十分钟让你看懂Python

张长青

软件设计二部

目录

一、什么是Python？

1、Python的概念

Python（KK 英语发音：/ˈpaɪθən/,是一种**面向对象**、直译式计算机程序设计语言，由Guido van Rossum于1989年底发明，第一个公开发行版发行于1991年。Python语法**简捷而清晰**，具有丰富和强大的类库。它常被昵称为胶水语言，它能够很轻松的把用其他语言制作的各种模块（尤其是C/C++）轻松地联结在一起。常见的一种应用情形是，使用python**快速生成程序的原型**（有时甚至是程序的最终界面），然后对其中有特别要求的部分，用更合适的语言改写，比如3D游戏中的图形渲染模块，速度要求非常高，就可以用C++重写。

2、Python的简介

Python是一种**解释型、面向对象、动态数据类型**的高级程序设计语言。自从20世纪90年代初Python语言诞生至今，它逐渐被广泛应用于处理系统管理任务和Web编程。Python[1]已经成为最受欢迎的程序设计语言之一。**2011年1月，它被TIOBE编程语言排行榜评为2010年度语言。**自从2004年以后，python的使用率是呈线性增长[2]。

3、Python的历史

Python的创始人为Guido van Rossum。1989年圣诞节期间，在阿姆斯特丹，Guido为了打发圣诞节的无趣，决心开发一个新的**脚本解释程序**，做为ABC 语言的一种继承。之所以选中Python（大蟒蛇的意思）作为程序的名字，是因为他是一个叫Monty Python的**喜剧团体的爱好者**。

4、Python的设计风格

Python在设计上坚持了清晰划一的风格，这使得Python成为一门易读、易维护，并且被大量用户所欢迎的、用途广泛的语言。设计者开发时总的指导思想是，对于一个特定的问题，**只要有一种最好的方法来解决就好了**。这在由Tim Peters写的python格言（称为The Zen of Python）里面表述为：There should be one-- and preferably only one --obvious way to do it. 这正好和Perl语言（另一种功能类似的高级动态语言）的中心思想TMTOWTDI（There's More Than One Way To Do It）完全相反。

5、Python的设计定位

Python的**设计哲学是“优雅”、“明确”、“简单”**。因此，Perl语言中“总是有多种方法来做同一件事”的理念在Python开发者中通常是难以忍受的。Python开发者的哲学是**“用一种方法，最好是只有一种方法来做一件事”**。在设计Python语言时，如果面临多种选择，Python开发者一般会拒绝花俏的语法，而选择明确的没有或者很少有歧义的语法。由于这种设计观念的差异，Python源代码通常被认为比Perl具备更好的可读性，并且能够支撑大规模的软件开发。这些准则被称为Python格言。在Python解释器内运行import this可以获得完整的列表。

6、Python的面向对象

Python是**完全面向对象**的语言。函数、模块、数字、字符串都是对象，并且完全支持继承、重载、派生、多继承，有益于增强源代码的复用性。**Python支持重载运算符和动态类型。**相对于Lisp这种传统的函数式编程语言，Python对函数式设计只提供了有限的支持。有两个标准库(functools, itertools)提供了Haskell和Standard ML中久经考验的函数式程序设计工具。

7、Python的扩展

Python本身被设计为**可扩充的**,并非所有的特性和功能都集成到语言核心。Python提供了**丰富的API和工具**，以便程序员能够轻松地使用C语言、C++、Cython来编写扩充模块。Python编译器本身也可以被集成到其它需要脚本语言的程序内。因此，很多人还把Python作为一种**“胶水语言”**（glue language）使用,使用Python将其他语言编写的程序进行集成和封装。

8、Python的执行

Python在执行时，首先会将.py文件中的源代码编译成Python的byte code（字节码），然后再由**Python Virtual Machine（Python虚拟机）**来执行这些编译好的byte code。这种机制的基本思想跟Java，.NET是一致的。然而，Python Virtual Machine与Java或.NET的Virtual Machine不同的是，Python的Virtual Machine是一种更高级的Virtual Machine，和Java 或.NET相比，Python的Virtual Machine距离真实机器的距离更远。或者可以这么说，Python的Virtual Machine是一种**抽象层次更高的Virtual Machine。**

