Android 算法篇

工程师 李洪江

一 集合分页算法

```
/**
 * 分页算法
 *
 * @param outDataSource   输出集合
 * @param inputDataSource 输入集合
 * @param column          列数
 */
public static void setPageData(List<Object> outDataSource,
                               List<? extends MYData> inputDataSource, int column) {
    int total = inputDataSource.size();
    int count = (total / column) + ((total % column > 0) ? 1 : 0);
    for (int i = 0; i < count; i++) {
        outDataSource.add(inputDataSource.subList(i * column, Math.min((i + 1) * column, total)));
    }
}
```

二 列表加载去重算法

【1】前加载去重复数据

```
public static <T extends MYData> void addUnRepeatObjectsFront(List<T> list, List<? extends T> dataList) {
    if (list == null || dataList == null) {
        return;
    }

    LinkedHashSet<String> set = new LinkedHashSet<>();

    for (int i = 0; i < list.size(); i++) {
        String id = list.get(i).getId();
        if (id != null) {
            set.add(id);
        }
    }

    for (int i = 0; i < dataList.size(); i++) {
        T data = dataList.get(i);
        String id = data.getId();
        if (id == null) {
            continue;
        }
        if (!set.contains(id)) {
            set.add(id);
        } else {
            dataList.remove(data);
        }
    }

    set.clear();
}
```

【2】后加载去重复数据

```
public static <T extends MYData> void addUnRepeatObjects(List<T> list, List<? extends T> dataList) {
    if (list == null || dataList == null) {
        return;
    }

    LinkedHashSet<String> set = new LinkedHashSet<>();

    for (int i = 0; i < list.size(); i++) {
        String id = list.get(i).getId();
        if (id != null) {
            set.add(id);
        }
    }

    for (int i = 0; i < dataList.size(); i++) {
        T data = dataList.get(i);
        String id = data.getId();
        if (id == null) { // 如果ID为null，则不做去重判断
            list.add(data);
            continue;
        }
        if (!set.contains(id)) {
            set.add(id);
            list.add(data);
        }
    }

    set.clear();
}
```

三 容器复用算法

【1】不复用写法-性能不好

```
public ArrayList<T> mData = new ArrayList<>();
public ViewGroup mContainer;

public abstract View getView(int position, View convertView,
                             ViewGroup parent);

private void notifyDataSetChanged() {
    mContainer.removeAllViews();
    int count = mData.size();
    for (int i = 0; i < count; i++) {
       View view = getView(i, null, mContainer);
       mContainer.addView(view);
    }
}
```

相对较好的一种写法，简易版适配器写法

```
public ArrayList<T> mData = new ArrayList<>();
public ViewGroup mContainer;

public abstract View getView(int position, View convertView,
                             ViewGroup parent);

public void notifyDataSetChangedOptimalStrategy() {
    int count = mData.size();
    for (int i = 0; i < count; i++) {
        View mCacheView = getOldView(i);
        if (mCacheView != null) {
            getView(i, mCacheView, mContainer);
            mCacheView.setVisibility(View.VISIBLE);
        } else {
            View view = getView(i, null, mContainer);
            mContainer.addView(view);
        }
    }

    int childCount = mContainer.getChildCount();

    for (int i = mData.size(); i < childCount; i++) {
        mContainer.getChildAt(i).setVisibility(View.GONE);
    }
}

private View getOldView(int index) {
    if (index >= mContainer.getChildCount()) {
        return null;
    }
    return mContainer.getChildAt(index);
}
```

四 递归算法

【1】递归上传文件

【2】递归遍历文件夹

【3】递归查找 View

五 奇偶算法

【1】在列表控制线显示时候。

六 总行数

```
/**
 * 计算总行数
 *
 * @param total  总数量
 * @param column 列数
 * @return
 */
public static int calcLineNum(int total, int column) {
    return (total / column) + ((total % column > 0) ? 1 : 0);
}
```

七 equals 算法

【1】对象 String 时候，直接判断是否相等

【2】判断 List 集合是否包含对象，一种是遍历集合处理，第二种是重写 equals 方法，使用集合自带的 contains 方法，快速判断。

【3】判断 HashSet 做单选，多选处理，重写 hash 方法和 equals 方法达到去重效果

八 字典排序算法-tree 数据结构使用

```
Set<String> set = new TreeSet<>();
for (Map.Entry<String, String> entry : mParams.entrySet()) {
    String vaule = entry.getValue();
    set.add(entry.getKey().concat(vaule != null ? vaule : ""));
}
```

