Python 温故

赖勇浩
Email: lanphaday@126.com
http://blog.csdn.net/lanphaday

适用人群

- 有一定Python编程经验
- 期望自己的代码更Pythonic
- 认同代码的优雅比执行效率更重要的观点
- 但又时刻关注执行效率且以劣化代码为耻

内容提要

- 数值类型
- 字符串
- 基本数据结构
- 异常处理
- 避免劣化代码
- Python与设计模式
- 单元测试
- 性能剖分

数值类型(1)

- Plain integers
- Long integers
- Booleans
- Floating point numbers
- Complex numbers

数值类型(2)

- >>> i = 1234567890 * 1234567890
- >>> i
- 1524157875019052100L
- >>> i = 12345678901234567890
- >>> i
- 12345678901234567890L
- >>> i /= 12345678901234567890
- >>> i
- 1L

数值类型(3)

- >>> 99 / 2L
- 49L
- >>> 3.1415926 // 0.7
- 4.0
- >>> round(3.1415926 / 0.7)
- 4.0
- >>> 2 ** 8
- 256
- >>> pow(2, 8)
- 256

字符串(1)

- >>> s = 'string'
- >>> s = "string"
- >>> s = """
- ... str
- ... ing"""
- >>> s
- '\nstr\ning'
- >>> 'string' * 4
- 'stringstringstringstring'
- >>> 4 * 'string'
- 'stringstringstringstring'

字符串(2)

- capitalize( ), title(), istitle()
- center(width[, fillchar]) , ljust(width[, fillchar]), rjust(width[, fillchar]), zfill(width)
- strip([chars]), lstrip([chars]), rstrip([chars])
- count(sub[, start[, end]]), find(sub[, start[, end]]), index(sub[, start[, end]]), rfind(sub[, start[, end]]), rindex(sub[, start[, end]])
- startswith(prefix[, start[, end]]), endswith(suffix[, start[, end]])
- join(seq), split([sep [,maxsplit]]), rsplit([sep [,maxsplit]])

字符串(3)

- find()找不到时返回-1，index()抛出ValueError异常
- >>> s = 'Return a copy of the string converted to lowercase'
- >>> s.split('o', 2)
- ['Return a c', 'py ', 'f the string converted to lowercase']
- >>> s.rsplit('o', 2)
- ['Return a copy of the string converted t', ' l', 'wercase']
- >>> s.split('o') == s.rsplit('o')
- True

字符串(4)

- >>> "%d, %f, %o"%(10, 10.0, 16)
- '10, 10.000000, 20'
- >>> t = (10, 10.0, 16) # 不能是list
- >>> "%d, %f, %o"%t
- '10, 10.000000, 20'
- >>> t = "%d, %f, %o"
- >>> t%(10, 10.0, 16)
- '10, 10.000000, 20'
- >>> "%(num)d, %(float)f, %(oct)o" \ %{'num':10, 'float':10.0, 'oct':16}
- '10, 10.000000, 20'

基本数据结构(1)

- list
- tuple
- >>> (1,2) + (1,2)
- (1, 2, 1, 2)
- >>> (1,2) + [1,3]
- Traceback (most recent call last):
- File "<stdin>", line 1, in ?
- TypeError: can only concatenate tuple (not "list") to tuple

- list.sort([cmp[, key[, reverse]]])
- list.reverse()

- >>> l = ['lai', 'Lai', 'Yonghao', 'yonghao']
- >>> l.sort()
- >>> l
- ['Lai', 'Yonghao', 'lai', 'yonghao']
- >>> l.sort(key = str.lower)
- >>> l
- ['Lai', 'lai', 'Yonghao', 'yonghao']

基本数据结构(2)

- dict
- d.clear(), d.copy()
- d.items(), d.keys(), d.values()
- d.iteritems, d.iterkeys, d.itervalues()
- d.pop(k[, v]), d.popitem()

- d.update([o])
- >>> d = {}
- >>> d.update({"lai":"yonghao", 1:2, "computer":hash('computer')})
- >>> d
- {1: 2, 'computer': -375145325, 'lai': 'yonghao'}
- >>> d.update(((3,5),(4,7)))
- >>> d
- {3: 5, 1: 2, 'computer': -375145325, 4: 7, 'lai': 'yonghao'}
- >>> d.update(zip(range(10), range(10)))
- >>> d
- {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 'lai': 'yonghao', 6: 6, 7: 7, 8: 8, 9: 9, 'computer': -375145325, 5: 5}

- dict.fromkeys(seq[, value])
- >>> dict.fromkeys(range(10))
- {0: None, 1: None, 2: None, 3: None, 4: None, 5: None, 6: None, 7: None, 8: None, 9: None}
- >>> dict.fromkeys(range(10), "lai")
- {0: 'lai', 1: 'lai', 2: 'lai', 3: 'lai', 4: 'lai', 5: 'lai', 6: 'lai', 7: 'lai', 8: 'lai', 9: 'lai'}

