
Google
Developer
Day2009

Android 用户界面编程技巧

Grace Kloba
2009.06.05

Google
Developer
Day2009

技巧和设计模式

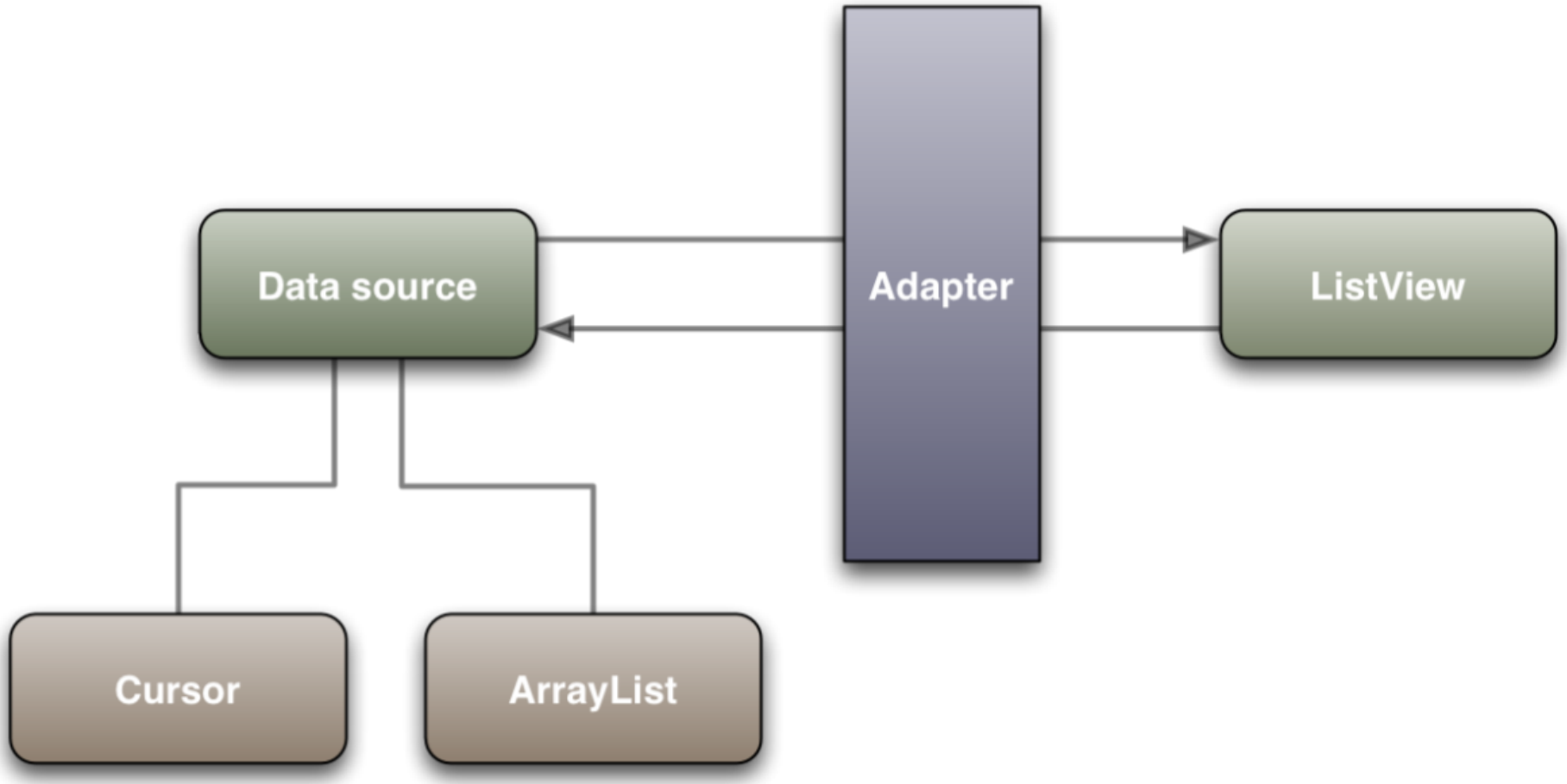
- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

