[**geohash算法原理及实现方式**](http://blog.csdn.net/wangxiafghj/article/details/9014363)

分类： [text mining](http://blog.csdn.net/wangxiafghj/article/category/1184218) 2013-06-03 16:04 5312人阅读 [评论](http://blog.csdn.net/wangxiafghj/article/details/9014363#comments)(0) 收藏 [举报](http://blog.csdn.net/wangxiafghj/article/details/9014363#report)

目录[(?)[+]](http://blog.csdn.net/wangxiafghj/article/details/9014363)

**原地址：**[**http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html**](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html)

**[geohash算法原理及实现方式](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html)**

**1、**[**geohash特点**](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html#a)

**2、**[**geohash原理**](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html#b)

**3、**[**geohash的php 、python、java、C#实现代码**](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html#d)

**4、**[**观点讨论**](http://www.cnblogs.com/dengxinglin/archive/2012/12/14/2817761.html#f)

 w微博:<http://weibo.com/dxl0321>

**geohash有以下几个特点：**

首先，geohash用一个字符串表示经度和纬度两个坐标。某些情况下无法在两列上同时应用索引 （例如MySQL 4之前的版本，Google App Engine的数据层等），利用geohash，只需在一列上应用索引即可。

其次，geohash表示的并不是一个点，而是一个矩形区域。比如编码wx4g0ec19，它表示的是一个矩形区域。 使用者可以发布地址编码，既能表明自己位于北海公园附近，又不至于暴露自己的精确坐标，有助于隐私保护。

第三，编码的前缀可以表示更大的区域。例如wx4g0ec1，它的前缀wx4g0e表示包含编 码wx4g0ec1在内的更大范围。 这个特性可以用于附近地点搜索。首先根据用户当前坐标计算geohash（例如wx4g0ec1）然后取其前缀进行查询 （SELECT \* FROM place WHERE geohash LIKE 'wx4g0e%'），即可查询附近的所有地点。

Geohash比直接用经纬度的高效很多。

**Geohash的原理**

**Geohash的最简单的解释就是：将一个经纬度信息，转换成一个可以排序，可以比较的字符串编码**

        首先将纬度范围(-90, 90)平分成两个区间(-90,0)、(0, 90)，如果目标纬度位于前一个区间，则编码为0，否则编码为1。

由于39.92324属于(0, 90)，所以取编码为1。

然后再将(0, 90)分成 (0, 45), (45, 90)两个区间，而39.92324位于(0, 45)，所以编码为0。

以此类推，直到精度符合要求为止，得到纬度编码为1011 1000 1100 0111 1001。

|  |  |  |  |
| --- | --- | --- | --- |
| 纬度范围 | 划分区间0 | 划分区间1 | 39.92324所属区间 |
| (-90, 90) | (-90, 0.0) | (0.0, 90) | 1 |
| (0.0, 90) | (0.0, 45.0) | (45.0, 90) | 0 |
| (0.0, 45.0) | (0.0, 22.5) | (22.5, 45.0) | 1 |
| (22.5, 45.0) | (22.5, 33.75) | (33.75, 45.0) | 1 |
| (33.75, 45.0) | (33.75, 39.375) | (39.375, 45.0) | 1 |
| (39.375, 45.0) | (39.375, 42.1875) | (42.1875, 45.0) | 0 |
| (39.375, 42.1875) | (39.375, 40.7812) | (40.7812, 42.1875) | 0 |
| (39.375, 40.7812) | (39.375, 40.0781) | (40.0781, 40.7812) | 0 |
| (39.375, 40.0781) | (39.375, 39.7265) | (39.7265, 40.0781) | 1 |
| (39.7265, 40.0781) | (39.7265, 39.9023) | (39.9023, 40.0781) | 1 |
| (39.9023, 40.0781) | (39.9023, 39.9902) | (39.9902, 40.0781) | 0 |
| (39.9023, 39.9902) | (39.9023, 39.9462) | (39.9462, 39.9902) | 0 |
| (39.9023, 39.9462) | (39.9023, 39.9243) | (39.9243, 39.9462) | 0 |
| (39.9023, 39.9243) | (39.9023, 39.9133) | (39.9133, 39.9243) | 1 |
| (39.9133, 39.9243) | (39.9133, 39.9188) | (39.9188, 39.9243) | 1 |
| (39.9188, 39.9243) | (39.9188, 39.9215) | (39.9215, 39.9243) | 1 |

