

目录

前言	1.1
正则简介	1.2
正则这个词是什么意思	1.2.1
正则语法	1.3
一图记住所有正则	1.3.1
环视断言	1.3.2
举例解释	1.3.2.1
正则应用	1.4
编程语言	1.4.1
工具和软件	1.4.2
数据库	1.4.3
其他领域	1.4.4
正则心得	1.5
正则调试工具	1.5.1
如何看懂别人的正则	1.5.2
如何自己写正则	1.5.3
附录	1.6
参考资料	1.6.1

应用广泛的超强搜索：正则表达式

- 最新版本：[v1.1](#)
- 更新时间：[20200212](#)

鸣谢

感谢我的老婆陈雪雪的包容理解和悉心照料，才使得我 `crifan` 有更多精力去专注技术专研和整理归纳出这些电子书和技术教程，特此鸣谢。

简介

详尽整理正则表达式的基本语法；正则的名称的由来；整理出一张脑图看懂和记住所有正则的语法；以及用图形象解释了环视断言的种类和含义；常见的正则的应用领域和具体应用案例；整理了正则的心得，包括如何看懂别的的正则和如何自己写出相对复杂的正则。以及整理了好用的调试正则的工具和在线网站。

源码+浏览+下载

本书的各种源码、在线浏览地址、多种格式文件下载如下：

Gitbook源码

- [crifan/super_search_regex: 应用广泛的超强搜索：正则表达式](#)

如何使用此Gitbook源码去生成发布为电子书

详见：[crifan/gitbook_template: demo how to use crifan gitbook template and demo](#)

在线浏览

- [应用广泛的超强搜索：正则表达式 book.crifan.com](#)
- [应用广泛的超强搜索：正则表达式 crifan.github.io](#)

离线下载阅读

- [应用广泛的超强搜索：正则表达式 PDF](#)
- [应用广泛的超强搜索：正则表达式 ePUB](#)
- [应用广泛的超强搜索：正则表达式 Mobi](#)

版权说明

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您版权，请通过邮箱联系我 `admin 艾特 crifan.com`，我会尽快删除。谢谢合作。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-12
20:27:39

正则简介

旧版教程

很早之前已写过正则的教程[正则表达式学习心得](#)

发布工具是[Docbook](#), 暂时不方便整合到此[Gitbook](#)的教程中。

以后有空再合并新旧教程。

就像[字符编码](#)一样, 正则表达式, 简称 正则, 是个应用领域很广的软件方面的基础知识之一。

教程目的

此教程主要讨论, 正则本身的通用逻辑。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-08 18:10:49

正则这个词是什么意思

先来说说：

- 正则这个词是什么意思

或者说：

- 正则 这个名称是什么意思？
- 正则 这个名字是怎么来的？
- 为何叫做正则？

先解释名词本身的称谓：

- 正则
 - 中文
 - 全称： 正则表达式
 - 英文
 - 全称： regular expression
 - 简称
 - re
 - 来自： RE gular expression
 - regex
 - 来自： REG ular EX pression

再来解释：正则的词的含义

首先要明白，正则主要存在的领域：字符串

在字符串的领域，即处理字符串相关内容时，往往会遇到搜索（和替换）

而对于普通字符串，比如：

asdlkg77832qth4uh

是不是你看起来，觉得，没什么特殊规律？

是的，我也看不出有什么规律。

所以，可称之为：普通字符串

而对于：

hello world, hello CrifanLi, hello Love, hello Python3, hello regex

是不是觉得有规律可循？

且如果要从其中找出：hello xxx中的xxx，且如果再加上限定条件：xxx首字母是大写

你是不是也能一眼看出其中的规律？

很明显，是的：除了第一个和最后一个，中间的3个，都符合条件

换句话说：这个字符串，有一定规律，有规律可循

再来说regular这个英文单词，其本意就是：有规律的，整齐的，（形状）规则的

如此，就容易理解：

regular expression = 正则表达式

就是指：

- 要处理的字符串是有规律的，有规律可循的
 - 而 正则表达式 = （用特定语法写出来的）字符串的规律

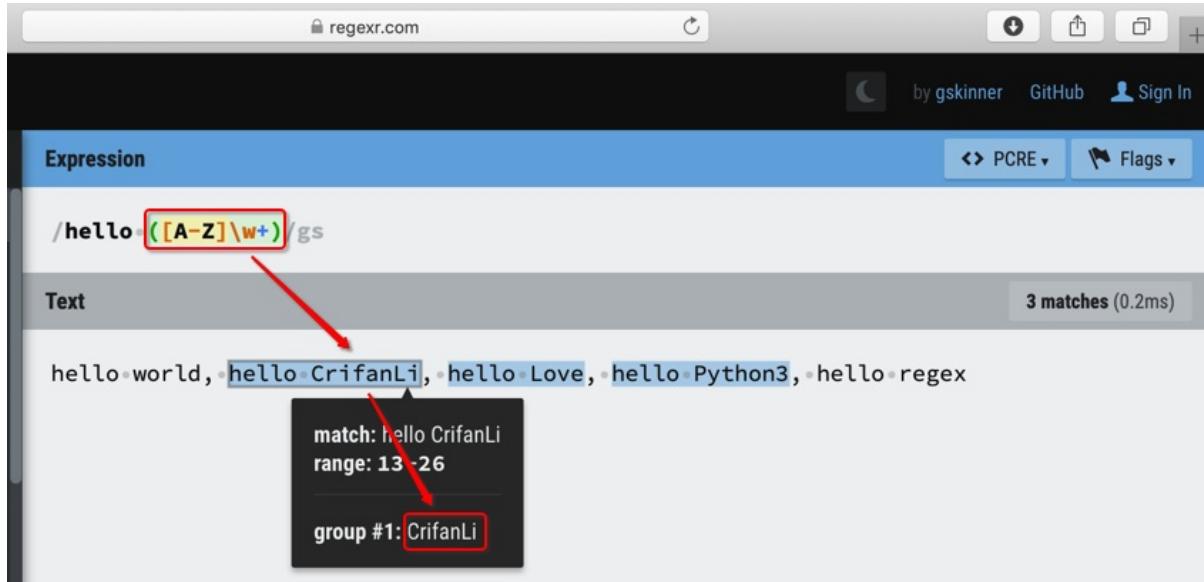
进一步说，要从上述字符串中，提取出：

- hello xxx 中的 xxx
 - 且再加上限定条件： xxx 中的首字母是大写

对应的正则就是：

hello ([A-Z][a-zA-Z0-9]+) 或： hello ([A-Z]\w+)

从[RegExr: Learn, Build, & Test RegEx](#)，可见匹配效果，如图：



汇总如下：

- 普通字符串：可视为没有规律的
 - 比如：`asd1kg7783qth4uht`
- 有规律的字符串：是我们要处理的，即搜索或替换等操作
 - 我们才去把字符串中的规律，用 正则表达式 去写出来
 - 从而匹配到我们要的字符串
 - 用于后续的其他操作
 - 比如：
 - 有规律的字符串：`hello world, hello CrifanLi, hello Love, hello Python3, hello regex`
 - 想要匹配和提取：xxx 中的 xxx，且 xxx 的首字母是大写
 - 对应正则表达式：`hello ([A-Z][a-zA-Z0-9]+)`
 - 或 `hello ([A-Z]\w+)`

正则这个词的含义

总结起来就是：

- `regular` = 正则：有规律的
 - 字符串中要处理的内容是有规律的
- `regular expression` = 正则表达式：用正则的语法把字符串的规律表达出来
 - 目的：实现字符串的处理，如搜索出匹配的值，字符串的成其他值等

