原文链接　http://blog.csdn.net/sinchb/article/details/8392827

事先说明哦，这不是一篇关于 Python 异常的全面介绍的文章，这只是在学习 Python 异常后的一篇笔记式的记录和小结性质的文章。什么？你还不知道什么是异常，额...

1.Python 异常类

Python 是面向对象语言，所以程序抛出的异常也是类。常见的 Python 异常有以下几个，大家只要大致扫一眼，有个映像，等到编程的时候，相信大家肯定会不只一次跟他们照面（除非你不用 Python 了）。

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
SystemExit	Python 解释器请求退出
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
KeyboardInterrupt	用户中断执行(通常是输入^C)
LookupError	无效数据查询的基类
IndexError	序列中没有没有此索引(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

异常	描述
NameError	尝试访问一个没有申明的变量
ZeroDivisionError	除数为 0
SyntaxError	语法错误
IndexError	索引超出序列范围
KeyError	请求一个不存在的字典关键字
IOError	输入输出错误（比如你要读的文件不存在）
AttributeError	尝试访问未知的对象属性

ValueError	传给函数的参数类型不正确，比如给 int()函数传入字符串形

2.捕获异常

Python 完整的捕获异常的语句有点像：

[html] view plaincopy

```
try:
    try_suite
except Exception1,Exception2,...,Argument:
    exception_suite
.....   #other exception block
else:
    no_exceptions_detected_suite
finally:
    always_execute_suite
```

额...是不是很复杂？当然，当我们要捕获异常的时候，并不是必须要按照上面那种格式完全写下来，我们可以丢掉 else 语句，或者 finally 语句；甚至不要 exception 语句，而保留 finally 语句。额，晕了？好吧，下面，我们就来一一说明啦。

2.1.try...except...语句

try_suite 不消我说大家也知道，是我们需要进行捕获异常的代码。而 except 语句是关键，我们 try 捕获了代码段 try_suite 里的异常后，将交给 except 来处理。

try...except 语句最简单的形式如下：

[python] view plaincopy

```
try:
    try_suite
except:
    exception block
```

上面 except 子句不跟任何异常和异常参数，所以无论 try 捕获了任何异常，都将交给 except 子句的 exception block 来处理。如果我们要处理特定的异常，比如说，我们只想处理除零异常，如果其他异常出现，就让其抛出不做处理，该怎么办呢？这个时候，我们就要给 except 子句传入异常参数啦！那个 ExceptionN 就是我们要给 except 子句的异常类（请参考异常类那个表格），表示如果捕获到这类异常，就交给这个 except 子句来处理。比如：

[python] view plaincopy

```
try:
    try_suite
except Exception:
    exception block
```

举个例子：

[python] view plaincopy

```
>>> try:
...     res = 2/0
... except ZeroDivisionError:
...     print "Error:Divisor must not be zero!"
...
Error:Divisor must not be zero!
```

看，我们真的捕获到了 ZeroDivisionError 异常！那如果我想捕获并处理多个异常怎么办呢？有两种办法，一种是给一个 except 子句传入多个异常类参数，另外一种是写多个 except 子句，每个子句都传入你想要处理的异常类参数。甚至，这两种用法可以混搭呢！下面我就来举个例子。

[python] view plaincopy

```
try:
    floatnum = float(raw_input("Please input a float:"))
    intnum = int(floatnum)
    print 100/intnum
```

```
except ZeroDivisionError:
    print "Error:you must input a float num which is large or equal then 1!"
except ValueError:
    print "Error:you must input a float num!"

[root@Cherish tmp]# python test.py
Please input a float:fjia
Error:you must input a float num!
[root@Cherish tmp]# python test.py
Please input a float:0.9999
Error:you must input a float num which is large or equal then 1!
[root@Cherish tmp]# python test.py
Please input a float:25.091
4
```

上面的例子大家一看都懂，就不再解释了。只要大家明白，我们的 except 可以处理一种异常，多种异常，甚至所有异常就可以了。

大家可能注意到了，我们还没解释 except 子句后面那个 Argument 是什么东西？别着急，听我一一道来。这个 Argument 其实是一个异常类的实例（别告诉我你不知到什么是实例），包含了来自异常代码的诊断信息。也就是说，如果你捕获了一个异常，你就可以通过这个异常类的实例来获取更多的关于这个异常的信息。例如：

[python] view plaincopy

```
>>> try:
...     1/0
... except ZeroDivisionError,reason:
...     pass
...
>>> type(reason)
<type 'exceptions.ZeroDivisionError'>
>>> print reason
integer division or modulo by zero
>>> reason
ZeroDivisionError('integer division or modulo by zero',)
>>> reason.__class__
<type 'exceptions.ZeroDivisionError'>
>>> reason.__class__.__doc__
```

```
'Second argument to a division or modulo operation was zero.'
>>> reason.__class__.__name__
'ZeroDivisionError'
```

上面这个例子，我们捕获了除零异常，但是什么都没做。那个 reason 就是异常类 ZeroDivisionError 的实例，通过 type 就可以看出。

2.2try … except…else 语句

现在我们来说说这个 else 语句。Python 中有很多特殊的 else 用法，比如用于条件和循环。放到 try 语句中，其作用其实也差不多：就是当没有检测到异常的时候，则执行 else 语句。举个例子大家可能更明白些：

[python] view plaincopy

```
>>> import syslog
>>> try:
...     f = open("/root/test.py")
... except IOError,e:
...     syslog.syslog(syslog.LOG_ERR,"%s"%e)
... else:
...     syslog.syslog(syslog.LOG_INFO,"no exception caught\n")
...
>>> f.close()
```

