【白话python连载（8）】python的异常处理--让bug无处遁形

* python常见异常的介绍

python程序在运行过程中可能会因为语法错误、逻辑错误或外部异常（网络中断、空间不足、键盘操作）而导致程序终止退出。而且这些问题的出现往往是不可避免的。每一个较为成型的健壮的程序都是经过不断测试发现错误、调试错误、处理错误而逐渐完善。因此python中也内置了较为完整的异常处理机制，来帮助程序开发人员更好的处理不同的程序问题。常见的异常如下：

|  |
| --- |
| 常见异常分类和实例 |
| NameError 使用的变量不存在 |
| AttributionError对象属性不存在，或者属性名不对所造成 |
| FileNotFoundError 读取文件失败，是因为文件在指定位置不存在 |
| ModuleNotFoundError 模块导入失败 模块名书写不对 |
| IndentationError代码对齐方式不对，代码书写缩进不正确 |
| IndexError列表下标超出索引 |
| keyErroy字典不存在某个键 |
| KeyboardInterrupt 用户中断操作，用户使用键盘输入了ctrl+c |
| SyntaxError 代码的逻辑语法不对，python3中的输出函数必须用（）包裹输出内容 |
| TypeError 传入的对象类型不符合要求 |
| ValueError 传入的对象的值的数据类型也要求不符合 |

* 自定义抛出异常

python内置的这个异常处理机制，可以帮助我们很快了解出错的信息，而且我们也可以借助print（输出语句）或assert（断言机制）【assert 表达式，表达式不为真时输出异常】找出出错的位置。促使我们有针对性的修正一些显而易见的错误。

除此以外，在编写特定功能的模块时，该模块有些地方也会常常出现问题，但是在现有的机制中没有合适的错误提示，当别人导入使用该模块可能不太能轻易找出这个异常，因此我们自己在编写程序的过程中也可以自定义一些异常情况，并将其抛出，让使用可以很方便的发现问题，解决问题。自定义的抛出异常如下：

如果是python中不存在的异常，需要先提前声明对应异常的类，不然会报名称异常的错。在程序中间容易出问题的地方使用raise语句抛出相关错误。自定义的错误抛出可以使得自己输出的相关功能模块更易复用。自定义抛出的异常必须是一个异常的实例或者是异常的类（也就是 Exception 的子类）。

|  |
| --- |
| *#!/usr/bin/python # -\*- coding: utf-8 -\*-* **class MyError**(TypeError)**: #定义异常的类  pass #代码占位不执行任何操作 def my\_abs**(*x*)**: #实现求绝对值的函数（要求输入的必须是整型或者浮点型）  if not** isinstance(*x*, (int, float))**:  raise MyError**(**'bad input type'**) #抛出异常  **if** *x* **>=** 0**:  return** *x* **else:  return -***x* **if** \_\_name\_\_ **== '\_\_main\_\_':** print(**my\_abs**(0))  print(**my\_abs**(**'a'**)) |
| *"D:\anaconda python\python3.6.exe" D:/vernacular/coding\_ziyi/Error.py*  *Traceback (most recent call last):*  *0*  *File "D:/vernacular/coding\_ziyi/Error.py", line 16, in <module>*  *print(my\_abs('a'))*  *File "D:/vernacular/coding\_ziyi/Error.py", line 9, in my\_abs*  *raise MyError('bad input type')*  *\_\_main\_\_.MyError: bad input type*  *Process finished with exit code 1* |

* 和python的bug say googbye

python程序在运行过程中，是按照主程序编写顺序依次执行每条语句，如果运行过程中在某条语句处检测到错误，则会立即终止程序运行，接下来的程序即使无错也无法再正常运行。为了跳过某些非人为因素导致的错误或者异常，或者比如在爬取网络数据时，不同网页的变量定义的不同可能会出现我们在获取数据时所对应的对象类型报错的情况。会希望跳过一些特殊的网站，而去获取其他类型符合要求的网站。这时需要借助python的处理异常的功能来实现。跳过可以直接忽略的那些报错内容。

|  |  |
| --- | --- |
| 基本框架 | 具体实例 |
|  |  |
|  | |
| 注意点：   1. 在异常处理机制中，只会输出一个except捕捉的异常内容。一旦捕获一个异常内容异常处理机制中后续的代码将不会再执行。直接跳到finally语句或接下来的程序块。 2. finally程序块无论之前是否捕捉到异常都会被执行过程，如果之前的except未捕捉到异常，经过finally处理后会中断程序并且提示相应的异常内容。若之前程序正常执行或被except捕捉到异常，则之后程序仍正常执行。 3. else程序块，只有在之前程序没有任何异常被捕获的的情况下才会执行。 4. try-except机制可以跨越多层次调用处理。只要在合适的层次处理异常捕获即可。如在主函数中捕获其他函数模块的异常信息，而不需要再在其他函数本身捕获异常。 5. 异常输出结果也是跨层次进行提示 ，从最新发现错位的位置寻根溯源，最终帮开发者找出最核心的出错原因。 | |

总结：

1、本次主要介绍了python程序中常见的一些异常，以便更好的了解报错的内容。

2、而对于抛出异常方式可以直接通过print（）或assert（）方法定位异常或者是通过自定义方式进行调试。

3、可以灵活使用try-except的检测和处理异常。根据捕获的异常信息确定异常原因，了解异常原因，而不影响后续程序的正常执行（非人为因素导致的）。

