pour contenir des données qui peuvent être représentées sous forme de texte. Certaines des opérations les plus utilisées sur les chaînes consistent à vérifier leur `length` , à les construire et à les concaténer à l'aide des opérateurs de chaîne + et += , à vérifier l'existence ou l'emplacement des sous-chaînes avec la `indexOf()` méthode ou à extraire des sous-chaînes avec la `substring()` méthode.

Création de chaînes

Les chaînes peuvent être créées en tant que primitives, à partir de littéraux de chaîne ou en tant qu'objets, à l'aide du `String()` constructeur :

```
const string1 = "A string primitive";
const string2 = 'Also a string primitive';
const string3 = `Yet another string primitive`;
```

```
const string4 = new String("A String object");
```

Les primitives de chaîne et les objets de chaîne peuvent être utilisés de manière interchangeable dans la plupart des situations. Voir " Primitives String et objets String " ci-dessous.

Les littéraux de chaîne peuvent être spécifiés à l'aide de guillemets simples ou doubles, qui sont traités de la même manière, ou à l'aide du caractère backtick `` ` ``. Cette dernière forme spécifie un modèle littéral : avec cette forme vous pouvez interpoler des expressions.

Accès personnage

Il existe deux façons d'accéder à un caractère individuel dans une chaîne. La première est la `charAt()` méthode :

```
return 'cat'.charAt(1) // returns "a"
```

L'autre méthode (introduite dans ECMAScript 5) consiste à traiter la chaîne comme un objet de type tableau, où les caractères individuels correspondent à un index numérique :

```
return 'cat'[1] // returns "a"
```

Lors de l'utilisation de la notation entre crochets pour l'accès aux caractères, la tentative de suppression ou d'attribution d'une valeur à ces propriétés échouera. Les propriétés impliquées ne sont ni inscriptibles ni configurables. (Voir `Object.defineProperty()` pour plus d'informations.)

Comparer des chaînes

En C, la `strcmp()` fonction est utilisée pour comparer des chaînes. En JavaScript, vous utilisez simplement les opérateurs inférieur à et supérieur à :

```
let a = 'a'
let b = 'b'
if (a < b) { // true
  console.log(a + ' is less than ' + b)
} else if (a > b) {
  console.log(a + ' is greater than ' + b)
} else {
  console.log(a + ' and ' + b + ' are equal.')
}
```

Un résultat similaire peut être obtenu en utilisant la `localeCompare()` méthode héritée par `String` les instances.

Notez que `a == b` compare les chaînes dans `a` et `b` pour être égaux de la manière habituelle sensible à la casse. Si vous souhaitez comparer sans tenir compte des caractères majuscules ou minuscules, utilisez une fonction similaire à celle-ci :

```
function isEqual(str1, str2)
{
    return str1.toUpperCase() === str2.toUpperCase()
} // isEqual
```

Les majuscules sont utilisées à la place des minuscules dans cette fonction, en raison de problèmes avec certaines conversions de caractères UTF-8.

Primitives de chaîne et objets String

Notez que JavaScript fait la distinction entre les `String` objets et les valeurs de chaîne primitives . (Il en est de même pour `Boolean` et `Numbers` .)

Les chaînes littérales (indiquées par des guillemets doubles ou simples) et les chaînes renvoyées par des `String` appels dans un contexte non constructeur (c'est-à-dire appelées sans utiliser le mot- `new` clé) sont des chaînes primitives. JavaScript convertit automatiquement les primitives en `String` objets, de sorte qu'il est possible d'utiliser

`String` des méthodes d'objet pour les chaînes primitives. Dans les contextes où une méthode doit être invoquée sur une chaîne primitive ou une recherche de propriété se produit, JavaScript enveloppera automatiquement la primitive de chaîne et appellera la méthode ou effectuera la recherche de propriété.

```
let s_prim = 'foo'
let s_obj = new String(s_prim)

console.log(typeof s_prim) // Logs "string"
console.log(typeof s_obj)  // Logs "object"
```

