RecyclerView 核心要点

Jiaheng

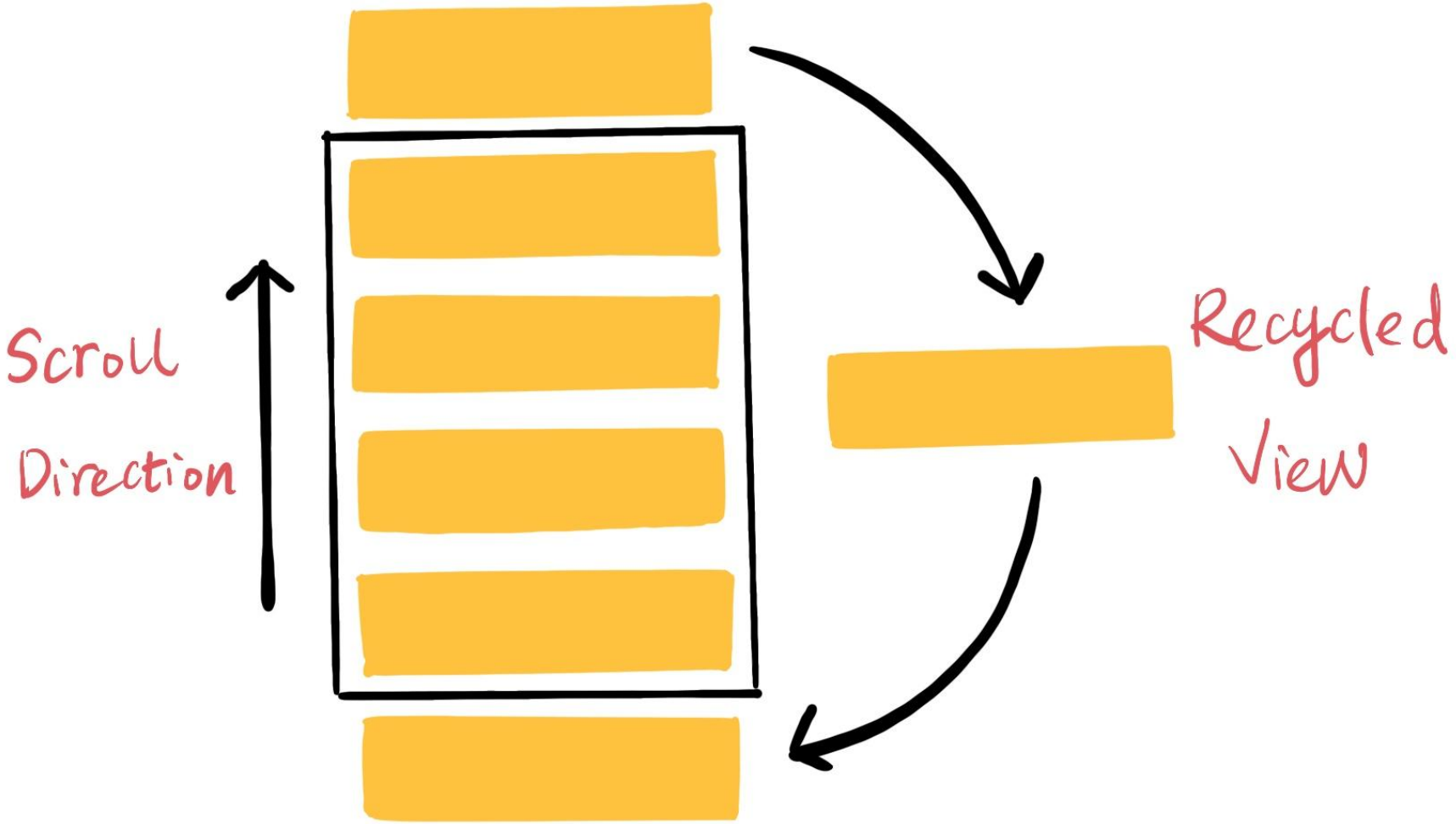
为什么讲这个主题？

- 绝大多数 App 都有列表页面，关系到核心用户体验
- 不看任何文档用 RecyclerView 写一个简单的列表？
- 仍然有不少公司在使用 ListView
- 即使使用 RecyclerView，只是很简单的应用

今天讲什么？

- RecyclerView 是什么？
- RecyclerView Demo
- ViewHolder 究竟是什么？
- RecyclerView 缓存机制
- 你可能不知道的 RecyclerView 性能优化策略
- 为什么 ItemDecoration 可以绘制分隔线？

A **flexible** view for providing
a **limited** window into
a **large** data set


Scroll
Direction
Recycled
View

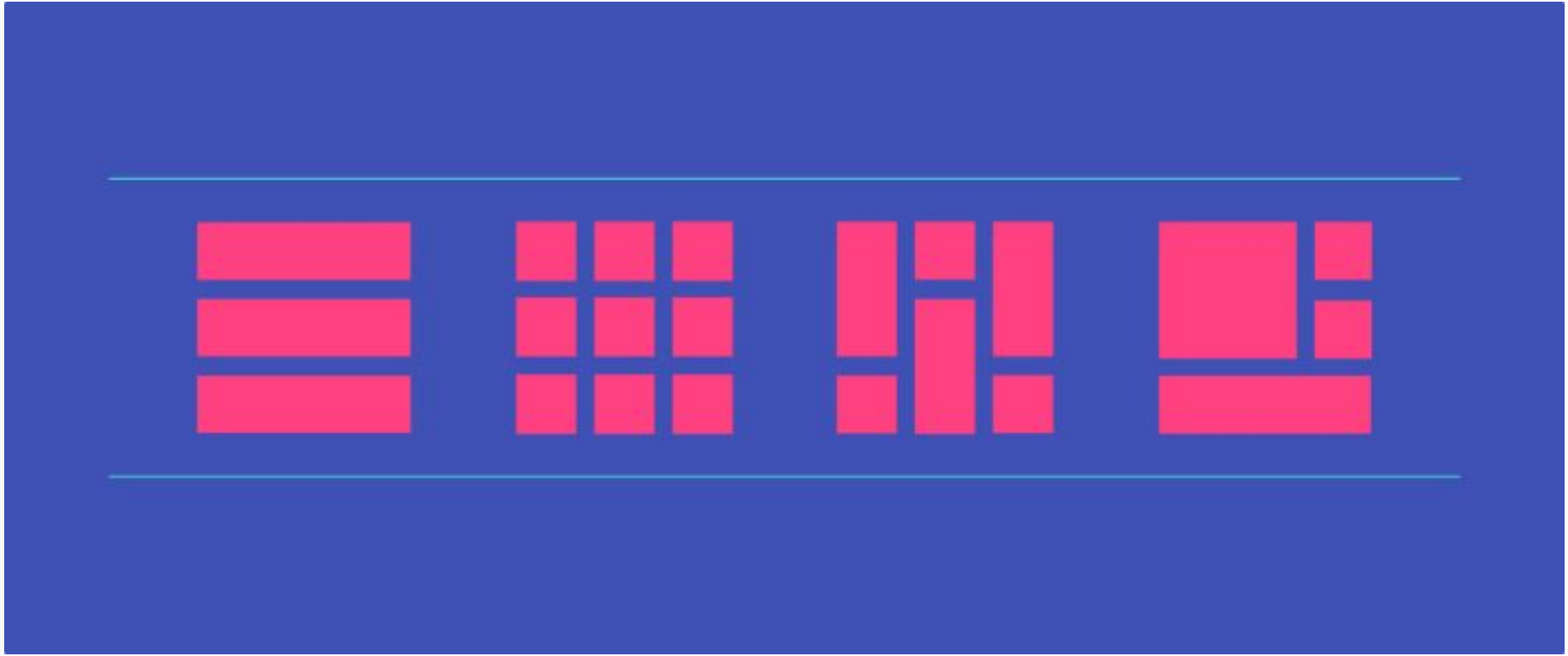
ListView 的局限

- 只有纵向列表一种布局
- 没有支持动画的 API
- 接口设计和系统不一致
  - setOnItemClickListener()
  - setOnItemLongClickListener()
  - setSelection()
- 没有强制实现 ViewHolder
- 性能不如 RecyclerView

