

## 13 | 空间检索（上）：如何用Geohash实现“查找附近的人”功能？

2020-04-24 陈东

检索技术核心20讲

[进入课程](#) >




讲述：陈东

时长 20:10 大小 18.48M



你好，我是陈东。

现在，越来越多的互联网应用在提供基于地理位置的服务。这些基于地理位置服务，本质上都是检索附近的人或者物的服务。比如说，社交软件可以浏览附近的人，餐饮平台可以查找附近的餐厅，还有出行平台会显示附近的车等。那如果你的老板希望你能为公司的应用开发相关的功能，比如说实现一个“查询附近的人”功能，你会怎么做呢？

一个很容易想到的方案是，把所有人的坐标取出来，计算每个人和自己当前坐标的距离， 后把它们全排序，并且根据距离远近在地图上列出来。但是仔细想想你就会发现，这种方案在大规模的系统中并不可行。

这是因为，如果系统中的人数到达了一定的量级，那计算和所有人的距离再排序，这会是一个非常巨大的代价。尽管，我们可以使用堆排序代替全排序来降低排序代价，但取出所有人的位置信息并计算距离，这本身就是一个很大的开销。

那在大规模系统中实现“查找附近的人功能”，我们有什么更高效的检索方案呢？今天我们就来聊聊这个问题。

## 使用非精准检索的思路实现“查找附近的人”

事实上，“查找附近的人”和“检索相关的网页”这两个功能的本质是非常相似的。在这两个功能的实现中，我们都没有明确的检索目标，也就都不需要非常精准的检索结果，只需要保证质量足够高的结果包含在 Top K 个结果中就够了。所以，非精准 Top K 检索也可以作为优化方案，来实现“查找附近的人”功能。那具体是如何实现的呢？

我们可以通过限定“附近”的范围来减少检索空间。一般来说，同一个城市的人往往会比不同城市的人距离更近。所以，我们不需要去查询所有的人，只需要去查询自己所在城市的人，然后计算出自己和他们的距离就可以了，这样就能大大缩小检索范围了。那在同一个城市中，我们也可以优先检索同一个区的用户，来再次缩小检索范围。这就是**非精准检索的思路**了。

在这种限定“附近”区域的检索方案中，为了进一步提高检索效率，我们可以将所有的检索空间划分为多个区域并做好编号，然后以区域编号为 key 做好索引。这样，当我们需要查询附近的人时，先快速查询到自己所属的区域，然后再将该区域中所有人的位置取出，计算和每一个人的距离就可以了。在这个过程中，划分检索空间以及对其编号是最关键的一步，那具体怎么操作呢？我们接着往下看。

## 如何对区域进行划分和编号？

对于一个完整的二维空间，我们可以用二分的思想将它均匀划分。也就是在水平方向上一分为二，在垂直方向上也一分为二。这样一个空间就会被均匀地划分为四个子空间，这四个子空间，我们可以用两个比特位来编号。在水平方向上，我们用 0 来表示左边的区域，用 1 来表示右边的区域；在垂直方向上，我们用 0 来表示下面的区域，用 1 来表示上面的区域。因此，这四个区域，从左下角开始按照顺时针的顺序，分别是 00、01、11 和 10。



## 区域划分和编号

接下来，如果要继续划分空间，我们依然沿用这个思路，将每个区域再分为四块。这样，整个空间就被划分成了 16 块区域，那对应的编号也会再增加两位。比如说，01 编号的区域被划分成了 4 小块，那这四小块的编号就是在 01 后面追加两位编码，分别为 01 00、01 01、01 10、01 11。依次类推，我们可以将整个空间持续细分。具体划分到什么粒度，就取决于应用对于“附近”的定义和需求了。

这种区域编码的方式有 2 个优点：

1. 区域有层次关系：如果两个区域的前缀是相同的，说明它们属于同一个大区域；
2. 区域编码带有分割意义：奇数位的编号代表了垂直切分，偶数位的编号代表了水平切分，这会方便区域编码的计算（奇偶位是从右边以第 0 位开始数起的）。

## 如何快速查询同个区域的人？

那有了这样的区域编码方式以后，我们该怎么查询呢？这就要说到区域编码的一个特点了：**区域编码能将二维空间的两个维度用一维编码表示**。利用这个特点，我们就可以使用一维空间中常见的检索技术快速查找了。我们可以将区域编码作为 key，用有序数组存储，这样就可以用二分查找进行检索了。

