目录

Python中正则表达式：re模块详解

- 最新版本： `v1.0`
- 更新时间： `20200210`

鸣谢

感谢我的老婆陈雪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。

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，以及给出详细的例子详尽阐述具体如何使用

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

此电子书教程的全部内容，如无特别说明，均为本人原创和整理。其中部分内容参考自网络，均已备注了出处。如有发现侵犯您版权，请通过邮箱联系我 `admin 艾特 crifan.com`，我会尽快删除。谢谢合作。

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
-

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：

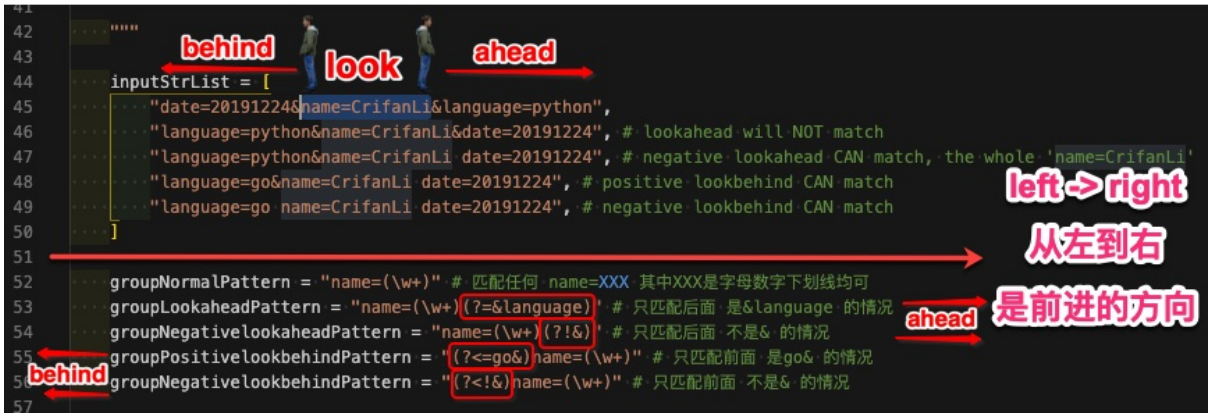
```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Author: Crifan Li
# Update: 20191223
# Function: Demo python re (?:…) non-capture usage

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

re.match

re.findall

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

TODO：

【已解决】Python中用正则re去搜索分组的集合

待整理过来

另外：好像此处属于 python中 iterator的特点 或 坑：访问一次就空了（而不一定是re的问题），待确认

re应用举例

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
-