Débuter en JavaScript

4

Les objets JavaScript

En regardant les exemples, on peut avoir l'impression qu'il existe un grand nombre d'objets JavaScript. En fait, ces objets appartiennent à quatre domaines différents :

- les objets JavaScript ;
- les objets du BOM ;
- les objets du DOM ;
- les objets créés par les développeurs.

Les objets JavaScript sont ceux fournis en standard par le langage, quel que soit le dispositif dans lequel il est implémenté. De ce fait, ils sont toujours disponibles, que le navigateur fonctionne sur un PC ou dans un téléphone mobile.

Parmi ces objets, on trouve ceux qui correspondent aux types de données que nous avons étudiés au chapitre 2 : `String` pour les chaînes de caractères, `Boolean` pour les valeurs booléennes, et `Number` pour les nombres. Ces objets encapsulent les types fondamentaux. Ils sont responsables des opérations de conversion ainsi que d'autres fonctionnalités.

Il existe également des objets particuliers tels que `Math`, `Date`, et `RegExp`. Ce dernier permet d'utiliser les *expressions rationnelles*, qui sont un outil puissant mais complexe servant à manipuler les chaînes de caractères de manière très précise.

JavaScript a également un objet d'agrégation, `Array`. Tous les objets JavaScript sont en fait des tableaux (*array* en anglais), bien que cela ne soit pas toujours évident lorsqu'on les manipule. Les objets standards JavaScript sont étudiés dans ce chapitre.

Le constructeur d'objets

Tous les objets sont dérivés d'un objet fondamental nommé **Object**. Cet objet est traité au chapitre 11, consacré à la création d'objets et de bibliothèques. La façon dont JavaScript gère l'extensibilité est un peu particulière. Bien que la version actuelle de JavaScript ne soit pas réellement orientée objets, ce langage possède le concept de constructeur et permet de créer des instances d'objets au moyen de la méthode **new**.

Tous les objets standards, à l'exception d'un seul, ont des méthodes et des propriétés associées à leur type. Certaines de ces méthodes et propriétés sont accessibles par l'intermédiaire des instances d'objets. D'autres sont statiques, ce qui signifie qu'elles ne sont accessibles qu'à partir de l'objet lui-même (et sont donc partagées par toutes les instances).

Le seul objet qui ne possède pas de propriétés ou de méthodes est l'objet **Boolean**. Les seules méthodes auxquelles il donne accès sont celles de l'objet **Object**. Nous l'utiliserons pour montrer comment il est possible de créer des instances d'objets, et nous passerons ensuite à des objets plus complexes.

Pour créer une nouvelle instance de l'objet **Boolean**, il suffit d'utiliser le mot-clé **new** de la façon suivante :

```
var reponse = new Boolean(true);
```

Une fois l'objet instancié, vous pouvez accéder à la valeur primitive qu'il encapsule (c'est-à-dire qu'il contient) à l'aide de la méthode **toValue** de l'objet **Object** :

```
if (reponse.toValue) ...
```

Vous pouvez également tester l'objet comme s'il s'agissait d'une primitive :

```
if (reponse) ...
```

L'objet **Boolean** ne dispose pas de fonctionnalités particulièrement enthousiasmantes, mais nous nous rattraperons en étudiant les autres objets !

L'objet Number

L'objet **Number** a des méthodes qui permettent d'effectuer des conversions en chaînes de caractères, en chaînes localisées (spécifiques à un pays), en valeurs exprimées avec une précision donnée, ou avec un certain nombre de décimales, ou en notation exponentielle. Il possède également quatre propriétés numériques, directement accessibles.

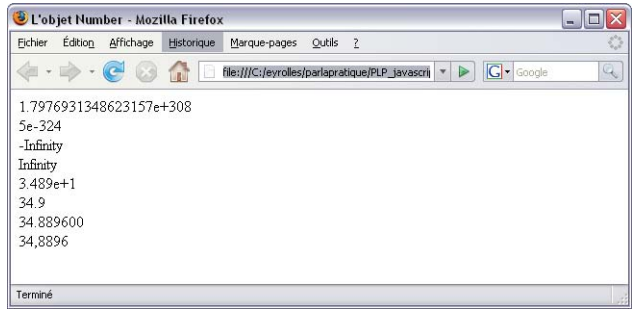
Plutôt que de donner la liste des propriétés et des méthodes de l'objet **Number**, nous exposerons dans l'exemple 4-1 la façon de les employer en affichant leur résultat ou leur valeur.

Exemple 4-1 – Les méthodes et propriétés de l'objet Number

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>L'objet Number</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
// Les propriétés de l'objet Number
document.writeln(Number.MAX_VALUE + "<br />");
document.writeln(Number.MIN_VALUE + "<br />");
document.writeln(Number.NEGATIVE_INFINITY + "<br />");
document.writeln(Number.POSITIVE_INFINITY + "<br />");
// Les méthodes spécifiques de l'objet Number
var newValue = new Number("34.8896");
document.writeln(newValue.toExponential(3) + "<br />");
document.writeln(newValue.toPrecision(3) + "<br />");
document.writeln(newValue.toFixed(6) + "<br />");
document.writeln(newValue.toLocaleString("fr") + "<br />");
//]]>
</script>
</body>
</html>
```

La figure 4-1 montre le résultat affiché par cet exemple.

Figure 4-1
L'affichage produit par l'exemple 4-1.



Dans l'exemple 4-1, deux constantes numériques – `MAX_VALUE` et `MIN_VALUE` – indiquent les valeurs maximales et minimales qui peuvent être représentées en JavaScript. Les valeurs `Infinity` et `-Infinity` sont celles qui sont retournées lorsqu'un débordement se produit ou lorsque les valeurs maximale ou minimale sont dépassées. Au chapitre 2, nous avons évoqué la constante globale `Infinity` dans la section intitulée *Le type number*. La propriété `POSITIVE_INFINITY` est égale à cette valeur.

Après avoir imprimé les constantes numériques, le programme crée une instance de l'objet `Number`. L'expression utilisée pour construire cette instance peut être une valeur numérique ou une chaîne représentant une telle valeur. S'il s'agit d'une chaîne et que son format n'est pas correct, la valeur de l'objet est `NaN`.

La première méthode invoquée est `toExponential`. Elle renvoie la valeur en notation scientifique avec la précision indiquée par son argument (ici, 3 chiffres après le point décimal). La seconde méthode est `toPrecision`. Elle renvoie la valeur en notation décimale, avec le nombre de chiffres significatifs indiqué. La troisième méthode est `toFixed`. Elle renvoie la valeur en notation décimale avec le nombre de chiffres décimaux indiqué. Enfin, la méthode `toLocaleString` renvoie une chaîne de caractères représentant la valeur exprimée selon les règles locales.

L'objet String

L'objet `String` est probablement le plus utilisé des objets standards JavaScript. Une instance de cet objet peut être créée explicitement au moyen du constructeur `new`, en lui passant en paramètre la valeur littérale de la chaîne :

```
var sObject = new String("Exemple de chaîne");
```

Cet objet a plusieurs méthodes. Certaines concernent l'utilisation des chaînes en HTML, d'autres non. Une des méthodes sans rapport avec HTML est `concat`, qui prend comme paramètres deux chaînes et retourne une chaîne composée en mettant les deux paramètres bout à bout. L'exemple 4-2 montre un exemple d'utilisation de cette méthode.

Exemple 4-2 – Création d'un objet String et utilisation de la méthode concat

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Etude des chaînes de caractères</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
```

Exemple 4-2 – Création d'un objet String et utilisation de la méthode concat (*suite*)

```
<script type="text/javascript">
//<![CDATA[
var sObj = new String();
var sTxt = sObj.concat("Ceci est ", "une nouvelle chaîne.");
document.writeln(sTxt);
//]]>
</script>
</body>
</html>
```

Il n'y a pas de limite au nombre de chaînes pouvant ainsi être assemblées. Toutefois, cette méthode est rarement employée car l'opérateur de concaténation de chaîne **+** fournit le même résultat plus facilement.

