Expressões Regulares

Felipe S. Oliveira Barros; Felipe dos S. Pinheiro; Luciano P. Soares.
Centro Universitário da Cidade do Rio de Janeiro, Escola de Ciências Exatas e Tecnologia, NUPAC - Núcleo de Projetos e Pesquisa em Aplicações Computacionais, Metrô-Carioca (Estação) - Av. Rio Branco e Largo da Carioca s/n

Email: {felipebarros.ti, felipe0pinheiro, lpsoares}@gmail.com

Resumo

O processamento de texto é uma atividade muito comum em diversas áreas de atuação, mas principalmente a Financeira (processamento de arquivos bancários com diferentes layouts) e a Computacional(criação de algoritmos para tratar arquivos texto dos mais diversos tipos)

Com o aumento da demanda e a necessidade cada vez maior de velocidade e eficiência no processamento textual surgiu uma ferramenta que não promete resolver todos os problemas, mas que pode ser uma verdadeira mão-na-roda para quem faz esse tipo de trabalho, são as Expressões Regulares, RegEx, ou apenas ER.

Expressões Regulares tem sido uma ferramenta eficiente para identificar caracteres particulares, palavras, subgrupo de caracteres e/ou combinações destes dentro de um texto. Expressões mais elaboradas são de difícil compreensão para quem está começando a lidar com elas, mas utilizá-las é de suma importância para quem quer reduzir consideravelmente as linhas de código produzidas ou o tempo gasto com processamento de arquivos .txt, csv ou qualquer outro formato de arquivo cujo conteúdo possa ser exibido em texto.

ER's são escritas em uma linguagem formal, são compostas por símbolos e caracteres com funções especiais que agrupados entre si e com caracteres literais formam uma expressão. São interpretadas por um processador de expressão regular, que analisa o texto e identifica as partes que "casam" , ou seja, obedecem exatamente a todas as condições da expressão fornecida. [1]

O assunto ER é muito extenso, assim como o seu imenso leque de utilidades. Este artigo tem como objetivo introduzir alguns conceitos básicos dobre RegEx e a criação de uma ferramenta utilizando as linguagens PHP, JavaScript e HTML para a construção de Expressões Regulares básicas. O público-alvo para a utilização do sistema é de pessoas que ainda não dominem o assunto, mas poderão com ele criar expressões básicas com apenas alguns cliques.

Abstract:

Word processing is a very common several areas, but mainly the Financial (banking with file processing different layouts) and Computer (creation of algorithms to handle text files from various types)

With increasing demand and growing need speed and efficiency in processing textual came a tool that does not promise to solve all problems, but can be a real hands-on wheel for those who make this kind of work, are the expressions Regular, RegEx, or only ER.

Regular Expressions has been an effective tool to identify particular characters, words, subset of characters and / or combinations of these within a text. More elaborate expressions are difficult to understanding for those just starting to deal with them, but using them is of paramount importance to anyone considerably reduce the lines of code produced or time spent on processing files. txt, csv or any other format file whose contents can be displayed in text.

ER's are written in a formal language, are composed by symbols and characters with special functions that grouped together and form a literal characters expression. Are interpreted by a processor regular expression, which examines text and identifies shares that "match", ie, obey exactly all the conditions of the given expression. [1]

The subject is very extensive ER, as well as its immense range of uses. This article aims to introduce some basic concepts and double RegEx creating a tool using the PHP, JavaScript and HTML to build Expressions Regular basic. The target audience for the use of system is the people who have not mastered the subject, but with it may create with only basic expressions a few clicks

1 Introdução

A história das expressões regulares tem origem nas ideias do matemático Stephen Cole Kleene (Hartford, 5 de janeiro de 1909 –Madison, 25 de janeiro de 1994 [2]) que, utilizando uma notação matemática própria chamada de "conjuntos regulares"para descrever modelos de computação(autômatos finitos), foi o criador da Algebra Kleene e das Expressões Regulares.

Não existe uma definição formal sobre o que seria uma expressão regular, tão pouco um padrão de texto que pode ser qualificado ou não como expressão regular.

As expressões regulares ajudam a reduzir a um valor mínimo as linhas de código em implementações que trabalham com texto. Atualmente são usadas para buscar e/ou substituir, dentro de uma string, uma cadeia de caracteres especifica, mas também pode ser utilizado para buscar e validar datas, horários, endereços IP e MAC, endereços de e-mail, etc [1] sendo usados não apenas por

programadores, mas também por qualquer prossional que trabalhe diretamente com textos.

O número de ferramentas que possibilitam o uso de expressões regulares é cada fez maior. Entre as mais conhecidas podemos citar BrOffice, NotePad++, GoogleDocs , Ed, vi e vim. Vale lembrar que nem todas as ferramentas implementem a mesma versão de RegEx, ou implementem versões incompletas, onde um ou outro metacaracter ou caracter especial não é reconhecido, por isso vale a pena dar uma olhada na documentação da ferramenta com qual deseja trabalhar com RegEx, afim de aproveitar ao máximo os recursos disponíveis.

