Python 编程艺术

享受高效无误且充满乐趣的编程

```
def hello():
        print 'hello world!'
```

什么样的函数返回 None？

没有消息就是最好的消息

对许多有经验的程序员来说
True / 1 并不是执行成功的意思

None 是最好的沉默

虽然 0 也不错，然而
在 Python 中默认的返回值是 None

函数不能既返回结果又返回错误

瘆人的设计
“如果 open() 返回 errno”

```
try:
    fileobj = open('filename')

# 虽然 open 能返回错误，但不能保证所有都捕获了啊
# 所以还是 catch 下比较保险？
except:
    raise # 暂且 raise 吧，鬼知道会是什么状况

if isinstance(fileobj, file):  # 正确返回 file 对象
    ... 正常的工作代码 ...

elif isinstance(fileobj, int): # 返回错误码，执行错误处理
    errno = fileobj
    raise IOError(errno, os.strerror(errno))

# 还会有其他状况？
# 既然已经有多种返回类型了，指不定还会不会节外生枝
# 这回还真无法预期了
else:
    raise IOError(-1, 'Unknown error')
```

其实 C 语言是一样一样的

因为 C 没有异常机制，而认为这种在 Python 中 return 一切的做法，显然是源于 C 语言的“优良作风”

—— 这是一种误解，其实 C 处理异常结果的思路和 Python 是完全一样的

Python 的标准接口

```
try:
    fobj = open('filename')
except IOError:
    ... 文件打开失败 . . .

... 只要没有异常，fobj 就一定是 file 对象 . . .

try:
    data = fobj.read()
except IOError:
    ... 读取失败 . . .

... 只要没有异常，data 就一定是 str 类型 . . .
```

最小惊讶原则

误区 1

函数返回值的可预见性，并不完全意味着仅返回一种返回值

误区 2

函数尽量返回一种返回值
并不意味着返回同一种“类型”
或基于同一种“基类”的数据

一个例子，虽然返回的都是 Tuple

```
if ... :
    return username, passwd
else:
    return name, age, gender
```

Duck Typing

文件

- 系统文件：open(...) 与 file(...)
  - 可以是任何介质，磁盘/内存文件/声卡/NFS/FUSE 等
- 字节流文件：StringIO(...)
- 套接字文件：socket.makefile(...)
- 管道：os.popen(...) 函数族
  - popen() 、popen2()、popen3() 、popen4()
- 标准输入输出：
  - sys.stdin 、sys.stdout 、sys.stderr
- 其他：gzip.open(..) 等

没有所谓基类（BaseFile）
却行为一致

将静态语言编程思想套用于动态语言是非常危险的

动态语言不需要预分配内存

危险的初始化

也没有所谓虚表

抛出 NotImplementedError 就安全了？

避免把问题演变成逻辑错误

解释器错误比运行期错误好

力图把错误提早到程序跑起来之前

程序崩溃比逻辑错误好

力图把可能存在的逻辑错误变成崩溃

尤其不要捕获 AttributeError

发现逻辑问题的最好机会

抛出 AttributeError 和 KeyError

而不是通过返回 None 来错失机会

了解各种 Python 异常的含义
尽量使用现有异常类型

Python 预设了丰富的异常类型
很多情况下设计额外的异常会和现有异常类型重合
发生混淆、影响系统稳定和增加维护成本

OSError & IOError

errno 在 Python 中的传播

坏消息是 IOError 并不是一种 OSError
好消息是 socket.error 可以用 IOError 获取

先 TypeError 而后 ValueError

int(None) / int('hello')

当你开始大量 raise
TypeError 和 ValueError
你就要注意了

raise 是用来补刀的

AssertError
合情合理的逻辑错误？

使用 assert 的场合

异常策略总结

不要做输入检测
不要编写保护性代码
不要容错，使程序崩溃
不要处理异常，尽量制造和抛出异常

一个脆弱的程序？

我们都知道没有完美的程序，出错在所难免，任何一个小异常都能让整个程序崩溃，对于一个已经开始运营的软件产品

—— 这是可以接受的吗？

单点复杂度

让专业的程序去做专业的事

SIMPLE IS BETTER

推论 1
检查只会让输入变得更不可靠

推论 2
越多的安全策略只会让程序变得更不安全

只有当错误不是错误

哪些异常是就地解决的？

同样的写法不同的命运

如何编写快速的 dict 检索逻辑

Python 异常的性能之迷

快却没有想象中那么快
慢却没有想象中那么慢

回到 Duck-Typing 这个主题

严重误区

Duck-Typing 是基于对象和
对象方法的吗？

如果把 for ... in obj 替换成 obj.__iter__()

Duck-Typing 是基于调用协议的

a, b = 'ab'

基于函数名和参数名的调用协议

调用者

（CPython、PyPy）

避免使用可修改对象作为函数的默认参数

显式总比隐藏好

```
login(username='xxx',
password='***')
```

暴露但不滥用

尽量消除私有和隐藏属性
尽可能暴露所有接口

不使用 _slots_

在所有层次上做到开放、透明

重复代码未必是坏事

放弃一部分封装
让用户明确知道自己在做什么

哪些情况是不需要封装的？

性能杀手

Python 的函数调用是重量级的

厚胶合层

横向复杂度与纵向复杂度

不要使用 super()

当用到 super() 时程序就已经失败了

Python / Unix 第一定律

程序复杂度只和行数相关

谢谢观赏

沈崴于上海 Python 聚会（五分钟）
2010 年 5 月 30 日

Python 编程艺术

凡事并无绝对
运用存乎一心