异常处理方法

**具体信息参考以下网站**

**<https://m.jb51.net/article/139842.htm>**

**<http://www.cnblogs.com/caicaihong/p/6402245.html>**

**<http://www.runoob.com/python/python-exceptions.html>**

**第一种：**

try:

正常的操作

......................

except:

发生异常，执行这块代码

......................

else:

如果没有异常执行这块代码

首先执行try语块中的语句，若发生错误，则执行except语块，若没有错误则执行完try语块后执行else语块（跳过except语块），这种结构不需要知道发生什么样的错误，反正是只要有错误，就跳转到except语块，否则不。**它能捕获所有的异常**

**例子：**

from urllib.request import Request,urlopen

req1=Request("http://www.ooxx-fishc.com")

req2=Request("http://www.fishc.com/ooxx.html")

try:

response1=urlopen(req1)

response2=urlopen(req2)

except:

print("we failed to reach a server.")

**结果：**we failed to reach a server.

**第二种：捕获多个异常(区分优先级)**

try:

<语句> #运行代码

except <异常1>：

<语句> #如果在try部份引发了'name1'异常，则执行这一语句

except <异常2>，<数据>:

<语句> #如果引发了'name2'异常，则执行这一语句，并获得附加的数据

else:

<语句> #如果没有异常发生

原理：如果在try语句中发生异常，则根据异常发生的类型匹配相应的except语句（根据异常进行匹配），并执行相应语句（异常1，异常2 后的语句），若匹配则执行，若不匹配则不会执行并跳过，如果都不匹配则抛出异常，而且只要try中语句发生错误则就会进行匹配跳转执行，则发生错误的语句的后面的语句就不会在执行了。

执行try下的语句，如果引发异常，则执行过程会跳到第一个except语句。

如果第一个except中定义的异常与引发的异常匹配，则执行该except中的语句。

如果引发的异常不匹配第一个except，则会搜索第二个except，允许编写的except数

量没有限制。

**如果所有的except都不匹配，则异常会传递到下一个调用本代码的最高层try代码中。**

**重要例子：**

try:

x=1

try:

fh.write("这是一个测试文件，用于测试异常!!")

except IOError:

print("Error: 没有找到文件或读取文件失败0")

finally:

print("关闭文件")

except:

print("Error: 没有找到文件或读取文件失败1")

**结果：**

关闭文件

Error: 没有找到文件或读取文件失败1

可见在内层的try语句中发生了错误，引发异常，但是没有相应的except语句相对应，执行了finally语句，由于在内层没有找到相匹配的except，因此将异常传递给了外层的try except语句，然后找到相应的except语句块执行，如果还没有找到匹配的，则会抛出异常。（finallly会在抛出异常之前执行）

**例子：**

from urllib.request import Request,urlopen

from urllib.error import URLError,HTTPError

req1=Request("http://www.ooxx-fishc.com")

req2=Request("http://www.fishc.com/ooxx.html")

try:

response1=urlopen(req1)

response2=urlopen(req2)

except URLError as e:

print("we failed to reach a server.")

print("Reason :",e.reason)

except HTTPError as e:

print("the server couldn\’ t fulfill the request.")

print("Error code :",e.code)

**结果：**

we failed to reach a server.

Reason : [Errno 11001] getaddrinfo failed

以上的例子中try语块中的两个语句都会发生错误，但是由于第一个语句发生异常，跳到except语句处进行匹配，匹配到则跳到相应的代码处执行，否则不执行，由于监控到第一个语句有误，则跳转到相应的except处执行，不再跳回带try语块继续执行后面的语句，因此第二个语句应该没有执行，就已经跳转了。，所以只输出了一种错误。Try…except…解决了程序出现错误以后则停止继续执行程序的问题。

**第三种：捕获多个异常（不区分优先级）**

try:

正常的操作

......................

except**(**Exception1[, Exception2[,...ExceptionN]]]**):**

发生以上多个异常中的一个，执行这块代码

......................

else:

如果没有异常执行这块代码

与第二种相类似，在匹配的时候具有更多的选择，只要有一个匹配的上就能跳转到except语句块处执行。

**例子：**

from urllib.request import Request,urlopen

from urllib.error import URLError,HTTPError

req1=Request("http://www.ooxx-fishc.com")

req2=Request("http://www.fishc.com/ooxx.html")

try:

response1=urlopen(req1)

response2=urlopen(req2)

except (URLError,HTTPError):

print("we failed to reach a server.")

结果：we failed to reach a server.

**第四种：**

如果try中的异常没有在exception中被指出，那么系统将会抛出Traceback(默认错误代码）,并且终止程序，接下来的所有代码都不会被执行，但如果有Finally关键字，则会在程序抛出Traceback之前（程序最后一口气的时候），执行finally中的语句。这个方法在某些必须要结束的操作中颇为有用，如释放文件句柄，或释放内存空间等。

try:

<语句>

finally:

<语句> #退出try时总会执行

这种结构不管如何finally语句块一定会被执行。

**例子：**

str1 = 'hello world'

try:

int(str1)

except IndexError as e:

print(e)

print(1)

except KeyError as e:

print(e)

print(2)

except ValueError as e:

print(e)

print(3)

else:

print('try内没有异常')

finally:

print('无论异常与否,都会执行我')

**结果：**

invalid literal for int() with base 10: 'hello world'

3

无论异常与否,都会执行我

**例子：**

str1 = 'hello world'

try:

int(str1)

try:

fh.write("这是一个测试文件，用于测试异常!!")

finally:

print("关闭文件")

fh.close()

except:

print("Error: 没有找到文件或读取文件失败")

**结果：**Error: 没有找到文件或读取文件失败

**raise（主动触发异常）**

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），如果存在，是跟踪异常对象。

**例子：**

def not\_zero(num):

try:

if num == 0:

raise ValueError('参数错误')

return num

except Exception as e:

print(e)

not\_zero(0)

**结果：**参数错误

解释：raise异常类名（参数），其实建立了一个异常类的实例，并且传入参数，如果不传入，则认为参数为空None。**e用来存储异常发生的具体信息**

**def temp\_convert(var):**

**try:**

**return int(var)**

**except ValueError[,Argument]:**

**print("参数没有包含数字\n",Argument)**

**temp\_convert("xyz");**

**上面的例子是带参数的，程序有问题**

第一种写法：

from urllib.request import Request,urlopen

from urllib.error import URLError,HTTPError

req=Request(someurl)

try:

response=urlopen(req)

except HTTPError as e:

print(“the server couldn\’ t fulfill the request.”)

print(“Error code :”,e.code)

except URLError as e:

print(“we failed to reach a server.”)

print(“Reason :”,e.reason)

else:

# everything is fine

第二种写法(推介写法)：

from urllib.request import Request,urlopen

from urllib.error import URLError

req=Request(someurl)

try:

response=urlopen(req)

except URLError as e:

if hasattr(e,’reason’):

print(“we failed to reach a server.”)

print(“Reason :”,e.reason)

elif hasattr(e,’code’):

print(“the server couldn\’ t fulfill the request.”)

print(“Error code :”,e.code)

else:

# everything is fine