1. **REGULAR EXPRESSION in PHP**

## **Giới thiệu**

Trước đây việc xử lý chuỗi trong PHP rất là khó khăn bởi nó cung cấp quá ít hàm xử lý chuỗi, hoặc nếu có thì các hàm đó lại không có một chuẩn chung để kiểm tra định dạng của chuỗi. Nhưng từ ngày **Regular Expression** ra đời thì vấn đề này đã được giải quyết.

Regular Expression là một chương trình mẫu (*biểu thức chính quy*) dùng để so khớp với dữ liệu. Ngoài cách ghi này ra thì người ta còn ghi tắt là ReExp, vì vậy kể từ bây giờ nếu mình có ghi RegExp thì ban phải hiểu đó là Regular Expression nhé.

Hiện trong PHP có một số hàm dùng để xử lý RegExp như preg\_match, preg\_match\_all, ... Công dụng nhiều nhất của Regular Expression PHP đó là dùng để tách chuỗi khi lấy nội dung từ website khác, hoặc nó dùng để kiểm tra định dạng dữ liệu truyền vào có hợp lệ hay không. Đây là hai vấn đề thường gặp khi bạn làm website, vì vậy bạn không nên bỏ qua series học RegExp​ này bởi vì series này mình viết khá đầy đủ và dư cho bạn sử dụng khi xây dựng dự án. Tuy nhiên nếu bạn chỉ muốn sử dụng đơn giản thì có thể đọc vài bài đầu về các cú pháp đơn giản của Regex và các hàm xử lý Regular Expression trong PHP là được.

Regular Expression được tích hợp vào hầu hết các ngôn ngữ lập trình hiện nay, mỗi ngôn ngữ sẽ có những hàm xử lý khác nhau nhưng về chuẩn biểu thức chính quy thì thì giống nhau, vì vậy nếu bạn đã từng sử dụng biểu thức Regex ở ngôn ngữ khác thì khi qua PHP sẽ không gây khó khăn cho bạn.

## **Bài 01: Regular Expression là gì?**

Đây là bài đầu tiên trong loạt bài tìm hiểu Regular Expression và trong bài này chung ta tìm hiểu khái niệm **Regular Expression là gì**?

### **1. Regular Expression là gì?**

Regular Expression hay còn gọi là biểu thức chính quy được dùng để xử lý chuỗi nâng cao thông qua biểu thức riêng của nó, những biểu thức này sẽ có những nguyên tắc riêng và bạn phải tuân theo nguyên tắc đó thì biểu thức của bạn mới hoạt động được. Ngoài cái tên gọi Regular Expression ra thì nó còn có thể viết tắt thành RegEx.

Nguyên tắc hoạt động của biểu thức RegEx là so khớp dựa vào khuôn mẫu, khuôn mẫu được xây dựng từ các quy tắc căn bản của biểu thức RegEx. Và để các bạn nắm rõ **regular expression là gì** thì tôi sẽ giới thiệu một hàm xử lý Regular Expression trong php đó là hàm preg\_match , sau đó đưa ra một vài ví dụ nhỏ để các bạn thực hành trước khi tìm hiểu các phần nâng cao hơn.

### **2. Hàm Preg\_match trong php**

Để hiểu rõ hơn hàm này thì bạn tìm hiểu hàm preg\_match trong PHP nhé.

Cú pháp:

|  |
| --- |
| preg\_match($pattern, $subject, $matches) |

Hàm này có tổng cộng 5 tham số nhưng ở đây tôi chỉ đưa ra 3 tham số quan trọng nhất và thường hay sử dụng:

* $pattern là chuỗi Regular Expression Pattern
* $subject là chuỗi nguồn mà chúng ta muốn so khớp với $pattern
* $matches là danh sách kết quả trả về đúng khi so khớp $pattern và $subject

Sau đây là một số ví dụ về Regular Expression, bạn có thể xem và làm theo chứ không nhất thiết phải hiểu nó 100% vì mục đích tôi muốn các bạn hiểu được ý nghĩa của  nó.

**Kiểm tra một chuỗi là số**

|  |
| --- |
| if (preg\_match('/^[0-9]+$/', '123', $maches)){      var\_dump($maches);  } |

**Kết quả**:

**array**

0 => string '123' *(length=3)*

Kết quả in ra màn hình là giá trị của biến $maches, như vậy chuỗi 123 đã so khớp với pattern /^[0-9]+$/

**Kiểm tra một chuỗi là các ký tự thường**

|  |
| --- |
| if (preg\_match('/^[a-z]+$/', 'thehalfheart', $maches)){      var\_dump($maches);  } |

**Kết quả**:

**array**

0 => string 'thehalfheart' *(length=12)*

Như vậy chuỗi thehalfheart đã được so khớp vỡi pattern /^[a-z]+$/

**Kiểm tra một chuỗi là các ký tự hoa**

|  |
| --- |
| if (preg\_match('/^[A-Z]+$/', 'THEHALFHEART', $maches)){      var\_dump($maches);  } |

**Kết quả**:

**array**

0 => string 'THEHALFHEART' *(length=12)*

Như vậy chuỗi THEHALFHEART đã được khớp với pattern /^[A-Z]+$/

### **3. Lời Kết**

Như vậy là bạn đã tìm hiểu xong khái niệm **Regular Expression là gì** và biết được một thuật ngữ khác đó là **biểu thức chính quy** nên nếu có ai hỏi thì trả lời thực chất cả hai khái niệm là một nhé.

Qua ba ví dụ trên bạn phải hiểu được Regular Expression sử dụng những biểu thức và nguyên tắc riêng để đưa ra khuôn so sánh và những biểu thức đó có thể mô tả bằng ngôn ngữ bình thường được.

## **Bài 02: Các quy tắc Regular Expression căn bản (phần 1)**

Ta có thể hiểu nôm na rằng “ **Regular Expression** hoạt động theo kiểu so khớp và trả về kết quả giống nhau của sự so khớp đó ”. Ví dụ bạn làm bánh thì cái khuông chính là biểu thức Regular Expression còn còn bánh được làm ra chính là sản phẩm của sự so khớp giữa bột vài cái khuông.

Trước tiên để dễ hiểu ta quy định một số danh từ nhé.

* **pattern**: được gọi là chuỗi Regular Expression (biểu thức chính quy)
* **subject**: chuỗi so khớp với pattern
* **matches**: kết qua so khớp

Đây chính là ba tham số của hàm preg\_match mà ở bài trước ta đã tìm hiểu.

### **1. Khai báo chuỗi RegEx**

Để khai báo một chuỗi**Regular Expression** ta chỉ cần khai báo bắt đầu bằng ký tự / và kết thúc cũng là ký tự /.

|  |
| --- |
| // Khai báo chuỗi biểu thức chính quy  $pattern = '/regular\_expression\_string/'; |

**Ví dụ:**

|  |
| --- |
| // Partern kiểm tra trong subject có tồn tại chuỗi abc không  $pattern = '/abc/';  $subject = 'abc';  if (preg\_match($pattern, $subject)) {      echo 'Chuỗi regex so khớp';  } else {  echo 'Chuỗi regex không so khớp';  } |

Trong ví dụ này chuỗi $pattern = '/abc/' có ý nghĩa là tìm trong chuỗi $subject xem có tồn tại chuỗi abc không. Vì $subject = 'abc' nên kết quả hàm preg\_match trả về kết quả đúng.

Giả sử giờ tôi thay đổi giá trị của $subject như sau thì theo bạn kết quả trả về đúng hay sai nhé:

|  |
| --- |
| // Partern kiểm tra trong subject có tồn tại chuỗi abc không  $pattern = '/abc/';  $subject = 'mabcd';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } else {  echo 'Chuỗi regex không so khớp';  } |

Kết quả trả về vẫn là đúng tại vì trong $subject tuy có giá trị là mabcd nhưng tách nhỏ ra thì nó tồn tại chuỗi abc nên kết quả so khớp là đúng.

### **2. Ký tự bắt đầu và kết thúc RegEx**

Trong lập trình để kiểm tra hai biến có chính xác bằng nhau hay không thì đơn giản ta chỉ cần dùng toán tử so sánh ==. Còn trong Regular Expression có làm được điều này không? Hoàn toàn được nhé các bạn, ta sẽ dùng ký tự bắt đầu ^ và ký tự kết thúc $ đặt vào đầu và cuối chuỗi $pattern, như vậy khi so khớp sẽ so sánh từ đầu đến cuối, tức là so sánh khớp hoàn toàn.

|  |
| --- |
| // So khớp tuyệt đối chuỗi nguồn với biểu thức chính quy  // ^: bắt đầu cho 1 chuỗi. vd: ^a: Chuỗi có kí tự bắt đầu là a  // $: kết thúc cho 1 chuỗi. vd: b$ : Chuỗi có kí tự kết thúc là b  // ^string$: so cứng, chỉ là string mới đúng  $pattern = '/^regular\_expression\_string$/'; |

**Ví dụ:**

|  |
| --- |
| // Partern kiểm tra trong subject co bang freetuts khong  $pattern = '/^freetuts$/';  $subject = 'freetuts';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } else {  echo 'Chuỗi regex không so khớp';  } |

Trong ví dụ này chuỗi $pattern = '/^freetuts$/' sẽ kiểm tra xem chuỗi $subject có bằng freetuts.net không, vì $subject = 'freetuts' nên kết quả trả về là đúng. Giả sử giờ bạn thay đổi biến $subject = 'sfreetutsd' thì kết quả sẽ sai, tại vì $subject tuy xuất hiện chữ freetuts nhưng ký tự bắt đầu lại là chữ s chứ không phải chứ f và ký tự kết thúc là chữ d chứ không phải chữ s.

Nếu tới đây bạn vẫn chưa hiểu thì tôi giải thích tiếp như sau, chuỗi $pattern = '/^freetuts$/' có ý nghĩa rằng:

* Ký tự thứ 1 là chữ f và là ký tự bắt đầu chuỗi
* Ký tự thứ 2 phải là chữ r
* ký tự thứ 3 phải là chữ e
* ...
* ký tự thứ 7 phải là chữ t
* ký tự thứ 8 phải là chữ s và là ký tự kết thúc chuỗi

### **3. RegEx phạm vi của chuỗi**

Giả sử cần kiểm tra một chuỗi có phải là chữ cái trong một phạm vi nào đó hay không thì ta sẽ dùng ký hiệu [min-max], trong đó min là ký tự bắt đầu, max là ký tự kết thúc. Hoặc [list\_char] trong đó list\_char là danh sách các ký tự cho phép.

|  |
| --- |
| // Kiểm tra chuỗi (ký tự) nguồn có nằm trong phạm vi được khai báo bởi biểu thức chính quy  $pattern = '/[min-max]/';  Hoặc  $pattern = '/[list\_character]/'; |

Sau đây là các ví dụ cách sử dụng phạm vi của chuỗi Regular Expression.

