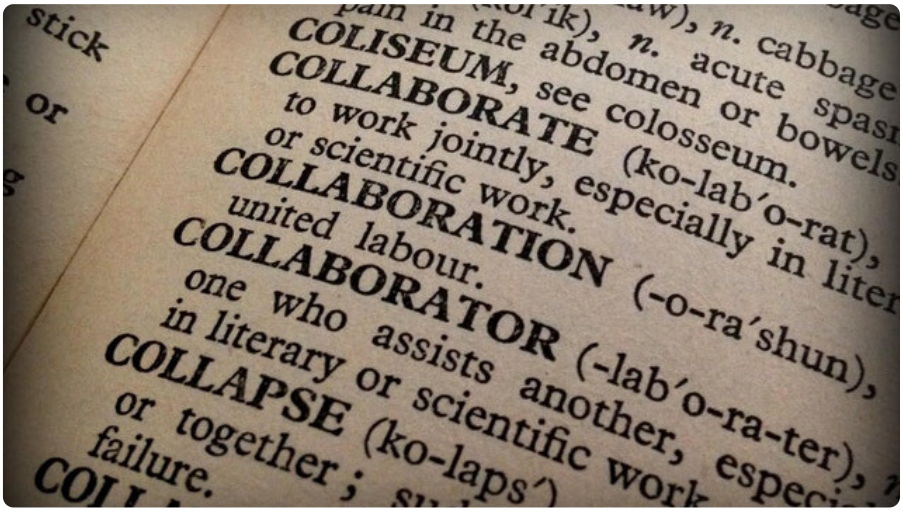
目录

09 知错能改—错误处理、异常机制

更新时间：2019-09-02 10:45:18



> 理想的书籍是智慧的钥匙。
>
> ——列夫·托尔斯泰

为什么需要错误处理

我们之前写的代码能够正常运行是建立在一个前提之下的，那就是假设所有的命令行输入或者函数参数都是正确无误的，并且执行过程中每个环节都是可靠和符合预期的。

当然，在程序的实际开发和使用过程中，这个前提是不能成立的，所有的假设都无法完全保证。比如：

- 用户与程序交互时输入不满足规则的内容。如，本应该输入年龄的地方输入了一个汉字，或者年龄的取值为负数，或者年龄远远超出人的正常寿命
- 函数或模块的使用者采用非预期的使用方式。如，函数期望的参数是整数型，结果传递了一个列表
- 程序外部的环境发生变化等。如：读取文件时，系统中不存在该文件；网络传输时，发生连接故障
- ……

这些错误发生在程序运行阶段，无法在编码阶段预知到它们是否会发生，但我们可以未雨绸缪，在代码中对潜在错误做出处理，以避免对程序运行造成破坏性影响。

> 说明：开发程序过程中还有一种常见的错误，就是开发者编写代码时的语法错误、编译错误以及运行时的 Bug。这些错误可以在开发时通过测试、调试、日志诊断等手段予以发现和解决，并不属于本章节所讲的错误处理机制的范畴。且**不能用错误处理机制来规避 Bug**。

首先错误发生的，需要先捕获到该错误，然后根据具体的错误内容或类型，选择后续处理的方式。

在 Python 中大多数情况下，错误是以抛出**异常**的形式报告出来。如列表的索引越界异常：

```
>>> fruit = ['apple', 'banana'][2]
Traceback (most recent call last):
     File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

上面提示发生了「IndexError」错误，这个 IndexError 就是异常的一种。在这里它直接被解释器捕捉到，然后将错误信息输出到了命令行中。

我们也可以自己来捕获异常，然后自定义处理方式。

try-except 语句捕获异常

异常的捕获使用 try-except 语句：

