7个提升Python程序性能的好习惯

掌握一些技巧，可尽量提高Python程序性能，也可以避免不必要的资源浪费。

1、使用局部变量

尽量使用局部变量代替全局变量：便于维护，提高性能并节省内存。

使用局部变量替换模块名字空间中的变量，例如 ls = os.linesep。一方面可以提高程序性能，局部变量查找速度更快；另一方面可用简短标识符替代冗长的模块变量，提高可读性。

2、减少函数调用次数

对象类型判断时，采用isinstance()最优，采用对象类型身份（id()）次之，采用对象值（type()）比较最次。

```
#判断变量num是否为整数类型
type(num) == type(0) #调用三次函数
type(num) is type(0) #身份比较
isinstance(num,(int)) #调用一次函数
```

不要在重复操作的内容作为参数放到循环条件中，避免重复运算。

```
#每次循环都需要重新执行len(a)
while i < len(a):
    statement

#len(a)仅执行一次
m = len(a)
while i < m:
    statement
```

如需使用模块X中的某个函数或对象Y，应直接使用from X import Y，而不是import X; X.Y。这样在使用Y时，可以减少一次查询（解释器不必首先查找到X模块，然后在X模块的字典中查找Y）。

3、采用映射替代条件查找

映射（比如dict等）的搜索速度远快于条件语句（如if等）。Python中也没有select-case语句。

```
#if查找
if a == 1:
    b = 10
elif a == 2:
    b = 20
...

#dict查找，性能更优
```

```
d = {1:10,2:20,...}
b = d[a]
```

4、直接迭代序列元素

对序列（str、list、tuple等），直接迭代序列元素，比迭代元素的索引速度要更快。

```
a = [1,2,3]

#迭代元素
for item in a:
    print(item)

#迭代索引
for i in range(len(a)):
    print(a[i])
```

5、采用生成器表达式替代列表解析

列表解析（list comprehension），会产生整个列表，对大量数据的迭代会产生负面效应。

而生成器表达式则不会，其不会真正创建列表，而是返回一个生成器，在需要时产生一个值（延迟计算），对内存更加友好。

```
#计算文件f的非空字符个数
#生成器表达式
l = sum([len(word) for line in f for word in line.split()])

#列表解析
l = sum(len(word) for line in f for word in line.split())
```

6、先编译后调用

使用eval()、exec()函数执行代码时，最好调用代码对象（提前通过compile()函数编译成字节码），而不是直接调用str，可以避免多次执行重复编译过程，提高程序性能。

正则表达式模式匹配也类似，也最好先将正则表达式模式编译成regex对象（通过re.complie()函数），然后再执行比较和匹配。

7、模块编程习惯

模块中的最高级别Python语句（没有缩进的代码）会在模块导入（import）时执行（不论其是否真的必要执行）。因此，应尽量将模块所有的功能代码放到函数中，包括主程序相关的功能代码也可放到main()函数中，主程序本身调用main()函数。

可以在模块的main()函数中书写测试代码。在主程序中，检测name的值，如果为'main'（表示模块是被直接执行），则调用main()函数，进行测试；如果为模块名字（表示模块是被调用），则不进行测试。

作者：爱coding，会编程的核电工程师。zhihu.com/people/zhong-yun-75-63