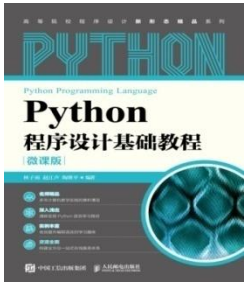
《Python程序设计基础教程（微课版）》

http://dblab.xmu.edu.cn/post/python



第13章 正则表达式

林子雨 博士/副教授

厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn

主页：http://dblab.xmu.edu.cn/linziyu

主讲教师

2017年度厦门大学奖教金获得者
2020年度厦门大学奖教金获得者

主讲教师：厦门大学 林子雨 博士/副教授
中国高校首个“**数字教师**”提出者和建设者
2009年7月从事教师职业以来
累计**免费**网络发布超过1500万字高价值教学和科研资料
网络浏览量超过1500万次

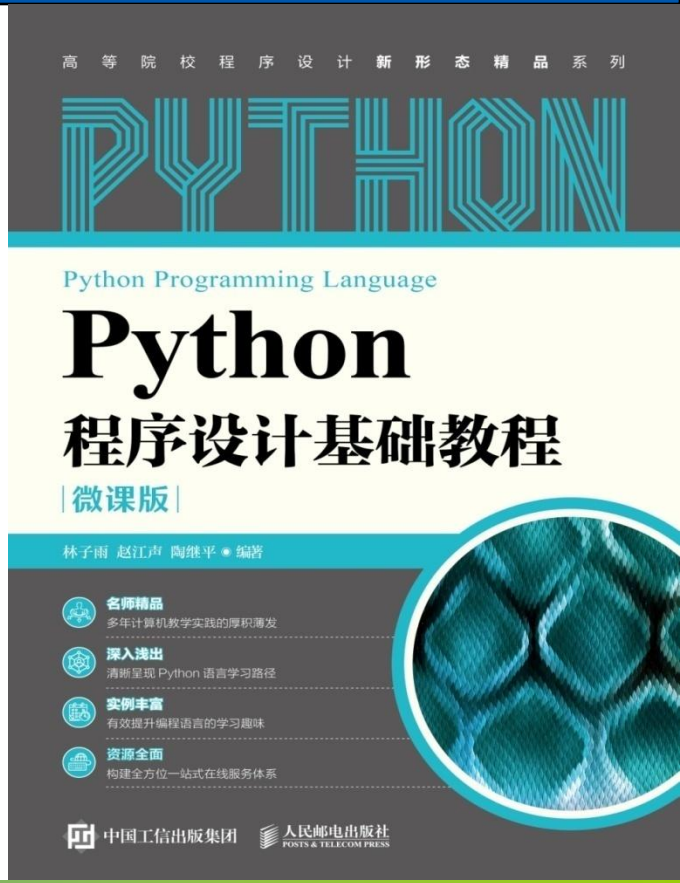
提纲

13.1 正则表达式概述
13.2 正则表达式基本规则
13.3 正则表达式的组
13.4 正则表达式的函数

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python



13.1正则表达式概述

正则表达式（Regular Expression）也可以称为“规则表达式”，在各大计算机语言中，一般记为“regex”，也经常简写为“re”。一个正则表达式应当由模式字符串和被搜索字符串共同组成。下面是一个使用正则表达式的具体实例，该实例使用了正则表达式来快速匹配短信文本中的验证码：

```
>>> import re
```

在使用正则表达式时，需要先引用Python的内置模块re。之后，使用re的相关函数进行操作。为了简洁起见，在本章接下来的所有实例中，就不再出现re模块的引用代码“import re”，都默认视为已经在IDLE中引用了re模块。

13.1正则表达式概述

```
>>> regex = re.compile(r".*验证码.*(\d{6,8}).*")
```

上面这行代码re.complie是将模式字符串编译为正则表达式对象。编译函数的第一个参数是用以描述搜索模式的原始字符串，也就是“.*验证码.*(\d{6,8}).*”，这个字符串应当称为“正则表达式中的模式字符串”，这里都将其简写为“模式字符串”。不过，在实际的沟通和交流中，为了减少学习和沟通成本，可以约定俗成地把“正则表达式中的模式字符串”直接称为“正则表达式”。

模式字符串规定了在查询中按照什么样的模式进行匹配。比如，本例中模式字符串“.*验证码.*(\d{6,8}).*”，实际要表达的含义是“查询原字符串，找出出现在‘验证码’三个字之后的、一个6位到8位长的数字”。

13.1正则表达式概述

```
>>> source = "此验证码只用于登录你的微信或更换绑定，验证码提供给他人将导致微信被盗。123456（微信验证码）。再次提醒，请勿转发"
>>> match = regex.search(source)
```

在上面的代码中，调用了正则表达式对象的search函数。search函数用来搜索某个字符串是否与规定的模式匹配，被搜索的字符串称为“原字符串”。在上面这个微信验证码短信的例子中，search函数可以找到一组数字满足模式字符串的要求，即“123456”。

```
>>> print(match.group(1))
123456
```

13.1正则表达式概述

（第1行代码）>>> import re
（第2行代码） >>> regex = re.compile(r".*验证码.*(\d{6,8}).*")
（第3行代码） >>> source = "此验证码只用于登录你的微信或更换绑定，验证码提供给他人将导致微信被盗。123456（微信验证码）。再次提醒，请勿转发"
（第4行代码） >>> match = regex.search(source)

众所周知，每一个表达式都包括运算符、操作数和计算结果。因此，在正则表达式中，search函数就可以理解为一个二元操作符，它拥有两个操作数，即“模式字符串”和“原字符串”，计算结果是一个名为“match”的对象。也就是说，在上面这个实例中，第2、3、4行代码合在一起才能称为一个正则表达式。

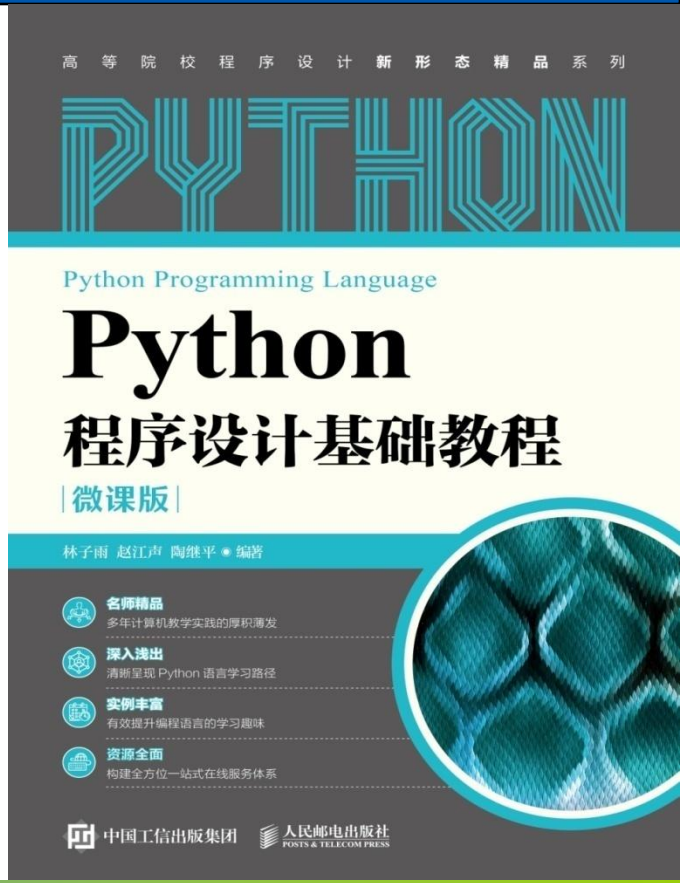
13.2 正则表达式的基本规则

13.2.1 正则表达式中的字符串类型
13.2.2 模式字符串中的普通字符
13.2.3 模式字符串中的转义字符
13.2.4 模式字符串的其它特殊字符

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python



13.2.1 正则表达式中的字符串类型

一个正则表达式其实包含了两个部分，分别是“模式字符串”和“原字符串”。根据第5章的内容我们知道，在Python中字符串的表现形式包括Unicode字符串（str）和字节串（bytes）。在使用正则表达式的过程中，模式字符串和原字符串的表现形式必须相同，即不能使用Unicode字符串匹配一个字节串，反之亦然。

