在 Vim 中，有四个与编码有关的选项，它们是：`fileencodings`、`fileencoding`、`encoding` 和 `termencoding`。在实际使用中，任何一个选项出现错误，都会导致出现乱码。因此，每一个 Vim 用户都应该明确这四个选项的含义。下面，我们详细介绍一下这四个选项的含义和作用。

1 encoding

`encoding` 是 Vim 内部使用的字符编码方式。当我们设置了 `encoding` 之后，Vim 内部所有的 buffer、寄存器、脚本中的字符串等，全都使用这个编码。Vim 在工作的时候，如果编码方式与它的内部编码不一致，它会先把编码转换成内部编码。如果工作用的编码中含有无法转换为内部编码的字符，在这些字符就会丢失。因此，在选择 Vim 的内部编码的时候，一定要使用一种表现能力足够强的编码，以免影响正常工作。

由于 `encoding` 选项涉及到 Vim 中所有字符的内部表示，因此只能在 Vim 启动的时候设置一次。在 Vim 工作过程中修改 `encoding` 会造成非常多的问题。如果没有特别的理由，请始终将 `encoding` 设置为 `utf-8`。为了避免在非 UTF-8 的系统如 Windows 下，菜单和系统提示出现乱码，可同时做这几项设置：

```
set encoding=utf-8
set langmenu=zh_CN.UTF-8
language message zh_CN.UTF-8
```

2 termencoding

`termencoding` 是 Vim 用于屏幕显示的编码，在显示的时候，Vim 会把内部编码转换为屏幕编码，再用于输出。内部编码中含有无法转换为屏幕编码的字符时，该字符会变成问号，但不会影响对它的编辑操作。如果 `termencoding` 没有设置，则直接使用 `encoding` 不进行转换。

举个例子，当你在 Windows 下通过 telnet 登录 Linux 工作站时，由于 Windows 的 telnet 是 GBK 编码的，而 Linux 下使用 UTF-8 编码，你在 telnet 下的 Vim 中就会乱码。此时有两种消除乱码的方式：一是把 Vim 的 `encoding` 改为 `gbk`，另一种方法是保持 `encoding` 为 `utf-8`，把 `termencoding` 改为 `gbk`，让 Vim 在显示的时候转码。显然，使用前一种方法时，如果遇到编辑的文件中含有 GBK 无法表示的字符时，这些字符就会丢失。但如果使用后一种方法，虽然由于终端所限，这些字符无法显示，但在编辑过程中这些字符是不会丢失的。

对于图形界面下的 GVim，它的显示不依赖 TERM，因此 `termencoding` 对

于它没有意义。在 GTK2 下的 GVim 中，`termencoding` 永远是 `utf-8`，并且不能修改。而 Windows 下的 GVim 则忽略 `termencoding` 的存在。

3 fileencoding

当 Vim 从磁盘上读取文件的时候，会对文件的编码进行探测。如果文件的编码方式和 Vim 的内部编码方式不同，Vim 就会对编码进行转换。转换完毕后，Vim 会将 `fileencoding` 选项设置为文件的编码。当 Vim 存盘的时候，如果 `encoding` 和 `fileencoding` 不一样，Vim 就会进行编码转换。因此，通过打开文件后设置 `fileencoding`，我们可以将文件由一种编码转换为另一种编码。但是，由前面的介绍可以看出，`fileencoding` 是在打开文件的时候，由 Vim 进行探测后自动设置的。因此，如果出现乱码，我们无法通过在打开文件后重新设置 `fileencoding` 来纠正乱码。

4 fileencodings

编码的自动识别是通过设置 fileencodings 实现的，注意是复数形式。fileencodings 是一个用逗号分隔的列表，列表中的每一项是一种编码的名称。当我们打开文件的时候，VIM 按顺序使用 fileencodings 中的编码进行尝试解码，如果成功的话，就使用该编码方式进行解码，并将 `fileencoding` 设置为这个值，如果失败的话，就继续试验下一个编码。

因此，我们在设置 `fileencodings` 的时候，一定要把要求严格的、当文件不是这个编码的时候更容易出现解码失败的编码方式放在前面，把宽松的编码方式放在后面。

例如，latin1 是一种非常宽松的编码方式，任何一种编码方式得到的文本，用 latin1 进行解码，都不会发生解码失败——当然，解码得到的结果自然也就是理所当然的“乱码”。因此，如果你把 `latin1` 放到了 `fileencodings` 的第一位的话，打开任何中文文件都是乱码也就是理所当然的了。

以下是滇狐推荐的一个 `fileencodings` 设置：

```
set fileencodings=ucs-bom,utf-8,cp936,gb18030,big5,euc-jp,euc-kr,latin1
```

其中，ucs-bom 是一种非常严格的编码，非该编码的文件几乎没有可能被误判为 ucs-bom，因此放在第一位。

utf-8 也相当严格，除了很短的文件外(例如许多人津津乐道的 GBK 编码的“联通”被误判为 UTF-8 编码的经典错误)，现实生活中一般文件是几乎不可能被误判的，因此放在第二位。

接下来是 cp936 和 gb18030，这两种编码相对宽松，如果放前面的话，会出现大量误判，所以就让它们靠后一些。cp936 的编码空间比 gb18030 小，所

以把 cp936 放在 gb18030 前面。

至于 big5、euc-jp 和 euc-kr，它们的严格程度和 cp936 差不多，把它们放在后面，在编辑这些编码的文件的时候必然出现大量误判，但这是 Vim 内置编码探测机制没有办法解决的事。由于中国用户很少有机会编辑这些编码的文件，因此我们还是决定把 cp936 和 gb18030 前提以保证这些编码的识别。

最后就是 latin1 了。它是一种极其宽松的编码，以至于我们不得不把它放在最后一位。不过可惜的是，当你碰到一个真的 latin1 编码的文件时，绝大部分情况下，它没有机会 fall-back 到 latin1，往往在前面的编码中就被误判了。不过，正如前面所说的，中国用户没有太多机会接触这样的文件。

如果编码被误判了，解码后的结果就无法被人类识别，于是我们就说，这个文件乱码了。此时，如果你知道这个文件的正确编码的话，可以在打开文件的时候使用 `++enc=encoding` 的方式来打开文件，如：

```
:e ++enc=utf-8 myfile.txt
```

5 fencview

根据前面的介绍，我们知道，通过 Vim 内置的编码识别机制，识别率是很低的，尤其是对于简体中文（GBK/GB18030）、繁体中文（Big5）、日文（euc-jp）和韩文（euc-kr）之间的识别。而对于普通用户而言，肉眼看出一个文件的编码方式也是很不现实的事情。因此，滇狐强烈推荐水木社区的 mbbill 开发的 fencview 插件。该插件使用词频统计的方式识别编码，正确率非常高。点击这里下载。

标签：VIM 教程 编码 乱码