**REGULAR EXPRESSION 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?**

|  |
| --- |
| **Mục luc:**   * 1. Regular Expression là gì? * 2. Hàm Preg\_match trong php   + Kiểm tra một chuỗi là số   + Kiểm tra một chuỗi là các ký tự thường   + Kiểm tra một chuỗi là các ký tự hoa * 2. Lời Kết |

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)**

|  |
| --- |
| **Mục luc:**   * 1. Khai báo chuỗi RegEx * 2. Ký tự bắt đầu và kết thúc RegEx * 3. RegEx phạm vi của chuỗi   + Regex kiểm tra có phải chữ cái in thường   + Regex kiểm tra một chữ cái in hoa   + Regex kiểm tra một ký tự là chữ số   + Regex kiểm một chữ cái in hoa hoặc in thường   + Regex kiểm tra một ký tự là số, chữ cái in hoa hoặc in thường   + Regex kiểm tra ký tự có nằm trong dãy không * 4. Xác định chiều dài của chuỗi Regex   + Regex kiểm tra là chữ in thường dài từ 5 đến 10 ký tự   + Regex kiểm tra là chữ số dài 3 đến 10 ký tự.   + Regex kiểm tra chuỗi có độ dài chính xác * 5. Lời kết |

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)**

|  |
| --- |
| **Mục luc:**   * 1. Regex đại diện cho một ký tự * 2. Ký hiệu đặc biệt cho các từ khóa Regex * 3. Regex A hoặc B * 4. Gom nhóm Regex lại * 5. Regex kiểm tra chiều dài không giới hạn   + Ký tự \*   + Ký tự +   + Ký tự ? * 6. Regex phủ định - NOT   + Các ký tự Regex đặc biệt * 7. Capturing value trong Regular Expression * 8. Greedy trong Regular Expression * 9. Các ví dụ về Regular Expression   + Kiểm tra chuỗi có phải định dạng năm hay không?   + kiểm tra chuỗi có phải là số hay không   + kiểm tra chuỗi phải là dạng ngay/thang/nam hay không   + Kiểm tra chỗi là ABC hoặc CDF * 10. Lời kết |

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, hoặc 0-9 ~ [a-zA-Z0-9]
* \W - Ngược lại với \w (nghĩa là các ký tự không thuộc các khoảng: a-z, A-Z, hoặc 0-9) ~[^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)**

Hàm preg\_match trong PHP

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

Một pattern (biểu thức so mẫu) có dạng:

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

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

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) |
| **$** | Kết thúc dòng(string) |
| **.** | Đại diện cho **một** ký tự bất kỳ |
| **+** | Lặp lại ký tự hay cụm ký tự đứng trước nó **(>=1)** **Ví dụ:** a+ : 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. |
| **\*** | Cũng là lặp lại ký tự hay cụm ký tự đứng trước nó **(>=0)** **Ví dụ:** a\* : 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. |
| **?** | Tồn tại hay không tồn tại ký tự hay cụm ký tự đứng trước nó |
| **\** | Dấu **\**đi kèm với 1 meta symbol sẽ làm mất ý nghĩa của meta symbol đó - trả về symbol bình thường. |

Ví dụ:

$patern = "/.../"; (trong dấu...ta thay các biểu thức sau)

**/^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.

**/^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.

Từ bây giờ thì ta sẽ sử dụng: $pattern = "/^...$/";(^:bắt đầu một dòng, $:kết thúc một dòng)

**/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!

**/ax+/**  :biểu diễn cho dòng: ax, axx, axxxx - nhưng không thể là "a" vì + đại diện cho >=1 ký tự.

**/ay\*/**  :biểu diễn cho dòng: a, ay, ayy, ayyyyyyyyy,....

**/ab?/**  :biểu diễn cho: a, ab (b có thể có hoặc không)

**/a\+b/** :biểu diễn cho dòng: "a+b"

Cách biểu diễn số và chữ cái

**\d : Biểu diễn một con số bất kỳ.**

Ví dụ:

$reg5="/^\d$/"; // chi co 1 chu so

$reg5a="/^\d\d$/"; // chi co 2 chu so

$reg5b = "/^10A\d\d\d\d\d\d$/"; // 10A123456

**\D : Ký tự bất kỳ không phải là chữ số (ngược với \d)**

Ví dụ:

