第九关 - 用魔法打败字符串：正则表达式

2022年12月26日 9:23

1. 项目分析

欢迎回到爬虫课堂，今天课堂主要分成两个部分。
前面，你需要跟我一起完善爬取歌词的代码。
完成这个代码后，也不能松懈。
因为你还需要学习一个非常重要的知识：正则表达式，对歌词内容进一步“提纯”。
先看看本关的项目目标吧。

1.1 项目目标

本关我们爬取的目标网站是《闪光音乐》 。

我们的目标是**根据歌手名称**，爬取《闪光音乐》网站中**与该歌手有关的所有歌曲名、歌手名及歌词信息，并存储到本地。**
比如说我们的目标歌手是周杰伦，那么我们就需要爬取网站中所有与周杰伦有关的歌曲信息。
为了搞清楚自己要做什么，我们先手动试试看该如何完成这个任务。
在这部分你可以跟着文字讲解，在网页上自行操作。这样你会有更深的感触。

1.2 网页分析

首先，进入网站瞧瞧，用谷歌 Chrome 浏览器打开《闪光音乐》网站： https://music.facode.cn/。
还是**以周杰伦为例。**假如我们要查找所有与周杰伦相关的歌曲信息，最简单的方法就是直接在首页搜索框里搜索关键字：周杰伦。

当我们在上方搜索框中搜索歌手并点击回车后，**会发现网站要求我们先登录。**

所以，我们需要先完成账号的登录才能拿到搜索结果。
那该怎么登录？点击左上角的【登录】按钮。

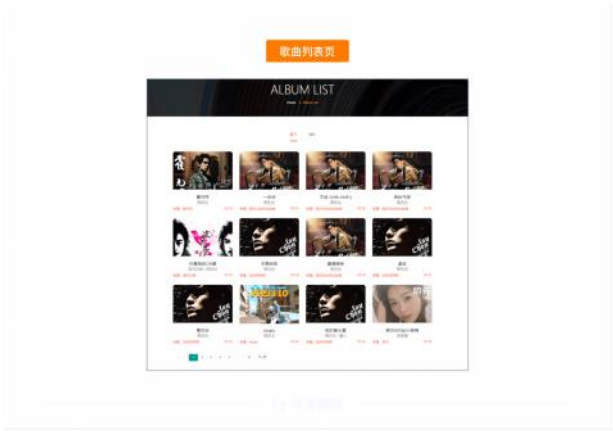
在弹出的页面中，填写账号信息后点击【登录】按钮，就能完成账号的登录。

但如果你还没有注册账号的话，那你就需要先完成注册才能登录。
点击登录页面下方的【去注册】按钮，进入注册页面。

弹出注册页面后，填写相关信息，点击【注册】按钮，就能完成账号的注册。

注册完成以后，我们再用新注册的账号登录网站即可。
好了，登录完成了，我们回到首页。
在搜索框中搜索“周杰伦”的名字。

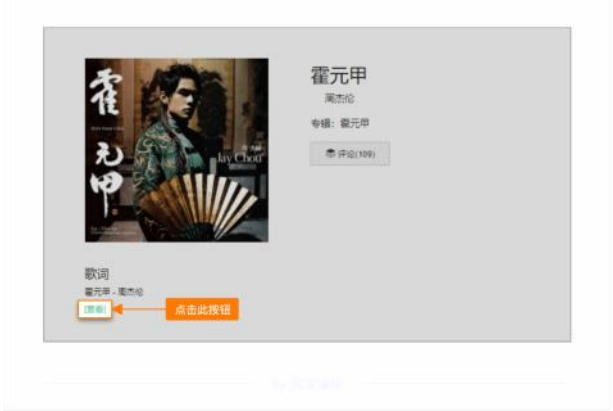
点击回车，确认搜索，就会弹出歌曲列表页：



不出我们所料，搜索得出了《闪光音乐》网站中所有周杰伦相关的歌曲。
可以看到，每一页会排布12首歌曲的搜索结果。而且，歌曲列表页中会显示每首歌的：**歌手名、歌曲名信息。**
但是关键的**歌词信息**还不在这里。可能需要我们点击具体歌曲详情页才能找到歌词。
我们试着点开页面的第一首歌《霍元甲》看看。

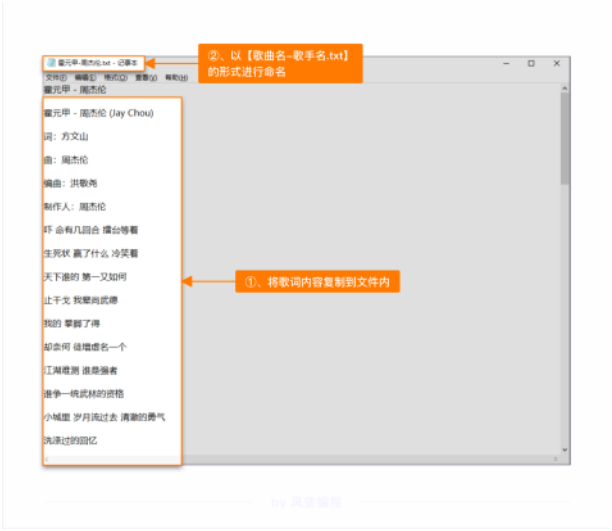
温馨提示，在实际操作网页时，需要点击歌曲名才能进入歌曲详情页。

在歌曲详情页中，我们可以在很显眼的位置看到歌词一栏。点击【查看】按钮，看看能不能找到具体的歌词。



没问题，具体的歌词信息就在里面。

如果是手动操作的话，接下来我们就需要将歌词信息复制粘贴到 `txt` 文档中，并以【歌曲名-歌手名.txt】的格式命名。就像下面这样：



这样复制一首歌的信息，还是很快的。但是数量一多就很麻烦了。
比如我们要目标歌手的所有歌曲信息，而不是一首、两首歌的歌曲信息。所以，还是用爬虫来完成这个项目会更高效。
那么，咱们该怎么用爬虫来完成它呢？
老规矩，我们先来体验一下我写好的项目代码吧。

