# SparseArray，ArrayMap，ArraySet

<https://blog.csdn.net/hq942845204/article/details/81293480>

<https://blog.csdn.net/u010687392/article/details/47809295>

<https://blog.csdn.net/qq_37704124/article/details/89715604>

<https://developer.android.com/reference/android/util/SparseArray>

<https://developer.android.com/reference/android/util/ArrayMap>

<https://developer.android.com/reference/android/util/ArraySet>

## （1） HashMap的缺陷

<https://blog.csdn.net/u010687392/article/details/47809295>

在Android中，HashMap是比较费内存的，我们在一些情况下可以使用SparseArray和ArrayMap来代替HashMap。

## （x） 是什么

### 【1】 SparseArray及兄弟类

#### 本身



<https://developer.android.com/reference/android/util/SparseArray>

SparseArray maps integers to Objects and, unlike a normal array of Objects, its indices can contain gaps（它的下标能包含间隔）. SparseArray is intended to be more memory-efficient than a [HashMap](https://developer.android.com/reference/java/util/HashMap), because it avoids auto-boxing keys and its data structure doesn't rely on an extra entry object for each mapping.

<https://blog.csdn.net/hq942845204/article/details/81293480>

hashmap的key值和value值为泛型，但是SparseArray 的key值只能为int 类型，value值为Object类型。

#### 兄弟类（LongSparseArray，SparseBooleanArray，SparseIntArray，SparseLongArray）

<https://blog.csdn.net/hq942845204/article/details/81293480>

LongSparseArray 和SparseArray 相比，唯一的不同就是key值为long，所以，既然为long ，那么相对SparseArray 来说，它可以存储的数据元素就比SparseArray 多。

然后轮到了SparseBooleanArray，SparseIntArray，SparseLongArray，这三兄弟相对SparseArray 来说就是value值是确定的，SparseBooleanArray的value固定为boolean类型，SparseIntArray的value固定为int类型，SparseLongArray的value固定为long类型。

稍作总结一下，如下：

|  |
| --- |
| SparseArray <int, Object>  LongSparseArray <long, Object>  SparseBooleanArray <int, boolean>  SparseIntArray <int, int>  SparseLongArray <int, long> |

### 【2】 ArrayMap



<https://developer.android.com/reference/android/util/ArrayMap>

ArrayMap is a generic key->value mapping data structure that is designed to be more memory efficient than a traditional [HashMap](https://developer.android.com/reference/java/util/HashMap). It keeps its mappings in an array data structure -- an integer array of hash codes for each item, and an Object array of the key/value pairs. This allows it to avoid having to create an extra object for every entry put in to the map, and it also tries to control the growth of the size of these arrays more aggressively：积极 (since growing them only requires copying the entries in the array, not rebuilding a hash map).

<https://blog.csdn.net/u010687392/article/details/47809295>

ArrayMap是一个<key,value>映射的数据结构，它设计上更多的是考虑内存的优化，内部是使用两个数组进行数据存储，一个数组记录key的hash值，另外一个数组记录key和value值，它和SparseArray一样，也会对key使用二分法进行从小到大排序，在添加、删除、查找数据的时候都是先使用二分查找法得到相应的index，然后通过index来进行添加、查找、删除等操作，所以，应用场景和SparseArray的一样，如果在数据量比较大的情况下，那么它的性能将退化至少50%。

### 【3】 ArraySet



<https://developer.android.com/reference/android/util/ArraySet>

ArraySet is a generic set data structure that is designed to be more memory efficient than a traditional [HashSet](https://developer.android.com/reference/java/util/HashSet). The design is very similar to [ArrayMap](https://developer.android.com/reference/android/util/ArrayMap), with all of the caveats:注意事项 described there. This implementation is separate from ArrayMap, however, so the Object array contains only one item for each entry in the set (instead of a pair for a mapping).