如果有效区域动态增加，那我们还可以使用二叉检索树、跳表等检索技术来索引。在一些系统的实现中，比如 Redis，它就可以直接支持类似的地理位置编码的存入和检索，内部的实现方式是，使用跳表按照区域编码进行排序和查找。此外，如果希望检索效率更高，我们还可以使用哈希表来实现区域的查询。

这样一来，当我们想要查询附近的人时，只需要根据自己的坐标，计算出自己所属区域的编码，然后在索引中查询出所有属于该区域的用户，计算这些用户和自己的距离，最后排序展现即可。

不过，这种非精准检索的方案，会带来一定的误差。也就是说，我们找到的所谓“附近的人”，其实只是和你同一区域的人而已，并不一定是离你最近的。比如说，你的位置正好处于一个区域的边缘，那离你最近的人，也可能是在你的邻接区域里。



邻接区域距离可能更近

好在，在“查找附近的人”这类目的性不明确的应用中，这样的误差我们也是可以接受的。但是，在另一些有精准查询需求的应用中，是不允许存在这类误差的。比如说，在游戏场景中，角色技能的攻击范围必须是精准的，它要求技能覆盖范围内的所有敌人都应该受到伤害，不能有遗漏。那这是怎么做到的呢？你可以先想一想，然后再来看我的分析。

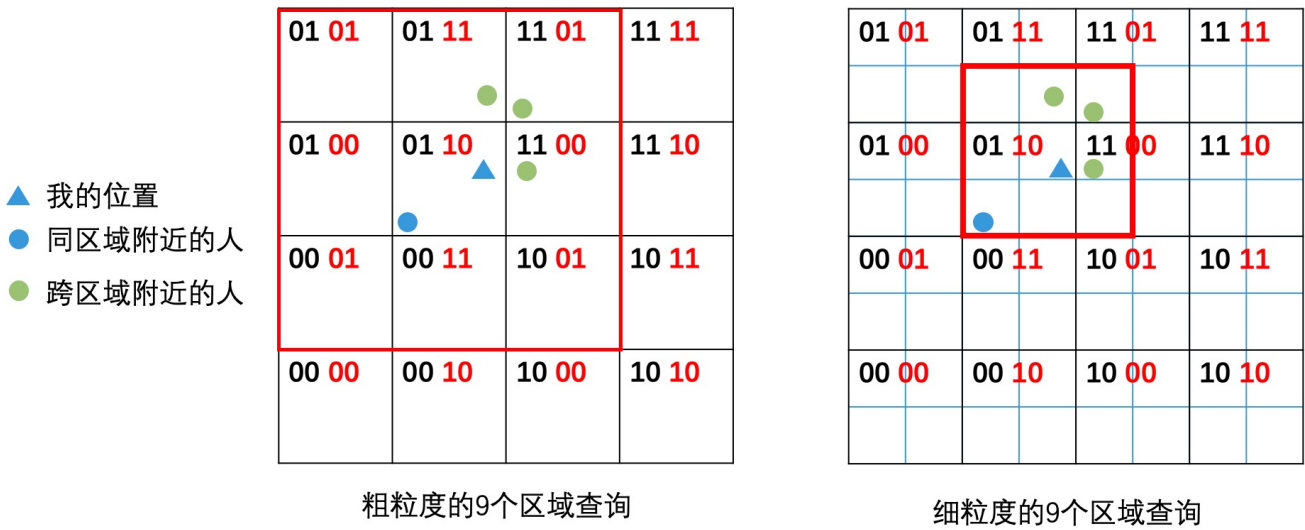
### 如何精准查询附近的人？



既然邻接区域的人距离我们更近，那我们是不是可以建立一个更大的候选集合，把这些邻接区域的用户都加进去，再一起计算距离和排序，这样问题是不是就解决了呢？我们先试着操作一下。