经度也用同样的算法，对(-180, 180)依次细分，得到116.3906的编码为1101 0010 1100 0100 0100。

|  |  |  |  |
| --- | --- | --- | --- |
| 经度范围 | 划分区间0 | 划分区间1 | 116.3906所属区间 |
| (-180, 180) | (-180, 0.0) | (0.0, 180) | 1 |
| (0.0, 180) | (0.0, 90.0) | (90.0, 180) | 1 |
| (90.0, 180) | (90.0, 135.0) | (135.0, 180) | 0 |
| (90.0, 135.0) | (90.0, 112.5) | (112.5, 135.0) | 1 |
| (112.5, 135.0) | (112.5, 123.75) | (123.75, 135.0) | 0 |
| (112.5, 123.75) | (112.5, 118.125) | (118.125, 123.75) | 0 |
| (112.5, 118.125) | (112.5, 115.312) | (115.312, 118.125) | 1 |
| (115.312, 118.125) | (115.312, 116.718) | (116.718, 118.125) | 0 |
| (115.312, 116.718) | (115.312, 116.015) | (116.015, 116.718) | 1 |
| (116.015, 116.718) | (116.015, 116.367) | (116.367, 116.718) | 1 |
| (116.367, 116.718) | (116.367, 116.542) | (116.542, 116.718) | 0 |
| (116.367, 116.542) | (116.367, 116.455) | (116.455, 116.542) | 0 |
| (116.367, 116.455) | (116.367, 116.411) | (116.411, 116.455) | 0 |
| (116.367, 116.411) | (116.367, 116.389) | (116.389, 116.411) | 1 |
| (116.389, 116.411) | (116.389, 116.400) | (116.400, 116.411) | 0 |
| (116.389, 116.400) | (116.389, 116.394) | (116.394, 116.400) | 0 |

接下来将经度和纬度的编码合并，奇数位是纬度，偶数位是经度，得到编码 11100 11101 00100 01111 00000 01101 01011 00001。

最后，用0-9、b-z（去掉a, i, l, o）这32个字母进行base32编码，得到(39.92324, 116.3906)的编码为wx4g0ec1。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 十进制 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| base32 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | b | c | d | e | f | g |
| 十进制 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| base32 | h | j | k | m | n | p | q | r | s | t | u | v | w | x | y | z |

解码算法与编码算法相反，先进行base32解码，然后分离出经纬度，最后根据二进制编码对经纬度范围进行细分即可，这里不再赘述。

**实现代码：**

php版本的实现方式：<http://blog.dixo.net/downloads/geohash-php-class/>  我下载了一个上传的

**php：**

geohash.class.php

**[php]** [view plaincopy](http://blog.csdn.net/wangxiafghj/article/details/9014363)

1. View Code
2. <?php
3. /\*\*
4. \* Geohash generation class
5. \* http://blog.dixo.net/downloads/
6. \*
7. \* This file copyright (C) 2008 Paul Dixon (paul@elphin.com)
8. \*
9. \* This program is free software; you can redistribute it and/or
10. \* modify it under the terms of the GNU General Public License
11. \* as published by the Free Software Foundation; either version 3
12. \* of the License, or (at your option) any later version.
13. \*
14. \* This program is distributed in the hope that it will be useful,
15. \* but WITHOUT ANY WARRANTY; without even the implied warranty of
16. \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17. \* GNU General Public License for more details.
18. \*
19. \* You should have received a copy of the GNU General Public License
20. \* along with this program; if not, write to the Free Software
21. \* Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
22. \*/