九 在写添加页面和编辑页面信息时候，复用界面写法。

【1】一种　使用状态字段来区分。

【2】一种　使用对象来区分。新数据和旧数据比较是否改变。

【3】涉及到深拷贝和浅拷贝知识。集合的 add 和 addAll 属于浅拷贝，修改对象属性值都会引起原来集合对象改变。

```
public class MYFamilyService extends MYData implements Cloneable {

   public String service_id;


   //实现对象深拷贝，对象实现cloneable接口
   @Override
   public MYFamilyService clone() {
       MYFamilyService clone = null;
       try {
           clone = (MYFamilyService) super.clone();
       } catch (CloneNotSupportedException e) {
           throw new RuntimeException(e); // won't happen
       }
       return clone;
   }

}
```

```
private ArrayList<MYFamilyService> deepClone(ArrayList<MYFamilyService> old) {
    ArrayList<MYFamilyService> copy = new ArrayList<>();
    for (int j = 0; j < old.size(); j++) {
        copy.add(old.get(j).clone());
    }
    return copy;
}
```

十 列表 item 拆分算法

【1】利用可见性拆分 item 项目。

【2】拆分 N 个小 Item。

十一 单选和多选算法

【1】不复用列表：

单选可采用保存 View 性能最好。

单选也可记录位置效率较好。

单选可采用遍历 View 性能一般。

```
private void setSingleSelect(ViewGroup mFamilyType, View v) {
    for (int i = 0; i < mFamilyType.getChildCount(); i++) {
        View temp = mFamilyType.getChildAt(i);
        mFamilyType.getChildAt(i).setSelected(temp == v);
    }
}

public Integer getHourseType() {
    int size = mFamilyType.getChildCount();
    for (int i = 0; i < size; i++) {
        if (mFamilyType.getChildAt(i).isSelected()) {
            return i + 1;
        }
    }
    return null;
}
```

多选可使用自身选中标记处理，最后遍历查看状态。

```
private void setMultiSelect(View v) {
    v.setSelected(!v.isSelected());
}
```

```
public ArrayList<Integer> getHourseCondition() {
    ArrayList<Integer> data = new ArrayList<>();
    int size = mFamilyCondition.getChildCount();
    for (int i = 0; i < size; i++) {
        if (mFamilyCondition.getChildAt(i).isSelected()) {
            data.add(i + 1);
        }
    }
    return data;
}
```

【2】复用 View，不能使用本身记忆。

单选采用 position 记忆法，需要全部刷新。

单选采用 HashSet 方法记忆，不需要全部刷新。

多选采用数据本身记忆，不需要全部刷新。

【3】分页单选和复选

十二 图片压缩算法

【1】比例压缩

【2】质量压缩

十三 View 可见性算法

不建议这样写

```
mPics.setVisibility(isImageData ? View.GONE : View.VISIBLE);
```

建议这样写

```
//优化代码
public void setVisible(int  visible,View  view){
    if (view.getVisibility()==visible){
        return;
    }
    view.setVisibility(visible);
}
```

十四 不同状态不同背景写法

【1】不建议代码设置不同背景写法

【2】建议使用选择器，利用 enable，check，selected

【3】所有组件都有 selected 和 enable 状态。

【4】只有 check 类型控件才有 check 组件。

十五 Arrays 和 Collections 使用。

Arrays 常用方法：sort 排序，asList 转化集合，binarySearch 二分查找数据，fill 填充方法，equals 比较数组等。

Collections 常用方法：sort 排序。

十六 集合迭代删除算法

```
//测试ArrayList迭代过程中删除元素，
//避免抛出 java.util.ConcurrentModificationException
List<User> list  = new ArrayList<User>();
for(int i = 0 ; i < 10 ; i ++){
    list.add(new User(i+""));
}
for(User temp:list){
    if(Integer.parseInt(temp.name) % 2 == 0){
        list.remove(temp);//这里引起异常，这种迭代方式新增删除都会引起异常
    }
    System.out.print(temp.name + ",");
}
System.out.println("=====");
Iterator<User> it = list.iterator();
while(it.hasNext()){        //正确做法
    //System.out.println(it.hasNext());
    User temp = it.next();
    if(Integer.parseInt(temp.name) % 2 == 0){
        it.remove();
    }
}

for(User temp:list){
    System.out.print(temp.name + ",");
}
```

```
private static void commonSceneWhenRemove(Map<String, String> source) {
      Iterator<Map.Entry<String, String>> iterator = source.entrySet().iterator();
      while (iterator.hasNext()) {
          Map.Entry<String, String> entry = iterator.next();
          if (entry.getKey().contains("1")) {
              iterator.remove();
          }
      }
      System.out.println(source);
  }
```