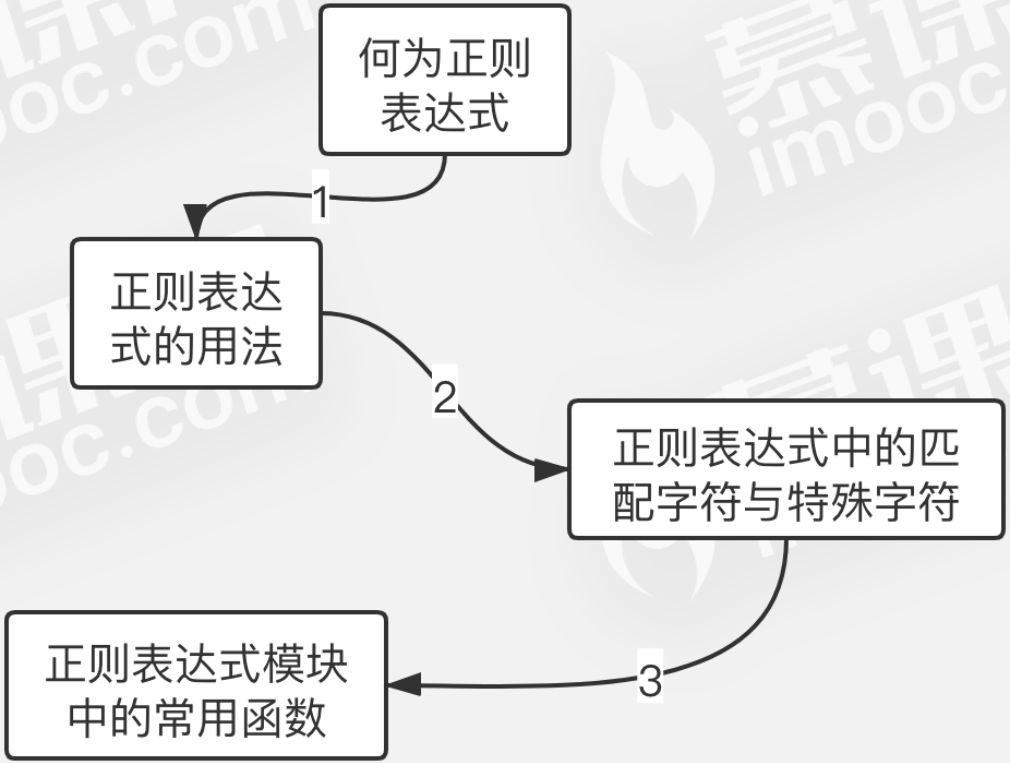
正则表达式与项目练习

本周内容

- Python中的正则表达式
- 综合项目实战-抽奖系统

正则表达式



综合项目实战—抽奖系统-效果展示

```
哦可惜  您没有中奖

Process finished with exit code 0
```

```
/Users/zhangdewei/.virtualenvs/gift/
恭喜您  获得  ipad  奖品

Process finished with exit code 0
```

