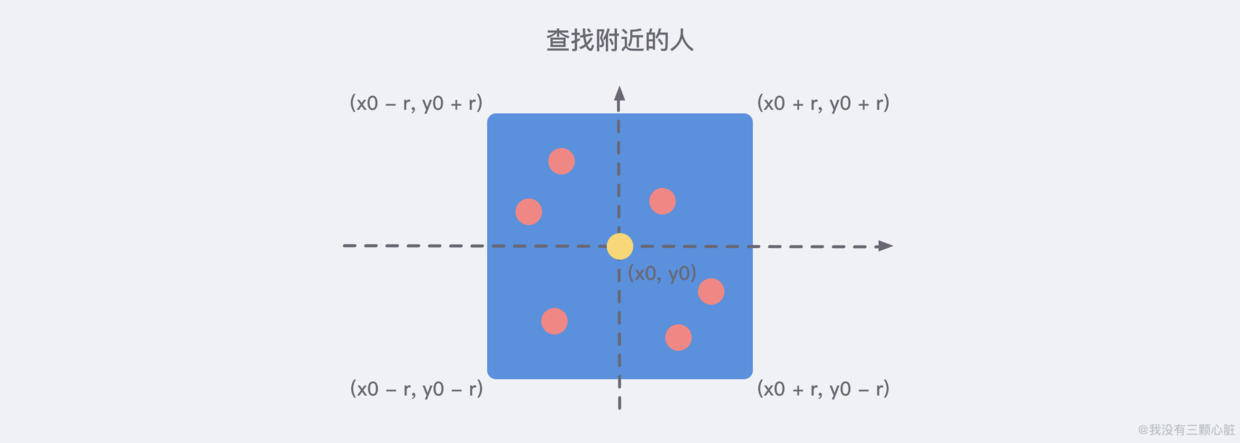
像微信“附近的人”，美团“附近的餐厅”，支付宝共享单车“附近的车”是怎么设计实现的呢？

一、使用数据库实现查找附近的人

我们都知道，地球上的任何一个位置都可以使用二维的经纬度来表示，经度范围 [-180, 180]，纬度范围 \*[-90, 90]，纬度正负以赤道为界，北正南负，经度正负以本初子午线 \*(英国格林尼治天文台) 为界，东正西负。比如说，北京人民英雄纪念碑的经纬度坐标就是(39.904610, 116.397724)，都是正数，因为中国位于东北半球。

所以，当我们使用数据库存储了所有人的经纬度信息之后，我们就可以基于当前的坐标节点，来划分出一个矩形的范围，来得知附近的人，如下图：



伪SQL语句：

SELECT id FROM positions WHERE x0 - r < x < x0 + r AND y0 - r < y < y0 + r

如果还想进一步地知道与每个坐标元素的距离并排序的话，就需要一定的计算。

当两个坐标元素的距离不是很远的时候，可以简单利用勾股定理就能够得出他们之间的距离。不过需要注意的是，地球不是一个标准的球体，经纬度的密度 是不一样的，所以我们使用勾股定理计算平方之后再求和时，需要按照一定的系数加权再进行求和。当然，如果不准求精确的话，加权也不必了。

优化之后的SQL语句：

SELECT

\*

FROM

users\_location

WHERE

latitude > '.$lat.' - 1

AND latitude < '.$lat.' + 1 AND longitude > '.$lon.' - 1

AND longitude < '.$lon.' + 1

ORDER BY

ACOS(

SIN( ( '.$lat.' \* 3.1415 ) / 180 )

\* SIN( ( latitude \* 3.1415 ) / 180 )

+ COS( ( '.$lat.' \* 3.1415 ) / 180 )

\* COS( ( latitude \* 3.1415 ) / 180 ) \* COS( ( '.$lon.' \* 3.1415 ) / 180

- ( longitude \* 3.1415 ) / 180 )

