第11章 正则表达式

《Python程序开发案例教程（第2版）》

学习目标/Target

- 熟悉元字符，能够归纳元字符的功能以及基本用法
- 熟悉预定义字符集，能够归纳预定义字符集的功能
- 了解re模块，能够说出re模块中常用函数或方法的作用
- 掌握预编译的方式，能够通过complie()函数将正则表达式预编译为正则对象
- 掌握匹配与搜索的方式，能够通过match()与search()函数实现匹配与搜索的功能

学习目标/Target

- 掌握匹配对象，能够通过匹配对象的方法获取匹配结果中的各项数据
- 掌握全文匹配的方式，能够通过findall()与finditer()函数实现全文匹配的功能
- 掌握检索替换的方式，能够通过sub()、subn()函数实现检索替换的功能
- 掌握文本分割的方式，能够通过split()函数实现文本分割的功能
- 熟悉贪婪匹配，能够区分贪婪匹配和非贪婪匹配的不同

章节概述/ Summary

网站开发人员需要对用户在注册页面提交的信息进行验证，包括手机号、用户名、邮箱等。由于这些信息遵循复杂的规则，仅使用条件语句判断将增加工作量。然而，正则表达式解决了这个问题。正则表达式是一种描述字符串结构的语法规则，它在字符串的查找、匹配、替换等方面非常强大，并且被许多编程语言广泛支持，包括Python。本章将介绍如何在Python中使用正则表达式。

目录/Contents

目录/Contents

目录/Contents

11.1 基础知识

11.1.1 元字符

- 熟悉元字符，能够归纳元字符的功能以及基本用法

11.1.1 元字符

元字符

在正则表达式中，元字符指具有特殊含义的专用字符，可以用来规定其前导字符在目标对象中出现的模式，前导字符是位于元字符之前的字符。正则表达式中的元字符一般由特殊字符和符号组成。

11.1.1 元字符

元字符

常用的元字符如表所示

元字符	功能说明
.	点字符，匹配任何一个字符，除了换行符
^	脱字符，匹配字符串的开头
$	美元符，匹配字符串的结尾
|	连接符，用于连接多个模式，匹配任意一个模式
[]	字符组，匹配其中出现的任意一个字符
-	连字符，表示范围，比如1-5等价于1、2、3、4、5
?	匹配其前导字符0次或1次
*	匹配其前导字符0次或多次
+	匹配其前导字符1次或多次
{n}/{m,n}	匹配其前导字符n次/匹配其前导字符m~n次
()	在模式中划分出子模式，并保存子模式的匹配结果

11.1.1 元字符

1.点字符

（1）点字符

J.m	Jam、Jom、J#m、Jim、J2m

（2）脱字符“^”和美元符“$”

^cat	category

（3）连接符

cat|dog	匹配cat或dog

（4）字符组

arg[vs]	argv或args

11.1.1 元字符

1.点字符

（5）连字符“-

[0-9]	0~9之间的一个数字

（6）元字符“?”

June?	Jun或June

（7）元字符“*”、“+”以及{n}/{m,n}

ht*p	hp、htp、http、htttp

（8）元字符

Jan(uary)?	Jan或January

11.1.2 预定义字符集

- 熟悉预定义字符集，能够归纳预定义字符集的功能

11.1.2 预定义字符集

异常的类型

正则表达式中预定义了一些字符集，字符集能以简洁的方式表示一些由元字符和普通字符表示的匹配规则。常见的预定义字符集如表所示。

元字符	功能说明
\w	匹配下画线或任何字母与数字
\s	匹配任意的空白字符，等价于[<空格>\t\r\n\f\v]
\d	匹配任意数字，等价于[0-9]
\b	匹配单词的边界
\W	与\w相反，匹配特殊字符
\S	与\s相反，匹配任意非空白字符的字符，等价于[^\s]
\D	与\d相反，匹配任意非数字的字符，等价于[^\d]
\B	与\b相反，匹配不出现在单词边界的元素
\A	仅匹配字符串的开头，等价于^
\Z	仅匹配字符串的结尾，等价于$

11.2 re模块

11.2 re模块

- 了解re模块，能够说出re模块中常用函数或方法的作用

11.2 re模块

re模块

Python中的re模块是正则表达式模块，该模块提供了文本匹配、文本检索替换、文本分割等功能。re模块中常用的函数及方法如表所示。

函数/方法	说明
compile()	对正则表达式进行预编译，并返回一个Pattern对象
match()	从字符串的开头位置开始匹配，若匹配成功返回Match对象，否则返回None
search()	从字符串的任意位置开始匹配，若匹配成功返回Match对象，否则返回None
split()	根据正则表达式将目标字符串进行分割，并返回一个分割后的列表
findall()	在目标字符串中从左至右查找与正则表达式模式匹配的所有非重叠子串，将返回一个包含这些子串的列表

11.2 re模块

re模块

Python中的re模块是正则表达式模块，该模块提供了文本匹配、文本检索替换、文本分割等功能。re模块中常用的函数及方法如表所示。

函数/方法	说明
finditer()	功能与findall()相同，但返回结果是迭代器对象
sub()	搜索目标字符串中与正则对象匹配的子串，使用指定字符串替换，并返回替换后的对象
subn()	搜索目标字符串中与正则对象匹配的子串，使用指定字符串替换，返回替换后的对象和替换次数
group()	返回全部Pattern对象
groups()	返回一个包含全部匹配的子组的元组，若匹配失败，则返回空元组

