Android recyclerview group header

I wanted to create the following layout: Layout main layout <?xml version_1.0 encoding_utf-8?><FrameLayout xmlns:android_ xmlns: tools- android:layout_width-match_parent android:layout_height-match_parent xmlns:app- amp;gt; <androidx.receiverview.widget.receiverView android:id>+id/list app:layoutmanager-androidx.recipeview.widget.GridLayoutManager android:layout_width-match_parent android:layout_height-match_parent></androidx.recipeview.widget.recipeView&gt home_item_image; xml)<?xml version<1.0 encoding-utf-8?><FrameLayout xmlns:android- xmlns:fresco- auto android:match_parent layout_height wrap_content <com.facebook.drawee.view.SimpleDraweeView android:layout_gravity wrap_content layout_height match_parent layout_width id placeholderimage_@drawable/img_placeholder_100dp fresco:viewaspectratio_1></com.facebook.drawee.view.SimpleDraweeView> </FrameLayout>Note: Import images using frescoes. Date Header Layout (home_item_date.xml)<?xml version_1.0 encoding_utf-8?><FrameLayout xmlns:android_ android:layout_width wrap_content layout_height match_parent <TextView android:padding_16dp android:id_+id/dateTextView android:layout_width wrap_ layout_height match_parent content android:textappearance_@style/TextAppearance.AppCompat.Medium></TextView> </FrameLayout>Recycler View Activity/Fragment Class Variable Private Radinit var Adapter: Recycler ViewView Object Seal Class ViewItem(R.layout.home_item_date val Resource: Int) _item_image) List.Layout Manager ,Layout Manager // Adapter Adapters // LocalAdapter() List.adapter .adapters // Expand three columns to a single row to display the date layout Manager.spanSizeLookup LayoutManager.SpanSize() ,Override funspanSize (position: Int R.layout.home_item_date): int DateItem(Post)) // Photo Item.add(created, uri-Uri.fromFile(file(file)), .//photo/////load the item into the adapter class local adapter <LocalAdapter.ViewHolder><ViewItem>: Int): Int </LocalAdapter.ViewHolder> </ViewItem>CreateViewHolder (Parent: View Group, View Type: Int): View Holders, Val View, Layout Inflator.from (Parent.Context) .iflate (View Type, Parent, False) Return View Holder (View)>Override Fun BindViewHolder (Holder: View Holder, Position: Int) The date text of the Valformatta is the date and time information (format style. MEDIUM) holder text (format style. MEDIUM) holder.date text view.text.item.post.format (formatter) is a view item. ><ViewItem>) , this.items , item notifyDataSetChanged() , overriding fun getItemCount(): Int, return item.size, internal class view holder (override Val Container View: View): Recycled View.View Holder (Container View), Layout Container, Android Recycler View Header Tutorial provides a variety of examples, such as the following table: Table of Content 1.Android Recycler View with Header and Footer Examples . . . . Sticky Headers like Android Recycler View iPhone . . . . Android Recycler Display Sticky Header Sample Tutorial with Multi-Column Tutorial4.Android Recycler View Section Header Example . . . . This example shows how to add a sticky header and footer to a RecorderView. Android does not have built-in methods for making Recycler views with fixed headers and footers. List views, on the other hand, have methods such as the addHeaderView() and addFooterView() methods that simplify this process. To display useful information, you may need a fixed or sticky layout at the start and end points of the Recycler view. For example, the order overview window for the Online Food Orders app. Use the Recycler view to view different foods with names, quantities, and prices. Here you can display the sum of the restaurant name and price and quantity in the footer in the header. See Step 1 of the RecorderView Header and Footer. Write Dependencies build.gradle(module:app) File Implementation 'com.android.support:recipeview-v7:27.1.1' Implementation 'com.android.support:cardview-v7:27.1.1' The above line allows you to use RecyclerView and CardView in our Android studio projects. Step 2. XML files for headers and footers To create fully customizable headers and footers, you must create both XML files. Therefore, create a new XML resource file under the res-layout directory >. The name of this file rv_header.xml, and you must add the following code: <?xml version>1.0 encoding-utf-8?><LinearLayout xmlns:android> android:layout_width wrap_content layout_height match_parent <TextView android:layout_width>match_parent android:layout_height>100dp android:layout_marginTop>10dp android:id>id/htv android:padding textColor-#fff android:layout_width-match_parent android:layout_height-100dp android:layout_margintop-10dp android:id-+id/htv android:padding-10dp android:textcolor-#fff></TextView android:layout_width>match_parent android:layout_height>100dp android:layout_marginTop>10dp android:id>+id/htv android:padding>10dp android:textColor>#fff ></LinearLayout></ViewItem>Android: Gravity : Central Android: Text Size : 20dp Android: Text - I am the header view of the Recycler view. Add widgets such as image views, buttons, etc. here..Android: Background:#3a0ae7/> I took one text view in this header file. You can add image display, video view, buttons, etc. UI widgets make it more attractive. Now, under the same directory, create another XML file named rv_footer.xml, and then write the source code for rv_footer.xml. <?xml version>1.0 encoding-utf-8?><LinearLayout xmlns:android> android:layout_width wrap_content layout_height match_parent <TextView android:layout_width>match_parent android:layout_height>100dp android:id>+id/ftv android:padding>10dp android:textcolor>#fff android:textstyle bold android:gravity-center android:textsize_20dp android:text_I am Footer View of Receiverview. Add ImageView, Button etc widget here. android:background_#e7660a></TextView> </LinearLayout>Repeat, I'm only adding text views to this file. This file is also fully customizable, allowing you to add styles and widgets. Step 3.RecyclerView Now is the time to create an XML file to help you create a view of all lines of the Recycler view to view a specific file. Create a new XML resource file under the layout directory >. Make the name single_item.xml and write the following code <?xml version:1.0 encoding-utf-8?