**Regex kiểm tra có phải chữ cái in thường**

|  |
| --- |
| // Pattern là chữ cái từ a -> z  $pattern = '/[a-z]/';  $subject = 'd';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm tra một chữ cái in hoa**

|  |
| --- |
| // Pattern là “chữ cái” từ A -> Z  $partern = '/[A-Z]/';  $subject = 'd';  if (preg\_match($partern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm tra một ký tự là chữ số**

|  |
| --- |
| // Pattern là chữ cái từ 0 -> 9  $pattern = '/[0-9]/';  $subject = 'd';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm một chữ cái in hoa hoặc in thường**

|  |
| --- |
| // Partern là ký tự in hoa hoặc in thường  $partern = '/[a-zA-Z]/';  $subject = 'f';  // $subject = 'F';  if (preg\_match($partern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm tra một ký tự là số, chữ cái in hoa hoặc in thường**

|  |
| --- |
| // Pattern là số, ký tự in hoa hoặc in thường  $pattern = '/[a-zA-Z0-9]/';  $subject = 'f';  //$subject = 'F';  //$subject = '9';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm tra ký tự có nằm trong dãy không**

|  |
| --- |
| // Pattern là chữ a hoặc số 2 hoặc chữ b  $pattern = '/[a2b]/';  $subject = 'a';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

Trong ví dụ này ta không sử dụng cú pháp [min, max] để định nghĩa nên Regular Expression sẽ hiểu là **một trong các ký** **tự**, tức là nếu:

* $subject có chứa ký tự **a**
* $subject có chứa ký tự **2**
* $subject có chứa ký tự **b**

thì regex sẽ trả về đúng.

### **4. Xác định chiều dài của chuỗi Regex**

Để xác định chiều dài của chuỗi pattern Regex ta dùng ký hiệu {min,max}, trong đó min là chiều dài tối thiểu và max là chiều dài tối đa.

|  |
| --- |
| // Kiểm tra độ dài chuỗi nguồn  $pattern = '/Regex\_string{length\_min , length\_max}/'; |

**Regex kiểm tra là chữ in thường dài từ 5 đến 10 ký tự**

|  |
| --- |
| // So khớp  // Chuỗi bắt đầu bằng kí tự thuộc [a-z] và kết thúc bằng 1 kí tự thuộc [a-z]  // Chỉ cho phép các kí tự có mặt trong [a-z] ở trong chuỗi (kí tự chữ thường)  // có độ dài chuỗi từ 5-10  $pattern = '/^[a-z]{5,10}$/';  $subject = 'fdsfdsa';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

Trong ví dụ này chạy lên kết quả sẽ trả về đúng. Tuy nhiên nếu bạn đổi giá trị biến $subject = 'a' thì kết quả sẽ trả về sai tại vì chiều dài chuỗi $subject chỉ bằng 1.

**Regex kiểm tra là chữ số dài 3 đến 10 ký tự.**

|  |
| --- |
| // Pattern là chữ số từ 3 đến 10 ký tự  // So khớp  // Chuỗi bắt đầu bằng kí tự thuộc [0-9] và kết thúc bằng 1 kí tự thuộc [0-9]  // Chỉ cho phép các kí tự có mặt trong [0-9] ở trong chuỗi (kí tự số)  // có độ dài chuỗi từ 3-10  $pattern = '/^[0-9]{3,10}$/';  $subject = '3232';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Regex kiểm tra chuỗi có độ dài chính xác**

Giả sử ta có bài toán kiểm tra một chuỗi có phải là chữ in thường dài 5 ký tự không? Ta có 2 cách để xử lý vấn đề này:

**Cách 1**: dùng cách thông thường $partern = '/^[a-z]{5,5}$/'

|  |
| --- |
| // Pattern là chữ thường dài 5 ký tự  $pattern = '/^[a-z]{5,5}$/';  $subject = 'sssss';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Cách 2**: Ta dùng cú pháp bình thường nhưng bớt đi một biến.

|  |
| --- |
| // Pattern là chữ thường dài 5 ký tự  $pattern = '/^[a-z]{5}$/';  $subject = 'sssss';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

### **5. Lời kết**

Trong bài này chúng ta đã tìm hiểu được một số biểu thức quan trọng trước khi vào học nâng cao, những biểu thức này đang chỉ ở mức căn bản nhưng nó là tiền đề để bạn hiểu và kết hợp với những kiến thức nâng cao mà mình sắp trình bày ở các bài tiếp theo.

## **Bài 03: Các quy tắc Regular Expression căn bản (phần 2)**

Tiếp tục serie học Regular Expression trong PHP thì trong bài này ta sẽ tiếp nối bài trước đó là tìm hiểu các quy tắc căn bản trong Regular Expression, bài này mình sẽ bổ sung một số quy tắc căn bản chứ chưa đi vào phần nâng cao.

### **1. Regex đại diện cho một ký tự**

Bài trước ta đã được học cách xác định phạm vi của một chuỗi bằng cách dùng cú pháp [min-max] hoặc [list\_char], nhưng giả sử tôi muốn một Regex chấp nhận một ký tự bất kì nào đó thì không thể sử dụng cú pháp đó được. Và thật may trong Regular Expression đã cho ta một cách đó là dùng ký tự chấm đơn “.” để định nghĩa cho một kí tự bất kì.

**Ví dụ**:

|  |
| --- |
| // Pattern là ký tự bất kỳ dài từ 3 đến 10 ký tự  $pattern = '/^.{3,10}$/';  $subject = '3232';  if (preg\_match($pattern, $subject)) {      echo 'Chuỗi regex so khớp';  } |

### **2. Ký hiệu đặc biệt cho các từ khóa Regex**

Giả sử giờ tôi cần kiểm tra một chuỗi có tồn tại dấu chấm . hay không? Nếu áp dụng kí tự .thì ta sẽ làm như sau:

|  |
| --- |
| // Kết quả ko mong đợi  $pattern = '/./';  $subject = 'demo';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

Chạy đoạn code này lên kết quả trả về true, điều này sai với kết quả ta mong đợi vì trong $subject không tồn tại dấu chấm “.” . Lý do Regular Expression đã hiểu nhầm ký tự dấu chấm ở $pattern là đại diện cho ký tự bất kì nên nó so khớp với $subject là hoàn toàn đúng.

Những ký tự như dấu chấm ., mở ngoặc và đóng ngoặc vuông [], hoặc những ký tự liên quan đến quy tắc của Regular Expression đều được quy về dạng **ký tự đặc biệt trong Regular Expression**. Vì thế để phân biệt giữa ký tự đặc biệt Regex và ký tự bình thường thì ta thêm dấu “\” vào đầu ký tự đó.

Như ở ví dụ trên tôi sẽ code lại như sau:

|  |
| --- |
| // Dấu chấm là ký tự đặc biệt trong regex nên phải thêm dấu \  $partern = '/\./';  $subject = 'demo';  if (preg\_match($partern, $subject)){      echo 'Chuỗi regex so khớp';  } |

Chạy lên kết quả như mong đợi đó là trả về FALSE vì $subject = 'demo', nếu bạn đổi giá trị$subject = '.' thì kết quả sẽ TRUE.

### **3. Regex A hoặc B**

Giả sử tôi cần kiểm tra $subject = 'A' hoặc bằng $subject = 'B' sẽ trả về đúng thì ta dùng dấu “|” , đây là kí hiệu biểu diễn mối quan hệ OR.

|  |
| --- |
| $pattern = '/^A|B$/';  $subject = 'A';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

Kết quả sẽ trả về TRUE. Bây giờ bạn thử thay đổi giá trị của $subject và kiểm tra lại nhé.

### **4. Gom nhóm Regex lại**

Đôi lúc ta muốn gom nhóm Regex lại cho dễ nhìn, việc này đơn giản ta chỉ cần đặt đoạn mã Regex bên trong cặp đóng và mở (). Khi sử dụng gom nhóm thì việc so khớp vẫn bình thường, tuy nhiên với kết quả về của biến $matches thì sẽ có sự thay đổi và chi tiết thế nào thì ở phần Capturing Group dưới đây mình sẽ đề cập tới.

**Ví dụ**:

|  |
| --- |
| // Gom nhóm A hoặc B lại thành 1 nhóm  $pattern = '/(A|B)/';  $subject = 'A';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

### **5. Regex kiểm tra chiều dài không giới hạn**

Ở bài trước ta đã được học để xác định chiều dài cho chuỗi Regex $pattern thì dùng cú pháp {min,max}, tuy nhiên vẫn còn một số quy tắc ngắn gọn hơn đó là sử dụng các ký tự \*, +, ? để thiết lập chiều dài cho chuỗi.

**Ký tự \***

Đại diện cho không hoặc nhiều ký tự.

**Ví dụ**: kiểm tra chuỗi có phải trống hoặc là gồm các chữ cái thường.

|  |  |
| --- | --- |
| // Chuỗi có phải trống hoặc có những chữ cái in thường  $pattern = '/[a-z]\*/';  $subject = 'dsada';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } | // Chuỗi có phải trống hoặc có những chữ cái in thường  $pattern = '/[a-z]{0,}/';  $subject = 's';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Ký tự +**

Đại diện cho 1 hoặc nhiều ký tự.

**Ví dụ**: kiểm tra chuỗi có ít nhất một chữ thường

|  |  |
| --- | --- |
| // chuỗi ít nhất có 1 ký tự chữ thường  $pattern = '/[a-z]+/';  $subject = 's';  if (preg\_match($pattern, $subject)){  echo 'Chuỗi regex so khớp';  } | //chuỗi ít nhất có 1 ký tự chữ thường  $pattern = '/[a-z]{1,}/';  $subject = 's';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Ký tự ?**

Đại diện cho một hoặc không có ký tự nào

**Ví dụ**:

|  |  |
| --- | --- |
| // chuỗi có 1 hoặc không có ký tự thường nào  $pattern = '/[a-z]?/';  $subject = 's';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } | // chuỗi có 1 hoặc không có ký tự thường nào  $pattern = '/[a-z]{0,1}/';  $subject = 's';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

### **6. Regex phủ định - NOT**

Ta dùng ký tự ^ để phủ định một Regex nào đó, ví dụ trả về đúng nếu $subject không phải là số. (đặt nó trong [^regRex] , nếu không nó chỉ định nghĩa cho kí tự bắt đầu 1 chuỗi. )

|  |
| --- |
| // Chuoi không có ký tự số  $pattern = '/[^0-9]{1,2}/';  $subject = 'sd';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi regex so khớp';  } |

**Các ký tự Regex đặc biệt**

Danh sách các ký tự Regex đặc biệt như sau:

* \d - Chữ số bất kỳ ~ [0-9]
* \D - Ký tự bất kỳ không phải là chữ số (ngược với \d) ~ [^0-9]
* \w - Ký tự từ a-z, A-Z, 0-9 hoặc “\_” ~ [a-zA-Z0-9\_]
* \W - Ngược lại với \w (nghĩa là các ký tự không thuộc các khoảng: a-z, A-Z, 0-9 hoặc “\_”) ~[^a-zA-Z0-9\_]
* \s - Khoảng trắng (space)
* \S - Ký tự bất kỳ không phải là khoảng trắng.

