第十章：错误与异常处理

Table of Contents

[1. 序言 2](#_Toc51539188)

[1.1 章节简介 2](#_Toc51539189)

[2. 什么是错误和错误处理机制 2](#_Toc51539190)

[3. try…except…finally 2](#_Toc51539191)

[4. 调用栈 5](#_Toc51539192)

[5. 记录错误 6](#_Toc51539193)

[6. raise error 7](#_Toc51539194)

# 序言

## 章节简介

本章介绍Python在错误处理方面的机制，主要有以下一些内容：

* 什么是错误和错误处理机制
* try…except…finally
* 调用栈
* 记录错误
* raise error

# 什么是错误和错误处理机制

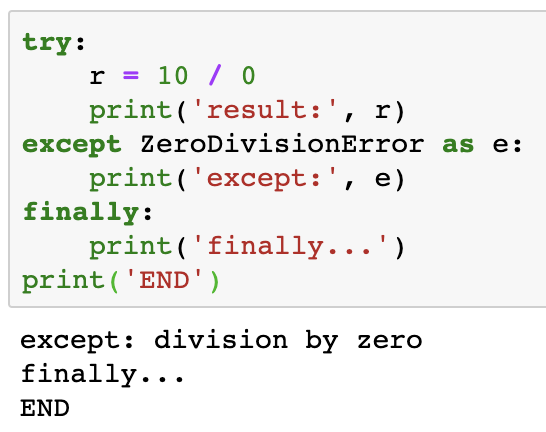
我们先来了解什么是错误(error)。错误即是一个事件，该事件会在程序执行过程中发生，影响了程序的正常执行。一般情况下，在Python无法正常处理程序时就会发生一个错误。错误是Python对象，表示一个错误。当Python脚本发生错误时我们需要捕获处理它，否则程序会终止执行。

跟其他高级语言类似，捕捉错误可以使用try/except语句。try/except语句用来检测try语句块中的错误，从而让except语句捕获异常信息并处理。如果你不想在异常发生时结束你的程序，只需在try里捕获它。

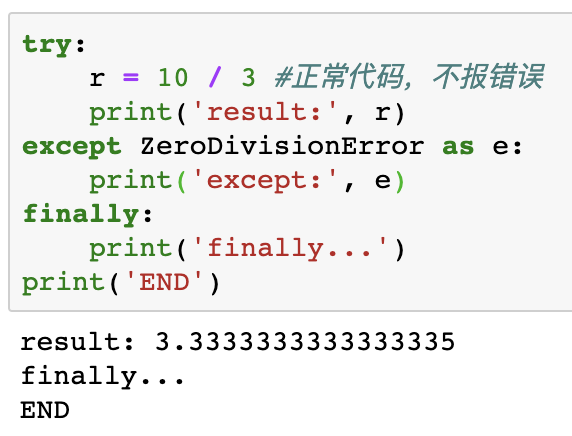
# try…except…finally

在Python中我们使用try…except…else…finally语句来处理程序运行时发生的错误。把可能发生错误的语句放在try模块里，用except来处理异常。except可以处理一个专门的异常，也可以处理一组圆括号中的异常，如果except后没有指定异常，则默认处理所有的异常。每一个try，都必须至少有一个except。finally块让你可以执行最终代码，不管try与except块的结果如何，finally块的代码都将执行。

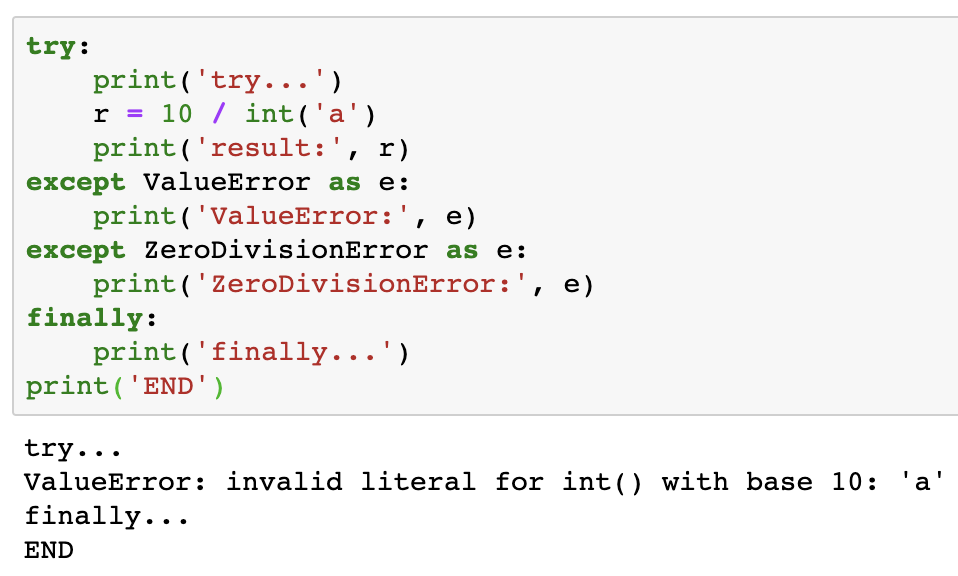
我们先来看一个使用try的示例。在我们的程序中如果有些代码可能会发生错误，就可以用try语句来包含这段代码，如果try中的代码在运行过程中报错，则代码不会再执行，而是直接跳转至错误处理代码，即except语句块，执行完except后，如果有finally语句块，则执行finally语句块，至此，执行完毕。下面的代码在计算10 / 0时会产生一个除法运算错误：从输出可以看到，当错误发生时，后续语句print('result:', r)不会被执行，except由于捕获到ZeroDivisionError，因此被执行。最后，finally语句被执行。这就是整个语句块中代码的流程：