Les propriétés et méthodes de l'objet `String` sont décrites dans le tableau 4-1.

Tableau 4-1 – Les propriétés et méthodes de l'objet String

Méthodes ou propriété	Description	Paramètre
`valueOf`	Retourne la valeur littérale contenue dans l'objet	Aucun
`length`	Propriété contenant la longueur du contenu	Utilisé sans parenthèses
`anchor`	Crée un point d'ancrage HTML (cible d'un lien)	Le nom du point d'ancrage
`big`, `blink`, `bold`, `italics`, `small`, `strike`, `sub`, `sup`	Retourne le contenu de l'objet formaté	Aucun
`charAt`, `charCodeAt`	Retourne le caractère ou le code du caractère à une position donnée	Un entier représentant la position du caractère recherché (à partir de 0)
`indexOf`	Retourne la position de la première occurrence d'une chaîne dans la valeur de l'objet	La chaîne recherchée
`lastIndexOf`	Retourne la position de la dernière occurrence d'une chaîne dans la valeur de l'objet	La chaîne recherchée
`link`	Retourne un lien HTML	L'URL utilisée pour l'attribut `href` du lien
`concat`	Concaténation de chaînes	La chaîne à ajouter à la valeur de l'objet
`split`	Décomposition d'une chaîne en fonction d'un séparateur	Le séparateur et le nombre maximal d'éléments

Tableau 4-1 – Les propriétés et méthodes de l'objet String (*suite*)

Méthodes ou propriété	Description	Paramètre
`slice`	Retourne une section d'une chaîne	Les positions de début et de fin de la section
`substring`, `substr`	Retourne une sous-chaîne	Les positions de début et de fin de la sous-chaîne
`match`, `replace`, `search`	Recherche et remplacement à l'aide d'expressions rationnelles	Chaîne représentant l'expression rationnelle utilisée
`toLowerCase`, `toUpperCase`	Changement de casse	Aucun

Les méthodes de mise en forme HTML (`anchor`, `link`, `big`, `blink`, `bold`, `italics`, `sub`, `sup`, `small` et `strike`) retournent une chaîne contenant la valeur littérale de l'objet et les balises HTML permettant de l'afficher sous la forme voulue. L'exemple 4-3 montre l'utilisation de ces méthodes.

Exemple 4-3 – Utilisation des méthodes de mise en forme HTML de l'objet String

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Méthodes de mise en forme HTML de l'objet String</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
var someString = new String("Ceci est la chaîne de test.");
document.writeln(someString.big());
document.writeln(someString.blink());
document.writeln(someString.sup());
document.writeln(someString.strike());
document.writeln(someString.bold());
document.writeln(someString.italics());
document.writeln(someString.small());
document.writeln(someString.link('http://www.eyrolles.com'));
//]]>
</script>
</body>
</html>
```

Un de ces éléments, `blink`, a été déclaré obsolète en HTML et n'est pas supporté du tout par XHTML. Toutefois, s'il est utilisé avec `document.writeln`, la page sera validée car les programmes de validation XHTML ne voient que le code source et non le résultat. Si vous copiez ce résultat dans un nouveau document, celui-ci ne pourra pas passer la validation.

Attention

Même si l'utilisation des méthodes de mise en forme HTML ne provoque pas d'erreur, il est préférable de les éviter car elles ne reposent pas sur les CSS. Et dans tous les cas, n'utilisez jamais `blink`. Il s'agit d'un héritage du temps où les développeurs de pages Web pensaient que l'intérêt d'une page se mesurait au nombre d'éléments clignotants. (Il y en a malheureusement encore !) Rien ne fera fuir les visiteurs plus rapidement que l'utilisation de `blink`.

La meilleure façon de tester les autres méthodes de l'objet `String` consiste à créer une page Web simple, comme celle de l'exemple 4-3, puis à remplacer le code par celui correspondant à chaque méthode décrite dans la suite de cette section.

Les méthodes `charAt` et `charCodeAt` retournent respectivement le caractère et le code Unicode correspondant à une position donnée. Elles prennent pour paramètre l'index du caractère à retourner :

```
var sObj = new String("Ceci est une chaîne de test");
var sTxt = sObj.charAt(3);
document.writeln(sTxt);
```

La valeur de l'index est comptée à partir de 0. Pour retourner le quatrième caractère, il faut donc utiliser la valeur 3.

Les méthodes `substr` et `substring`, ainsi que la méthode `slice`, retournent une partie de la chaîne en fonction d'un index de départ et d'une longueur :

```
var sTx = "Ceci est une chaîne de test";
var ssTxt = sText.substr(0, 4);
document.writeln(ssTxt);
```

Comme le montre cet exemple, les méthodes de l'objet `String` peuvent être employées avec des chaînes littérales ou avec l'objet `String`. JavaScript convertit automatiquement les chaînes littérales en objets avant d'appeler les méthodes (sans pour autant modifier les chaînes originales).

Les méthodes `indexOf` et `lastIndexOf` retournent l'index d'une sous-chaîne. `indexOf` donne l'index de la première occurrence trouvée et `lastIndexOf`, celui de la dernière.

```
var sTx = "Ceci est une chaîne de test";
var iVal = sText.indexOf("t");
document.writeln(iVal);
```

L'exemple 4-2 a montré comment les chaînes pouvaient être concaténées. Pour effectuer l'opération inverse, il suffit d'employer la méthode **split**. Cette méthode prend deux paramètres. Le premier est le caractère séparateur. Le second, facultatif, est le nombre de coupures à effectuer. L'exemple 4-4 découpe une chaîne à l'emplacement des trois premières virgules. Les valeurs résultant de ce découpage sont ensuite coupées au niveau du signe **=**.

Exemple 4-4 – Utilisation de la méthode split pour découper une chaîne en éléments

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>La méthode Split</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
var inputString =
'firstName=Shelley,lastName=Powers,state=Missouri,statement="Ceci est un test, de
la méthode split"';
var arrayTokens = inputString.split(',',3);
for (var i in arrayTokens) {
   document.writeln(arrayTokens[i] + "<br />");
   var newTokens = arrayTokens[i].split('=');
   document.writeln(newTokens[1] + "<br />");
}
//]]>
</script>
</body>
</html>
```

