CAPÍTULO 11 EXPRESIONES REGULARES

Las expresiones, son un lenguaje utilizado para describir patrones en cadenas de caracteres. Forman un pequeño y separado lenguaje, que está incluido en JavaScript (y en la gran mayoría de lenguajes de programación). No es un lenguaje fácil de leer, pero es una herramienta muy poderosa que simplifica mucho tareas de procesado de cadenas de caracteres.

Las expresiones regulares son utilizadas para buscar, reemplazar y extraer información de las cadenas de caracteres. Al igual que la gran mayoría de elementos en JavaScript, las expresiones regulares también son objetos. Los métodos que funcionan con expresiones regulares en JavaScript son las siguientes: `regexp.exec`, `regexp.test`, `string.match`, `string.replace`, `string.search`, y `string.split`. Las expresiones regulares tienen un rendimiento sensiblemente superior a las operaciones equivalentes sobre *strings*.

11.1 PRIMERA EXPRESIÓN REGULAR

Al igual que las cadenas de caracteres se escriben entre comillas dobles (`"`), los patrones de expresiones regulares se escriben entre barras (`/`).

```
var slash = /\//;
console.log("AC/DC".search(slash));
```

Este método de búsqueda se asemeja a `indexOf`, pero utiliza una expresión regular para la búsqueda en lugar de un *string*. Los patrones especificados por expresiones regulares, pueden hacer cosas que las cadenas de caracteres no pueden hacer, como que alguno de sus elementos coincida con más de un carácter.

Un ejemplo complejo de expresión regular puede ser el siguiente, un patrón para analizar URLs:

```
var parse_url = /^(?:([A-Za-z]+):)?(\/{0,3})([0-9.\-A-Za-z]+)(?::(\d+))?
(?:\/([^?#]*))?(?:\?([^#]*))?(?:#(.*))?$/;

var url = "http://www.ora.com:80/goodparts?q#fragment";
```

Utilicemos el método `exec`, que procesa la cadena de caracteres en función del patrón proporcionado, devolviendo un *array* que contiene las partes extraidas:

```
var result = parse_url.exec(url);

var names = ['url', 'scheme', 'slash', 'host', 'port',
     'path', 'query', 'hash'];
```

```
var i;
for (i = 0; i < names.length; i += 1) {
    document.writeln(names[i] + ':\n', result[i]);
}
```

Donde el resultado es el siguiente:

```
url:    http://www.ora.com:80/goodparts?q#fragment
scheme: http
slash:  //
host:   www.ora.com
port:   80
path:   goodparts
query:  q
hash:   fragment
```

11.2 CONTRUCCIÓN

La principal manera de construir expresiones regulares, es utilizando los literales, de la siguiente manera:

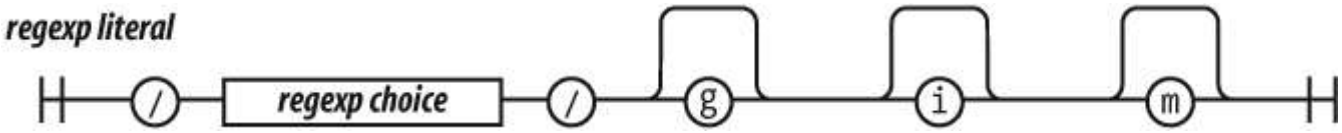

Figura 11.1 Construcción de una expresión literal

Podemos indicar tres parámetros al final de la expresión regular, que modifican ligeramente su comportamiento.

```
var my_regexp = /\s*/g;
```

Su significado es el siguiente:

Parámetro	Significado
`g`	Global (hace coincidir el patrón todas las veces posibles, aunque puede variar según el método).
`i`	Insensitive (ignora las mayúsculas y minúsculas).
`m`	Multiline (los caracteres `^` y `$` pueden coincidir con caracteres multilínea).

