可能是最好的正则表达式的教程笔记了吧...

正则表达式

1.1. 基本语法

通过一张图表来对正则表达式的基本进行一个回顾

single char	quantifiers(数量)	position(位置)
\d 匹配数字	* 0个或者更多	^一行的开头
\w 匹配word(数字、字母)	+ 1个或更多，至少1个	$一行的结尾
\W 匹配**非**word(数字、字母)	? 0个或1个,一个Optional	\b 单词"结界"(word bounds)
\s 匹配white space(包括空格、tab等)	{min,max}出现次数在一个范围内	
\S 匹配**非**white space(包括空格、tab等)	{n}匹配出现n次的	
. 匹配任何，任何的字符		

1.1.1. single char

假设你有一段字符如下：

```
1  These are some phone numbers 917-555-1234.  Also, you
2  can call me at 646.555.1234 and of course I'm always
3  reachable at (212)867-5309.
```


- `\w`

将匹配所有word，当然，() - 等字符除外

- `\w\w\w`

发现匹配的有' The se are som e pho ne number s ...' 注意正则表达式是匹配一个连续串的规则，所以可以看到三个字母的单词可以匹配到，6个单词的也可以匹配到。

- `\s\s`

匹配到一行中连续两个空格

quantifiers

假设我们有这一段话：

```
The colors of the rainbow have many colours
and the rainbow does not  have a single colour.
```

css 复制代码

我们想把所有的颜色找出来 `colors` `colours` `colour`

答案 `colou?rs?` 嗯，看起来很简单，很方便。

好了，现在想要匹配一行中的4个数字，或者一行中的5个字母等，这时候用quantifiers就非常方便了。

> 我现在想找5个字母组成的单词

- `\w{5}` 这样可以吗？嗯..不行的，看下它匹配的内容,如下: ' These are some phone numbe rs 915-555-1234...' 的确，我们模板给的很简单，它只找一行中，连续出现5个字母的序列。所以现在改进一下好了
- `\w{5}\s` 为了能找到单词，所以我希望5个字母后，跟一个空格的序列，这样应该可以了吧，看下匹配情况： ' These are some phone nu mbers 915-555-1234...' 嗯，是的，只有目前这些方法，是做不到的。 所以，我们需要第三个工具 "position"

1.1.2. position

回到刚才的问题之前，先熟悉下 `^` `$` 和 `\b`

```
This is somthing
is about
a blah
words
sequence of words
Hello and
GoodBye and
Go gogo!
```

来看下各种正则所匹配的内容

- `\w+` 这个应该毫无疑问，匹配所有的words
- `^\w+` 多了一个 `^` ，这样子，就只能匹配到每一行开头的单词了 `This is a words sequence Hello GoodBye Go`
- `\w+$` 这样就能匹配到每行的最后一个字母

回到刚才的问题

> 现在想找5个字母组成的单词

就变得很简单了，使用单词结界符 `\b`

答案就是 `\b\w{5}\b`

1.1.3. 找个电话号码吧

最后，找一个刚才出现的电话号 `123-456-1231`

用以上最基本的正则方法就是 `\d{3}-\d{3}-\d{4}` ，这样就找到了。 但是有的时候，电话号码是 `123.456.1234` 或者 `(212)867-4233` 的结构怎么办呢?

正则表达式中的 `或` 或者其他表达方式，下面一一来介绍。

1.2. 字符分类(char class)

前面记录了最基本的方法，接下来说一下分类符 `[]`

这个符号用来表示逻辑关系 或 ，比如 [abc] 表示a或者b或c. [-.] 表示符号 - 或者 . 号(**注意这里，在 [] 中的 . 号代表的就是这个符号，但是如果在其外面，表示个匹配所有。 所以如果不在 [] 之中，想要匹配'.'，就要通过转意符号 \.**)

