

<<Python核心编程（第二版）>>

图书基本信息

书名：<<Python核心编程（第二版）>>

13位ISBN编号：9787115178503

10位ISBN编号：711517850X

出版时间：2008-06

出版时间：人民邮电出版社

作者：[美]Wesley J. Chun（陳仲才）

页数：654

译者：CPUG

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核心编程（第二版）>>

内容概要

本书是经典的Python指导书，在第一版的基础上进行了全面升级。

全书分为两个部分：第1部分占据了大约三分之二的篇幅，阐释这门语言的“核心”内容，包括基本的概念和语句、语法和风格、Python对象、数字类型、序列类型、映射和集合类型、条件和循环、文件和输入/输出、错误和异常、函数和函数式编程、模块、面向对象编程、执行环境等内容；第2部分则提供了各种高级主题来展示可以使用Python做些什么，包括正则表达式、网络编程、网络客户端编程、多线程编程、图形用户界面编程、Web编程、数据库编程、扩展Python和一些其他材料。

本书适合Python初学者，以及已经入门但想继续学习和提高自身Python技巧的程序员。

<<Python核心编程 (第二版)>>

作者简介

作者：(美国)丘恩(Wesley J.Chun) 译者：宋吉广

<<Python核心编程 (第二版)>>

书籍目录

第1部分 Python核心	第1章 欢迎来到Python世界	1.1 什么是Python	1.2 起源
1.3 特点	1.3.1 高级	1.3.2 面向对象	1.3.3 可升级
1.3.4 可扩展	1.3.5 可移植性	1.3.6 易学	1.3.7 易读
1.3.8 健壮性	1.3.9 高效的快速原型开发工具	1.3.10 内存管理器	1.3.11 易维护
1.3.12 解释性和(字节)编译性	1.4 下载和安装Python	1.5 运行Python	1.5.1 命令
1.5.2 运行上的交互式解释器	1.5.3 从命令行启动脚本	1.5.4 集成开发环境	1.5.5 其他的集成开发环境和执行环境
1.6 Python文档	1.7 比较Python (Python与其他语言的比较)	1.8 其他实现	1.9 练习
第2章 快速入门	2.1 程序输出, print语句及“Hello World!”	2.2 程序输入和raw_input()内建函数	2.3 注释
2.4 操作符	2.5 变量和赋值	2.6 数字	2.7 字符串
2.8 列表和元组	2.9 字典	2.10 代码块及缩进对齐	2.11 if语句
2.12 while循环	2.13 for循环和range()内建函数	2.14 列表解析	2.15 文件和内建函数open()、file()
2.16 错误和异常	2.17 函数	2.17.1 如何定义函数	2.17.2 如何调用函数
2.17.3 默认参数	2.18 类	2.19 模块	2.19.1 如何导入模块
2.19.2 如何访问一个模块函数或访问一个模块变量	2.20 实用的函数	2.21 练习	第3章 Python基础
3.1 语句和语法	3.1.1 注释(#)	3.1.2 继续(\)	3.1.3 多个语句构成代码组(:)
3.1.4 代码组由不同的缩进分隔	3.1.5 同一行书写多个语句(;))	3.1.6 模块	3.2 变量赋值
3.2.1 赋值操作符	3.2.2 增量赋值	3.2.3 多重赋值	3.2.4 “多元”赋值
3.3 标识符	3.3.1 合法的Python标识符	3.3.2 关键字	3.3.3 内建
3.3.4 专用下划线标识符	3.4 基本风格指南	3.4.1 模块结构和布局	3.4.2 在主程序中书写测试代码
3.5 内存管理	3.5.1 变量定义	3.5.2 动态类型	3.5.3 内存分配
3.5.4 引用计数	3.5.5 垃圾收集	3.6 第一个Python程序	3.7 相关模块和开发工具
3.8 练习	4.1 Python对象	4.2 标准类型	4.3 其他内建类型
4.3.1 类型对象和type类型对象	4.3.2 None--Python的Null对象	4.4 内部类型	4.4.1 代码对象
4.4.2 帧对象	4.4.3 跟踪记录对象	4.4.4 切片对象	4.4.5 省略对象
4.4.6 xrange对象	4.5 标准类型操作符	4.5.1 对象值的比较	4.5.2 对象身份比较
4.5.3 布尔类型	4.6 标准类型内建函数	4.6.1 type()	4.6.2 cmp()
4.6.3 str()和repr() (及`操作符)	4.6.4 type()和isinstance()	4.6.5 Python类型操作符和内建函数总结	4.7 类型工厂函数
4.8 标准类型的分类	4.8.1 存储模型	4.8.2 更新模型	4.8.3 访问模型
4.9 不支持的类型	4.10 练习	第5章 数字	5.1 数字简介
5.1.1 如何创建数值对象并用其赋值(数字对象)	5.1.2 如何更新数字对象	5.1.3 如何删除数字对象	5.2 整型
5.2.1 布尔型	5.2.2 标准整型	5.2.3 长整型	5.2.4 整型和长整型的统一
5.3 双精度浮点型	5.4 复数	5.5 操作符	5.5.1 混合模式操作符
5.5.2 标准类型操作符	5.5.3 算术操作符	5.5.4 *位操作符(只适用于整型)	5.6 内建函数与工厂函数
5.6.1 标准类型函数	5.6.2 数字类型函数	5.6.3 仅用于整型的函数	5.7 其他数字类型
5.7.1 布尔“数”	5.7.2 十进制浮点型	5.8 相关模块	5.9 练习
第6章 序列:字符串、列表和元组	6.1 序列	6.1.1 标准类型操作符	6.1.2 序列类型操作符
6.1.3 内建函数(BIF)	6.2 字符串	6.3 字符串和操作符	6.3.1 标准类型操作符
6.3.2 序列操作符切片([]和[:])	6.4 只适用于字符串的操作符	6.4.1 格式化操作符(%)	6.4.2 字符串模板:更简单的替代品
6.4.3 原始字符串操作符(r/R)	6.4.4 Unicode字符串操作符(u/U)	6.5 内建函数	6.5.1 标准类型函数
6.5.2 序列类型函数	6.5.3 字符串类型函数	6.6 字符串内建函数	