11.3 ELEMENTOS

11.3.1 EL PUNTO "."

El punto se interpreta como *cualquier carácter*, es decir, busca cualquier carácter sin incluir los saltos de línea.

```
var point = /g.ant/;
var story = "We noticed the *gi\nant sloth*, hanging from a giant branc
h.";
console.log(story.search(point));      // 17
```

11.3.2 LA BARRA INVERTIDA O CONTRABARRA "\"

Se utiliza para "marcar" el siguiente carácter de la expresión de búsqueda, de forma que este adquiera un significado especial o deje de tenerlo. Es decir, la barra invertida no se utiliza nunca por sí sola, sino en combinación con otros caracteres. Al utilizarlo por ejemplo en combinación con el punto "." este deja de tener su significado normal y se comporta como un carácter literal.

De la misma forma, cuando se coloca la barra invertida seguida de cualquiera de los caracteres especiales listados a continuación, estos dejan de tener su significado especial y se convierten en caracteres de búsqueda literal.

- `\t`: representa un tabulador.
- `\r`: representa el "retorno de carro".
- `\n`: representa la "nueva línea".
- `\e`: representa la tecla "Esc" o "Escape".
- `\f`: se utiliza para representar caracteres ASCII o ANSI si se conoce su código. Por ejemplo el símbolo © es representado mediante "\fA9".
- `\x`: se utiliza para representar caracteres Unicode si se conoce su código. Por ejemplo, "\u02" representa el símbolo de centavos.
- `\u`: se utiliza para representar caracteres Unicode si se conoce su código. Por ejemplo, "\u00A2" representa el símbolo de centavos.
- `\d`: representa un dígito del 0 al 9.
- `\w`: representa cualquier carácter alfanumérico (incluidos los guiones bajos `_`).
- `\s`: representa un espacio en blanco (espacios en blanco, tabuladores y nuevas líneas).
- `\b`: marca el inicio y el final de una palabra.

Para los caracteres `d`, `w` y `s`, se puede utlizar su variante en mayúsculas para indicar justamente su significado opuesto.

- `\D`: representa cualquier carácter que no sea un dígito del 0 al 9.
- `\W`: representa cualquier carácter no alfanumérico.
- `\S`: representa cualquier carácter que no sea un espacio en blanco.

```
var digitSurroundedBySpace = /\s\d\s/;
console.log("1a 2 3d".search(digitSurroundedBySpace));     // 2
```