技巧和设计模式

- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

Adapter

- Adapter 是 ListView 和数据源之间的中间人

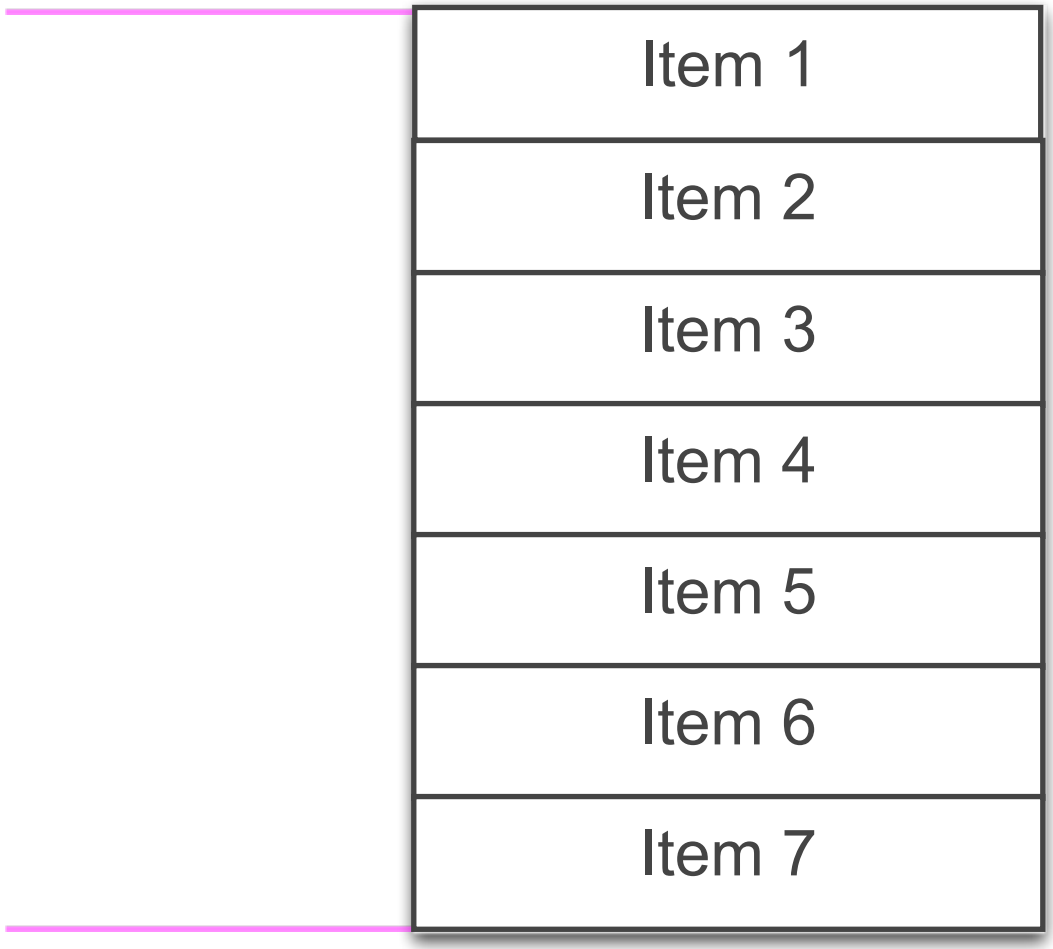


Adapter

- 当每条数据进入可见区时
  - Adapter 的 getView() 会被调用
  - 返回代表具体数据的视图
- 触摸滚动时，频繁调用
- 支持成百上千条数据

