正则表达式

Regular Expressions

https://xkcd.com/208/

使用场景

- 海量文本查找 & 替换
  - 从200MB日志文件中提取所有conversationId
  - 从几百万行gclog中查找耗时>1s的日志
  - 删除所有行首的空格
- 数据校验
  - Email / 手机号

支持正则的应用

- 编程语言
- 文本编辑器
- IDE
- Linux命令：grep, sed, awk
  - grep => g/re/p
  - **g**lobal search with **re**gex and **p**rint

定义：描述规则的语言

发现字符串背后的规则

https://alf.nu/RegexGolf

Backrefs – This doesn't really work as a tutorial. Not really clear what you're supposed to do here.

0

Match all of these…

- ✔ allochirally
- ✔ anticovenanting
- ✔ barbary
- ✔ calelectrical
- ✔ entablement
- ✔ ethanethiol
- ✔ froufrou
- ✔ furfuryl
- ✔ galagala
- ✔ heavyheaded
- ✔ linguatuline
- ✔ mathematic
- ✔ monoammonium
- ✔ perpera
- ✔ photophonic
- ✔ purpuraceous
- ✔ salpingonasal
- ✔ testes
- ✔ trisectrix
- ✔ undergrounder
- ✔ untaunted

and none of these…

- ✘ anticker
- ✘ corundum
- ✘ crabcatcher
- ✘ damnably
- ✘ foxtailed
- ✘ galvanotactic
- ✘ gummage
- ✘ gurniad
- ✘ hypergoddess
- ✘ kashga
- ✘ nonimitative
- ✘ parsonage
- ✘ pouchlike
- ✘ presumptuously
- ✘ pylar
- ✘ rachioparalysis
- ✘ scherzando
- ✘ swayed
- ✘ unbridledness
- ✘ unupbraidingly
- ✘ wellside

举例来说

- abc -> abc ac
- 13个数字 -> 13312345678 13312345
- http或https -> http https http
- key=后面的词 -> key=value

方言 / flavors

- 正则发明于1950年代，发展到现在有各种不同的实现，支持的特性也不同，PCRE、POSIX、ECMA…
- 参考：https://en.wikipedia.org/wiki/Comparison_of_regular_expression_engines
- 使用前先看说明

	"+" quantifier	Negated character classes	Non-greedy quantifiers[Note 1]	Shy groups[Note 2]	Recursion	Look-ahead	Look-behind	Backreferences[Note 3]	>9 indexable captures
Boost.Regex	Yes	Yes	Yes	Yes	Yes[Note 4]	Yes	Yes	Yes	Yes
Boost.Xpressive	Yes	Yes	Yes	Yes	Yes[Note 5]	Yes	Yes	Yes	Yes
CL-PPCRE	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
EmEditor	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No
FREJ	No[Note 6]	No	Some[Note 6]	Yes	No	No	No	Yes	Yes
GLib/GRegex	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
GNU grep	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	?
Haskell	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Helios RXPF	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
ICU Regex	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Java	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
JavaScript (ECMAScript)	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes
JGsoft	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Lua	Yes	Yes	Some[Note 7]	No	No	No	No	Yes	No
.NET	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
OCaml	Yes	Yes	No	No	No	No	No	Yes	No
OmniOutliner 3.6.2	Yes	Yes	Yes	No	No	No	No	?	?
PCRE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Perl	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
PHP	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Python	Yes	Yes	Yes	Yes	Yes[Note 8]	Yes	Yes	Yes	Yes
Qt/QRegExp	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes
R[Note 9]	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
RE2	Yes	Yes	Yes	Yes	No	No	No	No	Yes
Ruby	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
TRE	Yes	Yes	Yes	Yes	No	No	No	Yes	No
Vim	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No
RGX	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Tcl	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
TRegExpr	Yes	?	Yes	?	?	?	?	?	?
XRegExp	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes

正则基础 / PCRE

- Characters 字符
- Meta-characters 元字符
  -) ( ] [\ ^ $. | ? * +

Characters / 字符

- 直接匹配
- 字母：A-Z, a-z
- 数字：0-9
- 符号：!,@,#,%…

ps：meta-characters作为字面量直接匹配时使用\转义

Classes / 集合

- [ ]
- [Aa] : 匹配一个 A 或 a
- [a-z] : 匹配任意一个小写字母一次
- [^Aa] : 匹配不是 A 或 a 的字符一次
- [-A^a] : 匹配-,A,a或^

ps: ^位于集合内首位才作为元字符

集合别名

- \d
  - 匹配一个数字字符，等价于[0-9]
- \w
  - 匹配包括下划线的任何单词字符，等价于[A-Za-z0-9_]，注意Unicode正则表达式会匹配中文字符
- \s
  - 匹配任何空白字符，等价于 [\t(?:\n|\r\n)]
- \D, \W, \S : 上述取反

Dot / 点

- `.`
- 匹配任意非换行符，等价于 `[^\n]`
- 开销较高
- 匹配整个字符串 `.*`

Quantifiers 量词

- `{n}` : 匹配n次
- `{n,}` : 匹配至少n次
- `{n,m}` : 匹配n到m次
- `*` : 等价于 `{0,}`
- `+` : 等价于 `{1,}`
- `?` : 等价于 `{0,1}`

Non Greedy Quantifiers / 非贪婪量词

示例文本：`<hello>world</hello>`

- 量词默认是贪婪的（即匹配尽可能多的字符）
  - `<.*>` 匹配`<hello>world</hello>`
- 量词后面加上? 表示非贪婪（按方言可能有所差异）
  - `<.*?>` 匹配`<hello>`

示例

- 查找执行超过100ms的方法
  - 示例文本：method elapse:123 ms
  - \d{3,}
- 匹配http或https
  - https?
- 匹配json串中的kdtId
  - 示例文本“kdtId”:18116282,“name”:“123”
  - “kdtId”:[^,]+

Alternation / 选择

- |
- 匹配左侧或右侧的表达式，优先级最低
- `http|https`
- 和集合的区别?
- 选择的分支可以是任意长度字符串

Anchors / 锚

- 配置文本中的**位置**而非字符
- ^ ： 文本起始位置
- $ ： 文本结束位置
- \b ： 单词边界，\w和[^\w]之间的位置

示例

- `^\s+|\s+$`
- 匹配行首或行尾的空白字符

Groupings / 分组

- ( )
- 聚合
  - (abc)+, (a|b)c
- 捕获
  - 示例表达式：(\d\d):(\d\d):(\d\d)
  - 示例文本：11:23:15
    - match: 11:23:15
    - group(1): 11, group(2): 23, group(3): 15
  - 捕获的组可以在表达式中引用（backreference）
    - ([‘“]).+\1

https://regex101.com/r/cuggER/1

示例

- 示例文本

```
541959335 [DubboServerHandler-10.19.8.255:20883-thread-91] INFO c.y.s.c.c.c.a.DubboServiceAspect - [RPC SUCCESS] ImUsersServiceImpl . getOnlineAdmins with param:[{"kdtId":"40812901","channel":"dkf"}] response:{"data":[],"count":0,"success":true,"code":200,"message":"successful"} elapsed:5
```

- 统计方法名及耗时

```
\[RPC \w+\] (.+) with param.*elapsed:(\d+)
```

```
Group 1.    113-149 `ImUsersServiceImpl . getOnlineAdmins`
Group 2.    288-289 `5`
```

```
➜ ~ cat regdemo
541959335 [DubboServerHandler-10.19.8.255:20883-thread-91] INFO c.y.s.c.c.c.a.DubboServiceAspect - [RPC SUCCESS] ImUsersServiceImpl . getOnlineAdmins with param:[{"kdtId":"40812901","channel":"dkf"}] response:{"data":[],"count":0,"success":true,"code":200,"message":"successful"} elapsed:5
➜ ~ sed -n 's/.*\[RPC .*\] \(.*\) with param.*elapsed:\([0-9]*\)/\1 \2/p' regdemo
ImUsersServiceImpl . getOnlineAdmins 5
```

https://regex101.com/r/8yxtfn/1

Zero-Length Assertions / 零宽断言
Lookahead and Lookbehind / 正向预查&反向预查

- 正向肯定预查 `(?=pattern)`
  - `Win(?=95|98)` 匹配Win95中的Win，不匹配Win7中的Win
  - 预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始
  - 用`Win(?=95|98)(\d+)` 匹配 `Win9500`，group(1)是什么?
- 正向否定预查 `(?!pattern)`
- 反向肯定预查 `(?!pattern)`
  - `(?<=kdtId=)\d+` 匹配kdtId=123中的123，不匹配fansId=123中的123
- 反向否定预查 `(?<!pattern)`