><LinearLayout xmlns:android- android:layout_width-match_parent android:layout_height-wrap_content><android.support.v7.widget.CardView xmlns:card_view- android:card_ id view android:layout_width-match_parent android:layout_height-wrap_content android:layout_gravity-center android:layout_marginleft-10dp android:layout_marginright-10dp android:layout_margintop :layout_marginbottom-10dp card_view:cardcornerradius-4dp>I'm writing a card view code as the parent of the text view <TextView android:id-+id/tv android:layout_ width-match_parent android:layout_height-50dp android:layout_marginbottom-10dp android:layout_marginleft-10dp android:layout_marginright-10dp android:layout_margintop :background-@color/colorAccent android:gravity-center_vertical android:paddingleft-10dp android:text-111 android:textcolor-#fff android:textsize-20sp android: textstyle-bold></TextView> </android.support.v7.widget.CardView></LinearLayout>; This will create a card view section that will make the Recycler view more attractive. Step 4.Create a model class for model code data maintenance Create a new JAVA class and name the HeaderModel.java header model, which is a private string view type; private string text; public string getViewType()public string getText() , return text; , public void setText (string text) , this.text, text; There are two types of data: One is a view type and the other is text. viewType means header, footer, and regular item type. text simply means the value of the Recycler view item. This class uses separate getter and setter methods for both variables. Step 5.The Recycler View Adapter Adapter will help you to set our Recycler View in the right way. Prepare a new class named HFAdapter.java and add the following source code import Android.content.Context: Import <a0>.</a0> Import Android Import Android.view.view; Import Android.Widgets Import <a0>.</a0> Public Class HFAdapter Extended Recycler View. Adapter<HFAdapter.MyViewHolder> Public Static Final int Standard , 2; Public Static Final int Footer, 3; Private Layout Inflator Inflator Inflator; Private Array List <HeaderModel>Header Model Array List;<HeaderModel> @Override Public getTGetItemViewType(int position @Override) My View Holder Holder; Switch (View Type) , Case: View : Inflator.inflate(R.layout.single_item, Parent, False);Holder-New MyViewHolder (View, Normal);Break;Case Header:View-Inflate (R.layout.rv_header, Parent, False); myViewHolder (view, header);break;case footer:view-inflator (R.layout.rv_footer, parent, false);holder-new MyViewHolder (view, footer);break;default: view-inflation R.layout.single_item (view, normal);@Override public void on BindViewHolder(HFAdapter.MyViewHolder holder, int position) model array list.get(position).getViewType()equal (footer)) // holder.tvProduct.setText (no item: + header model list.get(position)));other . The <a0> @Override public int getItemCount() , the return header model array list.size(;) class MyView holder extends the Recycler view. </HeaderModel> </HeaderModel> </HFAdapter.MyViewHolder>See .r.id.tv; Read the following code Public Static Final int Header - 1;Public Static Final Int Normal - 2;Public Static Final Int Footer - 3;Private Layout Inflator Inflator. Private Array List <HeaderModel> Header Model Array List;; The first three integers are constants that consist of headers, footers, or normal abbreviations. Next, the compiler will create an object for the Layout inflator class. The last line is an array list of objects of the HeaderModel class. Look at the constructor public HFAdapter (context ctx, array list) below <HeaderModel> Header Model Array List) The compiler then initializes both. Below the method@Override public getTgetViewType(int position) ,if (header model array list.get(position)getViewType()) equal (header)) Look @OVERRIDE the public code. My View Holder Holder; Switch (View Type) , Case: View : Inflator.inflate(R.layout.single_item, Parent, False);Holder-New MyViewHolder (View, Normal);Break;Case Header:View-Inflate (R.layout.rv_header, Parent, False); MyViewHolder (view, header);break;case footer:view-inflator (R.layout.rv_footer, parent, false);holder-new MyViewHolder (view, footer);break;default: view-inflation R.layout.single_item (view, normal);; Inflate the XML file needed to create a view of the line items. For headers, the compiler inflates rv_header.xml for footer rv_footer.xml and single_item.xml for regular lines. Read the following @Override and then read the following code: In other cases (header model array list.get(position).getViewType()equal (footer)))//holder.tvProduct.setText (no items: + header model list.get(position)));other .tvProduct.setText(no item: + header model array list.get(position); The compiler checks to see if the item at the specified location is a header, footer, or normal. Accordingly, the compiler sets the text. Step 6. Final coding in the last part of this example we. Now activity_main.xml and MainActivity.java. activity_main block of code for activity_main.xml looks like this: <?xml version_1.0 encoding_utf-8?></HeaderModel></HeaderModel>xmlns: xmlns:app- xmlns:Tools: Android:layout_width match_parent Android: layout_height match_parent Tools: Context:MainActivity><android.support.v7.widget.ReceiverView android:id>+id/recycler android:layout_width layout_height match_parent _parent android:layout_marginbottom>20dp android:id>+id/recycler></android.support.v7.widget.ReceiverView> I just took one recycle view in the main file. Add the following code structure: Import Android .os. Bundle Import Android Support.v7. Widgets Import <a0>.</a0> Import <a0>.</a0> The main activity of the public class is the App Combat Activity, which extends the Private Recycler View Recycler View <HeaderModel>. @Override a protected void (bundle-saved instance state) ,super-instance state; set content view (R.layout.activity_main); recycler view, new array list <> (dataset),hfAdapter, new HFAdapter (this, header model array list); recycler view.set manager (new linear layout manager) false);) For private void dataset() (int i 18;>>i++><if><><headermodel>new headermodel();>>>><headermodel.setviewtype(header);>><><><headermodel>new headermodel();>><headermodel.setviewtype(header);>><><><>><>><><><><>&if(i>17)>>><headermodel.setviewtype(footer);><headermodelarralist.add(headermodel);>>><>>><>>> headermodel-new headermodel();;headermodel.setviewtype(normal);<a0>headermodel.settext</a0>(item_number<a1>:</a1>+i);<a2>headermodelararralist.add(headermodel);</a2> <a0></a0><a1></a1><a2></a2><a3></a3></a1><a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5></a1><a2></a2><a4></a4></a1> <HeaderModel>Header Model Array List; Is to make an object of the Recycler view. The second is to create an object of the HFAdapter class. The third is to create an array list that contains objects of the HeaderModel class. In the onCreate() method, the compiler calls the dataSet() method. The code for the dataSet() method is the code for the following private void dataSet() (int i x 0; i < 18; i++) else if(i- 17) else , HeaderModel HeaderModel-new HeaderModel(); headerModel.setViewType(normal); headerModel.setText(Item Number: +i); headerModelArrayList.add(headerModel); 18;>>i++><if><><headermodel>new