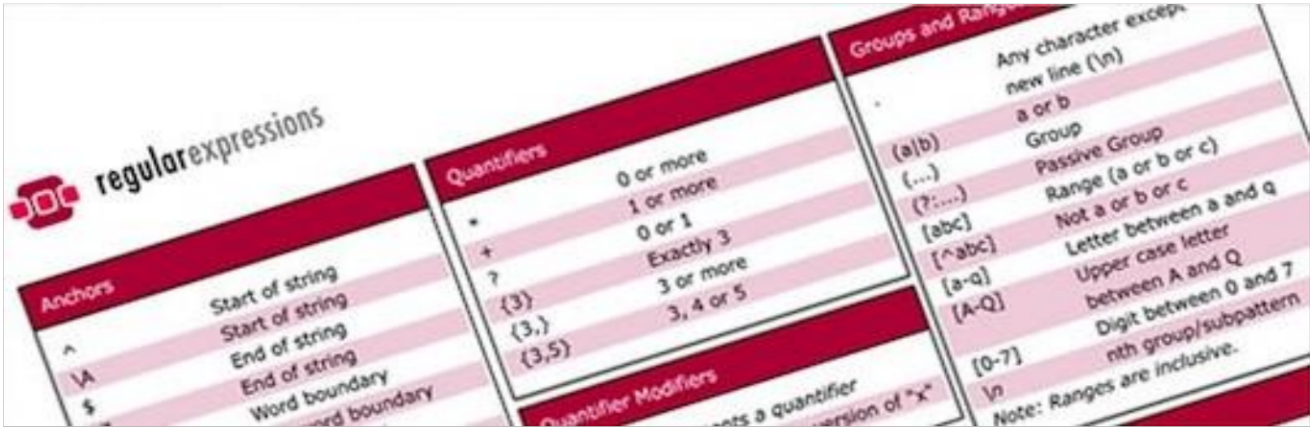
PHP (регулярное выражение) - что это такое? Примеры и проверка регулярных выражений

При работе с текстами в любом современном языке программирования разработчики постоянно встречаются с задачами проверки введенных данных на соответствие нужному шаблону, поиска и замены тестовых фрагментов и прочими типовыми операциями по обработке символьной информации. Разработка собственных алгоритмов проверки приводит к потере времени, несовместимости программного кода и сложности в его развитии и модернизации.

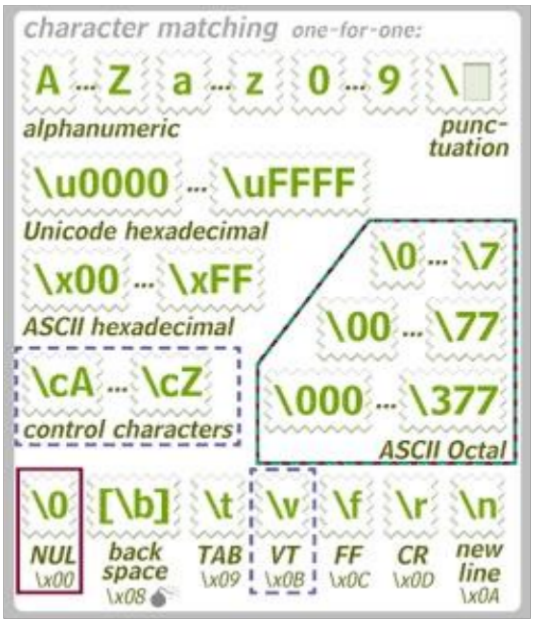
Бурное развитие Интернета и языков WEB-разработки потребовало разработки универсальных и компактных средств обработки текстовой информации при минимальном количестве требуемого для этого кода. Не является исключением и популярный среди начинающих и профессиональных разработчиков язык PHP. Регулярное выражение как язык текстовых шаблонов позволяет упростить задачи обработки текста и уменьшить программный код на десятки и сотни срок. Многие задачи вообще невозможно решить без их использования.



Регулярные выражения в PHP

Язык PHP содержит три механизма работы с регулярными выражениями - «ereg», «mb_ereg» и «preg». Наиболее распространенным является интерфейс «preg», функции которого обеспечивают доступ к библиотеке поддержки регулярных выражений *PCRE*, изначально разработанной для языка Perl, которая входит в комплект PHP. Preg-функции, ищут в заданной текстовой строке совпадения, согласно определенному шаблону на языке регулярных выражений.

