

Tiếng Việt ▼

# Biểu thức chính quy

« Previous

Next »

Biểu thức chính quy (regular expressions) là các mẫu dùng để tìm kiếm các bộ kí tự được kết hợp với nhau trong các chuỗi kí tự. Trong JavaScript thì biểu thức chính quy cũng đồng thời là các đối tượng, tức là khi bạn tạo ra một biểu thức chính quy là bạn có một đối tượng tương ứng. Các mẫu này được sử dụng khá nhiều trong JavaScript như phương thức `exec` và `test` của `RegExp`, hay phương thức `match`, `replace`, `search`, và `split` của `String`. Trong chương này, ta cùng tìm hiểu chi tiết hơn về biểu thức chính quy trong JavaScript.

## Tạo một biểu thức chính quy

Bạn có thể tạo ra một biểu thức chính quy bằng 1 trong 2 cách sau:

- Sử dụng cách mô tả chính quy thuần (regular expression literal) như sau:

```
1 | var re = /ab+c/;
```

Các đoạn mã chứa các mô tả chính quy thuần sau khi được nạp vào bộ nhớ sẽ dịch các đoạn mô tả đó thành các biểu thức chính quy. Các biểu thức chính quy được dịch ra này sẽ được coi như các hằng số, tức là không phải tạo lại nhiều lần, điều này giúp cho hiệu năng thực hiện tốt hơn.

- Tạo một đối tượng `RegExp` như sau:

```
1 | var re = new RegExp("ab+c");
```

Với cách này, các biểu thức chính quy sẽ được dịch ra lúc thực thi chương trình nên hiệu năng không đạt được như với việc sử dụng cách mô tả chính quy thuần. Nhưng ưu điểm là nó có thể thay đổi được, nên ta thường sử dụng chúng khi ta muốn nó có thể thay đổi được, hoặc khi ta chưa chắc chắn về các mẫu chính quy (pattern) như nhập từ bàn phím chẳng hạn.

---

## Cách viết một mẫu biểu thức chính quy

Một mẫu biểu thức chính quy là một tập các kí tự thường, như `/abc/`, hay một tập kết hợp cả kí tự thường và kí tự đặc biệt như `/ab*c/` hoặc `/Chapter (\d+)\.\d*/`. Trong ví dụ cuối, ta thấy rằng nó chứa cả các dấu ngoặc đơn ( `()` ) được sử dụng như các thiết bị nhớ, tức là các mẫu trong phần ( ) này sau khi được tìm kiếm có thể được nhớ lại để sử dụng cho các lần sau. Bạn có thể xem chi tiết hơn tại: [Sử dụng dấu ngoặc đơn tìm sâu con](#).

### Sử dụng mẫu đơn giản

Các mẫu đơn giản là các mẫu có thể được xây dựng từ các kí tự có thể tìm kiếm một cách trực tiếp. Ví dụ, mẫu `/abc/` sẽ tìm các đoạn 'abc' theo đúng thứ tự đó trong các chuỗi. Mẫu này sẽ khớp được với "Hi, do you know your abc's?" và "The latest airplane designs evolved from slabcraft.", vì cả 2 chuỗi này đều chứa đoạn 'abc'. Còn với chuỗi 'Grab crab', nó sẽ không khớp vì chuỗi này không chứa 'abc' theo đúng thứ tự, mà chỉ chứa 'ab c'.

### Sử dụng các kí tự đặc biệt

Các mẫu có thể chứa các kí tự đặc biệt cho các mục đích tìm kiếm nâng cao mà tìm kiếm trực tiếp sẽ khó khăn như tìm một đoạn chứa một hoặc nhiều hơn một kí tự b, hay tìm một hoặc nhiều kí tự dấu cách (while space). Ví dụ, mẫu `/ab*c/` có thể tìm các đoạn có chứa: một kí tự 'a', theo sau là không có hoặc có một hoặc có nhiều kí tự 'b', cuối cùng là một kí tự 'c' như chuỗi "cbbabbbbcdebc," sẽ được khớp với chuỗi 'abbbbc'.

Bảng dưới đây mô tả đầy đủ các kí tự đặc biệt có thể dùng với biểu thức chính quy.