抽奖系统功能模块介绍

知识点

功能

1: 父类的创建
2: json文件的读写，私有函数的定义
3: 字典的练习，循环的练习
4: 条件语句的练习
5: 异常的处理与抛出

2

base模块

1

避免业务逻辑，只做底层操作，如对用户的增删改查，奖品的增删改查，直接和storage模块关联

1: 类的继承
2: 多态的练习 super函数
3: 条件语句的练习
4: 循环的练习

2

admin模块

1

1: 继承base模块2: 用户的增删改查
3: 奖品的增删改查

1: 类的继承
2: 父类私有函数的调用
3: 启蒙与强化开发思维

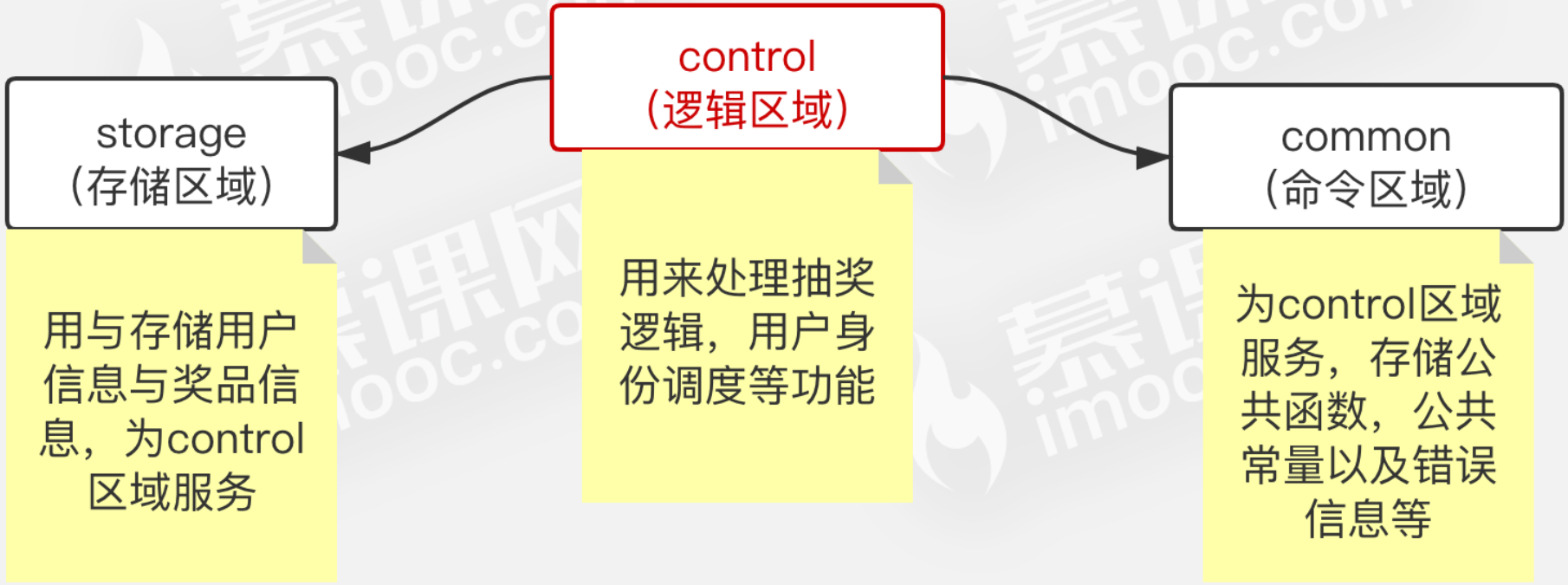
2

user模块

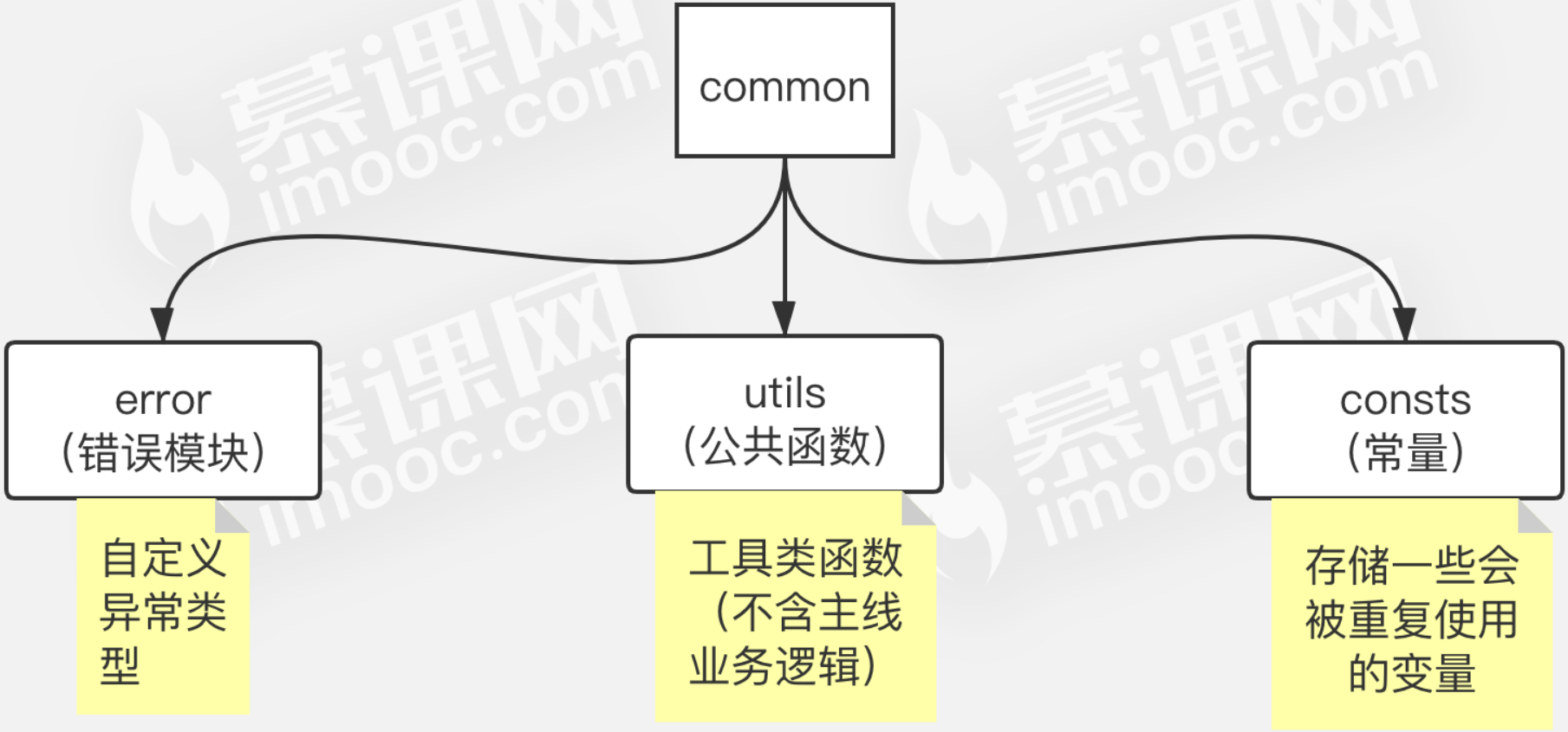
1

1: 用户身份验证
2: 抽奖功能

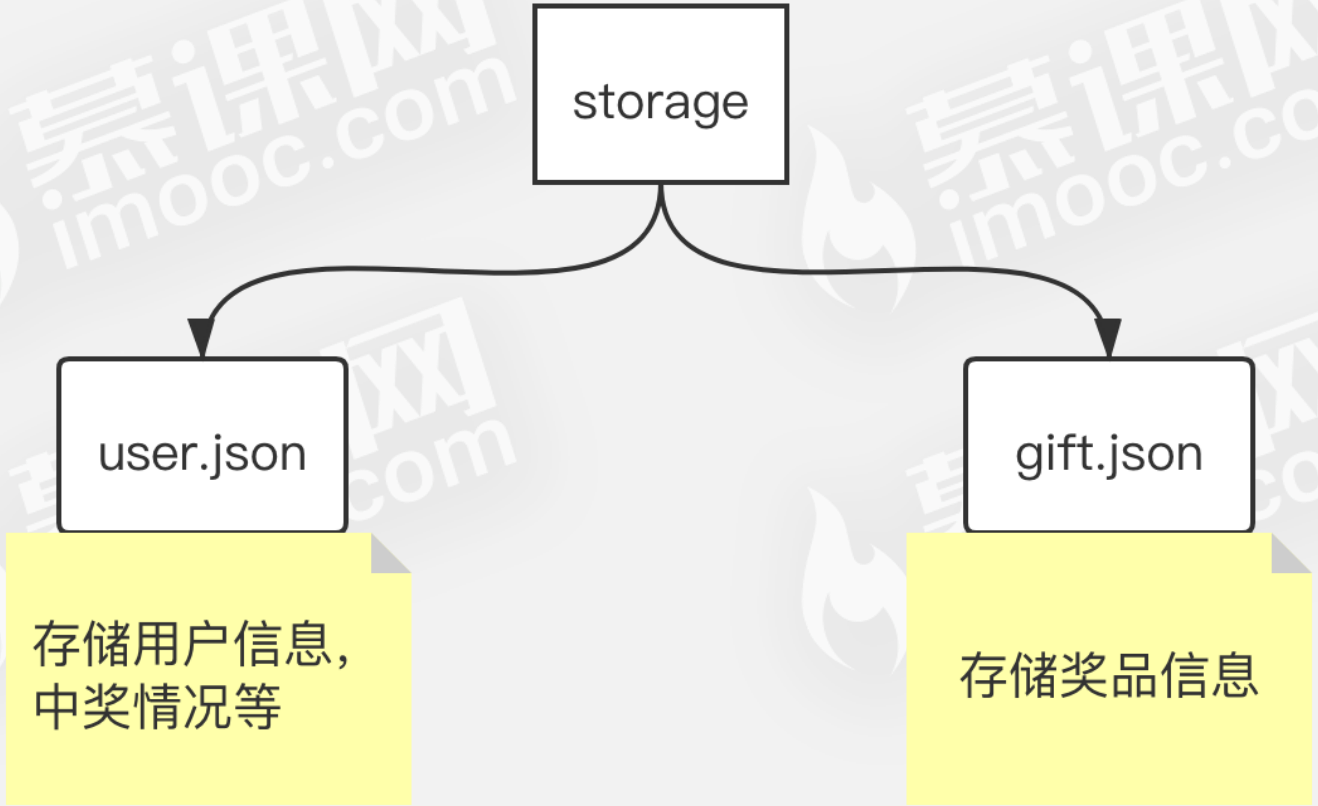
抽奖系统三大区域介绍


control
（逻辑区域）
storage
（存储区域）
common
