第3天服务器信息清洗

一、 字典

1. 字典特性介绍

> 字典在 Python 中极为重要，是属于映射类型的数据结构。 字典有一对儿大括号组成 `{}`, 字典内的元素都是成对儿出现的 `{"a": 1}`, 他们用英文的冒号(`:`)隔开, 左边叫做键(key),右边的叫值(value), 通常叫做键值对儿。 每个元素用英文的逗号 (`,`) 隔开
> `{"a": 1, "b": 2}`

2. 创建字典

> 创建字典可以使用一对儿大括号， 也可以使用 `dict()`

```
>>> d1 = {}
>>> type(d1)
<class 'dict'>
>>> d2 = dict()
>>> type(d2)
<class 'dict'>
>>> d3 = {"size": 8096}
>>>
```


3. 字典的 key 和 value

3.1 字典的 key

在一个字典中，key 不允许有重复

```
>>> {"a": 1, "a": 10}
{'a': 10}
```

并且必须是 python 中不可变的数据类型 如： 整型 浮点型 布尔值 字符串 元组

```
>>> {1: "整型", 1.1: "浮点型", False: "布尔值", "abc": "字符串", (1,2): "元组" }
{1: '整型', 1.1: '浮点型', False: '布尔值', 'abc': '字符串', (1, 2): '元组'}
>>>
```

最常用的是字符串，因为比较容易有语意，易读

```
>>> {"user": "shark", "age": 18}
{'user': 'shark', 'age': 18}
>>>
```


小扩展

注意： python 中认为 `1` 和 `True`，`0` 和 `False` 的值是相等的 但不是一个对象，因为它们的 id 不同

```
>>> 1 == True
True
>>> False == 0
True
>>> 1 is True
False
>>> id(1);id(True)
4551162464
4550779216
>>> 0 is False
False
```

所以你会看到如下的现象

```
>>> {1: "整型", True: "布尔型"}
{1: '布尔型'}
>>> {True: "布尔型", 1: "整型"}
{True: '整型'}
>>>
```

3.2 字典中的 value

字典中的 value 可以是 Python 中任意的一个数据对象:

- 整型、浮点型、布尔值、字符串、列表、元组、字典

```
>>> {'port': 3306, "height": 1.81, 'stat': True, "name": "dbserver"}
{'port': 3306, 'height': 1.81, 'stat': True, 'name': 'dbserver'}

>>> {"mysql-01": {
...     "cpu": 4,
...     "memroy": [4096,4096]
...     }
... }
{'mysql-01': {'cpu': 4, 'memroy': [4096, 4096]}}
>>>
```


- 函数对象等

```
>>> def foo():
...     print("hello")
...
>>> {"1": foo}
{'1': <function foo at 0x10f5b9e18>}
```

4. 获取字典中 key 和 value

4.1 检查字典中是否存在某个 key

> 可以使用 `in` 关键字

```
In [39]: dic_map = {
    ...:     "Manufacturer": "manufacturer",
    ...:     "Product Name": "pod_name",
    ...:     "Serial Number": "sn"
    ...: }

In [40]: line = '\tManufacturer: Alibaba Cloud'

In [41]: line = line.strip()

In [42]: k, v  = line.split(": ")

In [43]: k
Out[43]: 'Manufacturer'

In [44]: k in dic_map
Out[44]: True

In [45]:
```


4.2 使用 `[]` 获取指定 key 的 value

```
In [48]: dic_map['Manufacturer']
Out[48]: 'manufacturer
```

> 这种方式是非常的高效做法，推荐使用。但是有个问题，假设获取字典中不存在的 key 的值

4.3 使用字典的 `get()` 方法

```
>>> dic = {'a': 1, 'b': 2}
>>> dic.get('a')    # 获取到 'a' 对应的值(value)
1
>>>
>>> dic.get('c')    # key 不存在字典中，则返回 None
>>>
>>> v = dic.get('c')
>>> type(v)
<class 'NoneType'>
>>> dic.get('c', '5')  # key 不存在，返回指定的值
'5'
>>>
```

