正则表达式快速参考手册

胡志飞

<WisdomFusion[at]gmail[dot]com>

2012 年之旧文重拾于 2016 年 2 月 26 日
version 0.5.0

目　录

1 简介 **1**

2 基本语法 **2**

3 高级语法 **10**

4 举些例子吧！ **13**

5 正则表达式“流派” **15**

5.1 正则表达式流派 15

5.2 正则表达式引擎 15

6 应用场景 **18**

6.1 支持正则表达式的工具们 18

6.1.1 RegexBuddy 18

6.1.2 PowerGREP 20

6.1.3 在线正则表达式测试工具 22

6.1.4 grep 22

6.1.5 Vim 22

6.1.6 GNU Emacs 23

6.1.7 Adobe Dreamweaver 表格处理 25

6.1.8 UltraEdit 等编辑器 25

6.1.9 GREP for Adobe InDesign 25

6.2 猿类们经常用到的正则表达式 26

6.2.1 Perl 正则表达式王国 26

6.2.2 Python 27

6.2.3 PHP PCRE 29

6.2.4 .NET Framework 正则表达式 30

6.2.5 JavaScript 30

6.2.6 sed & awk 30

6.2.7 VBA 中使用正则表达式 30

6.2.8 More and more 32

6.3 Changelog 32

6.4 Disclaimer 32

1 简介 INTRODUCTION

文字处理无处不在无时不有，日常工作和学习大多数任务都和文字息息相关，编辑们写文章、整理资料，开发人员编码、处理用户提交的数据或请求接口数据，等等，这些都是以字符和字符串相关的任务，既然如些，掌握一个快速文字处理的方法就变得很有必要。

正则表达式，（Regular Expression，在代码中常简写为 regex、regexp 或 RE），计算机科学的一个概念。正则表达式使用字符来描述、匹配一系列符合某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。许多程序设计语言都支持利用正则表达式进行字符串操作。例如，在 Perl[①]中就内建了一个功能强大的正则表达式引擎。正则表达式这个概念最初是由 Unix 中的工具软件（例如 sed[②]和 grep[③]）普及开的。

需要注意的是，用什么工具，用什么编辑语言，正则表达式的语法有些差别，特性的支持也参差不齐，称之为正则表达式“流派”(第5部分详述)，所以要单独参考工具和编程语言本身的文档才行。本文档旨在给大家一个通用的、概括的正则表达式宏观印象，辅以实例和应用案例，同时针对个别常用但又不易理解的特性，给大家作详细说明和总结，抛砖引玉。

期望本文档能给大家一个快速的参考，快速的掌握正则表达式这个棒棒哒效率工具，让大家平时工作学习中更加得心应手！☺

说明

我对排版及专业出版知之甚微，只是在平时笔记和文档时，哪怕是自己写的太乱的话也不乐意翻看，印象笔记里的东东又过于零碎，故把旧文完善并整排[a]。然而，专业领域知识因涉猎过多而不精，难保周全和准确，但只要在自己知识圈内，我会劲力完成尽可能规范和可靠的文档呈现给大家，并不断完善更新，请批评指正，共同提高。

另外，因为文档中有大量图表，表格的宽度要求比较高，所以直接采用 Lanscape 横向布局，对于阅读不是最佳设置，但为了牵就“大表哥”目前折中一下这些处理。

[a]本文档 2012 年编写，现整拾，并加以完善。目前觉得 Markdown, org-mode适用于文本文档的快速编写，而 Adobe InDesign 和 LaTeX 适合更专业的文档和书籍的图文混排及设计。

[①]Perl被称为“实用报表提取语言”（Practical Extraction and Report Language），正则表达式特性的推动者，文本处理非常方便。

[②]sed是一种 UNIX/Linux 平台下的轻量级流编辑器，日常一般用于处理文本文件。

[③]grep，global search regular expression and print out the line，是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。

2 基本语法 Basic Syntax

语法部分结合了自己的理解和对正则表达式应用的一些心得，分类有不当之处，请指正。因为是总结性的参考文档，所以这里使用“大表哥”形式展示，列出语法的同时，关键语法举了几个栗子加强理解。

特性	语法	描述	举个栗子
字符	除 [\^$.|?*+() 以外的任意字符	除了 [\^$.|?*+() 以外的任意字符，{和} 也是文字文本，除了下面说到的成对出现的量词语法，如 {n} 和 {m,n} 等。	a 匹配 about 中的 a
	字符转义	\t , \? , * , \+ , \. , \\| , \{ , \} , \\ , \[, \] , \(, \)	\+ 匹配 + ； \?\- 匹配 ?-
	\n , \r 和 \t	Windows 文件格式换行符是\r\n，UNIX 文件格式换行符是\n，\t 匹配水平制表符	
	\cA 到 \cZ , \ca 到 \cz	Ctrl + A 到 Ctrl + Z ，与 ASCII 字符 \x01 到 \x1A 等价	
	\a , \e , \f , \v	依次为警报（\x07）、Esc 字符 (\x1B)、进纸符（\x0C）和垂直制表符（\x0B）	
	\Q ... \E	文字文本范围，被包含在 \Q 和 \E 之间的文字，都被视为普通文字，如 [\^$.|?*+(){} 也不再用转义了，这个最早是由 Perl 引入正则表达式的。	\Q+-*/\E 匹配的就是 +-*/
基本特性	. （点）	匹配除换行符之外的任意字符，有些正则表达式“流派”还支持点是否匹配换行符的开关。	. 匹配 about 中的任意一个字符