正则语法

下面整理出正则表达式的语法规则：

此处以Python为例，参考：[Python Regular Expression's Cheat Sheet](#)，正则表达式的基本语法如下：

正则表达式语法

文字版：

- Special Characters
 - \ escape special characters
 - common char need escape:
 - \
 - [
 -]
 - (
 -)
 - {
 - }
 - .
 - *
 - +
 - ?
 - ^
 - \$
 - |
 - . matches any character
 - ^ matches beginning of string
 - \$ matches end of string
 - [5b-d] matches any chars '5', 'b', 'c' or 'd'
 - [^a-c6] matches any char except 'a', 'b', 'c' or '6'
 - R|S matches either regex R or regex S
 - () creates a capture group and indicates precedence
- Quantifiers
 - * 0 or more (append ? for non-greedy)
 - + 1 or more (append ? for non-greedy)
 - ? 0 or 1 (append ? for non-greedy)
 - {m} exactly mm occurrences
 - {m, n} from m to n. m defaults to 0, n to infinity
 - {m, n}? from m to n, as few as possible
- Special sequences
 - \A start of string
 - \b matches empty string at word boundary (between \w and \w)
 - \B matches empty string not at word boundary
 - \d digit
 - \D non-digit
 - \s whitespace: [\t\n\r\f\v]
 - \S non-whitespace

- `\w` alphanumeric: `[0-9a-zA-Z_]`
- `\W` non-alphanumeric
- `\z` end of string
- `\g<id>` matches a previously defined group
- Extensions
 - `(?iLmsux)` Matches empty string, sets re.X flags
 - `(?:...)` Non-capturing version of regular parentheses
 - `(?P<name>...)` Creates a named capturing group
 - `(?P=name)` Matches whatever matched previously named group
 - `(?#...)` A comment; ignored.
 - `(?=...)` Lookahead assertion: Matches without consuming
 - `(?!...)` Negative lookahead assertion
 - `(?<=...)` Lookbehind assertion: Matches if preceded
 - `(?<!...)` Negative lookbehind assertion
 - `(?(id)yes|no)` Match 'yes' if group 'id' matched, else 'no'

一图记住所有正则

刚整理的脑图：

- 在线浏览
 - [一图让你看懂和记住所有正则表达式规则](#)
- 图片：

◦

环视断言Look Around

正则中，有一部分的语法叫做：Look Around，下面详细解释如下：

- 在线浏览
 - 正则表达式详解：look ahead和look behind
- 如图

◦

下面再详细解释如下：

断言

- 叫法
 - 英文：assertion
 - 中文：断言
- 含义：声称，判断，预言：是或否=不是
 - 是：
 - 是什么（样的）
 - 不是
 - 不是什么（样的）
- 正则表达中的断言
 - 叫法：
 - 是 -» 匹配
 - 不是 -» 不匹配
 - 含义
 - 广义：所有的正则表达式都是断言
 - 都是 匹配 或 不匹配 对应规则
 - 狹义：往往指的是零宽断言

零宽断言

- zero-width assertion = 零宽度断言 = 零宽断言
 - = zero-length assertion = 零长度断言
 - 包括
 - word level 单词级别
 - \b
 - \B
 - single line level 单行级别
 - ^
 - \$
 - multiple line 多行=whole string 整个字符串=whole document 整篇文档级别 level
 - \A
 - \Z
 - look around (assertion)= 环视断言
 - 包括
 - positive look ahead = look ahead
 - negative look ahead
 - positive look behind = look behind
 - negative look behind
 - -»
 - zero-width positive look ahead assertion
 - zero-width negative look ahead assertion
 - zero-width positive look behind assertion
 - zero-width negative look behind assertion

非捕获匹配

- non-capture match=非捕获匹配
 - 正则语法: (?:pattern)
 - 含义:
 - 去匹配match, 但是不捕获capture, 不消费consume 对应的字符串

> 某种程度上说, 此处的 look around, 也算是属于非捕获匹配

环视断言look around

- look around : 属于 zero-width assertion
- 含义
 - 概述: look around = 环视断言
 - 包含
 - (?=xxx) : (positive) look ahead (assertion) = 正向肯定断言
 - (?!xxx) : negative look ahead (assertion) = 正向否定断言
 - (?<=xxx) : (positive) look behind (assertion) = 反向肯定断言
 - (?<!xxx) : negative look behind (assertion) = 反向否定断言
 - -> 加上 零宽 前缀
 - zero-width positive look ahead assertion = 零宽正向肯定断言
 - zero-width negative look ahead assertion = 零宽正向否定断言
 - zero-width positive look behind assertion = 零宽反向肯定断言
 - zero-width negative look behind assertion = 零宽反向否定断言
 - 详解
 - look around = 环视断言

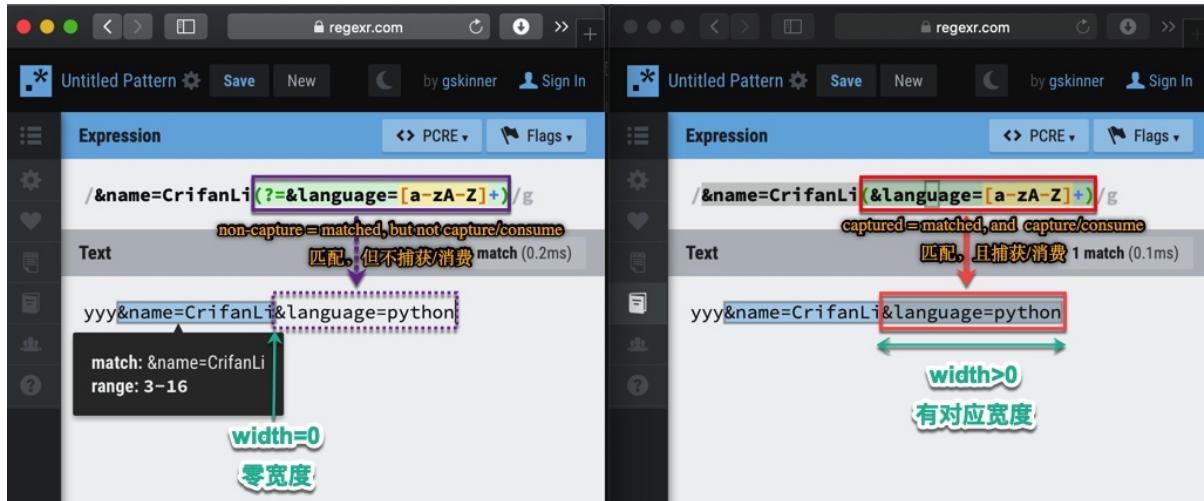
- `look ahead`
 - 英文:
 - `positive look ahead`
 - `look ahead`
 - `zero-width positive look ahead assertion`
 - 中文:
 - 一般译为: 零宽正向先行断言
 - 其他叫法:
 - (正向)前瞻性断言
 - (正向)超前断言
 - 正向预查
 - 零宽断言
 - 最佳翻译:
 - 正向肯定查找
 - 正向肯定断言
- `negative look ahead`
 - 英文:
 - `negative look ahead`
 - `zero-width negative look ahead assertion`
 - 中文:
 - 一般译为: 零宽负向先行断言
 - 其他叫法:
 - 负向先行断言
 - 否定超前断言
 - 最佳翻译:
 - 正向否定查找
 - 正向否定断言
- `look behind`
 - 英文:
 - `positive look behind`
 - `look behind`
 - `zero-width positive look behind assertion`
 - 中文:
 - 一般译为:
 - (正向)后行断言
 - (正向)后顾断言
 - 零宽正向后行断言
 - 正回顾后发断言
 - 零宽正回顾后发断言
 - 最佳翻译:
 - 反向肯定查找
 - 反向肯定断言
- `negative look behind`
 - 英文:
 - `negative look behind`
 - `zero-width negative look behind assertion`
 - 中文:
 - 一般译为:
 - 负向后行断言
 - 零宽负向后行断言
 - 负回顾后发断言

