Python 语言程序设计

陈 峦 副教授

13880209111，chluan@uestc.edu.cn

研究院大楼316#

第十一章 异常处理

- 异常（exception）是指程序运行过程中出现的错误或遇到的意外情况。
- 引发异常的原因有很多，如除数为0、下标越界、文件不存在、数据类型错误、命名错误、内存空间不够、用户操作不当，等等。

- 如果这些异常得不到有效的处理，会导致程序终止运行。
- 一个好的程序，应具备较强的容错能力，即，除了在正常情况能够完成所预想的功能外，在遇到各种异常的情况下，也能够做出恰当的处理。
- 这种对异常情况给予适当处理的技术就是异常处理。

- Python提供了一套完整的异常处理方法，在一定程度上可以提高程序的健壮性，即程序在非正常环境下仍能正常运行，并能把Python晦涩难懂的错误信息转换为友好的提示呈现给最终用户。

11.1 异常处理概述

- 程序中的错误通常分为语法错误、运行错误和逻辑错误。
- 语法错误是由于程序中使用了不符合语法规则的信息而导致的。
- 例如：缺少标点符号、表达式中括号不匹配、关键字拼写错误等，这类错误比较容易修改，因为编译器或解释器会指出错误的位置和性质；

- 运行错误则不容易修改，因为其中的错误是不可预料的，或者可以预料但无法避免的。
- 例如：内存空间不够、数组下标越界、文件打开失败等；
- 逻辑错误主要表现在程序运行后，得到的结果与设想的结果不一致，通常出现逻辑错误的程序都能正常运行，系统不会给出提示信息，所有很难发现。

- 要发现和改正逻辑错误需要仔细阅读和分析程序及其算法。
- 程序出现错误的原因有很多，包括用户操作不当。
- 例如：输入数据的格式不对，本应输入整数的时候输了浮点数，或者输了非法字符；
- 程序本身有漏洞。例如：算法不严谨、考虑不全面；
- 程序运行时出现了异常。例如：创建文件失败、除数为0等。

- 良好的程序应该对用户的不当操作做出提示，能识别多种情况下的程序运行状况，并选择适当的应对策略。
- 对于程序运行时可能出现的情况，程序应有相应的处理措施。

- 在程序中，对各种可预见的异常情况进行处理称为异常处理（exception handling）。
- 例如：除法运算时，应对除数进行判断，对除数是0的情况进行异常处理。

- 处理程序异常的方法有很多，其中最简单和最直接的办法是在发现异常时，由Python系统进行默认的异常处理。
- 如果异常对象未被处理或者捕捉，程序就会用所谓的回溯（Traceback）终止执行。

- 例：下面的语句，执行“print(A)”语句时出错，系统抛出异常。
- >>> a=3/4
- >>> print(A)
- Traceback (most recent call last):
- File "<pyshell#9>", line 1, in <module>
- print(A)
- NameError: name 'A' is not defined

- 标准错误信息包括两个部分：错误类型（如NameError）和错误说明（如name 'A' is not defined），两者用冒号分隔。
- Python系统还追溯错误发生的位置，并显示有关信息。
- 以上异常发生时，程序终止运行，如果能对异常进行捕获并处理，将不至于使整个程序运行失败。

- 默认异常处理只是简单地终止程序运行，并给出错误信息。
- 显然，这种处理异常的方法过于简单。
- 还可以用if语句来判断可能出现的异常情况，以便程序做出适当处理。
- 例如：在求两个数的商时，在进行除法运算之前判断除数是否为0，从而捕获并防止异常。

- 例：整除程序的简单异常处理方法。

```
def main():
    a,b=eval(input())
    if b==0:
        print("Divide 0!")
    else:
        s=a/b
        print(s)
main()
```

- 这种处理方式使得程序对正常执行过程的描述和异常处理交织在一起，程序的可读性不好。
- 为此，Python提供了一套完整的异常处理方法。

11.2 捕获和处理异常

- 在Python中，异常也是对象，可对它进行操作。
- Python也提供try语句处理异常。
- try语句有两种格式：try-except语句和try-finally语句。