<<Python核心编程 (第二版)>>

- 6.7 字符串的独特特性
 - 6.7.1 特殊字符串和控制字符
 - 6.7.2 三引号
 - 6.7.3 字符串不变性
- 6.8 Unicode
 - 6.8.1 术语
 - 6.8.2 什么是Unicode
 - 6.8.3 怎样使用Unicode
 - 6.8.4 Codec是什么
 - 6.8.5 编码解码
 - 6.8.6 把Unicode应用到实际应用中
 - 6.8.7 从现实中得来的教训
 - 6.8.8 Python的Unicode支持
- 6.9 相关模块
- 6.10 字符串关键点总结
- 6.11 列表
 - 6.12 操作符
 - 6.12.1 标准类型操作符
 - 6.12.2 序列类型操作符
 - 6.12.3 列表类型操作符和列表解析
- 6.13 内建函数
 - 6.13.1 标准类型函数
 - 6.13.2 序列类型函数
 - 6.13.3 列表类型内建函数
- 6.14 列表类型的内建函数
- 6.15 列表的特殊特性
- 6.16 元组
 - 6.17 元组操作符和内建函数
 - 6.17.1 标准类型操作符、序列类型操作符和内建函数
 - 6.17.2 元组类型操作符和内建函数、内建方法
 - 6.18 元组的特殊特性
 - 6.18.1 不可变性给元组带来了什么影响
 - 6.18.2 元组也不是那么“不可变”
 - 6.18.3 默认集合类型
 - 6.18.4 单元素元组
 - 6.18.5 字典的关键字
- 6.19 相关模块
- 6.20 *拷贝Python对象、浅拷贝和深拷贝
- 6.21 序列类型小结
- 6.22 练习
- 第7章 映像和集合类型
 - 7.1 映射类型：字典
 - 7.1.1 如何创建字典和给字典赋值
 - 7.1.2 如何访问字典中的值
 - 7.1.3 如何更新字典
 - 7.1.4 如何删除字典元素和字典
 - 7.2 映射类型操作符
 - 7.2.1 标准类型操作符
 - 7.2.2 映射类型操作符
 - 7.3 映射类型的内建函数和工厂函数
 - 7.3.1 标准类型函数[type()、str()和cmp()]
 - 7.3.2 映射类型相关的函数
 - 7.4 映射类型内建方法
 - 7.5 字典的键
 - 7.5.1 不允许一个键对应多个值
 - 7.5.2 键必须是可哈希的
 - 7.6 集合类型
 - 7.6.1 如何创建集合类型和给集合赋值
 - 7.6.2 如何访问集合中的值
 - 7.6.3 如何更新集合
 - 7.6.4 如何删除集合中的成员和集合
 - 7.7 集合类型操作符
 - 7.7.1 标准类型操作符(所有的集合类型)
 - 7.7.2 集合类型操作符(所有的集合类型)
 - 7.7.3 集合类型操作符(仅适用于可变集合)
 - 7.8 内建函数
 - 7.8.1 标准类型函数
 - 7.8.2 集合类型工厂函数
 - 7.9 集合类型内建方法
 - 7.9.1 方法(所有的集合方法)
 - 7.9.2 方法(仅适用于可变集合)
 - 7.9.3 操作符和内建方法比较