Les primitives de chaîne et les `String` objets donnent également des résultats différents lors de l'utilisation de `eval()` . Les primitives transmises à `eval` sont traitées comme du code source ; `String` les objets sont traités comme tous les autres objets, en retournant l'objet. Par example:

```
let s1 = '2 + 2'              // creates a string primitive
let s2 = new String('2 + 2')  // creates a String object
console.log(eval(s1))         // returns the number 4
console.log(eval(s2))         // returns the string "2 + 2"
```

Pour ces raisons, le code peut se casser lorsqu'il rencontre des `String` objets alors qu'il attend une chaîne primitive à la place, bien que généralement, les auteurs n'aient pas à se soucier de la distinction.

Un `String` objet peut toujours être converti en son homologue primitif avec la `valueOf()` méthode.

```
console.log(eval(s2.valueOf()))  // returns the number 4
```

Séquences d'échappement

Les caractères spéciaux peuvent être codés à l'aide de séquences d'échappement :

Séquence d'échappement	Point de code Unicode
`\0`	caractère nul (U+0000 NULL)
	guillemet simple (U+0027

Séquence d'échappement	Point de code Unicode
`\'`	guillemet simple (U+0027 APOSTROPHE)
`\"`	guillemet double (U+0022 GUILLEMET)
`\\`	barre oblique inverse (U+005C SOLIDE INVERSE)
`\n`	retour à la ligne (U+000A LINE FEED ; LF)
`\r`	retour chariot (U+000D RETOUR CHARIOT ; CR)
`\v`	tabulation verticale (TABULATION LIGNE U+000B)
`\t`	onglet (U+0009 TABULATION DE CARACTÈRES)
`\b`	retour arrière (U+0008 RETOUR ARRIÈRE)
`\f`	saut de page (U+000C FORM FEED)
`\uXXXX` … où `XXXX` est exactement 4 chiffres hexadécimaux dans la plage `0000` – `FFFF` ; par exemple, `\u000A` est le même que `\n` (LINE FEED); `\u0021` est " ! "	Point de code Unicode entre `U+0000` et `U+FFFF` (le plan multilingue de base Unicode)
`\u{X}` … `\u{XXXXXX}` … où `X` … `XXXXXX` correspond à 1–6 chiffres hexadécimaux dans la plage `0` – `10FFFF` ; par exemple, `\u{A}` est le même que `\n` (LINE FEED); `\u{21}` est " ! "	Point de code Unicode entre `U+0000` et `U+10FFFF` (l'intégralité d'Unicode)
`\xXX` … où `XX` est exactement 2 chiffres hexadécimaux dans la plage `00` – `FF` ; par exemple, `\x0A` est le même que `\n` (LINE FEED); `\x21` est " ! "	Point de code Unicode entre `U+0000` et `U+00FF` (les blocs Basic Latin et Latin-1 Supplement ; équivalent à ISO-8859-1)

Longues chaînes littérales

Longues chaînes littérales

Parfois, votre code inclura des chaînes très longues. Plutôt que d'avoir des lignes qui s'allongent sans fin ou qui s'enroulent au gré de votre éditeur, vous souhaiterez peut-être diviser spécifiquement la chaîne en plusieurs lignes dans le code source sans affecter le contenu réel de la chaîne. Vous pouvez le faire de deux manières.

Méthode 1

Vous pouvez utiliser l' opérateur + pour ajouter plusieurs chaînes ensemble, comme ceci :

```
let longString = "This is a very long string which needs " +
                 "to wrap across multiple lines because " +
                 "otherwise my code is unreadable."
```

Méthode 2

Vous pouvez utiliser le caractère barre oblique inverse (\ ) à la fin de chaque ligne pour indiquer que la chaîne continuera sur la ligne suivante. Assurez-vous qu'il n'y a pas d'espace ou tout autre caractère après la barre oblique inverse (à l'exception d'un saut de ligne) ou sous forme de retrait ; sinon ça ne marchera pas.

Ce formulaire ressemble à ceci :

```
let longString = "This is a very long string which needs \
to wrap across multiple lines because \
otherwise my code is unreadable."
```

Les deux méthodes ci-dessus produisent des chaînes identiques.