- 零宽负回顧后发断言
- 最佳翻译：

 - 反向否定查找
 - 反向否定断言

零宽度+非捕获 vs 普通有宽度+捕获

用下图来对比说明：



crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook 最后更新： 2020-02-11
21:53:14

举例解释

如前面概述 look around 包含：

- positive lookahead assertion : `(?=xxx)`
- negative lookahead assertion : `(?!xxx)`
- positive lookbehind assertion : `(?<=xxx)`
- negative lookbehind assertion : `(?<!xxx)`

用图解释：

```

41
42
43
44 inputStrList = [
45     "date=20191224&name=CrifanLi&language=python",
46     "language=python&name=CrifanLi&date=20191224", # lookahead will NOT match
47     "language=python&name=CrifanLi date=20191224", # negative lookahead CAN match, the whole 'name=CrifanLi'
48     "language=go&name=CrifanLi date=20191224", # positive lookbehind CAN match
49     "language=go name=CrifanLi date=20191224", # negative lookbehind CAN match
50 ]
51
52 groupNormalPattern = "name=(\w+)" # 匹配任何 name=XXX 其中XXX是字母数字下划线均可
53 groupLookaheadPattern = "name=(\w+)(?=language)" # 只匹配后面 是&language 的情况
54 groupNegativeLookaheadPattern = "name=(\w+)(?!=&)" # 只匹配后面 不是& 的情况
55 groupPositiveLookbehindPattern = "(?=<go&)name=(\w+)" # 只匹配前面 是go& 的情况
56 groupNegativeLookbehindPattern = "(?!=&)name=(\w+)" # 只匹配前面 不是& 的情况
57

```

举例来说明：

- 需求：
 - 要匹配的，普通的，字符串： `&name=CrifanLi`
 - 进一步要求：匹配的字符的 前面 或 后面 也有些其他，符合特定的规则的，字符才匹配，否则不匹配
 - 举例：
 - 后面也有字符，且是`&language=xxx`，其中`xxx`是字母，才匹配，否则不匹配
 - 比如：
 - 匹配： `yyy&name=CrifanLi&language=python`
 - 不匹配： `yyy&name=CrifanLi&gender=male`
 - 前面也有些字符，且是`date=2020`，才匹配，否则不匹配
 - 比如：
 - 匹配： `date=2020&name=CrifanLiyyy`
 - 不匹配： `nationality=china&name=CrifanLiyyy`
 - 后面不能出现也有些字符，比如`&OS=xxx`，其中`xxx`是数字，才匹配，否则不匹配
 - 比如
 - 匹配： `xxxname=CrifanLi&OS=mac`
 - 不匹配： `xxxname=CrifanLi&OS=20200205`
 - 前面不能出现某些字符，比如`java&`，才匹配，否则不匹配
 - 比如
 - 匹配： `python&name=CrifanLixxx`
 - 不匹配： `java&name=CrifanLixxx`
 - 背景知识：相对于，当前要匹配的字符串，从左向右 的方向属于 前进的方向
 - 后面 = 向前看 = 向右看 = 向 \rightarrow 看 = `look ahead`
 - 前面 = 向后看 = 向左看 = 向 \leftarrow 看 = `look behind`
 - 用图举例



- 将上述需求，转换成正则表达式

■ &name=CrifanLi(?=&language=[a-zA-Z]+)

■ 匹配：yyy&name=CrifanLi&language=python

The screenshot shows the regexr.com interface. The expression input field contains the regex pattern `/&name=CrifanLi(?=language=[a-zA-Z]+)/g`. The text input field contains the string `yyy&name=CrifanLi&language=python`. A tooltip is displayed over the word `language`, showing the match details: `match: &name=CrifanLi` and `range: 3-16`.

The Tools panel at the bottom has tabs for Replace, List, Details, and Explain. The Explain tab is selected, showing the breakdown of the regex components:

- i Character.** Matches a "i" character (char code 105). Case sensitive.
- (?= Positive lookahead.)** Matches a group after the main expression without including it in the result.
- & Character.** Matches a "&" character (char code 38).
- l Character.** Matches a "l" character (char code 108). Case sensitive.
- a Character.** Matches a "a" character (char code 97). Case sensitive.
- n Character.** Matches a "n" character (char code 110). Case sensitive.

Below the Tools panel, there is a note: ■ 不匹配: `yyy&name=CrifanLi&gender=male`

The screenshot shows the regexr.com interface. The expression input field contains the regex pattern: `/&name=CrifanLi(?=&language=[a-zA-Z]+)/g`. The text input field contains the string: `yyy&name=CrifanLi&gender=male`. The result panel shows "No match (0.1ms)". Below the input fields is a "Tools" section with buttons for Replace, List, Details, and Explain. The "Explain" button is selected. A tooltip for the character "i" is displayed: "Character. Matches a "i" character (char code 105). Case sensitive." A larger callout box provides detailed information about the lookahead group: "(?= Positive lookahead. Matches a group after the main expression without including it in the result. It contains the following matches: & Character. Matches a "&" character (char code 38). l Character. Matches a "l" character (char code 108). Case sensitive. a Character. Matches a "a" character (char code 97). Case sensitive. n Character. Matches a "n" character (char code 110). Case sensitive." At the bottom, there is a list of matches: "(?<=date=2020)&name=CrifanLi" and a note: "匹配: date=2020&name=CrifanLiyyy".

The screenshot shows the regexr.com interface. In the 'Expression' field, the regex `/ (?<=date=2020)&name=CrifanLi/g` is entered. Below it, in the 'Text' field, the string `date=2020&name=CrifanLiyyy` is shown. A tooltip box highlights the part `(?<=date=2020)` with the text "match: &name=CrifanLi" and "range: 9-22". In the 'Tools' section, there are buttons for Replace, List, Details, and Explain. The 'Details' button is selected. A tooltip for the lookbehind operator `(?<= Positive lookbehind)` explains: "Matches a group before the main expression without including it in the result." It lists three matches: "d Character. Matches a "d" character (char code 100). Case sensitive.", "a Character. Matches a "a" character (char code 97). Case sensitive.", and "t Character. Matches a "t" character (char code 116). Case sensitive.". Below this, a note says "不匹配: nationality=china&name=CrifanLiyyy".

The screenshot shows the regexr.com interface. In the 'Expression' field, the regex `/(?<=date=2020)&name=CrifanLi/g` is entered. Below it, in the 'Text' field, is the string `nationality=china&name=CrifanLiyyy|`. A status bar indicates 'No match (0.1ms)'. In the 'Tools' menu, 'Replace', 'List', 'Details', and 'Explain' are selected. A tooltip for the lookbehind operator (?<=) is displayed, stating: '(?<= Positive lookbehind. Matches a group before the main expression without including it in the result.' It lists three matches: 'd Character. Matches a "d" character (char code 100). Case sensitive.', 'a Character. Matches a "a" character (char code 97). Case sensitive.', and 't Character. Matches a "t" character (char code 116). Case sensitive.'

