



戴口罩 防感染

保护自己 也对他人负责



第六讲

(1)

正则表达式

清华大学计算机系





正则表达式

戴口罩 防感染

保护自己 也对他人负责



- 什么是正则表达式？
- 正则表达式有什么用？
- 如何用正则表达式？
- 很多语言都支持正则表达式



什么是正则表达式

➤ 在计算机科学中，指一个用来描述或者匹配一系列符合某个句法规则的字符串的**单个字符串**

✓ 010-\d{8}

✓ 010-627[789]\d{4}

➤ 鼻祖

✓ 美国新泽西州的Warren McCulloch

✓ 美国底特律的Walter Pitts

➤ 推广人

✓ Ken Thompson Unix创始人

✓ Jeffrey Friedl



正则表达式用途

戴口罩 防感染

保护自己 也对他人负责



- 模式匹配：是否满足某一特征的字符串
 - ✓ 匹配电话号码
 - ✓ 匹配URL
- 查找字符串
- 字符串替换



传统的字符串匹配

戴口罩 防感染

保护自己 也对他人负责



➤ 普通字符

- ✓ 字母、数字、汉字、下划线
- ✓ 没有被定义特殊意义的标点符号

➤ 如果每个字符都是普通字符，就是传统字符串匹配

➤ 如何进行复杂匹配？

- ✓ 匹配北京电话号码
- ✓ 匹配Email地址



戴口罩 防感染

保护自己 也对他人负责



通用正则表达式



与“多种字符”匹配的表达式

表达式	作用
<code>\d</code>	任意一个数字，0~9 中的任意一个
<code>\w</code>	任意一个字母、数字或下划线，也就是 A~Z, a~z, 0~9, _ 中任意一个
<code>\s</code>	包括空格、制表符、换页符等空白字符的其中任意一个
<code>.</code>	小数点可以匹配除了换行符（ <code>\n</code> ）以外的任意一个字符
<code>\D</code>	非 <code>\d</code>
<code>\W</code>	非 <code>\w</code>
<code>\S</code>	非 <code>\s</code>



例子

戴口罩 防感染

保护自己 也对他人负责



➤ 北京电话号码

✓ 010- \d\d\d\d\d\d\d

➤ 用户名长度为6的Gmail地址

✓ \w\w\w\w\w\w@gmail.com

匹配长度为100的？



匹配次数的符号

表达式	作用
$\{n\}$	表达式重复n次，比如：" $\backslash w\{2\}$ " 相当于 " $\backslash w\backslash w$ "; " $a\{5\}$ " 相当于 "aaaaa"
$\{m,n\}$	表达式至少重复m次，最多重复n次，比如：" $ba\{1,3\}$ "可以匹配 "ba"或"baa"或"baaa"
$\{m,\}$	表达式至少重复m次，比如：" $\backslash w\backslash d\{2,\}$ "可以匹配 "a12","_456","M12344"...
$?$	匹配表达式0次或者1次，相当于 $\{0,1\}$ ，比如：" $a[cd]?$ "可以匹配 "a","ac","ad"
$+$	表达式至少出现1次，相当于 $\{1,\}$ ，比如：" $a+b$ "可以匹配 "ab","aab","aaab"...
$*$	表达式不出现或出现任意次，相当于 $\{0,\}$ ，比如：" $a*b$ "可以匹配 "b","aaaab"...



例子

戴口罩 防感染

保护自己 也对他人负责



✓. 匹配单个字符

□ r.t

□ 匹配 rat rut

□ 不匹配 root

✓* 匹配0或多个字符

□ r.*t

□ 匹配 rt rot root

✓+ 匹配1或多个字符

□ 9+ 匹配 9999

✓? 匹配0个或1个字符

□ 9? 匹配 9



匹配任意字符

戴口罩 防感染
保护自己 也对他人负责



表达式	作用
[ab5@]	匹配 "a" 或 "b" 或 "5" 或 "@"
[^abc]	匹配 "a" , "b" , "c" 之外的任意一个字符
[f-k]	匹配 "f"~"k" 之间的任意一个字母
[^A-F0-3]	匹配 "A"~"F" , "0"~"3" 之外的任意一个字符