1.2 体验代码

整个爬取网站是一个动态网站，需要先登录，登录后，搜索歌手名字，查询歌手歌曲信息，歌词需要点击进歌曲详情页，再点查看 获得。

```
import requests
import  re

# 输入登入账号
phone = input(">输入登入手机号：")
password = input(">输入登入密码：")
print(">登入中，请稍候...")
name = input('请输入歌手名称：')

# 登录网站链接
login_url = 'https://music.facode.cn/index.php/Home/Index/login.html'
login_data_from = {'phone': phone,
                   'pass': password,
                   'verify': 214839}

# 登录请求
login_res = requests.post(login_url, data=login_data_from)
if login_res.status_code == 200:
    print('登录成功')
else:
    print('登录失败')


# 设置查询数据的链接
search_url = 'https://music.facode.cn/index.php/Home/Index/search_list.html'

# 设置歌词数据链接
lyrics_url = 'https://music.facode.cn/index.php/Home/Index/lyrics.html'

for page in range(1,11):
    search_data_from = {'value': name,
                        'info': 1,
                        'page': page}

    # 查询请求
    search_res = requests.post(search_url, data=search_data_from,cookies=login_res.cookies)

    # 获取json 数据 并转换成字典格式
    search_res_json = search_res.json()

    for song in search_res_json['voice']:
        # 歌词文件名（音乐名加作者）
        file_name = song['name'] + '-' + song['author'].replace('/', '')

        # 歌词数据页请求
        # 歌词数据页表单
        lyrics_data = {'id': song['id']}


        lyrics_res = requests.post(lyrics_url, cookies=login_res.cookies,data=lyrics_data)
```

```
# 获取歌词页json数据 并 转换成字典格式
lyrics_json = lyrics_res.json()

#如果不存在歌词，则跳过 歌词就是lyrics_json['data']
if lyrics_json['data'] == '暂无歌词':
    print(f'{file_name}--暂无歌词，继续查找-->')
    continue

else:
    # 用正则表达式 删除掉不需要的字符
    lyrics = re.sub('\[.*?]','',lyrics_json['data'])

with open(f'D:/PythonTest/风变python学习资料/Python爬虫/歌词体验/{file_name}.txt', 'w', encoding='utf-8-sig',newline='') as f:
    f.write(lyrics)

    print(f'{file_name}-->歌词文件保存成功-->')
```

4. 正则表达式

4.1 正则表达式是什么

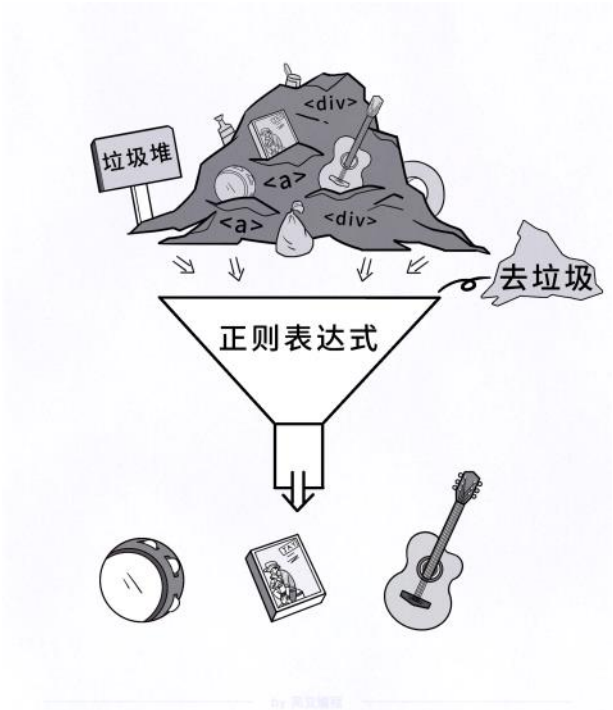
首先，第一个问题，正则表达式到底是什么呢?

正则表达式是一种用事先定义好的字符组合，来检索字符串文本内容的方法。

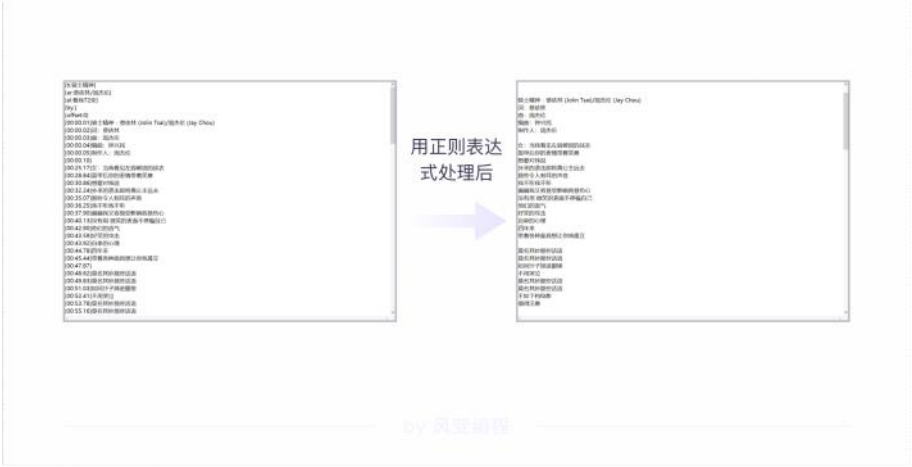
简单点说，就是符合某种规律的“内容公式”。

如果把杂乱的字符信息比做一个杂乱的物堆。

当我们确定了要过滤什么内容后，可以通过正则表达式总结出规律。这个正则表达式就会像一个无敌漏斗，来帮助我们从中筛选找到想要的字符串。



回到歌词文本，我们也可以通过正则表达式找出这一系列不必要的信息，然后删除。



咱们直接来体验一段代码。它会从一串杂乱的语句里**找出所有收入信息。**

具体代码内容不必细看，大体看一遍后直接运行即可。

```python
# 体验正则表达式
import re

# 待处理的字符串
content = '''
说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
而领导王小强的电话号码保密，收入是最好的，15300元/月。
'''

# 正则表达式
pattern = '[0-9]*元/月'

# 用正则表达式筛取其中的月收入数值，返回一个列表
income_list = re.findall(pattern, content)

# 打印列表，检验结果
print(income_list)
```

```
['10110元/月', '8100元/月', '15300元/月']
```

```python
#体验正则表达式
importre

#待处理的字符串
content='''
说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
而领导王小强的电话号码保密，收入是最好的，15300元/月。
'''

#正则表达式
pattern='[0-9]*元/月'

#用正则表达式筛取其中的月收入数值，返回一个列表
income_list=re.findall(pattern,content)

#打印列表，检验结果
print(income_list)
```