- 注意：向后看look behind，只支持 固定长度fixed length的内容的匹配
 - 不支持这种 data=\d+ 的不固定的写法，否则会报错：

The screenshot shows a browser window with the URL `regexpr.com` in the address bar. The main interface is titled "Expression" with "PCRE" selected as the engine. A red arrow points from the error message to the lookbehind assertion in the regular expression.

Expression:

```
/(?<=data=\d+)&name=CrifanLi/g
```

Text:

```
date=20200205&name=CrifanLiyyy|
```

Result: **ERROR (0.1ms)**

EXEC ERROR: Compilation failed:
lookbehind assertion is not fixed
length at offset 12

Below the text input, there are two bullet points:

- name=CrifanLi(?!&OS=\d+)
- 匹配: xxxname=CrifanLi&OS=mac

The screenshot shows the regexr.com interface. The expression input field contains `/name=CrifanLi(?!&OS=\d+)/g`. The text input field contains `xxxname=CrifanLi&OS=mac`. A tooltip indicates a match at index 3-15. The Tools panel is open, showing explanations for various regex symbols:

- (?!)** Negative lookahead. Specifies a group that can not match after the main expression (if it matches, the result is discarded).
- &** Character. Matches a "&" character (char code 38).
- O** Character. Matches a "O" character (char code 79). Case sensitive.
- S** Character. Matches a "S" character (char code 83). Case sensitive.
- =** Character. Matches a "=" character (char code 61).

■ 不匹配: `xxxname=CrifanLi&OS=20200205`

The screenshot shows the regexr.com interface. The expression input field contains `/name=CrifanLi (?!=&OS=\d+)/g`. The text area contains `xxxname=CrifanLi&OS=20200205`. A tooltip box is open over the text area, displaying "No match found in 28 characters." and "Insertion point: line 0, col 28, index 28". Below the text area, the tools menu is visible with options: Replace, List, Details, Explain, and a close button. A detailed explanation box is open, explaining the `(?!)` construct. It states: "(?! Negative lookahead. Specifies a group that can not match after the main expression (if it matches, the result is discarded)." Below this, four sub-explanations are listed: "& Character. Matches a "&" character (char code 38).", "O Character. Matches a "O" character (char code 79). Case sensitive.", "S Character. Matches a "S" character (char code 83). Case sensitive.", and "=" Character. Matches a "=" character (char code 61).". At the bottom left, there is a list of recent searches.

Expression: /name=CrifanLi (?!=&OS=\d+)/g

Text: xxxname=CrifanLi&OS=20200205

No match found in 28 characters.

Insertion point: line 0, col 28, index 28

Tools: Replace, List, Details, Explain

(?! Negative lookahead. Specifies a group that can not match after the main expression (if it matches, the result is discarded).

- & Character.** Matches a "&" character (char code 38).
- O Character.** Matches a "O" character (char code 79). Case sensitive.
- S Character.** Matches a "S" character (char code 83). Case sensitive.
- = Character.** Matches a "=" character (char code 61).

Recent Searches:
■ (?<!java&)name=CrifanLi
■ 匹配: python&name=CrifanLixxx

The screenshot shows the regexr.com interface. The expression input field contains `/(?<!java&)name=CrifanLi/g`. The text input field contains `python&name=CrifanLixxx`. A callout box highlights a match at position 7-19, labeled "match: name=CrifanLi range: 7-19". The tools panel is open, showing information about the `(?<!` construct. It defines it as "Negative lookbehind. Specifies a group that can not match before the main expression (if it matches, the result is discarded)". Below this, four character groups are listed: `j` Character. Matches a "j" character (char code 106). Case sensitive., `a` Character. Matches a "a" character (char code 97). Case sensitive., `v` Character. Matches a "v" character (char code 118). Case sensitive., and `a` Character. Matches a "a" character (char code 97). Case sensitive. At the bottom left, there is a note: "不匹配: java&name=CrifanLixxx".

The screenshot shows the regexr.com interface. In the 'Expression' field, the regular expression `/(?<!java&)name=CrifanLi/g` is entered. In the 'Text' field, the input is `java&name=CrifanLixxx`. A tooltip indicates 'No match found in 21 characters.' and 'Insertion point: line 0, col 21, index 21'. Below the text input, there's a 'Tools' section with buttons for Replace, List, Details, Explain, and a close button. A detailed explanation box is open, explaining the `(?<!` construct as a negative lookbehind that specifies a group that cannot match before the main expression (if it matches, the result is discarded). It lists four matches: 'j Character' (char code 106), 'a Character' (char code 97), 'v Character' (char code 118), and another 'a Character' (char code 97).

crifan.com, 使用署名4.0国际(CC BY 4.0)协议发布 all right reserved, powered by Gitbook 最后更新: 2020-02-09
23:09:14

正则应用

关于正则在不同的编程语言、软件或工具、系统等领域的应用，则详见独立教程：

正则表达式应用举例

此处只是概述，正则有哪些应用领域。

正则的应用领域很广，目前已知的有：

- 编程语言及第三方库
 - Perl 的 PCRE
 - PHP
 - Python 的 re
 - BeautifulSoup
 - JavaScript
 - C#
 - Java 的 java.util.regex
 - ActionScript 的 RegExp
 - Objective-C
 - Swift 的 NSRegularExpression
- 编辑器和IDE
 - VSCode
 - Sublime
 - Notepad++
 - Editplus
 - UltraEdit
- 数据库及相关工具
 - MySQL
 - Sequel Pro
 - MongoDB
 - Mongo Compass

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-08
18:14:02

编程语言

如前所述，正则在很多编程语言中也有应用，以及很多都有内置的正则的库：

Python 的 re

详见：

[Python中的正则表达式：re模块详解](#)

BeautifulSoup

BeautifulSoup 中的 `find` 和 `findAll` 的 `name` 或 `attr` 参数，也支持正则

详见：

[【教程】BeautifulSoup中使用正则表达式去搜索多种可能的关键字 – 在路上](#)

Perl 的 PCRE

PERL语言中的正则的库：`pcre`

官网：[PCRE - Perl Compatible Regular Expressions](#)

PHP

JavaScript

最新 js 的规范中：

[ECMAScript 2018 语言规范正式发布，改进正则表达式 - 开源中国社区](#)

支持了一些正则的（其他语言的正则早就支持的）高级功能：

- 支持 s (dotAll) 模式
- Unicode 属性转义 (Property Escape)
- 支持后行断言 (Lookbehind Assertions)
- 命名捕获组 (named capture group)

js的正则的心得

问号要写成 \\\? 而不是 \?