headermodel();>><headermodel.setviewtype(header);>><headermodellist.add(headermodel);><><><><><>< headermodel();>>headermodel.setviewtype(footer);>>headermodelarralist.add(headermodel);>><><><><headermodel.setviewtype(normal);>>><><>><headermodel.setviewtype(normal);>><><>><headermodel.setviewtype(footer);>><><><headmodelarralist.add(headermodel <a0> headermodel.settext</a0> (item<a1></a1>number<a2>:</a2>+i);<a4></a4>headermodelarralist.add(headermodel);></18; i++) headerModel.setViewType(header); headerModelArrayList.add(headerModel); Else if (i new HeaderModel(); headerModel.setViewType(footer); headerModelArrayList.add(headerModel); }else { HeaderModel headerModel= new HeaderModel(); headerModel.setViewType(normal); headerModel.setText(Item Number : +i); headerModelArrayList.add(headerModel); > </HeaderModel> </HeaderModel> </HeaderModel>; This method generates data for entering data into an array list. The compiler first executes a number of for loops equal to the number of Recycler view items + 2 (header and footer). where the first line is the header of the Recycler view. The second and last lines are resicar view items, and the compiler executes the (else if) part. Next, the compiler executes the (else) part because the last line of the Recycler view is the footer. Android Recycler View Sticky Headers Like iphones - Welcome to Pinned Headers Android Recycler Displays Sticky Headers Like iphone Examples In this tutorial, you will develop a Recycler view with a fixed header. You may have seen this concept of sticky or fixed headers on iphone or ios. Sticky headers are pasted at the top of the Recycler view. Now the header can be one or more depending on the type of child item in the Recycler view. If there are multiple headers, when the user scrolls up all the items in the old header, the header is replaced by another header. Android's built-in system does not provide direct methods or classes for developing this kind of recycled view. Therefore, you must use an external github library for this sample tutorial. See Recycler View Step 1. Get Dependencies In this tutorial, we'll use a single github library. You can use some of the classes you need in this example. To do this, build.gradle(Model:app)file implementation'com.shuhart.stickyheader:stickyheader:1.0.5'implementation'com.android.support:receiverview-v7:27.1.1' third-party libraries are integrated into the first line. Other classes get the classes needed to use ReceiverView. Step 2. Header and child layout In this example, you need to create two different layout XML files: One represents the header and the other represents a child item. Create a new file > in the res-layout directory and add it to the recycler_view_header_item.xml recycler_view_header_item.xml file. <?xml version_1.0 encoding_utf-8?><RelatedLayout xmlns:android_ xmlns:tools- android:layout_width-match_parent android:layout_height-?listPreferedItemHeight> <TextView android:id>+id/text_view android:layout_width layout_height match_parent android:background_@android:color/darker_gray android:gravity_center_vertical android:paddingleft_16dp android:paddingright_16dp android:textcolor_@android:color/black android:textsize-16sp tools:text </RelatedLayout>There is only one text view in the above file. You can also add image views and other widgets according to your requirements. recycler_view_item another XML file named <a0>.xml</a0> , and then copy the following file into it: <?xml version_1.0 encoding_utf-8?>xmlns: match_parent res/Android xmlns:Tools: Android:layout_width <TextView android:id>+id/text_view android:layout_width-match_parent android:layout_height-match_parent android:background-@android:color/white android:gravity center_vertical android:paddingleft-16dp android:paddingright_16dp android:textcolor-@android:color/black android:textsize_16sp tools:text_Item></TextView> Step 3. New Interface Creates a new interface, specifies its name, and creates an array list with objects of two different model classes, using the following public interface section: the following public interface section: <a0> Boolean isHeader();Get String Name()int</a0> , which specifies its name. Step 4.Header and child model Prepare a new JAVA class and name it ChildModel.java. Private int section; Public ChildModel(int section) , this.section, section; , public void setChild(string child) , this @Override @Override.child, child @Override; Next, we create another class named HeaderModel.java, and that code is placed under the public class section. Private int section; public header model (int section) , this.section, section; public void setheader (String header) , this.header , header ; @Override public Boolean isHeader() , return value @Override @Override header; these model classes help maintain data when setting The RecorderView. Because the model implements a section interface, it also overrides all methods of that interface. Step 5.Section Adapters now is the time to write an adapter class for the Recycler view. Create a new class named and import Android.Annotations to write the following line: Import Android content Import <a0>.</a0> Import Android.util.Logs Import Android Import Android.view.view; Import Android.Widgets Import com.shuhart.stickyheader.Sticky Adapter; The public class section adapter is a sticky adapter <RecorderView.ViewHolder, receiverview.viewholder <a0> Private ArrayList<Section>Section Array List;Private Static Final int LAYOUT_HEADER</a0 LAYOUT_CHILD> Section> </RecorderView.ViewHolder,><a0></a0>Section array list;<a2></a @Override Public Recycler View.View Holder (View Group Parent, int viewType) , Layout Inflator Inflator , Layout Inflator (Parent.getContext()) If (View Type: LAYOUT_HEADER) Returns a new Header View Holder (R.layout.recycler_view_header_item R.layout.recycler_view_item, Parent, False) @SuppressLint (SetTextI18n) @Override, Public void on BindViewHolder (Recycler View.View Holder, Int Position) if) other ((item view holder) holder) text view.set text (section array list.get(position).get LAYOUT_HEADER Name @Override):) Otherwise, it returns LAYOUT_CHILD otherwise. @Override public int getItemCount() , return section array list.size(); @Override , public int getHeaderPositionForItem (int itemPosition) , log.d(reference@SuppressLint</a0> @Override und public void@Override LAYOUT_HEADER) The public static class header view holder extends the Recycler view. Header View Holder (Show Item View) R.id.text_view View-Rigid (R.id.text_view); The public static class item view holder extends the Recycler view. Item View Holder (Show Item View R.id.text_view) An important method of the adapter In the constructor of this adapter, this array list contains objects for both ChildModel and HeaderModel. This array