同样，在使用正则表达式进行替换操作时，将要替换的字符串也必须和模式字符串、原字符串的类型保持一致。总之，在使用正则表达式时，涉及到的所有字符串都必须是同一类型。

13.2.2 模式字符串中的普通字符

正则表达式里可以包含所有普通或者特殊字符。普通字符的含义就是匹配它们本身，模式字符串中，除了如图13-1所示的字符以外，其它字符都是普通字符。

. ^ $ * + ? { } [ ] \ | ( )

图13-1 Python正则表达式中的特殊字符

13.2.2 模式字符串中的普通字符

比如，模式字符串“Python”在没有任何特殊设置的情况下，就只能匹配大小写与之一致的字符串。在使用过程中，如果只使用普通字符作为模式字符串进行匹配，那么目的一般是为了验证原字符串是否包含某些内容，那么，在这种情况下，作用就和字符串的find函数没有什么区别。具体实例如下：

```
>>> print(re.search("Python", "这是一本介绍Python的书"))
<re.Match object; span=(6, 12), match='Python'>
```

可以看出，在使用正则表达式匹配时，返回的匹配结果不仅表达了是否包含模式字符串，更包含了模式字符串所在的位置信息。

13.2.2 模式字符串中的普通字符

在模式字符串只出现一次的情况下，和find函数没有什么区别。因为，find函数也可以通过返回的索引值直接确定模式字符字串出现的起止位置。

下面是一个实例：

```
>>> print("这是一本介绍Python的书".find("Python"))
6
```

不过，如果模式字符串所对应的匹配出现了多次，那么search函数只返回第一个匹配的位置和内容。可是在实际使用过程中，有些场景（比如解析一个HTML），必须要得到所有的匹配。所以，就需要用到finditer函数，它可以返回一个迭代器，通过循环语句就可以取出所有的匹配，具体实例如下：

```
>>> for m in re.finditer("Python", "这是一本用Python写的、介绍Python的书"):
        print(m)
<re.Match object; span=(5, 11), match='Python'>
<re.Match object; span=(16, 22), match='Python'>
```

13.2.3 模式字符串中的转义字符

在介绍普通字符时，一共有列出14个特殊字符（见图13-1），其中，最重要的莫过于转义字符“\”，这个字符可以将特殊字符的含义消除，和Python字符串中的转义字符含义类似。比如，在正则表达式中，如果需要匹配一个字符串是否是一个存在于C盘的文件，就要写如下代码：

```
>>> re.search("C:\\\\.*", r"C:\msdia80.dll")
<re.Match object; span=(0, 14), match='C:\\msdia80.dll'>
```

可以看到，由于路径分隔符“\”在正则表达式中是一个特殊字符，所以在写的时候要使用转义字符消去其特殊含义，于是要在模式字符串里写成“\\”。但是，反斜杠“\”在Python的字符串里也是特殊字符，其含义同样是转义。这样，“\\”在经过Python字符串解释的时候，就变成了一个反斜杠。因此，为了表示两个反斜杠“\\”，就要写成四个反斜杠“\\\\”。

13.2.3 模式字符串中的转义字符

再比如，如果需要匹配“\]”这样的字符，由于这两个字符都是特殊字符，所以，在书写模式字符串时都要进行转义，应当写成“\\\]”。这种写法即不美观也不易读，所以，在这里应当使用第5章中学习到的原始字符串来书写，简单易读也好懂，具体实例如下：

```
>>> re.search(r"\\\]", r"在正则表达式中，|\][都是特殊字符")
<re.Match object; span=(9, 11), match='\\]'>
>>> re.search("\\\\\\]", r"在正则表达式中，|\][都是特殊字符")
>>> <re.Match object; span=(9, 11), match='\\]'>
```

13.2.4 模式字符串的其它特殊字符

•在匹配字符串时，显然不仅要匹配模式字符串中包含的字符，还可能要匹配一些不确定的字符，比如匹配手机号码时，只知道手机号是一个11位的数字，但不确定这11位分别是什么数字，所以这时就需要使用通用字符。

•所谓“通用字符”指的是可以表示一系列字符的字符。在正则表达式的默认配置中，模式字符串使用特殊字符“.”来代表除了换行符以外的一个任何字符，实例如下：

```
>>> re.search("133.......", "您好，您拨打的电话已经更换为：133****2457。现在这个号码已经不再使用了。")
<re.Match object; span=(15, 26), match='133****2457‘>
```

13.2.4 模式字符串的其它特殊字符

•可以看到，这个实例的目标是匹配一个133开头的手机号。众所周知，手机号是一个11位的数字。

•也就是说，133后面应当跟着8位数，因此，这个实例中就书写了8个可以匹配任意字符的“.”。

•于是，系统也为用户匹配出了出现在短信里的文本“133****2457”。但是，匹配结果的手机号“133****2457”出现了“*”，并不是数字，而通用字符“.”也将它们匹配出来了。

•如果我们不想出现包含“*”的匹配结果，就需要使用指定匹配的通用字符“[]”。

13.2.4 模式字符串的其它特殊字符

使用方括号包裹起来的内容应当视为一个字符，即意味着，出现在这个位置的字符必须是方括号中出现字符中的其中一个。比如模式字符串“[Pp]ython”的匹配结果可以是“Python”，也可以是“python”，但是绝对不可能是“Ppython”，具体实例如下：

```
>>> for m in re.finditer("[Pp]ython", "这是一本用Python写的介绍Python的书，不过你不可以把pyThon写成“Ppython”。"):
        print(m)
<re.Match object; span=(5, 11), match='Python'>
<re.Match object; span=(15, 21), match='Python'>
<re.Match object; span=(41, 47), match='python'>
```

在上面这个实例中，原字符串一共有4个python，但是，正则表达式的默认设置是区分大小写的，所以第三个“pyThon”显然无法匹配。

13.2.4 模式字符串的其它特殊字符

那么，如何书写模式字符串才能匹配一个数字呢？显然，一个数字只有0-9这十种情况，所以可以直接写成“[0123456789]”这样的形式。不过，这种写法显然太麻烦了，实际上可以使用“[0-9]”来表示。同理，“[4-6]”表示这个位置应当出现数字4、5或者6。

具体实例如下：

```
>>> re.search("[4-6]", "您说的这个号码是什么？我没有听清。当中的4位是什么？")
<re.Match object; span=(20, 21), match='4'>
```

在这个实例中，“[4-6]”就匹配了原字符串中唯一出现的数字4。同理，字母也可以使用这种写法，“[a-z]”表示的是所有小写字母，“[A-Z]”表示的是所有大写字母。

13.2.4 模式字符串的其它特殊字符

当然，这种写法还可以连用，比如，“[a-gx-zR-U]”就应当看成是“a-g”、“x-z”和“R-U”三段的组合。使用正常语言表达就是的a、b、c、d、e、f、g、x、y、z、R、S、T、U中的任何一个字母。具体实例如下：

```
>>> for m in re.finditer("[a-gx-zR-U]", "这是一本用Python写的介绍Python的书，不过你不可以把pyThon写成“Ppython”。"):
        print(m)
<re.Match object; span=(6, 7), match='y'>
<re.Match object; span=(16, 17), match='y'>
<re.Match object; span=(32, 33), match='y'>
<re.Match object; span=(33, 34), match='T'>
<re.Match object; span=(42, 43), match='y'>
```

13.2.4 模式字符串的其它特殊字符

回到本小节最开始的例子，要表示一个133开头的手机号，需要使用8个数字，那么应当写成“133[0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9]”。可以看出，这里重复写了8次[0-9]，这种写法显然有些麻烦，如果匹配的是具有20位的手机SIM卡的编号，这么写就不合适了。所以，在正则表达式中，可以使用花括号“{}”来表示一个字符出现了多少次，具体实例如下：

```
>>> re.search(r"\*{4}", r"不好意思先生，我说的是133****2457中间四位数是**5*啊")
<re.Match object; span=(14, 18), match='****'>
```

需要注意的是，花括号只能匹配出现在它前面的1个字符。在上面这个实例中，花括号前面看似有两个字符，即“*”。但是，由于“*”是特殊字符，所以转义字符串“*”实际只代表着一个字符，也就是星号“*”。因此，这里花括号中出现的数字4，指的就是前面的星号出现了4次。既然已经指定了次数，其它任何次数不正确的星号都不能匹配，这也是为什么后一个“**5*”无法被匹配。

