# 第十二章 异常处理

本节所讲内容：

12.1 认识异常

12.2 捕获异常

12.3 异常信息

12.4 触发异常

12.5 嵌套捕获异常

## 12.1 认识异常

作为Python初学者，在刚学习Python编程时，经常会看到一些报错信息，在前面我们没有提及，这章节我们会专门介绍。

Python有两种错误很容易辨认：语法错误和异常。

什么是异常？

由于逻辑或者语法导致的程序中断叫做异常。

异常即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。

一般情况下，在Python无法正常处理程序时就会发生一个异常。

异常是Python对象，表示一个错误。

当Python脚本发生异常时我们需要捕获处理它，否则程序会终止执行。

语法格式：

try:

可能触发异常的语句块

except [exceptionType]:

捕获可能触发的异常[可以指定处理的异常类型]

except [exceptionType as data]:

捕获异常并获取异常详细信息data

except:

没有指定异常类型，捕获任意异常

else：

没有触发异常时，执行的语句块

finally:

     print("finally")

注意：一般底层中的出现列表【】表明内部的数据可添加可不添加！

### 12.1.1 语法错误

#此时代码中没有‘：’

def test()  
 print(a)

运行结果如下：

File “E:/workspace/importTest/test02.py”, line 7

def test()

^

SyntaxError: invalid syntax

### 12.1.2 异常 print(p)

运行结果如下：

Traceback (most recent call last):

File "E:/workspace/importTest/test02.py", line 6, in <module>

print(p)

NameError: name 'p' is not defined

### 12.1.3 异常简介

看如下示例:

print('----test01----')  
open('123.txt','r')  
print('----test02----')

运行结果如下:

说明:

打开一个不存在的文件123.txt，当找不到123.txt 文件时，就会抛出给我们一个FileNotFoundError 类型的错误，[Errno 2] No such file or directory: '123.txt' （没有123.txt这样的文件或目录）

Python2中出现的是IOError：

IOError: [Errno 2] No such file or directory: '123.txt'

>>> 4+'4'

Traceback (most recent call last):

File "<input>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'int' and 'str'