Le résultat affiché par ce programme est le suivant :

```
firstName=Shelley
Shelley
lastName=Powers
```

```
Powers
state=Missouri
Missouri
```

Outre la méthode **split**, ce programme montre comment JavaScript gère automatiquement les conversions entre les chaînes littérales et les objets. La chaîne initiale est créée sous la forme d'une chaîne littérale affectée à une variable. La méthode **split** est ensuite appelée sur cette chaîne comme s'il s'agissait d'un objet **String** :

```
var arrayTokens = inputString.split(',',3);
```

Le moteur JavaScript traite cette ligne en convertissant la variable littérale en objet **String**, puis exécute l'appel de la méthode **split**. Il n'est donc pas nécessaire de créer un tel objet, même si votre intention est d'en appeler les méthodes. Il n'est même pas indispensable d'utiliser une variable. Les méthodes peuvent être appelées directement sur la chaîne littérale :

```
var arrayTokens = 'firstName=Shelley'.split('=');
document.writeln(newTokens[1]);
```

Le même principe est applicable à toutes les primitives, comme nous le montrerons avec **RegExp** dans la suite de ce chapitre. Bien que cet usage soit parfaitement autorisé en JavaScript, il n'est pas recommandé car il rend les programmes moins lisibles.

Info

Les tableaux, utilisés dans le cadre de l'exemple 4-4, seront étudiés dans la suite de ce chapitre.

Les méthodes **toUpperCase** et **toLowerCase** effectuent une conversion en majuscules ou en minuscules :

```
var someString = new String("Mélange de Majuscules et de Minuscules");
var newString = someString.toUpperCase(); // Conversion en majuscules
```

Ces méthodes sont très utiles pour comparer les chaînes. Elles peuvent ainsi être converties en majuscules ou en minuscules avant comparaison.

Info

Ces méthodes fonctionnent parfaitement avec les caractères accentués du français, y compris le « œ », du moins avec les navigateurs récents.

Il existe également une méthode statique nommée `fromCharCode`. Une méthode statique doit être appelée directement sur l'objet `String` et non sur une de ses instances :

```
var s = String.fromCharCode(345,99,99,76);
document.writeln(s);
```

Cette méthode prend pour argument une liste de codes Unicode séparés par des virgules et retourne la chaîne correspondante. Toutefois, il est également possible d'insérer des codes Unicode directement dans une chaîne, comme nous l'avons vu au chapitre 2.

Les dernières méthodes de l'objet `String` reposent sur l'utilisation des expressions rationnelles. JavaScript a un objet dédié à cet usage, `RegExp`, auquel la prochaine section est consacrée.

RegExp et les expressions rationnelles

Les *expressions rationnelles* sont des chaînes de caractères qui forment un motif susceptible d'être employé pour sélectionner des chaînes, effectuer des remplacements ou localiser des sous-chaînes. La plupart des langages supportent les expressions rationnelles et JavaScript ne fait pas exception.

Une expression rationnelle peut être créée explicitement à l'aide de l'objet `RegExp`, mais il est également possible d'utiliser une simple chaîne littérale. L'exemple suivant montre l'utilisation de l'objet `RegExp` :

```
var searchPattern = new RegExp('+s');
```

alors que la ligne suivante emploie une expression littérale :

```
var searchPattern = /+s/;
```

Dans les deux cas, le signe `+` est utilisé pour trouver toute combinaison de un ou plusieurs « s » dans une chaîne. Les barres obliques délimitent l'expression rationnelle en indiquant qu'il ne s'agit pas d'une chaîne de caractères ordinaire.

Les méthodes de RegExp : test et exec

L'objet `RegExp` n'a que deux méthodes dignes d'intérêt : `test` et `exec`. La méthode `test` détermine si une chaîne passée en paramètre correspond à l'expression régulière. Dans l'exemple suivant, le motif `/JavaScript est génial/` est comparé à une chaîne :

```
var re = /JavaScript est génial/;
var str = "JavaScript est génial";
if (re.test(str)) document.writeln("Il semble en effet qu'il le soit.");
```

La comparaison tient compte des majuscules et des minuscules. Si la chaîne testée est "Javascript est génial", la comparaison échoue. Pour qu'elle soit effectuée en confondant les majuscules et les minuscules, il suffit d'ajouter la lettre `i` après la seconde barre oblique :

```
var re = /JavaScript est génial/i;
```

Il est également possible d'employer les indicateurs `g` pour effectuer une comparaison globale, et `m` pour une comparaison sur plusieurs lignes. Dans le cas de l'utilisation de l'objet **`RegExp`**, l'indicateur est passé en second paramètre :

```
var searchPattern = new RegExp('+s', 'g');
```

Dans le code suivant, la méthode **`exec`** recherche le motif **`/JS*/`** dans une chaîne entière (`g`), sans tenir compte des majuscules et des minuscules (`i`) :

```
var re = /JavaScript*/ig;
var str = "cfdsJS *(&YJSjs 888JS";
var resultArray = re.exec(str);
while (resultArray) {
  document.writeln(resultArray[0]);
  resultArray = re.exec(str);
}
```

Le motif employé dans l'expression rationnelle est la lettre `J` suivie d'un nombre quelconque de `S`. L'indicateur `i` étant présent, les majuscules et les minuscules sont confondues. La chaîne `js` est donc trouvée également. À cause de la présence de l'indicateur `g`, l'index est placé à l'endroit où la dernière chaîne a été trouvée à chaque appel, et chaque appel retourne donc l'occurrence suivante. Par conséquent, quatre occurrences sont affichées. Lorsque qu'il n'y a plus d'occurrence, la valeur `null` est affectée au tableau et la boucle se termine.

Il existe d'autres indicateurs que ceux utilisés ici, comme les signes `+` ou `*` employés dans l'exemple précédent.

Dans la plupart des livres, on trouve un tableau présentant chacun de ces caractères et fournissant quelques exemples dans lesquels une combinaison d'indicateurs est mise en œuvre dans un motif long et complexe. De ce fait, de nombreux utilisateurs éprouvent des difficultés à mettre au point les motifs et leurs programmes ne donnent pas toujours les résultats souhaités. Les expressions rationnelles sont un outil suffisamment important pour justifier plusieurs exemples

concrets, du plus simple au plus complexe. Si vous maîtrisez déjà les expressions rationnelles, vous pouvez passer à la section suivante, sauf si vous êtes intéressé par une révision.

Bien que les méthodes de `RegExp` soient employées dans les applications, les expressions rationnelles sont surtout mises en œuvre à l'aide des méthodes `replace`, `match` et `search` de l'objet `String`. Les exemples donnés dans la suite de ce chapitre sont basés sur ces méthodes.

Utiliser les expressions rationnelles

Le premier caractère est la barre oblique inverse \ (*backslash* en anglais), également nommée *caractère d'échappement* car elle sert à modifier la signification des caractères suivants. Dans les expressions rationnelles JavaScript, ce caractère a deux utilisations. S'il précède un caractère littéral, par exemple la lettre `s`, il change de signification. `\s` correspond à un caractère d'espace non imprimable (espace, tabulation, fin de ligne, etc.). En revanche, si la barre oblique inverse est employée devant un caractère spécial, comme le signe `+`, ce caractère est alors traité comme un caractère littéral.

