目录

Python中正则表达式：re模块详解

- 最新版本： `v1.3`
- 更新时间： `20201212`

简介

整理Python中正则表达式re模块，解释常见正则函数的含义、语法，以及给出详细的例子详尽阐述具体如何使用。常见正则函数包括re.search、re.sub、re.match、re.findall、re.finditer等，最后总结出相关心得。以及详细对比re.search、re.findall、re.finditer的详细用法和区别。

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/python_regex_re_intro: Python中正则表达式：re模块详解

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

- Python中正则表达式：re模块详解 book.crifan.com
- Python中正则表达式：re模块详解 crifan.github.io

离线下载阅读

- Python中正则表达式：re模块详解 PDF
- Python中正则表达式：re模块详解 ePub
- Python中正则表达式：re模块详解 Mobi

版权说明

鸣谢

感谢我的老婆陈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。

更多其他电子书

本人 `crifan` 还写了其他 `100+` 本电子书教程，感兴趣可移步至：

crifan/crifan_ebook_readme: Crifan的电子书的使用说明

re简介

关于正则表达式

关于正则表达式，之前已整理了教程：

- 新
  - 应用广泛的超强搜索：正则表达式
- 旧
  - 正则表达式学习心得

关于 `Python` 中的 `re`

关于Python的正则表达式方面的教程，之前也有整理过：

- 旧
  - Python专题教程：正则表达式re模块详解
  - 【教程】详解Python正则表达式 – 在路上
    - 之前没完全写完

此处再次重新整理。

`re` 是 `Python` 中内置的正则表达式模块。功能十分强大。

先概述如下：

- 最常用：
  - 搜索：`re.search`
  - 替换：`re.sub`
  - 匹配：`re.match`
- 其他
  - 匹配所有：`re.findall`
  - 匹配所有，且每个都可获取匹配对象的详情：`re.finditer`
    - = `re.findall` + `re.search`

下面来详细介绍其相关功能。

官网资料

此处先贴出来，`Python` 官网关于 `re` 的的文档资料，供后续参考：

- 官网文档
  - 英文
    - re — Regular expression operations - Python 3
  - 中文

        - re --- 正则表达式操作 — Python 3 文档
- 官网教程
    - 英文
        - Regular Expression HOWTO — Python 3 documentation
    - 中文
        - 正则表达式HOWTO — Python 3 文档

re.search

TODO：之前已写过相关的帖子，抽空把内容整理至此。

- 【教程】详解Python正则表达式之：‘.’ dot 点 匹配任意单个字符
- 【教程】详解Python正则表达式之：‘^’ Caret 脱字符/插入符 匹配字符串开始
- 【教程】详解Python正则表达式之：‘$’ dollar 美元符号 匹配字符串末尾
- 【教程】详解Python正则表达式之：‘*’ star 星号 匹配0或多个
- 【教程】详解Python正则表达式之：[] bracket 中括号 匹配某集合内的字符
- 【教程】详解Python正则表达式之：‘|’ vertical bar 竖杠
- 【教程】详解Python正则表达式之：\s 匹配任一空白字符
- 【教程】详解Python正则表达式之：re.LOCALE re.L 本地化标志
- 【教程】详解Python正则表达式之：re.UNICODE re.U 统一码标志

和其他一些心得：

- 【已解决】Python 3中用正则匹配多段的脚本内容 – 在路上

官网文档：

- 英文
  - re.search — Regular expression operations — Python 3 documentation

    > re.search(pattern, string, flags=0)
    >
    > Scan through string looking for the first location where the regular expression pattern produces a match, and return a corresponding match object. Return None if no position in the string matches the pattern; note that this is different from finding a zero-length match at some point in the string.

- 中文
  - re.search --- 正则表达式操作 — Python 3 文档

    > re.search(pattern, string, flags=0)
    >
    > 扫描整个 字符串 找到匹配样式的第一个位置，并返回一个相应的 匹配对象。如果没有匹配，就返回一个 None； 注意这和找到一个零长度匹配是不同的。

分组group

TODO：把下面之前写的帖子的内容整理至此

- 【教程】详解Python正则表达式之： (…) group 分组
- 【教程】详解Python正则表达式之： (?…) extension notation 扩展助记符
- 【教程】详解Python正则表达式之： (?:…) non-capturing group 非捕获组
- 【教程】详解Python正则表达式之： (?P…) named group 带命名的组
- 【教程】详解Python正则表达式之： (?P=name) match earlier named group 匹配前面已命名的组
- 【教程】详解Python正则表达式之： (?(id/name)yes-pattern|no-pattern) 条件性匹配
- 【教程】详解Python正则表达式之： (?=…) lookahead assertion 前向匹配 /前向断言
- 【教程】详解Python正则表达式之： (?<=…) positive lookbehind assertion 后向匹配 /后向断言
- 【教程】详解Python正则表达式之： (?<=…) positive lookbehind assertion 后向匹配 /后向断言

命名的组

非捕获组

官网的

- 英文解释

```
(?...)
    This is an extension notation (a '?' following a '(' is not meaningful otherwise).
    The first character after the '?' determines what the meaning and further syntax of th
e construct is.
    Extensions usually do not create a new group; (?P name...) is the only exception to t
his rule.
    Following are the currently supported extensions.

(?:...)
    A non-capturing version of regular parentheses.
    Matches whatever regular expression is inside the parentheses,
    but the substring matched by the group cannot be retrieved after performing a match or
 referenced later in the pattern.

(?P name...)
    Similar to regular parentheses, but the substring matched by the group is accessible v
ia the symbolic group name name.
    Group names must be valid Python identifiers, and each group name must be defined only
 once within a regular expression.
    A symbolic group is also a numbered group, just as if the group were not named.
```

- 中文解释

```
(?…)
这是个扩展标记法 （一个 '?' 跟随 '(' 并无含义）。
'?' 后面的第一个字符决定了这个构建采用什么样的语法。
这种扩展通常并不创建新的组合； (?P name ...) 是唯一的例外。 以下是目前支持的扩展。

(?:…)
正则括号的非捕获版本。 匹配在括号内的任何正则表达式，
但该分组所匹配的子字符串 不能 在执行匹配后被获取或是之后在模式中被引用。

(?P name …)
（命名组合）类似正则组合，但是匹配到的子串组在外部是通过定义的 name 来获取的。
组合名必须是有效的Python标识符，并且每个组合名只能用一个正则表达式定义，只能定义一次。
一个符号组合同样是一个数字组合，就像这个组合没有被命名一样。

命名组合可以在三种上下文中引用。
如果样式是 (?P quote['"]).*?(?P=quote)
（也就是说，匹配单引号或者双引号括起来的字符串）：

引用组合 "quote" 的上下文          引用方法
在正则式自身内                    (?P=quote) (如示)
                                \1
```

```
处理匹配对象 m                    m.group('quote')
                                 m.end('quote') (等)
传递到 re.sub() 里的 repl 参数中   \g quote
                                 \g 1
                                 \1
```

