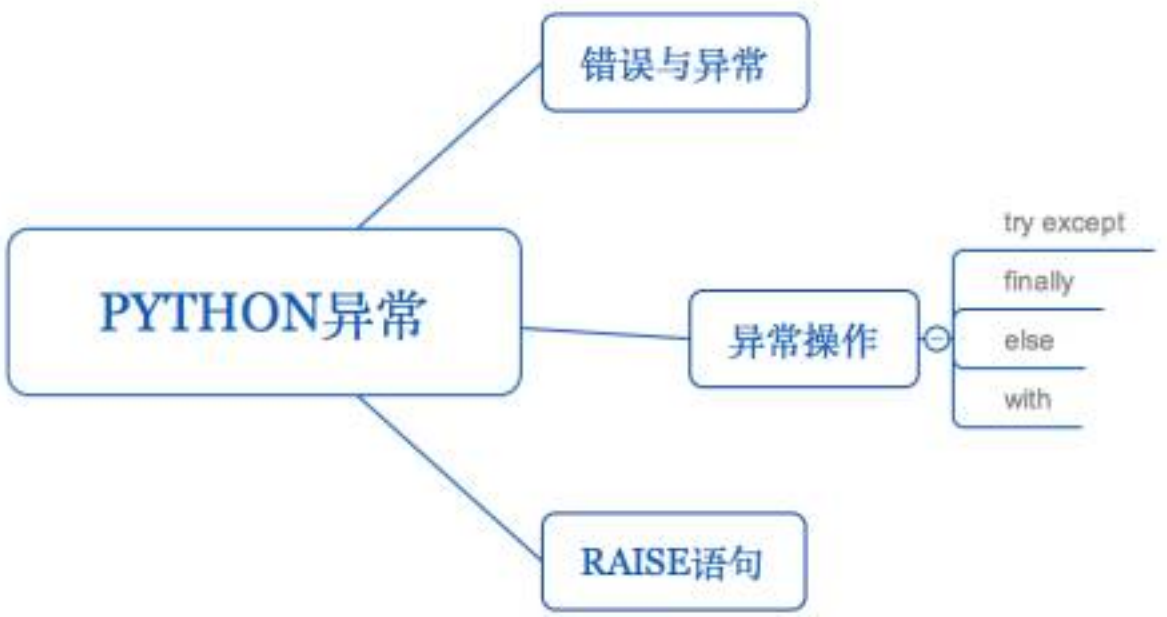
Python异常－人非圣贤，谁还不犯几个错?

Original 2016-12-11 哈哈小 菜猿观世界

通过本节的学习，你将了解以下内容：



说到异常，大家可能最先想到的是bug。在这里需要声明一下，bug与异常，两者之间还是有一定区别的。

bug就跟个小虫一样，躲在系统的某处，不被发现还好，发现了可就值钱了。

所以bug一般强调难以预见的错误，一种潜在的漏洞。

而异常，就不一样了，异常指的是程序在运行时出现的问题导致程序停止运行的现象。

相比较bug来讲，异常好像就在说：哥们，你程序这有问题，快去处理处理，处理不了，别想过我这关。

听起来还挺严重的。

对于面向对象的编程语言，几乎都有对异常的处理。

接下来我们就来具体看看python对异常怎么个玩法。

在进入异常的讲解之前，先来了解一下语法错误（syntax error）

语法错误，又叫做解释错误，对于python 而言，如果发现程序中有存在这种错误，那绝对不能忍，编译都不会让你通过，就更别说执行了。

这种错误就比较低级了，因为低级，所以最不应该犯，否者等它脸色大变，你就不堪忍受了。

这种错误低级到，好比哈哈小在写某篇原创时，手抽筋打错的几个字或一不留神的语句混乱。

1.语法错误

语法错误，大家可能在哈哈小的原创中鲜少见到，因为对于这种低级错误，平时没事，哈哈小实在不应该拿出来给大家看啊。

但是哈哈小相信，你们在编写程序时，一定有遇到。

如果大家用的python的版本是3.x，下面这个错误应该经常见到吧：

```
>>> while True: print 'Hello world'
SyntaxError: invalid syntax
>>>
```