L'exemple 4-5 recherche un espace non imprimable suivi d'un astérisque et remplace cette combinaison par un tiret. Le motif recherché est `\s\*`. Normalement, l'astérisque signifie « 0, une ou plusieurs occurrences », mais nous voulons ici le traiter comme un caractère littéral.

Exemple 4-5 – Utilisation du caractère d'échappement dans les expressions rationnelles

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Le caractère d'échappement dans les expressions rationnelles</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
var regExp = /\s\*/g;
var str = "Ceci *est *une *chaîne *de *test";
var resultString = str.replace(regExp,'-');
document.writeln(resultString);
//]]>
</script>
</body>
</html>
```

Le résultat affiché par ce programme est :

```
Ceci-est-une-chaîne-de-test
```

L'expression employée ici est très utile. Si vous souhaitez remplacer dans une chaîne toutes les occurrences d'espaces par des tirets, sans tenir compte du caractère suivant, utilisez simplement l'expression **/\s/g**.

Quatre indicateurs peuvent servir à définir le nombre d'occurrences : l'astérisque (*****) signifie zéro, une ou plusieurs occurrences. Le signe plus (**+**) signifie une ou plusieurs occurrences. Le point d'interrogation (**?**) correspond à zéro ou une occurrence. Le point signifie un caractère quelconque.

Attention

Il existe deux motifs intéressants : **/.*/** et **/.*?/**. Dans le premier cas, le point signifie n'importe quel caractère et l'astérisque correspond à un nombre quelconque. Par exemple, pour trouver n'importe quel texte entre guillemets, vous pourriez employer **/".*"/**. Si la chaîne examinée est :

```
test = "un" ou "plusieurs" caractères
```

La chaîne trouvée est **"un" ou "plusieurs"**, c'est-à-dire qu'elle s'étend du premier au dernier guillemet. Dans ce cas, l'emploi de **/.*?/** permet de trouver :

```
"test"
```

ce qui correspond au but recherché.

Dans l'exemple 4-6, la méthode **search** de l'objet **String** est employée pour rechercher une date constituée d'un numéro de jour suivi d'un espace, puis d'un nom de mois, un espace et un numéro d'année, le tout précédé par le caractère deux-points (**:**).

Exemple 4-6 – Motifs de caractères répétés

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Trouver une date</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
```

Exemple 4-6 – Motifs de caractères répétés (*suite*)

```
var regExp = /:\s*\d+\s+\D*\s+\d+/;
var str = "Ceci est une date : 12 novembre 2007";
var resultString = str.match(regExp);
document.writeln("Date " + resultString);
//]]>
</script>
</body>
</html>
```

Le motif commence par **:**, suivi de **\s***, correspondant à un nombre quelconque d'espaces. **\d+** correspond ensuite à un ou plusieurs chiffres, puis **\s+** à un ou plusieurs espaces. On trouve ensuite **\D***, qui correspond à zéro, un ou plusieurs caractères non numériques (autres que des chiffres). Le motif se termine par un ou plusieurs espaces (**\s+**) suivi(s) de un ou plusieurs chiffres (**\d+**).

La date trouvée est affichée sous la forme :

```
Date : 12 novembre 2007
```

Le motif **\D** correspond à tout caractère autre qu'un chiffre. Une autre façon de définir cette condition consiste à employer les crochets carrés pour définir une plage et de la faire précéder du caractère **^** :

```
[^0-9]
```

La même technique permet de trouver un chiffre. Il suffit d'omettre le caractère **^** :

```
[0-9]
```

Si les caractères à trouver appartiennent à plusieurs plages, il suffit de les lister :

```
[A-Za-z]
```

Une plage peut être réduite à un seul caractère :

```
[A-Za-zâäéèêëîïôöùûüç]
```

Le motif utilisé pour notre exemple peut donc être réécrit :

```
var regExp = /:\s*[0-9]+\s+[a-zéû]*\s+[0-9]+/;
```

(Les noms de mois n'utilisent que deux caractères accentués et ne prennent pas de majuscules en français.)

Le signe `^` est employé, ainsi que le caractère `$` pour trouver les débuts et fins de lignes. Le signe `^`, en dehors des crochets carrés, correspond à un début de ligne, et le signe `$` à une fin de ligne.

Dans l'exemple suivant, la séquence n'est pas trouvée car le caractère ne se trouve pas en début de ligne :

```
var regExp = /^Cel/i;
var str = "Cette chaîne est celle utilisée pour le premier test."
```

Alors qu'elle l'est dans cet exemple :

```
var regExp = /^Cel/i;
var str = "Celle-ci est utilisée pour le second test."
```

Si l'indicateur `m` est employé pour indiquer que la recherche est faite sur plusieurs lignes, la recherche suivante est réussie :

```
var regExp = /^Cel/im;
var str = "Cette chaîne est\ncelle utilisée pour le premier test."
```

Le caractère de fin de ligne fonctionne de la même façon. La recherche suivante échoue :

```
var regExp = /fin$/i;
var str = "La fin est proche";
```

mais celle-ci est réussie :

```
var regExp = /fin$/i;
var str = "C'est vraiment la fin";
```

ainsi que celle-ci :

```
var regExp = /fin$/im;
var str = "La fin\nest proche";
```

L'utilisation des parenthèses a une signification particulière. En effet, celles-ci permettent d'indiquer que la chaîne trouvée doit être mémorisée et placée dans le tableau de résultats :

```
var rgExp = /(\D*[0-9]*)/;
var str = "Exercice 13 du chapitre 4";
var resultArray = str.match(rgExp);
document.writeln(resultArray);
```

Cet exemple affiche deux fois la chaîne **Exercice 13** en séparant les deux occurrences par une virgule, indiquant qu'il s'agit de deux éléments distincts du tableau. Le premier élément est la chaîne trouvée et le second est la chaîne mémorisée en raison de la présence des parenthèses. Les parenthèses peuvent également être placées autour d'une partie du motif, par exemple **/\D*([0-9]*)/**. Le résultat est alors :

```
Exercice 13,13
```

Les parenthèses servent également à déplacer des éléments lorsqu'elles sont employées avec les caractères spéciaux **$1**, **$2**...**$9** qui permettent de mémoriser des sous-chaînes. L'exemple 4-7 trouve des paires de chaînes séparées par des tirets et les inverse.