13.2.4 模式字符串的其它特殊字符

不过，在一些情况下，字符出现次数其实是不固定的，比如，出现4次到6次都满足要求，那么就可以在花括号中给出这些匹配次数，具体实例如下：

```
>>> re.search("[cC][oO]{4,6}[lL]", "这真的是太cool了。VERY COOOOOL!VERY VERY COOOOOOOOOOL!!!")
<re.Match object; span=(16, 23), match='COOOOOL'>
```

在匹配时，系统会尽可能多地匹配重复部分，比如“ba{4,6}”在匹配原字符串“baaaaaa”时，会将整个字符串匹配到，而不是只匹配其中4个a。当然，逗号前后的数字都可以不写。如果写成“ba{,6}”，则说明这里的a应当出现0至6次。如果写成“ba{4,}”，则说明这里的a应当出现4次以上，直到无限次。如果逗号前后数字都不写，就是匹配任意次。

13.2.4 模式字符串的其它特殊字符

具体实例如下：

```
>>> re.search("\*{2,}", r"您那里信号是不是有问题啊，我这听到的都是*。一堆的*******")
<re.Match object; span=(25, 32), match='*******'>
```

在上面这个实例中，由于匹配会在重复次数中选择尽可能多的那一个，所以，第一个星号由于只出现一次，不能匹配，而后面连续出现的七个星号，就全部匹配成功了。这里需要注意，如果不指定具体的重复次数，逗号是一定不能省略的。“{2,}”指的是2次或者2次以上。“{2}”指的是只出现2次，这是完全不一样的含义。

13.2.4 模式字符串的其它特殊字符

在书写模式字符串时，有一些重复次数是非常常见的。比如任意次数，再比如1次或者1次以上。所以，可以使用特殊字符“*”代表重复任意次，也就是等价于“{,}”。可以使用特殊字符“+”表示出现1次或者一次以上，也就是等价于“{1,}”。可以使用特殊字符“?”表示出现0次或者1次，也就是等价于“{0,1}”。具体实例如下：

```
>>> for m in re.finditer("colou?r", "This colour is very good. I like this color."):
        print(m)
<re.Match object; span=(5, 11), match='colour'>
<re.Match object; span=(38, 43), match='color'>
```

13.2.4 模式字符串的其它特殊字符

再次强调，“*”，“+”和“?”都是用来简化书写的，在使用时它们也只能修饰之前的一个字符。比如，在下面的实例中，“信号+”就只能匹配“信号号”和“信号”，但不能匹配“信号信号”：

```
>>> for m in re.finditer("信号+", "没有啊，我这的信号信号非常...嗯？信号号~~"):
        print(m)
<re.Match object; span=(7, 9), match='信号'>
<re.Match object; span=(9, 11), match='信号'>
<re.Match object; span=(18, 21), match='信号号'>
```

13.2.4 模式字符串的其它特殊字符

在使用过程中，还有可能需要匹配所有不是指定字符的字符，这时，就可以使用特殊字符“^”对方括号中的内容“取非”，表达的含义是，除了这些字符以外，其它的都可以匹配。

13.2.4 模式字符串的其它特殊字符

```
>>> for m in re.finditer("[^*]", "喂*喂**，你**在**，我**不知***~~~"):
        print(m)
<re.Match object; span=(0, 1), match='喂'>
<re.Match object; span=(2, 3), match='喂'>
<re.Match object; span=(5, 6), match='，'>
<re.Match object; span=(6, 7), match='你'>
<re.Match object; span=(9, 10), match='在'>
<re.Match object; span=(12, 13), match='，'>
<re.Match object; span=(13, 14), match='我'>
<re.Match object; span=(16, 17), match='不'>
<re.Match object; span=(17, 18), match='知'>
<re.Match object; span=(21, 22), match='~'>
<re.Match object; span=(22, 23), match='~'>
<re.Match object; span=(23, 24), match='~'>
```

可以看到，这里就匹配了所有不是星号的字符。

13.2.4 模式字符串的其它特殊字符

这里需要注意的是，由于方括号内只表示一个字符，所以用来表示重复的特殊字符“*”、“+”和“？”以及之后要介绍的“|”都会失去含义，这时，它们就不再需要转义了。当然，写了转义也不会出问题。

13.2.4 模式字符串的其它特殊字符

```
>>> for m in re.finditer(r"[^\*]", r"喂*喂\\**，你**在**，我**不知***~~~"):
        print(m)
<re.Match object; span=(0, 1), match='喂'>
<re.Match object; span=(2, 3), match='喂'>
<re.Match object; span=(3, 4), match='\\'>
<re.Match object; span=(4, 5), match='\\'>
<re.Match object; span=(7, 8), match='，'>
<re.Match object; span=(8, 9), match='你'>
<re.Match object; span=(11, 12), match='在'>
<re.Match object; span=(14, 15), match='，'>
<re.Match object; span=(15, 16), match='我'>
<re.Match object; span=(18, 19), match='不'>
<re.Match object; span=(19, 20), match='知'>
<re.Match object; span=(23, 24), match='~'>
<re.Match object; span=(24, 25), match='~'>
<re.Match object; span=(25, 26), match='~'>
```

13.2.4 模式字符串的其它特殊字符

•由于记忆哪些需要转义哪些不需要转义会非常麻烦，所以，为了保证简化内容，建议初学者在使用时不管如何对于特殊字符全部使用转义。同时，上面这个实例中，其实想要的结果是去掉文本中的所有星号，这时其实不应该使用匹配，而应该使用之后要介绍的模式替换。

•不过，并不是所有匹配都可以按一个一个字符来的。比如，匹配一个网址是否是教育机构或者非营利性组织的。众所周知，判断网址的类型可以用顶级域名，在这里，我们假设所有网站都遵守这个规则，也就是说，教育机构的顶级域名一定是“.edu”，非营利性组织的顶级域名一定是“.org”。那么，如何判断字符串“你可以在http://python.org找到Python的详细资料，也可以访问http://www.xmu.edu.cn关注厦门大学的最新信息。”中包含多少个教育机构和非营利性组织的网址呢？这时就要用到特殊符号“()”。

13.2.4 模式字符串的其它特殊字符

具体实例如下：

```
>>> for m in re.finditer("(.edu)|(.org)", "你可以在http://python.org找到Python的详细资料，也可以访问http://www.xmu.edu.cn关注厦门大学的最新信息"):
        print(m)
<re.Match object; span=(17, 21), match='.org'>
<re.Match object; span=(54, 58), match='.edu'>
```

从执行结果可以看出，会找到所有.org和.edu的匹配

13.2.4 模式字符串的其它特殊字符

```
>>> re.search("(.edu)|(.org)", "你可以在http://python.org找到Python的详细资料，也可以访问http://www.xmu.edu.cn关注厦门大学的最新信息")
<re.Match object; span=(17, 21), match='.org'>
```

从执行结果可以看出，只要找到一个.org结果以后，就不会继续寻找.edu。

```
>>> re.search("(.edu)|(.org)", "你可以访问http://www.xmu.edu.cn关注厦门大学的最新信息，也可以在http://python.org找到Python的详细资料")
<re.Match object; span=(19, 23), match='.edu'>
```

从执行结果可以看出，只要找到一个.edu结果以后，就不会继续寻找.org。

13.2.4 模式字符串的其它特殊字符

•在括号中的正则表达式被称为一个组，关于组的更多内容将在13.3节介绍，在这里只需要知道每个组都会被当成一个整体即可。

•可以看到，匹配结果里将“.edu”和“.org”当成了一个整体进行匹配。另外，上例在模式字符串中使用了一个新的特殊字符“|”，这个特殊字符表示“或”，它连接的是两个正则表达式，匹配文字只要满足“或”连接的第一个正则表达式，就视为匹配成功，就不再继续匹配另一个正则表达式。

13.2.4 模式字符串的其它特殊字符

•具体实例如下：

```
>>> re.search("你好|是我", "你是我啊")
<re.Match object; span=(1, 3), match='是我'>
```

“|”连接的是两个正则表达式，所以这里第一个re.search语句的真实含义是匹配“你好”或者“是我”，那么在原字符串“你是我啊”里面，只存在“是我”。

13.2.4 模式字符串的其它特殊字符

```
>>> re.search("你(好|是)我", "你是我啊")
<re.Match object; span=(0, 3), match='你是我'>
```

使用圆括号限定了取“或”的范围，“|”连接的两个正则表达式是“好”和“是”，因此，这个re.search语句的真实含义是匹配一个“你”字，一个“好”或者“是”字，以及一个“我”字，等同于“你[好是]我”。