如果不想在程序出现异常就结束程序，就用捕获异常。

12.1.4 异常总结（了解一下）

BaseException # 所有异常的父类

- SystemExit # 由sys.exit()抛出的异常

- KeyBoardInterrupt # 通常由ctrl+c或者Delete抛出的异常

- GeneratorExit # 当生成器被关闭时抛出的异常

- Exception

- StopIteration # 迭代结束异常

- StopAsyncIteration # 由异步迭代的`\_\_anext\_\_()`抛出的异常

- ArithmeticError # 各种算数错误引起的异常

- FloatingPointError # 浮点数操作错误

- OverflowError # 结果超出范围

- ZeroDivisionError # ０为除数异常

- AssertionError # assert错误异常

- AttributeError # 属性引用异常

- BufferError # 缓存错误

- EOFError # 读不到数据

- ImportError # import错误

- ModuleNotFoundError # 找不多模块

- LookupError # 由索引和key值引起的异常

- IndexError # 索引错误

- KeyError # 字典key值错误

- MemortError # 内存溢出异常

- NameError # 本地和全局找不到变量名

- UnboundLocalError # 局部变量没有赋值

- OSError # system错误

- BlockingIOError # 调用阻塞异常错误

- ChildProcessError # 子进程

- ConnectionError # 连接

- BrokenPipeError # 管道读写异常

- ConnectionAbortedError # 连接失败

- ConnectionRefusedError # 连接拒绝

- ConnectionResetError # 连接重置

- FileExistsError # 创建文件和文件夹错误

- FileNotFoundError # 文件未找到

- InterruptedError # 中断错误

- IsADirectoryError # 文件操作用在文件夹上

- NotADirectoryError # 不是文件夹

- PermissionError # 权限

- ProcessLookupError # 进程不存在

- TimeoutError # 超时

- ReferenceError # 引用异常

- RuntimeError #

- NotImplementedError # 运行抽象方法

- RecursionError # 超出最大递归深度

- SyntaxError # 语法错误

- IndentationError # 缩进错误

- TabError # tab错误

- SystemError # 解释器中断

- TypeError # 类型错误

- ValueError # 赋值错误

- UnicodeError #

- UnicodeEncodeError # unicode编码错误

- UnicodeDecodeError # unicode解码错误

- UnicodeTranslateError # unicode转换错误

- Warning

- DeprecationWarning # 操作不赞成警告

- PendingDeprecationWarning # 表明此操作将来会被弃用

- UserWarning # 用于用户生成警告

- SyntaxWarning # 语法可疑警告

- RuntimeWarning # 运行警告

- FutureWarning # 将会改变警告

- ImportWarning # 导入警告

- UnicodeWarning # unicode相关警告

- BytesWarning # 字节相关警告

- ResourceWarning # 资源使用情况警告

## 12.2 捕获异常



### 12.2.1 捕获异常 try...except...

语法格式：  
try:

可能触发异常的语句块

except [exceptionType]:

捕获可能触发的异常[可以指定处理的异常类型]

except [exceptionType]:

捕获异常并获取附加数据

示例:

print('----test01----')  
try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----try ----')  
except FileNotFoundError:  
 print('这是一个FileNotFoundError的错误%s'% FileNotFoundError)  
except NameError:  
 print('这是一个NameError的错误：%s'%NameError)  
print('----test02----')

运行结果如下：

----test01----

这是一个NameError的错误：<class 'NameError'>

----test02----

说明:

此程序看不到任何错误，因为用except 捕获到了NameError异常，并添加了处理的方法。

总结:

#将可能出错的代码放进try中，把处理异常的代码放在except中。

try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----try----')  
 *# except负责抓捕出现的异常，并处理。*except FileNotFoundError:  
 print('这是一个FileNotFoundError的错误%s'%FileNotFoundError)  
except NameError:  
 print('这是一个NameError的错误：%s'%NameError)

运行结果如下：

这是一个NameError的错误：<class 'NameError'>

把可能出现问题的代码，放在try中；

把处理异常的代码，放在except中；

注意：只会抓捕到第一个出错的语句。

### 10.2.2 except捕获多个异常

抓捕多个异常，一般有三种方法，在这里我用三种案例给大家阐述具体的操作！在此之前大家先看下面这则示例。

示例：

try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----try----')  
except FileNotFoundError:  
 print('这是一个FileNotFoundError的错误%s'%FileNotFoundError)

运行结果如下:

Traceback (most recent call last):

----test01----

File "E:/workspace/impTest/test02.py", line 64, in <module>

print(name)

NameError: name 'name' is not defined

想一想:

上例程序，已经使用except来捕获异常了，为什么还会看到错误的信息提示？

原因:

except捕获的错误类型是FileNotFoundError，而此时程序产生的异常为 NameError ，所以except没有生效。

修改后的代码为:

print('----test01----')  
try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----test02----')  
except NameError:  
 print('这是一个NameError的错误：%s'%NameError)  
print('----test02----')

运行结果如下:

----test01----

这是 NameError的：<class 'NameError'>

----test02----

看到以上内容可以总结一个规律：

try 中的程序由上到下运行，然后找到第一个错误，直接抛出处理，那try下面第二个错误异常根本没有抓捕到。（等等……，好像哪里不对，如果我都明确知道你是什么异常，还用你这个方法干嘛—\_—）

想一想：有没有很好的方法，可以处理多个异常？

方法一：指定一个通用异常，在这里我们使用所有的异常的父类Exception达到我们想要的结果

语法：

try:

语句块

except Exception:

语句块

案例：

print('----start----')  
try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----test02----')  
except Exception:  
 print('这是一个Exception的错误：%s'%Exception)  
print('----end----')

运行结果如下：

----test01----

这是一个Exception的错误：<class 'Exception'>

----test02----

方法二：在except使用元组捕获多个异常的方式

print('----start----')  
try:  
 *#没有name变量，报错* print(name)  
 *#没有123.txt文件，报错* open('123.txt','r')  
 print('----try----')  
except (NameError,FileNotFoundError):  
 print('这是一个NameError的错误：%s'%NameError)  
 print('这是一个FileNotFoundError的错误%s' % FileNotFoundError)  
print('----end----')