26. /\*\*
27. \* Encode and decode geohashes
28. \*
29. \*/
30. class Geohash
31. {
32. private $coding="0123456789bcdefghjkmnpqrstuvwxyz";
33. private $codingMap=array();
35. public function Geohash()
36. {
37. //build map from encoding char to 0 padded bitfield
38. for($i=0; $i<32; $i++)
39. {
40. $this->codingMap[substr($this->coding,$i,1)]=str\_pad(decbin($i), 5, "0", STR\_PAD\_LEFT);
41. }
43. }
45. /\*\*
46. \* Decode a geohash and return an array with decimal lat,long in it
47. \*/
48. public function decode($hash)
49. {
50. //decode hash into binary string
51. $binary="";
52. $hl=strlen($hash);
53. for($i=0; $i<$hl; $i++)
54. {
55. $binary.=$this->codingMap[substr($hash,$i,1)];
56. }
58. //split the binary into lat and log binary strings
59. $bl=strlen($binary);
60. $blat="";
61. $blong="";
62. for ($i=0; $i<$bl; $i++)
63. {
64. if ($i%2)
65. $blat=$blat.substr($binary,$i,1);
66. else
67. $blong=$blong.substr($binary,$i,1);
69. }
71. //now concert to decimal
72. $lat=$this->binDecode($blat,-90,90);
73. $long=$this->binDecode($blong,-180,180);
75. //figure out how precise the bit count makes this calculation
76. $latErr=$this->calcError(strlen($blat),-90,90);
77. $longErr=$this->calcError(strlen($blong),-180,180);
79. //how many decimal places should we use? There's a little art to
80. //this to ensure I get the same roundings as geohash.org
81. $latPlaces=max(1, -round(log10($latErr))) - 1;
82. $longPlaces=max(1, -round(log10($longErr))) - 1;
84. //round it
85. $lat=round($lat, $latPlaces);
86. $long=round($long, $longPlaces);
88. return array($lat,$long);
89. }

92. /\*\*
93. \* Encode a hash from given lat and long
94. \*/
95. public function encode($lat,$long)
96. {
97. //how many bits does latitude need?
98. $plat=$this->precision($lat);
99. $latbits=1;
100. $err=45;
101. while($err>$plat)
102. {
103. $latbits++;
104. $err/=2;
105. }
107. //how many bits does longitude need?
108. $plong=$this->precision($long);
109. $longbits=1;
110. $err=90;
111. while($err>$plong)
112. {
113. $longbits++;
114. $err/=2;
115. }
117. //bit counts need to be equal
118. $bits=max($latbits,$longbits);
120. //as the hash create bits in groups of 5, lets not
121. //waste any bits - lets bulk it up to a multiple of 5
122. //and favour the longitude for any odd bits
123. $longbits=$bits;
124. $latbits=$bits;
125. $addlong=1;
126. while (($longbits+$latbits)%5 != 0)
127. {
128. $longbits+=$addlong;
129. $latbits+=!$addlong;
130. $addlong=!$addlong;
131. }

134. //encode each as binary string
135. $blat=$this->binEncode($lat,-90,90, $latbits);
136. $blong=$this->binEncode($long,-180,180,$longbits);
138. //merge lat and long together
139. $binary="";
140. $uselong=1;
141. while (strlen($blat)+strlen($blong))
142. {
143. if ($uselong)
144. {
145. $binary=$binary.substr($blong,0,1);
146. $blong=substr($blong,1);
147. }
148. else
149. {
150. $binary=$binary.substr($blat,0,1);
151. $blat=substr($blat,1);
152. }
153. $uselong=!$uselong;
154. }
156. //convert binary string to hash
157. $hash="";
158. for ($i=0; $i<strlen($binary); $i+=5)
159. {
160. $n=bindec(substr($binary,$i,5));
161. $hash=$hash.$this->coding[$n];
162. }

165. return $hash;
166. }
168. /\*\*
169. \* What's the maximum error for $bits bits covering a range $min to $max
170. \*/
171. private function calcError($bits,$min,$max)
172. {
173. $err=($max-$min)/2;
174. while ($bits--)
175. $err/=2;
176. return $err;
177. }
179. /\*
180. \* returns precision of number
181. \* precision of 42 is 0.5
182. \* precision of 42.4 is 0.05
183. \* precision of 42.41 is 0.005 etc
184. \*/
185. private function precision($number)
186. {
187. $precision=0;
188. $pt=strpos($number,'.');
189. if ($pt!==false)
190. {
191. $precision=-(strlen($number)-$pt-1);
192. }
194. return pow(10,$precision)/2;
195. }