**Bảng 4.1 Các kí tự đặc biệt trong biểu thức chính quy.**


---

Kí tự (kí hiệu, cờ)	Ý nghĩa
\	<p>Tìm với luật dưới đây:</p> <p>Một dấu gạch chéo ngược sẽ biến một kí tự thường liền kề phía sau thành một kí tự đặc biệt, tức là nó không được sử dụng để tìm kiếm thông thường nữa. Ví dụ, trường hợp kí tự 'b' không có dấu gạch chéo ngược này sẽ được khớp với các kí tự 'b' in thường, nhưng khi nó có thêm dấu gạch chéo ngược, '\b' thì nó sẽ không khớp với bất kì kí tự nào nữa, lúc này nó trở thành kí tự đặc biệt. Xem thêm phần <a href="#">word boundary character</a> để biết thêm chi tiết.</p> <p>Tuy nhiên nếu đứng trước một kí tự đặc biệt thì nó sẽ biến kí tự này thành một kí tự thường, tức là bạn có thể tìm kiếm kí tự đặc biệt này trong chuỗi của bạn như các kí tự thường khác. Ví dụ, mẫu /a*/ có '*' là kí tự đặc biệt và mẫu này sẽ bị phụ thuộc vào kí tự này, nên được hiểu là sẽ tìm khớp với 0 hoặc nhiều kí tự a. Nhưng, với mẫu /a\*/ thì kí tự '*' lúc này được hiểu là kí tự thường nên mẫu này sẽ tìm kiếm chuỗi con là 'a*'.  Đừng quên \ cũng là một kí tự đặc biệt, khi cần so khớp chính nó ta cũng phải đánh dấu nó là kí tự đặc biệt bằng cách đặt \ ở trước (\\).</p>
^	<p>Khớp các kí tự đứng đầu một chuỗi. Nếu có nhiều cờ này thì nó còn khớp được cả các kí tự đứng đầu của mỗi dòng (sau kí tự xuống dòng).</p> <p>Ví dụ, /^A/ sẽ không khớp được với 'A' trong "an A" vì 'A' lúc này không đứng đầu chuỗi, nhưng nó sẽ khớp "An E" vì lúc này 'A' đã đứng đầu chuỗi.</p> <p>Ý nghĩa của '^' sẽ thay đổi khi nó xuất hiện như một kí tự đầu tiên trong một lớp kí tự, xem phần <a href="#">complemented character sets</a> để biết thêm chi tiết.</p>
\$	<p>So khớp ở cuối chuỗi. Nếu gắn cờ multiline (đa dòng), nó sẽ khớp ngay trước kí tự xuống dòng.</p> <p>Ví dụ, /t\$/ không khớp với 't' trong chuỗi "eater" nhưng lại khớp trong chuỗi "eat".</p>

Kí tự (kí hiệu, cờ)	Ý nghĩa
*	<p>Cho phép kí tự trước nó lặp lại 0 lần hoặc nhiều lần. Tương đương với cách viết {0,}.</p> <p>Ví dụ, /bo*/ khớp với 'boooo' trong chuỗi "A ghost boooooed" nhưng không khớp trong chuỗi "A birth warbled".</p>
+	<p>Cho phép kí tự trước nó lặp lại 1 lần hoặc nhiều lần. Tương đương với cách viết {1,}.</p> <p>Ví dụ, /a+/ khớp với 'a' trong chuỗi "candy" và khớp với tất cả kí tự a liên nhau trong chuỗi "caaaaaaandy".</p>
?	<p>Cho phép kí tự trước nó lặp lại 0 lần hoặc 1 lần duy nhất. Tương đương với cách viết {0,1}.</p> <p>Ví dụ, /e?le?/ khớp với 'el' trong chuỗi "angel" và 'le' trong chuỗi "angle" hay 'l' trong "oslo".</p> <p>Nếu sử dụng kí tự này ngay sau bất kì kí tự định lượng nào trong số *,+,? hay {}, đều làm bộ định lượng "chán ăn" (dừng so khớp sau ngay khi tìm được kí tự phù hợp), trái ngược với đức tính "tham lam" vốn sẵn của chúng (khớp tất cả kí tự chúng tìm thấy). Ví dụ, áp dụng biểu mẫu /\d+/ cho "123abc" ta được "123". Nhưng áp /\d+?/ cho chính chuỗi trên ta chỉ nhận được kết quả là "1".</p> <p>Bạn có thể đọc thêm trong mục x(?=y) và x(?!y) của bảng này.</p>
.	<p>Dấu . khớp với bất kì kí tự đơn nào ngoại trừ kí tự xuống dòng.</p> <p>Ví dụ, /.n/ khớp với 'an' và 'on' trong chuỗi "no, an apple is on the tree", nhưng không khớp với 'no'.</p>