<https://blog.csdn.net/hq942845204/article/details/81293480>

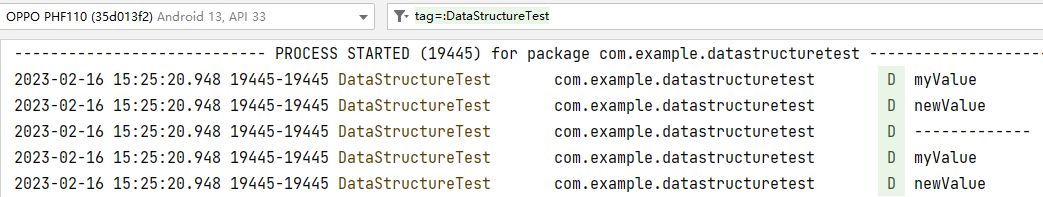
ArrayMap和ArraySet的关系就像HashMap和HashSet一样（但ArraySet不是用ArrayMap实现的），ArraySet和HashSet都是不能存储相同的元素。

## （x） 使用

### 【1】 SparseArray

|  |
| --- |
| *//声明* SparseArray<String> sparseArray = new SparseArray<>(); *//增加元素，append方式* sparseArray.append(0, "myValue"); *//增加元素，put方式* sparseArray.put(1, "myValue"); *//删除元素，二者等同* sparseArray.remove(1); sparseArray.delete(1); *//修改元素，put或者append相同的key值即可* sparseArray.put(1, "newValue"); sparseArray.append(1, "newValue"); *//查找，遍历方式1* for (int i = 0; i < sparseArray.size(); i++) {  Log.*d*(*TAG*, sparseArray.valueAt(i)); } Log.*d*(*TAG*, "-------------"); *//查找，遍历方式2* for (int i = 0; i < sparseArray.size(); i++) {  int key = sparseArray.keyAt(i);  Log.*d*(*TAG*, sparseArray.get(key)); } |

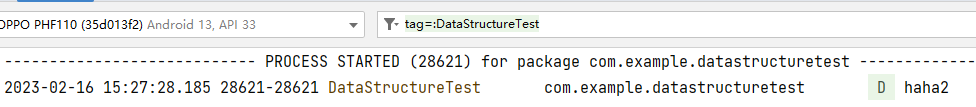
运行结果：



### 【2】 ArrayMap

|  |
| --- |
| ArrayMap<String, String> map = new ArrayMap<>(); *//增加* map.put("xixi", "haha"); *//删除* map.remove("xixi"); *//修改，put相同的key值即可* map.put("xixi2", "haha"); map.put("xixi2", "haha2"); *//查找，通过key来遍历* for (String key : map.keySet()) {  Log.*d*(*TAG*, map.get(key)); } |

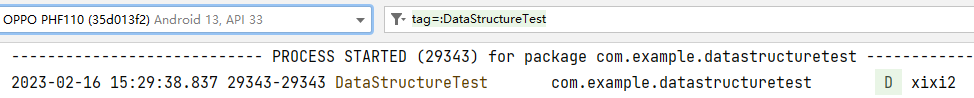
运行结果：



### 【3】 ArraySet

|  |
| --- |
| ArraySet<String> set = new ArraySet<>(); *//增加* set.add("xixi"); set.add("xixi2"); *//删除* set.remove("xixi"); *//查找* for (String val : set) {  Log.*d*(*TAG*, val); } |

运行结果：



## （x） 原理+源码分析

### 【1】 SparseArray

#### 《1》 原理概述

<https://blog.csdn.net/hq942845204/article/details/81293480>

<https://blog.csdn.net/u010687392/article/details/47809295>

* 通过两个数组来进行数据存储的，一个存储key，另外一个存储value
* 延迟删除机制
* 对于查询、添加、删除元素操作，先在存储key的数组中通过二分查找定位到要查询/添加/删除元素的位置，然后在该位置对存储key和存储value的数组分别进行添加/删除元素。
* 利用gc函数一次性压缩空间，提高效率

### 【2】 ArrayMap

#### 《1》 原理概述

<https://blog.csdn.net/hq942845204/article/details/81293480>

* 存储结构，两个数组存储，一个存key的hash，一个存key和value
* 和SparseArray相似，在查询、添加、删除元素时，先用二分查找在{存key的hash的数组}中找到对应index，然后分别在{存key的hash的数组}的index位置添加/删除hash，以及在{存key+value的数组}的index\*2处添加/删除key+value。
* 数组缓存设计：2个缓存数组mBaseCache（用来缓存容量为BASE\_SIZE的mHashes和mArray数组）和mTwiceBaseCache（用来缓存容量为BASE\_SIZE\*2的mHashes和mArray数组），主要作用是缓存小数组以避免在扩容时频繁地用new创建新的数组导致内存消耗。
* 删除元素时的数组容量及时收缩
* 删除元素时的下界控制，防止抖动

