*Лекция 13: Регулярные выражения*

### **Понятие регулярного выражения**

***Регулярное выражение*** (*regular expression*, сокращенно *РВ* ) – это технология, которая позволяет задать шаблон и осуществить поиск данных, соответствующих этому шаблону, в заданном тексте, представленном в виде строки.

Кроме того, с помощью *регулярных выражений* можно изменить и удалить данные, разбить строку по шаблону на подстроки и многое другое.

Одно из распространенных применений *РВ* – это проверка строки на соответствие каким-либо правилам. Например, следующее *РВ*предназначено для проверки того, что строка содержит корректный e-mail–*адрес*:

/^\w+([\.\w]+)\*\w@\w((\.\w)\*\w+)\*\.\w{2,3}$/

Выглядит, конечно, жутко, но зато работает. И если уметь пользоваться этим механизмом виртуозно, то жить становится легче.

Вернемся к нашему определению *РВ*. В нем несколько раз повторяется термин "*шаблон*". Что это такое? В принципе, интуитивно понятно, но попробуем все же пояснить.

Давайте подумаем, что представляет собой корректный e-mail–*адрес*. Это набор букв, цифр и символов подчеркивания, после которых идет *специальный символ* "собака" @, затем еще один такой же набор, содержащий имя сервера, точку ( .) и две или три буквы, указывающие на зону домена, к которой принадлежит *почтовый ящик* (ru, *com*, *org* и т.д.). Приведенное выше *РВ* формализует данное описание на языке, понятном компьютеру. И описывает не какой-то конкретный электронный *адрес*, а все возможные корректные электронные адреса. Таким образом, производится формальное задание *множества* правильных e-mail'ов с помощью шаблона *регулярного выражения*. Другие примеры шаблонов – это шаблоны MS *Word* и *html-формы*.

Механизм *регулярных выражений* задает правила построения шаблонов и осуществляет *поиск* данных по этому шаблону в указанной строке.

В дальнейшем изложении термины *РВ* и "*шаблон*" часто будут использоваться как синонимы, но важно понимать, что это не совсем одно и то же. *Шаблон* задает какой-то *тип данных*, а *РВ* – это механизм, который производит *поиск* и включает в себя *шаблон* и опции поиска, а также задает язык написания шаблонов.

### **Регулярные выражения в PHP**

*Регулярные выражения* пришли из *UNIX* и Perl. Как упоминалось выше, с помощью *регулярных выражений* можно искать и изменять текст, разбивать строку на подстроки и т.д. В *PHP* существуют такие удобные и мощные средства работы со строками, как *explode*(*разбиение* строки на подстроки), *strstr* (нахождение подстроки), *str\_replace* (замена всех вхождений подстроки). Возникает вопрос – зачем придумывать что-то еще?

Основное преимущество *РВ* заключается в том, что они позволяют организовать более гибкий *поиск*, т.е. найти то, о чем нет точного знания, но есть примерное *представление*. Например, нужно найти все семизначные номера телефонов, встречающиеся в тексте. Мы не ищем какой-то заранее известный нам номер телефона, мы знаем только, что искомый номер состоит из семи цифр. Для этого можно воспользоваться следующим *РВ*:

/\d{3}-\d{2}-\d{2}/m

В *PHP* существует два различных механизма для обработки *регулярных выражений*: *POSIX-совместимые* и *Perl-совместимые*(сокращенно *PCRE* ). Их *синтаксис* во многом похож, однако *Perl-совместимые регулярные выражения* более мощные и к тому же работают намного быстрее. Начиная с версии *PHP* 4.2.0, *PCRE* входят в набор базовых модулей и подключены по умолчанию. *POSIX-совместимые РВ* включены по умолчанию только в версию *PHP* для *Windows*.

Основные функции для работы с *Perl-совместимыми регулярными выражениями*: *preg\_match*(pattern, string, result[, flags])и *preg\_match\_all*(pattern, string, result, [flags]), где:

pattern – *шаблон регулярного выражения* ;

string – строка, в которой производится *поиск*;

result – содержит *массив* результатов (нулевой элемент массива содержит соответствие всему шаблону, первый – первому "захваченному" *подшаблону* и т.д.);

flags – необязательный *параметр*, определяющий то, как упорядочены результаты поиска.