13.2.4 模式字符串的其它特殊字符

```
>>> re.search("我|我们", "这是我们的歌")
<re.Match object; span=(2, 3), match='我'>
```

最后一个例子中，“|”连接的是“我”和“我们”。由于在匹配时，只要满足任何一个就不在匹配，所以匹配出“我”字之后，这个正则就不会再匹配后面的部分。因此，可以得出一个结论，这个正则表达式永远也不可能匹配出“我们”二字。

13.2.4 模式字符串的其它特殊字符

•现在回到前面手机号匹配的场景，要匹配一个133开头的11位手机号，就可以写作“133[0-9]{8}”。

•当然，实际使用过程中不可能只匹配133一个号段，需要匹配所有可用的号段。通过查询各大运营商使用号码段，可以知道 13开头的和18开头的，所有号段都能用，也就是130-139和180-189都有手机号存在。15和19开头的，除了154和194都可以用。17开头的，从170-178都可以用。14开头的，除了142和143其它都可以用。16开头的，只有2、5、6、7可以用。

13.2.4 模式字符串的其它特殊字符

所以，这个正则表达式可以写成如下形式：

1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}

由于15开头的号段分成了两个部分，书写时绝对不可以把“5[0-35-9]”书写成“5[0-3,5-9]”。表面上看，这里加了个逗号使得表达式更加易读了。但是，逗号在正则表达式里不是特殊字符，于是它就被视为了一个普通字符，因此，就会导致如下的严重错误：

```
>>> re.search("1(3[0-9]|4[01456879]|5[0-3,5-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "15,11122233")
<re.Match object; span=(0, 11), match='15,11122233'>
```

13.2.4 模式字符串的其它特殊字符

在正则表达式中，如果每个数字都要写成[0-9]其实也很麻烦。所以，有一些特殊的记法用来简化它，表13-1给出了一些常用的简化记法。

表13-1 常用的简化记法

记法	匹配内容
\d	匹配一个数字，可以是半角的，也可以是全角的。如果在ASCII码中，就等同于[0-9]
\D	除了数字其它都可以匹配，相当于[^0-9]
\w	匹配一个单词字符，在ASCII码中相当于[0-9A-Za-z_]。在Unicode中，则包括所有构成单词的字符，比如希腊字符、中文字符等
\W	匹配所有\w不能匹配的字符，在ASCII码中相当于[^0-9A-Za-z_]
\s	所有的空白符，在ASCII码中就是[\t\n\r\f\v]。在Unicode中，所有显示为空白的字符，比如全角的空格都可以匹配
\S	匹配所有\s不能匹配的字符，在ASCII码中相当于[^ \t\n\r\f\v]

13.2.4 模式字符串的其它特殊字符

在使用过程中，尤其是在表单验证中，不建议使用“\d”来代替“[0-9]”，因为这会使得原本不应该被匹配的内容变得可以匹配，具体实例如下：

```
>>> re.search("1(3[0-9]|4[01456879]|5[0-3,5-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "155２２２２３３３３")
<re.Match object; span=(0, 11), match='155２２２２３３３３'>
```

一般而言，短信服务提供商会要求输入的短信字符是半角字符，但是，上例中使用了“\d”，使得全角的字符也被匹配成功，这很显然是错误的。所以，表13-1中的记法如果无法正确区分包含哪些内容，请尽量不要使用除了“\w”以外的其它简要记法。之所以可以使用“\w”，是因为这个符号通常用来匹配文字，它一般不会产生什么问题。

13.2.4 模式字符串的其它特殊字符

•最后要介绍的特殊字符是出现在方括号外的“^”和写在任意位置的“$”。这两个比较简单，前者用以匹配字符串的开头，如果正则表达式的模式设置为多行模式，它还能匹配每一行的开头。

•后者用以匹配字符串的结束，在多行模式下，还可以匹配换行符。比如上例所示的正则表达式，其实它可以匹配“+8615522223333”。

13.2.4 模式字符串的其它特殊字符

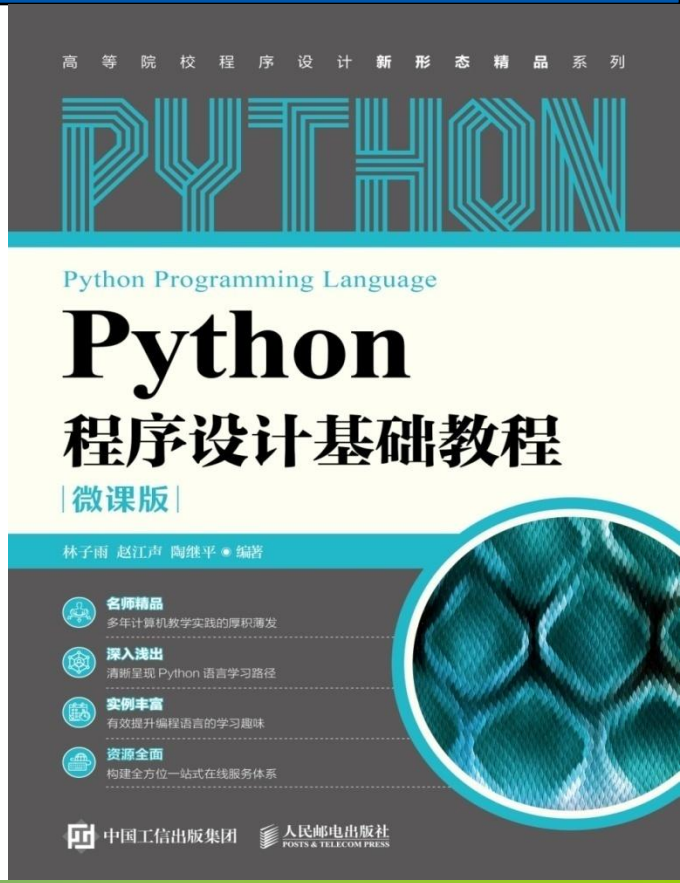
但是，如果加上“^”就不行了，因为这个符号代表当前位置必须是字符串的开头，实例如下：

```
>>> re.search("1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "+8615522223333")
<re.Match object; span=(3, 14), match='15522223333'>
>>> print(re.search("^1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "+8615522223333"))
None
```

特别注意一点，“^”只有出现在方括号外才有此含义。如果“^”是出现在方括号里的第一个字符，那么这个符号就表示取反。

13.3 正则表达式的组

13.3.1 捕获组
13.3.2 条件匹配
13.3.3 断言组



本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python

13.3.1 捕获组

•在使用正则表达式时，往往不仅需要匹配原字符串是否符合某规则，更重要的是从原字符串提取出想要的信息。比如本章开始的例子，就不仅需要验证一条短信息是否包含验证码，还需要将其中的验证码提取出来。这时就需要使用匹配结果中的“捕获组（Capture Group）”，也可以简称为“捕获”。

•在使用正则表达式的分组功能时，每一个括号内的内容就是一个捕获组，其中包含的内容就是捕获的内容。每个组的左括号出现的顺序就是捕获组的编号，编号从1开始。第0个捕获组则是被匹配的字符串本身，即相当于每个模式字符串都在最外层加一个括号。

13.3.1 捕获组

下面是一个具体实例：

```
>>> import re;
>>> regex = re.compile(r".*验证码.*(?P<code>\d{6,8}).*");
>>> source = "此验证码只用于登陆你的微信或更换绑定，验证码提供给他人将导致微信被盗。123456（微信验证码）。再次提醒，请勿转发";
>>> match = regex.search(source) ;
>>> print(match.group(0)) ;
此验证码只用于登陆你的微信或更换绑定，验证码提供给他人将导致微信被盗。123456（微信验证码）。再次提醒，请勿转发
>>> print(match.group(1)) ;
123456
>>> print(match.group("code"));
123456
```

13.3.1 捕获组

•在这个实例中，捕获组1的正则表达式是“(?P<code>\d{6,8})”，其中，“(?P<组名>)”是给捕获组命名的语法，其含义是，将该组命名为指定的名称，这种分组也称为“具名组”。

•在检查捕获时，group函数的参数就可以使用此名称来取代对应的顺序。

13.3.1 捕获组

在使用group函数时，还可以一次取多个捕获，这将会返回一个元组。下面的实例就从URL中获取了相关的信息：

```
>>> match = re.search("(((ht|f)tp(s?))\://){1}(www.|[a-zA-Z].)([a-zA-Z0-9\-\.]+)\.(com|edu|gov|mil|net|org|biz|info|name|museum|us|ca|uk)(\:[0-9]+)*", "https://www.xmu.edu.cn")
>>> match.group(1,2,3,4,5,6,7,8)
('https://', 'https', 'ht', 's', 'www.', 'xmu', 'edu', None)
```

