# Expressões Regulares no Perl 6

Breno G. de Oliveira
breno@rio.pm.org

# Antes de mais nada...

- Não sou especialista no assunto!
- Perl 6 é uma linguagem **em desenvolvimento**!

# O que mudou? *Até o nome mudou !!!*

Melhor perguntar o que NÃO mudou ;-)

- Capturando: **(...)**
- Quantificadores de repetição: *****, **+** e **?**
- Alternativas: **|**
- Escapando: ****
- Sufixos de busca mínima: **??**, ***?** e **+?**

# Variáveis

*$/* agora contém toda a string encontrada, e /(seus) (ítens) (individuais)/ podem ser encontrados em *$/[0]*, *$/[1]*. *$/[2]*, ... ou em *$0*, *$1*, *$2*, ...

# Variáveis

*$/* agora contém toda a string encontrada, e /(seus) (ítens) (individuais)/ podem ser encontrados em *$/[0]*, *$/[1]*. *$/[2]*, ... ou em *$0*, *$1*, *$2*, ...

## Mas e as variáveis $0 e $/ ???

- O antigo *$0*, que indicava o nome do arquivo contendo seu programa, agora virou *$*PROGRAM_NAME*

- O antigo *$/*, que indicava o(s) caractere(s) separadores de entrada (\n, \r\n, etc), está com seu futuro incerto.

# Modificadores

- Sintaxe extendida “/x” (ignora espaços e permite comentários) virou padrão

- “/s” (deixa “.” ser o caractere de nova linha) não existe mais: o “.” agora encontra *qualquer* caractere, incluindo o de nova linha.

→ *Para usar o “.” como antigamente, use “\N” (tudo EXCETO nova linha)*

- “/m” (deixar que “^” e “$” sejam encontrados ao lado de um caractere de nova linha) também não existe mais: agora “^” e “$” encontram sempre o início/final da string, como o “\A” (início do buffer), “\z” (fim do buffer) e “\Z” (“\n*\z”) (que também não existem mais, à propósito ;-)

→ *Isso significa que “$” não acha “\n”, então devemos procurar por “\n?$” se for o caso, ou por “^^” e “$$”*

- Não existe mais o “/s” (modificador de eval em substituições)

→ *E agora? Use s/padrão/ { codigo() } /*

# Modificadores

- Modificadores agora são colocados no início das buscas/substituições, sempre precedidos por “:” (dois pontos)

```
➔ m:g:i/\s* (\w*) \s* ,?/;
```

- Modificadores também têm versões longas

```
➔ :g          :global
➔ :i          :ignorecase
```

- Modificadores “:p” (:pos) e “:c” (:continue) analisam na posição atual da string e a partir dela, respectivamente.

- *Como :p está implicitamente preso à posição do padrão na string, é ideal para a construção de analisadores lexicos e parsers.*

# Modificadores

- Modificador ":ov" (:overlap) encontra as possíveis posições, inclusive as que se sobrepõem e retorna na forma de lista. ":ex" (:exaustive) varre absolutamente todas as opções da string, independente de sua posição de início).

```
$str = "abracadabra";

$str ~~ m:overlap/ a (.*) a /;  #  bracadabr cadabr dabr br

$str ~~ m:exaustive/ a (.*) a/; #  br brac bracad bracadabr c
                                #  cad cadabr d dabr br
```

- Vários outros modificadores:

```
:words   :bytes   :graphs   :langs
```

- Será permitido até criar seus próprios modificadores!

# Racionalização dos agrupadores (), [], {}

(...) ainda delimita grupo capturado, mas... a ordem dos grupos agora é hierárquica, e não linear.

- Em outras palavras **/ (.. (..) ..) (..) (..(...(..)...)..(..) ..) /** vira
  **$0**, **$0[0]**, **$1**, **$2**, **$2[0]**, **$2[0][0]** e **$2[1]**

[...] não define mais um grupo de caracteres, e sim delimita um grupo que não deve ser capturado em variáveis.

- Use <[...]> para definir grupos de caracteres

{...} não é mais um quantificador de repetições. Eles agora são utilizados para embutir código caso um trecho da ER seja encontrado.

