Формат записи параметров поиска ссылок и регулярных выражений в ПК XseoN

При парсинге ПК XseoN использует параметры поиска, заложенные в служебном файле. Посмотреть/отредактировать параметры поиска и регулярное выражение можно, нажав на кнопку **рег. выр.**

ниже приведено содержимое служебного файла, в котором записаны параметры парсинга поисковых систем и регулярное выражение

***********ПОИСК В ПОИСКОВЫХ СИСТЕМАХ********формат записи: Имя поисковой системы [текст перед ссылкой, до http://] [символ или текст после ссылки]***********
//текст перед ссылкой Google
Google [<a href="/url?q=] ["]
//текст перед ссылкой Yandex
Yandex [class="b-serp-item__title-link" href="] ["]
//текст перед ссылкой Yandex (блоги)
Yandex [<a class="b-serp-url__link" href="] ["]
//текст перед ссылкой Yahoo
Yahoo [yschttl spt" href=] ["]
//текст перед ссылкой Rambler
Rambler [<a target="_blank" href="] ["]
***********ПОИСК В КОНТЕНТЕ УРЛ********ОПИСАНИЕ - http://www.xseon.ru/index.php?option=com_content&view=article&id=115&Itemid=85. ВОЗМОЖНО ПАРСИТЬ С ПОМОЩЬЮ РЕГУЛЯРНОГО ВЫРАЖЕНИЯ ИЛИ С ПОМОЩЬЮ ЗАДАНИЯ КОНТЕНТА ДО И ПОСЛЕ ИСКОМОГО ОБЪЕКТА**********
URLListR [.([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3})D+([0-9]{2,5}).] [1]:[2]

Поиск ссылок в поисковых системах

В служебном файле данный раздел начинается с "*********ПОИСК В ПОИСКОВЫХ СИСТЕМАХ******** "

**********ПОИСК В ПОИСКОВЫХ СИСТЕМАХ********формат записи: Имя поисковой системы [текст перед ссылкой, до http://] [символ или текст после ссылки]*********
//текст перед ссылкой Google
Google [<a href="/url?q=] ["]
//текст перед ссылкой Yandex
Yandex [class="b-serp-item__title-link" href="] ["]
//текст перед ссылкой Yandex (блоги)
Yandex [<a class="b-serp-url__link" href="] ["]
//текст перед ссылкой Yahoo
Yahoo [yschttl spt" href=] ["]
//текст перед ссылкой Rambler
Rambler [<a target="_blank" href="] ["]

При парсинге ссылок в поисковых системах используется простое задание текста до ссылки и текста после ссылки:

Формат записи:

Google [<a href="/url?q=] ["]

где

Google - означает, что параметры указаны для парсинга ссылок в ПС Google,

далее идет пробел,

[<a href="/url?q=] - символы находящиеся перед ссылкой,

далее идет пробел,

["] - символы находящиеся после ссылки.

Так же задаются параметры для Yandex, Yahoo и Rambler.

Поиск объектов с помощью регулярных выражений

В служебном файле данный раздел начинается с "*********ПОИСК В КОНТЕНТЕ УРЛ******** "

**********ПОИСК В КОНТЕНТЕ УРЛ********ОПИСАНИЕ - http://www.xseon.ru/index.php?option=com_content&view=article&id=115&Itemid=85. ВОЗМОЖНО ПАРСИТЬ С ПОМОЩЬЮ РЕГУЛЯРНОГО ВЫРАЖЕНИЯ ИЛИ С ПОМОЩЬЮ ЗАДАНИЯ КОНТЕНТА ДО И ПОСЛЕ ИСКОМОГО ОБЪЕКТА********** URLListR [.([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3})D+([0-9]{2,5}).] [1]:[2]

При парсинге объектов поиска с помощью регулярных выражений может использоваться два варианта поиска:

1. используется простое задание текста до объекта поиска и текста после объекта

поиска (так же как и при парсинге ссылок в ПС):

Формат записи:

пример:

URLListC [prefix] [<a href="/url?q=] ["] [postfix]

где

URLLictC - означает, что параметры указаны для парсинга контента страницы при помощи простого задания текста до и после искомого объекта,

далее идет пробел,

[prefix] - набор символов, который подстанавливается перед результатом поиска,

[<a href="/url?q=] - символы находящиеся перед объектом поиска,

далее идет пробел,

[''] - символы находящиеся после объекта поиска,

далее идет пробел,

[postfix] - набор символов, который подстанавливается после результата поиска.

