

## <<Python标准库>>

### 图书基本信息

书名：<<Python标准库>>

13位ISBN编号：9787111378105

10位ISBN编号：7111378105

出版时间：2012-6-15

出版时间：机械工业出版社华章公司

作者：Doug Hellmann

页数：1016

译者：刘炽

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>

## &lt;&lt;Python标准库&gt;&gt;

## 前言

序 今天是2010年的感恩节。

不论人们是否身在美国，在这个节日里，大家都一边品尝丰盛的食物，一边欣赏橄榄球比赛，有些人可能会出门逛逛。

对我（以及其他很多人）来说，会借此机会回顾一下过去的岁月，想想那些让我们的生活充满色彩的人和事，向他们致以感谢。

当然，我们每天都该这么做，不过专门有一天来表达谢意有时会让我们想得更深远一些。

现在我坐在这里为本书写序，非常感谢能有机会做这件事，不过我想到的不只是本书的内容，也不只是作者本人（一个无比热情的社区成员），我所想的是这个主题本身—Python，具体来讲，还有它的标准库。

当前发布的每版Python都包含数百个模块，它们是多年来多位开发人员针对多个主题和多个任务共同完成的。

这些模块涵盖一切，从发送和接收Email，到GUI开发，再到内置的HTTP服务器都一应俱全。

就其本身而言，标准库的开发和维护是一项极其庞大的工作。

如果没有多年来一直维护它的人们，没有数以千计的人提交补丁、文档和提供反馈，它绝不会成为今天的模样。

这是一个惊人的成就，在Python日益普及的今天（不论是作为语言还是一种生态系统），标准库已经成为其中不可或缺的重要组成部分。

如果没有标准库，没有核心团队和其他人员的“内含动力”（batteries included）口号，Python绝对不可能走这么远。

它已经被成千上万的人和公司下载，并已安装在数百万服务器、台式机和其他设备上。

即使没有标准库，Python仍是一种很不错的语言，在教学、学习和可读性方面有扎实的基础。

基于这些优点，它本身也能发展得足够好。

不过标准库把它从一种有趣的体验变成为一个强大而有效的工具。

每一天，全世界的开发人员都在构建工具和完整的应用，他们所基于的只是核心语言和标准库。你不仅要能够明确概念，描述汽车是什么（语言），还要得到足够的部件和工具来自行组装一辆基本的汽车。

它可能并不完善，不过可以使你从无到有，这将是一个很好的奖励，会赋予你巨大的动力。

我曾反复对那些骄傲地看着我的人说：“看看我构建的应用，除了Python提供的，其他的工具通通都没有用到！”

不过，标准库也不是完美无缺的，它也有自己的缺陷。

由于标准库的规模、广度和它的“年龄”，毫无疑问有些模块有不同层次的质量、API简洁性和覆盖性。

有些模块存在“特性蔓延”的问题，或者无法跟进其覆盖领域中的最新进展。

通过众多不计酬劳的志愿者的帮助和辛勤工作，Python还在继续发展、壮大和改进。

不过，有些人对Python还有争议，不仅由于它的缺点，而且因为标准库并不一定构成其模块涵盖领域中“最顶尖的”解决方案（毕竟，“最佳”是一个不断改变和调整的目标），因而认为应当将它完全舍弃，尽管它还在不断改善。