Эти функции осуществляют *поиск* по шаблону и возвращают информацию о том, сколько раз произошло совпадение. Для *preg\_match()* это 0 (нет совпадений) или 1, поскольку *поиск* прекращается, как только найдено первое совпадение. *Функция**preg\_match\_all()* производит *поиск* до конца строки и поэтому находит все совпадения. Все точные совпадения содержатся в первом элементе массива result у каждой из этих функций (для *preg\_match\_all()* этот элемент – тоже *массив*).

Про "*захват*" элементов будет рассказано в разделе, посвященном *подвыражениям*.

Аналогом *preg\_match* является *булева функция* *POSIX*-расширения *ereg*(string pattern, string string [, array regs])

*Функция* *ereg()* возвращает TRUE, если совпадение найдено, и FALSE – в противном случае.

Приводимые далее примеры можно тестировать на перечисленных функциях. Например, так:

<?

//строка, в которой нужно что-то найти

$str = "Мой телефонный номер: ".

"33-22-44. Номер моего редактора: ".

"222-44-55 и 323-22-33";

//шаблон, по которому искать.

//Задает поиск семизначных номеров.

$pattern = "/\d{3}-\d{2}-\d{2}/m";

//функция, осуществляющая поиск

$num\_match = preg\_match\_all ($pattern,

$str, $result);

//вывод результатов поиска

for ($i=0;$i<$num\_match;$i++)

echo "Совпадение $i: ".

$result[0][$i]."<br>";

?>

### **Синтаксис регулярных выражений**

Строгое *определение* *регулярного выражения* выглядит довольно громоздко. Начнем с неформального описания.

*Регулярное выражение* представляет собой строку. Эта строка состоит из собственно *регулярного выражения* (шаблона), выделенного с помощью специального символа разделителя (это могут быть символы " / " , " | ", " { ", " !" и т.п ) и *модификатора*, влияющего на способ обработки *РВ*.

В дальнейшем это описание будет расширено.

Например, в *регулярном выражении* /\d{3}-\d{2}-\d{2}/m символ " / "является разделителем, \d{3}-\d{2}-\d{2} – непосредственно *регулярное выражение* (*шаблон*), а m – *модификатор*.

Мощь *регулярных выражений* порождена в основе своей их способностью включать в *шаблон* альтернативы и повторения. Они кодируются в шаблоне с помощью *метасимволов*. *Метасимвол* отличается от любого другого символа тем, что имеет специальное *значение*.

Одним из основных *метасимволов* является *обратный слэш* " \ ". Он меняет тип символа, следующего за ним, на противоположный, т.е. если это был обычный символ, то он МОЖЕТ превратиться в *метасимвол*, если это был *метасимвол*, то он теряет свое специальное *значение* и становится обычным символом (это нужно для того, чтобы вставлять в текст специальные символы как обычные). Например, символ d в обычном режиме не имеет никаких специальных значений, но \d есть *метасимвол*, означающий "любая цифра". Символ " ." в обычном режиме означает "любой единичный символ", а " \." означает просто точку.

Другое назначение *обратного слэша* – *кодирование* непечатных символов, таких как :

\n – cимвол перевода строки;

\e – символ *escape*;

\t – cимвол табуляции;

\xhh – символ в шестнадцатеричном коде, например \x41 есть буква A и т.д.

Еще одно назначение *обратного слэша* – обозначение генерируемых символьных типов, таких как:

\d – любая десятичная цифра ( 0-9 );

\D – любой символ, не являющийся десятичной цифрой;

\s – любой *пустой символ* (*пробел* или табуляция);

\S – любой символ, не являющийся пустым;

\w – символ, используемый для написания Perl-слов (это буквы, цифры и символ подчеркивания), так называемый "словарный символ";

\W – несловарный символ (все символы, кроме определяемых \w ).

Что имеется в виду под "символьным типом"? Просто каждый *метасимвол* принимает *значение* (одно) из класса возможных значений, заданных автоматически или вручную. *Символьные типы*, задаваемые пользователем, описываются с помощью квадратных скобок (подробнее об этом позже). Выше приведены *символьные типы*, *диапазон* значений которых заранее определен языком программирования.