**Ví dụ**: kiểm tra một chuỗi là số hoặc không phải là số

|  |
| --- |
| // Kiểm tra là số  $pattern = '/\d/';  $subject = '2';  if (preg\_match($pattern, $subject)){ echo 'Chuỗi regex so khớp';}  // Kiểm tra không phải là số  $pattern = '/\D/';  $subject = 'dsd';  if (preg\_match($pattern, $subject)){ echo 'Chuỗi regex so khớp';} |

### **7. Capturing value trong Regular Expression**

**Capturing value là gì?**để trả lời câu hỏi này tôi sẽ đưa ra một đoạn RegEx để mổ xẻ nó nhé $pattern = '/([a-z]+)([0-9]+)/'.

Trong đoạn $pattern trên ta sẽ phân tích thành 3 phần RegEx khác nhau:

* Phần 1: Là toàn bộ pattern, tức là ([a-z]+)([0-9]+)
* Phần 2: Là đoạn con nhỏ pattern đầu tiên ([a-z]+)
* Phần 3: Là đoạn con nhỏ cuối cùng ([0-9]+)

Những phần trên ta gọi là Captering Value Trong Regular Expression.

**Ví dụ**

|  |
| --- |
| preg\_match('/([a-z]+)([0-9]+)/', 'freetuts2014', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

**Kết quả:**

Array

(

[0] => freetuts2014

[1] => freetuts

[2] => 2014

)

Các bạn thấy:

* Phần tử thứ nhất tương ứng với Phần 1.
* Phần tử thứ hai tương ứng với Phần 2
* Phần tử thứ ba tương ứng với Phần 3

Các bạn phải nắm định nghĩa này nhé, nó rất là quan trong khi các bạn kết hớp xử lý với các hàm như preg\_replace, [preg\_match trong php](http://freetuts.net/ham-preg-match-trong-php-68.html)

### **8. Greedy trong Regular Expression**

Greedy là gì? Cũng như phần **Capturing Value** tôi sẽ đưa ra một ví dụ cho dễ hiểu nhé.

|  |
| --- |
| // Tìm chuỗi bắt đầu bằng h và kết thúc chữ o  preg\_match('/h(.+)o/', 'hello la xin chao', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Chạy và kết quả như sau:

Array (

[0] => hello la xin chao

[1] => ello la xin cha

)

Tại sao kết quả tứ lung tung thế nhỉ. Tại vì nó tìm chữ h và nó thấy chữ h ngay chuỗi hello đầu tiên, tiếp theo là một chuỗi bất kì và kết thúc là chữ o nên nó duyệt và thấy chữ o ở ngay chữ chao nên nó lấy từ đầu đến cuối. Như vậy có cách nào để lấy chứ **hello** đầu tiên không? Bạn xem code nhé.

|  |
| --- |
| // Tìm chuỗi bắt đầu bằng h và kết thúc chữ o  preg\_match('/h(.+?)o/', 'hello la xin chao', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Chạy lên kết quả sẽ là:

Array

(

[0] => hello

[1] => ell

)

Đây chính là kết quả ta mong đợi, sự khác biệt giữa 2 cách là ở cách 2 tôi thêm dấu ? ở kế ngay dấu + ở chuỗi pattern.

Vậy Greedy trong Regular Expression là tính chất tham ăn, nó lấy hết cho tới khi gặp ký tự cuối cùng. Và ta dùng dấu ? đặt sau các Regex để khắc phục tình trạng Greedy.

### **9. Các ví dụ về Regular Expression**

**Kiểm tra chuỗi có phải định dạng năm hay không?**

|  |
| --- |
| // Pattern đúng nếu là chữ A hoặc chữ B  // Kiểm tra một chuỗi có phải là định dạng năm không,  // Gợi ý: Định dạng năm có 4 ký tự số  $pattern = '/^\d{4}$/';  $subject = '1234';  if (preg\_match($pattern, $subject)){      echo 'Năm đúng';  } |

**Kiểm tra chuỗi có phải là số hay không**

|  |
| --- |
| $pattern = '/^[0-9]+$/';  $subject = '1234';  if (preg\_match($pattern, $subject)){      echo 'Chuoi la số';  } |

**Kiểm tra chuỗi phải là dạng ngay/thang/nam hay không**

Ví dụ ngày 20/06/2015

|  |
| --- |
| $pattern = '/^[0-9]{2}/[0-9]{2}/[0-9]{4}$/';  $subject = '12/10/2014';  if (preg\_match($pattern, $subject)){      echo 'Đúng định dạng ngày tháng năm';  } |

**Kiểm tra chỗi là ABC hoặc CDF**

|  |
| --- |
| // Chuỗi là ABC hoặc là CDF  $pattern = '/^(ABC)|(CDF)$/';  $subject = 'ABC';  if (preg\_match($pattern, $subject)){      echo 'Chuỗi so khớp';  } |

Ví du này tôi đã dùng kỹ thuật gom nhóm **Capturing Value** và dùng toán tử OR.

### **10. Lời kết**

Như vậy là mình đã giới thiệu với các bạn đầy đủ các quy tắc căn bản thường sử dụng trong Regular Expression, vẫn còn một số kiến thức nâng cao nữa như LookAhead, LookBehind, Condition, ... nhưng mình sẽ giới thiệu mở một bài khác. Và mình nhắc luôn là phần **Greedy** và **Capturing Value** rất là quan trọng nên bắt buộc bạn phải hiểu nó thì ở các bài tiếp theo mới tiếp thu bài học được.

## **Bài 04 Các quy tắc Regular Expression căn bản (tổng hợp)**

Muốn khai báo một biểu thức so mẫu:

|  |
| --- |
| $pattern = "/thiết\_lập quy\_tắc\_ở đây/"; |

Biểu thức so mẫu nằm giữa hai dấu "/", và hai dấu "/" này nằm trong cặp nháy đôi.

### **1. Meta symbol 1 (Các ký tự đặc biệt)**

Để viết regular expression ta cần làm quen với những ký tự có ý nghĩa đặc biệt trong một regular expression đó là các “meta symbol”

|  |  |
| --- | --- |
| **Meta Symbols** | **Ý nghĩa** |
| **^** | Bắt đầu dòng(string)  Ví dụ:  **/^a/** :Bắt đầu của dòng là ký tự a. Dòng này có thể là abcd, aaaa, a4684,.... miễn sao bắt đầu bằng ký tự a.  Nếu nó nằm trong [] thì nó mang nghĩa phủ định , nghĩa là các ký tự không thuộc các ký tự được khai báo bên trong []  Ví dụ:  /[^0-9]/ : Ký tự khác số (a-zA-Z và các kí tự đặc biệt đều thỏa mãn) |
| **$** | Kết thúc dòng(string)  Ví dụ:  **/^string$/**: Dòng này mang giá trị cứng là string vì nó bị chặn giữa ^ và $. Nghĩa là chỉ có chữ string mới đúng, còn lại thì sai hết. |
| **.** | Đại diện cho **một( chỉ một )** ký tự bất kỳ  **Ví dụ:**  **/a./**  :có thể mang giá trị: ab, a1, au, af .Nhưng không thể là abc, ab1, vì dấu . chi đại diện cho một ký tự! |
| **+** | Lặp lại ký tự hay cụm ký tự đứng trước nó **(>=1)** **Ví dụ:**  **/123a+/** : nghĩa là lặp lại chữ a và phải có ít nhất là một chữ a.  Do vậy mới ghi >=1 cho dễ hiểu. - 123a: 1(đúng) - 123aa: 1(đúng) - 123 : 0(sai). vì không có chữ a nào được lặp.  **/ax+/**  :biểu diễn cho dòng: ax, axx, axxxx,… nhưng không thể là "a" vì + đại diện cho >=1 ký tự được khai báo trước nó (nghĩa là ít nhất phải tồn tại 1x trong chuỗi đó). |
| **\*** | Cũng là lặp lại ký tự hay cụm ký tự đứng trước nó **(>=0)** **Ví dụ:**  **/123a\*/** : nghĩa là lặp lại chữ a, nếu không có chữ a cũng không sao. Do vậy mà >=0 -123a: (1) -123aaa: (1) -123: (1). Không có chữ a cũng không sao.  **/ay\*/**  :biểu diễn cho dòng: a, ay, ayy, ayyyyyyyyy,… |
| **?** | Tồn tại hay không tồn tại ký tự hay cụm ký tự đứng trước nó  **Ví dụ:**  **/ab?/**  :biểu diễn cho: a, ab (b có thể có hoặc không) |
| **\** | Dấu **\**đi kèm với 1 meta symbol sẽ làm mất ý nghĩa của meta symbol đó nên nó sẽ trả về symbol bình thường.  **Ví dụ:**  **/a\+b/** :biểu diễn cho dòng: "a+b" |