O objetivo deste artigo é introduzir RegEx e apresentar uma ferramenta online para criação de Expressões Regulares básicas, com foco em iniciantes e pessoas que não dominam o assunto mas desejam aprender como otimizar seu trabalho.

A seção 2 descreve o que é preciso(quais os símbolos e convenções usar) para se criar uma ER, utilizando exemplos simples. Trata-se de um assunto extenso, porém extremamente necessário para a compreensão das ER's criadas com o sistema desenvolvido aqui, ou qualquer outro.

A seção 3 fala tecnicamente do Online RegEx Build Assistant, desde a lógica utilizada na implementação até exemplos práticos de criação de Expressões.

Na seção 4, finalmente utilizamos as expressões geradas em nosso Assistente para resolver problemas reais e comparar resoluções com RegEx e soluções tradicionais.

2 Conceitos básicos

Na informática, o tempo que se leva para processar uma informação é muito valioso e as expressões regulares nos possibilitam resultados rápidos e precisos.

Podemos usá-las para procurar textos específicos, partes de textos, textos no inicio ou no fim de uma linha, textos com uma seqüência especifica de caractere, etc.

Ao utilizarmos uma ER procuramos por um padrão que "case" com o padrão especificado. O termo casar dentro de ER quer dizer combinar, e trata-se da idéia central das ER, pois sempre estamos procurando por uma string que case com as informações fornecidas.

Para fazer uso das ER utilizamos uma combinação de símbolos que chamamos de metacaracteres. Cada metacaracter tem a sua função específica no momento de casar uma ER. Os metacaracteres são dividdos de acordo com a sua função, que são:

- Metacaracter tipo Representante;

- Metacaracter tipo Quantificador; e
- Metacaracter tipo Ancora.

Existem outros tipos de metacaracteres, os estendidos, porém os 3 tipos citados acima são suficientes para escrever uma expressão regular básica e para a implementação do Assistente. Na informática, o tempo que se leva para processar uma informação é muito valioso e as expressões regulares nos possibilitam resultados rápidos e precisos.

Podemos usá-las para procurar textos específicos, partes de textos, textos no inicio ou no fim de uma linha, textos com uma seqüência especifica de caractere, etc.

2.1 Metacaracter tipo Representante

O metacaracter do tipo Representante representa um ou mais caracteres, contudo representam apenas uma posição.

2.1.1 O metacaracter ponto (.)

O metacaracter ponto (.), representa um único caractere, podendo casar com qualquer numero ou letra, inclusive o próprio ponto, na posição na qual ele foi colocado. Exemplos:
Tarefa: Procurar em um texto qualquer caracter que esteja entre aspas duplas.
Solução: "."
Tarefa: Procurar em um texo qualquer caracter entre aspas simples.
Solução: '.'

2.1.2 O metacaracter lista ([])

O metacaracter lista ([ ]), representa uma lista de argumentos que são permitidos no momento da junção, ou seja, o caracter que está sendo procurando naquela posição só será retornado na busca se estiver dentro dos colchetes. Ela torna-se mais eciente que o ponto, pois ela não aceita qualquer argumento, somente os especificados.
Exemplos:
Podemos utilizar esse recurso para encontrar palavras com uma possível grafia incorreta ou quando não se sabe se está começando em maiuscula ou minuscula. A ER <n[ãa]o> procura pela string "não" ou "nao" , <[Nn]ão> casa "Não" e "não" .

Se for preciso buscar um valor numérico entre 1 e 9 podemos usar a lista <[123456789]>,ou então <[1-9]> (muito mais cômodo). O símbolo "–" neste caso representa um intervalo, que também consegue indicar intervalo no alfabeto somente maiúsculo <[A-Z]> ou somente minúsculo <[a-z]>, isso não inclui as vogais acentuadas, "Ç" e "ç" . Se eu quiser uma lista com letras maiúsculas, minúsculas e números uma das maneiras é usar a ER <[A-Za-z0-9]>.

Vale esclarecer que tudo o que estiver dentro dos colchetes será exatamente o que estiver dentro dos colchetes exceto o "–" que irá representar o intervalo, quando estiver entre dois caracteres. Dessa maneira a ER [.] vai representar apenas um ponto. Para representar o "-" dentro da lista devemos colocá-lo como ultimo item da, ou seja, fora do seu lugar padrão como caracter especial, entre dois valores.