例子

戴口罩 防感染

保护自己 也对他人负责



- `[]` 匹配括号中的任何一个字符
 - ✓ `r[aou]t`
 - ✓ 匹配 `rat`、`rot`、`rut`
 - ✓ 不匹配 `root`
- `[0-9]` 匹配任何数字
- `[a-z]` 匹配a-z之间的任何字符
- `[^a-z]` 匹配除了a-z之间的字符
- 通过`{i}`匹配指定数目的字符
 - ✓ `A[0-9]{3}` 匹配 `A123`、`A348`



转义字符

戴口罩 防感染

保护自己 也对他人负责



➤ 转义字符 \

✓ \\ 表示 \

✓ * 表示 *

✓ \\$ 表示 \$

✓ \. 表示 .



贪婪与懒惰

戴口罩 防感染

保护自己 也对他人负责



➤ 贪婪匹配方式

- ✓ 最长匹配

➤ 表达式 $a.*b$

- ✓ 字符串 aabab

- ✓ 匹配结果 aabab

➤ 怎么得到结果 aab 和 ab?



贪婪与懒惰

戴口罩 防感染

保护自己 也对他人负责



*? 重复任意次，但尽可能少重复

+? 重复1次或更多次，但尽可能少重复

?? 重复0次或1次，但尽可能少重复

{n,m}? 重复n到m次，但尽可能少重复

{n,}? 重复n次以上，但尽可能少重复

表达式 $a.*?b$

字符串 aabab

懒惰匹配结果 aab 和 ab



字符边界

戴口罩 防感染

保护自己 也对他人负责



表达式 `de\w`

字符串 `abcdef`

匹配结果 `def`

如何匹配单词？

表达式	作用
<code>^</code>	与字符串开始的地方匹配，不匹配任何字符
<code>\$</code>	与字符串结束的地方匹配，不匹配任何字符
<code>\b</code>	匹配一个单词边界，也就是单词和空格之间的位置，不匹配任何字符



例子

戴口罩 防感染

保护自己 也对他人负责



✓\$

□are you\$

□匹配 how are you

□不匹配how old are you?

✓^

□^how are

□匹配 how are you

□不匹配how old are you

✓\b

□\bho\b

□匹配 ho are you

□不匹配how old are you



逻辑、分组

戴口罩 防感染

保护自己 也对他人负责



➤ \B 和 \b相反

- ✓ \Bthe\b

- ✓ 不匹配in the world

- ✓ 匹配otherwise

➤ | 或运算

- ✓ 0\d{2}-\d{8}|0\d{3}-\d{7}

➤ () 分组

- ✓ 整体被修饰，可以通过分组提取匹配的信息

- (\d{1,3}\.){3}\d{1,3} \\ ip地址

- ((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)



分组

戴口罩 防感染

保护自己 也对他人负责



➤ 表达式 $((ab^*)c)d)e$

➤ 字符串 abcdef

➤ 分组结果

✓ $\text{group}(0) = abcde$ 整个匹配结果

✓ $\text{group}(1) = abcd$ 第一组结果

✓ $\text{group}(2) = abc$

✓ $\text{group}(3) = ab$



分组

戴口罩 防感染

保护自己 也对他人负责



- `CS`
- 如何匹配出 <http://cs.tsinghua.edu.cn>
- `<a\s*href\s*=\s*"([^\"]*)" \s*title="([^\"]*)" [^>]*>`
- `group(1)`
 - ✓ <http://cs.tsinghua.edu.cn>
- `group(2)`
 - ✓ [CS](#)