2. используется регулярное выражение:

Формат записи:

пример:

URLListR [.([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3})D+([0-9]{2,5}).] Proxy: [1]:[2] no checked

где

URLLictR - означает, что параметры указаны для парсинга контента страницы при помощи регулярного выражения,

далее идет пробел,

[.([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3})D+([0-9]{2,5}).] - регулярное выражение,

далее идет пробел,

Proxy: [1]:[2] no checked - это строка параметров форматирования записи результата поиска в итоговый txt файл,

все что записано без квадратных скобок вставляется в итоговую строку без изменений, в выше приведенном примере это:

Proxy:

:

no checked

В квадратных скобках **[1]** и **[2]** записываются номера групп из регулярного выражения, Группа образуется заключением части регулярного выражения в круглые скобки (). В нашем примере:

([0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}.[0-9]{1,3}**)** - первая группа (адрес прокси сервера), ее номер по порядку
1
;

([0-9]{2,5}**)** - вторая группа (порт прокси сервера),ее номер по порядку **2**;

и т.д., может быть много.

Все что найдено по регулярному выражению целиком всегда попадает в группу под номером **0.**

Приминительно к нашему примеру:

Proxy: [1]:[2] no checked - будет записано: ***Proxy: 123.123.123.123:8080 no checked***

ВИДЕО РАБОТЫ ПАРСЕРА ССЫЛОК В РЕЖИМЕ ПОИСКА В КОНТЕНТЕ

Синтаксис регулярных выражений

В парсере используется библиотека TRegExpr, разработчик Andrey V. Sorokin

Введение

Регулярные выражения - это широкоиспользуемый способ описания шаблонов для поиска текста и проверки соответствия текста шаблону. Специальные **метасимволы** позволяют определять, например, что Вы ищете подстроку в начале входной строки или определенное число повторений подстроки.

TestRExp.dpr - программа для теста и отладки созданных Вами регулярных выражений.

Простое сравнение

Любой символ совпадает с самим собой, если он не относится к специальным метасимволам описанным чуть ниже.

Последовательность символов совпадает с такой же последовательностью во входной строке, так что шаблон "bluh" совпадет с подстрокой "bluh" во входной строке. Пока все просто, не так ли ?

Если необходимо, чтобы метасимволы или **escape-последовательности** воспринимались как обычные символы, их нужно предварять символом "", например, метасимвол "^" обычно совпадает с началом строк, однако, если записать его как "^", то он будет совпадать с символом "^", "\" совпадает с "" и т.д.

Примеры:
foobar *находит 'foobar'*
^FooBarPtr *находит '^FooBarPtr'*

Escape-последовательности

Любой символ может быть определен с помощью escape последовательности, так же как это делается в языках C или Perl: "n" означает начало строки, "t" - табуляцию и т.д.. Вообще, xnn, где nn это последовательность шестнадцатеричных цифр, означает символ с ASCII-кодом nn. Если необходимо определить двухбайтный (Unicode) символ, используйте формат 'x{nnnn}', где 'nnnn' - одна или более шестнадцатеричных цифр.

xnn *символ с шестнадцатеричным кодом nn*
x{nnnn} *символ с шестнадцатеричным кодом nnnn (более одного байта можно задавать только в режиме Unicode)*
t *табуляция (HT/TAB), можно также x09*
n *новая строка (NL), можно также x0a*
r *возврат каретки (CR), можно также x0d*
f *перевод формата (FF), можно также x0c*
a *звонок (BEL), можно также x07*
e *escape (ESC), можно также x1b*

Примеры:

foox20bar *находит 'foo bar' (обратите внимание на пробел посередине)*
tfoobar *находит 'foobar' предшествуемый табуляцией*

Перечни символов

Вы можете определить перечень, заключив символы в []. Перечень будет совпадать с любым **одним** символом перечисленным в нем.

