5.1 读写文本数据¶

问题¶

你需要读写各种不同编码的文本数据，比如ASCII，UTF-8或UTF-16编码等。

解决方案¶

使用带有 `rt` 模式的 `open()` 函数读取文本文件。如下所示：

```
# Read the entire file as a single string
with open('somefile.txt', 'rt') as f:
    data = f.read()

# Iterate over the lines of the file
with open('somefile.txt', 'rt') as f:
    for line in f:
        # process line
        ...
```

类似的，为了写入一个文本文件，使用带有 `wt` 模式的 `open()` 函数， 如果之前文件内容存在则清除并覆盖掉。如下所示：

```
# Write chunks of text data
with open('somefile.txt', 'wt') as f:
    f.write(text1)
    f.write(text2)
    ...

# Redirected print statement
with open('somefile.txt', 'wt') as f:
    print(line1, file=f)
    print(line2, file=f)
    ...
```

如果是在已存在文件中添加内容，使用模式为 `at` 的 `open()` 函数。

文件的读写操作默认使用系统编码，可以通过调用 `sys.getdefaultencoding()` 来得到。 在大多数机器上面都是utf-8编码。如果你已经知道你要读写的文本是其他编码方式， 那么可以通过传递一个可选的 `encoding` 参数给open()函数。如下所示：

```
with open('somefile.txt', 'rt', encoding='latin-1') as f:
    ...
```

Python支持非常多的文本编码。几个常见的编码是ascii, latin-1, utf-8和utf-16。 在web应用程序中通常都使用的是UTF-8。 ascii对应从U+0000到U+007F范围内的7位字符。 latin-1是字节0-255到U+0000至U+00FF范围内Unicode字符的直接映射。 当读取一个未知编码的文本时使用latin-1编码永远不会产生解码错误。 使用latin-1编码读取一个文件的时候也许不能产生完全正确的文本解码数据， 但是它也能从中提取出足够多的有用数据。同时，如果你之后将数据回写回去，原先的数据还是会保留的。

讨论¶

读写文本文件一般来讲是比较简单的。但是也几点是需要注意的。首先，在例子程序中的open语句给被使用到的文件创

读写文本文件一般来讲是比较简单的。但是也有点是需要注意的。首先，在例子程序中的with语句结被使用到的文件创建了一个上下文环境， 但 with 控制块结束时，文件会自动关闭。你也可以不使用 with 语句，但是这时候你就必须记得手动关闭文件：

```
f = open('somefile.txt', 'rt')
data = f.read()
f.close()
```

另外一个问题是关于换行符的识别问题，在Unix和Windows中是不一样的(分别是 \n 和 \r\n)。 默认情况下，Python会以统一模式处理换行符。 这种模式下，在读取文本的时候，Python可以识别所有的普通换行符并将其转换为单个 \n 字符。 类似的，在输出时会将换行符 \n 转换为系统默认的换行符。 如果你不希望这种默认的处理方式，可以给 open() 函数传入参数 newline='' ，就像下面这样：

```
# Read with disabled newline translation
with open('somefile.txt', 'rt', newline='') as f:
    ...
```

为了说明两者之间的差异，下面我在Unix机器上面读取一个Windows上面的文本文件，里面的内容是 hello world!\r\n ：

```
>>> # Newline translation enabled (the default)
>>> f = open('hello.txt', 'rt')
>>> f.read()
'hello world!\n'

>>> # Newline translation disabled
>>> g = open('hello.txt', 'rt', newline='')
>>> g.read()
'hello world!\r\n'
>>>
```

最后一个问题就是文本文件中可能出现的编码错误。 但你读取或者写入一个文本文件时，你可能会遇到一个编码或者解码错误。比如：

```
>>> f = open('sample.txt', 'rt', encoding='ascii')
>>> f.read()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.3/encodings/ascii.py", line 26, in decode
    return codecs.ascii_decode(input, self.errors)[0]
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 in position
12: ordinal not in range(128)
>>>
```

如果出现这个错误，通常表示你读取文本时指定的编码不正确。 你最好仔细阅读说明并确认你的文件编码是正确的(比如使用UTF-8而不是Latin-1编码或其他)。 如果编码错误还是存在的话，你可以给 open() 函数传递一个可选的 errors 参数来处理这些错误。 下面是一些处理常见错误的方法：

```
>>> # Replace bad chars with Unicode U+fffd replacement char
>>> f = open('sample.txt', 'rt', encoding='ascii', errors='replace')
>>> f.read()
'Spicy Jalape?o!'
>>> # Ignore bad chars entirely
>>> g = open('sample.txt', 'rt', encoding='ascii', errors='ignore')
>>> g.read()
'Spicy Jalapeo!'
```

```
Spicy Jalapeo!
>>>
```

如果你经常使用 `errors` 参数来处理编码错误，可能会让你的生活变得很糟糕。 对于文本处理的首要原则是确保你总是使用的是正确编码。当模棱两可的时候，就使用默认的设置(通常都是UTF-8)。