Пример использования приведенных выше *метасимволов*:

/\d\d\d plus \d is \w\w\w/

Это *РВ* означает: трехзначное число, за которым следует *подстрока* plus, любая цифра, затем is и *слово* из трех словарных символов. В частности, данному *РВ* удовлетворяют строки: " 123 plus 3 is sum ", " 213 plus 4 is 217 ".

Вообще различают два *множества* *метасимволов*: те, что распознаются в любом месте шаблона, за исключением внутренности квадратных скобок, и те, что распознаются внутри квадратных скобок.

Квадратные скобки [ ] применяются для *описания подмножеств* и внутри *регулярного выражения* рассматриваются как один символ, который может принимать значения, перечисленные внутри этих скобок. Однако если первым символом внутри скобок является ^, то значением символьного класса могут быть только символы, НЕ перечисленные внутри скобок.

**Примеры**:

1. Символьный класс [абвгд] задает один из символов а, б, в, г, д, а класс [^абвгд] задает любой символ, кроме а, б, в, г, д.
2. Если написать [2бул]ки], то это выражение интерпретируется как один из символов 2, б, у, л, за которым следует строка ки], потому что первая встретившаяся закрывающая квадратная скобка (разбор происходит слева направо) заканчивает определение символьного класса. То есть это *РВ* совпадет с одной из строк 2ки], бки], уки] или лки].
3. С помощью *РВ* [0-9А-Яа-я] можно задать любую букву или цифру.

*Метасимволы*, распознаваемые вне квадратных скобок, можно разделить на группы следующим образом: определяющие положение искомого текста в строке, связанные с *подвыражениями*, ограничивающие символьный *класс*, *квантификаторы* и *перечисление*альтернатив.

**Примеры (^ и $)**

1. Пусть дан такой текст, записанный в виде строки:
2. $str = "11 aaa bbb ".
3. "ccc 22 ddd ".

"eee ggg 33";

Таблица 13.1. *Метасимволы, распознаваемые ВНУТРИ квадратных скобок*

|  |  |
| --- | --- |
| ***Метасимвол*** | **Значение** |
| \ | Переходный символ со множеством назначений |
| ^ | Отрицание класса, но только если это первый символ (например, " ^\d " задает все, кроме цифр) |
| - | Задает *диапазон символов* (например, " 0-9 " задает все цифры, " A-Z " – все латинские буквы) |
| ] | Вычисляет символьный класс |

*Регулярное выражение* /\d\d/m может быть сопоставлено следующим подстрокам: 11, 22, 33. Если в начале *РВ* стоит ^, то совпадения ищутся в начале строки, поэтому выражение /^\d\d/m найдет только 11.

Когда в конце *РВ* стоит знак доллара $, поиск производится в конце строки, поэтому выражение /\d\d$/m найдет только 33.

Шаблону же /^\d\d\d$/ будет удовлетворять строка, целиком состоящая из трехзначного числа (т.е. она и начинается и заканчивается этим числом).

1. Найдем все html-теги, расположенные в начале каждой строки файла 1.htm.
2. <?
3. //считываем файл в строку
4. $str = file\_get\_contents('1.htm');
5. $pattern = "!^<[^/]+>!mU";
6. // осуществляем поиск
7. $n = preg\_match\_all ($pattern,
8. $str, $res);
9. // выводим результаты
10. for ($i=0;$i<$n;$i++)
11. echo htmlspecialchars($res[0][$i]).
12. "<br>";

?>

Таблица 13.2. *Метасимволы*, *распознаваемые ВНЕ квадратных скобок*

|  |  |
| --- | --- |
| ***Метасимвол*** | **Значение** |
| \ | Переходный символ со множеством назначений |
| ^ | Объявляет начало объекта (или строки в многострочном режиме). То есть этот символ определяет, что искомый текст должен находиться в начале строки. Альтернатива: " \A " |
| $ | Объявляет конец объекта (или строки в многострочном режиме). То есть этот символ определяет, что искомый текст должен находиться в конце строки. Альтернативы: " \Z ", " \z " |
| . | Совпадает с любым символом, кроме символа перевода строки (по умолчанию) |
| [ | Начинает определение символьного класса |
| ] | Заканчивает определение символьного класса |
| | | Разделяет перечисление *альтернативных вариантов* |
| ( | Начинает *подшаблон* (регулярное *подвыражение* ) |
| ) | Заканчивает *подшаблон* |
| ? | Расширяет значение " ( ", *квантификаторов* 0 или 1, и *квантификатор* минимизации |
| \* | 0 или больше повторений ( *квантификатор* ) |
| + | 1 или больше повторений ( *квантификатор* ) |
| { | Начинает минимальный/максимальный *квантификатор* |
| } | Заканчивает минимальный/максимальный *квантификатор* |

