Android Lists and Grids with Xamarin

The RecyclerView

Why Talk about Lists and Grids?

- Most apps (not including games) render some form of repeating content (list or grid)
- Improperly using lists or grids can have performance implications
- It is important to know how to use them

Basic Terminology

- View - a UI component or widget, i.e. TextView, ListView, GridView
- Adapter - a class that manages the interaction between a view and a collection of data
- ViewHolder - a class that caches references to view objects

Lists in Android

- ListView is a view that renders some collection of content where each item gets its own template view
- ListViews are assigned an adapter that manages the interaction with the actual data
- The adapter exposes a method that returns a view (item) in the list and binds it to a data model from the collection
- The adapter should attempt to reuse an item that was scrolled off of the screen instead of creating new ones
- The adapter should also cache references to items inside of the view (ViewHolder pattern)

ViewHolder Pattern

- Each item in the list is composed of the subviews that render the data, i.e. TextView, ImageView, ButtonView
- Your code must obtain references to these views in order to populate the data in the list's item
- This is relatively expensive
- The ViewHolder pattern has you cache those view references in a class and attach it to the inflated view
- When you reuse an item you should always check for an attached ViewHolder before creating the references to the views

Performance Implications

- You don’t want to create a new view for each item in the list since only a subset might actually be rendered at a time
- Only the visible items are important
- Garbage collection can become problematic if items are destroyed when they scroll off of the screen
- It is *highly* recommended to reuse items when they are no longer needed to avoid GC
- Item reuse combined with the ViewHolder pattern will result in better performance and smoother scrolling

Grids in Android

- Uses the GridView view instead of the ListView
- The coding patterns are the same
- The GridView exposes some additional properties to allow for customization, i.e. how many columns

Introducing the RecyclerView

- Lollipop, a.k.a Android 5.0, released in 2014 with a new view called the RecyclerView
- The RecyclerView is intended to replace both the ListView and the GridView
- The RecyclerView has the item reuse and ViewHolder design patterns baked in to the implementation

ViewHolder Pattern

- Android will call “onCreateViewHolder” instead of calling “onCreateView” to create each item in the list
- This is responsible for inflating the view and creating references to any subviews and returning them wrapped in a ViewHolder
- “onCreateViewHolder” is not responsible for binding any data
- Once the ViewHolder has been created then Android will call “onBindViewHolder” to bind the current data item to the views referenced by the ViewHolder

Item Reuse

- Android will only call “onCreateViewHolder” when a new item is actually needed
- Otherwise it will just call “onBindViewHolder” to bind an old view to new data for reuse (or recycled, hence the name)
- Because you cannot tell if an item is a recycled item or a new item, you must *always* initialize every subview to make sure that stale data does not get rendered

What about Grids?

- The RecyclerView also requires a LayoutManager in addition to the adapter
- The LayoutManager is responsible for how each item renders in relation to the others
- Out of the box you can use the stock LinearLayoutManager for lists and the GridLayoutManager to render grids
- The GridLayoutManager expects you to define the number of columns in the grid, as well as an optional helper method to determine the column span for each item
- The RecyclerView is now the one-stop shop for rendering collections

Backwards Compatibility?

- The RecyclerView was initially introduced in Lollipop so older devices could not use it
- Later in 2014 the RecyclerView was added to a support library to add support for it to older devices
- Make sure to include the v7 RecyclerView support library to ensure that it will work for all users

Demo

- Basic linear list
- Basic grid

Decorators

- Unlike the ListView, the RecyclerView does not render divider lines between items in the list
- A common workaround for this is to use an ItemDecorator to draw a divider for each item
- The RecyclerView exposes an ItemDecoration class that you can use to modify how items render in the list
- Essentially you draw directly into the canvas for each item
- “onDraw” will draw prior to rendering the item itself
- “onDrawOver” will draw after rendering the item

Demo

- List with divider

Use with Cards

- Cards are a common way to render forms of data, especially with Material Design
- Android’s CardView is commonly used with the RecyclerView

Demo

- List with Cards

Touch Interaction

- The use of Cards normally suggests some form of touch interaction, i.e. swipe to delete or drag and drop
- The ItemTouchHelper class allows you to attach touch behaviors to items in your list
- The ItemTouchHelper is a subclass of ItemDecorator
- You can define which directions an item can be swiped
- You can define which directions an item can be dragged
- You can register a callback for when an item has been fully swiped or dropped somewhere else in the list/grid

Demo

- List with swipeable Cards

Scroll Awareness

- Another new feature in Lollipop is the CoordinatorLayout
- Among other things, the CoordinatorLayout allows child views to notify when they are scrolled
- A common use case for this is to hide the floating action button when scrolled
- Another common use case is to expand or contract the toolbar when scrolling up or down

Scroll Awareness Implementation

- You can implement a custom scroll behavior by implementing the CoordinatorLayout.Behavior class
- The two important methods are “onStartNestedScroll” and “onNestedScroll”
- “onStartNestedScroll” allow you to define the axes that you care about
- “onNestedScroll” gives you a reference to the scrolled item as well as some direction offsets
- The behavior must be added to the view that is supposed to *respond* to the scroll event

Demo

- List with FAB that is only visible when the list is at the very top

Toolbar Interaction

- Lollipop also added a few helper layout wrappers to make it easier to do some scroll interaction
- The AppBarLayout should wrap your Toolbar view
- The Toolbar can then define scrollFlags to indicate what it should do for different scroll scenarios
- You can also nest a CollapsingToolbarLayout between your Toolbar and your AppBarLayout to allow for a large toolbar that shrinks as you scroll

Demo

- List with disappearing toolbar
- List with collapsing toolbar

Device Rotation

- It is worth mentioning that rotating your device will trigger your activity to restart
- If you want to maintain your scrolling location and behavior states then you will need to properly save and restore the state
- Android automatically will restore views that have ids
- The source code for the demo is set up to properly save and restore state

Questions?

ilm
Knowledge meets passion

Contact Information

- https://www.linkedin.com/in/joelpeterson2
- joel.peterson@ilmservice.com

- Source code: https://github.com/jpeters5392/RecyclerViewDemo