RegExp — Lurkmore

В эту статью нужно добавить как можно больше жадных квантификаторов.

Также сюда можно добавить интересные факты, картинки и прочие кошерные вещи.

Я нихуя не понял!

В этой статье слишком много мусора, что затрудняет её понимание.
Данный текст необходимо очистить, ~~либо вообще снести нахуй~~

«Я не являюсь сторонником такого кода, в котором нарисован единственный регэксп в пару-тройку-другую сотен символов длиною, который не в подъем уму уразуметь... »

— *Мицгол*

Регэксп (moon. , евр. рас. : ביטוי רגולרי, он же *RegExp*, *RegEx* и ещё *Регулярка*, а обозначает оно Regular Expression(s)) — фича в языках программирования, позволяющая сократить 9000 строк кода до одной строчки непонятной хуйни. Сей термин восходит корнями к дискретной математике и означает он систему лексического анализа текста для выделения из него составных частей. Чуть менее, чем полностью состоит из символов `.%?*` и других.

Я знаю регулярные выражения!

Суть

Главное в регэкспе — это специальные, или **служебные** символы. Например, известные обычным юзерам wildcards вроде «*.txt» (найти все все файлы с расширением txt) на регэкспе выглядит примерно как «`.*\.txt`». На неокрепший разум познание регекспов действует разрушительно — появляется непреодолимое желание совать регексы **везде**, даже когда можно обойтись более простыми и читаемыми средствами.

Модули, обеспечивающие поддержку функций работы с регулярками, можно найти для любого высокоуровневого неэзотерического языка, поэтому — краткий список языков, в которых **НЕЛЬЗЯ** использовать регексы: Brainfuck, ассемблер и 1C.

Применение регэкспов

«If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. »

— *stackoverflow*

Стандартом де-факто для регулярных выражений является **PCRE** — Perl Compatible Regular Expression. Регулярки данного стандарта понимают фактически все современные языки программирования. Существуют также и другие стандарты, например винрарный POSIX, однако широкого распространения они не получили и используются преимущественно PHP-быдлокодерами[1] или трухардкор-красноглазиками

Пример, который соответствует большинству корректных адресов E-mail, например FuckMyBrain@pechenki.com:

```
^[-.\w]+@(?:[a-z\d]{2,}\.)+[a-z]{2,6}$
```

Того же результата и даже более омниприменимого результата можно добиться и другими, более сложными регэкспами. Самый полный известный регексп, проверяющий корректность e-mail в соответствии с грамматикой RFC822, занимает более страницы машинописного текста. И таки он тоже неидеален.

Приятная особенность всех регулярок заключается в том, что ни один даже самый профессиональный девелопер не может за сколь либо приемлемый срок сказать, чему соответствует та или иная регулярка, глядя на неё. Более того, даже если её придумал сам программист, то если он не прокомментировал её, уже через месяц или неделю она будет для него столь же загадочна и непонятна, как и любая другая. У всех остальных же вообще при виде подобного непотребства случается коллапс ганглия. Впрочем, существуют более удобные способы записи регулярных выражений, позволяющие подробно их комментировать.

> Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

— *Jamie Zawinski in comp.lang.emacs*

Регулярки как правило применяются тремя способами:

- Сопоставление (match) — смысл действия в том чтобы выяснить, соответствует ли определенный текст заданному регулярному выражению. Например текст «Пыщь!!!!!11» не соответствует вышеприведенному регэкспу.
- Поиск (find) — позволяет выдрать из текста все последовательности символов, соответствующие регулярке. Например при выполнении данной операции с вышеприведенной регуляркой к любой веб-странице на выходе прогер получит хуй, ибо сий регэксп начинается с ^ и кончается $, что означает что выражение соответствует не просто последовательности символов, а целой строке, или даже всему тексту. Если же убрать данные символы из концов выражения, то счастливый юзер получит все e-mail адреса, указанные на странице, что позволит ему люто, бешено рассылать спам.
- Замена (replace) — позволяет не просто найти, но и ~~уничтожить~~ заменить определенные последовательности символов в строке, что позволяет например легко менять корни слов в тексте, делать всякие там фильтры мата или вчерную пиздить целые сайты, меняя на них ссылки на ходу.

  > Распространенной практикой является использование в письменной речи (а не только в исходниках программ) оператора замены s/// из sed (также есть в perl и VIM). Например, допустив опечатку, можно написать следующим сообщением, скажем, s/монад/номад/g, означает «надо бы заменить „монад“ на „номад“». Еще так можно скорректировать предыдущего оратора в треде или подсказать ему другое направление мысли.