注释

戴口罩 防感染

保护自己 也对他人负责



➤ `(?#comment)` 来包含注释

➤ `2[0-4]\d(?#200-249)|25[0-5](?#250-255)|[01]?\d\d(?#0-199)`



例子

戴口罩 防感染

保护自己 也对他人负责



- “ $\text{\$}(\backslash\text{d}+.\text{?}\backslash\text{d}^*)$ ” 匹配 $\text{\$}10.9, \text{\$}20.5$
- $^{\wedge}[\text{A-Za-z}]^+\text{\$}$ 匹配由26个英文字母组成的字符串
- $^{\wedge}[\text{A-Z}]^+\text{\$}$ 匹配由26个英文字母的大写组成的字符串
- $^{\wedge}[\text{a-z}]^+\text{\$}$ 匹配由26个英文字母的小写组成的字符串
- $^{\wedge}[\text{A-Za-z0-9}]^+\text{\$}$ 匹配由数字和26个英文字母组成的字符串
- $^{\wedge}\backslash\text{w}^+\text{\$}$ 匹配由数字、26个英文字母或者下划线组成的字符串



例子

戴口罩 防感染

保护自己 也对他人负责



$\wedge[1-9]\backslash d^*\$$ // 匹配正整数
 $\wedge-[1-9]\backslash d^*\$$ // 匹配负整数
 $\wedge-?[1-9]\backslash d^*\$$ // 匹配整数
 $\wedge[1-9]\backslash d^*|0\$$ // 匹配非负整数 (正整数 + 0)
 $\wedge-[1-9]\backslash d^*|0\$$ // 匹配非正整数 (负整数 + 0)
 $\wedge[1-9]\backslash d^*\backslash.\backslash d^*|0\backslash.\backslash d^*[1-9]\backslash d^*\$$ // 匹配正浮点数
 $\wedge-([1-9]\backslash d^*\backslash.\backslash d^*|0\backslash.\backslash d^*[1-9]\backslash d^*)\$$ // 匹配负浮点数
 $\wedge-?([1-9]\backslash d^*\backslash.\backslash d^*|0\backslash.\backslash d^*[1-9]\backslash d^*|0?\backslash.0+|0)\$$ // 匹配浮点数
 $\wedge[1-9]\backslash d^*\backslash.\backslash d^*|0\backslash.\backslash d^*[1-9]\backslash d^*|0?\backslash.0+|0\$$ // 匹配非负浮点数 (正浮点数 + 0)
 $\wedge-([1-9]\backslash d^*\backslash.\backslash d^*|0\backslash.\backslash d^*[1-9]\backslash d^*)|0?\backslash.0+|0\$$ // 匹配非正浮点数 (负浮点数 + 0)



例子

戴口罩 防感染

保护自己 也对他人负责



- 匹配国内电话号码: `\d{3}-\d{8}|\d{4}-\d{7}`
- 匹配身份证: `\d{15}|\d{18}`
- 匹配ip地址: `\d+\.\d+\.\d+\.\d+`
- 匹配Email地址的正则表达式:
 - ✓ `\w+([-+.] \w+)*@ \w+([-.] \w+)*\.\w+([-.] \w+)*`
- 匹配网址URL的正则表达式:
 - ✓ `https?:\/\/[^\s]*`
- 匹配帐号是否合法(字母开头, 允许5-16字符, 允许字母数字下划线):
 - ✓ `^[a-zA-Z][a-zA-Z0-9_]{4,15}$`



戴口罩 防感染

保护自己 也对他人负责



Python的正则表达式



Python使用正则表达式

戴口罩 防感染

保护自己 也对他人负责



➤ 包re支持正则表达式

- ✓ 字符串匹配
- ✓ 字符串分解
- ✓ 字符串替换



从开头找到第一个匹配match

- `re.match`: 字符串匹配
- `re.match`的函数原型为: `re.match(pattern, string, flags)`
 - ✓ 第一个参数是正则表达式
 - ✓ 第二个参数表示要匹配的字符串;
 - ✓ 第三个参数是标致位, 用于控制正则表达式的匹配方式, 如: 是否区分大小写, 多行匹配等等。
- `re.match`只匹配字符串的开始, 如果字符串开始不符合正则表达式, 则匹配失败, 函数返回`None`; 如果匹配成功, 则返回一个`Match`对象



