App 运行时发生 OOM 的原因你知道哪几种？如何避免？

1.资源对象没关闭造成的内存泄露，try catch finally 中将资源回收放到 finally 语句可以有效避免 OOM。资源性对象比如：

1-1，Cursor
1-2，调用 registerReceiver 后未调用 unregisterReceiver()
1-3，未关闭 InputStream/OutputStream
1-4，Bitmap 使用后未调用 recycle()

2.作用域不一样，导致对象不能被垃圾回收器回收，比如：

2-1，非静态内部类会隐式地持有外部类的引用，
2-2，Context 泄露
概括一下，避免 Context 相关的内存泄露，记住以下事情：
1、 不要保留对 Context-Activity 长时间的引用（对 Activity 的引用的时候，必须确保拥有和 Activity 一样的生命周期）
2、尝试使用 Context-Application 来替代 Context-Activity 3、如果你不想控制内部类的生命周期，应避免在 Activity 中使用非静态的内部类，而应该使用静态的内部类，并在其中创建一个对 Activity 的弱引用。
这种情况的解决办法是使用一个静态的内部类，其中拥有对外部类的 WeakReference。
2-3，Thread 引用其他对象也容易出现对象泄露。
2-4，onReceive 方法里执行了太多的操作

3.内存压力过大

3-1，图片资源加载过多，超过内存使用空间，例如 Bitmap 的使用
3-2，重复创建 view，listview 应该使用 convertview 和 viewholder

如何避免内存泄露：

1.使用缓存技术，比如 LruCache、DiskLruCache、对象重复并且频繁调用可以考虑对象池
2.对于引用生命周期不一样的对象，可以用软引用或弱引用 SoftReferner WeakReferner
3.对于资源对象 使用 finally 强制关闭
4.内存压力过大就要统一的管理内存

写段代码，定义一个字符串常量，字符串中只有大小写字母和整数，输出字符串中的出现最多的数字的和？例如 ” 9fil3dj11P0jAsf11j ” 中出现最多的是 11 两次，输出 22.

```
public int func(String s) {
    int maxCount = 0;
        int maxVal = 0;
    String[] strs = s.split("[^0-9]");
    HashMap<Integer,Integer> map = new HashMap<Integer,Integer>();
    for(int i = 0; i < strs.length; i++) {
        if(!"".equals(strs[i])) {
            int key = Integer.valueOf(strs[i]);
```

```
                if(map.containsKey(key)) {
                        map.put(key, map.get(key) + 1);
                    }
                    else {
                        map.put(key, 1);
                    }
                    if(maxCount < map.get(key)) {
                        maxCount = map.get(key);
                        maxValue = key;
                    }
                }
            }
            return maxValue * maxCount;
    }
```