Android Paging分页库讲解

前言：

这篇文章讲讲述android分页加载库的实现原理并模拟源码实现分页加载

问题：

通常我们在列表展示加载数据的时候，在数据量比较大的情况下，我们不会将对象一次性全部加载到内存，此时需要一个分页加载机制，比如优先加载一段，比如20条数据，如果用户想要继续滑动查看更多，可以选择加载更多数据，从而节省内存，提高APP性能

对此现象，开发者往往有很多自定义加载更多，Android在新版本也推出了自己的paging分页加载库

依赖：

implementation "androidx.paging:paging-runtime:2.1.2"

下面我们将模拟源码实现Android的Paging分页加载更多

1. 列表RecyclerView数据来源于Adapter，这里我们优先定义我们的Adapter

public abstract class PagedListAdapter<T, VH extends RecyclerView.ViewHolder> extends RecyclerView.Adapter<VH> {  
  
 private final PagedListDiffer<T> differ;  
  
 public PagedListAdapter(ItemCallback<T> diffCallback) {  
 differ = new PagedListDiffer<>(diffCallback);  
 }  
  
 @Override  
 public int getItemCount() {  
 return differ.getItemCount();  
 }  
  
 public void submitList(@Nullable List<T> list) {  
 differ.submitList(list);  
 }

}

这里我们定义一个PagedListAdapter，由于持有的List数据类型未知，我们就使用泛型T来代替，毕竟暂时不关心数据类型

2. 从上面可以看出，我们的Adapter里面持有一个differ变量，在构造函数里面创建该对象，并对外提供一个ItemCallback

数据由differ持有，并实现获取数据个数getItemCount方法

2.1 同时我们在对子类提供一个getItem方法，方便子类在onBindViewHolder获取位置数据

public T getItem(int position){  
 return differ.getItem(position);  
}  
  
//*todo 子类覆盖*@Override  
public void onBindViewHolder(@NonNull VH holder, int position) {  
 T item = getItem(position);  
 //设置Item数据  
}

这里便完成了我们的Adapter，将内容获取与选择都交给differ完成，并在构造函数创建该differ，实现自定义比较，其中比较的内容接口ItemCallback如下：

public interface ItemCallback<T> {  
 */\*\*  
 \* 数据是否相同  
 \** ***@param*** *oldItem 旧数据  
 \** ***@param*** *newItem 新数据  
 \** ***@return*** *是否相同  
 \*/* boolean areItemsTheSame(@NonNull T oldItem, @NonNull T newItem);  
}

这样的话，differ就可以由外部传递的callback，来比较数据是否相同

这里我们可以默认实现返回false，确保内部数据都是有效且正确加载的

那么实现我们的submitList方法：

public class PagedListDiffer<T> {  
 private List<T> oldData;  
  
 private ItemCallback<T> diffCallback;  
  
 public PagedListDiffer(@NonNull ItemCallback<T> diffCallback) {  
 }  
  
 public int getItemCount() {  
 return oldData.size();  
 }  
  
 public T getItem(int index) {  
 return oldData.get(index);  
 }  
  
 public void submitList(@Nullable List<T> newList) {  
 List<T> notSameData = new ArrayList<>();  
 // for each 比较 旧数据oldData与新数据newList 不同  
 // diffCallback.areItemsTheSame();  
 // 修改notSameData  
 appendData(notSameData);  
 }  
  
 private void appendData(List<T> notSameData) {  
 oldData.addAll(notSameData);  
 }  
}

3. 目前为止已经实现了数据的添加，那么如何加载更多数据呢，以及何时自动触发加载更多呢？

我们看Differ类，其中T getItem(int index) 方法返回数据，在获取数据的时候我们会触发加载更多，如果index已经到末尾了，是应该这个时候触发加载更多了，也就是说：

public T getItem(int index) {  
 if(index == (oldData.size() - 1) ) {  
 // 在末尾了，需要加载更多  
 //presenter加载更多  
 }  
 return oldData.get(index);  
}

此时的PagedListDiffer虽然知道了数据加载时机，但是没有presenter去执行加载任务，我们可以传递一个presenter去给differ执行，也可以包装到List中去加载，这里Android选择了后者，定义了PagedList类：

下面我们也实现我们的PagedList类：

public class PagedList<T> implements DataSource.LoadCallback<T> {  
  
 */\*\*  
 \* 真实列表数据  
 \*/* List<T> data;  
  