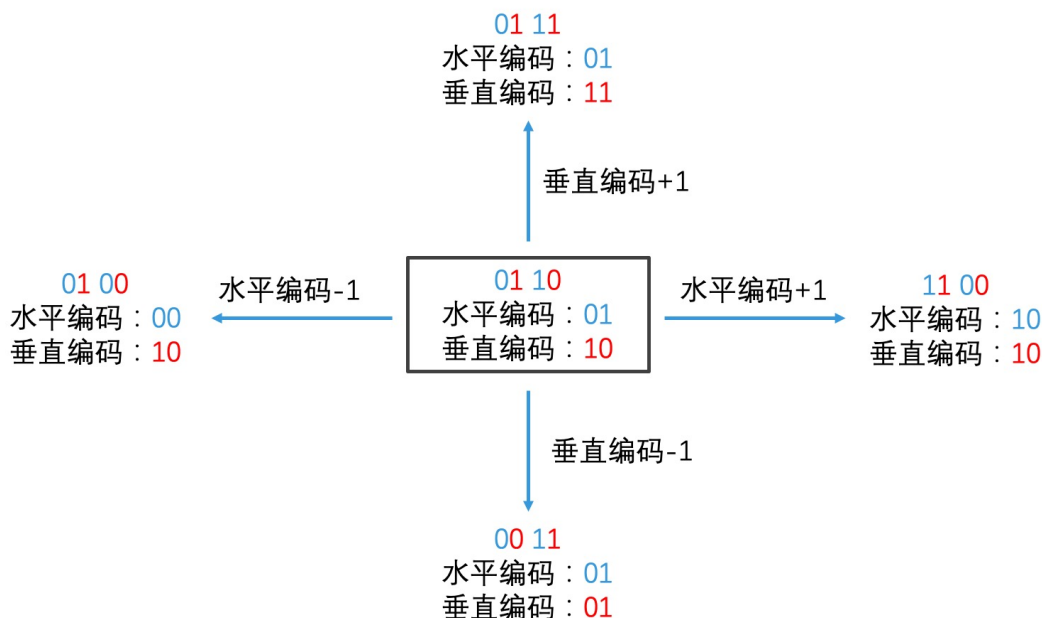
对于目标所在的当前区域，我们可以根据期望的查询半径，以当前区域为中心向周围扩散，从而将周围的区域都包含进来。假设，查询半径正好是一个区域边长的一半，那我们只要将目标区域周围一圈，也就是 8 个邻接区域中的用户都加入候选集，这就肯定不会有遗漏了。这时，虽然计算量提高了 8 倍，但我们可以给出精准的解了。

如果要降低计算量，我们可以将区域划分的粒度提高一个量级。这样，区域的划分就更精准，在查询半径不变的情况下，需要检索的用户数量就会更少（查询范围对比见下图两个红框部分）。



更细粒度地划分区域

知道了要查询的区域有哪些，那我们怎么快速寻找这些区域的编码呢？这就要回到我们区域编码的方案本身了。前面我们说了，区域编码可以根据奇偶位拆成水平编码和垂直编码这两块，如果一个区域编码是 0110，那它的水平编码就是 01，垂直编码就是 10。那该区域右边一个区域的水平编码的值就比它自己的大 1，垂直编码则相同。因此，**我们通过分解出当前区域的水平编码和垂直编码，对对应的编码值进行加 1 或者减 1 的操作，就能得到不同方向上邻接的 8 个区域的编码了。**



区域编码规则

以上，就是精准查询附近人的检索过程，我们可以总结为两步：第一步，先查询出自己所属的区域编码，再计算出周围 8 个邻接区域区域编码；第二步，在索引中查询 9 次，取出所有属于这些区域中的人，精准计算每一个人和自己的距离，最后排序输出结果。

## 什么是 Geohash 编码？

说到这，你可能会会有疑问了，在实际工作中，用户对应的都是实际的地理位置坐标，那它和二维空间的区域编码又是怎么联系起来的呢？别着急，我们慢慢说。

实际上，我们会将地球看作是一个大的二维空间，那经纬度就是水平和垂直的两个切分方向。在给出一个用户的经纬度坐标之后，我们通过对地球的经纬度区间不断二分，就能得到这个用户所属的区域编码了。这么说可能比较抽象，我来举个例子。

我们知道，地球的纬度区间是 $[-90, 90]$ ，经度是 $[-180, 180]$ 。如果给出的用户纬度（垂直方向）坐标是 39.983429，经度（水平方向）坐标是 116.490273，那我们求这个用户所属的区域编码的过程，就可以总结为 3 步：

1. 在纬度方向上，第一次二分，39.983429 在 $[0, 90]$ 之间， $[0, 90]$ 属于空间的上半边，因此我们得到编码 1。然后在 $[0, 90]$ 这个空间上，第二次二分，39.983429 在 $[0, 45]$ 之间，

[0,45]属于区间的下半边，因此我们得到编码 0。两次划分之后，我们得到的编码就是 10。

- 2. 在经度方向上，第一次二分，116.490273 在[0,180]之间，[0,180]属于空间的右半边，因此我们得到编码 1。然后在[0,180]这个空间上，第二次二分，116.490273 在[90,180]之间，[90,180]还是属于区间的右半边，因此我们得到的编码还是 1。两次划分之后，我们得到的编码就是 11。
- 3. 我们把纬度的编码和经度的编码交叉组合起来，先是经度，再是纬度。这样就构成了区域编码，区域编码为 1110。