- 7.10 集合类型总结表
- 7.11 相关模块
- 7.12 练习
- 第8章 条件和循环
 - 8.1 if语句
 - 8.1.1 多重条件表达式
 - 8.1.2 单一语句的代码块
 - 8.2 else语句
 - 8.3 elif(即else-if)语句
 - 8.4 条件表达式(即“三元操作符”)
 - 8.5 while语句
 - 8.5.1 一般语法
 - 8.5.2 计数循环
 - 8.5.3 无限循环
 - 8.6 for语句
 - 8.6.1 一般语法
 - 8.6.2 用于序列类型
 - 8.6.3 用于迭代器类型
 - 8.6.4 range()内建函数
 - 8.6.5 xrange()内建函数
 - 8.6.6 与序列相关的内建函数
 - 8.7 break语句
 - 8.8 continue语句
 - 8.9 pass语句
 - 8.10 再谈else语句
 - 8.11 迭代器和iter()函数
 - 8.11.1 什么是迭代器
 - 8.11.2 为什么要迭代器
 - 8.11.3 如何迭代
 - 8.11.4 使用迭代器
 - 8.11.5 可变对象和迭代器
 - 8.11.6 如何创建迭代器
 - 8.12 列表解析
 - 8.13 生成器表达式
 - 8.14 相关模块
 - 8.15 练习
- 第9章 文件和输入输出
 - 9.1 文件对象
 - 9.2 文件内建函数(open()和file())
 - 9.2.1 工厂函数file()
 - 9.2.2 通用换行符支持(UNS)
 - 9.3 文件内建方法
 - 9.3.1 输入
 - 9.3.2 输出
 - 9.3.3 文件内移动
 - 9.3.4 文件迭代
 - 9.3.5 其他
 - 9.3.6 文件方法杂项
 - 9.4 文件内建属性
 - 9.5 标准文件
 - 9.6 命令行参数
 - 9.7 文件系统
 - 9.8 文件执行
 - 9.9 永久存储模块
 - 9.9.1 pickle和marshal模块
 - 9.9.2 DBM风格的模块
 - 9.9.3 shelve模块
 - 9.10 相关模块
 - 9.11 练习
- 第10章 错误和异常
 - 10.1 什么是异常
 - 10.1.1 错误
 - 10.1.2 异常
 - 10.2 Python中的异常
 - 10.3 检测和处理异常
 - 10.3.1 try-except语句
 - 10.3.2 包装内建函数
 - 10.3.3 带有多重except的try语句
 - 10.3.4 处理多个异常的except语句
 - 10.3.5 捕获所有异常
 - 10.3.6 “异常参数”
 - 10.3.7 在应用使用我们封装的函数
 - 10.3.8 else子句
 - 10.3.9 finally子句
 - 10.3.10 try-finally语句
 - 10.3.11 try-except-else-finally：厨房一锅端
 - 10.4 上下文管理
 - 10.4.1 with语句
 - 10.4.2 *上下文管理协议
 - 10.5 *字符串作为异常
 - 10.6 触发