198. /\*\*
199. \* create binary encoding of number as detailed in http://en.wikipedia.org/wiki/Geohash#Example
200. \* removing the tail recursion is left an exercise for the reader
201. \*/
202. private function binEncode($number, $min, $max, $bitcount)
203. {
204. if ($bitcount==0)
205. return "";
207. #echo "$bitcount: $min $max<br>";
209. //this is our mid point - we will produce a bit to say
210. //whether $number is above or below this mid point
211. $mid=($min+$max)/2;
212. if ($number>$mid)
213. return "1".$this->binEncode($number, $mid, $max,$bitcount-1);
214. else
215. return "0".$this->binEncode($number, $min, $mid,$bitcount-1);
216. }

219. /\*\*
220. \* decodes binary encoding of number as detailed in http://en.wikipedia.org/wiki/Geohash#Example
221. \* removing the tail recursion is left an exercise for the reader
222. \*/
223. private function binDecode($binary, $min, $max)
224. {
225. $mid=($min+$max)/2;
227. if (strlen($binary)==0)
228. return $mid;
230. $bit=substr($binary,0,1);
231. $binary=substr($binary,1);
233. if ($bit==1)
234. return $this->binDecode($binary, $mid, $max);
235. else
236. return $this->binDecode($binary, $min, $mid);
237. }
238. }





245. ?>

**python：**

python版本的geohash：[python-geohash](http://code.google.com/p/python-geohash/)

**java：**

java版本的geohash,实现：<http://code.google.com/p/geospatialweb/source/browse/#svn/trunk/geohash/src>

**[java]** [view plaincopy](http://blog.csdn.net/wangxiafghj/article/details/9014363)

1. View Code
2. import java.io.File;
3. import java.io.FileInputStream;
4. import java.util.BitSet;
5. import java.util.HashMap;

8. public class Geohash {
10. private static int numbits = 6 \* 5;
11. final static char[] digits = { '0', '1', '2', '3', '4', '5', '6', '7', '8',
12. '9', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'j', 'k', 'm', 'n', 'p',
13. 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z' };
15. final static HashMap<Character, Integer> lookup = new HashMap<Character, Integer>();
16. static {
17. int i = 0;
18. for (char c : digits)
19. lookup.put(c, i++);
20. }
22. public static void main(String[] args)  throws Exception{
24. System.out.println(new Geohash().encode(45, 125));
26. }
28. public double[] decode(String geohash) {
29. StringBuilder buffer = new StringBuilder();
30. for (char c : geohash.toCharArray()) {
32. int i = lookup.get(c) + 32;
33. buffer.append( Integer.toString(i, 2).substring(1) );
34. }
36. BitSet lonset = new BitSet();
37. BitSet latset = new BitSet();
39. //even bits
40. int j =0;
41. for (int i=0; i< numbits\*2;i+=2) {
42. boolean isSet = false;
43. if ( i < buffer.length() )
44. isSet = buffer.charAt(i) == '1';
45. lonset.set(j++, isSet);
46. }
48. //odd bits
49. j=0;
50. for (int i=1; i< numbits\*2;i+=2) {
51. boolean isSet = false;
52. if ( i < buffer.length() )
53. isSet = buffer.charAt(i) == '1';
54. latset.set(j++, isSet);
55. }
57. double lon = decode(lonset, -180, 180);
58. double lat = decode(latset, -90, 90);
60. return new double[] {lat, lon};
61. }
63. private double decode(BitSet bs, double floor, double ceiling) {
64. double mid = 0;
65. for (int i=0; i<bs.length(); i++) {
66. mid = (floor + ceiling) / 2;
67. if (bs.get(i))
68. floor = mid;
69. else
70. ceiling = mid;
71. }
72. return mid;
73. }