Шаблон ограничен восклицательными знаками. Первая " ^ " значит, что мы ищем совпадения в начале строк, потом идет символ " < " – его и ищем в строке, после него должно идти все, что угодно, кроме *слэша* (конструкция " [^/] " ), " + " говорит, что стоящий перед ним символ повторяется один и более раз и заканчивается все это символом " > ". Таким образом, выделяются все теги в начале строк.

1. Мы хотим убедиться, что имя автора было записано правильно (сначала фамилия с большой буквы, потом инициалы через точку) и находится в конце строки.
2. <?
3. //считываем файл в строку
4. $str = file\_get\_contents('1.htm');
5. $pattern = "!\s[А-Я][А-Яа-я]+".
6. "\s([А-Я]\.\s\*)([А-Я]\.\s\*)$!m";
7. // шаблон ограничен восклицатель-
8. // ными знаками, m – модификатор,
9. // включающий многострочный режим.
10. // Первый \s означает, что перед
11. // фамилией должен идти пустой
12. // символ (например, пробел).
13. // [А-Яа-я] задает одну из букв
14. // алфавита в любом регистре, а в
15. // комбинации со знаком плюс
16. // определяет, что эта буква
17. // повторяется один и более
18. // раз. Следующий \s означает, что
19. // между фамилией и инициалами
20. // должен быть пробел.
21. // Далее идет подвыражение,
22. // определяющее инициалы.
23. // Это буква от А до Я, после
24. // которой стоит точка ('\.')
25. // Экранируем точку, чтобы
26. // избавиться от ее специального
27. // значения. После буквы с точкой
28. // может идти или не идти пробел
29. // или несколько. Вся конструкция
30. // повторяется минимум два раза.
31. // Последний символ $ означает,
32. // что фамилия с инициалами
33. // должны находиться в конце
34. // строки.
35. // осуществляем поиск
36. $n = preg\_match\_all ($pattern,
37. $str, $res);
38. // выводим результаты
39. for ($i=0;$i<$n;$i++)
40. echo htmlspecialchars($res[0][$i]).
41. "<br>";

?>

**Примеры ( | и .)**

1. Пусть имеется некий текст. Нам нужно найти всех упомянутых в нем людей со званиями.

<?

$str = "Доцент Смирнов совершил ".

"открытие. Его учителем была ".

"профессор Иванова. ".

"Этим открытием Смирнов ".

"завоевал себе степень ".

"доктора. Раньше он был ".

"только кандидат.";

$pattern = "/(профессор|доцент)".

"\s[А-Я][А-Яа-я]+(\s|\.)/i";

// осуществляем поиск

$n = preg\_match\_all ($pattern, $str,

$res);

// выводим результаты

for ($i=0;$i<$n;$i++)

echo htmlspecialchars($res[0][$i]).

"<br>";

?>

*Метасимвол* прямая черта " | " позволяет задавать *альтернативные варианты*. В примере мы хотели найти всех профессоров или доцентов. Для этого было создано *подвыражение* " (профессор|доцент) ". После звания через *пробел* фамилия человека, которому оно принадлежит, – для этого существует комбинация " \s[А-Я][А-Яа-я]+ ". После фамилии идет либо опять *пробел*, либо точка, если это конец предложения. Получаем опять два *альтернативных варианта*: " (\s|\.) " (здесь точка экранируется *обратным слэшем*, чтобы она понималась как обычная точка, без специального значения).

#### **Подвыражения (подшаблоны)**

В нескольких примерах мы уже использовали *подвыражения*. Настало время разобраться, что же это такое и какими свойствами они обладают

В *РВ* *подшаблоны* выделяют, заключая в круглые скобки. Для их обозначения кроме термина " *подшаблон* " также используют термин " *подвыражение* ". *Подшаблоны* могут быть вложенными. Выделение части *регулярного выражения* в виде регулярного *подвыражения*делает следующее.