运行结果如下：

----test01----

这是一个NameError的错误：<class 'NameError'>

这是一个FileNotFoundError的错误<class 'FileNotFoundError'>

----test02----

注意：这种方法不严谨

案例：

print('---test01---')  
try:  
 open('123.txt', 'r')  
 print(name)  
 print('---try----')  
*#抓捕多个异常就用于元组，但是这个方法不严谨*except (FileNotFoundError,NameError,TypeError,IOError):  
 print('这是一个FileNotFoundError的错误%s' % FileNotFoundError)  
 print('这是一个NameError的错误%s' % NameError)  
 print('这是一个TypeError的错误%s' % TypeError)  
 print('这是一个IOError的错误%s' % IOError)  
*# except NameError:*print('---test02---')

运行结果如下：

---test01---

这是一个FileNotFoundError的错误<class 'FileNotFoundError'>

这是一个NameError的错误<class 'NameError'>

这是一个TypeError的错误<class 'TypeError'>

这是一个IOError的错误<class 'OSError'>

---test02---

方法三：except子句后不带任何异常名称，捕获所有异常

try:  
 print(a)  
 open('123.txt','r')  
*#有异常但是不知道具体什么异常*except:  
 print('有异常，但是哥就不想打印……')

运行结果如下：

有异常，但是哥就不想打印……

## 12.3 异常信息

### 12.3.2 获取异常的信息描述

1. as 打印详细信息

try:  
 print(a)  
*#NameError 是异常，e 是为了存储异常的详细信息*except NameError as e:  
 print('NameError 的详细信息为%s'%e)

运行结果如下：

NameError 的详细信息为name 'a' is not defined

2. as处理多个异常详细信息

想一想：能不能打印出多个异常的详细信息？

try:  
 print(a)  
 open('123.txt','r')  
*#NameError， FileNotFoundError是异常，e 是为了存储异常的详细信息*except (FileNotFoundError ,NameError) as e:  
 print('详细信息为%s'%(e))

运行结果如下：

详细信息为name 'a' is not defined

说明：上面例子中的e相当于异常中的一个对象。

### 12.3.3 else

我们应该对else并不陌生，在if中，它的作用是当条件不满足时执行的实行语句；同样在try...except...中也是如此，即如果没有捕获到异常，那么就执行else中的事情。

语法格式：

try:

需要执行的操作

except 异常类 as e:

如果发生异常，执行以下代码

except 异常类 as f:

如果发生异常，执行以下代码

else:

如果没有发生异常，执行的命令

try:  
 num = 10  
 print(num)  
except Exception as e:  
 print('有异常，…… %s'% e)  
*#else对应except来说的*else:  
 print('没有发生异常')

运行结果如下:

10

没有发生异常

如果没有except  
try:  
 num = 10  
 print(num)  
else:  
 print('没有发生异常')

运行结果如下：

File "E:/workspace/importTest/test02.py", line 32

else:

^

SyntaxError: invalid syntax

注意，else一定要有except否则会出现语法错误

### 12.3.4 try...finally...

try...finally...语句用来表达这样的情况：

在程序中，如果一个段代码必须要执行，即无论异常是否产生都要执行，那么此时就需要使用finally。 比如文件关闭，释放锁，把数据库连接返还给连接池等。

格式：

try:

需要执行的操作

except 异常类 as e:

如果发生异常，执行以下代码

except 异常类 as f:

如果发生异常，执行以下代码

else:

如果没有发生异常，执行的命令

Finally:

最终执行的代码

案例：

try:  
 *# num = 10* print(num)  
except Exception as e:  
 print('打印出错信息%s'%e)  
*#finally 无论前面是否错误，最终都会执行*finally:  
 print('\_\_finally\_\_')

运行结果对比：

第一次：

10

\_\_finally\_\_  
第二次：

打印出错信息name 'num' is not defined

\_\_finally\_\_

说明：

不管try语句有没有执行成功，都会执行finally当中的代码逻辑。

## 12.4 触发异常

你可以用raise语句来引发一个异常，异常/错误对象必须有一个名字，且它们应是Error或Exception类的子类，我们可以使用raise语句自己触发异常。

raise语法格式如下：

raise [Exception [, args [, traceback]]]

语句中 Exception 是异常的类型（例如，NameError）参数标准异常中任一种，args 是自已提供的异常参数。

最后一个参数是可选的（在实践中很少使用）跟踪异常对象（了解一下）。

抛出异常后，raise 后面的代码不再执行，为了能够捕获异常，"except"语句必须要用相同的异常来抛出类对象或者字符串。

### 12.4.1 使用raise抛出系统的异常

class Test(object):  
 def start(self):  
 try:  
 content = int(input('请输入数字:'))  
 if content < 5:  
 *#raise 抛出异常后，后面的代码不在执行* raise Exception('你输入的数字小于3')  
 *#print（‘异常已经抛出了’）  
 #except后面的异常和raise抛出的一样* except Exception as info:  
 print(info)  
 else:  
 print('恭喜你，没有异常')  
  
t = Test()  
t.start()