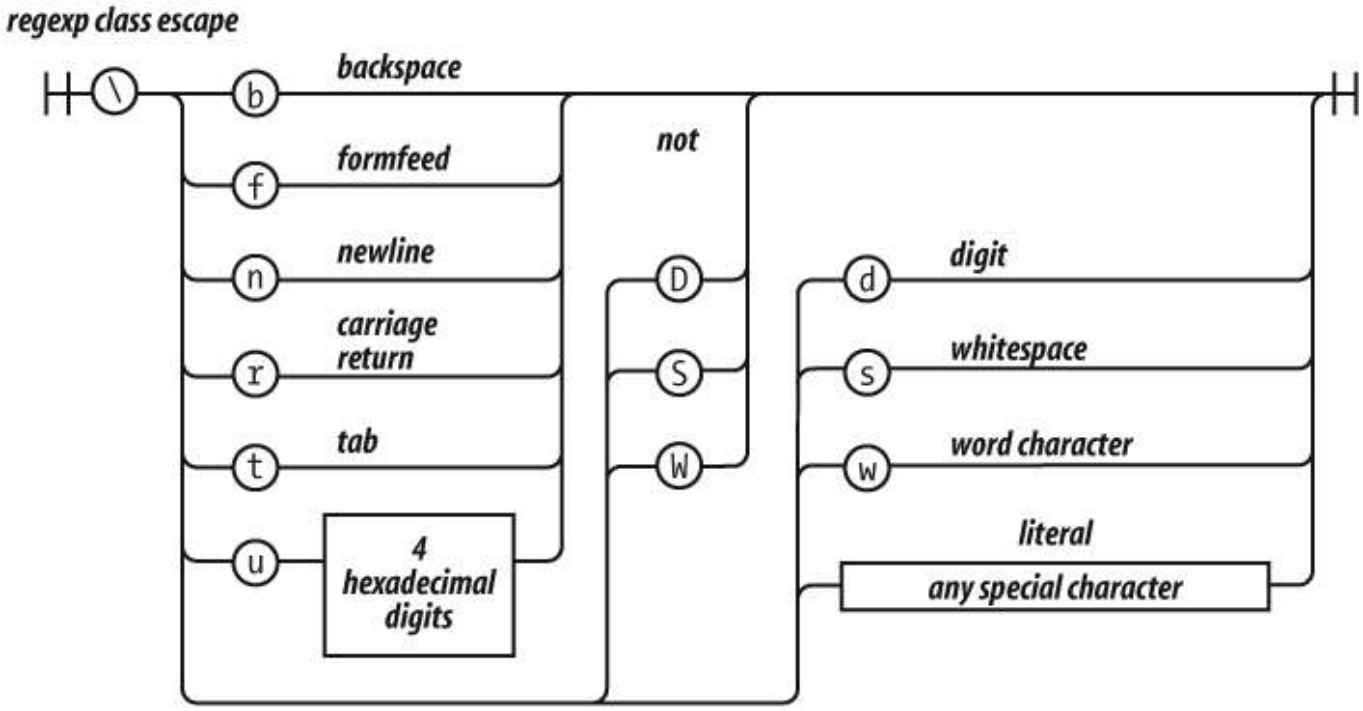

Figura 11.2 Caracteres de escape

11.3.3 LOS CORCHETES "[ ]"

La función de los corchetes es representar "clases de caracteres", es decir, agrupar caracteres en grupos o clases. Dentro de los corchetes es posible utilizar el guion "-" para especificar rangos de caracteres. Hay que tener en cuenta que dentro de los corchetes, los metacaracteres (`$`, `*`, `+`, `?`) pierden su significado y se convierten en literales cuando se encuentran dentro de estos corchetes.

```
var asteriskOrBrace = /[{*]/;
var story = "We noticed the *giant sloth*, hanging from a giant branch."
;
console.log(story.search(asteriskOrBrace));       // 15
```

Cuando al inicio de este grupo de caracteres se encuentra el carácter `^`, permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado.

```
var notABC = /[^ABC]/;
console.log("ABCBACCBBADABC".search(notABC));     // 10
```

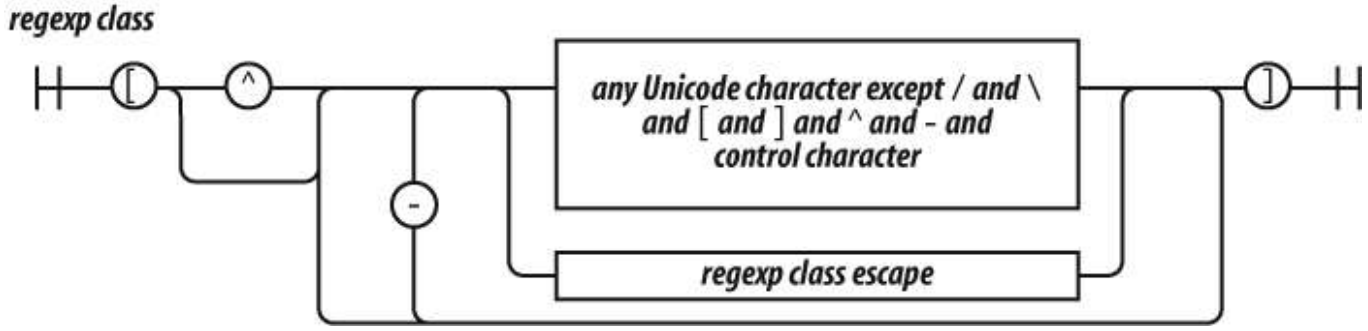

Figura 11.3 Caracteres de clase

11.3.4 LA BARRA "|"

Sirve para indicar una de varias opciones.

```
var cardinalPoints = /north|south|east|west/i;
console.log(cardinalPoints.test("At north"));                    // true
console.log(cardinalPoints.test("I'm from Southampton"));        // true
```