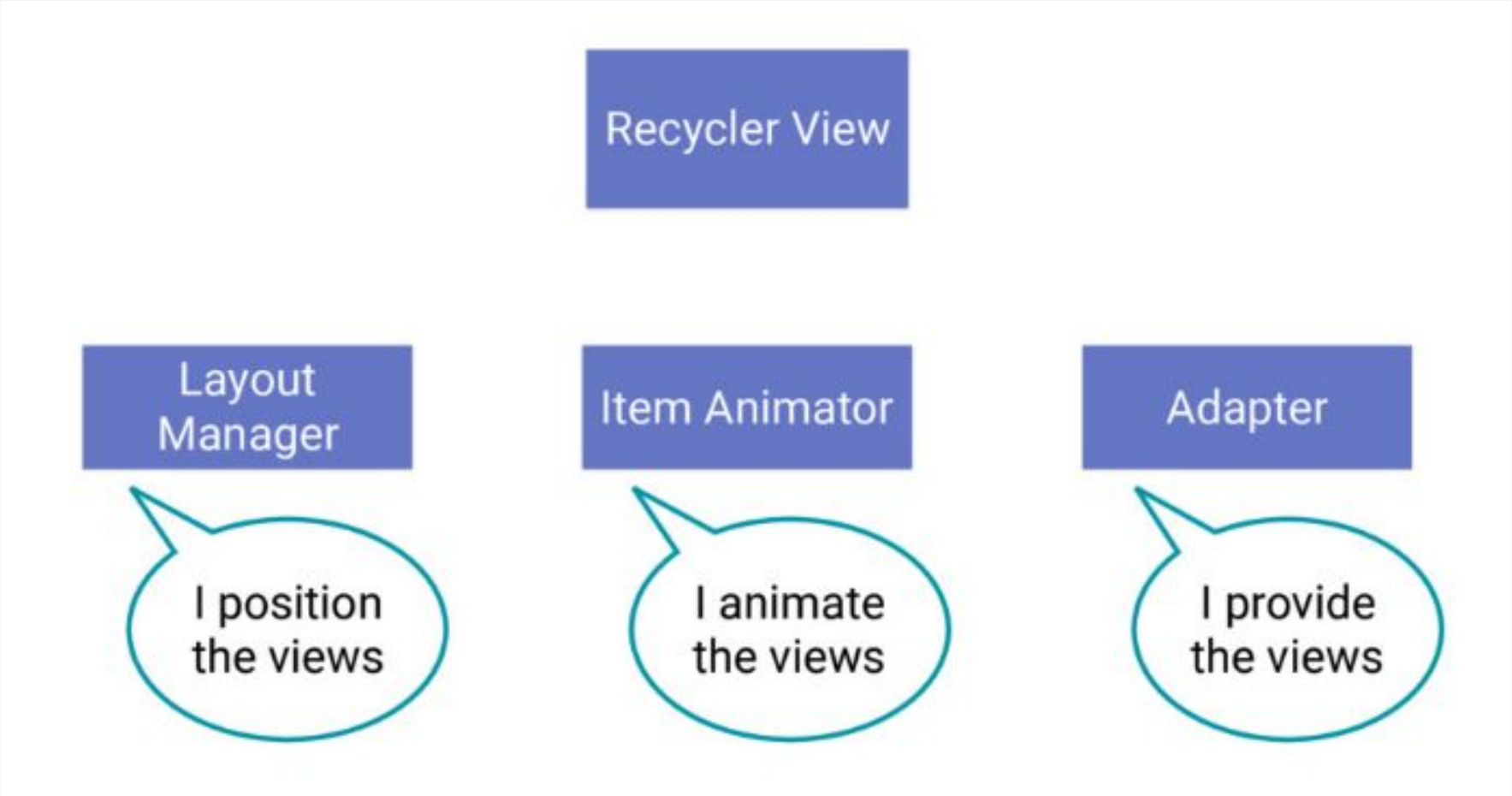
RecyclerView 的优势

- 默认支持 Linear, Grid, Staggered Grid 三种布局
- 友好的 ItemAnimator 动画 API
- 强制实现 ViewHolder
- 解耦的架构设计
- 相比 ListView 更好的性能

LayoutManager 支持的布局



RecyclerView 的重要组件



简单 Demo

View holder 究竟是什么？

- View holder 和 item view 是什么关系？一对一？一对多？多对一？
- View holder 解决的是什么问题？
- View holder 的 ListView item view 的复用有什么关系？

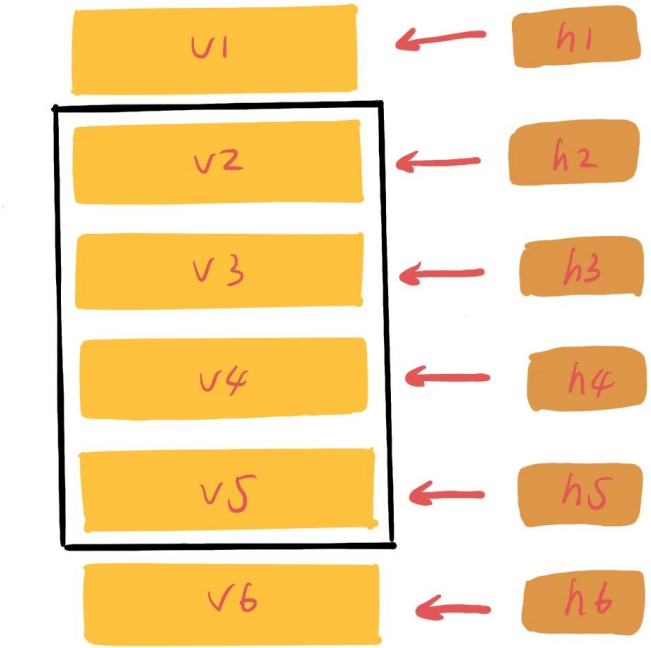
没有实现 view holder 的 getView()

```
public class SimpleListViewAdapter extends BaseAdapter {
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = LayoutInflater.from(parent.getContext())
                                        .inflate(R.layout.list_view_item, parent, false);
        }
        ImageView avatar = convertView.findViewById(R.id.user_avatar);
        TextView name = convertView.findViewById(R.id.user_name);
        TextView title = convertView.findViewById(R.id.user_title);

        User user = getItem(position);
        Glide.with(parent.getContext()).load(user.avatarUrl).into(avatar);
        name.setText(user.name);
        title.setText(user.title);

        return convertView;
    }
}
```

实现了 view holder 的 getView()

```
public class SimpleListViewAdapter extends BaseAdapter {
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        UserViewHolder holder;
        if (convertView == null) {
            convertView = LayoutInflater.from(parent.getContext())
                                        .inflate(R.layout.list_view_item, parent, false);
            holder = new UserViewHolder(convertView);
            holder.avatar = convertView.findViewById(R.id.user_avatar);
            holder.name = convertView.findViewById(R.id.user_name);
            holder.title = convertView.findViewById(R.id.user_title);
            convertView.setTag(holder);
        } else {
            holder = (UserViewHolder)convertView.getTag();
        }

        User user = getItem(position);
        Glide.with(parent.getContext()).load(user.avatarUrl).into(holder.avatar);
        holder.name.setText(user.name);
        holder.title.setText(user.title);

        return convertView;
    }
}
```