Kí tự (kí hiệu, cờ)	Ý nghĩa
(x)	<p>Khớp 'x' và nhớ kết quả so khớp này, như ví dụ ở dưới. Các dấu ngoặc tròn được gọi là các dấu ngoặc có nhớ.</p> <p>Biểu mẫu <code>/(foo) (bar) \1 \2/</code> khớp với <code>'foo'</code> và <code>'bar'</code> trong chuỗi <code>"foo bar foo bar"</code>. <code>\1</code> và <code>\2</code> trong mẫu khớp với 2 từ cuối.</p> <p>Chú ý rằng <code>\1</code>, <code>\2</code>, <code>\n</code> được sử dụng để so khớp các phần trong regex, nó đại diện cho nhóm so khớp đằng trước. Ví dụ: <code>/(foo) (bar) \1 \2/</code> tương đương với biểu thức <code>/(foo) (bar) foo bar/</code>.</p> <p>Cú pháp <code>\$1</code>, <code>\$2</code>, <code>\$n</code> còn được sử dụng trong việc thay thế các phần của một regex. Ví dụ: <code>'bar foo'.replace(/(...) (...)/, '\$2 \$1')</code> sẽ đảo vị trí 2 từ <code>'bar'</code> và <code>'foo'</code> cho nhau.</p>
(?:x)	<p>Khớp 'x' nhưng không nhớ kết quả so khớp. Những dấu ngoặc tròn được gọi là những dấu ngoặc không nhớ, nó cho phép bạn định nghĩa những biểu thức con cho những toán tử so khớp. Xem xét biểu thức đơn giản <code>/(?:foo){1,2}/</code>. Nếu biểu thức này được viết là <code>/foo{1,2}/</code>, <code>{1,2}</code> sẽ chỉ áp dụng cho kí tự 'o' ở cuối chuỗi 'foo'. Với những dấu ngoặc không nhớ, <code>{1,2}</code> sẽ áp dụng cho cả cụm 'foo'.</p>
x(?:=y)	<p>Chỉ khớp 'x' nếu 'x' theo sau bởi 'y'.</p> <p>Ví dụ, <code>/Jack(?:=Sprat)/</code> chỉ khớp với 'Jack' nếu đằng sau nó là 'Sprat'. <code>/Jack(?:=Sprat Frost)/</code> chỉ khớp 'Jack' nếu theo sau nó là 'Sprat' hoặc 'Frost'. Tuy nhiên, cả 'Sprat' và 'Frost' đều không phải là một phần của kết quả so khớp trả về.</p>
x(?:!y)	<p>Chỉ khớp 'x' nếu 'x' <b>không</b> được theo sau bởi 'y'.</p> <p>Ví dụ: <code>/\d+(?!\.\.)/</code> chỉ khớp với số không có dấu <code>.</code> đằng sau. Biểu thức <code>/\d+(?!\.\.)/.exec("3.141")</code> cho kết quả là <code>'141'</code> mà không phải <code>'3.141'</code>.</p>

Kí tự (kí hiệu, cờ)	Ý nghĩa
<code>x y</code>	<p>Khớp 'x' hoặc 'y'</p> <p>Ví dụ, <code>/green red/</code> khớp với 'green' trong chuỗi "green apple" và 'red' trong chuỗi "red apple".</p>
<code>{n}</code>	<p>Kí tự đứng trước phải xuất hiện n lần. n phải là một số nguyên dương.</p> <p>Ví dụ, <code>/a{2}/</code> không khớp với 'a' trong "candy", nhưng nó khớp với tất cả kí tự 'a' trong "caandy", và khớp với 2 kí tự 'a' đầu tiên trong "caaandy".</p>
<code>{n,m}</code>	<p>Kí tự đứng trước phải xuất hiện từ n đến m lần. n và m là số nguyên dương và <math>n \leq m</math>. Nếu m bị bỏ qua, nó tương đương như <math>\infty</math>.</p> <p>Ví dụ, <code>/a{1,3}/</code> không khớp bất kì kí tự nào trong "cndy", kí tự 'a' trong "candy", 2 kí tự 'a' đầu tiên trong "caandy", và 3 kí tự 'a' đầu tiên trong "caaaaaaandy". Lưu ý là "caaaaaaandy" chỉ khớp với 3 kí tự 'a' đầu tiên mặc dù chuỗi đó chứa 7 kí tự 'a'.</p>
<code>[xyz]</code>	<p>Lớp kí tự. Loại mẫu này dùng để so khớp với một kí tự bất kì trong dấu ngoặc vuông, bao gồm cả <a href="#">escape sequences</a>. Trong lớp kí tự, dấu chấm (.) và dấu hoa thị (*) không còn là kí tự đặc biệt nên ta không cần kí tự thoát đứng trước nó. Bạn có thể chỉ định một khoảng kí tự bằng cách sử dụng một kí tự gạch nối (-) như trong ví dụ dưới đây:</p> <p>Mẫu <code>[a-d]</code> so khớp tương tự như mẫu <code>[abcd]</code>, khớp với 'b' trong "brisket" và 'c' trong "city". Mẫu <code>/[a-z.]+/</code> và <code>/[\w.]+/</code> khớp với toàn chuỗi "test.i.ng".</p>
<code>[^xyz]</code>	<p>Lớp kí tự phủ định. Khi kí tự ^ đứng đầu tiên trong dấu ngoặc vuông, nó phủ định mẫu này.</p> <p>Ví dụ, <code>[^abc]</code> tương tự như <code>[^a-c]</code>, khớp với 'r' trong "brisket" và 'h' trong "chop" là kí tự đầu tiên không thuộc khoảng a đến c.</p>