二、为什么要学Phython？

1、学习Python的几个目的

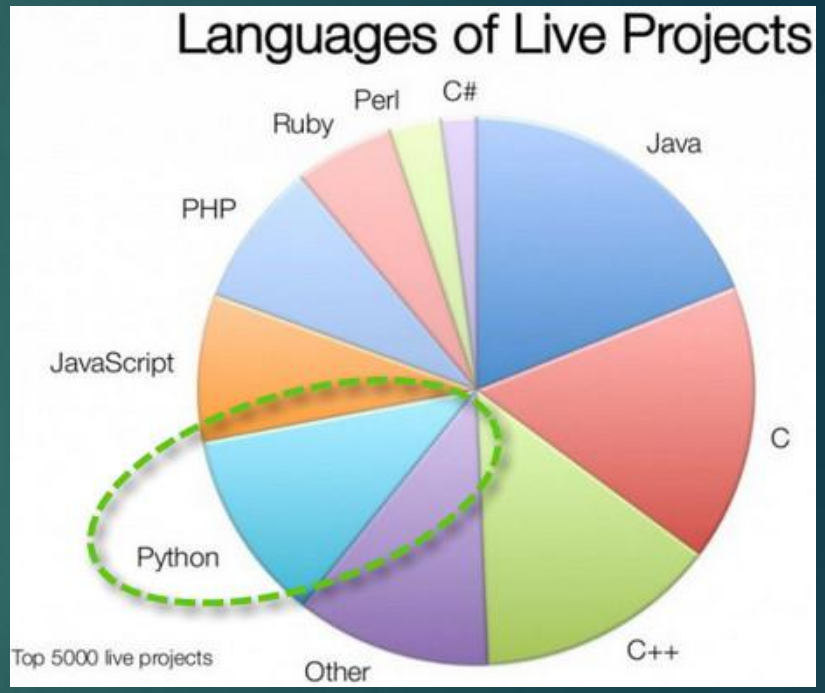
① 读懂脚本程序；

② 编写Phython程序；

③ 开阔编程思想；

2、更多的新项目采用PTYTHON

Black Duck Software, Inc. 曾发布一项内容十分周详的调查报告，题为”Open Source By The Numbers“（报告人是Rich Sands），该调查发现，当今最活跃的编程语言是C/C++，跟随其后的是Java，**Python**，JavaScript等编程语言，如图所示。该调查报告的最后结论是：”New live projects trending towards Python, PHP, JavaScript and away from C-family languages“，意思是说，**新的活跃研究项目都倾向（trending towards）使用Python**，PHP与JavaScript编程，而远离（away from）C编程语言大家族。



3、Python 八荣八耻

以动手实践为荣，以只看不练为耻;
以打印日志为荣，以单步跟踪为耻;
以空格缩进为荣，以制表缩进为耻;
以单元测试为荣，以人工测试为耻;
以模块复用为荣，以复制粘贴为耻;
以多态应用为荣，以分支判断为耻;
以Pythonic为荣，以冗余拖沓为耻;
以总结分享为荣，以跪求其解为耻。

三、PHYTHON的基本语法

1、语言基础

Python是一门**解释型**语言，因为不需要编译和链接的时间，它可以帮你省下一些开发时间。解释器可以**交互式使用**，这样就可以很方便的测试语言中的各种功能，以便于编写发布用的程序，或者进行自下而上的开发。还可以当它是一个**随手可用的计算器**。

1、语言基础

Python可以写出很紧凑和可读性很强的程序。用Python写的程序通常比同样的C或C++程序要短得多，这是因为以下几个原因：

① **高级数据结构**使你可以在一个单独的语句中表达出很复杂的操作；

② 语句的组织**依赖于缩进**，而不是begin/end块；

③ **不需要变量或参数声明。**

2、Python解释器

2.1 调用解释器

通常Python的解释器被安装在目标机器的 /usr/local/bin/python 目录下；把 /usr/local/bin 目录放进你的UNIXShell 的搜索路径里，确保它可以通过输入 python来启动。输入一个文件结束符（UNIX上是**Ctrl+D**，Windwos上是**Ctrl+Z**）解释器会以0值退出（就是说，没有什么错误，正常退出——译者）

