

## <<Python源码剖析>>

### 图书基本信息

书名：<<Python源码剖析>>

13位ISBN编号：9787121068744

10位ISBN编号：7121068745

出版时间：2000-1

出版时间：电子工业出版社

作者：陈儒

页数：480

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## <<Python源码剖析>>

### 内容概要

作为主流的动态语言，Python不仅简单易学、移植性好，而且拥有强大丰富的库的支持。此外，Python强大的可扩展性，让开发人员既可以非常容易地利用C/C++编写Python的扩展模块，还能将Python嵌入到C/C++程序中，为自己的系统添加动态扩展和动态编程的能力。

为了更好地利用Python语言，无论是使用Python语言本身，还是将Python与C/C++交互使用，深刻理解Python的运行原理都是非常重要的。

本书以CPython为研究对象，在C代码一级，深入细致地剖析了Python的实现。

书中不仅包括了对大量Python内置对象的剖析，更将大量的篇幅用于对Python虚拟机及Python高级特性的剖析。

通过此书，读者能够透彻地理解Python中的一般表达式、控制结构、异常机制、类机制、多线程机制、模块的动态加载机制、内存管理机制等核心技术的运行原理，同时，本书所揭示的动态语言的核心技术对于理解其他动态语言，如Javascript、Ruby等也有较大的参考价值。

本书适合于Python程序员、动态语言爱好者、C程序员阅读。

## &lt;&lt;Python源码剖析&gt;&gt;

## 书籍目录

第0章 PYTHON源码剖析--编译PYTHON0.1 PYTHON总体架构0.2 PYTHON源代码的组织0.3 WINDOWS环境下编译PYTHON0.4 UNIX/LINUX环境下编译PYTHON0.5 修改PYTHON源代码0.6 通往PYTHON之路0.7 一些注意事项第1部分 PYTHON内建对象第1章 PYTHON对象初探1.1 PYTHON内的对象1.1.1 对象机制的基石——PyObject1.1.2 定长对象和变长对象1.2 类型对象1.2.1 对象的创建1.2.2 对象的行为1.2.3 类型的类型1.3 PYTHON对象的多态性1.4 引用计数1.5 PYTHON对象的分类第2章 PYTHON中的整数对象2.1 初识PYINTOBJECT对象2.2 PYINTOBJECT对象的创建和维护2.2.1 对象创建的3种途径2.2.2 小整数对象2.2.3 大整数对象2.2.4 添加和删除2.2.5 小整数对象池的初始化2.3 HACK PYINTOBJECT第3章 PYTHON中的字符串对象3.1 PYSTRINGOBJECT与PYSTRING\_TYPE3.2 创建PYSTRINGOBJECT对象3.3 字符串对象的INTERN机制3.4 字符缓冲池3.5 PYSTRINGOBJECT效率相关问题3.6 HACK PYSTRINGOBJECT第4章 PYTHON中的LIST对象4.1 PYLISTOBJECT对象4.2 PYLISTOBJECT对象的创建与维护4.2.1 创建对象4.2.2 设置元素4.2.3 插入元素4.2.4 删除元素4.3 PYLISTOBJECT对象缓冲池4.4 HACK PYLISTOBJECT第5章 PYTHON中的DICT对象5.1 散列表概述5.2 PYDICTOBJECT5.2.1 关联容器的entry5.2.2 关联容器的实现5.3 PYDICTOBJECT的创建和维护5.3.1 PyDictObject对象创建5.3.2 PyDictObject中的元素搜索5.3.3 插入与删除5.3.4 操作示例5.4 PYDICTOBJECT对象缓冲池5.5 HACK PYDICTOBJECT第6章 最简单的PYTHON模拟——SMALL PYTHON6.1 SMALL PYTHON6.2 对象机制6.3 解释过程6.4 交互式环境第2部分 PYTHON虚拟机第7章 PYTHON的编译结果--CODE对象与PYC文件7.1 PYTHON程序的执行过程7.2 PYTHON编译器的编译结果--PYCODEOBJECT对象7.2.1 PyCodeObject对象与pyc文件7.2.2 Python源码中的PyCodeObject7.2.3 pyc文件7.2.4 在Python中访问PyCodeObject对象7.3 PYC文件的生成7.3.1 创建pyc文件的具体过程7.3.2 向pyc文件写入字符串7.3.3 一个PyCodeObject, 多个PyCodeObject7.4 PYTHON的字节码7.5 解析PYC文件第8章 PYTHON虚拟机框架8.1 PYTHON虚拟机中的执行环境8.1.1 Python源码中的PyFrameObject8.1.2 PyFrameObject中的动态内存空间8.1.3 在Python中访问PyFrameObject对象8.2 名字、作用域和名字空间8.2.1 Python程序的基础结构——module8.2.2 约束与名字空间8.2.3 作用域与名字空间8.3 PYTHON虚拟机的运行框架8.4 PYTHON运行时环境初探第9章 PYTHON虚拟机中的一般表达式9.1 简单内建对象的创建9.2 复杂内建对象的创建9.3 其他一般表达式9.3.1 符号搜索9.3.2 数值运算9.3.3 信息输出第10章 PYTHON虚拟机中的控制流10.1 PYTHON虚拟机中的IF控制流10.1.1 研究对象--if\_control.py10.1.2 比较操作10.1.3 指令跳跃10.2 PYTHON虚拟机中的FOR循环控制流10.2.1 研究对象——for\_control.py10.2.2 循环控制结构的初始化10.2.3 迭代控制10.2.4 终止迭代10.3 PYTHON虚拟机中的WHILE循环控制结构10.3.1 研究对象——while\_control.py10.3.2 循环终止10.3.3 循环的正常运转10.3.4 循环流程改变指令之continue10.3.5 循环流程改变指令之break10.4 PYTHON虚拟机中的异常控制流10.4.1 Python中的异常机制10.4.2 Python中的异常控制语义结构第11章 PYTHON虚拟机中的函数机制11.1 PYFUNCTIONOBJECT对象11.2 无参函数调用11.2.1 函数对象的创建11.2.2 函数调用11.3 函数执行时的名字空间11.4 函数参数的实现11.4.1 参数类别11.4.2 位置参数的传递11.4.3 位置参数的访问11.4.4 位置参数的默认值11.4.5 扩展位置参数和扩展键参数11.5 函数中局部变量的访问11.6 嵌套函数、闭包与DECORATOR11.6.1 实现闭包的基石11.6.2 闭包的实现11.6.3 Decorator第12章 PYTHON虚拟机中的类机制12.1 PYTHON中的对象模型12.1.1 对象间的关系12.1.2 和12.2 从TYPE对象到CLASS对象12.2.1 处理基类和type信息12.2.2 处理基类列表12.2.3 填充tp\_dict12.3 用户自定义CLASS12.3.1 创建class对象12.4 从CLASS对象到INSTANCE对象12.5 访问INSTANCE对象中的属性12.5.1 instance对象中的\_\_dict\_\_12.5.2 再论descriptor12.5.3 函数变身12.5.4 无参函数的调用12.5.5 带参函数的调用12.5.6 Bound Method和Unbound Method12.6 千变万化的DESCRIPTOR第3部分 PYTHON高级话题第13章 PYTHON运行环境初始化13.1 线程环境初始化13.1.1 线程模型回顾13.1.2 初始化线程环境13.2 系统MODULE初始化13.2.1 创建\_\_builtin\_\_ module13.2.2 创建sys module13.2.3 创建\_\_main\_\_ module13.2.4 设置site-specific的module的搜索路径13.3 激活PYTHON虚拟机13.3.1 交互式运行方式13.3.2 脚本文件