11.3 预编译

11.3 预编译

- 掌握预编译的方式，能够通过complie()函数将正则表达式预编译为正则对象

11.3 预编译

预编译

如果需要对一个正则表达式重复使用，那么可以使用compile()函数对其进行预编译，以节省每次编译正则表达式的开销。complie()函数的语法格式如下：

语法格式

```
compile(pattern, flags=0)
```

- pattern：表示要编译的正则表达式。
- flags：用于指定正则匹配的模式。

预编译

参数flags用于指定正则匹配的模式，该参数的常用取值如表所示。

取值	说明
re.I	忽略大小写的模式，使匹配对大小写不敏感
re.L	用于本地化的识别匹配，它会根据当前区域设置影响一些预定义字符集的解释，这些预定义字符集包括\w、\W、\b、\B等
re.M	多行模式，使得^和$匹配每一行的开头和结尾，而不仅仅是整个字符串的开头和结尾
re.S	使.匹配所有字符，包括换行符
re.U	根据Unicode字符集解析字符
re.A	根据ASCII字符集解析字符
re.X	使用更灵活的格式书写正则表达式，正则表达式可以是多行、忽略空白字符、加入注释等，使用户更容易理解

11.3 预编译

预编译

complie()函数在编译成功后会返回一个Pattern对象。例如，使用complie()函数将正则表达式预编译为Pattern对象，具体代码如下：

示例

```
import re
regex_obj = re.compile(r'\d')
words = 'Today is July 26, 2023.'
print(regex_obj.findall(words))
```

11.3 预编译

预编译

如果想要匹配这一组字符串中所有的英文字母，可以在使用compile()函数预编译正则对象时设置flags参数，将正则表达式的匹配模式设置为re.I，忽略英文字母的大小写，示例代码如下：

示例

```
import re
# 将正则表达式[a-z]+预编译为正则对象，同时指定匹配模式为re.I
regex_one = re.compile(r'[a-z]+', re.I)
words = 'Today is March 28, 2019.'
print(regex_one.findall(words))
```

11.4 匹配与搜索

11.4.1 使用match()函数进行匹配

- 掌握匹配与搜索的方式，能够通过match()函数实现匹配功能

11.4.1 使用match()函数进行匹配

使用match()函数进行匹配

match()函数用于从字符串的开头开始匹配正则表达式模式，它会尝试将模式与字符串的开头部分进行匹配，如果匹配成功，match()函数返回一个Match对象，也就是匹配对象；如果匹配失败，它返回 None。

语法格式

```
match(pattern, string, flags=0)
```

- pattern：表示正则表达式，它的取值既可以是正则对象，也可以是包含正则表达式的字符串。
- string：表示待匹配的目标字符串。
- flags：表示使用的匹配模式，默认值为0，说明不使用任何匹配模式。

11.4.1 使用match()函数进行匹配

使用match()函数进行匹配

使用match()函数对指定的字符串进行匹配与搜索，示例代码如下：

示例

```
import re
date_one = "Today is July 26, 2023."
date_two = "26 July 2023"
print(re.match(r"\d", date_one))
print(re.match(r"\d", date_two))
```

11.4.2 使用search()函数进行匹配

- 掌握匹配与搜索的方式，能够通过search()函数实现s
  搜索功能

11.4.2 使用search()函数进行匹配

使用search()函数进行匹配

search()函数会查找整个字符串，直到找到第一个满足模式的子字符串。如果找到匹配项，search()函数返回一个匹配对象；如果没有找到匹配项，它会返回 None。

语法格式

```
search(pattern, string, flags=0)
```

- pattern：表示正则表达式，它的取值既可以是正则对象，也可以是包含正则表达式的字符串。
- string：表示待匹配的目标字符串。
- flags：表示使用的匹配模式，默认值为0，说明不使用任何匹配模式。

11.4.2 使用search()函数进行匹配

使用search()函数进行匹配

使用search()函数对指定的字符串进行匹配与搜索，示例代码如下：

示例

```
import re
info_one = "I was born in 2000."
info_two = "20000505"
print(re.search(r"\d", info_one))
print(re.search(r"\D", info_two))
```

11.4.3 实例1：手机号运营商

- 根据任务分析实现实例1：手机号运营商

11.4.3 实例1：手机号运营商

一个手机号码由11位数字组成，前3位数字表示网络识别号，第4~7位数字表示地区编号，第8-11位数字表示用户编号。因此，我们可以通过手机号前3位数字的网络识别号辨别手机号所属运营商。

运营商	号码段
中国移动	134、135、136、137、138、139、147、148、150、151、152、157、158、159、165、178、182、183、184、187、188、198
中国联通	130、131、132、140、145、146、155、156、166、185、186、175、176
中国电信	133、149、153、180、181、189、177、173、174、191、199

本实例要求编写程序，实现判断输入的手机号码是否合法以及判断其所属的运营商的功能。

11.4.3 实例1：手机号运营商

实现思路

① 使用列表保存价格信息。

② 定义空列表用于保存用户选购商品的价格。

③ 接收输入的最大价格和最小价格。

④ 从价格列表中获取每个商品价格。

⑤ 判断商品价格区间。

⑥ 将商品价格进行排序。

11.4.3 实例1：手机号运营商

实现步骤

① 在Chapter10项目中创建01_belong.py文件。

② 在01_belong.py中编写代码。

③ 运行01_belonge.py文件。

11.5 匹配对象