Item view 和 view holder 一一对应



不实现 view holder 还会复用 item view 吗？

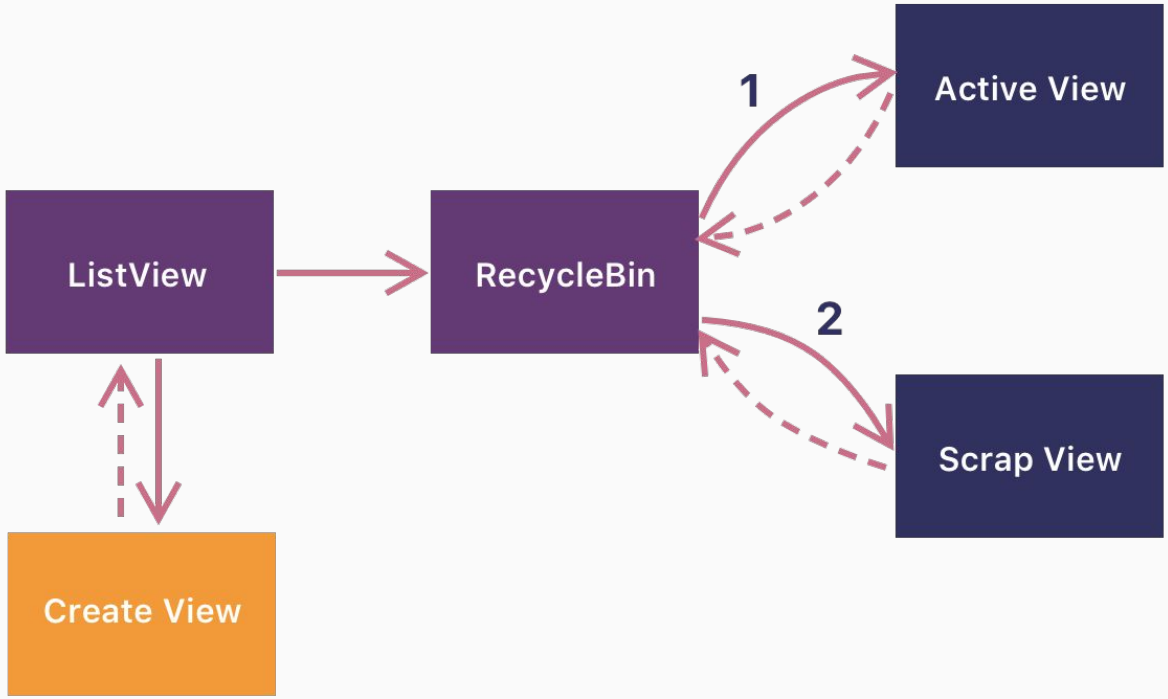
```
public class SimpleListViewAdapter extends BaseAdapter {
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if (convertView == null) {
            convertView = LayoutInflater.from(parent.getContext())
                                        .inflate(R.layout.list_view_item, parent, false);
        }
        ImageView avatar = convertView.findViewById(R.id.user_avatar);
        TextView name = convertView.findViewById(R.id.user_name);
        TextView title = convertView.findViewById(R.id.user_title);

        User user = getItem(position);
        Glide.with(parent.getContext()).load(user.avatarUrl).into(avatar);
        name.setText(user.name);
        title.setText(user.title);

        return convertView;
    }
}
```

View holder 最佳实践

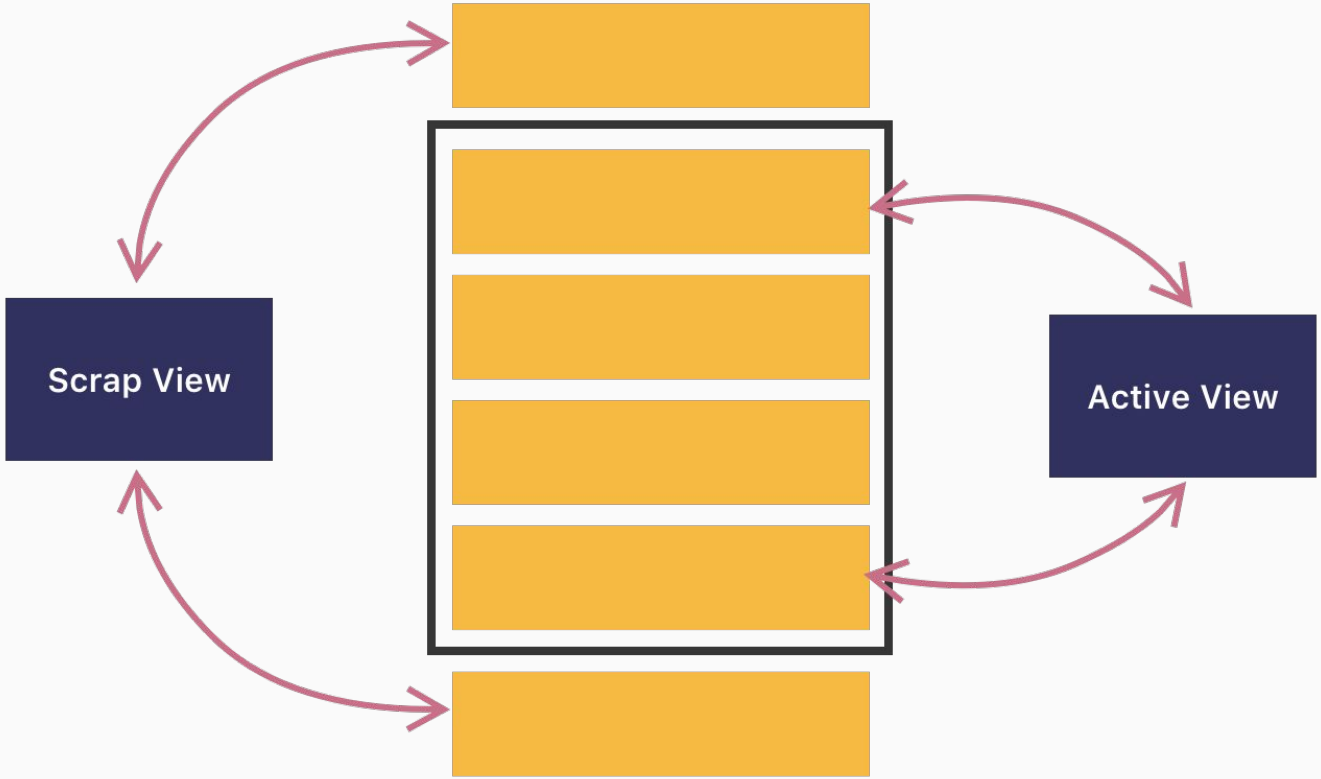
```
static class UserViewHolder extends RecyclerView.ViewHolder {
    ImageView avatar;
    TextView name;

    UserViewHolder(@NonNull View itemView) {
        super(itemView);
        avatar = itemView.findViewById(R.id.avatar);
        name = itemView.findViewById(R.id.name);
    }

    void bindTo(User user) {
        // bind data to UI
    }
}

// void onBindViewHolder(UserViewHolder holder, int position) {
//     holder.bindTo(userList.get(position));
// }
```