详见：[【已解决】js中正则匹配问号出错：Invalid regular expression Nothing to repeat](#)

括号 [] 中的 \w 要写成 \\\w 才可以

详见：[【已解决】js中正则无法匹配短横线](#)

js中对命名的组支持的不好

详见： [\[不去解决\] js中的正则如何写named group命名的组](#)

C#

Java 的 `java.util.regex`

ActionScript 的 `RegExp`

Objective-C

Swift 的 `NSRegularExpression`

swift中也支持正则

但不是很好用：

[\[未解决\] Swift中字符串的正则表达式处理：判断字符串是否符合某个类型 – 在路上](#)

Ruby

有个网站专门用于测试Ruby正则的网站：

[Rubular a Ruby regular expression editor](#)

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-09
21:58:47

工具和软件

很多工具和软件中的搜索和替换功能中，往往也支持正则：

VSCode

Sublime

Sublime中也支持正则。

具体的语法：

```

> Regular Expressions Cheat Sheet
> A regular expression specifies a set of strings that matches it. This cheat sheet is based off Python 3's Regular Expressions (http://docs.python.org/3/library/re.html) but is designed for searches within Sublime Text.

> Special Characters
\ Escapes special characters or signals a special sequence.
. Matches any single character except a newline.
^ Matches the start of the string.
$ Matches the end of the string.
* Greedily matches 0 or more repetitions of the preceding RE.
*? Matches 0 or more repetitions of the preceding RE.
+ Greedily matches 1 or more repetitions of the preceding RE.
+? Matches 1 or more repetitions of the preceding RE.
? Greedily matches 0 or 1 repetitions of the preceding RE.
?? Matches 0 or 1 repetitions of the preceding RE.
A|B Matches A, if A is unmatched then matches B, where A and B are arbitrary REs.
{m} Matches exactly m many repetitions of the previous RE.
{m,n} Greedily matches from m many to n many repetitions of the previous RE.
{m,n}? Matches m many to n many repetitions of the previous RE.

[...] Indicates a set of characters to match.
[amk] Matches 'a', 'm', or 'k'.
[a-z] Matches 'a' through 'z'.
[a-f0-7] Matches 'a' through 'f' or '0' through '7'.
[a\,z] Matches 'a', '\', or 'z'.
[a-] Matches 'a' or '-'.
[-a] Matches 'a' or '-'.
[(+*)] Matches '(', '+', '*', or ')'. [] matches special characters literally.
[\w] Matches the character class for '\w'. See character classes.
[^5] Matches anything other than '5'. '^' forms the complementary set only as the first character in a set.
[]() Matches ']', '(', and ')'. ']' is taken literally only as the first character in a set.
[()\\] Matches ']', '(', and ')'.

(...) Matches the RE inside the parenthesis and assigns a new group.
(?P<name>...) The RE matched is accessible by the group indicated by name.

(?:...) Extension notation which changes a RE's behavior. These do not assign a new group.
(?:aiLmsux) Sets the corresponding flag to each letter. Does not work within Sublime Text.
(?:...) A non-capturing version of parenthesis. The matched substring cannot be retrieved later.
(?P name) Matches the substring matched by the group named name.
(?#...) A comment, the contents are ignored.
(?) Lookahead assertion, the preceding RE only matches if this matches.
(?) Negative lookahead assertion, the preceding RE only matches if this doesn't match.
(?) Positive lookbehind assertion, the following RE will only match if preceded with this fixed length RE.
(?) Negative lookbehind assertion, the following RE will only match if not preceded with this fixed length RE.

> Character classes
\1 Matches the contents of the group labelled by the same number. Acceptable numbers are 1-99.
\A Matches at the start of the current string.
\b Matches the empty string at the beginning or end of a word. \b matches the boundary between \w and \W.

```

\B	Matches the empty string not at the beginning or end of a word.
\d	Matches any Unicode decimal digit, including 0-9.
\D	Matches any Unicode non-decimal digit.
\s	Matches any Unicode whitespace character, including ' ', \t, \n, \r, \f and \v.
\S	Matches any Unicode non-whitespace character.
\w	Matches any Unicode word character, including a-z, A-Z, and 0-9.
\W	Matches any Unicode non-word character.
\Z	Matches at the end of the string.
\a	Matches the ASCII Bell.
\f	Matches the ASCII Formfeed.
\n	Matches the ASCII Linefeed.
\r	Matches the ASCII Carriage Return.
\t	Matches the ASCII Horizontal Tab.
\v	Matches the ASCII Vertical Tab.

Notepad++

Editplus

UltraEdit

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-08
21:57:01

数据库

正则在数据库方面也有一些应用：

- MySQL
 - Sequel Pro
- MongoDB
 - Mongo Compass

MySQL

MySQL中通配符中的逻辑，也是基本上都是参考了正则的语法和含义：

与LIKE搭配的通配符语法	含义	说明
%	0或多个字符	类似于正则（或Windows系统中Access）中的：星号 *
-	单个字符	类似于正则（或Windows系统中Access）中的：？问号
[AllowableCharList]	允许的字符的列表	
[^NotAllowableCharList] 或 [!NotAllowableCharList]	不允许的字符的列表	

详见：

[【整理】MSQL中的通配符搜索：LIKE和% – 在路上](#)

MySQL数据库工具：[Sequel Pro](#)

详见：

[【基本解决】Sequel Pro中设置通配符正则的过滤条件](#)

MongoDB

Python中用Mongo中去搜索文件名，通过regex实现不区分大小写：

[【未解决】Mongo中让搜索支持不区分大小写](#)

根据[官网解释](#)

MongoDB 中的 \$regex 用的是：[PCRE = Perl Compatible Regular Expressions](#)

对应语法：

[\\$regex — MongoDB Manual](#)

其中也有：

- \$options
 - i
 - m

- x
- s
- 等等

和其他地方，比如Python中也很类似：

[re — Regular expression operations — Python 3.7.2 documentation](#)

- `re.I = re.IGNORECASE`
- `re.M = re.MULTILINE`
- `re.S = re.DOTALL`
- `re.X = re.VERBOSE`

MongoDB的GUI工具：Mongo Compass

类似的，MongoDB的图形化工具 `Mongo Compass`，在界面中搜索内容，也支持有限的正则表达式。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-09 22:18:30

其他领域

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-08
17:59:37

正则心得

下面整理一些，在正则使用和开发期间的一些心得。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-08
22:10:23

正则调试工具

对于：

- 写正则：把想要的规则用正则语法写出来
- 看别人的正则：看懂已有正则的含义

期间，可以借助于一些好用的工具，更好的实现自己的目的。

正则可视化

jex.im

新发现一个网站，可视化效果更好：

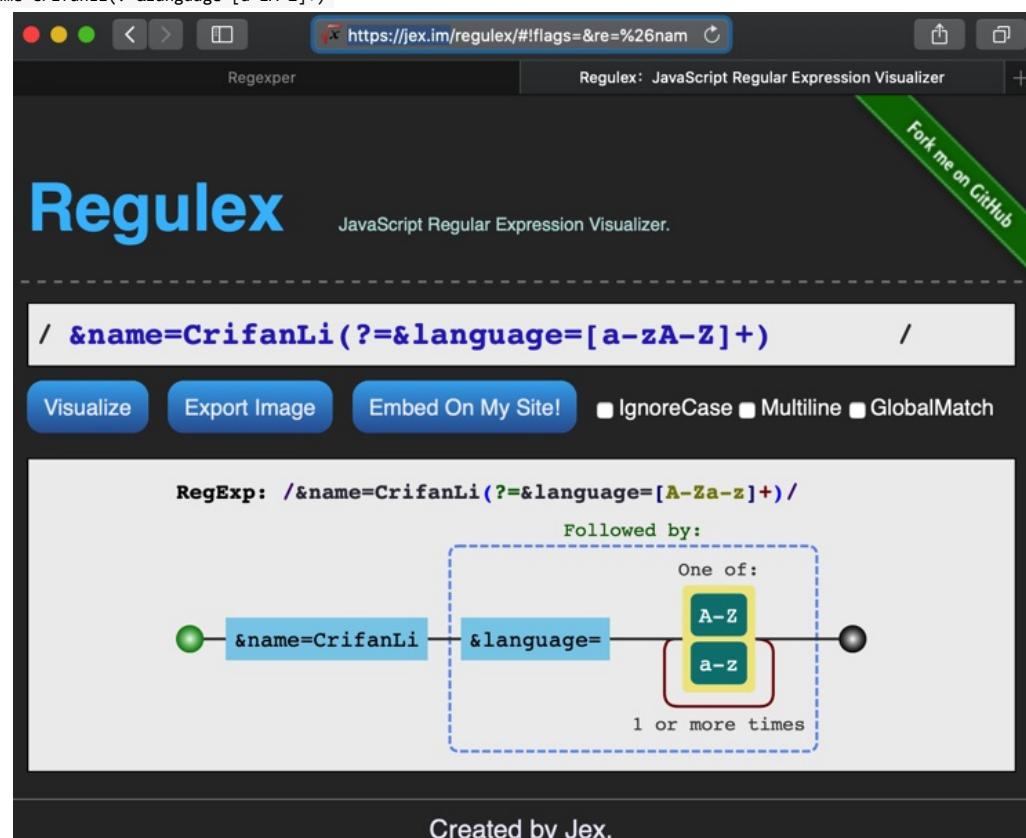
- 网站：[Regulex: JavaScript Regular Expression Visualizer](https://jex.im/regulex/)