### **2. Meta symbol 2 (Các ký tự đặc biệt)**

Cách biểu diễn số và chữ cái

|  |  |
| --- | --- |
| **Meta Symbols** | **Ý nghĩa** |
| **\d** | **Biểu diễn một con số bất kỳ. ([0-9])**  Ví dụ:  $reg5="/^\d$/"; // chỉ có 1 chữ số  $reg5a="/^\d\d$/"; // chỉ có 2 chữ số  $reg5b = "/^10A\d\d\d\d\d\d$/"; // 10A123456 |
| **\D** | **Ký tự bất kỳ không phải là chữ số (ngược với \d ~ [^0-9])**  Ví dụ:  $reg = "/^\D$/"; // biểu diễn 1 ký tự bất kì không phải là số  $reg ="/^\d\D\d\D$/"; // Biểu diễn 1 chuỗi có dạng như là 1a2b. Với a,b là ký tự bất kỳ không phải là số |
| **\w** | **Là “ ký tự từ ” : a-z, A-Z, 0-9 , hoặc dấu gạch dưới “\_” ([a-zA-Z0-9\_])**  Ví dụ:  $reg6 = "/^\w\w\w$/"; // chỉ gồm 3 ý tự thuộc [a-z], [A->Z], [0->9] , hoặc “\_”. Không có các ký tự đặc biệt khác  $reg7 ="/^\w+\@\w+\.\w+$/"; // kiểm tra email teo@yahoo.com |
| **\W** | **Ký tự không phải là ký tự từ (nghĩa là các ký tự không thuộc các khoảng: [a-z], [A-Z], [0-9] và “\_” ~ [^a-zA-Z0-9\_])**  Nghĩa là chỉ chứa các ký tự đặc biệt  Ví dụ:  $reg = "/^\W$/"; // chứa 1 ký tự không thuộc: [a-z] , [A-Z], [0-9] |
| **\s** | **Có nghĩa: có >=1 khoảng trắng.**  Ví dụ:  $reg ="/^\w+\s\d+$/"; // biểu diễn a 1, g 22, abc 123 |
| **\S** | **Ký tự bất kì không phải là khoảng trắng (ngược với \s)** |

### **3. Dấu ngoặc trong RegEx**

|  |  |
| --- | --- |
| **Ký tự ngoặc** | **Ý nghĩa** |
| **()** | **Dùng để gom các ký tự thành một nhóm**  Ví dụ:  $reg = "/^\d(ab)+$/"; bieu dien 1ab, 1abab, 1ababab. Lặp lại ab vô hạn |
| **{}** | **Dùng để lặp ký tự hay cụm ký tự đứng trước nó với số lượng xác định**  + {3}: lặp lại chỉ 3 lần.  + {3,}: lặp lại >=3.  + {3,10}: lặp lại từ 3 đến 10 lần.  Ví dụ:  $reg8 = "/^\d{3}$/"; // lap chi 3 so  $reg8a = "/^\d{3,}$/"; // lap >=3  $reg9 = "/^a(xy){2,}$/"; // lap xy toi thieu tu 2 lan tro len  $reg9a = "/^a(xy|12){2,}$/"; // lap xy hoac 12 toi thieu 2 lan tro len |
| **|** | **Để lựa chọn hoặc cái này hoặc cái khác nếu có nhiều lựa chọn được đưa ra**  Ví dụ:  $reg16 = "/^[a-zA-Z0-9]+\.(jpg|png|gif|jpeg|JPG|PNG|GIF|JPEG)$/"; // kiem tra file hinh co duoi khac nhau |
| **[]** | **Để chỉ đoạn , một tập các ký tự cho phép được biểu diễn bởi 1 khoảng**  + [a-z] : chỉ các chữ từ a đến z  + [A-Z] : chỉ các chữ từ A đến Z  + [0-9] : chỉ các số từ 0-9  + [a-zA-z0-9] : các chữ từ a-z, A-Z, 0-9  Ví dụ:  $reg13 = "/^[3-7]$/"; // chi co 1 so trong khoang tu 3 den 7 |

### **4. Website kiểm tra RegEx online**

Để test bạn có thể sử dụng một trong các website sau:

<http://regexr.com/>

<http://www.regexpal.com/>

<https://regex101.com/>

## Bài 05: Hàm preg\_match trong php

Nếu bạn theo dõi serie học Regular Expression  từ đầu cho tới giờ thì hàm preg\_match() không còn xa lạ gì nữa, nhưng nếu bạn là khách viếng thăm và đây là bài đầu tiên của bạn thì mình khuyên bạn nên quay lại đọc từ bài “Regular Expression là gì”  thì sẽ tốt hơn. Tuy nhiên mình vẫn giới thiệu từ đầu để phù hợp với mọi đối tượng luôn.

### 1. Hàm preg\_match() trong PHP

Hàm preg\_match() được dùng để kiểm tra, so khớp và lấy kết quả của việc so sánh chuỗi dựa vào biểu thức chính quy Regular Expression (nghĩa là so sánh/kiểm tra xem chuỗi đầu vào có đúng định dạng yêu cầu hoặc có chứa chuỗi thỏa mãn định dạng hay không , trong đó Regular Expression chính là định dạng được thiết kế theo mục đích đặt ra để kiểm tra chuỗi đó ), hàm này có ba tham số và có cú pháp như sau:

|  |
| --- |
| // Kiểm tra chuỗi đầu vào có đúng định dạng khai báo trong $pattern hay không  preg\_match ( $pattern , $subject, &$matches) |

**Trong đó**:

* $pattern là biểu thức **Regular Expression**
* $subject là chuỗi cần kiểm tra
* $matches là kết quả trả về, đây là một tham số truyền vào ở dạng tham chiếu. (nó là mảng các phần được so khớp từ chuỗi đầu vào với biểu thức RegEx , mảng được lấy index theo các nhóm – phân chia bằng cặp dấu ngoặc “()” trong RegEx)

**Kết quả**:

* Kết quả của hàm preg\_match() sẽ trả về TRUE nếu so khớp (đúng định dạng yêu cầu) và FALSE nếu không khớp (không đúng với định dạng yêu cầu).

**Ví dụ**: kiểm tra một chuỗi có phải là một dãy số hay không.

|  |
| --- |
| $pattern = '/^[0-9]+$/';  $subject = '0979306603';  if (preg\_match($pattern, $subject, $matches)){      echo 'Đây là một dãy số';  } |

Nếu bạn thắc mắc tại sao biến $matches **không tồn tại** và mình truyền vào nhưng chương trình chạy không bị lỗi thì mình xin giải thích rằng đó là vì tham số thứ ba $matches là một tham chiếu nên bạn có thể truyền được như vậy, và bạn có thể đặt cho nó một cái tên bất kỳ chứ không nhất thiết phải là $matches.

Bây giờ mình sẽ thực hành một số ví dụ để bạn hiểu rõ hơn về hàm này nhé.

### 2. Các ví dụ hàm preg\_match trong php

Mình xin nhắc lại kết quả của hàm preg\_match() sẽ trả về TRUE/FALSE nên nếu bạn muốn lấy kết quả thì phải thông qua biến $matches.

**Ví dụ 1**: xem thử giá trị của biến $matches

|  |
| --- |
| $subject = "freetuts.net";  $pattern = '/net/';  preg\_match($pattern, $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => net

)

Như vậy kết quả trả về của biến $matches là một mảng các ký tự được so khớp.

Nếu bạn thắc mắc tại sao mảng trả về lại ở dạng [0] => net thì xem các ví dụ dưới đây nhé.

**Ví dụ 2**: Capturing Value - gom nhóm 1 cấp

|  |
| --- |
| $subject = "freetuts.net";  $pattern = '/(net)/';  preg\_match($pattern, $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => net

[1] => net

)

**Ví dụ 3**: Capturing Value - gom nhóm 2 cấp

|  |
| --- |
| $subject = "freetuts.net";  $pattern = '/((net))/';  preg\_match($pattern, $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => net

[1] => net

[2] => net

)

Nhận xét:

Cả 3 ví dụ trên đều cùng một mục đích là lấy ra chuỗi net, tuy nhiên cách khai báo lại khác nhau:

* Ví dụ 1: $partern = '/net/'
* Ví dụ 2: $partern = '/(net)/'
* Ví dụ 3: $partern = '/((net))/'

**Và kết quả cũng khác nhau:**

* Ví dụ 1: chuỗi $partern = '/net/' => index số 0 trong $matches là kết quả của toàn chuỗi pattern /net/.
* Ví dụ 2: chuỗi $partern = '/(net)/' => index số 0 trong $matches là kết quả của toàn chuỗi pattern /(net)/, index số 1 trong $matches là kết quả của biểu thức trong cặp đóng mở (net)
* Ví dụ 3: Chuỗi $partern = '/((net))/' => index số 0 trong $matches là kết quả của toàn chuỗi pattern /((net))/, index số 1 trong $matches là kết quả biểu thức trong cặp đầu tiên ((net)), index số 2 trong $matches là kết quả biểu thức trong cặp ngoặc cuối cùng (net)

Nếu bạn không hiểu khái niệm về **Capturing Value** thì quay lại các bài trước dể đọc nhé:

* [Quy tắc Regular Expression căn bản (p1)](http://freetuts.net/cac-quy-tac-regular-expression-can-ban-65.html)
* [Quy tắc Regular Expression căn bản (p2)](http://freetuts.net/cac-quy-tac-regular-expression-can-ban-tiep-theo-66.html)

Kết luận:

**Dựa vào nhận xét trên ta có được kết luận như sau**:

* Kết quả của biến $matches là một mảng các Capturing Value được so khớp, nhưng bao nhiêu phần tử thì phụ thuộc vào biểu thức Regular Expression.
* Biểu thức sẽ duyệt từ trái sang phải và toàn bộ Regex sẽ là nhóm thứ nhất (tức là phần tử 0), duyệt tiếp và mỗi lần gặp cặp biểu thức Regex (được gom nhóm bởi *()*) thì sẽ coi như là phần tử tiếp theo.

### 3. Một vi dụ khác hàm preg\_match

**Ví dụ 1:** Chúng ta sẽ làm một ví dụ lấy domain của http://freetuts.net nằm trong một chuỗi cho trước.

|  |
| --- |
| $subject = "website hoc mien phi [http://freetuts.net](http://freetuts.net/) rat la hay";  $pattern = '/(http)(.+)(net)/';  preg\_match($pattern, $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả sẽ trả về 4 phần tử tại vì trong pattern có 3 đoạn gom nhóm Regex, kết hợp với toàn bộ regex nữa nên kết quả sẽ là 4 phần tử.

Array

(

[0] => http://freetuts.net

[1] => http

[2] => ://freetuts.

[3] => net

)

**Ví dụ 2:** Bonus cho các bạn thêm một ví dụ nữa đó là lấy chuỗi nằm trong cặp ngoặc vuông [].

|  |
| --- |
| // Lấy chuỗi nằm giữa 2 dấu [ và ]  $subject = "Chuỗi cần lấy là [chuỗi này] mọi người";  $pattern = '/\[(.+)\]/';  if (preg\_match($pattern, $subject, $matches)){      // kêt quả matches sẽ có 2 phần tử      // phần tử thứ nhất là toàn regex =>  "[chuỗi này]"      // chuỗi thứ 2 là kết quả đoạn regex bên trong dấu () => "chuỗi này"      echo "kết quả toàn chuỗi là: <br/>";      echo '<pre>';      print\_r($matches);      echo '</pre>';        echo "Kết quả muốn lấy là: <br/>";      echo $matches[1];  } |

### 4. Lời kết

Bài này mình thấy khá là hay bởi vì kết hợp RegEx với hàm **preg\_match() trong PHP** bạn có thể lấy được bất kì một chuỗi con nào miễn là có một quy tắc để lấy, hàm này không những dùng để lấy chuỗi con dựa vào việc so khớp mà nó còn có tác dụng kiểm tra kết quả so khớp và trả về TRUE/FALSE.