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
	x|y	条件分支，匹配 | 的左侧或右侧的字符串。使用条件分支特性时，要注意各个条件的顺序。	abc|def|xyz 匹配 abc 或 def 或 xyz
字符类	[...]	匹配字符类中列举的任意一个字符	[abc] 匹配 a 或 b 或 c [aeiou] 匹配任何一个英文元音字母 [.!?] 匹配 . 或 ! 或 ?
	[\^\]]	在字符类中，要匹配 ^-]\这几字符，得使用\转义	[\^\]] 匹配 ^ 或]
	[^...]	排除型字符类，^（脱字符，caret）紧跟 [之后，可以把字符类中列举的字符排除匹配范围，也就是所这个字符类将匹配任意一个不在列出字符范围内的字符	[^a-d] 匹配除了 a,b,c,d 之外的任意一个字符
	\d , \w , \s	\d 匹配数字，与 [0-9] 等价；\w 匹配任意一个字母或数字或下划线或汉字；\s 匹配任意一个空白符	[\d\s] 匹配一个数字或空白符
	\D , \W , \S	是 \d , \w 和 \s 的反义字符类。\D 匹配任意非数字的字符；\W 匹配任意不是字母、数字、下划线、汉字的字符；\S 匹配任意不是空白符的字符	\D 匹配任意非数字的字符
	[\b]	在字符类中，[\b] 为Backspace退格键字符	

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
POSIX	[:alnum:]	匹配所有大小写字母及数字	等价于 [0-9a-zA-Z]
	[:alpha:]	匹配所有大小写字母	等价于 [a-zA-Z]
	[:ascii:]	匹配所有 ASCII 字符，查看完整ASCII 字符列表	等价于 [\x01-\x7F]
	[:blank:]	匹配半角空格和制表符	等价于 [\t]
	[:cntrl:]	匹配所有 ASCII 0 到 31 之间的控制符	等价于 [\x01-\x1F]
	[:digit:]	匹配所有数字	等价于 [0-9]
	[:graph:]	匹配所有可打印的字符	
	[:lower:]	匹配所有小写字母	等价于 [a-z]
	[:print:]	匹配所有可打印字符和空格	
	[:punct:]	匹配所有标点符号	
	[:space:]	空白字符	等价于 [\t\n\r\f\v]
	[:upper:]	匹配所有大写字母	等价于 [A-Z]
	[:word:]	字母、数字和下划线	等价于 [a-zA-Z0-9_]
	[:xdigit:]	匹配所有十六进制字符	等价于 [0-9a-fA-F]

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
锚点	^	匹配字符串开始位置或行首位置	单行模式下 ^. 在 foo\nbar 中匹配 f；在多行模式下，同时还匹配换行后的 b
	$	匹配字符串结尾位置或行尾位置	.$ 在 foo\nbar 中匹配 r；在多行模式下，同时还匹配换行符前的 o
	\A	字符串开头位置（类似 ^，但不受处理多行选项的影响）	\Ae 在 example 这个字符串中匹配开头的 e
	\Z	字符串结尾位置或行尾位置（不受处理多行选项的影响）	e\Z 在 example 这个字符串中匹配结尾的 e
	\b	单词分界位置，单词开头或结尾	.\b 在字符串 abc 中匹配 c
	\B	匹配不是单词开头或结尾的位置	\B.\B 在字符串 abc 中匹配 b
	\<	单词开头	
	\>	单词结尾	

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
	\G	该锚点匹配上一个正则表达式匹配到的结尾处，通常用于实现“继续匹配”，该特性目前仅在 Perl 中得以实现，为了避免 \G 匹配失败后从头重新匹配重个文符串，需加 c 修饰符。	实例在后文中体现，见6.2.1
量词	?	前导字符重复零次或一次，贪婪的[①]：当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能多的字符。	abc? 匹配 abc 或 ab，如果可能，优先匹配前者
	??	前导字符重复零次或一次，非贪婪[②]：当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能少的字符。与贪婪相反。	abc?? 匹配 ab 或 abc
	*	前导字符重复零次或更多次，贪婪的	
	*?	前导字符重复零次或更多次，非贪婪	
	+	前导字符重复一次或更多次，贪婪的	

TO BE CONTINUED...

[①]贪婪模式：当正则表达式中包含能接受重复的限定符时，通常的行为是（在使整个表达式能得到匹配的前提下）匹配尽可能多的字符。考虑这个表达式：a.*b，它将会匹配最长的以 a 开始，以 b 结束的字符串。如果用它来搜索 aabab 的话，它会匹配整个字符串 aabab。这被称为贪婪匹配。

[②]相反，非贪婪，即匹配尽可能少的字符，只要在量词后面加上一个问号?。如 a.*?b 匹配最短的，以 a 开始，以 b 结束的字符串。如果把它应用于 aabab 的话，它会匹配 aab（第一到第三个字符）和 ab（第四到第五个字符）。

（续表）

特性	语法	描述	举个栗子
	+?	前导字符重复一次或更多次，非贪婪	
	{n}	前导字符重复 n 次	
	{n,m}	前导字符重复 n 到 m 次，其中 $n >= 0$，$m >= n$	
	{n,}	前导字符重复 n 次或更多次，其中 $n >= 0$	
	{,m}	前导字符最多重复 m 次，基中 $m >= 0$	
分组与反向引用	(regex)	匹配 regex，并捕获文本到自动命名的组里	(abc){3} 匹配 abcabcabc
	(?:regex)	匹配 regex，不捕获匹配的文本，也不给此分组分配组号	(?:abc){3} 匹配 abcabcabc，无分组
	\1 到 \9	反向引用，用于重复搜索前面某个分组匹配的文本。例如，\1 代表分组 1 匹配的文本。有些与此正则表达式流派支持多于 9 的分组。左括弧顺序即是分组序号的顺序。见图1。	(abc|def)=\1 匹配 abc=abc 或 def=def，而不是 abc=def 或 def=abc
	\10 到 \99	反向引用，分组 10 到 99	
	\g{1} 到 \g{99}	Perl 语法中，反向引用语法优化[①]	避免出现歧义，同时用负数分组还能倒序引用。
	\g{-1}, \g{-2}, ...	倒数第 1 个分组，倒数第 2 个分组，...	