<<Python核心编程 (第二版)>>

异常	10.7 断言	10.8 标准异常	10.9 *创建异常	10.10 (现在)为什么用异常
练习	10.11 到底为什么要异常	10.12 异常和sys模块	10.13 相关模块	10.14
函数操作符	第11章 函数和函数式编程	11.1 什么是函数?	11.2 调用函数	11.2.1
创建函数	11.1.1 函数vs过程	11.1.2 返回值与函数类型	11.2.2 关键字参数	11.2.3 默认参数
11.3.4 函数属性	11.3.1 def语句	11.3.2 声明与定义比较	11.2.4 参数组	11.3
传递函数	11.3.4 函数属性	11.3.5 内部/内嵌函数	11.3.3 前向引用	11.4
11.6 可变长度的参数	11.5 Formal Arguments	11.5.1 位置参数	11.3.6 *函数(与方法)装饰器	11.4
数(字典)	11.6.1 非关键字可变长参数(元组)	11.5.2 默认参数	11.6.2 关键字变量参数	11.7.1 匿名函数与lambda
偏函数应用	11.6.3 调用带有可变长参数对象函数	11.7 函数式编程	11.7.2 内建函数apply()、filter()、map()、reduce()	11.7.3
变量作用域和名称空间	11.8 变量作用域	11.8.1 全局变量与局部变量	11.8.2 globa语句	11.8.6
模块	11.8.3 作用域的数字	11.8.4 闭包	11.8.5 作用域和lambda	11.8.6
、覆盖	11.9 *递归	11.10 生成器	11.10.1 简单的生成器特性	11.11 练习
12.4.2 from-import语句	11.10.2 加强的生成器特性	11.11 练习	第12章 模块	12.1 什么是
12.5 模块导入的特性	12.2 模块和文件	12.2.1 模块名称空间	12.2.2 搜索路径和路径搜索	12.3.1 名称查找、确定作用域
(load)	12.3 名称空间	12.3.1 名称空间与变量作用域比较	12.3.2 名称查找、确定作用域	12.4.1 import语句
12.5.5 关于__future__	12.3.3 无限制的名称空间	12.4 导入模块	12.4.1 import语句	12.4.4 扩展的import语句(as)
12.5.8 “新的”导入钩子	12.4.2 from-import语句	12.4.3 多行导入	12.4.4 扩展的import语句(as)	12.5.2 导入(import)和加载
globals()和locals()	12.5.1 载入时执行模块	12.5.2 导入(import)和加载	12.5.2 导入(import)和加载	12.5.4 被导入到导入者作用域的名字
12.7.2 使用from-import导入包	12.5.3 导入到当前名称空间的名称	12.5.4 被导入到导入者作用域的名字	12.5.4 被导入到导入者作用域的名字	12.5.7 从ZIP文件中导入模块
12.8.2 阻止属性导入	12.5.5 关于__future__	12.5.6 警告框架	12.5.7 从ZIP文件中导入模块	12.6.1 __import__()
12.8.5 导入循环	12.5.8 “新的”导入钩子	12.6 模块内建函数	12.6.1 __import__()	12.6.2
面向对象编程	globals()和locals()	12.6.3 reload()	12.7 包	12.7.1 目录结构
的关系	12.7.2 使用from-import导入包	12.8 模块的其他特性	12.8.1 自动载入的模块	12.8.4 源代码编码
建类	12.8.2 阻止属性导入	12.8.3 不区分大小的导入	12.8.4 源代码编码	12.9 相关模块
Methods	12.8.5 导入循环	12.8.6 模块执行	12.9 相关模块	12.10 练习
13.5.1 初始化:通过调用类对象来创建实例	12.8.6 模块执行	12.9 相关模块	12.10 练习	第13章 面
13.5.3 __new__()“构造器”方法	12.9 相关模块	12.10 练习	第13章 面	13.1 引言
属性	13.1 引言	13.2 面向对象编程	13.2.1 面向对象设计与面向对象编程的关系	13.2.2 现实中的问题
性	13.2 面向对象编程	13.2.1 面向对象设计与面向对象编程的关系	13.2.2 现实中的问题	13.2.3 *常用术语
13.7 绑定和方法调用	13.2.2 现实中的问题	13.2.3 *常用术语	13.3 类	13.3.1 创
静态方法和类方法	13.3.2 声明与定义	13.4 类属性	13.3.1 创	13.3.2
用函数修饰符	13.4 类属性	13.4.1 类的数据属性	13.3.2	13.4.2
__bases__类属性	13.4.1 类的数据属性	13.4.2 特殊的类属性	13.4.2 特殊的类属性	13.5 实例
13.11.4 多重继承	13.4.2 特殊的类属性	13.5 实例	13.5 实例	13.5.1 初始化:通过调用类对象来创建实例
13.12.2 isinstance()	13.5.1 初始化:通过调用类对象来创建实例	13.5.2 __init__()“构造器”方法	13.5.2 __init__()“构造器”方法	13.5.3 __new__()“构造器”方法
13.12.4 dir()	13.5.3 __new__()“构造器”方法	13.5.4 __del__()“解构器”方法	13.5.4 __del__()“解构器”方法	13.6 实例
类	13.5.4 __del__()“解构器”方法	13.6 实例	13.6 实例	13.6.1 “实例化”实例属性(或创建一个更好的构造器)
