with关键字的使用

对于系统资源如文件、数据库连接、socket 而言，应用程序打开这些资源并执行完业务逻辑之后，必须做的一件事就是要关闭（断开）该资源。

比如 Python 程序打开一个文件，往文件中写内容，写完之后，就要关闭该文件，否则会出现什么情况呢？极端情况下会出现 "Too many open files" 的错误，因为系统允许你打开的最大文件数量是有限的。

同样，对于数据库，如果连接数过多而没有及时关闭的话，就可能会出现 "Can not connect to MySQL server Too many connections"，因为数据库连接是一种非常昂贵的资源，不可能无限制的被创建。

来看看如何正确关闭一个文件。

- 普通版:

```
def m1():
    f = open("output.txt", "w")
    f.write("python之禅")
    f.close()
```

这样写有一个潜在的问题，如果在调用 write 的过程中，出现了异常进而导致后续代码无法继续执行，close 方法无法被正常调用，因此资源就会一直被该程序占用者释放。那么该如何改进代码呢？

- 进阶版:

```
def m2():
    f = open("output.txt", "w")
    try:
        f.write("python之禅")
    except IOError:
        print("oops error")
    finally:
        f.close()
```

改良版本的程序是对可能发生异常的代码处进行 try 捕获，使用 try/finally 语句，该语句表示如果在 try 代码块中程序出现了异常，后续代码就不再执行，而直接跳转到 except 代码块。而无论如何，finally 块的代码最终都会被执行。因此，只要把 close 放在 finally 代码中，文件就一定会关闭。

- 高级版:

```
def m3():
    with open("output.txt", "r") as f:
        f.write("Python之禅")
```

一种更加简洁、优雅的方式就是用 with 关键字。open 方法的返回值赋值给变量 f，当离开 with 代码块的时候，系统会自动调用 f.close() 方法， with 的作用和使用 try/finally 语句是一样的。

上下文管理器

with语句实质上是一个上下文管理器，with语句后的对象都会有 `__enter__()` 和 `__exit__()` 方法。在进入到上下文时，会自动调用 `__enter__()` 方法，程序正常执行完成，或者出现异常中断的时候，都会调用 `__exit__()` 方法。

```
class MyContext(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __enter__(self):
        print('调用了enter方法')
        return self

    def test(self):
        1 / 0
        print(self.name + '调用了test方法')

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('调用了exit方法')
        print(exc_type, exc_val, exc_tb)

with MyContext('zhangsan', 18) as context:
    context.test()
```