LDAT 自动化测试系统

项目: Linux Desktop Auto-test Tool 基于 LDTP 等工具的自动化测试系统

文档 ID: LDAT-python exception

文档类型: 开源项目文档

版本: ver1.0

作者: Vans

日期: 2015.09.28

是否被检查: 是/否

是否已提交: 是/否

密级: 无

关键词: Python, Exception,Errors

目录

Python Exception.....3
本文背景:.....3
Python Exceptions 介绍.....3
Built-In Exception --内置异常.....3
Python 异常的处理.....6
1.捕捉异常.....6
2.Try 语句工作原理.....6
3.检测所有异常.....7
4. try-finally 语句.....7
5. 异常的参数.....7
7.用户自定义异常.....8
附录：.....8

Python Exception

本文背景:

鉴于 LOONGSON(龙芯中科) ldat 项目作为开源的项目进行推广，在使用 python 进行 case 脚本编写时，必然会遇到 python exception 的触发和处理，在此对 Python Exception 进行简单整理，方便 python 新入者能够正确使用 Python exception 机制。

Python Exceptions 介绍

Python 的异常是指在 python 程序执行之间遇到的错误。它通常不是致命的，大多数异常都不是由程序来处理的，但是依然会显示错误信息。当一个内置的异常发生时，异常的名称就会被打印出来，这条规则对所有的 Python 内置异常都是非常有用的。标准异常名称是内置的标识符，而不是保留关键字。一般来说屏幕上输出的异常包含一个堆栈跟踪信息，但是它不会显示从标准输入中读取的行。

Built-In Exception --内置异常

异常应该是类的对象，异常定义在 exceptions 模块里。此模块从不需要显式地导入，异常和异常模块在内置命名空间被提供。对于异常类，在 try 语句和 except 子句之间，提到一个特殊的类，这个 except 从句同时也处理由此异常类派生的任何异常类，而不会去处理该异常类的父类或基类。

未通过子类相关的两个异常类从不等价，即使它们有相同的额名称。除非提到它们有相一个关联值，表示错误原因的详细原因。这个相关联的值，是 raise()函数的第 2 个参数。如果异常类是从标准的根类 BaseException 派生而来，相关联的值作为异常实例的的 args 属性。

用户代码可以引发内置异常，这可以用来测试异常处理程序，或者报告一个错误的状况。内置异常类可以作为自定义异常类的父类，鼓励程序员从 Exception 或者它的子类中派生用于自定义的异常类，而不是从 BaseException 进派生。

下面列出常见的 python 异常类:

(1)BaseException

它是多有内置异常类的基类，但是用于定义的异常类并不直接从此异常类继承，通常情况下用户自定义的异常类可以 Exception 继承。如果对 BaseException 类的实例调用 str()或者 unicode(),这个实例的参数的描述信息，将被返回，或者在没有参数的情况下返回一个空字符串。此异常类的参数是一个元组，此参数供该类的构造函数使用，一些内置的异常类(例如: IOError) 期待一定数量的参数和分配一个特殊意义的的元组元素，然而有些异常类，仅仅使用一个简单的字符串来输出错误信息。

(2)Exception

所有内置，non-system-exiting 的异常都派生于此类，所有用户自定义的异常类也派生于此类。

(3)StandargError

是除了 StopIteration, GeneratorExit, KeyboardInterrupt and SystemExit 之外的所有类的基类，此类本省派生于 Exception。

(4)ArithmeticError

此类是那些内置的各种算术错误的基类，例如 OverflowError， ZeroDivisionError，FloationgPointError.