TO BE CONTINUED...

[①]如果想实现类似 (.)\1 的效果，若紧跟的字符和分组号相同，如 (.)\111，这样正则表达式引擎就不知所措，如果用 \g{1}11 这种语法就不会有歧义，最重要的是这种语法或以用负数分组号倒序选用分组。

（续表）

特性	语法	描述	举个栗子
	(?<name>regex)	命令分组	命名分组的最大好处是反向引用时不用再怕弄错分组了。
	\k<name>	反向引用命令分组，Perl 中也可以使用\g{name}。	
替换表达式	\1 到 \9	替换表达式中使用反向引用，和匹配表达式中用法相同。	
	\`	正则表达式匹配部分之前的字符串，Perl 语言中也作 `${^PREMATCH}`	
	\&	正则表达式匹配的部分，有的流派使用 \0，Perl 语言中也作 `${^MATCH}`	
	\’	正则表达式匹配部分之后的字符串，Perl 语言中也作 `${^POSTMATCH}`	
	\L	后续替换字符转换为小写	
	\U	后续替换字符转换为大写	
	\E	关闭前面的 \L 和 \U	

关于正则表达式分组序号，这里有必要作一下解释，简单正则表达式，分组号很明了，但是一旦复杂起来，尤其是带有多层嵌套时，怎么解？这里有个小窍门，如图1所示，左括号决定分组序号，也就是说，不管有多复杂，只要盯着左括号，就能很快分辨出各个分组序号。

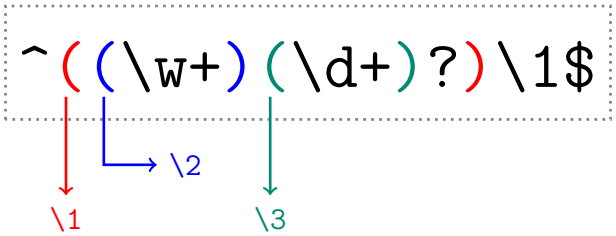

图 1: 正则表达式分组序号

3 高级语法 Advanced Syntax

之所以本文档中称这些正则为“高级语法”，一是有些不常用，二是有些语法不太好理解，故有此一说。

特性	语法	描述	举个栗子
模式修饰符	`(?i)`	打开忽略大小写模式，之后的模式不分大小写。模式修饰符有 i, s, m, x 四种，分别是忽略大小写（**I**gnoreCase）、单行模式（**S**ingleline）、多行模式（**M**ultiline）和注释模式	`(?i)te(?-i)st` 匹配 `TEst`，而不匹配 `TEST`
	`(?-i)`	关闭忽略大小写模式，之后的模式不分大小写	
	`(?s)`	打开单行模式，之后的模式不支持多行	默认情况下 `.` 是不匹配换行的，打开该模式后，待匹配的字符将为视为“一行”。
	`(?-s)`	关闭单行模式，之后的模式支持多行	
	`(?m)`	打开多行模式，之后的模式支持多行	
	`(?-m)`	`^` 和 `$` 匹配行首和行尾	
	`(?x)`	打开宽松和注释模式	打开该模式后，可以在正则表达式中插和空白和换行，使正则表达式可读性增强。
	`(?-x)`	关闭宽松和注释模式	
	`(?i-sm)`	打开 i 和 m 模式，关闭 s 模式	以上几种模式可以组合使用
	`(?i-sm:regex)`	在 `(?i-sm:regex)` 子模式内打开 i 和 m 模式，关闭 s 模式	

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
注释	(?#comment)	注释	
环视	(?=Regex)	肯定前瞻（Positive Lookahead） 子表达式能够（肯定）匹配右侧（前瞻）文本，该断言检查的文本和行文方向一致，故称之为**前瞻** **前瞻**（Lookahead）和**后顾**（Lookbehind）统称为**环视**（Lookaround），属于**零宽断言**（Zero-Length Assertions）	\b\w+(?=ing\b)，匹配以 ing 结尾的单词的前面部分，不含 ing，零宽断言仅用来判断前方或后方有没有某些字符，而不包含在匹配结果中
	(?!Regex)	否定前瞻（Negative Lookahead） 子表达式不能（否定）匹配右侧（前瞻）文本	\d{3}(?!\d) 匹配三位数字，而且这三位数字的后面不能是数字
	(?<=regex)	肯定后顾（Positive Lookbehind） 子表达式能够（肯定）匹配 左侧（后顾）文本，该断言检查的文本和行文方向相反，故称之为**后顾**	(?<=\bre)\w+\b 会匹配以 re 开头的单词的后半部分，不含 re 又，(?<=\s)\d+(?=\s) 匹配前后有空白符间隔的数字，不含空白符
	(?<!regex)	否定后顾（Negative Lookbehind） 子表达式不能（否定）匹配 左侧（后顾）文本	(?<![a-z])\d{7} 匹配前面不是小写字母的七位数字 又，(?<=<(\w+)>).*(?=<\/\1>) 匹配没有设置属性的 HTML 标签的内容，如 我是要匹配的内容

TO BE CONTINUED...