2、Python解释器

2.2 参数传递

调用解释器时，脚本名和附加参数之传入一个名为sys.argv的字符串列表。没有脚本和参数时，它至少也有一个元素：sys.argv[0]此时为空字符串。脚本名指定为‘-’（表示标准输入）时，sys.argv[0]被设置为‘-’，使用**-c** 指令时，sys.argv[0]被设定为‘-c’。 **-c** 命令之后的参数不会被 Python 解释器的选项处理机制所截获，而是留在 sys.argv 中，供脚本命令操作。

2、Python解释器

2.3 交互模式

从tty读取命令时，我们称解释器工作于交互模式。这种模式下它根据主提示符来执行，主提示符通常标识为三个大于号（“>>>”）；继续的部分被称为从属提示符，由三个点标识（“...”）。在第一行之前，解释器打印欢迎信息、版本号和授权提示:

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

2、Python解释器

2.4 错误处理

有错误发生时，解释器打印一个错误信息和栈跟踪（监视）器？。交互模式下，它返回主提示符，如果从文件输入执行，它在打印栈跟踪器后以非零状态退出。（异常可以由try语句中的except子句来控制，这样就不会出现上文中的错误信息）有一些非常致命的错误会导致非零状态下退出，这由通常由内部矛盾和内存溢出造成。所有的**错误信息都写入标准错误流**；命令中执行的普通输出写入标准输出。在主提示符或附属提示符输入中断符（通常是Control-C or DEL）就会取消当前输入，回到主命令行。

2、Python解释器

2.5 执行Python脚本

Python脚本可以像Shell脚本那样直接执行，只要在脚本文件开头写一行命令，指定文件和模式：

```
#! /usr/bin/env python
```

“#!”必须是文件的前两个字符，目的是将程序路径通知解释器，在某些平台上，第一行必须以Unix风格的行结束符（“\n”）结束，不能用Mac（“\r”）或Windows（“\r\n”）的结束符。注意，“#”是Python中是行注释的起始符。

脚本可以通过 chmod 命令指定执行模式和许可权。

```
$ chmod +x myscript.py
```

2、使用Python解释器

2.6 输入输出标志

区分输入和输出的方法是看是否有**提示符（“>>>”和“..”）**，想要重复这些例子的话，你就要在提示符显示后输入所有的一切；**没有以提示符开始的行，是解释器输出的信息。**需要注意的是示例中的从属提示符用于多行命令的结束，它表示你需要输入一个空行。

Python中的**注释以符号“#”起始，**一直到当前行的结尾。注释可能出现在一行的开始，也可能跟在空格或程序代码之后，但不会出现在字符串中，字符串中的#号只代表#号。

3、Python基本类型

3.1 数值

解释器的行为就像是一个计算器。你可以向它输入一个表达式，它会返回结果。表达式的语法简明易懂：+，-，*，/和大多数语言中的用法一样（比如C或Pascal），括号用于分组。例如：

```
>>> 2+2
4
>>> (50-5*6)/4
5
```

3、Python基本类型

3.1 数值

等号（“=”）用于给变量赋值，被分配的值是只读的。

```
>>> width = 20
>>> height = 5*9
>>> width * height
900
```

同一个值可以同时赋给几个变量：

```
>>> x = y = z = 0
>>> x
0
```

3、Python基本类型

3.2 字符串（string）

Python还可以通过几种不同的方法操作字符串。

字符串用单引号或双引号标识：

```
>>> 'spam eggs'
'spam eggs'
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

3、Python基本类型

3.3 None类型

None类型是一个特殊的常量，表示出错。

逻辑表达式：PYTHON中没有boolean类型，除了0以外，其他都是真。但是Python中‘假’有很多种，包括None，0,0.0，“”(空字符串), [](空list)，()(空元祖)，{}(空字典)。

Phython中的逻辑运算符有： and , or , not

3、Python基本类型

3.4 链表(列表list)

Python 已经有了几个复合数据类型，用于组织其它的值。最通用的是链表，它写为中括之间用逗号分隔的一列值（子项），链表的子项不一定是同一类型的值。

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
```

