异常

1. 简介

当Python检测到一个错误时，解释器就无法继续执行了，反而出现了一些错误的提示，这就是所谓的"异常"

异常，不仅仅是错误，相对于正常而言，如银行取钱过程。

1.2 有哪些异常

```
# 索引异常
# IndexError: list index out of range
mylist = [1, 2] # 0 1
mylist[3]

# 语法异常
# SyntaxError: invalid character in identifier
mytuple = （1, 2） # 中文括号

# 属性异常
# AttributeError: module 'sys' has no attribute 'open'
import sys
sys.open('a.txt')


# 文件找不到异常
# FileNotFoundError: [Errno 2] No such file or directory: 'a.txt'
open('a.txt')

# 除零异常
# ZeroDivisionError: division by zero
1/0

# 类型异常
# TypeError: unsupported operand type(s) for +: 'int' and 'list'
100 + [1, 2]

# 模块未找到
# ModuleNotFoundError: No module named 'noexist'
import noexist

# 变量未定义
# NameError: name 'b' is not defined
a = b

# 键不存在
# KeyError: 'tel'
```

1.3 异常的类型

所有的异常，都继承自BaseException类，所有的常规异常都继承自Exception类，包含关系系如下:

- BaseException：所有异常的基类
- KeyboardInterrupt：用户中断执行
- SystemExit：解释器请求退出
- Exception：常规错误
    - NameError：访问未申明变量
    - ZeroDivisionError：除零错误
    - SyntaxError：解释器语法错误
    - IndexError：索引超出序列范围
    - IOError：输入、输出错误
- 异常关系：https://docs.python.org/3/library/exceptions.html#base-classes

1.4 为什么要使用异常

- 把错误处理和真正的工作分开来
- 封装完善、成熟的异常机制，帮助程序员减少多余切重复的校验工作
- 代码更易组织，更清晰，复杂的工作任务更容易实现
- 毫无疑问，更安全了，不至于由于一些小的疏忽而使程序意外崩溃了
- 所以它提供了一种方法使得程序的控制流可以安全的跳转到上层（或者上上层）的错误处理模块中去。

1.5 捕捉异常

```
# 捕获异常
try:
    print('---1----')
    open('nofile.txt','r')
    print('---2---')
except FileNotFoundError as e:
    print('捕获了异常')

# 捕获多个异常
try:
    print('---1----')
    print(num)
    open('nofile.txt','r')
    print('---2---')
except (FileNotFoundError,NameError):
    print('捕获了异常')

# 获取异常数据的内容
try:
    print('---1----')
    print(num)
    open('nofile.txt','r')
    print('---2---')
```

```
except (FileNotFoundError,NameError) as e:
    print('捕获了异常:',e)

# 捕获所有异常
try:
    print('---1----')
    1/0
    print(num)
    open('nofile.txt','r')
    print('---2---')
except Exception as e:
    print('捕获了异常:',e)
```

1.6 异常处理

(1) 语法：

```
try:
    可能触发异常的语句
except 错误类型1 [as 变量1]：
    处理语句1
except 错误类型2 [as 变量2]：
    处理语句2
except Exception [as 变量3]：
    不是以上错误类型的处理语句
else:
    未发生异常的语句
finally:
    无论是否发生异常的语句
```

(2) 作用：将程序由异常状态转为正常流程。

(3) 说明：

as 子句是用于绑定错误对象的变量，可以省略

except子句可以有一个或多个，用来捕获某种类型的错误。

else子句最多只能有一个。

finally子句最多只能有一个，如果没有except子句，必须存在。

如果异常没有被捕获到，会向上层(调用处)继续传递，直到程序终止运行。

- try...except...else...
    - 将可能出现问题的代码，放在try语句块中，当异常发生，就会被程序捕获并调转到except执行
      except中，放处理异常的代码，except后跟要捕获的异常的类型，用逗号分隔捕获多个异常。
      如果没有捕获到异常，就执行else当中的语句，else可选。
- try...finally...
    - 无论是否发生异常，都会执行finally后的代码块，一般用于关闭文件，销毁线程，释放锁等操作。

- 捕获异常的信息描述：except...as...