（续表）

特性	语法	描述	举个栗子
固化分组	(?>regex)	贪婪子表达式，也称“**固化分组**”，使用它可以加快匹配失败的速度，如 Subject 这个字符串，现用 ^\w: 对其进行匹配，正则表达式引擎发现 Subject 不匹配，就会试图匹配 Subjec，一直尝试到 S，发现都不匹配才得出无法匹配的结论。如果使用固化分组 ^(?>\w+):，它会直接试图使用 \w+ 去匹配 Subjec 字符串，而不会一一回溯，发现 \w+ 后面没有 :，立即报告失败。	如果字符串中没有第二个 x 的时候，x(?>\w+)x 要比 x\w+x 高效得多
递归匹配	(?R)	递归匹配，Perl 中使用 (?R) 或 (?0)，Ruby 2.0[①]中使用 \g<0>，PCRE 这 3 种都支持。该特性适用于匹配相类似 HTML 标签的场合，它能匹配同等个数的开始和结束标签。	a(?R)?z 匹配 az、aazz 或 aaazzz [②] \{([^{}]|(?R))*\} 匹配 {1, {2, 3}} 这种嵌套结构。

[①]Ruby 是一门开源的动态编程语言，注重简洁和效率。重整本文档时，Ruby 最新版本是 2.3.0。详见Ruby 语言官方网站。

[②]拿 a(?R)?z 匹配 aaazzz 为例，首先表达式中 a 匹配第 1 个 a，然后表达式 (?R) 告诉正则引擎从头匹配整个正则表达式，进入第 1 层递归，a 匹配到第 2 个 a，再次遇到 (?R) 使得正则进入第 2 层递归，再次重新匹配整个正则表达式，此时匹配到第 3 个 a。进入第 3 层递归时，发现已无法匹配到 a 字符，因为只有 3 个 a，又因为 ? 使得 (?R) 是可选项，那么正则表达式开始匹配后面的 z 正则字符，匹配到了第 1 个 z 字符。现在，正则引擎已到达正则表达式的结尾，跳出第 3 层递归，但还有 2 层递归需要继续匹配，同样发现后面的 z 匹配到了第 2 个 z 字符，继续跳出第 1 层递归，匹配最后一个 z 字符，完成整个表达式的匹配过程。

4 举些例子吧！ Regex Examples

Email 地址

`^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$` 匹配形如 WisdomFusion@gmail.com 的邮箱地址。

日期

`^\d{4}\-(0?[1-9]|1[012])\-(0?[1-9]|[12][0-9]|3[01])$` 匹配形如 yyyy-mm-dd 格式的日期。

非零负整数

`^\-[1-9][0-9]*$` 匹配如 $-2, -1024, \ldots$之类的非零负整数。

匹配浮点数

`[-+]?([0-9]*\.[0-9]+|[0-9]+)`. 匹配如 3.1415926 的浮点数（带小数位的）。

去除重复行

查找`^(.*)(\r?\n\1)+$`，替的为`\1`。

匹配用户名

```
^[a-z0-9_-]{3,16}$
```

网址 URL

```
^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$
```

IPv4 地址

```
^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$
```

HTML 标记

```
<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)
```

UBB 代码清理

把`\[/?(?:font|size)([^\]]+)?\]` 替换为空。

汉字

```
[^u4E00-u9FA5]
```

手机号码

```
^(?<national>\+?(?:86)?)(?<separator>\s?-?)(?<phone>(?<vender>(13|15|18)[0-9])(?<area>\d{4})(?<id>\d{4}))$
```

5 正则表达式“流派” Regex Flavors

在标准制定之前，各家正则自成一派，不同工具不同编程语言各不相干。为了理清正则表达式的混乱局面，POSIX[①]把各种常见的流派分为两大类：*Basic Regular Expressions*(BREs) 和 *Extended Regular Expressions*(EREs)。POSIX 程序必须支持其中的任意一种标准。

5.1 正则表达式流派

关于正则表达式“流派”，大家第一印象可能认为不同的工具和编程语言就是“流派”，其实不然，从上述中关于流派的描述可以看出“不同的流派 ≠ 不同的工具或编程语言”。不过，这个概念不是那么重要，因为在实际应用中，我们关于的是工具或编程语言支持哪些特性、同一个特性的语法有什么区别等。表3（POSIX 正则表达式流派概览）中列表出两大流派对正则表达式特性支持的情况，随着文档的完善和丰富，特性的总结将会更加全面。

标准定了，流派也有了，那么各家支持的如何呢？表4（若干工具的正则表达式流派对比）作了简单的对比，可以看出，不同的工具和编程语言，虽没有统一语言，但相差不大，在具体使用某种工具或语言时需要单独了解和掌握。

5.2 正则表达式引擎

regex engine

[①]诞生于 1986 年的 POSIX 是 Portable Operating System Interface（可移植操作系统接口）的缩写，这是一系列标准，确保操作系统之间的移植性。

正则表达式特性	BREs	EREs
点、^、$、[...]、[^...]	✓	✓
“任意数目”量词	*	*
+ 和? 量词		+ ?
区间量词	\{m,n\}	{m,n}
分组	\{…\}	
量词可否作用于括号	✓	✓
反向引用	\1 到\9	
多选结构		✓

表 3: POSIX 正则表达式流派概览

特性	grep	egrep	GNU Emacs	Tcl	Perl	.NET	Java
*、^、$、[…]	✓	✓	✓	✓	✓	✓	✓
? + |	\? \+ |	? + |	? + |	? + |	? + |	? + |	? + |
分组	\(…\)	(…)	\(…\)	(…)	(…)	(…)	(…)
(?:…)					✓	✓	✓
单词分界符		\<\>	\<\>、\b\B	\n\M\y	\b\B	\b\B	\b\B
\w、\W		✓	✓	✓	✓	✓	✓
反向引用	✓	✓	✓	✓	✓	✓	✓

表 4: 若干工具的正则表达式流派对比

6 应用场景 Application Scenarios

学以致用这是最后的落脚点，要不我费这么大力气整理这个文档做什么呢！这部分我将从我自身角度出发，展示一下日常工作中正在用或曾用过的支持正则表达式的编辑器和编程语言们，希望能过这些更具体的实例和应用，让大家对正则表达式有更进一步的认识，培养凡事都要批量处理的“懒习惯”。