代码演示如何使用：

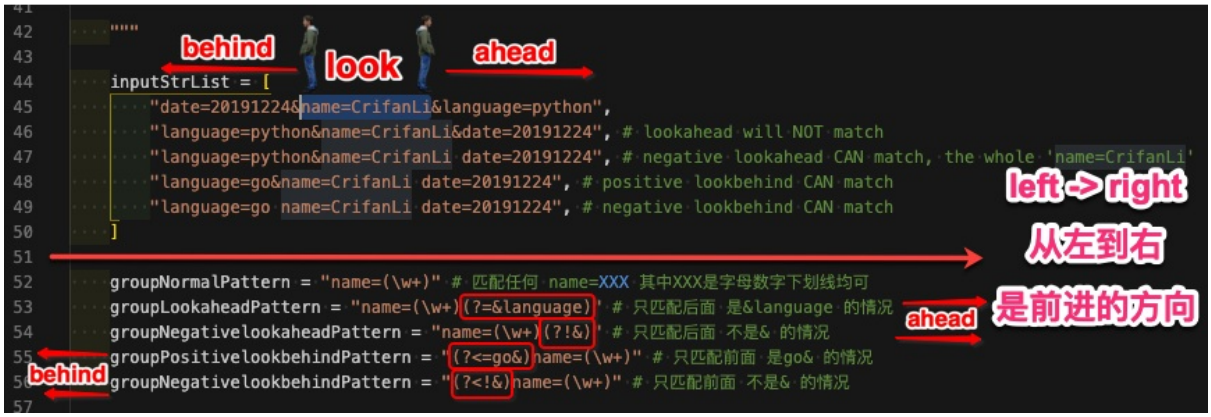
```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Author: Crifan Li
# Update: 20191223
# Function: Demo python re (?:…) non-capture usage

import re

def demoReNonCapturingGroup():
    normalGroupPattern = """(请?点击[“”"'])(.+?)([“”"'])"""
    nonCapturingGroupPattern = """(?:请?点击[“”"'])(?:.+?)(?:[“”"'])"""
    namedGroupPattern = """(请?点击[“”"'])(?P<strToClick>.+?)([“”"'])"""

    inputStrList = [
        "请点击“升级”按钮取210000经验,需50元宝提取经验",
        "点击”+“, 放入吞噬道具",
        "请点击”战骑“",
    ]
    for eachInputStr in inputStrList:
        print("="*60)
        foundNormalGroup = re.search(normalGroupPattern, eachInputStr)
        foundNonCapturingGroup = re.search(nonCapturingGroupPattern, eachInputStr)
        foundNamedGroup = re.search(namedGroupPattern, eachInputStr)

        if foundNormalGroup and foundNonCapturingGroup and foundNamedGroup:
            print("-"*60)
            print("eachInputStr=%s -> " % eachInputStr)

            matchedWholeStrNormal = foundNormalGroup.group(0)
            print("matchedWholeStrNormal=%s" % matchedWholeStrNormal)
            matchedWholeStrNonCapturing = foundNonCapturingGroup.group(0)
            print("matchedWholeStrNonCapturing=%s" % matchedWholeStrNonCapturing)
            matchedWholeStrNamed = foundNamedGroup.group(0)
            print("matchedWholeStrNamed=%s" % matchedWholeStrNamed)

            matchedGroupListNormal = foundNormalGroup.groups()
            print("matchedGroupListNormal=%s" % (matchedGroupListNormal,))
            matchedGroupListNonCapturing = foundNonCapturingGroup.groups()
            print("matchedGroupListNonCapturing=%s" % (matchedGroupListNonCapturing,))
            matchedGroupListNamed = foundNamedGroup.groups()
            print("matchedGroupListNamed=%s" % (matchedGroupListNamed,))

            strToClickNormal = foundNormalGroup.group(2)
            print("strToClickNormal=%s" % strToClickNormal)
```

```
            strToClickNamed = foundNamedGroup.group("strToClick")
            print("strToClickNamed=%s" % strToClickNamed)

            pass

    pass

    # ================================================================
    # ----------------------------------------------------------------
    # eachInputStr=请点击“升级”按钮取210000经验,需50元宝提取经验 ->
    # matchedWholeStrNormal=请点击“升级”
    # matchedWholeStrNonCapturing=请点击“升级”
    # matchedWholeStrNamed=请点击“升级”
    # matchedGroupListNormal=('请点击“', '升级', '”')
    # matchedGroupListNonCapturing=()
    # matchedGroupListNamed=('请点击“', '升级', '”')
    # strToClickNormal=升级
    # strToClickNamed=升级
    # ================================================================
    # ----------------------------------------------------------------
    # eachInputStr=点击”+“，放入吞噬道具 ->
    # matchedWholeStrNormal=点击”+“
    # matchedWholeStrNonCapturing=点击”+“
    # matchedWholeStrNamed=点击”+“
    # matchedGroupListNormal=('点击”', '+', '“')
    # matchedGroupListNonCapturing=()
    # matchedGroupListNamed=('点击”', '+', '“')
    # strToClickNormal=+
    # strToClickNamed=+
    # ================================================================
    # ----------------------------------------------------------------
    # eachInputStr=请点击”战骑“ ->
    # matchedWholeStrNormal=请点击”战骑“
    # matchedWholeStrNonCapturing=请点击”战骑“
    # matchedWholeStrNamed=请点击”战骑“
    # matchedGroupListNormal=('请点击”', '战骑', '“')
    # matchedGroupListNonCapturing=()
    # matchedGroupListNamed=('请点击”', '战骑', '“')
    # strToClickNormal=战骑
    # strToClickNamed=战骑

    # 结论：
    # non-capturing group = 非捕获组
    # 含义：正常去匹配，但是匹配后的match的group中，是无法获取到对应的值的
    # 用途：（感觉唯一的用途就只是）在匹配时，用group去匹配，容易看懂相关内容的逻辑和关系，但是匹配的结果中，不关心这部分的内容

if __name__ == "__main__":
    demoReNonCapturingGroup()
```

环视断言

- `环视断言` = `look around (assertion)`
  - 包括
    - `look ahead (assertion)` = 正向断言
      - `positive lookahead assertion` : `(?=xxx)`
      - `negative lookahead assertion` : `(?!xxx)`
    - `look behind (assertion)` = 反向断言
      - `positive lookbehind assertion` : `(?<=xxx)`
      - `negative lookbehind assertion` : `(?<!xxx)`

如果觉得look ahead和look behind很费解的话，看这个图，就容易懂了：



总体就2个逻辑：

- 站在 当前所要匹配的内容
  - 往哪看
    - ahead：向前 向右 ➡ 当前字符串继续往后的方向
      - 从左到右 叫做 向前，属于正向
    - behind：向左 ◀ 向后 ⬅ 当前字符串之前的方向
      - 所以会额外加上一个 `<` 小于号 表示向后看的意思
        - `(?<=xxx)`
        - `(?<!xxx)`
  - positive/negative：
    - positive=正面的，肯定的，用 等于号 `=` ，意思是：`=xxx`
    - negative=负面的，否定的，用 不等于号 `!` ，意思是：`!=xxx`

==》因此推导出：

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

官网文档：

```
(...)
    Matches whatever regular expression is inside the parentheses, and indicates the start
 and end of a group;
    the contents of a group can be retrieved after a match has been performed,
    and can be matched later in the string with the \number special sequence, described be
low.
    To match the literals '(' or ')', use \( or \), or enclose them inside a character cla
ss: [(], [)].


(?...)
    Matches if ... matches next, but doesn't consume any of the string.
    This is called a lookahead assertion.
    For example, Isaac (?=Asimov) will match 'Isaac ' only if it's followed by 'Asimov'.


(?!...)
    Matches if ... doesn't match next.
    This is a negative lookahead assertion.
    For example, Isaac (?!Asimov) will match 'Isaac ' only if it's not followed by 'Asimov'
.


(?<=...)
    Matches if the current position in the string is preceded by a match for ... that ends
 at the current position.
    This is called a positive lookbehind assertion.
    (?<=abc)def will find a match in 'abcdef', since the lookbehind will back up 3 charact
ers and check if the contained pattern matches.
    The contained pattern must only match strings of some fixed length, meaning that abc o
r a|b are allowed, but a* and a{3,4} are not.
    Note that patterns which start with positive lookbehind assertions will not match at t
he beginning of the string being searched;


(?<!...)
    Matches if the current position in the string is not preceded by a match for ....
    This is called a negative lookbehind assertion.
    Similar to positive lookbehind assertions, the contained pattern must only match strin
gs of some fixed length.
    Patterns which start with negative lookbehind assertions may match at the beginning of
 the string being searched.
```

代码详细解释：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Author: Crifan Li
# Update: 20191224
```

```
# Function: Demo python re lookahead and lookbehind group

import re

def demoReLookAheadBehind():
    inputStrList = [
        "date=20191224&name=CrifanLi&language=python",
        "language=python&name=CrifanLi&date=20191224", # lookahead will NOT match
        "language=python&name=CrifanLi date=20191224", # negative lookahead CAN match, the
 whole 'name=CrifanLi'
        "language=go&name=CrifanLi date=20191224", # positive lookbehind CAN match
        "language=go name=CrifanLi date=20191224", # negative lookbehind CAN match
    ]

    groupNormalPattern = "name=(\w+)" # 匹配任何 name=XXX 其中XXX是字母数字下划线均可
    groupLookaheadPattern = "name=(\w+)(?=&language)" # 只匹配后面 是&language 的情况
    groupNegativelookaheadPattern = "name=(\w+)(?!&)" # 只匹配后面 不是& 的情况
    groupPositivelookbehindPattern = "(?<=go&)name=(\w+)" # 只匹配前面 是go& 的情况
    groupNegativelookbehindPattern = "(?<!&)name=(\w+)" # 只匹配前面 不是& 的情况

    for curIdx, eachInputStr in enumerate(inputStrList):
        print("\n%s [%d] %s %s" % ("="*20, curIdx, eachInputStr, "="*20))

        print("%s %s %s" % ("-"*10, "normal group", "-"*10))
        foundGroupNormal = re.search(groupNormalPattern, eachInputStr)
        print("foundGroupNormal=%s" % foundGroupNormal)
        if foundGroupNormal:
            wholeMatchStrNormal = foundGroupNormal.group(0)
            print("wholeMatchStrNormal=%s" % wholeMatchStrNormal)
            matchedGroupsNormal = foundGroupNormal.groups()
            print("matchedGroupsNormal=%s" % (matchedGroupsNormal, ))
            foundName = foundGroupNormal.group(1)
            print("foundName=%s" % foundName)

        print("%s %s %s" % ("-"*10, "lookahead group", "-"*10))
        foundGroupLookahead = re.search(groupLookaheadPattern, eachInputStr)
        print("foundGroupLookahead=%s" % foundGroupLookahead)
        if foundGroupLookahead:
            wholeMatchStrLookahead = foundGroupLookahead.group(0)
            print("wholeMatchStrLookahead=%s" % wholeMatchStrLookahead)
            matchedGroupsLookahead = foundGroupLookahead.groups()
            print("matchedGroupsLookahead=%s" % (matchedGroupsLookahead, ))
            foundName = foundGroupLookahead.group(1)
            print("foundName=%s" % foundName)

        print("%s %s %s" % ("-"*10, "negative lookahead group", "-"*10))
        foundGroupNegativelookahead = re.search(groupNegativelookaheadPattern, eachInputStr
)
        print("foundGroupNegativelookahead=%s" % foundGroupNegativelookahead)
        if foundGroupNegativelookahead:
            wholeMatchStrNegativelookahead = foundGroupNegativelookahead.group(0)
            print("wholeMatchStrNegativelookahead=%s" % wholeMatchStrNegativelookahead)
            matchedGroupsNegativelookahead = foundGroupNegativelookahead.groups()
```