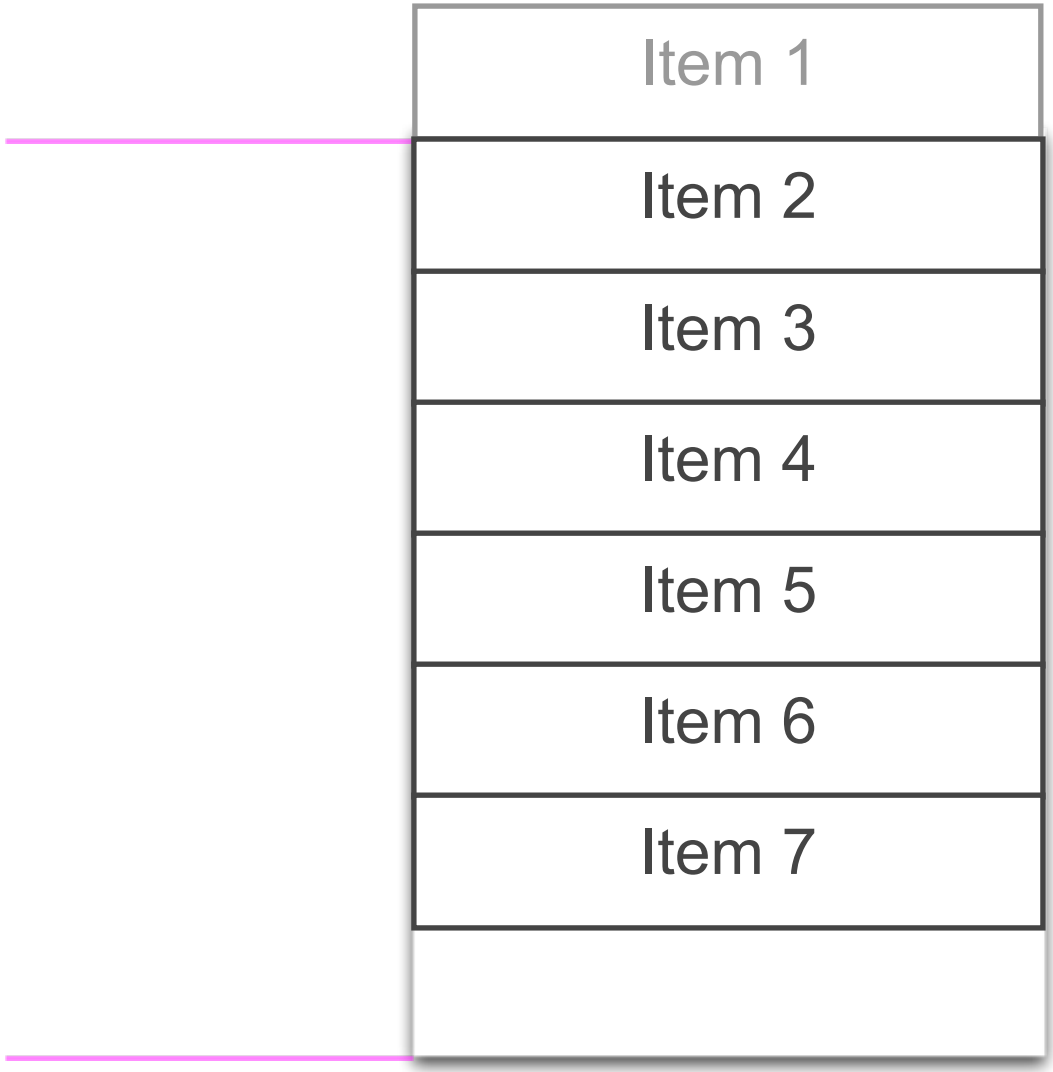
Adapter

剖析ListView



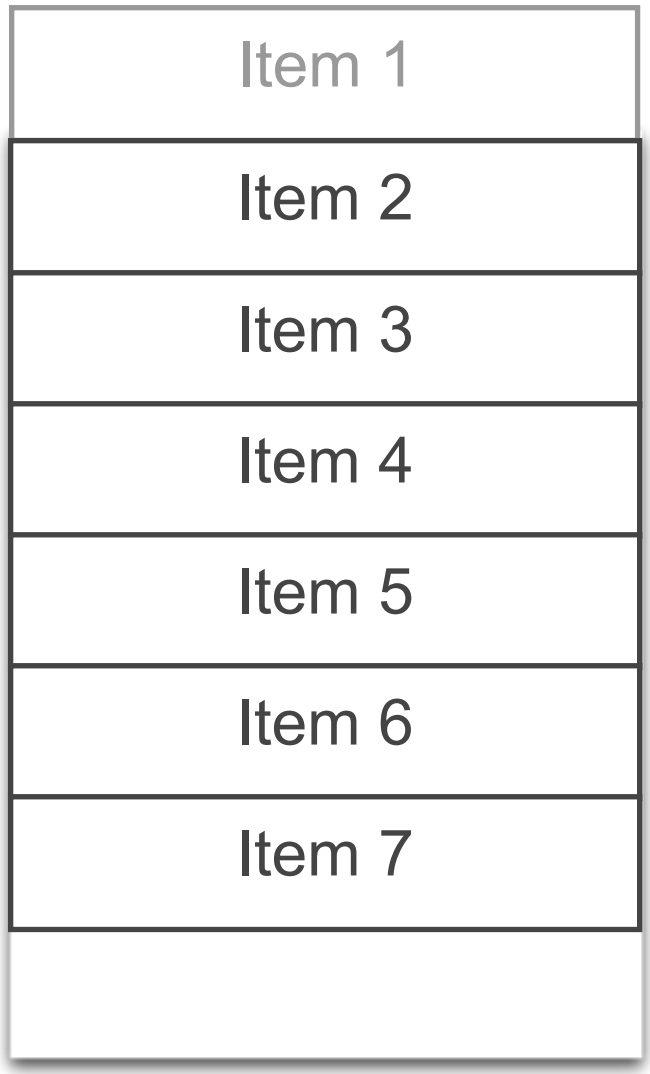
Adapter

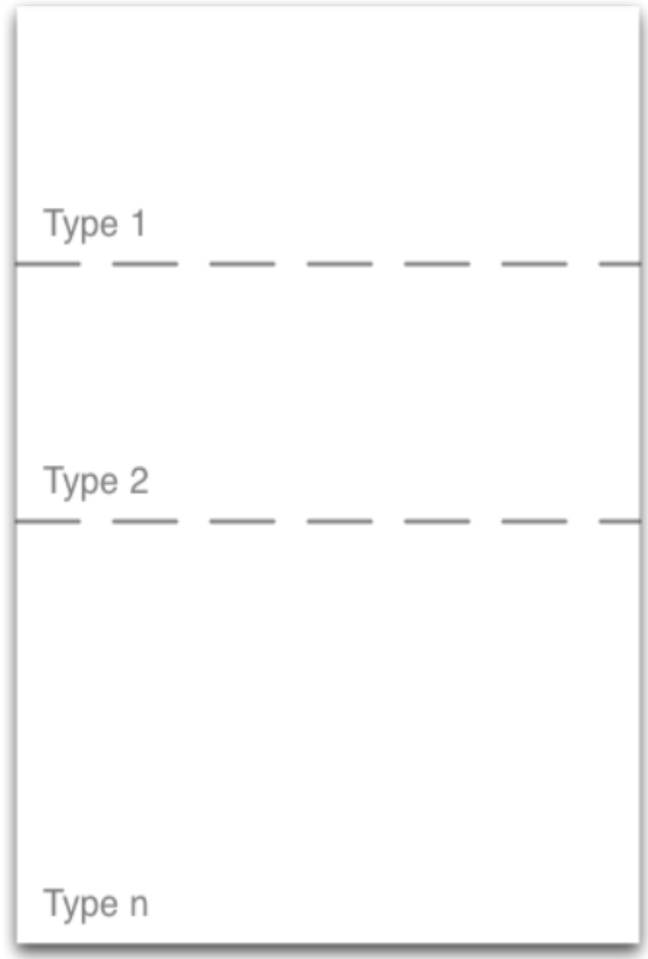
剖析ListView



Adapter

剖析ListView



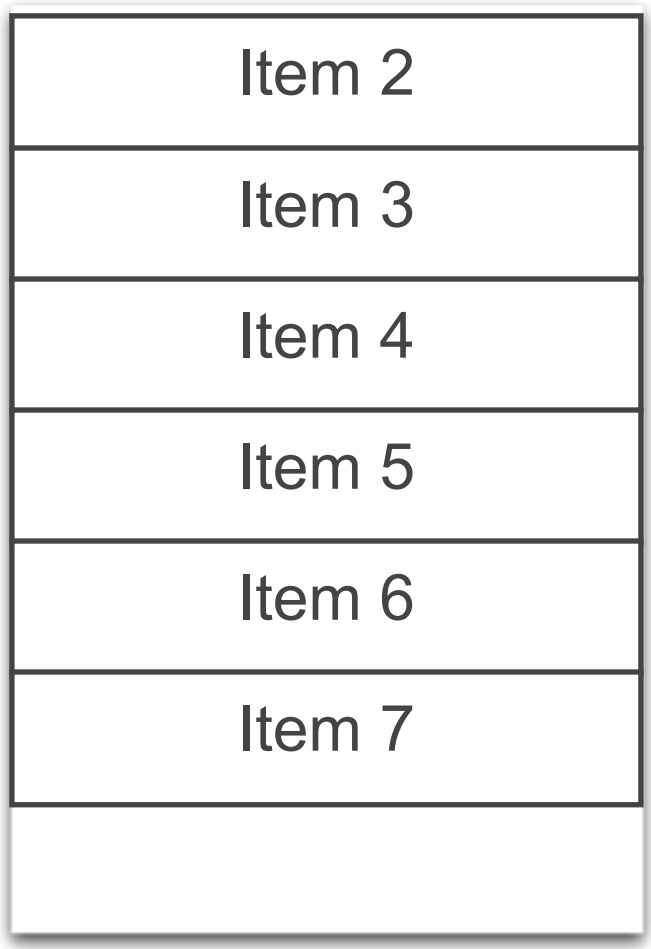


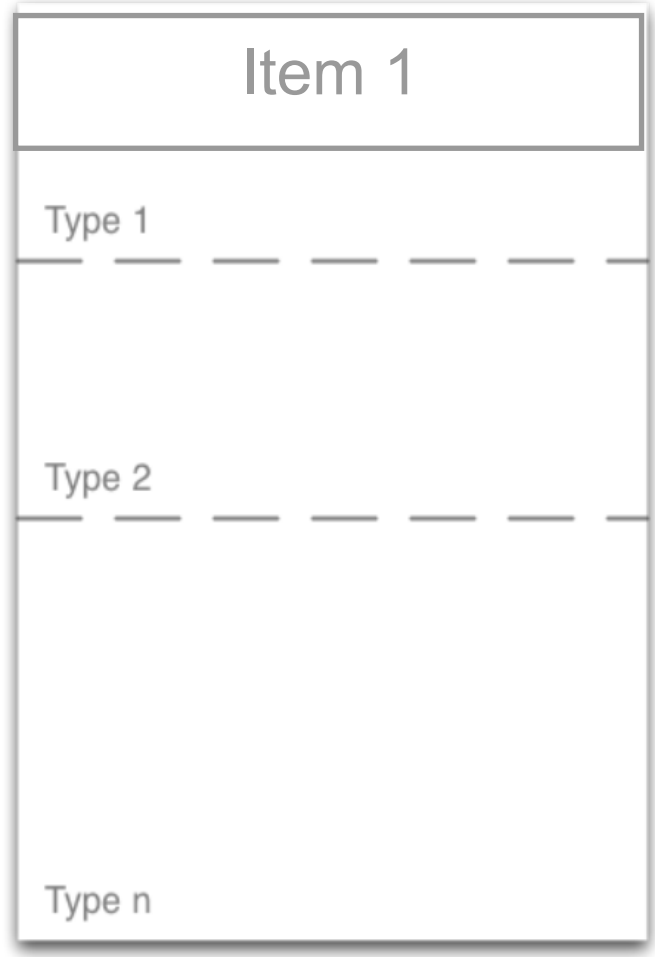
Recycler

Google Developer Day2009

Adapter

剖析ListView



