16. 附录

16.1. 交互模式

16.1.1. 错误处理

当发生错误时，解释器会打印错误信息和错误堆栈。在交互模式下，将返回到主命令提示符；如果输入内容来自文件，在打印错误堆栈之后，程序会以非零状态退出。（这里所说的错误不包括 try 语句中由 except 所捕获的异常。）有些错误是无条件致命的，会导致程序以非零状态退出；比如内部逻辑矛盾或内存耗尽。所有错误信息都会被写入标准错误流；而命令的正常输出则被写入标准输出流。

将中断字符（通常为 Control-C 或 Delete ）键入主要或辅助提示会取消输入并返回主提示符。 [1] 在执行命令时键入中断引发的 KeyboardInterrupt 异常，可以由 try 语句处理。

16.1.2. 可执行的Python脚本

在BSD等类Unix系统上，Python脚本可以直接执行，就像shell脚本一样，第一行添加:

```
#!/usr/bin/env python3.5
```

（假设解释器位于用户的 PATH ）脚本的开头，并将文件设置为可执行。 #! 必须是文件的前两个字符。在某些平台上，第一行必须以Unix样式的行结尾（'\n'）结束，而不是以Windows（'\r\n'）行结尾。请注意，散列或磅字符 '#' 在Python中代表注释开始。

可以使用 **chmod** 命令为脚本提供可执行模式或权限。

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

在Windows系统上，没有“可执行模式”的概念。 Python安装程序自动将 .py 文件与 python.exe 相关联，这样双击Python文件就会将其作为脚本运行。扩展也可以是 .pyw ，在这种情况下，会隐藏通常出现的控制台窗口。

16.1.3. 交互式启动文件

当您以交互方式使用Python时，每次启动解释器时都会执行一些标准命令，这通常很方便。您可以通过将名为 PYTHONSTARTUP 的环境变量设置为包含启动命令的文件名来实现。这类似于Unix shell的 .profile 功能。

此文件只会在交互式会话时读取，而非在 Python 从脚本读取指令或是在给定 /dev/tty 为指令的明确来源时（后者反而表现得像是一个交互式会话）。 该文件执行时所在的命名空间与交互式指令相同，所以它定义或导入的对象可以在交互式会话中直接使用。 你也可以在该文件中更改提示符 sys.ps1 和 sys.ps2。

如果你想从当前目录中读取一个额外的启动文件，你可以使用像 if os.path.isfile('.pythonrc.py'): exec(open('.pythonrc.py').read()) 这样的代码在全局启动文件中对它进行编程。如果要在脚本中使用启动文件，则必须在脚本中显式执行此操作:

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
    with open(filename) as fobj:
        startup_file = fobj.read()
    exec(startup_file)
```

16.1.4. 定制模块

Python提供了两个钩子来让你自定义它：sitecustomize 和 usercustomize。要查看其工作原理，首先需要找到用户site-packages目录的位置。启动Python并运行此代码:

```
>>> import site
>>> site.getusersitepackages()
'/home/user/.local/lib/python3.5/site-packages'
```

现在，您可以在该目录中创建一个名为 usercustomize.py 的文件，并将所需内容放入其中。它会影响Python的每次启动，除非它以 -s 选项启动，以禁用自动导入。

sitecustomize 以相同的方式工作，但通常由计算机管理员在全局 site-packages 目录中创建，并在 usercustomize 之前被导入。有关详情请参阅 site 模块的文档。

备注

[1] GNU Readline 包的问题可能会阻止这种情况。

3.10.7

转向