迭代器(RandSeq和Anylter)	13.6.1 “实例化”实例属性(或创建一个更好的构造器)	13.6.2 查看实例属性	13.6.2 查看实例属性	13.6.3 特殊的实例属性
13.15 *授权	13.6.2 查看实例属性	13.6.3 特殊的实例属性	13.6.3 特殊的实例属性	13.6.4 建类型属性
(Python 2.2+)	13.6.3 特殊的实例属性	13.6.4 建类型属性	13.6.4 建类型属性	13.6.5 实例属性vs类属性
__getattr__()特殊方法	13.6.4 建类型属性	13.6.5 实例属性vs类属性	13.6.5 实例属性vs类属性	13.7 绑定和方法调用
	13.6.5 实例属性vs类属性	13.7 绑定和方法调用	13.7 绑定和方法调用	13.7.1 调用绑定方法
	13.7 绑定和方法调用	13.7.1 调用绑定方法	13.7.1 调用绑定方法	13.7.2 调用非绑定方法
	13.7.1 调用绑定方法	13.7.2 调用非绑定方法	13.7.2 调用非绑定方法	13.8
	13.7.2 调用非绑定方法	13.8 使用函数修饰符	13.8 使用函数修饰符	13.8.1 staticmethod()和classmethod()内建函数
	13.8 使用函数修饰符	13.8.1 staticmethod()和classmethod()内建函数	13.8.1 staticmethod()和classmethod()内建函数	13.8.2 使用
	13.8.1 staticmethod()和classmethod()内建函数	13.8.2 使用	13.8.2 使用	13.9 组合
	13.8.2 使用	13.9 组合	13.9 组合	13.10 子类 and 派生
	13.9 组合	13.10 子类 and 派生	13.10 子类 and 派生	13.11 继承
	13.10 子类 and 派生	13.11 继承	13.11 继承	13.11.1
	13.11 继承	13.11.1 通过继承覆盖方法	13.11.2 通过继承覆盖方法	13.11.3 从标准类型派生
	13.11.1 通过继承覆盖方法	13.11.2 通过继承覆盖方法	13.11.2 通过继承覆盖方法	13.11.3 从标准类型派生
	13.11.2 通过继承覆盖方法	13.11.3 从标准类型派生	13.11.3 从标准类型派生	13.12.1 issubclass()
	13.11.3 从标准类型派生	13.12.1 issubclass()	13.12.1 issubclass()	13.12.2 isinstance()
	13.12.1 issubclass()	13.12.2 isinstance()	13.12.2 isinstance()	13.12.3 hasattr()、getattr()、setattr()、delattr()
	13.12.2 isinstance()	13.12.3 hasattr()、getattr()、setattr()、delattr()	13.12.3 hasattr()、getattr()、setattr()、delattr()	13.12.4 dir()
	13.12.3 hasattr()、getattr()、setattr()、delattr()	13.12.4 dir()	13.12.4 dir()	13.12.5 super()
	13.12.4 dir()	13.12.5 super()	13.12.5 super()	13.12.6 vars()
	13.12.5 super()	13.12.6 vars()	13.12.6 vars()	13.13 用特殊方法定制
	13.12.6 vars()	13.13 用特殊方法定制	13.13 用特殊方法定制	13.13.1 简单定制(RoundFloat2)
	13.13 用特殊方法定制	13.13.1 简单定制(RoundFloat2)	13.13.1 简单定制(RoundFloat2)	13.13.2 数值定制(Time60)
	13.13.1 简单定制(RoundFloat2)	13.13.2 数值定制(Time60)	13.13.2 数值定制(Time60)	13.13.3
	13.13.2 数值定制(Time60)	13.13.3	13.13.3	13.13.4 *多类型定制(NumStr)
	13.13.3	13.13.4 *多类型定制(NumStr)	13.13.4 *多类型定制(NumStr)	13.14 私有化
	13.13.4 *多类型定制(NumStr)	13.14 私有化	13.14 私有化	13.15 *授权
	13.14 私有化	13.15 *授权	13.15 *授权	13.15.1 包装
	13.15 *授权	13.15.1 包装	13.15.1 包装	13.15.2 实现授权
	13.15.1 包装	13.15.2 实现授权	13.15.2 实现授权	13.16 新式类的高级特性
	13.15.2 实现授权	13.16 新式类的高级特性	13.16 新式类的高级特性	13.16.1 新式类的通用特性
	13.16 新式类的高级特性	13.16.1 新式类的通用特性	13.16.1 新式类的通用特性	13.16.2 __slots__类属性
	13.16.1 新式类的通用特性	13.16.2 __slots__类属性	13.16.2 __slots__类属性	13.16.3
	13.16.2 __slots__类属性	13.16.3	13.16.3	13.16.4 描述符
	13.16.3	13.16.4 描述符	13.16.4 描述符	13.16.5 元类和__metaclass__
	13.16.4 描述符	13.16.5 元类和__metaclass__	13.16.5 元类和__metaclass__	