（命令区域）
用与存储用户信息与奖品信息，为control区域服务
用来处理抽奖逻辑，用户身份调度等功能
为control区域服务，存储公共函数，公共常量以及错误信息等

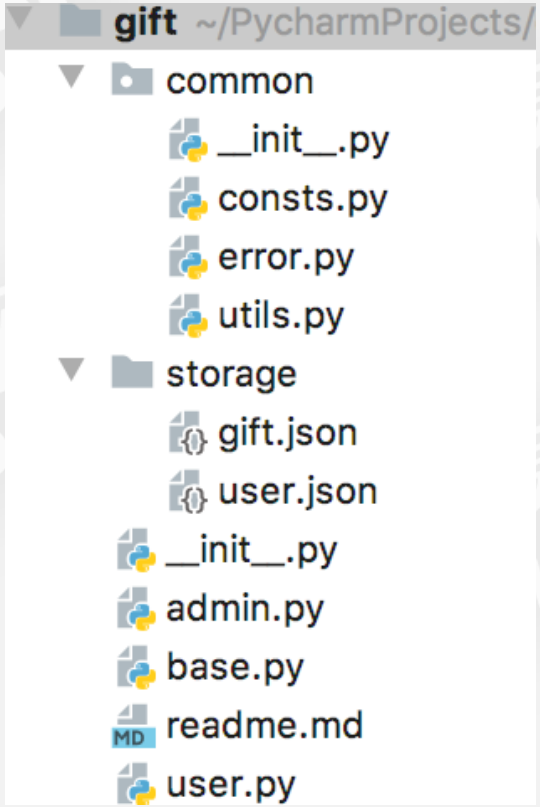
抽奖系统-common模块



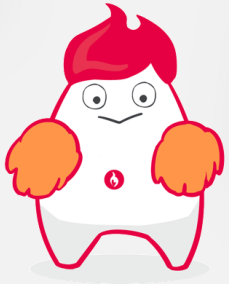
抽奖系统-storage模块



抽奖系统代码结构


gift ~/PycharmProjects/
common
__init__.py
consts.py
error.py
utils.py
storage
gift.json
user.json
__init__.py
admin.py
base.py
readme.md
user.py

正则表达式



本节课内容

- 什么是正则表达式
- 正则表达式的使用场景
- 正则表达式模块 re 的用法

什么是正则表达式

正则表达式的应用场景

- 判断一个字符串是否符合规则
- 取出制定数据
- 爬虫岗位较为核心的技术
- 彩票网站匹配彩票信息

正则表达式模块-re

```
import re


str_data = 'hello xiaomu, this is a good day!'
result = re.search('h([a-zA-Z])s', str_data)
print(result.groups())
```

只匹配一次

```
('i',)
```

正则表达式模块-re

匹配多次

```
import re

str_data = '本期彩票结果是:10,20,1,5,7,21,12'
result = re.findall('(\d+,\d+,\d+,\d+,\d+,\d+,\d+)', str_data)
print(result)
```

```
['10,20,1,5,7,21,12']
```

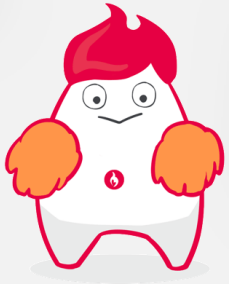
匹配字符串的需要条件

- 正则表达式模块--re
- 匹配“针”的规则
- 取“针”的大海--字符串

[h(a-zA-Z)s]

(\d+,\d+.....)