Recursion

- `az,aazz,aaazzz`
  - `a(?R)?z`
- `deed, abcdcba`
  - `\b(?'word'(?'oddword'(?'oddletter'[a-z])(?P>oddword)\k'oddletter'|[a-z])|(?'evenword'(?'evenletter'[a-z])(?P>evenword)?\k'evenletter'))\b`

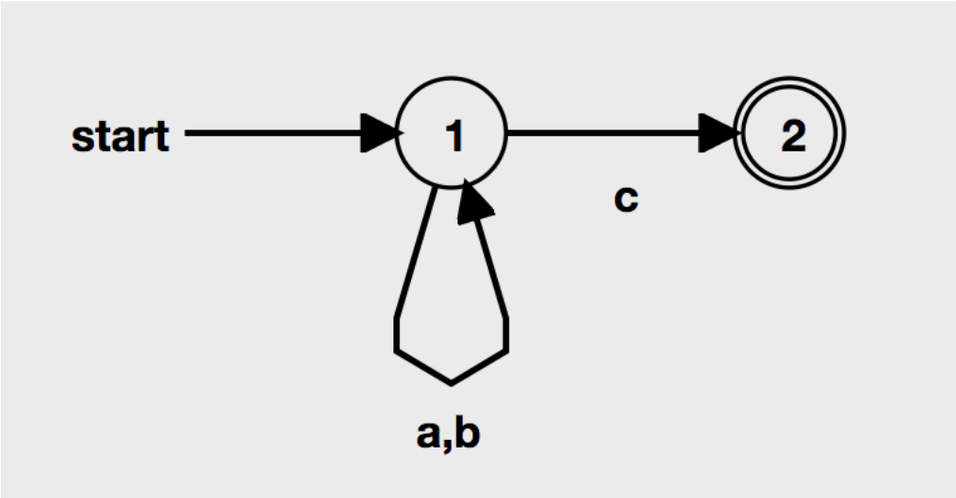
https://www.regular-expressions.info/recurse.html

DFA & NFA

- 确定有限状态自动机（Deterministic Finite Automaton）：对每个状态和输入符号对有**唯一**的下一个状态
- 非确定有限状态自动机（Non-deterministic Finite Automaton）：对每个状态和输入符号对可以有**多个可能**的下一个状态