Kí tự (kí hiệu, cờ)	Ý nghĩa
<code>[\b]</code>	Khớp với kí tự dịch lùi - backspace (U+0008). Bạn phải đặt trong dấu ngoặc vuông nếu muốn so khớp một kí tự dịch lùi. (Đừng nhầm lẫn với mẫu <code>\b</code> ).
<code>\b</code>	<p>Khớp với kí tự biên. Kí tự biên là một kí tự giả, nó khớp với vị trí mà một kí tự không được theo sau hoặc đứng trước bởi một kí tự khác. Tương đương với mẫu <code>(^\w \w\$ \w\w \w\W)</code>. Lưu ý rằng một kí tự biên được khớp sẽ không bao gồm trong kết quả so khớp. Nói cách khác, độ dài của một kí tự biên là 0. (Đừng nhầm lẫn với mẫu <code>[\b]</code>)</p> <p>Ví dụ:</p> <p><code>/\bm/</code> khớp với 'm' trong chuỗi "moon";</p> <p><code>/oo\b/</code> không khớp 'oo' trong chuỗi "moon", bởi vì 'oo' được theo sau bởi kí tự 'n';</p> <p><code>/oon\b/</code> khớp với 'oon' trong chuỗi "moon", bởi vì 'oon' ở cuối chuỗi nên nó không được theo sau bởi một kí tự;</p> <p><code>/\w\b\w/</code> sẽ không khớp với bất kì thứ gì, bởi vì một kí tự không thể theo sau một kí tự biên và một kí tự thường.</p> <div> <p> <b>Chú ý:</b> Engine biên dịch biểu thức chính quy trong Javascript định nghĩa một <b>lớp kí tự</b> là những kí tự thường. Bất kỳ kí tự nào không thuộc lớp kí tự bị xem như một kí tự ngắt. Lớp kí tự này khá hạn chế: nó bao gồm bộ kí tự La-tinh cả hoa và thường, số thập phân và kí tự gạch dưới. Kí tự có dấu, như "é" hay "ü", không may, bị đối xử như một kí tự ngắt.</p> </div>
<code>\B</code>	<p>Khớp với kí tự không phải kí tự biên. Mẫu này khớp tại vị trí mà kí tự trước và kí tự sau nó cùng kiểu: hoặc cả hai là kí tự hoặc cả hai không phải là kí tự. Bắt đầu và kết thúc chuỗi không được xem là những kí tự.</p> <p>Ví dụ, <code>/\B. ./</code> khớp với 'oo' trong "noonday", và <code>/y\B. /</code> khớp với 'ye' trong "possibly yesterday."</p>

Kí tự (kí hiệu, cờ)	Ý nghĩa
<code>\cX</code>	X là một kí tự trong khoảng A tới Z. Mẫu này khớp với một kí tự điều khiển trong một chuỗi.  Ví dụ: <code>/\cM/</code> khớp với control-M (U+000D) trong chuỗi.
<code>\d</code>	Khớp với một kí tự số. Tương đương với mẫu <code>[0-9]</code> .  Ví dụ: <code>/\d/</code> hoặc <code>/[0-9]/</code> khớp với '2' trong chuỗi "B2 is the suite number."
<code>\D</code>	Khớp với một kí tự không phải là kí tự số. Tương đương với mẫu <code>[^0-9]</code> .  Ví dụ: <code>/\D/</code> hoặc <code>/[^0-9]/</code> khớp với 'B' trong "B2 is the suite number."
<code>\f</code>	Khớp với kí tự phân trang - form feed (U+000C).
<code>\n</code>	Khớp với kí tự xuống dòng - line feed (U+000A).
<code>\r</code>	Khớp với kí tự quay đầu dòng - carriage return (U+000D).
<code>\s</code>	Khớp với một kí tự khoảng trắng, bao gồm trống - space, tab, phân trang - form feed, phân dòng - line feed. Tương đương với <code>[ \f\n\r\t\v\u00a0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]</code> .  Ví dụ: <code>/\s\w*/</code> khớp với ' bar' trong "foo bar."
<code>\S</code>	Khớp với một kí tự không phải khoảng trắng. Tương đương với <code>[^\f\n\r\t\v\u00a0\u1680\u180e\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200a\u2028\u2029\u202f\u205f\u3000]</code> .  Ví dụ: <code>/\S\w*/</code> khớp với 'foo' trong chuỗi "foo bar."
<code>\t</code>	Khớp với kí tự tab (U+0009).
<code>\v</code>	Khớp với kí tự vertical tab (U+000B).