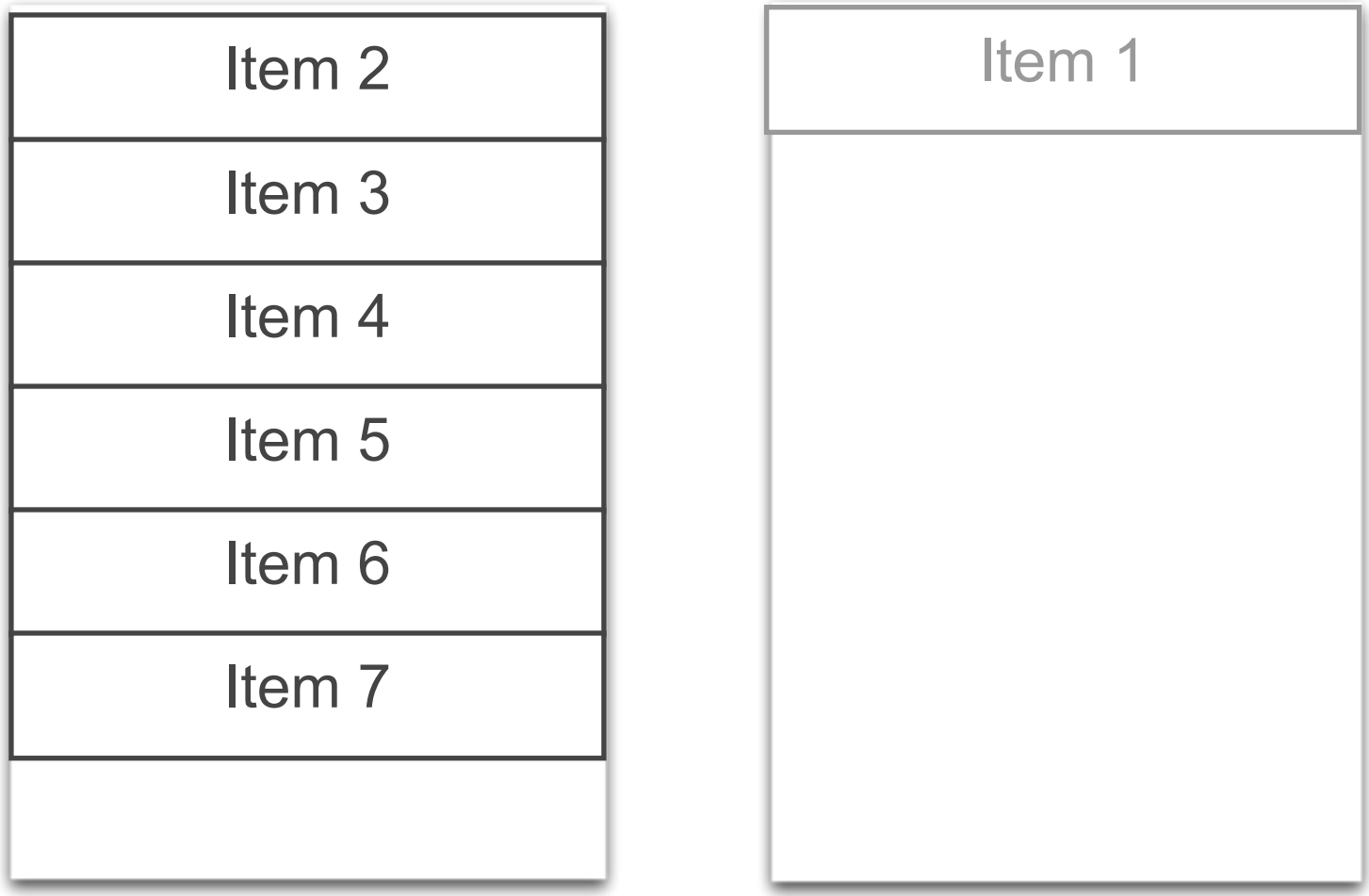


Recycler

Google Developer Day2009

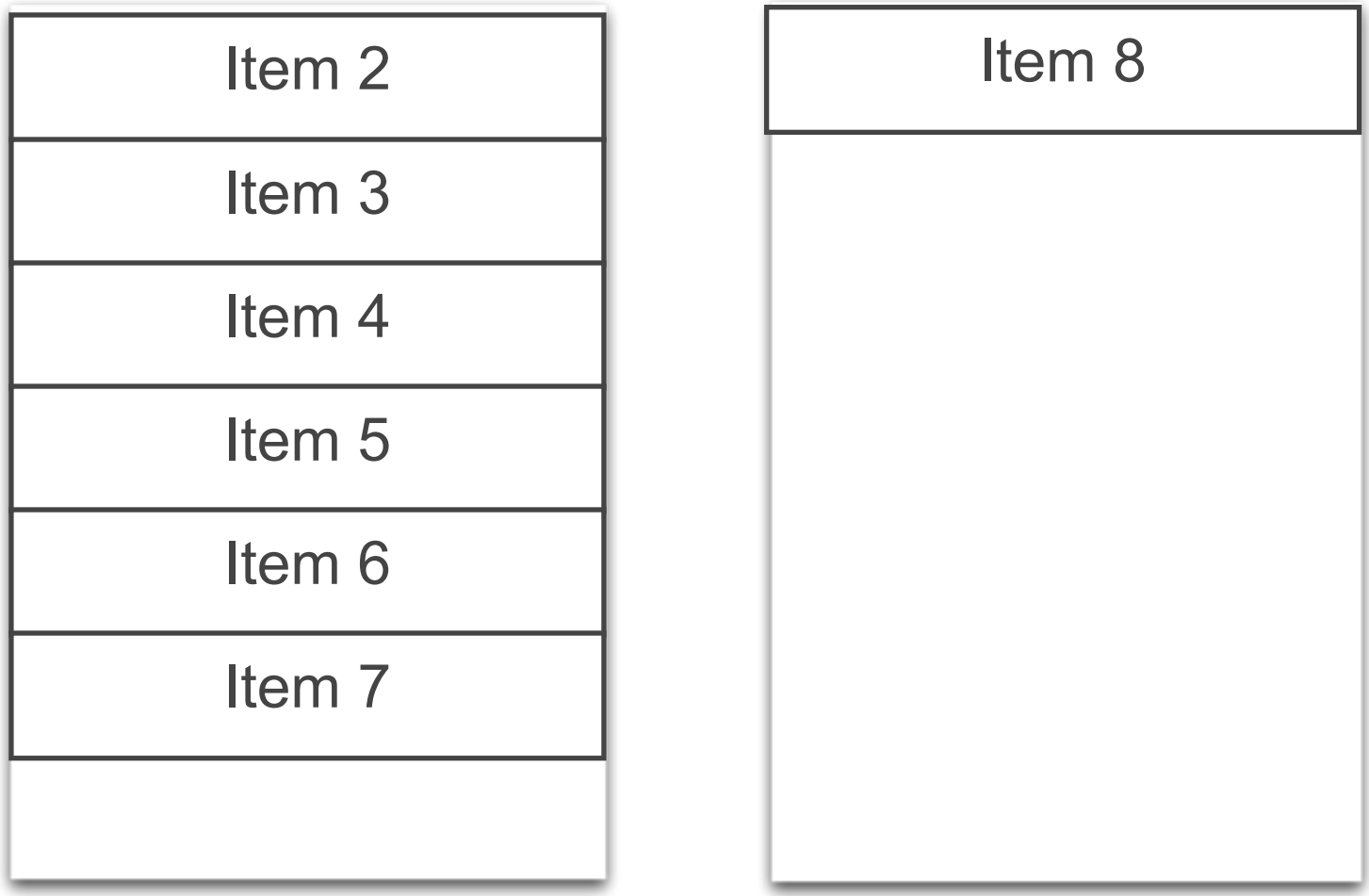
Adapter

剖析ListView



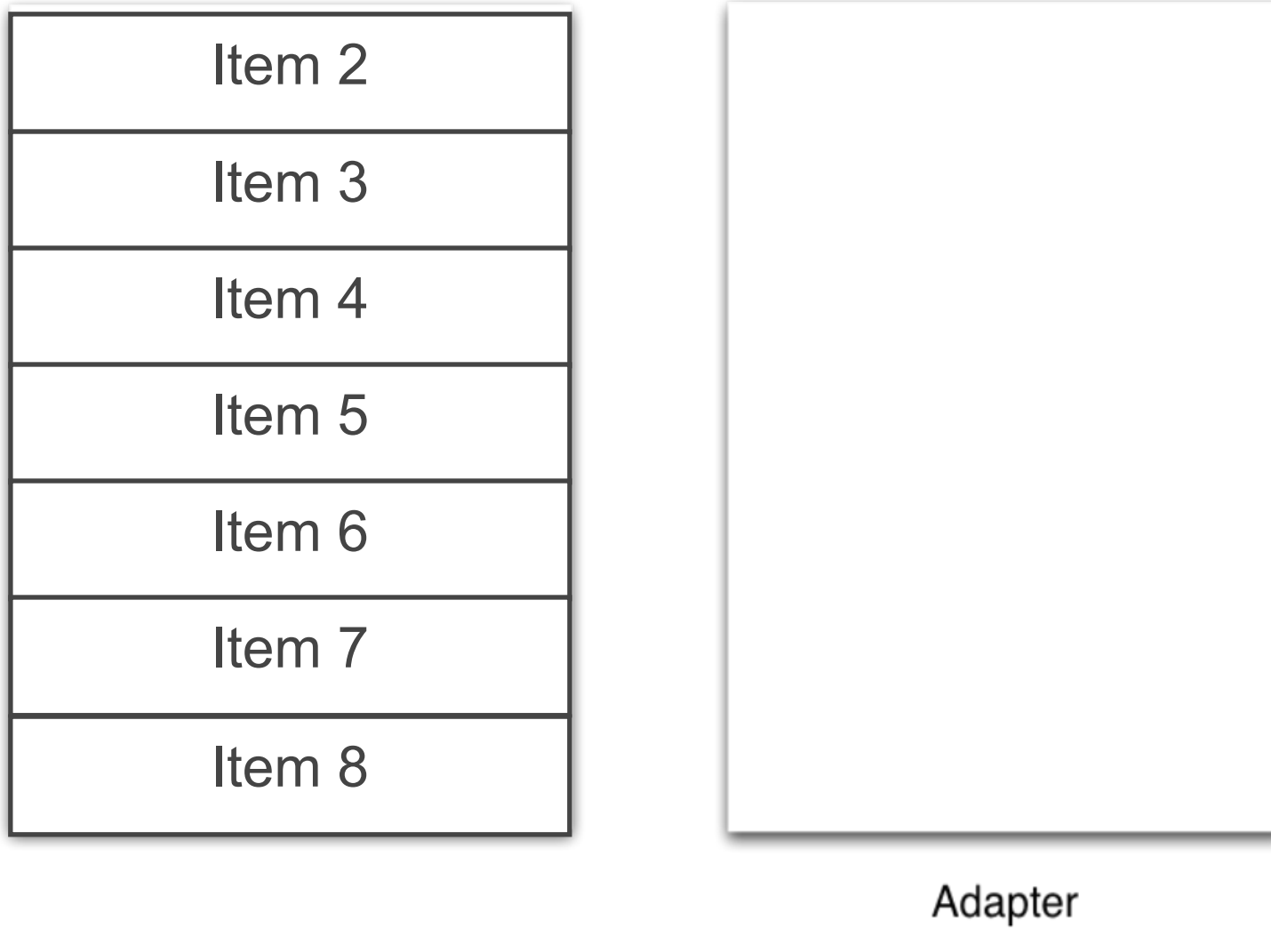
Adapter

剖析ListView



Adapter

剖析ListView



Adapter

显示每条数据的 XML 布局文件


```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal">
    <ImageView android:id="@+id/icon"
        android:layout_width="48dip"
        android:layout_height="48dip" />
    <TextView android:id="@+id/text"
        android:layout_gravity="center_vertical"
        android:layout_width="0dip"
        android:layout_weight="1.0"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Adapter

最简单的方法，最慢最不实用