<<Python核心编程 (第二版)>>

13.17 相关模块和文档	13.18 练习	第14章 执行环境	14.1 可调用对象
14.1.1 函数	14.1.2 方法	14.1.3 类	14.1.4 类的实例
14.2 代码对象	14.3 可执行的对象声明和内建函数	14.3.1 callable ()	14.3.2 compile ()
14.3.3 eval ()	14.3.4 exec	14.3.5 input ()	14.3.6 使用Python
在运行时生成和执行Python代码	14.4 执行其他 (Python) 程序	14.4.1 导入	
14.4.2 execfile ()	14.4.3 将模块作为脚本执行	14.5 执行其他 (非Python) 程序	
14.5.1 os.system ()	14.5.2 os.popen ()	14.5.3 os.fork ()、os.exec* ()	
、os.wait* ()	14.5.4 os.spawn* ()	14.5.5 subprocess 模块	14.5.6 相关函数
14.6 受限执行	14.7 结束执行	14.7.1 sys.exit () and SystemExit	
14.7.2 sys.exitfunc ()	14.7.3 os._exit () 函数	14.7.4 os.kill () Function	
14.8 各种操作系统接口	14.9 相关模块	14.10 练习	第2部分 高级主题
15.1 引言/动机	15.2 正则表达式使用的特殊符号和字符	15.2.1 用	第15章
正则表达式	15.2.2 匹配任意一个单个的字符 (.)	15.2.3	
管道符号 () 匹配多个正则表达式模式	15.2.4 创建字符类 ([])		
从字符串的开头或结尾或单词边界开始匹配 (^/\$/ \b /\B)	15.2.6 使用闭包操作符 (*, +, ?, {) 实现多次出现/重复匹配	15.2.8 用圆括号 (()) 组建组	
15.2.5 指定范围 (-) 和否定 (^)	15.3 正则表达式和Python语言	15.3.1 re模块: 核心函数和方法	15.3.2 使
用compile () 编译正则表达式	15.3.3 匹配对象和group ()、groups () 方法	15.3.4	
用match () 匹配字符串	15.3.5 search () 在一个字符串中查找一个模式 (搜索与匹配的比较)	15.3.6 匹配多个字符串 ()	15.3.7 匹配任意单个字符 (.)
15.3.8 创建字符集合 ([])	15.3.9 重复、特殊字符和子组	15.3.10 从字符串的开头或结尾匹配及在单词边界上的匹配	15.3.11 用findall () 找到每个出现的匹配部分
15.3.12	15.3.13 用split () 分割 (分隔模式)	15.4	
用sub () (和subn ()) 进行搜索和替换	15.4.1 匹配一个字符串	15.4.2 搜索与匹配的比较, “贪婪”匹配	
15.4.3	15.5 练习	第16章 网络编程	16.1 引言
16.1.1 什么是客户端/服务器架构	16.1.2 客户端/服务器网络编程	16.2 套接字: 通信端点	16.2.1 什么是套接
16.2.2 套接字地址: 主机与端口	16.2.3 面向连接与无连接	16.3 Python中的	
的网络编程	16.3.1 socket () 模块函数	16.3.2 套接字对象 (内建) 方法	
16.3.3 创建一个TCP服务器	16.3.4 创建TCP客户端	16.3.5 运行我们的客户端	
与TCP服务器	16.3.6 创建一个UDP服务器	16.3.7 创建一个UDP客户端	
16.3.8 执行UDP服务器和客户端	16.3.9 Socket模块属性	16.4 *SocketServer模块	
16.4.1 创建一个SocketServerTCP服务器	16.4.2 创建SocketServerTCP客户端	16.5 Twisted框架介绍	16.5.1 创建一个Twisted
16.4.3 执行TCP服务器和客户端	16.5.2 创建一个Twisted Reactor TCP客户端	16.5.3 执行TCP服务	
器和客户端	16.6 相关模块	16.7 练习	第17章 网络客户端编程
17.1 什么是	17.2 文件传输	17.2.1 文件传输网际协议	17.2.2 文件传输协议
因特网客户端	17.2.3 Python和FTP	17.2.4 ftplib.FTP类方法	17.2.5 交互式FTP示
例	17.2.6 客户端FTP程序举例	17.2.7 FTP的其他方面	17.3 网络新闻
17.3.1 Usenet与新闻组	17.3.2 网络新闻传输协议 (NNTP)	17.3.3 Python	
和NNTP	17.3.4 nntplib.NNTP类方法	17.3.5 交互式NNTP举例	17.3.6 客户
端程序NNTP举例	17.3.7 NNTP的其他方面	17.4 电子邮件	17.4.1 电子邮件
系统组件和协议	17.4.2 发送电子邮件	17.4.3 Python和SMTP	17.4.4
smtplib.SMTP类方法	17.4.5 交互式SMTP示例	17.4.6 SMTP的其他方面	
17.4.7 接收电子邮件	17.4.8 POP和IMAP	17.4.9 Python和POP3	17.4.10
交互式POP3举例	17.4.11 poplib.POP3类方法	17.4.12 客户端程序SMTP和POP3举例	
17.5 相关模块	17.5.1 电子邮件	17.5.2 其他网络协议	17.6 练习
第18章 多线程编程	18.1 引言/动机	18.2 线程和进程	18.2.1 什么是进程

