正则表达式（regular expressions）是具有增强查找/字符串替换操作功能的表达式。我们通常在文本编辑器里编辑文本时使用它。目的有以下四个方面：

1)检查文本内容中是不是包含一个给定的模式

2）查找任何匹配的模式

3）从文本中提取信息

4）对文本内容进行修改

绝大多数高级编程语言都支持正则表达式，

**一）什么是正则表达式**

一个正则表达式就是一个没有长度限制的String。通常这个String很短，下面是一些例子：

* I had a \S+ day today
* [A-Za-z0-9\-\_]{3,16}
* \d\d\d\d-\d\d-\d\d
* v(\d+)(\.\d+)\*
* TotalMessages="(.\*?)"
* <[^<>]>

这个字符串实际上是一个极小的计算程序。并且正则表达式是一个语法小而简洁，领域特定的编程语言。牢记以下几点，它们不应在你学习过程中感到惊讶：

1）每一个正则表达式都可以被分解成一个顺序进行的指令集合：”在这儿查找、然后在哪儿查找、然后。。。“

2）每一个正则表达式都有输入（给定文本）和输出（模式匹配的结果，有时是修改后的文本）。

3）存在语法错误（并不是每个字符串都是一个合法的正则表达式）

4）语法有些怪异，或者说是恐怖

5）一个正则表达式有时候可以被编译以便使其运行更快

正则表达式一直有着显著的改变，但是本文将集中介绍在几乎每个正则表达式实现都包含的正则表达式核心语法。

练习

获取一个支持正则的文本编辑器。我推荐Notepad++。

下载一篇很长的散文故事比如Gutenberg出版社出版的H. G. Wells的《时光机器》然后打开它。

下载一部字典，比如这个，解压然后打开。一切准备就绪，稍后开始练习。

提示： 正则表达式与文件通配符语法完全不兼容，比如\*.xml

**二）正则表达式基本语法**

**1）常量（literals）**

正则表达式由含义如其字面所示的常量字符和具有特殊含义的元字符构成。下面是一些示例，元字符由下划线标出。

* I had a \S+ day today
* [A-Za-z0-9\-\_]{3,16}
* \d\d\d\d-\d\d-\d\d
* v(\d+)(\.\d+)\*
* TotalMessages="(.\*?)"
* <[^<>]\*>

大多数的字符，包括字母数字字符都以字面量的形式出现，这意味着他们查找的是自身，例如，对于正则表达式

Cat

含义为：”查找字符c，其后面紧跟字母a，然后又紧跟字符t“

目前为止感觉良好。这的确很像

一个普通的查找对话框

Java中的String.indexOf()函数

PHP中的strpos()函数

等等

提示：除非特别说明，正则表达式是区分大小写的。然而，绝大多数实现都会提供一个标记来开启不区分大小写的功能。

句点（.）

**2）句点（dot）**

我们第一个元字符是句号（译者注：句点，英文句号）.。一个.表示匹配任何单个字符。下面这个正则表达式c.t代表“先找到c，接着找到任何单个字符，再找到t”。

在一段文本中，这个表达式将会找到cat，cot，czt，甚至字面值为c.t的字符串（c，句点，t），但是不包括ct或者coot。

在正则表达式里，空格是有效的。正则表达式 'c t' 代表”先找到 'c'，接着找到空格，再找到 't'“。

任何元字符如果用一个反斜杆\进行转义就会变成字面值。所以上述的正则表达式c\.t就代表“先找到c，接着找到句号，再找到t”。

反斜杠是一个元字符，这意味着它也可以使用反斜杠转义。所以正则表达式c\\t代表“先找到c，接着找到反斜杆，再找到t”。

注意！ 在一些实现中，. 会匹配除了 换行符 的任意字符。这意味着“换行符”在不同的实现中也会变化。 要查看你的文档。在这篇文章中， 我会确保. 会匹配任意字符。

在其它情况下， 通常会有一个标记来调整这种行为，那就是`DOTALL`或类似的标记

**3）字符组：**

字符组为一个由方括号括起来的字符集合，这表示：匹配查找任何一个字符集合里的字符。

正则表达式c[aeiou]t表示“找到c后跟一个元音字母，再找到t”。在一段文本中，将会匹配到cat，cet，cit，cot和cut。

正则表达式[0123456789]表示找到一个数字

正则表达式[a]和a意义相同：“找到a”

