08 学习正则表达式—基础入门

更新时间：2019-07-04 13:28:11

> 从不浪费时间的人，没有工夫抱怨时间不够。
>
> ——杰弗逊

正则表达式是一门古老的技术，同时它也是一项在各种编程语言中必备的文字分析技术。它在网络爬虫开发中发挥着极其重要的作用，常用于定义链接规则与非结构化的文字内容提取。

再论 `LinkExtractor`

在继续讲述 `LinkExtractor` 之前，先来回顾一下 `CrawlSpider` 的工作流程：

1. 当蜘蛛启动后 `CrawlSpider` 将会自动向 `start_urls` 发出请求(所以我们将 `start_urls` 内定义的 URL 称之为"种子页")
2. 当向"种子页"发出的请求返回响应对象后，`CrawlSpider` 就会开始对 `rules` 属性中定义的 `Rule` 实例进行迭代:
    - 对符合 LinkExtractor 匹配规则的链接提取出来，并发出第二次的请求。
    - 如果请求返回则会检查 `callback` 参数是否有指定回调函数，有则执行。
    - 如果 `follow` 参数被设置为 `True` 则继续对当前返回的请求执行 rules 属性内的规则继续深入下一层。

基于以上的流程应该对 `LinkExtractor` 在 `CrawlSpider` 中所起到的作用有了更进一步的认识，`LinkExtractor` 定义的是链接元素的匹配规则，在 `LinkExtractor` 众多的参数定义之中最为常用的就是 `allow` 与 `deny` 这两个效果相反的参数了，它们所定义的内容就是本节所需要讨论的重点:**正则表达式**。

先从一个最常见的例子入手来学习最简单的正则表达式。

假如在返回的网页内容中包含了以下的链接对象:

```
<a href="/news/hot-spot"> 今日热点 </a>
<a href="/news/locals"> 本地新闻 </a>
<a href="/login"> 登入 </a>
<a href="/news/worlds"> 国际新闻 </a>
```

如果我们只想要提取新闻相关的链接，也就是说要排除 `<a href="/login"> 登入 </a>` 这个元素，那么我们可以这样来写 `LinkExtractor` 的构造函数:

```
LinkExtractor(allow=r'/news/')
```

上述代码的意思就是提取 `href` 属性中以 `'/news'` 字符串开始的 `<a>` 元素。如果要用 `deny` 参数的话可以写成这样:

```
LinkExtractor(deny=r'/login')
```

这两段代码的效果是完全一样的。

再举另一个例子，如果在网页上有以下的链接元素：

```
<a href="/news/1">今日热点</a>
<a href="/news/2">本地新闻</a>
<a href="/login">登入</a>
<a href="/news/worlds">国际新闻</a>
```

如果我们只想要提取"今日热点"与"本地新闻"这种类型的链接应该如何定义 `allow` 中的表达式呢？

```
LinkExtractor(allow=r'\/news\/\d+')
```

如果没有系统化地学过正则表达式，估计代码写到这里你已经开始出现各种的疑问了，这些通配符组成的字符串到底如何理解？接下来，我们就借此机会系统化地学习一下：这套在任何语言都会出现的正则表达式背后的原理到底是什么，它是怎么来提取内容的，又应该如何去编写。

在结构化的文档中使用正则表达式的机会并不多，因为通常结构化文档都会配有一些便于操作它们对应的语言的工具包，例如在 python 中操作 xml 你会用 lxml 一样。但是，如果你用语言自身操作字符串的方式去从一个文本中查找、替换某个或某几个字符或字符串时，代码就会显非常臃肿，而且执行速度也非常低下。正则表达式就是为了解释各种通用的非结构化的文本或长符串操作而诞生的一种表达式语法，在任何语言中都会对它支持，所以说它几乎是一种在任何语言中必学的知识。

> 正则表达式（regular expression）描述了一种字符串匹配的模式（pattern），可以用来检查一个串是否含有某种子串，将匹配的子串替换，或者从某个串中取出符合某个条件的子串等。

构造正则表达式的方法和创建数学表达式的方法一样。也就是用多种元字符与运算符将小的表达式结合在一起来创建更大的表达式。正则表达式的组件可以是单个的字符、字符集合、字符范围、字符间的选择，或者所有这些组件的任意组合。