1.2.1. 分类的简单应用

字符序列：

```vbnet
The lynk is quite a link don't you think? l nk l(nk
```

正则表达式： l[yi (]nk

结果:

```
lynk  link  l nk   l(nk
```

很容易理解的，就是表达 或 逻辑。

1.2.2. 匹配所有可能的电话号码

好了，现在回到之前遗留的问题，有以下字段，请匹配所有可能的电话号码：

```vbnet
These are some phone numbers 915-134-3122. Also,
you can call me at 643.123.1333 and of course,
I'm always reachable at (212)867-5509
```

好的，一步一步来，刚才我们使用 \d{3}-\d{3}-\d{4} 匹配了连字符的情况。现在我们可以很轻松的把 . 这种情况加进去了

第一步： \d{3}[-.]\d{3}[-.]\d{4}

第二步: 为了能够匹配括号，可以使用?来，因为这是一个option选择。所以最后就成了

```
\(?\d{3}[-.)]\d{3}[-.]\d{4}
```

这里还是要说明，在[]中，特殊字符不需要转义，可以直接使用，比如 [.()] ,但是在外面，是需要转义的 \(\. 等

1.2.3. []的特殊语法

刚才介绍了最简单和基本的功能，但是有些特殊的地方需要注意

1. -连接符是第一个字符时

比如 `[-.]` 的含义是连字符 `-` 或者点符 `.` 。 但是，如果当连字符不是第一个字符时，比如 `[a-z]` ，这就表示是从字母a到字符z。

2. []中的^

`^` 在之前介绍中，是表示一行开头，但是在 `[]` 中，有着不同的含义。 `[ab]` 表示a或者b `[^ab]` 啥都行，只要不是a或b(anythings except a and b)，相当于取反

1.2.4. []和()

除了使用 `[]` 表示或逻辑, `()` 也是可以的。用法是 `(a|b)` 表示a或者b

比如下面的例子，匹配所有email

kotlin 复制代码

```
gaoyaqi411@126.com
dyumc@google.net
sam@sjtu.edu
```

思路:

首先要想我到底相匹配什么，这里我想匹配的是

1. 任何一个以words开头的，一个或更多 `\w+`
2. 紧接着是一个 `@` 符号 `\w+@`
3. 接着有一个或者更多的words `\w+@\w+`
4. 接着一个 `.` 标点 `\w+@\w+\.`
5. 接着一个 `com` `net` 或 `edu` `\w+@\w+\.(com|net|edu)`

还是提醒注意第四步的 `\.` 转义符号

好了，这样几可以匹配以上的所有邮箱了。但是还有一个问题，因为邮箱用户名是可以有 `.` 的，比如 `vincent.ko@126.com`

其实仍然很简单，修复如下： `[\w.]+@\w+\.(com|net|edu)`

1.2.5. 总结

1. `[]` 的作用，用英文表达就是"alternation",表达一个或的逻辑；
2. `/[-.(]/` 在符号中的连字符 `-` 放在第一位表示连字符本身，如果放在中间，表示"从..到..", 比如 `[a-z]` 表示a-z
3. `[.)]` 括号中的特殊符号不需要转义，就表示其本身
4. `[^ab]` 括号中的 `^` 表示非，anythings except `a` and `b`
5. `(a|b)` 也可表示选择，但是它有更强大的功能....

所以， `()` 的强大功能是什么呢？ 分组捕获，这对序列的替换、交换是很有帮助的。 后面一节进行学习记录

1.3. 分组捕获(capturing groups)

什么是分组捕获，现在回到之前电话号码的例子

arduino 复制代码

```
212-555-1234
915-412-1333

//我想要保留区号，把后面的电话号码变为通用性的
👇👇👇👇👇👇👇👇👇👇👇👇

212-xxx-xxxx
915-xxx-xxxx
```

按照之前的做法 `\d{3}-\d{3}-\d{4}` ,这种匹配的方式，是将整个电话号码作为一个组(group)匹配起来。 我们把 `212-555-1234` 这样的叫 `Group0` 。

这个时候，如果我们加了一个括号 `\d{3}-(\d{3})-\d{4}` ，那么匹配到的 `555` 就叫 `Group1` 。 以此类推，如果有两个小括号 `\d{3}-(\d{3})-(\d{4})` 那么分组就是下面的情况：

yaml 复制代码

```
212-555-1234   Group0
555            Group1
1234           Group2
```

1.3.1. 选择分组

现在组已经分好，那么如何选择已经匹配的分组?