运行结果如下：

请输入数字:2

你输入的数字小于3

注意：抛出异常后，raise 后面的代码不在执行，为了能够捕获异常，"except"语句必须用相同的异常来抛出类对象或者字符串。

### 12.4.2 使用raise抛出自定义异常

*#自定义一个异常MyException，继承Exception*class MyException(Exception):  
 *#重写父类的\_\_init\_\_方法* def \_\_init\_\_(self,num,atleast):  
 *#调用父类的\_\_init\_\_* super().\_\_init\_\_()  
 self.num = num  
 self.atleast = atleast  
class Test(object):  
 def start(self):  
 try:  
 content = int(input('请输入数字:'))  
 if content < 5:  
 *#抛出自定义异常，相当于创建一个实例对象* raise MyException(content,3)  
 *#相当于创建这个对象赋值给s* except MyException as s:  
 print('当前大小:%s,最少的数字:%s'%(s.num,s.atleast))  
 else:  
 print('输入的结果符合要求')  
t = Test()  
t.start()

运行结果如下:

请输入数字:5

输入的结果符合要求

注意：

以上程序中，关于代码super（）.\_\_init\_\_()的说明

这一行代码，可以调用也可以不调用，建议调用，因为\_\_init\_\_方法往往是用来对创建完的对象进行初始化工作，如果在子类中重写了父类的\_\_init\_\_方法，即意味着父类中的很多初始化工作没有做，这样就不保证程序的稳定了，所以在以后的开发中，如果重写了父类的\_\_init\_\_方法，最好是先调用父类的这个方法，然后再添加自己的功能。

### 12.4.3 工作自定义异常

在工作中我们时常要定义自己异常，减少程序的耦合性，简单来说，为了处理各模块之间的依赖性问题，提高重用。

工作中，当我们遇到这个自定义的异常时候，都会统一封装到一个py文件里.例如 生成imp\_except.py文件，imp\_except 里面封装一个代码：

Imp\_except.py中：

*#自定义一个异常MyException，继承Exception*class MyException(Exception):  
 *#重写父类的\_\_init\_\_方法* def \_\_init\_\_(self,num,atleast):  
 *#调用父类的\_\_init\_\_* super().\_\_init\_\_()  
 self.num = num  
 self.atleast = atleast

然后在其他py中导入该模块  
from imp\_except import MyException

class Test(object):  
 def start(self):  
 try:  
 content = int(input('请输入数字:'))  
 if content < 5:  
 *#抛出自定义异常，相当于创建一个实例对象* raise MyException(content,3)  
 *#相当于创建这个对象赋值给s* except MyException as s:  
 print('当前大小:%s,最少的数字:%s'%(s.num,s.atleast))  
 else:  
 print('输入的结果符合要求')  
t = Test()  
t.start() 注意： 工作中的自定义异常，常常统一处理。

## 12.5 嵌套捕获异常

### 12.5.1 try的嵌套使用

示例如下：

import time  
try:  
 f = open('123.txt',encoding='utf-8')  
 try:  
 while True:  
 *#自动打印三行* for i in range(3):  
 print(f.readline())  
 *#睡两秒* time.sleep(2)  
 *#再读一行* content = f.readline()  
 *#打印* print(content)  
 *#如果我们读取的内容0 则break* if len(content) == 0:  
 break  
 *#无论如何都要关闭文件* finally:  
 f.close()  
except:  
 print('没有这个文件')

### 12.5.2 函数嵌套调用

def test01():  
 print('----test01----')  
 print(num)  
def test02():  
 print('----test02----')  
def test03():  
 try:  
 print('----test03----')  
 test01()  
 except Exception as result:  
 print('捕获异常:%s'%result)  
test03()

运行结果如下：

----test03----

----test01----

捕获异常:name 'num' is not defined

总结：

12.1 认识异常

12.2 异常处理

12.3 捕获异常

12.4 触发异常

12.5 嵌套捕获异常