Ngoài ra chúng ta hay sử dụng hàm **preg\_match()** để validate dữ liệu như kiểm tra định dạng Email, Phone, .. và một chức năng khá hay khác là bóc tách dữ liệu từ website khác.

## **Bài 06: Hàm preg\_replace trong php**

Trong bài này chúng ta tìm hiểu một hàm khá giống với hàm str\_replace đó là hàm preg\_replace.

### **1. Hàm preg\_replace trong PHP**

Hàm preg\_replace dùng để replace một chuỗi nào đó khớp với đoạn Regular Expression truyền vào. Hàm này có chức năng tương tự như **str\_replace** nhưng có sự khác biệt là một bên dùng regex một bên không dùng.

|  |
| --- |
| preg\_replace ( $pattern, $replacement, $subject) |

**Trong đó:**

* $partern: là chuỗi Regular Expression
* $replacement: là chuỗi replace thành
* $subject: là string muốn duyệt và replace

Kết quả trả về của hàm preg\_replace là chuỗi đã được replace.

### **2. Ví dụ hàm preg\_relace trong PHP**

**Ví dụ 1**: Replace chuỗi hi thành chuỗi hello trong chuỗi hi everybody, hi there

|  |
| --- |
| $partern = '/hi/';  $subject = 'hi everybody, hi there';  $replacement = 'hello';  echo preg\_replace($partern, $replacement, $subject); |

Kết quả xuất ra màn hình là chuỗi hello everybody, hello there đã được replace

**Ví dụ 2**: Xóa thẻ h1 trong chuỗi <h1>Welcome to freetuts.net</h1>

Nhận xét: Ta thấy để xóa thẻ h1 thì sẽ phải replace cho cả thẻ đóng </h1> và thẻ mở <h1> thành ký tự trống ''. Ta sẽ có 2 cách xử lý cho bài này.

**Cách 1**: Dùng toán tử OR trong Regular Expression, tức là sẽ kiểm tra **nếu là thẻ đóng hoặc thẻ mở** thì sẽ bị replace thành ký tự trống. Chuỗi regex sẽ như sau $partern = '/(<h1>)|(<\/h1>)/'. Tôi đã dùng dấu mở và đóng () để gộp 2 nhóm thẻ mở và thẻ đóng lại, luôn luôn làm như vậy nha các bạn.

|  |
| --- |
| $partern = '/(<h1>)|(<\/h1>)/';  $subject = '<h1>Welcome to freetuts.net</h1>';  $replacement = '';  echo preg\_replace($partern, $replacement, $subject); |

Các bạn chạy lên và dùng firebug soi sẽ thấy mất thẻ h1.

**Cách 2**: Dùng toán tử kiểm tra có hoặc không đó là dấu ?. Bạn quay lại bài trước để xem các ví dụ và cách dùng dấu này nhé.

Các bạn thấy thẻ mở <h1> và thẻ đóng </h1> chỉ khác nhau ở chỗ dấu /. Như vậy chuỗi partern sẽ như sau $partern = '/(<\/?h1>)/'. Vì dấu / là ký tự đặc biệt trong Regular Expression nên tôi đã thêm một dấu \ để đặt trước nó. Những điều này tôi đã nói ở bài trước rồi nên bạn quay lại xem nhé. Dấu ? là có ý nghĩa có hoặc không có ký tự, tức là <h1> hoặc </h1>

|  |
| --- |
| $partern = '/(<\/?h1>)/';  $subject = '<h1>Welcome to freetuts.net</h1>';  $replacement = '';  echo preg\_replace($partern, $replacement, $subject); |

Chạy lên kết quả tương đương như cách 1.

**Ví dụ 3**: Dùng Regular Expression xóa tất cả những thẻ html (h1, h2, h3, h4, h5) của một đoạn text

Nếu không dùng hàm preg\_replace kết hợp với Regular Expression thì trong php có cung cấp 1 hàm để xử lý vấn đề này, đó là hàm strip\_tags**.**Nhưng trong đề bài yêu cầu dùng Regular Expresision nên phải làm vậy thôi.

Ý tưởng: Ở ví dụ 2 ta đã xóa được một thẻ h1, như vậy muốn xóa các thẻ khác thì chỉ cần copy và thay đổi giá trị thẻ là được, cách làm như sau:

|  |
| --- |
| $subject = '<h1>This is h1</h1><h2>This is h2</h2><h3>This is h3</h3><h4>This is h4 </div><h5> This is h5 </h5>';  // H1  $subject = preg\_replace('/<\/?h1>/', '', $subject);  // H2  $subject = preg\_replace('/<\/?h2>/', '', $subject);  // H3  $subject = preg\_replace('/<\/?h3>/', '', $subject);  // H4  $subject = preg\_replace('/<\/?h4>/', '', $subject);  // H5  $subject = preg\_replace('/<\/?h5>/', '', $subject);    echo $subject; |

Nhưng các bạn thấy liệu có hay không? có cách này để đưa vào một partern mà xử lý hết các thẻ luôn hay không? Câu trả lời là có nhé. Bạn kết hợp ví dụ 1 và ví dụ 2 lại sẽ có được chuỗi partern mong muốn. Các bạn xem bài giải sau:

|  |
| --- |
| $subject = '<h1>This is h1</h1><h2>This is h2</h2><h3>This is h3</h3><h4>This is h4 </div><h5> This is h5 </h5>';  echo preg\_replace('/(<\/?h1>)|(<\/?h2>)|(<\/?h3>)|(<\/?h4>)|(<\/?h5>)/', '', $subject); |

Rất gọn đúng không nào, các bạn tham khảo thêm một cách giải nữa:

|  |
| --- |
| $subject = '<h1>This is h1</h1><h2>This is h2</h2><h3>This is h3</h3><h4>This is h4 </div><h5> This is h5 </h5>';  echo preg\_replace('/<\/?h(1|2|3|4|5)>/', '', $subject); |

Cách này tôi dùng toán tử OR cho các số 1 hoặc 2 hoặc 3 hoặc 4 hoặc 5. Đây là cách gọn và tối ưu nhất đúng không nào.

|  |
| --- |
| <?php  $subject1 = '<h1>This is h1</h1><h2>This is h2</h2><h3>This is h3</h3><h4>This is h4 </div><h5> This is h5 </h5>';  $pattern = '/(<\/?h[1-5]>){1,5}/';  $temp = preg\_replace($pattern , '', $subject1);  echo $temp; |

### **3. Lời kết**

Qua 3 ví dụ trên chắc hẳn bạn đã biết các sử dụng hàm preg\_replace trong php kết hợp với Regular Expression rồi nhỉ. Có rất nhiều bài toán khác nhau và ứng dụng khác nhau nên không thể quy ra một công thức chung được, các bạn cố gắng nắm vững các biểu thức Regular Expression để kết hợp là ok.

## **Bài 07: Hàm preg\_match\_all trong php**

Hàm preg\_match dùng để lấy kết quả theo biểu thức RegEx nhưng nó chỉ lấy đúng một kết quả duy nhất. Ví dụ bạn muốn lấy chuỗi '*freetuts*' trong chuỗi '*freetuts và freetuts và freetuts*' thì nó chỉ trả về đúng một giá trị '*freetuts*'. Vậy để lấy hết tất cả ba chữ thì phải làm thế nào? Ta sẻ sử dụng hàm preg\_match\_all.

### **1. Hàm preg\_match\_all trong PHP**

Hàm preg\_match\_all cũng có chức năng giống như preg\_match đó là so khớp và trả về kết quả của việc so khớp đó.

Tuy nhiên vẫn có sự khác biệt đó là:

* preg\_match chỉ chỉ trả về một kết quả cho mỗi regex con.
* preg\_match\_all sẽ trả về hết kết quả so khớp chứ không phải là kết quả đầu tiên như preg\_match.

**Ví dụ**: lấy đoạn chuỗi bên trong cặp nháy kép của chuỗi '*xin chào "các bạn", chào mừng đến "freetuts.net"*'.

**Sử dụng hàm preg\_match:**

Trong ví dụ này có hai chuỗi nằm trong cặp nháy kép đó là chuỗi '*các bạn*' và chuỗi '*freetuts.net*'. Bây giờ ta sử dụng hàm preg\_match để lấy xem kết quả thế nào.

|  |
| --- |
| $subject = 'Hello "Everybody", welcome to "freetuts.net"';  preg\_match('/"(.+)"/', $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả sẽ là:

Array

(

[0] => "Everybody", welcome to "freetuts.net"

[1] => Everybody", welcome to "freetuts.net

)

Mục đích của ta chỉ lấy đoạn text bên trong cặp nháy kép, nhưng nó trả về dài quá :D. Lý do là nó duyệt từ dấu nháy đầu chuỗi cho đến cuối chuỗi nên kết quả mới như vậy, đây gọi là hiện tượng Greedy mà chúng ta đã học ở bài các quy tắc Regular Expression căn bản. Vậy để lấy đoạn text trong cặp dấu nháy thứ nhất thì ta phải thêm dấu ? đằng sâu dấu + của chuỗi partern trên, có ý nghĩa là lấy kết quả match đầu tiên (chống greedy).

|  |
| --- |
| $subject = 'Hello "Everybody", welcome to "freetuts.net"';  preg\_match('/"(.+?)"/', $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả sẽ là:

Array

(

[0] => "Everybody"

[1] => Everybody

)

**Sử dụng hàm preg\_match\_all:**

Như vậy các bạn thấy preg\_match chỉ lấy được một kết quả duy nhất. Có cách nào lấy hết kết quả không? Để trả lời ta tìm hiểu hàm preg\_match\_all nhé.

* Cú pháp: preg\_match\_all ($pattern, $subject, &$matches)
* Trong đó:
  + $partern là biểu thức Regular Expression
  + $subject là chuỗi muốn kiểm tra
  + &$matches là biến lưu kết quả sau khi match

Như ở ví dụ trên, tức là lấy tất cả các chuỗi bên trong cặp nháy kép của chuỗi '*xin chào "các bạn", chào mừng đến "freetuts.net"*'.

|  |
| --- |
| $subject = 'Hello "Everybody", welcome to "freetuts.net"';  preg\_match\_all('/"(.+?)"/', $subject, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả là :

Array

(

[0] => Array

(

[0] => "Everybody"

[1] => "freetuts.net"

)

[1] => Array

(

[0] => Everybody

[1] => freetuts.net

)

)

Kết quả nó trả về một mảng gồm 2 phần tử và giá trị của mỗi phần tử giống như hàm kết quả của hàm preg\_match nên mình không giải thích thêm về phần này nhé.