正则表达式-匹配字符



本节课内容

- 正则表达式的匹配字符

正则表达式中的特殊字符

特殊字符	描 述
\d	匹配任何十进制数字，与[0-9]一致（\D 与\d 相反，不匹配任何非数值型的数字）
\w	匹配任何字母数字字符（\W 与之相反）
\s	匹配任何空格字符，与[\n\t\r\v\f]相同（\S 与之相反）
\A(\Z)	匹配字符串的起始（结束）
.	匹配任何字符（除了\n 之外）

正则表达式的使用

```
In [1]: import re

In [2]: data = 'hello dewei you are 33 age old'
```

正则表达式的使用

```
In [3]: print(re.findall('\d', data))
['3', '3']
```

\d 匹配了数字

正则表达式的使用

```
In [4]: print(re.findall('\s', data))
[' ', ' ', ' ', ' ', ' ', ' ']
```

\s 匹配 了空格 \n

正则表达式的使用

```
In [5]: print(re.findall('\w', data))
['h', 'e', 'l', 'l', 'o', 'd', 'e', 'w', 'e', 'i', 'y', 'o', 'u', 'a', 'r', 'e', '3', '3', 'a', 'g', 'e', 'o
', 'l', 'd']
```

\w 匹配 字母　　数字

正则表达式的使用

```
In [13]: print(re.findall('\Ahello', data))
['hello']

In [14]: print(re.findall('\Ahellos', data))
[]
```

\A 匹配字符串开头（想想 startswith）

正则表达式的使用

```
In [16]: print(re.findall('old\Z', data))
['old']

In [17]: print(re.findall('aold\Z', data))
[]
```

\Z 匹配字符串结尾（想想 endswith)

正则表达式的使用

```
In [18]: print(re.findall('.', data))
['h', 'e', 'l', 'l', 'o', ' ', 'd', 'e', 'w', 'e', 'i', ' ', 'y', 'o', 'u', ' ', 'a', 'r', 'e', ' ', '3', '3
', ' ', 'a', 'g', 'e', ' ', 'o', 'l', 'd']
```

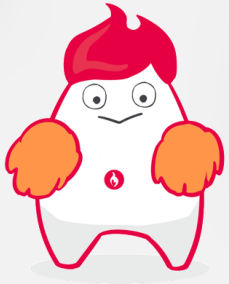
- . 匹配了 空格\n 字母 数字

思考

```
In [5]: print(re.findall('\w', data))
['h', 'e', 'l', 'l', 'o', 'd', 'e', 'w', 'e', 'i', 'y', 'o', 'u', 'a', 'r', 'e', '3', '3', 'a', 'g', 'e', 'o
', 'l', 'd']
```

- 如何返回复数呢?

正则表达式-量词符号与组



正则表达式中的符号

符号	描 述
re1|re2	匹配正则表达式 re1 或者 re2
^	匹配字符串起始部分
$	匹配字符串终止部分
*	匹配 0 次或者多次前面出现的正则表达式
+	匹配 1 次或者多次前面出现的正则表达式

正则表达式中的符号

符号	描 述
{N}	匹配 N 次前面出现的正则表达式
{M,N}	匹配 M ~ N 次前面出现的正则表达式
[...]	匹配来自字符集的任意单一字符
[..x-y..]	匹配 x ~ y 范围中的任意单一字符
[^...]	不匹配此字符集中出现的任何一个字符，包括某一范围的字符（如果在此字符集中出现）
\	将特殊字符无效化

正则表达式的应用

```
In [28]: data = 'dewei@imooc.com'

In [29]: print(re.findall('dewei|com|imooc', data))
['dewei', 'imooc', 'com']
```

| 或的关系 只要存在就能捕获

匹配到的数据只按字符串顺序返回，而不是按照匹配规则返回

正则表达式的应用

```
In [30]: print(re.findall('^dewei', data))
['dewei']

In [31]: print(re.findall('^haha', data))
[]
```

^ 等同于 \A

正则表达式的应用

```
In [32]: print(re.findall('com$', data))
['com']

In [33]: print(re.findall('net$', data))
[]
```

$ 等同于 \Z

正则表达式的应用

```
In [42]: data = 'dewei@imooc.com'

In [43]: print(re.findall('\w*', data))
['dewei', '', 'imooc', '', 'com', '']
```

w* 匹配0次或多次数字或字母

正则表达式的应用

```
In [44]: data = 'dewei@imooc.com'

In [45]: print(re.findall('\w+', data))
['dewei', 'imooc', 'com']
```

w+ 匹配1次或多次数字或字母

@和. 属于0次范围，不会被匹配出来

正则表达式的应用

```
In [53]: data = 'dewei@imooc.com'

In [54]: print(re.findall('\w{3}', data))
['dew', 'imo', 'com']
```

```
In [48]: print(re.findall('[a-z]{3}', data))
['dew', 'imo', 'com']
```

