EXPRESSÕES REGULARES

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

29 de abril de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

CALENDÁRIO

Provas

- prova 1 dia 18 de maio
- prova 2 dia 22 de junho
- sub fechada (a ser marcada)
- recuperação dia 20 de julho
- média das provas = $P_1 + 2P_2 \ / \ 3$
- média final = 0.7 * média das provas + 0.3 * média dos trabalhos

```
      Abril                 Maio
Do Se Te Qu Qu Se Sá  Do Se Te Qu Qu Se Sá
                1  2   1  2  3  4  5  6  7
 3  4  5  6  7  8  9   8  9 10 11 12 13 14
10 11 12 13 14 15 16  15 16 17 18 19 20 21
17 18 19 20 21 22 23  22 23 24 25 26 27 28
24 25 26 27 28 29 30  29 30 31

      Junho                 Julho
Do Se Te Qu Qu Se Sá  Do Se Te Qu Qu Se Sá
          1  2  3  4                 1  2
 5  6  7  8  9 10 11   3  4  5  6  7  8  9
12 13 14 15 16 17 18  10 11 12 13 14 15 16
19 20 21 22 23 24 25  17 18 19 20 21 22 23
26 27 28 29 30        2 4 25 26 27 28 29 30
31
```

LISTAGEM DE DIRETÓRIOS

```
Path dir = ...;
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file: stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    System.err.println(x);
}
```

LISTAGEM DE DIRETÓRIOS

```
Path dir = ...;
try (DirectoryStream<Path> stream = Files.newDirectoryStream(dir)) {
    for (Path file: stream) {
        System.out.println(file.getFileName());
    }
} catch (IOException | DirectoryIteratorException x) {
    System.err.println(x);
}
```

Filtragem usando um glob

```
Path dir = ...;
try (DirectoryStream<Path> stream =
     Files.newDirectoryStream(dir, "*.{java,class,jar}")) {
    for (Path entry: stream) {
        System.out.println(entry.getFileName());
    }
} catch (IOException x) {
    System.err.println(x);
}
```

PERCORRENDO UMA ÁRVORE DE DIRETÓRIOS

Para percorrer recursivamente um diretório você deve implementar uma classe com a interface `FileVisitor` (ou estender de `SimpleFileVisitor`)

A interface `FileVisitor`

preVisitDirectory chamado antes das entradas de um diretório serem visitadas

postVisitDirectory chamado depois que todas as entradas forem visitadas. Se um erro ocorrer, será passado para o método

visitFile chamado quando um arquivo for visitado; os `BasicFileAttributes` também são passados

visitFileFailed chamado quando um arquivo não pôde ser acessado

EXEMPLO

```
import static java.nio.file.FileVisitResult.*;

public static class PrintFiles
    extends SimpleFileVisitor<Path> {

    // Imprime informação sobre o tipo de arquivo
    @Override
    public FileVisitResult visitFile(Path file,
                                     BasicFileAttributes attr) {
        if (attr.isSymbolicLink()) {
            System.out.format("Symbolic link: %s ", file);
        } else if (attr.isRegularFile()) {
            System.out.format("Regular file: %s ", file);
        } else {
            System.out.format("Other: %s ", file);
        }
        System.out.println("(" + attr.size() + "bytes)");
        return CONTINUE;
    }
```

EXEMPLO

```
    // Imprime cada diretório visitado
    @Override
    public FileVisitResult postVisitDirectory(Path dir,
                                              IOException exc) {
        System.out.format("Directory: %s%n", dir);
        return CONTINUE;
    }

    // Se um erro ocorrer durante o acesso a um arquivo,
    // mostre pro usuário.
    // Se você não sobrescrever esse método e um erro
    // ocorrer, um IOException será lançado
    @Override
    public FileVisitResult visitFileFailed(Path file,
                                           IOException exc) {
        System.err.println(exc);
        return CONTINUE;
    }
}
```

Para iniciar o processo

```
Path startingDir = ...;
PrintFiles pf = new PrintFiles();
Files.walkFileTree(startingDir, pf);

// para controlar como o diretório será percorrido é possível
// passar opções definidas no enum FileVisitOption e definir o
// limite do número de níveis visitados
Path startingDir = ...;
EnumSet<FileVisitOption> opts = EnumSet.of(FOLLOW_LINKS);
Finder finder = new Finder(pattern);
Files.walkFileTree(startingDir, opts, Integer.MAX_VALUE, finder);
```