(5)BufferError

当一个 buffer 相关的操作没有被执行时就会触发此异常。

(6)LookupError

当一个键或索引在一个映射或者序列上使用无效时，就会出发此异常，并且它是 IndexError ，KeyError 类的基类。可以直接通过 codecs.lookup() 来触发此异常类。

(7)EnvironmentError

此基类是那些发生在 Python 系统之外的异常:IOError,OSError. 这种异常类通常在创建的时候，带有一个 2 元素的元组，元组的第一个元素是有关实例的错误号属性(它被认为是一个错误号)，元组的第 2 个元素是有关标准错误的属性(与错误的信息有关。)

(8)AsserttionError

当一个 assert 语句执行失败是触发此异常。

(9)AttributeError

当一个属性引用或者一个 assignment 失败时，引发此异常。当一个对象不支持属性引用或者属性分配时，TypeError 将会引发异常。

(10)EOFError

当一个内置的函数(input(), 或者 raw_input()) 遇到一个文件结束条件，没有读取任何数据。File.read() file.readline()方法在遇到 EOF 时，会返回一个空字符串。

(11)FloatingPointError

当一个浮点操作失败时，将会引发此异常。

(12)GeneratorExit

当一个 generator 调用 close 方法时将会引发此异常。此异常直接继承自 BaseException，而不是 StandardError。

(13)IOError

当一个 IO 操作（例如一个 print 语句， 或者内置的 built-in open() 函数，或则一个文件对象的方法调用失败时）一个 IO 相关的原因例如”file not found” or “disk full” 引发的异常。此类继承自 EnvironmentError 在 python 2.6 之后此类作为 sock.error 的基类。

(14)NameError

当一个局部或者全局的名称没有找到时，引发此异常。这仅限于非限定名称，关联值时包括不能被找到的名称的错误信息。

(15)RuntimeError

当检测到的错误不属于任何其他类别的范畴，相关的值是一个字符串，用来指出错误在哪里。

(16)SystemExit

这个异常由 sys.exit()函数引发，当它们没有被处理时，python 解释器退出。与此异常相关联的是一个整型值。用来指定 system exit 的退出状态。如实值为 None，退出状态为 0. 如果此值是其他的类型，则对象的值是 1.

python 标准异常

异常名称	描述
BaseException	所有异常的基类
SystemExit	解释器请求退出
KeyboardInterrupt	用户中断执行(通常是输入^C)
Exception	常规错误的基类
StopIteration	迭代器没有更多的值
GeneratorExit	生成器(generator)发生异常来通知退出
StandardError	所有的内建标准异常的基类
ArithmeticError	所有数值计算错误的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除(或取模)零 (所有数据类型)
AssertionError	断言语句失败
AttributeError	对象没有这个属性
EOFError	没有内建输入,到达 EOF 标记
EnvironmentError	操作系统错误的基类
IOError	输入/输出操作失败
OSError	操作系统错误
WindowsError	系统调用失败
ImportError	导入模块/对象失败
LookupError	无效数据查询的基类
IndexError	序列中没有此索引(index)
KeyError	映射中没有这个键
MemoryError	内存溢出错误(对于 Python 解释器不是致命的)
NameError	未声明/初始化对象 (没有属性)
UnboundLocalError	访问未初始化的本地变量
ReferenceError	弱引用(Weak reference)试图访问已经垃圾回收了的对象
RuntimeError	一般的运行时错误
NotImplementedError	尚未实现的方法
SyntaxError	Python 语法错误
IndentationError	缩进错误
TabError	Tab 和空格混用
SystemError	一般的解释器系统错误

TypeError	对类型无效的操作
ValueError	传入无效的参数
UnicodeError	Unicode 相关的错误
UnicodeDecodeError	Unicode 解码时的错误
UnicodeEncodeError	Unicode 编码时错误
UnicodeTranslateError	Unicode 转换时错误
Warning	警告的基类
DeprecationWarning	关于被弃用的特征的警告
FutureWarning	关于构造将来语义会有改变的警告
OverflowWarning	旧的关于自动提升为长整型(long)的警告
PendingDeprecationWarning	关于特性将会被废弃的警告
RuntimeWarning	可疑的运行时行为(runtime behavior)的警告
SyntaxWarning	可疑的语法的警告
UserWarning	用户代码生成的警告

Python 异常的处理

异常是 python 的对象，用来表示一个错误，当 Python 脚本发生异常时，需要进行捕获处理，否则程序就会终止执行。Python 提供了两个重要的的功能来处理 Python 程序在运行中出现的异常和错误

(1)python 异常处理

(2)断言 Assertions

1.捕捉异常

异常的捕获 捕捉异常可以使用 try/except 语句，该语句用来检测 try 语句块中的错误，从而让 except 语句捕捉异常信息并处理。如果你不想在异常发生时结束你的程序，只需咋 try 里步骤它。

