



4.6 上下文管理器

■ 异常处理

```
try:
    f = open("mypython.txt")
    print(f.read())
except IOError as e:
    print(e)
finally:
    f.close()
```

■ with语句

```
with open("mypython.txt") as f:
    print(f.read())
```

- 使用with语句替代try-finally 语句，代码更加的简洁清晰。
- 对于需要**对资源进行访问**的任务，无论在代码运行过程中，是否发生异常，都会执行必要的清理操作，**释放资源**。



■ with语句

```
with open("mypython.txt") as f:  
    print(f.read())
```

运行结果：

```
Python3.0
```

在with语句完成时，会**自动关闭文件**。如果再次读取这个文件，就会出现文件关闭的**错误提示信息**。

```
print(f.read())
```

```
ValueError                                Traceback (most recent call last)  
<ipython-input-5-9a7692027948> in <module>()  
----> 1 print(f.read())  
ValueError: I/O operation on closed file.
```



■ 上下文管理器

- 上下文管理器是Python中的一种**协议**，它保证了每次代码执行的一**致性**
- 一旦进入上下文管理器，就一定会**按照规定的步骤**退出
- 如果合理的设计了退出上下文管理器的步骤，就能够很好的**处理异常**。
- 上下文管理器被最多用到的场景是**资源清理**操作。
- **实现上下文管理器**，只要在类定义时，实现**`__enter__()`方法**和**`__exit__()`方法**即可

```
class A():
    def __init__(self, val_a):
        self.a=val_a

    def __enter__(self):
        print("calss A's __enter__function.")

    def __exit__(self, exc_type, exc_val, exc_tb):
        print("calss A's __exit__function.")
```



□ 实现上下文管理器

```
class A():  
  
    def __init__(self, val_a):  
        self.a=val_a  
  
    def __enter__(self):  
        print("calss A's __enter__ function.")  
  
    def __exit__(self, exc_type, exc_val, exc_tb):  
        print("calss A's __exit__ function.")
```

如果with语句块正常结束，那么这三个参数全部是None。如果发生异常，那么这三个参数的值分别异常的类、实例和跟踪记录。

- 在with语句块**执行前**，首先会执行**enter()**方法
- 当with语句块**执行结束**，无论是否出现异常，都会调用**__exit__()**方法
- 通常将**清除、释放资源**的操作写在**__exit__()**方法中



□ 使用with语句访问上下文管理器

上下文管理器对象，或函数(返回值是上下文管理器对象)

with 上下文管理器表达式 [as 变量]:
语句块

```
with open("mypython.txt") as f:  
    print(f.read())
```



例：模拟实现一个文件类

```
class File():  
  
    def __init__(self, filename, mode):  
        self.filename = filename  
        self.mode = mode  
  
    def __enter__(self):  
        print("执行__enter__()方法")  
        self.f = open(self.filename, self.mode)  
        return self.f  
  
    def __exit__(self, *args):  
        print("执行__exit__方法")  
        self.f.close()
```



以读的方式打开当前目录下的mypythontxt文件，并输出文件内容。

```
with File('mypythontxt', 'r') as f:  
    print(f.read())
```

运行结果：

```
执行__enter__()方法  
Python3.0  
执行__exit__方法
```

- 在执行with语句块之前，首先执行了__enter()__方法，然后再执行with语句块，最后执行__exit()__方法。
- 采用这种方式读取文件时，即使执行过程中出现了异常，__exit()__方法也会被执行，完成关闭文件的操作。