```
public View getView(int pos, View convertView,
        ViewGroup parent){
    View item = mInflater.inflate(R.layout.list_item, null);

    ((TextView) item.findViewById(R.id.text)).
            setText(DATA[pos]);
    ((ImageView) item.findViewButId(R.id.icon)).
            setImageBitmap((pos & 1) == 1 ? mIcon1 : mIcon2);

    return item;
}
```

Adapter

利用 convertView 回收视图，效率提高 200%

```
public View getView(int pos, View convertView,
        ViewGroup parent){
    if (convertView == null) {
        convertView = mInflater.inflate(
                R.layout.list_item, null);
    }

    ((TextView) convertView.findViewById(R.id.text)).
            setText(DATA[pos]);
    ((ImageView) convertView.findViewButId(R.id.icon)).
            setImageBitmap((pos & 1) == 1 ? mIcon1 : mIcon2);

    return convertView;
}
```

Adapter

使用 ViewHolder 模式, 效率再提高 50%

```
static class ViewHolder {
    TextView text;
    ImageView icon;
}
```

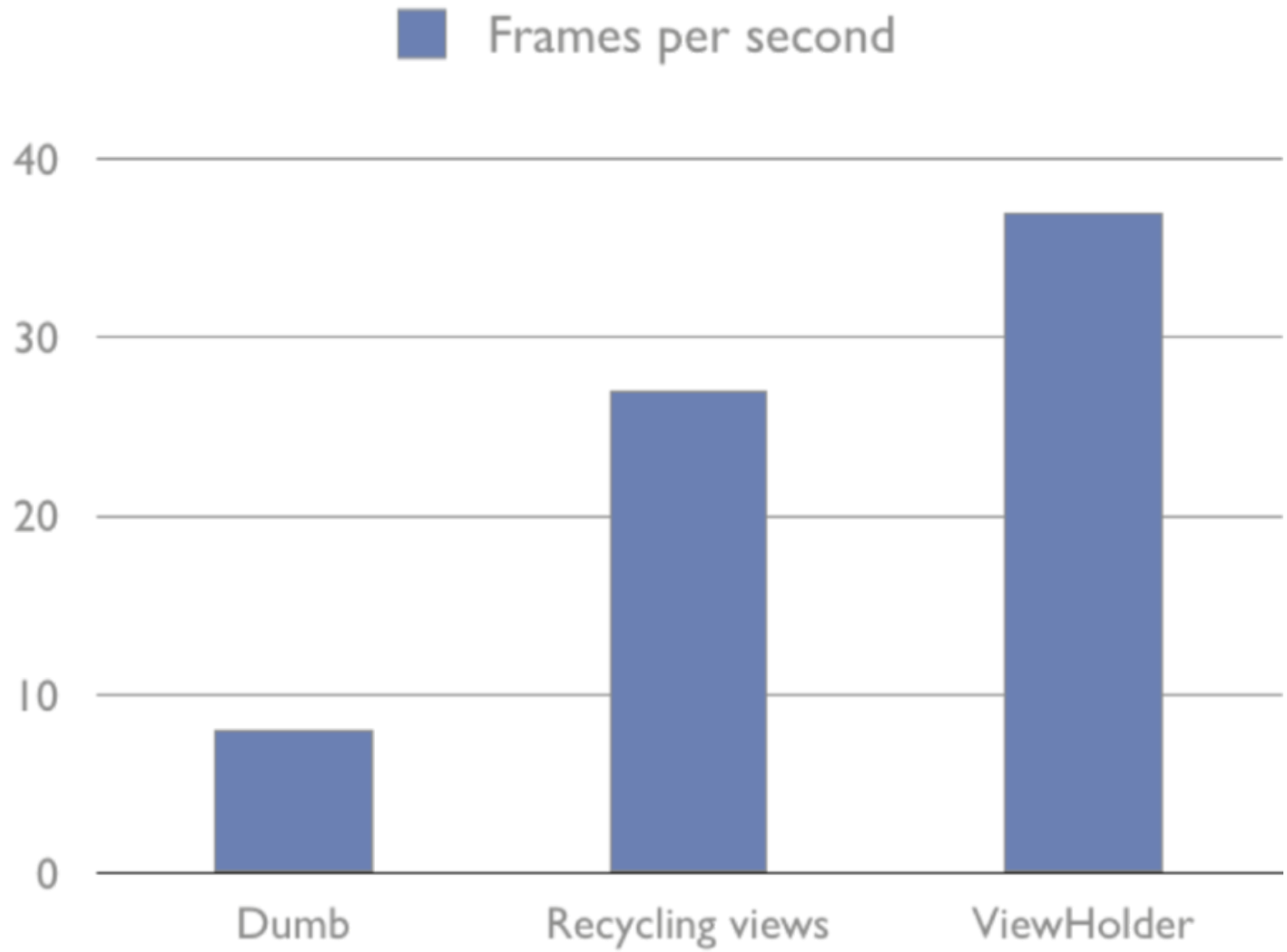
Adapter

使用 ViewHolder 模式, 效率再提高 50%