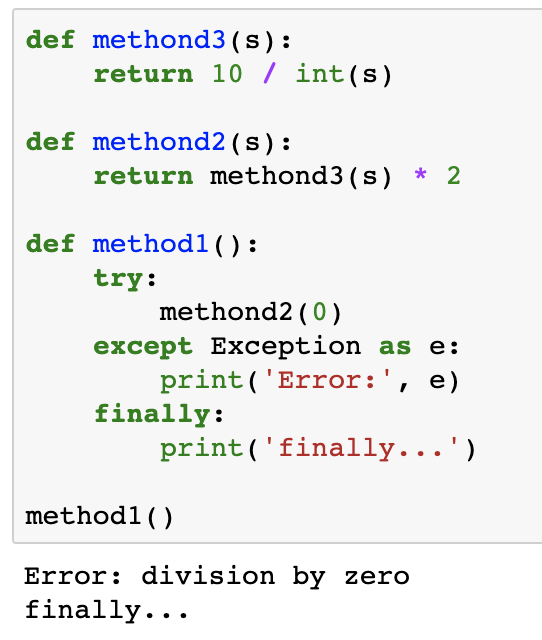
如果程序正常执行不报错，那么输出结果如下。即except语句块不会被执行，但是finally如果有，则一定会被执行（可以没有finally语句，但如果有finally，不管程序有没有Error，Finally中的代码都会被执行）。



Python中内置错误类型有很多种类，我们可以在try...except...finally代码块中定义多个except语句块。对于try中可能抛出的不同类型的错误，应该由不同的except语句块处理。在下面的示例中，int()函数可能会抛出ValueError，所以我们用一个except捕获ValueError，用另一个except捕获ZeroDivisionError：



使用try...except还有一个优点，就是可以在代码的外层捕获错误，比如在下面的示例中，methond1方法调用method2方法，而methond2方法调用methond3方法。当我们的代码里调用method1时，只要把methond1()的调用放在try语句块中，不管method1里如何实现调用了多少其他变量和方法，所有的可能的代码抛出 的错误都会被捕获。



# 调用栈

当我们的程序没有捕获错误并进行处理时，错误会被Python识别并将错误信息输出并终止和退出程序，我们来看看输出的错误信息的trackback。

下面这段程序仍然是抛出了一个ZeroDivisionError，除了红色的显示错误类别的信息。还有一段trackback的信息。首先trackback告诉我们在当前的模块里(可以当做我们的python脚本)发生了错误，发生错误的代码位于文件的第十行，也就是调用method1()处。我们再往下看，trackback里的信息的第二段可以看到method1()中，虚线箭头的指向代码的第8行，这里告诉我们实际是程序的第八行报错，也就是method1()代码中调用method2()的地方地方。下面的信息以此类推，直到trackback里的最后一段，绿色虚线告诉我们就是在return 10/int(s)的一行发生了错误。而这一行代码中，因为除数是0而报错。我们成功的根据trackback里的信息跟踪了代码的调用层级，并发现了报错的根本原因，在平时的工作中，我们要学会查看trackback跟踪错误发生的代码处。

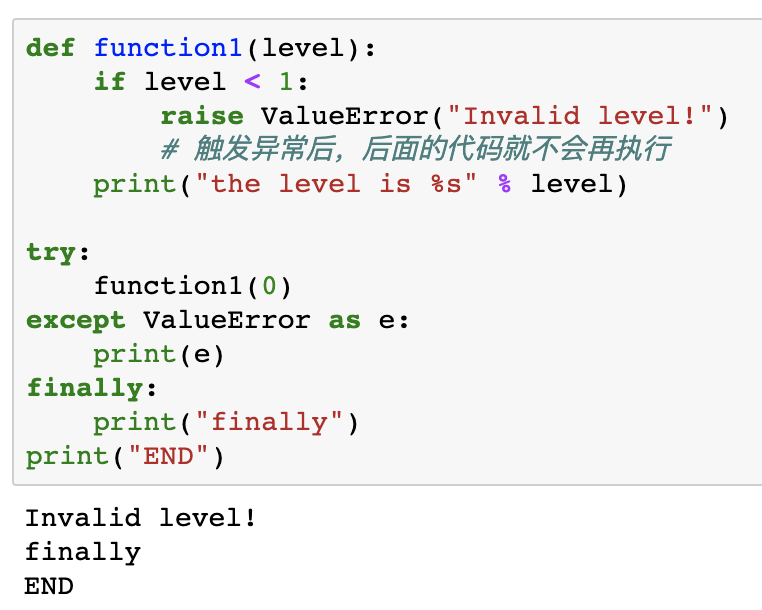


# 记录错误

我们在得到错误对象时，有时需要记录Error的信息及trackback，这中场合下我们会使用Python内置的logging模块，将程序抛出的error和trackback记录到日志文件中，方便我们时候进行跟踪寻找代码的错误根源。

# raise error

最后我们再来看错误处理中的raise。我们可以使用raise语句自己触发异常，当程序出现错误，python会自动抛出错误，也可以通过raise显示地抛出错误。一旦执行了raise语句，raise后面的语句将不能执行。可能有人会问代码发现错误时不是应该马上处理吗，为什么要用raise抛出去？这是因为捕获错误目的只是记录一下，便于后续追踪。但是，由于当前函数不知道应该怎么处理该错误，所以，最恰当的方式是继续往上抛，让顶层调用者去处理。而上层调用者会根据错误类型和trackback，决定如何处理错误。这是实际编程中常见的处理方式。



上面的代码里，function1()发现参数小于1后，主动抛出一个ValueError，同时function1()的执行中止，返回到上一层即function1()的调用者，调用者在try语句块中捕获和这个error并进行祥一行的处理。