Universe Scientific Publishing
Computer System Networking and Telecommunications
计算机系统网络和电信，2019,1(2)
ISSN:2661-3719 (Print); 2661-3727 (Online)

Analysis of Python-based Exception Handling Mechanisms

Honghui WANG Ke WANG Chenxi LI
School of Computer and Software, Jincheng College, Sichuan University, Chengdu, 611731

Abstract

With the rapid development of the society towards the information society and the ever-changing big data technology, Python language is popular for its concise and clear syntax, good readability and functional expansibility. It has more than 20 years of development history, has been mature and stable. The trend for scientific computation in Python is gaining momentum. Programs are written by people, so errors are inevitable. Fortunately, there are techniques that can be used to avoid errors, and there are techniques that can identify errors in a program and fix them. In this paper, some common Python exceptions and the characteristics of exception handling mechanism are briefly analyzed, and user information management system combined with file operation is illustrated as an example.

Key Words

Python; Exception handling; File operations

DOI:10.18686/jsjxt.v1i2.690

基于 Python 的异常处理机制的分析

王洪汇 王 科 李晨曦
四川大学锦城学院计算机与软件学院，四川成都，611731

摘 要

随着社会正快速向信息社会的发展，以及日新月异的大数据技术，Python 语言因为其简洁清晰的语法特点、良好的可读性还有功能的扩展性受到欢迎。它已经有二十多年的发展历史，已经成熟和稳定。Python 进行科学计算的趋势越来越猛。程序是由人编写的所以在所难免会产生错误，幸运的是我们可以用一些技术避免错误，而且还有一些技术可以识别出程序中的错误并且修复它。本文就 Python 的一些常见的异常以及异常处理机制的特征等进行了简要概括分析，并将用户信息管理系统结合文件操作为实例进行阐述。

关键词

Python；异常处理；文件操作

1.理解异常

1.1 程序中的错误

1.1.1 编译时的错误

Python 是一门先编译后解释的语言，在“解释”之前执行的第一个工作和 Java 一样是编译。如果程序存在语法错误，Python 编译器就会打印出文件名、行号、出错行等相关的错误信息，并使用“^”标记在该行程序中检测出错误的位置。这样编译时出现的语法错误是易于修复的。

1.1.2 运行时的错误

如果编译通过也就是说没有语法错误就会运行程序，运行时发生了未处理的异常，Python 就将终止执行程序。当 Python 检测到一个错误时，解释器就会指出当前流已经没有办法再继续执行下去，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意外采取的行为。异常即一个时间，该事件会在程序执行过程中发生，并且会影响程序的正常执行。

1.2 异常

Python 中的异常是在运行过程中，特定环境下，特定条件下引发的一个错误。比如说打开了一个不存在的文件，或者一些序列的类型超出了它的边界，或是不兼容的类型之间进行运算等等这些都叫做异常。这些异常并不是语法错误导致的。因此，在运行 Python 程序的过程当中，将首先检查语法错误。而程序的逻辑错误，是属于设计错误，并不属于程序本身的错误[1]。

1.3 异常的用途

Python 中，异常有多种用途，有几种比较常见的用途：错误处理、特殊情况处理、终止行为等。如果编译通过也就是说没有语法错误就会运行程序，运行时发生了未处理的异常，Python 就将终止执。可以在程序代码中捕捉和响应错误，或者忽略已经发生的错误，但是如果忽略错误的话默认的异常处理行为就会启动，程序会停止并且打印出错误的信息，可以通过异常捕捉将程序从异常中回复[2]；有的时候如果发生了比较罕见的异常，很难调整代码来处理。通常会在异常处理机制种取处理这些棘手的情况；终止行为无论程序中是否发生异常都可以确定一定会进行的运算。