```
            print("matchedGroupsNegativelookahead=%s" % (matchedGroupsNegativelookahead, ))

            foundName = foundGroupNegativelookahead.group(1)
            print("foundName=%s" % foundName)

        print("%s %s %s" % ("-"*10, "positive lookahead group", "-"*10))
        foundGroupPositivelookbehind = re.search(groupPositivelookbehindPattern, eachInput
Str)
        print("foundGroupPositivelookbehind=%s" % foundGroupPositivelookbehind)
        if foundGroupPositivelookbehind:
            wholeMatchStrPositivelookbehind = foundGroupPositivelookbehind.group(0)
            print("wholeMatchStrPositivelookbehind=%s" % wholeMatchStrPositivelookbehind)
            matchedGroupsPositivelookbehind = foundGroupPositivelookbehind.groups()
            print("matchedGroupsPositivelookbehind=%s" % (matchedGroupsPositivelookbehind,
))
            foundName = foundGroupPositivelookbehind.group(1)
            print("foundName=%s" % foundName)

        print("%s %s %s" % ("-"*10, "positive lookahead group", "-"*10))
        foundGroupNegativelookbehind = re.search(groupNegativelookbehindPattern, eachInput
Str)
        print("foundGroupNegativelookbehind=%s" % foundGroupNegativelookbehind)
        if foundGroupNegativelookbehind:
            wholeMatchStrNegativelookbehind = foundGroupNegativelookbehind.group(0)
            print("wholeMatchStrNegativelookbehind=%s" % wholeMatchStrNegativelookbehind)
            matchedGroupsNegativelookbehind = foundGroupNegativelookbehind.groups()
            print("matchedGroupsNegativelookbehind=%s" % (matchedGroupsNegativelookbehind,
))
            foundName = foundGroupNegativelookbehind.group(1)
            print("foundName=%s" % foundName)

    # ==================== [0] date=20191224&name=CrifanLi&language=python ==========
=====
    # ---------- normal group ----------
    # foundGroupNormal=<re.Match object; span=(14, 27), match='name=CrifanLi'>
    # wholeMatchStrNormal=name=CrifanLi
    # matchedGroupsNormal=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- lookahead group ----------
    # foundGroupLookahead=<re.Match object; span=(14, 27), match='name=CrifanLi'>
    # wholeMatchStrLookahead=name=CrifanLi
    # matchedGroupsLookahead=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- negative lookahead group ----------
    # foundGroupNegativelookahead=<re.Match object; span=(14, 26), match='name=CrifanL'>
    # wholeMatchStrNegativelookahead=name=CrifanL
    # matchedGroupsNegativelookahead=('CrifanL',)
    # foundName=CrifanL
    # ---------- positive lookahead group ----------
    # foundGroupPositivelookbehind=None
    # ---------- positive lookahead group ----------
    # foundGroupNegativelookbehind=None
```

```
    # ==================== [1] language=python&name=CrifanLi&date=20191224 ==============
=====
    # ---------- normal group ----------
    # foundGroupNormal=<re.Match object; span=(16, 29), match='name=CrifanLi'>
    # wholeMatchStrNormal=name=CrifanLi
    # matchedGroupsNormal=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- lookahead group ----------
    # foundGroupLookahead=None
    # ---------- negative lookahead group ----------
    # foundGroupNegativelookahead=<re.Match object; span=(16, 28), match='name=CrifanL'>
    # wholeMatchStrNegativelookahead=name=CrifanL
    # matchedGroupsNegativelookahead=('CrifanL',)
    # foundName=CrifanL
    # ---------- positive lookahead group ----------
    # foundGroupPositivelookbehind=None
    # ---------- positive lookahead group ----------
    # foundGroupNegativelookbehind=None

    # ==================== [2] language=python&name=CrifanLi date=20191224 ==============
=====
    # ---------- normal group ----------
    # foundGroupNormal=<re.Match object; span=(16, 29), match='name=CrifanLi'>
    # wholeMatchStrNormal=name=CrifanLi
    # matchedGroupsNormal=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- lookahead group ----------
    # foundGroupLookahead=None
    # ---------- negative lookahead group ----------
    # foundGroupNegativelookahead=<re.Match object; span=(16, 29), match='name=CrifanLi'>
    # wholeMatchStrNegativelookahead=name=CrifanLi
    # matchedGroupsNegativelookahead=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- positive lookahead group ----------
    # foundGroupPositivelookbehind=None
    # ---------- positive lookahead group ----------
    # foundGroupNegativelookbehind=None

    # ==================== [3] language=go&name=CrifanLi date=20191224 ====================

    # ---------- normal group ----------
    # foundGroupNormal=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrNormal=name=CrifanLi
    # matchedGroupsNormal=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- lookahead group ----------
    # foundGroupLookahead=None
    # ---------- negative lookahead group ----------
    # foundGroupNegativelookahead=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrNegativelookahead=name=CrifanLi
    # matchedGroupsNegativelookahead=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- positive lookahead group ----------
```

```
    # foundGroupPositivelookbehind=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrPositivelookbehind=name=CrifanLi
    # matchedGroupsPositivelookbehind=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- positive lookahead group ----------
    # foundGroupNegativelookbehind=None

    # ==================== [4] language=go name=CrifanLi date=20191224 ====================

    # ---------- normal group ----------
    # foundGroupNormal=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrNormal=name=CrifanLi
    # matchedGroupsNormal=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- lookahead group ----------
    # foundGroupLookahead=None
    # ---------- negative lookahead group ----------
    # foundGroupNegativelookahead=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrNegativelookahead=name=CrifanLi
    # matchedGroupsNegativelookahead=('CrifanLi',)
    # foundName=CrifanLi
    # ---------- positive lookahead group ----------
    # foundGroupPositivelookbehind=None
    # ---------- positive lookahead group ----------
    # foundGroupNegativelookbehind=<re.Match object; span=(12, 25), match='name=CrifanLi'>
    # wholeMatchStrNegativelookbehind=name=CrifanLi
    # matchedGroupsNegativelookbehind=('CrifanLi',)
    # foundName=CrifanLi

if __name__ == "__main__":
    demoReLookAheadBehind()
```

对于代码中的结果，总结起来就是：

- name=(\w+)：普通的group
  - 匹配结果
    - 5个都匹配
      - date=20191224&name=CrifanLi&language=python
      - language=python&name=CrifanLi&date=20191224
      - language=python&name=CrifanLi date=20191224
      - language=go&name=CrifanLi date=20191224
      - language=go name=CrifanLi date=20191224
    - 匹配到内容都是：
      - name=CrifanLi
  - 解析：因为只是普通的(xxx)的组，没有限制，所以都能匹配到
- name=(\w+)(?=&language)：lookahead=positive lookahead=正向先行断言
  - 匹配结果
    - 只匹配了1个：
      - date=20191224&name=CrifanLi&language=python

        - 其余4个都不匹配
            - language=python&name=CrifanLi&date=20191224
            - language=python&name=CrifanLi date=20191224
            - language=go&name=CrifanLi date=20191224
            - language=go name=CrifanLi date=20191224
    - 解析：
        - (?=&language) 表示 后面一定是 &language
            - 而上面4个的后面，分别是：
                - &date=
                - date=
                - date=
                - date=
            - 所以都不匹配
- name=(\w+)(?!&)：negative look ahead=负向先行断言
    - 匹配结果
        - 5个都匹配到了，但是匹配的内容不一样
            - 2个匹配到了：name=CrifanL
                - date=20191224&name=CrifanLi&language=python
                - language=python&name=CrifanLi&date=20191224
            - 3个匹配到了：name=CrifanLi
                - language=python&name=CrifanLi date=20191224
                - language=go&name=CrifanLi date=20191224
                - language=go name=CrifanLi date=20191224
    - 解析
        - 注意 前2个匹配到的 最后没有i，是CrifanL，而不是CrifanLi
        - 因为(?!&)表示 后面不能是 &
            - 所以类似于
                - name=CrifanLi&language
                - name=CrifanLi&date
            - 这种，只能匹配到L，而不是i，因为i后面是&，此处要求后面不能是&
- (?<=go&)name=(\w+)：positive look behind=正向后行断言
    - 匹配结果
        - 只匹配到1个
            - language=go&name=CrifanLi date=20191224
    - 解析
        - 因为此处(?<=go&)意思是，前面一定要是 go& 所以只有这个匹配
        - 其他的
            - 20191224&name=
            - python&name=
            - python&name=
            - go name=
        - 中name=的前面 都不符合条件
- (?<!&)name=(\w+)：negative look behind=负向后行断言
    - 匹配结果

        - 只匹配到1个：
            - language=go name=CrifanLi date=20191224
    - 解析
        - 因为(?<!&)的意思是：前面不能是 &
            - 所以只有
                - go name=
            - 这个符合，而其余的
                - 20191224&name=
                - python&name=
                - python&name=
                - go&name=
            - name=前面都是&，所以不符合条件，不匹配

(positive) look behind

官网解释：