RecyclerView 缓存机制

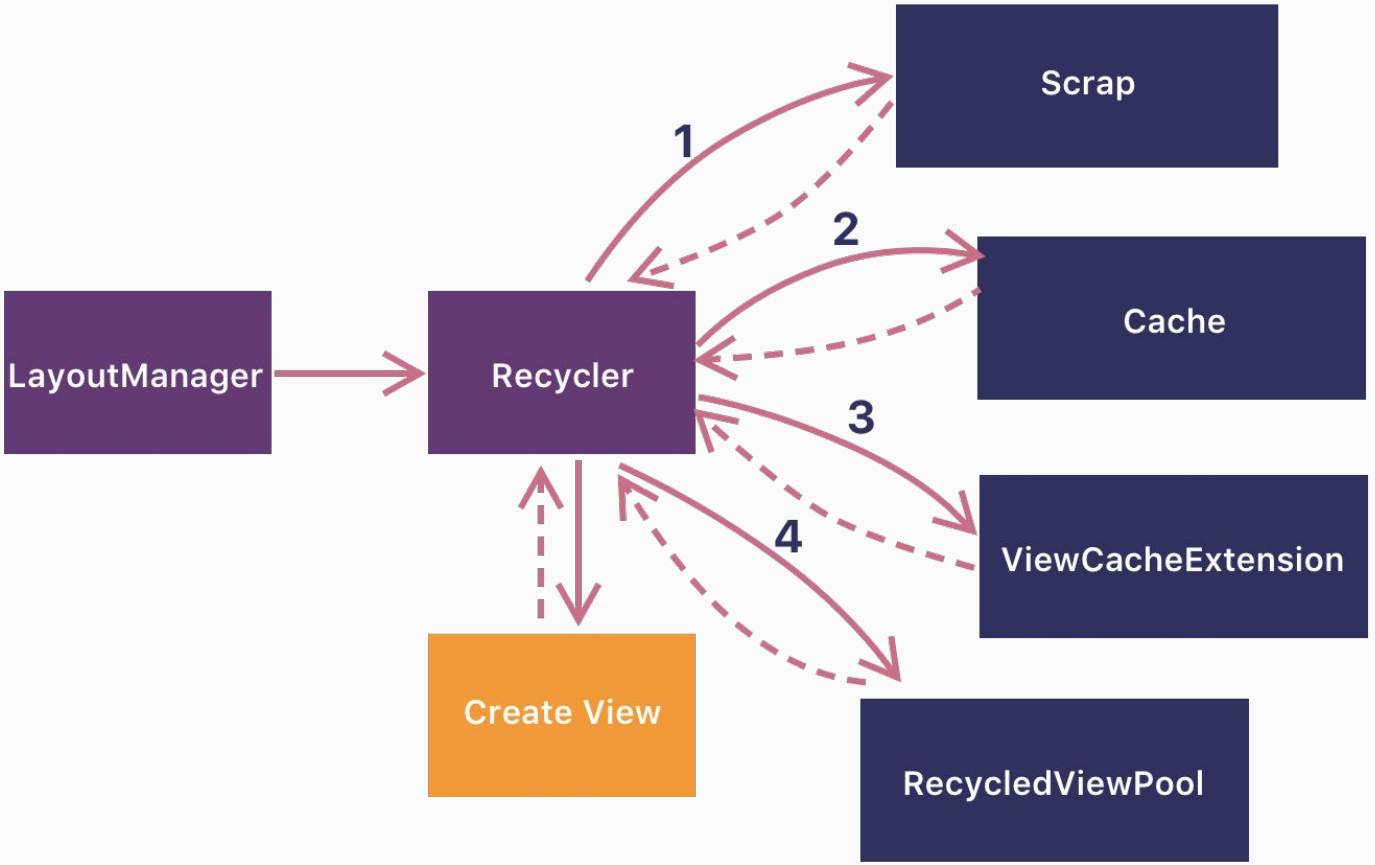
ListView 缓存图示一



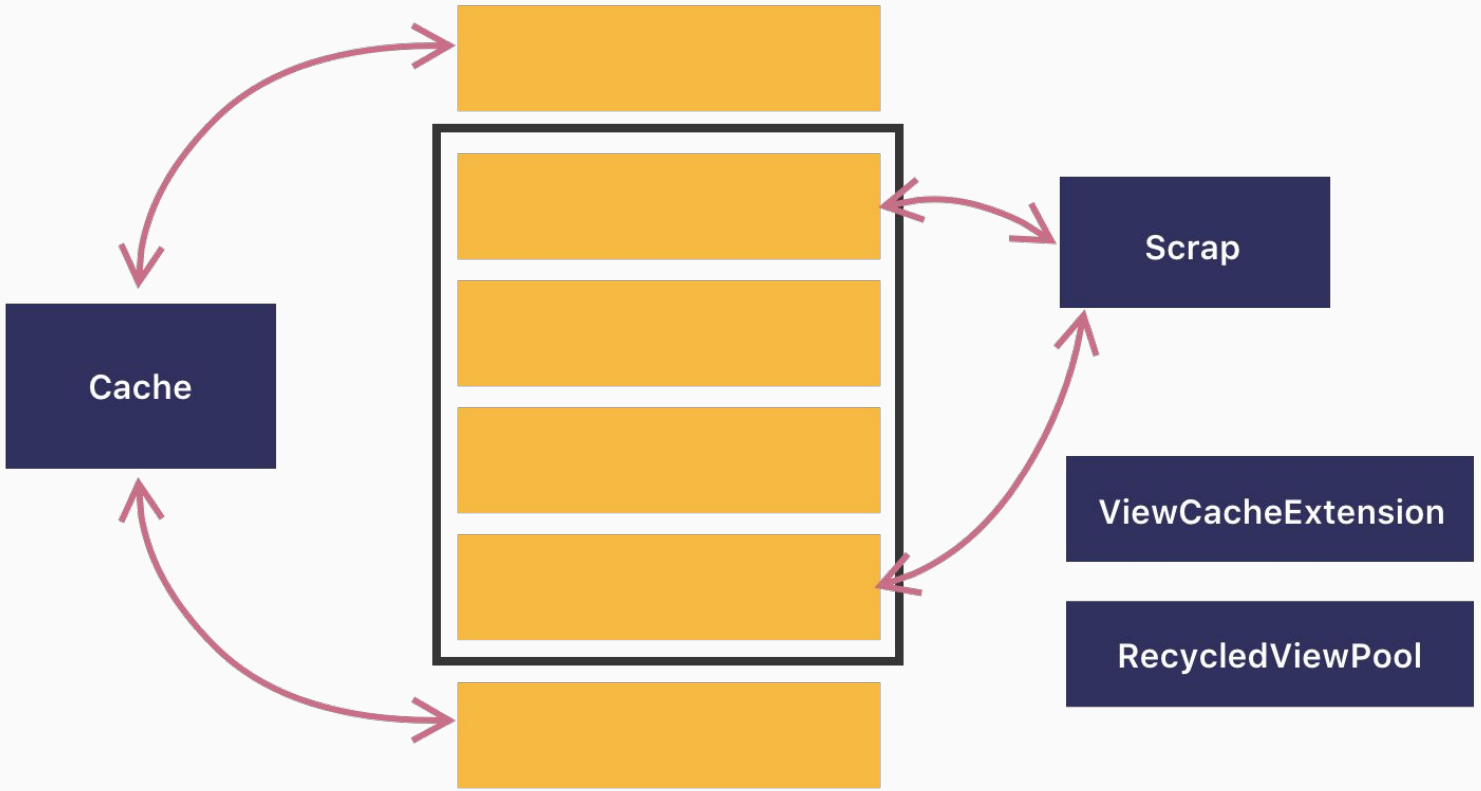
ListView 缓存图示二



RecyclerView 缓存图示一



RecyclerView 缓存图示二



ViewCacheExtension Example

- 广告卡片
  - 每一页一共有 4 个广告
  - 这些广告短期内不会发生变化
- 每次滑入一个广告卡片，一般情况下都需要重新绑定
- Cache 只关心 position，不关心 view type
- RecycledViewPool 只关心 view type，都需要重新绑定
- 在 ViewCacheExtension 里保持 4 个广告 Card 缓存

注意：列表中 item/广告的 impression 统计

- ListView 通过 getView() 统计
- RecyclerView 通过 onBindViewHolder() 统计？**可能错误！**
- 通过 onViewAttachedToWindow() 统计

你可能不知道的 RecyclerView 性能优化策略

在 onBindViewHolder 里设置点击监听？

onBindViewHolder 里设置点击监听器会导致重复创建对象

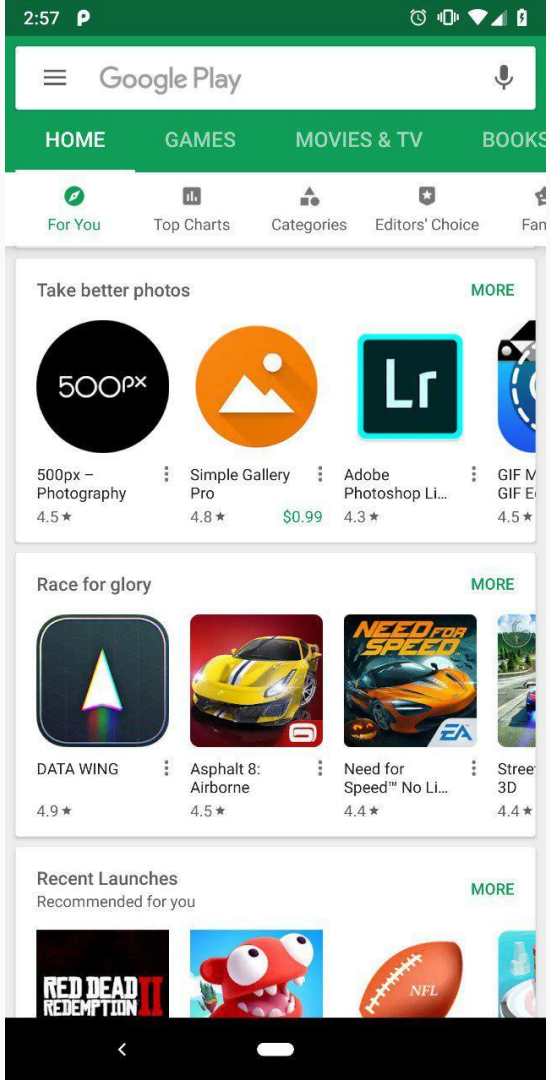
```
public class SimpleAdapter extends RecyclerView.Adapter {

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // ...
            }
        });
    }
}
```

