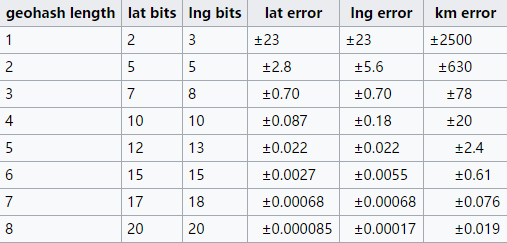
### **认识GeoHash**

　GeoHash算法将二维经纬度坐标直接转换成字符串，每一个字符串代表一个矩形区域，也就是说，这个矩形区域内所有的点（经纬度坐标）都共享相同的GeoHash字符串，字符串的长度越大，矩形的区域就越小，经度也就越高。字符串相似的表示距离相近，这样可以利用字符串的前缀匹配来查询附近的POI信息

### **GeoHash算法的步骤**

　地球纬度区间是[-90,90]，经度区间是[-180,180]，通过区间法对经度和纬度分别进行计算，假如我们获取到的当前坐标为经度-0.12866, 纬度38.534413，以纬度为例：

1. 将纬度平均分成两个区间：[-90,0),[0,90]，成为左区间和右区间，可以判断出38.534413属于右区间，则值为1,（如果属于左区间则值为0）；
2. 接着将右区间继续划分，就变成了[0,45),[45,90],此时，38.534413属于左区间，则值为0
3. 递归上述过程，则区间的值会越来越逼近38.534413
4. 随着算法的进行，我们将会得到一个序列，序列的长度跟递归的次数有关，但是一定要保证的是经度和纬度的序列长度是一样的，我这里设置的递归长度是30，经度和纬度加起来就是60，
5. 根据算法我们最终得到经度的序列为[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0]，纬度的序列为[1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0]，然后我们根据此序列再组合一个新的序列偶数位放经度，奇数位放纬度，把2串编码组合生成新串[0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0]，实际上这个序列是一串二进制
6. 最后，我们将这个新串转换成十进制再使用0-9、b-z（去掉a, i, l, o）对这组十进制进行编码得到的字符串eyzgjnfr4p0p就是最终的GeoHash编码。下图是网上给出的不同编码长度给出的精度：

[](http://www.tianshangkun.com/images/geoHash.png)  
[](http://www.tianshangkun.com/images/geoHashLength.png)

　得到GeoHash的值之后可以同样的保存近数据库中，每次查询离我最近的数据的时候理论上来说可以根据精确度截取GeoHash的值进行模糊查询，但是这样查询是有问题的，因为你没法保证你每次查询时你的当前坐标刚好在这个矩形的正中间，如果你的坐标处在矩形的边界，那么你就无法获取其附近的数据，那么这个问题怎么解决呢？其实也很简单，以你当前所在位置的矩形为九宫格的中间格子，再获取其相邻的8个矩形。

　获取其他8个区域的GeoHash也很简单，上面的表已经给出了每一个区域内经度和纬度的宽度，那么直接加减后就可以得出周边相邻的8个格子，我这里自己写了一套GeoHash的算法，得出当前坐标的GeoHash的值以及相邻8个格子的值，代码比较多就不贴上来了，在GitHub上有一个别人写好的JAVA版本的可以直接拿来用，叫[geohash-java](https://github.com/kungfoo/geohash-java" \t "http://www.tianshangkun.com/2017/05/04/%E8%8E%B7%E5%8F%96%E6%88%91%E9%99%84%E8%BF%91%E7%9A%84%E5%95%86%E5%BA%97%E6%96%B9%E6%B3%95-%E4%BA%8C-GeoHash%E7%AE%97%E6%B3%95/_blank),貌似maven的仓库中也有

<dependency>

<groupId>ch.hsr</groupId>

<artifactId>geohash</artifactId>

<version>1.3.0</version>

</dependency>

使用的方法也很简单

GeoHash geoHash = GeoHash.withCharacterPrecision(30.259498, 120.167673, 6);//参数6 表示在一格的尺寸为宽1.2km,高609.4m

System.out.println(geoHash.toBase32());//获取当前的GeoHash的值

GeoHash[] adjacent = geoHash.getAdjacent();//获取整个九宫格的GeoHash的值

for (GeoHash hash : adjacent) {

System.out.println(hash.toBase32());

}

我们算出的九个格子的geoHash值为'wtmknk','wtmknm','wtmknt','wtmkns','wtmkne','wtmkn7','wtmkn5','wtmknh','wtmknj'

上面是我们生成查询条件，我们要查询数据库中geohash的值包含上面结果的记录，下面我们看一下Mysql怎么设计 。

Mysql 5.7 以后提供了很多内置GEO函数用来解决几何问题，其中有一个很重要的函数就是st\_geohash函数，st\_geohash接收经纬度参数和一个geohash的生成长度值，这个长度值与矩形的尺寸有关，具体查看上面的对照表，如st\_geohash(120.167673,30.259498,6) 表示在Point(120.167673,30.259498)附近1.2km,609.4m范围内

下面我们创建一个表nodes,这张表有一个geom列，类型为geometry，一个虚拟列geohash,geohash列存储的是geom中的点经过st\_geohash计算后的结果。

CREATE TABLE nodes (

geom geometry NOT NULL,

geohash VARCHAR(128) AS (st\_geohash(geom,6)) VIRTUAL,

SPATIAL KEY idx\_nodes\_geom (geom),

key idx\_nodes\_geohash(geohash)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

我们模拟一些数据，创建一个存储过程用来添加数据

create PROCEDURE aaa()

begin

DECLARE lon double;

DECLARE lat double;

DECLARE i int;

set i=0;

WHILE (i <1000) do

set lon=120.167673+rand()\*0.0001;

set lat=30.259498+RAND()\*0.0001;

insert into nodes(geom) values (ST\_GeomFromText(CONCAT('POINT(',lon,' ',lat,')')));

set i=i+1;

end while;

end;

执行存储过程 call aaa ，插入数据。

下面我们查找经纬度Point(120.167673,30.259498)附近的点

SELECT

ST\_Distance\_Sphere(Point(120.167673,30.259498), geom) as distance,

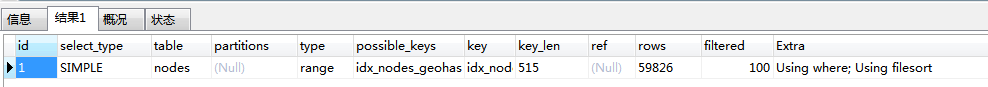
ST\_AsText(geom) geom

FROM nodes FORCE INDEX(idx\_nodes\_geohash)

WHERE geohash IN ('wtmknk','wtmknm','wtmknt','wtmkns','wtmkne','wtmkn7','wtmkn5','wtmknh','wtmknj')

ORDER BY distance LIMIT 300,400 ;

这样的话我们就获取到了某一个经纬度附近的点。我们explain一下这个查询语句:



查询使用了索引idx\_nodes\_geohash ,经测试当库中有12万条数据时，查询速度0.5秒左右