【整理】关于Python脚本开头两行的：#!/usr/bin/python和# -*- coding: utf-8 -*-的作用 - 指定文件编码类型

2013 年 7 月 19 日 下午 12:49 crifan 已有25545人围观 5个评论

#!/usr/bin/python

是用来说明脚本语言是python的

是要用/usr/bin下面的程序（工具）python，这个解释器，来解释python脚本，来运行python脚本的。

-*- coding: utf-8 -*-

是用来指定文件编码为utf-8的

详情可以参考：

PEP 0263 — Defining Python Source Code Encodings

在此，详细的（主要是翻译）解释一下，为何要加这个编码声明，以及如何添加编码声明：

使用文件编码声明以前所遇到的问题

Python 2.1中，想要输入Unicode字符，只能用基于Latin-1的"unicode-escape"的方式输入 -> 对于其他非Latin-1的国家和用户，想要输入Unicode字符，就显得很繁琐，不方便。

希望是：

编程人员，根据自己的喜好和需要，以任意编码方式输入字符串，都可以，这样才正常。

建议选用的方案

所以，才有人给Python官方建议，所以才有此PEP 0263。

此建议就是：

允许在Python文件中，通过文件开始处的，放在注释中的，字符串形式的，声明，声明自己的python文件，用何种编码。

由此，需要很多地方做相应的改动，尤其是Python文件的解析器，可以识别此种文件编码声明。

具体如何声明python文件编码？

上面已经说了，是，文件开始处的，放在注释中的，字符串形式的，声明。

那具体如何声明，以什么样的格式去声明呢？

其实就是，你之前就见过的，这种：

```
# -*- coding: utf-8 -*-
```

对此格式的详细解释是：

1. 如果没有此文件编码类型的声明，则python默认以ASCII编码去处理
   - 如果你没声明编码，但是文件中又包含非ASCII编码的字符的话，python解析器去解析的python文件，自然就会报错了。
2. 必须放在**python**文件的第一行或第二行
3. 支持的格式，可以有三种：
   A. 带等于号的：

```
# coding=<encoding name>
```

   B. 最常见的，带冒号的（大多数编辑器都可以正确识别的）：

```
#!/usr/bin/python
```

```
# -*- coding: <encoding name> -*-
```

C. vim的：

```
#!/usr/bin/python
# vim: set fileencoding=<encoding name> :
```

4. 更加精确的解释是：
   - 符合正则表达式：

```
"coding[:=]\s*([-\w.]+)"
```

   - 的都可以，很明显，如果你熟悉正则表达式，也就可以写出来，其他一些合法的编码声明，以utf-8为例，比如：

```
coding:          utf-8
coding=utf-8
coding=                  utf-8
encoding:utf-8
crifanEncoding=utf-8
```

5. 为了照顾特殊的Windows中的[带BOM（'\xef\xbb\xbf'）的UTF-8]：
   A. 如果你的python文件本身编码是带BOM的UTF-8，即文件前三个字节是：'\xef\xbb\xbf'，那么：
      i. 即使你没有声明文件编码，也自动当做是UTF-8的编码
      ii. 如果你声明了文件编码，则必须是声明了（和你文件编码本身相一致的）UTF-8
         a. 否则（由于声明的编码和实际编码不一致，自然）会报错

文件编码声明的各种例子

针对上面的规则，下面给出各种，合法的，非法的，例子，供参考：

合法的python文件编码声明

1. 带声明了解释器的，Emacs风格的，（注释中的）文件编码声明
   A. 例子1：

```
#!/usr/bin/python
# -*- coding: latin-1 -*-
import os, sys
...
```

   B. 例子2：

```
#!/usr/bin/python
# -*- coding: iso-8859-15 -*-
import os, sys
...
```

C. 例子3：

```
#!/usr/bin/python
# -*- coding: ascii -*-
import os, sys
...
```

2. 不带声明了解释器的，直接用纯文本形式的：

```
# This Python file uses the following encoding: utf-8
import os, sys
...
```

3. 文本编辑器也可以有多种（其他的）定义编码的方式：

```
#!/usr/local/bin/python
# coding: latin-1
import os, sys
...
```

- 很明显，其中的没用-*-，直接用了coding加上编码值

4. 不带编码声明的，默认当做ASCII处理：

```
#!/usr/local/bin/python
import os, sys
...
```

非法的python文件编码声明举例

1. 少了coding:前缀

```
#!/usr/local/bin/python
# latin-1
import os, sys
...
```

2. 编码声明不在第一行或第二行：

```
#!/usr/local/bin/python
#
# -*- coding: latin-1 -*-
import os, sys
...
```

3. 不支持的，非法的字符编码（字符串）声明：

```
#!/usr/local/bin/python
# -*- coding: utf-42 -*-
import os, sys
...
```

python文件编码声明所遵循的理念

1.单个的完整的python源码文件中，只用单一的编码。

->

不允许嵌入了多种的编码的数据

否则会导致（python解释器去解析你的python文件时）报编码错误。

不太懂这段：

> Any encoding which allows processing the first two lines in the way indicated above is allowed as source code encoding, this includes ASCII compatible encodings as well as certain multi-byte encodings such as Shift_JIS. It does not include encodings which use two or more bytes for all characters like e.g. UTF-16. The reason for this is to keep the encoding detection algorithm in the tokenizer simple.

2.这段也不太懂：

> Handling of escape sequences should continue to work as it does now, but with all possible source code encodings, that is standard string literals (both 8-bit and Unicode) are subject to escape sequence expansion while raw string literals only expand a very small subset of escape sequences.

3.Python的分词器+编译器，会按照如下的逻辑去工作：

1. 读取文件
2. 不同的文件，根据其声明的编码去解析为Unicode
3. 转换为UTF-8字符串
4. 针对UTF-8字符串，去分词
5. 编译之，创建Unicode对象

要注意的是：

Python中的标识符，都是ASCII的。

其余的内容，不翻译了。

至此，已经解释的够清楚了。