错误和异常

try

问题引入

为了知道程序运行中出现了哪些错误，考虑定义一个错误码，通过错误码来判断是否出错，出错的原因是什么

```
def a():
    a = 1
    b = 1+ 1
    return -1

def foo():
    r = a()
    if r==(-1):
        return (-1)
    # do something
    return r

def bar():
    r = foo()
    if r==(-1):
        print('Error')
    else:
        pass
bar()
```

```
Error
```

问题：bar() 调用 foo() 调用 a()，返回码 a -> foo -> bar 层层上报，造成大量的冗余代码

引入try

通过try来捕获异常

```
# 未捕获异常，程序去识别异常，并且中断程序，后面的语句不会执行
# r = 10/0
# print("ending")



# 捕获异常
```

```
try:
    print('try...')
    # 分母为0 会触发except
    r = 10 / 0
    print('result:', r)
except ZeroDivisionError as e:
# except:
    # 重试
    # logging
    print('except:', e)
finally:
    print('finally...')
print('END')
```

```
try...
except: division by zero
finally...
END
```

解释：

1. 把可能会执行错误的语句放到 try 下
2. 当try中某行触发错误，该行以后的代码不执行
3. try中代码，若出现异常，则跳转到 except，并执行execpt中内容；若全部执行成功，则不进入except
4. 无论try中是否又异常，都会执行finally中的内容。finally内容可以不写

连续捕获

可以通过多个except来捕获不同类型异常

```
try:
    print('try...')
    # 触发第一个异常
#     r = 10 / int('a')
    # 触发第二个异常
    r = 10 / 0
    print('result:', r)
except ValueError as e:
    print('ValueError:', e)
except ZeroDivisionError as e:
    print('ZeroDivisionError:', e)
else:
    print("当try中无异常时，会执行本代码")
finally:
    print('无论什么情况都会执行的代码')
print('END')
```

```
try...
ZeroDivisionError: division by zero
无论什么情况都会执行的代码
END
```

错误类的继承

1. 错误类有继承的关系，捕获父类错误会把子类错误也捕获
2. BaseException是基类，所有错误类都继承自它

```python
# 汉字对应的unicode范围为\u4E00~\u9FA5,9FA5-4E00=30101,即有30101个汉字
# encode() 作用是将str 转换为bytes 类型
# decode()作用是将bytes 转换为str 类型
try:
    # 超出范围的会触发异常
    a = "\u3E00".encode('gbk')
    print(a, type(a))
#     b = a.decode()
#     print(b, type(b))
except ValueError as e:
    print('ValueError')
except UnicodeError as e:
    print('UnicodeError')
```

```
ValueError
```

- 继承关系

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- 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
```

多层调用

父层可以捕获子层中触发的异常，所以只要在合适的地方写try，而不用每个地方都去写

```python
def foo(s):
    return 10 / int(s)

def bar(s):
    return foo(s) * 2

def main():
    try:
        bar('0')
    except Exception as e:
        print('Error:', e)
    finally:
        print('finally...')
main()
```

```
Error: division by zero
finally...
```

调用栈

若不捕获异常，会一直查找，直到python解释器捕获，强制终止

```python
def foo(s):
    return 10 / int(s)

def bar(s):
    return foo(s) * 2

def main():
    bar('0')

main()
```

```
---------------------------------------------------

ZeroDivisionErrorTraceback (most recent call last)

<ipython-input-93-189c120ae7d8> in <module>
      8     bar('0')
      9
---> 10 main()
```

```
<ipython-input-93-189c120ae7d8> in main()
      6
      7 def main():
----> 8     bar('0')
      9
     10 main()
```

```
<ipython-input-93-189c120ae7d8> in bar(s)
      3
      4 def bar(s):
----> 5     return foo(s) * 2
      6
      7 def main():
```

```
<ipython-input-93-189c120ae7d8> in foo(s)
      1 def foo(s):
----> 2     return 10 / int(s)
      3
      4 def bar(s):
      5     return foo(s) * 2
```

```
ZeroDivisionError: division by zero
```

记录错误

通过logging记录出错的日志

```
import logging

def foo(s):
    return 10 / int(s)

def bar(s):
    return foo(s) * 2

def main():
    try:
        bar('0')
    except Exception as e:
        logging.exception(e)

main()
print("hello")
```

```
ERROR:root:division by zero
Traceback (most recent call last):
  File "<ipython-input-94-e81fa1b7ad43>", line 11, in main
    bar('0')
  File "<ipython-input-94-e81fa1b7ad43>", line 7, in bar
    return foo(s) * 2
  File "<ipython-input-94-e81fa1b7ad43>", line 4, in foo
    return 10 / int(s)
ZeroDivisionError: division by zero
```

```
hello
```

raise

除了捕获异常，还可以手动抛出异常

```
class FooError(ValueError):
    pass

def foo(s):
    n = int(s)
    if n==0:
        raise FooError('无效的参数%s' % s)
    return 10 / n

foo('0')
```

```
---------------------------------------------

FooError       Traceback (most recent call last)

<ipython-input-95-8795b901c7bd> in <module>
      8     return 10 / n
      9
---> 10 foo('0')
```

```
<ipython-input-95-8795b901c7bd> in foo(s)
      5     n = int(s)
      6     if n==0:
----> 7         raise FooError('无效的参数%s' % s)
      8     return 10 / n
      9
```

```
FooError: 无效的参数0
```

- 捕获了之后再抛出异常

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

def bar():
    try:
        foo('0')
    except ValueError as e:
        # 这里是为了记录日志
        print('ValueError!')
        # 不带参数会原样抛出
        raise

bar()
```

```
ValueError!
```

```
---------------------------------------------

ValueError     Traceback (most recent call last)

<ipython-input-96-1d5c390fe5d6> in <module>
     14         raise
     15
---> 16 bar()
```

```
<ipython-input-96-1d5c390fe5d6> in bar()
      7 def bar():
      8     try:
----> 9         foo('0')
     10     except ValueError as e:
     11         # 这里是为了记录日志
```

```
<ipython-input-96-1d5c390fe5d6> in foo(s)
      2     n = int(s)
      3     if n==0:
----> 4         raise ValueError('invalid value: %s' % s)
      5     return 10 / n
      6
```

```
ValueError: invalid value: 0
```

```

```