

# 异常 Exception

## 错误 Error

逻辑错误：算法写错了，例如加法写成了减法

笔误：例如变量名写错了，语法错误

函数或类使用错误，其实这也属于逻辑错误

总之，错误是可以避免的

## 异常 Exception

本意就是意外情况

这有个前提，没有出现上面说的错误，也就是说程序写的没有问题，但是在某些情况下，会出现一些意外，导致程序无法正常的执行下去。

例如open函数操作一个文件，文件不存在，或者创建一个文件时已经存在了，或者访问一个网络文件，突然断网了，这就是异常，是个意外的情况。

异常不可能避免

## 错误和异常

在高级编程语言中，一般都有错误和异常的概念，异常是可以捕获，并被处理的，但是错误是不能被捕获的。

举例

对比异常和错误

```
1  with open('testabc') as f:
2      pass
3
4  # 异常
5  Traceback (most recent call last):
6      File "test.py", line 4, in <module>
7          with open('testabc') as f:
8      FileNotFoundError: [Errno 2] No such file or directory: 'testabc'
9
10 def OA():
11     pass
12
13 # 错误
14 File "test.py", line 3
15     def OA():
16         ^
17 SyntaxError: invalid syntax
```

一个健壮的程序

- 尽可能的避免错误
- 尽可能的捕获、处理各种异常

## 产生异常

产生:

- raise 语句显式的抛出异常
- Python解释器自己检测到异常并引发它

```
1 def foo():
2     print('before')
3     1/0 # 除零异常
4     raise Exception('my exception') # raise主动抛出异常
5     print('after')
6 bar()
```

程序会在异常抛出的地方中断执行，如果不捕获，就会提前结束程序（其实是终止当前线程的执行）

## 异常的捕获

```
1 try:
2     待捕获异常的代码块
3 except [异常类型]:
4     异常的处理代码块
```

```
1 def foo():
2     try:
3         print('before')
4         c = 1/0
5         print('after')
6     except:
7         print('catch u')
8         print('finished')
9
10 foo()
11 print('==== end ====')
```

上例执行到 `c = 1/0` 时产生异常并抛出，由于使用了try...except语句块则捕捉到了这个异常，异常生成位置之后语句将不再执行，转而执行对应的except部分的语句，最后执行try...except语句块之外的语句。

## 捕获指定类型的异常

```
1 def foo():
2     try:
3         print('before')
4         c = 1/0
5         print('after')
6     except ArithmeticError: # 指定捕获的类型
7         print('catch u')
8         print('finished')
9
10 foo()
11 print('==== end ====')
```

## 异常类及继承层次

```
1  # Python异常的继承
2
3  BaseException
4  +-- SystemExit
5  +-- KeyboardInterrupt
6  +-- GeneratorExit
7  +-- Exception
8      +-- RuntimeError
9          |  +-- RecursionError
10     +-- MemoryError
11     +-- NameError
12     +-- StopIteration
13     +-- StopAsyncIteration
14     +-- ArithmeticError
15         |  +-- FloatingPointError
16         |  +-- OverflowError
17         |  +-- ZeroDivisionError
18     +-- LookupError
19         |  +-- IndexError
20         |  +-- KeyError
21     +-- SyntaxError
22     +-- OSError
23         |  +-- BlockingIOError
24         |  +-- ChildProcessError
25         |  +-- ConnectionError
26         |      |  +-- BrokenPipeError
27         |      |  +-- ConnectionAbortedError
28         |      |  +-- ConnectionRefusedError
29         |      |  +-- ConnectionResetError
30         |  +-- FileExistsError
31         |  +-- FileNotFoundError
32         |  +-- InterruptedError
33         |  +-- IsADirectoryError
34         |  +-- NotADirectoryError
35         |  +-- PermissionError
36         |  +-- ProcessLookupError
37         |  +-- TimeoutError
38
```

## BaseException及子类

### BaseException

所有内建异常类的基类是BaseException

### SystemExit

sys.exit()函数引发的异常，异常不捕获处理，就直接交给Python解释器，解释器退出。

```

1 import sys
2
3 print('before')
4 sys.exit(1)
5 print('SysExit')
6 print('after') # 是否执行?

```

```

1 # 捕获这个异常
2 import sys
3 try:
4     print('before')
5     sys.exit(1)
6     print('after')
7 except SystemExit: # 换成Exception能否捕获
8     print('SysExit')
9 print('outer') # 是否执行?

```

如果except语句捕获了该异常，则继续向后面执行，如果没有捕获住该异常SystemExit，解释器直接退出程序。

注意捕获前后程序退出状态码的变化。

## KeyboardInterrupt

对应的捕获用户中断行为Ctrl + C

```

1 import time
2
3 try:
4     while True:
5         time.sleep(1)
6         print('running')
7 except KeyboardInterrupt:
8     print("Ctrl + c")
9 print('=' * 30)

```

## Exception及子类

Exception是所有内建的、非系统退出的异常的基类，自定义异常类应该继承自它

### SyntaxError 语法错误

Python将这种错误也归到异常类下面的Exception下的子类，但是这种错误是不可捕获的

```

1 def a():
2     try:
3         0a = 5
4     except:
5         pass
6
7 # 错误
8 File "test2.py", line 3
9     0a = 5
10     ^
11 SyntaxError: invalid syntax