3、Python基本类型

3.4 链表(列表list)

链表也以零开始，可以被切片，联接，等等：

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a[0]
'spam‘
>>> a[3]
1234
```

```
>>> a[-2]
100
>>> a[1:-1]
['eggs', 100]
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
```

3、Python基本类型

3.5 元祖（Tuples）

一个元组由数个逗号分隔的值组成，元组在输出时总是有括号的，以便于正确表达嵌套结构。在输入时可能有或没有括号都可以，不过经常括号都是必须的（如果元组是一个更大的表达式的一部分）。

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
```

```
>>>u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

3、Python基本类型

3.6 字典（Dictionaries）

字典以关键字为索引，关键字可以是任意不可变类型，通常用字符串或数值。理解字典的最佳方式是把它看做无序的关键字：值对（key:value pairs）集合，关键字必须是互不相同的（在同一个字典之内）。一对大括号创建一个空的字典：{}。

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

3、Python基本类型

3.7 序列（Sequence）

序列包括string，list，tuple，它们都有一些通用的操作。

① in 判断某个object是不是在一个sequence中；

② len(seq)得到sequence的长度；

③ 通过下标获取其中的元素,seq[i]；

④ 通过带冒号的下标获取子sequence，seq[start:end]；

⑤ 用+表示连接两个sequence；

⑥ 用*表示重复一个sequence，“a”*3表示“aaa”；

4、Python程序流程

4.1 if语句——语法格式

```
if <expr1>: <one_line_statement>

if <expr1>:
    <statement-block>

if <expr1>:
    <statement-block>
else:
    <statement-block>
```

```
if <expr1>:
    <statement-block>
elif <expr2>:
    <statement-block>
elif <expr3>:
    <statement-block>
...
else:
    <statement-block>
```

4、Python程序流程

4.1 if语句——使用说明

① if后面的表达式可以是任何表达式，除了None，0，“”(空字符串)，[](空list)，{}(空Dictionary)，()(空Tuple)以外，其他都是真。

② **表达式以冒号分割。**

③ 若是一行的简单表达式，可以直接跟在冒号后面。

④ 若是多行的表达式，就要使用缩进的方式，表示一组语句。**Pyhon认为同样缩进长度的语句是一组。**

⑤ Python中没有switch语句，使用elif来完成这个功能。

⑥ else表示条件不满足时应该执行的语句，别忘了后面的冒号。

4、Python程序流程

4.2 for循环语句——语法格式

```
for x in <sequence>:
    <statement-block>
else:
    <else-block>
```

4、Python程序流程

4.2 for循环语句——使用说明

① sequence表示任何string，tuple，dictionary。

② break语句可以强制退出循环。

③ continue语句可以继续执行下一次循环。

④ else语句是可有可无的。若有，则表示如果每个序列中的元素都循环到了，那么就执行else-block。

⑤ else语句可以理解为，如果循环没有被break语句强行中断，或者说循环正常结束后，就会执行else中的语句。

4、Python程序流程

4.3 while循环语句——语法格式

```
while <expr1>:
    <block>
else:
    <else-block>
```

4、Python程序流程

4.3 while循环语句——使用说明

① 只要expr1是真，那么就一直执行block中的语句。

② 如果没有break跳出循环，也就是expr1为假的时候，执行else block中的语句。

4、Python程序流程

4.4 break 和continue 语句，以及循环中的else 子句

① break语句和C中的类似，用于跳出最近的一级for或while循环。

② continue 语句是从C中借鉴来的，它表示循环继续执行下一次迭代。

③ 循环可以有一个else子句;它在循环迭代完整个列表（对于for）或执行条件为false（对于while）时执行，但循环被break中止的情况下不会执行。

4、Python程序流程

4.5 pass 语句

pass 语句什么也不做。它用于那些语法上必须要有什么语句，但程序上什么也不要做的场合，例如：