Kí tự (kí hiệu, cờ)	Ý nghĩa
<code>\w</code>	Khớp với tất cả kí tự là chữ, số và gạch dưới. Tương đương với mẫu <code>[A-Za-z0-9_]</code> .  ví dụ, <code>/\w/</code> khớp với 'a' trong "apple," '5' trong "\$5.28," và '3' trong "3D."
<code>\W</code>	Khớp với tất cả kí tự không phải là chữ. Tương đương với mẫu <code>[^A-Za-z0-9_]</code> .  ví dụ, <code>/\W/</code> hoặc <code>/[^A-Za-z0-9_]/</code> khớp với '%' trong "50%."
<code>\n</code>	Trong đó, n là một số nguyên dương, một tham chiếu ngược tới chuỗi khớp thứ n trong biểu thức (đếm từ trái sang, bắt đầu bằng 1).  ví dụ, <code>/apple(,)\sorange\1/</code> hay <code>/apple(,)\sorange,/</code> khớp với 'apple, orange,' trong chuỗi "apple, orange, cherry, peach."
<code>\0</code>	Khớp với kí tự NULL (U+0000). Lưu ý: không được thêm bất kì một kí tự số nào sau 0, vì <code>\0&lt;các-kí-tự-số&gt;</code> là một biểu diễn hệ bát phân <a href="#">escape sequence</a> .
<code>\xhh</code>	Khớp với kí tự với mã code là hh (2 số trong hệ thập lục phân)
<code>\uhhhh</code>	Khớp với kí tự có mã hhhh (4 số trong hệ thập lục phân).

Mã hóa escape chuỗi người dùng nhập vào bằng một hàm thay thế đơn giản sử dụng biểu thức chính quy:

```

1 | function escapeRegExp(string){
2 |     return string.replace(/[.*+?^${}()|[\]\\"/g, "\\$&");
3 | }

```

## Sử dụng ngoặc tròn

Ngoặc tròn bao quanh bất kỳ phần nào của biểu thức chính quy sẽ khiến phần kết quả so khớp được nhớ. Mỗi lần nhớ, chuỗi con có thể được gọi lại để sử dụng, mô tả trong [Using Parenthesized Substring Matches](#).

Ví dụ, mẫu `/Chapter (\d+)\.\d*/` khớp đúng với 'Chapter ' theo sau bởi một hoặc nhiều kí tự số, sau nữa là một dấu chấm thập phân, cuối cùng có thể là 0 hoặc nhiều kí tự số. Bên cạnh đó, dấu ngoặc tròn được sử dụng để nhớ một hoặc nhiều kí tự số đầu tiên được khớp.

Mẫu này được tìm thấy trong chuỗi "Open Chapter 4.3, paragraph 6", nhớ '4' nhưng không được tìm thấy trong chuỗi "Chapter 3 and 4", bởi vì chuỗi đó không có dấu chấm sau kí tự số '3'.

Để so khớp một chuỗi con không nhớ, đặt `?:` ở vị trí đầu tiên trong ngoặc. Ví dụ, `(?:\d+)` khớp với một hoặc nhiều kí tự số nhưng không nhớ kết quả so khớp.

---

## Làm việc với biểu thức chính quy

Biểu thức chính quy được sử dụng với phương thức `test` và `exec` của lớp `RegExp` hoặc phương thức `match`, `replace`, `search` và `split` của chuỗi. Những phương thức này được giải thích chi tiết trong [JavaScript Reference](#).

