一口气说出 4种“附近的人”实现方式，面试官笑了

Original 程序员内点事 程序员内点事 2020-04-15

引言

昨天一位公众号粉丝和我讨论了一道面试题，个人觉得比较有意义，这里整理了一下分享给大家，愿小伙伴们面试路上少踩坑。面试题目比较简单：“让你实现一个附近的人功能，你有什么方案？”，这道题其实主要还是考察大家对于技术的广度，本文介绍几种方案，给大家一点思路，避免在面试过程中语塞而影响面试结果，如有不严谨之处，还望亲人们温柔指正！

“附近的人” 功能生活中是比较常用的，像外卖app附近的餐厅，共享单车app里附近的车辆。既然常用面试被问的概率就很大，所以下边依次来分析基于 mysql数据库 、 Redis 、 MongoDB 实现的 “附近的人” 功能。

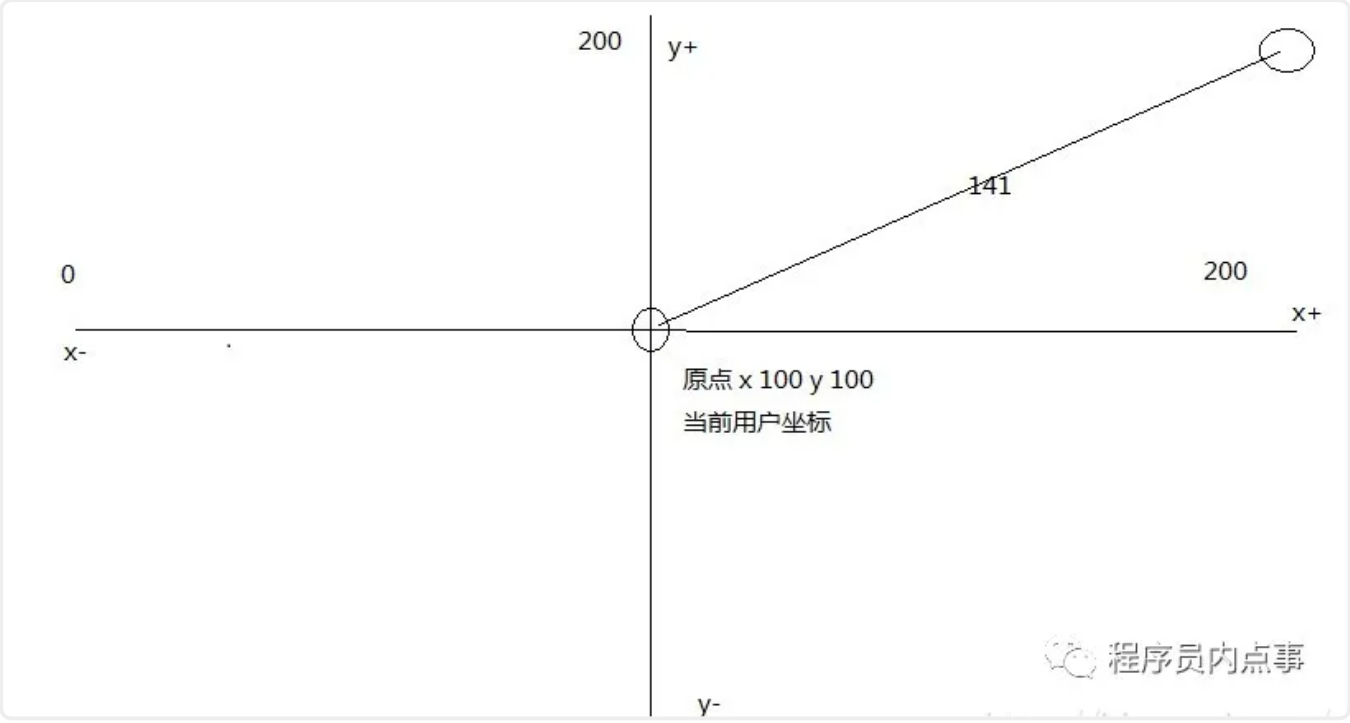
科普：世界上标识一个位置，通用的做法就使用经、纬度。经度的范围在 (-180, 180]，纬度的范围在(-90, 90]，纬度正负以赤道为界，北正南负，经度正负以本初子午线 (英国格林尼治天文台) 为界，东正西负。比如：望京摩托罗拉大厦的经、纬度（116.49141，40.01229）全是正数，就是因为我国位于东北半球。