) \* 6380 ASC

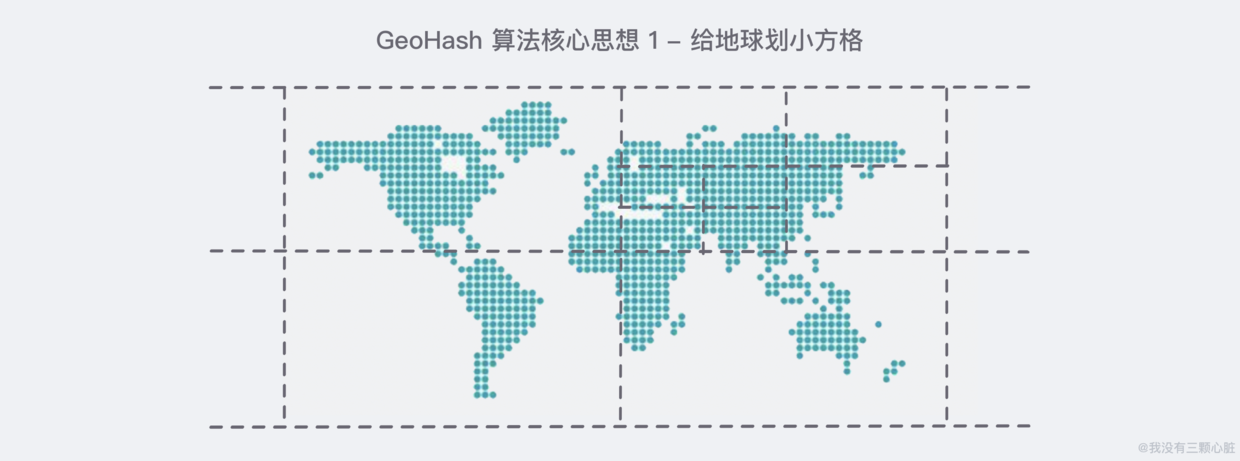
LIMIT 10 ';

为了满足高性能的矩形区域算法，数据表也需要把经纬度坐标加上双向复合索引(x, y)，这样可以满足最大优化查询性能。

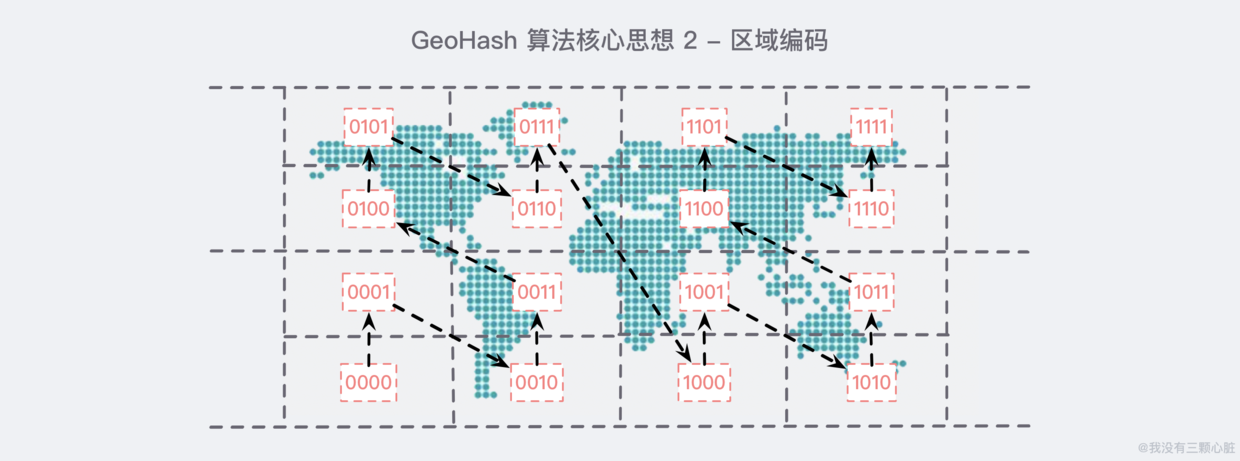
二、GeoHash算法简述

这是业界比较通用的，用于地理位置距离排序的一个算法，Redis也采用了这样的算法。GeoHash算法将二维的经纬度数据映射到一维的整数，这样所有的元素都将在挂载到一条线上，距离靠近的二维坐标映射到一维后的点之间距离也会很接近。当我们想要计算「附近的人时」，首先将目标位置映射到这条线上，然后在这个一维的线上获取附近的点就行了。

它的核心思想就是把整个地球看成是一个二维的平面，然后把这个平面不断地等分成一个一个小的方格，每一个坐标元素都位于其中的唯一一个方格中，等分之后的方格越小，那么坐标也就越精确，类似下图：