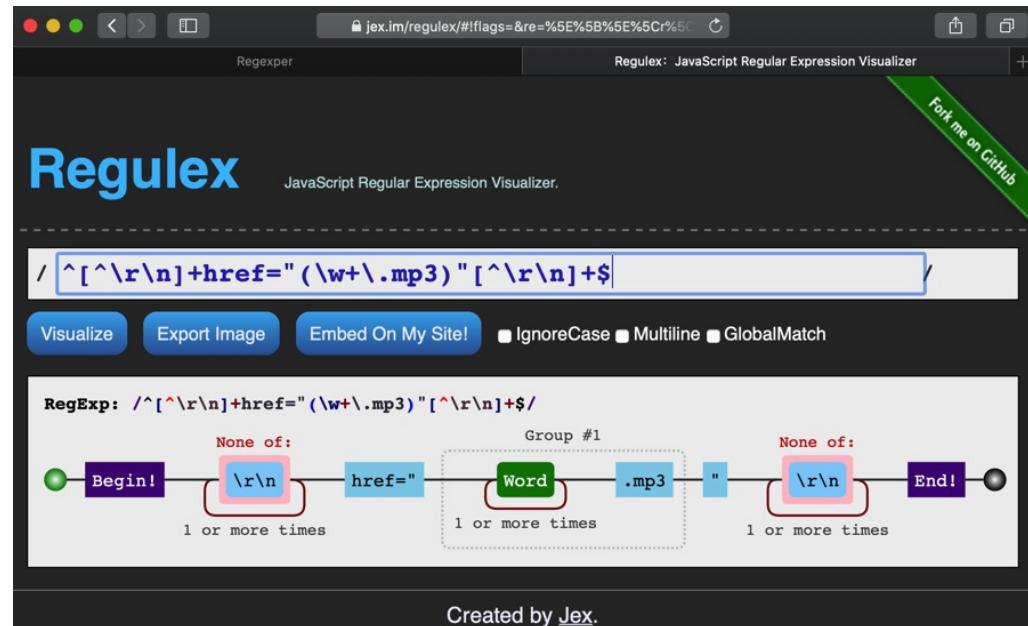
○ 特点：

- 可视化效果好看
- 支持更多选项

○ 举例

- `&name=CrifanLi(?=&language=[a-zA-Z]+)`





regexer.com

- 网站: <https://regexer.com>
 - 特点:
 - 可以把正则表达式，很好的可视化，便于人们理解其内部逻辑和含义
 - 能识别高级语法，如 `look ahead`
 - 效果:
 - `&name=CrifanLi(?=&language=[a-zA-Z]+)`

Regxer

vscode 插件: 正则表达式预览
javallone/regexper-static... +

REGEXPER

You thought you only had two problems... ↗ Changelog
↗ Documentation
↗ Source on GitLab

`&name=CrifanLi(?=&language=[a-zA-Z]+)`

Display [Download SVG](#) // [Download PNG](#) // [Permalink](#)

positive lookahead
One of:
"a" - "z"
"A" - "Z"

Created by Jeff Avallone // Generated images licensed: [\(cc\) BY](#)

`^[^\r\n]+href="(\\w+\\.mp3)"[^\\r\\n]+$`

Regxer

You thought you only had two problems... ↗ Changelog
↗ Documentation
↗ Source on GitLab

`^[^\r\n]+href="(\\w+\\.mp3)"[^\\r\\n]+$`

Display [Download SVG](#) // [Download PNG](#) // [Permalink](#)

Start of line

None of:
carriage return (0x0D)
line feed (0x0A)

'href='

group #1

word

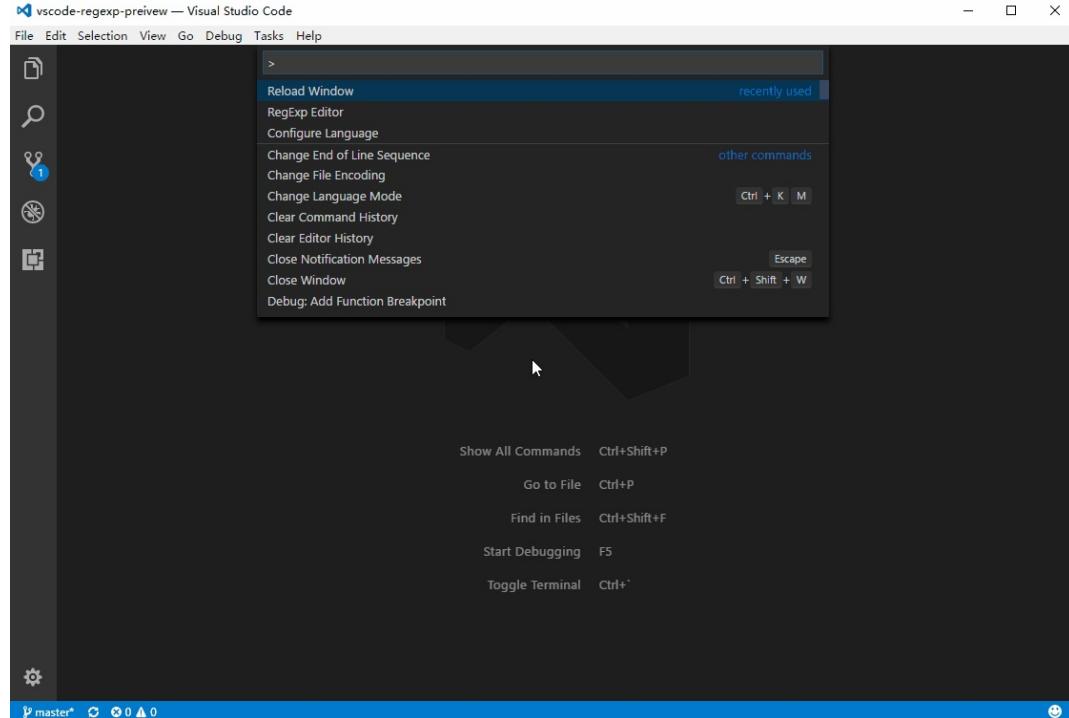
'.mp3'

None of:
carriage return (0x0D)
line feed (0x0A)