一、“附近的人”原理

“附近的人” 也就是常说的 LBS (Location Based Services，基于位置服务)，它围绕用户当前地理位置数据而展开的服务，为用户提供精准的增值服务。

“附近的人” 核心思想如下：

1. 以 “我” 为中心，搜索附近的用户
2. 以 “我” 当前的地理位置为准，计算出别人和 “我” 之间的距离
3. 按 “我” 与别人距离的远近排序，筛选出离我最近的用户或者商店等



二、什么是GeoHash算法?

在说 “附近的人” 功能的具体实现之前，先来认识一下 GeoHash 算法，因为后边会一直和它打交道。定位一个位置最好的办法就是用 经、纬度 标识，但 经、纬度 它是二维的，在进行位置计算的时候还是很麻烦，如果能通过某种方法将二维的 经、纬度 数据转换成一维的数据，那么比较起来就要容易的多，因此 GeoHash 算法应运而生。

GeoHash 算法将二维的经、纬度转换成一个字符串，例如：下图中9个 GeoHash 字符串代表了9个区域，每一个字符串代表了一矩形区域。而这个矩形区域内其他的点（经、纬度）都用同一个 GeoHash 字符串表示。

比如： WX4ER 区域内的用户搜索附近的餐厅数据，由于这区域内用户的 GeoHash 字符串都是 WX4ER ，故可以把 WX4ER 当作 key ，餐厅信息作为 value 进行缓存；而如果不使用 GeoHash 算法，区域内的用户请求餐厅数据，用户传来的经、纬度都是不同的，这样缓存不仅麻烦且数据量巨大。

GeoHash 字符串越长，表示的位置越精确，字符串长度越长代表在距离上的误差越小。下图 geohash 码精度表：

geohash码长度	宽度	高度
1	5,009.4km	4,992.6km
2	1,252.3km	624.1km

geohash码长度	宽度	高度
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

而且字符串越相似表示距离越相近，字符串前缀匹配越多的距离越近。比如：下边的经、纬度就代表了三家距离相近的餐厅。

商户	经纬度	Geohash字符串
串串香	116.402843,39.999375	wx4er9v
火锅	116.3967,39.99932	wx4ertk
烤肉	116.40382,39.918118	wx4erfe

让大家简单了解什么是 GeoHash 算法，方便后边内容展开， GeoHash 算法内容比较高深，感兴趣的小伙伴自行深耕一下，这里不占用过多篇幅（其实是我懂得太肤浅，哭唧唧~）。