4.4 循环字典的 key 和 value

> 字典对象的 `items()` 方法会获取到字典的 key 和 value, 它是一个可迭代对象

```

In [49]: dic_map.values()
Out[49]: dict_values(['manufacturer', 'pod_name', 'sn'])

In [50]: for k, v in dic_map.items():
    ...:     print(k, v)
Manufacturer manufacturer
Product Name pod_name
Serial Number sn

In [51]:
```

4.5 向字典中添加键值对

- `[]` 方式

```
>>> info = {}
>>> info["cpu"] = 4
>>> info["memory"] = [4096, 4096]
>>> info
{'cpu': 4, 'memory': [4096, 4096]}
>>>
```

生产示例：

> 字段映射

```
In [51]: dic_map
Out[51]:
{'Manufacturer': 'manufacturer',
 'Product Name': 'pod_name',
 'Serial Number': 'sn'}

In [52]: li = ['Manufacturer: Alibaba Cloud',
    ...:       'Product Name: Alibaba Cloud ECS',
    ...:       'Version: pc-i440fx-2.1',
    ...:       'Serial Number: 0f7e3d86-7742-4612-9f93-e3a9e4754157']

In [53]: prod_info = {}

In [54]: for line in li:
    ...:     k, v = line.split(": ")
    ...:     if k in dic_map:
    ...:         prod_info[dic_map[k]] = v  # 获取的一个字典的值作为另一个字典的键
    ...:

In [55]: prod_info
```

```
Out[55]:
{'manufacturer': 'Alibaba Cloud',
 'pod_name': 'Alibaba Cloud ECS',
 'sn': '0f7e3d86-7742-4612-9f93-e3a9e4754157'}
```

- `update` 方式

```
>>> disk = {"disk": [10240]}
>>> info.update(disk)
>>> info
{'cpu': 4, 'memory': [4096, 4096], 'disk': [10240]}
>>>
```

4.6 字典的编程之道：用字典实现 case 语句

程序源码


```
data = {
    "0": "zero",
    "1": "one",
    "2": "two",
}

while True:
    arg = input(">>:")
    v = data.get(arg, "nothing")
    print(v)
```

执行程序

```
$ python3 hello.py
>>:0
zero
>>:1
one
>>:9
nothing
>>:
```

> 思考题，如何给上例添加 输入 'q' 退出

二、集合

1 集合特性介绍

```
在 python 中集合看起来像是只有 key 的字典

{'disk','cpu','memory','motherboard'}

在 python 解释器中表现为 set

集合内的元素不允许重复
```


2. 高效创建集合和转换

`set()`

```
>>> s1 = set()
>>> type(s1)
<class 'set'>
>>>
```

转换

```
In [99]: set('disk')
Out[99]: {'d', 'i', 'k', 's'}

In [100]: set(['disk','cpu','memory'])
Out[100]: {'cpu', 'disk', 'memory'}

In [101]: set(('disk','cpu','memory'))
Out[101]: {'cpu', 'disk', 'memory'}

In [102]: set({'disk': '560G','cpu': '4'})
Out[102]: {'cpu', 'disk'}
```

3. 集合运算

- `&` 交集

获取两个集合都有的元素

```
In [55]: s1 = {"192.168.1.51",  "192.168.1.45"}

In [56]: s2 = {"192.168.1.51", "192.168.1.78"}

In [57]: s1 & s2
Out[57]: {'192.168.1.51'}
```

- `|` 并集

把两个集合的元素合并在一起，产生一个新的集合

```
In [60]: s1 | s2
Out[60]: {'192.168.1.45', '192.168.1.51',
'192.168.1.78'}
```

- `-`差集

返回第一个集合中独有的元素。 就是只保留在第一个集合中出现并且不在第二个集合中出现的元素。

```
In [55]: s1 = {"192.168.1.51", "192.168.1.45"}

In [56]: s2 = {"192.168.1.55", "192.168.1.51"}

In [61]: s1 - s2
Out[61]: {'192.168.1.45'}

In [62]: s2 - s1
Out[62]: {'192.168.1.78'}
```

- `^`异或运算

获取两个集合的分别独有的元素，组合为一个新的集合对象。

```
In [55]: s1 = {"192.168.1.51", "192.168.1.45"}

In [56]: s2 = {"192.168.1.55", "192.168.1.51"}

In [63]: s1 ^ s2
Out[63]: {'192.168.1.45', '192.168.1.78'}
```