

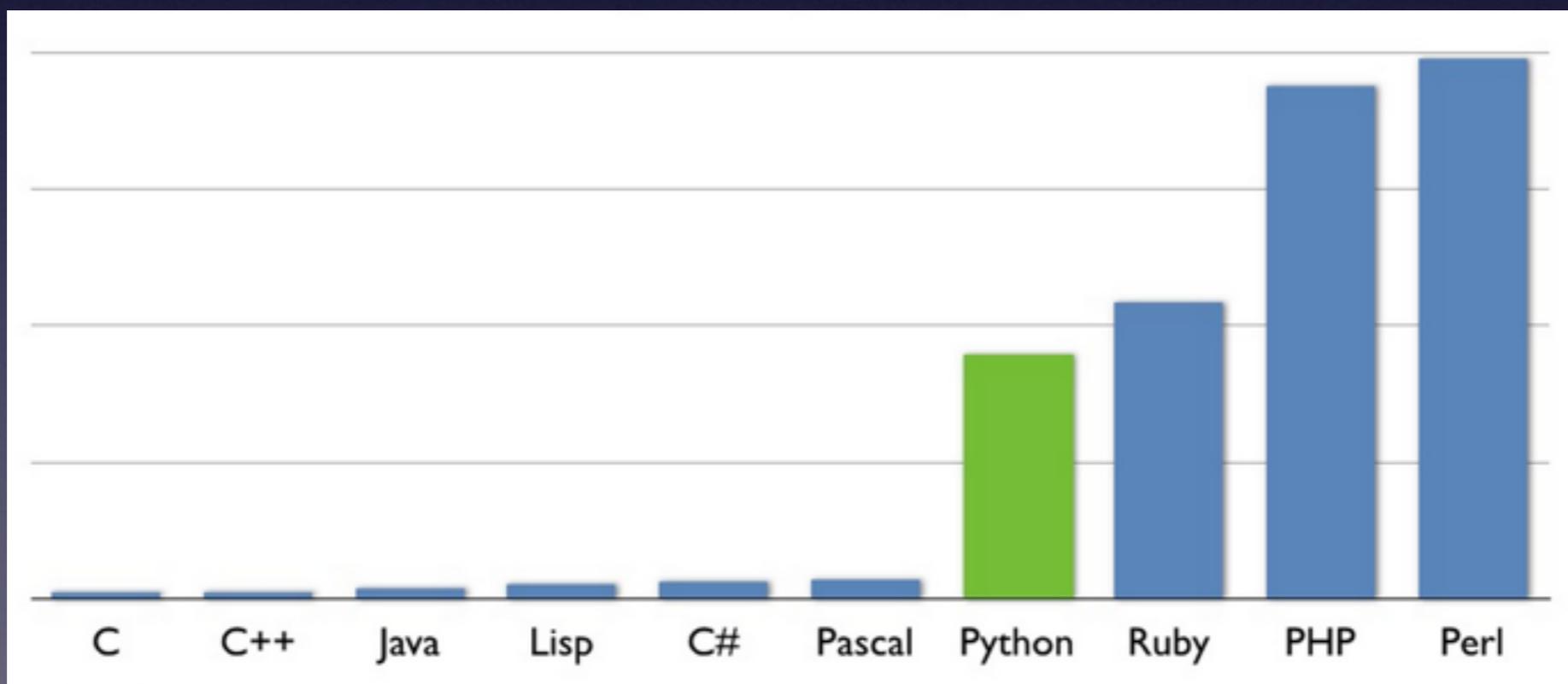


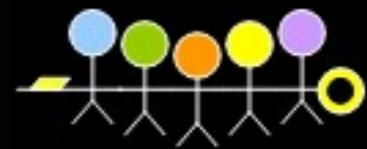
编写“高性能”Python代码



前言

- 为什么是“高性能”
- Python(CPython)的速度





主要内容

- 如何找出代码性能瓶颈
 - Python的一些性能tips
 - 其他和性能相关的内容杂谈
-  性能调优是一个非常庞大的主题，这儿无法涵盖所有方面，重要的是分享思路



找出瓶颈

- 优化的第一步：找出瓶颈
 - 永远不要去猜代码的性能瓶颈
 - 瓶颈经常会出现你意想不到的地方
- `json.dumps(data)` 会成为你的代码瓶颈吗？
 - 当data异常庞大时...



找出瓶颈

- 使用专业工具
 - profile/cprofile
 - line_profile(https://github.com/rkern/line_profiler)
 - ipython: %prun

```
4534 function calls (4521 primitive calls) in 0.073 seconds
```

```
Ordered by: internal time
```

```
ncalls  tottime  percall  cumtime  percall  filename:lineno(function)
   1     0.033    0.033    0.033    0.033  {method 'connect' of '_socket.socket' objects}
   9     0.031    0.003    0.031    0.003  {method 'recv' of '_socket.socket' objects}
   1     0.001    0.001    0.001    0.001  {_socket.getaddrinfo}
   3     0.001    0.000    0.001    0.000  {built-in method decompress}
```



比较方案

- 使用工具来比较方案的好坏
- import timeit
 - timeit.Timer(...).timeit()
- ipython: %timeit

```
In [13]: %timeit function2()  
100 loops, best of 3: 17.7 ms per loop
```



编写高性能Python代码

- 优化你的代码
- 使用更快的工具
- 超越CPython



优化你的代码

- 合理使用内置数据类型
 - 判断元素是否在其中

```
l = range(10000)

def function1():
    return 9000 in l
```

131 μ s

```
s = set(range(10000))

def function2(item):
    return item in s
```

190 ns

700x faster!