三、基于Mysql

此种方式是纯基于 mysql 实现的，未使用 GeoHash 算法。

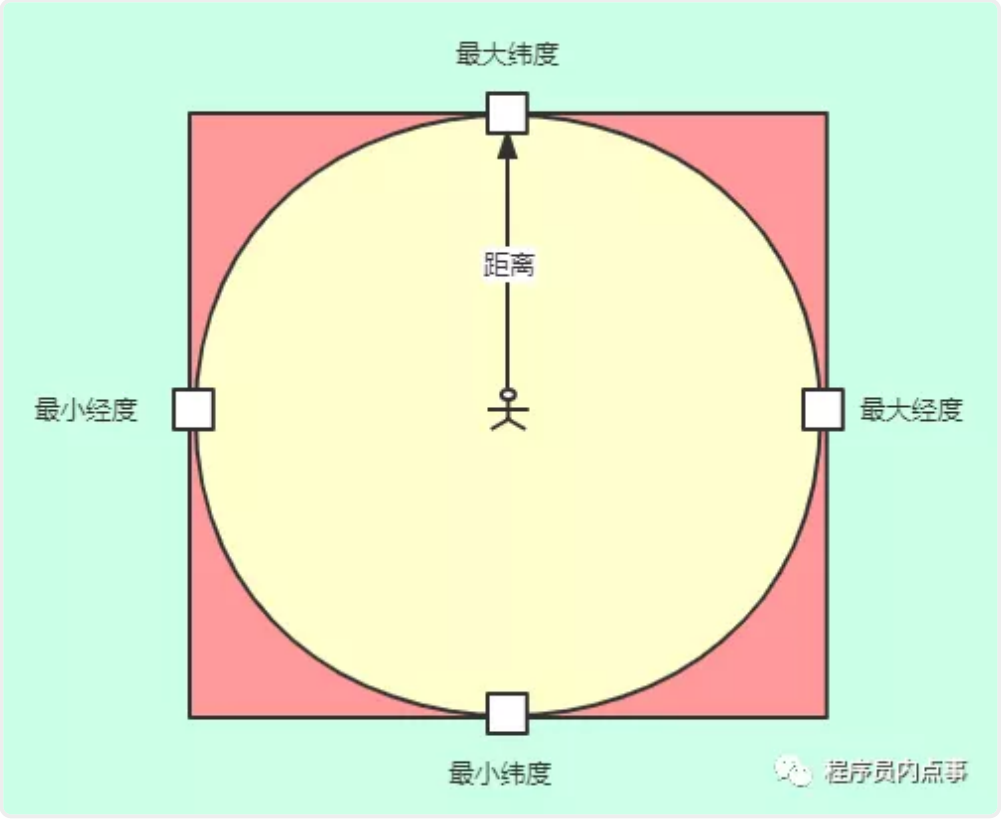
1、设计思路

以用户为中心，假设给定一个500米的距离作为半径画一个圆，这个圆型区域内的所有用户就是符合用户要求的 “附近的人”。但有一个问题是圆形有弧度啊，直接搜索圆形区域难度太大，根本无法用经、纬度直接搜索。

但如果在圆形外套上一个正方形，通过获取用户经、纬度的最大最小值（经、纬度 + 距离），再根据最大最小值作为筛选条件，就很容易将正方形内的用户信息搜索出来。

那么问题又来了，**多出来一些面积肿么办？**

我们来分析一下，多出来的这部分区域内的用户，到圆点的距离一定比圆的半径要大，那么我们就计算用户中心点与正方形内所有用户的距离，筛选出所有距离小于等于半径的用户，圆形区域内的所用户即符合要求的 “附近的人” 。



在这里插入图片描述

2、利弊分析

纯基于 mysql 实现 “附近的人”，优点显而易见就是简单，只要建一张表存下用户的经、纬度信息即可。缺点也很明显，需要大量的计算两个点之间的距离，非常影响性能。

3、实现

创建一个简单的表用来存放用户的经、纬度属性。

```
CREATE TABLE `nearby_user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL COMMENT '名称',
  `longitude` double DEFAULT NULL COMMENT '经度',
  `latitude` double DEFAULT NULL COMMENT '纬度',
  `create_time` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

计算两个点之间的距离，用了一个三方的类库，毕竟自己造的轮子不是特别圆，还有可能是方的，啊哈哈哈~

```
<dependency>
    <groupId>com.spatial4j</groupId>
    <artifactId>spatial4j</artifactId>
    <version>0.5</version>
</dependency>
```

获取到外接正方形后，以正方形的最大最小经、纬度值搜索正方形区域内的用户，再剔除超过指定距离的用户，就是最终的 附近的人 。

```
    private SpatialContext spatialContext = SpatialContext.GEO;