Como a gramática da língua Portuguesa e algumas outras necessitam de caracteres acentuados é possível fazer uso de caracteres especiais chamados de POSIX. Como na lista [A-Za-z] não estão inclusos os caracteres acentuados, é possível substitui-la pela classe[:upper:] (os colchetes fazem parte da classe), que inclui os caracteres acentuados bem como o "Ç" e o "ç" .

classe POSIX	SIMILAR	SIGNIFICA
[:upper:]	[A-Z]	letras maiúsculas
[:lower:]	[a-z]	letras minúsculas
[:alpha:]	[A-Za-z]	maiúsculas/minúsculas
[:alnum:]	[A-Za-z0-9]	letras e números
[:digit:]	[0-9]	números
[:xdigit:]	[0-9A-a-f]	números hexadecimais
[:punct:]	[.,!?:...]	sinais de pontuação
[:blank:]	[\t]	espaço e TAB
[:space:]	[\t, \n, \r, \f, \v]	caracteres brancos
[:cntrl:]	-	caracteres de controle
[:graph:]	[^\t,\n,\r, \f, \v]	caracteres imprimíveis
[:print:]	[^,\t,\n, \r,\f, \v]	imprimíveis e o espaço

Figura 1 - Tabela de classes PÒSIX

2.1.3 O metacaracter lista negada ([^...])

O metacaracter lista negada ([^...]) possui a mesma sintaxe da lista, aceita intervalos literários, numéricos e classes Posix, mas o seu signicado é inverso. O que está fora da lista é aceito e o que está dentro não, ou seja, na ER <[^0-9]> seria aceito qualquer caractere exceto números. Épossível também excluir o alfabeto minúsculo ou maiúsculo com as ER's <[^a-z]> e <[^A-Z]> respectivamente ou usar as classes POSIX's <[^[:lower:]]> e <[^[:upper:]]> .

2.2 Metacaracteres Quanticadores

Os metacaracteres do tipo quantificador dizem quantas vezes um determinado caractere ou matacaracter pode aparecer na string. Esse tipo de metacaracter é conhecido como guloso, pois se for usado permitindo uma repetição, ele tentará usar o numero máximo de repetições possíveis.

2.2.1 O Metacaracter Opcional (?)

Muito útil para procurar palavras no singular e no plural, o metacaracter opcional (?) é usado quando queremos a ocorrência de 0 ou 1 caractere, somente para o caractere marcado e não para a palavra toda.
Exemplo:

- <festas?> retorna ou “festa ” “festas ” pois o "s"foi marcado como opcional.
- <fala[r!]?> retona “falar” , “fala” e “fala” .
- <1581?8> retorna “1581” ou “1588” pois o “1” foi marcado como opcional.

2.2.2 O Metacaracter Asterisco (*)

O metacaracter asterisco permite o caractere aparecer nenhuma vez ou em uma quantidade infinita de vezes.
Exemplos:

- <6*0> retorna “0”, “60”, “660”, “6660”, “66660”, ..., “66666666666666660 ”,...
- <b[ip]*> retorna “b”, “bi”, “bii”, “bip”, “biipp”, “bpi”, “bppii”.....

Ao utilizar a combinação do “.”e do “*”, ou seja “.*”que retornar qualquer coisa em qualquer quantidade.

2.2.3 Metacaracter Mais (+)

Semelhante ao asterisco o mais (+) diferencia-se em um único ponto, o caracter deve aparecer pelo menos uma vez, no asterisco ele não precisava aparecer. Útil quando queremos no mínimo uma aparição. Usando os mesmos exemplos do metacaracter asterisco podemos perceber que “0” sozinho não é retornado e que “b ” sozinho também não:

- <6+0> retorna “60”, “660”, “6660”, “66660”, ..., “66666666666666660 ”,...
- <b[ip]+> retorna “bi”, “bii”, “bip”, “biipp”, “bpi”, “bppii”.....

2.2.4 O Metacaracter chaves ({})

O metacaracter chaves ( ) é usado para especificar quantas repetições de cada caractere queremos ter. Se escrevermos a,b queremos dizer algo de a até b. Exemplo:

- <{1,9}> retorna "9", "99", "999", "9999", "99999",... e "999999999".

Partindo dessa armação podemos perceber que é possível uma substituição da combinação de chaves pelos metacaracteres asterisco, opcional e mais.

2.3 Metacaracteres Âncora

Esse tipo de metacaracter não faz junção de outro caractere ou determinauma quantidade, ele marca uma posição especifica na linha. Por causa da função que exerce, o metacaracter não pode ser quanticado, ou seja, os metacaracteres asterisco, mais e opcional não exercem influência sobre estes.

2.3.1 O metacaracter Circunflexo (^)

Indica que estamos procurando por algo no começo da linha. É possível combinar a lista negada com o circunexo.
Exemplo:

- <^[0-9]> indica que estamos procurando linhas que começam com números.números.

- <^[^a-z]> indica as linhas que não começam com letra minúscula.

