
1.8 正则表达式

1.8.1 正则表达式

正则表达式是用来匹配与查找字符串的,从网上爬取数据自然或多或少会用到正则表达式。Python 的正则表达式要先引入 `re` 模块,正则表达式以 `r` 引导,例如:

```
import re
reg=r"\d+"
m=re.search(reg,"abc123cd")
print(m)
```

其中 `r"\d+"` 正则表达式表示匹配连续的多个数值, `search` 是 `re` 中的函数,从 `"abc123cd"` 字符串中搜索连续的数值,得到 `"123"`,返回一个匹配对象,因此程序的结果如下:

```
<_sre.SRE_Match object; span=(3, 6), match='123'>
```

从结果看出,在指定的字符串中找到了连续的数值,它们是 `"123"`, `span(3,6)` 表示开始位置是 3,结束位置是 6,这正好是 `"123"` 在 `"abc123cd"` 中的位置。

Python 中关于正则表达式的规则比较多,下面将介绍主要的内容,详细内容读者可以参考相关资料。

1、字符 `"\d"` 匹配 0-9 之间的一个数值。

例如:

```
import re
reg=r"\d"
m=re.search(reg,"abc123cd")
print(m)
```

结果找到了第一个数值 `"1"`:

```
<_sre.SRE_Match object; span=(3, 4), match='1'>
```

2、字符 `"+"` 重复前面一个匹配字符一次或者多次。

例如:

```
import re
reg=r"b\d+"
m=re.search(reg,"a12b123c")
print(m)
```

结果找到了 `"b123"`:

```
<_sre.SRE_Match object; span=(3, 7), match='b123'>
```

注意: `r"b\d+"` 第一个字符要匹配 `"b"`,后面是连续的多个数字,因此是 `"b123"`,不是 `"a12"`。

3、字符 `"*"` 重复前面一个匹配字符零次或者多次。

`"*"` 与 `"+"` 类似,但有区别,例如:

```
import re
reg=r"ab+"
m=re.search(reg,"acabc")
print(m)
reg=r"ab*"
```

```
m=re.search(reg,"acabc")
```

```
print(m)
```

结果:

```
<_sre.SRE_Match object; span=(2, 4), match='ab'>
```

```
<_sre.SRE_Match object; span=(0, 1), match='a'>
```

由此可见 `r"ab+"`匹配的是"ab", 但是 `r"ab*"`匹配的是"a", 因为 `r"ab*"`表示"b"可以重复零次, 但是"+"却要求"b"重复一次以上。

4、字符"?"重复前面一个匹配字符零次或者一次。

例如:

```
import re
```

```
reg=r"ab?"
```

```
m=re.search(reg,"abbcabc")
```

```
print(m)
```

结果:

```
<_sre.SRE_Match object; span=(0, 2), match='ab'>
```

匹配结果是"ab", 其中 b 重复一次。

5、字符"."代表任何一个字符, 但是没有特别声明时不代表字符"\n"。

例如:

```
import re
```

```
s="xaxby"
```

```
m=re.search(r"a.b",s)
```

```
print(m)
```

结果"."代表了字符"x"

```
<_sre.SRE_Match object; span=(1, 4), match='axb'>
```

6、"|"代表把左右分成两个部分。

例如:

```
import re
```

```
s="xaabababy"
```

```
m=re.search(r"ab|ba",s)
```

```
print(m)
```

结果匹配"ab"或者"ba"都可以:

```
<_sre.SRE_Match object; span=(2, 4), match='ab'>
```

7、特殊字符使用反斜线"\ "引导, 例如"\r"、"\n"、"\t"、"\"分别表示回车、换行、制表符号与反斜线自己本身。

例如:

```
import re
```

```
reg=r"a\nb?"
```

```
m=re.search(reg,"ca\nbcabc")
```

```
print(m)
```

结果匹配"a\nb":

```
<_sre.SRE_Match object; span=(1, 4), match='a\nb'>
```

8、字符"\b"表示单词结尾，单词结尾包括各种空白字符或者字符串结尾。

例如：

```
import re
reg=r"car\b"
m=re.search(reg,"The car is black")
print(m)
结果匹配"car"，因为"car"后面是以个空格：
<_sre.SRE_Match object; span=(4, 7), match='car'>
```