```
try:
    代码块1
except:
    代码块2
```

执行流程是，从 try 下的 代码块1 开始执行，若其中有异常抛出，那么异常将会被捕获，直接跳转并执行 except 下的 代码块2 。若 代码块1 一切正常，并没有异常抛出，那么 代码块2 将不会被执行。

也就是说 代码块1 是我们想要正常运行的代码，而 代码块2 是当错误发生时用于处理错误的代码。

来看一个使用 try-except 时发生异常的例子：

```
>>> try:
…     fruit = [‘apple’, ‘banana’][2]
…     print(fruit)
… except:
…     print(‘列表索引越界啦’)
…
列表索引越界啦
```

这里的执行流程是，执行 try 下的 ['apple', 'banana'][2] ，此时由于索引越界而产生异常，代码 print(fruit) 将被跳过，转而执行 except 下的 print('列表索引越界啦') 。

再来看一个无异常的例子：

```
>>> try:
…     fruit = [‘apple’, ‘banana’, ‘cherry’][2]
```

```
...     print('列表索引越界啦')
...
cherry
```

可以看到无异常抛出时， try 下的代码被全部执行， except 下的代码不会被执行。

捕获指定的异常

之前我们没有直接指定要捕获的异常类型，所以所有类型的异常都会被捕获。

我们也可以显式地指定要捕获的异常种类。方法是：

```
try:
    代码块1
except 异常X as e:
    代码块2
```

和之前的区别在于，多出了 异常X as e 这一部分。异常X 是指定的要捕获的异常名，如 IndexError 、 NameError 。 as e 语句是将异常对象赋予变量 e ，这样 e 就可以在 代码块2 中使用了，如获取错误信息。

如下是捕获指定异常的例子：

```
>>> try:
...     fruit = ['apple', 'banana'][2]
... except IndexError as e:
...     print('出现索引越界错误：', e)
...
出现索引越界错误： list index out of range
```

这里我们显式地指定要捕获 IndexError 异常，并且将异常中的错误信息输出出来。

显式指定异常时，只有被指定的异常会被捕获，其余异常将会被忽略。

捕获指定的多个异常

上面是指定并捕获一个异常，当然也可以在一个 try 语句下指定并捕获多个异常。有两种方式：

```
try:
    代码块1
except (异常X, 异常Y, 异常Z) as e:
    代码块2
```

```
try:
    代码块1
except 异常X as e:
    代码块2
```

```
    代码块4
```

如上，第一种方式是将多个异常放在一个 except 下处理，第二种方式将多个异常分别放在不同的 except 下处理。无论用哪种方式，异常抛出时，Python 会根据异常类型去匹配对应的 except 语句，然后执行其中代码块，若异常类型未能匹配到，则异常会继续抛出。那么这两种方式有什么区别呢？

- 第一种方式适用于多种异常可用相同代码进行处理的情况。
- 第二种情况适用于每个异常需要用不同代码进行处理的情况。

try-except-finally 语句

在之前介绍的 try-except 语句之后，还可以紧跟 finall 语句，如下：

```
try:
    代码块1
except 异常X as e:
    代码块2
finally:
    代码块3
```

它的执行流程是，

1. 首先执行 代码块1
2. 若发生异常则执行 代码块2 ，否则跳过 代码块2
3. 无论是否发生异常都执行 代码块3

也就是说在 try-except 执行流程的基础上，紧接着执行 finally 下的代码块，且 finally 下的代码必定会被执行。

finally 有什么用？举个例子，我们有时会在 try 下使用一些资源（比如文件、网络连接），而无论过程中是否有异常产生，我们在最后都应该释放（归还）掉这些资源，这时就可以将释放资源的代码放在 finally 语句下。

常见的异常类型

下表中是 Python 常见的内置异常：

异常名	含义
Exception	大多数异常的基类
SyntaxError	无效语法
NameError	名字（变量、函数、类等）不存在
ValueError	不合适的值
IndexError	索引超过范围
ImportError	模块不存在
IOError	I/O 相关错误
TypeError	不合适的类型

KeyError	字典的键值不存在
ZeroDivisionError	除法中被除数为 0

除此之外内置异常还有很多，待日后慢慢积累掌握。

raise 语句主动抛出异常

之前的示例中，异常是在程序遇到错误无法继续执行时，由解释器所抛出，我们也可以选择自己主动抛出异常。

主动抛出异常的方法是使用 raise 语句：