这些人遗漏了一个事实：标准库不仅是促使Python不断成功的一个重要组成部分，而且尽管存在瑕疵，它还是一个绝妙的资源。

不过我有意忽略了一个巨大的领域：文档。

标准库的文档很不错，还在继续改进和发展。

由于标准库的庞大规模和广度，相应的文档规模也很惊人。

在数以千计的开发人员和用户的努力下，我们有成百上千页文档，实在让人佩服。

每天都有数万人在使用这些文档创建应用—可能简单到只是一页的脚本，也可能很复杂，比如控制大型机械手的软件。

## &lt;&lt;Python标准库&gt;&gt;

正是由于文档，我们才会看到本书。

所有好的文档和代码都有一个起点——关于目标“是什么”以及“将是什么”；要有一个核心概念。

从这个内核出发，才有了角色（API）和故事情节（模块）。

谈到代码，有时代码会从一个简单的想法开始：“我想解析一个字符串，查找一个日期。

”不过等结束时，你可能已经查看了数百个单元测试、函数以及你编写的其他代码，你会坐下来，发现自己构建的东西远远超出了原先的设想。

文档也是如此，特别是代码文档。

在我看来，代码文档中最重要的部分就属于这种例子。

要写有关一个API中某一部分的描述，你可能会写上几本书，可以用华丽的文字和经过深思熟虑的用例来描述松耦合的接口。

不过，如果第一次查看这个描述的用户无法将这些华丽的文字、仔细考量的用例和API签名结合在一起，构建出有意义的应用并解决他们的问题，这一切就完全是徒劳。

人们建立重要连接所用的网关也属于这种例子，这些逻辑会从一个抽象概念跳转到具体的事物。

“了解”思想和API是一回事；知道它如何使用则是另外一回事。

如果你不仅想要学到东西，还希望改善现状，这会很有帮助。

这就把我们重新引回Python。

本书作者Doug Hellmann在2007年创建了一个名为“Python Module of the Week”的博客。

在这个博客中，他全面研究了标准库的众多模块，采用一种“示例为先”的方式介绍各个模块如何工作以及为什么。

从读到它的第一天起，它就成为我除了核心Python文档之外的又一个必访之地。

他的作品已经成为我以及Python社区其他人不可缺少的必备资源。

Doug的文章填补了当前我所看到的Python文档的一个重大空白：对例子的迫切需求。

用一种有效而简单的方式展示如何做以及为什么这么做，这绝非易事。

我们已经看到，这也是一项很重要、很有价值的工作，对人们每一天的工作都有帮助。

人们频繁地给我发邮件，告诉我：“你看过Doug的这个帖子吗？

实在太棒了！

”或者“为什么这个不能放在核心文档里呢？

它能帮助我了解到底是怎么做的！

”当我听说Doug准备花些时间进一步完善他现有的工作，把它变成一本书，让我能把它放在桌子上反复翻阅，以备急用，我真是太兴奋了。

Doug是一位非凡的技术作者，而且能敏锐地捕捉到细节。

有一整本书专门讲解实际例子，介绍标准库中一百多个模块是如何工作的，而且是他来写这本书，实在让我欣喜若狂。

所以，我要感谢Python，感谢标准库（包括它的瑕疵），感谢我拥有的这个活力充沛有时也存在问题的庞大Python社区。

我要感谢核心开发小组的辛勤工作，包括过去、现在，还有将来。

我还要感谢这么多社区成员提供的资源、投入的时间和做出的努力，其中Doug Hellmann更是卓越的代表，正是这些让这个社区和生态系统如此生机勃勃。

最后，我要感谢本书。

继续向它的作者表示敬意，这本书会在未来几年得到充分利用。

Jesse Noller Python核心开发人员 PSF Board成员 Nasuni公司首席工程师

## &lt;&lt;Python标准库&gt;&gt;

## 内容概要

本书由资深Python专家亲自执笔，Python语言的核心开发人员作序推荐，权威性毋庸置疑。

对于程序员而言，标准库与语言本身同样重要，它好比一个百宝箱，能为各种常见的任务提供完美的解决方案，所以本书是所有Python程序员都必备的工具书！

本书以案例驱动的方式讲解了标准库中一百多个模块的使用方法（如何工作）和工作原理（为什么要这样工作），比标准库的官方文档更容易理解（一个简单的示例比一份手册文档更有帮助），为Python程序员熟练掌握和使用这些模块提供了绝佳指导。

全书一共19章，系统而全面地对Python标准库中的一百多个模块进行了生动的讲解。

这些模块主要包括：文本处理工具模块、与数据结构相关的模块、与算法有关的模块、管理日期和时间值的模块、用于数学计算的模块、管理文件系统的模块、用于数据存储与交换的模块、用于数据压缩与归档的模块、用于加密的模块、...

(展开全部) 本书由资深Python专家亲自执笔，Python语言的核心开发人员作序推荐，权威性毋庸置疑。

对于程序员而言，标准库与语言本身同样重要，它好比一个百宝箱，能为各种常见的任务提供完美的解决方案，所以本书是所有Python程序员都必备的工具书！

本书以案例驱动的方式讲解了标准库中一百多个模块的使用方法（如何工作）和工作原理（为什么要这样工作），比标准库的官方文档更容易理解（一个简单的示例比一份手册文档更有帮助），为Python程序员熟练掌握和使用这些模块提供了绝佳指导。

全书一共19章，系统而全面地对Python标准库中的一百多个模块进行了生动的讲解。

这些模块主要包括：文本处理工具模块、与数据结构相关的模块、与算法有关的模块、管理日期和时间值的模块、用于数学计算的模块、管理文件系统的模块、用于数据存储与交换的模块、用于数据压缩与归档的模块、用于加密的模块、与进程和线程相关的模块、与网络通信和Email相关的模块、构建模块、支持处理多种自然语言和文化设置的模块、开发工具模块、与运行时特性相关的模块，等等。