9、"[]"中的字符是任选择一个，如果字符是 ASCII 码中连续的一组，那么可以使用"-"符号连接，例如[0-9]表示 0-9 的其中一个数字，[A-Z]表示 A-Z 的其中一个大写字符，[0-9A-Z]表示 0-9 的其中一个数字或者是 A-Z 的其中一个大写字符。

例如：

```
import re
reg=r"x[0-9]y"
m=re.search(reg,"xyx2y")
print(m)
结果匹配"x2y":
<_sre.SRE_Match object; span=(2, 5), match='x2y'>
```

10、"^"出现在[]的第一个字符位置，就代表取反，例如[^ab0-9]表示不是 a、b，也不是 0-9 的数字。

例如：

```
import re
reg=r"x[^ab0-9]y"
m=re.search(reg,"xayx2yxcy")
print(m)
结果匹配"xcy":
<_sre.SRE_Match object; span=(6, 9), match='xcy'>
```

11、"\s"匹配任何空白字符，等价于"[\r\n\x20\t\f\v]"。

例如：

```
import re
s="1a ba\tbxy"
m=re.search(r"a\s",s)
print(m)
结果匹配"a b":
<_sre.SRE_Match object; span=(1, 4), match='a b'>
```

12、"\w"匹配包括下划线子内的单词字符，等价于"[a-zA-Z0-9_]"。

例如：

```
import re
```

```
reg=r"\w+"
m=re.search(reg,"Python is easy")
print(m)
结果匹配"Python":
<_sre.SRE_Match object; span=(0, 6), match='Python'>
```

13、"^"匹配字符串的开头位置。

例如：

```
import re
reg=r"^ab"
m=re.search(reg,"cabcab")
print(m)
结果：
None
```

没有匹配到任何字符，因为"cabcab"中虽然有"ab"，但不是"ab"开头。

14、"\$"字符匹配字符串的结尾位置。

例如：

```
import re
reg=r"ab$"
m=re.search(reg,"abcb")
print(m)
匹配结果是最后一个"ab"，而不是第一个"ab":
<_sre.SRE_Match object; span=(3, 5), match='ab'>
```

15、使用括号(...)可以把(...)看成一个整体，经常与"+"、"*"、"?"的连续使用，对(...)部分进行重复。

例如：

```
import re
reg=r"(ab)+"
m=re.search(reg,"ababcb")
print(m)
结果匹配"abab"，"+"对"ab"进行了重复：
<_sre.SRE_Match object; span=(0, 4), match='abab'>
```

1.8.2 查找匹配字符串

正则表达式库 `re` 的 `search` 函数使用正则表达式对要匹配的字符串进行匹配，如果匹配不成功就返回 `None`，如果匹配成功就返回一个匹配对象。匹配对象调用 `start()` 函数得到匹配字符串的开始位置，匹配对象调用 `end()` 函数得到匹配字符串的结束位置。`search` 虽然只返回第一次匹配的结果，但是我们只要连续使用 `search` 函数就可以找到字符串中全部匹配的字符串。

例：匹配找出英文句子中所有单词

我们可以使用正则表达式 `r"[A-Za-z]+\b"` 匹配单词，它表示匹配由大小写字母组成的连续多个字符，一般是一个单词，之后 `"\b"` 表示单词结尾。

```
import re
s="I am testing search function"
reg=r"[A-Za-z]+\b"
m=re.search(reg,s)
while m!=None:
    start=m.start()
    end=m.end()
    print(s[start:end])
    s=s[end:]
    m=re.search(reg,s)
```

结果:

```
I
am
testing
search
function
```

程序开始匹配到一个单词后 `m.start()`,`m.end()`就是单词的起始位置, `s[start:end]`为截取的单词, 之后程序再次匹配字符串 `s=s[end:]`, 即字符串的后半段, 一直到匹配完毕为止就找出每个单词。