- d.get(k[, v]), d.setdefault(k[, v])
- >>> d = {}
- >>> d.get(100)
- >>> print d.get(100)
- None
- >>> d.setdefault(100, 'lai')
- 'lai'
- >>> d.get(100)
- 'lai'
- >>> d
- {100: 'lai'}
- >>> d.setdefault(100, 'yonghao')
- 'lai'
- >>> d
- {100: 'lai'}

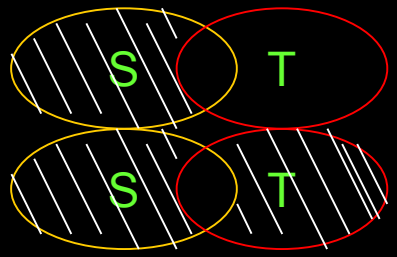
- 不得不说的defaultdict
- new in version 2.5
- defaultdict([default_factory[, ...]])
- __missing__(key)
- 当找不到key时由__getitem__调用
- 当default_factory为None，抛出KeyError
- 否则dict[key] = default_factory()， 并返回dict[key]，就像dict.setdefault()

- 自定义对象作为dict的key
- 覆盖__hash__
- def __hash__(self):pass
- 覆盖__eq__
- def __eq__(self, other):pass

基本数据结构(3)

- set, frozenset
- s.issubset(t)　　s <= t
- >>> s = set(range(5))
- >>> s <= set(range(10))
- True
- >>> s.issubset(range(10))
- >>> s <= range(10)
- Traceback (most recent call last):
- 　File "<stdin>", line 1, in ?
- TypeError: can only compare to a set
- s.issuperset(t)　　s >= t

- s.union(t) s | t
- s.intersection(t) s & t
- s.difference(t) s - t
- s.symmetric_difference(t) s ^ t
- s.update(t) s |= t
- s.intersection_update(t) s &= t
- s.difference_update(t) s -= t
- s.symmetric_difference_update(t) s ^= t



- s.add(x), s.remove(x), s.discard(x), s.pop(), s.clear()
- remove()在找不到x时抛出KeyError
- pop()在空集时抛出KeyError

基本数据结构(4)

- heapq
- 基于list的小顶堆，只提供维护heap的API
- heappush(heap, item), heappop(heap)
- heapify(x)
- heapreplace(heap, item)
- nlargest(n, iterable[, key])
- nsmallest(n, iterable[, key])

基本数据结构(5)

- bisect
- 基于list的有序序列，只提供维护的API
- bisect_left(list, item[, lo[, hi]]), bisect(...)
- 返回插入list索引
- insort_left(list, item[, lo[, hi]]), insort(...)
- 将元素插入list
- 皆假定list已经有序，操作后仍然有序

- >>> l = range(10)
- >>> bisect.insort(l, 5)
- >>> l
- [0, 1, 2, 3, 4, 5, 5, 6, 7, 8, 9]
- >>> l.remove(bisect.bisect(l, 4))
- >>> l
- [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

基本数据结构(6)

- from collections import deque
- deque([iterable])
- append(x), appendleft(x), extend(it), extendleft(it)
- pop(), popleft()
- clear(), remove(x)
- rotate(n),

- >>> de = deque(range(10))
- >>> de
- deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
- >>> de.extend(range(5))
- >>> de
- deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4])
- >>> de.extendleft(range(5))
- >>> de
- deque([4, 3, 2, 1, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4])

- remove(x), new in version 2.5
- x不存在，抛出ValueError异常
- >>> de = deque(range(10))
- >>> de
- deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
- >>> de.rotate(5)
- >>> de
- deque([5, 6, 7, 8, 9, 0, 1, 2, 3, 4])
- for i in xrange(5):de.appendleft(de.pop())

异常处理

- 精准地捕获异常
- try:
  do_something()
  except:
  pass
- 不要！！！

避免劣化代码(1)

- 避免劣化不是优化
- 时刻注意，时刻进行
- 代码要高效，但执行效率不是最重要的
- 简单、清晰、可重用
- 用算法提高执行效率而不是代码
- 用库解决问题
- 详读manual，关注每一版本的what's new

- 劣化代码：
  - if x != []: # 非空
  - do_something()
- 推荐代码：
  - if x:
  - do_something()
- 理由：
  - Python是非强类型语言
  - 更好的扩展性
  - 效率
  - >>> t = timeit.Timer("if x != []:pass", "x = []")
  - >>> t.timeit()
  - 0.24600005149841309
  - >>> t = timeit.Timer("if x:pass", "x = []")
  - >>> t.timeit()
  - 0.07499980926513671 9
  - >>> 0.24600005149841309 / 0.07499980926513671 9
  - 3.280009028139821 8

- 劣化代码：
  - for i in xrange(len(seq)):
  - foo(seq[i], i)
- 推荐代码：
  - for i, item in enumerate(seq):
  - foo(item, i)
- 理由：
  - 效率
  - 简洁