Para controlar o processo

- CONTINUE
- TERMINATE
- SKIP_SUBTREE
- SKIP_SIBLINGS

API DE E/S LEGADA

Em Java ≤ 7, a principal classe de acesso a arquivos era a `java.io.File`, mas ela tinha alguns problemas:

- muitos métodos não lançavam exceções quando falhavam; era impossível determinar a causa exata do erro
- o método `rename` não era multiplataforma
- não havia a noção de links simbólicos
- tinha acesso restrito e ineficiente aos metadados de permissões, dono do arquivo, etc.
- desempenho ruim em diretórios com muitos arquivos

Métodos úteis

- `File.toPath()`
- `Path.toFile()`

EXPRESSÕES REGULARES

INTRODUÇÃO

- *Expressões regulares* são uma forma de descrever um conjunto de strings que possuem uma mesma característica
- podem ser usadas para buscar, editar ou manipular strings
- possuem uma sintaxe própria, usada em diferentes linguagens de programação e (Perl, Tcl, Python, awk, etc.) programas (grep, sed, emacs, vim, etc.)
- implementado em Java pelo pacote `java.util.regex`

O pacote `java.util.regex` é consiste principalmente dessas três classes principais:

Pattern representação (compilada) de uma expressão regular

Matcher é a classe que interpreta um padrão e procura em um conjunto de strings dados como entrada por casamentos desse padrão

PatternSyntaxException exceção não verificada que indica um erro de sintaxe em uma expressão regulares

EXEMPLO DE USO

```
while (true) {
    Pattern pattern =
    Pattern.compile(console.readLine("%nEnter your regex: "));

    Matcher matcher =
    pattern.matcher(console.readLine("String to search: "));

    boolean found = false;
    while (matcher.find()) {
        console.format("I found the text" +
            " \"%s\" starting at " +
            "index %d and ending at index %d.%n",
            matcher.group(),
            matcher.start(),
            matcher.end());
        found = true;
    }
    if(!found){
        console.format("No match found.%n");
    }
}
```

Busca por `foo` em “foo”

```
Enter your regex: foo
Enter input string to search: foo
I found the text "foo" starting at index 0 and ending at index 3.
```

Busca por `foo` em “foo”

```
Enter your regex: foo
Enter input string to search: foo
I found the text "foo" starting at index 0 and ending at index 3.
```

Busca por `foo` em “foofoofoo”

```
Enter your regex: foo
Enter input string to search: foofoofoo
I found the text "foo" starting at index 0 and ending at index 3.
I found the text "foo" starting at index 3 and ending at index 6.
I found the text "foo" starting at index 6 and ending at index 9.
```

- alguns caracteres especiais podem mudar como o casamento de um padrão é feito
- `< ( [ { \^- =; $ ! | ] } ) ? * + . >`

Exemplo

```
Enter your regex: cat.
Enter input string to search: cats
I found the text "cats" starting at index 0 and ending at index 4.
```

CLASSES DE CARACTERES

Construção	Descrição
[abc]	a, b, or c (uma classe simples)
[^abc]	Qualquer caracter exceto a, b ou c (negação)
[a-zA-Z]	a até z ou A até Z, inclusive (intervalo)
[a-d[m-p]]	a até d ou m até p: [a-dm-p] (união)
[a-z&&[def]]	d, e ou f (intersecção)
[a-z&&[^bc]]	a até z, exceto para b e c: [ad-z] (subtração)
[a-z&&[^m-p]]	a até z mas não m até p: [a-lq-z] (subtração)

Exemplo

```
Enter your regex: [bcr]at
Enter input string to search: bat
```

Exemplo

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
```

Exemplo

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
I found the text "cat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: rat
```

Exemplo

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
I found the text "cat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: rat
I found the text "rat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: hat
```

Exemplo

```
Enter your regex: [bcr]at
Enter input string to search: bat
I found the text "bat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: cat
I found the text "cat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: rat
I found the text "rat" starting at index 0 and ending at index 3.