$reg = "/^\D$/"; // bieu dien mot ky tu bat kỳ khong phai la so

$reg ="/^\d\D\d\D$/"; bieu dien mọt chuoi co dang 1a2b. Với a,b là ký tự bất kỳ khong phai la so

**\w - Ký tự từ a-z, A-Z, hoặc 0-9**

Ví dụ:

$reg6 = "/^\w\w\w$/"; // chi gom 3 ky tu từ a->z, A->Z, 0->9. Khong co ky tu dac biet

$reg7 ="/^\w+\@\w+\.\w+$/"; // kiem tra email teo@yahoo.com

**\W - Ngược lại với \w (nghĩa là các ký tự không thuộc các khoảng: a-z, A-Z, hoặc 0-9)**

Nghĩa là chỉ chứa các ký tự đặc biệt

Ví dụ:

$reg = "/^\W$/"; // chua mọt ký tu khong thuoc tù a-z , A-Z, 0-9

**\s+ - Có nghĩa: có >=1 khoảng trắng.**

Ví dụ:

$reg ="/^\w+\s\d+$/"; // bieu dien a 1, g 22, abc 123

Ý nghĩa các dấu ngoặc tròn, ngoặc nhọn, ngoặc vuông

Ngoặc tròn: () : dùng để gom các ký tự thành một nhóm

Ví dụ:

$reg = "/^\d(ab)+$/"; bieu dien 1ab, 1abab, 1ababab. Lặp lại ab vô hạn

Ngoặc nhọn {}: 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

Dấu gạch thẳng đứng để lựa chọn hoặc cái này hoặc cái kia

Ví dụ:

$reg16 = "/^[a-zA-Z0-9]+\.(jpg|png|gif|jpeg|JPG|PNG|GIF|JPEG)$/"; // kiem tra file hinh co duoi khac nhau

Dấu ngoặc vuông để chỉ đoạn ký tự cho phép

+ [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

3. Hàm preg\_replace trong PHP

Hàm này dùng để tìm một string theo quy tắc và thay thế string đó theo mình muốn.

Cú pháp:

preg\_replace(mixed regex, mixed replace, mixed subject);

mixed regex: biểu thức so mẫu

mixed replace: mẫu thay thế.

mixed: chuỗi mẹ

Ví dụ:

preg\_replace("/(\d{3})/","<u>$1</u>","abc 123 efg");

Tìm chuỗi 123 và gạch dưới nó. Kết quả abc 123 efg

Chú ý:

- Cứ 1 nhóm trong dấu ngoặc tròn được đại diện bằng các biến $ là được đánh dấu lần lượt từ trái sang phải, bắt đầu là $1, tiếp theo là $2 , $3...

- Nếu thay thế nó bằng một cái khác thì không có $1, $2, $3...Còn nếu chỉ thay đổi nó thì dùng $1, $2, $3.. như ví dụ ở trên.

Ví dụ

preg\_replace("/^\<div(\s+)(id\=\'teo\')?\>(\s+)?(.\*)(\s+)?\<\/div\>$/","<b><u>$4</u></b>","<div id='teo'> Nguyen Van Teo </div>");

Tìm <div id='teo'> Nguyen Van Teo </div> và thay đổi nó thành <b><u>Nguyen Van Teo</u></b>

Lưu ý:

Biểu thức so mẫu viết liền, không khoảng trắng. Nếu chuỗi mẹ có khoảng trắng thì mình thêm vào biểu thức so mẫu là (\s+). Khi đó nhớ đếm cụm dấu ngoặc để xác định biến $ cho chính xác.

4. Hàm preg\_split trong PHP

Hàm này có tác dụng cắt chuỗi giống như hàm explore, nhưng hàm explore phải kiếm đúng theo ký tự, còn hàm này kiếm theo quy tắc

Cú pháp:

preg\_split(string pattern, string subject);

string pattern: biểu thức so mẫu

string subject: chuỗi mẹ

Chú ý: hàm này trả về một mảng

Ví dụ:

$macth = preg\_split("/\#/","abc#123#efg");

print\_r($macth);