可以看到，所有符合条件的字符串都被筛选出来，并返回到一个列表中。

那在 Python 中，我们是怎么应用正则表达式的呢？

4.2 re 模块与 re.findall() 函数

在Python里面用正则表达式，我们需要用到它内置的re 模块。

re 模块的导入方法为：

```
1  import re
```

在刚刚那段代码中，我们就用到了 re 模块中的 **findall()** 函数。

它可以在字符串中找到正则表达式所匹配的所有内容，并返回一个列表。

如果没有找到匹配的，则返回空列表。 它的语法格式是：👇

```
1  re.findall(pattern, string)
```

findall() 函数内的：

1） **pattern** 表示用于匹配的正则表达式；

2） **string** 表示需要匹配的字符串。

在刚刚那串代码中，正则表达式就是 **[0-9]*元/月** ，需要匹配的字符串就是变量 content 中的内容。

至于[0-9]*是什么意思，为什么这么写？我们接下来就会学到。

其实，正则表达式说复杂也不复杂，它主要由 **普通字符** 以及 **元字符** 组成。咱们先来看普通字符。

4.3 普通字符

在正则表达式中，普通字符包括**所有大写和小写字母，所有数字和其他任何没有被指定为元字符的符号（比如中文）**。

写在正则表达式里面的普通字符都能直接匹配它们。也就是说，**普通字符会匹配字符串中和它相同的内容。**

比如说，我们的正则表达式就写一个普通字符'月'。

阅读完下面的代码后直接运行即可，重点看第11行代码。

```
1  import re
2  # 待处理的字符串
3  content = '''
4  说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
5  但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
6  而领导王小强的电话号码保密，收入是最好的，15300元/月。
7  '''
8
9  # 正则表达式
10 pattern = '月'
11
12 # 用正则表达式筛取其中的月收入数值，返回一个列表
13 income_list = re.findall(pattern, content)
14
15 # 打印列表，检验结果
16 print(income_list)
```

```
['月', '月', '月', '月']
```

```
importre
#待处理的字符串
content='''
说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
而领导王小强的电话号码保密，收入是最好的，15300元/月。
'''

#正则表达式
pattern='月'
```

```
#用正则表达式筛取其中的月收入数值，返回一个列表
income_list=re.findall(pattern,content)

#打印列表，检验结果
print(income_list)
```

可以看到，字符串中所有的'月'都被匹配出来了。

换到其他普通字符也都是一样的。

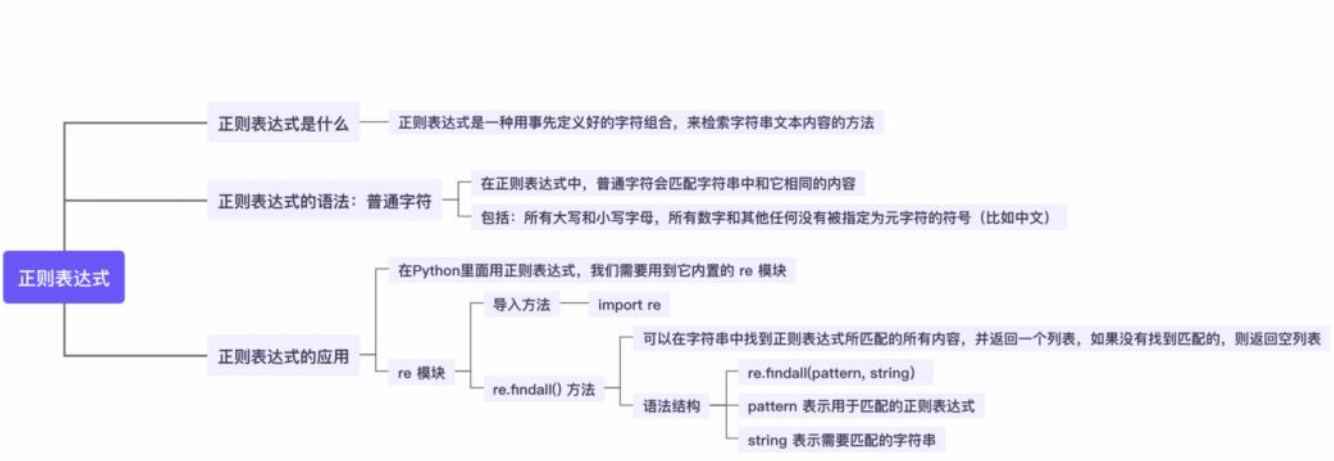
来，实操一下。将代码第11行正则表达式内容修改为'电话'，看是否能匹配出字符串所有的'电话'。

```
import re
# 待处理的字符串
content = '''
说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
而领导王小强的电话号码保密，收入是最好的，15300元/月。
'''

# 正则表达式
pattern = '电话号码'

# 用正则表达式筛取其中的月收入数值，返回一个列表
income_list = re.findall(pattern, content)

# 打印列表，检验结果
print(income_list)
```

```
['电话号码', '电话号码', '电话号码']
```

```
importre
#待处理的字符串
content='''
说来，组长许大力的电话号码是13426755462，目前收入情况是10110元/月。
但员工刘聪明差点，他电话号码为18926234431，月收入8100元/月。
而领导王小强的电话号码保密，收入是最好的，15300元/月。
'''

#正则表达式
pattern='电话号码'

#用正则表达式筛取其中的月收入数值，返回一个列表
income_list=re.findall(pattern,content)

#打印列表，检验结果
print(income_list)
```

到这里，我们先简单总结一下现在所学的知识：



但是，你可能会有个疑问，这个普通字符有啥用呢？就像 str.replace() 一样，我们还是无法匹配大多数非固定的字符内容。

稍安勿躁，这是因为我们还需要搭配元字符才能发挥正则表达式的功效。

除了普通字符，正则表达式中还有很多特殊的字符，我们叫它元字符。

当它们出现在正则表达式字符串中时，不是表示直接匹配这些字符，而是表达一些特别的含义。

这里最常见的有：**字符类元字符**。

4.4 元字符 - 字符类

字符类元字符代表它**能与一组或一类字符匹配。**

字符类元字符中比较常见的就是 . 。

. 能匹配除**换行符（如 \n）之外的任何单个字符。**

不要小看它，用它搭配普通字符，我们就可以更多元地匹配字符了。

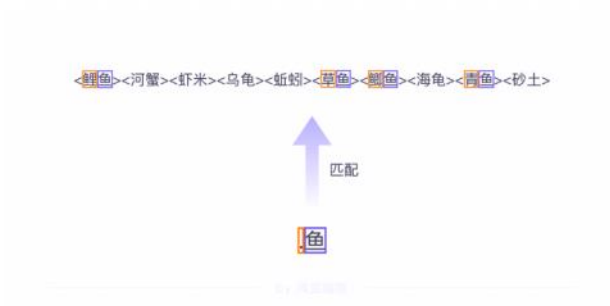
比如这有一个字符串文本组成的变量 river。👇

```
river = '''
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><乌龟><蚯蚓><草鱼><鲫鱼><海龟><青鱼><砂土>
'''
```

如果我们要从这里“捞”出所有的“鱼”。也就是找到以“鱼”字结尾，且前面只有一个字符的词。

我们的正则表达式就可以写为 .鱼 。

这里的 . 是元字符， 鱼 是普通字符，搭配在一起的意思是只要'.鱼'格式的两个字符文本就匹配。



比如说鲤鱼、草鱼、鲫鱼、青鱼…之类的都会被成功匹配。

来，实际运行一下这段代码看看，重点看第10行。👇

```
# 找出字符串中所有的鱼
import  re

river = '''
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><乌龟><蚯蚓><草鱼><鲫鱼><海龟><青鱼><砂土>
'''
fish_list = re.findall('.鱼',river)  # .能匹配除换行符（如 \n）之外的任何单个字符。
print(fish_list)
```

```
['鲤鱼', '草鱼', '鲫鱼', '青鱼']
```

```
#找出字符串中所有的鱼
importre
```

```
river='''
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><乌龟><蚯蚓><草鱼><鲫鱼><海龟><青鱼><砂土>
'''
fish_list=re.findall('.鱼',river)#.能匹配除换行符（如\n）之外的任何单个字符。
print(fish_list)
```