正则表达式是由普通字符（例如，字符a到z）和特殊字符（称为"元字符"）组成的文字模式。模式描述在搜索文本时要匹配一个或多个字符串。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

简单模式

我们首先要了解最简单的正则表达式。 由于正则表达式用于对字符串进行操作，因此我们将从最常见的任务开始：匹配字符。

普通字符

普通字符包括没有显式指定为元字符的所有可打印和不可打印字符。包括所有大写和小写字母，所有数字，所有标点符号和一些其他符号。

非打印字符

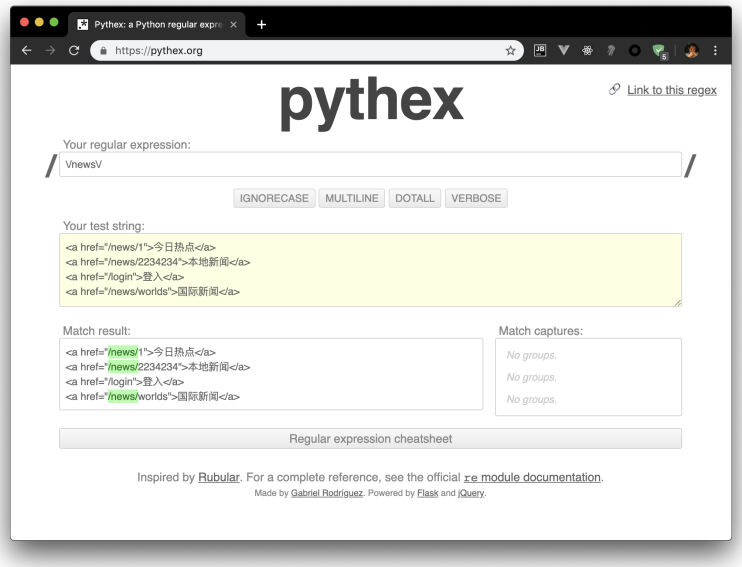
非打印字符也可以是正则表达式的组成部分。下表列出了表示非打印字符的转义序列。

字符	描述
`\cx`	匹配由 x 指明的控制字符。例如， `\cM` 匹配一个 Control-M 或回车符。x 的值必须为 `A-Z` 或 `a-z` 之一。否则，将c视为一个原义的 'c' 字符
`\f`	匹配一个换页符。等价于 `\x0c` 和 `\cL`
`\n`	匹配一个换行符。等价 于 `\x0a` 和 `\cJ`
`\r`	匹配一个回车符。等价于 `\x0d` 和 `\cM`
`\s`	匹配任何空白字符，包括空格、制表符、换页符等。等价于 `[ \f\n\r\t\v]`
`\S`	匹配任何非空白字符。等价于 `[^ \f\n\r\t\v]`
`\t`	匹配一个制表符。等价于 `\x09` 和 `\cI`
`\v`	匹配一个垂直制表符。等价于 `\x0b` 和 `\cK`

我们的第一个例子就是采用普通字符与非打印字符的匹配规则，这也是正则表达式中最简单也是最容易懂的内容：

```
allow=r'\/news\/'
```

在继续深入学习之前我们需要一个工具来测试我们正则表达式是否能像设想的那样正确地匹配，我推荐使用 https://pythex.org/ 这个在线正则表达式测试工具：



pythex 的使用非常简单，只要在 “Your regular expression” 中输入我们设计的正则表达式，在 “Your test string:” 中输入进行测试的匹配内容，它就会自动将匹配到的内容以高亮方式显示。