五花八门的工具，很多，也很杂，但总有一款适合你！小到日常用的编辑器，大到一门编程语言，如果想熟练使用正则表达式，还是需要花点儿时间细看一下本文的。用 Windows 的同学可能有这个感触：Windows 下的记事本太鸡肋，Word 处理方式主要是“通配符”而不是正则表达式，局限性太大，而且比较繁烦。而本文总结的工具和语言将让你挣脱束缚，事半功倍。

6.1 支持正则表达式的工具们

首先，我们先来看看那些支持正则表达式的工具们（出场顺序不代表偏重和强弱，大家自行选用，还有众多好工具因篇幅不能一一列举）。

6.1.1 RegexBuddy

JGsoft 开发的一个强大的正则表达式测试工具，这款是正则测试界最最强大的工具了，**没有之一**，要墙裂向大家推介的哦！☺

图2中所示，①正则表达式区域，②替换字符区域，③暂存使用的正则表达式以备后用，④待测试的文字，⑤替换后的文字。没有标注的区域还有一大堆很贴心的正则表达式创建功能，比如不同语言的选择、模式的开关、不同流派正则表达式的转换、GREP、正则表达式库等等，正如其名“正则表达式好兄弟”，我平时更喜欢“正则基友”这个简称，☺ 建议安装长期占有之。

这里总结了一些功能特色：

- 特性全，而且正则流派支持很全，同一样语言的不同版本的正则特性的细微差别都研究得透透的。按 Alt + F1 可以打开流派列表。
- 实时正则表达式匹配测试，正则表达式创建功能更带有正则的解释功能，不懂的正则可以看看这个功能。
- 可以把正则在不同流派间转换，并可以使成指定编程语言的代码。
- 历史功能像个正则收藏夹，创建好的正则可以保存起来以备后用。
- 功能强大，好处多多。你，值得拥有！