2.3 finally 子句

finally 子句是无论是否检测到异常，都会执行的一段代码。我们可以丢掉 except 子句和 else 子句，单独使用 try…finally，也可以配合 except 等使用。

例如 2.2 的例子，如果出现其他异常，无法捕获，程序异常退出，那么文件 f 就没有被正常关闭。这不是我们所希望看到的结果，但是如果我们把 f.close 语句放到 finally 语句中，无论是否有异常，都会正常关闭这个文件，岂不是很 妙

[python] view plaincopy

```
>>> import syslog
>>> try:
...     f = open("/root/test.py")
... except IOError,e:
...     syslog.syslog(syslog.LOG_ERR,"%s"%e)
... else:
...     syslog.syslog(syslog.LOG_INFO,"no exception caught\n")
... finally:
>>>     f.close()
```

大家看到了没，我们上面那个例子竟然用到了 try,except,else,finally 这四个子句！:-)，是不是很有趣？到现在，你就基本上已经学会了如何在 Python 中捕获常规异常并处理之。

3.两个特殊的处理异常的简便方法

3.1 断言（assert）

什么是断言，先看语法：

[python] view plaincopy

```
assert expression[,reason]
```

其中 assert 是断言的关键字。执行该语句的时候，先判断表达式 expression，如果表达式为真，则什么都不做；如果表达式不为真，则抛出异常。reason 跟我们之前谈到的异常类的实例一样。不懂？没关系，举例子！最实在！

[python] view plaincopy

```
>>> assert len('love') == len('like')
>>> assert 1==1
>>> assert 1==2,"1 is not equal 2!"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: 1 is not equal 2!
```

我们可以看到，如果 assert 后面的表达式为真，则什么都不做，如果不为真，就会抛出 AssertionErro 异常，而且我们传进去的字符串会作为异常类的实例的具体信息存在。其实，assert 异常也可以被 try 块捕获：

[python] view plaincopy

```
>>> try:
...     assert 1 == 2 , "1 is not equal 2!"
... except AssertionError,reason:
...     print "%s:%s"%(reason.__class__.__name__,reason)
...
AssertionError:1 is not equal 2!
>>> type(reason)
<type 'exceptions.AssertionError'>
```

3.2.上下文管理（with 语句）

如果你使用 try,except,finally 代码仅仅是为了保证共享资源（如文件，数据）的唯一分配，并在任务结束后释放它，那么你就有福了！这个 with 语句可以让你从 try,except,finally 中解放出来！语法如下：

[python] view plaincopy

```
with context_expr [as var]:
    with_suite
```

是不是不明白？很正常，举个例子来！

[python] view plaincopy

```
>>> with open('/root/test.py') as f:
...     for line in f:
...         print line
```

上面这几行代码干了什么?

（1）打开文件/root/test.py

（2）将文件对象赋值给 f

（3）将文件所有行输出

（4）无论代码中是否出现异常，Python 都会为我们关闭这个文件，我们不需要关心这些细节。

这下，是不是明白了，使用 with 语句来使用这些共享资源，我们不用担心会因为某种原因而没有释放他。但并不是所有的对象都可以使用 with 语句，只有支持上下文管理协议（context management protocol）的对象才可以，那哪些对象支持该协议呢？如下表：

file

decimal.Context

thread.LockType

threading.Lock

threading.RLock

threading.Condition

threading.Semaphore

threading.BoundedSemaphore

至于什么是上下文管理协议，如果你不只关心怎么用 with,以及哪些对象可以使用 with，那么我们就不比太关心这个问题：）

4.抛出异常(raise)

如果我们想要在自己编写的程序中主动抛出异常，该怎么办呢？raise 语句可以帮助我们达到目的。其基本语法如下：

[python] view plaincopy

```
raise [SomeException [, args [,traceback]]
```

第一个参数，SomeException 必须是一个异常类，或异常类的实例

第二个参数是传递给 SomeException 的参数，必须是一个元组。这个参数用来传递关于这个异常的有用信息。

第三个参数 traceback 很少用，主要是用来提供一个跟中记录对象（traceback）

下面我们就来举几个例子。

[python] view plaincopy

```
>>> raise NameError
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError
>>> raise NameError()  #异常类的实例
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError
```

```
>>> raise NameError,("There is a name error","in test.py")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
>>> raise NameError("There is a name error","in test.py") #注意跟上面一个例子的区别
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: ('There is a name error', 'in test.py')
>>> raise NameError,NameError("There is a name error","in test.py") #注意跟上面一个例子的区别
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: ('There is a name error', 'in test.py')
```

其实，我们最常用的还是，只传入第一个参数用来指出异常类型，最多再传入一个元组，用来给出说明信息。如上面第三个例子。

5.异常和 sys 模块

另一种获取异常信息的途径是通过 sys 模块中的 exc_info()函数。该函数回返回一个三元组:(异常类，异常类的实例，跟中记录对象)

[python] view plaincopy

```
>>> try:
...     1/0
... except:
...     import sys
...     tuple = sys.exc_info()
...
>>> print tuple
(<type 'exceptions.ZeroDivisionError'>, ZeroDivisionError('integer division or modulo by zero',), <traceback object at 0x7f538a318b48>)
>>> for i in tuple:
...     print i
...
<type 'exceptions.ZeroDivisionError'> #异常类
integer division or modulo by zero #异常类的实例
<traceback object at 0x7f538a318b48> #跟踪记录对象
```