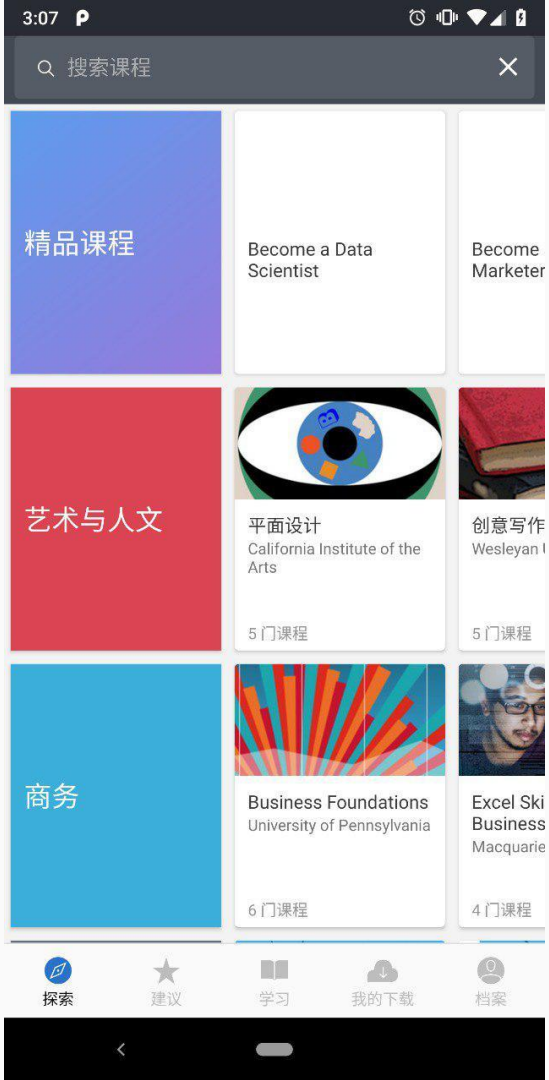
在 onCreateViewHolder 里设置点击监听！

View - ViewHolder - View.OnClickListener 三者一一对应

```
public class SimpleAdapter extends RecyclerView.Adapter {

    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        final SimpleViewHolder holder = new SimpleViewHolder();
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // ...
            }
        });
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int position) {

    }
}
```

LinearLayoutManager.setInitialPrefetchItemCount()





使用 LinearLayoutManager.setInitialPrefetchItemCount()

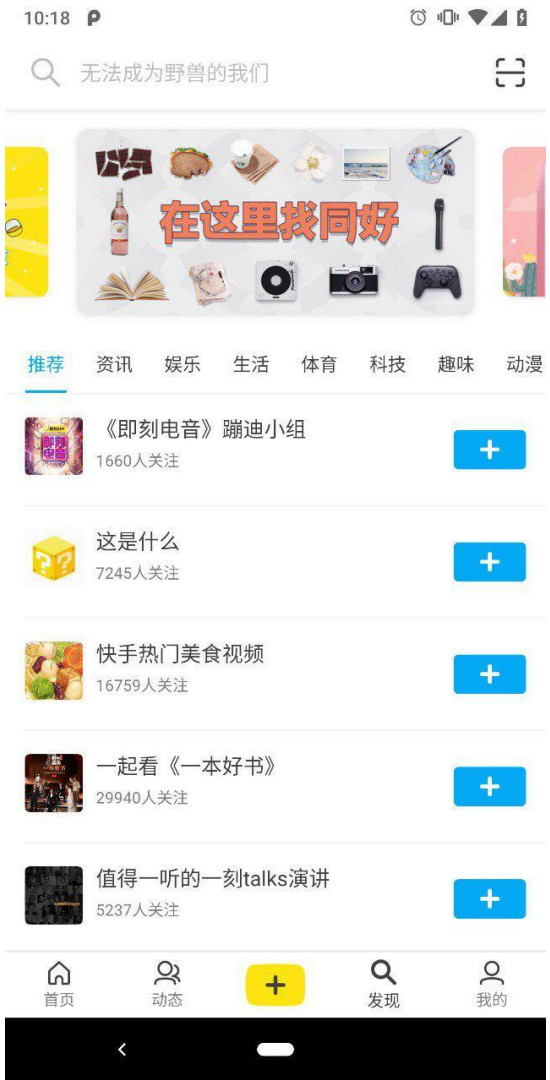
- 用户滑动到横向滑动的 item RecyclerView 的时候，由于需要创建更复杂的 RecyclerView 以及多个子 view，可能会导致页面卡顿
- 由于 RenderThread 的存在，RecyclerView 会进行 prefetch
- LinearLayoutManager.setInitialPrefetchItemCount(横向列表初次显示时可见的 item 个数)
  - 只有 LinearLayoutManager 有这个 API
  - 只有嵌套在内部的 RecyclerView 才会生效

RecyclerView.setHasFixedSize()

```
// 伪代码
void onContentsChanged() {
    if (mHasFixedSize) {
        layoutChildren();
    } else {
        requestLayout();
    }
}
```

如果 Adapter 的数据变化不会导致 RecyclerView 的大小变化 → RecyclerView.setHasFixedSize(**true**)

多个 RecyclerView 共用 RecycledViewPool



共用 RecycledViewPool 代码

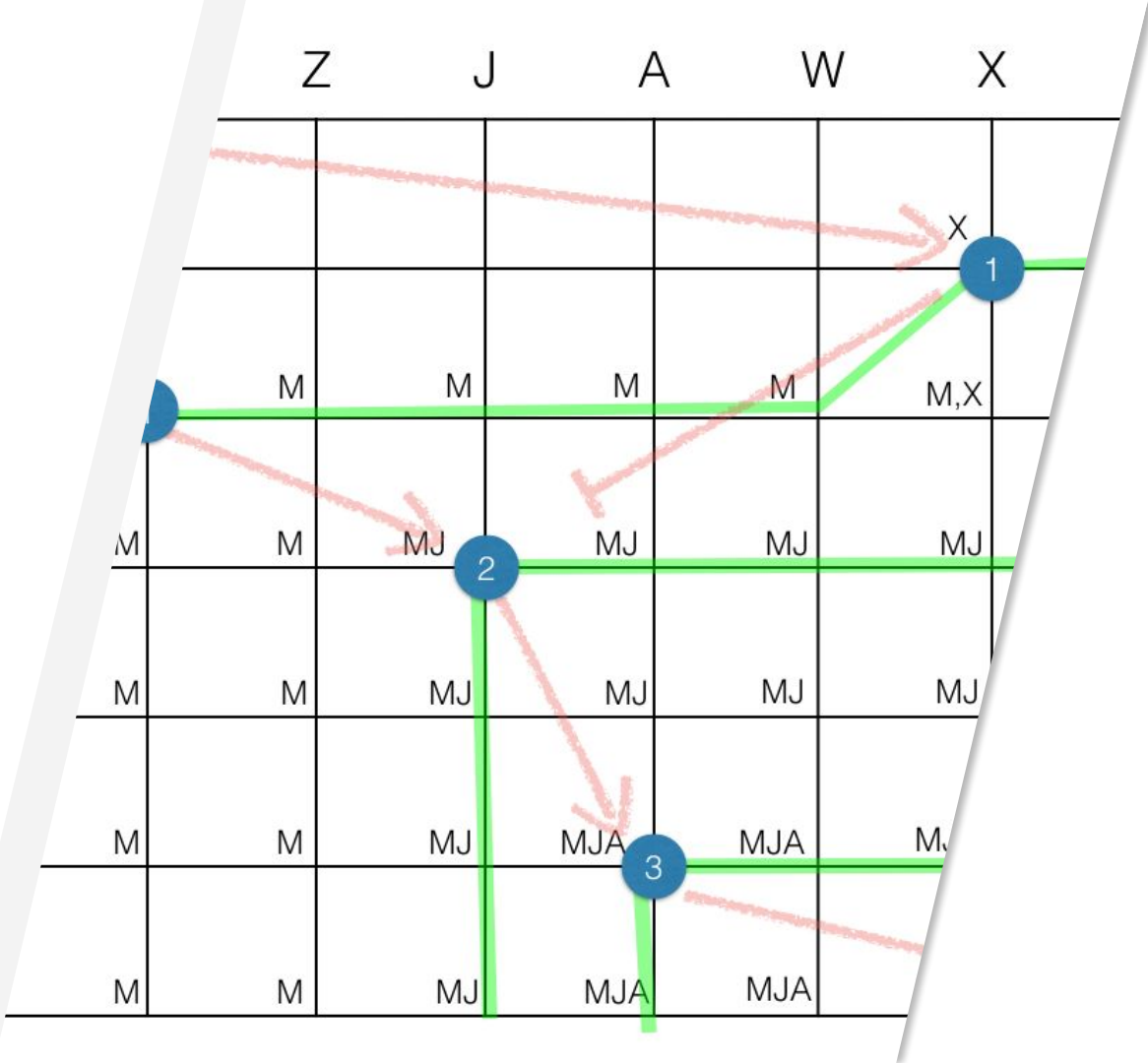
```
RecyclerView.RecycledViewPool recycledViewPool =
                    new RecyclerView.RecycledViewPool();
recyclerView1.setRecycledViewPool(recycledViewPool);
recyclerView2.setRecycledViewPool(recycledViewPool);
recyclerView3.setRecycledViewPool(recycledViewPool);
```