Если первый символ перечня (сразу после "[") - "^", то такой перечень совпадает с любым символом **не** перечисленным в перечне.

Примеры:
foob[aeiou]r *находит 'foobar', 'foober' и т.д. но не 'foobbr', 'foobcr' и т.д.*
foob[^aeiou]r *находит 'foobbr', 'foobcr' и т.д.. но не 'foobar', 'foober' и т.д.*

Внутри перечня символ "-" может быть использован для определения **диапазонов** символов, например a-z представляет все символы между "a" и "z", включительно.

Если Вам необходимо включить в перечень сам символ "-", поместите его в начало или конец перечня или предварите ". Если Вам необходимо поместить в перечень сам символ ']', поместите его в самое начало или предварите ".

Примеры:
[-az] *'a', 'z' и '-'*
[az-] *'a', 'z' и '-'*
[a-z] *'a', 'z' и '-'*
[a-z] *все 26 малых латинских букв от 'a' до 'z'*
[n-x0D] *#10, #11, #12, #13.*
[d-t] *цифра, '-' или 't'.*
[]-a] *символ из диапазона ']'..'a'.*

Метасимволы

Метасимволы - это специальные символы, являющиеся важнейшим понятием в регулярных выражениях. Существует несколько групп метасимволов.

Метасимволы - разделители строк

^ *начало строки*
$ *конец строки*
A *начало текста*
Z *конец текста*
. *любой символ в строке*

Примеры:

^foobar *находит 'foobar' только если он в начале строки*
foobar$ *находит 'foobar' только если он в конце строки*
^foobar$ *находит 'foobar' только если это единственное слово в строке*
foob.r *находит 'foobar', 'foobbr', 'foob1r' и т.д.*

Метасимвол "^" по умолчанию совпадает только в начале входного текста, а метасимвол "$" - только в конце текста. Внутренние разделители строк, имеющиеся в тексте, не будут совпадать с "^" и "$".
Однако, если Вам необходимо работать с текстом как с многострочным, чтобы "^" совпадал после каждого разделителя строки внутри текста, а "$" - перед каждым разделителем, то Вы можете включить модификатор /m .

Метасимволы A и Z аналогичны "^" и "$", но на них не действует модификатор /m , т.е. они всегда совпадают только с началом и концом всего входного текста.

Метасимвол "." по умолчанию совпадает с любым символом, однако, если Вы выключите модификатор /s , то '.' не будет совпадать с разделителями строк.

TRegExpr интерпретирует разделители строк так, как это рекомендовано на www.unicode.org (http://www.unicode.org/unicode/reports/tr18/):

"^" совпадает с началом входного текста, а также, если включен модификатор /m , с точкой непосредственно следующей после x0Dx0A, x0A или x0D (если Вы используете Unicode-версию TRegExpr, то также x2028 или x2029 или x0B или x0C или x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности x0Dx0A.

"$" совпадает с концом входного текста, а также, если включен модификатор /m , с точкой непосредственно предшествующей x0Dx0A, x0A или x0D (если Вы используете Unicode-версию TRegExpr, то также x2028 или x2029 или x0B или x0C или x85). Обратите внимание, что он не совпадает в промежутке внутри последовательности x0Dx0A.

"." совпадает с любым символом, но если выключен модификаторr /s , то "." не совпадает с x0Dx0A и x0A и x0D (если Вы используете Unicode-версию TRegExpr, то не совпадает также с x2028 и x2029 и x0B и x0C и x85).

Обратите внимание, что "^.*$" (шаблон для пустой строки) не совпадает с пустой строкой вида x0Dx0A, но совпадает с x0Ax0D.

Вы можете перенастроить вышеописанное поведение при обработке многострочных текстов - см. описания свойств LineSeparators и LinePairedSeparator, скажем, Вы можете перенастроиться на использование только Unix-разделителей строк n или только DOS/Windows-разделителей rn или же смешанных разделителей (так и настроено по умолчанию) или вообще определить свои собственные разделители

строк!

Метасимволы - стандартные перечни символов

w	*буквенно-цифровой символ или "_"*
W	*не w*
d	*цифровой символ*
D	*не d*
s	*любой "пробельный" символ (по умолчанию - [tnrf])*
S	*не s*

Стандартные перечни w, d и s можно использовать и внутри **перечней символов**.