果然如前面图片显示的匹配结果一样，匹配出了'鲤鱼'、'草鱼'、'鲫鱼'、'青鱼'。

你试着练习看看。

这一次，需要你用正则表达式捞出所有的“龟”，即匹配出所有以“龟”字结尾的词。

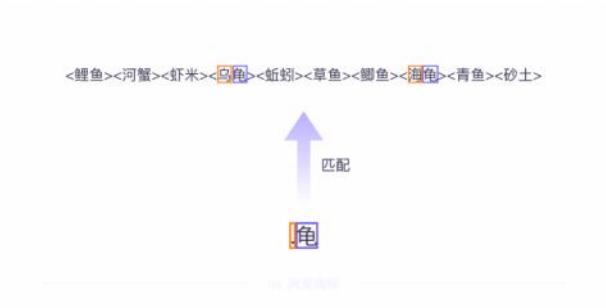
按提示要求，补充下面代码第10行，打印出对应结果。

```
1  # 找出字符串中所有的龟
2  import  re
3
4  river = '''
5  在这条长河里面，游满了：
6  <鲤鱼><河蟹><虾米><乌龟><蚯蚓><草鱼><鲫鱼><海龟><青鱼><砂土>
7  '''
8
9  turtle_list = re.findall('.龟',river)
10 turtle_list
```

```
['乌龟', '海龟']
```

```
#找出字符串中所有的龟
importre

river='''
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><乌龟><蚯蚓><草鱼><鲫鱼><海龟><青鱼><砂土>
'''

turtle_list=re.findall('.龟',river)
turtle_list
```

可以看到，所有以“龟”字结尾的词都被匹配出来了。



你可能会想，.指代的范围也太大了吧。有没有什么小一点的，可以给我们自由控制取值范围的方法呢？

我们可以用中括号[]。

将同一个位置可能出现的各种普通字符用 [] 框起来，他们就可以组成一个**字符组**。

这个字符组表示**匹配中括号内指定的几个字符之一**。

比如说：

1) [abcd]可以匹配a，b，c，d里面的任意一个字符，可以被简单表示为[a-d]，代表a到d;

2) [12345]可以匹配1，2，3，4，5里的任意一个字符，可以被简单表示为[1-5]，代表1到5;

3) [龟鱼鸡鸭鹅]可以匹配其内部的特定字符：'龟'、'鱼'、'鸡'、'鸭'、'鹅'。

回到最开始的例子，假设我们想用正则表达式把字符串中的数字都提取出来，就可以用 **[0-9]** 来匹配。

阅读下面的代码，然后直接运行。

```
#提取字符串中所有的数字
import re

# 待处理的字符串
content = '''
A组长许大力电话号码为13426755462，月收入1.1万/月。
'''
number = re.findall('[0-9]',content ) # []表示一个范围，这里表示0到9任意数字
number

```

```
['1', '3', '4', '2', '6', '7', '5', '5', '4', '6', '2', '1', '1']
```

```
#提取字符串中所有的数字
importre

#待处理的字符串
content='''
A组长许大力电话号码为13426755462，月收入1.1万/月。
'''
number=re.findall('[0-9]',content)#[]表示一个范围，这里表示0到9任意数字
number
```

如此一来，我们就可以匹配出字符串中所有的数字了。

需要注意的是，一个字符组和一个 . 、一个普通字符一样，都只能匹配一个字符。

接下来，咱们来道练习吧。

还是这条字符串文本组成的变量 river。这次要你捞出里面所有的"鱼"和"龟"，即匹配出所有以"鱼"或"龟"字结尾的词。

按题设要求，补充下面第10行代码，完成练习。

温馨提示，**一个[龟鱼]可以匹配一个'龟'或一个'鱼'。** 👇

```
# 提取字符串所有的龟和鱼
import re

# 待处理的字符串
river = '''
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><水龟><蚯蚓><草鱼><鲫鱼><饼龟><青鱼><砂土>
'''
river_data = re.findall('.[鱼龟]',river)
river_data

```

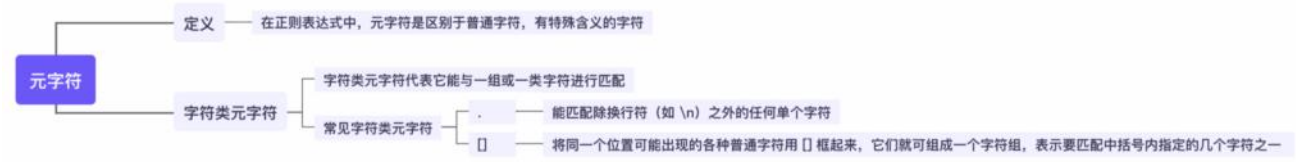
```
['鲤鱼', '水龟', '草鱼', '鲫鱼', '饼龟', '青鱼']
```

```
#提取字符串所有的龟和鱼
importre

#待处理的字符串
river='''
```

```
在这条长河里面，游满了：
<鲤鱼><河蟹><虾米><水龟><蚯蚓><草鱼><鲫鱼><饼龟><青鱼><砂土>
'''
river_data=re.findall('[鱼龟]',river)
river_data
```

我们需要将“鱼”、“龟”用中括号 [] 框起来形成一个字符组。其他不变，我们就能匹配到所有以“鱼”或“龟”字结尾的词了。

好了，字符类元字符我们就先学到这里，简单总结一下：👇



关于字符类元字符有太多种类了，我们不可能在一节课中全部学完。
但是别担心，我这里给你整合了常见的字符类元字符。

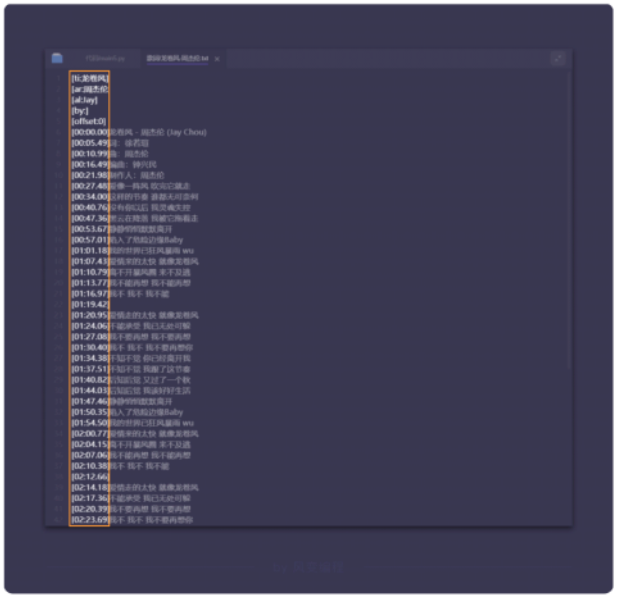
常见字符类元字符

字符	说明
.	匹配除换行符（\n、\r）之外的任何单个字符。
\d	匹配一个数字字符
\D	匹配一个非数字字符
\n	匹配一个换行符
\s	匹配任何空白字符，包括空格、制表符、换页符等等
\S	匹配任何非空白字符
\w	匹配字母、数字、下划线字符
\W	匹配非字母、数字、下划线字符

