14.6 处理多个异常¶

问题¶

你有一个代码片段可能会抛出多个不同的异常，怎样才能不创建大量重复代码就能处理所有的可能异常呢？

解决方案¶

如果你可以用单个代码块处理不同的异常，可以将它们放入一个元组中，如下所示：

```
try:
    client_obj.get_url(url)
except (URLError, ValueError, SocketTimeout):
    client_obj.remove_url(url)
```

在这个例子中，元祖中任何一个异常发生时都会执行 `remove_url()` 方法。 如果你想对其中某个异常进行不同的处理，可以将其放入另外一个 `except` 语句中：

```
try:
    client_obj.get_url(url)
except (URLError, ValueError):
    client_obj.remove_url(url)
except SocketTimeout:
    client_obj.handle_url_timeout(url)
```

很多的异常会有层级关系，对于这种情况，你可能使用它们的一个基类来捕获所有的异常。例如，下面的代码：

```
try:
    f = open(filename)
except (FileNotFoundError, PermissionError):
    pass
```

可以被重写为：

```
try:
    f = open(filename)
except OSError:
    pass
```

`OSError` 是 `FileNotFoundError` 和 `PermissionError` 异常的基类。

讨论¶

尽管处理多个异常本身并没什么特殊的，不过你可以使用 `as` 关键字来获得被抛出异常的引用：

```
try:
    f = open(filename)
except OSError as e:
    if e.errno == errno.ENOENT:
        logger.error('File not found')
    elif e.errno == errno.EACCES:
        logger.error('Permission denied')
    else:
        logger.error('Unexpected error: %d', e.errno)
```

这个例子中， `e` 变量指向一个被抛出的 `OSError` 异常实例。 这个在你想更进一步分析这个异常的时候会很有用，比如基于某个状态码来处理它。

同时还要注意的时候 `except` 语句是顺序检查的，第一个匹配的会执行。 你可以很容易的构造多个 `except` 同时匹配的情形，比如：

```
>>> f = open('missing')
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'missing'
>>> try:
...     f = open('missing')
... except OSError:
...     print('It failed')
... except FileNotFoundError:
...     print('File not found')
...
It failed
>>>
```

这里的 `FileNotFoundError` 语句并没有执行的原因是 `OSError` 更一般，它可匹配 `FileNotFoundError` 异常， 于是就是第一个匹配的。 在调试的时候，如果你对某个特定异常的类成层级关系不是很确定， 你可以通过查看该异常的 `__mro__` 属性来快速浏览。比如：

```
>>> FileNotFoundError.__mro__
(<class 'FileNotFoundError'>, <class 'OSError'>, <class 'Exception'>,
 <class 'BaseException'>, <class 'object'>)
>>>
```

上面列表中任何一个直到 `BaseException` 的类都能被用于 `except` 语句。