我们将以上的这种只采用普通字符与非打印字符所组成的正则表达式称之为**简单匹配**模式。

使用特殊字符

特殊字符就是指一些有特殊含义的字符，如前文中的 `'\/news\/\d+'` 中的 `\d`，简单地说就是表示数字的意思。

许多元字符要求我们在试图匹配它们时需要特别对待。若要匹配这些特殊字符，必须先使字符"转义"，即将反斜杠字符 `\` 放在它们前面。

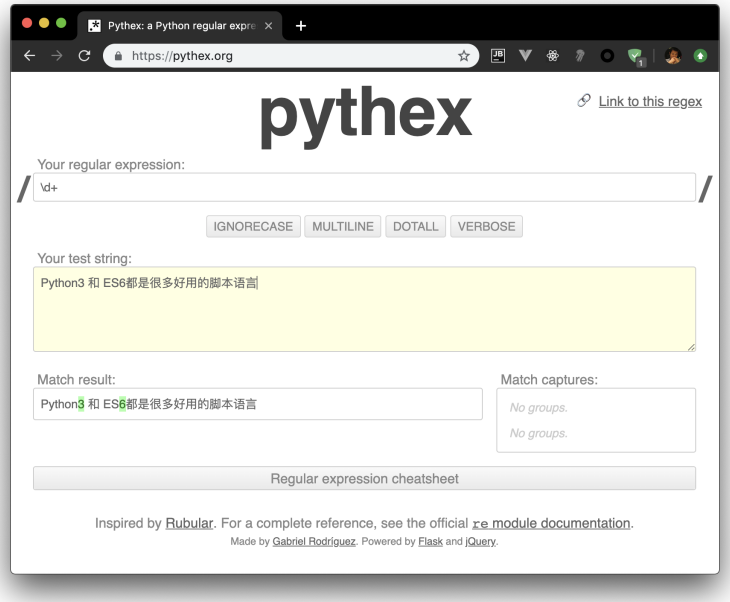
正则表达式就是一种字符学，它提供了一系列的具有特殊作用的符号来操作字符串，我尝试过硬背这个字符表，最终大都会被快速忘记。最后还是将它们按作用进行分块学习会更有效。

字母与数字的匹配符

此类符号比较容易理解，它们的作用就是用于匹配内容中的字符，单词或数字的。

特殊字符	描述
`.`	匹配除换行符之外的任何单个字符。
`\w`	匹配字母或数字或下划线或汉字
`\W`	匹配一个不可以组成单词的字符，例如 `[\W]` 匹配"KaTeX parse error: Expected 'EOF', got '中' at position 6: 5.98"中的，等价于 `[^a-zA-Z0-9]`。
`\s`	匹配任意的空白符,包括空格，制表符(Tab)，换行符，中文全角空格
`\S`	匹配任意非空白符与 `\s` 相反。
`\d`	匹配数字
`\D`	匹配一个非数字字符。
`[xyz]`	一个字符集合。匹配括号中的任意字符，包括转义序列。你可以使用破折号（`-`）来指定一个字符范围。对于点(`.`)和星号(`*`)这样的特殊符号在一个字符集中没有特殊的意义。他们不必进行转义，不过转义也是起作用的。例如，`[abcd]` 和 `[a-d]` 是一样的。他们都匹配 `"brisket"` 中的 `‘b’`,也都匹配 `“city”` 中的 `‘c’`。
`[^xyz]`	一个反向字符集。也就是说， 它匹配任何没有包含在方括号中的字符。你可以使用破折号（`-`）来指定一个字符范围。任何普通字符在这里都是起作用的。例如，`[^abc]` 和 `[^a-c]` 是一样的。他们匹配 `"brisket"` 中的 `‘r’`，也匹配 `“chop”` 中的 `‘h’`。

例如：`\d+` 匹配1个或更多连续的数字。这里的`+`是和`*`类似的重复匹配字符，不同的是`*`匹配重复任意次(可能是0次)，而`+`则匹配重复1次或更多次。