```
public View getView(int pos, View convertView, ViewGroup parent){
    ViewHolder holder;
    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.list_item, null);
        holder = new ViewHolder();
        holder.text = (TextView) convertView.findViewById(
                R.id.text));
        holder.icon = (ImageView) convertView.findViewButId(
                R.id.icon));
        convertView.setTag(holder);
    } else {
        holder = (ViewHolder) convertView.getTag();
    }
    holder.text.setText(DATA[pos]);
    holder.icon.setImageBitmap((pos & 1) == 1 ? mIcon1 : mIcon2);
    return convertView;
}
```

Adapter

更新率比较



Google Developer Day2009

技巧和设计模式

- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

背景和图像

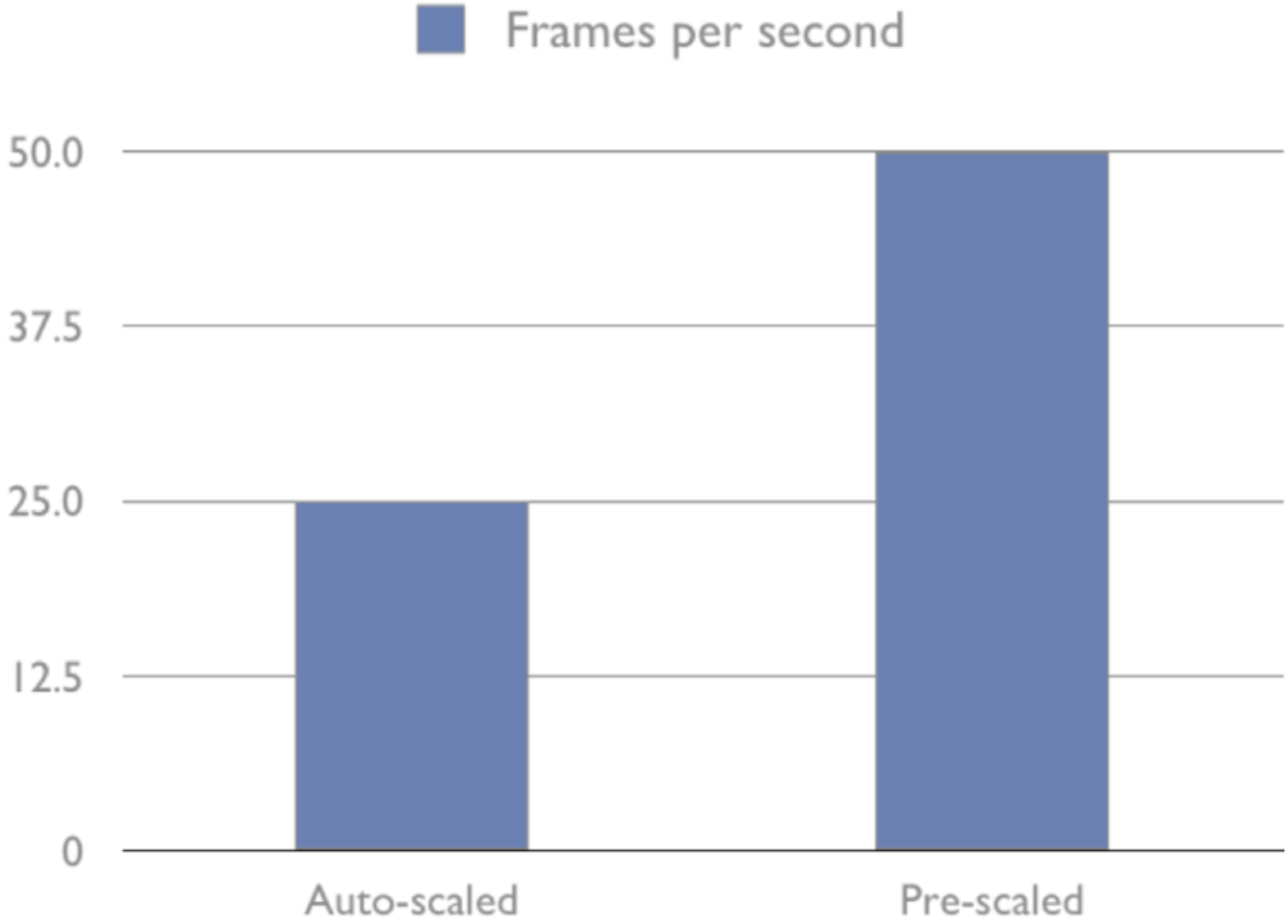
选择恰当的图像尺寸

- 视图背景图像总会填充整个视图区域
  - 图像尺寸不合适会导致自动缩放
  - 避免实时缩放
  - 最好预先缩放到视图大小

```
originalImage = Bitmap.createScaledBitmap(
        originalImage,     // 被缩放图像
        view.getWidth(),   // 视图宽度
        view.getHeight(), // 视图高度
        true);             // 双线性过滤器
```

背景和图像

更新率比较



背景和图像

窗口背景

- 默认情况下，窗口有一个不透明的背景
- 有时可以不需要
  - 最高层的视图是不透明的
  - 最高层的视图覆盖整个窗口
    - layout_width = fill_parent
    - layout_height = fill_parent
- 更新看不见的背景是浪费时间

背景和图像

删除窗口背景

- 方法一：修改编码

```
@Override
public void onCreate(Bundle icicle){
    super.onCreate(icicle);
    setContentView(R.layout.mainview);

    // 删除窗口背景
    getWindow().setBackgroundDrawable(null);