这里有两种方法，第一种使用 $ 符号，比如 $1 代表 555 , $2 代表 1234 ； 第二种，使用 \ ,比如 \1 代表 555 。两种的使用场景不一样，先讲 $

现在为了满足最开始的要求，我们可以这么做

```css
reg: \(?(\d{3})[-.)]\d{3}[-.]\d{4}

replace: $1-xxx-xxxx
```

ps: 这里可以直接用JS的replace函数进行操作，但是正则不是JS专属的，所以这里先介绍通用方法，之后对JS部分进行总结

1.3.2. 实景训练

1. 现在有一个名单列表，但是姓和名称是反的，我需要把他交换过来

```erlang
shiffina, Daniel
shifafl, Daniell
shquer, Danny
...
```

实现方法:

```
reg: (\w+),\s(\w+)

replace: $2 $1
```

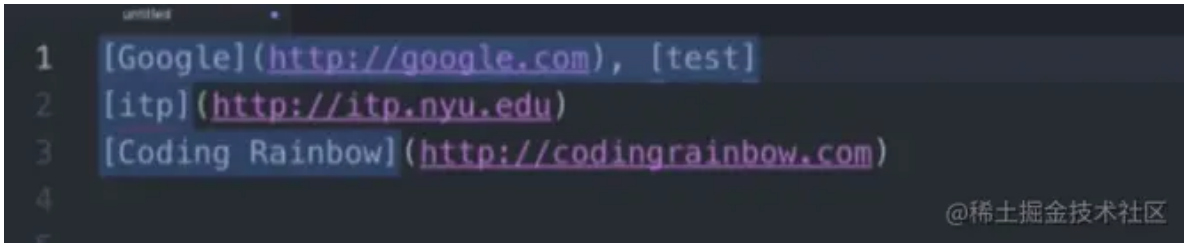
注意： $0 是所有匹配到的，所以第一个加括号的是 $1

2. 匹配markdown中的link标签，并替换为html标签

```less
[google](http://google.com)
[itp](http://itp.nyu.edu)
[Coding Rainbow](http://codingrainbow.com)
```

解析： 这道题有些坑，需要慢慢来。

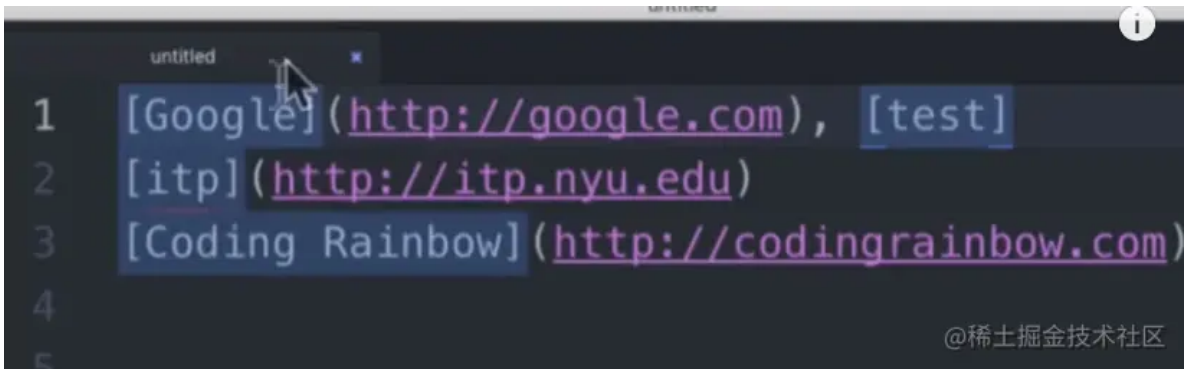
看到这个，第一个想考虑匹配 `[google]` 这个东西，立马想到正则表达式 `\[.*\]` 。 这个是巨大的坑，在当前来看，它的确能正确匹配到上面的三条。 但是如果文本是这样的：