Giờ bạn thay đổi chuỗi $subject = 'Hello "Everybody", welcome to "freetuts.net", thanks for "like it"'. Bạn chạy và kết quả sẽ là:

Array

(

[0] => Array

(

[0] => "Everybody"

[1] => "freetuts.net"

[2] => "like it"

)

[1] => Array

(

[0] => Everybody

[1] => freetuts.net

[2] => like it

)

)

Rất đơn giản đúng không nào. Bây giờ lấy kết quả thì có 2 lựa chọn:

* Nếu lấy có dấu ngoặc thì chọn phần tử thứ nhất
* Nếu lấy không có dấu ngoặc thì chọn phần tử thứ 2

Giải thích tại sao lại có 2 phần tử thì như bài trước, phần tử thứ nhất là toàn chuỗi toàn partern, phần tử thứ 2 là kết quả của đoạn RegEx (.+?).

### **2. Ví dụ hàm preg\_match\_all trong PHP**

Mình sẽ đưa ra một ví dụ để các bạn thực hành như sau: Lấy tất cả nội dung bên trong tất cả thẻ div của một file html.

Ta có hai vấn đề cần quan tâm:

* Để lấy tất cả các đoạn text trong tất cả thẻ div thì ta phải dùng đến hàm preg\_match\_all trong php.
* Đoạn $pattern sẽ có dạng $pattern = '/<div>(.\*?)<\/div>/'. Lưu ý phải có dấu ? nha bạn, vì có dấu ? thì máy sẽ hiểu là lấy từng kết quả chứ không phải duyệt hết chuỗi rồi lấy (gọi là Greedy, bạn quay lại bài trước để xem phần này nhé).

|  |
| --- |
| $subject = '<div>Div1</div><div>Div2</div><div>Div3</div>';  preg\_match\_all('/<div>(.\*?)<\/div>/', $subject, $matches);  echo '<pre>';  var\_dump($matches);  echo '</pre>'; |

Kết quả sẽ là:

array(2)

{

  [0]=> array(3)

    {

        [0]=> string(15) "Div1"

        [1]=> string(15) "Div2"

        [2]=> string(15) "Div3"

}

  [1]=> array(3)

    {

      [0]=> string(4) "Div1"

      [1]=> string(4) "Div2"

      [2]=> string(4) "Div3"

    }

}

### **3. Lời kết**

Như vậy để lấy tất cả kết quả thì ta sử dụng hàm preg\_match\_all còn chỉ lấy một kết quả đầu tiên thì ta sử dụng hàm preg\_match. Thực tế thì bạn sử dụng hàm preg\_match\_all để thay thế cho hàm preg\_match cũng được bằng cách lấy phần tử đầu tiên, tuy nhiên cách này rất rườm rà nên tùy vào trường hợp mà sử dụng cho hợp lý.

## **Bài 08: None Capturing Group trong Regular Expression**

Chúng ta đã được học qua khái niệm Capturing Value trong bài [các quy tắc Regular Expression căn bản](http://freetuts.net/cac-quy-tac-regular-expression-can-ban-65.html)thì trong bài này ta sẽ đến một khái niệm khác đó là **None Capturing Group**, đây là một khái niệm ngược lại với Capturing Value trong Regular Expression.

### **1. None Capturing Group là gì?**

Như ta biết Capturing Value sẽ xác định giá trị trả về cho từng biểu thức RegEx con, nhưng đôi lúc ta lại muốn biểu thức con  đó không xuất hiện trong kết quả trả về thì làm thế nào? Để giải quyết vấn đề này ta dùng **None Capturing Group** trong Regular Expression.

**Cú pháp:** /regex**:?**regex/ trong đó dấu :? chính là cú pháp khai báo None Capturing Group.

Để dễ hiểu hơn ta tìm hiểu một số ví dụ nhé (*Code PHP*).

**Ví dụ**: Lấy tất cả chuỗi trong dấu ngoặc kép *lấy luôn dấu ngoặc kép*)

|  |
| --- |
| preg\_match('/"(?:.+)"/', 'program "Hello World"', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Chạy và kết quả là:

Array

(

[0] => "Hello World"

)

Nếu bạn đã đọc qua phần **Capturing Value** và bài hàm preg\_match trong php thì bạn sẽ thấy có sự bất thường ở đây. Lẽ ra phải trả về 2 kết quả vì ta có:

* Pattern toàn bộ:/"(?:.+)"/
* Pattern con: (?:.+)

Nhưng vì ta đã dùng **None Capturing Group** ở biểu thức RegEx con (?:.+) nên biểu  thức con này không được liệt kê kết quả trả về.

Nếu ta không dùng None Capturing Group thì code sẽ như sau:

|  |
| --- |
| preg\_match('/"(.+)"/', 'program "Hello World"', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Chạy kết quả là:

Array

(

[0] => "Hello World"

[1] => Hello World

)

Bạn so sánh 2 kết quả và tự suy nghĩ nhé :D.

**Từ ví dụ trên ta có kết luận**:

**None Capturing** Group là kỹ thuật không cho một đoạn Regex nào đó xuất hiện trong kết quả trả về

### **2. Một ví dụ None Capturing Group**

Bây giờ mình sẽ làm một vài ví dụ để bạn dễ nắm bắt hơn.

**Ví dụ 2**: Cho chuỗi$subject = 'program (None capturing group "Welcome To You")', Hãy lấy nội dung chữ None capturing group trong chuỗi trên, kết quả trả về bỏ đi đoạn Welcome To You

|  |
| --- |
| preg\_match('/\((.+)"(?:.+)"\)/', 'program (None capturing group "Welcome To You")', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => (None capturing group "Welcome To You")

[1] => None capturing group

)

Nếu bạn bỏ đi biểu thức None Capturing Group thì kết quả sẽ như sau:

|  |
| --- |
| preg\_match('/\((.+)"(.+)"\)/', 'program (None capturing group "Welcome To You")', $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => (None capturing group "Welcome To You")

[1] => None capturing group

[2] => Welcome To You

)

### **3. Lời Kết**

Phần None Capturing Group này thực ra cũng ít sử dụng, nhưng bạn cũng nên biết nó để xử lý nhuần nhuyễn RegEx hơn nhé. Bài tiếp theo mình nghiên cứu Lookbehind trong Regular Expression

## **Bài 09: Negative Và Positive Lookbehind trong Regular Expression**

Lookbehind dùng để xác định sự chọn lựa ở phía trước của một chuỗi Regular Expression, gồm có 2 loại là **Negative Lookbehind** và **Positive Lookbehind**. Đây là một quy tắc rất hay sử dụng để xử lý chuỗi nên nếu bạn bỏ qua phần này là một sai lầm đấy.

Để dễ hình dung tôi đưa ra một ví dụ nhé:

**Ví dụ 1**: Hãy lấy tất cả những phần đuôi email của *thehalfheart*trong chuỗi thehalfheart@gmail.com, freetuts@facebook.com,thehalfheart@yahoo.com. Kết quả muốn lấy sẽ là *@gmail.com* và *@yahoo.com*

Nếu dùng các quy tắc Regular Expression căn bản thì ta sẽ làm như sau:

|  |
| --- |
| $string = 'thehalfheart@gmail.com, freetuts@facebook.com, thehalfheart@yahoo.com';  preg\_match\_all('/@[^,]+/', $string, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => Array

(

[0] => @gmail.com

[1] => @facebook.com

[2] => @yahoo.com

)

)

Đây không phải là kết quả mong đợi của ta vì dư @facebook.com. Giờ nếu ta đưa luôn chuỗi *thehalfheart* vào chuỗi Regular Expression như thế nào? Ta cùng xem code nhé:

|  |
| --- |
| $string = 'thehalfheart@gmail.com, freetuts@facebook.com, thehalfheart@yahoo.com';  preg\_match\_all('/thehalfheart@[^,]+/', $string, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => Array

(

[0] => thehalfheart@gmail.com

[1] => thehalfheart@yahoo.com

)

)

Kết quả đúng là trả về 2 email của *thehalfheart* nhưng lại dư chữ *thehalfheart* nên không đúng như yêu cầu đề bài.

Vậy giải pháp của ta là sử dụng **Positive Lookbehind trong Regular Expression**để đưa vào chuỗi RegEx, nhưng trước tiên ta xem cú pháp nó đã:

### **Positive Lookbehind**

**Cú pháp**: **(?<=)**

**Ý tưởng**:

Ở bài toán **Ví dụ 1** ta thấy thông tin cần lấy là phần đuôi của email có phần đầu là *thehalfheart.* Như vậy ta sẽ dùng Positive Lookbehind để chọn lựa, nếu nói tiếng việt thì thế này: Lấy phần đuôi của email có phần đầu bằng thehalfheart. Các bạn xem bài giải nhé:

|  |
| --- |
| $string = 'thehalfheart@gmail.com, freetuts@facebook.com, thehalfheart@yahoo.com';  preg\_match\_all('/(?<=thehalfheart)@[^,]+/', $string, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => Array

(

[0] => @gmail.com

[1] => @yahoo.com

)

)

Thật tuyệt vời, kết quả như mong đợi =))

**Giải thích**:

Ở chuỗi partern trong đoạn code trên ta tôi đã đưa vào một đoạn Positive Lookbehind (?<=thehalfheart) nên hệ thống sẽ hiểu là bắt đầu bằng *thehalfheart* và vì chữ *thehalfheart* nằm trong phần Lookbehind nên nó không nằm trong kết quả trả về.

### **Negative Lookbehind**

**Cú pháp**: **(?<!)**

Ngược với Positive, Negative sẽ lựa chọn với điều kiện là không giống. Giờ ta sẽ làm một ví dụ cho nó xôn nha.

**Ví dụ 2**: Lấy phần đuôi của email không phải là thehalfheart trong chuỗi sau: thehalfheart@gmail.com,freetuts@facebook.com, thehalfheart@yahoo.com, room@freetuts.net. Có nghĩa kết quả trả về là *@facebook.com* và*@freetuts.net*.