> 异常处理流程：
>
> - try --> 异常 --> except --> finally
> - try --> 无异常 --> else --> finally

```python
print('程序开始')
try:
    mydict = {'name':'zhangsan'}
    # mydict['tel']
    # try中抛出异常后面的代码不会执行
    # 1/0
    # a = b
    print('after mydict[tel]') # 没有打印
# 使用 except 捕捉异常
# 从上到下 挨个匹配，只会执行一个异常处理
except KeyError as e: # 捕捉到异常才执行
    print('e:',e)
    print('in except 捕捉异常')
except ZeroDivisionError as e:
    print('e:',e)
    print('in except ZeroDivisionError')
# 同时捕获多个异常
except (NameError, TypeError) as e:
    print('e:',e)
    print('in except NameError, TypeError')
# 捕获所有异常，Exception 是所有其他异常的父类
except Exception as e:
    print('e:', e)
    print('in except Exception')
else: # 可选
    print('当没有发生异常的时候执行')
finally: # 可选
    print('无论有没有异常都会执行')

print('程序结束')
```

1.7 异常抛出 raise

- 之前的代码，都是故意使用错误的代码产生的异常，通过raise可以手动抛出一个异常
  `raise IndexError`
  `raise IndexError()`
- 在Python中，所有的异常都是一个异常类的实例

(1) 作用：抛出一个错误，让程序进入异常状态。

(2) 目的：在程序调用层数较深时，向主调函数传递错误信息要层层return比较麻烦，所以人为抛出异常，可以直接传递错误信息。

```python
class Wife:
    def __init__(self, age):
        self.age = age
```

```python

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, value):
        if 20 <= value <= 60:
            self.__age = value
        else:
            # 创建异常  -- 抛出 错误信息
            raise Exception("我不要","if 20 <= value <= 60",1001)

# -- 接收 错误信息
while True:
    try:
        age = int(input("请输入你老婆年龄: "))
        w01 = Wife(age)
        break
    except Exception as e:
        print(e.args) # ('我不要', 'if 30 <= value <= 60', 1001)
```

1.8 异常传递

```python
def A():
    print(' 进入 A')
    1/0
    print(' 退出 A')
def B():
    print(' 进入 B')
    try:
        A() # 调用 A
    except Exception as e:
        print('捕捉到异常 in B, e:', e)
        raise # 再次将该异常抛出
    print(' 退出 B')
def C():
    print(' 进入 C')
    try:
        B() # 调用 B
    except Exception as e:
        print('捕捉到异常 in C, e:', e)
    print(' 退出 C')
C()
```

1.9 自定义异常

- 异常类可以用户自定义，只需要继承对应的异常类并重写相关信息

```
class InputShortException(Exception):

    def __init__(self, msg):
        super().__init__()
        self.__msg = msg # 存储异常信息

    def __str__(self):
        return self.__msg

def input_value(text, min_len):
    '''
        接收用户输入
    '''
    value = input(text)
    if len(value) < min_len:
        # 抛出异常
        error = "{} 至少 {} 长度，实际长度是{} 字符".format(text,
min_len,len(value))
        raise InputShortException(error)
    return value

print('程序启动')
print('录入用户信息')
try:
    name = input_value('姓名：', 2)
    tel = input_value('电话：', 11)
    addr = input_value('地址：', 4)
except InputShortException as e: # 异常提示
    print('错误提示：', e)
else: # 验证成功后执行
    print('您输入的信息是：')
    print('姓名：', name)
    print('电话：', tel)
    print('地址：', addr)
print('程序结束')
```

- 在触发错误的地方抛出异常
- 捕获异常并处理的方式和普通的系统异常处理方式一致