由于print不加括号直接输出字符串这种语法仅于python3.x以下的版本支持，所以，如果你试图在python3.x的版本中使用这种语法，python理解不了你的行为，于是python编译器会很听话地给你报这么一个错，而且还会给你脸色看（是不是看到了具有警示意义的红色？）。

```
SyntaxError: invalid syntax
```

这种格式跟我们之前谈到的字典是不是很是相似？ 错误以健值对的形式抛出，分号前面表示错误的类型，后面表示具体的错误内容。

在这里，这句话的意思是：你有个语法错误，是犯了不合理的语法这么一个错。

好吧，我承认这个错报的有点尴尬。

Anyway, 进入今天核心part－异常

2.异常（exceptions）

毋庸置疑，不管是上文提到的漏洞，语法错误，还是这里马上要讲到的异常，统统都称错误。这是一个好比今天就是今天，绝不可能是明天一样的事实吧。

嗯，是这样的。

官话有言：在程序执行时检测的错误叫异常，并且这种错误是很致命的（*Errors detected during execution are called exceptions and are not unconditionally fatal*）

说那么多，没个例子呈上，我还不如左手握手打发时间。

别急，例子说来就来：

```
>>> 1/0
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    1/0
ZeroDivisionError: division by zero

>>> f=open("exception.txt")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    f=open("exception.txt")
FileNotFoundError: [Errno 2] No such file or directory: 'exception.txt'
```

```
>>> '2'+2
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    '2'+2
TypeError: Can't convert 'int' object to str implicitly

>>>  3*x
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    3*x
NameError: name 'x' is not defined
>>>
```

比较一下语法错误，异常做事情好像要靠谱一些，多给了点信息：

```
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    1/0
```

通过这部分的信息，能让我们知道出现异常的是哪个文件File "<pyshell#2>"，位于该文件的哪一行line 1，以及这个异常是哪个模块抛出来的in <module>1/0。

上面几个例子是我们在python编程时，常能看到的错误。

第一个是除零异常－小明同学应该都知道这点吧。

第二个是文件未发现异常－哈哈小没创建文件就直接去打开了。

第三是类型转换异常－字符串类型的'2'和整型的2相加，python傻的分不清。

第四个是变量为声明异常－我发誓，在这之前我没对x定义过。

以上列举的错误，仅仅只是python异常处理系统极少的一部分。哈哈小只是小试引导，大家在今后的学习过程中，自会日益增进。

感兴趣的同学可以入官方网站了解更多。

这里附上pyhton异常层次结构图：

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

（截图自：https://docs.python.org/3/library/exceptions.html#bltin-exceptions）

别看图了，看这里：

附这张图，一来是让大家知道，程序能引发的异常是异常庞大的，就好比我们生活中能犯的错误也是各种各样，各式奇葩的。

二来是，python对异常的操作是分层分类结构化管理的。

3.处理异常

有发现，上文出现的异常都是python自动给我们处理好的，抛出的异常信息也是python定义好的。

如果我们要张扬个性，对异常自定义，输出一些自己stye的异常信息，那该怎么做呢?

try except语句就是专门来做这种事的。

3.1 try except语句

以上文的第一个例子为讲解：

```
>>> try:
        6/0
        print("haha")
except ZeroDivisionError:
        print("嘿，哥们，你拿个数除以0，不带这么玩的")

嘿，哥们，你拿个数除以0，不带这么玩的
>>>
```

try except语句基本语法就如上图所示。

try语句块内容为监测范围，except语句后面接要对异常处理的代码。

except后面除了必写参数Exception（要处理异常的类型）外，还有一个可选参数reason（这个名字自己随便给，不一定叫reason）。

如果在本例中，ZeroDivisionError后面接上as resaon，然后再用print讲reason打印出来，其结果就跟python自处理的结果没什么区别了。

```
>>> try:
        6/0
        print("haha")
except ZeroDivisionError as reason:
        print(reason)

division by zero
>>>
```