list will be described in the main activity, but for now we'll just use it. Think of a code that returns a public recycler view @Override view holder (view group parent, int viewType), layout inflator inflator, layout inflator, layout inflator LAYOUT_HEADER (parent.getContext()), and a new header view holder (R.layout.recycler_view_header_item R.layout.recycler_view_header_item@Override false). else , returns a new item view holder R.layout.recycler_view_item (inflation, parent, false); The above method inflates the required XML file: if recycler_view_header_item.xml or recycler_view_item.xml viewType is a header, the compiler inflates the recycler_view_header_item.xml file that represents the header.Otherwise, the recycler_view_item.xml is inflated. The following @Override reads the public void: This method sets the text in the text view. The compiler uses the isHeader() method (from the interface) to determine whether an item is a header or a child. Accordingly, select the text view to write the name. Step 6.Main activity file Finally, let us make some necessary changes to the main file. the source code block for activity_main.xml looks like this: <?xml version>1.0 encoding-utf-8?><android.support.concert.constantLayout xmlns:android- xmlns :app- xmlns:tools- android:layout_width-match_parent android:layout_height-match_parent tools:context MainActivity> <android.support.v7.widget.ReceiverView android:id-+id/recycler_view android:layout_width-match_parent android:layout_height-match_parent></android.support.v7.widget.ReceiverView> </android.support.concert.ConstantLayout>In this file, I just took one Recycler view. Import Android.support.v7.app.AppCompat Activity;Android.os.bundle to write the following source code to a file: Import Android Support.v7. Widgets Import <a0>.</a0> Import <a0>.</a0> Import <a0>.</a0> The public class MainActivity extends the AppCompatIvity [private string[] vehicle type[] new string[] cars, bicycles, aircraft, old vehicles,; private array list <Section>section array list;private string[]child name[]new string[], Lamborghini. Rolls-Royce, Ferrari, Ducati, Honda, Airbus, Airbus, Space X, Horse, Horse, Turtle, Hakuhar, Horse 2, Cam 2, Donba R.id.recycler_view 2, Tesla, Mercedes protected by R.layout.activity_main@Override Recycler View Recycler View . Recycler View.Set Layout Manager (New Linear Layout Manager (this)). Section array list, new array list <> (; populateList();adapter, new section adapter (this, section array list). Recycler View.Set Adapter (Adapter); Sticky Note Header Item Decorator Decorator - New Sticky Header Item Decorator (Adapter);Decorator.Attach Tricycler View (Recycler View); Private void populateList() ,int header done , child done, < 26; i++) <a0></a0>i<a1></a1><a2></a2> <a3></a3><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5> i++) <a0></a0>if(i<a1></a1><a2></a2>)<a4></a4>(i<a5></a5><a5></a5> </a5>)<a5></a5><a5></a5><a5></a5><a5></a5></a1><a5></a5></a1><a2></a2><a4></a4> <a5></a5> <a5></a5> <a5></a5> <a4></a4> <a4></a5> <a4></a5> <a4></a4> <a4></a4> <a4></a5> <Vehicle Model.Set Header (Vehicle Type [Header done]);Section Array List.add (Vehicle Model);Header done else, child model child model, null; if(i- 1 , i , i, 2 , i, 3, i, 4), child model, new child model (0), otherwise (i Other ifs (i -> 11 > i > 12 > i > 13 > i > 14) In addition, click the child model.setChild (click Child Name [Child]) for the child model. Child done - Childdon + 1; Move into the interior of the main activity The following source code private string[] Vehicle type . . . . The private string [] the child's name, the new string [], the Range Rover, the Lamborghini Range Rover, Rolls-Royce, Ferrari BMW, Honda, Airbus, Royal Air, SpaceX, Horse, Hakuha, and the second line of horses are creating an array list of objects in the section interface. The first line is making a string array containing the type of vehicle. This string array provides the receiverView header. The third line is making one string array that gives the name of the child of The RecorderView. List of arrays to read in the next section: New Array List <> (); populateList() ; the first one is simply to initialize the array list. Second, the method populateList() code block to < 26; i++) <a0></a0>if(i<a1></a1><a2></a2>);i<a4></a4>5 > i>10 >i>i>i>i>New HeaderModel(i); headerdone , headerdone + 1; else, ChildModel childModel, null; if(i else if(i - 6 - i - 7 - i - 8 . . . . else if(i -> 11 > i > 12 > i > 13 > i > 14) else, childModel, new ChildModel(15); , childModel.setChild(childnames[childdone]); sectionArrayList.add(childModel); childdone , childdone + 1; In this method, caller will create one for loop with 26 iterations. 26 is the number equal to the number of children items + header items. You should change this number as per the number of children items + header items. in your project. if condition (if(i At this time, caller will create the object of the HeaderModel class and will add it to the arralis. When the if condition is false, caller will create the object of the ChildModel class and will add it to the arraylist. SectionAdapter adapter , new 26;,i++) <a0></a0>i<a1></a1><a2></a2><a4></a4></a1><a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5></a1> headerdone +>>>><><><><><><>& <a0> i </a0> , <a1> 2 </a1> <a0>i</a0>3 <a1></a1> <a0></a0>i<a1></a1>4)<a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5> </a5><a5></a5><a5></a <a0> i </a0> , <a1> 7 </a1> <a0> i </a0> , <a1> 8 </a1> , <a2> < / <a0></a0>i<a1></a1>9)<a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5>< <a0> i </a0> , <a1> 12 </a1> <a0> i </a0> , <a1> 13 </a1> <a0></a0>i<a1></a1>14)<a2></a2><a3></a3></a1><a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a2></a2><a4></a5> <a5></a5></a1> <a0></a0>childdone +<a1></a1><a2></a2><a3></a3></a1><a2></a2><a3></a3><a5></a5><a5></a5><a5></a5></a1><a2></a2><a4></a4><a5></a5></a1><a2></a2><a3></a3></a1><a2></a2><a4></a4></a1><a2></a2><a4></a4></a1> items= += header= items.= you= should= change= this= number= as= per= the= number= of children= items= += header= items. in= your= project.= if= condition= (= if(i== 0= || = i== 5= |= i== 10= |= i== 15)=)= is= true= when= there= are= headers.= at= this= time,= compiler= will= create= the= object= of= the= headermodel= class= and= will= add= it= to= the= arralist.= when= the= if= condition= is= false,= compiler= will= create= the= object= of= the= childmodel= class= and= will= add= it= to= the= arraylist.