**[Python模块学习]chardet模块识别字节包编码**

原创 2016年09月10日 19:30:33

* 标签：
* [python](http://so.csdn.net/so/search/s.do?q=python&t=blog) /
* [编码](http://so.csdn.net/so/search/s.do?q=%E7%BC%96%E7%A0%81&t=blog)
* 2473

**chardet模块**

***by: uizuizui***

这个模块是[the auto-detection code in Mozilla](http://lxr.mozilla.org/seamonkey/source/extensions/universalchardet/src/base/" \t "_blank)的一部分源码加上模块作者的一些改进而成的。模块作者保留了源代码作者代码注释。

功能描述：chardet模块即[Universal Encoding Detector](https://github.com/chardet/chardet" \t "_blank)，功能正如其名，是一个用于字符检测的模块。它可以帮助我们识别一段未知格式的字节是属于什么编码格式。从而可以顺利地将bytes转换为str。

安装： pip install chardet

**简单用法**

chardet的使用非常简单，主模块里面只有一个函数detect。detect有一个参数，要求是bytes类型。bytes类型可以通过读取网页内容、open函数的rb模式、带b前缀的字符串、encode函数等途径获得。

示例代码：

import chardet

some\_string = '你好，世界。'.encode('utf-8') # encode方法返回一个bytes

# b'\xe4\xbd\xa0\xe5\xa5\xbd\xef\xbc\x8c\xe4\xb8\x96\xe7\x95\x8c\xe3\x80\x82'

result = chardet.detect(some\_string) # 调用检测接口

print(result)

# {'encoding': 'utf-8', 'confidence': 0.99}

如上所示，detect函数返回一个字典，字典里有两个key-value对。其中一个的key值为encoding，代表chardet推断出来的编码格式。另一个key值为confidence，代表可信度。可信度是一个0到1之间float值，0代表不可信，1代表百分之百可信。

**高级用法**

当用于检测的文档特别大时，可以chardet的子模块chardet.universaldetector。这个模块允许我们分多次（逐行读取或者自行断行读取）检测文本的编码格式，当达到一定的阈值时便可以提前退出检测。这样做可以有效地节省资源，提高程序效率，同时保证检测结果的准确性。

示例代码：

from chardet.universaldetector import UniversalDetector

detector = UniversalDetector() # 初始化一个UniversalDetector对象

f = open('test.txt', 'rb') # test.txt是一个utf-8编码的文本文档

for line in f:

detector.feed(line) # 逐行载入UniversalDetector对象中进行识别

if detector.done: # done为一个布尔值，默认为False，达到阈值时变为True

break

detector.close() # 调用该函数做最后的数据整合

f.close()

print(detector.result)

# {'confidence': 1.0, 'encoding': 'UTF-8-SIG'}

**需要注意的是：如果对多个不同来源的文本进行检测，在每次检测完毕时，必须调用一次UniversalDetector对象的reset函数，将之前的检测数据清除。否则会导致后面的检测结果混乱。**

**目前支持的编码格式**

Universal Encoding Detector目前支持以下编码格式：

* Big5, GB2312/GB18030, EUC-TW, HZ-GB-2312, and ISO-2022-CN (Traditional and Simplified - Chinese 繁体和简体中文)
* EUC-JP, SHIFT\_JIS, and ISO-2022-JP (Japanese 日文)
* EUC-KR and ISO-2022-KR (Korean 韩文)
* KOI8-R, MacCyrillic, IBM855, IBM866, ISO-8859-5, and windows-1251 (Russian 俄文)
* ISO-8859-2 and windows-1250 (Hungarian 匈牙利文)
* ISO-8859-5 and windows-1251 (Bulgarian 保加利亚文)
* ISO-8859-1 and windows-1252 (Western European languages 西欧文字)
* ISO-8859-7 and windows-1253 (Greek 希腊文)
* ISO-8859-8 and windows-1255 (Visual and Logical Hebrew 视觉顺序和逻辑顺序的希伯来文)
* TIS-620 (Thai 泰文)
* UTF-32 BE, LE, 3412-ordered, or 2143-ordered (with a BOM 带字节序标记)
* UTF-16 BE or LE (with a BOM 带字节序标记)
* UTF-8 (with or without a BOM 带或者不带字节序标记)
* ASCII