Основы синтаксиса



В рамках короткой статьи невозможно подробно описать весь синтаксис регулярных выражений, для этого существует специальная литература. Приведем только основные элементы для показа широких возможностей для разработчика и понимания примеров кода.

В php регулярное выражение формально определяется очень сложно, и поэтому упростим описание. Регулярное выражение представляет из себя текстовую строку. Она состоит из выделенного разделителем шаблона и модификатора, указывающего каким образом его обрабатывать. Возможно включение в шаблоны различных альтернатив и повторений.

Например, в выражении */\d{3}-\d{2}-\d{2}/m* разделителем будет «/», далее идет шаблон, а символ «m» будет модификатором.

Вся мощь регулярных выражений кодируется с помощью метасимволов. Основным метасимволов языка является обратный слэш «\». Он меняет тип следующего за ним символа на противоположный т.е. обычный символ превращается в метасимвол и наоборот. Другим важным метасимволом является прямая черта « | » задающий альтернативные варианты шаблона. Еще примеры метасимволов:

^	Начало объекта или строки
(	Начало подшаблона
)	Окончание подшаблона
{	Начало квантификатора
}	Конец квантификатора
\d	десятичная цифра от 0 до 9
\D	любой символ, не являющийся цифрой
\s	пустой символ, пробел, табуляция
\w	словарный символ

PHP обрабатывая регулярные выражения, пробел рассматривает как отдельный значимый символ, поэтому выражения АБВГДЕ и АБВ ГДЕ являются разными.

Подшаблоны

В PHP регулярные подшаблоны выделяются круглыми скобками и иногда называются «подвыражениями» и выполняют следующие функции:

1. *Выделение альтернатив.* Например, шаблон *жар(кое|птица|)* совпадет со словами «жар», «жарптица» и «жаркое». А без скобок это будет только пустая строка, «птица» и «жаркое».

2. *«Захватывающий» подшаблон.* Это означает, что если в шаблоне совпала подстрока, то в качестве результата возвращаются все совпадения. Для наглядности приведем пример. Дано следующее регулярное выражение:

```
победитель получает((золотую|позолоченный)(медаль|кубок))
```

и строка для поиска совпадений: «победитель получает золотую медаль». Результатом кроме исходной фразы в результатах поиска будут выданы: «золотую медаль», «медаль», «золотую».

Операторы повторений (квадрификаторы)

При составлении регулярных выражений очень часто необходимо анализировать повторения чисел и символов. Это не является проблемой, если повторений не очень много, но что делать, когда мы не знаем их точного числа? В таком случае необходимо использовать специальные метасимволы.

Для описания повторений применяются квадрификаторы – метасимволы для задания количества. Квадрификаторы бывают двух типов:

- Общие, заключенные в скобки
- Сокращенные.

Общий квантификатор задет минимальное и максимальное количество разрешенных повторений элемента в виде двух чисел в фигурных скобках, например, так x{2,5}. Если максимальное количество повторений неизвестно второй аргумент не указывается - x{2,}.

Сокращенные квантификаторы представляют собой символы для наиболее распространенных повторений для избегания лишней перегрузки синтаксиса. обычно используются три сокращения:

* - ноль и больше повторений, что эквивалентно {0,}
+ - одно и более повторения, т.е. {1,}
? – ноль или только одно повторение - {0,1}

Примеры регулярных выражений

Изучая регулярные выражения, примеры являются лучшим учебником для начинающих. Мы приведем несколько, показывающих их широкие возможности при минимуме усилий. Все программные коды полностью совместимы с версиями PHP 4.x и выше. Для полного понимания синтаксиса и использования всех возможностей языка рекомендуем книгу *Дж. Фридла «Регулярные выражения»*, где полностью рассматривается синтаксис, и имеются примеры регулярных выражений не только на PHP, а и для языков Python, Perl, MySQL, Java, Ruby и C#.

Проверка корректности адреса E-mail

Задача

Существует Интернет-страница на которой у посетителя запрашивается адрес email. Регулярное выражение должно проверять правильность полученного адреса перед отправкой сообщений. Проверка не дает гарантии, что указанный почтовый ящик реально существует и принимает письма. Но отсеять заведомо неправильные адреса она может.