我们把模式字符串分拆成以下几个部分就比较容易理解了：

第1行：(((ht|f)tp(s?))\://){1}
第2行：(www\.|[a-zA-Z]+\.)
第3行：([a-zA-Z0-9\-\.]+)
第4行：\.
第5行：(com|edu|gov|mil|net|org|biz|info|name|museum|us|ca|uk)
第6行：(\:[0-9]+)*

13.3.1 捕获组

•事实上，在这个捕获组里，有太多的信息是没有用的。那么如果需要返回指定的信息，除了可以使用“(?P<组名>)”的方式命名，还可以使用非捕获组“(?:)”的方式。

•非捕获组的意思是，在使用group函数或者groups函数时，不会将这些内容视为一个捕获。比如上例中，“(ht|f)”如果不使用非捕获组，就会出现元组中的第三个值“ht”。如果使用了非捕获组“(?:ht|f)”，那么，就不会有“ht”这个值了，如下例所示：

```
>>> match = re.search("((?:(?:ht|f)tp(?:s?))\://){1}(www\.|[a-zA-Z]+\.)([a-zA-Z0-9\-\.]+)\.(com|edu|gov|mil|net|org|biz|info|name|museum|us|ca|uk)(\:[0-9]+)*", "https://www.xmu.edu.cn")
>>> match.groups()
('https://', 'www.', 'xmu', 'edu', None)
```

13.3.1 捕获组

我们把模式字符串分解开就比较容易理解了：

第1行：((?:(?:ht|f)tp(?:s?))\://){1}

第2行：(www\.|[a-zA-Z]+\.)

第3行：([a-zA-Z0-9\-\.]+)

第4行：\.

第5行：(com|edu|gov|mil|net|org|biz|info|name|museum|us|ca|uk)

第6行：(\:[0-9]+)*

13.3.1 捕获组

如果不使用非捕获组，我们获得的是如下匹配结果：
('https://', 'https', 'ht', 's', 'www.', 'xmu', 'edu', None)
如果使用非捕获组，则获得如下匹配结果：
('https://', 'www.', 'xmu', 'edu', None)

从结果中可以看出，使用非捕获组以后'https', 'ht', 's'就没有出现在结果中了，这是因为，(?:s?)里面使用了非捕获组，所以不会捕获's'，(?:ht|f)里面使用了非捕获组，所以不会捕获'ht'，(?:(?:ht|f)tp(?:s?))的第1个问号和冒号表示非捕获组，因此就不会捕获'https'。

13.3.1 捕获组

•对于非捕获组而言，它仅是不产生捕获，对于圆括号原本的组的含义和用法保持不变。

•显而易见的是，在正则表达式中需要分组的内容肯定远多于需要捕获的内容，所以在正常的代码开发过程中，一般都会使用具名组的方式书写。

13.3.2 条件匹配

正则表达式中的组除了包含之前介绍的功能以外，还可以用来限定当前位置的内容是否满足指定条件，这种匹配方法称为“条件匹配”。条件匹配有两种情况：

（1）根据前文是否出现某些字符，判断当前位置应当匹配什么值。使用的语法是“(?(前文组的id/前文组的名字)存在匹配|不存在匹配)”；

（2）在指定位置匹配前文已经出现过的内容，使用的语法是“(?P=前文组的名字)”。

13.3.2 条件匹配

比如在电子邮件应用中，许多电子邮箱的名字都会使用尖括号包裹，那么在匹配时，如果需求是电子邮箱包括尖括号，则需要匹配出一对尖括号和邮件地址，如果电子邮箱不包括尖括号，则匹配出邮件地址，如果只有一边有尖括号，则不匹配，那么就要写成如下的形式：

13.3.2 条件匹配

```
>>> re.search(r"(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$)",
"<some_email@domain.com>")
<re.Match object; span=(0, 23), match='<some_email@domain.com>'>
>>> re.search(r"(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$)",
"some_email@domain.com")
<re.Match object; span=(0, 21), match='some_email@domain.com'>
>>> print(re.search(r"(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$)",
"some_email@domain.com>"))
None
```

这里使用的语法是“(?(id/组名)存在匹配|不存在匹配)”，也就是正则表达式"(<)?(\w+@\w+(?:\.\w+)+)(?(1)>|$)"的最后一部分“(?(1)>|$)”，它表示的是判断ID为1的组是否存在，如果存在，则匹配“>”，如果不存在则匹配行尾“$”。可以看到，匹配结果里只包含两种情况，要么是包含尖括号的电子邮箱地址，要么是不包含尖括号的电子邮箱地址。

13.3.2 条件匹配

在正则表达式的组中，不仅可以使用是否存在作为条件进行匹配，还可以使用前文已经捕获的内容进行匹配。比如，匹配一个字符串的文本内容是否包含另一个字符串，就可以使用如下的正则表达式进行匹配：

13.3.2 条件匹配

```
>>> re.search("(?P<q>['\"]).*(?P=q)", "'python'")
<re.Match object; span=(0, 8), match="'python'">
>>> re.search("(?P<q>['\"]).*(?P=q)", "'python\"")
>>> re.search("(?P<q>['\"]).*(?P=q)", "\"python\"")
<re.Match object; span=(0, 8), match='“python”'>
```

在上面这个实例中，先匹配了一个具名组q，其内容是单引号或者双引号，然后在后面使用“(?P=组名)”的语法匹配前面已经出现过的组中的内容，所以，如果前面使用了单引号，那么后面只有出现单引号才能匹配成功，如果前面使用了双引号，那么后面只有出现双引号才能匹配成功。这种匹配方式经常用于匹配HTML的语法是否正确，因为在HTML中，属性名需要使用引号包裹，但是和Python一样，单引号或者双引号只要配对使用就可以。如果需要检查引号是否配对，就必须使用条件匹配的方式。注意，与指定条件匹配不同，这里不能使用编号，只能使用具名组的方式进行匹配。

13.3.3 断言组

断言（Assert）是一种判断，比如“我说你的枪里没有子弹”，这就是一个断言。在正则表达式中，根据判断文本的方向，可以分为“先行”和“后行”。根据判断是存在还是不存在，可以分为“肯定”和“否定”（也可以使用“正向”和“负向”）。所以，正则表达式中的断言一共可以分为4种：先行肯定断言、先行否定断言、后行肯定断言以及后行否定断言。接下来，将一一举例介绍。

所谓“后行断言”，指的是被断言的内容出现在断言之后的一种情况。比如，现在银行发送的短信验证码往往都有一个短信编号，那么在匹配短信编号时，就需要使用后行断言。

13.3.3 断言组

例如，收到一条银行发来的短信内容为“【某某银行】您正在登录手机银行，短信验证码：330033（短信编号：135361），请勿泄露短信验证码”，现在需要匹配其中的短信验证码。

观察短信编号和验证码不难发现，两者都是一个6位的数字。如果使用前文所述的“验证码.*(?P<code>\d{6,8})”就会有如下的结果：

```
>>> re.search(r"验证码.*(?P<code>\d{6,8})", "【某某银行】您正在登录手机银行，短信验证码：330033（短信编号：135361），请勿泄露短信验证码")
<re.Match object; span=(18, 40), match='验证码：330033（短信编号：135361'>
```

13.3.3 断言组

实际上，该实例需要匹配的是出现在“验证码”三个字后面的数字。在这里“出现在‘验证码’三个字后面”就是一个断言。断言本身不会产生匹配，也不消耗正在匹配的字符，它只做一个判断。在该实例中，需要匹配的内容在断言的后面，所以这种断言是“后行断言”。再者，需要匹配的是包含“验证码”字样，所以这种断言是肯定断言。综上所述，该实例所述的就是“后行肯定断言”，其书写方法如下所示：

```
>>> re.search(r"(?<=验证码.)(?P<code>\d{6,8})", "【某某银行】您正在登录手机银行，短信验证码：330033（短信编号：135361），请勿泄露短信验证码")
<re.Match object; span=(22, 28), match='330033'>
```