- 劣化代码：
  - for i in xrange(len(seq1)):
  - foo(seq1[i], seq2[i])
- 推荐代码：
  - for i, j in zip(seq1, seq2):
  - foo(i, j)
  - for i, j in itertools.izip(seq1, seq2):
  - foo(i, j)
- 理由：
  - 简洁
  - 效率（第二种）

- 劣化代码：
  - l = []
  - for i in seq:
  - 　l.append(foo(i))
- 推荐代码：
  - l = map(foo, seq)

  - for i in itertools.imap(foo, seq):
  - 　bar(i)
- 理由：
  - 简洁
  - 效率

- 劣化代码：
  - for i in xrange(len(seq)-1, -1, -1):
  - 　foo(seq[i])

  - l = seq[:]
  - l.reverse()
  - for i in l:
  - 　foo(i)
- 推荐代码：
  - for i in reversed(seq):
  - 　foo(i)
- 理由：
  - 效率
  - 简洁

- 劣化代码：
  - def foo(seq, bgn, end):
  - i = 0
  - while(bgn < end):
  - bar(seq[bgn], i)
  - bgn += 1
  - i += 1
  - def foo(seq, bgn, end):
  - tmp_seq = seq[bgn:end]
  - for i, item in enumerate(tmp_seq):
  - bar(item, i)
- 推荐代码：
  - def foo(seq, bgn, end):
  - for begin, i in itertools.izip(xrange(bgn, end), itertools.counter()):
  - bar(seq[begin], i)
- 理由：
  - 简洁
  - 效率

- 劣化代码：
  - for i in seq:
  - 　if pred(i):
  - 　　　foo(i)
- 推荐代码：
  - for i in itertools.ifilter(pred, seq):
  - 　foo(i)
- 理由：
  - filter
  - ifilterfalse
  - 效率

- 劣化代码：
  - s = ''
  - for i in seq:
  - s += chr(i)
- 推荐代码：
  - ''.join(map(chr, seq))
- 理由：
  - 简洁，略高效率
  - 追求更高效率
  - array.array('B', seq).tostring()
  - 47:30:12

避免劣化代码(2)

- import xx_module
- from xx_module import *
- from xx_module import foo,bar
- 名字空间污染
- 效率
- 适当选择

Python与设计模式

- 略

单元测试(1)

- unittest
- 单元测试是一种白盒测试方法
- 程序员应该主动
- 双扣和百变双扣大量使用单元测试
- 减轻QC的工作量，程序更稳定
- 新斗地主也在使用
- 推荐，项目质量的利器

单元测试(2)

- 只讲TestCase和TestSuite
- import unittest
- class DefaultWidgetSizeTestCase(unittest.TestCase):
- def runTest(self):
- widget = Widget('The widget')
- self.assertEqual(widget.size(), (50, 50), 'incorrect default size')
- if __name__ == '__main__':
- unittest.main()

单元测试(3)

- TestCase
- setUp(), setDown(), id()
- assert_(expr[, msg]), assert_equal(...)等
- failUnless(expr[, msg]), failUnleeEqual(...)

单元测试(4)

- TestSuite
- addTest(test)
- test是TestCase或者TestSuite对象
- addTests(tests)
- tests是一序列TestCase或者TestSuite

性能剖分(1)

- 做好单元测试有利于做性能剖分
- 不要太早
- 改进性能通常意味着损失可维护性
- 风险（进度、质量和士气）
- 毫无疑义，性能是很容易获取的

性能剖分(2)

- profile
- >>> def foo():
- ... l = range(10)
- ... l.sort()
- ... return l
- >>> import profile
- >>> profile.run('foo()')
- 6 function calls in 0.000 CPU seconds

- Ordered by: standard name

- ncalls tottime percall cumtime percall filename:lineno(function)
- 1 0.000 0.000 0.000 0.000 :0(range)
- 1 0.000 0.000 0.000 0.000 :0(setprofile)
- 1 0.000 0.000 0.000 0.000 :0(sort)
- 1 0.000 0.000 0.000 0.000 <stdin>:1(foo)
- 1 0.000 0.000 0.000 0.000 <string>:1(?)
- 1 0.000 0.000 0.000 0.000 profile:0(foo())
- 0 0.000 0.000 profile:0(profiler)

性能剖分(3)

- profile.run(command[, filename])
- ncalls　　被调用次数
- tottime　　函数体运行总时间
- percall　　平均一次调用时间
- cumtime　　调用总时间含子函数
- percall　　平均调用时间含子函数
- filename:lineno(function)

性能剖分(4)

- 输出漂亮的性能剖分报表
- class Stats(filename[, stream=sys.stdout[, ...]])
- sort_stats(key[, ...])
- print_stats([restriction, ...])
- print_callers([restriction, ...])
- print_callees([restiction, ...])
- add(filename[, ...])

性能剖分(5)

- 更好的剖分器cProfile， 用法基本上profile
- timeit——用以测定一小段代码的性能
- class Timer([stmt='pass' [, setup='pass' [, timer=<timer function>]]])
- timeit([number=1000000])
- repeat([repeat=3 [, number=1000000]])
- print_exc([file=None])

谢谢大家！