一些转义的例子：

\[a\]表示“找到一个左方括号紧跟着一个a，再跟着一个右方括号”。

[\[\]ab]表示“匹配一个左方括号或者右方括号或者a或者 b”。

[\\\[\]]表示“匹配一个反斜杆或者一个左方括号或者一个右方括号”。（呕！）

在字符类中顺序和重复字符并不重要。[dabaaabcc]跟[abcd]一样。

重要的提示

在字符类内部的“规则”和在字符类内部的规则有所不同。一些字符在字符类内部扮演着元字符的角色，但在字符类外部则充当字面值。还有一些字符做着相反的事。一些字符在两种情形都为元字符，但在各自情形里代表不同的含义。

特别地, .表示“匹配任意字符”，但是[.]表示“匹配句点”。不能并为一谈。

**4）字符组区间**

你可以在字符类中使用连字符来表示一个字母或数字的区间：

[b-f]和[bcdef]都表示“找到一个b或c或d或 e或f”。

[A-Z]和[ABCDEFGHIJKLMNOPQRSTUVWXYZ]都表示“匹配大写字母”。

[1-9]和[123456789]都表示“匹配一个非零数字”。

连字符在字符类外部使用时并没有特别都含义。正则表达式a-z表示“找到一个a接着跟着一个连字符，然后匹配一个z”。

区间和单独的字符可能会共存于同一个字符类：

[0-9.,]表示“匹配一个数字或者一个句点或者一个逗号”。

[0-9a-fA-F]表示“匹配一位十六进制数”。

[a-zA-Z0-9\-]表示“匹配一个字母数字字符或连字符”。

虽然你可以尝试在区间内以非字母数字字符结束（比如abc[!-/]def），但这在其它实现中的语法不一定对。即使语法正确，但在这个区间内很难看出包含了哪个字符。请谨慎使用（我的意思是不要这么干）。

同样的，区间端点的范围应该一致。即使像[A-z]这种表达式在你选择的实现中合法，但结果可能不如你愿。（补充：可以有Z到a的区间范围）。

注意。 区间是字符的区间，不是数字的区间。正则表达式[1-31]表示“找到一个1或一个 2或一个3”，不是“找到一个从1到31的整数"。

**5）字符组否定**

你可以通过在最开始的位置使用插入符号（译者注：^）来否定一个字符类。

[^a]表示“匹配除了a的任意字符”。

[^a-zA-Z0-9]表示“找到一个非字母也非数字的字符”。

[\^abc]表示“找到一个插入符或者a或者b或者c”。

[^\^]表示“找到除了插入符外的任意字符”。（呕！）

**6）字符组补充（规定的简易表示）**

正则表达式\d含义与[0-9]一致：“匹配一个数字”。（为了匹配一个反斜杆后跟一个d，可以使用\\d。）

\w的含义与[0-9A-Za-z\_]一致：“匹配一个单词字符（译者注：字母或数字或下划线或汉字）”。

\s表示“匹配任意空白字符（空格，tab，回车或者换行）”。

此外，

\D同[^0-9]：“匹配任意非数字的字符”。

\W同[^0-9A-Za-z\_]：“匹配任意非单词字符（译者注：匹配任意不是字母，数字，下划线，汉字的字符）”。

\S表示“匹配任意不是空白符的字符”。

这些字符类都很常见，你必须学会。

你可能也注意到了，句点.本质上是一个包含任意字符的字符类。

许多实现提供了很多额外的字符类或标记，它们通过扩展现有的字符类来覆盖ASCII之外范围的字符。提示：Unicode包含更多的“数字字符”而不仅仅是0到9，这一点同样对于“单词”和“空格”也适用。注意你的文档所写。

**7）乘法器**

你可以在一个字面值或者字符类后跟着一个大括号来使用乘法器。

正则表达式a{1}同a，表示“匹配一个a”。

a{3}表示“找到一个a后再跟一个a，最后找到一个a”。

a{0}表示“匹配空字符”。就其本身而言，这似乎没有用处。如果你在任何一段文本中使用该表达式，你会在你刚开始搜索的端点处立即得到一个匹配。即使你的文本为空字符串结果也为真。

a\{2\}代表“找到一个a，跟着一个左大括号，接着跟匹配一个2，然后跟着一个右大括号”。

在字符类中大括号没有特别的含义。[{}]代表“匹配一个左大括号或者一个右大括号”。

注意，乘法器没有记忆。该正则表达式[abc]{2}表示“匹配a或者b或者c，接着匹配a或者b或者c。这跟“匹配aa或ab或ac或ba或bb或bc或ca或cb或cc”相同。这跟“匹配aa或bb或cc”含义不同！

**8）乘法器区间**

乘法器可能会有区间：

x{4,4}跟x{4}一样。

colou{0,1}r表示“匹配colour或color。

a{3,5}表示“匹配aaaaa或aaaa或aaa”。

值得注意的是优先选择更长的匹配，因为乘法器是贪婪的。如果你输入的文本是I had an aaaaawful day，该正则表达式就会在aaaaawful中匹配到aaaaa。不会在第三个a后就停止匹配。