## <<Python标准库>>

### 作者简介

Doug Hellmann目前是Racemi公司的一位高级开发人员，也是Python Software

Foundation的信息交流主管。

从1.4版开始他就一直在做Python编程，曾在大量UNIX和非UNIX平台上参与项目开发，涉及领域包括地图、医疗新闻播报、金融和数据中心自动化。

为《Python

Magazine》做了一年普通专栏作家后，他在2008—2009年成为这家杂志的主编。

自2007年以来，Doug在他的博客上发表了颇受关注的“Python Module of the Week”系列。

他居住在乔治亚州的Athens。

## &lt;&lt;Python标准库&gt;&gt;

## 书籍目录

译者序

序

前言

第1章 文本

1.1 string—文本常量和模板

1.1.1 函数

1.1.2 模板

1.1.3 高级模板

1.2 textwrap—格式化文本段落

1.2.1 示例数据

1.2.2 填充段落

1.2.3 去除现有缩进

1.2.4 结合dedent和fill

1.2.5 悬挂缩进

1.3 re—正则表达式

1.3.1 查找文本中的模式

1.3.2 编译表达式

1.3.3 多重匹配

1.3.4 模式语法

1.3.5 限制搜索

1.3.6 用组解析匹配

1.3.7 搜索选项

1.3.8 前向或后向

1.3.9 自引用表达式

1.3.10 用模式修改字符串

1.3.11 利用模式拆分

1.4 difflib—比较序列

1.4.1 比较文本体

1.4.2 无用数据

1.4.3 比较任意类型

第2章 数据结构

2.1 collections—容器数据类型

2.1.1 Counter

2.1.2 defaultdict

2.1.3 deque

2.1.4 namedtuple

2.1.5 OrderedDict

2.2 array—固定类型数据序列

2.2.1 初始化

2.2.2 处理数组

2.2.3 数组与文件

2.2.4 候选字节顺序

2.3 heapq—堆排序算法

2.3.1 示例数据

2.3.2 创建堆

## &lt;&lt;Python标准库&gt;&gt;

- 2.3.3 访问堆的内容
- 2.3.4 堆的数据极值
- 2.4 bisect—维护有序列表
  - 2.4.1 有序插入
  - 2.4.2 处理重复
- 2.5 Queue—线程安全的FIFO实现
  - 2.5.1 基本FIFO队列
  - 2.5.2 LIFO队列
  - 2.5.3 优先队列
  - 2.5.4 构建一个多线程播客客户程序
- 2.6 struct—二进制数据结构
  - 2.6.1 函数与Struct类
  - 2.6.2 打包和解包
  - 2.6.3 字节序
  - 2.6.4 缓冲区
- 2.7 weakref—对象的非永久引用
  - 2.7.1 引用
  - 2.7.2 引用回调
  - 2.7.3 代理
  - 2.7.4 循环引用
  - 2.7.5 缓存对象
- 2.8 copy—复制对象
  - 2.8.1 浅副本
  - 2.8.2 深副本
  - 2.8.3 定制复制行为
  - 2.8.4 深副本中的递归
- 2.9 pprint—美观打印数据结构
  - 2.9.1 打印
  - 2.9.2 格式化
  - 2.9.3 任意类
  - 2.9.4 递归
  - 2.9.5 限制嵌套输出
  - 2.9.6 控制输出宽度
- 第3章 算法
  - 3.1 functools—管理函数的工具
    - 3.1.1 修饰符
    - 3.1.2 比较
  - 3.2 itertools—迭代器函数
    - 3.2.1 合并和分解迭代器
    - 3.2.2 转换输入
    - 3.2.3 生成新值
    - 3.2.4 过滤
    - 3.2.5 数据分组
  - 3.3 operator—内置操作符的函数接口
    - 3.3.1 逻辑操作
    - 3.3.2 比较操作符
    - 3.3.3 算术操作符

## &lt;&lt;Python标准库&gt;&gt;

- 3.3.4 序列操作符
- 3.3.5 原地操作符
- 3.3.6 属性和元素“获取方法”
- 3.3.7 结合操作符和定制类
- 3.3.8 类型检查
- 3.4 contextlib—上下文管理器工具
  - 3.4.1 上下文管理器API
  - 3.4.2 从生成器到上下文管理器
  - 3.4.3 嵌套上下文
  - 3.4.4 关闭打开的句柄
- 第4章 日期和时间
  - 4.1 time—时钟时间
    - 4.1.1 壁挂钟时间
    - 4.1.2 处理器时钟时间
    - 4.1.3 时间组成
    - 4.1.4 处理时区
    - 4.1.5 解析和格式化时间
  - 4.2 datetime—日期和时间值管理
    - 4.2.1 时间
    - 4.2.2 日期
    - 4.2.3 timedelta
    - 4.2.4 日期算术运算
    - 4.2.5 比较值
    - 4.2.6 结合日期和时间
    - 4.2.7 格式化和解析
    - 4.2.8 时区
  - 4.3 calendar—处理日期
    - 4.3.1 格式化示例
    - 4.3.2 计算日期
- 第5章 数学计算
  - 5.1 decimal—定点数和浮点数的数学运算
    - 5.1.1 Decimal
    - 5.1.2 算术运算
    - 5.1.3 特殊值
    - 5.1.4 上下文
  - 5.2 fractions—有理数
    - 5.2.1 创建 Fraction实例
    - 5.2.2 算术运算
    - 5.2.3 近似值
  - 5.3 random—伪随机数生成器
    - 5.3.1 生成随机数
    - 5.3.2 指定种子
    - 5.3.3 保存状态
    - 5.3.4 随机整数
    - 5.3.5 选择随机元素
    - 5.3.6 排列
    - 5.3.7 采样

## &lt;&lt;Python标准库&gt;&gt;

- 5.3.8 多个并发生成器
- 5.3.9 SystemRandom
- 5.3.10 非均匀分布
- 5.4 math—数学函数
  - 5.4.1 特殊常量
  - 5.4.2 测试异常值
  - 5.4.3 转换为整数
  - 5.4.4 其他表示
  - 5.4.5 正号和负号
  - 5.4.6 常用计算
  - 5.4.7 指数和对数
  - 5.4.8 角
  - 5.4.9 三角函数
  - 5.4.10 双曲函数
  - 5.4.11 特殊函数
- 第6章 文件系统
  - 6.1 os.path—平台独立的文件名管理
    - 6.1.1 解析路径
    - 6.1.2 建立路径
    - 6.1.3 规范化路径
    - 6.1.4 文件时间
    - 6.1.5 测试文件
    - 6.1.6 遍历一个目录树
  - 6.2 glob—文件名模式匹配
    - 6.2.1 示例数据
    - 6.2.2 通配符
    - 6.2.3 单字符通配符
    - 6.2.4 字符区间
  - 6.3 linecache—高效读取文本文件
    - 6.3.1 测试数据
    - 6.3.2 读取特定行
    - 6.3.3 处理空行
    - 6.3.4 错误处理
    - 6.3.5 读取Python源文件
  - 6.4 tempfile—临时文件系统对象
    - 6.4.1 临时文件
    - 6.4.2 命名文件
    - 6.4.3 临时目录
    - 6.4.4 预测名
    - 6.4.5 临时文件位置
  - 6.5 shutil—高级文件操作
    - 6.5.1 复制文件
    - 6.5.2 复制文件元数据
    - 6.5.3 处理目录树
  - 6.6 mmap—内存映射文件
    - 6.6.1 读文件
    - 6.6.2 写文件