2.3.2 O Metacaracter Cifrão ($)

O metacaracter cifrão é usado da mesma forma que o circunflexo, mas o cifrão indica o fim de uma linha. Exemplo

- <[0-9]$> indica uma linha que termina com números.
- <^$> indica uma linha vazia.
- <..... $> indica uma linha que termine com 5 caracteres.
- <^.10,50$> indica linhas que tenham entre 10 e 50 caracteres.

2.3.3 O metacaracter borda (\b)

O metacaracter borda (\b) marca o inicio onde a palavra inicia ou onde ela termina, ou ainda os dois simultaneamente, útil para encontrar palavras exatas e não parciais. A seguir um exemplo do comportamento da ER com a palavra dia [1].

- <dia> retorna "dia", "diafragma", "radial", "melodia", "bom-dia"!
- <\bdia> retorna "dia", "diafragma".
- <dia\b> retorna "dia", "melodia", "bom-dia!"
- <\bdia\b> retorna "dia".

2.4 Metaceracteres Extendidos.

Existem metacarecteres que não possuem uma especicação relacionada com os anteriores por isso são denominados metacaracteres extendidos[1] e são amplamente usados para facilitar escrita de uma ER.

2.4.1 O Metacaracter Escape (\)

Se quisermos representar um matacaracter na forma literal, podemos fazê-lo através de uma lista. No entanto a lista é mais adequada para representar um conjunto de caracteres. Nesse caso a solução mais viável seria o uso do metacaracter escape.
Tarefa: Representar através de uma ER um numero de CPF no formato xxx.xxx.xxx-xx

- <[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}-[0-9]{1,2}>

2.4.2 O Metacaracter grupo "()"

O metacaracter grupo "()"é muito utilizado para agrupar outros metacaracteres, caracteres e subgrupos tornando seu uso essecial para simplicar uma expressão regular. Todo o conteudo que estiver entre parenteses será identicado como um grupo, e este deverá casar dentro da RegEx.
Exemplos:
<(www)?google.com(.br)> retorna www.google.com.br, www.google.com, google.com e google.com.br

2.4.3 O Metacaracter OU (|)

O metacaracter OU (|) funciona como um operador booleano, possibilitando alternativas diversas para a expressão, ao usá-lo podemos casar todas estas possibilidades, usando o metacaracter "( )"tornamos o ou"(|) mais poderoso. Tarefa: Encontrar endereços de internet que sejam https:// ou estilo ftp:// [1]. Resposta: <(http://|ftp://)>

2.5 Precedência entre metacaracteres

Os metacarcteres possuem uma precedência para melhor entendermos e determinamos uma RegEx.

Tipo de meta	Exemplo	Procedência
quantificador	ab+	maior
concatenação	ab	média
ou	ab|c	menor

Figura 2 - Tabela de procedência entre os metacaracteres.

Desda forma uma RegEx 12* não seria uma RegEx 1 com 2 em qualquer quantidade e sim 1 seguido de 2 em qualquer quantidade.

3 O Software Regex Build Assistent

4 Avaliação da ferramenta

A motivação, que levou ao desenvolvimento da ferramenta foi o uso de expressão regular no código de programas, no entanto isso nem sempre é possivel devido ao pouco ou nenhum conhecimento por parte dos programadores sobre expressão regular. Dividimos os teste com a ferramentas em duas partes:

- Comparação de um códigos para validar emails com expressão regular e o outro sem expressão regular.
- Teste de utilização da ferramenta por 3 usuários, 1 usuário possuia conhecimentos de expressão regular e 2 usuarios sem conhecimentos.

4.1 Validar e-mails em C

As funções a seguir são usados para validar emails na linguagem C #:

- Função sem expressão regular:

```
public static bool ValidarEmail(string email)
{
bool validEmail = false;
int indexArr = email.IndexOf('@');
if (indexArr > 0)
{
int indexDot = email.IndexOf('.', indexArr);
if (indexDot - 1 > indexArr)
{
if (indexDot + 1 < email.Length)
{
string indexDot2 = email.Substring(indexDot + 1, 1);
if (indexDot2 != ".")
{

validEmail = true;

}
}
}
}
return validEmail;
}
```

- Função com expressão regular:

```
string email = txtEmail.Text;
Regex rg = new Regex (@"^[A-Za-z0-9](([_ \.\]?[a-zA-Z0-9]+)*)@([A-Za-z0-9]+) (([\.\-]?[a-zA-Z0-9]+)*)\.([A-Za-z]{2,})$");
if (rg.IsMatch( email ) )
{
Response.Write("Email Valido!");
}
else
{
Response.Write("Email Inválido!");
}
```

Referências

[1]JARGAS, Aurélio Marinho. Expressões Regulares - Uma abordagem divertida. 3 ed. Novatec, 2009.