**Bảng 4.2 Những phương thức được sử dụng trong biểu thức chính quy**

Phương thức	Mô tả
<code>exec</code>	Một phương thức của <code>RegExp</code> dùng để tìm kiếm chuỗi phù hợp với mẫu so khớp. Nó trả về một mảng chứa kết quả tìm kiếm.
<code>test</code>	Một phương thức của <code>RegExp</code> dùng để kiểm tra mẫu có khớp với chuỗi hay không. Nó trả về giá trị <code>true</code> hoặc <code>false</code> .
<code>match</code>	Một phương thức của chuỗi dùng để tìm kiếm chuỗi phù hợp với mẫu so khớp. Nó trả về một mảng chứa kết quả tìm kiếm hoặc <code>null</code> nếu không tìm thấy.
<code>search</code>	Một phương thức của chuỗi dùng để tìm kiếm chuỗi phù hợp với mẫu so khớp và trả về vị trí của chuỗi đó hoặc <code>-1</code> nếu không tìm thấy.
<code>replace</code>	Một phương thức của chuỗi dùng để tìm kiếm một chuỗi theo mẫu so khớp và thay thế chuỗi con được khớp với một chuỗi thay thế.
<code>split</code>	Một phương thức của chuỗi dùng một biểu mẫu chính quy hoặc một chuỗi bất biến để ngắt chuỗi đó thành một mảng các chuỗi con.

Khi bạn muốn biết một mẫu có được tìm thấy trong chuỗi hay không, sử dụng phương thức `test` hoặc `search`; để lấy nhiều thông tin hơn (nhưng chậm hơn) sử dụng phương thức `exec` hoặc `match`.

Như ví dụ dưới đây, phương thức `exec` được dùng để tìm chuỗi phù hợp theo mẫu so khớp.

```
1 | var myRe = /d(b+)d/g;
2 | var myArray = myRe.exec("cdbbdsbz");
```

Nếu bạn không cần truy cập những thuộc tính khác của biểu thức chính quy, sử dụng cách sau:

```
1 | var myArray = /d(b+)d/g.exec("cdbbdsbz");
```

Nếu bạn muốn khởi tạo một biểu thức chính quy từ một chuỗi:

```
1 | var myRe = new RegExp("d(b+)d", "g");
2 | var myArray = myRe.exec("cdbbdsbz");
```

Với những mã này, so khớp thành công và trả về một mảng kết quả với những thuộc tính được liệt kê trong bảng dưới đây.

**Bảng 4.3 Kết quả so khớp**

Đối tượng	Thuộc tính hoặc chỉ số	Mô tả	Trong vd này
myArray		Chuỗi kết quả so khớp và toàn bộ chuỗi con được nhớ.	<code>["dbbd", "bb"]</code>
	<code>index</code>	Vị trí của chuỗi tìm thấy, đếm từ 0	<code>1</code>
	<code>input</code>	Chuỗi gốc được nhập vào	<code>"cdbbdsbz"</code>
	<code>[0]</code>	Những kí tự cuối cùng được khớp.	<code>"dbbd"</code>
myRe	<code>lastIndex</code>	Vị trí nơi mà bắt đầu so khớp lần sau. (Thuộc tính này chỉ được gán nếu biểu thức chính quy sử dụng lựa chọn g, được mô tả trong <a href="#">Advanced Searching With Flags</a> ).	<code>5</code>

Đối tượng	Thuộc tính hoặc chỉ số	Mô tả	Trong vd này
	source	Chuỗi biểu thức chính quy, được cập nhật tại thời điểm biểu thức được tạo, không phải tại lúc thực thi.	"d(b+)d"

Như dạng thứ 2 của ví dụ trên, bạn có thể dùng một biểu thức chính quy được khởi tạo mà không gán nó tới một biến. Tuy nhiên, nếu bạn làm thế, mỗi lần xuất hiện là một biểu thức chính quy mới. Vì lí do này, nếu bạn không gán nó vào một biến, bạn sẽ không thể truy cập các thuộc tính của biểu thức chính quy đó nữa. Ví dụ bạn có đoạn script sau:

```
1 | var myRe = /d(b+)d/g;
2 | var myArray = myRe.exec("cdbbdsbz");
3 | console.log("The value of lastIndex is " + myRe.lastIndex);
```

Kết quả hiển thị là:

```
1 | The value of lastIndex is 5
```

Tuy nhiên nếu bạn chạy:

```
1 | var myArray = /d(b+)d/g.exec("cdbbdsbz");
2 | console.log("The value of lastIndex is " + /d(b+)d/g.lastIndex);
```

Thì kết quả hiển thị sẽ là:

```
1 | The value of lastIndex is 0
```

Sự xuất hiện của `/d(b+)d/g` trong 2 lệnh trên là những đối tượng biểu thức chính quy khác nhau và vì thế có những giá trị khác nhau cho thuộc tính `lastIndex`. Nếu bạn cần truy cập những thuộc tính của một biểu thức chính quy, bạn nên gán nó tới một biến.

