# Python 异常处理

python提供了两个非常重要的功能来处理python程序在运行中出现的异常和错误。你可以使用该功能来调试python程序。

* 异常处理: 本站Python教程会具体介绍。
* 断言(Assertions):本站Python教程会具体介绍。

## python标准异常

|  |  |
| --- | --- |
| **异常名称** | **描述** |
|  |  |
| BaseException | 所有异常的基类 |
| SystemExit | 解释器请求退出 |
| KeyboardInterrupt | 用户中断执行(通常是输入^C) |
| Exception | 常规错误的基类 |
| StopIteration | 迭代器没有更多的值 |
| GeneratorExit | 生成器(generator)发生异常来通知退出 |
| StandardError | 所有的内建标准异常的基类 |
| ArithmeticError | 所有数值计算错误的基类 |
| FloatingPointError | 浮点计算错误 |
| OverflowError | 数值运算超出最大限制 |
| ZeroDivisionError | 除(或取模)零 (所有数据类型) |
| AssertionError | 断言语句失败 |
| AttributeError | 对象没有这个属性 |
| EOFError | 没有内建输入,到达EOF 标记 |
| EnvironmentError | 操作系统错误的基类 |
| IOError | 输入/输出操作失败 |
| OSError | 操作系统错误 |
| WindowsError | 系统调用失败 |
| ImportError | 导入模块/对象失败 |
| LookupError | 无效数据查询的基类 |
| IndexError | 序列中没有此索引(index) |
| KeyError | 映射中没有这个键 |
| MemoryError | 内存溢出错误(对于Python 解释器不是致命的) |
| NameError | 未声明/初始化对象 (没有属性) |
| UnboundLocalError | 访问未初始化的本地变量 |
| ReferenceError | 弱引用(Weak reference)试图访问已经垃圾回收了的对象 |
| RuntimeError | 一般的运行时错误 |
| NotImplementedError | 尚未实现的方法 |
| SyntaxError | Python 语法错误 |
| IndentationError | 缩进错误 |
| TabError | Tab 和空格混用 |
| SystemError | 一般的解释器系统错误 |
| TypeError | 对类型无效的操作 |
| ValueError | 传入无效的参数 |
| UnicodeError | Unicode 相关的错误 |
| UnicodeDecodeError | Unicode 解码时的错误 |
| UnicodeEncodeError | Unicode 编码时错误 |
| UnicodeTranslateError | Unicode 转换时错误 |
| Warning | 警告的基类 |
| DeprecationWarning | 关于被弃用的特征的警告 |
| FutureWarning | 关于构造将来语义会有改变的警告 |
| OverflowWarning | 旧的关于自动提升为长整型(long)的警告 |
| PendingDeprecationWarning | 关于特性将会被废弃的警告 |
| RuntimeWarning | 可疑的运行时行为(runtime behavior)的警告 |
| SyntaxWarning | 可疑的语法的警告 |
| UserWarning | 用户代码生成的警告 |

## 什么是异常？

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。

## 异常处理

捕捉异常可以使用try/except语句。

try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。

如果你不想在异常发生时结束你的程序，只需在try里捕获它。

语法：

以下为简单的try....except...else的语法：

try:<语句>#运行别的代码except<名字>：<语句>#如果在try部份引发了'name'异常except<名字>，<数据>:<语句>#如果引发了'name'异常，获得附加的数据else:<语句>#如果没有异常发生

try的工作原理是，当开始一个try语句后，python就在当前程序的上下文中作标记，这样当异常出现时就可以回到这里，try子句先执行，接下来会发生什么依赖于执行时是否出现异常。

* 如果当try后的语句执行时发生异常，python就跳回到try并执行第一个匹配该异常的except子句，异常处理完毕，控制流就通过整个try语句（除非在处理异常时又引发新的异常）。
* 如果在try后的语句里发生了异常，却没有匹配的except子句，异常将被递交到上层的try，或者到程序的最上层（这样将结束程序，并打印缺省的出错信息）。
* 如果在try子句执行时没有发生异常，python将执行else语句后的语句（如果有else的话），然后控制流通过整个try语句。

### 实例

下面是简单的例子，它打开一个文件，在该文件中的内容写入内容，且并未发生异常：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

try:

fh = open("testfile","w")

fh.write("这是一个测试文件，用于测试异常!!")exceptIOError:

print"Error: 没有找到文件或读取文件失败"else:

print"内容写入文件成功"

fh.close()

以上程序输出结果：

$ python test.py 内容写入文件成功

$ cat testfile # 查看写入的内容这是一个测试文件，用于测试异常!!