**注意：由于内部的相似性，某些情况下会出现检测错误。最常出问题的是匈牙利文，会出现报告的编码是两种之中的另一种。希腊文的检测也经常错误的将ISO-8859-7识别为匈牙利文的ISO-8859-2。**

**关于检测时出现的奇怪错误**

*该模块在检测ANSI编码（在中文版windows系统上就是gbk）的时候会出现一些奇怪的错误，博主正在研究英文文档，希望能在上面找到答案。如果有后续，会同步更新至本文。*

接上文：

**问题来源**：某些情况下检测ANSI编码的文本文档和gb2312编码的字节包会出现错误。

**过程**：博主分别对不同情况下录入、不同长度和不同编码的字节包进行了测试。并仔细看了一遍官方文档。大概理出了个思路。

**解析**：官方文档上有一段话，我先原文摘下来。

If UniversalDetector detects a high-bit character in the text, but none of the other multi-byte or single-byte encoding probers return a confident result, it creates a Latin1Prober (defined in latin1prober.py) to try to detect English text in a windows-1252 encoding. This detection is inherently unreliable, because English letters are encoded in the same way in many different encodings. The only way to distinguish windows-1252 is through commonly used symbols like smart quotes, curly apostrophes, copyright symbols, and the like. Latin1Prober automatically reduces its confidence rating to allow more accurate probers to win if at all possible.

大概意思是，当UniversalDetector在解析一些字节的时候，如果没有相应的探测器给出报告，它就会调用一个叫做Latin1Prober的探测器试图用英文编码windows-1252来解析该字节包，这个探测器非常不可信（官方吐槽…）。通常情况下英文字母和一些特殊符号在不同的编码里是一样的，所以这个探测器会给出很高的可信度。这个探测器会自动地压低自己给出的可信度以让其他探测器可以优先通过。

再根据以下这段原文：

The main entry point for the detection algorithm is universaldetector.py, which has one class, UniversalDetector. (You might think the main entry point is the detect function in chardet/**init**.py, but that’s really just a convenience function that creates a UniversalDetector object, calls it, and returns its result.)

大致意思是：检测算法的入口是UniversalDetector，chardet.detect函数仅仅只是方便用户使用的语法糖。

**可以得出类似的机制也会出现在detect函数中**。尽管Latin1Prober已经做了优化，但是还是会有一些情况下它给出了远超实际的可信度。比如下面这个实验：

>>> import chardet

>>> chardet.detect('哈'.encode('gb2312'))

{'confidence': 0.73, 'encoding': 'windows-1252'}

博主还做了几个其他的实验，得出了一个结论：**当字节包的长度不够长时，chardet给出的结论是很不可靠的，因为它可能在调用一个不相干的探测器时，该探测器给出了一个超过阈值的可信度，或者两个编码格式刚好有共通的字符**，然后就不再往下检测。这么做很容易出现不可靠的检测结果。所以最好不要检测很少量的字节。同时，在检测开篇有大段其他字符的文档时，最好先手动处理掉无关的符号（*不一定会发生错误，因为程序根据初次遍历结果对探测器的顺序做了优先级排序，但是保不准会出错*），以求最准确的结果。

[How Universal Encoding Detector Works](http://chardet.readthedocs.io/en/latest/how-it-works.html)这个网址详细地说明了模块是如何运作的。建议能看懂英文并且有耐心的读者通读一遍。博主只是选择性地快速阅读了一遍，不能保证把原作者的意思传达到各位这里，但是也能保证偏差不会太大。

参考资料：   
[chardet doc](http://chardet.readthedocs.io/en/latest/index.html)   
[python 模块 chardet下载方法及介绍](http://blog.csdn.net/aqwd2008/article/details/7506007)

***感谢以上模块提供者和内容提供者***

版权声明：以上内容遵循CC BY-NC-SA协议(https://creativecommons.org/licenses/by-nc-sa/4.0/deed.zh)，欢迎转载，转载请注明作者。 https://blog.csdn.net/jy692405180/article/details/52496599