1.3.1 错误处理

错误处理是异常处理的典型应用，在程序中捕捉可能发生的错误来进行处理，比如说打印错误，或是写异常日志等等。可以使用 try 进行捕获异常，并在发生错误时执行异常处理代码。以下为定义的保存用户基本修改信息文件的方法：

```
#保存创建的用户资料，若存在异常则打印异常信息
        def save_info(self):
            try:
             #打开 userdata.txt
             #x 方式针对不存在源文件时创建新文件
                userfile_txt = open("userdata.txt", "x")
            except Exception as e:
                userfile_txt = open("userdata.txt","w+")
                self.log.write(str(e)+"\n")
            for info in self.user_info:
             #把 name 和 password 存入 userdata.txt 中
                userfile_txt.write(str(info.name+" "+info.password)+"\n")
                userfile_txt.close()
```

该代码的含义为打开一个名为 userdata 的 txt 文件，“x”方式表示创建一个新的文件来写入内容，但是如果之前有这个文件存在，那么就会报错。若该过程出现异常，便用 w+方式打开文件，表示若文件不存在则创建一个名为 userdata 的可读可写的 txt 文件，这个方法会覆盖源文件内容。处理异常后将异常信息打印到日志中去。此功能用来记录程序运行时发生的异常信息。如果不想覆盖文件原内容可用 a+为打开方式在文件中追加内容。

1.3.2 终极行为

在 try 中使用 finally 进行终极行为，不管代码是否出现异常，finally 部分代码都会执行，比如在读写文件时，文件处理完要关闭，用 finally 来定义文件关闭操作。不管文件读写操作过程中是否出现了异常，文件都必须要执行关闭语句。

1.3.3 非常规的流程控制

还有一个就是当利用异常处理实现非常规的控制时，就可以使用 raise 语句，根据需要主动抛出异常，这些异常可以是内置定义的异常，也可以是用户自定义异常。实现流程的跳
转。

2.Python 中的异常层级

Python 面向对象程序设计通常使用类体系也就是说某个类继承自另外的类或数据类型 在 python 中，这一类体系的起点是 object 类。每个其他类都是从这个类（或继承这个类的其他类）继承而来，子类是一个继承其他类的类[3]。因而，所有的 python 类（object 本身除外）都是子类，因为这些类继承自 object 类。异常类都有共同的超类 BaseException，它的子类是 Exception、KeyboardInterrupt 和 SystemExit。BaseException 是所有异常类的顶级超类，不能直接继承。BaseException 为它的子类提供了默认的打印和状态保存的行为，在创建异常类的实例对象的时候可以用一个字符串来作为异常表述的信息，在打印异常类实例的时候，就会显示异常描述信息。当用户使用依赖于操作系统的组合

（CTRL+C）显示地打断程序的执行时会抛出 KeyboardInterrupt 类型的异常，sys 模块的 sys.exit()会抛出 SystemExit 类型的异常。Exception 是其他所有内建异常类的基类。

以下图一为 Python 异常层级的图表，下面所述的所有内置异常名字都以 Error 结尾。在 Python 中，“错误”和“异常”经常可以交互使用。在使用多个 expect 块时，必须对其排序，从异常体系中最底层到异常体系中最高层的异常。