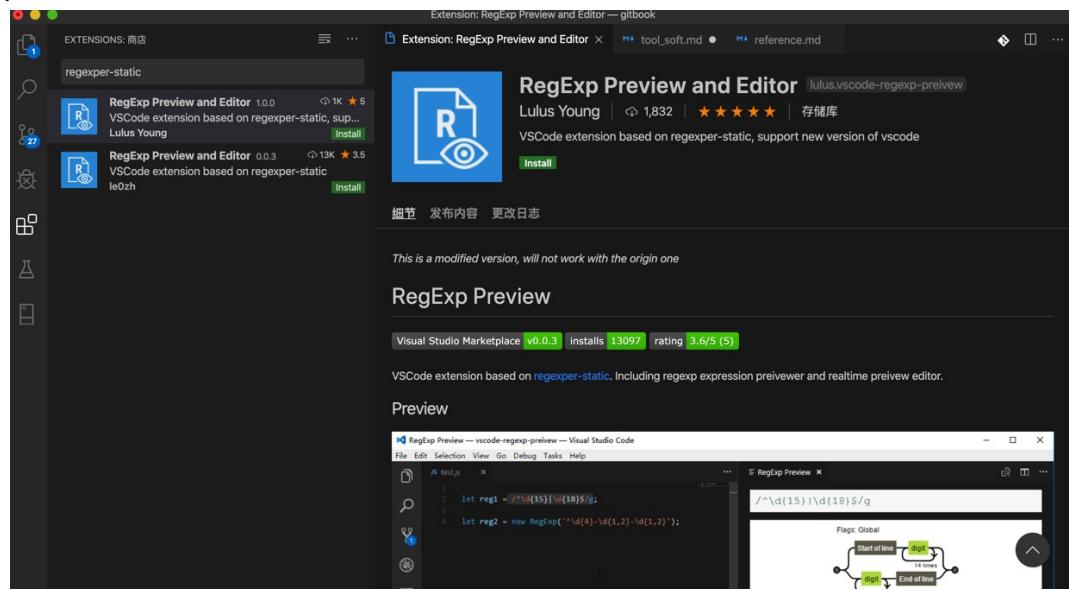
End of line

Created by Jeff Avallone // Generated images licensed: [\(cc\) BY](#)

- VSCode的插件: `regexper-static`
 - 效果: 插件官方gif动图



- 安装:

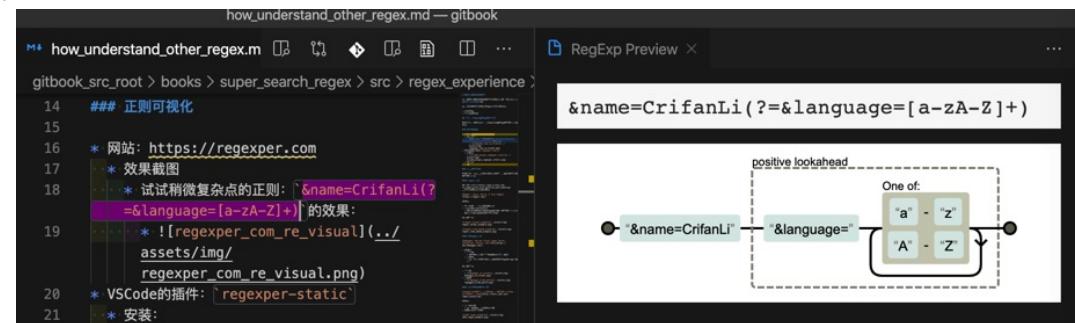


- 使用:

- 选中要可视化的正则表达式字符串->右击-> RegExp Preview



- 效果：



在线正则调试网站

另外，有几个在线网站，可以用来学习和理解正则，调试正则表达式：

regexr.com

之前折腾[这个](#)期间，发现个好用的网站：

[RegExr: Learn, Build, & Test RegEx](https://www.regexr.com)

优点：

- 匹配内容鼠标移动可高亮显示
- 每个字符精确含义解释
 - 可以结构化的，层次清晰的，列出正则表达中每个字符，每个group组等的详细含义。

效果如图：

Expression: /<.+?>/g

Text:

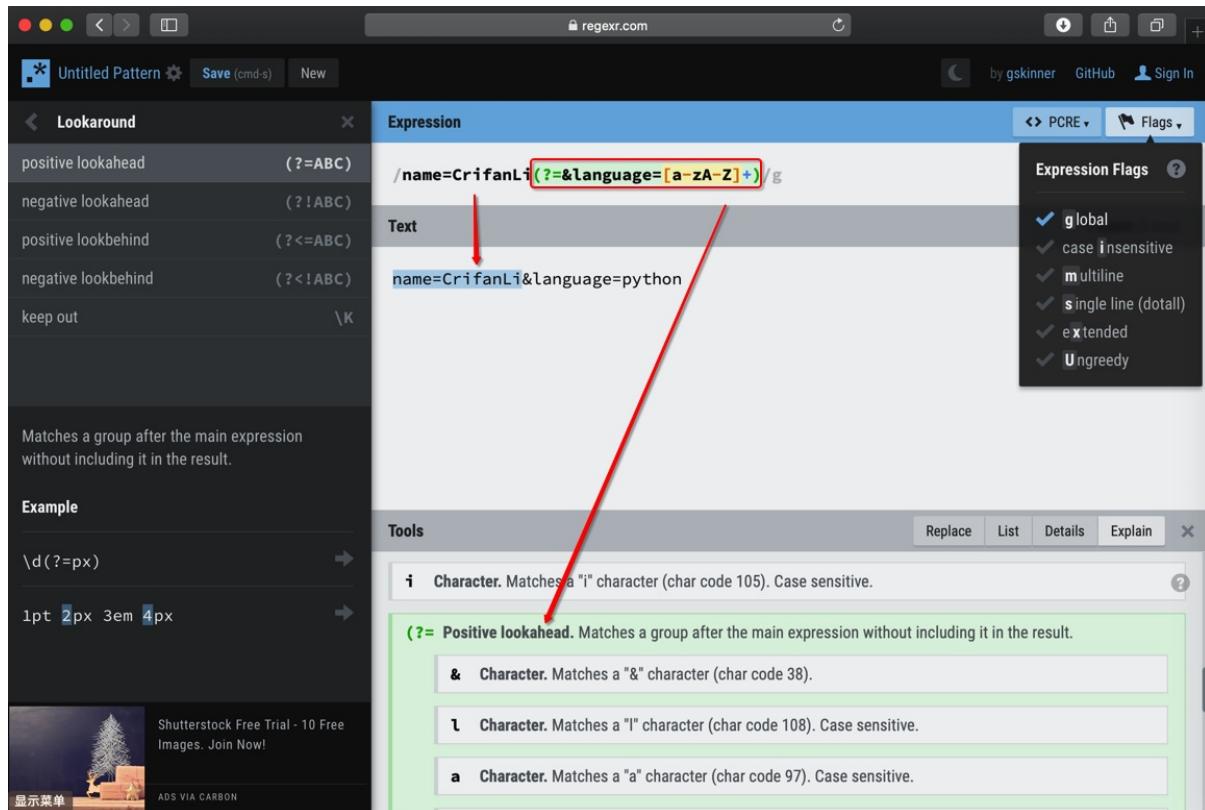
```
<anyChar>
<anyChar>
[<anyChar>]
<testABC123%^&+->
[<testABC123%^&+->
<123testABC%^&+->
[<ABC123%^&+-test>]
```

Tools:

Roll-over elements below to highlight in the Expression above. Click to open in Reference.