11.2.1 Python中的异常类

- Python中提供了一些异常类，同时也可以自己定义自己的异常。
- 所有异常都是基类Exception的成员。
- 所有异常都从基类Exception继承（直接或间接），而且都在exceptions模块中定义。

- Python自动将所有异常名称放在内置命名空间中，所以程序不必导入exceptions模块即可使用异常。
- 一旦引发而且没有捕捉SystemExit异常，程序执行就会终止。
- 如果交互式会话遇到一个未被捕捉的SystemExit异常，会话就会终止。

Python 中常见的异常类

异常类名	说　　明
Exception	所有异常类的基类
AttributeError	尝试访问未知的对象属性时引发
IOError	试图打开不存在的文件时引发
IndexError	使用序列中不存在的索引时引发
KeyError	使用字典中不存在的关键字时引发
NameError	找不到变量名字时引发
SyntaxError	语法错误时引发
TypeError	传递给函数的参数类型不正确时引发

异 常 类 名	说　　明
ValueError	函数应用于正确类型的对象，但是该对象使用不适合的值时引发
ZeroDivisionError	在做除法或模除操作中除数为 0 时引发
EOFError	发现一个不期望的文件或输入结束时引发
SystemExit	Python 解释器请求退出时引发
KeyboardInterrupt	用户中断执行（通常是按 Ctrl+C 快捷键）时引发
ImportError	导入模块或对象失败时引发
IndentationError	缩进错误时引发

- 尽管内置的异常类包含了大部分的情况，但是有时候还是需要创建自己的异常类，只要确保从**Exception**类继承（不管是直接的还是间接的）。

11.2.2 使用ty-except语句

- try-except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。
- 如果不希望在异常发生时结束运行的程序，只需在try中捕获它。

1. 最简单形式的异常处理

- try-except语句最简单的格式：
- try:
- 　语句块
- except:
- 　异常处理语句块

- 异常处理过程：
- 执行try后面的语句块，如果执行正常，语句块执行结束后转向执行try-except语句的下一条语句；
- 如果引发异常，则转向异常处理语句块，执行结束后转向ty-except语句的下一条语句。

- 例：整除程序的异常处理。

```
def main():
    a,b=eval(input())
    try:
        s=a/b
        print(s)
    except:
        print("Divide 0!")
main()
```

2. 分类异常处理

- 简单形式的try-except语句不加区分地对所有异常进行相同的处理.
- 如果需要对不同类型的异常进行不同处理，则可使用具有多个异常处理分支的try-except语句。

- 一般格式为：
- try:
- 　语句块
- except 异常类型1[as 错误描述]:
- 　异常处理语句块1
- ……
- except 异常类型n[as 错误描述]:
- 　异常处理语句块n
- except:
- 　默认异常处理语句块
- else:
- 　语句块

- 异常处理过程：
- 执行try后面的语句块，如果执行正常，在语句块执行结束后转向执行try-except语句的下一条语句；
- 如果引发异常，则系统依次检查各个except子句，试图找到与所发生异常相匹配的异常类型。
- 如果找到了，就执行相应的异常处理语句块。
- 如果找不到，则执行最后一个except子句下的默认异常处理语句块（最后一个不含错误类型的except子句是可选的）；

- 如果在执行try语句块时没有发生异常，Python系统将执行else语句后的语句（如果有else的话）。
- 异常处理结束后转向try-except语句的下一条语句。
- “as错误描述”子句为可选项。

- 例：整除程序的分类异常处理。
- 分析：
- 输入数据时，输入的数据个数不足或输入的是字符串而非数值均可导致TypeError。
- 输入的除数为0可导致ZeroDivisionError，等等。
- 程序可以捕获未预料到的异常类型。
- 程序没有异常时给出提示。

```
def main():
    a,b=eval(input())
    try:
        s=a/b
        print(s)
    except TypeError:
        print("数据类型错!")
    except ZeroDivisionError as e:
        print("除数为0错!",e)
    except:
        print("发生异常!")
    else:
        print("程序执行正确!")
main()
```

3. 异常处理的嵌套

- 异常处理可以嵌套。
- 如果外层try子句中的语句块引发异常，程序将直接跳转到与外层try对应的except子句，而内部的try子句将不会被执行。

- 例：
- try:
- 　　s="Python"
- 　　s=s+4
- 　　try:
- 　　　　print(s[0]+s[1])
- 　　　　print(s[0]-s[1])
- 　　except TypeError:
- 　　　　print("字符串不支持减法运算")
- except:
- 　　print("产生异常!")