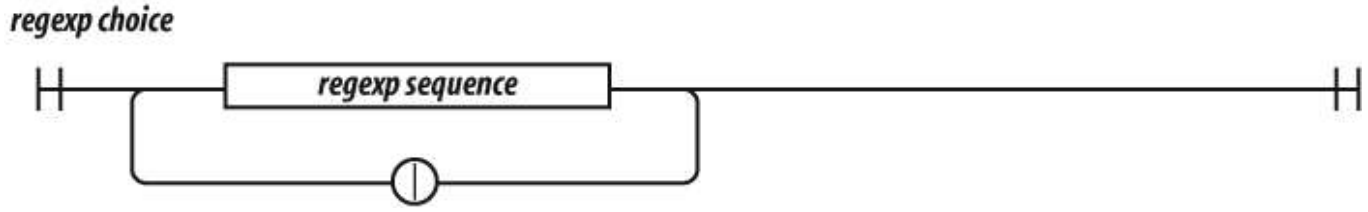

Figura 11.4 Carácter para indicar posibles opciones

11.3.5 EL SIGNO DE DÓLAR "$"

Representa el final de la cadena de caracteres o el final de la línea, si se utiliza el modo multi-línea. No representa un carácter en especial sino una posición. Si se utiliza la expresión regular ".$", el motor encontrará todos los lugares donde un punto finalice la línea, lo que es útil para avanzar entre párrafos.

```
console.log(/a+/.test("blah"));          // true
console.log(/a+$/.test("blah"));         // false
console.log(/a+$/.test("blahaa"));       // true
```

11.3.6 EL ACENTO CIRCUNFLEJO " "

Éste carácter tiene una doble funcionalidad, en función de si utiliza individualmente y si se utiliza en conjunto con otros caracteres especiales. Su funcionalidad como carácter individual: el carácter `^` representa el inicio de la cadena (de la misma forma que el signo de dólar `$` representa el final de la cadena).

```
console.log(/a+/.test("blah"));          // true
console.log(/^a+/.test("blah"));         // false
console.log(/^a+/.test("aaaablah"));       // true
```

Cuando se utiliza en conjunto con los corchetes, permite encontrar cualquier carácter que NO se encuentre dentro del grupo indicado.

```
var notABC = /[^ABC]/;
console.log("ABCBACCBBADABC".search(notABC));      // 10
```

La utilización en conjunto de los caracteres especiales `^` y `$` permite realizar validaciones de forma sencilla. Por ejemplo `^\d$` permite asegurar que la cadena a verificar representa un único dígito, mientras que `^\d\d/\d\d/\d\d\d\d$` permite validar una fecha en formato corto.

11.3.7 LOS PARÉNTESIS "()"

De forma similar que los corchetes, los paréntesis sirven para agrupar caracteres, sin embargo existen varias diferencias fundamentales entre los grupos establecidos por medio de corchetes y los grupos establecidos por paréntesis:

- Los caracteres especiales conservan su significado dentro de los paréntesis.
- Los grupos establecidos con paréntesis establecen una "etiqueta" o "punto de referencia" para el motor de búsqueda, que puede ser utilizada posteriormente.
- Utilizados en conjunto con la barra `|` permiten hacer búsquedas opcionales.
- Utilizados en conjunto con otros caracteres especiales que se detallan posteriormente, ofrece funcionalidad adicional.

```
var holyCow = /(sacred|holy) (cow|bovine|bull|taurus)/i;
console.log(holyCow.test("Sacred bovine!"));      // true
```