上述的列表中的字符我就不一一举例了，下面就提供一些在实际中会常用到的关于数字的正则表达式：

作用	正则表达式
数字	`^[0-9]*$`
n位的数字	`^\d{n}$`
至少n位的数字	`^\d{n,}$`
m-n位的数字	`^\d{m,n}$`
零和非零开头的数字	`^(0|[1-9][0-9]*)$`
非零开头的最多带两位小数的数字	`^([1-9][0-9]*)+(.[0-9]{1,2})?$`
带1-2位小数的正数或负数	`^(\-)?\d+(\.\d{1,2})?$`
正数、负数、和小数	`^(\-|\+)?\d+(\.\d+)?$`
有两位小数的正实数	`^[0-9]+(.[0-9]{2})?$`
有1~3位小数的正实数	`^[0-9]+(.[0-9]{1,3})?$`
非零的正整数	`^[1-9]\d*$` 或 `^([1-9][0-9]*){1,3}$` 或 `^\+?[1-9][0-9]*$`
非零的负整数	`^\-[1-9][]0-9"*$` 或 `^-[1-9]\d*$`
非负整数	`^\d+$` 或 `^[1-9]\d*\|0$`
非正整数	`^-[1-9]\d*|0$` 或 `^((-\d+)\|(0+))$`
非负浮点数	`^\d+(\.\d+)?$` 或 `^[1-9]\d*\.\d*\|0\.\d*[1-9]\d*\|0?\.0+\|0$`
非正浮点数	`^((-\d+(\.\d+)?)|(0+(\.0+)?))$` 或 `^(-([1-9]\d*\.\d*\|0\.\d*[1-9]\d*))\|0?\.0+\|0$`
正浮点数	`^[1-9]\d*\.\d*|0\.\d*[1-9]\d*$` 或 `^(([0-9]+\.[0-9]*[1-9][0-9]*)\|([0-9]*[1-9][0-9]*\.[0-9]+)\|([0-9]*[1-9][0-9]*))$`
负浮点数	`^-([1-9]\d*\.\d*|0\.\d*[1-9]\d*)$` 或 `^(-(([0-9]+\.[0-9]*[1-9][0-9]*)\|([0-9]*[1-9][0-9]*\.[0-9]+)\|([0-9]*[1-9][0-9]*)))$`
浮点数	`^(-?\d+)(\.\d+)?$` 或 `^-?([1-9]\d*\.\d*\|0\.\d*[1-9]\d*\|0?\.0+\|0)$`

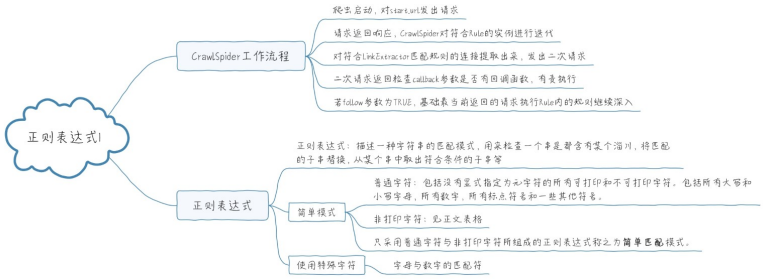
以下是一个关于字符的正则速查表:

作用	正则表达式
双字节汉字	`[^\x00-\xff]`
用户名	`^[a-z0-9_-]{3,16}$`
密码	`^[a-z0-9_-]{6,18}$`

作用	正则表达式
18位身份证号	`^(\d{6})(\d{4})(\d{2})(\d{2})(\d{3})([0-9]|X)$`
年-月-日 格式	`([0-9]{3}[1-9]|[0-9]{2}[1-9][0-9]{1}|[0-9]{1}[1-9][0-9]{2}|[1-9][0-9]{3})-(((0[13578]|1[02])-(0[1-9]|[12][0-9]|3[01]))|((0[469]|11)-(0[1-9]|[12][0-9]|30))|(02-(0[1-9]|[1][0-9]|2[0-8])))`
QQ 号	`[1-9][0-9]{4,}`
邮政编码	`[1-9]\d{5}(?!\d)`
十六进制值	`^#?([a-f0-9]{6}`
电子邮箱	`^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$` 或 `^[a-z\d]+(\.[a-z\d]+)*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$`
URL	`^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$/`
IPv4 地址	`((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?)`
IPv6地址	`^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$`
HTML 标签	`^<([a-z]+)([^<]+)*(?:>(.*)<\/\1>|\s+\/>)$`

小结

这节课带着大家回顾了一下 `LinkExtractor`，`LinkExtractor` 会把符合你定义在其中的正则表达式的链接提取出来让我们的蜘蛛来进行爬取，其中正则表达式的定义十分的关键，是你的蜘蛛能否继续进行爬取的首要条件。在这里我带大家简单认识了一下正则表达式，下节课我们来继续深入的学习正则这个无双利器。



07 常见爬网方式与网页结构分析思路

09 学习正则表达式—构造复杂模式