Constructeur

`String()`
: Crée un nouvel `String` objet. Il effectue la conversion de type lorsqu'il est appelé en tant que fonction, plutôt qu'en tant que constructeur, ce qui est généralement plus utile.

Méthodes statiques

`String.fromCharCode(num1 [, ...[, numN]])`
: Renvoie une chaîne créée à l'aide de la séquence spécifiée de valeurs Unicode.

`String.fromCodePoint(num1 [, ...[, numN)`
: Renvoie une chaîne créée à l'aide de la séquence de points de code spécifiée.

`String.raw()`

Renvoie une chaîne créée à partir d'une chaîne de modèle brute.

Propriétés d'occurrence

`String.prototype.length`
Reflète le `length` de la chaîne. Lecture seulement.

Méthodes d'instance

`String.prototype.at(index)` 
Renvoie le caractère (exactement une unité de code UTF-16) au spécifié `index`. Accepte les entiers négatifs, qui comptent à rebours à partir du dernier caractère de la chaîne.

`String.prototype.charAt(index)`
Renvoie le caractère (exactement une unité de code UTF-16) au spécifié `index`.

`String.prototype.charCodeAt(index)`
Renvoie un nombre qui correspond à la valeur de l'unité de code UTF-16 au niveau donné `index`.

`String.prototype.codePointAt(pos)`
Renvoie un nombre entier non négatif qui correspond à la valeur du point de code du point de code encodé UTF-16 commençant au spécifié `pos`.

`String.prototype.concat(str [, ...strN ])`
Combine le texte de deux chaînes (ou plus) et renvoie une nouvelle chaîne.

`String.prototype.includes(searchString [, position])`
Détermine si la chaîne d'appel contient `searchString`.

`String.prototype.endsWith(searchString [, length])`
Détermine si une chaîne se termine par les caractères de la chaîne `searchString`.

`String.prototype.indexOf(searchValue [, fromIndex])`
Renvoie l'index dans l' **`String`** objet appelant de la première occurrence de `searchValue`, ou `-1` s'il n'est pas trouvé.

`String.prototype.lastIndexOf(searchValue [, fromIndex])`
Renvoie l'index dans l'objet appelant **`String`** de la dernière occurrence de `searchValue`, ou `-1` s'il est introuvable.

`String.prototype.localeCompare(compareString [, locales [, options]])`

Renvoie un nombre indiquant si la chaîne de référence `compareString` vient avant, après ou est équivalente à la chaîne donnée dans l'ordre de tri.

`String.prototype.match(regexp)`

Utilisé pour faire correspondre une expression régulière `regexp` à une chaîne.

`String.prototype.matchAll(regexp)`

Renvoie un itérateur de toutes `regexp` les correspondances de .

`String.prototype.normalize([form])`

Renvoie le formulaire de normalisation Unicode de la valeur de la chaîne d'appel.

`String.prototype.padEnd(targetLength [, padString])`

Complète la chaîne actuelle à partir de la fin avec une chaîne donnée et renvoie une nouvelle chaîne de longueur `targetLength` .

`String.prototype.padStart(targetLength [, padString])`

Complète la chaîne actuelle depuis le début avec une chaîne donnée et renvoie une nouvelle chaîne de longueur `targetLength` .

`String.prototype.repeat(count)`

Renvoie une chaîne constituée des éléments de l'objet répétés plusieurs `count` fois.

`String.prototype.replace(searchFor, replaceWith)`

Utilisé pour remplacer les occurrences de `searchFor` using `replaceWith` . `searchFor` peut être une chaîne ou une expression régulière, et `replaceWith` peut être une chaîne ou une fonction.

`String.prototype.replaceAll(searchFor, replaceWith)`

Utilisé pour remplacer toutes les occurrences de `searchFor` using `replaceWith` . `searchFor` peut être une chaîne ou une expression régulière, et `replaceWith` peut être une chaîne ou une fonction.

`String.prototype.search(regexp)`

Recherche une correspondance entre une expression régulière `regexp` et la chaîne d'appel.