76. public String encode(double lat, double lon) {
77. BitSet latbits = getBits(lat, -90, 90);
78. BitSet lonbits = getBits(lon, -180, 180);
79. StringBuilder buffer = new StringBuilder();
80. for (int i = 0; i < numbits; i++) {
81. buffer.append( (lonbits.get(i))?'1':'0');
82. buffer.append( (latbits.get(i))?'1':'0');
83. }
84. return base32(Long.parseLong(buffer.toString(), 2));
85. }
87. private BitSet getBits(double lat, double floor, double ceiling) {
88. BitSet buffer = new BitSet(numbits);
89. for (int i = 0; i < numbits; i++) {
90. double mid = (floor + ceiling) / 2;
91. if (lat >= mid) {
92. buffer.set(i);
93. floor = mid;
94. } else {
95. ceiling = mid;
96. }
97. }
98. return buffer;
99. }
101. public static String base32(long i) {
102. char[] buf = new char[65];
103. int charPos = 64;
104. boolean negative = (i < 0);
105. if (!negative)
106. i = -i;
107. while (i <= -32) {
108. buf[charPos--] = digits[(int) (-(i % 32))];
109. i /= 32;
110. }
111. buf[charPos] = digits[(int) (-i)];
113. if (negative)
114. buf[--charPos] = '-';
115. return new String(buf, charPos, (65 - charPos));
116. }
118. }

**C#：**

**[csharp]** [view plaincopy](http://blog.csdn.net/wangxiafghj/article/details/9014363)

1. using System;
3. namespace sharonjl.utils
4. {
5. public static class Geohash
6. {
7. #region Direction enum
9. public enum Direction
10. {
11. Top = 0,
12. Right = 1,
13. Bottom = 2,
14. Left = 3
15. }
17. #endregion
19. private const string Base32 = "0123456789bcdefghjkmnpqrstuvwxyz";
20. private static readonly int[] Bits = new[] {16, 8, 4, 2, 1};
22. private static readonly string[][] Neighbors = {
23. new[]
24. {
25. "p0r21436x8zb9dcf5h7kjnmqesgutwvy", // Top
26. "bc01fg45238967deuvhjyznpkmstqrwx", // Right
27. "14365h7k9dcfesgujnmqp0r2twvyx8zb", // Bottom
28. "238967debc01fg45kmstqrwxuvhjyznp", // Left
29. }, new[]
30. {
31. "bc01fg45238967deuvhjyznpkmstqrwx", // Top
32. "p0r21436x8zb9dcf5h7kjnmqesgutwvy", // Right
33. "238967debc01fg45kmstqrwxuvhjyznp", // Bottom
34. "14365h7k9dcfesgujnmqp0r2twvyx8zb", // Left
35. }
36. };
38. private static readonly string[][] Borders = {
39. new[] {"prxz", "bcfguvyz", "028b", "0145hjnp"},
40. new[] {"bcfguvyz", "prxz", "0145hjnp", "028b"}
41. };
43. public static String CalculateAdjacent(String hash, Direction direction)
44. {
45. hash = hash.ToLower();
47. char lastChr = hash[hash.Length - 1];
48. int type = hash.Length%2;
49. var dir = (int) direction;
50. string nHash = hash.Substring(0, hash.Length - 1);
52. if (Borders[type][dir].IndexOf(lastChr) != -1)
53. {
54. nHash = CalculateAdjacent(nHash, (Direction) dir);
55. }
56. return nHash + Base32[Neighbors[type][dir].IndexOf(lastChr)];
57. }
59. public static void RefineInterval(ref double[] interval, int cd, int mask)
60. {
61. if ((cd & mask) != 0)
62. {
63. interval[0] = (interval[0] + interval[1])/2;
64. }
65. else
66. {
67. interval[1] = (interval[0] + interval[1])/2;
68. }
69. }
71. public static double[] Decode(String geohash)
72. {
73. bool even = true;
74. double[] lat = {-90.0, 90.0};
75. double[] lon = {-180.0, 180.0};
77. foreach (char c in geohash)
78. {
79. int cd = Base32.IndexOf(c);
80. for (int j = 0; j < 5; j++)
81. {
82. int mask = Bits[j];
83. if (even)
84. {
85. RefineInterval(ref lon, cd, mask);
86. }
87. else
88. {
89. RefineInterval(ref lat, cd, mask);
90. }
91. even = !even;
92. }
93. }
95. return new[] {(lat[0] + lat[1])/2, (lon[0] + lon[1])/2};
96. }
98. public static String Encode(double latitude, double longitude, int precision = 12)
99. {
100. bool even = true;
101. int bit = 0;
102. int ch = 0;
103. string geohash = "";
105. double[] lat = {-90.0, 90.0};
106. double[] lon = {-180.0, 180.0};
108. if (precision < 1 || precision > 20) precision = 12;
110. while (geohash.Length < precision)
111. {
112. double mid;
114. if (even)
115. {
116. mid = (lon[0] + lon[1])/2;
117. if (longitude > mid)
118. {
119. ch |= Bits[bit];
120. lon[0] = mid;
121. }
122. else
123. lon[1] = mid;
124. }
125. else
126. {
127. mid = (lat[0] + lat[1])/2;
128. if (latitude > mid)
129. {
130. ch |= Bits[bit];
131. lat[0] = mid;
132. }
133. else
134. lat[1] = mid;
135. }
137. even = !even;
138. if (bit < 4)
139. bit++;
140. else
141. {
142. geohash += Base32[ch];
143. bit = 0;
144. ch = 0;
145. }
146. }
147. return geohash;
148. }
149. }
150. }

