

Python基础 Day12 异常处理

异常概述

在程序运行过程中，经常会遇到各种各样的错误，这些错误统称“异常”。

下面这段代码将产生异常：

```
def division():
    num1 = int(input("请输入被除数: ")) #接收用户输入的数据，并转换为int类型
    num2 = int(input("请输入除数: "))
    result = num1 // num2 # 执行除法运算，且不要小数
    print(result)

if __name__ == "__main__":division() # 这个程序的入口，并调用division函数
```

正常情况下的输出为：

```
λ cd day12
F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo.py
请输入被除数: 4
请输入除数: 2
2
```

但是如果用户输入的内容不合法，将会产生异常：

```
λ python demo.py
请输入被除数: 4
请输入除数: 0
Traceback (most recent call last):
  File "demo.py", line 7, in <module>
    if __name__ == "__main__":division() # 这个程序的入口，并调用division函数
  File "demo.py", line 4, in division
    result = num1 // num2 # 执行除法运算，且不要小数
ZeroDivisionError: integer division or modulo by zero
```

产生这个错误的原因是：0作为了除数。但是对于一般用户来讲，可能并不清楚这是什么。因此，需要我们在编程时做好异常处理。尽量不要让用户看到如上的错误。而是以一种更好理解的形式来告诉用户。

异常处理语句

1. try ... except语句

在使用时，将可能出现问题的代码内容放在try语句块中。这样当try语句块的内容发生错误时，就会执行except中的内容了。具体语法：

```
try:
```

```
block1
except [ExceptionName [as alias]]:
    block2
```

block1:可能产生问题的代码块

[ExceptionName [as alias]]: 可选参数, 用于指定要捕获的异常。ExceptionName 为要捕获的异常的名称, 比如: ZeroDivisionError, [as alias], 则表示为该异常起一个别名, 方便后面使用。

block2: 当发生异常时要执行的代码块。

使用try ... except语句, 当程序出错时, 输出错误信息后, 程序会继续往下执行。

```
def division():
    num1 = int(input("请输入被除数: ")) #接收用户输入的数据, 并转换为int类型
    num2 = int(input("请输入除数: "))
    result = num1 // num2 # 执行除法运算, 且不要小数
    print(result)

if __name__ == "__main__":
    try:
        division() # 调用division()函数
    except ZeroDivisionError: # 捕获异常类型为ZeroDivisionError
        print("输入错误: 除数不能为0")
```

运行结果为:

```
λ python demo1.py
请输入被除数: 4
请输入除数: 2
2

F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo1.py
请输入被除数: 4
请输入除数: 0
输入错误: 除数不能为0
```

可以很好的看出来是发生了什么样的错误。

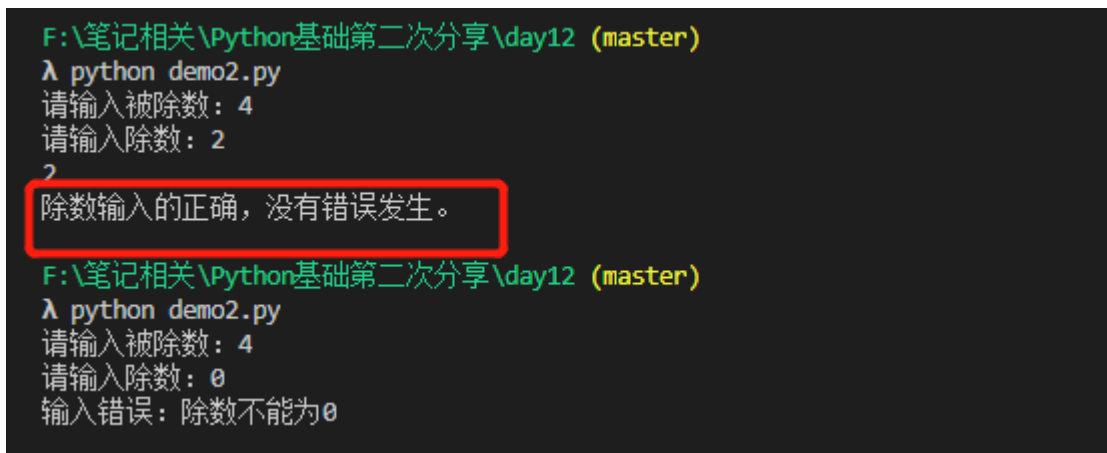
2. try ... except..else语句

当程序没有发生异常时, 我们也可通过这样的语句来获取信息。

```
def division():
    num1 = int(input("请输入被除数: ")) #接收用户输入的数据, 并转换位int类型
    num2 = int(input("请输入除数: "))
    result = num1 // num2 # 执行除法运算, 且不要小数
    print(result)

if __name__ == "__main__":
    try:
        division() # 调用division()函数
    except ZeroDivisionError: # 捕获异常类型为ZeroDivisionError
        print("输入错误: 除数不能为0")
    else:
        print("除数输入的正确, 没有错误发生。")
```