    /**
     * 获取附近 x 米的人
     *
     * @param distance 搜索距离范围 单位km
     * @param userLng  当前用户的经度
     * @param userLat  当前用户的纬度
     */
    @GetMapping("/nearby")
    public String nearBySearch(@RequestParam("distance") double distance,
                               @RequestParam("userLng") double userLng,
                               @RequestParam("userLat") double userLat) {
        //1.获取外接正方形
        Rectangle rectangle = getRectangle(distance, userLng, userLat);
        //2.获取位置在正方形内的所有用户
        List<User> users = userMapper.selectUser(rectangle.getMinX(), rectangle.getMaxX(), rectangle.getMi
```

```
        //3.剔除半径超过指定距离的多余用户
        users = users.stream()
            .filter(a -> getDistance(a.getLongitude(), a.getLatitude(), userLng, userLat) <= distance)
            .collect(Collectors.toList());
        return JSON.toJSONString(users);
    }

    private Rectangle getRectangle(double distance, double userLng, double userLat) {
        return spatialContext.getDistCalc()
            .calcBoxByDistFromPt(spatialContext.makePoint(userLng, userLat),
                    distance * DistanceUtils.KM_TO_DEG, spatialContext, null);
    }
```

由于用户间距离的排序是在业务代码中实现的，可以看到SQL语句也非常的简单。

```
<select id="selectUser" resultMap="BaseResultMap">
    SELECT * FROM user
    WHERE 1=1
    and (longitude BETWEEN ${minlng} AND ${maxlng})
    and (latitude BETWEEN ${minlat} AND ${maxlat})
</select>

```

四、Mysql + GeoHash

1、设计思路

这种方式的设计思路更简单，在存用户位置信息时，根据用户经、纬度属性计算出相应的 geohash 字符串。**注意**：在计算 geohash 字符串时，需要指定 geohash 字符串的精度，也就是 geohash 字符串的长度，**参考上边的 geohash 精度表。**

当需要获取 附近的人 ，只需用当前用户 geohash 字符串，数据库通过 WHERE geohash Like 'geocode% ' 来查询 geohash 字符串相似的用户，然后计算当前用户与搜索出的用户距离，筛选出所有距离小于等于指定距离（附近500米）的，即 附近的人 。

2、利弊分析

利用 GeoHash 算法实现 “附近的人” 有一个问题，由于 geohash 算法将地图分为一个个矩形，对每个矩形进行编码，得到 geohash 字符串。可我当前的点与邻近的点很近，但恰好我们分别在两个区域，明明就在眼前的点偏偏搜不到，实实在在的灯下黑。

如何解决这一问题?

为了避免类似邻近两点在不同区域内，我们就需要同时获取当前点（ WX4G0 ）所在区域附近 8个区域的 geohash 码，一并进行筛选比较。

在这里插入图片描述

3、实现

同样要设计一张表存用户的经、纬度信息，但区别是要多一个 geo_code 字段，存放geohash字符串，此字段通过用户经、纬度属性计算出。使用频繁的字段建议加上索引。

```
CREATE TABLE `nearby_user_geohash` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(255) DEFAULT NULL COMMENT '名称',
  `longitude` double DEFAULT NULL COMMENT '经度',
  `latitude` double DEFAULT NULL COMMENT '纬度',
  `geo_code` varchar(64) DEFAULT NULL COMMENT '经纬度所计算的geohash码',
  `create_time` datetime DEFAULT NULL ON UPDATE CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`),
  KEY `index_geo_hash` (`geo_code`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