```
>>> while True:
...       pass        # Busy-wait for keyboard interrupt...
```

5、Python函数

5.1 函数定义

```
def <function_name> ( <parameters_list> ):
    <code block>
```

说明：

① function_name是函数名称，冒号用来分割函数的内容。

② Parameters_list是参数列表。参数也没有类型，可以传递任何类型的值给函数，由函数的内容定义函数的接口。如果传递的参数类型不是函数想要的，那么函数可以抛出异常。

③ code block是和if，while一样的代码，应该有相同的缩进。

④ 函数没有返回值类型，**return可以返回任意类型。**

5、Python函数

5.2 函数参数默认值

最有用的形式是给一个或多个参数指定默认值。这样创建的函数可以在调用时使用更少的参数。

```
def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'): return 1
        if ok in ('n', 'no', 'nop', 'nope'): return 0
```

这个函数还可以用以下的方式调用：ask_ok('Do you really want to quit?')，或者像这样：ask_ok('OK to overwrite the file?',2)。

5、Python函数

5.3 函数参数关键字

函数可以通过参数关键字的形式来调用，形如“keyword = value”。例如，以下的函数：

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action)
    print("if you put", voltage, "Volts through it.")
    print("-- Lovely plumage, the", type print "-- It's", state, "!")
```

可以用以下的任一方法调用：

```
parrot(1000)
parrot(action = 'VOOOOOOM', voltage = 1000000)
parrot('a thousand', state = 'pushing up the daisies')
```

5、Python函数

5.3 函数参数关键字

① 参数列表中的每一个关键字都必须来自于形式参数，每个参数都有对应的关键字。

② 形式参数有没有默认值并不重要。

③ 实际参数不能一次赋多个值。

④ 形式参数不能在同一次调用中同时使用位置和关键字绑定值。

5、Python函数

5.4 函数可变参数列表

一个最不常用的选择是可以让函数调用可变个数的参数。这些参数被包装进一个拓扑。在这些可变个数的参数之前，可以有零到多个普通的参数：

```
def fprintf(file, format, *args):
    file.write(format % args)
```

5、Python函数

5.5 嵌套函数

这种机制提供了一个函数范围的概念，某些函数只在某些函数的内容才能看到。如果需要公开的话，可以通过返回值返回内部函数。

```
>>> def outfun(a,b):
...         def innerfun(x,y):
...         return x+y
...       return innerfun(a,b)
```

5、Python函数

5.6 lambda关键字

通过lambda关键字，可以创建很小的匿名函数。这里有一个函数返回它的两个参数的和：“lambda a, b: a+b”。Lambda 形式可以用于任何需要的函数对象。出于语法限制，它们只能有一个单独的表达式。语义上讲，它们只是普通函数定义中的一个语法技巧。类似于嵌套函数定义，lambda形式可以从包含范围内引用变量：

```
>>> def make_incrementor(n):
        return lambda x: x + n...
>>> f = make_incrementor(42)
>>> f(0)
42
```

```
>>> f = lambda a,b: a+b
>>> f(1,2)
3
>>> f("abc","def")
'abcdef'
```

5、Python函数

5.7 文档字符串(DocString函数描述)

第一行应该是关于对象用途的简介。简短起见，不用明确的陈述对象名或类型，因为它们可以从别的途径了解到。这一行应该以大写字母开头，以句号结尾。如果文档字符串有多行，第二行应该空出来，与接下来的详细描述明确分隔。接下来的文档应该有一或多段描述对象的调用约定、边界效应等。

5、Python函数

5.7 文档字符串(DocString函数描述)

```
>>> def my_function():
... """Do nothing, but document it
....
....     No, really, it doesn't do anything
...."""
... pass
...
>>> print my_function.__doc__
this function do nothing, just demostrate the use of the doc string.
```

6、Python类

6.1 类定义

定义一个类的时候，会创建一个新的命名空间，将其作为局部作用域使用——因此，所以对局部变量的赋值都引入新的命名空间。特别是函数定义将新函数的命名绑定于此。

```
class ClassName:
    <statement-1>
    ...
    <statement-N>
```