DiffUtil

- DiffUtil is a utility class that can calculate the difference between two lists and output a list of update operations that converts the first list into the second one.
- 局部更新方法 notifyItemXXX() 不适用于所有情况
- notifyDataSetChange() 会导致整个布局重绘，重新绑定所有 ViewHolder，而且会失去可能的动画效果
- DiffUtil 适用于整个页面需要刷新，但是有部分数据可能相同的情况

Myers Diff Algorithm

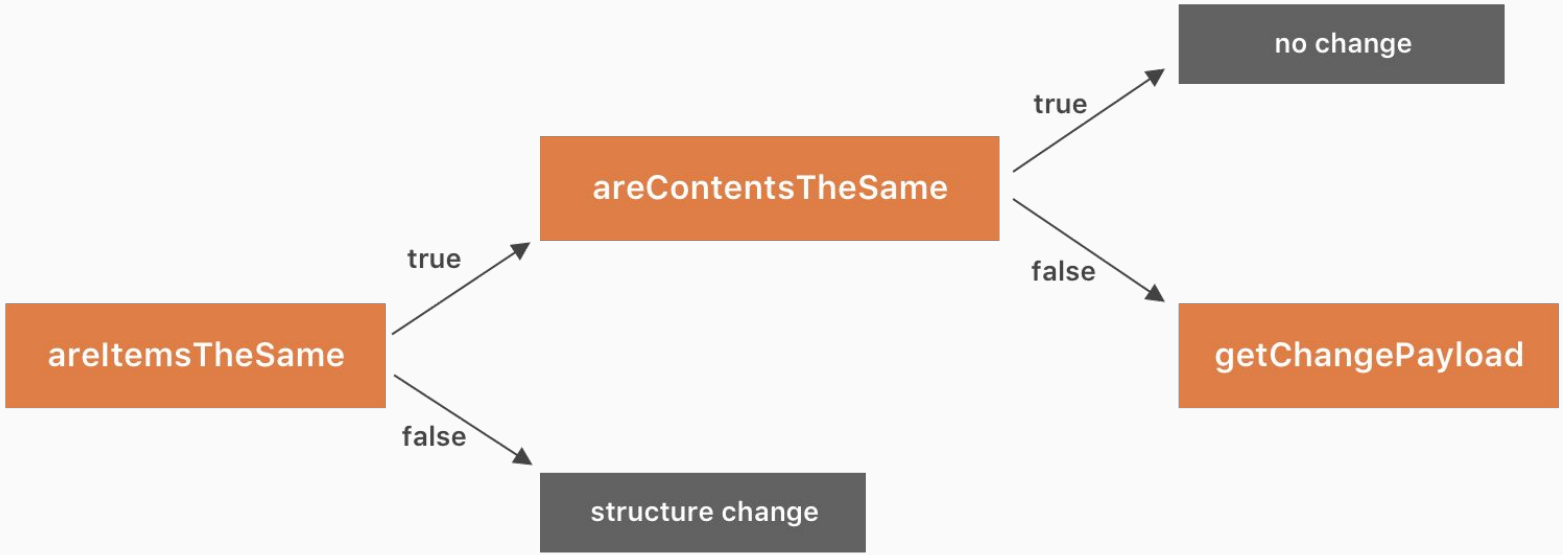
http://blog.robertelder.org/diff-algorithm/



Show you the code

```java
public abstract static class Callback {
    public abstract int getOldListSize();

    public abstract int getNewListSize();

    public abstract boolean areItemsTheSame(int oldItemPosition, int newItemPosition);

    public abstract boolean areContentsTheSame(int oldItemPosition, int newItemPosition);

    public Object getChangePayload(int oldItemPosition, int newItemPosition) {
        return null;
    }
}
```

DiffUtil.Callback 逻辑



```java
public class UserDiffCallback extends DiffUtil.Callback {
    private List<User> oldList;
    private List<User> newList;
    public UserDiffCallback(List<User> oldList, List<User> newList) {
        this.oldList = oldList;
        this.newList = newList;
    }
    @Override
    public int getOldListSize() { return oldList.size();}
    @Override
    public int getNewListSize() { return newList.size();}
    @Override
    public boolean areItemsTheSame(int oldItemPosition, int newItemPosition) {
        return oldList.get(oldItemPosition).id == newList.get(newItemPosition).id;
    }
    @Override
    public boolean areContentsTheSame(int oldItemPosition, int newItemPosition) {
        User oldUser = oldList.get(oldItemPosition);
        User newUser = newList.get(newItemPosition);
        return oldUser.id == newUser.id && oldUser.name.equals(newUser.name)
                && oldUser.profession.equals(newUser.profession);
    }
}
```

```java
public class UserDiffCallback extends DiffUtil.Callback {
    private List<User> oldList;
    private List<User> newList;
    @Nullable
    @Override
    public Object getChangePayload(int oldItemPosition, int newItemPosition) {
        User oldUser = oldList.get(oldItemPosition);
        User newUser = newList.get(newItemPosition);
        Bundle payload = new Bundle();
        if (oldUser.id != newUser.id) {
            payload.putLong(User.KEY_ID, newUser.id);
        }
        if (!oldUser.name.equals(newUser.name)) {
            payload.putString(User.KEY_NAME, newUser.name);
        }
        if (!oldUser.profession.equals(newUser.profession)) {
            payload.putString(User.KEY_PROF, newUser.profession);
        }
        if (payload.size() == 0) return null;
        return payload;
    }
}
```

```java
public class ShowcaseRVAdapter extends RecyclerView.Adapter<ShowcaseRVAdapter.UserViewHolder> {
    private List<User> userList;

    public ShowcaseRVAdapter() {
        userList = new ArrayList<>(UserRepo.USER_LIST);
    }

    public void swapData(List<User> newList, boolean diff) {
        if (diff) {
            DiffUtil.DiffResult diffResult = DiffUtil
                    .calculateDiff(new UserDiffCallback(userList, newList), false);
            userList = newList;
            diffResult.dispatchUpdatesTo(this);
        } else {
            userList = newList;
            notifyDataSetChanged();
        }
    }
}
```

```java
public class ShowcaseRVAdapter extends RecyclerView.Adapter<ShowcaseRVAdapter.UserViewHolder> {
    @Override
    public void onBindViewHolder(@NonNull ShowcaseRVAdapter.UserViewHolder holder, int position) {
        User user = userList.get(position);
        holder.name.setText(user.name);
    }

    @Override
    public void onBindViewHolder(@NonNull UserViewHolder holder, int position, @NonNull List<Object> payloads) {
        if (payloads.isEmpty()) {
            onBindViewHolder(holder, position);
        } else {
            Bundle payload = (Bundle) payloads.get(0);
            for (String key : payload.keySet()) {
                switch (key) {
                case User.KEY_NAME:
                    holder.name.setText(payload.getString(key));
                    break;
                }
            }
        }
    }
}
```

DiffUtil 的效率

- 100 items and 10 modifications: avg: 0.39 ms, median: 0.35 ms
- 100 items and 100 modifications: 3.82 ms, median: 3.75 ms
- 100 items and 100 modifications without moves: 2.09 ms, median: 2.06 ms
- 1000 items and 50 modifications: avg: 4.67 ms, median: 4.59 ms
- 1000 items and 50 modifications without moves: avg: 3.59 ms, median: 3.50 ms
- 1000 items and 200 modifications: 27.07 ms, median: 26.92 ms
- 1000 items and 200 modifications without moves: 13.54 ms, median: 13.36 ms

在列表很大的时候异步计算 diff

- 使用 Thread/Handler 将 DiffResult 发送到主线程
- 使用 RxJava 将 calculateDiff 操作放到后台线程
- 使用 Google 提供的 AsyncListDiffer (Executor) / ListAdapter