首先根据用户经、纬度信息，在指定精度后计算用户坐标的 geoHash 码，再获取到用户周边8个方位的 geoHash 码在数据库中搜索用户，最后过滤掉超出给定距离（500米内）的用户。

```
1   private SpatialContext spatialContext = SpatialContext.GEO;
2
3     /***
4      * 添加用户
5      * @return
6      */
7     @PostMapping("/addUser")
8     public boolean add(@RequestBody UserGeohash user) {
9         //默认精度12位
10         String geoHashCode = GeohashUtils.encodeLatLon(user.getLatitude(),user.getLongitude());
11        return userGeohashService.save(user.setGeoCode(geoHashCode).setCreateTime(LocalDateTime.no
12     }
13
14
15  /**
16      * 获取附近指定范围的人
17      *
18      * @param distance 距离范围（附近多远的用户） 单位km
19      * @param len      geoHash的精度（几位的字符串）
20      * @param userLng  当前用户的经度
21      * @param userLat  当前用户的纬度
22      * @return json
23      */
24     @GetMapping("/nearby")
25     public String nearBySearch(@RequestParam("distance") double distance,
26                                @RequestParam("len") int len,
27                                @RequestParam("userLng") double userLng,
28                                @RequestParam("userLat") double userLat) {
29
30
31         //1.根据要求的范围，确定geoHash码的精度，获取到当前用户坐标的geoHash码
32         GeoHash geoHash = GeoHash.withCharacterPrecision(userLat, userLng, len);
33         //2.获取到用户周边8个方位的geoHash码
34         GeoHash[] adjacent = geoHash.getAdjacent();
35
36         QueryWrapper<UserGeohash> queryWrapper = new QueryWrapper<UserGeohash>()
37             .likeRight("geo_code",geoHash.toBase32());
38         Stream.of(adjacent).forEach(a -> queryWrapper.or().likeRight("geo_code",a.toBase32()));
39
40         //3.匹配指定精度的geoHash码
41         List<UserGeohash> users = userGeohashService.list(queryWrapper);
42         //4.过滤超出距离的
43         users = users.stream()
44                 .filter(a ->getDistance(a.getLongitude(),a.getLatitude(),userLng,userLat)<= distance)
45                 .collect(Collectors.toList());
46         return JSON.toJSONString(users);
47     }
48
49
50     /***
51      * 球面中，两点间的距离
```

```
52          * @param longitude 经度1
53          * @param latitude  纬度1
54          * @param userLng   经度2
55          * @param userLat   纬度2
56          * @return 返回距离，单位km
57          */
58         private double getDistance(Double longitude, Double latitude, double userLng, double userLat) {
59             return spatialContext.calcDistance(spatialContext.makePoint(userLng, userLat),
60                     spatialContext.makePoint(longitude, latitude)) * DistanceUtils.DEG_TO_KM;
61         }
```

五、Redis + GeoHash

Redis 3.2 版本以后，基于 geohash 和数据结构 Zset 提供了地理位置相关功能。通过上边两种 mysql 的实现方式发现， 附近的人 功能是明显的读多写少场景，所以用 redis 性能更会有很大的提升。

1、设计思路

redis 实现 附近的人 功能主要通过 Geo 模块的六个命令。

- GEOADD ： 将给定的位置对象（纬度、经度、名字）添加到指定的key;
- GEOPOS ： 从key里面返回所有给定位置对象的位置（经度和纬度）；
- GEODIST ： 返回两个给定位置之间的距离;
- GEOHASH ： 返回一个或多个位置对象的Geohash表示;
- GEORADIUS ： 以给定的经纬度为中心，返回目标集合中与中心的距离不超过给定最大距离的所有位置对象;
- GEORADIUSBYMEMBER ： 以给定的位置对象为中心，返回与其距离不超过给定最大距离的所有位置对象。

以 GEOADD 命令和 GEORADIUS 命令简单举例：

```
1    GEOADD key longitude latitude member [longitude latitude member ...]
```

其中， key 为集合名称， member 为该经纬度所对应的对象。

GEOADD 添加多个商户“火锅店”位置信息：

```
1    GEOADD hotel 119.98866180732716  30.27465803229662 火锅店
```

GEORADIUS 根据给定的经纬度为中心，获取目标集合中与中心的距离不超过给定最大距离（500米内）的所有位置对象，也就是 “附近的人” 。

```
1    GEORADIUS key longitude latitude radius m|km|ft|mi [WITHCOORD] [WITHDIST] [WITHHASH] [ASC|DESC
```