{N} 对于匹配到的数据只获取N个

[a-zA-Z0-9] 等同于 \w

正则表达式的应用

```
In [76]: data = 'dewei@imooc.com'

In [77]: print(re.findall('\w{1,5}', data))
['dewei', 'imooc', 'com']
```

{N,M} 给出匹配到的数据范围

```
In [78]: print(re.findall('\w{1, 5}', data))
[]
```

注意：N 和 M 中间的逗号左右不可以有空格

正则表达式的应用

```
In [82]: data = 'dewei@imooc.com'

In [83]: print(re.findall('[^dewei]', data))
['@', 'm', 'o', 'o', 'c', '.', 'c', 'o', 'm']
```

字符集中 ^ 不再代表开始的意思，而是过滤掉

组的概念

符号	描 述
()	在匹配规则中只要指定的数据

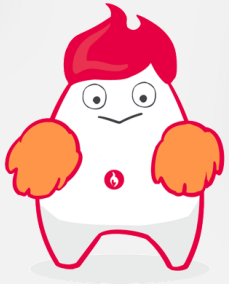
组的应用

```
In [90]: result = re.search('hello (.*) name is (.*)', test)

In [91]: result.groups()
Out[91]: ('my', 'dewei')

In [92]: result.group(1)
Out[92]: 'my'

In [93]: result.group(2)
Out[93]: 'dewei'
```

贪婪与非贪婪

- 0次或多次属于贪婪模式
- 通过？组合变成非贪婪模式

正则表达式-量词符号



本课内容

- findall()的使用
- search()的使用
- group()与groups()的使用
- split()正则替换
- compile的使用
- match的使用
- re 的 额外匹配要求

findall()的使用

- findall(pattern，string [, flags])

- 查找字符串中所有（非重复）出现的正则表达式模式，并返回一个匹配列表

search()的使用

- search(pattern， string， flags=0)

- 使用可选标记搜索字符串中第一次出现的正则表达式模式。
  如果匹配成功，则返回匹配对象； 如果失败，则返回 None

group()与groups()

- group(num)返回整个匹配对象，或者编号为 num 的特定子组

- groups(): 返回一个包含所有匹配子组的元组（如果没有成功匹配，则返回一个空元组）

group()与groups()的使用

```
In [90]: result = re.search('hello (.*) name is (.*)', test)

In [91]: result.groups()
Out[91]: ('my', 'dewei')

In [92]: result.group(1)
Out[92]: 'my'

In [93]: result.group(2)
Out[93]: 'dewei'
```

split()正则替换

- split(pattern， string， max=0)

- 根据正则表达式的模式分隔符， split 函数将字符串分割为列表，然后返回成功匹配的列表，分隔最多操作 max 次（默认分割所有匹配成功的位置）

split()正则替换应用

```
In [94]: data = 'hello world'

In [95]: print(re.split('\W', data))
['hello', 'world']
```

re模块-match

- match(pattern， string， flags=0)

- 尝试使用带有可选的标记的正则表达式的模式来匹配字符串。如果匹配成功，就返回匹配对象； 如果失败，就返回 None

re模块-match使用

```
[105]: data = 'hello world'

[106]: result = re.match('hello', data)

[107]: result.group()
[107]: 'hello'
```

re模块-compile

- compile(pattern， flags = 0)

- 定义一个匹配规则的对象

re模块-compile使用

```
In [129]: data = 'hello my email is dewei@imooc.com i like python'

In [130]: re_obj = re.compile('email is (.*?) i')

In [131]: result = re_obj.findall(data)

In [132]: result
Out[132]: ['dewei@imooc.com']
```

re的额外匹配要求

属性	描述
re.I、 re.IGNORECASE	不区分大小写的匹配
re.L、 re.LOCALE	根据所使用的本地语言环境通过\w、 \W、 \b、 \B、 \s、 \S 实现匹配
re.M、 re.MULTILINE	^和$分别匹配目标字符串中行的起始和结尾，而不是严格匹配整个字符串本身的起始 和结尾
re.S、 rer.DOTALL	“.”（点号）通常匹配除了\n（换行符）之外的所有单个字符；该标记表示“.”（点号） 能够匹配全部字符
re.X、 re.VERBOSE	通过反斜线转义， 否则所有空格加上#（以及在该行中所有后续文字）都被忽略，除非 在一个字符类中或者允许注释并且提高可读性