DFA

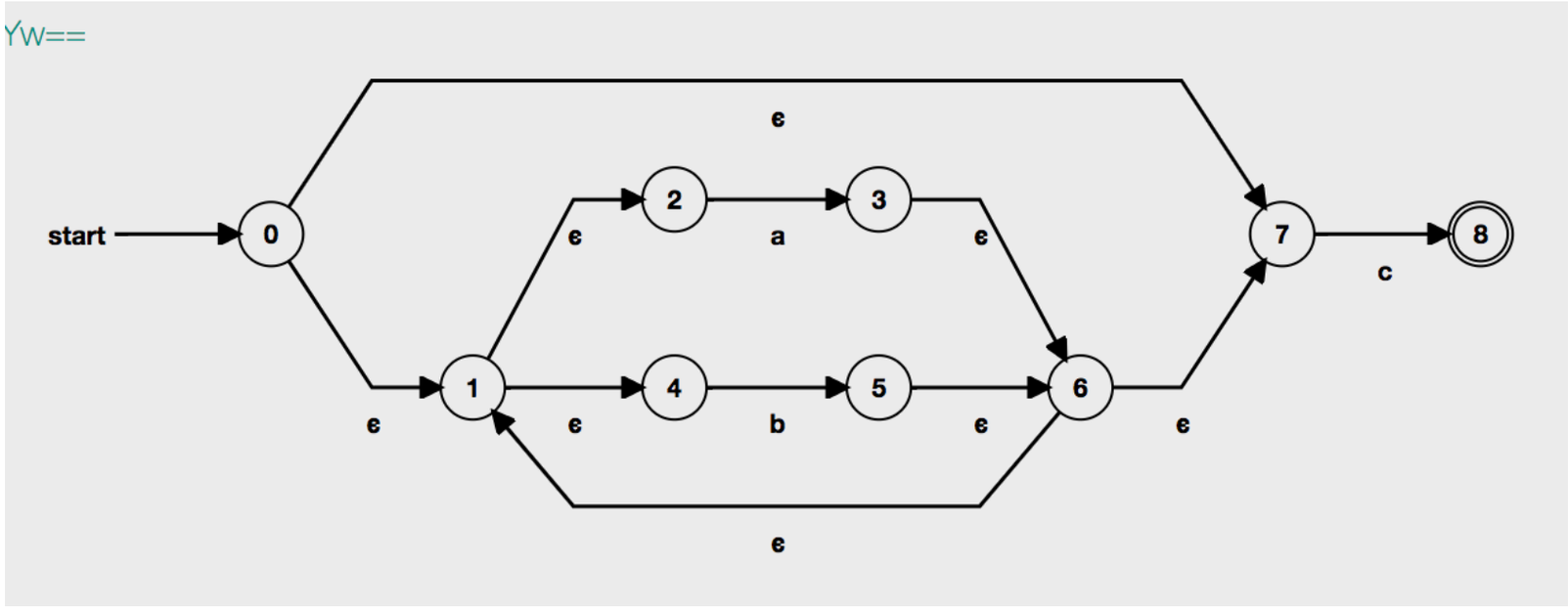
- `(a|b)*c`



https://cyberzhg.github.io/toolbox/min_dfa?regex=KGF8YikqYw==

NFA

- `(a|b)*c`



https://cyberzhg.github.io/toolbox/regex2nfa?regex=KGF8YikqYw==

Demo

- http://ivanzuzak.info/noam/webapps/fsm_simulator/

DFA vs NFA

- DFA是基于文本的（Text-Directed），去匹对相配正则表达式，文本中每个字符只会扫描一遍，O(n)
- NFA是基于表达式的（Regex-Directed），去匹对相配文档，可能会多次扫描文本中同一个字符（回溯）

DFA vs NFA

`abc => ab|abc`

- DFA，返回最长匹配，abc
- NFA，返回最左匹配，ab
- POSIX NFA，返回最长匹配 abc

DFA vs NFA

`abc => abd|abc` (NFA)

-

1. /abd|abc/ abc
2. /abd|abc/ abc
3. /abd|abc/ abc
4. /abd|abc/ abc
5. /abd|abc/ abc
6. /abd|abc/ abc
7. /abd|abc/ abc
8. /abd|abc/ abc

常见应用对应的引擎类型

引擎类型	程序
DFA	awk(大多数版本)、egrep（大多数版本）、flex、lex、MySQL、Procmail
传统型 NFA	GNU Emacs、Java、grep（大多数版本）、less、more、.NET语言、PCRE library、Perl、PHP（所有三套正则库）、Python、Ruby、set（大多数版本）、vi
POSIX NFA	mawk、Mortice Lern System's utilities、GUN Emacs（明确指定时使用）
DFA/NFA混合	GNU awk、 GNU grep/egrep、 Tcl

Stack Overflow Outage Postmortem

- On July 20, 2016 we experienced a 34 minute outage starting at 14:44 UTC
- **^[\s\u200c]+|[\s\u200c]+$**
- The malformed post contained roughly 20,000 consecutive characters of whitespace on a comment line, but not at the end
- Backtracking 20,000+19,999+19,998+…+3+2+1 = 200,010,000 times

http://stackstatus.net/post/147710624694/outage-postmortem-july-20-2016

DEBUG DATA

```
1.  /\s+$/        b
2.  /\s+$/        b
3.  /\s+$/        b
4.  /\s+$/        b
5.  /\s+$/        b
6.  /\s+$/        b
7.  /\s+$/        b
8.  /\s+$/        b
9.  /\s+$/        b
10. /\s+$/        b
11. /\s+$/        b
12. /\s+$/        b
13. /\s+$/        b
14. /\s+$/        b
15. /\s+$/        b
16. /\s+$/        b
17. /\s+$/        b
18. /\s+$/        b
19. /\s+$/        b
```

https://regex101.com/r/VUGv3S/1

Backtracking

参考

- https://www.regular-expressions.info/
- https://regex101.com/
- 精通正则表达式