看到了，第一行的内容会全部匹配下来，而不能区分 `[google]` 和 `[test]` 。 之所以这样，**是因为 `.` 是贪婪的，他表示所有，所有能匹配到的，所以当然也包括了 `]` ，一直到这一行的最后一个 `]` ，它才停止。**

所以为了让它能正确匹配，需要去掉这种贪婪的属性。这里用到 `?` 。 当 `?` 放在了 `quantifiers` 符号后，表示去掉贪婪属性，匹配到终止条件，即可停下。

`\[.*?\]` 这样子，就可以将 `[google]` 和 `[test]` 分开,效果如下：



接下来完成所有内容：

```ini
reg: \[(.*?)\]\((http.*?)\)

replace: <a href="$2">$1</a>
```

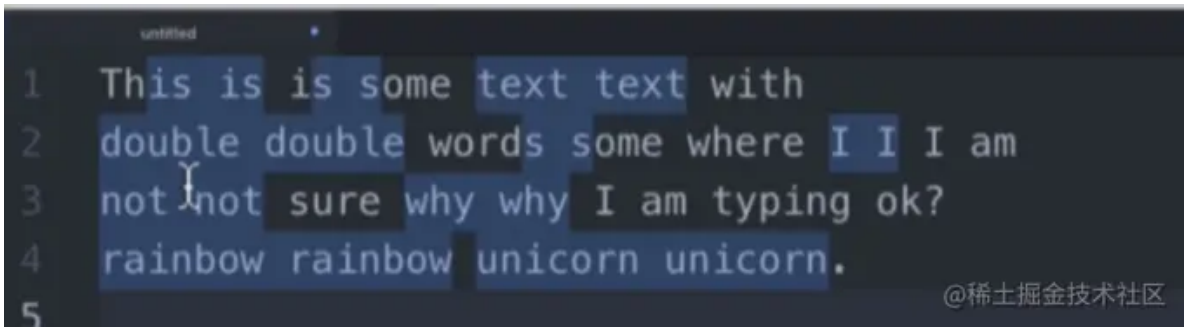
1.3.3. 使用 `\` 选择器

`$` 选择符是在替换的时候进行的标志或选择，但是如果在正则表达式本身，就要使用 `\` 选择了。比如以下的场景

```kotlin
This is is a a dog , I think think this is is really
a a good good dog. Don't you you thinks so so ?
```

我们想要匹配比如 `is is  so so` 这样连续的序列，就用到了下面的表达方式: `(\w+)\s\1`

效果:



嗯，差不多达到效果，但是有一些小的bug。比如第一句话 `This is is a` 这个就匹配不准确，会把第一个This的后面字母匹配进去。 **这就用到第一节说的字符结界 `\b` 了**，就变成了 `\b(\w+)\s\1\b`

好了，大功告成，就不贴效果图了，自行脑补就好了。

1.3.4. 总结

1. 分组捕获，使用()进行数据分组，编号0代表整个匹配项，选择的分组从1号开始
2. 选择器可以使用 `$1` 和 `\1` ，但是使用场景不同， `\` 用在正则表达式自己身上
3. `?` 符号可以禁止贪婪属性，放在 `.*` 之后，表示一次匹配遇到重点就可以停止。否则将会一直向后匹配。

1.4. 在JavaScript中的应用

在js中，主要的正则表达式都是涉及到string的应用。

```ini
var str = "hello"
var r = /w+/
```

这两个分别是string和reg的字面量创建方法。当要使用正则来进行操作的时候，使用了 `r.test()` 和 `str.match()` 以及 `str.replace` 等方法。

1.4.1. reg.test()

正则表达式本身有一个test的方法，这个方法只能测试是否包含，返回一个bool变量。

```ini
var r = /\d{3}/;
var a = '123';
var b = '123ABC';
var c = 'abc';

r.test(a)  //true
r.test(b) //true
r.test(c) //false
```

嗯，这个很简单，而且用的实际不多，下面着重讲str上的一些方法。

1.4.2. str.match()