1. Локализует множество альтернатив.

Например, шаблон

жар(кое|птица)

совпадает с одним из слов " жаркое ", " жарптица " и " жар ". Тогда как без скобок это было бы " жаркое ", " птица " и пустая строка.

1. Устанавливает *подшаблон* как "захватывающий" *подшаблон*. Это значит, что, когда какая-то подстрока в тексте совпала с шаблоном, все подстроки, которые совпали с *подшаблонами* этого *РВ*, тоже возвращаются в качестве результата. Скобки, обозначающие начало *подшаблона*, пересчитываются слева направо (начиная с 1) для того, чтобы узнать, сколько *подшаблонов*нужно захватить.

Например, имеется такой шаблон:

победитель получит

((золотую|позолоченный)

(медаль|кубок))

и строка, в которой ищутся совпадения с этим шаблоном: " победитель получит золотую медаль ". Тогда кроме этой фразы будут еще захвачены и выданы как результаты поиска следующие совпадения в *подвыражениях*: " золотую медаль ", " золотую", " медаль ", пронумерованные 1, 2, 3 соответственно.

Однако это не всегда удобно. Для того чтобы избавиться от "захватывающего" эффекта *подвыражения*, после открывающей скобки пишут " ?:". Тогда это *подвыражение* в результат поиска не включается и при нумерации остальных *подшаблонов* с "захватывающим" эффектом не учитывается.

победитель получит

((?:золотую|позолоченный)

(медаль|кубок))

Пример 13.1. Применение ?:

Тогда в условиях предыдущего примера получим искомую строку " победитель получит золотую медаль " и строки " золотую медаль ", " медаль ", пронумерованные 1 и 2 соответственно.

Если в html-файле название находится после <body> и отделено от него только пробелами или переводами строк, заключено в тег <h1> и после него тоже может идти сколько-то пробелов и переводов строк, то его можно найти с помощью следующего скрипта:

<?

//считываем файл в строку

$str = file\_get\_contents('1.htm');

$pattern = "/<body.\*?>[\n\s]\*<h1>".

"(.\*?)<\/h1>[\n\s]\*/m";

// осуществляем поиск

$n = preg\_match\_all ($pattern, $str, $res);

echo $res[1][0]; // выводим заголовок

?>

Пример 13.2. Поиск названия в html-файле

Заметим, что здесь выводится первое захваченное *подвыражение*, поскольку нам интересно только само название, а не все *РВ*. Так как в этом *РВ* есть только одно *подвыражение*, то его значение содержится в нулевом элементе первого массива результатов.

#### **Повторения (квантификаторы)**

В предыдущих примерах мы часто писали комбинации типа \d\d. Это значит, что цифра должна повторяться два раза. А что же делать, если повторений очень много или мы не знаем, сколько именно? Оказывается, нужно использовать специальные *метасимволы*.

Повторения описываются с помощью так называемых *квантификаторов* ( *метасимволов*, задающих количественные отношения). Существует два типа *квантификаторов*: общие (задаются с помощью фигурных скобок) и сокращенные (это исторически сложившиеся сокращения наиболее распространенных *квантификаторов* ).

*Квантификаторы* могут следовать за любым из перечисленных элементов:

* одиночный символ (возможно, в комбинации с *обратным слэшем* );
* *метасимвол* "точка";
* символьный класс;
* *обратная ссылка* (о них расскажем позднее);
* *подшаблон*.

*Общие квантификаторы* задают минимальное и максимальное число дозволенных повторений элемента; эти два числа, разделенные запятой, заключаются в фигурные скобки. Числа не должны превышать 65 536 и первое число должно быть меньше или равно второму. Например,

x{1,3}

говорит о том, что символ "x" должен повторяться минимум один, а максимум три раза. Соответственно этому шаблону удовлетворяют строки: x, xx, xxx.

Если второй параметр отсутствует, но запятая есть, то повторений может быть сколько угодно. Таким образом,

[aeuoi]{2,}

значит, что любой из символов " a ", " e ", " u ", " o ", " i " в строке может повторяться два и более раз, а *регулярное выражение*

\d{3}

задает ровно три цифры.