**Bài giải**:

|  |
| --- |
| $string = 'thehalfheart@gmail.com, freetuts@facebook.com, thehalfheart@yahoo.com, room@freetuts.net';  preg\_match\_all('/(?<!thehalfheart)@[^,]+/', $string, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Kết quả:

Array

(

[0] => Array

(

[0] => @facebook.com

[1] => @freetuts.net

)

)

Ôi trời, kêt quả đúng rồi, sao hay thế nhỉ?

**Giải thích**:

Tương tự như giải thích phần Positive, chuỗi partern có một đoạn Negative Lookahead (?<!thehalfheart) nên hệ thống sẽ hiểu là lấy những email nào có phần đầu không phải là *thehalfheart*, và chữ *thehalfheart* nằm trong lookahead nên nó sẽ không có trong kết quả trả về.

### **Lời Kết**

Qua 2 ví dụ trên căn bản các bạn cũng hình dung được phần nào rồi nhỉ, các bạn hãy làm theo ví dụ, tự ra đề và tự giải để thưởng thức thành quả của bài học **Positive và Positive Lookahead Trong Regular Expressoin** nhé

## **Bài 10: Negative Và Positive Lookahead trong Regular Expression**

Nếu Lookbehind dùng để kiểm tra điều kiện ở phía trước chuỗi thì Lookahead ngược lại, nó sẽ kiểm tra điều kiện ở phía sau. Cũng tương tự như [lookbehind](http://freetuts.net/negative-va-positive-lookbehind-trong-regular-expression-72.html)sẽ có 2 dạng, đó là Positive và Negative.

Không có gì bằng thấy tận mắt, sờ tận tay đúng không nào? Nên ta sẽ bắt đầu một ví dụ nhé.

**Ví dụ 1**: Lấy phần đầu của email có phần đuôi là @gmail.com trong chuỗi 'freetuts@gmail.com thehalfheart@facebook.comhoclaptrinh@yahoo.com freetuts.net@gmail.com'.

Hình như ví dụ này hơi quen quen, các bạn nhớ ra chưa nhỉ ? Nếu nhớ ra thì comment ở dưới để tôi biết với nhé :D.

Ta sẽ giải bài này thông qua **Positive Lookahead Regular Expression**.

### **Positive Lookahead Trong Regular Expression**

**Cú pháp: (?=)**

Ở ví dụ một ta sẽ dùng Positive Lookahead để xác định câu partern 'chuỗi nào có phần đuôi bằng *@gmail.com* thì sẽ chọn'. Các bạn xem code nhé.

|  |
| --- |
| $str = 'freetuts@gmail.com thehalfheart@facebook.com hoclaptrinh@yahoo.com freetuts.net@gmail.com';  preg\_match\_all('/[^\s]+(?=@gmail\.com)/', $str, $matches);  echo '<pre>';  print\_r($matches);  echo '</pre>'; |

Chạy lên và kết quả là:

Array

(

[0] => Array

(

[0] => freetuts

[1] => freetuts.net

)

)

Bạn có thấy chuỗi partern mình có thêm một đoạn (?=@gmail\.com) không, đoạn này chính là Positive Lookahead xác định chuỗi @*gmail.com*. Còn đoạn partern đầu tiên [^\s]+ là tìm dãy ký tự cho tới khi gặp khoảng trắng. Và kết quả đúng như mong đợi.

### **Negative Lookahead trong regular expression**

**Cú pháp: (?!)**

Ở phần trên ta đã tìm được phần đầu của những email có phần đuôi là @gmail.com, bây giờ giả sử tôi muốn tìm kiếm phần đầu của những email có phần đuôi không phải là @gmail.com. Áp dụng cú pháp và tương tự như Pisitive Lookahead ta có lời giải như sau:

|  |
| --- |
|  |

1. **REGULAR EXPRESSION in JAVASCRIPT**

## **Phần 1: Lý thuyết**

* 1. **RegEx là gì?**

RegEx (Regular Expression) là một biểu thức thường được dùng để mô tả một mẫu chuỗi (string pattern).

### **Vậy, mẫu chuỗi là gì?**

Mẫu chuỗi, diễn giải thô ra có nghĩa là các chuỗi văn bản tuân thủ theo một quy luật sắp xếp (mẫu) nào đó.  
  
Ví dụ:   
- Một số được viết dưới dạng một chuỗi với các chữ số, có thể có chứa tối đa 1 dấu phẩy ở giữa.  
- Dữ liệu kiểu ngày tháng: Thường được viết theu mẫu: dd/mm/yyyy: Hai chữ số đầu tiên xác định ngày, tiếp theo là dấu gạch chéo, tiếp đến là 2 chữ số xác định tháng, tiếp theo là dấu gạch chéo và kết thúc bởi bốn chữ số xác định năm.  
- Địa chỉ email: xxx@yyy.zz.gìđó: Bao gồm 1 chuỗi văn bản được chia thành 2 phần: Phần user name và phần domain, 2 phần này cách nhau bởi dấu @. Phần domain có luật riêng theo cách đặt tên của domain...  
...  
  
**3. Khái niệm về biểu thức:**  
  
Một biểu thức bao gồm các toán tử và các toán hạng  
Toán tử: Các phép toán (VD: cộng, trừ, nhân, chia, lặp, gộp, so sánh…)  
Toán hạng: Biến, giá trị được dùng để tính toán.  
  
**4. Vậy regex được dùng để làm gì và nó có tác dụng gì?**  
  
Trong các kỹ thuật xử lý chuỗi (VD: Tìm kiếm, thay thế...) thông thường, chúng ta cần phải xác định các chuỗi tường minh để tìm kiếm. Các chuỗi này được gọi là từ khoá.  
  
Tuy nhiên, trong một số trường hợp ngoại lệ khác, chúng ta không thể chỉ định rõ chuỗi từ khoá đó, mà chỉ có thể mô tả được quy luật để sinh ra chuỗi từ khoá đó.  
  
VD: Làm thế nào để lấy được toàn bộ địa chỉ email trong một văn bản?  
Rõ ràng với câu hỏi này, bằng mắt thừong thì ta có thể lấy được, còn bằng con đường lập trình thông thường theo kiểu tìm kiếm bằng từ khoá là các địa chỉ email thì chúng ta ... bó tay.   
  
Nhưng nếu có regex, chúng ta có thể thiết lập ra quy luật của một địa chỉ email và yêu cầu chương trinh tìm kiếm theo luật đó.  
  
**Cú pháp khai báo regex trong JavaScript**  
  
Chúng ta có thể khai báo biến chứa regex như sau:

Code:

var tên\_biến= /mẫu\_chuỗi/cách\_tìm;

Tham số cách\_tìm xác định cách thức tìm kiếm mẫu chuỗi, nó có thể chứa các giá trị sau đây:  
g: ("global" matching): Lặp lại quá trình tìm kiếm cho đến khi kết thúc chuỗi. Khi sử dụng phương thức replace(), giá trị này sẽ thay thế tất cả các cụm từ tìm thấy thay vì chỉ cụm từ đầu tiên.  
i: so sánh không phân biệt hoa thường.  
m: so sánh ở chế độ đa dòng.  
Có thể phối hợp nhiều cách\_tìm. VD: gi. Cũng có thể không cần đưa tham số này vào  
  
**Các phương thức có hỗ trợ regEx trong JavaScript:**  
  
Giả sử có một biến chuỗi S:  
S.search(): Tìm chuỗi phù hợp với biểu thức đã cho và trả về vị trí của chuỗi đó  
S.match(): Tìm chuỗi theo mẫu và trả về một mảng các giá trị tìm thấy  
S.replace() Thay thế chuỗi bằng chuỗi khác phù hợp với mẫu tìm kiếm.  
...  
Các bạn có thể tìm hiểu cách sử dụng các hàm này trên mạng, ở đây tôi chỉ sử dụng hàm match để demo cách sử dụng regex. Nên nhớ, hàm match() này trả về một mảng các giá trị phù hợp với mẫu chuỗi cần tìm.  
  
**Ví dụ đơn giản đầu tiên về regex:**  
Đoạn mã dưới đây trả về mảng các chuỗi có chứa chữ thu:

Code:

<html>

<body>

<script type="text/javascript">

var str = "Ngoai hien giot mua thu thanh thot roi";

var patt1 = /thu/; // Mẫu chuỗi xác định một chữ thu

var result = str.match(patt1);

document.write(result); // In ra chữ thu

</script>

</body>

</html>

Đoạn ví dụ thứ 2 chỉ tìm kiếm các mẫu chuỗi chứa duy nhất chữ t:

HTML Code:

<html>

<head>

<script>

var s="Thu mot ti, thu hai Ti";

//regex: t/g xác định việc tìm kiếm tất cả các chuỗi là chữ t,

//và tiến hành tìm từ đầu đến cuối chuỗi nhờ tham số cách tìm /g.

document.write(s.match(/t/g)); //Kết quả: t,t,t

//regex: t/gi xác định việc tìm kiếm tất cả các chuỗi là chữ t,

//và tiến hành tìm từ đầu đến cuối chuỗi không phân biệt chữ hoa, chữ thường nhờ tham số cách tìm /gi.

document.write(s.match(/t/gi)); //Kết quả: T,t,t,t,T

</script>

</head>

</html>

**Các phép toán tìm kiếm theo vị trí**  
  
^: Trả về chuỗi kết quả trong trường hợp chuỗi này nằm ở vị trí **đầu**của chuỗi gốc  
$: Trả về chuỗi kết quả trong trường hợp chuỗi này nằm ở vị trí **cuối**của chuỗi gốc  
\b: Trả về chuỗi kết quả trong trường hợp chuỗi này nằm ở vị trí **đầu của một từ** trong chuỗi gốc. Nếu cần so sánh ở vị trí cuối từ, hãy đặt biểu thức \b ở vị trí cuối từ.  
\B: Trả về chuỗi kết quả trong trường hợp chuỗi này **không nằm ở vị trí đầu của một từ** trong chuỗi gốc. (tuỳ thuộc vào vị trí đặt \B ở đầu hoặc cuối một từ)  
?=: Trả về chuỗi kết quả nếu theo sau chuỗi đó là một chuỗi nào đó được chỉ định trước  
?!: Trả về chuỗi kết quả nếu sau chuỗi đó không phải là một chuỗi nào đó được chỉ định trước  
  
Ví dụ: 

HTML Code:

<html>

<head>

<script>

var s="Thu mot ti, thu hai Ti";

document.write(s.match(/^Thu/i)); //Kết quả: Thu, vì chữ Thu nằm ở đầu chuỗi s

document.write(s.match(/t$/gi)); //Kết quả: null, vì ko tìm thấy chữ t nằm ở cuối chuỗi s

document.write(s.match(/ti$/gi)); //Kết quả: Ti, vì chữ Ti nằm ở vị trí cuối chuỗi s và thoả mãn điều kiện tìm kiếm /gi (tìm toàn cục ko phân biệt chữ hoa, chữ thường)

document.write ("<BR>");

document.write(s.match(/\bti/gi));//Kết quả: ti,Ti vì 2 chữ này nằm ở vị trí đầu của 2 từ ti, Ti trong chuỗi s

document.write ("<BR>");

document.write(s.match(/\Bai/gi));//Kết quả: ai, vì chữ này nằm ở phía cuối của chữ hai trong chuỗi s

document.write ("<BR>");

document.write(s.match(/th(?=u)/gi));//Kết quả: Th,th, vì 2 chữ này có theo sau (?=) là u, là 2 chữ Thu, thu trong chuỗi gốc.

document.write ("<BR>");

document.write(s.match(/t(?!h)/gi));//Kết quả: t,t,T, vì 3 chữ này không theo sau (?!) bởi chữ h, tương ứng với t , ti, Ti trong chuỗi gốc.