先把这张图保存下来。在课后，你还可以多尝试感受一下不同字符类元字符的玩法。

虽然我们现在可以匹配出不同类别的字符串了，但这样还不够。

因为，歌词文本中需要被剔除的内容长度也是不一样的。



所以，除了要控制匹配的字符类型外，我们还需要控制匹配的字符数量。

这里就涉及了一种新的元字符：**数量词**。

4.5 元字符 - 数量词

数量词**能指定匹配一定数量的字符**，它也被称为限定符。

在正则表达式中，最常用的数量词有`*`。它表示匹配**前面的子表达式任意次，包括0次。**

这里的子表达式常常指的是正则表达式中的一个字符或字符组。

咱们具体从一个例子出发来看。

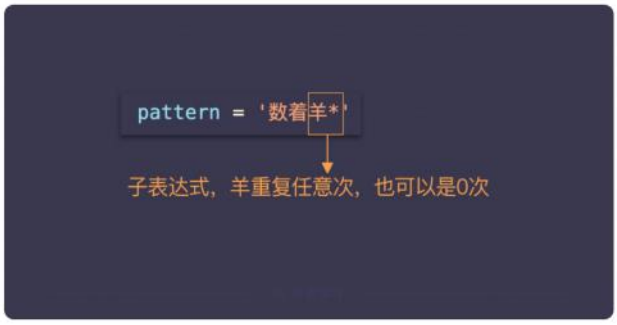
阅读完下面的代码后，重点看第 10 行代码，看完后直接运行即可。👇

```
1  import re
2
3  content = '''
4  我数着羊羊羊羊羊羊羊，呼噜噜
5  我数着羊羊羊，呼噜噜
6  我数着，呼噜噜
7  '''
8
9  # 正则表达式
10 pattern = '数着羊*'    #    *表示匹配前面的子表达式一次或多次，包括0次 ，*放在羊后面，代表这里的羊可以是任意个，也可以是0个
11
12 # 用正则表达式提筛取其中的信息，返回一个列表
13 sheep = re.findall(pattern, content)
14
15 # 打印列表，检验结果
16 print(sheep)
```

```
['数着羊羊羊羊羊羊羊', '数着羊羊羊', '数着']
```

```
importre

content='''
我数着羊羊羊羊羊羊羊，呼噜噜
我数着羊羊羊，呼噜噜
我数着，呼噜噜
'''

#正则表达式
pattern='数着羊*'#*表示匹配前面的子表达式一次或多次，包括0次,*放在羊后面，代表这里的羊可以是任意个，也可以是0个

#用正则表达式提筛取其中的信息，返回一个列表
sheep=re.findall(pattern,content)

#打印列表，检验结果
print(sheep)
```

`*`放在羊后面，代表这里的羊可以是任意个，也可以是0个。



所以，'数着羊羊羊羊羊羊羊'、'数着羊羊羊'，这种包含多个“羊”的情况可以匹配成功。'数着'这种0个“羊”的情况也可以匹配成功。

还有一种数量词与 * 类似，那就是`+`。

`+`表示**匹配前面的子表达式一次或多次，不包括0次。**

还是一样，咱们简单体验一段代码，阅读完直接运行即可。👇

```
import re

content = '''
我数着羊羊羊羊羊羊羊，呼噜噜
我数着羊羊羊，呼噜噜
我数着，呼噜噜
'''

# 正则表达式
pattern = '数着羊+'    #  +表示匹配前面的子表达式一次或多次，不包括0次

# 用正则表达式提筛取其中的信息，返回一个列表
sheep = re.findall(pattern, content)

# 打印列表，检验结果
print(sheep)
```

```
['数着羊羊羊羊羊羊羊', '数着羊羊羊']
```

```
importre

content='''
我数着羊羊羊羊羊羊羊，呼噜噜
我数着羊羊羊，呼噜噜
我数着，呼噜噜
'''

#正则表达式
```

```
pattern='数着羊+'#+表示匹配前面的子表达式一次或多次，不包括0次

#用正则表达式提筛取其中的信息，返回一个列表
sheep=re.findall(pattern,content)

#打印列表，检验结果
print(sheep)
```

因为少了0次的情况，所以这次匹配的结果只有**'数着羊羊羊羊羊羊羊'**、**'数着羊羊羊'**。

最后还有一种比较常见的数量词是**？**，它表示**匹配前面的子表达式0次或一次。**

还是一样，咱们简单体验一段代码，阅读完直接运行即可。👇

```python
import re

content = '''
我数着羊羊羊羊羊羊羊羊，呼噜噜
我数着羊羊羊，呼噜噜
我数着，呼噜噜
'''

# 正则表达式
pattern = '数着羊?'  # ?它表示匹配前面的子表达式0次或一次

# 用正则表达式提筛取其中的信息，返回一个列表
sheep = re.findall(pattern, content)

# 打印列表，检验结果
print(sheep)
```

```
['数着羊', '数着羊', '数着']
```

```
importre

content='''
我数着羊羊羊羊羊羊羊，呼噜噜
我数着羊羊羊，呼噜噜
我数着，呼噜噜
'''

#正则表达式
pattern='数着羊?'#?它表示匹配前面的子表达式0次或一次

#用正则表达式提筛取其中的信息，返回一个列表
sheep=re.findall(pattern,content)

#打印列表，检验结果
print(sheep)
```

因为只匹配0次和1次，所以这次匹配的结果只有'数着羊'、'数着羊'、'数着'。

但除了以上常规用法外，？还会根据你的使用方法不同，而被识别成不同的作用。这一点我等下就会讲到。

好了，再回到咱们那条“河”，试着捞下“鱼”。

但这次，河里汇入了其他不同大小品种的“鱼”。

```
river = '''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
```

因为 <> 是普通字符，那我们用正则表达式 <.鱼> 来试“捞”一下，看看能不能“捞”出每一条鱼。

阅读完下面的代码后重点看第10行，然后直接运行即可。👇

```
# 捞出河里的每条鱼
import  re
river = '''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish = re.findall('<.鱼>',river)

fish
```

```
['<鲤鱼>', '<草鱼>', '<青鱼>']
```

```
#捞出河里的每条鱼
importre
river='''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish=re.findall('<.鱼>',river)

fish
```