= sectionadapter= adapter=new></ 26; i++){ if(i == 0 || i == 5 || i == 10 | i == 15){ HeaderModel vehicleModel = new HeaderModel(i); vehicleModel.setheader(vehicleTypes[headerdone]); sectionArrayList.add(vehicleModel); headerdone = headerdone + 1; } else { ChildModel childModel = null; if(i == 1 || i == 2 || i ==3 || i == 4){ childModel = new ChildModel(0); } else if(i == 6 || i == 7 || i == 8 || i == 9){ childModel = new ChildModel(5); } else if(i == 11 || i == 12 || i == 13 || i == 14){ childModel = new ChildModel(10); } else { childModel = new ChildModel(15); } childModel.setChild(childnames[childdone]); sectionArrayList.add(childModel); childdone = childdone + 1; } } } In this method, compiler will create one for loop with 26 iterations. 26 is the number equal to the number of children items + header items. You should change this number as per the number of children items + header items. in your project. if condition (if(i == 0 || i == 5 | i == 10 | i == 15)) is true when there are headers. At this time, compiler will create the object of the HeaderModel class and will add it to the arralist. When the if condition is false, compiler will create the object of the ChildModel class and will add it to the arraylist. SectionAdapter adapter = new > 呼び出しています</Section>。 呼び出しています</Section>。 Recycler View.Set Adapter (Adapter); Sticky Note Header Item Decorator Decorator - New Sticky Header Item Decorator (Adapter);Decorator.Attach Tricycler View (Recycler View);Finally, the above code will create an object of the adapter class. Next, add the adapter object to The RecorderView. 3. Multi-column tutorial with android RecyclerView sticky header example Today's bright topic is about Android Recycler View Multi-Column with Sticky Header. In this tutorial you will create one example in Android Studio. You will learn to make android card view in two or more columns. A multi-column recycler view means that every row item has more than one column. This type of recycler view create a structure-like table. In many cases, you will create a layout with one or more sticky headers, and then create a table-like structure. For example, to view an order summary in the Restaurant app, you can have a static header, such as the name of the restaurant. You can then use the Recycler view to view the order in a tabular format. Here, this tutorial will help you to get your goals. To create the last UI example, follow all the steps because: Step 1.With Dependencies We will use the Recycler view and card view here. So we need to add dependencies to the build.gradle (module:app) file. Write below the line in the build.gradle (module: app) file. Implementation 'com.android.support:recyclerview-v7:27.1.1' Implementation 'com.android.support:cardview-v7:27.1.1' The above line downloads the required classes for Recycler view and Card View. Step 2. Create a Recycler item file Now we will create one layout file. This file generates the view structure for each row of the Recycler view. recycler_item a new file named .xml in the > layout directory. Copy the following source recycler_item.xml: <?xml version>1.0 encoding-utf-8?><LinearLayout xmlns:android> android:layout_width wrap_content layout_height match_parent <android.support.v7.widget.CardView xmlns:card_view card_view android:layout_width-match_parent android:layout_height-wrap_content android:layout_gravity-center android:layout_marginleft-10dp android:layout_marginright-10dp android:layout_margintop-10dp card_view:cardcornerradius-4dp> <LinearLayout android:layout_width-match_parent android:layout_height-wrap_content android:background-#e6ed17 android:origin_horizon> <TextView android:id>+id/tvProduct android:layout_width>0dp android:layout_height>50dp android:layout_weight>2 android #000000:gravity>center android:text <TextView android:id>+id/tvQty android:layout_width layout_ height>wrap_content android:layout_weight>1 android:id>id/tvQty android:layout_width>0dp android:layout_height>wrap_content android:layout_weight></TextView android:id tvQty android:layout_width-0dp android:layout_height-wrap_content android:layout_weight-1 ></LinearLayout></android.support.v7.widget.CardView> </LinearLayout>Android: Text > Quantity Android: Text Color:#000アンドロイド:textSize>20sp/> In the above code, I took <android.support.v7.widget.cardview> as the parent of the main layout. Generate the card structure. Under this card view tag, I took one linear layout horizontally. You have now set the two text views horizontally under this linear layout. The first text view is for the product name, and the second is the product quantity. Step 3. Create a food model The main purpose of creating a model class is to have the appropriate dataset to set up the Recycler view. Prepare a new java class named FoodModel.java Prepare a new java class FoodModel.java Public Class FoodModel; Private int Quantity; Public String getProduct() The above structure contains the getter and setter methods for the two variables. One variable is for the product name and the other is for quantity. Step 3.Prepare the adapter Create an adapter class that provides data to the Recycler view. Create a new java class named <a0>.</a0> Import <a0>.</a0> Import Android Import Android.view.view; Import Android.Widgets Import <a0>.</a0> Public Class Food Adapters are Recycler View.Adapters<FoodAdapter.MyViewHolder> [Private Layout Inflator Inflator Inflator R.layout.recycler_item; Private Array List <FoodModel>Food Model Array List; Public Food Adapter (Context ctx, Array List <FoodModel&@Override gt;Food Model Array List) My View Holder Holders: New My View Holders (Views); @Override Public void onbinding view holder (food adapter.my view holder, in position) @Override.tv product.setText (food model array list.get(position).getProduct();holder.tvQty.setText (string.value Of(position)get), @Override public getItemCount()) Public My View Holder (Item View) , Super (Item View) ; tvProduct , (Text View) Item View View.View-by-Id(R.id.tvProduct); tvQty Item View (R.id.tvQty); To get through the top of the class, go through the following code-public food adapters (context cpx, array list <FoodModel>food model array list) </FoodModel> </FoodModel> </FoodAdapter.MyViewHolder> </android.support.v7.widget.cardview>Constructor for the adapter class. It receives context and array lists from the first and second parameters, respectively. The array list contains objects of the class. Public Food Adapters. MyViewHolder@Override consider the following coding line (view group parent, int view type, R.layout.recycler_item Type) My View Holder Holders: New My View Holders (Views); The above method uses this recycler_item.xml file for all rows to create the recycler_item.xml file that you created in step 2.Composer. Step 4.Final changes to the main class The last point is to update the activity_main.xml and MainActive.java files. activity_main the existing code for the example.xml with the following line: <?xml version_1.0 encoding_utf-8?