*Сокращенные квантификаторы* задают наиболее используемые количественные отношения (повторения). Они придуманы для удобства, чтобы не перегружать и без того сложные выражения лишним синтаксисом.

Исходя из исторических традиций три наиболее часто встречающихся *квантификатора* имеют следующие обозначения:

\* эквивалентно {0,} – то есть это ноль и более повторений;

+ эквивалентно {1,} – то есть это одно и более повторений;

? эквивалентно {0,1} – то есть это ноль или одно повторение.

Есть еще один важный момент, на который стоит обратить внимание при изучении *квантификаторов*. По умолчанию все *квантификаторы "жадные"*, они стараются захватить как можно больше повторений элемента. То есть если указать, что символ должен повторяться один и более раз (например, с помощью \* ), совпадение произойдет со строкой, содержащей наибольшее число повторений указанного символа. Это может создать проблемы, например, при попытке выделить комментарии в программе на языке Cи или PHP. Комментарии в Cи и PHP записываются между символами /\* и \*/, внутри которых тоже могут встречаться символы \* и /. И попытка выявить Си-комментарии с помощью шаблона

/\\* .\* \\*/

в строке

/\* первый комментарий \*/

не комментарий

/\* второй комментарий \*/

не увенчается успехом из-за *"жадности"* элемента " .\* " (будет найдена также строка "не комментарий").

Для решения этой проблемы нужно написать знак вопроса после *квантификатора*. Тогда он перестанет быть *"жадным"* и попытается захватить как можно меньшее число повторений элемента, к которому он применен ( *квантификатор* применяется к элементу, что стоит перед ним). Так что шаблон

/\\* .\*? \\*/

успешно выделяет Си-комментарии.

В PHP существует *опция PCRE\_UNGREEDY*, которая делает все *квантификаторы* *"не жадными"* по умолчанию и *"жадными"*, если после них идет знак вопроса.

<?

//Рассмотрим html-файл, где имеется

//следующая строка:

$str = "<div id=1>Привет</div> ".

"<p>Текст, не заключенный в тег ".

"div</p><div id=2>Пока</div>";

// Если мы хотим найти текст,

// содержащийся между тегами div,

// естественно написать такой шаблон:

$pattern = "!<div id=1>.\*</div>!si";

// Но этот шаблон слишком "жадный"

// и захватит также и текст,

// заключенный в нашем примере между

// тегами <p>. Чтобы этого избежать,

// нужно написать следующий шаблон,

// отличающийся только наличием знака

// вопроса, который запрещает

// квантификатору быть "жадным".

$pattern1 = "!<div id=1>.\*?</div>!si";

// Запускаем поиск в строке $str

// совпадений с шаблонами

// $pattern и $pattern1

$s = preg\_match\_all ($pattern, $str,

$res);

$js = preg\_match\_all ($pattern1,

$str, $res1);

//выводим результаты поиска

// функция htmlspecialchars позволяет

// выводить html без

// его обработки браузером

echo "Жадный шаблон:".

htmlspecialchars($res[0][0]).

"<br>";

echo "Нежадный шаблон:".

htmlspecialchars($res1[0][0]);

?>

Пример 13.3. Использование "жадных" квантификаторов

Результаты работы скрипта:

"Жадный" шаблон:<div id=1>Привет</div>

<p>Текст,не заключенный в тег div</p>

<div id=2>Пока</div>

"Нежадный" шаблон:<div id=1>Привет</div>

Теперь мы в принципе можем решить задачу выделения содержания из html-файла, если оно заключено в теге <div id=content>. Предлагаем читателям проделать это самостоятельно.

### **Модификаторы PCRE**

Еще один немаловажный элемент *регулярного выражения* – это *список* применяемых к нему *модификаторов*. ***Модификаторы*** – это выдаваемая интерпретатору *регулярных выражений* инструкция по обработке данного выражения. Например, считать, что все символы *регулярного выражения* соответствуют как большим, так и маленьким буквам в строке, где производится поиск. Примеры *модификаторов* приведены в [таблице 13.3](https://www.intuit.ru/studies/courses/42/42/lecture/27199?page=4#table.13.3).