AsyncListDiffer / ListAdapter 代码示例

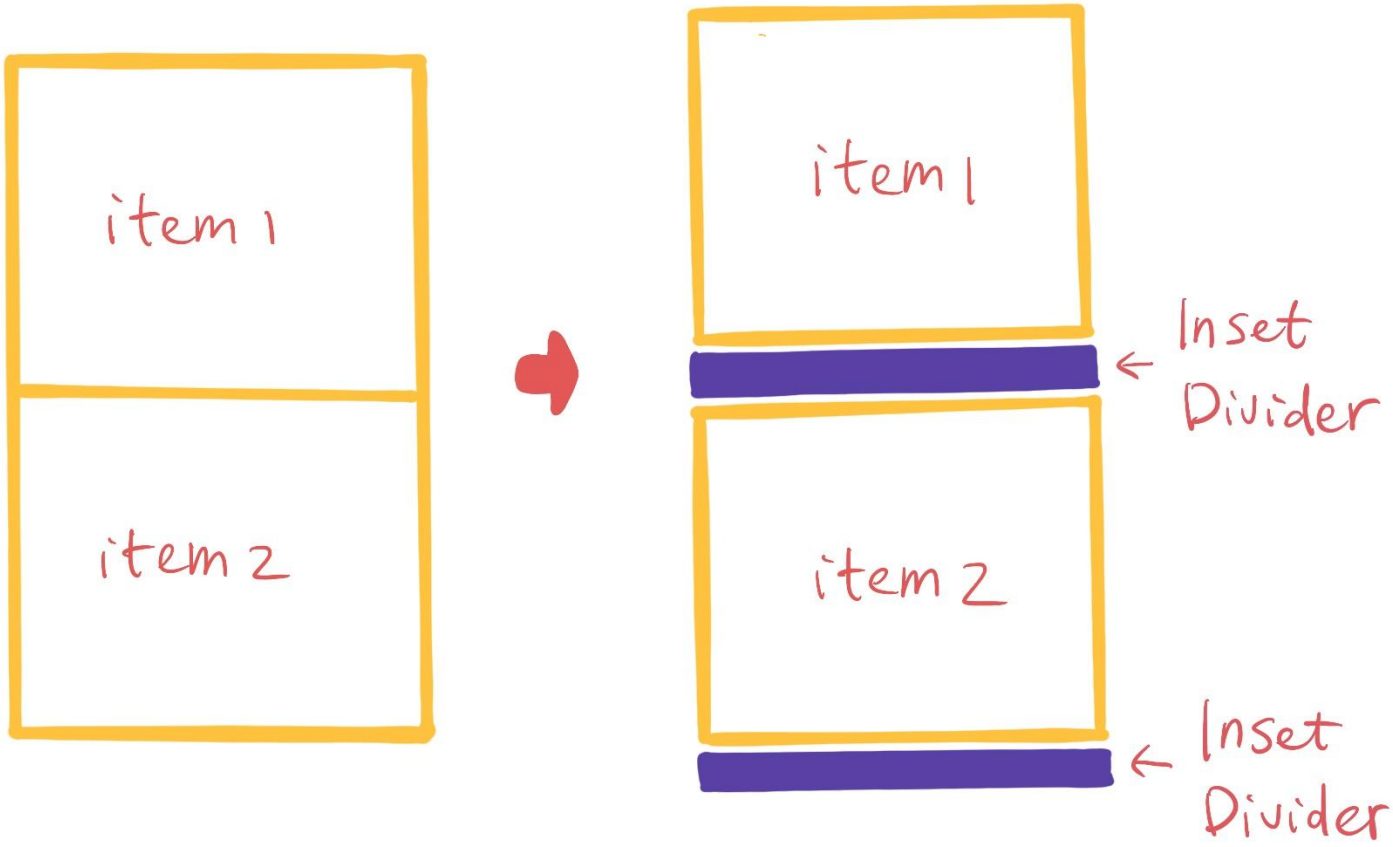
- https://developer.android.com/reference/androidx/recyclerview/widget/AsyncListDiffer
- https://developer.android.com/reference/androidx/recyclerview/widget/ListAdapter

为什么 ItemDecoration 可以绘制分隔线？

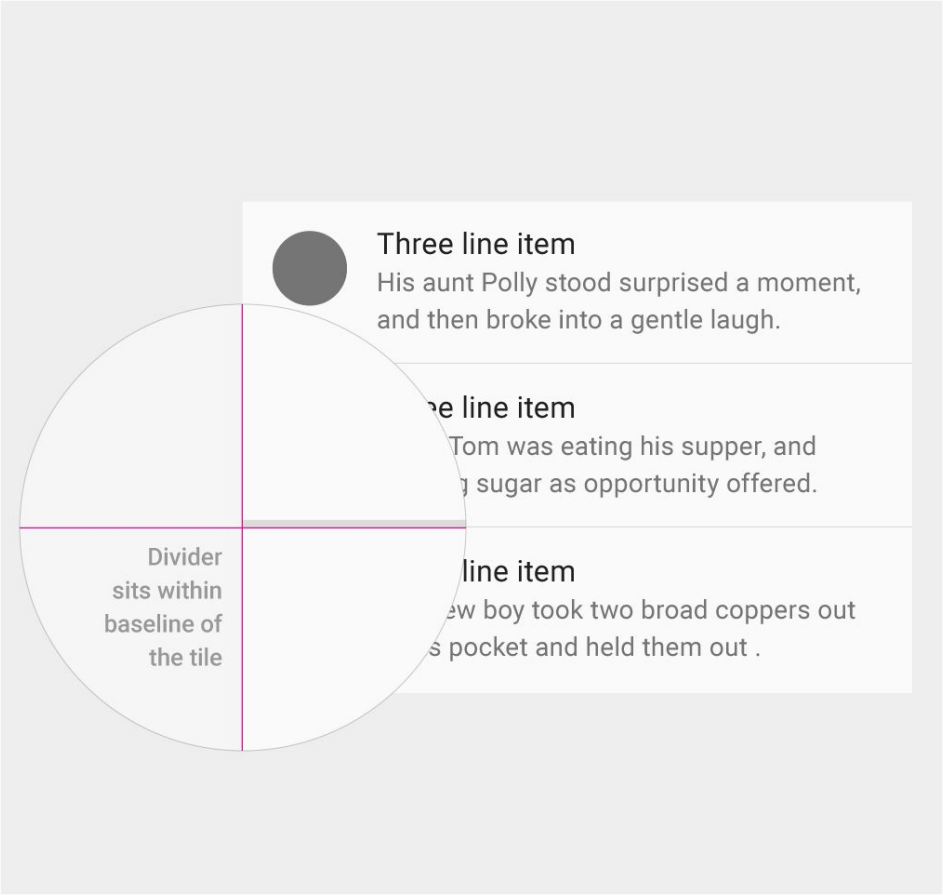
一条分隔线要这么复杂？

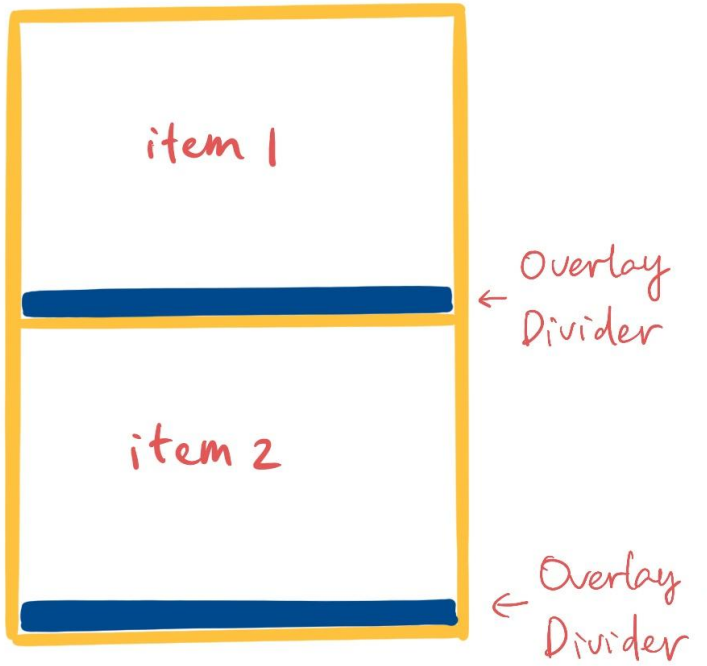
```
/**
 * Current orientation. Either {@link #HORIZONTAL} or {@link #VERTICAL}.
 */
private int mOrientation;

private final Rect mBounds = new Rect();

/**
 * Creates a divider {@link RecyclerView.ItemDecoration} that can be used with a
 * {@link LinearLayoutManager}.
 *
 * @param context Current context, it will be used to access resources.
 * @param orientation Divider orientation. Should be {@link #HORIZONTAL} or {@link #VERTICAL}.
 */
public DividerItemDecoration(Context context, int orientation) {
    final TypedArray a = context.obtainStyledAttributes(ATTRS);
    mDivider = a.getDrawable(0);
    if (mDivider == null) {
        Log.w(TAG, "@android:attr/listDivider was not set in the theme used for this "
                + "DividerItemDecoration. Please set that attribute all call setDrawable()");
    }
    a.recycle();
```