Примеры:

foobdr	*находит 'foob1r', "foob6r' и т.д. но не 'foobar', 'foobbr' и т.д.*
foob[ws]r	*находит 'foobar', 'foob r', 'foobbr' и т.д. но не 'foob1r', 'foob=r' и т.д.*

TRegExpr использует свойства SpaceChars и WordChars для того, чтобы определять стандартные перечни w, W, s, S, т.е. Вы легко можете переопределить состав этих перечней.

Метасимволы - границы слов

b	*Совпадает на границе слова*
B	*Совпадает не на границе слова*

Граница слова (b) это точка между двумя символами, один из которых удовлетворяет w, а другой - W (в любом порядке), при этом перед началом и после конца строки подразумевается W.

Метасимволы - повторения

После любого элемента регулярного выражения может следовать очень важный тип метасимвола - **повторитель**. Используя их Вы можете определить число допустимых повторений предшествующего символа, метасимвола или подвыражения.

*	*ноль или более раз ("жадный"), то же что {0,}*
+	*один или более раз ("жадный"), то же что {1,}*
?	*ноль или один раз ("жадный"), то же что {0,1}*
{n}	*точно n раз ("жадный")*
{n,}	*не менее n раз ("жадный")*
{n,m}	*не менее n но не более m раз ("жадный")*
*?	*ноль или более раз ("не жадный"), то же что {0,}?*

+? *один или более раз ("не жадный"), то же что {1,}?*
?? *ноль или один раз ("не жадный"), то же что {0,1}?*
{n}? *точно n раз ("не жадный")*
{n,}? *не менее n раз ("не жадный")*
{n,m}? *не менее n но не более m раз ("не жадный")*

Т.о. {n,m} задает **минимум** n повторов и **максимум** - m. Повторитель {n} эквивалентен {n,n} и задает точно n повторов. Повторитель {n,} задает минимум n повторов. Теоретически величина параметров n и m не ограничена, но рекомендуется не задавать большие значения, поскольку в некоторых ситуациях это может потребовать существенных затрат времени и ОЗУ при обработке такого повторителя в связи с рекурсивным характером работы.

Если фигурные скобки встречаются в "неправильном" месте, где они не могут быть восприняты как повторитель, то они вопринимаются просто как символы.

Примеры:
foob.*r *находит 'foobar', 'foobalkjdflkj9r' и 'foobr'*
foob.+r *находит 'foobar', 'foobalkjdflkj9r' но не 'foobr'*
foob.?r *находит 'foobar', 'foobbr' и 'foobr' но не 'foobalkj9r'*
fooba{2}r *находит 'foobaar'*
fooba{2,}r *находит 'foobaar', 'foobaaar', 'foobaaaar' и т.д.*
fooba{2,3}r *находит 'foobaar', или 'foobaaar' но не 'foobaaaar'*

Небольшое пояснение по поводу "жадности". "Жадные" варианты повторителей пытаются захватить как можно большую часть входного текста, в то время как "не жадные" - как можно меньшую. Например, 'b+' как и 'b*' примененные к входной строке 'abbbbc' найдут 'bbbb', в то время как 'b+?' найдет только 'b', а 'b*?' - вообще - пустую строку; 'b{2,3}?' найдет 'bb', в то время как 'b{2,3}' найдет 'bbb'.

Вы можете переключить все повторители в выражении в "не жадный" режим, воспользовавшись модификатором /g .

Метасимволы - варианты

Вы можете определить перечень **вариантов**, используя метасимвол "|" для их разделения, например "fee|fie|foe" найдет "fee" или "fie" или "foe", (так же как "f(e|i|o)e"). В качестве первого варианта воспринимается все от предыдущего метасимвола "(" или "[" или от начала выражения до первого метасимвола "|", в качестве последнего - все от последнего "|" до конца выражения или до ближайшего метасимвола ")". Обычно, чтобы не запутаться, набор вариантов всегда заключают в скобки, даже если без этого можно было бы обойтись.
Варианты пробуются начиная с первого и попытки завершаются сразу же как удастся подобрать такой при котором совпадет вся последующая часть выражения. Это означает, что варианты не обязательно обеспечат "жадное" поведение. Например,

если применить выражение "foo|foot" ко входной строке "barefoot", то будет найдено "foo" так это первый вариант который позволил совпасть всему выражению. Обратите внимание, что метасимвол "|" воспринимается как обычный символ внутри перечней символов, например, [fee|fie|foe] означает ровно то же самое что и [feio|].