范围单位： m | km | ft | mi --> 米 | 千米 | 英尺 | 英里。

- WITHDIST ： 在返回位置对象的同时，将位置对象与中心之间的距离也一并返回。距离的单位和用户给定的范围单位保持一致。
- WITHCOORD ： 将位置对象的经度和维度也一并返回。
- WITHHASH ： 以 52 位有符号整数的形式，返回位置对象经过原始 geohash 编码的有序集合分值。这个选项主要用于底层应用或者调试，实际中的作用并不大。
- ASC | DESC ： 从近到远返回位置对象元素 | 从远到近返回位置对象元素。
- COUNT count ： 选取前N个匹配位置对象元素。（不设置则返回所有元素）
- STORE key ： 将返回结果的地理位置信息保存到指定key。
- STORedisT key ： 将返回结果离中心点的距离保存到指定key。

例如下边命令：获取当前位置周边500米内的所有饭店。

```
1    GEORADIUS hotel 119.98866180732716  30.27465803229662 500 m WITHCOORD
```

Redis 内部使用有序集合(zset)保存用户的位置信息， zset 中每个元素都是一个带位置的对象，元素的 score 值为通过经、纬度计算出的52位 geohash 值。

2、利弊分析

redis 实现 附近的人 效率比较高，集成也比较简单，而且还支持对距离排序。不过，结果存在一定的误差，要想让结果更加精确，还需要手动将用户中心位置与其他用户位置计算距离后，再一次进行筛选。

3、实现

以下就是 Java redis 实现版本，代码非常的简洁。

```
@Autowired
    private RedisTemplate<String, Object> redisTemplate;

    //GEO相关命令用到的KEY
    private final static String KEY = "user_info";

    public boolean save(User user) {
        Long flag = redisTemplate.opsForGeo().add(KEY, new RedisGeoCommands.GeoLocation<>(
                user.getName(),
                new Point(user.getLongitude(), user.getLatitude()))
        );
        return flag != null && flag > 0;
    }

    /**
     * 根据当前位置获取附近指定范围内的用户
     * @param distance 指定范围 单位km， 可根据{@link org.springframework.data.geo.Metrics} 进行设置
     * @param userLng 用户经度
     * @param userLat 用户纬度
     * @return
     */
    public String nearBySearch(double distance, double userLng, double userLat) {
        List<User> users = new ArrayList<>();
        // 1.GEORADIUS获取附近范围内的信息
        GeoResults<RedisGeoCommands.GeoLocation<Object>> reslut =
            redisTemplate.opsForGeo().radius(KEY,
                    new Circle(new Point(userLng, userLat), new Distance(distance, Metrics.KILOMETERS)),
                    RedisGeoCommands.GeoRadiusCommandArgs.newGeoRadiusArgs()
                            .includeDistance()
                            .includeCoordinates().sortAscending());
        //2.收集信息，存入list
        List<GeoResult<RedisGeoCommands.GeoLocation<Object>>> content = reslut.getContent();
        //3.过滤掉超过距离的数据
        content.forEach(a-> users.add(
                new User().setDistance(a.getDistance().getValue())
                .setLatitude(a.getContent().getPoint().getX())
                .setLongitude(a.getContent().getPoint().getY())));
        return JSON.toJSONString(users);
    }
```

六、MongoDB + 2d索引

1、设计思路

MongoDB 实现附近的人，主要是通过它的两种地理空间索引 2dsphere 和 2d 。两种索引的底层依然是基于 Geohash 来进行构建的。但与国际通用的 Geohash 还有一些不同，具体参考官方文档。

2dsphere 索引仅支持球形表面的几何形状查询。

2d 索引支持平面几何形状和一些球形查询。虽然 2d 索引支持某些球形查询，但 2d 索引对这些球形查询时，可能会出错。所以球形查询尽量选择 2dsphere 索引。