```
(?<=...)
    Matches if the current position in the string is preceded by a match for ... that ends
 at the current position.
    This is called a positive lookbehind assertion.
    (?<=abc)def will find a match in 'abcdef', since the lookbehind will back up 3 charact
ers and check if the contained pattern matches.
    The contained pattern must only match strings of some fixed length, meaning that abc o
r a b are allowed, but a* and a{3,4} are not.
    Note that patterns which start with positive lookbehind assertions will not match at t
he beginning of the string being searched;

(?P<name>...)
    Similar to regular parentheses, but the substring matched by the group is accessible v
ia the symbolic group name name.
    Group names must be valid Python identifiers, and each group name must be defined only
 once within a regular expression.
    A symbolic group is also a numbered group, just as if the group were not named.
```

用代码详细解释：

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Author: Crifan Li
# Update: 20191224
# Function: Demo python re lookbehind group

import re

def demoReLookbehind():
    namedGroupPattern = "(SID=)(?P<sidValue>[^&]+)"
    lookBehindGroupPattern = "(?<=SID=)(?P<sidValue>[^&]+)"
```

```
    """
    正则含义的解释:
        (?<=SID=)[^&]+
            (?<=SID=) 属于(?<=XXX)，其中XXX是"SID="这个固定长度的4个字符的字符串 用于匹配你要的值的前面的部分 SID=YYY中的 SID=
            [^&]+
                [] 中括号中是所允许出现的字符
                ^ 是取反，除了...之外的，所以^&就是除了&字符之外的，因为你要匹配的字符串是 SID=8E3qOreRiOnbhkl84Uc&YYY 可以避免匹配到 最后的&和YYY
                + 表示1个或更多个 直到遇到 不允许出现的&字符，匹配此处的 即从8到c，即8E3qOreRiOnbhkl84Uc
    """

    inputStrList = [
        "GeneralSearch&SID=8E3qOreRiOnbhkl84Uc&preferencesSaved",
    ]
    for eachInputStr in inputStrList:
        print("="*60)
        foundNamedGroup = re.search(namedGroupPattern, eachInputStr)
        foundLookbehindGroup = re.search(lookBehindGroupPattern, eachInputStr)
        if foundNamedGroup and foundLookbehindGroup:
            print("foundNamedGroup=%s" % foundNamedGroup)
            print("foundLookbehindGroup=%s" % foundLookbehindGroup)

            groupsNamedGroup = foundNamedGroup.groups()
            print("groupsNamedGroup=%s" % (groupsNamedGroup, ))
            groupsLookbehindGroup = foundLookbehindGroup.groups()
            print("groupsLookbehindGroup=%s" % (groupsLookbehindGroup, ))

            wholeStrNamedGroup = foundNamedGroup.group(0)
            print("wholeStrNamedGroup=%s" % wholeStrNamedGroup)
            wholeStrLookbehindGroup = foundLookbehindGroup.group(0)
            print("wholeStrLookbehindGroup=%s" % wholeStrLookbehindGroup)

            sidValueNamedGroup = foundNamedGroup.group("sidValue")
            print("sidValueNamedGroup=%s" % sidValueNamedGroup)
            sidValueLookbehindGroup = foundLookbehindGroup.group("sidValue")
            print("sidValueLookbehindGroup=%s" % sidValueLookbehindGroup)

    # foundNamedGroup=<re.Match object; span=(14, 37), match='SID=8E3qOreRiOnbhkl84Uc'>
    # foundLookbehindGroup=<re.Match object; span=(18, 37), match='8E3qOreRiOnbhkl84Uc'>
    # groupsNamedGroup=('SID=', '8E3qOreRiOnbhkl84Uc')
    # groupsLookbehindGroup=('8E3qOreRiOnbhkl84Uc',)
    # wholeStrNamedGroup=SID=8E3qOreRiOnbhkl84Uc
    # wholeStrLookbehindGroup=8E3qOreRiOnbhkl84Uc
    # sidValueNamedGroup=8E3qOreRiOnbhkl84Uc
    # sidValueLookbehindGroup=8E3qOreRiOnbhkl84Uc

if __name__ == "__main__":
    demoReLookbehind()
```

新： 2020-02-11 22:05:55

re.sub

官网文档:

- 英文
  - re.sub — Regular expression operations — Python 3 documentation

> re.sub(pattern, repl, string, count=0, flags=0)
>
> Return the string obtained by replacing the leftmost non-overlapping occurrences of pattern in string by the replacement repl. If the pattern isn't found, string is returned unchanged. repl can be a string or a function; if it is a string, any backslash escapes in it are processed. That is, `\n` is converted to a single newline character, `\r` is converted to a carriage return, and so forth. Unknown escapes of ASCII letters are reserved for future use and treated as errors. Other unknown escapes such as `\&` are left alone. Backreferences, such as `\6`, are replaced with the substring matched by group `6` in the pattern. For example:
>
> ```
> >> re.sub(r'def\s+([a-zA-Z_][a-zA-Z_0-9]*)\s*\(\s*\):',
> ...        r'static PyObject*\npy_\1(void)\n{',
> ...        'def myfunc():')
> 'static PyObject*\npy_myfunc(void)\n{'
> ```
>
> If repl is a function, it is called for every non-overlapping occurrence of pattern. The function takes a single match object argument, and returns the replacement string. For example:
>
> ```
> >> def dashrepl(matchobj):
> ...     if matchobj.group(0) == '-': return ' '
> ...     else: return '-'
> >> re.sub('-{1,2}', dashrepl, 'pro----gram-files')
> 'pro--gram files'
> >> re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE)
> 'Baked Beans & Spam'
> ```
>
> The pattern may be a string or a pattern object.
>
> The optional argument count is the maximum number of pattern occurrences to be replaced; count must be a non-negative integer. If omitted or zero, all occurrences will be replaced. Empty matches for the pattern are replaced only when not adjacent to a previous empty match, so `sub('x*', '-', 'abxd')` returns `'-a-b--d-'`.
>
> In string-type repl arguments, in addition to the character escapes and backreferences described above, `\g<name>` will use the substring matched by the group named `name`, as defined by the `(?P<name>...)` syntax. `\g<number>` uses the corresponding group number; `\g<2>` is therefore equivalent to `\2`, but isn't ambiguous in a

> replacement such as `\g<2>0`. `\20` would be interpreted as a reference to group 20, not a reference to group 2 followed by the literal character `'0'`. The backreference `\g<0>` substitutes in the entire substring matched by the RE.
>
> - Change Log
>   - Changed in version 3.1: Added the optional flags argument.
>   - Changed in version 3.5: Unmatched groups are replaced with an empty string.
>   - Changed in version 3.6: Unknown escapes in pattern consisting of `'\'` and an ASCII letter now are errors.
>   - Changed in version 3.7: Unknown escapes in repl consisting of `'\'` and an ASCII letter now are errors.
>   - Changed in version 3.7: Empty matches for the pattern are replaced when adjacent to a previous non-empty match.