<<Python核心编程 (第二版)>>

18.2.2 什么是线程 (GIL)	18.3 Python、线程和全局解释器锁	18.3.1 全局解释器锁
18.3.2 退出线程	18.3.3 在Python中使用线程	18.3.4 没有线程支持的情况
18.5.1 Thread类	18.3.5 Python的threading模块	18.4 thread模块
18.5.2 斐波那契、阶乘和累加和	18.5 threading模块	18.5.3 threading模块中的其他函数
18.5.4 生产者-消费者问题和Queue模块	18.6 相关模块	18.7 练习
第19章 图形用户界面编程	19.1 简介	19.1.1 什么是Tcl、Tk和Tkinter
19.1.2 安装和使用Tkinter 533	19.1.3 客户端/服务器架构 534	19.2 Tkinter与Python编程 534
19.2.1 Tkinter模块：把Tk引入你的程序	19.2.2 GUI程序开发简介	19.3.1 标签组件
19.2.3 顶层窗口：Tkinter.Tk()	19.2.4 Tk组件	19.3 Tkinter举例
19.3.2 按钮组件	19.3.3 标签和按钮组件	19.3.4 标签、按钮和进度条组件
19.3.5 偏函数应用举例	19.3.6 中级Tkinter范例	19.4 其他GUI简介
19.4.1 Tk Interface eXtensions (Tix)	19.4.2 Python MegaWidgets (PMW)	19.4.3 wxWidgets和wxPython
19.4.4 GTK+和PyGTK	19.5 相关模块和其他GUI	19.6 练习
第20章 Web编程	20.1 介绍	20.1.1 Web应用：客户端/服务器计算
20.1.2 因特网	20.2 使用Python进行Web应用：创建一个简单的Web客户端	20.2.1 统一资源定位符
20.2.2 urlparse模块	20.2.3 urllib模块	20.2.4 urllib2模块
20.3 高级Web客户端	20.4 CGI：帮助Web服务器处理客户端数据	20.4.1 CGI介绍
20.4.2 CGI应用程序	20.4.3 cgi模块	20.5 建立CGI应用程序
20.5.1 建立Web服务器	20.5.2 建立表单页	20.5.3 生成结果页
20.5.4 生成表单和结果页面	20.5.5 全面交互的Web站点	20.6 在CGI中使用Unicode编码
20.7 高级CGI	20.7.1 Mulitipart表单提交和文件的上传	20.7.2 多值字段
20.7.3 cookie	20.7.4 使用高级CGI	20.8 Web (HTTP) 服务器
20.9 相关模块	20.10 练习	第21章 数据库编程
21.1 介绍	21.1.1 持久存储	21.1.2 基本的数据库操作和SQL语言
21.1.3 数据库和Python	21.2 Python数据库应用程序程序员接口 (DB-API)	21.2.1 模块属性
21.2.2 连接对象	21.2.3 游标对象	21.2.4 类型对象和构造器
21.2.5 关系数据库	21.2.6 数据库和Python：接口程序	21.2.7 使用数据库接口程序举例
21.3 对象-关系管理器 (ORM)	21.3.1 考虑对象，而不是SQL	21.3.2 Python和ORM
21.3.3 雇员数据库	21.3.4 总结	21.4 相关模块
21.5 练习	第22章 扩展Python 623	22.1 引言/动机
22.1.1 什么是扩展	22.1.2 为什么要扩展Python	22.2 创建Python扩展
22.2.1 创建您的应用程序代码	22.2.2 用样板来包装你的代码	22.2.3 编译
22.2.4 导入和测试	22.2.5 引用计数	22.2.6 线程和全局解释器锁 (GIL)
22.3 相关话题	22.4 练习	第23章 其他话题
23.1 Web服务	23.1.1 客户端COM编程	23.1.2 微软Excel
23.2 用Win32的COM来操作微软Office	23.2.1 微软PowerPoint	23.2.2 微软Outlook
23.2.3 微软Word 第1部分 Python核心	23.2.4 中等规模的例子	23.3 用Jython写Python和Java的程序
23.2.5 什么是Jython	23.3.1 什么是Jython	23.3.2 Swing GUI开发 (Java或者Python!)
23.2.6 微软PowerPoint	23.4 练习	23.2.4 微软PowerPoint
23.2.5 微软Outlook	23.2.6 中等规模的例子	23.3 用Jython写Python和Java的程序
23.3.1 什么是Jython	23.3.2 Swing GUI开发 (Java或者Python!)	23.4 练习

