Python异常处理

在Python中 异常会根据错误自动的被触发 也能由代码触发和截获 异常由以下四个语句来处理

try/except #捕获异常并恢复
try/finally #无论异常是否发生 都执行清理行为
raise #手工在代码中触发异常
assert # 有条件的在代码中触发异常
with/as ##实现环境管理

异常的角色

错误处理
每当在运行时检测到程序错误时 Python就会引发异常 可以在程序代码中捕获和响应错误 或者忽略已经发生的异常， Python代码的默认异常处理行为是当代码发生错误的时候 停止程序 并打印出错误消息. 如果不想启动这种默认行为 就要写try语句来捕获异常 并且从异常中恢复 当检测到错误的时候 Python会跳到try处理器 而程序会在try之后重新继续执行

事件通知
当发生错误的时候 可以在程序之间传递有效状态的信号

终止行为
try/finally 保证一定会进行指定的操作 无论程序是否异常 如关闭文件

非常规控制流程
它是一种高级的 goto 可以作为实现非常规的流程控制

具体例子：

1.捕获异常
```
In [2]: try:
   ...:     print 3/0   ##这里会报错 因为分母不可以为0
   ...: except:
   ...:     print 'except'
   ...:
except
```
#当 try代码块执行时 触发异常,python会跳转到处理器, 执行except下面的语句， try

不仅会捕捉异常 也会从异常中恢复执行

```
In [3]: try:
   ...:     print  3/0
   ...: except:
   ...:     print 'except'
   ...: print 'continue'
   ...:
except
continue
```

```
a = 10
try:
    #c = a / 0
    c = a / d
except ZeroDivisionError:
    print 'this is except!!!!'
except NameError:
    print 'this is nameError!!!'

print 'hello!!'
~
```

```
a = 10
try:
    c = a / 0
    #c = a / d
except Exception as e:
    print 'this is except !!! %s' %e

print 'hello!!'
```

2.引发异常

在Python中程序员可以人为的来生成一个异常，通过raise来生成异常

```
In [4]: raise IndexError
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-4-ef7a68a2d97e> in <module>()
----> 1 raise IndexError

IndexError:

In [5]: try:
   ...:     raise IndexError
   ...: except:
   ...:     print 'manual raise'
   ...:
manual raise
```

3 用户自定义异常
raise只能触发内置异常 当然我们也可以定义自己的新的异常 它特定于我们自己的程序 用户定义的异常可以通过类编写，它继承了内置的异常类Exception

```
class badString(Exception):
    pass

try:
    raise badString
except:
    print 'success catch'
~
[root@yeslab python]# python demo2.py
success catch
```

4.终止行为
try/finally ##不管错误是否发生 它都会执行

```
In [4]: try:
   ...:     print 3/0
   ...: except:
   ...:     print 'except'
```

```
...: finally:
...:     print 'finally'
...: print 'after try'
...:
except
finally
after try   ##代码发生错误 finally下面的语句依然会执行
```

异常代码编写的细节

try/except/else/finally #其中 else 和 finally是可选的

在这个机构的语句中 try 下面的代码是此语句的主要动作， 在执行这个代码的过程中except来捕捉异常 else的作用在于没发生异常的时候要执行的语句

如

```
try:
    print 3 / 1
except:
    print 'except'
else:
    print 'else'
finally:
    print 'finally'
print 'after try'
```

结果

```
except
finally
after try
~
```

```
try:
    print 3 / 0
except:
    print 'except'
else:
    print 'else'
finally:
    print 'finally'
```

```
print 'after try'
~
结果#
3
else
finally
after try
```

在使用 except 进行捕捉的时候， 如果要想捕捉一切错误 空的except就可以做到
如果想要捕捉指定的错误 以及如果想要捕捉 指定的一组错误 可使用下面的语法

```
In [1]: try:
   ...:     print 3 / 0
   ...: except ZeroDivisionError:
   ...:     print 'zero'
   ...:
zero

In [2]: try:
   ...:     l = [1, 2, 3]
   ...:     l[4]
   ...: except ZeroDivisionError:  ##只捕捉除0错误
   ...:     print 'zero'
   ...:
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-2-6b677378168b> in <module>()
      1 try:
      2     l = [1, 2, 3]
----> 3     l[4]
      4 except ZeroDivisionError:
      5     print 'zero'

IndexError: list index out of range  ##因为只捕捉除0错误 所以索引错误无法被捕捉

In [4]: try:
    print 3 / 0
except (ZeroDivisionError, IndexError):  ##同时捕捉一组错误
    print '123'
```

...:
123

assert语句

语法 assert <test>, <data>
当test 为假的时候 引发异常 AssertError, 并且data作为错误内容展示出来, 如果test为真则没有发生错误

In [1]: assert 0, 'abc'

AssertionError Traceback (most recent call last)
<ipython-input-1-bc7a908c1c73> in <module>()
----> 1 assert 0, 'abc'
AssertionError: abc
In [2]: assert 1, '123'

with/as环境管理器
with/as语句的设计是作为常见的 try/finally 用法模式的替代方案， with/as语句也是用于定义必须执行的终止或清理行为 无论步骤中是否发生异常

基本使用
with expression [as variable]
    with-block
这里的 expression 返回一个对象 从而支持环境管理协议, 并将这个对象赋值给 as 后面的变量
例如
with open('20160913.log') as file:
    for line in file:
        print line
对 open的调用 会返回一个简单文件对象 赋值给变量myfile, with语句所使用的环境管理协议 会保证由myfile所引用的文件对象会自动关闭
类似:
file = open('20160913.log')
try:
    for line in file:
        print line
finally:

```
    file.close()
# 通过 with/as  与 try/finally  功能相同 但是明显前者更加的精简
```

异常的打印和状态

```
In [4]: try:
   ...:     print 3 / 0
   ...: except ZeroDivisionError as x:
   ...:     print x
   ...:
integer division or modulo by zero
```

定制打印信息

```
In [1]: class myBad(Exception):
   ...:     pass
   ...:
In [3]: try:
    raise myBad('error')
except myBad as e:
    print e
   ...:
error
```

使用traceback获取详细的异常信息

```
import traceback
try:
    1 / 0
except:
    print 'error'


try:
    1 / 0
except:
    traceback.print_exc()


f= open('error.log', 'a+')
```

```
try:
    1 / 0
except:
    traceback.print_exc(file=f)  ###将报错写入文件

f.flush()
f.close()
```

```
[root@yeslab python]# python demo20.py
error   ##第一个错误  我们只能看到 Except 捕捉到错误之后所打印的输出 没有详细信息可以看到


Traceback (most recent call last):
  File "demo20.py", line 9, in <module>
    1 / 0
ZeroDivisionError: integer division or modulo by zero  ## 可以看到哪个脚本  哪一行出错
```