可以看到老办法已经不奏效了，只能“捞”出中等大小的鱼：'<鲤鱼>'、'<草鱼>'、'<青鱼>'。

像<大鲤鱼>、<小青鱼>、<鱼>这种我们就“捞”不出来了。

怎么办呢？我们需要加入数量词，匹配鱼前面的多个字符字符。

那如果我们的正则表达式改成 <.*鱼>，是不是就可以捞出超过两个字的鱼了呢？闲话少说，直接试试看。👇

```python
# 捞出河里的每条鱼
import  re
river = '''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish = re.findall('<.*鱼>',river)  # 这里*号匹配鱼前面任意字符，包括0个字符

fish
```

```
['<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>']
```

```
#捞出河里的每条鱼
importre
river='''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish=re.findall('<.*鱼>',river)#这里*号匹配鱼前面任意字符，包括0个字符

fish
```

嗯？为什么匹配了这么长一段？

因为在正则表达式中，`.*`是贪婪的。这种组合符号我们叫它**贪婪限定符**。

贪婪限定符，就是**在表达式匹配成功的前提下，它会尽可能多地匹配内容。**

在前面代码中，虽然有许多符合条件的选项，但是它会尽可能多地匹配字符的结果。

如果我们不想要匹配那么多内容该怎么办呢？

我们可以使用**惰性限定符**，也叫做**非贪婪限定符**。

我们前面讲过 ? 会根据你的使用方法不同，而被识别成不同的作用。

当？位于在贪婪限定符的后面时，它的作用就是将其变为惰性限定符。

既然.*是贪婪的，那么加上一个？就能变成惰性限定符.*?。

与贪婪限定符相反，所谓惰性，就是**在表达式匹配成功的前提下，尽可能少的匹配。**

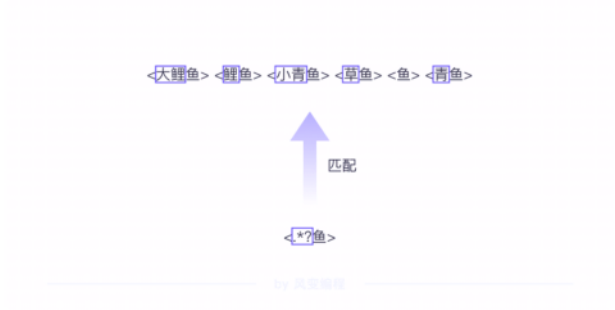
还是原先的代码，我将正则表达式改成了惰性限定符的方式，来看看匹配结果是怎样的。

```
# 捞出河里的每条鱼
import  re
river = '''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish = re.findall('<.*?鱼>',river)

fish
```

```
['<大鲤鱼>', '<鲤鱼>', '<小青鱼>', '<草鱼>', '<鱼>', '<青鱼>']
```

```
#捞出河里的每条鱼
importre
river='''
在这条长河里面，游满了：
<大鲤鱼><鲤鱼><小青鱼><草鱼><鱼><青鱼>
'''
fish=re.findall('<.*?鱼>',river)

fish
```

可以看到，这次我们成功把每一个“鱼”都捞起来了。惰性限定符对字符串的匹配情况如下：👇



它会尽可能少地匹配。

来，你也练习一下吧。先看一个字符串：

```
source = '<div class="word"><a>闪光音乐-最闪的音乐网站</a></div>'
```

假设我们需要把以上字符串中的所有 html 标签都提取出来，正则表达式该怎么写呢？

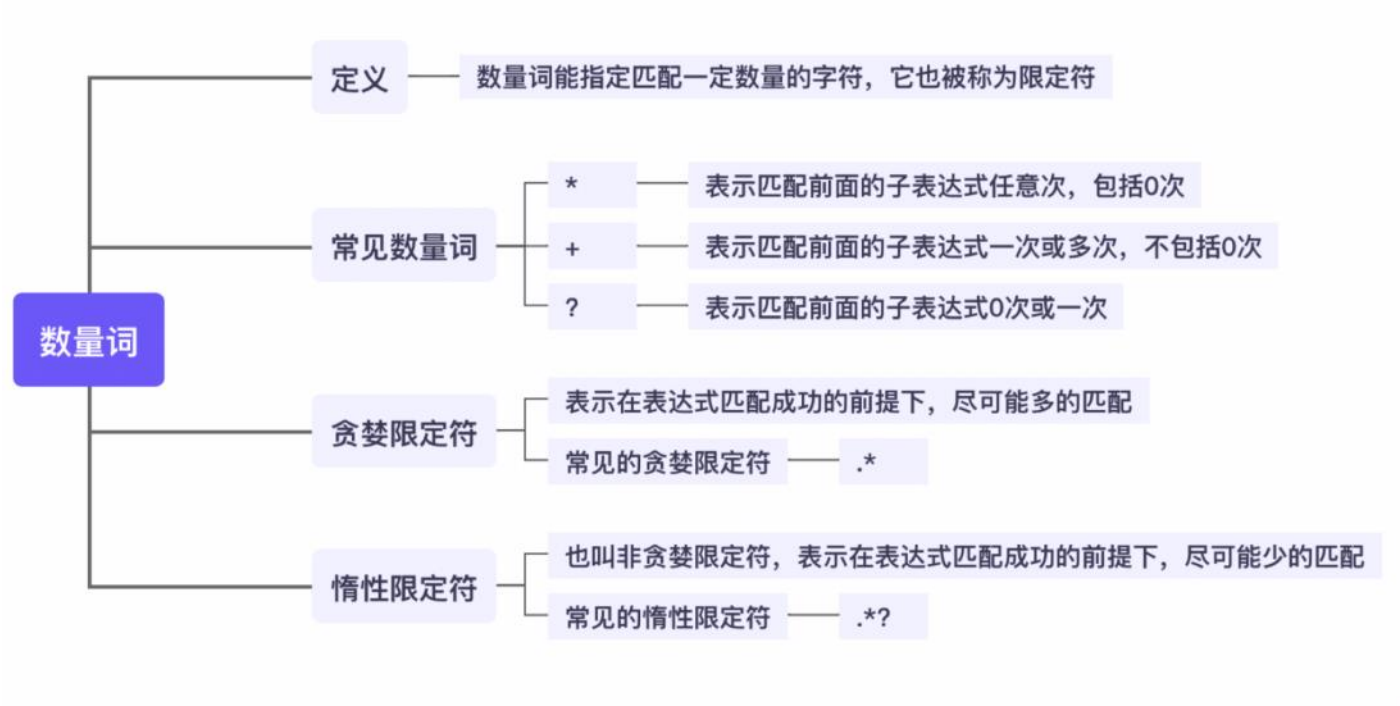
```
import re

# 创建变量 source
source = '<div class="word"><a>闪光音乐-最闪的音乐网站</a></div>'

# 正则表达式
pattern = '<.*?>'    # 提取全部html标签，用惰性限定符 .*? 搭配普通字符 <> 就能取出。

# 用正则表达式提筛取其中的信息，返回一个列表
html_tag = re.findall(pattern, source)

# 打印列表，检验结果
print(html_tag)
```

```
['<div class="word">', '<a>', '</a>', '</div>']
```

```
importre

#创建变量source
source='<divclass="word"><a>闪光音乐-最闪的音乐网站</a></div>'

#正则表达式
pattern='<.*?>'#提取全部html标签，用惰性限定符.*?搭配普通字符<>就能取出。

#用正则表达式提筛取其中的信息，返回一个列表
html_tag=re.findall(pattern,source)

#打印列表，检验结果
print(html_tag)
```