### 一些总结

<https://blog.csdn.net/hq942845204/article/details/81293480>

其实我们对比一下它们二者，有很多共性,，也就是它们的出发点都是不变的就是以时间换空间，比如它们都有即时空间压缩机制，SparseArray 采用的延迟删除和gc机制来保证无用空间的及时压缩，ArrayMap 采用的删除时通过逻辑判断来处理空间的压缩，其实说白了，它们的设计者都是力求一种极致充分的内存利用率，这也必然导致了所带来的问题，例如插入删除逻辑复杂等，不过这和我们使用起来一点都不违背，我们只需要在合适的场景进行取舍即可。

## （x） 优缺点

### 【1】 SparseArray及兄弟类

#### 《1》 优点

<https://blog.csdn.net/hq942845204/article/details/81293480>

* 通过它的三兄弟可以避免存取元素时的装箱和拆箱（关于装箱和拆箱带来的效率问题，可以查看我的这篇文章，[Java装箱和拆箱详解](https://blog.csdn.net/hq942845204/article/details/81019902)）
* 频繁的插入删除操作效率高（延迟删除机制保证了效率）
* 会定期通过gc函数来清理内存，内存利用率高
* 放弃hash查找，使用二分查找，更轻量

#### 《2》 缺点

<https://blog.csdn.net/hq942845204/article/details/81293480>

* 二分查找的时间复杂度O(log n)，大数据量的情况下，效率没有HashMap高
* key只能是int 或者long

### 【2】 ArrayMap

#### 《1》 优点

<https://blog.csdn.net/hq942845204/article/details/81293480>

* 在数据量少时，内存利用率高，及时的空间压缩机制
* 迭代效率高，可以使用索引来迭代（keyAt()方法以及valueAt() 方法），相比于HashMap迭代使用迭代器模式，效率要高很多

#### 《2》 缺点

<https://blog.csdn.net/hq942845204/article/details/81293480>

* 存取复杂度高，花费大
* 二分查找的O(log n )时间复杂度远远大于HashMap
* ArrayMap没有实现Serializable，不利于在Android中借助Bundle传输。

## （x） 应用场景

### 【1】 SparseArray及兄弟类

<https://blog.csdn.net/hq942845204/article/details/81293480>

* item数量为 <1000级别的
* 存取的key或value为指定类型的，比如boolean、int、long，可以避免自动装箱和拆箱问题。

### 【2】 ArrayMap

<https://blog.csdn.net/hq942845204/article/details/81293480>

* item数量为 <1000 级别的，尤其是在查询多，插入数据和删除数据不频繁的情况
* Map中包含子Map对象

### 【3】 SparseArray及兄弟类+ArrayMap+HashMap

<https://blog.csdn.net/hq942845204/article/details/81293480>

如果觉得这些优缺点一大堆，还是很迷，我就再精简一下二者使用的取舍：

(1) 首先二者都是适用于数据量小的情况，但是SparseArray以及他的三兄弟们避免了自动装箱和拆箱问题，也就是说在特定场景下，比如你存储的value值全部是int类型，并且key也是int类型，那么就采用SparseArray，其它情况就采用ArrayMap。

(2) 数据量多的时候当然还是使用HashMap啦

<https://blog.csdn.net/u010687392/article/details/47809295>

SparseArray和ArrayMap都差不多，使用哪个呢？  
假设数据量都在千级以内的情况下：

1、如果key的类型已经确定为int类型，那么使用SparseArray，因为它避免了自动装箱的过程，如果key为long类型，它还提供了一个LongSparseArray来确保key为long类型时的使用