## &lt;&lt;Python源码剖析&gt;&gt;

运行方式13.3.3 启动虚拟机13.3.4 名字空间第14章 PYTHON模块的动态加载机制14.1 IMPORT前奏曲14.2 PYTHON中IMPORT机制的黑盒探测14.2.1 标准import14.2.2 嵌套import14.2.3 import package14.2.4 from与import14.2.5 符号重命名14.2.6 符号的销毁与重载14.3 IMPORT机制的实现14.3.1 解析module/package树状结构14.3.2 加载module/pakcage14.3.3 from与import14.4 PYTHON中的IMPORT操作14.4.1 import module14.4.2 import package14.4.3 from & import14.4.4 import & as14.4.5 reload14.4.6 内建module : imp14.5 与MODULE有关的名字空间问题第15章 PYTHON多线程机制15.1 GIL与线程调度15.2 初见PYTHON THREAD15.3 PYTHON线程的创建15.3.1 建立多线程环境15.3.2 创建线程15.4 PYTHON线程的调度15.4.1 标准调度15.4.2 阻塞调度15.5 PYTHON子线程的销毁15.6 PYTHON线程的用户级互斥与同步15.6.1 用户级互斥与同步15.6.2 Lock对象15.7 高级线程库——THREADING15.7.1 Threading Module概述15.7.2 Threading的线程同步工具15.7.3 Threading中的Thread第16章 PYTHON的内存管理机制16.1 内存管理架构16.2 小块空间的内存池16.2.1 Block16.2.2 Pool16.2.3 arena16.2.4 内存池16.3 循环引用的垃圾收集16.3.1 引用计数与垃圾收集16.3.2 三色标记模型16.4 PYTHON中的垃圾收集16.4.1 可收集对象链表16.4.2 分代的垃圾收集16.4.3 Python中的标记-清除方法16.4.4 垃圾收集全景16.4.5 Python中的gc模块16.4.6 总结