Таблица 13.3. *Наиболее часто используемые модификаторы*

|  |  |
| --- | --- |
| **Обозначение** | **Описание** |
| i ( PCRE\_CASELESS) | Если указан этот *модификатор*, то буквы в шаблоне совпадают с буквами и верхнего, и нижнего регистра в строке |
| m ( PCRE\_MULTILINE) | По умолчанию строка, подающаяся на вход интерпретатору *РВ*, рассматривается как состоящая из одной линии. Этот *модификатор* включает поддержку многострокового режима |
| s ( PCRE\_DOTALL ) | Если установлен этот *модификатор*, то *метасимвол* точка " ." совпадает с любым символом, ВКЛЮЧАЯ символ перевода строки |
| x ( PCRE\_EXTENDED) | Заставляет интерпретатор игнорировать пробелы между символами в шаблоне, за исключением пробелов, *экранированных* *обратным слэшем* или находящихся внутри символьного класса, а также между неэкранированным символом # вне символьного класса и *символом новой строки* |
| U ( PCRE\_UNGREEDY) | Этот *модификатор* инвертирует *"жадность" квантификаторов*, т.е. они становятся *"нежадными"* по умолчанию и *"жадными"* если предшествуют символу " ?" |

### **Регулярные выражения для "продвинутых"**

В последующих разделах обсуждаются более сложные конструкции работы с *регулярными выражениями*, без которых в принципе можно обойтись. Поэтому мы не будем в них особо углубляться, а приведем лишь общие сведения.

#### **Обратная ссылка**

Вне определения символьного класса (это тот, что задается квадратными скобками) комбинация *обратный слэш* и цифра больше нуля (например, \1 ) называется ***обратной ссылкой*** и представляет собой ссылку на захваченное ранее регулярное *подвыражение*. Этих *подвыражений* ровно столько, сколько открывающихся круглых скобок (после которых нет знака вопроса) стоит левее данного элемента.

*Обратная ссылка* совпадает с конкретным выбранным значением *подвыражения*, на которое она ссылается, а не с любым возможным значением этого *подвыражения*. Таким образом, шаблон

(ответствен|надеж)ный проявляет \1ность

совпадет со строками " ответственный проявляет ответственность ", " надежный проявляет надежность " и не совпадет со строкой " ответственный проявляет надежность ".

*Обратные ссылки* могут использоваться внутри *подвыражений*. При первом использовании *подвыражения* ссылка внутри него не срабатывает, но при последующих повторениях *подшаблона* она работает, как описано выше.

#### **Утверждения**

*Утверждение* – это проверка символов, следующих до или после текущего символа. Простейшие *утверждения* закодированы последовательностями \A, \Z, ^, $ и т.д. Более сложные *утверждения*кодируются с помощью *подшаблонов*. Постараемся вкратце описать, как это делается.

Существует два типа *утверждений*: те, что смотрят на текущую позицию в исходной строке ( *"смотрящие вперед"* ), и те, что смотрят на символы перед текущей позицией ( *"смотрящие назад"* ).

*Утверждения*, закодированные *подшаблонами*, сравниваются как обычные *подшаблоны*, за исключением того, что при их обработке не происходит изменения текущей позиции.

*"Смотрящие вперед" утверждения* ищут совпадения в строке за текущей позицией поиска и начинаются с (?= для позитивных *утверждений* и с (?! для негативных. Например,

\w+(?=;)

совпадает со словом, заканчивающимся точкой с запятой (не включая точку с запятой в результат поиска), и

foo(?!bar)

совпадает с любым появлением foo, после которого нет bar. Как все происходит? Берем строку и ищем в ней foo. Как только нашли, заглядываем вперед (текущая позиция при этом не меняется) и смотрим, идет ли далее bar. Если нет, то совпадение с шаблоном найдено, иначе продолжаем поиск.

***Регулярное выражение***

(?!foo)bar

не найдет все вхождения bar, перед которыми нет foo, потому что оно *"смотрит вперед"*, а перед ним никаких символов нет. Поэтому в данном шаблоне ?!foo всегда верно.

*"Смотрящие назад" утверждения* ищут совпадения перед текущей позицией. Позитивные *утверждения* этого типа начинаются с (?<=,негативные – с (?<! . *Смотрящим назад утверждениям* позволено искать только строки фиксированной длины, т.е. в них нельзя использовать *квантификаторы*. Например,

(?<!foo)bar

находит все появления bar, перед которыми нет foo.