6、Python类

6.2 类对象引用

类对象支持两种操作：属性引用和实例化。

属性引用使用和Python中所有的属性引用一样的标准语法：obj.name。类对象创建后，类命名空间中所有的命名都是有效属性名。所以如果类定义是这样：

```
class MyClass:
    "A simple example class"
    i = 12345
    def f(self):
        return 'hello world‘
```

那么 MyClass.i 和 MyClass.f 是有效的属性引用，分别返回一个整数和一个方法对象。

6、Python类

6.3 类的实例化

类的实例化使用函数符号。只要将类对象看作是一个返回新的类实例的无参数函数即可。例如（假设沿用前面的类）：

```
>>>x = MyClass()
```

以上创建了一个新的类实例并将该对象赋给局部变量x。这个实例化操作创建了一个空的对象。创建为有初始状态的类可以通过定义一个名为__init__() 的特殊方法来实现，像下面这样：

```
def __init__(self):
    self.data = []
```

类定义了 __init__() 方法的话，类的实例化操作会自动为新创建的类实例调用 __init__() 方法。

6、Python类

6.4 类的继承

如果一种语言不支持继承就，“类”就没有什么意义。派生类的定义如下所示：

```
class DerivedClassName(BaseClassName):
    <statement-1>
    ...
    <statement-N>
```

命名 BaseClassName （示例中的基类名）必须与派生类定义在一个作用域内。

6、Python类

6.5 类的多继承

Python同样有限的支持多继承形式。多继承的类定义形如下例：

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    ...
    <statement-N>
```

这里唯一需要解释的语义是解析类属性的规则。顺序是深度优先，从左到右。

7、Python模块和包

7.1 模块(Module)

一个module不过是一些函数，class放在一个*.py文件中，例如test.py：

```
"""
this only is a very simple test module
"""
age=0 # a sample attribute
def sayHello(): # a sample function in a module
print "Hello"
if __name__ == "__main__"
sayHello()
```

7、Python模块和包

7.2 包(Package)

Package是一组module的集合，用以下方法创建一个package。

① 在当前目录下创建一个目录testpackage；

② 在testpackage目录中创建一个空文件__init__.py；

③ 在testpackage目录中创建一个testmodule.py文件，里面含有一些代码。

```
>>> import testpackage.testmodule
>>> testpackage.testmodule.sayHello()
Hello
```

Package是一种组织module的方法，提供了一个name space，防止发生名称冲突。

7、Python模块和包

7.3 命名空间(name space)

命名空间是从命名到对象的映射。以下有一些命名空间的例子：内置命名（像 abs()这样的函数，以及内置异常名）集，模块中的全局命名，函数调用中的局部命名。某种意义上讲对象的属性集也是一个命名空间。关于命名空间需要了解的一件很重要的事就是不同命名空间中的命名没有任何联系，例如两个不同的模块可能都会定义一个名为“maximize”的函数而不会发生混淆——用户必须以模块名为前缀来引用它们。

不同的命名空间在不同的时刻创建，有不同的生存期。包含内置命名的命名空间在Python解释器启动时创建，会一直保留，不被删除。

四、如何阅读Phython程序？

如何阅读python程序？

1、熟悉基本语法；

2、熟悉面向过程和面向对象编程逻辑；

3、熟悉Python基本类库；

4、多写、多练、多调试，才能有感觉。

写在后面的话

好吧，我承认，使用《十分钟让你看懂Python》这个标题，只是一个噱头。如果你之前不了解Python，看完这个PPT，顶多可以让你有个入门的感觉，离真正看懂还差得远。但这是一个很好的开始。可以让你真实地感受到，除了Java，除了C/C++，还有一门很有意思的语言可以使用。

如果你只有一把锤子，你会把所有的问题都看出钉子。但是很显然，锤子并不是解决所有问题的最佳方式。所以，赶快丰富一下你的工具箱吧，多学会一门语言，就多了一种工具，多了一种思维方式的改变。你会发现，你能比以前看到更多，想到更多。

Thank you Very much!

Email: adamzhang@126.com