你会发现，在上面的例子中，我们只二分了两次。实际上，如果区域划分的粒度非常细，我们就要持续、多次二分。而每多二分一次，我们就需要增加一个比特位来表示编码。如果经度和纬度各二分 15 次的话，那我们就需要 30 个比特位来表示一个位置的编码。那上面例子中的编码就会是 11100 11101 00100 01111 00110 11110。



计算编码的过程示意图

这样得到的编码会特别长，那为了简化编码的表示，我们可以以 5 个比特位为一个单位，把长编码转为 base32 编码，最终得到的就是 wx4g6y。这样 30 个比特位，我们只需要用 6 个字符就可以表示了。

这样做不仅存储会更简单，而且具有相同前缀的区域属于同一个大区域，看起来也非常直观。**这种将经纬度坐标转换为字符串的编码方式，就叫作 Geohash 编码。**大多数应用都会使用 Geohash 编码进行地理位置的表示，以及在很多系统中，比如，Redis、MySQL 以及 Elastic Search 中，也都支持 Geohash 数据的存储和查询。

十进制 编码	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
base 32 编码	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
十进制 编码	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
base 32 编码	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z



十进制转为base32编码字符对照表

那在实际转换的过程中，由于不同长度的 Geohash 代表不同大小的覆盖区域，因此我们可以结合 GeoHash 字符长度和覆盖区域对照表，根据自己的应用需要选择合适的 Geohash 编码长度。这个对照表让我们在使用 Geohash 编码的时候方便很多。

Geohash编码长度	宽度	高度
1	5009.4km	4992.6km
2	1252.3km	624.1km
3	156.5km	156km
4	39.1km	19.5km
5	4.9km	4.9km
6	1.2km	609.4m
7	152.9m	152.4m
8	38.2m	19m
9	4.8m	4.8m
10	1.2m	59.5cm
11	14.9cm	14.9cm
12	3.7cm	1.9cm



字符长度和覆盖区域对照表



不过，Geohash 编码也有缺点。由于 Geohash 编码的一个字符就代表了 5 个比特位，因此每当字符长度变化一个单位，区域的覆盖度变化跨度就是 32 倍 ( $2^5$ )，这会导致区域范围划分不够精细。

因此，当发现粒度划分不符合自己应用的需求时，我们其实可以将 Geohash 编码转换回二进制编码的表示方式。这样，编码长度变化的单位就是 1 个比特位了，区域覆盖度变化跨度就是 2 倍，我们就可以更灵活地调整自己期望的区域覆盖度了。实际上，在许多系统的底层实现中，虽然都支持以字符串形式输入 Geohash 编码，但是在内存中的存储和计算都是以二进制的方式来进行的。

## 重点回顾

今天，我们重点学习了利用空间检索的技术来查找附近的人。

首先，我们通过将二维空间在水平和垂直方向上不停二分，可以生成一维的区域编码，然后我们可以使用一维空间的检索技术对区域编码做好索引。

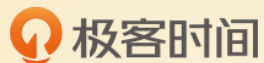
在查询时，我们可以使用非精准的检索思路，直接检索相应的区域编码，就可以查找到“附近的人”了。但如果要进行精准检索，我们就需要根据检索半径将扩大检索范围，一并检索周边的区域，然后将所有的检索结果进行精确的距离计算，最终给出整体排序。这也是一个典型的“非精准 Top K 检索 - 精准 Top K 检索”的应用案例。因此，当你需要基于地理位置，进行查找或推荐服务的开发时，可以根据具体需求，灵活使用今天学习到的检索方案。

此外，我们还学习了 Geohash 编码，Geohash 编码是很常见的一种编码方式，它将真实世界的地理位置根据经纬度进行区域编码，再使用 base32 编码生成一维的字符串编码，使得区域编码在显示和存储上都更加方便。

## 课堂讨论

1. 如果一个应用期望支持“查找附近的人”的功能。在初期用户量不大的时候，我们使用什么索引技术比较合理？在后期用户量大的时候，为了加快检索效率，我们又可以采用什么检索技术？为什么？
2. 如果之前的应用选择了 5 个字符串的 Geohash 编码，进行区域划分（区域范围为  $4.9 \text{ km} * 4.9 \text{ km}$ ），那当我们想查询 10 公里内的人，这个时候该如何进行查询呢？使用什么索引技术会比较合适呢？