## Sử dụng nhiều dấu ngoặc tròn

Sử dụng nhiều ngoặc tròn trong một biểu thức chính quy cho ta nhiều kết quả so khớp tương ứng được nhớ. Cho ví dụ, `/a(b)c/` khớp với 'abc' và nhớ 'b'. Để gọi lại những kết quả so khớp, sử dụng những phần tử của mảng `[1]...`, `[n]`.

Số lượng các chuỗi con trong những ngoặc tròn là không giới hạn. Mảng trả về giữ lại tất cả mọi thứ được tìm thấy.

## Ví dụ

Đoạn mã JavaScript dưới đây sử dụng phương thức `replace()` để giao hoán các từ trong chuỗi. Trong chuỗi thay thế, ta dùng `$1` và `$2` để chỉ các chuỗi khớp với mẫu trong ngoặc ở vị trí thứ 1 và 2.

```
1 var re = /(\w+)\s(\w+)/;
2 var str = "John Smith";
3 var newstr = str.replace(re, "$2, $1");
4 console.log(newstr);
```

Kết quả hiển thị là: "Smith, John".

## Tìm kiếm nâng cao với cờ

Biểu thức chính quy có 4 cờ cho phép tìm kiếm toàn cục và hoa thường. Những cờ này có thể được đứng riêng hoặc đứng gần nhau, và được coi như một phần của biểu thức chính quy.

**Bảng 4.4 Cờ của biểu thức chính quy**

Cờ	Mô tả
<code>g</code>	Tìm kiếm toàn cục.
<code>i</code>	Tìm kiếm không phân biệt hoa thường.
<code>m</code>	Tìm đa dòng.
<code>y</code>	Thực thi một tìm kiếm "dính" - so khớp được bắt đầu ở vị trí hiện tại trong chuỗi mục tiêu.

Firefox 3 note

Từ Firefox 3 đã hỗ trợ cờ y. Cờ y thất bại nếu so khớp không thành công ở vị trí hiện tại trong chuỗi mục tiêu.

Để sử dụng cờ trong biểu thức chính quy, dùng cú pháp này:

```
1 | var re = /pattern/flags;
```

hoặc

```
1 | var re = new RegExp("pattern", "flags");
```

Lưu ý rằng cờ là một phần hợp thành một biểu thức chính quy. Chúng không thể được thêm hoặc gỡ bỏ sau đó.

Ví dụ, `re = /\w+\s/g` tạo một biểu thức chính quy dùng để tìm kiếm một hoặc nhiều ký tự theo sau bởi một khoảng trắng, và nó tìm kiếm tổ hợp này xuyên suốt chuỗi mục tiêu.

```
1 | var re = /\w+\s/g;
2 | var str = "fee fi fo fum";
3 | var myArray = str.match(re);
4 | console.log(myArray);
```

Kết quả hiển thị là `["fee ", "fi ", "fo "]`.

Trong ví dụ này, bạn cũng có thể thay thế dòng này:

```
1 | var re = /\w+\s/g;
```

bằng:

```
1 | var re = new RegExp("\\w+\\s", "g");
```

và nhận được cùng một kết quả giống nhau.

Cờ m được sử dụng để xác định rằng một chuỗi đầu vào nên được đối xử như nhiều dòng. Nếu dùng cờ này, so khớp ^ và \$ ở đầu và cuối ở mọi dòng trong chuỗi đầu vào thay vì ở đầu và cuối của toàn bộ chuỗi.

---

## Ví dụ minh họa

Các ví dụ sau đây cho thấy một số cách sử dụng các biểu thức chính quy.