- 中文
  - re.sub --- 正则表达式操作 — Python 3 文档

> re.sub(pattern, repl, string, count=0, flags=0)
>
> 返回通过使用 *repl* 替换在 *string* 最左边非重叠出现的 *pattern* 而获得的字符串。 如果样式没有找到，则不加改变地返回 *string*。 *repl* 可以是字符串或函数；如为字符串，则其中任何反斜杠转义序列都会被处理。 也就是说， `\n` 会被转换为一个换行符， `\r` 会被转换为一个回车附，依此类推。 未知的 `ASCII` 字符转义序列保留在未来使用，会被当作错误来处理。 其他未知转义序列例如 `\&` 会保持原样。 向后引用像是 `\6` 会用样式中第 6 组所匹配到的子字符串来替换。 例如:
>
> ```
> >> re.sub(r'def\s+([a-zA-Z_][a-zA-Z_0-9]*)\s*\(\s*\):',
> ...        r'static PyObject*\npy_\1(void)\n{',
> ...        'def myfunc():')
> 'static PyObject*\npy_myfunc(void)\n{'
> ```
>
> 如果 *repl* 是一个函数，那它会对每个非重复的 *pattern* 的情况调用。这个函数只能有一个 匹配对象 参数，并返回一个替换后的字符串。比如
>
> ```
> >> def dashrepl(matchobj):
> ...     if matchobj.group(0) == '-': return ' '
> ...     else: return '-'
> >> re.sub('-{1,2}', dashrepl, 'pro----gram-files')
> 'pro--gram files'
> >> re.sub(r'\sAND\s', ' & ', 'Baked Beans And Spam', flags=re.IGNORECASE)
> 'Baked Beans & Spam'
> ```
>
> 样式可以是一个字符串或者一个 样式对象 。
>
> 可选参数 *count* 是要替换的最大次数；*count* 必须是非负整数。如果忽略这个参数，或者设置为0，所有的匹配都会被替换。空匹配只在不相临连续的情况被更替，所以 `sub('x*', '-', 'abxd')` 返回 `'-a-b--d-'` 。

> 在字符串类型的 *repl* 参数里，如上所述的转义和向后引用中，`\g<name>` 会使用命名组合 name，（在 `(?P<name>…)` 语法中定义） `\g<number>` 会使用数字组；`\g<2>` 就是 `\2`，但它避免了二义性，如 `\g<2>0` 。 `\20` 就会被解释为组20，而不是组2后面跟随一个字符 `'0'` 。向后引用 `\g<0>` 把 *pattern* 作为一整个组进行引用。
>
> - 更新日志
>   - 在 3.1 版更改: 增加了可选标记参数。
>   - 在 3.5 版更改: 不匹配的组合替换为空字符串。
>   - 在 3.6 版更改: *pattern* 中的未知转义（由 `'\'` 和一个 `ASCII` 字符组成）被视为错误。
>   - 在 3.7 版更改: *repl* 中的未知转义（由 `'\'` 和一个 `ASCII` 字符组成）被视为错误。
>   - 在 3.7 版更改: 样式中的空匹配相邻接时会被替换。

举例

把

```
最后更新：`20190825`
```

中的日期换成最新的，比如 `20200919`

则可以用代码：

```
oldReamdMdStr = "最后更新：`20190825`"
newLastUpdateStr = "最后更新：`20200919`"
newReamdMdStr = re.sub("最后更新：`\d+`", newLastUpdateStr, oldReamdMdStr)
```

详细过程可参考：

- 【已解决】给crifan的gitbook的template添加部署时更新github.io的README

re.match

官网文档：

- 英文
  - re.match — Regular expression operations — Python 3 documentation

    > re.match(pattern, string, flags=0)
    >
    > If zero or more characters at the beginning of string match the regular expression pattern, return a corresponding match object. Return `None` if the string does not match the pattern; note that this is different from a zero-length match.
    >
    > Note that even in MULTILINE mode, re.match() will only match at the beginning of the string and not at the beginning of each line.
    >
    > If you want to locate a match anywhere in string, use search() instead (see also search() vs. match()).

- 中文
  - re.match --- 正则表达式操作 — Python 3 文档

    > re.match(pattern, string, flags=0)
    >
    > 如果 *string* 开始的0或者多个字符匹配到了正则表达式样式，就返回一个相应的 匹配对象 。 如果没有匹配，就返回 `None` ；注意它跟零长度匹配是不同的。
    >
    > 注意即便是 MULTILINE 多行模式， re.match() 也只匹配字符串的开始位置，而不匹配每行开始。
    >
    > 如果你想定位 *string* 的任何位置，使用 search() 来替代（也可参考 search() vs. match() ）

re.findall

官网文档:

- 英文
  - re.findall — Regular expression operations — Python 3 documentation

    > re.findall(pattern, string, flags=0)
    >
    > Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found. If one or more groups are present in the pattern, return a list of groups; this will be a list of tuples if the pattern has more than one group. Empty matches are included in the result.
    >
    > Changed in version 3.7: Non-empty matches can now start just after a previous empty match.

- 中文
  - re.findall --- 正则表达式操作 — Python 3 文档

    > re.findall(pattern, string, flags=0)
    >
    > 对 string 返回一个不重复的 pattern 的匹配列表， string 从左到右进行扫描，匹配按找到的顺序返回。如果样式里存在一到多个组，就返回一个组合列表；就是一个元组的列表（如果样式里有超过一个组合的话）。空匹配也会包含在结果里。
    >
    > 在 3.7 版更改: 非空匹配现在可以在前一个空匹配之后出现了。

re.finditer

`re.finditer` = iterator of `re.findall` =针对 `re.findall` 所返回的字符串相对，每个都是一个匹配的对象 `MatchedObject`

对比：

- `findall` ： 返回 `str` 的list
- `finditer` ： 返回 `MatchObject` 的迭代器 `iterator`
  - 可以针对每个匹配到的字符串，获取其中的 `sub group` 了

可以理解为：

`re.finditer` = `re.findall` + 每个字符串再用 `re.search` 处理

官网文档：

- 英文
  - re — Regular expression operations — Python 3.8.3 documentation
    > re.finditer(pattern, string, flags=0)
    >
    > Return an iterator yielding match objects over all non-overlapping matches for the RE pattern in string. The string is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.
    >
    > Changed in version 3.7: Non-empty matches can now start just after a previous empty match.
- 中文
  - re --- 正则表达式操作 — Python 3.8.3 文档
    > re.finditer(pattern, string, flags=0)
    >
    > pattern 在 string 里所有的非重复匹配，返回为一个迭代器 iterator 保存了 匹配对象 。string 从左到右扫描，匹配按顺序排列。空匹配也包含在结果里。
    >
    > 在 3.7 版更改: 非空匹配现在可以在前一个空匹配之后出现了。

心得

iter对象无法被重复访问

Python中的iterator类型变量，只能被访问（使用/遍历）一次，就空了。无法被重复使用。

用代码举例如下：

```
matchIter = re.finditer(someP, someStr)
for eachMatch in matchIter:
  print("matchIter=%s" % matchIter)
```

```
resultList = list(matchIter) # -> will error, matchIter already None
```

举例

更新crifan的github.io中README.md中book的list

之前折腾：

【已解决】给crifan的gitbook的template添加部署时更新github.io的README

期间，需要对于：

```
    * [老少皆宜的运动：羽毛球](https://crifan.github.io/all_age_sports_badminton/website)
    * [app抓包利器：Charles](https://crifan.github.io/app_capture_package_tool_charles/website)
    * [安卓应用的安全和破解](https://crifan.github.io/android_app_security_crack/website)
```

去提取出，每个book的中文名称 `bookTitle` ，和对应的git仓库地址 `bookRepoName`

可以写成：

```
allBookMatchList = re.finditer("\[(?P<bookTitle>[^]]+)\]\(https://crifan\.github\.io/(?P<bookRepoName>\w+)/website\)", curGitbookListMd)
```

即可获取到对应匹配到的，Match对象的列表

其中每个元素都是一个Match对象，即每个元素都相当于用 `re.search` 去匹配到的结果

所以可以用循环，依次从每个Match对象中，提取出我们要的 `bookTitle` 和 `bookRepoName` 了：

```
  curBookDict = {}
  for curIdx, eachBookMatch in enumerate(allBookMatchList):
    # print("eachBookMatch=%s" % eachBookMatch)
    # print("%s %s %s" % ("-"*10, curIdx, "-"*10))
    bookTitle = eachBookMatch.group("bookTitle")
    # print("bookTitle=%s" % bookTitle)
    bookRepoName = eachBookMatch.group("bookRepoName")
    # print("bookRepoName=%s" % bookRepoName)
    curBookDict[bookRepoName] = bookTitle
```

match vs findall vs finditer

背景需求

希望从自己的Crifan的电子书的使用说明的README.md中提取相关book的信息，比如url，name，title等。

re.search处理单行，返回匹配对象 `Match Object`

对于普通的，单行字符串：

```
* [科学上网](https://book.crifan.com/books/scientific_network_summary/website)
```

一般用 `re.search` 代码去查找：

```
foundBook = re.search("https?://book\.crifan\.com/books/(?P<bookName>\w+)/website", singleLineMdStr)
print("foundBook=%s" % foundBook)
```

代码解析：

- `https?` = `https` 加上问号 `?`
  - 表示： `http` 后面，可能有 `s` ，也可能没有 `s`
  - 用于匹配： `http://xxx` 和 `https://xxx` 的链接
- `(?P<bookName>\w+)`
  - 典型的 `named group` = `命名的组` 的写法
    - 语法： `(?P<yourGroupName>match_pattern)`
      - 此处
        - `yourGroupName` = `bookName`
          - 用于：若匹配到，后续可通过 `foundBook.group("bookName")` 提取这部分的值
        - `match_pattern` = `\w+`
          - 表示：至少1个（1个或更多个）的大小写字母或下划线
          - 用于匹配：此处的 `scientific_network_summary`