乘法器是贪婪的，但它不会忽略一个更好的匹配。如果你的输入文本为I had an aaawful daaaaay，之后这个正则表达式会在第一次的匹配中于aaawful找到aaa。只有在你说“给我找到另一个匹配”的时候，它才会继续搜索然后在daaaaay中找到aaaaa。

乘法器区间可能是开区间：

a{1,}表示“在一列中找到一个或多个a”。然而你的乘法器将会是贪婪的。在找到第一个a后，它将会尽可能匹配到更多的a。

.{0,}表示“匹配任何情形”。不管你的输入文本是什么——甚至为空——这个正则表达式都会匹配整个字符串然后返回给你。

**9）乘法器补充**

?代表的含义与{0,1}相同。比如说，colou?r表示“匹配colour或color”。

\*等于{0,}。比如说，.\*表示“匹配一切”，跟上面提到的一样。

+等于{1,}。比如说，\w+表示“匹配一个单词”。这里的“单词”是1个或多个“单词字符”的序列，就像\_var或AccountName1。

这些乘法器都很常见，你必须掌握。还有：

\?\\*\+表示“匹配一个问号，接着找到一个星号，然后跟着一个加号”。

[?\*+]表示“找到一个问号或者一个星号或者一个加号”。

**10）Non-greed**

正则表达式".\*"表示“找到一个双引号，接着找到尽可能多的字符，最后再找到一个双引号”。注意一下被.\*匹配的内部字符，很可能包含多个双引号。这通常不是非常有用。

乘法器可通过追加问号来实现惰性。这里对优先顺序进行了反转：

\d{4,5}?表示“匹配\d\d\d\d或\d\d\d\d\d”。其实跟\d{4}行为一致。

colou??r就是colou{0,1}?r，表示“找到color或colour”。和colou?r行为一致。

".\*?"表示“匹配一个双引号，跟着一个尽可能少的字符，再跟着一个双引号”。这个不像上面两个例子，实际上很有用。

**11）Alternation**

你可以使用管道符号来实现匹配多种选择：

cat|dog表示“匹配cat或dog”。

red|blue|和red||blue以及|red|blue都是同样的意思，“匹配red或blue或空字符串”。

a|b|c跟[abc]一样。

cat|dog|\|表示“匹配cat或dog或管道符号”。

[cat|dog]表示“找到a或c或d或d或g或o或t或一个管道符号”。

**12)Grouping**

组合可能会包含空字符串：

(red|blue|)表示“匹配red或blue或空字符串”。

abc()def等同于abcdef

可能你会在组合中使用乘法器：

(red|blue)?等同于(red|blue|)。

\w+(\s+\w+)\*代表“找到一个或多个单词，它们以空格隔开”。

**13)word boundary**

单词边界是一个单词字符和非单词字符之间的位置。记住，一个单词字符是\w，它是[0-9A-Za-z\_]，一个非单词字符是\W，也就是[^0-9A-Za-z\_]。文本的开头和结尾总是当作单词边界。

输入的文本it's a cat有八个单词边界。如果我们在cat后追加一个空格，这里就会有九个单词边界。

正则表达式\b表示“匹配一个单词边界”。

\b\w\w\w\b表示“匹配一个三个字母的单词”。

a\ba表示“找到a，跟着一个单词边界，接着找到b”。不管输入文本是什么，这个正则表达式永远都不会成功找到一个匹配。

单词边界不是字符。它们宽度为零.下面的正则表达式表示相同的含义：

(\bcat)\b

(\bcat\b)

\b(cat)\b

\b(cat\b)

**14)行边界**

正则表达式^表示“匹配开始行”。

正则表达式$表示“匹配结束行”。

^$表示“匹配空行”。

^.\*$将会匹配整个文本，因为换行符是一个字符，所以.会匹配它。为了匹配单行，要使用惰性乘法器，^.\*?$。

\^\$表示“匹配尖符号后跟着一个美元符号”。

[$]表示“匹配一个美元符”。然而，[^]是非法单正则表达式。要记住的是尖符号在方括号中时有不同的特殊含义。把尖符号放在字符类中，这么用[\^]。

像单词边界一样，行边界也不是字符。它们宽度为零。下面的正则表达式表示相同的含义：

(^cat)$

(^cat$)

^(cat)$

^(cat$)

**15）文本边界**

很多实现提供一个标记，通过改变它来改变^和$的含义。从“行开始”和“行结束”变成“文本开始”和“文本结束”。

其它的一些实现提供单独的元字符\A和\z来达到这个目的。