- 例：

```
try:
    s="Python"
    #s=s+4
    try:
        print(s[0]+s[1])
        print(s[0]-s[1])
    except TypeError:
        print("字符串不支持减法运算")
except:
    print("产生异常!")
```

11.2.3 使用try-finally语句

- finally子句是指无论是否发生异常都将执行相应的语句块。
- 语句格式为：
- try:
- 　语句块
- finally:
- 　语句块

- 例：将输入的字符串写入到文件中，直至按Q键结束。如果按Ctrl+C快捷键，则终止程序运行，最后要保证打开的文件能正常关闭。

```
try:
    fh=open("test.txt","w")
    while True:
        s=input()
        if s.upper()=="Q":break
        fh.write(s+"\n")
except KeyboardInterrupt:
    print("按Ctrl+C时程序终止!")
finally:
    print("正常关闭文件！ ")
    fh.close()
```

11.3 断言处理

- 在编写程序时，在程序调试阶段往往需要判断程序执行过程中变量的值，根据变量的值来分析程序的执行情况。
- 可以使用**print()**函数打印输出结果，也可以通过断点跟踪调试查看变量，但使用断言更加灵活高效。

- 断言的主要作用是帮助调试程序，以保证程序运行的正确性。
- 使用断言（assert）语句可以声明断言。
- 其格式为：

- assert 逻辑表达式
- assert 逻辑表达式,字符串表达式
- assert语句有1个或2个参数。
- 第1个参数是一个逻辑值，如果该值为True，则什么都不做。如果该值为False，则断言不通过，抛出一个AssertionError异常。
- 第2个参数是错误的描述，即断言失败时输出的信息，也可以省略不写。

- 例：a整除程序的断言处理。
- a,b=eval(input())
- assert b!=0,'除数不能为0!'
- c=a/b
- print(a,"/",b,"=",c)

```
2,3
2 / 3 = 0.6666666666666666
```

```
2,0
Traceback (most recent call last):
  File "C:/Users/Administrator/AppData/Local/Programs
/Python/Python36-32/a.py", line 2, in <module>
    assert b!=0,'除数不能为0!'
AssertionError: 除数不能为0!
```

- **AssertionError**异常可以被捕获，并像使用在**try-except**语句中的任何其他异常处理，但如果不处理，它们将终止程序并产生回溯。

- 例：AssertionError异常处理。
- try:
- assert 1==3
- except AssertionError:
- print("Assertion error!")

11.4 主动引发异常与自定义异常类

- 前面的异常类都是由Python库中提供的，产生的异常也都是由Python解释器引发的。
- 在程序设计过程中，有时需要在编写的程序中主动引发异常，还可能需要定义表示特定程序错误的异常类。

11.4.1 主动引发异常

- 当程序出现错误时，Python会自动引发异常，也可以通过raise显式地引发异常。
- 一旦执行了raise语句，raise后面的语句将不能执行。

- 在Python中，要想自行引发异常，最简单的形式就是输入关键字raise，后跟要引发的异常的名称。
- 异常名称标识出具体的类，Python异常处理是这些类的对象。
- 执行raise语句时，Python会创建指定的异常类的一个对象。
- raise语句还可指定对异常对象进行初始化的参数。

- raise语句的格式为：
- raise 异常类型[(提示参数)]
- 其中，提示参数用来传递关于这个异常的信息，它是可选的。
- 例：
- >>> raise Exception("抛出一个异常")
- Traceback (most recent call last):
- File "<pyshell#11>", line 1, in <module>
- raise Exception("抛出一个异常")
- Exception: 抛出一个异常