**C#代码来自：https://github.com/sharonjl/geohash-net**

[geohash演示](http://openlocation.org/geohash/geohash-js/)：http://openlocation.org/geohash/geohash-js/

**观点讨论**

引用阿里云以为技术专家的博客上的讨论：

1.两个离的越近，geohash的结果相同的位数越多，对么？  
这一点是有些用户对geohash的误解，虽然geo确实尽可能的将位置相近的点hash到了一起，可是这并不是严格意义上的(实际上也并不可能，因为毕竟多一维坐标)，  
例如在方格4的左下部分的点和大方格1的右下部分的点离的很近，可是它们的geohash值一定是相差的相当远，因为头一次的分块就相差太大了，很多时候 我们对geohash的值进行简单的排序比较，结果貌似真的能够找出相近的点，并且似乎还是按照距离的远近排列的，可是实际上会有一些点被漏掉了。  
上述这个问题，可以通过搜索一个格子，周围八个格子的数据，统一获取后再进行过滤。这样就在编码层次解决了这个问题。  
2.既然不能做到将相近的点hash值也相近，那么geohash的意义何在呢？  
我觉得geohash还是相当有用的一个算法，毕竟这个算法通过无穷的细分，能确保将每一个小块的geohash值确保在一定的范围之内，这样就为灵活的周边查找和范围查找提供了可能。

常见的一些应用场景

A、如果想查询附近的点？如何操作

查出改点的gehash值，然后到数据库里面进行前缀匹配就可以了。

B、如果想查询附近点，特定范围内，例如一个点周围500米的点，如何搞？

可以查询结果，在结果中进行赛选，将geohash进行解码为经纬度，然后进行比较

 \*在纬度相等的情况下：

 \*经度每隔0.00001度，距离相差约1米；

 \*每隔0.0001度，距离相差约10米；

 \*每隔0.001度，距离相差约100米；

 \*每隔0.01度，距离相差约1000米；

 \*每隔0.1度，距离相差约10000米。

 \*在经度相等的情况下：

 \*纬度每隔0.00001度，距离相差约1.1米；

 \*每隔0.0001度，距离相差约11米；

 \*每隔0.001度，距离相差约111米；

 \*每隔0.01度，距离相差约1113米；

 \*每隔0.1度，距离相差约11132米。

参考资料：

<http://iamzhongyong.iteye.com/blog/1399333>

<http://tech.idv2.com/2011/06/17/location-search/>

<http://blog.sina.com.cn/s/blog_62ba0fdd0100tul4.html>

来源： <<http://blog.csdn.net/wangxiafghj/article/details/9014363>>