</script>

</head>

</html>

**Các lớp ký tự trong regex:**  
  
[xyz]: Tìm một ký tự bất kỳ nằm trong tập ký tự giữa cặp dấu ngoặc vuông  
[x-z]: Tìm một ký tự bất kỳ nằm trong tập ký tự từ x đến z  
[^xyz]: Tìm một ký tự bất kỳ không thuộc tập ký tự giữa cặp dấu ngoặc vuông.  
.: Tìm bất kỳ một ký tự nào không phải là ký tự xuống dòng mới (new line) hoặc ký tự kết thúc dòng (line terminator).  
\w: Tìm một ký tự dạng a-Z, 0-9 và dấu gạch dưới.  
\W: Ngược lại với \w  
\d: Tìm một ký tự thuộc tập ký tự từ 0 đến 9  
\D: Ngược lại với \d: Tìm một ký tự không nằm trong tập ký tự từ 0 đến 9  
\s: Tìm ký tự cách (dấu cách)  
\S:Tìm một ký tự không phải là dấu cách  
  
Ví dụ: 

HTML Code:

<html>

<body>

<script type="text/javascript">

var str = "1 voi 1 la 2, 2 them 2 la 4";

// trả về mảng chứa các ký tự trong đoạn từ e tới h

document.write(str.match(/[e-h]/gi));//Kết quả: h,e, với h,e nằm trong chữ them.

document.write ("<BR>");

// Trả về mảng các ký tự không thuộc tập các ký tự từ a-m và ký tự u:

document.write(str.match(/[^a-mu ]/gi));//Kết quả: 1,v,o,1,2,,,2,t,2,4

document.write ("<BR>");

// Tìm các chuỗi con bắt đầu bởi chữ oi, tiếp theo sau đó là 1 ký tự thuộc lớp \W:

document.write(str.match(/oi\W/gi));//oi

document.write ("<BR>");

// Tìm các ký tự là các chữ số:

document.write(str.match(/\d/gi));//1,1,2,2,2,4

document.write ("<BR>");

// Tìm các chuỗi bắt đầu bởi 1 ký tự thuộc lớp \D, sau đó là 1 ký tự không thuộc nhóm từ b-t:

document.write(str.match(/\D[^b-t ]/g));//v, 1,la, 2, 2, 2,la, 4

document.write ("<BR>");

// Tìm các dấu cách:

document.write(str.match(/\s/g));//, , , , , , , ,

</script>

</body>

</html>

**Các phép lặp trong regex**  
  
RegEx cho phép tìm kiếm lặp bên trong biểu thức:  
{x}: Lặp một ký tự hoặc một biểu thức con trước đó x lần  
{x,y}: Lặp một ký tự hoặc một biểu thức con trước đó từ x đến y lần  
{x,}: Lặp một ký tự hoặc một biểu thức con trước đó >= x lần  
?: Lặp một ký tự hoặc một biểu thức con trước đó 0 hoặc 1 lần  
\*: Lặp một ký tự hoặc một biểu thức con trước đó >=0 lần  
+: Lặp một ký tự hoặc một biểu thức con trước đó >=1 lần  
  
Ví dụ:

HTML Code:

<html>

<body>

<script type="text/javascript">

var str = "Ai mua xoi deeeeee";

// Tìm một chuỗi bao gồm các chữ cái d hoặc e (lớp [de], lặp lại 3 lần:

document.write (str.match(/[de]{3}/g));//dee,eee

// Tìm một chuỗi bao gồm các chữ cái d hoặc e, lặp lại >=3 lần:

document.write (str.match(/[de]{3,}/g));//deeeeee

// Tìm một chuỗi bao gồm các chữ cái d hoặc e, lặp lại từ 3 đến 5 lần:

document.write (str.match(/[de]{3,5}/g))//deeee

// Tim chu d, sau chu d la 0 hoac 1 chu e

document.write (str.match(/de?/g));

// Tim chu d, sau chu d la 0 hoac n chu e

document.write (str.match(/de\*/g));

// Tim chu d, sau chu d la 1 hoac n chu e

document.write (str.match(/de+/g));

//Tim tat ca cac tu bao gom cac chu cai tu a den z

document.write (str.match(/[a-z]\*/gi));

//Tim tat ca cac tu bao gom cac chu cai tu a den z, sau tu la mot dau cach:

document.write (str.match(/[a-z]\* +/gi));

</script>

</body>

</html>

**Gộp nhóm các biểu thức**  
Chúng ta có thể sử dụng các dấu ngoặc tròn () để gộp nhóm như trong các biểu thức toán học thông thường.  
  
(): Tìm kiếm một nhóm các ký tự bên trong cặp dấu ngoặc và lưu vào chuỗi kết quả.  
(?: ): Tìm kiếm chuỗi kết quả không chứa tập ký tự nằm trong cặp dấu ngoặc.  
|: Phép toán hoặc, được sử dụng để kết hợp các mệnh đề với nhau vào chung một biểu thức  
  
Ví dụ:

HTML Code:

<html>

<body>

<script type="text/javascript">

var str = "Email: mail@ttt.com, test@ttt.com.vn, sinhln@hnue.edu.vn";

// Tim tat ca cac dia chi email trong chuoi

document.write (str.match(/([a-zA-Z0-9\_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+/gi));

//Tim tat ca cac so trong chuoi

str=" 12312 and 343 and 54q";

document.write (str.match(/[0-9]+/g));

// hoac

document.write(str.match(/\d+/g));

</script>

</body>

</html>

***Phần 2: Một số trường hợp ứng dụng***  
  
  
  
  
  
  
Qua phần đó, các bạn hẳn đã hiểu được regex là cái gì, và bước đầu có thể đọc hiểu được ý nghĩa của một chuỗi regex. Bài này đi vào tìm hiểu một số ví dụ ứng dụng regex để xử lý các bài toán trong thực tiễn.  
  
**I. Tìm kiếm và thay thế**  
Hẳn các bạn đã biết, phương thức replace() trong Javascript nếu ở chế độ bình thường thì nó sẽ chỉ tìm và thay thế được duy nhất chuỗi đầu tiên được tìm thấy.  
VD:  
  
Code:  
<script language="javascript">  
var str="chuoi 1, chuoi 2, chuoi 3";  
document.write (str.replace ("chuoi","cam")); // Ket qua: cam 1, chuoi 2, chuoi 3   
</script>  
  
Vậy làm thế nào để thay thế được tất cả các từ "chuoi" thành "cam"?  
  
Ở đây, chúng ta có thể sử dụng các phép tìm kiếm lặp của regex:

HTML Code:

<script language="javascript">

var str="chuoi 1, chuoi 2, chuoi 3";

// Chuỗi cần tìm là 1 regex: /chuoi/gi, xác định tìm kiếm từ chuoi lặp lại cho đến cuối chuỗi gốc và không phân biệt hoa thường

document.write (str.replace (/chuoi/gi,"cam")); // Ket qua: cam 1, cam 2, cam 3

</script>

**II. Hỗ trợ kiểm tra dữ liệu theo mẫu regex**  
  
Ở đây các bạn cần làm quen với một phương thức trong lớp RegExp của javascript, đó là phương thức test().  
  
Cú pháp:  
biến\_regex.test(chuỗi\_gốc)  
  
Phương thức này trả về true nếu như chuỗi gốc khớp với giá trị trả về của biến\_regex. Ngược lại là false.  
  
Ví dụ:

HTML Code:

<script language="javascript">

// Khai báo 1 chuỗi thông thường

var str="Hello world!";

//Khai báo 1 chuỗi mẫu (biểu thức regex)

var patt=/Hello/g;

var result=patt.test(str);

document.write("Ket qua: " + result); // Tra ve true vi tim thay tu Hello trong chuoi str

//Một mẫu chuỗi khác

patt=/phpvn/g;

result=patt.test(str);

document.write("<br />Ket qua: " + result); // Tra ve false vi khong thay chuoi phpvn trong chuoi str.

</script>

**a. Kiểm tra xem chuỗi nhập vào có phải là 1 số hay không?**  
  
Bình thường, chúng ta có thể sử dụng các hàm kết hợp như: isNaN, parseInt, parseFloat... để kiểm tra xem đó có phải là một số hay không? Ở đây tôi cung cấp một cách khác dùng regex:  
  
Đoạn mã dưới đây sẽ chỉ kiểm tra xem đây có phải là 1 số nguyên dương hay không:

HTML Code:

<script language="javascript">

function phpvn\_isDigit(myString)

{

var reg = /^\d+$/;

return (reg.test(myString));

}

document.write (phpvn\_isDigit("12345"));

</script>

Đoạn mã dưới đây sẽ kiểm tra xem có phải là đầu vào là 1 số nguyên hay ko (có thể có thêm dấu +/- ở trước):

HTML Code:

function phpvn\_isInt(myString)

{

var reg= /^(\+|-)?\d+$/;

return (reg.test(fData));

}

Tiếp tục nâng cấp kiểm tra xem chuỗi đầu vào có phải là một số thực hay không.  
Một chuỗi được coi là 1 số thực nếu như nó bắt đầu bởi 1 chuỗi số, có 1 dấu chấm ở giữa và kết thúc bởi 1 chuỗi số:

HTML Code:

function phpvn\_isFloat(myString)

{

var reg= /^(\+|-)?((\d+(\.\d+)?)|(\[0].\d+))$/;

return (reg.test(myString));

}

Nhưng mẫu chuỗi trên sẽ trả về false nếu người dùng gõ 1 chuỗi dạng 123,123,123.45