- < Character. Matches a "<" character (char code 60).
- . Dot. Matches any character except line breaks.
- + Quantifier. Match 1 or more of the preceding token.
- ? Lazy. Makes the preceding quantifier lazy, causing it to match as few characters as possible.

鼠标移动到上面能实时对应上面的正则中的字符



debuggex.com

[Debuggex: Online visual regex tester. JavaScript, Python, and PCRE](#)

- 优点:
 - 图形化
 - 正则表达式的图形化的表示，便于理解
 - 进度条
 - 对于匹配内容，拖动进度条可以清晰的看到是否匹配

效果如图：

- 图形化

- - 进度条

◦

脚本之家的正则测试

[JavaScript正则在线测试工具 - 正则表达式工具 - 脚本之家在线工具](#)

优点：

- 是中文网站
 - 对于英文不熟悉的容易看懂
- 高亮显示匹配内容

JavaScript正则表达式在线测试工具

本工具提供了多个版本的JS正则在线测试工具，每一个版本都各具特色，功能不一，欢迎大家使用。

脚本之家版

着色功能版

替换功能版

正则和匹配结果会自动着色，如果有匹配结果将会用黄色和蓝色交替着色。

匹配选项： 全局 不区分大小写 对^\$前后换行也支持 符号.匹配所有

正则表达式：(必填)

```
<.+?>
```

要匹配的文本：(必填)

```
<anyChar>
<anyChar>
[<anyChar>]
<testABC123%&+->
[ testABC123%&+->
<123testABC%^&+->
[ <ABC123%^&+-test>]
```

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-08
22:59:02

如何看懂别人的正则

关于如何看懂别人写的正则，核心思路就一句话：从左到右，挨个分析。

对于已有正则，想要去分析其含义，可以：

- 自己分析
 - 关于如何分析，详见：
 - [【教程】如何教你看懂复杂的正则表达式 – 在路上](#)
- 借助别的（正则调试）工具
 - 详见前面整理的：正则调试工具

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-12
20:18:22

如何自己写正则

如何如何写正则，如何写出自己需要的，相对复杂的正则。

详见：

[【教程】以Python中的re模块为例，手把手教你，如何从无到有，写出相对复杂的正则表达式 – 在路上](#)

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新：2020-02-12 20:20:57

附录

下面列出相关参考资料。

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-08
17:47:33

参考资料

- [re — Regular expression operations — Python 3.8.1 documentation](#)
- [re --- 正则表达式操作 — Python 3.8.1 文档](#)
- [正则表达式学习心得](#)
- [Replacement Strings Reference: Context and Case Conversion](#)
- [Popular Tools, Utilities and Programming Languages That Support Regular Expressions](#)
- [【问题解答】正则表达式里边<.+?>什么意思](#)
- [RegExr: Learn, Build, & Test RegEx](#)
- [Debuggex: Online visual regex tester. JavaScript, Python, and PCRE](#)
- [JavaScript正则在线测试工具 - 正则表达式工具 - 脚本之家在线工具](#)
- [PCRE - Perl Compatible Regular Expressions](#)
- [Python中的正则表达式：re模块详解](#)
- [\\$regex — MongoDB Manual](#)
- [re — Regular expression operations — Python 3.7.2 documentation](#)
- [【教程】BeautifulSoup中使用正则表达式去搜索多种可能的关键字 – 在路上](#)
- [Sublime Text \[Regular expression - Cheat sheet\]](#)
- [Sublime Text Regular Expression Cheat Sheet - jdiao's blog](#)
- [【已解决】VSCode中如何使用正则表达式去替换且被替换中使用分组group](#)
- [Search and Replace — Sublime Text Unofficial Documentation](#)
- [Perl Regular Expression Syntax - 1.44.0](#)
- [Perl Format String Syntax - 1.44.0](#)
- [Regex in Sublime Text by Jake Albaugh on CodePen](#)
- [vscode插件：正则表达式预览 | le0zh's blog](#)
- [javallone/regexpers-static: Regular Expression Visualization Site](#)
- [分享5个可视化的正则表达式编辑工具-CSDN.NET](#)
- [CJex/regulex: Regular Expression Excited!](#)
- [rscarvalho/pyrex: pyrex is a Python Regular Expression Online Tester](#)
- [PyRegex](#)
- [Pythex: a Python regular expression editor](#)
- [re — Regular expression operations — Python 3.8.1 documentation](#)
- [Regex Tester - Javascript, PCRE, PHP](#)
- [Rubular a Ruby regular expression editor](#)
- [你是如何学会正则表达式的？ - 知乎](#)
- [learn-regex/README-cn.md at master · ziishaned/learn-regex](#)
- [“正则”是什么意思？ - 知乎](#)
- [正则表达式 - JavaScript | MDN](#)
- [JavaScript 正则表达式 | 菜鸟教程](#)
- [JavaScript RegExp 对象](#)
- [javascript正则表达式 - 司徒正美 - 博客园](#)
- [JS正则表达式入门，看这篇就够了 - 个人文章 - SegmentFault 思否](#)
- [How do you access the matched groups in a JavaScript regular expression](#)
- [Capturing groups](#)
- [String.prototype.match\(\) - JavaScript | MDN](#)
- [【已解决】js中正则匹配问号出错：Invalid regular expression Nothing to repeat](#)
- [【已解决】js中正则无法匹配短横线](#)
- [\[不去解决\] js中的正则如何写named group命名的组](#)
- [\[未解决\] Swift中字符串的正则表达式处理：判断字符串是否符合某个类型 – 在路上](#)
- [【基本解决】Sequel Pro中设置通配符正则的过滤条件](#)

- [【整理】MSQL中的通配符搜索：LIKE和% – 在路上](#)
- [正则表达式中的断言\(assertions\) - 简书](#)
- [正则表达式：零宽断言\(lookaround\) - 许炎的个人博客](#)
- [re — Regular expression operations — Python 3.8.1 documentation](#)
- [re --- 正则表达式操作 — Python 3.8.1 文档](#)
- [正则表达式：零宽断言\(lookaround\) - 许炎的个人博客](#)
- [Lookahead and Lookbehind Tutorial—Tips & Tricks](#)
- [Regex defined zero-width assertions - Java 9 Regular Expressions](#)
- [Zero-width assertions - Java 9 Regular Expressions](#)
- [Regex Tutorial - Lookahead and Lookbehind Zero-Length Assertions](#)
- [Regex Tutorial - \b Word Boundaries](#)
- [Regex Tutorial - Start and End of String or Line Anchors](#)
- [Zero-Width Assertions « Quantifiers « Regular Expressions « C# Book](#)
- [正则表达式——02：零宽断言 - 简书](#)
- [正则表达式后行断言 · 探索 ES2018 和 ES2019 - 众成翻译](#)
- [正则表达式之——先行断言\(lookahead\)和后行断言\(lookbehind\)_赶路人儿-CSDN博客](#)
- [People Looking Down Png - People Standin #1297885 - PNG Images - PNGio](#)
- [WAIPER RAKUTENICHIBATEN: HOUSTON Houston U.S. forces CWU-45/P flight jacket 5CW45P/ military forces thing men | Rakuten Global Market](#)
- [【已解决】Mac中编辑图片水平左右翻转](#)
- [【教程】如何教你看懂复杂的正则表达式 – 在路上](#)
- [【教程】以Python中的re模块为例，手把手教你，如何从无到有，写出相对复杂的正则表达式 – 在路上](#)
-

crifan.com, 使用[署名4.0国际\(CC BY 4.0\)协议](#)发布 all right reserved, powered by Gitbook最后更新: 2020-02-12 20:21:01