### Thay đổi thứ tự trong một chuỗi

Ví dụ sau đây minh họa sự cấu thành các biểu thức chính quy và việc sử dụng các phương thức `string.split()` và `string.Replace()`. Nó làm sạch một chuỗi đầu vào được định dạng có chứa tên (first name ở vị trí đầu tiên) cách nhau bởi khoảng trống, các tab và chỉ một dấu chấm phẩy duy nhất. Cuối cùng, nó đảo ngược thứ tự tên (last name ở vị trí đầu tiên) và sắp xếp lại danh sách.

```
1 // The name string contains multiple spaces and tabs,
2 // and may have multiple spaces between first and last names.
3 var names = "Harry Trump ;Fred Barney; Helen Rigby ; Bill Abel ; Chris Ha
4
5 var output = ["----- Original String\n", names + "\n"];
6
7 // Prepare two regular expression patterns and array storage.
8 // Split the string into array elements.
9
10 // pattern: possible white space then semicolon then possible white space
11 var pattern = /\s*;\s*/;
12
13 // Break the string into pieces separated by the pattern above and
14 // store the pieces in an array called nameList
15 var nameList = names.split(pattern);
16
17 // new pattern: one or more characters then spaces then characters.
18 // Use parentheses to "memorize" portions of the pattern.
19 // The memorized portions are referred to later.
```

```
20 pattern = /(\w+)\s+(\w+)/;
21
22 // New array for holding names being processed.
23 var bySurnameList = [];
24
25 // Display the name array and populate the new array
26 // with comma-separated names, last first.
27 //
28 // The replace method removes anything matching the pattern
29 // and replaces it with the memorized string-second memorized portion
30 // followed by comma space followed by first memorized portion.
31 //
32 // The variables $1 and $2 refer to the portions
33 // memorized while matching the pattern.
34
35 output.push("----- After Split by Regular Expression");
36
37 var i, len;
38 for (i = 0, len = nameList.length; i < len; i++){
39     output.push(nameList[i]);
40     bySurnameList[i] = nameList[i].replace(pattern, "$2, $1");
41 }
42
43 // Display the new array.
44 output.push("----- Names Reversed");
45 for (i = 0, len = bySurnameList.length; i < len; i++){
46     output.push(bySurnameList[i]);
47 }
48
49 // Sort by last name, then display the sorted array.
50 bySurnameList.sort();
51 output.push("----- Sorted");
52 for (i = 0, len = bySurnameList.length; i < len; i++){
53     output.push(bySurnameList[i]);
54 }
55
56 output.push("----- End");
57
58 console.log(output.join("\n"));
```



## Sử dụng những kí tự đặc biệt để xác thực đầu vào

Trong ví dụ dưới đây, ta mong chờ người dùng nhập một số điện thoại. Khi người dùng ấn nút "Check", đoạn script sẽ kiểm tra tính xác thực của số vừa nhập. Nếu số đó hợp lệ (khớp với chuỗi kí tự được xác định bởi biểu thức chính quy), website sẽ hiển thị một tin nhắn cảm ơn người dùng và xác nhận số đó. Nếu số đó không hợp lệ, website sẽ thông báo người dùng rằng số điện thoại vừa nhập không hợp lệ.

```
var re = /(?:\d{3}|\(\d{3}\))([-\/\.] )\d{3}\1\d{4}/;
```

Trong mẫu ngoặc tròn không nhớ (`?:`, biểu thức chính quy tìm 3 kí tự số `\d{3}` hoặc `|` 3 kí tự số trong cặp ngoặc tròn, (kết thúc mẫu ngoặc tròn không nhớ), sau đó là một kí tự gạch ngang, dấu chéo ngược, hoặc dấu chấm thập phân, và khi tìm thấy, nhớ kí tự vừa tìm được, tìm tiếp 3 kí tự số, theo sau là một so khớp được nhớ ở lần đầu tiên (`[-\/\.]`), cuối cùng là 4 kí tự số.

Sự kiện `change` được kích hoạt khi người dùng nhấp Enter.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
5      <meta http-equiv="Content-Script-Type" content="text/javascript">
6      <script type="text/javascript">
7        var re = /(?:\d{3}|\(\d{3}\))([-\/\.] )\d{3}\1\d{4}/;
8        function testInfo(phoneInput){
9          var OK = re.exec(phoneInput.value);
10         if (!OK)
11           window.alert(OK.input + " isn't a phone number with area code!");
12         else
13           window.alert("Thanks, your phone number is " + OK[0]);
14       }
15     </script>
16   </head>
17   <body>
18     <p>Enter your phone number (with area code) and then click "Check".
19     <br>The expected format is like ###-###-####.</p>
20     <form action="#">
```

```
21     <input id="phone"><button onclick="testInfo(document.getElementById  
22     </form>  
23 </body>  
24 </html>
```

[« Previous](#)[Next »](#)

---

🕒 Sửa đổi lần cuối: 23 thg 3, 2019, bởi những người đóng góp MDN

## Các chủ đề liên quan

### *JavaScript*

#### **Tutorials:**

► Complete beginners

▼ JavaScript Guide

Introduction

Grammar and types

Control flow and error handling

Loops and iteration

Functions

Expressions and operators

Numbers and dates

Text formatting

Regular expressions

Indexed collections

Keyed collections

Working with objects

Working with objects

Details of the object model

Using promises

Iterators and generators

Meta programming

JavaScript modules

► Intermediate

► Advanced

#### References:

► Built-in objects

► Expressions & operators

► Statements & declarations

► Functions

► Classes

► Errors

► Misc



# Tìm hiểu cách phát triển web tốt nhất

Nhận bản tin mới nhất và lớn nhất từ MDN được gửi thẳng vào hộp thư đến của bạn.

*Hiện tại các bản tin chỉ được viết bằng tiếng Anh.*

**Đăng ký ngay**