## &lt;&lt;Python源码剖析&gt;&gt;

## 章节摘录

第1章 Python对象初探 对象是Python中最核心的一个概念，在Python的世界中，一切都是对象，一个整数是一个对象，一个字符串也是一个对象。

更为奇妙的是，类型也是一种对象，整数类型是一个对象，字符串类型也是一个对象。

换句话说，面向对象理论中的“类”和“对象”这两个概念在Python中都是通过Python内的对象来实现的。

在Python中，已经预先定义了一些类型对象，比如int类型、string类型、dict类型等，这些我们称之为内建类型对象。

这些类型对象实现了面向对象中“类”的概念；这些内建类型对象通过“实例化”，可以创建内建类型对象的实例对象，比如int对象、string对象、dict对象。

类似的，这些实例对象可以视为面向对象理论中“对象”这个概念在Python中的体现。

同时，Python还允许程序员通过class A (object) 这样的表达式自己定义类型对象。

基于这些类型对象，同样可以进行“实例化”的操作，创建的对象称为“实例对象”。

Python中不光有着这些千差万别的对象，这些对象之间还存在着各种复杂的关系，从而构成了我们称之为“类型系统”或“对象体系”的东西。

Python中的对象体系是一个庞大而复杂的体系，如果说在本书的第一章我就试图将这个体系阐释清楚，这只能说明我是个疯子。

在本章中，我们的重点将放在了解对象在Python内部是如何表示的，更确切地说，因为Python是由C实现的，所以我们首先要弄清楚的一个问题就是：对象，这个神奇的东西，在C的层面，究竟长得是个什么模样，究竟是三头六臂，还是烈焰红唇。

除此之外，我们还将了解到类型对象在C的层面是如何实现的，并初步认识类型对象的作用及它与实例对象的关系。

总之，本章对Python对象体系的介绍力求简洁，但是并不肤浅，有的地方甚至会相当深入。

因此，在本章的阅读中，如果有什么疑难的地方，没有关系，先放下，只要有一个直观的感觉就可以了，这并不妨碍你阅读接下来的内容。

本章的目的是为能够顺利而快速地进入对内建对象的剖析打下必要的基础，至于对Python对象体系的详细剖析，会在第2部分的最后一章中介绍到。

只有到了那个时候，我们才有足够的能力将这个体系看个明白。

1.1 Python内的对象 从1989年Guido在圣诞节揭开Python的大幕开始，一直到现在，Python经历了一次一次的升级，但是其实现语言一直都是ANSI C。

我们知道，C并不是一个面向对象的语言，那么在Python中，它的对象机制是如何实现的呢？

对于人的思维来说，对象是一个比较形象的概念，而对于计算机来说，对象却是一个抽象的概念。

它并不能理解这是一个整数，那是一个字符串，对于计算机来说，它所知道的一切都是字节。

通常的说法是，对象是数据以及基于这些数据的操作的集合。

在计算机中，一个对象实际上就是一片被分配的内存空间，这些内存可能是连续的，也可能是离散的，这都不重要，重要的是这片内存在更高的层次上可以作为一个整体来考虑，这个整体就是一个对象。

在这片内存中，存储着一系列的数据以及可以对这些数据进行修改或读取操作的一系列代码。

在Python中，对象就是为C中的结构体在堆上申请的一块内存，一般来说，对象是不能被静态初始化的，并且也不能在栈空间上生存。

唯一的例外就是类型对象，Python中所有的内建的类型对象（如整数类型对象，字符串类型对象）都是被静态初始化的。

在Python中，一个对象一旦被创建，它在内存中的大小就是不变的了。

这就意味着那些需要容纳可变长度数据的对象只能在对象内维护一个指向一块可变大小的内存区域的指针。

## <<Python源码剖析>>

为什么要设定这样一条特殊的规则呢，因为遵循这样的规则可以使通过指针维护对象的工作变得非常的简单。

一旦允许对象的大小可在运行期改变，我们就可以考虑如下的情形。

在内存中有对象A，并且其后紧跟着对象B。

如果运行期某个时刻，A的大小增大了，这意味着必须将A整个移动到内存中的其他位置，否则A增大的部分将覆盖原本属于B的数据。

只要将A移动到内存中的其他位置，那么所有指向A的指针就必须立即得到更新，光是想一想，就知道这样的工作是多么的繁琐。

## <<Python源码剖析>>

### 编辑推荐

《Python源码剖析：深度探索动态语言核心技术》适合于Python程序员、动态语言爱好者、C程序员阅读。

作为主流的动态语言，Python不仅简单易学、移植性好，而且拥有强大丰富的库的支持。此外，Python强大的可扩展性，让开发人员既可以非常容易地利用C/C++编写Python的扩展模块，还能将Python嵌入到C/C++程序中，为自己的系统添加动态扩展和动态编程的能力。

为了更好地利用Python语言，无论是使用Python语言本身，还是将Python与C/C++交互使用，深刻理解Python的运行原理都是非常重要的。

## <<Python源码剖析>>

### 版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>