13.3.3 断言组

后行肯定断言在书写时，被断言的内容写在后面。在上面这个实例中，“*(?<=验证码.)*(?P<code>\d{6,8})”中的斜体部分的就是该断言。前导符“?<=”里，“<”代表着这个断言是一个后行断言，为方便记忆，可以视为一个箭头，指的是被断言内容的方向。“=”代表着这个断言是一个肯定断言。断言的内容是“验证码.”这4个字符，其中，“.”按照正则表达式中的理解，解释为一个任意的字符。

后行肯定断言指的是只有在出现断言内容之后，才能出现需要匹配的内容。比如，在上面这个实例中，符合“(?P<code>\d{6,8})”的字符串及其前四个字符分别是“验证码：330033”和“信编号：135361”。在这里，只有前一个符合断言内容。所以，返回的结果只有短信验证码。

13.3.3 断言组

需要注意的是，在Python中后行断言的内容必须是一个定长的字符串，比如下面的断言就不正确：

```
>>> re.search(r"(?<=(验证码)|(短信验证码))(?P<code>\d{6,8}", "【某某银行】您正在登录手机银行，短信验证码：330033（短信编号：135361），请勿泄露短信验证码")
Traceback (most recent call last):
    File "<pyshell#96>", line 1, in <module>
re.search(r"(?<=(验证码)|(短信验证码))(?P<code>\d{6,8}", "【某某银行】您正在登录手机银行，短信验证码：330033（短信编号：135361），请勿泄露短信验证码")
…
```

上面的断言之所以错误，是因为断言的内容是3个或者5个字符，这是不符合要求的。这里需要说明的是，不是所有语言都要求后行断言的内容是定长字符串，比如JavaScript。

13.3.3 断言组

再来看另外一个实例。假设原字符串是“购买时间截至1月10日，下载时间截至1月20日，招标开始时间1月30日”，现在需要匹配除下载时间以外的其它时间。

这里，显然需要断言的是除下载以外的时间，所以这里是一个否定断言。要匹配的内容出现在断言“除下载以外”的后面，所以这是一个后行断言。后行否定断言的书写方法如下：

13.3.3 断言组

```
>>> for m in re.finditer(r"((?<!下载)时间)(截至)?(?P<time>(\d+月\d+日))", "
购买时间截至1月10日，下载时间截至1月20日，招标开始时间1月30日"):
        print(m.group("time"))
1月10日
1月30日
```

在该实例中，“(***(?<!下载)***时间)(截至)?(?P<time>(\d+月\d+日))”中的斜体部分是断言。该断言的前导符为“?<!”，其中，“<”代表这个断言是一个后行断言，“!”是取非的符号，代表着这个断言是一个否定断言。所以，该断言是一个后行否定断言，它意味着在出现断言之后，才可以出现需要匹配的内容“时间”。所以，符合条件的就只有第一个时间和第三个时间。

13.3.3 断言组

在学习了后行断言之后，再来看先行断言。所谓“先行断言”就是指先出现需要匹配的内容，再出现需要断言的内容。与后行断言不同，几乎所有常见的语言里，先行断言都不要求断言内容必须为固定长度，可以是任意长度。

例如，现有一个密码要求如下：

（1）可以使用任意字符；

（2）长度必须在8-12位之间；

（3）必须拥有至少一个小写字母；

（4）必须拥有至少一个大写字母；

（5）必须拥有至少一个数字。

现在需要编写一个正则表达式，验证某个输入的字符串是否为符合以上条件的密码。

在这里，可以将这个问题转换为：在字符串的开始之后的内容，一定有出现过一个大写字母、一个小写字母和一个数字，并且有8-12位的限制。

13.3.3 断言组

•现在一步步书写该问题对应的模式字符串。
•首先是最简单的，字符串长度为8-12位的任意字符，可以写为“^.{8-12}$”，这里必须要使用“^”和“$”来限制字符串的开始和结束，否则任何一个长度大于8的字符串都可以匹配。
•再之后，为“^”增加先行断言“必须拥有至少一个小写字母”，由于这里要求的是“必须拥有”，所以书写为肯定断言，使用先导符是“=”，即书写为“^***(?=.*[a-z])***.{8,12}$”。
•同样地，斜体部分是断言，观察其前导符“?=”，与后行断言相比，不需要加入“<”。这种断言表示的是，先出现被匹配内容，再出现断言。在表达式“^(?=.*[a-z]).{8,12}$”中，被匹配内容指的是字符串开始符“^”。断言内容指的是“.*[a-z]”，也就是有一个小写字母，至于出现在哪里无所谓。这里不能书写为“[a-z].*”，因为这样写表达的含义是以小写字母开头。

13.3.3 断言组

那么，还有两个条件该如何书写呢？可以发现，这里的条件之间都是“与”关系，所以，直接在断言之后再书写断言就可以了。所以整体的正则表达式如下：

```
>>> re.search(r"^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9]).{8,12}$", "Dd54x76y")
<re.Match object; span=(0, 8), match='Dd54x76y'>
```

这个实例中，断言之间的关系都是“与”关系。如果该实例的后三个条件是“或”关系，那么就使用一个断言“(?=.*([a-z]|[0-9][A-Z]))”就能解决，不需要书写三遍。

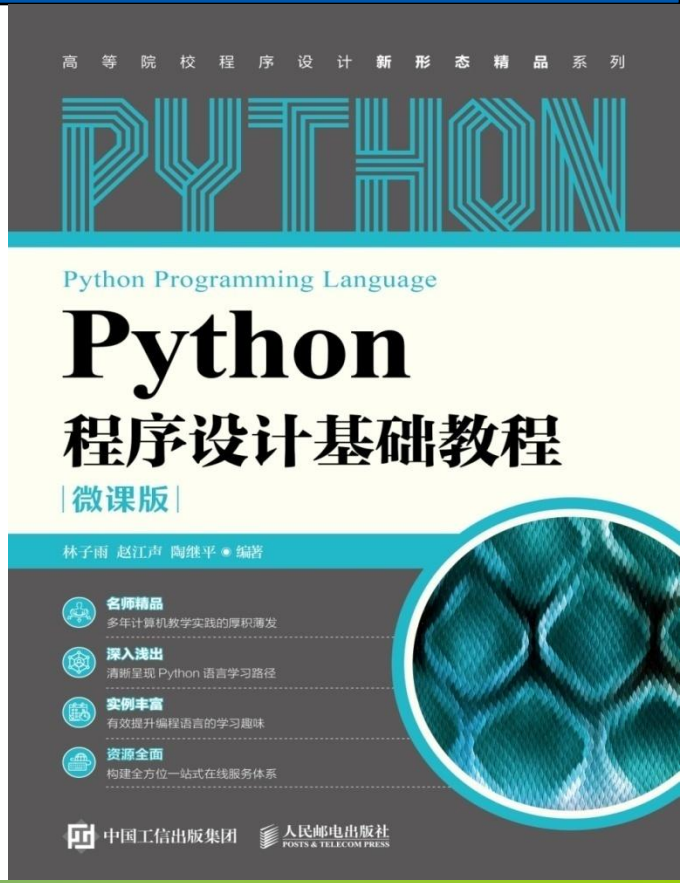
13.3.3 断言组

到目前为止，我们已经介绍了三种断言，按照上述规律，想必不难总结出，还没有介绍的先行否定断言的前导符是“?!”。这里给出一个具体实例，假设原字符串是“Python是1门非常热门的语言，2020年，热门指数已经达到了9%。在其之后的是热门指数为7%的C语言”，现在需要匹配其中不是“热门指数”的数字。可以看到，所谓“热门指数”在字符串中表现为一个百分数。所以，需要匹配的内容相当于被匹配的数字后面没有百分号。因此应当书写为如下形式：

```
>>> for m in re.finditer(r"\d+(?!%)", "Python是1门非常热门的语言，2020年，热门指数已经达到了9%。在其之后的是热门指数为7%的C语言"):
        print(m)
<re.Match object; span=(7, 8), match='1'>
<re.Match object; span=(17, 21), match='2020'>
```

13.4 正则表达式的函数

13.4.1 正则表达式的使用方法
13.4.2 正则对象和匹配规则
13.4.3 正则对象的常用成员函数
13.4.4 正则表达式里的match对象



本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《 Python程序设计基础教程（微课版） 》教材官方网站：
http://dblab.xmu.edu.cn/post/python

13.4.1 正则表达式的使用方法

正则表达式的使用方法主要有两种。第一种是将模式字符串编译为正则表达式对象（也称“正则对象”），然后每一次都调用正则对象的成员函数对原字符串进行匹配，具体实例如下：

```
>>> import re
>>> pattern = re.compile(r'1[34578]\d{9}')
>>> string = '收件人是林书凡，联系电话为：13612345678'
>>> result = pattern.sub("1**********", string)
>>> print(result)
收件人是林书凡，联系电话为：1**********
```