得到的是一个 `Match Object`

```
foundBook: re.Match object; span: (9, 73), match: 'https://book.crifan.com/books/scientific_network_'
```

后来典型写法就是提取出所需的部分，比如

```
if foundBook:
```

```
    bookName = foundBook.group("bookName")
    print("bookName=%s" % bookName)
```

得到book的name：

```
bookName scientific_network_summary
```

re.findall处理多行，返回整个匹配字符串或group部分的字符串

对于多行字符串：

```
* 推荐工具
  * [科学上网](https://book.crifan.com/books/scientific_network_summary/website)
  * 编辑器和IDE
    * [总结](https://book.crifan.com/books/editor_ide_summary/website/)
    * 好用工具
      * [VSCode](http://book.crifan.com/books/best_editor_vscode/website)
```

返回单条全部内容

如果只是想要获取所有的book的url，则可以用 `re.findall` ，且不要加组group：

```
allBookUrlList = re.findall("https?://book\.crifan\.com/books/\w+/website", multipleLineDd
Str)
print("allBookUrlList=%s" % allBookUrlList)
```

输出：

```
allBookUrlList ['https://book.crifan.com/books/scientific_network_summary/website', 'https
://book.crifan.com/books/editor_ide_summary/website', 'http://book.crifan.com/books/best_e
ditor_vscode/website']
```

返回单条中的部分内容=组的内容

如果只想要获取，每条匹配的内容中的其中一部分，即其中某个或某几个group组的内容，则加上group：

```
allBookNameList = re.findall("https?://book\.crifan\.com/books/(?P<bookName>\w+)/website",
 multipleLineDdStr, re.S)
print("allBookNameList=%s" % allBookNameList)
```

输出：

```
# allBookNameList=['scientific_network_summary', 'editor_ide_summary', 'best_editor_vscode
']
```

re.findall的两种用法对比

- re.findall的两种用法对比
  - 规则：`"https?://book\.crifan\.com/books/\w+/website"`
    - 不带group组
      - 返回：单条匹配的所有内容，即whole single match string
        - 举例：
          - `'https://book.crifan.com/books/scientific_network_summary/website'`
  - 规则：`https?://book\.crifan\.com/books/(?P<bookName>\w+)/website`
    - 带group组
      - 返回：单条匹配到的所有内容中的其中一部分，即group组的部分，matched group string
        - 举例：
          - `'scientific_network_summary'`

re.finditer处理多行，返回多个(re.search所返回的)匹配对象

而对于re.finditer，可以视为 = `re.findall` + `re.search`

即：就像 `re.findall` 一样，返回多个值，但是每个值，不是 `re.findall` 直接返回（匹配到的）字符串，而是(re.search所返回的)匹配的对象 `Match Object`

下面用具体例子来解释：

`re.finditer` 内部就像 `re.findall`

和之前一样，想要返回所有的book的url，则可以写成：

```
allBookUrlMatchObjIterator = re.finditer("https?://book\.crifan\.com/books/\w+/website", multipleLineDdStr)
print("allBookUrlMatchObjIterator=%s" % allBookUrlMatchObjIterator)
```

此处注意返回的是 `iterator`：

```
# allBookUrlMatchObjIterator=<callable_iterator object at 0x11063f160>
```

可以将其转换为 `list`：

```
allBookUrlMatchObjList = list(allBookUrlMatchObjIterator)
print("allBookUrlMatchObjList=%s" % allBookUrlMatchObjList)
```

得到了 `Match Object` 的 `list`：

```
# allBookUrlMatchObjList=[<re.Match object; span=(19, 83), match='https://book.crifan.com/books/scientific_network_>, <re.Match object; span=(108, 164), match='https://book.crifan.com/books/editor_ide_summary/>, <re.Match object; span=(195, 250), match='http://book.crif
```

```
an.com/books/best_editor_vscode/w>]
```

接着就可以对list去枚举处理了：

```
for curIdx, eachMatchObj in enumerate(allBookUrlMatchObjList):
  singleMatchWholeStr = eachMatchObj.group(0)
  print("[%d] singleMatchWholeStr=%s" % (curIdx, singleMatchWholeStr))
```

从每个匹配的对象获取所需要的完整的url值：

```
# [0] singleMatchWholeStr=https://book.crifan.com/books/scientific_network_summary/website
# [1] singleMatchWholeStr=https://book.crifan.com/books/editor_ide_summary/website
# [2] singleMatchWholeStr=http://book.crifan.com/books/best_editor_vscode/website
```

`re.finditer` 内部就像 `re.search`

而如果想要一次性的匹配得到多个匹配对象 `Match Object`，且每个都可以提取对应的group组的值，则可以给规则中加上group组：

```
allBookNameMatchObjIterator = re.finditer("https?://book\.crifan\.com/books/(?P<bookName>\
w+)/website", multipleLineDdStr)
print("allBookNameMatchObjIterator=%s" % allBookNameMatchObjIterator)
```

同理先是得到 `iterator`：

```
# allBookNameMatchObjIterator=<callable_iterator object at 0x10e9a9fd0>
```

再转换为 `list`：

```
allBookNameMatchObjList = list(allBookNameMatchObjIterator)
print("allBookNameMatchObjList=%s" % allBookNameMatchObjList)
```

即多个匹配对象：

```
# allBookNameMatchObjList=[<re.Match object; span=(19, 83), match='https://book.crifan.com
/books/scientific_network_>, <re.Match object; span=(108, 164), match='https://book.crifan
.com/books/editor_ide_summary/>, <re.Match object; span=(195, 250), match='http://book.cri
fan.com/books/best_editor_vscode/w>]
```

然后继续针对每个匹配对象去处理：

```
for curIdx, eachMatchObj in enumerate(allBookNameMatchObjList):
  singleMatchWholeStr = eachMatchObj.group(0)
  singleMatchBookName = eachMatchObj.group("bookName")
  print("[%d] singleMatchWholeStr=%s, singleMatchBookName=%s" % (curIdx, singleMatchWholeS
tr, singleMatchBookName))
```

即可，既能获取到 单个匹配的完整字符串，又能获取到 每个匹配的全部字符串中特定的组的内容了：

```
# [0] singleMatchWholeStr=https://book.crifan.com/books/scientific_network_summary/website
, singleMatchBookName=scientific_network_summary
# [1] singleMatchWholeStr=https://book.crifan.com/books/editor_ide_summary/website, single
MatchBookName=editor_ide_summary
# [2] singleMatchWholeStr=http://book.crifan.com/books/best_editor_vscode/website, singleM
atchBookName=best_editor_vscode
```

附录：

完整代码详见：

reSearchFindallFinditer.py

re应用举例

下面给出之前自己用 `re` 实现过的各种需求和实际的案例，供参考。

re.search实例

提取csdn帖子地址

```
foundLastListPageUrl = re.search('<a\s+?href="(?P<lastListPageUrl>/\w+?/article/list/\d+)"
>尾页</a>', homeRespHtml, re.I)
logging.debug("foundLastListPageUrl=%s", foundLastListPageUrl)

if(foundLastListPageUrl):
    lastListPageUrl = foundLastListPageUrl.group("lastListPageUrl")
```

详见：

https://github.com/crifan/BlogsToWordpress/blob/master/libs/crifan/blogModules/BlogCsdn.py

从内容中

```
<a href="/chenglinhust/article/list/22">尾页</a>
```

提取出

```
/chenglinhust/article/list/22
```

提取csdn帖子的标题

```
foundTitle = re.search('<span class="link_title"><a href="[\w/]+?">\s*(<font color="red">\
[置顶\]</font>)?\s*(?P<titleHtml>.+?)\s*</a>\s*</span>', html, re.S)
logging.debug("foundTitle=%s", foundTitle)

if(foundTitle):
    titleHtml = foundTitle.group("titleHtml")
    logging.debug("titleHtml=%s", titleHtml)
```

详见：

https://github.com/crifan/BlogsToWordpress/blob/master/libs/crifan/blogModules/BlogCsdn.py

从内容中

```
<span class="link_title"><a href="/v_july_v/article/details/6543438">
<font color="red">[置顶]</font>
程序员面试、算法研究、编程艺术、红黑树4大系列集锦与总结
</a></span>
```

或

```
<span class="link_title"><a href="/chdhust/article/details/7252155">
        windows编程中wParam和lParam消息

        </a>
        </span>
```

提取出

```
程序员面试、算法研究、编程艺术、红黑树4大系列集锦与总结
```

或

```
windows编程中wParam和lParam消息
```

re.sub实例

Evernote的content处理

去掉最开始的xml头

处理前：

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>\n<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">\n<en-note>xxx
```

代码：

```
re.sub('<\?xml version="1.0" encoding="UTF-8" standalone="no"\?>\s*', "", noteHtmlWithXml)
```

处理后：

```
<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">\n<en-note>xxx
```

去掉开始的en-note头

处理前：

