异常处理与调试

错误

- 有的错误是程序编写有问题造成的,比如本来应该输出整数结果输出了字符串,这种错误我们通常称之为 bug,bug 是必须修复的。
- 有的错误是用户输入造成的,比如让用户输入 email 地址,结果得到一个空字符串,这种错误可以通过检查用户输入来做相应的处理。
- 还有一类错误是完全无法在程序运行过程中预测的,比如写入文件的时候,磁盘满了,写不进去了,这类错误也称为异常,在程序中通常是必须处理的,否则,程序会因为各种问题终止并退出。

错误处理

- 在程序运行的过程中,如果发生了错误,可以事先约定返回一个错误代码;
- Python语言通常都内置了一套 try...except...finally... 的错误处理机制

错误处理

```
try:
        print 'try...'
        r = 10 / 0
        print 'result:', r
except ZeroDivisionError, e:
        print 'except:', efinally:
        print 'finally...'
        print 'END'
```

错误处理

- 用 try 来运行可能会出错的代码;
- 如果执行正确,则except 语句块不会执行;
- 如果执行错误,直接跳转至错误处理代码，即except语句块;
- 如果有 finally 语句块,不管try语句块内容是否正确，都会执行 finally 语句块

错误处理

- 错误有很多种类,如果发生了不同类型的错误,应该由不同的 except语句块处理。因此可以有多个 except 来捕获不同类型的错误。

```
eroDivisionError:', e
            print 'ValueError:', e
except ZeroDivisionError, e:
            print 'ZeroDivisionError:', e
```

错误处理

- Python 的错误其实也是 class,所有的错误类型都继承自BaseException;
- 在使用except 捕获该类型的错误,还把其子类也“一网打尽”;
- 常见的错误类型和继承关系看这里:

https://docs.python.org/2/library/exceptions.html#exception-hierarchy

读懂复杂的错误

- 解读错误信息是定位错误的关键。我们从上往下可以看到整个错误的调用函数链。

```
def foo(s):
        return 10 / int(s)
def bar(s):
        return foo(s) * 2
def main():
        bar('0')
main()
```

记录错误

- 不捕获错误,Python 解释器会打印出错误信息,但程序也被结束;
- 捕获错误,就可以把错误信息打印出来,然后分析错误原因,同时,让程序继续执行下去。
- Python 内置的 logging 模块可以记录错误信息。

```
logging.exception(e)
```

抛出错误

- 错误是 class,捕获一个错误就是捕获到该 class 的一个实例;
- Python 的内置函数会抛出很多类型的错误,我们自己编写的函数也可以抛出错误。
- 可以定义一个错误的 class,选择好继承关系,然后,用raise 语句抛出一个错误的实例;
- 尽量使用 Python 内置的错误类型

抛出错误

```
class FooError(StandardError):
        pass
def foo(s):
        n = int(s)
        if n==0:
                raise FooError('invalid value: %s' % s)
        return 10 / n
```

调试- print

第一种方法简单直接粗暴有效,就是用 print 把可能有问题的变量打印出来看看。用 print 最大的坏处是将来还得删掉它,运行结果也会包含很多垃圾信息。

调试- 断言

- 凡是用 print 来辅助查看的地方,都可以用断言(assert)来替代:;
- 如果断言失败, assert 语句本身就会抛出 AssertionError

```
assert n!=0

assert hello() = "hello"
```

- Python 解释器执行时可以用 -O 参数来关闭 assert，把所有的 assert 语句当成 pass。

调试- logging

- logging 不会抛出错误,而且可以输出到文件;
- logging.info() 就可以输出一段文本到日志文件中。
- logging.basicConfig(level=logging.INFO)指定记录信息的级别,有

debug , info , warning , error等几个级别。

调试- logging

```
import logging

s = '0'

n = int(s)

logging.basicConfig(filename="/home/kiosk/hello.log

",level=logging.INFO)

logging.info('n=%d' %n)

print 10/n
```

调试- pdb

- pdb

  pdb让程序以单步方式运行,随时查看运行状态。n 可以单步执行代码,p 变量名 来查看变量，q 结束调试,退出程序。

- pdb.set_trace

  在可能出错的地方放一个 pdb.set_trace() ,就可以设置一个断点。程序会自动在 pdb.set_trace() 暂停并进入 pdb 调试环境, p 查看变量, c 继续运行。

over !