13.4.1 正则表达式的使用方法

另一种方法是使用re模块里的相关方法直接操作模式字符串和原字符串。
具体实例如下：

```
>>> import re
>>> pattern = r'1[34578]\d{9}'
>>> string = '收件人是林书凡，联系电话为：13612345678'
>>> result = re.sub(pattern,'1**********',string)
>>> print(result)
收件人是林书凡，联系电话为：1**********
```

13.4.1 正则表达式的使用方法

•对于这两种不同的使用方法而言，如果一个正则表达式需要在程序里大量地重复使用，那么使用编译为正则对象的方法比较合适，因为这样处理效率是最高的。反之，如果一个正则表达式只是在某些指定场合使用，那么直接调用re里的方法就比较合适，因为这样代码量小，也简单易读。

•但是，无论使用哪种方法，它们的方法名和调用参数都差不多。只需要记忆函数名，具体的参数表在使用过程中查看IDE的智能提示就可以了。

13.4.2 正则对象和匹配规则

•在使用正则表达式的第一种使用方法中，首先需要使用re的compile方法将模式字符串编译为正则对象。

•该方法的原型是“re.compile(pattern, flags=0)”，其中，第二个可选参数flags是一个枚举对象，指定了正则表达式所使用的默认规则，比如，在默认规则下，正则表达式是区分大小写的。修改和配置flags参数就可以改变这些规则。

13.4.2 正则对象和匹配规则

常用的规则有：

（1）按ASCII码匹配，使用re.A或者re.ASCII。这使得\w、\W、\d、\D、\s、\S、\b和\B都只匹配ASCII码，具体的匹配内容已经在表13-1中有给出。这种匹配规则可以很好地处理之前所说的半角或者全角问题。使用re.A后，就可以看到会有如下所示的结果：

```
>>> print(re.search("1(3[0-9]|4[01456879]|5[0-3,5-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "155２ ２ ２ ２ ３ ３ ３ ３", re.A))
None
>>> re.search("1(3[0-9]|4[01456879]|5[0-3,5-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}", "15522223333", re.A)
<re.Match object; span=(0, 11), match='15522223333'>
```

13.4.2 正则对象和匹配规则

（2）按Unicode匹配。在新版Python的正则表达式中，本身就是按Unicode字符匹配的，所以不需要加任何标记符。由于之前版本并不是这样的，所以保留了re.U使得早期版本的Python代码在新版编译器中不会失效，但是，在新版代码中re.U基本没有任何意义。

（3）忽略大小写。在默认的匹配过程中，正则表达式是区分大写小的，如果需要忽略，就可以使用re.I或者re. IGNORECASE。在ASCII码的情况下，相当于[a-z]可以匹配[A-Z]，反之亦然。

（4）多行匹配。在正常情况下，无论字符串有多少个换行符，正则表达式都将它们视为一行进行处理，其中，换行符都只是视为一个普通的字符。所以，行首标记符“^”只能匹配为字符串索引位置0，行尾标记符“$”只能匹配字符串索引位置-1。当中所有的换行符都是普通字符。

13.4.2 正则对象和匹配规则

•不过，使用re.M或者re.MULTILINE的情况下，行首标记符除了匹配字符串的开头外，还可以匹配每一个换行符的后一个字符，也就是显示出来的字符串的每一行的开头。

•而行尾标记符除了匹配字符串的结尾以外，还可以匹配换行符的前一个字符，也就是显示出来的字符串的每一行的结尾。这是一个非常常用的匹配方式，请务必牢记。

13.4.2 正则对象和匹配规则

具体实例如下：

```
>>> mobile = re.compile("^1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}$", re.A)
>>> print(mobile.search('''13355552222
13344447777
13322224444
'''))
None
>>> mobile = re.compile("^1(3[0-9]|4[01456879]|5[0-35-9]|6[2567]|7[0-8]|8[0-9]|9[0-35-9])\d{8}$", re.M)
>>> mobile.search('''13355552222
13344447777
13322224444
''')
<re.Match object; span=(0, 11), match='13355552222'>
```

13.4.2 正则对象和匹配规则

（5）在默认的匹配规则中，特殊字符“.”只能匹配除换行符外的其它所有字符。如果在特殊情况下，希望匹配包括换行符在内的所有字符，就需要re.S或者re.DOTALL。

以上就是常用的正则表达式匹配规则，当然还有re.DEBUG，re.X和re.L等不常用的匹配规则。这些规则的具体使用方法以及生效情况，请参阅对应版本的Python帮助文档。

13.4.3 正则对象的常用成员函数

1.search函数

在正则表达式的使用过程中，如果仅需要搜索原字符串中是否出现符合条件的文本，那么这时就应该使用search函数。假设正则对象为pattern，则调用search函数的语法如下：

pattern.search(string[, pos[, endpos]])

其中，第一个参数指的是原字符串，第二个参数指的是字符串的开始位置，第三个参数指的是字符串的结束位置。其中，参数pos如果不指定，相当于搜索原字符串。参数endpos如果指定了，那么即是搜索从pos至endpos-1位置的字符，并且如果endpos<pos，那么由于字符串切片结果为空，故不会有任何结果返回。由于这两个参数的作用是对原字符串进行切片，所以可以省略，即如果rx是一个编译后的正则对象，rx.search(string,0,50) 等价于rx.search(string[0:50])。

13.4.3 正则对象的常用成员函数

search函数的返回值有两种可能性，如果可以找到对应匹配，那么则返回对应结果。如果不能找到对应匹配，那么返回None。具体实例如下：

```
>>> pattern = re.compile("d")
>>> pattern.search("dog")
<re.Match object; span=(0, 1), match='d'>
>>> print(pattern.search("dog", 1))
None
```

需要特别指出的是，根据Python语言关于if语句里对于真值的判定，只要返回的对象没有特别规定真假，那么，有对象为真，没对象为假。所以，在使用if语句判定时，只需要把search函数放在if的条件部分即可，不需要写额外的判断。

13.4.3 正则对象的常用成员函数

2.match函数

与search函数功能相似但是在使用过程中非常容易混淆的是match函数，其参数与search完全相同，但是，该函数只能从字符串的“开始位置”搜索。观察如下的实例：

```
>>> pattern = re.compile("og")
>>> pattern.search("dog")
<re.Match object; span=(1, 3), match='og'>
>>> print(pattern.match("dog"))
None
```

可以看到，在匹配过程中，search函数可以匹配字符串任何位置出现的、模式字符串所描述的内容，但是match函数则不行。在单行模式下，match函数相当于search函数的模式字符串中，加入了行首标识符“^”。在多行模式下，相当于在search函数下加入了含断言的行首标识符“(?<!\n)^”。

13.4.3 正则对象的常用成员函数

3.fullmatch函数

与match函数类似的是fullmatch函数，它相当于在单行模式下search函数同时加入了行首标记符“^”和行尾标记符“$”，即只有整个字符串都符合条件，才会得到匹配。具体实例如下：

```
>>> pattern = re.compile(r"\w+@\w+(?:\.\w+)+")
>>> pattern.search("我的电子邮箱是 email@domain.com ")
<re.Match object; span=(8, 24), match='email@domain.com'>
>>> print(pattern.fullmatch("我的电子邮箱是 email@domain.com "))
>>> None
>>> pattern.fullmatch("email@domain.com")
<re.Match object; span=(0, 16), match='email@domain.com'>
```

13.4.3 正则对象的常用成员函数

4. findall和finditer函数

在使用search、match、fullmatch时，一次都只能返回一个匹配。那么如果需要一次返回多个匹配，就需要使用findall和finditer。需要注意的不同点是，findall返回的是一个列表，而finditer返回的是一个迭代器。这两个函数的搜索模式与search是完全相同的。并且，如果在使用findall时指定了多个捕获组，则会返回元组的列表。具体实例如下：

```
>>> pattern = re.compile(".")
>>> pattern.findall("dog")
['d', 'o', 'g']
>>> pattern.finditer("dog")
<callable_iterator object at 0x00000236D06B9F40>
>>> pattern = re.compile("(a)(p)")
>>> pattern.findall("apple")
[('a', 'p')]
```