### 实例

下面是简单的例子，它打开一个文件，在该文件中的内容写入内容，但文件没有写入权限，发生了异常：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

try:

fh = open("testfile","w")

fh.write("这是一个测试文件，用于测试异常!!")exceptIOError:

print"Error: 没有找到文件或读取文件失败"else:

print"内容写入文件成功"

fh.close()

在执行代码前为了测试方便，我们可以先去掉 testfile 文件的写权限，命令如下：

chmod -w testfile

再执行以上代码：

$ python test.py Error:没有找到文件或读取文件失败

## 使用except而不带任何异常类型

你可以不带任何异常类型使用except，如下实例：

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

以上方式try-except语句捕获所有发生的异常。但这不是一个很好的方式，我们不能通过该程序识别出具体的异常信息。因为它捕获所有的异常。

## 使用except而带多种异常类型

你也可以使用相同的except语句来处理多个异常信息，如下所示：

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

## try-finally 语句

try-finally 语句无论是否发生异常都将执行最后的代码。

try:<语句>finally:<语句>#退出try时总会执行raise

### 实例

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

try:

fh = open("testfile","w")

fh.write("这是一个测试文件，用于测试异常!!")finally:

print"Error: 没有找到文件或读取文件失败"

如果打开的文件没有可写权限，输出如下所示：

$ python test.py Error:没有找到文件或读取文件失败

同样的例子也可以写成如下方式：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

try:

fh = open("testfile","w")

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()exceptIOError:

print"Error: 没有找到文件或读取文件失败"

当在try块中抛出一个异常，立即执行finally块代码。

finally块中的所有语句执行后，异常被再次触发，并执行except块代码。

参数的内容不同于异常。

## 异常的参数

一个异常可以带上参数，可作为输出的异常信息参数。

你可以通过except语句来捕获异常的参数，如下所示：

try:

正常的操作

......................exceptExceptionType,Argument:

你可以在这输出Argument的值...

变量接收的异常值通常包含在异常的语句中。在元组的表单中变量可以接收一个或者多个值。

元组通常包含错误字符串，错误数字，错误位置。

### 实例

以下为单个异常的实例：

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 定义函数def temp\_convert(var):

try:

returnint(var)

exceptValueError,Argument:

print"参数没有包含数字\n",Argument

# 调用函数

temp\_convert("xyz");

以上程序执行结果如下：

$ python test.py 参数没有包含数字

invalid literal forint()withbase10:'xyz'

### 触发异常

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）参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"。

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。

### 实例

一个异常可以是一个字符串，类或对象。 Python的内核提供的异常，大多数都是实例化的类，这是一个类的实例的参数。

定义一个异常非常简单，如下所示：

def functionName( level ):

if level <1:

raiseException("Invalid level!", level)

# 触发异常后，后面的代码就不会再执行

**注意：**为了能够捕获异常，"except"语句必须有用相同的异常来抛出类对象或者字符串。

例如我们捕获以上异常，"except"语句如下所示：

try:

正常逻辑except"Invalid level!":

触发自定义异常else:

其余代码

### 实例

#!/usr/bin/python# -\*- coding: UTF-8 -\*-

# 定义函数def mye( level ):

if level <1:

raiseException("Invalid level!", level)

# 触发异常后，后面的代码就不会再执行

try:

mye(0)// 触发异常except"Invalid level!":

print1else:

print2

执行以上代码，输出结果为：

$ python test.py Traceback(most recent call last):

File"test.py", line 11,in<module>

mye(0)