和前面“捞鱼”是一样的，我们用惰性限定符 .*? 搭配普通字符 <>，就能取出目标字符。

好了，数量词我们就先学到这里，简单总结一下：👇



回到项目中，我们要筛选的内容也就是中括号：[xxx] 这种格式内的内容。

想必和捞鱼的格式一样，写成 **[.*?]** 应该就可以了。

来，我们直接用它来试试看筛选歌词文本。

5. 项目代码优化

咱们先简单做个尝试，看看是否这类内容是否可以被［.*?］筛选出来。

```
import re

# 模拟一段待处理的歌词字符串
lyrics = '''
[00:23.14]我可能是一段歌词1
[00:24.44]我可能是一段歌词2
'''

# 用正则表达式筛取其中的信息，返回一个列表
words = re.findall('[.*?]', lyrics)

# 打印列表，检验结果
print(words)
```

```
['.', '.']
```

```
importre

#模拟一段待处理的歌词字符串
lyrics='''
[00:23.14]我可能是一段歌词1
[00:24.44]我可能是一段歌词2
'''

#用正则表达式筛取其中的信息，返回一个列表
words=re.findall('[.*?]',lyrics)

#打印列表，检验结果
print(words)
```

从终端结果来看，似乎有点尴尬，就匹配出了俩点。

别慌，还记得我们前面学过 [] 是特殊字符吗？因为中括号会将其里面的内容组成一个字符组，而非仅仅表达字面的中括号意思。

所以一个正则表达式 `[.*?]` 只会在 `.`、`*`、`?` 三个符号中匹配其中一个。

我们才无法匹配出自己想要的内容。

那这该咋办？

我们需要**对元字符转义**，这样它就不会被当作特殊字符了。

5.1 元字符的转义

具体的代码，我们用反斜杠 `\` 来实现操作。

用法很简单，在需要转义的字符前面加个反斜杠就行。

阅读完下面的代码后，重点看第十行，直接运行即可。👇

```
import re

# 模拟一段待处理的歌词字符串
lyrics = '''
[00:23.14]我可能是一段歌词1
[00:24.44]我可能是一段歌词2
'''

# 用正则表达式筛取其中的信息，返回一个列表
words = re.findall('\[.*?]', lyrics)   #  \ 反斜杠转义

# 打印列表，检验结果
print(words)
```

```
['[00:23.14]', '[00:24.44]']
```

```
importre

#模拟一段待处理的歌词字符串
lyrics='''
[00:23.14]我可能是一段歌词1
[00:24.44]我可能是一段歌词2
'''

#用正则表达式筛取其中的信息，返回一个列表
words=re.findall('\[.*?]',lyrics)#\反斜杠转义

#打印列表，检验结果
print(words)
```

完美！ `'[00:23.14]'`、 `'[00:24.44]'` 都被精准筛取出来了。

你可能会有疑问，为什么给前面的中括号反斜杠就行了，不需要给后面的反斜杠呢？

这是因为一对中括号才组成"字符组"意义。我们将把前面的中括号转义为普通字符了以后，后面的中括号没有成对的符号，自然也就变成普通字符了。

当然除了 [] 外、我们需要元字符表示其字面意思的时候都可以用反斜杠转义它。

像是前面教的 `.`、`*` 这些都是可以的。感兴趣的同学可以在课后多多尝试噢。

现在我们已经能用正则表达式匹配出我们要剔除的目标了。接下来，我们只需要掌握删除匹配内容的方法就好了。

这里我们可以用 re 模块中的一个函数 `re.sub()`。

5.2 re.sub() 函数

re.sub() 函数能用于**替换字符串中的匹配项，并返回被替换后的字符串。**

我们可以用 re.sub() 函数将目标字符串替换为空字符串，来实现删除目标字符的效果。

在用法上，re.sub() 函数的语法结构是这样的：👇

```
re.sub(pattern, repl, string)
```

在这里涉及的参数有：

1） **pattern** 表示用于匹配的正则表达式；

2） **repl** 表示要替换的字符串；

3） **string** 表示需要匹配的字符串。

简单来说，参数 string 中能与正则表达式 pattern 匹配到的内容，会被替换为 repl。

具体怎么操作？我们来看段代码，关键看第七行。

```
import re

# 模拟一段待处理的歌词字符串
lyrics = '歌手：周杰伦\n[00:00.00]不能说的秘密\n[00:03.00]词:方文山、曲:周杰伦\n'

# 用正则表达式替换其中的信息
result = re.sub('\[.*?]', '', lyrics)

# 打印并检验结果
print(result)
```

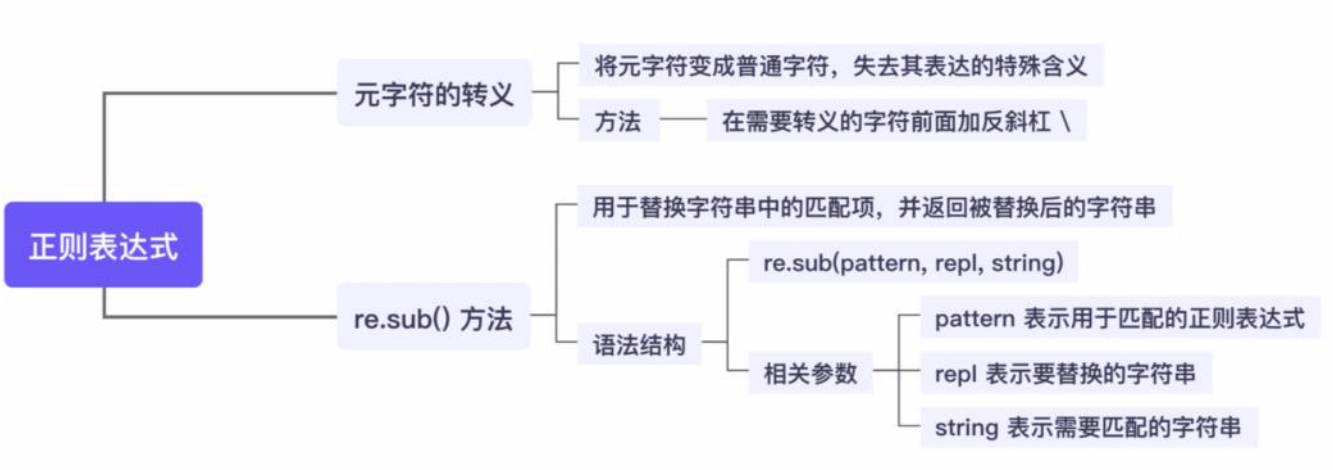
```
歌手：周杰伦
不能说的秘密
词:方文山、曲:周杰伦
```


```
importre

#模拟一段待处理的歌词字符串
lyrics='歌手：周杰伦\n[00:00.00]不能说的秘密\n[00:03.00]词:方文山、曲:周杰伦\n'

#用正则表达式替换其中的信息
result=re.sub('\[.*?]','',lyrics)

#打印并检验结果
print(result)
```