## &lt;&lt;Python标准库&gt;&gt;

- 6.6.3 正则表达式
- 6.7 codecs—字符串编码和解码
  - 6.7.1 Unicode入门
  - 6.7.2 处理文件
  - 6.7.3 字节序
  - 6.7.4 错误处理
  - 6.7.5 标准输入和输出流
  - 6.7.6 编码转换
  - 6.7.7 非Unicode编码
  - 6.7.8 增量编码
  - 6.7.9 Unicode数据和网络通信
  - 6.7.10 定义定制编码
- 6.8 StringIO—提供类文件API的文本缓冲区
- 6.9 fnmatch—UNIX式glob模式匹配
  - 6.9.1 简单匹配
  - 6.9.2 过滤
  - 6.9.3 转换模式
- 6.10 dircache—缓存目录列表
  - 6.10.1 列出目录内容
  - 6.10.2 标注列表
- 6.11 filecmp—比较文件
  - 6.11.1 示例数据
  - 6.11.2 比较文件
  - 6.11.3 比较目录
  - 6.11.4 程序中使用差异
- 第7章 数据持久存储与交换
  - 7.1 pickle—对象串行化
    - 7.1.1 导入
    - 7.1.2 编码和解码字符串数据
    - 7.1.3 处理流
    - 7.1.4 重构对象的问题
    - 7.1.5 不可pickle的对象
    - 7.1.6 循环引用
  - 7.2 shelve—对象持久存储
    - 7.2.1 创建一个新shelf
    - 7.2.2 写回
    - 7.2.3 特定shelf类型
  - 7.3 anydbm—DBM数据库
    - 7.3.1 数据库类型
    - 7.3.2 创建一个新数据库
    - 7.3.3 打开一个现有数据库
    - 7.3.4 错误情况
  - 7.4 whichdb—识别DBM数据库格式
  - 7.5 sqlite3—嵌入式关系数据库
    - 7.5.1 创建数据库
    - 7.5.2 获取数据
    - 7.5.3 查询元数据

## &lt;&lt;Python标准库&gt;&gt;

- 7.5.4 行对象
- 7.5.5 查询中使用变量
- 7.5.6 批量加载
- 7.5.7 定义新列类型
- 7.5.8 确定列类型
- 7.5.9 事务
- 7.5.10 隔离级别
- 7.5.11 内存中数据库
- 7.5.12 导出数据库内容
- 7.5.13 SQL中使用Python函数
- 7.5.14 定制聚集
- 7.5.15 定制排序
- 7.5.16 线程和连接共享
- 7.5.17 限制对数据的访问
- 7.6 xml.etree.ElementTree—XML操纵API
  - 7.6.1 解析XML文档
  - 7.6.2 遍历解析树
  - 7.6.3 查找文档中的节点
  - 7.6.4 解析节点属性
  - 7.6.5 解析时监视事件
  - 7.6.6 创建一个定制树构造器
  - 7.6.7 解析串
  - 7.6.8 用元素节点构造文档
  - 7.6.9 美观打印XML
  - 7.6.10 设置元素属性
  - 7.6.11 由节点列表构造树
  - 7.6.12 将XML串行化至一个流
- 7.7 csv—逗号分隔值文件
  - 7.7.1 读文件
  - 7.7.2 写文件
  - 7.7.3 方言
  - 7.7.4 使用字段名
- 第8章 数据压缩与归档
  - 8.1 zlib—GNU zlib压缩
    - 8.1.1 处理内存中数据
    - 8.1.2 增量压缩与解压缩
    - 8.1.3 混合内容流
    - 8.1.4 校验和
    - 8.1.5 压缩网络数据
  - 8.2 gzip—读写GNU Zip文件
    - 8.2.1 写压缩文件
    - 8.2.2 读压缩数据
    - 8.2.3 处理流
  - 8.3 bz2—bzip2压缩
    - 8.3.1 内存中一次性操作
    - 8.3.2 增量压缩和解压缩
    - 8.3.3 混合内容流