经过划分的地球，需要对其进行编码：



经过这样顺序的编码之后，会发现一些规律：

* 横着的所有编码中，第2位和第4位都是一样的，例如第一排第一个0101和第二个0111，他们的第2位和第4位都是1；
* 竖着的所有编码中，第1位和第3位是递增的，例如第一排第一个0101，如果单独把第1位和第3位拎出来的话，那就是00，同理看第一排第二个 0111，同样的方法第1位和第3位拎出来是01，刚好是00递增一个；

通过这样的规律我们就把每一个小方块儿进行了一定顺序的编码，这样做的好处是显而易见的：每一个元素坐标既能够被唯一标识在这张被编码的地图上，也不至于暴露特别的具体的位置，因为区域是共享的，我可以告诉你我就在公园附近，但是在具体的哪个地方你就无从得知了。

总之，我们通过上面的思想，能够把任意坐标变成一串二进制的编码了，类似于11010010110001000100这样(注意经度和维度是交替出现的哦..)，通过这个整数我们就可以还原出元素的坐标，整数越长，还原出来的坐标值的损失程序就越小。对于"附近的人"这个功能来说，损失的一点经度可以忽略不计。

最后就是一个Base32(09, az, 去掉 a/i/l/o 四个字母)的编码操作，让它变成一个字符串，例如上面那一串儿就变成了wx4g0ec1。

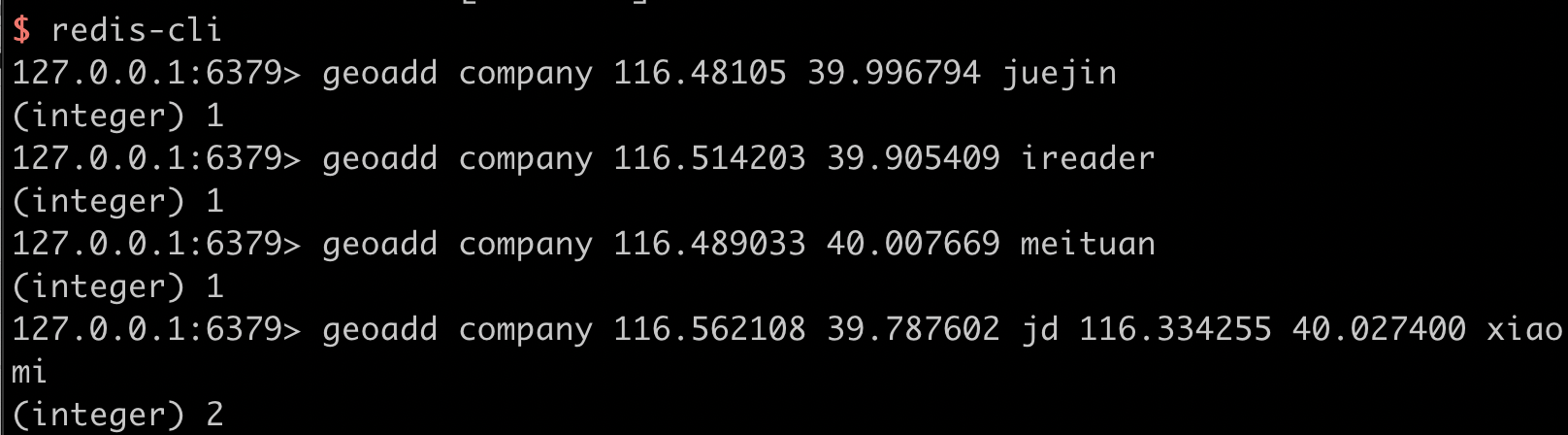
在 Redis 中，经纬度使用52位的整数进行编码，放进了zset里面，zset的 value是元素的key，score是GeoHash的52位整数值。zset的score虽然是浮点数，但是对于 52 位的整数值来说，它可以无损存储。

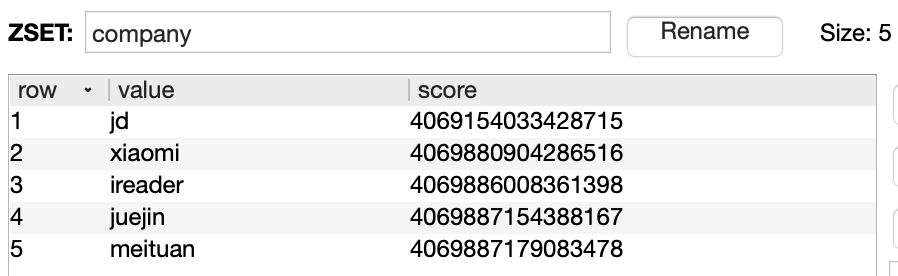
三、在Redis中使用Geo

在使用Redis进行Geo查询时，我们要时刻想到它的内部结构实际上只是一个zset(skiplist)。通过zset的score排序就可以得到坐标附近的其他元素（实际情况要复杂一些，可以这样理解），通过将score还原成坐标值就可以得到元素的原始坐标了。

