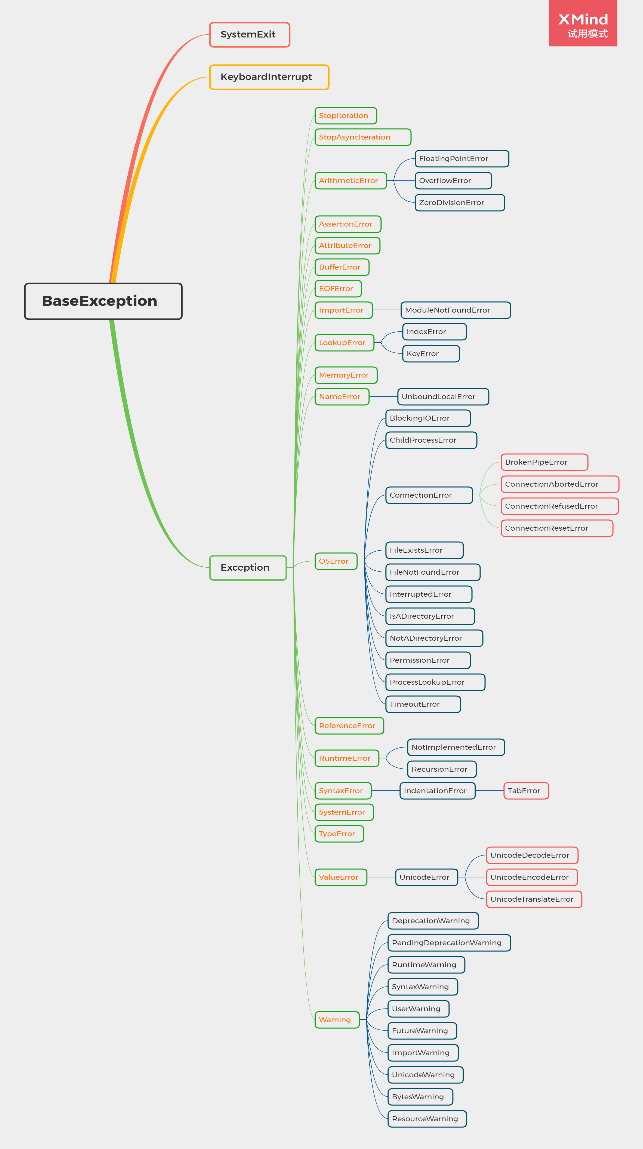
《Python实用系列---异常处理》

大家好，前几篇推文写的是关于xlwings的相关知识，介绍了其相关API的使用方法，大家在使用xlwing对Excel进行操作的时候，就可以使用推文中介绍的方法完成自己的任务。关于其与VBA的结合的使用，等后期遇到比较好项目的时候我再跟大家介绍。

从现在开始，本公众号将开启新的写作《Python实用系列》，主要是针对自己在项目中，对那些平时使用频率较高的，觉得重要的知识点进行总结和归纳，以更好的帮助大家在写代码的时候起到举一反三的作用，闲话不多说，这个系列的开篇我们就从-异常处理讲起！

百度百科中对异常处理的介绍为：**异常处理**，是[编程语言](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E8%AF%AD%E8%A8%80/9845131)或计算机硬件里的一种机制，用于处理软件或信息系统中出现的异常状况（即超出程序正常执行流程的某些特殊条件）。从这个角度来看，异常处理是保证程序正常运行关键，因此异常处理是非常重要的！

在Python中，处理异常的基类是：BaseException，各个异常具体如下所示：



**一．异常分类**

基类BaseException中有一个异常类Warning，这个Warning我们应该遇见的次数很多了，比如我们在运行代码的时候，比如某一个要模块版本过低的时候会出现这样的告警。

记下来我们来介绍上述所列的几个常见异常：

|  |  |
| --- | --- |
| AttributeError | 试图访问类中成员不存在的时候所抛出的异常 |
| IndexError | 试图访问超出对象索引的范围的时候所抛出的异常 |
| KeyError | 试图在字典中查找一个不存在的键的时候抛出的异常 |
| NameError | 试图访问一个不存在的变量时抛出的异常 |
| OSError | 操作系统产生的异常 |
| SyntaxError | 语法错误的时候抛出的异常（比较常见） |
| TypeError | 使用类型错误抛出的异常 |