Inset Divider



Overlay Divider



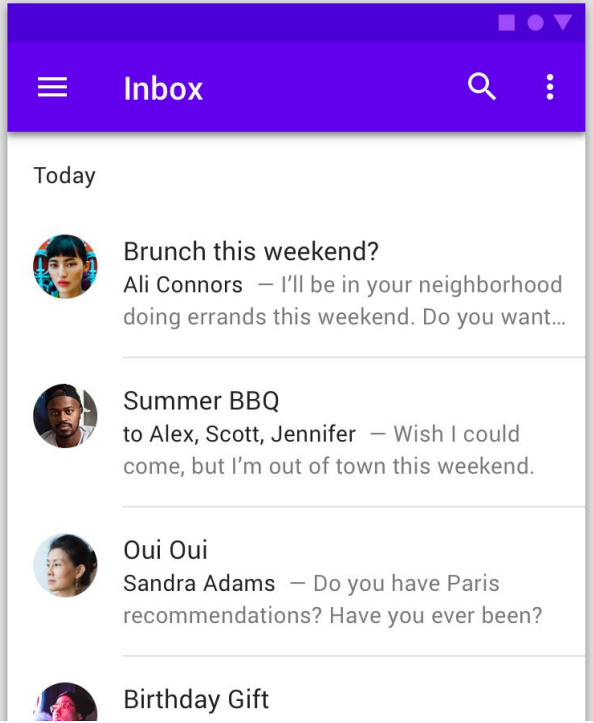


Overlay Divider 代码

```
public class OverlayDivider extends RecyclerView.ItemDecoration {

    @Override
    public void onDrawOver(Canvas c, RecyclerView parent, RecyclerView.State state) {
        if (mOrientation == VERTICAL_LIST) {
            drawVertical(c, parent);
        } else {
            drawHorizontal(c, parent);
        }
    }

    protected void drawVertical(Canvas c, RecyclerView parent) {}

    protected void drawHorizontal(Canvas c, RecyclerView parent) {}

    // 不需要覆写 getItemOffsets()
}
```

快速提问

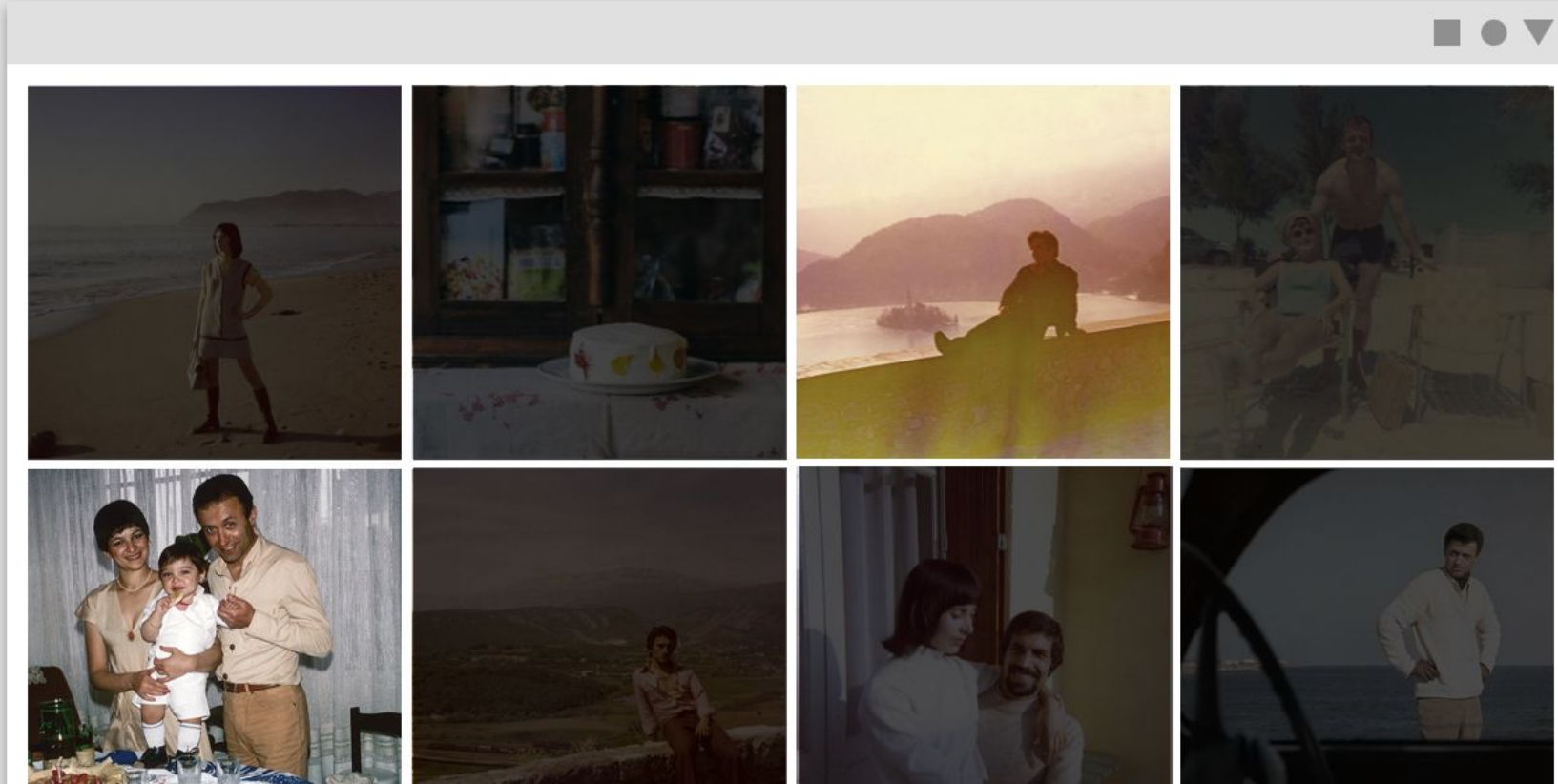
这样的分隔线应该怎么画？



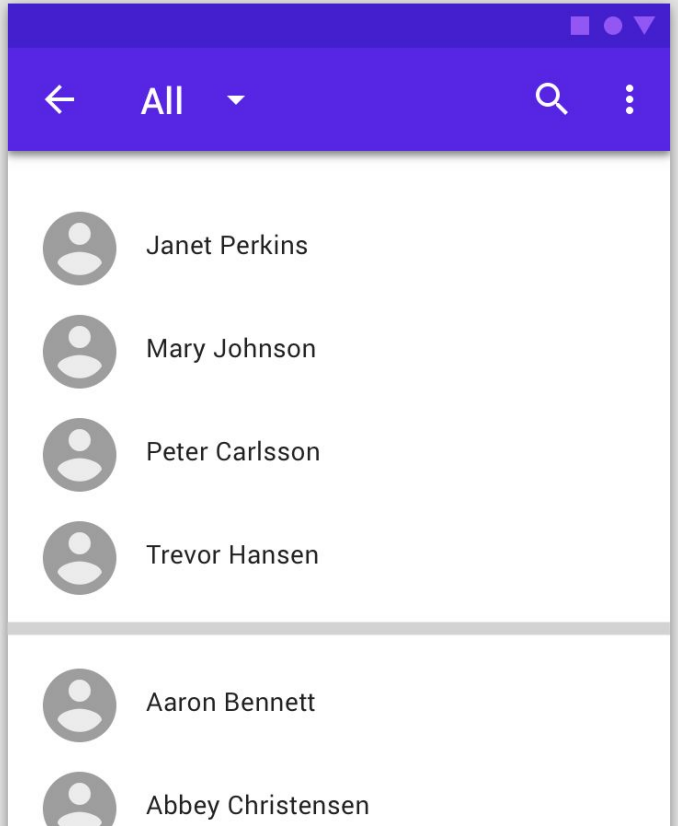
ItemDecoration 还可以做什么？

- Drawing dividers between items
- Highlights
- Visual grouping boundaries

Highlights



Visual grouping boundaries



RecyclerView 更多知识

- https://github.com/h6ah4i/android-advancedrecyclerview
- https://advancedrecyclerview.h6ah4i.com/

谢谢大家！