Enter your regex: [bcr]at
Enter input string to search: hat
No match found.
```

CARACTERES PREDEFINIDOS

Construção	Descrição
.	Qualquer caracter (quebras de linha podem ou não serem casadas)
\d	Um dígito: [0-9]
\D	Qq coisa menos dígito: [^0-9]
\s	Um caractere de espaço em branco: [\t \n \x0B \f \r]
\S	Qq coisa menos caractere em branco: [^\s]
\w	Um caractere: [a-zA-Z_0-9]
\W	Qq coisa menos caractere: [^\w]

Obs:

No código é necessário usar um caractere de escape:

```
private final String REGEX = "\\d"; // um único dígito
```

EXEMPLOS

```
Enter your regex: .
Enter input string to search: @
I found the text "@" starting at index 0 and ending at index 1.

Enter your regex: .
Enter input string to search: 1
I found the text "1" starting at index 0 and ending at index 1.

Enter your regex: \S
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \w
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.

Enter your regex: \w
Enter input string to search: !
No match found.
```

QUANTIFICADORES

Guloso	Relutante	Possessivo[1]	Significado
X?	X??	X?+	X, uma única vez ou nenhuma
X*	X*?	X*+	X, zero ou mais vezes
X+	X+?	X++	X, uma ou mais vezes
X{n}	X{n}?	X{n}+	X, exatamente n vezes
X{n,}	X{n,}?	X{n,}+	X, ao menos n vezes
X{n,m}	X{n,m}?	X{n,m}+	X, ao menos n e não mais do que m

[1] Mais sobre a diferença entre as colunas em um minuto.

EXEMPLOS — BUSCA EM STRINGS VAZIAS

```
Enter your regex: a?
Enter input string to search:
```

EXEMPLOS — BUSCA EM STRINGS VAZIAS

```
Enter your regex: a?
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a*
Enter input string to search:
```

EXEMPLOS — BUSCA EM STRINGS VAZIAS

```
Enter your regex: a?
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a*
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a+
Enter input string to search:
```

EXEMPLOS — BUSCA EM STRINGS VAZIAS

```
Enter your regex: a?
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a*
Enter input string to search:
I found the text "" starting at index 0 and ending at index 0.

Enter your regex: a+
Enter input string to search:
No match found.
```

EXEMPLOS — RESULTADOS VAZIOS

```
Enter your regex: a?
Enter input string to search: a
```

EXEMPLOS — RESULTADOS VAZIOS

```
Enter your regex: a?
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a*
Enter input string to search: a
```

EXEMPLOS — RESULTADOS VAZIOS

```
Enter your regex: a?
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a*
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a+
Enter input string to search: a
```

EXEMPLOS — RESULTADOS VAZIOS

```
Enter your regex: a?
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a*
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
I found the text "" starting at index 1 and ending at index 1.

Enter your regex: a+
Enter input string to search: a
I found the text "a" starting at index 0 and ending at index 1.
```

EXEMPLOS — RESULTADOS VAZIOS

E se ao invés de “a” tivessemos como entrada “aaaaa”?

```
Enter your regex: a?
Enter input string to search: aaaaa
I found the text "a" starting at index 0 and ending at index 1.
I found the text "a" starting at index 1 and ending at index 2.
I found the text "a" starting at index 2 and ending at index 3.
I found the text "a" starting at index 3 and ending at index 4.
I found the text "a" starting at index 4 and ending at index 5.
I found the text "" starting at index 5 and ending at index 5.

Enter your regex: a*
Enter input string to search: aaaaa
I found the text "aaaaa" starting at index 0 and ending at index 5.
I found the text "" starting at index 5 and ending at index 5.

Enter your regex: a+
Enter input string to search: aaaaa
I found the text "aaaaa" starting at index 0 and ending at index 5.
```

GRUPOS DE CAPTURA E CLASSES DE CARACTERES

```
Enter your regex: (dog)3
Enter input string to search: dogdogdogdogdogdog
I found the text "dogdogdog" starting at index 0 and ending at index 9
I found the text "dogdogdog" starting at index 9 and ending at index 1