Решение

Как и в любом языке программирования на ***PHP регулярные выражения e-mail***-проверки адреса могут быть реализованы разными способами, и примеры в этой статье не являются окончательным и единственным вариантом. Поэтому в каждом случае мы будем приводить перечень требований, которые нужно учесть при программировании, а конкретная реализация полностью зависит от разработчика.

Итак, выражение, проверяющее правильность E-mail должно проверять следующие условия:

1. Наличие в исходной строке символа @ и отсутствие пробелов.

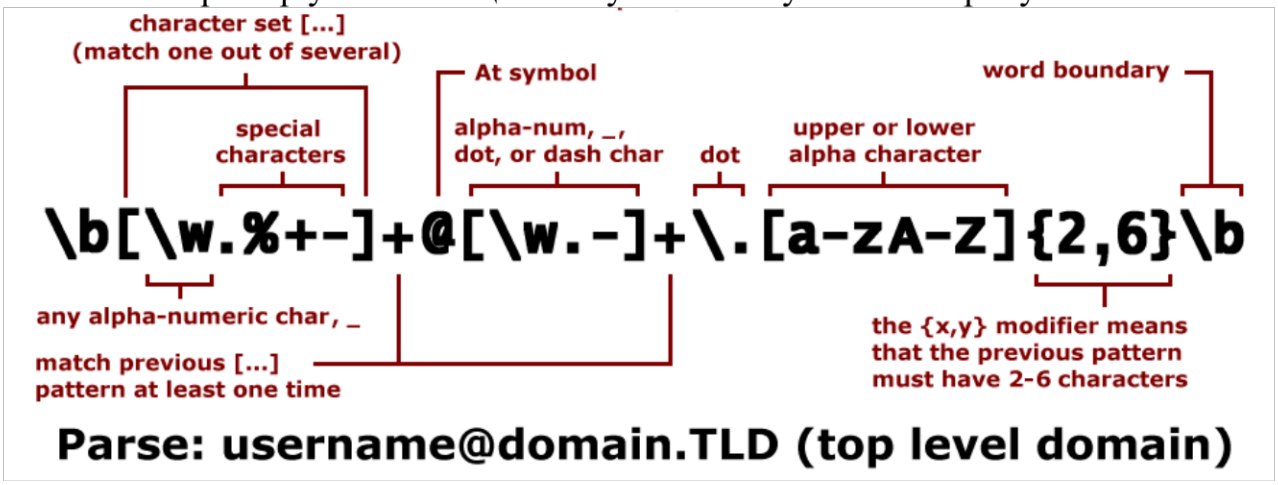
2. *Доменная часть* адреса, за символом @, содержит только допустимые символы для доменных имен. То же относится и к *имени пользователя*.

3. При проверке имени пользователя необходимо определить наличие специальных символов, таких как апостроф или вертикальная черта. Такие символы относятся к потенциально опасным и могут содержаться в таких видах нападений, как SQL-инъекции. Избегайте таких адресов.

4. Имена пользователя допускают наличие только одной точки, которая не может быть первым или последним символом в строке.

5. Доменное имя должно быть не меньше двух и не более шести символов.

Пример учитывающий все указанные условия на рисунке:



Проверка правильности адресов URL

Задача

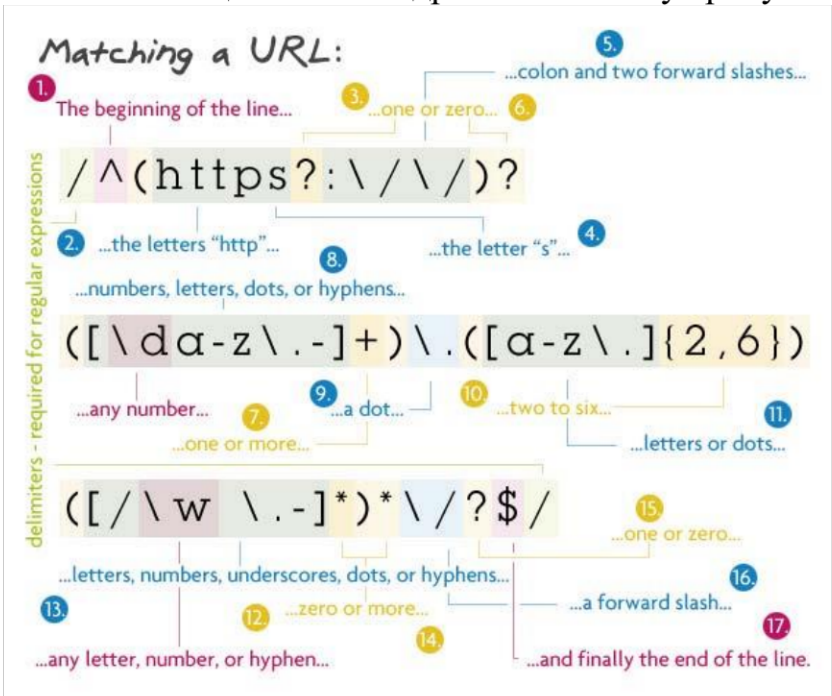
Проверить, является ли заданная текстовая строка допустимым адресом URL. Еще раз отметим, что это только один из вариантов решения.

Решение

Итоговый вариант ***регулярного выражения URL***-проверки выглядит следующим образом:

```
/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/
```

Теперь разберем его составляющие более подробно используя рисунок:



п 1.	Перед адресом URL не должно быть никаких символов
п 2.	Проверяем наличие обязательного префикса «http»
п 3.	Не должно быть символов
п 4.	Если присутствует «s» то URL указывает на защищенное соединение «https»
п.5.	Обязательный фрагмент «//»
п.6.	Нет символов
п. 7-9.	Проверка правильности домена первого уровня и наличие точки
п. 10-13	Контроль правильности написания домена второго уровня и точки
п. 14-17	Файловая структура URL — набор цифр, букв, подчёркиваний, дефисов, точек и слэш в конце

Проверяем номера кредитных карт

Задача

Необходимо реализовать проверку правильности введенного номера пластиковой карты наиболее распространенных платежных систем. Кроме улучшения работы службы поддержки онлайн-проверка номера создает удобство для клиента. Рассмотрен вариант только для карт Visa и MasterCard.

Решение

При создании выражения необходимо учитывать возможное наличие во введенном номере пробелов. Цифры номера на карте разделены на группы для упрощения чтения и диктовки номера. Поэтому вполне естественно, что человек может попытаться ввести номер,таким образом т.е. используя пробелы.

Написать универсальное выражение, учитывающее возможные пробелы и дефисы сложнее чем просто отбросить все символы кроме цифр. Поэтому в выражении рекомендуется использовать метасимвол /D, который удаляет все символы, кроме цифр.

Теперь можно переходить непосредственно к проверке номера. Все компании, выпускающие кредитные карты используют уникальный формат номера. В примере это используется, и клиенту нет необходимости вводить наименование компании – она определяется по номеру. Карты Visa всегда начинаются с 4 и имеют длину номера 13 или 16 цифр, MasterCard начинается в диапазоне 51-55 с длиной номера 16. В итоге получаем такое выражение:

```
^(?:
(?<visa>4[0-9]{12}(?:[0-9]{3})?)|
(?<mastercard>5[1-5][0-9]{14}\d)|
)$
```

Перед обработкой заказа можно провести дополнительную проверку последней цифры номера, которая вычисляется по алгоритму Луна.

Проверка телефонных номеров

Задача

Проверка корректности введенного телефонного номера

Решение

Количество цифр, в стационарных и мобильных телефонных номерах значительно различаются в зависимости от страны, поэтому универсально проверить, используя ***регулярные выражения, номер телефона*** на правильность невозможно. Но международные номера имеют строгий формат и отлично подходят для проверки по шаблону. Тем более, что все больше национальных телефонных операторов стараются соответствовать единому стандарту. Структура номера следующая:

+CCC.NNNNNNNNNxEEEE

где,
C – это код страны, состоящий от 1 до 3 цифр,
N – номер длиной до 14 цифри

E – необязательное расширение.

Плюс является обязательным элементом, а знак x присутствует только при необходимости расширения.