从终端结果来看，与正则相匹配的字符串，会替换为无内容的空字符串''。

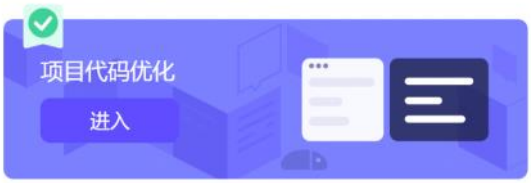
如此一来，我们就可以去掉不想要的 [xxx] 格式的内容了。

最后简单总结一下以上的知识点：👇



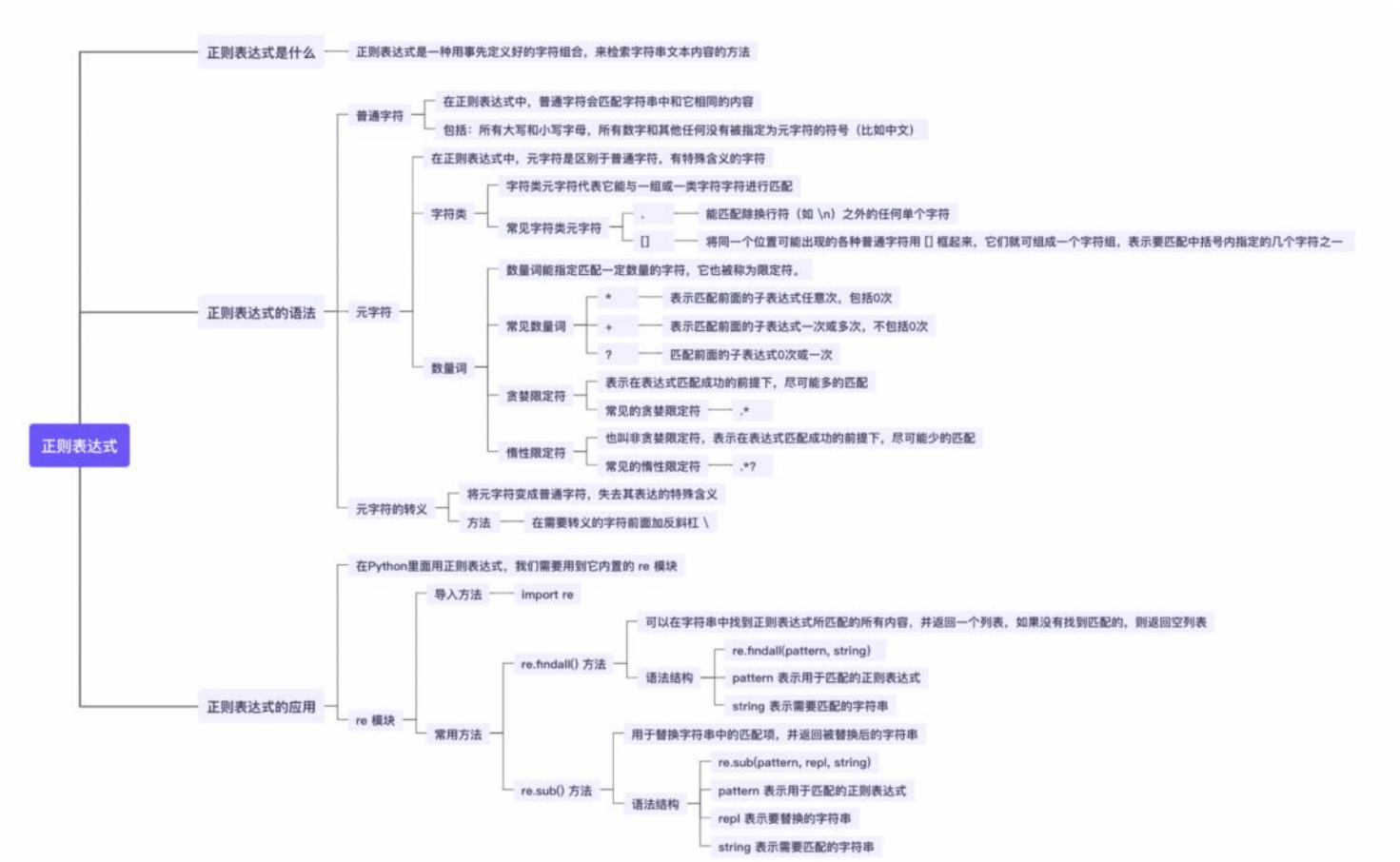
5.3 代码优化

好了，剩下的工作就交给你啦！



6. 总结与复习

先总结一下今天的内容，来看下图：



下面简单复习一下今天的重点知识。

今天我们主要学习的是**正则表达式**。

正则表达式的元字符中，我们主要学习了字符类元字符、数量词。

其中字符类的元字符，我们接触到了`.`还有`[]`，其中：

1）`.` 能匹配除换行符（如 \n）之外的任何单个字符；

2）`[]` 可以将同一个位置可能出现的各种普通字符用[]框起来组成一个字符组。其中，字符组的任何一个字符都可进行匹配。

其他字符类元字符可以参考下面这张表。👇

常见字符类元字符

字符	说明
.	匹配除换行符（\n、\r）之外的任何单个字符。
\d	匹配一个数字字符
\D	匹配一个非数字字符
\n	匹配一个换行符
\s	匹配任何空白字符，包括空格、制表符、换页符等等
\S	匹配任何非空白字符
\w	匹配字母、数字、下划线字符
\W	匹配非字母、数字、下划线字符

然后，我们学习了数量词中的`*`、`+`、`?`。

常见数量词

字符	说明
*	匹配前面的子表达式零次或多次。
+	匹配前面的子表达式一次或多次。
?	匹配前面的子表达式零次或一次。

此外，我们还重点学习了贪婪限定符和惰性限定符，其中：

1）贪婪限定符，就在表达式匹配成功的前提下，它会尽可能多地匹配内容，常见的有`.*`；

2）惰性限定符，就是在表达式匹配成功的前提下，尽可能少的匹配，常见的有`.*?` 。

我们还应用了 re 模块中的 **re.findall()** 函数、 **re.sub()** 函数，它们分别能用于查找、替换字符串的文本内容。

当然，还有许多语法值得我们去拓展学习。同学们可以在课后通过教辅来展开学习。

今天的课程到这里就结束了，咱们下节课再见！