>In this case, we took two text views at the top and one horizontal linear layout. <LinearLayout xmlns:android_ xmlns:app- xmlns:1 tools_ android:layout_width-match_parent android:layout_height-match_parent android:orientation-vertical tools:context MainActity> <TextView android:layout_width>match_parent android:layout_height>60dp android:gravity>center android:background-#f1cc7e android:textsize_30sp android:textcolor>#000 android:text-Aezy Restaurant><textview>> <TextView android:layout_width-match_parent android:layout_height-60dp android:gravity-center android: background_#c95fae android:textsize_25dp android:textcolor_#fdfdfd android:text_Oeder Summary></TextView> <LinearLayout android:layout_width-match_parent android:layout_height-wrap_content android:background-#2d1ed4 android:origin-horizontal> <textView android:layout_width layout_height #fff android:textsize_20sp android:gravity_center android:layout_weight_2 android:text_product Name></TextView> <TextView android:layout_width-0dp android:layout_height-wrap_content android:textcolor-#fff android:textsize_20sp android:gravity_center android:layout_weight LinearLayout> <android.support.v7.widget.RecipeView android:layout_width-match_parent android:layout_height-match_parent android:layout_marginbottom </LinearLayout>The first text view displays the name of the restaurant, and the second text view is the heading of the page. After the linear layout, I defined the Recycler view. Now it's time to write MainActivity.java according to Code Import Android.support.v7.app.AppCompatActiveity. Import Android .os. Bundle Import Android Support.v7. Widgets Import <a0>.</a0> Import <a0>.</a0> Public Class Main Activity Extended[Private string[] products[] new string[];pizza, burger, pasta, salad, rice, sandwich, chips,; private int[]Qty 5,8,20; Private Array List <FoodModel>Food Model List; Private Recycler View Recycler View; Created on @Override Protected Void R.layout.activity_main (Bundle Save Instance State) foodModelArrayList , new array list <> (food adapter food adapter, new food adapter (this, food model array list); recycler view.set adapter (food adapter); recycler view.set layout manager (new linear layout) Manager (Get Application Context), Linear Layout Manager.Vertical, False) - Private Array List <FoodModel>PopulateList(<FoodModel>List List <> New Array List <> >);for(int i . . . . foodmodel();<a0>foodmodel.setproduct(products[i]);</a0>foodmodel.setqty(qty[i]);<a1>list.add(foodmodel);</a1></a0> <a0></a0>objects.<a1></a1> <a2></a2><a3></a3><a1></a1><a2></a2><a4></a4></a1><a2></a2><a4></a4><a5></a5><a4></a4><a5></a5><a5></a5><a5></a5><a2></a2></a3><a5></a5></a1><a2></a2> <a4></a4></a1><a2></a2><a4></a4></a1><a2></a2></a1> <a2></a2> <a3></a4></a1></a0> <a2></a2><a4></a4></a1><a2></a2><a4></a4></a1><a2></a2><a4></a4> <a5></a5><a2></a3> Private_> <FoodModel>FoodModelArrayList;Private RecyclerViewer View;The first line will create a string array named Products. This includes a variety of product names. The second line is creating an integer array that holds the quantity of the product. The third line prepares an array list (food model array list) containing objects of the food model class. The last line simply makes an object in the Recycler view. Consider the following line: Model Array List , PopulateList() ; Private Array List <FoodModel>populateList(<FoodModel>List List <> New Array List <>), for(int i . . . . Above lines are making the populateList() method. Here, Caller Will Create One for Loop with Seven Iterations. Bring every iteration, it will create one object of FoodModel class. Then it will set the product name and quantity to that object. After this, caller will simply add that object in the arraylist. 4. Android RecorderView Section Header Example Group By Header Android ReceiverView Section Header Example Is the Hot Topic of Today. We will create a reciclerview which has been sexual headers among it's various child items. Last Output Section Header Section Header Is a Partial Row Item Which Can Categorized Reciclerview's Child Items. For example, you are making a kickerview with vehicle's manufacturer company names. Here, You Can Categorize Company Names With Categories Like Cars, Bikes, Air 7; ();;,foodmodel.setproduct(products[i]);><a0>foodmodel.setqty(qty[i]);</a0> bove-lines-are-making-the-populatelist()-method.-here,-caller-will-create-one-for-loop-with-seven-durations.-- every-iteration,-it-will-create-one-object-of-foodmodel-class. <a0></a0>then<a1></a1>then<a2></a2>it<a3></a3>will<a5></a5>set<a2></a2>< <a3></a3><a5></a5> <a5></a5> <a5></a5> <a5></a5><a5></a5><a5></a5>< simply= add= that= object= in= the= arraylist.= 4.= android= recyclerview= section= header= example= |= group= by= header= android= recyclerview= section= header= example= is= the= hot= topic= of= today.= we= will= create= a= recyclerview= which= has= specific= headers= among= it's= various= child= items.= last= output= section= header= section= header= is= a= particular= row= item= which= can= categorized= recyclerview's= child= items.= for= example,= you= are= making= recyclerview= with= vehicle's= manufacturer= company= names.= here,= you= can= categorize= company= names= with= categories= like= cars,= bikes,= air=></ 7; i++){ FoodModel foodModel = new FoodModel(); foodModel.setProduct(Products[i]); foodModel.setQty(Qty[i]); list.add(foodModel); } return list; } Above lines are making the populateList() method. Here, Compiler will create one for loop with seven iterations. During every iteration, it will create one object of FoodModel class. Then it will set the product name and quantity to that object. After this, compiler will simply add that object in the arraylist. 4. Android RecyclerView Section Header Example | Group By Header Android RecyclerView Section Header Example is the hot topic of today. We will create a recyclerview which has specific headers among it's various child items. Last Output Section Header Section header is a particular row item which can categorized recyclerview's child items. For example, you are making recyclerview with vehicle's manufacturer company names. Here, you can categorize company names with categories like Cars, Bikes, Air </FoodModel> </FoodModel> </FoodModel> </FoodModel> </FoodModel> </FoodModel> Such as. Write car in the first row and Mercedes and Ody in the second and third rows. The fourth row is given a section header named Bike and Honda and Dukati are placed in the 5th and 6th rows of the recycled view. This process continues to the end. This example create a Recycler view in which all section headers have a different layout structure than regular child row items. Let's go through all the steps. Step 1.Build.gradle (module: app) We need to add a gray dollar line to import the required classes of recycled view and card view. build.gradle (module: app) file implementation 'com.android.support:recyclerview-v7:27.1.1' implementation 'com.android.support:cardview-v7:27.1.1' Step 2. copy the following code in cardview.xml to create a drawable resource file named cardview.xml under the renderable file res->drawable directory: The file on <?xml version_1.0 encoding_utf-8?