Exemple 4-7 – Déplacer des chaînes avec les expressions rationnelles

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Déplacement de chaînes</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
var rgExp = /(\w*)-*(\w*)/
var str = "Java--Script";
var resultStrng = str.replace(rgExp,"$2-$1");
document.writeln(resultStrng);
//]]>
</script>
</body>
</html>
```

Ce programme affiche :

```
Script-Java
```

Notez que les deux tirets ont aussi été remplacés par un seul. Dans cet exemple, nous avons également introduit un motif très utile, `\w`, qui correspond à n'importe quel caractère alphanumérique ou au trait de soulignement (_). C'est l'équivalent de `[A-Za-z0-9_]`. La séquence inverse est `\W` qui correspond à tous les autres caractères.

Les derniers caractères utilisables dans les expressions rationnelles que nous examinerons sont le caractère `|` et les accolades `{}`. La barre verticale indique un choix. La séquence suivante signifie « la lettre a ou la lettre b » :

```
a|b
```

Il est possible de grouper plusieurs options :

```
a|b|c
```

Les accolades indiquent une répétition. Ainsi, la séquence suivante correspond à deux « s » :

```
s{2}
```

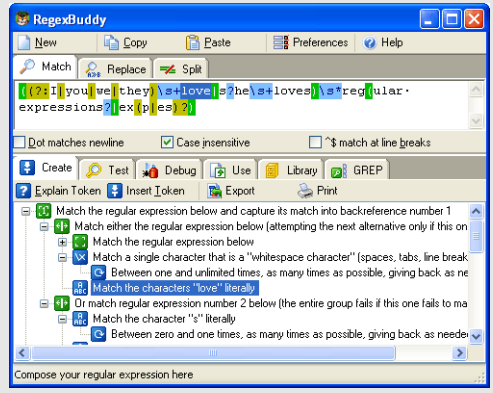
Les expressions rationnelles sont extrêmement utiles pour valider les champs de formulaires, comme nous le verrons au chapitre 7.

Mieux utiliser les expressions rationnelles

Dans ce chapitre, nous n'avons fait qu'aborder rapidement les expressions rationnelles, juste assez pour en comprendre le principe. Si vous utilisez de nombreux formulaires avec vos pages Web, ou avec Ajax, vous trouverez des informations complémentaires dans le livre *Maîtrise des expressions régulières* de Jeffrey E.F. Friedl (O'Reilly).

De nombreux outils sont disponibles dans ce domaine, et nous vous suggérons d'en essayer quelques-uns. Si vous travaillez sous Unix ou Mac OS X, **`grep`** est un utilitaire qui permet de trouver des chaînes dans des fichiers. Il existe aussi une version Windows nommée *PowerGrep*.

Figure 4-2
Le programme RegexBuddy.



Mieux utiliser les expressions rationnelles (*suite*)

Certains outils permettent également de tester les expressions rationnelles. Sur Macintosh, vous pouvez utiliser `CocoaRegex`, un programme qui peut être téléchargé gratuitement. Il existe des outils équivalents pour Windows et Linux. Certains peuvent même être employés en ligne, comme celui proposé par le site *regular-expressions.info*, où vous trouverez également le programme *RegexBuddy*, illustré à la figure 4-2. Pour trouver ces outils, recherchez « javascript regular expression tools ». Quant à la recherche de « javascript regular expression », elle affiche une liste de sites consacrés à ce sujet, sites qui proposent des listes de motifs utiles ainsi que des cours d'initiation.

Des objets utiles : Date et Math

Les objets JavaScript `Date` et `Math` offrent un accès à des fonctionnalités que vous ne soupçonnez peut-être pas, jusqu'au moment où vous vous demandez « Comment peut-on bien faire pour... ». Ils ont été créés spécifiquement pour permettre de manipuler des dates et des expressions mathématiques. Ni plus, ni moins.

Date

L'objet `Date` permet de créer une date et d'accéder à tous ces composants : année, mois, jour, heure, minute, seconde, etc. La création d'une date sans paramètre permet d'obtenir la date et l'heure de l'ordinateur sur lequel fonctionne le navigateur :

```
var dtNow = new Date();
```

Au moment où ces lignes sont écrites (ce n'est pas une heure pour travailler !), le résultat est égal à :

```
Wed Nov 29 2006 21:20:07 GMT+0100
```

Il est également possible d'utiliser un paramètre pour créer une date spécifique. Il « suffit » d'indiquer le nombre de millisecondes écoulées depuis le 1er janvier 1970 à 12:00:00 :

```
var dtMilliseconds = new Date(5999000920);
document.writeln(dtMilliseconds.toUTCString());
```

Ces lignes affichent le résultat suivant :

```
Wed, 11 Mar 1970 10:23:20 GMT
```

Vous pouvez aussi créer une date en utilisant une chaîne de caractères, à condition d'utiliser le format correct :

```
var nowDt = new Date("March 12, 1980 12:20:25");
```

Les heures et les minutes sont facultatives. Si elles ne sont pas indiquées, la valeur par défaut est 0. La date peut être spécifiée aussi sous forme numérique, dans l'ordre année, mois (de 0 à 11), jour, heure, minute, seconde et milliseconde :

```
var newDt = new Date(1977,12,23);
var newDt = new Date(1977,11,24,19,30,30,30);
```

Une fois l'objet **Date** obtenu, plusieurs méthodes (dont certaines sont statiques) permettent d'accéder à ses champs et de les manipuler de diverses façons.

Les méthodes statiques doivent être appelées sur l'objet **Date** lui-même, et non sur une de ses instances. **Date.now** donne la date et l'heure du moment. **Date.parse** donne le nombre de millisecondes écoulées depuis le 1er janvier 1970. **Date.UTC** retourne également le nombre de millisecondes de la manière suivante :

```
var numMs = Date.UTC(1977,11,24,19,30,30,30);
```

Les méthodes de l'objet **Date** permettent de lire les différents champs de la date :

- **getFullYear**
- **getHours**
- **getMilliseconds**
- **getMinutes**
- **getMonth**
- **getSeconds**
- **getYear**

Les équivalents UTC sont :

- **getUTCFullYear**
- **getUTCHours**
- **getUTCMilliseconds**
- **getUTCMinutes**
- **getUTCMonth**
- **getUTCSeconds**
- **getUTCYear**

La plupart des méthodes **get** ont un équivalent **set** qui permet de modifier ces champs. Par exemple, **setYear** modifie l'année, et **setUTCMonth** fait de même pour le mois UTC.

Parmi les méthodes dont la signification n'est pas aussi évidente, **getDate** retourne le numéro du jour du mois, alors que **getDay** retourne le jour de la semaine, en commençant à 0 pour dimanche :

```
var dtNow = new Date();
alert(dtNow.getDay());
```

La méthode **getTimezoneOffset** retourne le nombre de minutes (+ ou -) de décalage entre le temps local de l'ordinateur et le temps UTC. Ce texte étant écrit à Paris en hiver, l'affichage obtenu est -60.

Six méthodes permettent de convertir une date en chaîne de caractères mise en forme :

toString

Affiche l'heure locale.

toGMTString

Affiche l'heure GMT.

toLocaleDateString *et* **toLocaleTimeString**

Affichent respectivement l'heure et la date au format d'un pays.

toLocaleString

Convertit la date en fonction du format local.

toUTCString

Affiche la date au format UTC.

L'exemple 4-8 montre l'utilisation de ces méthodes et de celles que nous avons présentées précédemment.

Exemple 4-8 – Utilisation des méthodes de mise en forme de la date

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Mise en forme de la date</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head></body>
<script type="text/javascript">
```

Exemple 4-8 – Utilisation des méthodes de mise en forme de la date (*suite*)

```
//<![CDATA[
var dtNow = new Date();
// set day, month, year
dtNow.setDate(18);
dtNow.setMonth(10);
dtNow.setYear(1954);
dtNow.setHours(7);
dtNow.setMinutes(2);
// output formatted
document.writeln(dtNow.toString() + "<br />");
document.writeln(dtNow.toLocaleString() + "<br />");
document.writeln(dtNow.toLocaleDateString() + "<br />");
document.writeln(dtNow.toLocaleTimeString() + "<br />");
document.writeln(dtNow.toGMTString() + "<br />");
document.writeln(dtNow.toUTCString());

//]]>
</script>
</body>
</html>
```

Étant donné le nombre d'options, il est parfois délicat de choisir la bonne. En règle générale, il est préférable d'utiliser le format local du navigateur de l'utilisateur pour toutes les actions isolées, comme passer une commande sur un site marchand. En revanche, dans le cas d'une audience internationale, par exemple pour indiquer l'heure des commentaires sur un blog, il est préférable d'utiliser le temps UTC afin que tous les visiteurs aient une référence commune.