В конечном счете бытовой программист с помощью регэкспов может делать, например, такие вещи:

- Проверять правильность ввода пользователем данных в различных интерфейсах.
- Изменять поведение скриптов с юзерской стороны в зависимости от того, где находится (каков полный адрес страницы) юзер, что удобно на сайтах с изменяющимся контентом.
- Автоматически обрабатывать выдачу различных поисковиков и сервисов типа Vkontakte, дабы собрать базу емылов/асечек и онанировать на нее.
- Обрабатывать выдачу яндекса, чтобы пиздить контент и заполнять интернеты богомерзкими говносайтами, чуть более чем полностью состоящими из рекламы и приносящими своему создателю 2-3 доллара в день.
- Обрабатывать спизженное, чтобы получить псевдоуникальный контент и тем самым чуть-чуть поднять свой бредосайт над своими менее удачливыми собратьями, обогатившись на цент-два в день.
- Парсить странички с проном, выкачивая его тоннами
- Быстренько пофиксить баг на сайте, заюзав ob_start/ob_get_contents + сабж, чем доставив лютую анальную боль будущему баг-фиксеру
- ~~Грабить корованы~~
- ??????
- PROFIT

Следует, однако, понимать, что использование регулярных выражений для парсинга HTML знаменует пришествие Zalgo. Пруфлинк: http://stackoverflow.com/questions/1732348/regex-match-open-tags-except-xhtml-self-contained-tags

Пример

Чтобы во всей полноте ощутить регулярность регулярных выражений, приведём пример несложного выражения в perl flavor, которое проверяет почтовый адрес на соответствие RFC 822 (на самом деле, этот стандарт несколько устарел и в данный момент актуальнее ~~RFC 2822~~ RFC 5322, но он не столь нагляден).

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t
```

Для справки также приводится функция, с помощью которой данное регулярное выражение конструируется (написана она, собственно, аккурат по RFC):

```
sub make_rfc822re {
#   Basic lexical tokens are specials, domain_literal, quoted_string, atom, and
#   comment.  We must allow for lwsp (or comments) after each of these.
#   This regexp will only work on addresses which have had comments stripped
#   and replaced with lwsp.

    my $specials = '()<>@,;:\\\".\\[\\]';
```

```
    my $controls = '\\000-\\031';

    my $dtext = "[^\\[\\]\\r\\\\]";
    my $domain_literal = "\\[(?:$dtext|\\\\.)*\\]$lwsp*";

    my $quoted_string = "\"(?:[^\\\"\\r\\\\]|\\\\.|$lwsp)*\"$lwsp*";

#   Use zero-width assertion to spot the limit of an atom.  A simple
#   $lwsp* causes the regexp engine to hang occasionally.
    my $atom = "[^$specials $controls]+(?:$lwsp+|\\Z|(?=[\\[\"$specials]))";
    my $word = "(?:$atom|$quoted_string)";
    my $localpart = "$word(?:\\.$lwsp*$word)*";

    my $sub_domain = "(?:$atom|$domain_literal)";
    my $domain = "$sub_domain(?:\\.$lwsp*$sub_domain)*";

    my $addr_spec = "$localpart\@$lwsp*$domain";

    my $phrase = "$word*";
    my $route = "(?:\@$domain(?:,\@$lwsp*$domain)*:$lwsp*)";
    my $route_addr = "\\<$lwsp*$route?$addr_spec\\>$lwsp*";
    my $mailbox = "(?:$addr_spec|$phrase$route_addr)";

    my $group = "$phrase:$lwsp*(?:$mailbox(?:,\\s*$mailbox)*)?;\\s*";
    my $address = "(?:$mailbox|$group)";

    return "$lwsp*$address";
}
```

Если бы не было программистов

Следует заметить, что регулярные выражения существуют не только в ОйТи, но и в сугубо матанских областях знаний, например, в той же дискретной математике. Но там владение регэкспами приносит значительно меньше профита, и посему рассматриваться нами в рамках данной статьи не будет. Кому интересно — читайте педивикию по сабжу.

См. также

- Неведомая ёбаная хуйня
- Когнитивный диссонанс
- Я нихуя не понял
- ЩИТО

Ссылки

- Визуальный редактор регекспов, для желающих приобщиться, но ленящихся лезть в дебри языков программирования
- Преобразователь регекспов в няшную инфографику, понятную даже ~~дураку~~ неспециалисту
- Краткий список спецсимволов от создателей огнелиса
- Список модификаторов регекспов
- Как сломать интернет одним регулярным выражением

На регэкспах(!) можно делать игры:

- **Умные девушки**/55238 пишут тетрис на sed.
- Умные парни пишут на sed шахматы (пруфец, исходники).

Примечания

1. ↑ В виду того, что они не знают, что в PHP 5.3.0 признаны устаревшими (вызов кидает DEPRECATED) и окончательно исчезнут в PHP6

Языки программирования
++i + ++i 1C AJAX BrainFuck C Sharp C++ Dummy mode Erlang Forth FUBAR God is real, unless explicitly declared as integer GOTO Haskell Ifconfig Java JavaScript LISP My other car Oracle Pascal Perl PHP Prolog Pure C Python RegExp Reverse Engineering Ruby SAP SICP Tcl TeX Xyzzy Анти-паттерн Ассемблер Быдлокодер Выстрелить себе в ногу Грязный хак Дискета ЕГГОГ Индусский код Инжалид дежице Капча КОИ-8 Костыль Лог Метод научного тыка Очередь Помолясь Проблема 2000 Программист Процент эс Рекурсия Свистелки и перделки Спортивное программирование СУБД Тестировщик Умение разбираться в чужом коде Фаза Луны Фортран Хакер Языки программирования

w:Regexp en.w:Regexp