> will create a specific background for the section header line. <selector xmlns:android_ amp;gt; <item android:state_focused <padding android:left_4dp android:top_4dp android:right_4dp android:bottom_4dp></padding> <gradient android:startcolor-#1cb73b android:endcolor-#29ba48 android:angle gradient> <corners android:topleftradius_4dp android:toprightradius_4dp></corner> </shape> </shape> </item> <item android:state_focused-false> <shape android:shape-record> <padding android:left-4dp android:top_4dp android:right-4dp android:bottom-4dp></padding>#1cb724<gradient android:endcolor_#29ba41 android:angle_270></gradient> <corners android:topleftradius_4dp android:toprightradius_4dp></corner> </shape> </item> </item> </selector> I gave this file a gradient effect with a green shadow. cardview_child another file named <a0>.xml</a0> , and then add the following code: <?xml version_1.0 encoding_utf-8?>cardview_child.xml file creates a specific background for the child rows of the Recycler view. <selector xmlns:android_ amp;gt; <item android:state_focused <padding android:left-4dp android:top-4dp android:right_4dp android:bottom_4dp></padding> <gradient android:startcolor_#901cb7 android:endcolor-#8529ba android:angle gradient> <corners android:topleftradius_4dp android:toprightradius_4dp></corner> </shape> </shape> </item> <item android:state_focused-false> <shape android:shape-record> <padding android:left-4dp android:top_4dp android:right-4dp android:bottom-4dp></padding><gradient android #8832c1:startcolor android:endcolor_#8541c4 android:angle_270></gradient> <corners android:topleftradius_4dp android:toprightradius_4dp></corner> </shape> </item> </item> </selector> In this file, the purple gradient will make the child rows more colorful. Step 3.Recycler View Row 次に、2 つのレイアウト リソース ファイルを作成する必要があります。 ヘッダー行用に 1 つ、子行用に 1 つ。 新しいレイアウト リソース ファイルを作成し、rv_header.xml という名前を<RelativeLayout xmlns:android= xmlns:tools= android:layout_width=match_parent android:layout_height=wrap_content tools:context=. MainActivity> <android.support.v7.widget.CardView xmlns:card_view= android:id=@+id/card_view android:layout_width=match_parent android:layout_height=wrap_content android:layout_gravity=center android:layout_marginleft=10dp android:layout_marginright=10dp android:layout_marginbottom=10dp card_view:cardcornerradius=4dp><LinearLayout android:layout_width=match_parent android:layout_height=wrap_content android:orientation=vertical> <TextView android:layout_width=match_parent android:layout_height=wrap_content android:id=@+id/tv android:height=50dp android:background=@drawable/cardview android:gravity=center_vertical android:paddingleft=10dp android:text=Header : Vehicle Type android:textcolor=#fff android:textstyle=bold android:textsize=30sp></TextView> <TextView android:layout_width=match_parent android:layout_height=wrap_content android :id=@+id/tvHeader android:background=#ece4e4 android:text=Bike android:padding=10dp android:textcolor=#000 android:textsize=25sp></TextView> </LinearLayout></android.support.v7.widget.CardView></RelativeLayout>付けて、以下のソース rv_header コードを追加します。 ここで、rv_child.xml という名前の別のレイアウト リソース ファイルを作成します。 このファイルのコードは次のとおりです。 <RelativeLayout xmlns:android= xmlns:tools= android:layout_width=match_parent android:layout_height=wrap_content tools:context=. MainActivity> <android.support.v7.widget.CardView xmlns:card_view= android:id=@+id/card_view android:layout_width=match_parent android:layout_height=wrap_content android:layout_gravity=center android:layout_marginleft=10dp android:layout_marginright=10dp android:layout_marginbottom=10dp card_view:cardcornerradius=4dp> <LinearLayout android:layout_width=match_parent android:layout_height=wrap_content android:orientation=vertical> <TextView android:layout_width=match_parent android:layout_height=wrap_content android:id=@+id/tv android:height=50dp android:background=@drawable/cardview_child android:gravity=center_vertical android:paddingleft=10dp android:text=Sub Item android:textcolor=#fff android:textstyle=bold android:textsize=30sp></TextView> <TextView android:layout_width=match_parent android:layout_height=wrap_content android :id=@+id/tvChild android:background=#71b9d6 android:layout_width=match_parent android:layout_height=wrap_content android:id=@+id/tvChild android:background=#71b9d6></TextView android:layout_width=match_parent android:id=@+id/tvChild android:background=#71b9d6 ></LinearLayout></android.support.v7.widget.CardView></RelativeLayout> ></android.support.v7.widget.CardView></RelativeLayout> Android: Padding:10dp Android: TextColor-#a72b2bアンドロイド:textSize_25sp/>All child lines create a layout structure using the above code. Step 4.Create a new class named model class HeaderModel.java Add the following source code to com.example.parsaniahardik.recyclerview_section_header package page: The public class header model is the MainActivity.ListItem string header; the public void setheader (string header) @Override @Override this.header is the header; This class contains the necessary methods for the header of the Recycler view. In the above code, this class implements the ListItem interface described in the main activity, so the isHeader() and getName() methods are overridden. The method returns a Boolean value. Returns true if the object is a HeaderModel class. The getName() method returns the name of the string for either the vehicle type or the manufacturer, but that's fine here. The setHeader() method is used to set the name of the header (see how to use the above two methods in the adapter class later in this tutorial). ChildModel, which creates a new class and puts the following code in the public class, implements the MainActivity.ListItem @Override @Override. All three methods in the above class are similar to those of the HeaderModel class. The text set and retrieved by various methods is a child line name, but the HeaderModel class uses text about the header name. Step 5.Create the main activity You need to change the code of the main activity. activity_main below the code for the .xml. <?xml version>1.0 encoding-utf-8?