```

## ArithmeticError

所有算术计算引发的异常，其子类有除零异常等

## LookupError

使用映射的键或序列的索引无效时引发的异常的基类：IndexError, KeyError

## 自定义异常类

从Exception继承的类

```
1 class MyException(Exception):
2     pass
3
4 try:
5     raise MyException()
6 except MyException: # 捕获自定义异常
7     print('catch u')
```

## 多种捕获

except可以指定捕获的类型，捕获多种异常

```
1 import sys
2 class MyException(Exception):
3     pass
4
5 try:
6     a = 1/0
7     raise MyException()
8     open('t')
9     sys.exit(1)
10 except ZeroDivisionError:
11     print('zero')
12 except ArithmeticError:
13     print('arith')
14 except MyException: # 捕获自定义异常
15     print('catch u')
16 except Exception:
17     print('exception')
18 except: # 写在最后，缺省捕获
19     print('error')
20
21 print('====end====')
```

### 捕获规则

- 捕获是从上到下依次比较，如果匹配，则执行匹配的except语句块
- 如果被一个except语句捕获，其他except语句就不会再次捕获了
- 如果没有任何一个except语句捕获到这个异常，则该异常向外抛出
- `except:` 称为缺省捕获，缺省捕获必须是最后一个捕获语句

### 捕获的原则

- 从小到大，从具体到宽泛

## as子句

先看一个例子

```
1 # raise 能抛出什么样的异常？
2 class A: pass
3
4 try:
5     # 1/0
6     raise 1
7     # raise "abc"
8     # raise A
9     # raise A()
10    # raise {}
11 except: # 写在最后，缺省捕获
12     print('catch u')
13
14 print('====end====')
```

raise真的什么类型都能抛出吗？

被抛出的异常，应该是异常类的实例，如何获得这个对象呢？使用as子句

```
1 # raise 能抛出什么样的异常？
2 class A: pass
3
4 try:
5     # 1/0
6     raise 1
7     # raise "abc"
8     # raise A
9     # raise A()
10    # raise {}
11 except Exception as e: # 写在最后，缺省捕获
12     print(type(e), e) # 抛出TypeError类型异常实例
13
14 print('====end====')
```

raise语句

- raise后要求应该是BaseException类的**子类或实例**，如果是类，将被**无参实例化**。自定义应该是Exception子类
- raise后什么都没有，表示抛出最近一个被激活的异常，如果没有被激活的异常，则抛类型异常。这种方式较少用，它用在except中

## finally子句

finally最终，即最后一定要执行的，try...finally语句块中，不管是否发生了异常，都要执行finally的部分

```
1 try:
2     f = open('test.txt')
3 except FileNotFoundError as e:
4     print('{} {} {}'.format(e.__class__, e.errno, e.strerror))
5 finally:
6     print('清理工作')
7     f.close() #
```

注意上例中的f的作用域，解决的办法是在外部定义f

finally中一般放置资源的清理、释放工作的语句

```
1 f = None
2 try:
3     f = open('test.txt')
4 except Exception as e:
5     print('{}'.format(e))
6 finally:
7     print('清理工作')
8     if f:
9         f.close()
```

也可以在finally中再次捕捉异常

```
1 try:
2     f = open('test.txt')
3 except Exception as e:
4     print('{}'.format(e))
5 finally:
6     print('清理工作')
7     try:
8         f.close()
9     except Exception as e:
10        print(e)
```

## 语句嵌套和捕获

- 异常语句内部可以嵌入到try块、except块、finally块中
- 异常在内部产生后，如果没有捕获到，就会继续向外部抛出
- 如果外部也没能捕获，将继续再向外部抛出，直至异常代码所在线程，导致线程崩溃
- finally中有return、break语句，则异常就不会继续向外抛出

```

1  try:
2      try:
3          1/0
4      except KeyError as e:
5          print(1, e)
6      finally:
7          print(2, 'inner fin')
8  except FileNotFoundError as e:
9      print(3, e)
10 finally:
11     print(4, 'outer fin')

```

## else子句

```

1  try:
2      ret = 1 * 0
3  except ArithmeticError as e:
4      print(e)
5  else:
6      print('OK')
7  finally:
8      print('fin')

```

else子句，没有任何异常发生，则执行

## 总结

```

1  try:
2      <语句>      #运行别的代码
3  except <异常类>:
4      <语句>      # 捕获某种类型的异常
5  except <异常类> as <变量名>:
6      <语句>      # 捕获某种类型的异常并获得对象
7  else:
8      <语句>      #如果没有异常发生
9  finally:
10     <语句>      #退出try时总会执行

```

1. 如果try中语句执行时发生异常，搜索except子句，并执行第一个匹配该异常的except子句
2. 如果try中语句执行时发生异常，却没有匹配的except子句，异常将被递交到外层的try，如果外层不处理这个异常，异常将继续向外层传递。如果都不处理该异常，则会传递到最外层，如果还没有处理，就终止异常所在的线程
3. 如果在try执行时没有发生异常，如有else子句，可执行else子句中的语句
4. 无论try中是否发生异常，finally子句最终都会执行