尽管两种索引的方式不同，但只要坐标跨度不太大，这两个索引计算出的距离相差几乎可以忽略不计。

2、实现

首先插入一批位置数据到 MongoDB ， collection 为起名 hotel ，相当于 MySQL 的表名。两个字段 name 名称， location 为经、纬度数据对。

```
db.hotel.insertMany([
 {'name':'hotel1', location:[115.993121,28.676436]},
 {'name':'hotel2', location:[116.000093,28.679402]},
 {'name':'hotel3', location:[115.999967,28.679743]},
 {'name':'hotel4', location:[115.995593,28.681632]},
 {'name':'hotel5', location:[115.975543,28.679509]},
 {'name':'hotel6', location:[115.968428,28.669368]},
 {'name':'hotel7', location:[116.035262,28.677037]},
 {'name':'hotel8', location:[116.024770,28.68667]},
 {'name':'hotel9', location:[116.002384,28.683865]},
 {'name':'hotel10', location:[116.000821,28.68129]},
])
```

接下来我们给 location 字段创建一个 2d 索引，索引的精度通过 bits 来指定， bits 越大，索引的精度就越高。

```
db.coll.createIndex({'location':"2d"}, {"bits":11111})
```

用 geoNear 命令测试一下， near 当前坐标（经、纬度）， spherical 是否计算球面距离， distanceMultiplier 地球半径，单位是米，默认6378137， maxDistance 过滤条件（指定距离内的用户），开启弧度需除 distanceMultiplier ， distanceField 计算出的两点间距离，字段别名（随意取名）。

```
db.hotel.aggregate({
    $geoNear:{
        near: [115.999567,28.681813], // 当前坐标
        spherical: true, // 计算球面距离
        distanceMultiplier: 6378137, // 地球半径,单位是米,那么的除的记录也是米
        maxDistance: 2000/6378137, // 过滤条件2000米内，需要弧度
        distanceField: "distance" // 距离字段别名
    }
})
```

看到结果中有符合条件的数据，还多出一个字段 distance 刚才设置的别名，代表两点间的距离。

```
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e58"), "name" : "hotel10", "location" : [ 116.000821, 28.68129 ], "
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e51"), "name" : "hotel3", "location" : [ 115.999967, 28.679743 ], "
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e50"), "name" : "hotel2", "location" : [ 116.000093, 28.679402 ], "
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e57"), "name" : "hotel9", "location" : [ 116.002384, 28.683865 ], "
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e52"), "name" : "hotel4", "location" : [ 115.995593, 28.681632 ], "
{ "_id" : ObjectId("5e96a5c91b8d4ce765381e4f"), "name" : "hotel1", "location" : [ 115.993121, 28.676436 ], "
```

总结

本文重点并不是在具体实现，旨在给大家提供一些设计思路，面试中可能你对某一项技术了解的并不深入，但如果你的知识面宽，可以从多方面说出多种设计的思路，能够侃侃而谈，那么会给面试官极大的好感度，拿到offer的概率就会高很多。而且“附近的人” 功能使用的场景比较多，尤其是像电商平台应用更为广泛，所以想要进大厂的同学，这类的知识点还是应该有所了解的。

代码实现借鉴了一位大佬的开源项目，这里有前三种实现方式的demo，感兴趣的小伙伴可以学习一下，GitHub地址： https://github.com/larscheng/larscheng-learning-demo/tree/master/NearbySearch ，。

END

小福利：

获取到一些课 ， 嘘~，**免费** 送给小伙伴们。关注公众号

往期精彩回顾

为了不复制粘贴，我被逼着学会了JAVA爬虫
一口气说出 9种 分布式ID生成方式，面试官有点懵了
面试总被问分库分表怎么办？这些知识点你要懂
基于 Java 实现的人脸识别功能（附源码）
面试被问分布式ID怎么办？滴滴（Tinyid）甩给他
9种分布式ID生成之美团（Leaf）实战

Modified on 2020/04/22

People who liked this content also liked

聊聊spring事务失效的12种场景，太坑了
苏三说技术