Enter your regex: [abc]3
Enter input string to search: abccabaaaccbbbc
I found the text "abc" starting at index 0 and ending at index 3.
I found the text "cab" starting at index 3 and ending at index 6.
I found the text "aaa" starting at index 6 and ending at index 9.
I found the text "ccb" starting at index 9 and ending at index 12.
I found the text "bbc" starting at index 12 and ending at index 15.
```

DIFERENÇA ENTRE CASAMENTO GULOSO, RELUTANTE E POSSESSIVO

A diferença é sutil e se refere a como os caracteres de uma string são consumidos durante a comparação feita pelo `Matcher`

Guloso força o `Matcher` a consumir toda a string antes de testar o padrão. Caso não case, descarta o último caractere e tenta novamente

Relutante consome a string caractere a caractere procurando por um casamento, só testar a string inteira no final

Possessivo sempre consome a string inteira e testa uma única vez se há casamento

EXEMPLO

```
Enter your regex: .*foo  // greedy quantifier
Enter input string to search: xfooxxxxxxfoo
I found the text "xfooxxxxxxfoo" starting at 0 and ending at 13.

Enter your regex: .*?foo  // reluctant quantifier
Enter input string to search: xfooxxxxxxfoo
I found the text "xfoo" starting at index 0 and ending at index 4.
I found the text "xxxxxxfoo" starting at index 4 and ending at 13.

Enter your regex: .*+foo // possessive quantifier
Enter input string to search: xfooxxxxxxfoo
No match found.
```

1. o `.*` consome a string toda e não sobra caractere pra casar com “foo”. Só quando os três últimos caracteres forem “regurgitados” é que encontra um casamento
2. primeiro o `Matcher` consome “” (não casa), depois consome “x” (1º casamento) e assim por diante
3. a string inteira é consumida pelo `.*+`, não deixando nada pra casar com “foo”. O casamento falha e o `Matcher` termina

GRUPOS DE CAPTURA

- podemos organizar o resultado de um casamento em subexpressões entre parênteses chamadas *grupos*
- grupos são contados pelo número de abre parênteses da esquerda p/ direita
- um mesmo padrão pode ter vários grupos
- ex: o padrão `((A)(B(C)))` tem 4 grupos:

1. `((A)(B(C)))`
2. `(A)`
3. `(B(C))`
4. `(C)`

REFERÊNCIAS A GRUPOS

- os valores casados nos grupos de captura são armazenados na memória e podem ser utilizados dentro do mesmo padrão
- uma referência é especificada por um “ \” seguido do número do grupo a ser referenciado

Exemplo:

```
Enter your regex: (\d\d)\1
Enter input string to search: 1212
```

REFERÊNCIAS A GRUPOS

- os valores casados nos grupos de captura são armazenados na memória e podem ser utilizados dentro do mesmo padrão
- uma referência é especificada por um “ \” seguido do número do grupo a ser referenciado

Exemplo:

```
Enter your regex: (\d\d)\1
Enter input string to search: 1212
I found the text "1212" starting at index 0 and ending a

Enter your regex: (\d\d)\1
Enter input string to search: 1234
```

REFERÊNCIAS A GRUPOS

- os valores casados nos grupos de captura são armazenados na memória e podem ser utilizados dentro do mesmo padrão
- uma referência é especificada por um “ \” seguido do número do grupo a ser referenciado

Exemplo:

```
Enter your regex: (\d\d)\1
Enter input string to search: 1212
I found the text "1212" starting at index 0 and ending a

Enter your regex: (\d\d)\1
Enter input string to search: 1234
No match found.
```

LIMITES

Padrões de limites permitem especificar onde você quer que o padrão apareça na string de entrada.

Limitante	Descrição
^	Início de uma linha
$	Fim de uma linha
\b	Um separador de palavras
\B	Negação de \b
\A	Início da entrada
\G	O final do casamento anterior
\z	O fim da entrada
\Z	\z ou posição logo antes de uma quebra de linha em \z

EXEMPLOS

```
Enter your regex: ^dog$
Enter input string to search: dog
I found the text "dog" starting at index 0 and ending at index 3.

Enter your regex: ^dog$
Enter input string to search:       dog
No match found.

Enter your regex: \bdog\b
Enter input string to search: The dog plays in the yard.
I found the text "dog" starting at index 4 and ending at index 7.

Enter your regex: \bdog\b
Enter input string to search: The doggie plays in the yard.
No match found.
```

BIBLIOGRAFIA

- The Java™ Tutorials – Basic I/O: https://docs.oracle.com/javase/tutorial/essential/io/
- The Java™ Tutorials – Regular Expressions: https://docs.oracle.com/javase/tutorial/essential/regex/