```
<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">xxx
```

或：

```
<!DOCTYPE en-note SYSTEM "http://xml.evernote.com/pub/enml2.dtd">\nxxx
```

或：

```
<!DOCTYPE en-note SYSTEM 'http://xml.evernote.com/pub/enml2.dtd'>xxx
```

代码：

```
noteHtmlNoEnNote = re.sub("""<!DOCTYPE en-note SYSTEM (("|('))http://xml\.evernote\.com/pub/enml2\.dtd(("|('))>\s*""", "", eachNoteWithEnNote)
```

处理后：

```
xxx
```

代码解析：

- `(("")|('))`：用于匹配双引号 `"` 或 `'` 都支持
- `\s*`：用于匹配最后的 `\n`

处理en-meida的tag

处理前：

```
<en-media hash="7c54d8d29cccfcfe2b48dd9f952b715b" type="image/png"></en-media>
```

代码：

```
re.sub("(?P<enMedia><en-media\s+[^<>]+)>\s*</en-media>", "\g<enMedia> />", noteHtmlEnMedia)
```

处理后：

```
<en-media hash "7c54d8d29cccfcfe2b48dd9f952b715b" type "image/png" />
```

替换最顶层tag

处理前：

```
<html>
  xxx
  yyy
  zzz
</html>
```

代码：

```
noteContent = re.sub('<html>(?P<contentBody>.+)</html>', "<en-note>\g<contentBody></en-note>", noteHtml)
```

处理后：

```
<en-note>
  xxx
  yyy
  zzz
</en-note>
```

代码解析：

- `(?P<contentBody>.+)` 和 `\g<contentBody>`
  - `(?P<contentBody>.+)`：是替换前的 pattern，是普通的命名的组 `named group`
  - `\g<contentBody>`：是被替换为的 `repl` 的内容，其中可以用 `\g<groupName>`，表示引用替换内容中的命名的组
    - 此处用来：引用之前html的内容主体contentBody部分的字符串

re.finditer实例

从模式对话中提取出每段对话的信息

输入字符串：

```
Place: School canteen
Topic: food
Tittle:Have lunch
Age: 3-4
J: What did you have for lunch?
L: I ate rice, fish and bread.
J: Do you like rice?
L: Yes, I do.
J: Do you like fish?
L: Yes, I do.
J: Do you like bread?
L: No, I don’t.
J: What did you drink?
L: I drank milk.
J: Do you like milk?
L: Yes, I do.

Place: home
Topic: house
Tittle: Doing housework
Age: 4-5
J: Do you like cooking, mom?
M: Yes, I do a lot. What about you?
J: Mom, you know me. I can’t cook.
M: But can you help me wash dishes?
J: Yes, I can help you.
M: Let’s make a deal, ok?
J: What kind of deal?
M: I’m going to cook.
J: And then?
M: Then you wash the dishes after the meal.
J: That’s ok. I’ d like to help you mom.
M: You are a good boy.

。。。
```

代码：

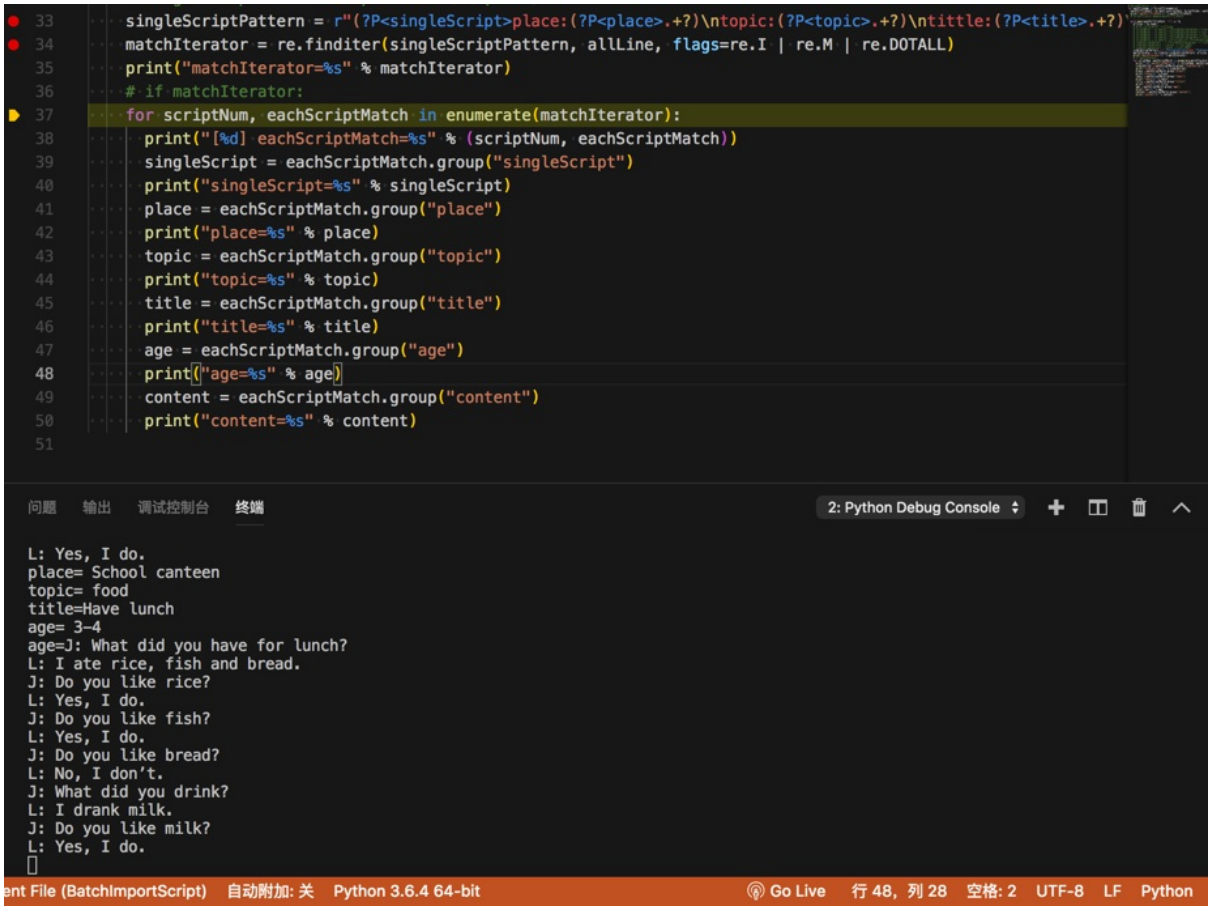
```
    singleScriptPattern = r"(?P<singleScript>place:(?P<place>.+?)\ntopic:(?P<topic>.+?)\nt
ittle:(?P<title>.+?)\nage:(?P<age>.+?)\n(?P<content>.+?))\n{2,1000}"
    matchIterator = re.finditer(singleScriptPattern, allLine, flags re.I | re.M | re.DOTALL
)
    print("matchIterator=%s" % matchIterator)
```

```
    # if matchIterator:
    for scriptNum, eachScriptMatch in enumerate(matchIterator):
      print("[%d] eachScriptMatch=%s" % (scriptNum, eachScriptMatch))
      singleScript = eachScriptMatch.group("singleScript")
      print("singleScript=%s" % singleScript)
      place = eachScriptMatch.group("place")
      print("place=%s" % place)
      topic = eachScriptMatch.group("topic")
      print("topic=%s" % topic)
      title = eachScriptMatch.group("title")
      print("title=%s" % title)
      age = eachScriptMatch.group("age")
      print("age=%s" % age)
      content = eachScriptMatch.group("content")
      print("content=%s" % content)
```

log输出：

```
matchIterator <callable_iterator object at 0x10e3f7b70>
[0] eachScriptMatch <_sre.SRE_Match object; span=(1, 309), match 'Place: School canteen\nT
opic: food\nTittle:Have l
singleScript Place: School canteen
Topic: food
Tittle:Have lunch
Age: 3-4
J: What did you have for lunch?
L: I ate rice, fish and bread.
J: Do you like rice?
L: Yes, I do.
J: Do you like fish?
L: Yes, I do.
J: Do you like bread?
L: No, I don't.
J: What did you drink?
L: I drank milk.
J: Do you like milk?
L: Yes, I do.
place= School canteen
topic= food
title Have lunch
age  3-4
age J: What did you have for lunch?
L: I ate rice, fish and bread.
J: Do you like rice?
L: Yes, I do.
J: Do you like fish?
L: Yes, I do.
J: Do you like bread?
L: No, I don't.
J: What did you drink?
L: I drank milk.
J: Do you like milk?
```

