执行模型(Execution model)

ZhangXu

2019 年 1 月 28 日

1 程序的结构

程序由代码块构成，*block*是一段Python程序文本，作为一个执行单元。包括:**一个模块，一个函数体和一个类定义**。交互式输入的每个命令都是一个块。脚本文件（作为解释器的标准输入提供的文件或指定为解释器的命令行参数的文件）是代码块。脚本命令（在带有-c选项的解释器命令行上指定的命令）是代码块。传递给内置函数eval（）和exec（）的字符串参数是一个代码块。

代码块在*executionframe*执行。一个框架包含一些管理信息（用于调试），并确定代码块执行完成后执行的持续时间和方式。

2 命名和绑定(Naming and binding)

2.1 绑定名称

名称指向对象。名称由名称绑定操作引入。

以下构造绑定名称:形式参数到函数，import语句，类和函数定义(这些定义绑定定义块中的类或函数名称)，以及在赋值过程中作为标识符的目标，for循环标头，或者在with语句中as之后的部分或except子句中。来自from...import *的Import语句绑定导入模块中定义的所有名称，但以下划线开头的名称除外。此表单仅能在模块级别使用。

在del语句中出现的目标也被认为是为此目的绑定的（尽管实际的语义是取消绑定名称）。

每个赋值或导入语句都发生在由类或函数定义或模块级别（顶级代码块）定义的块中。

如果名称绑定在块中，则它是该块的局部变量，除非声明为nonlocal或global。如果名称在模块级别绑定，则它是全局变量。(模块代码块的变量是局部的和全

局的)如果在代码块中使用了变量但在那里没有定义，则它是一个free variable。

程序文本中每次出现的名称都是指由以下名称解析规则建立的名称的绑定。

2.2 名称解析

*scope*定义块内的可见性。如果在块中定义了局部变量，则其范围包括该块。如果定义发生在函数块中，则作用域将扩展到定义块中包含的任何块，除非包含的块为名称引入了不同的绑定。(即函数局部变量可被函数中更小代码块相同的局部变量名称而顶替，虽名称相同，但其无关)

在代码块中使用名称时，将使用最近的封闭范围解析该名称。代码块可见的所有此类范围的集合称为块的环境（*evironment*）。

如果根本找不到名称，则会引发NameError异常。如果当前作用域是函数作用域，并且名称引用尚未绑定到使用该名称的值的本地变量，则会引发UnboundLocalError异常。 UnboundLocalError是NameError的子类。

如果名称绑定操作发生在代码块中的任何位置，则块中名称的所有使用都将被视为对当前块的引用。在绑定之前在块中使用名称时，这可能会导致错误。这条规则很微妙。Python缺少声明，并允许在代码块中的任何位置进行名称绑定操作。可以通过扫描块的整个文本以确定名称绑定操作来确定代码块的局部变量。

如果global语句发生在块中，则语句中指定的名称的所有使用都将引用该名称在顶级命名空间中的绑定。通过搜索全局命名空间在顶级命名空间中解析名称，例如包含代码块的模块的命名空间，以及内置命名空间，builtins模块的命名空间。首先搜索全局命名空间。如果在那里找不到名称，则搜索内置命名空间。global语句必须在名称的所有使用之前。

global语句与同一块中的名称绑定操作具有相同的范围。如果自由变量的最近封闭范围包含全局语句，则将free variable视为全局变量。

nonlocal语句使相应的名称引用最近的封闭函数范围中的先前绑定的变量。如果任何封闭的函数作用域中不存在给定的名称，则在编译时引发SyntaxError。

模块的命名空间在第一次导入模块时自动创建。脚本的主要模块始终称为__main__。

exec（）和eval（）的类定义块和参数在名称解析的上下文中是特殊的。类定义是可以使用和定义名称的可执行语句。这些引用遵循名称解析的常规规则，有一个例外是：未绑定的局部变量在全局命名空间中查找。**类定义的名称空间成为类的属性字典**。类块中定义的名称范围仅限于类块;它没有扩展到方法的代

码块 - 这包括comprehensions和generator expressions，因为它们是使用函数作用域实现的。

2.3 内置和限制执行

Cpython implementation detail: 用户不应该触摸__builtins__;它严格来说是一个实现细节。想要覆盖builtsin命名空间中的值的用户应该import builtsin模块并适当地修改其属性。

与代码块执行相关联的builtins命名空间实际上是通过在其全局命名空间中查找名称__builtins__来找到的;这应该是字典或模块（在后一种情况下，使用模块的字典）。默认情况下，在__main__模块中，__builtins__是内置builtins模块;在任何其他模块中，__ builtins__是bulitins模块本身的字典的别名。

2.4 与动态功能的交互

free variables的名称解析在运行时发生，而不是在编译时发生。这意味着以下代码将打印42：

- i=10
- def f():
- print(i)
- i=42
- f()

eval（）和exec（）函数无权访问用于解析名称的完整环境。可以在调用者的本地和全局名称空间中解析名称。free variable不在最近的封闭命名空间中解析，而是在全局命名空间中解析。exec（）和eval（）函数具有可选参数以覆盖全局和本地命名空间。如果仅指定了一个名称空间，则将其用于两者。

3 Exceptions 例外

例外是一种打破代码块正常控制流程以处理错误或其他异常情况的方法。检测到错误时会一个异常被*raised*;它可以由周围的代码块或直接或间接调用发生错误的代码块处理。

Python解释器在检测到运行时错误（例如除以零）时引发异常。 Python程

序也可以使用raise语句显式引发异常。使用try ... except语句指定异常处理程序。这种语句的finally子句可用于指定不处理异常的清理代码，但是在前面的代码中是否发生异常时执行。

Python使用错误处理的“termination”模型：异常处理程序可以找出发生了什么并继续在外层执行，但它无法修复错误原因并重试失败操作（除非从顶部重新输入有问题代码的部分）。

当根本不处理异常时，解释器终止程序的执行，或返回其交互式主循环。在任何一种情况下，它都会打印堆栈回溯，除非异常是SystemExit。

例外由类实例标识。根据实例的类选择except子句：它必须引用实例的类或其基类。该实例可以由处理程序接收，并且可以携带关于异常情况的附加信息。

NOTE: 异常消息不是Python API的一部分。它们的内容可能会在没有警告的情况下从一个版本的Python更改为下一个版本，并且不应该依赖于将在多个版本的解释器下运行的代码。