11.4.2 自定义异常类

- Python允许自定义异常类，用于描述Python中没有涉及的异常情况，自定义异常类必须继承Exception类，自定义异常类名一般以Error或Exception为后缀，表示这是异常类。
- 自定义异常使用raise语句引发，而且只能通过人工方式引发。

- 例：创建异常类NumberError.py。
- class NumberError(Exception):
- 　　def __init__()(self,data):
- 　　　　self.data=data

- 例：处理学生成绩时，成绩不能为负数。利用前面创建的NumberError异常类，处理出现负数成绩的异常。

```
from NumberError import *   #导入已创建的异常类NumberError
def average(data):
    sum=0
    for x in data:
        if x<0:raise NumberError("成绩为负!")   #成绩为负时引出异常
        sum+=x
    return sum/len(data)
def main():
    score=eval(input("输入学生成绩:"))   #将学生成绩存入元组
    print(average(score))
main()
```

```
#分别输入1~5时，分析程序的运行结果。
class AError(Exception):
    def __init__(self,data):
        self.data=data
def main():
    x=eval(input())
    try:
        if x==1:
            raise AError("AAA")
        if x==2:
            raise AError("BBB")
        if x==3:
            raise TypeError("CCC")
        if x==4:
            raise KeyError("DDD")
        print("in try")
    except AError as a:
        print("in AError",a)
    except TypeError as b:
        print("in TypeError",b)
    except:
        print("in XXXError")
    else:
        print("in else")
    finally:
        print("in finally")
main()
```

自测题

- 一、选择题
- 1．下列关于Python异常处理的描述中，不正确的是（　　）。D
- A．异常处理可以通过try-except语句实现。
- B．任何需要检测的语句必须在try语句块中执行，并由except语句处理异常。
- C．raise语句引发异常后，它后面的语句不再执行。
- D．except语句处理异常最多有两个分支。

- 2． 以下关于异常处理try语句块的说法，不正确的是（　　）。C
- A．finally语句中的代码段始终要被执行
- B．一个try块后接一个或多个except块
- C．一个try块后接一个或多个finally块
- D．try必须与except或finally块一起使用

- 3．Python异常处理机制中没有（　　）语句。B
- A．try　　B．throw　　C．assert　　D．finally
- 4．如果以负数作为平方根函数math.sqrt()的参数，将产生（　　）。C
- A．死循环　　B．复数
- C．ValueError异常　　D．finally

- 5．下列程序的输出结果是（　　）。D
- try:
- 　　x=1/2
- except ZeroDivisionError:
- 　　print('AAA')
- A．0　　B．0.5
- C．AAA　　D．无输出

- 6．下列程序的输出结果是（　　）。A
- x=10
- raise Exception("AAA")
- x+=10
- print("x=",x)
- A．Exception: AAA　　B．10
- C．20　　D．x=20

- 二、填空题
- 1．Python提供了（　）机制来专门处理程序运行时错误，相应的语句是（　）。异常处理，try-except
- 2．在Python中，如果异常并未被处理或捕捉，程序就会用（　）错误信息终止程序的执行。Traceback（回溯）

- 3．Python提供了一些异常类，所有异常都是（　）类的成员。Exception
- 4．异常处理程序将可能发生异常的语句块放在（　）语句中，紧跟其后可放置若干个对应的（　）语句。如果引发异常，则系统依次检查各个（　）语句，试图找到与所发生异常相匹配的（　）。try，except，except，异常类型

- 5．下列程序的输出结果是（　）。BBB
- try:
- 　print(2/'0')
- except ZeroDivisionError:
- 　print('AAA')
- except Exception:
- 　print('BBB')

- 三、问答题
- 1．程序的逻辑错误能否算作异常？为什么？
- 2．什么叫异常？异常处理有何作用？在Python中如何处理异常？
- 3．什么是Python内置异常类的基类？列举五个常见的异常类。
- 4．语句try-except和try-finally有何不同？
- 5．assert语句和raise语句有何作用？