## &lt;&lt;Python标准库&gt;&gt;

- 8.3.4 写压缩文件
- 8.3.5 读压缩文件
- 8.3.6 压缩网络数据
- 8.4 tarfile—Tar归档访问
  - 8.4.1 测试Tar文件
  - 8.4.2 从归档文件读取元数据
  - 8.4.3 从归档抽取文件
  - 8.4.4 创建新归档
  - 8.4.5 使用候选归档成员名
  - 8.4.6 从非文件源写数据
  - 8.4.7 追加到归档
  - 8.4.8 处理压缩归档
- 8.5 zipfile—ZIP归档访问
  - 8.5.1 测试ZIP文件
  - 8.5.2 从归档读取元数据
  - 8.5.3 从归档抽取归档文件
  - 8.5.4 创建新归档
  - 8.5.5 使用候选归档成员名
  - 8.5.6 从非文件源写数据
  - 8.5.7 利用ZipInfo实例写
  - 8.5.8 追加到文件
  - 8.5.9 Python ZIP归档
  - 8.5.10 限制
- 第9章 加密
  - 9.1 hashlib—密码散列
    - 9.1.1 示例数据
    - 9.1.2 MD5示例
    - 9.1.3 SHA1示例
    - 9.1.4 按名创建散列
    - 9.1.5 增量更新
  - 9.2 hmac—密码消息签名与验证
    - 9.2.1 消息签名
    - 9.2.2 SHA与MD
    - 9.2.3 二进制摘要
    - 9.2.4 消息签名的应用
- 第10章 进程与线程
  - 10.1 subprocess—创建附加进程
    - 10.1.1 运行外部命令
    - 10.1.2 直接处理管道
    - 10.1.3 连接管道段
    - 10.1.4 与其他命令交互
    - 10.1.5 进程间传递信号
  - 10.2 signal—异步系统事件
    - 10.2.1 接收信号
    - 10.2.2 获取注册的处理程序
    - 10.2.3 发送信号
    - 10.2.4 闹铃

## &lt;&lt;Python标准库&gt;&gt;

- 10.2.5 忽略信号
- 10.2.6 信号和线程
- 10.3 threading—管理并发操作
  - 10.3.1 Thread对象
  - 10.3.2 确定当前线程
  - 10.3.3 守护与非守护线程
  - 10.3.4 列举所有线程
  - 10.3.5 派生线程
  - 10.3.6 定时器线程
  - 10.3.7 线程间传送信号
  - 10.3.8 控制资源访问
  - 10.3.9 同步线程
  - 10.3.10 限制资源的并发访问
  - 10.3.11 线程特定数据
- 10.4 multiprocessing—像线程一样管理进程
  - 10.4.1 multiprocessing基础
  - 10.4.2 可导入的目标函数
  - 10.4.3 确定当前进程
  - 10.4.4 守护进程
  - 10.4.5 等待进程
  - 10.4.6 终止进程
  - 10.4.7 进程退出状态
  - 10.4.8 日志
  - 10.4.9 派生进程
  - 10.4.10 向进程传递消息
  - 10.4.11 进程间信号传输
  - 10.4.12 控制资源访问
  - 10.4.13 同步操作
  - 10.4.14 控制资源的并发访问
  - 10.4.15 管理共享状态
  - 10.4.16 共享命名空间
  - 10.4.17 进程池
  - 10.4.18 实现MapReduce
- 第11章 网络通信
  - 11.1 socket—网络通信
    - 11.1.1 寻址、协议簇和套接字类型
    - 11.1.2 TCP/IP客户和服务
    - 11.1.3 用户数据报客户和服务
    - 11.1.4 UNIX域套接字
    - 11.1.5 组播
    - 11.1.6 发送二进制数据
    - 11.1.7 非阻塞通信和超时
  - 11.2 select—高效等待I/O
    - 11.2.1 使用select()
    - 11.2.2 有超时的非阻塞I/O
    - 11.2.3 使用poll()
    - 11.2.4 平台特定选项