```
raise ValueError()
```

也可以同时指明错误原因：

```
raise ValueError("输入值不符合要求")
```

我们用示例来学习为什么要主动抛出异常，以及如何主动抛出异常。

之前我们在学习函数的时候写过这样一个函数：

```
def stage_of_life(age):
    if age <= 6:
        return '童年'
    elif 7 <= age <=17:
        return '少年'
    elif 18 <= age <= 40:
        return '青年'
    elif 41 <= age <= 65:
        return '中年'
    else:
        return '老年'
```

显然这个函数没有应对可能出错的情况。比如函数的 age 参数不能任意取值，要符合人类的年龄范围才行，如果取值超出范围就需要向函数调用方报告错误，这时就可以采取主动抛出异常的方式。

我们在函数内检验输入值的有效性，若输入有误则向外抛出异常，新增第 2 和第 3 行代码：

```
def stage_of_life(age):
    if age < 0 or age > 150:
        raise ValueError("年龄的取值不符合实际，需要在 0 到 150 之间")

    if age <= 6:
        return '童年'
    elif 7 <= age <=17:
        return '少年'
    elif 18 <= age <= 40:
        return '青年'
    elif 41 <= age <= 65:
        return '中年'
```

这里检查 age 的范围是否在 0～150 之间，若不是则使用 raise 抛出 ValueError 异常，表示取值错误。

用不为 0——150 的数字执行下函数看看：

```
>>> stage_of_life(-11)
Traceback (most recent call last):
  File “ ”, line 1, in
  File “ ”, line 3, in stage_of_life
ValueError: 年龄的取值不符合实际，需要在 0 到 150 之间

>>> stage_of_life(160)
Traceback (most recent call last):
  File “ ”, line 1, in
  File “ ”, line 3, in stage_of_life
ValueError: 年龄的取值不符合实际，需要在 0 到 150 之间
```

总结

在 Python 中大多数情况下，错误是以抛出**异常**的方式报告出来，可以针对潜在的异常来编写处理代码。

可使用 try-except 语句捕获异常

异常的捕获使用 try-except 语句：

```
try:
    代码块1
except 异常X as e:
    代码块2
```

捕获多个异常：

```
try:
    代码块1
except (异常X, 异常Y, 异常Z) as e:
    代码块2
```

```
try:
    代码块1
except 异常X as e:
    代码块2
except 异常Y as e:
    代码块3
except 异常Z as e:
    代码块4
```

finally 语句紧接着 try-except 的流程执行：

```
except 异常X as e:
    代码块2
finally:
    代码块3
```

使用 raise 语句可主动抛出异常：

```
raise ValueError()
```

← 08 将代码放进盒子—函数　　10 定制一个模子—类 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

singvis

如果异常太多，是不是可以这么写:
try:
代码块1
except:
代码块2

0 回复 2019-09-03

黄浮云 回复 **singvis**

这种写法是捕获所有的异常。try 下代码块可能抛出多种异常时，不建议用这种方式，因为不加区分地捕获所有异常，一方面不太好做针对性的处理；另一方面这可能会捕获到未预期的异常，并且错过对它的处理，未预期的异常应尽早暴露以便尽早修复。

回复 2019-09-16 21:07:28

singvis 回复 **singvis**

异常太多那如何处理呢？或者有更好的写法？

回复 2019-09-16 21:09:52

Spider007 回复 **singvis**

对于异常，处理方法就是罗列所有可能的异常，并一一处理。这样子才能提高用户的体验。而不是经常程序报错。

回复 2019-11-10 11:34:04

千学不如一看，千看不如一练