File"test.py", line 7,in mye

raiseException("Invalid level!", level)Exception:('Invalid level!',0)

## 用户自定义异常

通过创建一个新的异常类，程序可以命名它们自己的异常。异常应该是典型的继承自Exception类，通过直接或间接的方式。

以下为与RuntimeError相关的实例,实例中创建了一个类，基类为RuntimeError，用于在异常触发时输出更多的信息。

在try语句块中，用户自定义的异常后执行except块语句，变量 e 是用于创建Networkerror类的实例。

classNetworkerror(RuntimeError):

def \_\_init\_\_(self, arg):

self.args = arg

在你定义以上类后，你可以触发该异常，如下所示：

try:

raiseNetworkerror("Bad hostname")exceptNetworkerror,e:

print e.args

# Python异常处理总结

本文较为详细的罗列了Python常见的异常处理，供大家参考，具体如下：

**1. 抛出异常和自定义异常**

Python用异常对象(exception object)表示异常情况，遇到错误后，会引发异常。如果异常对象并未被处理或捕捉，程序就会用所谓的回溯(Traceback,一种错误信息)终止执行。

1. **.raise 语句**

Python中的raise 关键字用于引发一个异常，基本上和C#和Java中的throw关键字相同，如下所示：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # -- coding: utf-8 --    def ThorwErr():    raise Exception("抛出一个异常")    # Exception: 抛出一个异常  ThorwErr() |

raise关键字后面是抛出是一个通用的异常类型(Exception)，一般来说抛出的异常越详细越好，Python在exceptions模块内建了很多的异常类型，通过使用dir函数来查看exceptions中的异常类型，如下：

|  |  |
| --- | --- |
| 1  2  3 | import exceptions    # ['ArithmeticError', 'AssertionError'.....]  print dir(exceptions) |

**传递异常**

捕捉到了异常，但是又想重新引发它(传递异常)，可以使用不带参数的raise语句即可：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # -- coding: utf-8 --  class MuffledCalculator:    muffled = False    def calc(self,expr):      try:        return eval(expr)      except ZeroDivisionError:        if self.muffled:          print 'Division by zero is illegal'        else:          raise |

**②.自定义异常类型**

Python中也可以自定义自己的特殊类型的异常，只需要要从Exception类继承(直接或间接)即可：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1 | class SomeCustomException(Exception):    pass |

**2. 捕捉异常**

和C#中的try/catch类似，Python中使用try/except关键字来捕捉异常，如下：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | # -- coding: utf-8 --    try:    print 2/0  except ZeroDivisionError:    print '除数不能为0' |

**①.捕捉多个异常**

在一个except语句只捕捉其后声明的异常类型，如果可能会抛出的是其他类型的异常就需要再增加一个except语句了，或者也可以指定一个更通用的异常类型比如：Exception，如下：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | # -- coding: utf-8 --  try:    print 2/'0'  except ZeroDivisionError:    print '除数不能为0'  except Exception:    print '其他类型异常' |

为了捕获多个异常，除了声明多个except语句之外，还可以在一个except语句之后将多个异常作为元组列出来即可：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3 | # -- coding: utf-8 --  try:    print 2/'0'  except (ZeroDivisionError,Exception):    print '发生了一个异常' |

**②.获取异常信息**

每个异常都会有一些异常信息，一般情况下我们应该把这些异常信息记录下来：

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3  4 | # -- coding: utf-8 --  try:    print 2/'0'  except (ZeroDivisionError,Exception) as e:    # unsupported operand type(s) for /: 'int' and 'str'    print e |

**3. finally子句**

finally子句和try子句联合使用但是和except语句不同，finally不管try子句内部是否有异常发生，都会执行finally子句内的代码。所有一般情况下，finally自己常常用于关闭文件或者在Socket中。

[?](http://www.jb51.net/article/53783.htm)

|  |  |
| --- | --- |
| 1  2  3  4  5 | # -- coding: utf-8 --  try:    print 2/'0'  except (ZeroDivisionError,Exception):    print '发生了一个异常'  finally:    print '不管是否发生异常都执行' |