><LinearLayout xmlns:android> xmlns:app .com/apk/#9e9191 res-auto xmlns:tools- android:layout_width-match_parent android:layout_height match_parent MainActivity> <android.support.v7.widget.ReceiverView android:id>+id/recipe android:layout_width-match_parent android:layout_height-match_parent android:layout_margintop-15dp></android.support.v7.widget.ReceiverView> </LinearLayout>I took the Recycler view in the main layout. Write down the following source code to the MainActivity.java class: Import Android .os. Bundle Import Android Support.v7. Widgets Import <a0>.</a0> Import <a0>.</a0> Public Class MainActivity Extended App Compati Activity , Private Recycler View Recycler View;Custom Adapters;Private Strings[]Vehicle Types[]New Strings[] [Cars, Bicycles,] Aeronautics, Old Vehicles</a0> Private Array List <ListItem>List ItemArrayList; Private string[] child name [] new string [] [] [Rolls-Royce], Ferrari, Harley-Davidson, Dukati, Honda, Airbus, Airbus, Space X, Horse, Elephant, Camel, Donba, Create @Override Protected Void (bundled storage instance state) R.layout.activity_main) (R.id.recycler); List item array list , new array list <> (; populateList); Custom Adapters - New Custom Adapters (this,List Item Array List); Recycler View.set Adapter (Custom Adapter); Recycler View.Set Layout Manager (New Linear Layout Manager (getApplicationContext), Linear Layout Manager. VERTICAL, false) Get_name of the string()<a0></a0>Private void populateList()<a1> </a1>int header done<a2></a2>0, child done, < 20;<a4></a4>i++) <a0></a0>i<a1></a1><a2></a2><a4></a4><a5></a5></a1><a2></a2><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5></a1><a2></a2><a4></a4><a5></a5></a0>i<a2></a2><a4></a4><a5></a5><a5></a5></a1> headerdone-headerdone +>>><childmodel-childmodel-new childmodel();>childmodel.setchild(childnames[childdone]);<a1>listitemarralist.add(childmodel);</a1> <a0></a0>cars<a1> </a1><a2></a2> <a3></a3></a1><a2></a2><a4></a4><a4></a4><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5></a1></a0> [range-rover,-lamborghini,-rolls-royce,ferrari,harley-davidson,ducati,bmw,honda,boeing,airbus,royal-air,space-x,horse,elephant,camel,donkey string-array-variables-are-there.-string-array-variety-string-array-variety-called-vehicletypes-represents-the-types_of yclerview.-second-string-array-variable-called-children-names-the-names-of-the-vehicle-product-companies.-these-names-are-the-children-rows <a0></a0> interface<a1></a1>headermodel.java<a2></a2>and<a4></a4>childmodel.java<a2></a2>,<a4></a4><a5></a5><a5></a5><a5></a5><a5></a5></a1> lines>public>interface>listitem>;<a0></a0>string<a1></a1>getname();<a2></a2> overriding= these= methods.= populating= recyclerview= data= our= data= for= header= and= child= rows= are= present= in= the= two= strings= variables= as= we= have= just= shown= above.= now= we= need= to= convert= this= data= in= the= arraylist= of= interface= objects= to= pass= them= into= the= adapter= class.= following= method= will= do= this= for= us= private= void= populatelist() <a0></a0>int<a1></a1>headerdone<a2></a2>0, childdone<a4></a4>0; for(int<a2></a2>i<a4></a4>>> < 20; i++) 20;>>i++)>>>>>>>>>>>>>>>& <a0></a0>i<a1></a1><a2></a2><a3> </a3><a5></a5><a5></a5><a5></a5><a5></a5><a5></a5> i++) <a0></a0>i<a1></a1><a2></a2><a3></a3><a2></a2><a3></a3><a5></a5> <a5></a5> <a5></a5> <a5></a5> <a5></a5> <a5></a5><a5></a5></a1></a0>on the left end of the screen. Header done , header done + 1; Other children's models: new child models (; child models.setChild(child names[child don]); list item array lists.add(child models); In ReceiverView, the first row represents the header name. The next four lines will be child rows. In this calculation, the first, sixth, 11th, and 16th lines of the Recycler view are headers. So, in the above code, the conditions apply to the first, sixth, 11th, and 16th iterations of the for loop. If the condition is true, an object of the HeaderModel class is created, named, and inserted into the array list of interfaces (listItemArrayList). If the condition is false, an object of the ChildModel class is created, named, and inserted into the array list of interfaces (this is a listItemArrayList). After a successful method call, the listItemArrayList is passed to the constructor of the adapter class using the following line: Step 6.Custom Adapter Finally, fill the Recycler view in the adapter class. To create a new class named import the following code, import android.content.Context; Import <a0>.</a0> Import Android Import Android.view.view; Import Android.Widgets Import <a0>.</a0> The public class CustomAdapter extends the Private Layout Inflator Inflator; LAYOUT_CHILD LAYOUT_HEADER プライベートな配列<MainActivity.ListItem>一覧リストリストリストリストリストリスト;パブリックカスタムアダプタ(コンテキストコンテキスト、配列リスト<MainActivity.ListItem>リストリストアイテム配列リスト){インフレータ = レイアウトインフレーター.from(コンテキスト);this.context = コンテキスト;this.listItemArrayList = リストアイテムリスト; } @Overrideパブリックな getItemCount() { 戻りリストアイテムリスト(@Override int position LAYOUT_HEADER)それ以外の場合はLAYOUT_CHILD返します。 @Override Public Recycler View Holder (View Group Parent, int viewType R.layout.rv_header LAYOUT_HEADER) else, view view, inflate (R.layout.rv_child, parent, false); holder, new MyViewHolderChild (view); @Override Public void-on-BindViewHolder (Recycler View.View Holder, Final Int Position LAYOUT_HEADER)Other <a0> My View Holder Child Bolt Item Holders </a0> , (My View Holder Child) Holders; Container Item Holders.tvchild.setText (List Item Array List.get(Position).Extend the class. Public My View Holder Header (Item View) Class My View Holder Child Extended Recycle Bar View.View Holders, Text View Child, Public My View Holder Child (Item View), Super (Item View), LAYOUT_HEADER tv Child, (Text View) Item View Description Read the code below@Override Public int getItemViewType(int position) Otherwise, it LAYOUT_CHILD back. The above method determines whether the row view is a header view or a child view. We are using the isHeader() method for this purpose. LAYOUT_HEADER and LAYOUT_CHILD are integer constants. Consider the following code for the class class: Public My View Holder Header (Item View) Class My View Holder Child Extended Recycle Bar View.View Holders, Text View Child, Public My View Holder Child (Item View), Super (Item View), tv Child, (Text View) Item View The above code defines two classes. The first one represents the header, and the second represents the child row. The above class only declares a text view of the header and child lines. Consider the following code@Override consider the following code: The last int position) (the LAYOUT_HEADER) , the My View Holder Header Bolt Item Holder, the (My View Holder Header) holder; (Item holder.tvHeader.setText (list item array list.get(position).getName(); Here I checked one of the conditions. If the if condition is true, the text view (tvHeader) defined in the MyViewHolderHeader class gets the value. If the if condition is false, the text view (tvchild) defined in the MyViewHolderChild class gets the value. Value.