**1. AttributeError**

class Car():  
 def \_\_init\_\_(self):self.car\_lenth = 3.5  
 self.car\_height = 1.7  
 self.car\_name = **'Benz'**new\_car = Car  
print(new\_car.car\_width)

上述代码最后一行代码试图访问不存在的属性car\_width，这将抛出一下的异常：

AttributeError: type object 'Car' has no attribute 'car\_width'

**2. IndexError**

def len\_list(number):  
 print(number[len(number)] + 1)  
len\_list([1,2,3,4,5])

上述代码试图访问超出列表的索引进行取值，这将抛出异常：

IndexError: list index out of range

**3. KeyError**

def visit\_dic(dic\_num):  
 print(dic\_num[**'4'**])  
dic\_123 = {**'1'**:**'我'**, **'2'**:**'爱'**, **'3'**:**'python学堂'**}  
visit\_dic(dic\_123[**'4'**])

上述代码试图访问字典dic\_123中不存在的键值，这将抛出KeyError: '4'异常

**4. NameError**

def name\_err():  
 print(name)  
name\_err()

上述代码中，函数name\_err()中试图打印不存在的name对象，将抛出异常：

NameError: name 'name' is not defined

**5. OSError**

def open\_file(filename):  
 file=open(filename)  
file = **r'C:\Users\TEN\Desktop\'1.txt'**open\_file(file)

上述代码试图打开桌面一个不存在的文件1.txt，这将抛出OSError异常：

FileNotFoundError: [Errno 2] No such file or directory: "C:\\Users\\TEN\\Desktop\\'1.txt"

值的一提的是在python3.3版本以后，IOError(有关于输入输出的异常)将并到OSError中，看上述Xmind图可知FileNotFoundError属于OSError类。

**6. SyntaxError**

def syn\_error():  
 print(**'python知识学堂'**)  
 print(**'今天天气好)**syn\_error()

上述代码第二行的print打印测时候，少了一个“’”，将抛出异常：

SyntaxError: EOL while scanning string literal

**7. TypeError**

def add\_number(parameter1, parameter2):  
 sum = parameter1 + parameter2  
 return sum  
total = add\_number(1, 2, 3)  
print(total)

上述代码的函数add\_number接受2个参数，第五行代码传入了3个参数，这将抛出类型错误的异常：

TypeError: add\_number() takes 2 positional arguments but 3 were given

综上，我们看了7种类型的异常。那么很显然想到的就是，出现的这些异常我们该怎么处理呢？接下来我们就将介绍处理异常的一些方法：

**二．Try-except**

异常是使用try-except来进行处理的，使用try-except的好处就是可以让python执行指定的操作，可以人为定义一些面向用户友好的一些错误信息，帮助用户定位错误。**在实际的项目中，我们把可能出现异常的代码块放在try中，以便排除异常并保证其余代码正常运行而不退出。**

比如我们使用try-except处理一个ZeroDivisionError异常：

try:  
 print(100/0)  
except ZeroDivisionError:  
 print(**'分母不能为0'**)

这里出现错误的代码被放在了try中，如果try中的代码没有问题，python将跳过except中的代码；如果try中的代码出现问题，则会执行except中的代码。

这里有一个技巧，就是else模块的使用，比如：

try:  
 print(100/1)  
except ZeroDivisionError:  
 print(**'分母不能为0'**)  
else:  
 print(**'python'**)

上述代码输出：100.0，python。使用else的好处就是可以让代码继续执行，即使遇到一些可能的错误，作为一种选择性的继续执行时非常nice的。

在抛出异常的时候，我们可以使用except error\_type as errorname: 的形式,那么使用print(errorname)就可以打印异常对象了，十分方便。

try:  
 print(100/0)  
except ZeroDivisionError as zeror:  
 print(**'分母不能为0'**)  
 print(zeror)  
else:  
 print(**'python'**)

**三．多except**

在实际的项目中，我们在一段代码中需要处理的异常不只一个，比如要同时处理ArithmeticError和ValueError。ArithmeticError是 ZeroDivisionError的父类，这个时候我们就可以使用多Try-except方法了，该方法就可以进行多重异常的捕获了哦。

try:  
 file = open(**r'C:\Users\TEN\Desktop\1.txt'**) *# 桌面不存在文件1.txt* print(100/0)  
 print(100/2)  
except ZeroDivisionError as zeror:  
 print(**'分母不能为0'**)  
except FileNotFoundError as fn:  
 print(fn)  
 print(**'python'**)  
else:  
 print(**'运行结束'**)