## &lt;&lt;Python标准库&gt;&gt;

## 11.3 SocketServer—创建网络服务器

## 11.3.1 服务器类型

## 11.3.2 服务器对象

## 11.3.3 实现服务器

## 11.3.4 请求处理器

## 11.3.5 回应示例

## 11.3.6 线程和进程

## 11.4 asyncore—异步I/O

## 11.4.1 服务器

## 11.4.2 客户

## 11.4.3 事件循环

## 11.4.4 处理其他事件循环

## 11.4.5 处理文件

## 11.5 asynchat—异步协议处理器

## 11.5.1 消息终止符

## 11.5.2 服务器和处理器

## 11.5.3 客户

## 11.5.4 集成

## 第12章 Internet

## 12.1 urlparse—分解URL

## 12.1.1 解析

## 12.1.2 反解析

## 12.1.3 连接

## 12.2 BaseHTTPServer—实现Web服务器的基类

## 12.2.1 HTTP GET

## 12.2.2 HTTP POST

## 12.2.3 线程与进程

## 12.2.4 处理错误

## 12.2.5 设置首部

## 12.3 urllib—网络资源访问

## 12.3.1 利用缓存实现简单获取

## 12.3.2 参数编码

## 12.3.3 路径与URL

## 12.4 urllib2—网络资源访问

## 12.4.1 HTTP GET

## 12.4.2 参数编码

## 12.4.3 HTTP POST

## 12.4.4 增加发出首部

## 12.4.5 从请求提交表单数据

## 12.4.6 上传文件

## 12.4.7 创建定制协议处理器

## 12.5 Base64—用ASCII编码二进制数据

## 12.5.1 Base64编码

## 12.5.2 Base64解码

## 12.5.3 URL安全的变种

## 12.5.4 其他编码

## 12.6 robotparser—网络蜘蛛访问控制

## &lt;&lt;Python标准库&gt;&gt;

- 12.6.1 robots.txt
- 12.6.2 测试访问权限
- 12.6.3 长久蜘蛛
- 12.7 Cookie—HTTP Cookie
  - 12.7.1 创建和设置Cookie
  - 12.7.2 Morsel
  - 12.7.3 编码值
  - 12.7.4 接收和解析Cookie首部
  - 12.7.5 候选输出格式
  - 12.7.6 废弃的类
- 12.8 uuid—全局惟一标识符
  - 12.8.1 UUID 1—IEEE 802 MAC地址
  - 12.8.2 UUID 3和5—基于名字的值
  - 12.8.3 UUID 4—随机值
  - 12.8.4 处理UUID对象
- 12.9 json—JavaScript对象记法
  - 12.9.1 编码和解码简单数据类型
  - 12.9.2 优质输出和紧凑输出
  - 12.9.3 编码字典
  - 12.9.4 处理定制类型
  - 12.9.5 编码器和解码器类
  - 12.9.6 处理流和文件
  - 12.9.7 混合数据流
- 12.10 xmlrpclib—XML-RPC的客户端库
  - 12.10.1 连接服务器
  - 12.10.2 数据类型
  - 12.10.3 传递对象
  - 12.10.4 二进制数据
  - 12.10.5 异常处理
  - 12.10.6 将调用结合在一个消息中
- 12.11 SimpleXMLRPCServer—一个XML-RPC服务器
  - 12.11.1 一个简单的服务器
  - 12.11.2 备用API名
  - 12.11.3 加点的API名
  - 12.11.4 任意API名
  - 12.11.5 公布对象的方法
  - 12.11.6 分派调用
  - 12.11.7 自省API
- 第13章 Email
  - 13.1 smtpplib—简单邮件传输协议客户
    - 13.1.1 发送Email消息
    - 13.1.2 认证和加密
    - 13.1.3 验证Email地址
  - 13.2 smtpd—示例邮件服务器
    - 13.2.1 邮件服务器基类
    - 13.2.2 调试服务器
    - 13.2.3 代理服务器

## &lt;&lt;Python标准库&gt;&gt;

- 13.3 imaplib—IMAP4客户库
  - 13.3.1 变种
  - 13.3.2 连接到服务器
  - 13.3.3 示例配置
  - 13.3.4 列出邮箱
  - 13.3.5 邮箱状态
  - 13.3.6 选择邮箱
  - 13.3.7 搜索消息
  - 13.3.8 搜索规则
  - 13.3.9 获取消息
  - 13.3.10 完整消息
  - 13.3.11 上传消息
  - 13.3.12 移动和复制消息
  - 13.3.13 删除消息
- 13.4 mailbox—管理邮件归档
  - 13.4.1 mbox
  - 13.4.2 Maildir
  - 13.4.3 其他格式
- 第14章 应用构建模块
  - 14.1 getopt—命令行选项解析
    - 14.1.1 函数参数
    - 14.1.2 短格式选项
    - 14.1.3 长格式选项
    - 14.1.4 一个完整的例子
    - 14.1.5 缩写长格式选项
    - 14.1.6 GNU选项解析
    - 14.1.7 结束参数处理
  - 14.2 optparse—命令行选项解析器
    - 14.2.1 创建OptionParser
    - 14.2.2 短格式和长格式选项
    - 14.2.3 用getopt比较
    - 14.2.4 选项值
    - 14.2.5 选项动作
    - 14.2.6 帮助消息
  - 14.3 argparse—命令行选项和参数解析
    - 14.3.1 与optparse比较
    - 14.3.2 建立解析器
    - 14.3.3 定义参数
    - 14.3.4 解析命令行
    - 14.3.5 简单示例
    - 14.3.6 自动生成的选项
    - 14.3.7 解析器组织
    - 14.3.8 高级参数处理
  - 14.4 readline—GNU Readline库
    - 14.4.1 配置
    - 14.4.2 完成文本
    - 14.4.3 访问完成缓冲区

## &lt;&lt;Python标准库&gt;&gt;

- 14.4.4 输入历史
- 14.4.5 hook
- 14.5 getpass—安全密码提示
  - 14.5.1 示例
  - 14.5.2 无终端使用getpass
- 14.6 cmd—面向行的命令处理器
  - 14.6.1 处理命令
  - 14.6.2 命令参数
  - 14.6.3 现场帮助
  - 14.6.4 自动完成
  - 14.6.5 覆盖基类方法
  - 14.6.6 通过属性配置Cmd
  - 14.6.7 运行shell命令
  - 14.6.8 候选输入
  - 14.6.9 sys.argv的命令
- 14.7 shlex—解析shell语法
  - 14.7.1 加引号的字符串
  - 14.7.2 嵌入注释
  - 14.7.3 分解
  - 14.7.4 包含其他Token源
  - 14.7.5 控制解析器
  - 14.7.6 错误处理
  - 14.7.7 POSIX与非POSIX解析
- 14.8 ConfigParser—处理配置文件
  - 14.8.1 配置文件格式
  - 14.8.2 读取配置文件
  - 14.8.3 访问配置设置
  - 14.8.4 修改设置
  - 14.8.5 保存配置文件
  - 14.8.6 选项搜索路径
  - 14.8.7 用接合合并值
- 14.9 日志—报告状态、错误和信息消息
  - 14.9.1 应用与库中的日志记录
  - 14.9.2 记入文件
  - 14.9.3 旋转日志文件
  - 14.9.4 详细级别
  - 14.9.5 命名日志记录器实例
- 14.10 fileinput—命令行过滤器框架
  - 14.10.1 M3U文件转换为RSS
  - 14.10.2 进度元数据
  - 14.10.3 原地过滤
- 14.11 atexit—程序关闭回调
  - 14.11.1 示例
  - 14.11.2 什么情况下不调用atexit函数
  - 14.11.3 处理异常
- 14.12 sched—定时事件调度器
  - 14.12.1 有延迟地运行事件

