Android RecyclerView 사용하기

Android L 프리뷰 버전에서 등장한 RecyclerView는 이번 Android 5.0 프리뷰 버전과 함께 Support-Library-v7의 최신 버전에 정식으로 추가되었다.

RecyclerView는 기존의 ListView보다 유연하고 성능이 향상된 고급 위젯이다. 기존의 ListView는 커스텀하기에는 구조적인 문제로 많은 제약이 따랐으며, 구조적인 문제로 인해 성능 문제가 있었다. RecyclerView는 이런 고질적인 문제를 해결하기 위해 좀 더 다양한 형태로 개발자가 커스텀할 수 있도록 유연하며 성능에 중점을 두어 만들어졌다.

ListView에 비해 RecyclerView의 가장 큰 변경 사항은 LayoutManager과 ViewHolder, Item에 대한 뷰의 변형이나 애니메이션 할 수 있는 개념이 추가되었다. 이들은 리스트의 아이템이 표시될 재활용 뷰를 관리하는 데 사용되며 아이템의 포지션에 따른 레이아웃의 배치를 결정하게 된다. 또한 불필요한 뷰의 생성을 피하기 위해 레이아웃을 관리하는 역할을 하게 된다.

주요 클래스

Adapter

기존의 ListView에서 사용하는 Adapter와 같은 개념으로 데이터와 아이템에 대한 View생성

ViewHolder

재활용 View에 대한 모든 서브 뷰를 보유

LayoutManager

아이템의 항목을 배치

ItemDecoration

아이템 항목에서 서브뷰에 대한 처리

ItemAnimation

아이템 항목이 추가, 제거되거나 정렬될때 애니메이션 처리

LayoutManager

RecyclerView에서 가장 흥미로운 부분이다. RecyclerView를 생성 시 반드시 생성되어야 하며 이를 통해 모든 아이템의 뷰의 레이아웃을 관리한다. 수평/수직 배치뿐만 아니라 그리드 형태의 다양하게 레이아웃을 배치할 수 있다.

기본적으로 제공하는 LayoutManager

- LinearLayoutManager - 수평/수직의 스크롤 리스트
- GridLayoutManager - 그리드 리스트
- StaggeredGridLayoutManage - 높이가 불구칙적인 형태의 그리드 리스트

이외에 개발자는 LayoutManager를 확장하여 다양한 형태를 만들 수 있다. 아래와 같이 사용한다.

```
LinearLayoutManager layoutManager = new LinearLayoutManager(context);
layoutManager.setOrientation(LinearLayoutManager.VERTICAL);
layoutManager.scrollToPosition(currPos);
recyclerView.setLayoutManager(layoutManager);
```

ViewHolder

ViewHolder는 기존의 ListView에서 많이 사용하고 구글 안드로이드 팀에서도 오랫동안 추천된 패턴이다. 하지만 이를 사용하는 것을 강제적으로 제한하지 않았지만 **RecyclerView에서는 Adapter와 ViewHolder를 반드시 같이 사용**할 수밖에 없는 구조로 바뀌었다. ViewHolder패턴을 사용하지 않은 개발자는 약간의 훈련이 필요할 것이다.

```
public final static class ListItemViewHolder extends RecyclerView.ViewHolder {
    TextView label;
    TextView dateTime;

    public ListItemViewHolder(View itemView) {
        super(itemView);
        label = (TextView) itemView.findViewById(R.id.txt_label_item);
        dateTime = (TextView) itemView.findViewById(R.id.txt_date_time);
    }
}
```

사용법은 ViewHolder를 확장 후 서브 클래스를 findViewById()를 통해 저장해놓으면 된다. 이렇게 함으로 한번 생성한 클래스를 통해 서브 클래스(뷰)를 빠르게 다시 액세스 할 수 있다.

Adapter

ListView에서 Adpater와 동일한 형태의 구조로 해당 아이템의 데이터와 뷰간의 처리를 한다. 다음과 같은 3가지의 인터페이스를 구현해야 한다.

`public ListItemViewHolder onCreateViewHolder(ViewGroup parent, int viewType)`
제네릭 형식의 변수로 ViewHolder를 생성

`public void onBindViewHolder(ListItemViewHolder holder, int position)`
만들어진 ViewHolder에 데이터를 넣는 작업, ListView의 getView()와 동일

`public int getItemCount()`
데이터의 갯수

```
public class RecyclerViewDemoAdapter extends RecyclerView.Adapter {

    private List items;
    RecyclerViewDemoAdapter(List modelData) {
        if (modelData == null) {
            throw new IllegalArgumentException(
                    "modelData must not be null");
        }
        this.items = modelData;
    }

    @Override
    public ListItemViewHolder onCreateViewHolder(
            ViewGroup viewGroup, int viewType) {
                    View itemView = LayoutInflater.from(viewGroup.getContext())
                            .inflate(R.layout.item_demo_01, viewGroup, false);
        return new ListItemViewHolder(itemView, viewType);
    }

    @Override
    public void onBindViewHolder(
            ListItemViewHolder viewHolder, int position) {
        DemoModel model = items.get(position);
        viewHolder.label.setText(model.label);
        String dateStr = DateUtils.formatDateTime(
                viewHolder.label.getContext(),
                model.dateTime.getTime(),
                DateUtils.FORMAT_ABBREV_ALL);
        viewHolder.dateTime.setText(dateStr);
    }

    @Override
    public int getItemCount() {
        return items.size();
    }

    public final static class ListItemViewHolder
            extends RecyclerView.ViewHolder {
        // ViewHolder
    }
}
```

기존의 ListView와 동일한 구조라서 쉽게 구현이 가능하며, Adapter의 기본 클래스를 확장한 형태(CursorAdapter, ArrayAdapter)는 없다.