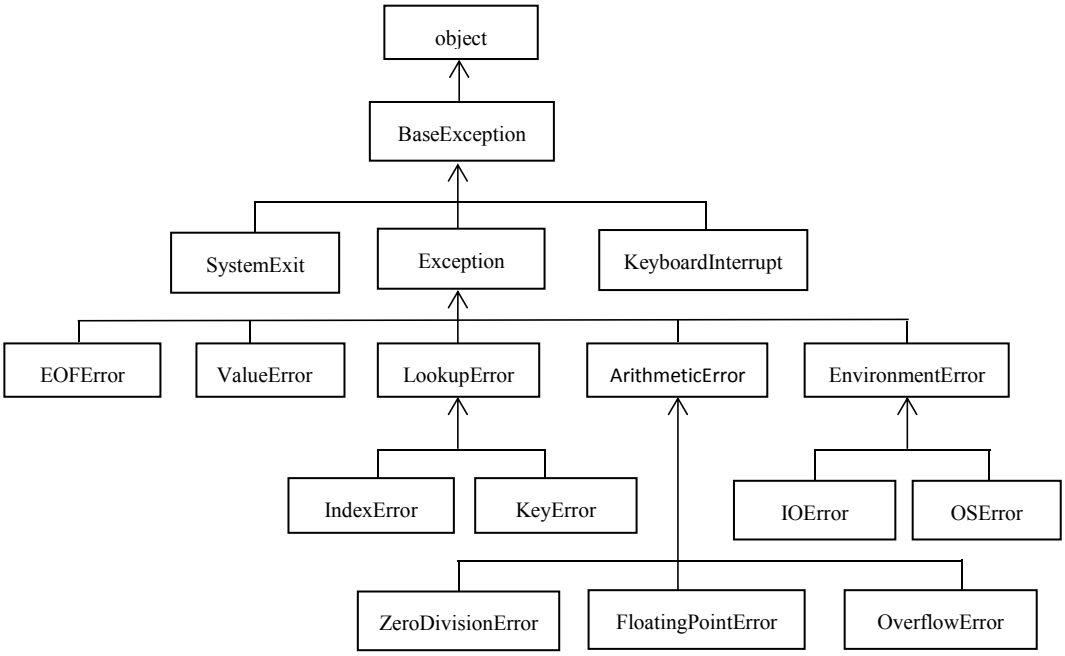

图 1 Python 异常层级图表

3.异常处理机制

3.1 什么是异常处理

异常处理是 Python 的一种高级工具，当程序代码执行过程中一旦出现异常，程序就会停止执行工作，在异常处理时当然不希望程序崩溃，这时就需要发现异常就跳转到异常处理的部分，不让程序停在那里 down 机（口语里面我们简单的把停掉机器叫做 down 机）。异常在程序运行过程中是可以捕捉处理的，这样可以避免我们的程序一旦出现异常，就会崩溃，或者卡死在那里。所以说我们对程序的异常进行处理是一种良好的编程习惯。

3.2 异常处理的基本结构

3.2.1 try/except/else

try 语句中放可能出现异常的代码,如果出现异常执行，except 用来捕捉相应异常类型,else 是 try 语句中的代码没有发生异常时执行的语句[4]。是可以省略的。当一次需要捕捉多个异常时，可以通过写多个 except 来指定这段代码可以同时处理几种异常，同时指定多种异常的时候，如果想使用相同的异常处理代码进行统一处理，就是我们有多种异常，但在处理过程中只用一段代码来处理，可以用 except……as。下面为一个处理读已注册用户信息文件时异常的代码：

```
def read_info(self):
        old_info = []
        try:
            userfile_txt = open("userdata.txt")# 打开文件，并返回一个文件对象
        except:
            print("暂未保存数据信息")
            return
        while True:
            info = userfile_txt.readline()# 读取文件内容
            if not info:
```

```
                    break
                info = info.rstrip()
            #将已有用户信息分别对应 name 和 password
                name,password = info.split(" ")

old_info.append(user(name,password))
            userfile_txt.close()
            return old_info
```

打开文件，若没有读到文件或是读取过程中出现问题，则提示“暂未保存数据信息”。然后进行循环读取文件中的内容，readline 表示每次读取一行。如果没有读到内容则退出循环，rstrip 去掉末尾的空格，split 将文件中的数据以空格为分界线放入 name 和 password 中，然后将已注册的用户读取进来。

3.2.2 捕捉所有异常

当不清楚会出现什么异常，就在 except 后面不写任何语句，不管出现任何异常，都会在 except 中处理。异常处理还可以进行嵌套，当发生异常时，内部没有捕获到的异常，外部可以捕捉。