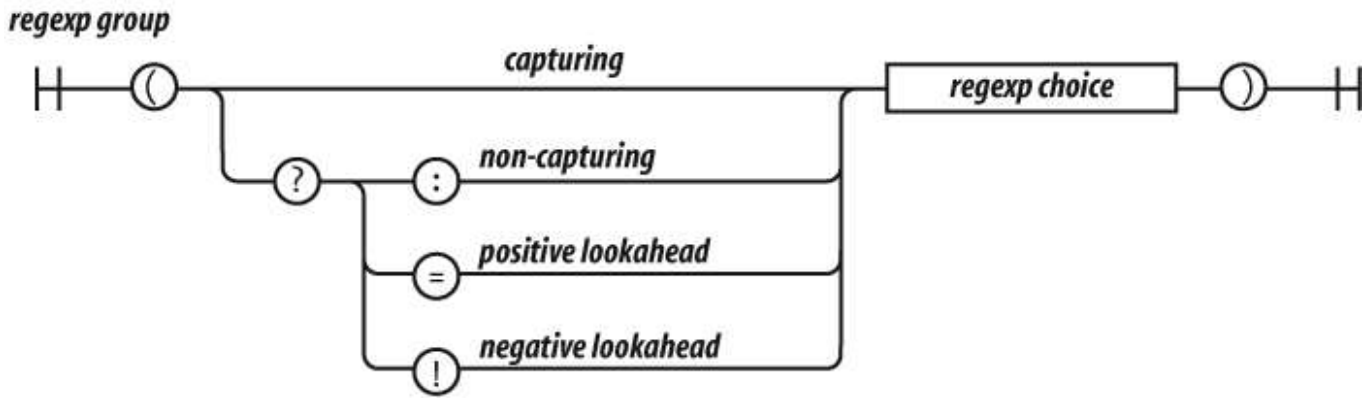

Figura 11.5 Grupos de caracteres

11.3.8 EL SIGNO DE INTERROGACIÓN "?"

El signo de interrogación especifica que una parte de la búsqueda es opcional. En conjunto con los paréntesis, permite especificar que un conjunto mayor de caracteres es opcional

```
var nov = /Nov(\.|iembre|ember)?/;
console.log(nov.test("Nov"));              // true
```

```
console.log(nov.test("Nov."));          // true
console.log(nov.test("Noviembre"));     // true
console.log(nov.test("November"));      // true
```

Como hemos indicado antes, los paréntesis pueden ser utilizados como puntos de referencia dentro de las expresiones regulares. Si no es necesario, podemos evitar este comportamiento de la siguiente manera, liberando al motor de búsqueda de un trabajo extra:

```
var nov = /Nov(?:\.|iembre|ember)?/;
```

11.3.9 LAS LLAVES "{}"

Comúnmente las llaves son caracteres literales cuando se utilizan por separado en una expresión regular. Para que adquieran su función de metacaracteres es necesario que encierren uno o varios números separados por coma y que estén colocados a la derecha de otra expresión regular de la siguiente forma:

```
var date = /^\d{2}\/\d{2}\/\d{2,4}$/;
console.log(date.test("05/05/1982"));     // true
console.log(date.test("05/05/82"));       // true
```

Son utilizados para indicar el número de veces que puede darse una coincidencia. Un número entre llaves (`{4}`) indica el número exacto de coincidencias. Dos números separados por una coma (`{2,4}`), indica que puede coincidir al menos tantas veces como el primer número (`2`), y cómo máximo tántas veces como el segundo número (`4`). De manera similar, `{2,}` significa que al menos ocurre dos veces y `{,4}`, que como máximo ocurre cuatro veces.

11.3.10 EL ASTERISCO "*"

El asterisco sirve para encontrar algo que se encuentra repetido 0 o más veces. Por ejemplo:

```
var exp = /[a-zA-Z]\d*/;
console.log(exp.test("A"));           // true
console.log(exp.test("B0"));          // true
console.log(exp.test("c01"));         // true
console.log(exp.test("abc01234"));    // true
console.log(exp.test("01234"));       // false
```

El asterisco equivale a la expresión `{0,}`.

11.3.11 EL SIGNO DE SUMA "+"

Se utiliza para encontrar una cadena que se encuentre repetida una o más veces. A diferencia del asterisco, la expresión `[a-zA-Z]\d+` encontrará "H1" pero no encontrará "H".

```
var exp = /[a-zA-Z]\d+/;
console.log(exp.test("A"));           // false
```

```
console.log(exp.test("B0"));          // true
console.log(exp.test("c01"));         // true
console.log(exp.test("abc01234"));    // true
console.log(exp.test("01234"));       // false
```

El signo de suma equivale a la expresión `{1,}`.

11.3.12 GRUPOS ANÓNIMOS

Los grupos anónimos se establecen cada vez que se encierra una expresión regular en paréntesis, por lo que la expresión `([a-zA-Z]\w*?)` define un grupo anónimo que tendrá como resultado que el motor de búsqueda almacenará una referencia al texto que corresponda a la expresión encerrada entre los paréntesis.