2、如果key类型为其它的类型，则使用ArrayMap

## （x） 实例

### 【1】 SparseArray

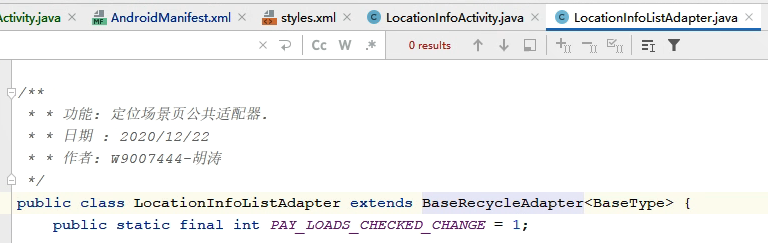
#### 例1：

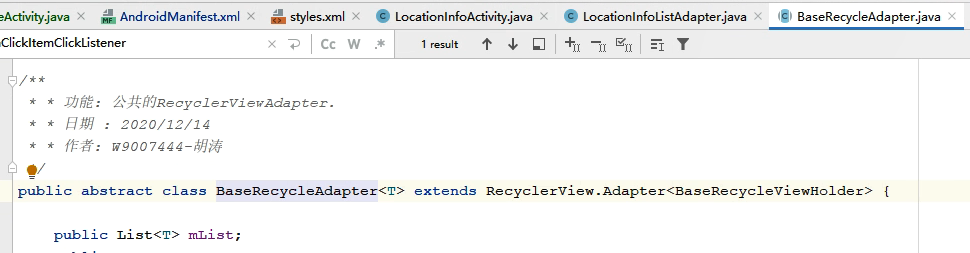
图形用户界面, 文本, 应用程序, 电子邮件

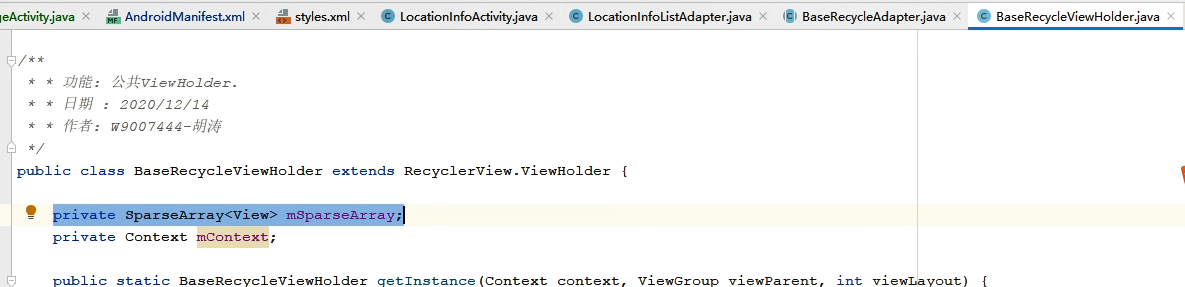
描述已自动生成

#### 例2：

Oppo-Smarthome app-场景-》添加场景-》添加条件-》位置变化时

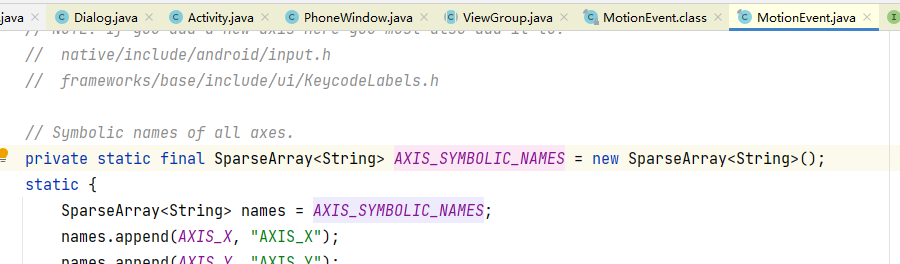


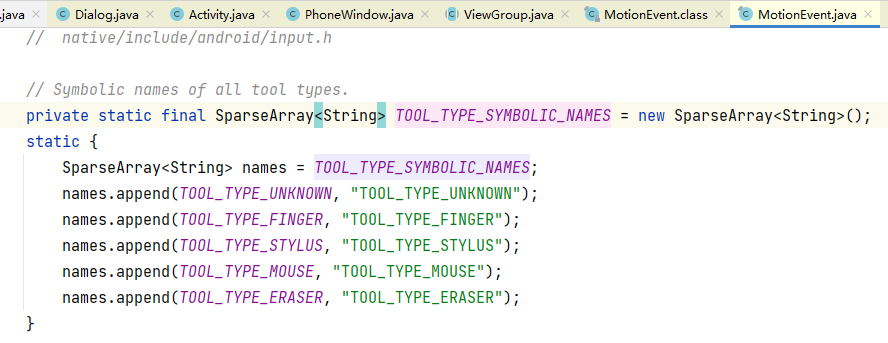




胡涛说因为key固定int型，所以当时用这个。

#### 例3：





### 【2】 ArrayMap

#### 例1：

图形用户界面, 文本

描述已自动生成