В результате имеем следующее выражение:

```
^\+[0-9]{1,3}\.[0-9]{4,14}(?:x.+)?$
```

Числа в диапазоне

Задача

Необходимо обеспечить совпадение целого числа из определенного диапазона. Дополнительно необходимо, чтобы находили ***регулярные выражения только цифры*** из диапазона значений.

Решение

Приведем несколько выражений для нескольких наиболее распространенных случаев:

Определяем час от 1 до 24	^(1[0-2]|[1-9])$
День внутри месяца 1-31	^(3[01]|[12][0-9]|[1-9])$
Секунда или минута 0-59	^[1-5]?[0-9]$
Число от 1 до 100	^(100|[1-9]?[0-9])$
День года 1-366	^(36[0-6]|3[0-5][0-9]|[12][0-9]{2}|[1-9][0-9]?)$

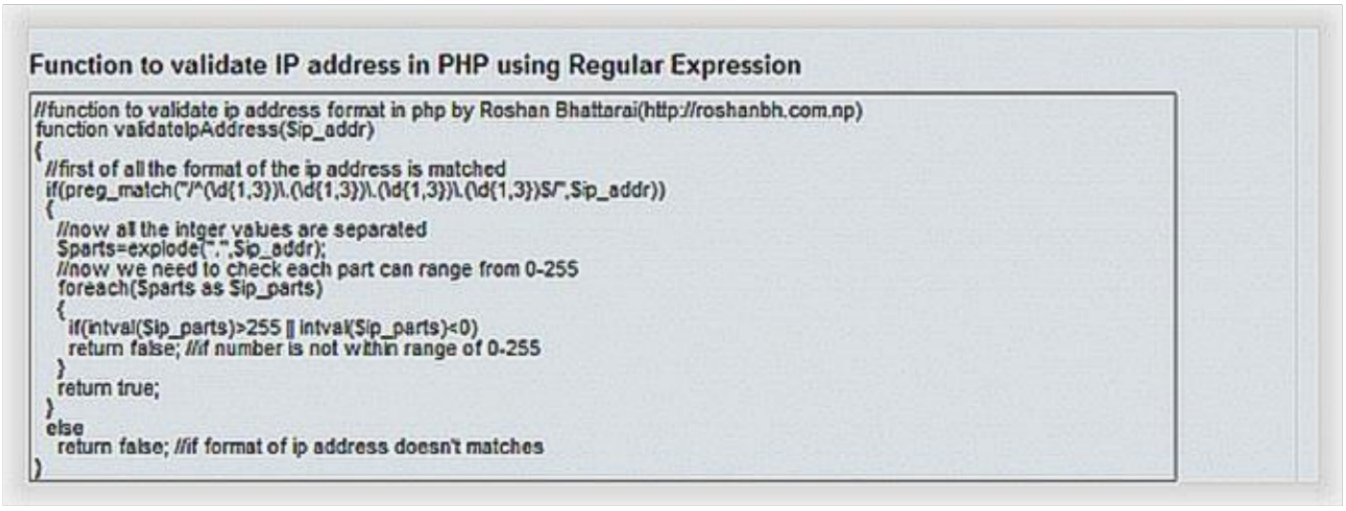
Поиск IP-адреса

Задача

Необходимо определить является ли заданная строка допустимым IP-адресом в формате IPv4 в диапазоне от 000.000.000.000-255.255.255.255.

Решение

Как и в любой задаче на языке ***PHP регулярное выражение*** имеет множество варинтов. Например, такое:



Онлайн-проверка выражений.

Проверка регулярных выражений на правильность для начинающих программистов может быть затруднительной из-за сложности синтаксиса, отличающегося от «обычных» языков программирования. Для решения данной проблемы в сети Интернет существует множество онлайн-тестеров выражений, позволяющих легко проверить правильность созданного шаблона на реальном тексте. Программист вводит выражение и данные для проверки и мгновенно видит результат обработки.

На сайтах обычно присутствует справочный раздел, где подробно описываются ***регулярные выражения, примеры*** и отличия реализации для наиболее распространенных языков программирования.

Но полностью доверять результатам онлайн-сервисов не рекомендуется всем разработчикам на ***PHP. Регулярное выражение***, написанное и проверенное лично повысит квалификацию и гарантирует отсутствие ошибок.