《Python统计计算》

（秋季学期）

翟祥
北京林业大学

编程基础

第8章 异常处理

- 异常是指程序运行时引发的错误
- 引发错误的原因有很多，例如除零、下标越界、文件不存在、网络异常、类型错误、名字错误、字典键错误、磁盘空间不足，等等。
- 如果这些错误得不到正确的处理将会导致程序终止运行，而合理地使用异常处理结果可以使得程序更加健壮，具有更强的容错性。
- 也可以使用异常处理结构为用户提供更加友好的提示。

8.1 什么是异常

```
>>> x, y = 10, 5
>>> a = x / y
>>> A
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
print A
NameError: name 'A' is not defined

>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: division by zero
```

8.1 什么是异常

```
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: Can't convert 'int' object to str
    implicitly
```

8.1 什么是异常

- 语法错误和逻辑错误不属于异常，但有些语法错误往往会导致异常，例如由于大小写拼写错误而访问不存在的对象。
- 当Python检测到一个错误时，解释器就会指出当前流已无法继续执行下去，这时候就出现了异常。异常是指因为程序出错而在正常控制流以外采取的行为。

8.1 什么是异常

- 异常分为两个阶段：
  - 第一个阶段是引起异常发生的错误；
  - 第二个阶段是检测并处理阶段。
- 不建议使用异常来代替常规的检查，如if...else判断。
- 应避免过多依赖于异常处理机制。
- 当程序出现错误，python会自动引发异常，也可以通过raise显式地引发异常。

8.2 Python中的异常类

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
```

(续)

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

8.2 Python中的异常类

- 可以继承Python内置异常类来实现自定义的异常类。

```
#myexcept.py
#coding=gbk
class ShortInputException(Exception):
    '''你定义的异常类。'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast
#主程序
try:
    s = raw_input('请输入 --> ')
    if len(s) < 3:
        raise ShortInputException(len(s), 3)
except EOFError:
    print '你输入了一个结束标记EOF'
except ShortInputException, x:              #except ShortInputException as x:
    print 'ShortInputException: 输入的长度是 %d, 长度至少应是 %d' % (x.length, x.atleast)
else:
    print '没有异常发生.'
```

8.2 Python中的异常类

```
>>> class MyError(Exception):
        def __init__(self, value):
            self.value = value
        def __str__(self):
            return repr(self.value)
>>> try:
        raise MyError(2*2)
    except MyError as e:
        print('My exception occurred, value:', e.value)

My exception occurred, value: 4
>>> raise MyError('oops!')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
__main__.MyError: 'oops!'
```

8.2 Python中的异常类

- 如果自己编写的某个模块需要抛出多个不同的异常，可以先创建一个基类，然后创建多个派生类分别表示不同的异常。

```
class Error(Exception):
    pass


class InputError(Error):
    """Exception raised for errors in the input.
    Attributes:
        expression -- input expression in which the error
  occurred  message -- explanation of the error
    """
    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

```
class TransitionError(Error):
    """Raised when an operation attempts a state
 transition that's not allowed.
    Attributes:
        previous -- state at beginning of transition
        next -- attempted new state
        message -- explanation of why the specific
 transition is not allowed
    """
    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
```

8.3.1 try...except结构

- try子句中的代码块放置可能出现异常的语句，except子句中的代码块处理异常。

```
try:
    try块                    #被监控的语句
except Exception[ as reason]:
    except块                 #处理异常的语句
```

- 当需要捕获所有异常时，可以使用BaseException，代码格式如下：

```
try:
        ……
except BaseException as e: #不建议这样做
        ……                #处理所有错误
```

8.3.1 try...except结构

```
>>> while True:
        try:
            x = int(input("Please enter a number: "))
            break
        except ValueError:
            print( “It’s not a valid number.  Try again...")
```

8.3.1 try...except结构

- except子句可以在异常类名字后指定一个变量。

```
>>> try:
        raise Exception('spam', 'eggs')
    except Exception as inst:
        print(type(inst))    # the exception instance
        print(inst.args)     # arguments stored in .args
        print(inst)          # __str__ allows args to be printed directly,
                             # but may be overridden in exception subclasses
        x, y = inst.args     # unpack args
        print('x =', x)
        print('y =', y)

<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

8.3.2 try...except...else结构

- 如果try范围内捕获了异常，就执行except块；如果try范围内没有捕获异常，就执行else块。

```
a_list = ['China', 'America', 'England', 'France']
print '请输入字符串的序号'
while True:
    n = input()
    n = int(n)
    try:
        print(a_list[n])
    except IndexError:
        print('列表元素的下标越界，请重新输入字符串的序号')
    else:
        break
```

8.3.2 try...except...else结构

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

8.3.3 带有多个except的try结构

- 带有多个except的try结构

```
try:
    try块                                  #被监控的语句
except Exception1:
    except块1                              #处理异常1的语句
except Exception2:
    except块2                              #处理异常2的语句
```

8.3.3 带有多个except的try结构

```
try:
    x=input('请输入被除数: ')
    y=input('请输入除数: ')
    z=float(x) / y
except ZeroDivisionError:
    print '除数不能为零'
except TypeError:
    print '被除数和除数应为数值类型'
except NameError:
    print '变量不存在'
else:
    print x, '/', y, '=', z
```

8.3.3 带有多个except的try结构

