探针数据二次过滤算法

1. 探针数据结构如下：  
   Class ProbeScanningData{  
   timeStamp; //数据被探针扫描的时间  
   mallID; //商场ID  
   shopID; //商铺ID  
   probeMac; //探针设备的MAC  
   clientMac; //扫描到的用户的MAC  
   rssi; //接收到的信号强度  
   }
2. 实际log文件中存储的数据格式如下：  
   timestamp mallID shopID probeMac clientMac rssi  
   为了方便起见，用一个ProbeScanningData的实例对象probeScanningData来代表log文件中一行的数据
3. 把同一个用户的MAC记为clientMac\_01的一天的探针数据按照时间先后顺序放入一个列表list中，格式如下：

|  |
| --- |
| probeScanningData\_0 |
| probeScanningData\_1 |
| probeScanningData\_2 |
| probeScanningData\_3 |
| probeScanningData\_4 |
| probeScanningData\_5 |
| probeScanningData\_6 |
| probeScanningData\_7 |
| probeScanningData\_8 |
| probeScanningData\_9 |
| probeScanningData\_10 |
| probeScanningData\_11 |
| probeScanningData\_12 |
| . |
| . |
| . |
| . |
| . |
| . |
| probeScanningData\_n |

说明：这些数据是改clientMac\_01的一天的数据，在同一个商铺里面被多个探针扫描到，去过多个商铺。

4．由于在同一个商铺里面可能会被其他商铺的探针扫描到，这些数据视为噪声数据，需要被过滤掉，先定义一个初始变量probeScanningDataInit用来存储list的第一个值。

5．遍历list，从list的第一个数据开始遍历，取出list的第一个数据，也就是list.get(0), 用

probeScanningDataInit.shopid和list.get(0).shopid进行比较，如果相等则把list.get(0)这个数据加入到一个新的list中，记为newlist，然后把 list.get(0)赋值给probeScanningDataInit. 然后再进行下一次比较。如果probeScanningDataInit.shopid不等于list.get(0).shopid，则判断如果probeScanningDataInit.rssi - list.get(0).rssi>阈值(根据实际情况和试验得出)，表示前一个探针数据的信号强度比当前这个大很多，则进行下一次比较，不把当前值加到newlist中，如果不是，则取当前数据之后的5(根据实际情况可改)个数据进行比较，如果这个5个数据中与probeScanningDataInit.shopid一样的数据最多，说明当前用户还在probeScanningDataInit.shopid的商铺中，然后进行下一次比较，如果5个数中与probeScanningDataInit.shopid不一样的数量多，则判断当前数据的shopid是否和 5个数中与probeScanningDataInit.shopid不一样的数量多的数据的shopid一样，如果一样则把当前数据加入到newlist中，把当前数据赋值给probeScanningDataInit，然后进行下一次比较；如果不是，进行一下比较；

6．伪代码如下：

//初始变量probeScanningDataInit用来存储list的第一个值

probeScanningDataInit = list.get(0);

for(int i=0; i<=list.size();i++){

if(probeScanningDataInit == list.get(i)){

//newList是存储过滤后的数据的列表

newList.add(list.get(i));

probeScanningDataInit = list.get(i);

}else{

probeScanningDataTemp = list.get(i);

//threshold 是一个经验值可以改变

if(probeScanningDataInit.rssi – probeScanningDataTemp> threshold ){

//继续下一次比较，不把当前数据加入到newList中

continue;

}else{

//定义一个列表存储i之后的5个数据,并加入数据

newlistTemp;

for(int j = i+1; j < i+5;j++){

listTemp.ada(list.get(j));

}

1. 统计listTemp列表中相同shopID数据的数量

If(列表中相同shopid数据最多的shopid ==probeScanningDataInit.shopid ){

continue;

}else{

if(listTemp列表中相同shopid最多的shopid == probeScanningDataTemp.shopid){

nenList.add(probeScanningDataTemp);

probeScanningDataTemp = probeScanningDataTemp;

}else{

continue;

}

}

}

}

return newList;

}