与test()不同，不只是返回bool变量，它会返回你所匹配到的内容。

```javascript
var r = /compus/
var reg = /w+/
var s = "compus, I know something about you"
r.test(s)  //true
s.match(r)  //["compus"]
s.match(reg) //["compus"]
```

等等，好像有点问题，为什么最后一个返回的也是"compus"？这不科学。

好吧，实际上，match()返回了第一个可以匹配的序列。想要实现之前的效果，就要用到JS里关于正则的几个flag

1.4.2.1. flag

这个标志就在建立正则的时候就要有的，主要有三个

flag	含义
g	全部的，给我匹配全部的
i	忽略大小写
m	多行匹配

所以为了解决刚才的问题，只要这样子设置reg就可以了

javascript 复制代码

```
var reg = /w+/g
```

看下面一个练习

javascript 复制代码

```
var str = "Here is a Phone Number 111-2313 and 133-2311"

var r = /\d{3}[-.]\d{4}/
var rg = /\d{3}[-.]d{4}/g

console.log(str.match(r)); //["111-2313"]
console.log(str.match(rg));//["111-2313","133-2311"]
```

嗯，找电话号码，是的，很方便。但是还有一个问题，刚才说的分组，那么match会返回分组吗?

css 复制代码

```
var sr = /(\d{3})[-.]\d{4}/
var srg = /(\d{3})[-.]\d{4}/g

console.log(str.match(sr)); //["111-2313","111"]
console.log(str.match(srg)); //["111-2313","133-2311"]
```

所以结论是： 当使用了全局flag g 的时候，不会返回分组，而是全部的匹配结果；如果没有使用 g ，会将匹配到的结果和分组以数组的形式返回。

那么如何实现全局的分组?

1.4.3. reg.exec()

从字面意思来看，正则表达式的执行方法。 这个方法可以实现匹配全局，并返回分组的结果。

> reg.exec()每次调用，返回一个匹配的结果，匹配结果和分组以数组的形式返回，不断的调用即可返回下一个结果，直到返回 `null`

ini 复制代码

```ini
var str = "Here is a Phone Number 111-2313 and 133-2311" ;
var srg = /(\d{3})[-.]\d{4}/g;
var result = srg.exec(str);
while(result !== null) {
    console.log(result);
    result = srg.exec(str);
}
```

result包含的内容可能比想象中的多，它是一个数组，比如第一次执行，他的结果为：

复制代码

```
["133-2311", "133", index: 36,
input: "Here is a Phone Number 111-2313 and 133-2311" groups: undefined]
```

1.4.4. str.split

现在来到了更强的功能上，先说下split，我们知道split是将字符串按照某个字符分隔开，比如有以下一段话，需要将其分割成单词。

ini 复制代码

```ini
var s = "unicorns and rainbows And, Cupcakes"
```

分割成单词，首先想到的是空格隔开，于是可以用下面方式实现

perl 复制代码

```perl
var result = s.split(' ');
var result1 = s.split(/\s/);
//完全一样的效果
//["unicorns", "and", "rainbows", "And,", "Cupcakes"]
```

嗯，这样体现不出来正则的强大，而且最主要的是没有实现要求。因为还有一个"And,"。所以要用正则了，匹配条件是 逗号或者空格

```perl
result = s.split(/[,\s]/);

//["unicorns", "and", "rainbows", "And", "", "Cupcakes"]
```

结果仍然和需要的有出入，因为多了一个""。 我们并不是想让它分割的依据是 逗号或者空格 ,依据应该是 逗号或空格所在的连续序列 。 在原来的基础上加一个 + ， 改成 /[,\s]+/ ， 这个含义就是 一个单独的逗号，或者一个单独的空格

```perl
result = s.split(/[,\s]+/);
// ["unicorns", "and", "rainbows", "And", "Cupcakes"]
```

