第十章：错误和异常

本章将介绍什么是异常, 异常处理, 以及 Python 对异常的支持. 我们还会介绍如何在代码里生成异常. 最后, 我们会涉及如何创建自定义的异常类.

目录

[10.1异常 1](#_Toc350539884)

[10.2python中的检测和处理异常 2](#_Toc350539885)

[1、try-except 2](#_Toc350539886)

[2、try-finally 4](#_Toc350539887)

[3、try-except-finally 复合语句. 4](#_Toc350539888)

[10.3触发异常 5](#_Toc350539889)

[10.4上下文管理 5](#_Toc350539890)

[10.5断言 5](#_Toc350539891)

[10.6标准异常 5](#_Toc350539892)

[10.7创建异常 5](#_Toc350539893)

## 10.1异常

1、异常基础

对异常的最好描述是: 它是因为程序出现了错误而在正常控制流以外采取的行为. 这个行为又分为两个阶段: 首先是引起异常发生的错误, 然后是检测(和采取可能的措施)阶段.

第一个阶段是在发生了一个异常条件(有时候也叫做例外的条件)后发生的. 只要检测到错误

并且意识到异常条件, 解释器会引发一个异常. 引发也可以叫做触发, 引发或者生成. 解释器通过它通知当前控制流有错误发生. Python 也允许程序员自己引发异常. 无论是 Python 解释器还是程序员引发的, 异常就是错误发生的信号. 当前流将被打断。

用来处理这个错误并采取相应的操作. 这就是第二阶段.对异常的处理发生在第二阶段. 异常引发后, 可以调用很多不同的操作. 可以是忽略错误(记录错误但不采取任何措施, 采取补救措施后终止程序), 或是减轻问题的影响后设法继续执行程序.所有的这些操作都代表一种继续, 或是控制的分支. 关键是程序员在错误发生时可以指示程序如何执行.

和其他支持异常处理的语言类似, Python 采用了 "try/尝试" 块和 "catching/捕获" 块的概念, 而且它在异常处理方面更有"纪律性". 我们可以为不同的异常创建不同的处理器, 而不是盲目地创建一个"catch-all/捕获所有"的代码.

2、Python 中的异常

NameError: 尝试访问一个未申明的变量

ZeroDivisionError: 除数为零

SyntaxError: Python 解释器语法错误

IndexError:请求的索引超出序列范围

KeyError:请求一个不存在的字典关键字

IOError: 输入/输出错误

AttributeError: 尝试访问未知的对象属性

3、所有异常的基类是Exception, 所以, 可以使用Exception捕获所有的异常…当然, 也可以使用裸except语句捕获多数的异常

4、异常发生时, 可以使用sys.exc\_info()或得到当前的系统执行信息, 得到的结果元组类似下面的格式: (<type 'exceptions.IndexError'>, IndexError('list index out of range',), <traceback object at 0x00C15EE0>) 第一个元素表明异常的类型, 第二个元素是异常, 第三个是堆栈信息

5、python2.5之后的异常体系结构

- BaseException

|- KeyboardInterrupt

|- SystemExit

|- Exception

|- (all other current built-in exceptions) 所有当前内建异常

## 10.2python中的检测和处理异常

try 语句有两种主要形式: try-except 和 try-finally .这两个语句是互斥的, 也就是说你只能使用其中的一种.

### 1、try-except

一个 try 语句可以对应一个或多个 except 子句，你可以使用 try-except 语句检测和处理异常. 你也可以添加一个可选的 else 子句处理没有探测到异常的时执行的代码.

1）

>>> try:

... f = open('blah', 'r')

... except IOError, e:

... print 'could not open file:', e

...

could not open file: [Errno 2] No such file or directory

2）带有多个 except 的 try 语句

def safe\_float(obj):

try:

retval = float(obj)

except ValueError:

retval = 'could not convert non-number to float'

except TypeError:

retval = 'object type cannot be converted to float'

return retval

3）处理多个异常的 except 语句

def safe\_float(obj):

try:

retval = float(obj)

except (ValueError, TypeError):

retval = 'argument must be a number or numeric string'

return retval

4）捕获所有异常

try:

:

except BaseException, e:

# handle all errors

底线: 避免把大片的代码装入 try-except 中然后使用 pass 忽略掉错误. 你可以捕获特定的异常并忽略它们, 或是捕获所有异常并采取特定的动作. 不要捕获所有异常,然后忽略掉它们.

5）else 子句

try:

3rd\_party\_module.function()

except:

log.write("\*\*\* caught exception in module\n")

else:

log.write("\*\*\* no exceptions caught\n")

log.close()

### 2、try-finally

一个 try 语句只能对应一个 finally 子。finally 子句是无论异常是否发生,是否捕捉都会执行的一段代码.

### 3、try-except-finally 复合语句.

下面是 try-except-else-finally 语法的示例:

try:

A

except MyException: B

else: C

finally: D

等价于 Python 0.9.6 至 2.4.x 中如下的写法:

try:

try:

A

except MyException:

B

else: C

finally:

D

当然,无论如何,你都可以有不止一个的 except 子句,但最少有一个 except 语句,而 else 和

finally 都是可选的.A,B,C 和 D 是程序(代码块).程序会按预期的顺序执行.(注意:可能的顺序是A-C-D[正常]或A-B-D[异常]).无论异常发生在A,B,和/或C都将执行finally块.旧式写法依然有效,所以没有向后兼容的问题.

## 10.3触发异常

到目前为止,我们所见到的异常都是由解释器引发的.由于执行期间的错误而引发.程序员在编写 API 时也希望在遇到错误的输入时触发异常,为此,Python 提供了一种机制让程序员明确的触发异常:这就是 raise 语句.

## 10.4上下文管理

## 10.5断言

## 10.6标准异常

## 10.7创建异常