Attention

L'objet **`Date`** se comporte de la même façon sur tous les navigateurs, sauf en ce qui concerne la méthode **`getYear`**. Cette méthode n'était pas compatible avec le passage à l'an 2000 et retournait l'année moins 1900. La spécification ECMA a introduit une nouvelle méthode nommée **`getFullYear`**. Firefox et les autres navigateurs compatibles ECMAScript Version 3 disposent de cette méthode. Internet Explorer 6, en revanche, a une méthode **`getYear`** qui renvoie la date sur quatre chiffres.

Math

JavaScript fait la différence entre l'arithmétique et les mathématiques, puisque les opérateurs arithmétiques (que nous avons étudiés au chapitre 2), ne sont pas associés à l'objet **`Math`**.

Celui-ci dispose de méthodes et de propriétés mathématiques telles que `LN10`, qui donne le logarithme décimal, et `log(x)` qui retourne le logarithme naturel. En revanche, il n'est pas concerné par les opérations arithmétiques simples comme l'addition et la soustraction.

Contrairement aux autres objets JavaScript, `Math` a uniquement des propriétés statiques. Cela signifie qu'il n'est jamais nécessaire d'en créer une instance pour accéder à ses fonctionnalités. Les méthodes et propriétés sont accessibles à partir de l'objet lui-même :

```
var newValue = Math.SQRT1;
```

Comme pour les autres objets, les propriétés sont accessibles à l'aide de l'opérateur de propriété, représenté par un point :

```
Math.propriété
```

Les propriétés suivantes de l'objet `Math` représentent des valeurs numériques. Elles sont données dans l'ordre de la spécification ECMA–262 :

`E`

Valeur de *e*, la base des logarithmes naturels (ou *logarithmes népériens*).

`LN10`

Le logarithme naturel de 10.

`LN2`

Le logarithme naturel de 2.

`LOG2E`

Approximation de la réciproque de `LN2`, c'est-à-dire le logarithme en base 2 de *e*.

`LOG10E`

Approximation de la réciproque de `LN10`, c'est-à-dire le logarithme en base 10 de *e*.

`PI`

La valeur de π.

`SQRT1_2`

La racine carrée de ½.

`SQRT2`

La racine carrée de 2.

La programmation de calculs mathématiques dépend de l'environnement, et en particulier de la manière dont les fonctions mathématiques sont implémentées par le navigateur et par le système d'exploitation, ainsi que de l'architecture de l'ordinateur. On peut donc s'attendre à de petites variations dans les résultats des fonctions trigonométriques, mais cela n'altère pas leur utilité dans le contexte qui nous intéresse.

Les méthodes mathématiques

Les méthodes de l'objet **Math** sont d'une utilisation relativement simple. L'argument passé en paramètre est d'abord converti au format correct. Vous n'avez aucune conversion à effectuer explicitement.

La fonction **abs** prend un argument représentant une valeur numérique et retourne sa valeur absolue. Dans l'exemple suivant, **pVal** prend la valeur 3.45 :

```
var nVal = -3.45;
var pVal = Math.abs(nVal);
```

Plusieurs méthodes sont disponibles pour les calculs trigonométriques : **sin**, **cos**, **tan**, **acos**, **asin**, **atan** et **atan2**. Elles retournent respectivement le sinus, le cosinus, la tangente, l'arc cosinus, l'arc sinus, l'arc tangente et l'angle défini par un point x et l'origine. Leurs arguments sont numériques, comme indiqué ci-après :

Math.sin(x)

Un angle, en radians.

Math.cos(x)

Un angle, en radians.

Math.tan(x)

Un angle, en radians.

Math.acos(x)

Un nombre compris entre -1 et 1.

Math.asin(x)

Un nombre compris entre -1 et 1.

Math.atan(x)

Un nombre.

Math.atan2(x, y)

Les coordonnées *x* et *y* d'un point.

La méthode **Math.ceil** arrondit une valeur à l'entier supérieur. Dans l'exemple suivant, **pVal** prend la valeur 4 :

```
var nVal = 3.45;
var pVal = Math.ceil(nVal);
```

Dans l'exemple suivant, **pVal** prend la valeur -3 :

```
var nVal = -3.45;
var pVal = Math.ceil(nVal);
```

La méthode **Math.floor** arrondit une valeur à l'entier inférieur. Dans l'exemple suivant, **pVal** prend la valeur 3 :

```
var nVal = 3.45;
var pVal = Math.floor(nVal);
```

Dans l'exemple suivant, **pVal** prend la valeur -4 :

```
var nVal = -3.45;
var pVal = Math.floor(nVal);
```

La méthode **Math.round** retourne l'entier le plus proche, positif ou négatif. 3.45 est arrondi à 3 et -3.85 est arrondi à -4.

Math.exp(x) calcule la valeur de *e*, la base des logarithmes naturels, à la puissance de son argument :

```
var nVal = Math.exp(4); // égal à e^4
```

Math.pow élève une valeur à une puissance :

```
var nVal = Math.pow(3, 2); // égal à 3^2 ou 9
```

Math.min et **Math.max** comparent plusieurs valeurs et retournent la plus petite ou la plus grande :

```
var nal = 1.45;
var nVal2 = 4.5;
var nVal3 = -3.33;
var nResult = Math.min(nVal, nVal2, nVal3); // retourne -3.33
var nResult2 = Math.max(, nVal2, nVal3); // retourne 4.5
```

La dernière méthode, **Math.random**, retourne un nombre aléatoire compris entre 0 (inclus) et 1 (exclu) :

```
var nValue = Math.random();
```

Les limites de cette méthode pourraient vous décourager de l'utiliser. Toutefois, il suffit de multiplier le résultat par 10, 100 ou une valeur quelconque pour obtenir un résultat plus utile. Il n'est pas possible de spécifier un intervalle, mais ce résultat peut être obtenu à l'aide d'une boucle, comme le montre l'exemple 4-9 :