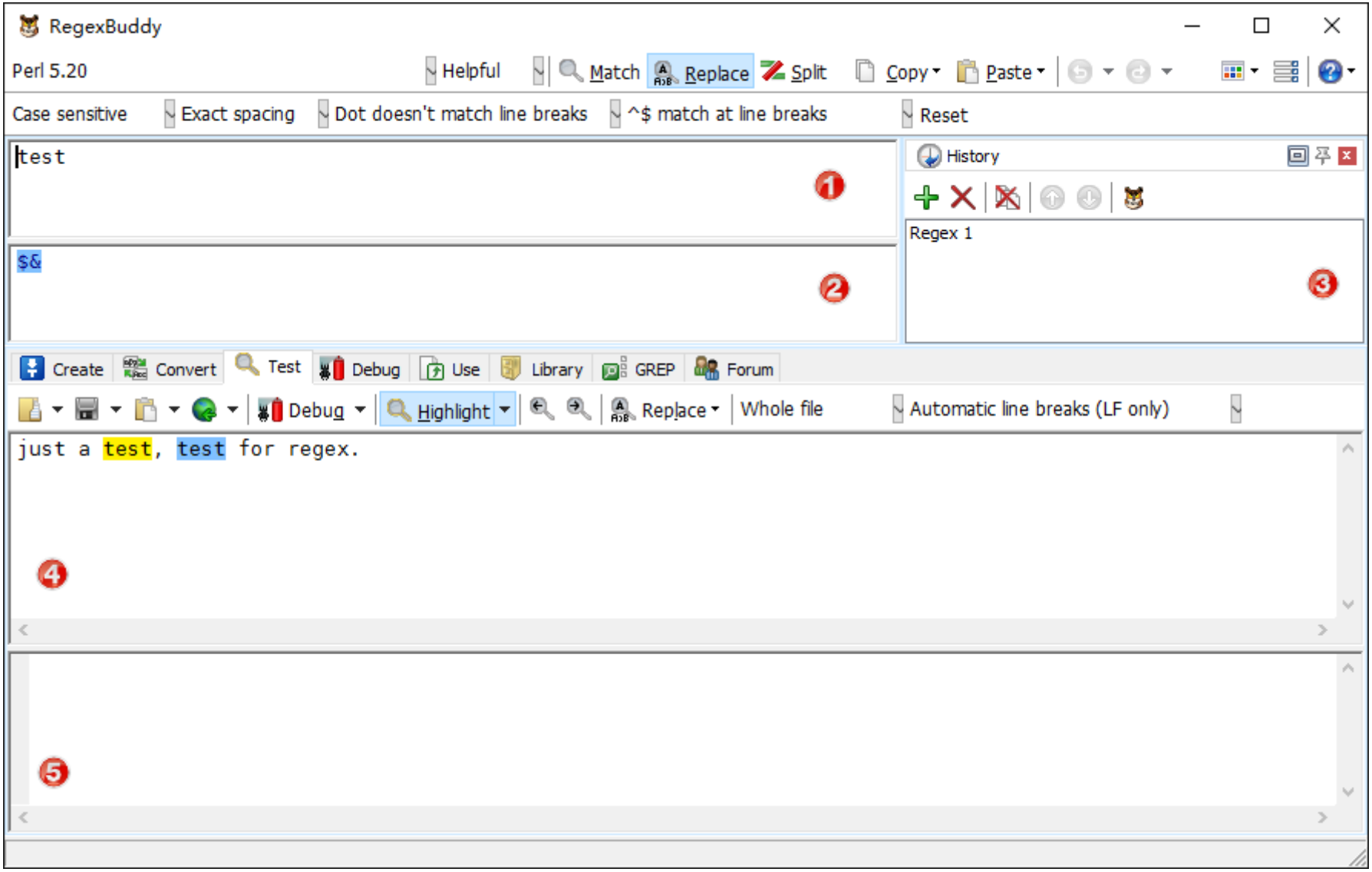

图 2: RegexBuddy 界面

6.1.2 PowerGREP

PowerGREP 是 RegexBuddy 的兄弟软件，同是 JGsoft 开发，是 grep 在 Windows 平台的实现和增强。

图3是该软件的界面展示，乍一看选项很多，简直有些杂乱无章，但细看一下就明白很有条理，而且这个最初印象是来自于它的功能强大。类 UNIX 系统[1]的同学，一眼就能看出，这比 grep 的功能要强很大，find, sed 等命令的一些功能也都包括了。如 Action Type 中列出的 Search（搜索）、Collect data（收集数据）、List files（列出文件）、Search and replace（查换替换）等等，左侧边栏可以针对文件的修改文期、文件大小等作搜索条件限制。可以看出，JGsoft 这家厂商出的软件还是很实用的。

[1]类 Unix 系统（英文：Unix-like）指各种传统的 Unix 系统（比如 Mac OS、FreeBSD、OpenBSD、Solaris）以及各种与传统 Unix 类似的系统（例如 Minix、Linux 等）。它们虽然有的是自由软件，有的是商业软件，但都相当程度地继承了原始 UNIX 的特性，有许多相似处，并且都在一定程度上遵守 POSIX 规范。

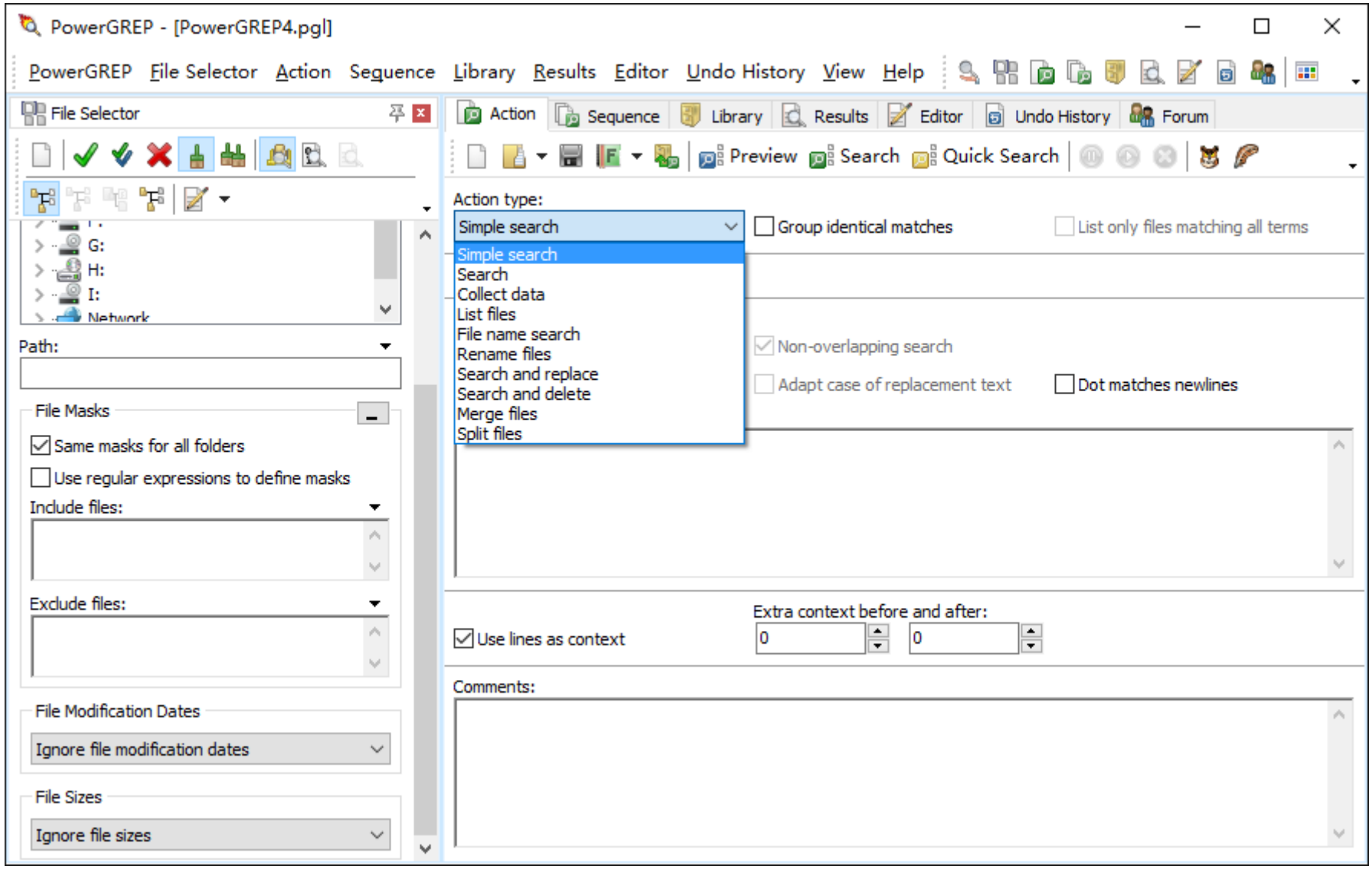

图 3: PowerGREP 界面

6.1.3 在线正则表达式测试工具

https://www.debuggex.com/
目前只支持 JavaScript, Python, PCRE。

6.1.4 grep

grep, egrep, fgrep

6.1.5 Vim

编辑器之神 Vim[①]，既然十几年来一直被冠以如此美誉，可不是吃素的，文字编辑方面是当之无愧的老大，而文字编辑最不可少的就是查找替换，正则表达式自然就是最大的功臣之一。Vim 有 4 种查找模式：**magic**（开关\m，默认模式，无须手动添加开关）, **very magic**（开关\v）, **nomagic**（开关\M，类似正则表达式里的取反）和 **very nomagic**（开关\V）。这几种模式我会在专门的 vi/Vim 文档里详述，这里主要举一些默认的 **magic** 模式查找替换的实例子吧：