## &lt;&lt;Python标准库&gt;&gt;

- 14.12.2 重叠事件
- 14.12.3 事件优先级
- 14.12.4 取消事件
- 第15章 国际化和本地化
- 15.1 gettext—消息编目
  - 15.1.1 转换工作流程概述
  - 15.1.2 由源代码创建消息编目
  - 15.1.3 运行时查找消息编目
  - 15.1.4 复数值
  - 15.1.5 应用与模块本地化
  - 15.1.6 切换转换
- 15.2 locale—文化本地化API
  - 15.2.1 探查当前本地化环境
  - 15.2.2 货币
  - 15.2.3 格式化数字
  - 15.2.4 解析数字
  - 15.2.5 日期和时间
- 第16章 开发工具
- 16.1 pydoc—模块的联机帮助
  - 16.1.1 纯文本帮助
  - 16.1.2 HTML帮助
  - 16.1.3 交互式帮助
- 16.2 doctest—通过文档完成测试
  - 16.2.1 开始
  - 16.2.2 处理不可预测的输出
  - 16.2.3 Traceback
  - 16.2.4 避开空白符
  - 16.2.5 测试位置
  - 16.2.6 外部文档
  - 16.2.7 运行测试
  - 16.2.8 测试上下文
- 16.3 unittest—自动测试框架
  - 16.3.1 基本测试结构
  - 16.3.2 运行测试
  - 16.3.3 测试结果
  - 16.3.4 断言真值
  - 16.3.5 测试相等性
  - 16.3.6 近似相等
  - 16.3.7 测试异常
  - 16.3.8 测试固件
  - 16.3.9 测试套件
- 16.4 traceback—异常和栈轨迹
  - 16.4.1 支持函数
  - 16.4.2 处理异常
  - 16.4.3 处理栈
- 16.5 cgitb—详细的traceback报告
  - 16.5.1 标准traceback转储

## &lt;&lt;Python标准库&gt;&gt;

- 16.5.2 启用详细traceback
- 16.5.3 traceback中的局部变量
- 16.5.4 异常属性
- 16.5.5 HTML输出
- 16.5.6 记录traceback
- 16.6 pdb—交互式调试工具
  - 16.6.1 启动调试工具
  - 16.6.2 控制调试工具
  - 16.6.3 断点
  - 16.6.4 改变执行流
  - 16.6.5 用别名定制调试工具
  - 16.6.6 保存配置设置
- 16.7 trace—执行程序流
  - 16.7.1 示例程序
  - 16.7.2 跟踪执行
  - 16.7.3 代码覆盖
  - 16.7.4 调用关系
  - 16.7.5 编程接口
  - 16.7.6 保存结果数据
  - 16.7.7 选项
- 16.8 profile和pstats—性能分析
  - 16.8.1 运行性能分析工具
  - 16.8.2 在上下文中运行
  - 16.8.3 pstats：保存和处理统计信息
  - 16.8.4 限制报告内容
  - 16.8.5 调用图
- 16.9 timeit—测量小段Python代码的执行时间
  - 16.9.1 模块内容
  - 16.9.2 基本示例
  - 16.9.3 值存储在字典中
  - 16.9.4 从命令行执行
- 16.10 compileall—字节编译源文件
  - 16.10.1 编译一个目录
  - 16.10.2 编译sys.path
  - 16.10.3 从命令行执行
- 16.11 pycldr—类浏览器
  - 16.11.1 扫描类
  - 16.11.2 扫描函数
- 第17章 运行时特性
  - 17.1 site—全站点配置
    - 17.1.1 导入路径
    - 17.1.2 用户目录
    - 17.1.3 路径配置文件
    - 17.1.4 定制站点配置
    - 17.1.5 定制用户配置
    - 17.1.6 禁用site模块
  - 17.2 sys—系统特定的配置

## &lt;&lt;Python标准库&gt;&gt;

- 17.2.1 解释器设置
- 17.2.2 运行时环境
- 17.2.3 内存管理和限制
- 17.2.4 异常处理
- 17.2.5 底层线程支持
- 17.2.6 模块和导入
- 17.2.7 跟踪程序运行情况
- 17.3 os—可移植访问操作系统特定特性
  - 17.3.1 进程所有者
  - 17.3.2 进程环境
  - 17.3.3 进程工作目录
  - 17.3.4 管道
  - 17.3.5 文件描述符
  - 17.3.6 文件系统权限
  - 17.3.7 目录
  - 17.3.8 符号链接
  - 17.3.9 遍历目录树
  - 17.3.10 运行外部命令
  - 17.3.11 用os.fork()创建进程
  - 17.3.12 等待子进程
  - 17.3.13 Spawn
  - 17.3.14 文件系统权限
- 17.4 platform—系统版本信息
  - 17.4.1 解释器
  - 17.4.2 平台
  - 17.4.3 操作系统和硬件信息
  - 17.4.4 可执行程序体系结构
- 17.5 resource—系统资源管理
  - 17.5.1 当前使用情况
  - 17.5.2 资源限制
- 17.6 gc—垃圾回收器
  - 17.6.1 跟踪引用
  - 17.6.2 强制垃圾回收
  - 17.6.3 查找无法收集的对象引用
  - 17.6.4 回收阈限和代
  - 17.6.5 调试
- 17.7 sysconfig—解释器编译时配置
  - 17.7.1 配置变量
  - 17.7.2 安装路径
  - 17.7.3 Python版本和平台
- 第18章 语言工具
  - 18.1 warnings—非致命警告
    - 18.1.1 分类和过滤
    - 18.1.2 生成警告
    - 18.1.3 用模式过滤
    - 18.1.4 重复的警告
    - 18.1.5 候选消息传送函数