欢迎在留言区畅所欲言，说出你的思考过程和最终答案。如果有收获，也欢迎把这一讲分享给你的朋友。



# 检索技术核心 20 讲

从搜索引擎到推荐引擎，带你吃透检索

陈东

奇虎 360 商业产品事业部  
资深总监



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 12 | 非精准Top K检索：如何给检索结果的排序过程装上“加速器”？

下一篇 14 | 空间检索（下）：“查找最近的加油站”和“查找附近的人”有何不同？

## 精选留言 (6)

写留言



一步

2020-04-25

1:

用户量不大的时候：直接可以进行计算具体，这里因为用户的数据是一致在变化的，所以保存的数据结构可以使用 树 和 跳表，都可以在  $\log(n)$  的时间复杂度中进行查询

用户量很大的时候，可以使用倒排索引，以区域的 key 建 倒排索引

2: 这里可以利用区域编码的特性，在同一大区域下的小区域的前缀是一样的...

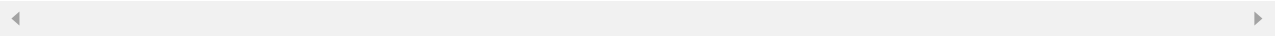
展开

作者回复: 回答得很清晰！

第一个问题，树和跳表都可以，都支持动态修改，并且支持范围查询，可以在 $\log n$ 时间复杂度内

完成查询。如果用户量变大，又希望查询效率高。可以用倒排索引。

第二个问题，查询周边两圈就可以了。这样相信你能更好地理解Geohash的原理。当然，还有更多灵活的查询方式，我们下一讲再介绍。

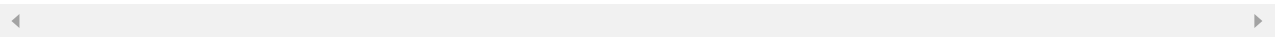


范闲

2020-04-27

- 1.小数据量怎么做都可以.大数据量倒排索引+跳表，倒排加速
- 2.扩大查询范围，可以将GeoHash的编码减少一位。如果一开始是6位，可以变成5位去查。

作者回复: 回答得很简明扼要。重点把握得不错。



一步

2020-04-25

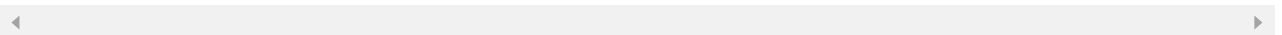
这里 GeoHash 编码字符串的长度和 选定几个比特位作为一个单元和编码方式有关

这里是默认 选5个比特位作为一个单元 和 使用 base32 进行编码吗？

展开 ∨

作者回复: 是的。字符串长度和选择几个比特位作为编码单元是相关的。

一般来说，对于Geohash，我们选择5个比特位作为一个单元，使用base32编码。



qinsi

2020-04-24

把地球的球体表面投影到平面的话，相同Geohash编码长度对应的覆盖区域的大小会随着纬度高低变化的吧

作者回复: 你这个问题很好。的确是这样的，由于地球是球面，直接用经纬度划分格子的话，高纬度地区和低纬度地区的一个格子的范围是不同的。这里其实存在一个误差。因此，真要精准估计区域范围的话，我们应该每隔一个经度或者纬度就累加一个偏差值才对。

不过，在查找附近的人的这类需求上，本来就是非精准检索，因此这个范围表更多是一个参考。帮助我们快速地决定需要使用的Geohash的长度。如果对精度有要求的话，可以按照按文中的思路，扩大范围，减少一个编码长度，取出更大范围的候选集进行精准计算。



那一刻

2020-04-24

问题1. 如老师讲过的，在用户量不大的情况下可以采用堆排序对全部用户排序，得到结果。而当用户量增大的时候，需要对二维空间进行编码的方式来解决。区别在于数据量小的情况堆排序可以解决问题，而且不需要额外的空间。当时数据量大的情况，时间复杂度增加，需要来增加空间复杂度，也就是对于空间位置进行二分编码的方式，来降低时间复杂度。...

展开 ▾

作者回复: 问题1: 数据量小的时候怎么做都行。如果出于节约空间的角度出发，用数组和链表(跳表)都可以。这样也可以做二分查找，效率也不算差。当数据量大的时候，如果觉得二分查找还不够快，那么还可以建立倒排索引，直接以编码作为key。

问题2: 不错，我们是扩展周边区域就可以了。不过如果只扩展周围一圈的邻接区域，那么扩展半径只是4.9公里。如果我希望的扩展半径是10公里，你会怎么做呢？



夜空中最亮的星 (华仔...)

2020-04-24

干货满满，很受益

展开 ▾

💬 1