Exemple 4-9 – Un générateur de nombres aléatoires

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Nombres aléatoires</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script type="text/javascript">
//<![CDATA[
var quoteArray = new Array(5);
quoteArray[0] = "Chaque homme doit inventer son chemin. [Sartre]";
quoteArray[1] = "L'homme est imparfait, mais ce n'est pas étonnant quand on pense
à l'époque où il a été créé. [Alphonse Allais]";
quoteArray[2] = "Méfiez-vous des gens qui ne s'ennuient jamais. On s'ennuie
toujours avec eux. [Gilbert Cesbron]";
quoteArray[3] = "Quand quelqu'un dit: je me tue à vous le dire! Laisse-le mourir.
[Jacques Prévert]";
quoteArray[4] = "Il faut user de tout avec modération, surtout de la modération.
[Richard Bates]";
function getQuote() {
   iValue = Math.random(); // nombre aléatoire dans l'intervalle [0, 1[
   alert(iValue);
   iValue *= 5; // multiplié par 5 pour un résultat dans [0, 5[
   alert(iValue);
   iValue = Math.floor(iValue); // arrondi à l'entier inférieur : [0, 4]
   alert(iValue);
   alert(quoteArray[iValue]);
}
//]]>
</script>
</head>
```

Exemple 4-9 – Un générateur de nombres aléatoires (*suite*)

```
<body onload="getQuote();">
</body>
</html>
```

Un tableau est créé pour contenir cinq citations. La fonction `getQuote` est appelée chaque fois que la page est chargée. Elle utilise plusieurs fonctions des objets `Number` et `Math` pour déterminer une valeur aléatoire comprise entre 0 et 4 (bornes incluses). Cette valeur est ensuite utilisée pour accéder à un des éléments du tableau et l'afficher. (Les étapes intermédiaires sont également affichées.)

Les tableaux JavaScript

En JavaScript, lorsqu'il y a un objet, il y a également un littéral. Comme nous l'avons vu aux chapitres précédents, il y a un objet `String` ainsi que des chaînes littérales. La même chose est vraie pour `Boolean` et les valeurs `true` et `false`, ou pour `Number` et les valeurs numériques. Dans le cas des expressions rationnelles, nous avons rarement utilisé l'objet `RegExp`, mais plutôt des expressions littérales. Il en est de même pour les tableaux.

Construction des tableaux

Un tableau JavaScript est un objet, au même titre que `String` ou `Math`. En tant que tel, il est créé à l'aide d'un constructeur :

```
var newArray = new Array('un', 'deux');
```

Un tableau est aussi une valeur littérale et l'utilisation de l'objet `Array` n'est pas nécessaire :

```
var newArray = ['un', 'deux'];
```

Dans ce dernier cas, le moteur JavaScript convertit le tableau littéral en objet de type `Array` et affecte le résultat à la variable. Les éléments du tableau sont ensuite accessibles à l'aide de leur index, qui représente leur position dans le tableau comptée à partir de 0 :

```
alert(newArray[0]); // affiche 'un'
```

L'index commençant à 0, la valeur maximale est inférieure de 1 à sa longueur. Dans un tableau de 5 éléments, les index vont de 0 à 4.

Les tableaux peuvent avoir plusieurs dimensions. JavaScript implémente ce concept sous forme de tableaux de tableaux : dans un tableau à deux dimensions, chaque élément d'un tableau à une dimension est lui-même un tableau à une dimension. L'exemple suivant crée un tableau à deux dimensions :

```
var threedPoints = new Array();
threedPoints[0] = new Array(1.2, 3.33, 2.0);
threedPoints[1] = new Array(5.3, 5.5, 5.5);
threedPoints[2] = new Array(6.4, 2.2, 1.9);
```

Si le tableau contient les coordonnées *x*, *y* et *z* d'un point dans l'espace, l'accès à la coordonnée *z* du troisième point peut être effectué de la manière suivante :

```
var newZPoint = threedPoints[2][2]; // les index commencent à 0
```

Attention

Ne confondez pas : bien que les données représentées dans ce tableau soient des points de l'espace à trois dimensions, le tableau ne comporte que deux dimensions.

Il est très simple d'ajouter des dimensions :

```
threedPoints[2][2] = new Array(4.4, 4.6, 44); // etc.
var newthreedZPoint = threedPoints[2][2][1];
```

Comme vous pouvez le voir, le nombre d'éléments d'un tableau n'est pas statique. Il n'est pas nécessaire de le connaître lors de sa création. Vous pouvez déclarer un tableau avec un certain nombre d'éléments, puis en ajouter lorsque cela est nécessaire. Il est également possible de forcer la taille d'un tableau en commençant par y placer son dernier élément :

```
var testArray = new Array();
testArray[99] = 'une valeur'; // le tableau comporte maintenant 100 positions
```

Pour connaître la taille d'un tableau, il suffit de consulter sa propriété `length` :

```
alert(testArray.length); // affiche 100
```

La propriété `length` ne donne la longueur du tableau que dans une dimension :

```
alert(threedPoints[2][2].length); // affiche 3
alert(threedPoints[2].length); // affiche 3
alert(threedPoints.length); // affiche 3
```

Outre **length**, l'objet **Array** a d'autres propriétés et méthodes. La méthode **splice**, par exemple, permet d'insérer ou de supprimer des éléments dans un tableau, ce qui est très pratique. Dans l'exemple suivant, cette méthode est employée pour ajouter deux éléments, puis en retirer deux à partir de l'index 2 (correspondant au troisième élément) :

```
var fruitArray = new Array('pomme', 'pêche', 'orange', 'citron', 'prune',
'cerise');
var removed = fruitArray.splice (2, 2, 'melon, banane');
document.writeln(removed + "<br />");
document.writeln(fruitArray);
```

Ce programme affiche :

```
orange,citron
pomme,pêche,melon,banane,prune,cerise
```

Les éléments supprimés sont retournés par la méthode **splice** sous forme d'un tableau.

La méthode **slice** permet d'obtenir une « tranche » d'un tableau :

```
fruitArray.slice(2,4); // melon,banane,prune
```

La méthode **concat** permet de mettre bout à bout deux tableaux :

```
var newFruit = fruitArray.concat(removed); // pomme,pêche,melon,
banane,prune,cerise,orange,citron
```

Ni **concat** ni **slice** n'altèrent le tableau original. Ces méthodes retournent simplement un nouveau tableau contenant le résultat.

Dans ces exemples, nous avons affiché directement les tableaux Dans ce cas, JavaScript les convertit en chaînes de caractères en utilisant la virgule comme séparateur par défaut. Vous pouvez utiliser la méthode **join** pour obtenir un séparateur différent :

```
var strng = fruitArray.join();
```

Il est également possible d'inverser l'ordre des éléments à l'aide de la méthode **reverse** :