13.4.3 正则对象的常用成员函数

5.split函数

search、match、fullmatch、findall和finditer五个函数是关于正则表达式在使用匹配功能时需要用到的函数。但正则表达式可以完成的功能远不止此，正则表达式还拥有切割字符串的功能。比如，在某些应用场景下，对于一段文本内容，如果需要将文本按所有标点符号切割，使用字符串的split方法就非常麻烦，而使用正则表达式的split函数就很简单。split函数有两个参数，其中，第一个参数是原字符串，第二个可选参数是切割的最大次数，默认为0，即无限次。具体实例如下：

```
>>> pattern = re.compile(r"\W+")
>>> pattern.split("使用正则表达式的split函数很简单，如图1-1所示。")
['使用正则表达式的split函数很简单', '如图1', '1所示', '']
```

13.4.3 正则对象的常用成员函数

（1）在使用切割标点符号的时候，“\W”的效果并不好，可以看到字符串“图1-1”应当是一个整体，而在切割时由于“\W”匹配的是非文本字符，所以将“-”也视为了标点符号。这再一次说明了本章前面曾提到的在使用过程中不推荐使用“\W”、“\D”等通用字符的正确性。

```
>>> pattern = re.compile(r"(\W+)")
>>> pattern.split("使用正则表达式的split函数很简单，如图1-1所示。")
['使用正则表达式的split函数很简单', '，', '如图1', '-', '1所示', '。', '']
```

13.4.3 正则对象的常用成员函数

（2）在使用split函数中，分割匹配的正则表达式加括号和不加括号是不一样的。其区别是，如果分割字符串加了括号，那么括号内的内容也将会出现在切割后的字符串里。注意观察上面第二次切割的结果字符串，里面有出现标点符号。

（3）如果在使用split函数中，必须要使用分组，又不希望切割内容出现在结果中，那么就可以使用前文所介绍的非捕获组，将这些组忽略掉。

```
>>> pattern = re.compile(r"(?:\W+)")
>>> pattern.split("使用正则表达式的split函数很简单，如图1-1所示。")
['使用正则表达式的split函数很简单', '如图1', '1所示', '']
```

13.4.3 正则对象的常用成员函数

6. sub函数

sub函数用于替换字符串中的匹配项，假设正则对象为pattern，则调用search函数的语法如下：

pattern.sub(repl, string, count=0)

其中，第一个参数repl表示替换的字符串；第二个参数string表示要被替换的原始字符串；第三个参数count，是可选参数，默认为0，如果其值不为0，则该值意味着最大替换次数，一般不使用。

13.4.3 正则对象的常用成员函数

具体实例如下：

```
>>> import re
>>> phone = "0592-123-4567 # 这是一个厦门市的电话号码"
>>> # 删除字符串中的Python注释
>>> pattern = re.compile(r'#.*$')
>>> num = pattern.sub("", phone)
>>> print("电话号码是: ",num)
电话号码是:  0592-123-4567
>>> # 删除非数字字符
>>> pattern = re.compile(r'\D')
>>> num = pattern.sub("",phone)
>>> print("电话号码是：",num)
电话号码是：  05921234567
```

13.4.4 正则表达式里的match对象

•在表达式里，除了表达式的运算符和操作数以外，最重要的当然就是表达式的返回结果，正则表达式也不例外。在正则表达式里，返回一种名为“match”的结果。比如，search和match函数，都会返回一个“match”对象。match对象本身可以视为True值，所以可以直接用在if函数的判断语句里。在这个对象里，也有一些成员函数需要注意，它们也非常常用。

•group函数就是一个最常用的函数，该函数可以直接返回匹配里对应组的内容，它的参数是组的编号或者具名组的名称。该函数已经在捕获组里有介绍过，这里再重复一次，如果没有参数，返回的是匹配全文。如果只有一个参数，返回的是对应组。如果有多个参数，返回的是对应组的元组。并且，如果捕获组里拥有具名组，那么还可以用组名代替组的序号。

13.4.4 正则表达式里的match对象

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.group()
'Malcolm Reynolds'
>>> m.group(0)
'Malcolm Reynolds'
>>> m.group(1)
'Malcolm'
>>> m.group(2)
'Reynolds'
>>> m.group("first_name")
'Malcolm'
>>> m.group("last_name")
'Reynolds'
>>> m.group(1, "last_name")
('Malcolm', 'Reynolds')
```

13.4.4 正则表达式里的match对象

当然，也可以使用groups函数一次性返回所有组的元组，实例如下：

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.groups()
('Malcolm', 'Reynolds')
```

13.4.4 正则表达式里的match对象

在使用时，有些组其实是不会参与匹配的。比如，匹配一个小数，那么假定小数点不存在，则小数部分也不存在。那么这时，可以使用groups函数的参数来指定空匹配时的默认值。该参数默认为None，即返回一个空值。具体实例如下：

```
>>> m = re.match(r"(\d+)\.?(\d+)?", "24")
>>> m.groups()
('24', None)
>>> m.groups('0')
('24', '0')
>>> m.groups('1')
('24', '1')
```

13.4.4 正则表达式里的match对象

不过，元组往往不太好操作。在操作时可能更希望使用字典，那么这时就可使用groupdict函数返回一个字典。该函数同样有一个默认参数，其含义等同于groups。具体实例如下：

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.groupdict()
{'first_name': 'Malcolm', 'last_name': 'Reynolds'}
```

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学与技术系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/post/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，厦门大学信息学院实验教学中心主任，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过400万次，累计访问量超过1500万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

林子雨大数据系列教材
用于导论课、专业课、实训课、公共课

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

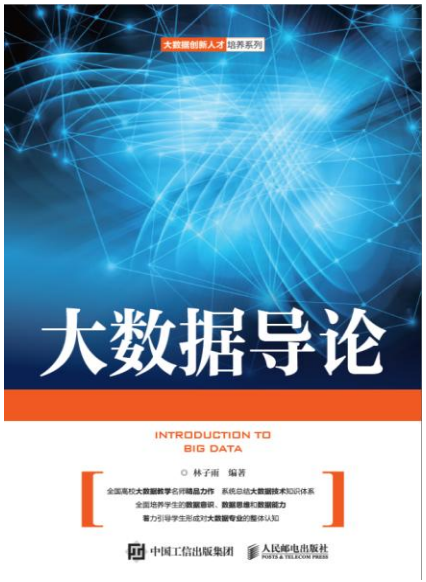
- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/



开设大数据专业导论课的优质教材

扫一扫访问教材官网

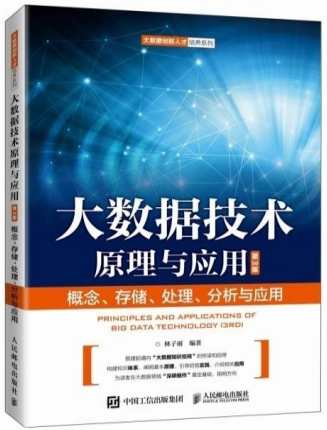
附录F：《大数据技术原理与应用（第3版）》教材

《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-54405-6 定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。
欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

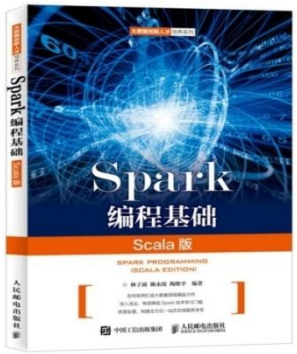
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

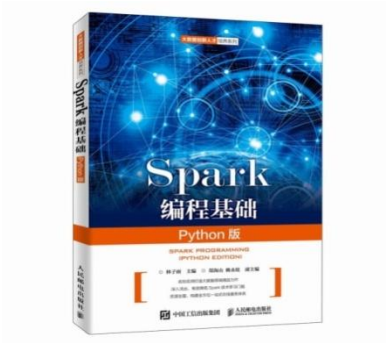
披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9

教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3

教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

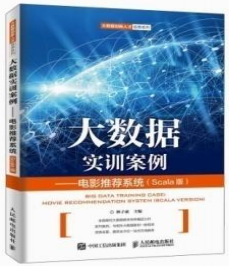
扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

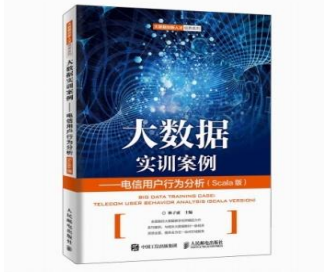
附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）
《电信用户行为分析》（已经于2019年5月出版）
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！
http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页


Thank You!