## &lt;&lt;Python标准库&gt;&gt;

- 18.1.6 格式化
- 18.1.7 警告中的栈层次
- 18.2 abc—抽象基类
  - 18.2.1 为什么使用抽象基类
  - 18.2.2 抽象基类如何工作
  - 18.2.3 注册一个具体类
  - 18.2.4 通过派生实现
  - 18.2.5 abc中的具体方法
  - 18.2.6 抽象属性
- 18.3 dis—Python字节码反汇编工具
  - 18.3.1 基本反汇编
  - 18.3.2 反汇编函数
  - 18.3.3 类
  - 18.3.4 使用反汇编进行调试
  - 18.3.5 循环的性能分析
  - 18.3.6 编译器优化
- 18.4 inspect—检查现场对象
  - 18.4.1 示例模块
  - 18.4.2 模块信息
  - 18.4.3 检查模块
  - 18.4.4 检查类
  - 18.4.5 文档串
  - 18.4.6 获取源代码
  - 18.4.7 方法和函数参数
  - 18.4.8 类层次结构
  - 18.4.9 方法解析顺序
  - 18.4.10 栈与帧
- 18.5 exceptions—内置异常类
  - 18.5.1 基类
  - 18.5.2 产生的异常
  - 18.5.3 警告类型
- 第19章 模块与包
  - 19.1 imp—Python的导入机制
    - 19.1.1 示例包
    - 19.1.2 模块类型
    - 19.1.3 查找模块
    - 19.1.4 加载模块
  - 19.2 zipimport—从ZIP归档加载Python代码
    - 19.2.1 示例
    - 19.2.2 查找模块
    - 19.2.3 访问代码
    - 19.2.4 源代码
    - 19.2.5 包
    - 19.2.6 数据
  - 19.3 pkgutil—包工具
    - 19.3.1 包导入路径
    - 19.3.2 包的开发版本

## <<Python标准库>>

19.3.3 用PKG文件管理路径

19.3.4 嵌套包

19.3.5 包数据

## &lt;&lt;Python标准库&gt;&gt;

## 章节摘录

版权页： 垂持久存储数据以供长期使用，这包括两个方面：在对象的内存中表示和存储格式之间来回转换数据，以及处理转换后数据的存储区。

标准库包含很多模块，可以在不同情况下处理这两个方面。

有两个模块可以将对象转换为一种可以传输或存储的格式[这个过程称为串行化 ( serializing ) ]。

最常用的是使用pickle完成持久存储，因为它可以与其他一些具体存储串行化数据的标准库模块集成，如shelve。

不过对基于Web的应用，json则更为常用，因为它能更好地与现有Web服务存储工具集成。

一旦将内存中对象转换为一种可以保存的格式，下一步就是确定如何存储这个数据。

如果数据不需要以某种方式索引，依序先后写入串行化对象的简单平面文件就很适用。

Python包括一组模块可以在一个简单的数据库中存储键-值对，需要索引查找时会使用某种变种DBM格式。

要利用DBM格式，最直接的方式就是shelve。

可以打开shelve文件，并通过一个类字典的API来访问。

保存到数据库的对象会自动pickle并保存，而无须调用者做任何额外的工作。

不过shelve有一个缺点，使用默认接口时，没有办法预测将使用哪一个DBM格式，因为它会根据创建数据库的系统上有哪些可用的库来进行选择。

如果应用不需要在库配置不同的主机之间共享数据库文件，那么选择哪种格式并不重要；不过，如果必须保证可移植性，可以使用这个模块中的某个类来确保选择一个特定的格式。

对于Web应用，由于这些应用处理的就是JSON格式的数据，因此使用json和anydbm可以提供另一种持久存储机制。

直接使用anydbm会比shelve稍微多做一些工作，因为DBM数据库键和值必须是字符串，而且在数据库中访问值时不会自动地重新创建对象。

大多数Python发布版本都提供了sqlite3进程中关系数据库，可以采用比键-值对更复杂的组织来存储数据。

它将数据库存储在内存中或者存储在一个本地文件中，所有访问都来自同一个进程，所以不存在网络通信延迟。

sqlite3的紧凑性使它尤其适合嵌入到桌面应用或Web应用的开发版本中。

## <<Python标准库>>

### 编辑推荐

《Python标准库》的读者应该是中等水平的Python程序员，所以尽管书中对所有源代码都做了讨论，但只有一部分会逐行给出解释。

每节会通过源代码和完全独立的示例程序的输出来重点介绍一个模块的特性。

我会尽可能简洁地介绍各个特性，使读者能够把重点放在所展示的模块或函数上，而不会因支持代码而分心。

## <<Python标准库>>

### 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>