(1)语法

try..expect..else 的语法

```
try:
    <语句> #运行别的代码
except <名字>:
    <语句> #如果在 try 部分引发了'名字'异常，此处的语句执行。
except <名字>,<数据>:
    <语句> #如果引发了'name' 异常，获得附加的数据
else:
    <语句> #如果没有异常发生
```

2.Try 语句工作原理

当开始一个 try 语句后，python 就在当前程序的上下文中作标记，这样当异常发生的

时候，就可以回到这里。Try 子句先执行，接下来发生什么依赖于执行时是否出现异常。

- 如果当 try 后的语句执行时发生异常，python 就跳回到 try 并执行第一个匹配该异常的 except 子句，异常处理完毕，控制流就通过整个 try 语句（除非在处理异常时又引发新的异常）。
- 如果在 try 后的语句里发生了异常，却没有匹配的 except 子句，异常将被递交到上层的 try，或者到程序的最上层（这样将结束程序，并打印缺省的出错信息）。
- 如果在 try 子句执行时没有发生异常，python 将执行 else 语句后的语句（如果有 else 的话），然后控制流通过整个 try 语句。

3.检测所有异常

使用 except 不带任何的异常类型： 使用不带异常类型的 except 语句，会捕获所有发生的异常，但这不是一个很好的方式，我们不能通过该程序识别出具体的异常信息，因为它将捕捉所有的异常。如果使用 except 跟踪多个异常，在异常的类型名称之间使用逗号分割。例如:except (Ex1,Ex2,Ex3,....): pass

4.try-finally 语句

使用 try-finally 语句是无论发生何种异常都将执行最后的代码。try: <语句> finally: <语句> # 退出 try 时总会执行。实例 1(打开一个没有权限的文件):

```
#!/usr/bin/python
try:
    fh = open("testfile", "w")
    try:
        fh.write("This is my test file for exception handling!!")
    finally:
        print "Going to close the file"
        fh.close()
except IOError:
    print "Error: can\'t find file or read data"
```

上当 try 语句块中抛出一个异常后立刻执行 finally 代码,finally 代码执行完毕后，异常再次被抛出，执行 except 代码。

5.异常的参数

一个异常可以带上参数，并且此参数可以作为异常的信息参数，可以通过通过 except 语句来捕捉异常的参数。

```
try:
    Some statement
Except ExceptionType, Argument:
        You can print value of argument here....
```

实例:

```
#!/usr/bin/python
# Define a function here.def temp_convert(var):
try:
    return int(var)
```

```
except ValueError, Argument:
    print "The argument does not contain numbers\n", Argument
```

6.触发异常

触发异常使用 raise 语句: raise [Exception [,args [, tracebace]]] . Raise 语句可以让程序员在需要的地方强制触发某种类型的异常。Raise 异常的参数是可以选的，如果没有提供参数，则为 None。最后一个参数也是可选的，如果存在，是跟踪异常对象。

实例:

```
def functionName( level ):
   if level < 1:
      raise "Invalid level!", level
  # The code below to this would not be executed
   # if we raise the exception
```

注意：为了能够捕获异常，"except"语句必须有用相同的异常来抛出类对象或者字符串。

例如我们捕获以上异常，"except"语句如下所示：

```
try:
    Business Logic here...except "Invalid level!":
Exception handling here...else:
    Rest of the code here..
```

7.用户自定义异常

通过创建一个新类，程序员可以命名自己的异常，典型的自定义异常应该继承自 Exception。

实例:

以下为与 RuntimeError 相关的实例,实例中创建了一个类，基类为 RuntimeError，用于在异常触发时输出更多的信息。在 try 语句块中，用户自定义的异常后执行 except 块语句，变量 e 是用于创建 Networkerror 类的实例。

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

在你定义以上类后，你可以触发该异常，如下所示：

```
try:
    raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

以上内容摘自:

http://www.runoob.com/python/python-exceptions.html

附录:

更多详细内容参见 Pyhon 官方文档:

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

https://docs.python.org/2/tutorial/errors.html