命令	说明
:%s/child/children/g	基本文本替换
:%s/\<child\>/chinldren/g	匹配整个单词
:%s/^V^M//g	把 ^M 替换掉[a]
:g/^$/d	删除空行

[a] UNIX 下换行符是\n，ASCII 码是 0xA，Windows 下是\r\n，0xD 和 0xA 的组合，而\r 在 Vim 显示的是形如 ^M 的字符，输入方式是Ctrl+V Ctrl+M。

[①] “编辑器之神”Vim和“神的编辑器”Emacs 粉丝之间的圣战说来也有小一二十年了，但本人对工具没有抱有任何偏见，我始终认为“用合适的工具”才是正确的做法和想法，而不是整日里争一时口舌之快，回过头来一看一事无成。

6.1.6 GNU Emacs

GNU Emacs 中正则表达式异常强大，除了基本的正向正则查找C-M-s、反向正则查找C-M-r、正则替换 `M-x replace-regexp`、查询式正则替换 `M-x query-replace-regexp`、保留行 `M-x keep-lines`、删除行 `M-x flush-lines`、...等等，Emacs 正则表达式替换时还能直接执行 LISP[①] Form[②]。

[①]出生自 1958 年，但目前仍很活跃的一门编程语言，尤其在 AI 领域中。

[②]简言之：可以直接执行 LISP 代码，利用强大的编程语言提供的函数方法去处理文本。

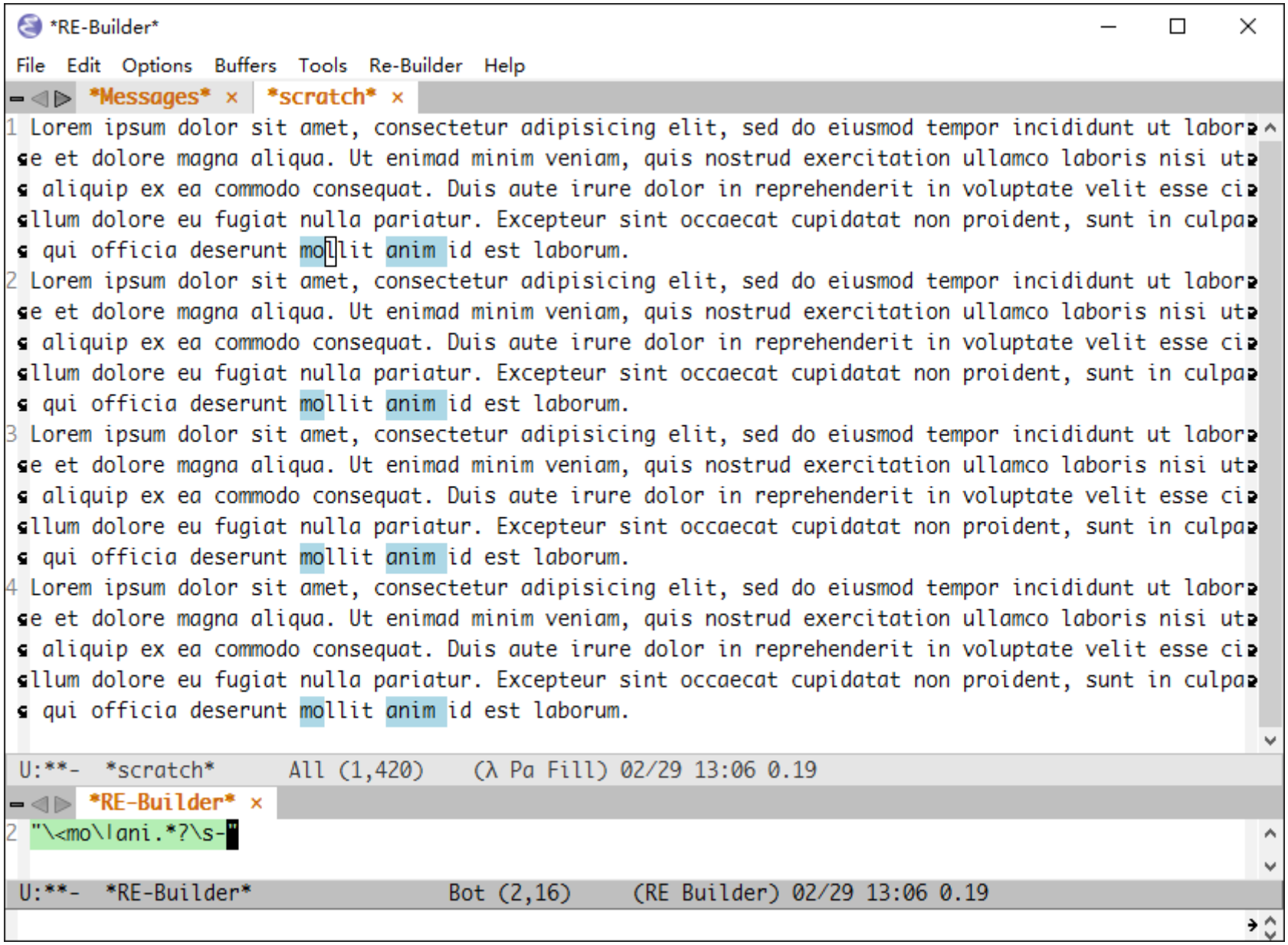

图 4: GNU Emacs 中 RE-Builder 模式

6.1.7 Adobe Dreamweaver 表格处理

Dw

6.1.8 UltraEdit 等编辑器

UltraEdit, EmEditor, Notepad++, Editplus, ...etc.