Es posible hacer referencia a estos grupos dentro de la propia expresión regular, o utilizarlos para extraer partes de la cadena de caracteres, si utilizamos los métodos que corresponden.

```
var exp = /<([a-zA-Z]\w*?)>.*?<\/\1>/;
console.log(exp.test("<font>Text</font>"));       // true
console.log(exp.test("<h1>Text</font>"));     // false
```

11.4 MÉTODOS

11.4.1 REGEXP.EXEC(STRING)

Éste método es el más poderoso (y lento) de todos los métodos que se pueden utilizar en expresiones regulares. Si la cadena de caracteres satisface el patrón, este método devuelve un *array*. El elemento `0` contiene la cadena de caracteres que coincide con la expresión regular, el elemento `1` el texto que coincide con el grupo 1, el elemento `2` el texto que coincide con el grupo 2... Si el patrón falla, la función devuelve `null`.

11.4.2 REGEXP.TEST(STRING)

Éste método es el más simple (y rápido) de todos los métodos que se pueden utilizar en expresiones regulares. Si la cadena de caracteres satisface el patrón, este método devuelve `true`, y `false` en caso contrario.

11.4.3 STRING.MATCH(REGEXP)

Éste método compara la cadena de caracteres con la expresión regular, pero su comportamiento depende del parámetro `g`. Si éste parámetro no está presente, el comportamiento es el mismo que so llamamos al método `regexp.exec(string)`. En cambio, si está presente, devuelve un array con todas las coincidencias, pero excluye los grupos.

```
var text = '<html><body bgcolor=linen><p>' +
           'This is <b>bold<\/b>!<\/p><\/body><\/html>';
```

```
var tags = /[^<>]+|<(\/?)([A-Za-z]+)([^<>]*)>/g;

var a, i;
a = text.match(tags);
for (i = 0; i < a.length; i += 1) {
    document.writeln(('// [' + i + '] ' + a[i]).entityify());
}

// The result is
// [0] <html>
// [1] <body bgcolor=linen>
// [2] <p>
// [3] This is
// [4] <b>
// [5] bold
// [6] </b>
// [7] !
// [8] </p>
// [9] </body>
// [10] </html>
```

11.4.4 STRING.REPLACE(SEARCHVALUE, REPLACEVALUE)

Éste método busca y reemplaza los valores indicados, devolviendo un nuevo string. El parámetro `searchValue` puede ser una cadena de caracteres o una expresión regular. Si es un `string`, únicamente la primera aparición es reemplazada, lo que puede ser un poco confuso.

```
var result = "mother_in_law".replace('_', '-');
console.log(result);      //mother-in_law
```

Si el parámetro es una expresión regular, y contiene el parámetro `g`, entonces reemplazará todas las apariciones. Si no contiene este parámetro, entonces sólo se reemplaza la primera aparición.

El parámetros `replaceValue` puede ser un *string* o una fución. Si el parámetro es una función, ésta será llamada por cada coincidencia, y el *string* devuelto por la función será utilizado como cadena de reemplazo. El parámetro pasado a esta función corresponde con la coincidencia de texto.

```
var character = {
                 '<' : '&lt;',
                 '>' : '&gt;',
                 '&' : '&amp;',
                 '"' : '&quot;'
                };
```

```
var entities = "<&>";
entities.replace(/[<>&"]/g,  function (c) {
            return character[c];
          });
```

11.4.5 STRING.SEARCH

Éste método de búsqueda es igual al método `indexOf`, a excepción de que éste método toma como parámetro una expresión regular en vez de un *string*. Devuelve la posición del primer carácter que coincide con la expresión regular, si hay alguno, y `-1` si no hay coincidencias. En este caso, el parámetro `g` es ignorado.

```
var text = 'and in it he says "Any damn fool could';
var pos = text.search(/["']/);     // 18
```

11.4.6 STRING.SPLIT(SEPARATOR, LIMIT)

Éste método crea un array de *strings* , dividiendo el *string* original en trozos. El parámetro `separator` puede ser un *string* o una expresión regular. El parámetro opcional `limit` indica el número máximo de trozos en los que se va a dividir el *string*.

```
var c = '|a|b|c|'.split('|');   // c is ['', 'a', 'b', 'c', '']

var text = 'last,  first  ,middle';
var d = text.split(/\s*,\s*/);
// d is [
//       'last',
//       'first',
//       'middle'
//      ]
```

> **EJERCICIO 13**
>
> Ver enunciado

ÍNDICE DE CONTENIDOS