Redis提供的Geo指令有6个：

#### 1、增加（geoadd）





Redis没有直接提供Geo的删除指令，但是我们可以通过zset相关的指令来操作Geo数据，所以元素删除可以使用zrem指令即可。

#### 2、距离（geodist）

geodist指令可以用来计算两个元素之间的距离，携带集合名称、2个名称和距离单位。

127.0.0.1:6379> geodist company juejin ireader km

"10.5501"

127.0.0.1:6379> geodist company juejin juejin km

"0.0000"

距离单位可以是m、km、ml（英里）、ft（尺）

#### 3、获取元素位置（geopos）

geopos指令可以获取集合中任意元素的经纬度坐标，可以一次获取多个。

127.0.0.1:6379> geopos company juejin

1) 1) "116.48104995489120483"

2) "39.99679348858259686"

可以看到获取的经纬度坐标和geoadd进去的坐标有轻微的误差，原因是Geohash对二维坐标进行的一维映射是有损的，通过映射在还原回来的值会出现较小的差别。对于【附近的人】这种功能来说，这点误差根本不是事。

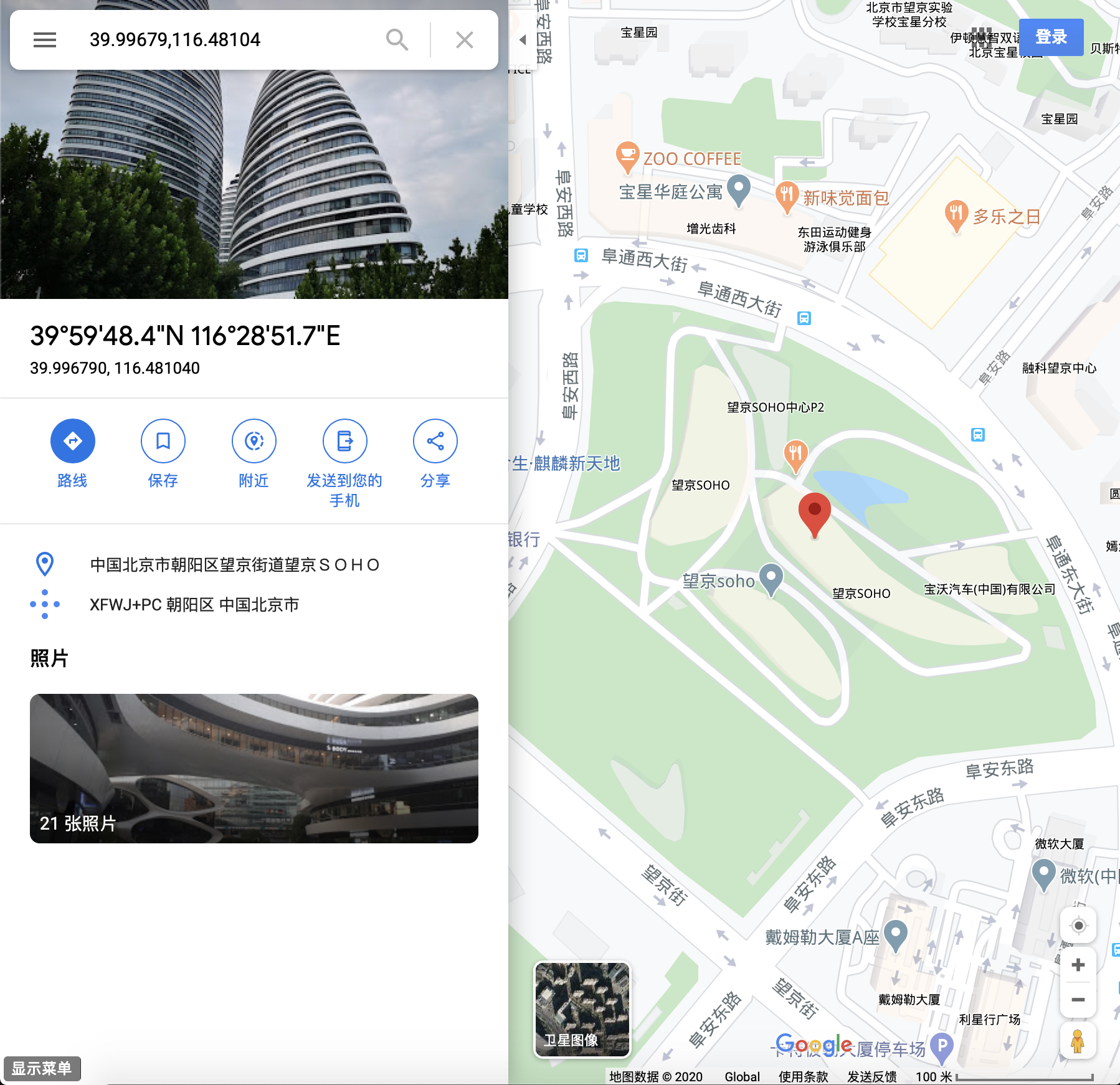
#### 4、获取元素的hash值（geohash）

geohash可以获取元素的经纬度编码字符串，它是base32编码。可以使用这个编码值去[http://geohash.org/${hash}](http://geohash.org/$%7bhash%7d)中进行直接定位，它是Geohash的标准编码值。

127.0.0.1:6379> geohash company juejin

1) "wx4gd94yjn0"

打开地址<http://geohash.org/wx4gd94yjin0>，观察地图指向的位置是否正确：



#### 5、附近的公司

georadiusbymember指令是最为关键的指令，它可以用来查询指定元素附近的其他元素，它的参数非常复杂。

# 范围 20 公里以内最多 3 个元素按距离正排，它不会排除自身

127.0.0.1:6379> georadiusbymember company ireader 20 km count 3 asc

1) "ireader"

2) "juejin"

3) "meituan"

# 范围 20 公里以内最多 3 个元素按距离倒排

127.0.0.1:6379> georadiusbymember company ireader 20 km count 3 desc

1) "jd"