Примеры:
foo(bar|foo) *находит 'foobar' или 'foofoo'.*

Метасимволы - подвыражения

Метасимволы (...) могут также использоваться для задания подвыражений - по завершении поиска выражения Вы можете обратиться к любому подвыражению используя свойства MatchPos, MatchLen и Match, а также подставлять подвыражения в некий шаблон, используя метод Substitute).

Подвыражения нумеруются слева направо, в порядке появления открывающих скобок. Первое подвыражение имеет номер '1' (выражение в целом - '0', к нему можно обращаться в Substitute как '$0' так и '$&').

Примеры:
(foobar){8,10} *находит строку содержащую 8, 9 или 10 копий 'foobar'*
foob([0-9]|a+)r *находит 'foob0r', 'foob1r' , 'foobar', 'foobaar', 'foobaar' и т.д.*

Метасимволы - обратные ссылки

Метасимволы от 1 до 9 воспринимаются как обратные ссылки. <n> совпадает с ранее найденным подвыражением #<n>.

Примеры:
(.)1+ *находит 'aaaa' и 'cc'.*
(.+)1+ *также находит 'abab' и '123123'*
□□(["']?)(d+)1 *находит "13" (в дв.кавычках), или '4' (в один.кавычках) или 77 (без кавычек) и т.д.*

Модификаторы

Модификаторы служат для изменения режимов работы TRegExpr.

Вы можете изменять модификаторы нескольких способами.
Любой модификатор может меняться с помощью специальной конструкции внутри регулярного выражения.
Также, Вы можете присвоить значение соответствующему свойству экземпляра объекта TRegExpr (например, ModifierX для изменения модификатора /x, или ModifierStr для изменения сразу нескольких модификаторов). Значения по умолчанию для новых экземпляров объектов TRegExpr определены в глобальных константах, например RegExprModifierX определяет значение по умолчанию для ModifierX.

i

Регистро-независимый режим (по умолчанию использует выбранный в ОС язык по умолчанию

m

Воспринимать входной текст как многострочный, при этом метасимволы "^" и "$" будут совпад

s

Воспринимать входной текст как одну строку. При этом метасимвол "." совпадает с любым сим

g

Не стандартный модификатор. Выключая его Вы переключаете все повторители в "не жадны

x

Позволяет форматировать шаблон чтобы обеспечить более легкую читаемость (см. описание

r

Не стандартный модификатор. Если включен, то диапазоны вида а-я включают также букву

Модификатор /x заставляет TRegExpr игнорировать пробелы, табуляции и разделители строк, что позволяет форматировать текст выражения. Кроме того, если встречается символ #, то все последующие символы до конца строки воспринимаются как комментарий, например:

```
(
(abc) #Комментарий 1
/ #Пробелы внутри выражения также игнорируются
(efg) #Комментарий 2
)
```

Естественно, это означает что, если Вам нужно вставить в выражение пробел, табуляцию или разделитель строки или #, то в расширенном (/x) режиме это можно сделать только предваряя их '/' или используя /xnn (внутри перечней символов все эти символы воспринимаются как обычно)

Расширения Perl

(?imsxr-imsxr)
Позволяет изменять значения модификаторов

Примеры:
(?i)Saint-Petersburg *находит 'Saint-petersburg' и 'Saint-Petersburg'*
(?i)Saint-(?-i)Petersburg *находит 'Saint-Petersburg' но не 'Saint-petersburg'*
(?i)(Saint-)?Petersburg *находит 'Saint-petersburg' и 'saint-petersburg'*
((?i)Saint-)?Petersburg *находит 'saint-Petersburg', но не 'saint-petersburg'*

(?#text)
Комментарий, просто игнорируется. Обратите внимание, что в комментарии такого вида невозможно поместить символ ")", поскольку он воспринимается как конец комментария.