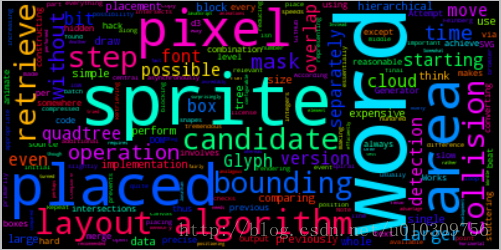
### [生成词云之python中WordCloud包的用法](http://blog.csdn.net/u010309756/article/details/67637930)

分类：***python***

 （1339）  （2）

效果图：   


这是python中使用wordcloud包生成的词云图。

下面来介绍一下wordcloud包的基本用法。

class wordcloud.WordCloud(font\_path=None, width=400, height=200, margin=2, ranks\_only=None, prefer\_horizontal=0.9,mask=None, scale=1, color\_func=None, max\_words=200, min\_font\_size=4, stopwords=None, random\_state=None,background\_color='black', max\_font\_size=None, font\_step=1, mode='RGB', relative\_scaling=0.5, regexp=None, collocations=True,colormap=None, normalize\_plurals=True)

* 1

这是wordcloud的所有参数，下面具体介绍一下各个参数：

font\_path : string //字体路径，需要展现什么字体就把该字体路径+后缀名写上，如：font\_path = '黑体.ttf'

width : int (default=400) //输出的画布宽度，默认为400像素

height : int (default=200) //输出的画布高度，默认为200像素

prefer\_horizontal : float (default=0.90) //词语水平方向排版出现的频率，默认 0.9 （所以词语垂直方向排版出现频率为 0.1 ）

mask : nd-array or None (default=None) //如果参数为空，则使用二维遮罩绘制词云。如果 mask 非空，设置的宽高值将被忽略，遮罩形状被 mask 取代。除全白（#FFFFFF）的部分将不会绘制，其余部分会用于绘制词云。如：bg\_pic = imread('读取一张图片.png')，背景图片的画布一定要设置为白色（#FFFFFF），然后显示的形状为不是白色的其他颜色。可以用ps工具将自己要显示的形状复制到一个纯白色的画布上再保存，就ok了。

scale : float (default=1) //按照比例进行放大画布，如设置为1.5，则长和宽都是原来画布的1.5倍。

min\_font\_size : int (default=4) //显示的最小的字体大小

font\_step : int (default=1) //字体步长，如果步长大于1，会加快运算但是可能导致结果出现较大的误差。

max\_words : number (default=200) //要显示的词的最大个数

stopwords : set of strings or None //设置需要屏蔽的词，如果为空，则使用内置的STOPWORDS

background\_color : color value (default=”black”) //背景颜色，如background\_color='white',背景颜色为白色。

max\_font\_size : int or None (default=None) //显示的最大的字体大小

mode : string (default=”RGB”) //当参数为“RGBA”并且background\_color不为空时，背景为透明。

relative\_scaling : float (default=.5) //词频和字体大小的关联性

color\_func : callable, default=None //生成新颜色的函数，如果为空，则使用 self.color\_func

regexp : string or None (optional) //使用正则表达式分隔输入的文本

collocations : bool, default=True //是否包括两个词的搭配

colormap : string or matplotlib colormap, default=”viridis” //给每个单词随机分配颜色，若指定color\_func，则忽略该方法。

fit\_words(frequencies) //根据词频生成词云

generate(text) //根据文本生成词云

generate\_from\_frequencies(frequencies[, ...]) //根据词频生成词云

generate\_from\_text(text) //根据文本生成词云

process\_text(text) //将长文本分词并去除屏蔽词（此处指英语，中文分词还是需要自己用别的库先行实现，使用上面的 fit\_words(frequencies) ）

recolor([random\_state, color\_func, colormap]) //对现有输出重新着色。重新上色会比重新生成整个词云快很多。

to\_array() //转化为 numpy array

to\_file(filename) //输出到文件

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 38
* 39
* 40
* 41
* 42
* 43
* 44
* 45
* 46

例子：   
想要生成的词云的形状：   
   
图中黑色部分就是词云的将要显示的部分，白色部分不显示任何词。

下面是一个文本文档：

How the Word Cloud Generator Works

The layout algorithm for positioning words without overlap is available on GitHub under an open source license as d3-cloud. Note that this is the only the layout algorithm and any code for converting text into words and rendering the final output requires additional development.

As word placement can be quite slow for more than a few hundred words, the layout algorithm can be run asynchronously, with a configurable time step size. This makes it possible to animate words as they are placed without stuttering. It is recommended to always use a time step even without animations as it prevents the browser’s event loop from blocking while placing the words.

The layout algorithm itself is incredibly simple. For each word, starting with the most “important”:

Attempt to place the word at some starting point: usually near the middle, or somewhere on a central horizontal line.

If the word intersects with any previously placed words, move it one step along an increasing spiral. Repeat until no intersections are found.

The hard part is making it perform efficiently! According to Jonathan Feinberg, Wordle uses a combination of hierarchical bounding boxes and quadtrees to achieve reasonable speeds.

Glyphs in JavaScript

There isn’t a way to retrieve precise glyph shapes via the DOM, except perhaps for SVG fonts. Instead, we draw each word to a hidden canvas element, and retrieve the pixel data.

Retrieving the pixel data separately for each word is expensive, so we draw as many words as possible and then retrieve their pixels in a batch operation.

Sprites and Masks

My initial implementation performed collision detection using sprite masks. Once a word is placed, it doesn't move, so we can copy it to the appropriate position in a larger sprite representing the whole placement area.

The advantage of this is that collision detection only involves comparing a candidate sprite with the relevant area of this larger sprite, rather than comparing with each previous word separately.

Somewhat surprisingly, a simple low-level hack made a tremendous difference: when constructing the sprite I compressed blocks of 32 1-bit pixels into 32-bit integers, thus reducing the number of checks (and memory) by 32 times.

In fact, this turned out to beat my hierarchical bounding box with quadtree implementation on everything I tried it on (even very large areas and font sizes). I think this is primarily because the sprite version only needs to perform a single collision test per candidate area, whereas the bounding box version has to compare with every other previously placed word that overlaps slightly with the candidate area.

Another possibility would be to merge a word’s tree with a single large tree once it is placed. I think this operation would be fairly expensive though compared with the analagous sprite mask operation, which is essentially ORing a whole block.

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29

从这个文本中生成一个词云，代码如下：

#!/usr/bin/python# -\*- coding: utf-8 -\*-#coding=utf-8

#导入wordcloud模块和matplotlib模块

from wordcloud import WordCloud

import matplotlib.pyplot as plt

from scipy.misc import imread

#读取一个txt文件

text = open('test.txt','r').read()

#读入背景图片

bg\_pic = imread('3.png')

#生成词云

wordcloud = WordCloud(mask=bg\_pic,background\_color='white',scale=1.5).generate(text)

image\_colors = ImageColorGenerator(bg\_pic)#显示词云图片

plt.imshow(wordcloud)

plt.axis('off')

plt.show()

#保存图片

wordcloud.to\_file('test.jpg')

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33

运行结果：   
