数据结构与算法（Python版）

Python数据类型的性能（上）

陈斌 北京大学 gischen@pku.edu.cn

Python数据类型的性能

- 前面我们了解了“大O表示法”以及对不同的算法的评估
- 下面来讨论下Python两种内置数据类型上各种操作的大O数量级

  列表list和字典dict

  这是两种重要的Python数据类型，后面的课程会用来实现各种数据结构

  通过运行试验来估计其各种操作运行时间数量级

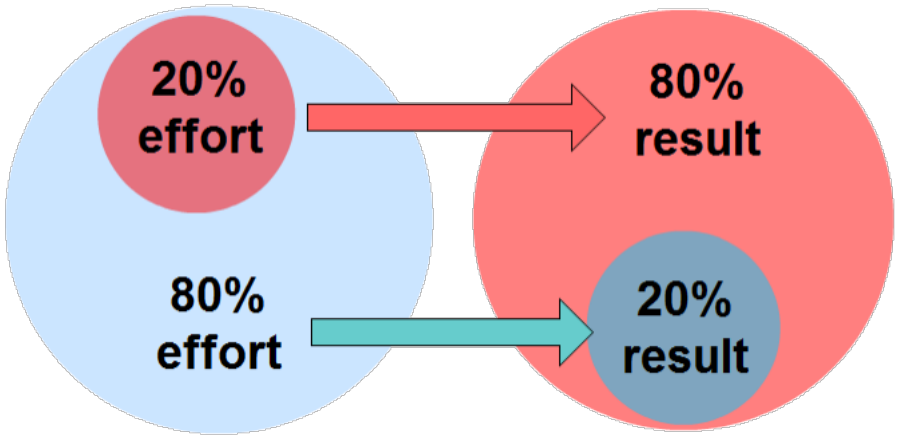
对比list和dict的操作

类型	list	dict
索引	自然数i	不可变类型值key
添加	append、extend、insert	b[k]=v
删除	pop、remove*	pop
更新	a[i]=v	b[k]=v
正查	a[i]、a[i:j]	b[k]、copy
反查	index(v)、count(v)	无
其它	reverse、sort	has_key、update

List列表数据类型

- list类型各种操作（interface）的实现方法有很多，如何选择具体哪种实现方法?
- 总的方案就是，让**最常用**的操作**性能最好**，牺牲不太常用的操作

  80/20准则：80%的功能其使用率只有20%



List列表数据类型常用操作性能

- 最常用的是：按索引取值和赋值（v = a[i], a[i]= v）

  由于列表的随机访问特性，这两个操作执行时间与列表大小无关，均为O(1)

- 另一个是列表增长，可以选择append()和__add__() “+”

  lst.append(v)，执行时间是O(1)

  lst= lst+ [v]，执行时间是O(n+k)，其中k是被加的列表长度

  选择哪个方法来操作列表，决定了程序的性能

4种生成前n个整数列表的方法

- 首先是循环连接列表（+）方式生成
- 然后是用append方法添加元素生成
- 接着用**列表推导式**来做
- 最后是range函数调用转成列表

```
def test1():
    l = []
    for i in range(1000):
        l = l + [i]

def test2():
    l = []
    for i in range(1000):
        l.append(i)

def test3():
    l = [i for i in range(1000)]

def test4():
    l = list(range(1000))
```

使用timeit模块对函数计时

- 创建一个Timer对象，指定需要**反复运行**的语句和只需要**运行一次**的“安装语句”
- 然后调用这个对象的timeit方法，其中可以指定反复运行多少次

```
from timeit import Timer
t1= Timer("test1()", "from __main__ import test1")
print "concat %f seconds\n" % t1.timeit(number= 1000)

t2= Timer("test2()", "from __main__ import test2")
print "append %f seconds\n" % t2.timeit(number= 1000)

t3= Timer("test3()", "from __main__ import test3")
print "comprehension %f seconds\n" % t3.timeit(number= 1000)

t4= Timer("test4()", "from __main__ import test4")
print "list range %f seconds\n" % t4.timeit(number= 1000)
```

4种生成前n个整数列表的方法计时

- 我们看到，4种方法运行时间差别很大

  列表连接（concat）最慢，List range最快，速度相差近200倍。

  append也要比concat快得多

  另外，我们注意到列表推导式速度是append两倍的样子

```
>>>
concat 1.889487 seconds

append 0.091561 seconds

comprehension 0.038418 seconds

list range 0.009710 seconds
```

List基本操作的大O数量级

Operation	Big-O Efficiency
index []	O(1)
index assignment	O(1)
append	O(1)
pop()	O(1)
pop(i)	O(n)
insert(i,item)	O(n)
del operator	O(n)
iteration	O(n)
contains (in)	O(n)
get slice [x:y]	O(k)
del slice	O(n)
set slice	O(n+k)
reverse	O(n)
concatenate	O(k)
sort	O(n log n)
multiply	O(nk)