2) "meituan"

3) "juejin"

# 三个可选参数 withcoord withdist withhash 用来携带附加参数

# withdist 很有用，它可以用来显示距离

127.0.0.1:6379> georadiusbymember company ireader 20 km withcoord withdist withhash count 3 asc

1) 1) "ireader"

2) "0.0000"

3) (integer) 4069886008361398

4) 1) "116.5142020583152771"

2) "39.90540918662494363"

2) 1) "juejin"

2) "10.5501"

3) (integer) 4069887154388167

4) 1) "116.48104995489120483"

2) "39.99679348858259686"

3) 1) "meituan"

2) "11.5748"

3) (integer) 4069887179083478

4) 1) "116.48903220891952515"

2) "40.00766997707732031"

Redis还提供了根据坐标值来查询附近的元素，这个指令更加有用，它可以根据用户的定位来计算【附近的车】，【附近的餐馆】等，它的参数和georadiusbymember基本一致，除了将目标元素改成经纬度坐标值：

127.0.0.1:6379> georadius company 116.514202 39.905409 20 km withdist count 3 asc

1) 1) "ireader"

2) "0.0000"

2) 1) "juejin"

2) "10.5501"

3) 1) "meituan"

2) "11.5748"

四、注意事项

在一个地图应用中，车的数据、餐馆的数据、人的数据可能会有百万千万条，如果使用Redis的Geo数据结构，它们将全部放在一个zset集合中。在Redis 的集群环境中，集合可能会从一个节点迁移到另一个节点，如果单个key的数据过大，会对集群的迁移工作造成较大的影响，在集群环境中单个key对应的数据量不宜超过1M，否则会导致集群迁移出现卡顿现象，影响线上服务的正常运行。

所以，这里建议Geo的数据使用单独的Redis实例部署，不使用集群环境。如果数据量过亿甚至更大，就需要对 Geo 数据进行拆分，按国家拆分、按省拆分，按市拆分，在人口特大城市甚至可以按区拆分。这样就可以显著降低单个 zset 集合的大小。

如果数据量过亿甚至更大，就需要对 Geo 数据进行拆分，按国家拆分、按省拆分，按市拆分，在人口特大城市甚至可以按区拆分。这样就可以显著降低单个 zset 集合的大小。