3.2.3 主动引发异常

try 语句中引导会出现的代码，还有主动引发异常通过 raise 或是 assert 语句来主动引发异常。assert 语句在“测试表达式”的值为假时，引发 AssertionError 异常，data 作为异常描述信息。raise 后面跟一个异常类的名字时，就会创建这个类的实例对象，然后引发异常。raise 后面跟一个异常类实例对象的名字时，就会引发它对应的异常。如果 raise 后面什么都不加的话就是默认重新引发刚刚发生的异常。在使用 raise 来主动引发异常时，可以给异常指定描述信息[5]。以下为删除和修改用户信息为例:

```
    #删除用户信息
        def del_user_info(self,del_name = ''):
            if not del_name:
                del_name = input("请输入删除的用户姓名：")
            for info in self.user_info:
                if del_name == info.name:
                    return info
            raise IndexError("no match %s" %del_name)
    #修改用户信息
        def mod_user_info(self):
            mod_name = input("请输入修改的用户名：")
            for info in self.user_info:
                if mod_name == info.name:
                    a = int(input("请输入新密码："))
                    info = userinfo(mod_name,a)
                    return info
            raise IndexError("no match %s" %mod_name)
```

该代码的功能是由用户指定删除和修改目标用户的名字进行删除，若输入的目标用户在信息库里是没有的，则会主动引发 IndexError 并打印自定义的描述信息。比如我们要删除一个不存在的用户“abc”时，我们输入目标用户名称，当检测到不存在时就会打印我们自定义的异常内容。no match 加上目标用户名称。

```
>>>请输入删除的用户姓名：abc
no match abc
```

如果没有处理这个异常的话，在程序运行时删除一个不存在的用户就会报错。如：

```
>>>请输入删除的用户姓名：abc
Traceback (most recent call last):
  File "<stdin>", line 116, in entrence
    self.user_info.remove(self.del_user_info())
ValueError: list.remove(x): x not in list
```

3.2.4 异常的嵌套

Python 允许在异常结构处理的内部嵌套另一个异常处理结构。在发生异常时，内部没有捕捉处理的异常可以被外层捕捉。

3.3 自定义异常类

内置的异常类只能提供常规的异常处理，如果是特殊的异常处理，比如将异常写入文件里，或者某个行业特别规定的一个异常处理，就需要自己写一个异常类。当想抛出一个异常但是发现所有的内置异常都没有办法满足我们的要求，我们就可以定义一个新的符合自己需求的异常。例：我们定义一个名为 test 的空的异常类，

它的超类为 Exception，然后用 raise 来引发这个自定义的异常类。

```
>>>class test(Exception):pass
…
>>>raise test("这是一个自定义异常类")
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
__main__.test: 这是一个自定义异常类
```

4.结束语

使用异常做程序流控可以用于一些方便的程序设计。异常不是一个我们需要完全去避免的坏事，一个异常发生不是意味着你要阻止它发生，相反，它是代码之间交互信息的方式。在不需要调用函数显示地检查返回值的情况下，我们可以使用异常来处理例外情况或者错误条件。

参考文献

[1]陈晶. 基于消息队列通信中间件的 PDM 与 ERP 系统集成的研究[D]. 河海大学, 2004.

[2]李宇翔. 网络化智能血沉分析仪的研究与设计[D].南京邮电大学,2017.

[3]曹志月. 时空数据模型的研究及其在时空地图可视化系统中的应用[D]. 2001.

[4]兆翦. 解析媒体矩阵(MediaMatrix)(四十九)NWare 软件应用(11)——Python 编程基础(续 5)[J]. 音响技术, 2008(8):13-15.

[5]Aaron Hall. Manually raising (throwing) an exception in Python[EB/OL].(2014-06-05)[2019-03-25].https://stackoverflow.com/questions/2052390/manually-raising-throwing-an-exception-in-python

作者简介

第一作者：王洪汇（1998-），女，汉，四川省成都市，本科，四川大学锦城学院，研究方向：大数据技术

第二作者（通讯作者）：王科（1985—），男，讲师，硕士研究生，四川大学锦城学院，研究方向 ：云计算

第三作者：李晨曦（1998-7），男，汉，贵州省贵阳市，本科，四川大学锦城学院，研究方向：大数据技术开发