程序正常运行且上述代码输出：

[Errno 2] No such file or directory: "C:\\Users\\TEN\\Desktop\\'1.txt"

python

上述代码使用多个except来进行不同的异常处理，大家可以好好研究一下其中的打印为什么是这样。

**三．嵌套Try-except**

在实际的项目中我们可能还会遇到这样的情况：在一个try-except中处理了一个异常，在这个异常处理之中我们还要进行异常的处理，此时就需要使用嵌套Try-except了，正好python中也是支持这个嵌套的。我们来看一个实际的例子，来感受一下：

def trying\_test():  
 try:  
 try:  
 print(100/0)  
 except:  
 print(**'分母不能为0'**)  
 try:  
 file = open(**r'C:\Users\TEN\Desktop\1.txt'**)  
 except FileNotFoundError as fn:  
 print(fn)  
 except:  
 print(**'python'**)  
 finally:  
 print(**'最后的打印'**)  
trying\_test ()

上述代码运行输出：

分母不能为0

[Errno 2] No such file or directory: "C:\\Users\\TEN\\Desktop\\'1.txt"

最后的打印

上述两个嵌套的try-except被嵌套在第一个代码中，精确的说就是两个try-except被放在try代码块中，当然了实际的开发中我们也可与将要被嵌套的try-except放在except的代码块中。Python的traceback提供了一个print\_exc()函数可以帮助我们打印一些异常的信息，有兴趣的小伙伴可以查阅一下相关的文档。

有的小伙伴注意到上述中有一个finally，这个finally的一个好处就是在异常抛出的之后，这个finally总会执行的，比如在诸如打开文件、写数据到文件的操作失败的时候就可以进行关闭文件的操作。

**四．With-as**

我们在看其他人代码的时候，会看到这样的代码：with open(filename) as f 的样式。使用with语句的好处，就是到达语句末尾时，会自动关闭文件，这就不需要使用try-except的方式捕捉异常进行关闭文件的操作了。

如果直接使用open()方法打开文件，如果出现异常，如读取过程中文件不存在或异常，则直接出现错误，close方法无法执行，文件无法关闭的，我们来看看使用with open…的方式：

def open\_file(filename):  
 try:  
 with open(filename) as f:  
 file = open(filename)  
 print(**'文件名:'**,file.name)  
 except OSError as E:  
 print(**'open file error'**)  
filename = **r'C:\Users\TEN\Desktop\1.txt'**open\_file(filename)

**五．异常类的定义**

在实际的工程项目中，我们有时候需要定义一个区别于系统的异常类，还好python提供了一个可以自己定义异常类的方法，我们来看下面的代码：

*# 自定义异常类,继承于Exception基类*class Division\_Error(Exception):  
 *# 当输出有误时，抛出此异常  
 # 初始化* def \_\_init\_\_(self, value, name):  
 self.value = value  
 self.name = name  
 *# 返回异常类对象的相关信息* def \_\_str\_\_(self):  
 if self.value % 2 != 0 and len(self.name) < 5:  
 return (**"{}不是偶数，{}长度小于4"**.format(self.value,self.name) )  
try:  
 *# 抛出异常* print(**'尝试显示抛出异常'**)  
 diverr = Division\_Error(1,**'name'**)  
 raise diverr  
 *#或raise Division\_Error(1,'name')'*except Division\_Error as diverr:  
 print(**'diverr: {}'**.format(diverr))

上述代码中自定了一个继承于基类的Division\_Error类，用于抛出一些异常，在try的代码块中，我们使用raise用于显示的抛出异常。

**五．总结**

上述就是Python中异常处理的相关知识了，介绍了python中几个异常的类型以及使用小例子具体介绍了他们的用法，接着我们介绍了异常的处理方法，这些方法在实际的项目中都非常的实用，小伙伴们可以参考其中的一些介绍进行实际的操作。

好了，这次的异常处理就介绍到这，各位小伙伴觉得写得不错的话，欢迎分享和点格在看，感谢大家的支持。