- Use **{...} para quantificar repetições

Foi adicionada também a chamada “fail”, para testar condições dentro da ER usando {...}

- **/ (\d+) { $0 < 256 or fail } /**

# Interpolação de Variáveis

- No Perl 6, as variáveis não são interpoladas, e sim passadas “cruas” para o sistema de regras, que então decide sobre como tratá-las.

- O tratamento padrão para variáveis é como um literal <'...'>, ou seja, não trata o conteúdo da variável como um subpadrão.
  - Em outras palavras, em Perl 6 a ER  / $var / é igual à / \Q$var\E /  no Perl 5

- Já as listas são tratadas como uma alternação de seus elementos.
  - Ou seja: / @lista /  é tratado como / [ @lista[0] | @lista[1] | @lista[2] | ... /

- Por fim, hashes interpolados buscam sempre a maior chave possível, devolvendo-a como um literal.
  - Dependendo do valor apontado pela chave, este pode ser executado, processado como subregra ou simplesmente ignorado

# Metasintaxe extensível

- Dentro de <...> definimos nossa própria estrutura de padrões
- Parâmetros especiais incluem:

```
/ <before padrão> /      # equivalente ao falecido  /(?=padrão)/
/ <after padrão> /       # equivalente ao falecido /?<padrão/
```

- <?...> não armazena o que for encontrado
- <$...> indica regra indireta (regra dentro de variável, etc)
- <&...> é equivalente a <{ ... }>
- <'...'> e <"..."> fazem busca literal por *'...'* ou *"..."*. No caso das aspas duplas, se houver dentro dela uma *$variável*, seu valor é interpolado e buscado como literal (sem ser interpretado)

# Reformulação dos escapes

- Propriedades “\p” e “\P” viraram regras intrinsecas da gramática ( à saber: “<prop ...>” e “<!prop ...>” )

- As sequências “\L...\E”, “\U...\E” e “\Q...\E” não existem mais.
  - Nos raros casos em que precisar delas, use “<{ lc $regra }>”, etc...

- Referências “\1”, “\2”, etc. não existem mais.
  - Use $0, $1, etc., já que as variáveis não são mais interpoladas

- Novas sequências de escape: “\h” e “\v” buscam por caracteres de espaço horizontais e verticais, respectivamente (inclusive Unicode)

- “\s” agora busca por qualquer caractere Unicode de espaço

- “\G” não existe mais – use “:p” no lugar

# Nada é ilegal

- O padrão *nulo* (“//”) agora é ilegal.

- Para buscar por o que quer que a busca anterior tenha procurado, use *<prior>:*

→ */<prior>/*

- Para buscar por strings de comprimento nulo, use <null>:

→ */<null>/*
→ *split /<?null>/, $string;  # o “?” antes da regra diz pra não guardar o valor encontrado*

# E a transliteração?

O tr/// agora virou trans(), cujo argumento é uma lista de pares

```
$str.=trans( 'a..e' => 'A..E', 'xyz' => 'XYZ');
```

```
$str.=trans( 'A' => 'a', 'B' => 'b', 'C' => 'c');
```

```
$str.=trans( [' ' , '<', '>', '&'] =>
             [ ' ', '&lt;', '&gt;', '&amp;' ]
           );
```

# Oh! E agora, quem poderá nos ajudar ???

- Calma, calma... o modificador “:perl5” permite que você use a sintaxe antiga para suas expressões regulares! (só não vale botar os modificadores no final ;-)

→ *m:perl5/(?mi)^[a-z]{1,2}(?=\s)/*

# Conclusão?

- Sistema de Expressões Regulares completamente redesenhado
- Sintaxe muito diferente das ER's do Perl 5
- Muito mais poder (ficou mais fácil dominar o mundo) !
- Agora eu entendo *menos ainda* sobre ER's ;-)

# Para saber mais...

- **Apocalipse 5**
  (http://dev.perl.org/perl6/doc/design/apo/A05.html)

- **Synopsis 5**
  (http://dev.perl.org/perl6/doc/design/syn/S05.html)

- **Exegesis 5**
  (http://dev.perl.org/perl6/doc/design/exe/E05.html)

- **Pugs: Implementação do Perl 6 em Haskell**
  (http://www.pugscode.org)

# Obrigado!

# Duvidas?