flags

戴口罩 防感染

保护自己 也对他人负责



- re.I 表示大小写忽略
- re.M 多行模式 使 \wedge $\$$ 匹配除了string开始结束外，还匹配一行的开始和结束
- re.S “.” 匹配包括 ‘ $\backslash n$ ’在内的任意字符，否则 . 不包括 ‘ $\backslash n$ ’
- re.X 为了写正则表达式，会忽略一些空格和#后面的注释
- re.L 本地化 $\backslash w$ 也匹配本地化语言，例如汉字



例子

戴口罩 防感染

保护自己 也对他人负责



```
import re
text = "JGood is a handsome boy, he is cool,
       clever, and so on..."
m = re.match("(\w+)\s", text)
if m:
    print(m.group(), '\n')
else:
    print('not match')
>>> JGood
```




re.M

戴口罩 防感染

保护自己 也对他人负责



```
import re
```

```
print(re.match("^\d+$", "123\n 234\n"))
```

None

```
print(re.match("^\d+$", "123\n 234\n",  
re.M).group())
```

['123']



找到一个结果search

➤ re.search和re.match参数相同

- ✓ re.search函数会在字符串内查找模式匹配,直到找到第一个匹配然后返回,如果字符串没有匹配,则返回None;
- ✓ re.match与re.search的区别: re.match只匹配字符串的开始,如果字符串开始不符合正则表达式,则匹配失败,函数返回None;
- ✓ 而re.search匹配整个字符串,直到找到一个匹配。



如何得到返回结果

戴口罩 防感染

保护自己 也对他人负责



- `rst = re.match()`
- `rst = re.search()`
- 找到的起始位置 `rst.start()`
- 找到的结束位置 `rst.end()`
- 找到的区间 `rst.span ()`
- 找到的组 `rst.group ()`



例子

戴口罩 防感染

保护自己 也对他人负责



```
import re
text = "JGood is a handsome boy, he is cool, clever, and so
on..."
m = re.search("\s(\w+)\s", text)
if m:
    print(m.group(1), m.start(), m.end(), m.span(), '\n')
else:
    print('not match')
m = re.match("\s(\w+)\s", text)
print(m)
➤ is 5 9 (5, 9)
➤ None
```



找到所有结果

戴口罩 防感染

保护自己 也对他人负责



➤ re.findall

- ✓ re.findall可以获取字符串中**所有匹配**的字符串，**返回一个列表**
- ✓ 如：re.findall('\w*oo\w*', text)；获取字符串中，包含'oo'的所有单词。

➤ re.finditer

- ✓ re.finditer可以获取字符串中**所有匹配**的字符串，**返回一个迭代器**
- ✓ 如：re.finditer('\w*oo\w*', text)；返回一个迭代器。



例子

戴口罩 防感染

保护自己 也对他人负责



```
import re  
s= '12 drummers drumming, 11 pipers  
   piping, 10 lords a-leaping'  
l = re.findall ('\s+\d+', s)  
print(l)
```

```
[' 11', ' 10']
```




例子

戴口罩 防感染

保护自己 也对他人负责



```
import re
s= '12 drummers drumming, 11 pipers
   piping, 10 lords a-leaping'
iter = re.finditer('\s+\d+', s)
for i in iter:
    print(i.group(), i.span())
```

11 (21, 24)

10 (39, 42)



re.M

戴口罩 防感染

保护自己 也对他人负责



```
import re
```

```
print(re.findall("\s\d+$", "I love 123\n  
Do you love 234\n"))
```

```
[' 234']
```

```
print(re.findall("\s\d+$", "I love 123\n  
Do you love 234\n", re.M))
```

```
[' 123', ' 234']
```