```
import sys
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

8.3.3 带有多个except的try结构

- 将要捕获的异常写在一个元组中，可以使用一个except语句捕获多个异常：

```
import sys
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except (OSError, ValueError,RuntimeError, NameError):
    pass
```

8.3.4 try...except...finally结构

- 在该结构中，无论是否发生异常，finally子句中的内容都会执行，常用来做一些清理工作以释放try子句中申请的资源。

```
try:
        ……
finally:
        ……                  #无论如何都会执行
```

```
>>> try:
        3/0
    except:
        print(3)
    finally:
        print(5)
3
5
```

8.3.4 try...except...finally结构

```
try:
    f = open('test.txt', 'r')
    line = f.readline( )
    print line
finally:
    f.close( )
```

8.3.4 try...except...finally结构

- 上面的代码，使用异常处理结构的本意是为了防止文件读取操作出现异常而导致文件不能正常关闭，但是如果因为文件不存在而导致文件对象创建失败，那么finally子句中关闭文件对象的代码将会抛出异常从而导致程序终止运行。

```
>>> try:
        f = open('test.txt', 'r')
        line = f.readline()
        print(line)
    finally:
        f.close()

Traceback (most recent call last):
  File "<pyshell#17>", line 6, in <module>
    f.close()
NameError: name 'f' is not defined
```

8.3.4 try...except...finally结构

- 如果try子句中的异常没有被处理，或者在except子句或else子句中出现了异常，那么这些异常将会在finally子句执行完后再次抛出。

```
>>> try:
        3/0
    finally:
        print(5)

5
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    try:3/0
finally:
    print(5)
ZeroDivisionError: division by zero
```

8.3.4 try...except...finally结构

```
>>> def divide(x, y):
        try:
            result = x / y
        except ZeroDivisionError:
            print("division by zero!")
        else:  #没有异常时才执行
            print("result is", result)
        finally:
            print("executing finally clause")
```

```
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in divide
TypeError: unsupported operand
    type(s) for /: 'str' and 'str'
```

8.3.4 try...except...finally结构

- 最后，使用带有finally子句的异常处理结构时，应尽量避免在finally子句中使用return语句，否则可能会出现出乎意料的错误，例如下面的代码：

```
>>> def demo_div(a, b):
        try:
          return a/b
        except:
          pass
        finally:
          return -1
>>> demo_div(1, 0)
-1
>>> demo_div(1, 2)
-1
>>> demo_div(10, 2)
-1
```

8.4 断言与上下文管理

- 断言与上下文管理是两种比较特殊的异常处理方式，在形式上比异常处理结构要简单一些。

8.4.1 断言

- 断言语句的语法是：

  assert *expression[as reason]*

  当判断表达式expression为真时，什么都不做；如果表达式为假，则抛出异常。
- assert语句一般用于开发程序时对特定必须满足的条件进行验证，仅当__debug__为True时有效。当Python脚本以-O选项编译为字节码文件时，assert语句将被移除以提高运行速度。

8.4.1 断言

```
try:
    assert 1 == 2 , "1 is not equal 2!"
except AssertionError as  reason:
    print("%s:%s"%(reason.__class__.__name__, reason))
```

运行结果：

```
AssertionError:1 is not equal 2!
```

8.4.2 上下文管理

- 使用with自动关闭资源，可以在代码块执行完毕后还原进入该代码块时的现场。
- 不论何种原因跳出with块，不论是否发生异常，总能保证文件被正确关闭，资源被正确释放。
- with语句的语法如下：

```
with context_expr [as var]:
    with块
```

8.4.2 上下文管理

- Python2

```
with open('d:\\test.txt') as f:
    for line in f:
        print line
```

- Python3

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

8.5 用sys模块回溯最后的异常

```
import sys
try:
    block
except:
    tuple = sys.exc_info()
    print(tuple)
```

sys.exc_info( )的返回值tuple是一个三元组(type, value/message，traceback)，这里的

- type —— 异常的类型
- value/message —— 异常的信息或者参数
- traceback —— 包含调用栈信息的对象。

8.5 用sys模块回溯最后的异常

- 当发生异常时，Python会回溯异常，给出大量的提示，可能会给程序员的定位和纠错带来一定的困难，这时可以使用sys模块来回溯最近一次异常。

```
>>> 1/0

Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    1/0
ZeroDivisionError: integer division or modulo by zero>>> import sys
>>> try:
            1/0
        except:
            r = sys.exc_info()
            print(r)

(<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero',),
        <traceback object at 0x000000000375C788>)
```

8.5 用sys模块回溯最后的异常

- sys.exc_info()可以直接定位最终引发异常的原因，结果也比较简洁，但是缺点是难以直接确定引发异常的代码位置。

8.5 用sys模块回溯最后的异常

```
>>> def A():
1/0
>>> def B():
A()
>>> def C():
B()
>>> C()
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    C()
  File "<pyshell#34>", line 2, in C
    B()
  File "<pyshell#31>", line 2, in B
    A()
  File "<pyshell#28>", line 2, in A
    1/0
ZeroDivisionError: integer division or modulo by zero
>>> try:
        C()
    except:
        r = sys.exc_info()
        print(r)
(<type 'exceptions.ZeroDivisionError'>, ZeroDivisionError('integer division or modulo
by zero',), <traceback object at 0x0134C990>)
```

python™

谢谢Q/A