 */\*\*  
 \* 数据加载类  
 \*/* private DataSource dataSourceLoader;  
  
 public PagedList(DataSource dataSourceLoader) {  
 this.dataSourceLoader = dataSourceLoader;  
 }  
  
 public T get(int position) {  
 return data.get(position);  
 }  
  
 public void loadAround(int index) {  
 if (index == (data.size() - 1)) {  
 // 在末尾了，需要加载更多  
 // presenter加载更多  
 dataSourceLoader.loadMore(this);  
 }  
 }  
  
 public int size() {  
 return data.size();  
 }  
  
 @Override  
 public void onLoadMore(List<T> data) {  
 this.data.addAll(data);  
 }  
}

使用该类包装List数据，以及DataSource数据源，该数据源提供加载方法，给内部实现加载更多，并使用DataSource.LoadCallback提供回调该类实现数据的添加，如果我们的PagedListDiffer如下：

public class PagedListDiffer<T> {  
 private PagedList<T> oldData;  
  
 private ItemCallback<T> diffCallback;  
  
 public PagedListDiffer(@NonNull ItemCallback<T> diffCallback) {  
 }  
  
 public int getItemCount() {  
 return oldData.size();  
 }  
  
  
 public T getItem(int index) {  
 oldData.loadAround(index);  
 return oldData.get(index);  
 }  
}

public class DataSource {  
  
 public interface LoadCallback<T>{  
 void onLoadMore(List<T> data);  
 }  
  
  
 public <T> void loadMore(LoadCallback<T> callback) {  
 List<T> more = new ArrayList<>();  
  
 // load more  
 //....  
 //  
 callback.onLoadMore(more);  
 }  
}

此时完成简单的加载更多

PagedList 数据接口优化

使用DataSource加载的数据，通过append加载的数据我们可以使用简单的List去追加更多，但是很多时候我们想要更加详细的数据，比如分页次数，下面我们来看一下android大神如何记录分页数据的（更加详细的数据信息）

PagedList#List<T> data 数据集改造

我们定义一个类PagedStorage来记录详细分页列表数据

public class PagedStorage<T> {  
 private final ArrayList<List<T>> mPages;  
  
 // page count default 20  
 private int mPageSize = 20;  
 // total of mPages's items  
 private int mStorageCount;  
  
 public PagedStorage(ArrayList<List<T>> mPages) {  
 this.mPages = mPages;  
 }  
}

我们定义mPages记录每页的数据，mpageSize代表的是页大小，这里设置20，mStorageCount记录的是总数量

获取index数据

public T getItem(int localIndex){  
 int pageInternalIndex = 0;  
 int localPageIndex = 0;  
 final int localPageCount = mPages.size();  
 for (localPageIndex = 0; localPageIndex < localPageCount; localPageIndex++) {  
 //寻找页  
 int pageSize = mPages.get(localPageIndex).size();  
 if (pageSize > pageInternalIndex) {  
 // stop, found the page  
 break;  
 }  
 // 寻找页的index下标  
 pageInternalIndex -= pageSize;  
 }  
 List<T> page = mPages.get(localPageIndex);  
 if (page == null || page.size() == 0) {  
 // can only occur in tiled case, with untouched inner/placeholder pages  
 return null;  
 }  
 return page.get(pageInternalIndex);  
}

获取第一个元素与最后一个元素

T getFirstLoadedItem() {  
 return mPages.get(0).get(0);  
}  
  
T getLastLoadedItem() {  
 List<T> page = mPages.get(mPages.size() - 1);  
 return page.get(page.size() - 1);  
}

添加元素：

void prependPage(@NonNull List<T> page) {  
 final int count = page.size();  
 if (count == 0) {  
 // Nothing returned from source, stop loading in this direction  
 return;  
 }  
 mPages.add(0, page);  
 mStorageCount += count;  
}  
  
void appendPage(@NonNull List<T> page) {  
 final int count = page.size();  
 if (count == 0) {  
 // Nothing returned from source, stop loading in this direction  
 return;  
 }  
 mPages.add(page);  
 mStorageCount += count;  
}

添加到第一个位置/最后一个位置

下面衍生添加任意页位置

public void insertPage(int position, @NonNull List<T> page) {  
 if (position <= 0) {  
 prependPage(page);  
 } else if (position >= size()) {  
 appendPage(page);  
 } else {  
 // 只允许正确长度的page  
 if (page.size() == mPageSize) {  
 mPages.add(position, page);  
 }  
 }  
}

这样PagedStorage类就完成了，如果需要记录更加详细的信息，我们可以在PagedStorage中继续添加，并定义成员变量

优势：

在加载getItem中判断index是否到合适的位置需要加载更多了，数据由pagedList管理并使用dataSource加载

注意：

自定义需要实现异步加载，并添加同步回调，否则数据容易出错/APP ANR