ItemDecoration

각 아이템 항목별로 오프셋을 추가하거나 아이템을 꾸미는 작업을 하게 된다. 예를 들어 스크롤 시 콘텐츠의 내용에 따라 View의 높이가 달라져 레이아웃의 위치를 이동해야 하는 작업하는 경우 여기에서 처리하면 된다.

필요시 다음과 같은 3가지를 구현해야 한다.

```
public void onDraw(Canvas c, RecyclerView parent)
public void onDrawOver(Canvas c, RecyclerView parent)
public void getItemOffsets(Rect outRect, int itemPosition, RecyclerView parent)
```

LayoutManager에서 getItemOffsets()의 호출을 통해 아이템의 레이아웃의 크기를 측정 하기 때문에 위의 예시는 getItemOffsets()에서 작업하면된다.

ItemAnimtor

ListView에서 아이템별 애니메이션을 일으키기 위해 notifyDataSetChanged()를 호출해 모든 아이템 변경이 발생할 때 처리를 하였으나 notifyItemChanged(int position), notifyItemInserted(int position), notifyItemRemoved(int position)를 통해 ItemAnimator를 통해 특정 아이템에 대한 애니메이션을 발생할 수 있다.

```
public final void notifyItemInserted(int position)
public final void notifyItemRemoved(int position)
```

예제

```
RecyclerView.ItemDecoration itemDecoration =
        new DividerItemDecoration(this, DividerItemDecoration.VERTICAL_LIST);
recyclerView.addItemDecoration(itemDecoration);

recyclerView.setItemAnimator(new CustomItemAnimator());

public class CustomItemAnimator extends ItemAnimator {

    public CustomItemAnimator() {
        setAddDuration(300);
        setRemoveDuration(300);
    }
```

```
    @Override
    protected boolean prepHolderForAnimateRemove(ViewHolder holder) {
        return true;
    }

    @Override
    protected ViewPropertyAnimatorCompat animateRemoveImpl(ViewHolder holder) {
        return ViewCompat.animate(holder.itemView)
                .rotationX(90)
                .translationY( - (holder.itemView.getMeasuredHeight() / 2));
    }

    @Override
    protected void onRemoveCanceled(ViewHolder holder) {
        ViewCompat.setRotationX(holder.itemView, 0);
        ViewCompat.setTranslationY(holder.itemView, 0);
    }

    @Override
    protected boolean prepHolderForAnimateAdd(ViewHolder holder) {
        ViewCompat.setRotationX(holder.itemView, 90);
        ViewCompat.setTranslationY(holder.itemView,
                    - (holder.itemView.getMeasuredHeight() / 2));
        return true;
    }

    @Override
    protected ViewPropertyAnimatorCompat animateAddImpl(ViewHolder holder) {
        return ViewCompat.animate(holder.itemView)
                .rotationX(0)
                .translationY(0);
    }

    @Override
    protected void onAddCanceled(ViewHolder holder) {
        ViewCompat.setRotationX(holder.itemView, 0);
        ViewCompat.setTranslationY(holder.itemView, 0);
    }
}
```

그 외

ListView OnItemClickListener, OnItemLongClickListener의 아이템 터치에 대한 리스너가

RecyclerView에서는 addOnItemTouchListener, setOnClickListener를 통해 처리하게 되었다. 이는 GestureDetecter를 통해 좀 더 다양한 이벤트를 식별할 수 있는 유연한 구조로 바뀌었다. 한편으로는 유연하지만 한편으로는 좀 더 어려워졌다고 볼 수 있다.

addHeaderView(), addFooterView()는 어디로?

없어졌다. 이를 위해 Adapter의 onCreateViewHolder()에서 itemType으로 header나 footer를 추가하는 방식으로 처리해야 한다.

마치며..

ListView에서 RecyclerView를 바로 사용하기는 개념상 달라진 점이 없기 때문에 무리가 없겠으나 ListView에 있는 몇몇 기능들이 RecyclerView에서는 지원하지 않아 직접 처리해야 하는 경우가 있어 보인다. 그리고 확실시 ListView에 다양하게 확장할 수 있도록 유연해졌다. 하지만 유연해진 만큼 손이 많이 가는 부분도 없지 않아 있다.

참고 코드:https://github.com/writtmeyer/recyclerviewdemo