```
L: Yes, I do.
```



匹配特定模式的成语

背景需求：

> 问题描述:使用Python正则表达式，进行汉语成语的模式搜索
>
> 搜索目的地：汉语成语词典
>
> 搜索目标：几种特殊模式的成语，例如：
>
> （1）xxyy模式的，高高兴兴，快快乐乐
>
> （2）数字模式的，三心二意，一泻千里
>
> （3）动物模式的，鸡鸣狗盗，狐假虎威
>
> （4）……
>
> 先将汉语成语文件准备好，再在文件中，使用正则表达式，进行搜索。搜索结果，显示在屏幕上，同时保存到一个结果文件中。

代码：

```
# Function:
```

```
#   请教怎样使用Python正则表达式，进行汉语成语模式搜索-CSDN论坛
#   https://bbs.csdn.net/topics/396860414
# Author: Crifan
# Update: 20200619

import re

seperator = "-"

idiomStr = """高高兴兴
快快乐乐
快乐至上
欢欢喜喜
欢天喜地

一心一意
三心二意
一泻千里
三番五次
一鼓作气
以一敌万

鸡鸣狗盗
狐假虎威
兔死狐悲
狗急跳墙
"""

def printSeperatorLine(curTitle):
    print("%s %s %s" % (seperator*30, curTitle , seperator*30))

def printEachMatchGroup(someIter):
    for curIdx, eachMatch in enumerate(someIter):
        curNum = curIdx + 1
        # print("eachMatch=%s" % eachMatch)
        eachMatchWholeStr = eachMatch.group(0)
        print("[%d] %s" % (curNum, eachMatchWholeStr))

printSeperatorLine("xxyy模式成语")

# xxyyP = "(\S)\1(\S)\2"
# xxyyP = "(\S)\1(\S)\2"
# xxyyP = "(\S)\1"
# xxyyP = "(.)\1"
# xxyyP = "(.)\\1"
# xxyyP = "(?:P\S)\\1(\S)\\2"
xxyyP = "(\S)\\1(\S)\\2"
# foundAllXxyy = re.findall(xxyyP, idiomStr, re.S)
# foundAllXxyy = re.search(xxyyP, idiomStr, re.S)
# foundAllXxyyIter = re.finditer(xxyyP, idiomStr, re.S)
foundAllXxyyIter = re.finditer(xxyyP, idiomStr)
# print("foundAllXxyy=%s" % foundAllXxyy)
# for curIdx, eachMatch in enumerate(foundAllXxyyIter):
```

```
#     curNum = curIdx + 1
#     # print("eachMatch=%s" % eachMatch)
#     eachMatchWholeStr = eachMatch.group(0)
#     print("[%d] %s" % (curNum, eachMatchWholeStr)
printEachMatchGroup(foundAllXxyyIter)

# print("%s %s %s" % (seperator*30, "数字模式成语" , seperator*30))
printSeperatorLine("数字模式成语")

# refer:
#   个,十,百,千,万……兆 后面是什么?-作业-慧海网
#   https://www.ajpsp.com/zuoye/4174539
zhcnDigitList = [
    "一",
    "二",
    "三",
    "四",
    "五",
    "六",
    "七",
    "八",
    "九",
    "十",
    "百",
    "千",
    "万",
    "亿",
    "兆",
    "京",
    "垓",
    "秭",
    "穰",
    "沟",
    "涧",
    "正",
    "载",
]
zhcnDigitListGroup = "|".join(zhcnDigitList)
zhcnDigitP = "(%s)\S(%s)\S" % (zhcnDigitListGroup, zhcnDigitListGroup)
zhcnDigitIter = re.finditer(zhcnDigitP, idiomStr, re.S)
printEachMatchGroup(zhcnDigitIter)

printSeperatorLine("动物模式成语")

animalList = [
    "鸡",
    "鸭",
    "猫",
    "狗",
    "猪",
    "兔",
    "狐",
    "狼",
```

```
    "虎",
    "豹",
    "狮",
    # TODO: 添加更多常见动物
]
animalGroup = "|".join(animalList)
animalP = "(%s)\S(%s)\S" % (animalGroup, animalGroup)
animalIter = re.finditer(animalP, idiomStr, re.S)
printEachMatchGroup(animalIter)
```

输出：

```
------------------------------ xxyy模式成语 ------------------------------
[1] 高高兴兴
[2] 快快乐乐
[3] 欢欢喜喜
------------------------------ 数字模式成语 ------------------------------
[1] 一心一意
[2] 三心二意
[3] 一泻千里
[4] 三番五次
------------------------------ 动物模式成语 ------------------------------
[1] 鸡鸣狗盗
[2] 狐假虎威
[3] 兔死狐悲
```

re学习心得

学习了 `Python` 的正则 `re` 后，会对其他一些知识理解更加深入，更能触类旁通：

`Python` 的 `str` 的 `startswith`

举例：

```
imgUrl = "Books/55434/Books_55434_205865_Book_20190407102211.jpg"
isImgUrlValid = imgUrl.startswith("http")
# isImgUrlValid=False

imgUrl = "http://img.xiaohuasheng.cn/Douban/Book/s11215709.jpg"
isImgUrlValid = imgUrl.startswith("http")
# isImgUrlValid=True
```

其中的 `startswith` ：

```
isImgUrlValid = originCoverImgUrl.startswith("http")
```

等价于，可以换为：

```
isMatchHttp = re.match("^http", originCoverImgUrl)
isOriginCoverImgUrlValid = isMatchHttp
```

而之前用startswith还有个缺点：

万一想要同时判断一个url是否是http或https开头的，却要写两个startswith的判断

再去逻辑或才能得到要的结果：

```
imgUrl = "http://img.xiaohuasheng.cn/Douban/Book/s11215709.jpg"
# imgUrl = "https://img.xiaohuasheng.cn/Douban/Book/s11215709.jpg"isHttpUrl = imgUrl.start
swith("http")
isHttpsUrl = imgUrl.startswith("https")
isValidUrl = isHttpUrl or isHttpsUrl
```

而改为正则去匹配：

```
imgUrl = "http://img.xiaohuasheng.cn/Douban/Book/s11215709.jpg"
# imgUrl = "https://img.xiaohuasheng.cn/Douban/Book/s11215709.jpg"isHttpOrHttpsUrl = re.ma
tch("^https?", imgUrl)
isValidUrl = isHttpOrHttpsUrl
```

就显得更加简洁和高效。

如此举一反三，触类旁通，可以把正则在字符串的搜索和替换等领域发挥出更大的价值。

附录

下面列出相关参考资料。

参考资料

- re — Regular expression operations - Python 3
- re --- 正则表达式操作 — Python 3 文档
- re — Regular expression operations — Python 3.8.1 documentation
- re --- 正则表达式操作 — Python 3.8.1 文档
-
- 【教程】详解Python正则表达式 – 在路上
- 【教程】详解Python正则表达式之：‘.’ dot 点 匹配任意单个字符
- 【教程】详解Python正则表达式之：‘^’ Caret 脱字符/插入符 匹配字符串开始
- 【教程】详解Python正则表达式之：‘$’ dollar 美元符号 匹配字符串末尾
- 【教程】详解Python正则表达式之：‘*’ star 星号 匹配0或多个
- 【教程】详解Python正则表达式之：[] bracket 中括号 匹配某集合内的字符
- 【教程】详解Python正则表达式之：‘|’ vertical bar 竖杠
- 【教程】详解Python正则表达式之：(…) group 分组
- 【教程】详解Python正则表达式之：(?…) extension notation 扩展助记符
- 【教程】详解Python正则表达式之：(?:…) non-capturing group 非捕获组
- 【教程】详解Python正则表达式之：(?P…) named group 带命名的组
- 【教程】详解Python正则表达式之：(?P=name) match earlier named group 匹配前面已命名的组
- 【教程】详解Python正则表达式之：(?(id/name)yes-pattern|no-pattern) 条件性匹配
- 【教程】详解Python正则表达式之：(?=…) lookahead assertion 前向匹配 /前向断言
- 【教程】详解Python正则表达式之：(?<=…) positive lookbehind assertion 后向匹配 /后向断言
- 【教程】详解Python正则表达式之：(?<=…) positive lookbehind assertion 后向匹配 /后向断言
- 【教程】详解Python正则表达式之：\s 匹配任一空白字符
- 【教程】详解Python正则表达式之：re.LOCALE re.L 本地化标志
- 【教程】详解Python正则表达式之：re.UNICODE re.U 统一码标志
- 【已解决】Python中用正则re去搜索分组的集合
- 【已解决】Python 3中用正则匹配多段的脚本内容
-
- 正则表达式之——先行断言(lookahead)和后行断言(lookbehind) - 赶路人儿
- 正则表达式后行断言 • 探索 ES2018 和 ES2019 - 众成翻译
- 無聊技術研究: RegExp 應用： lookahead , lookbehind
- 正则表达式：零宽断言(lookaround) - 许炎的个人博客
- 正则表达式中 Lookaround - 知乎
- 3分钟内理解Python的re模块中match、search、findall、finditer的区别*python,match,search*不若乘风来-CSDN博客
- 【已解决】Python中用正则re去搜索分组的集合
- 【已解决】用Python的正则re去匹配特定模式的成语
-