运行结果为:



```
F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo2.py
请输入被除数: 4
请输入除数: 2
2
除数输入的正确, 没有错误发生。
F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo2.py
请输入被除数: 4
请输入除数: 0
输入错误: 除数不能为0
```

3. try ... except...finally语句

finally语句的作用是, 无论程序有没有异常, 都会去执行finally下面的语句块。

```
def division():
    num1 = int(input("请输入被除数: ")) #接收用户输入的数据, 并转换位int类型
    num2 = int(input("请输入除数: "))
    result = num1 // num2 # 执行除法运算, 且不要小数
    print(result)

if __name__ == "__main__":
    try:
        division() # 调用division()函数
    except ZeroDivisionError: # 捕获异常类型为ZeroDivisionError
        print("输入错误: 除数不能为0")
    else:
        print("除数输入的正确, 没有错误发生。")
    finally:
        print("-----程序运行结束-----")
```

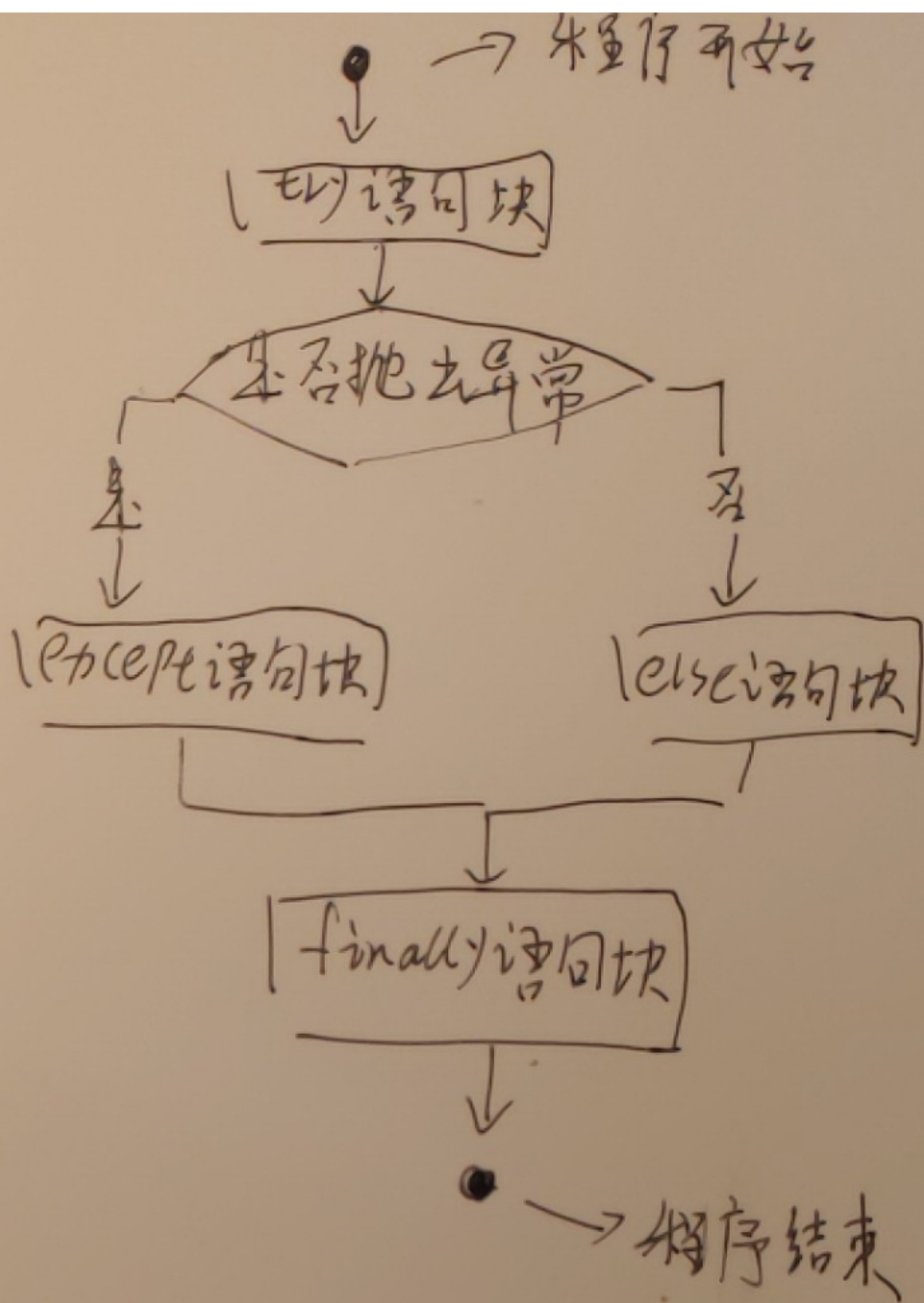
运行结果为:

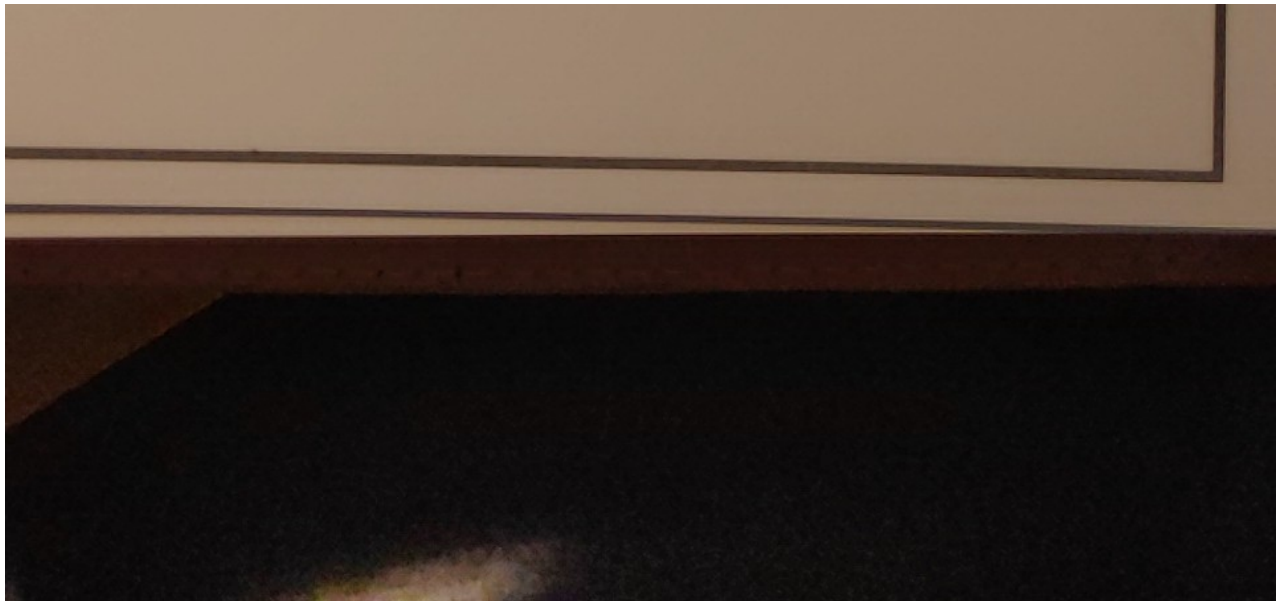
```
F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo3.py
请输入被除数: 4
请输入除数: 2
2
除数输入的正确, 没有错误发生。
-----程序运行结束-----

F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo3.py
请输入被除数: 4
请输入除数: 0
输入错误: 除数不能为0
-----程序运行结束-----
```

通用用于在程序运行结束后, 释放资源。比如open函数, 可以将close函数放在finally语句块中, 这样如论程序是否正常结束, 都会去关闭文件。

至此已经介绍了3中捕获异常的方式了, 我下面画一张图来展示各个语句的执行关系。(没有数位板, 只能在纸上画好, 上传上来。字丑, 请见谅。)





4. 使用raise语句抛出异常

如果某个函数或方法可能产生异常，但又不像在当前函数方法中处理这个异常，则可以使用raise语句在函数或方法中抛出一样

```
def division():
    num1 = int(input("请输入被除数: ")) #接收用户输入的数据，并转换位int类型
    num2 = int(input("请输入除数: "))
    if num2 == 0:
        raise ValueError("除数不能为0")
    result = num1 // num2 # 执行除法运算，且不要小数
    print(result)

if __name__ == "__main__":
    try:
        division() # 调用division()函数
    except ZeroDivisionError: # 捕获异常类型为ZeroDivisionError
        print("输入错误: 除数不能为0")
    except ValueError as e:
        print("输入错误:", e)
```

运行结果:

```
F:\笔记相关\Python基础第二次分享\day12 (master)
λ python demo4.py
请输入被除数: 4
请输入除数: 0
输入错误: 除数不能为0
```

raise ValueError("除数不能为0")

括号中的内容如果为空，则在抛出异常时，不附加任何异常描述信息