其实reason的内容正是python自处理的内容。

有发现，在6/0后面一句print（"haha"） 好像并没有被打印出来。

这是因为，一旦在try中发现异常，异常抛出，被excerpt捕获到，那么就算try中还有未执行的代码，都将不再被执行。

如果try语句下的监测范围存在多种异常的可能，我们需要捕获多个异常，可以将except语句改造如下：

```
except (A,B,C):
    #...
```

```
>>> try:
        6/0
        f=open("exception.txt")
        result='2'+2
        print("haha")
except (ZeroDivisionError,FileNotFoundError,TypeError) as reason:
        print(reason)
```

或

```
except (A):
    #....
except (B):
    #...
except (C):
    #...
...
```

```
>>> try:
        6/0
        f=open("exception.txt")
        result='2'+2
        print("haha")
```

```
except ZeroDivisionError as reason:
    print(reason)
except FileNotFoundError as reason:
    print(reason)
except TypeError as reason:
    print(reason)
```

3.2 finally语句

哈哈小前面讲过，如果try监测内容某一句发生错误，异常抛出，except捕获，后面的代码不再执行。

设想这么一种情况

```
f=open("finallyDemo.txt","w")
f.write("this is a test")
3*i
f.close()
```

如果检测内容如上所示，当程序执行到倒数第二行，会抛一个变量未定义异常，所以最后一行文件关闭代码close将不再执行。

哈哈小强调过，文件要随开随关，所以这怎么了得?

没关系，finally语句就是专门来解决这一类问题的，很多其它面相对象的语言例如java也是采用finally语句进行处理的。

我们来看看finally语句怎么使?

```
>>> try:
        f=open("finallyDemo.txt","w")
        f.write("this is a test")
        3*i
except (OSError,NameError):
```

```
        print("error exists")
finally:
        f.close()


14
error exists
```

解决上述问题，只需把close代码放到finally语句块下就可以了，finally的做的是收尾工作（clean－up actions），最后的工作，必做的工作。

就好比某些人不管这一年下来，是事事平安，还是祸事连三，年底都要去做的一件事－烧香拜佛一样。

3.3 with语句

在python中，还有一种特殊的收尾工作处理方式，就是利用with语句，使用如下：

```
>>> with open("withDemo.txt") as f:
    for line in f:
        print(line,end="")
```

python既然是一门脚本语言，看这意思也不难理解吧。

with open("withDemo.txt") as f 在告诉我们这么一件事：

打开withDemo.txt文件，如果没什么错误就把值赋给f变量，如果有错误，那抱歉了，f，我不能给你这个文件的指引了。

像with这种工作在python中叫做预定义收尾工作（predefined clea-up actions）。

3.4 else语句

else语句，这个相比大家再熟悉不过了，不管是循环还是分支，它都能派上用场。

不过今天讲的内容是异常，实在想不通这货在这出现是干嘛来了。

这时else可能就不高兴了，底气十足厉声说道：我的用途可大了去了，让你们见识见识我在异常处理中的用途：

```
>>>try:
        result=6/1
        print("haha")
except ZeroDivisionError as reason:
        print(reason)
else:
        print("no error, haha")
```

看到没有，没有什么错误发生时，你还有什么要执行的代码，尽管找我就是了。

4. raise语句

哈哈小在上文多次提到抛出，捕获两字。

这两字，请大家记住，既有画面感又助攻下文所讲。

学过java的同学，应该立马能想到，这里的raise是与java的throw对应的。

raise用法很简单：

rasie＋要丢出异常的类型

我随手一丢一异常

```
>>> raise FileNotFoundError
Traceback (most recent call last):
  File "<pyshell#53>", line 1, in <module>
    raise FileNotFoundError
```

```
FileNotFoundError
```

再丢把异常成双对

```
>>>raise TypeError
Traceback (most recent call last):
  File "<pyshell#54>", line 1, in <module>
    raise TypeError
TypeError
>>>
```

讲到此处，大家稍作思考，就会得到一个来之不易的“原来如此”

其实哈哈小之前给的例子，像6/0，2*i 等，不难发现，其实它们就等同分别执行了raise TypeError和 raise NameError两句代码。

文章结尾，哈哈小最后补充一句：人非圣贤，谁还不犯几个错？犯错不要紧，最重要的是，一定要把所有可能出现的错误一并收拢，全全捕获，事后处理。

下节预告：从下节开始，我们开始接触面向对象编程的概念。

结束语：我们不做知识的搬运工，幸能成为思想的启迪者。