章节摘录

插图：第1部分 Python核心第1章 欢迎来到Python世界开篇将介绍一些Python的背景知识，包括什么是Python、Python的起源和它的一些关键特性。

一旦你来了兴致，我们就会向你介绍怎样获得Python，以及如何在你的系统上安装并运行它。

本章最后的练习将会帮助你非常自如地使用Python，包括使用交互式解释器，以及创建并运行脚本程序。

1.1 什么是Python Python是一门优雅而健壮的编程语言，它继承了传统编译语言的强大性和通用性，同时也借鉴了简单脚本和解释语言的易用性。

它可以帮你完成工作，而且一段时间以后，你还能看明白自己写的这段代码。

你会对自己如此快地学会它和它强大的功能感到十分的惊讶，更不用提你已经完成的工作了！

只有你想不到，没有Python做不到。

1.2 起源 Guido van Rossum于1989年底始创了Python，那时，他还在荷兰的CWI（Centrum voor Wiskunde en Informatica，国家数学和计算机科学研究院）。

1991年初，Python发布了第一个公开发行人版。

这一切究竟是如何开始的呢？

像C、C++、Lisp、Java和Perl一样，Python来自于某个研究项目，项目中的那些程序员利用手边现有的工具辛苦地工作着，他们设想并开发出了更好的解决办法。

那时van Rossum是一位研究人员，对解释型语言ABC有着丰富的设计‘经验，这个语言同样也是在CWI开发的。

但是他不满足其有限的开发能力。

已经使用并参与开发了像ABC这样的高级语言后，再退回到C语言显然是不可能的。

他所期望的工具有一些是用于完成日常系统管理任务的，而且它还希望能够访问Amoeba分布式操作系统的系统调用。

尽管van Rossum也曾想过为Amoeba开发专用语言，但是创造一种通用的程序设计语言显然更加明智，于是在1989年末，Python的种子被播下了。

1.3 特点 尽管Python已经流行了超过15年，但是一些人仍旧认为相对于通用软件开发产业而言，它还是个新丁。

我们应当谨慎地使用“相对”这个词，因为“网络时代”的程序开发，几年看上去就像几十年。

当人们询问：“什么是Python？”

的时候，很难用任何一个具象来描述它。

人们更倾向于一口气不加思索地说出他们对Python的所有感觉，Python是一（请填写），这些特点究竟又是什么呢？

为了让你能知其所以然，我们下面会对这些特点进行逐一地阐释。

1.3.1 高级伴随着每一代编程语言的产生，我们会达到一个新的高度。

汇编语言是献给那些挣扎在机器代码中的人的礼物，后来有了FORTRAN、C和Pascal语言，它们将计算提升到了崭新的高度，并且开创了软件开发行业。

伴随着C语言诞生了更多的像C++、Java这样的现代编译语言。

我们没有止步于此，于是有了强大的、可以进行系统调用的解释型脚本语言，例如Tcl、Perl和Python。

这些语言都有高级的数据结构，这样就减少了以前“框架”开发需要的时间。

像Python中的列表（大小可变的数组）和字典（哈希表）就是内建于语言本身的。

在核心语言中提供这些重要的构建单元，可以鼓励人们使用它们，缩短开发与代码量，产生出可读性更好的代码。

在C语言中，对于混杂数组（Python中的列表）和哈希表（Python中的字典）还没有相应的标准库，所以它们经常被重复实现，并被复制到每个新项目中。

这个过程混乱而且容易产生错误。

<<Python核心编程（第二版）>>

编辑推荐

《Python核心编程(第2版)》提供了对Python核心特性系统的专家级讲解；开发复杂的应用程序和软件所需的强大深入的视角：易用的图表，详细描述了Pyffion模块、操作符、函数和方法：大量的专业级实例代码，从小的代码片段到功能齐全的应用程序一应俱全。

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>