- $O(n)$ vs $O(1)$, 越大的list区别越大



算法上的优化会带来极大的性能差距



优化你的代码

- 懒惰计算
- range vs xrange

```
def function1():  
    for item in range(1000000):  
        pass
```

30.1 ms

```
def function2():  
    for item in xrange(1000000):  
        pass
```

17.3 ms

1.74x faster!

- yield / dict.iteritems() / itertools
- [x for x in I] vs (x for x in I)



空间分配上的优化通常会带来性能上的提升



优化你的代码

- 正则表达式
- 不要忘了编译 re.compile

```
log_string = 'date:2015-03-04 message=This is a log message'  
  
def function1():  
    return re.match(r'^date:([ ^ ]+) message=(.+)$', log_string).groups()
```

2.68 μ s

```
RE_LOG = re.compile(r'^date:([ ^ ]+) message=(.+)$', log_string)  
  
def function2():  
    return RE_LOG.match(log_string).groups()
```

1.69 μ s

1.58X faster!



优化你的代码

- 简单的字符串处理
 - 有时候简单的字符串处理会更快

```
def function3():  
    s_date, s_name = log_string.split(None, 1)  
    return s_date[5:], s_name[8:]
```

850 ns

2X faster!

- 正则表达式在处理很长的字符串时性能较慢
- 💡 强大的规则有时会带来性能的损失，试着简单思考



优化你的代码

- 列表迭代 vs for循环

```
l = range(10000)

def function2():
    result = []
    for i in l:
        if i % 2 == 0:
            result.append(i)
    return result
```

1.12 ms

```
l = range(10000)

def function1():
    return [i for i in l if i % 2 == 0]
```

819 μ s

1.36x faster

- 列表迭代接近map循环速度，是c级别的



在Python中，更简洁的代码经常也会是更优秀的代码



优化你的代码

- 不可忽视的函数调用开销

```
def f(x):  
    return x * 2  
  
def function1():  
    for i in xrange(10000):  
        v = f(i)
```

1.42 ms

```
def function2():  
    for i in xrange(10000):  
        v = i * 2
```

558 μ s

2.54x faster

- 很大的循环内，如果可以inline，尽量inline



了解你所使用的编程语言细节 (**import dis**)



追求性能的同时，不要忘记代码可读性



优化你的代码

- 局部变量的访问速度会比较快(import dis看看)
- `.sort(key=xxx)` vs `.sort(cmp=xxx)`
 - 尽量使用key，因为它拥有更好的性能。(为什么?)



使用更快的工具

- 充分利用优秀的内建模块
 - deque / bisect / heapq / collections ...

```
def function1():  
    l = []  
    for i in xrange(10000):  
        l.insert(0, i)
```

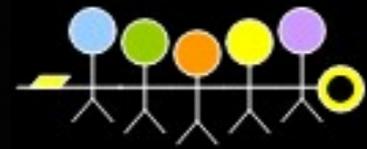
24.1 ms

```
from collections import deque  
  
def function2():  
    l = deque()  
    for i in xrange(10000):  
        l.appendleft(i)
```

1.02 ms

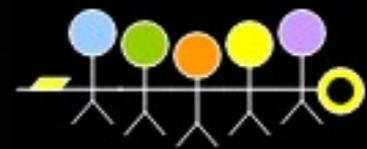
23.6X faster!

- heapq.nlargest(number, l)



使用更快的工具

- 尝试使用更快速的第三方模块
 - c实现的比pure python的要快
 - cPickle vs pickle / StringIO vs cStringIO ...
- 或者自己实现extension
 - CPython API
 - Cython: Python的语法, C的性能

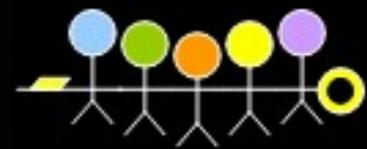


超越CPython

- pypy
 - JIT (Just-in-Time compiler)
- pyston by Dropbox(<https://github.com/dropbox/pyston>)
- ...

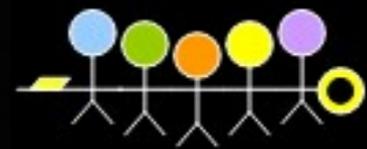


代码之外



并行的魅力

- 现在是多核的时代
- multithreading
 - 无处不在的GIL
 - 无法充分利用多核资源，适用范围有限
- 使用多进程
 - multiprocessing
 - concurrent.futures



异步IO

- 支撑高并发应用的关键
- 异步网络框架：Python的问题是选择太多
 - callback: twisted / tornado
 - coroutine: eventlet / greenlet / stackless python
- 挑选最适合你的



千万别忘了缓存

- 缓存无处不在
 - “最快的代码是什么都不干的代码”
 - 不变动的结果永远只计算一次
- 让人头疼的缓存过期问题
- 也许我们下次可以好好聊聊缓存



最后的话

- ***“Premature optimization is the root of all evil”***
-- DonaldKnuth



Q & A

欢迎任何问题!