`String.prototype.slice(beginIndex[, endIndex])`

Extrait une section d'une chaîne et renvoie une nouvelle chaîne.

`String.prototype.split([sep [, limit] ])`

Renvoie un tableau de chaînes rempli en divisant la chaîne d'appel aux occurrences de la sous-chaîne `sep`

la seule chaîne "sep".

`String.prototype.startsWith(searchString [, length])`
Détermine si la chaîne d'appel commence par les caractères de chaîne `searchString`.

`String.prototype.substring(indexStart [, indexEnd])`
Renvoie une nouvelle chaîne contenant les caractères de la chaîne appelante à partir de (ou entre) l'index (ou les index) spécifié.

`String.prototype.toLocaleLowerCase( [locale, ...locales])`
Les caractères d'une chaîne sont convertis en minuscules tout en respectant les paramètres régionaux actuels.

Pour la plupart des langages, cela renverra la même chose que `toLowerCase()`.

`String.prototype.toLocaleUpperCase( [locale, ...locales])`
Les caractères d'une chaîne sont convertis en majuscules tout en respectant les paramètres régionaux actuels.

Pour la plupart des langages, cela renverra la même chose que `toUpperCase()`.

`String.prototype.toLowerCase()`
Renvoie la valeur de la chaîne d'appel convertie en minuscules.

`String.prototype.toString()`
Renvoie une chaîne représentant l'objet spécifié. Remplace la `Object.prototype.toString()` méthode.

`String.prototype.toUpperCase()`
Renvoie la valeur de la chaîne d'appel convertie en majuscules.

`String.prototype.trim()`
Supprime les espaces blancs du début et de la fin de la chaîne. Fait partie de la norme ECMAScript 5.

`String.prototype.trimStart()`
Supprime les espaces blancs à partir du début de la chaîne.

`String.prototype.trimEnd()`
Supprime les espaces blancs à partir de la fin de la chaîne.

`String.prototype.valueOf()`
Renvoie la valeur primitive de l'objet spécifié. Remplace la `Object.prototype.valueOf()` méthode

`Object.prototype.valueOf()` methods.

`String.prototype.@@iterator()`
Retourne un nouvel objet itérateur qui itère sur les points de code d'une valeur String, renvoyant chaque point de code sous la forme d'une valeur String.

Méthodes d'encapsulation HTML

> **Avertissement :** Obsolète. Évitez ces méthodes.
>
> Ils sont d'une utilité limitée, car ils ne fournissent qu'un sous-ensemble des balises et attributs HTML disponibles.

`String.prototype.anchor()`
`<a name="name">` (cible hypertexte)

`String.prototype.big()`
`<big>`

`String.prototype.blink()`
`<blink>`

`String.prototype.bold()`
`<b>`

`String.prototype.fixed()`
`<tt>`

`String.prototype.fontcolor()`
`<font color="color">`

`String.prototype.fontsize()`
`<font size="size">`

`String.prototype.italics()`
`<i>`

`String.prototype.link()`
`<a href="url">` (lien vers URL)

`String.prototype.small()`
`<small>`

`String.prototype.strike()`

`<strike>`

`String.prototype.sub()`
`<sub>`

`String.prototype.sup()`
`<sup>`

Exemples

Conversion de chaîne

Il est possible de l'utiliser comme alternative `String` plus fiable , car il fonctionne lorsqu'il est utilisé sur et . Par example: `toString()` `null` `undefined`

```
var nullVar = null;
nullVar.toString();       // TypeError: nullVar is null
String(nullVar);          // "null"

var undefinedVar;
undefinedVar.toString();  // TypeError: undefinedVar is undefined
String(undefinedVar);     // "undefined"
```

Caractéristiques

spécification
Spécification du langage ECMAScript (ECMAScript) # sec-string-objects

Compatibilité du navigateur

Signaler les problèmes avec ces données de compatibilité sur GitHub

`String`	
Chrome	1
Bord	12
Firefox	1
Internet Explorer	3
Opéra	3