1.4.4.1. 单词分割

好了，拓展一下，实现一个段落的单词分割，一个正则表达式就是

```ini
result = s.split(/[,.!?\s]+/)
```

当然，有个最简单的方法，我们可以这样去做

```ini
result = s.split(/\W+/);
```

接着，如果我们想将一个段落的句子都分隔开，一个可以实现的表达式就是

```ini
result = s.split(/[.,!?]+/)
```

最后，有一个小需求，就是在分割句子的同时，还想把相应的分隔符保留下来。

```csharp
var s =
"Hello,My name is Vincent. Nice to Meet you!What's your name? Haha."
```

这是一个小小的ponit，记住**如果想要保留分隔符，只要给匹配的内容分组即可**

```css
var result = s.split(/([.,!?]+)/)
//["Hello", ",", "My name is Vincent", ".", " Nice to Meet you", "!", "What's your name", "?", " Hah
```

可以看到，这样就会把分隔符也存储起来。

1.4.5. str.replace()

replace也是字符串的方法，它的基本用法是 `str.replace(reg,replace|function)` ，第一个参数是正则表达式，代表匹配的内容，第二个参数是替换的字符串或者一个回掉函数。

注意，replace不会修改原字符串，只是返回一个修改后的字符串;除此外，正则表达式如果没有使用 `g` 标志，也和 `match` 一样，只匹配/替换第一个

1.4.5.1. 最简单的替换

> 替换一个序列中的元音字母(aeiou)，将其替换成一个double。 比如x->xx

javascript 复制代码

```
var s = "Hello,My name is Vincent."
var result = s.replace(/([aeiou])/g,"$1$1")
//"Heelloo,My naamee iis Viinceent."
```

注意，第二个参数必须是字符串; 注意不要忘记加 `g`

1.4.5.2. 牛x哄哄的function参数来了

嗯，这才是最强大的地方，第二参数传入function，先看一个最简单的示例

javascript 复制代码

```
var s = "Hello,My name is Vincent. What is your name?"
var newStr = s.replace(/\b\w{4}\b/g,replacer)
console.log(newStr)
function replacer(match) {
    console.log(match);
    return match.toUpperCase();
}
/*
name
What
your
name
Hello,My NAME is Vincent. WHAT is YOUR NAME?
*/
```

所以，函数的参数是匹配到的内容，返回的是需要替换的内容。好了，基本示例解释了基本用法，那么之前讨论的分组怎么办？如何实现分组呢？

javascript 复制代码

```
//分组

function replacer(match,group1,group2) {
    console.log(group1);
    console.log(group2);
}
```

如果正则表达式分组处理，那么在回调函数中，函数的第二个、第三参数就是group1,group2。这样子，就可以做很多神奇的事情

1.4.5.3. 综合练习题

1. 判断一个字符串中出现次数最多的字符，并统计次数

matlab 复制代码

```
var s = 'aaabbbcccaaabbbaaa';
var a = s.split('').sort().join("");  //"aaaaaaaaabbbbbbccc"
var ans = a.match/(\w)\1+/g);
ans.sort(function(a,b) {
    return a.length - b.length;
})
console.log('ans is : ' + ans[ans.length-1])
```

1.4.6. 总结

1. 在js中，正则表达式字面量 `/reg/` 和字符串字面量 `"str"` 用于创建正则和字符串。其中正则上有两个方法 `reg.test()` 和 `reg.exec()`
2. `reg.test(str)` 方法，返回布尔变量，用于指示是否有所匹配； `reg.exec(str)` 有点类似与迭代器，每次执行，返回匹配结果和分组，直到返回为 `null` 结束。
3. 字符串方法主要有 `str.match(reg)` , `str.split(reg)` 和 `str.replace(reg,str|function)` 三种方法。
4. `match` 比较特殊，如果正则包含了分组，且没有 `g` 标志，则返回匹配内容和分组; 如果没有分组，且有 `g` 标志，返回所有匹配内容
5. `split` 方法主要用于字符串分割，如果想要保存分隔符，记得将匹配内容分组(用小括号包起来)

6. `replace` 是最强大的方法，当使用回掉函数时，返回值就是替换值; 参数分别为 匹配值 `group1` `group2` ...