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

Na área de Ciência da Computação expressão regular tem sido uma técnica eficiente para identificar caracteres particulares ou palavras e até mesmo um subgrupo de caracteres. Nem sempre de compreensão fácil, as expressões regulares são uma forte ferramenta para o profissional que as manipulam podendo reduzir linhas de códigos e reduzir o tempo gasto no desenvolvimento. Expressões regulares são escritas numa linguagem formal sendo interpretada por um processador de expressão regular, este vai analisar o texto e identificar as partes que casam com a expressão fornecida. O objetivo deste artigo é propõe uma ferramenta para o desenvolvimento de um sistema que constrói uma expressão regular básica para uma pessoa que não tenha intimidade ou desconheça esta técnica.

Abstract:

TEXTO EM INGLES DO AZULAY....

1 Introdução

A história das expressões regulares (também conhecidas como ReGex ou ER) originou-se das ideias do matemático Stephen Cole Kleene, o responsável por integrar as expressões regulares e a teoria da computação fazendo uso de autômato finitos.

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 códigos. Atualmente são usadas para buscar ou substituir, dentro de uma string, uma cadeia de caracteres especifica, mas também pode ser utilizado para buscar e validar data, horário, número IP, endereço de e-mail, etc [1] sendo usados não apenas por programadores, mas também por qualquer profissional que trabalhe diretamente com textos. Atualmente o numero de ferramentas que possibilitam o uso de expressões regulares é cada fez maior entre as ferramentas mais conhecidas podemos citar BrOffice, NotePad++, GoogleDocs , Ed, vi e vim.

O objetivo deste artigo é mostrar a elaboração de um sistema, na linguagem PHP, que auxilie na criação de ER para iniciantes e pessoas que não conhecem o assunto mas desejam otimizar seu trabalho. A seção 2 descreve os conhecimentos utilizados para a criação de uma ER utilizando exemplos simples apenas para entendimento de uma expressão regular, tratasse de um assunto extenso mais extremamente necessário para a elaboração do sistema, que será a base de conhecimento utilizado na criação do software. A seção 3 descreve a elaboração do RegEx Build Assistent.A seção 4 mostra uma comparação de algoritmos tradicionais, nas linguagens C # e javascript, e algoritmos utilizando expressões regulares e tambem a experiencias de usuários com o sistema.

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 usar as expressões regulares para procurar textos específicos, partes de textos, textos no inicio ou no fim de uma linha ou procurar textos com uma seqüência de especifica de caractere.

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 o qual chamamos de metacaracteres. Cada metacaracter tem a sua função específica no momento de casar uma ER. Os metacaracteres, de acordo com a função, os que estudaremos estão divididos em:

- Metacaracter tipo Representante;
- Metacaracter tipo Quantificador; e
- Metacaracter tipo Ancora.

Existem outros tipos de metacaracteres, os estendido, mas somente esses três tipos de metacaracteres são suficientes para escrevermos uma expressão regular básica e é suficiente para a implementação do software. As sintaxes dos metacacteres também podem variar de um programa para o outro.

2.1 Metacaracter tipo Representante

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

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.
Exemplo:

Procurar em texto qualquer caracter que esteja entre as duplas.
Solução: "."
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, a string que está sendo procurada somente poderá conter os argumentos que estiverem dentro dos colchetes. Ela torna-se mais eficiente que o ponto, pois ela não aceita qualquer argumento, somente os especificados dentro dos colchetes. Exemplos:

Podemos utilizar esse recurso para encontrar palavras com uma possível grafia errada. Na ER n[ãa]o procura pela string não ou nao. Conseguimos inclusive procurar caracteres escritos maiúsculos e minúsculos, na ER [Nn]ão podemos encontrar as strings Não e não.

Se necessitarmos procurar um valor numérico entre 1 e 9 o correto é usar a ER [1-9] para representar o intervalo numérico pois e bem mais cômodo digitar [1-9] do que [123456789]. Podemos representar o intervalo do alfabeto somente maiúsculo [A-Z] ou somente minúsculo [a-z] isso não inclui as vogais acentuadas, Ç e o 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 e não vai representar uma lista de qualquer coisa (existe realmente uma maneira de representar isso que será visto mais adiante). Para representar o -"devemos colocá-lo como ultimo item da lista ou seja fora do seu lugar padrão.

Nosso idioma necessita de caracteres acentuados em muitas palavras por isso fazemos uso de caracteres especiais chamados de caracteres POSIX. Na lista [A-Za-z] não estão inclusos os caracteres acentuados para isso utilizamos a classe [:upper:] (os parênteses fazem parte da classe), que inclui os caracteres acentuados bem como o Ç e o ç , nessa situação somente os maiúsculos segue uma tabela que mostra a classe POSIX uma lista equivalente e o seu significado.

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 significado é inverso. Quando usamos a lista negada queremos dizer que está fora da lista é aceito e o que está dentro não, ou seja, na ER [^0-9] seria aceito qualquer caractere exceto número, mas sempre haverá um retorno. Podemos 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 Quantificadores

Os metacaracteres do tipo quantificador dizem quantas vezes um determinado caractere ou matacaracter pode aparecer na string. Esse tipo de metacaracter é dito gulosos, 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 palvra toda.Podemos, com os conceitos já vistos, implementar uma lista opcional como segue abaixo.

- festas? retorna festa ou festas pois o "s"foi marcado como opcional.
- 1581?8 retorna 15818 ou 1588 pois o "1"foi marcado como opcional.
- fala[r!]? retona falar, fala! e fala.

2.2.2 O Metacaracter Asterisco (*)

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

- 6*0 retorna 0, 60, 660, 6660, 66660, ..., 66666666666666660,...

- b[ip]* retorna b, bi, bii, bip, biipp, bpi, bppii.....

Podemos utilizar a combinação do . e o * obtendo dessa maneira a ER .* que irá 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 a string 0 sozinha não é retornada e que b sozinho também não:

- 6+0 retorna 60, 660, 666066660,.....,
- b[ip]+ retorna bip, biip, bipp, biiip....

2.2.4 O Metacaracter chaves ({})

O metacaractere 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 afirmação 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 define uma quantidade, ele marca uma posição especifica na linha. Por causa da função que exerce, o metacaracter não pode ser quantificado, 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 circunflexo. Exemplo:

- ^[0-9] indica que estamos procurando linhas que começam com 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 especificação 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 fazelo através de uma lista. No entanto a lista é mais adeguado para representar um conjunto de caracteres. Nesse caso a solução mais viável seria o uso do metacaracter escape.
Exemplo: Representar atraves 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 metacaractes, caracteres e subgrupos tornando seu uso essecial para simplificar uma expressão regular. Todo o conteudo que estiver entre parenteses será identificado como um grupo, e este deverá casar dentro da regex.
Exemplos:
(www)?google.com(ḃr) 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.
Exemplo:
Encontrar endereços de internet que sejam https:// ou estilo ftp:// [1].
(http://|ftp://)

2.5 Precedência entre metacaracteres

Os metacarcteres possuem uma precedência para melhor entendermos e determinamos uma RegEx.A seguir uma tabela que mostra a precedência entre caracteres baseada no livro de[1].

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.

2.6 O Software Regex Build Assistent

2.7 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.

2.7.1 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.