分组

戴口罩 防感染

保护自己 也对他人负责



➤ 匹配感兴趣的部分并将其分成几个小组

➤ `m = re.match('(a(b)c)d', 'abcd')`

`m.group(0)`

`'abcd'`

➤ `m.group(1)`

`'abc'`

➤ `m.group(2)`

`'b'`



命名分组

戴口罩 防感染

保护自己 也对他人负责



- `m = re.search(r'(?P<word>a.*?b)(ab)', 'aabab')`
- `m.group('word')`
 - `aab`
- `m.group(1)`
 - `aab`
- `m.group(2)`
 - `ab`



贪婪 vs 不贪婪

戴口罩 防感染

保护自己 也对他人负责



- `s = '<html><head><title>Title</title>'`
- `print(re.match('<.*>', s).span())`
✓ (0, 32)
- `print(re.match('<.*>', s).group())`
`<html><head><title>Title</title>`
- `print(re.match('<.*?>', s).span())`
✓ (0, 6)
- `print(re.match('<.*?>', s).group())`
`<html>`



编译为对象

戴口罩 防感染

保护自己 也对他人负责



➤ re.compile

✓ 把正则表达式编译成一个正则表达式对象

✓ compile(pattern, [flags])

□ 根据正则表达式字符串 pattern 和可选的 flags 生成正则表达式对象，其中 flags 有下面的定义：

- I 表示大小写忽略
- M 多行模式 使 \wedge \$ 匹配除了 string 开始结束外，还匹配一行的开始和结束
- S 单行模式 “.” 匹配包括 ‘\n’ 在内的任意字符，否则 . 不包括 ‘\n’

```
p = re.compile( '(blue|white|red)')
```

```
p.search("I like blue and red")
```




Compile 用法

戴口罩 防感染
保护自己 也对他人负责



- `reobj = re.compile(regex)`
- `match = reobj.search(subject)`
- `match = reobj.match(subject)`
- `result = reobj.findall(subject)`
- `iter = reobj.finditer(subject)`



例子

戴口罩 防感染

保护自己 也对他人负责



```
import re
p = re.compile('\s+\d+')
s=p.search('12 drummers drumming, 11 pipers piping,
10 lords a-leaping')
print(s.group())
```

11

```
m=p.match('12 drummers drumming, 11 pipers piping,
10 lords a-leaping')
print(m)
```

none



例子

戴口罩 防感染

保护自己 也对他人负责



```
import re
p = re.compile('\s+\d+')
print(p.findall('12 drummers drumming, 11 pipers piping, 10
lords a-leaping'))
[' 11', ' 10']
s= '12 drummers drumming, 11 pipers piping, 10 lords a-
leaping'
p = re.compile('\s+\d+')
iter = p.finditer(s)
for i in iter:
    print(i.group(), i.span())
11 (21, 24) 10 (39, 42)
```




替换 sub

戴口罩 防感染

保护自己 也对他人负责



➤ re.sub

✓ re.sub用于替换字符串中的匹配项。

✓ re.sub的函数原型为：

`re.sub(replacement, string, [count = 0])`

✓ 第1个参数是替换后的字符串；

✓ 第2个参数是替换的字符串；

✓ 第3个参数是替换个数。默认为0，表示每个匹配项都替换。



例子

戴口罩 防感染

保护自己 也对他人负责



```
p = re.compile( '(blue|white|red)')
```

```
print(p.sub( 'color', 'blue socks and red shoes'))
```

color socks and color shoes

```
print(p.sub( 'color', 'blue socks and red shoes', count=1))
```

color socks and red shoes



切分字符串split

戴口罩 防感染

保护自己 也对他人负责



➤ re.split

- ✓ 用来分割字符串
- ✓ `split(string , [maxsplit = 0])`
- ✓ `p = re.compile('\s')`
- ✓ `print(p.split('This is a test, short and sweet, of split().'))`
 - `['This', 'is', 'a', 'test,', 'short', 'and', 'sweet,', 'of', 'split().']`
- ✓ `print(p.split('This is a test, short and sweet, of split().', 3))`
 - `['This', 'is', 'a', 'test, short and sweet, of split().']`



戴口罩 防感染

保护自己 也对他人负责



谢谢！

Q&A?