6.1.9 GREP for Adobe InDesign

ID

6.2 猿类们经常用到的正则表达式

6.2.1 Perl 正则表达式王国

m/<regexp>/

- /i 不区分大小写
- /s
- /x
- /o

s/<pattern>/<repacement>/

/g	全局替换
\U	
\L	
\E	
\l 和 \u	只作用于下一个字符

tr/<pattern>/<replacement>/

\G 锚点实例，这个特性不是很常用，只因上文中表格处不好举例，这里特添加一例作补充说明。

```
while ($string =~ m/</g) {
    if ($string =~ m/\Gb>/ic) {
        # Bold
```

```
    } elsif ($string =~ m/\Gi>/ic) {
        # Italics
    } else {
        # ...etc...
    }
}
```

6.2.2 Python

Python

https://docs.python.org/3/library/re.html

```
import re

fh = open("simpsons_phone_book.txt")
for line in fh:
    if re.search(r"J.*Neu",line):
        print(line.rstrip())
fh.close()
```

```
import re

l = ["555-8396␣Neu,␣Allison",
     "Burns,␣C.␣Montgomery",
     "555-5299␣Putz,␣Lionel",
     "555-7334␣Simpson,␣Homer␣Jay"]

```

```
for i in l:
    res = re.search(r"([0-9-]*)\s*([A-Za-z]+),\s+(.*)", i)
    print(res.group(3) + "␣" + res.group(2) + "␣" + res.group(1))
```

```
import re

example_codes = ["SW1A␣0AA", # House of Commons
                 "SW1A␣1AA", # Buckingham Palace
                 "SW1A␣2AA", # Downing Street
                 "BX3␣2BB", # Barclays Bank
                 "DH98␣1BT", # British Telecom
                 "N1␣9GU", # Guardian Newspaper
                 "E98␣1TT", # The Times
                 "TIM␣E22", # a fake postcode
                 "A␣B1␣A22", # not a valid postcode
                 "EC2N␣2DB", # Deutsche Bank
                 "SE9␣2UG", # University of Greenwhich
                 "N1␣0UY", # Islington, London
                 "EC1V␣8DS", # Clerkenwell, London
                 "WC1X␣9DT", # WC1X 9DT
                 "B42␣1LG", # Birmingham
                 "B28␣9AD", # Birmingham
                 "W12␣7RJ", # London, BBC News Centre
                 "BBC␣007" # a fake postcode
                ]

pc_re = r"[A-z]{1,2}[0-9R][0-9A-Z]?␣[0-9][ABD-HJLNP-UW-Z]{2}"
```

```
24
25 for postcode in example_codes:
26     r = re.search(pc_re, postcode)
27     if r:
28         print(postcode + " matched!")
29     else:
30         print(postcode + " is not a valid postcode!")
```

6.2.3 PHP PCRE

PHP 中使用 PCRE[①]

```
1 $subject='Give me 10 eggs';
2 $pattern='~\b(\d+)\s*(\w+)$~';
3
4 $success = preg_match($pattern, $subject, $match);
5 if ($success) {
6     echo "Match: ".$match[0]."<br />";
7     echo "Group 1: ".$match[1]."<br />";
8     echo "Group 2: ".$match[2]."<br />";
9 }
```

Output: Match: 10 eggs Group 1: 10 Group 2: eggs

```
1 $subject='Give me 12 eggs then 12 more.';
2 $pattern='~\d+~';
3 $newstring = preg_replace($pattern, "6", $subject);
```

[①]PCRE, Perl Compatible Regular Expressions

```
echo $newstring;
```

The Output: Give me 6 eggs then 6 more.

6.2.4 .NET Framework 正则表达式

.NET Framework 正则表达式

6.2.5 JavaScript

javascript

6.2.6 sed & awk

sed
awk

6.2.7 VBA 中使用正则表达式

VBA[1]是不直接支持正则表达式的，需要借助 VBScript RegExp Object，具体请参考Microsoft Beefs Up VBScript with Regular Expressions。

```
Sub IndentParaWithRegEx()
' PowerPoint VBA 批量给指定字符开头段落加动画
Dim oSld As Slide
Dim oShp As Shape
Dim i As Integer
' 正则相关变量
Dim regx As Object, oMatch As Object
```

[1]Visual Basic for Application，Microsoft Office 套件宏语言。

```
strPattern = "^开头字符串"

Set regx = CreateObject("vbscript.regexp")
With regx
    .Global = True
    .IgnoreCase = True
    .Pattern = strPattern
End With

For Each oSld In ActivePresentation.Slides
    For Each oShp In oSld.Shapes
        If oShp.HasTextFrame Then
            If oShp.TextFrame2.HasText Then
                With oShp.TextFrame2.TextRange
                    For i = 1 To .Paragraphs.Count
                        With .Paragraphs(i)
                            ' 可能会出现多个匹配项的
                            If (regx.Test(.Text) = True) Then
                                .ParagraphFormat.FirstLineIndent = 0
                            End If
                        End With
                    Next i 'para
                End With
            End If 'has text
        End If 'has textframe
    Next oShp
```

```
Next oSld
End Sub
```

6.2.8 More and more

还有更多

6.3 Changelog

changelog

6.4 Disclaimer

本文档参考了大量参考文档及网页内容，同时，个别实例摘录自图书或网络文章，如发现雷同之处，触犯你方利益，请及时联系，我会作相应的调整和更改，谢谢合作！