```
fruitArray.reverse();
```

Dans certains cas, l'ordre des éléments d'un tableau est sans importance. Dans d'autres, en revanche, la bonne exécution du programme implique que l'ordre soit préservé. C'est le cas, par exemple, si le tableau sert de support à une structure de *queue*. Plusieurs méthodes

permettent d'utiliser les tableaux, par exemple *queue* ou *liste*, comme nous allons le voir dans la prochaine section.

Les queues FIFO

Les tableaux peuvent être employés pour gérer des queues de type FIFO (*First In First Out*). Dans ce type de structure, le premier élément entré est le premier disponible. Quatre méthodes de l'objet **Array** permettent de gérer les queues et les listes : **push**, **pop**, **shift** et **unshift**.

La méthode **push** ajoute un élément à la fin d'un tableau, alors que la méthode **unshift** fait de même au début. Toutes deux retournent la longueur du tableau.

La méthode **pop** retire le dernier élément et la méthode **shift** retire le premier. Toutes deux retournent l'élément supprimé.

Ces quatre méthodes modifient le tableau de façon permanente. L'exemple 4-10 montre la réalisation d'une queue FIFO en JavaScript.

Exemple 4-10 – Une queue FIFO utilisant les méthodes de l'objet Array

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>FIFO</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
<body>
<script type="text/javascript">
//<![CDATA[
//Création d'une queue FIFO et insertion d'éléments au moyen de la méthode push
var fifoArray = new Array();
fifoArray.push("Pomme");
fifoArray.push("Banane");
var ln = fifoArray.push("Cerise");
// Affichage de la longueur et du contenu du tableau
document.writeln("Longueur : " + ln + " - Contenu du tableau : " + fifoArray + "<br
/>");
// Suppression d'éléments à l'aide de la méthode pop
for (var i = 0; i < ln; i++) {
   document.writeln(fifoArray.shift() + "<br />");
}
// Affichage de la longueur
document.writeln("Longueur : " + fifoArray.length + "<br /><br />");
```

Exemple 4-10 – Une queue FIFO utilisant les méthodes de l'objet Array (*suite*)

```
// La même chose avec shift et unshift
var fifoNewArray = new Array();
fifoNewArray.unshift("Apprendre");
fifoNewArray.unshift("Java");
ln = fifoNewArray.unshift("Script");
document.writeln("Longueur : " + ln + " - Contenu du tableau : " + fifoNewArray +
"<br />");
// unshift
for (i = 0; i < ln; i++) {
  document.writeln(fifoNewArray.pop() + "<br />");
}
document.writeln("Longueur : " + fifoNewArray.length );i
//]]>
</script>
</body>
</html>
```

La première chose remarquable dans cet exemple est l'association de **shift** et **push** d'une part, et de **unshift** et **pop** d'autre part. La méthode **push** ajoute un élément en fin de tableau (dernier entré) et la méthode **shift** retire un élément en début de tableau (premier entré). Utilisée avec **pop**, la méthode **push** permet de créer une structure de pile *LIFO* (*Last In First Out*), ce qui n'est pas le but recherché ici.

Il en va de même pour **unshift** et **pop**. La méthode **unshift** ajoute un élément au début du tableau et la méthode **pop** en retire un à la fin, ce qui convient également dans le cadre de notre exemple. En d'autres termes, il est possible de réaliser une structure FIFO en utilisant le couple **unshift/pop** ou le couple **push/shift**. La différence est le sens du tableau. Pour une structure LIFO, on utilisera le couple **push/pop** (sommet de la pile à la fin du tableau) ou **unshift/shift** (sommet de la pile au début du tableau). L'avantage de la seconde méthode est que la partie active de la pile se trouve toujours à l'index 0.

Avec le couple **unshift/pop**, les nouveaux éléments sont ajoutés au début du tableau. Les éléments retirés le sont en fin de tableau. L'ordre d'entrée des éléments est ainsi conservé.

Le résultat affiché par ce programme est le suivant :

```
Longueur : 3 - Contenu du tableau : Pomme,Banane,Cerise
Pomme
Banane
Cerise
Longueur : 0
```

```
Longueur : 3 - Contenu du tableau : Script,Java,Apprendre
Apprendre
Java
Script
Longueur : 0
```

L'exemple 4-10 montre également comment une boucle peut être employée pour parcourir un tableau. Plutôt que d'afficher explicitement chaque élément, la boucle effectue une itération sur chaque élément. Cette technique permet d'afficher le contenu d'un tableau dont la longueur est variable.

Le plus souvent, la variable utilisée pour parcourir un tableau est incrémentée (ou décrémentée pour inverser l'ordre des éléments) à chaque itération :

```
for (var i = 0; i < someArray.length; i++) {
  alert(someArray[i]);
}
```

Toutefois, rien ne vous oblige à utiliser l'index. Une itération en sens inverse peut être obtenue de la manière suivante :

```
for (var i = someArray.length - 1; i >=0; i--) {
  alert(someArray[i]);
}
```

Une autre possibilité consiste à employer la boucle **`for...in`** pour accéder à chaque élément du tableau :

```
var programLanguages = new Array('C++', 'Pascal', 'FORTRAN', 'BASIC', 'C#',
'Java', 'Perl', 'JavaScript');
for (var itemIndex in programLanguages) {
  document.writeln(programLanguages[itemIndex] + "<br />");
}
```

Il existe d'autres méthodes associées à l'objet **`Array`**, mais elles nécessitent l'utilisation d'une fonction *callback* qui sera étudiée au chapitre 5. Pour l'instant, nous allons nous intéresser aux tableaux associatifs.

Les tableaux associatifs : des tableaux qui n'en sont pas vraiment

Nous avons abordé les tableaux associatifs pour la première fois au chapitre 3. Contrairement aux tableaux que nous venons de voir, les tableaux associatifs n'utilisent pas d'index pour accéder à leurs éléments. Il n'est donc pas possible d'utiliser la syntaxe :

```
assocArray[1]
```

Les tableaux associatifs peuvent être créés à l'aide du constructeur **Array**, mais cette pratique n'est pas recommandée à cause de la confusion que cela entraîne entre les deux types de tableaux. La technique conseillée est la suivante :

```
var assocArray = new Object();
assocArray["un"] = "Premier élément";
assocArray["deux"] = "Deuxième élément";
```

Pour accéder aux éléments du tableau, nous utilisons la même syntaxe que pour accéder aux propriétés des objets tels que **Math** ou **Date** :

```
alert(assocArray.un);
```

Les tableaux associatifs seront mis en œuvre dans les derniers chapitres de ce livre. Nous n'allons donc pas approfondir le sujet pour l'instant. Il est toutefois important de se souvenir que lorsque l'on parle de tableaux en JavaScript, il s'agit le plus souvent de la version qui utilise les index. Pour faire référence aux autres types de tableaux, on utilise explicitement l'expression *tableaux associatifs*, ou on parle tout simplement d'*objets*.

Questions

1. Les listes qui utilisent la virgule comme séparateur sont un format de données très courant. Comment pouvez-vous créer un tableau à partir d'une telle liste ?
2. Le caractère spécial **\b** définit une limite de mot, alors que **\B** correspond à l'inverse, c'est-à-dire à une position à l'intérieur d'un mot. Définissez une expression rationnelle permettant de trouver toutes les occurrences du mot **est** et de les remplacer par **était** dans la phrase suivante :

```
Il est temps de mettre une veste car le temps n'est plus estival.
```

3. Créez un programme qui donne la date du jour, puis ajoute une semaine, avant d'afficher le résultat.

4. Soit le nombre 34.44. Comment pouvez-vous l'arrondir à l'entier inférieur ? Et à l'entier supérieur ?
5. Utilisez une expression rationnelle pour remplacer toutes les ponctuations par des virgules dans la chaîne suivante :

```
var str = "pomme.orange-fraise,citron-.prune";
```

Utilisez le résultat pour remplir un tableau et affichez chacun de ses éléments.

Les réponses se trouvent dans l'annexe.