В начале лекции мы хотели научиться находить в html-файле упоминание об авторе. Это можно сделать с помощью *"смотрящих назад" утверждений* в *РВ* (хотя можно и проще).

<?

//считываем файл в строку

$str = file\_get\_contents('1.htm');

$pattern = "/(?<=Автор:)\s[А-Я]".

"[а-я]\*\s([А-Я]\.\s\*){1,2}/m";

// осуществляем поиск

$n = preg\_match\_all ($pattern, $str, $res);

// выводим результаты

for ($i=0;$i<$n;$i++)

echo htmlspecialchars($res[0][$i]).

"<br>";

?>

Пример 13.4. Смотрящие назад утверждения

Часть *РВ* после *утверждения* определяет, что мы ищем строку (ФИО), которая начинается с пробела, большой буквы, затем идут маленькие буквы в произвольном количестве, пробел и инициалы через точку. *Утверждение* задает то, что перед данной строкой должно стоять "Автор:".

Дату можно вычислить похожим образом. Оставляем это в качестве упражнения.

#### **Условные подвыражения**

Как в любом языке программирования, в *РВ* существуют условные конструкции. Применяются они к *подвыражениям*. То есть можно заставить процессор *РВ* выбирать *подшаблон* в зависимости от условия или выбирать между двумя *альтернативными шаблонами* в зависимости от результата *утверждения* или от того, совпал ли предыдущий захваченный *подшаблон*. Существуют две формы *условных подвыражений*:

(?(условие)шаблон\_выполняемый\_если\_

условие\_верно)

(?(условие)шаблон\_если\_условие\_верно

|шаблон\_если\_условие\_неверно)

Существует два типа условий. Если текст между круглыми скобками состоит из последовательности цифр, то условие удовлетворяется, если захваченное *подвыражение* с этим номером ранее совпало.

( \( )? [^()]+ (?(1) \) )

Пример 13.5. Условные подвыражения

Первая часть этого *РВ* опционально совпадает с открывающейся круглой скобкой, и если этот символ присутствует, то устанавливает его как первое захваченное *подвыражение*.

Вторая часть совпадает с одним или более символами, не заключенными в круглые скобки.

Третья часть *РВ* – это *условное подвыражение*, которое проверяет, совпало ли первое множество скобок или нет (попалась ли нам в строке открывающая круглая скобка). Если попалась, то есть объект (строка) начинается с символа " ( ", то условие верно и вычисляется условный шаблон, а именно требуется наличие закрывающей круглой скобки. В противном случае *подшаблон* ни с чем не совпадает.

Если условие – не последовательность цифр, то оно должно быть *утверждением*. Это может быть позитивное или негативное *"смотрящее вперед"* или *"смотрящее назад"* *утверждение*.

(?(?=[^a-z]\*[a-z])\d{2}-[a-z]{3}-\d{2}

|\d{2}-\d{2}-\d{2})

Пример 13.6. Условные подвыражения

Условие здесь – позитивное *"смотрящее вперед" утверждение*. Оно совпадает с любой последовательностью не букв, после которых идет буква. Другими словами, оно проверяет присутствие хотя бы одной буквы в строке для поиска. Если буква найдена, то производится сравнение по первому *альтернативному варианту* шаблона (\d{2}-[a-z]{3}-\d{2}), иначе – по второму (\d{2}-\d{2}-\d{2}). Этому шаблону удовлетворяют строки двух типов: dd-*aaa*-dd или dd-dd-dd, где d – любая цифра, a – любая буква.

### **Заключение**

Итак, мы рассмотрели механизм *регулярных выражений*, их *синтаксис* и семантику, показали примеры их использования. Безусловно, эта лекция не охватывает все тонкости данного механизма. О *регулярных выражениях* пишут целые книги! Мы же постарались лишь в общих чертах познакомить читателей с их основами. Вообще механизм *регулярных выражений* присутствует почти во всех языках программирования с небольшими отличиями, но суть остается той же. Так что надеемся, что знания, полученные в процессе чтения этой лекции, помогут при изучении других языков и пригодятся на практике.