    ...
}
```

背景和图像

删除窗口背景

- 方法二：修改 XML 声明
  - 首先确定你的 res/values/styles.xml 有

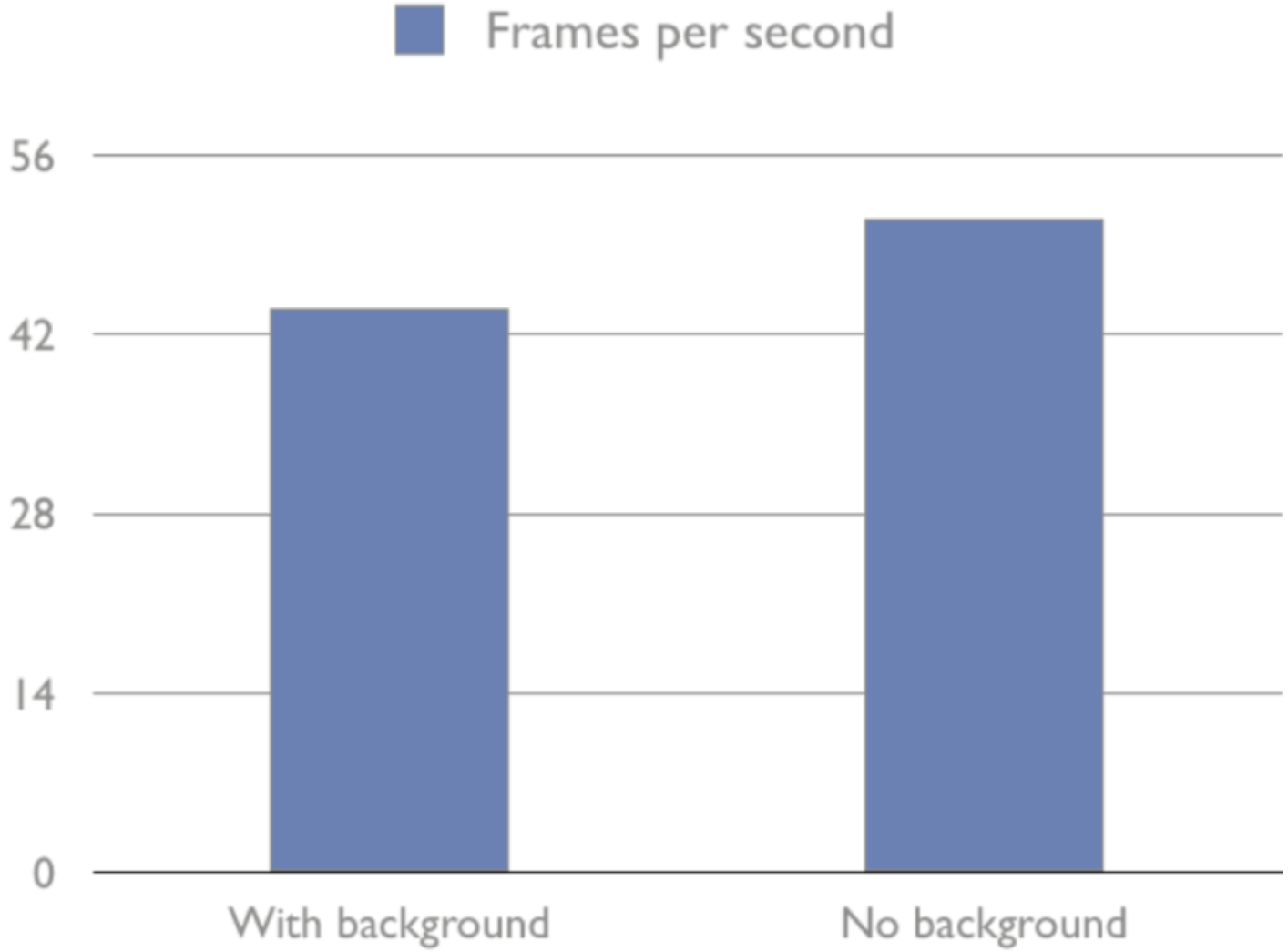
```
<resources>
    <style name="NoBackgroundTheme" parent="android:Theme">
        <item name="android:windowBackground">@null</item>
    </style>
</resources>
```

  - 然后编辑 AndroidManifest.xml

```
<activity android:name="MyApplication"
        android:theme="@style/NoBackgroundTheme">
...
</activity>
```

背景和图像

更新率比较


Frames per second
56
42
28
14
0
With background
No background

技巧和设计模式

- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

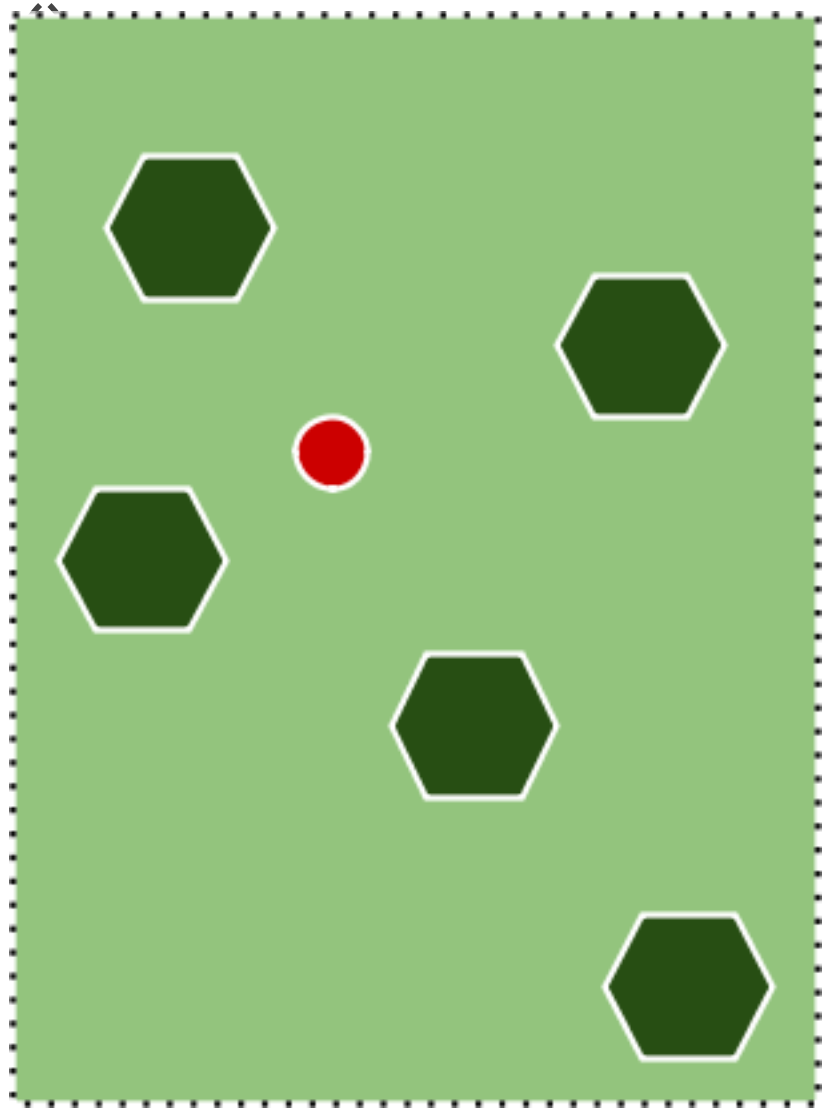
更新请求

- 当屏幕需要更新时，调用 invalidate()
  - 简单方便
  - 但会更新整个视图，太贵了
- 最好先找到无效区域，然后调用
  - invalidate(Rect dirty);
  - invalidate(int left, int top, int right, int bottom);

更新请求

应用实例：在屏幕上触摸移动小图标

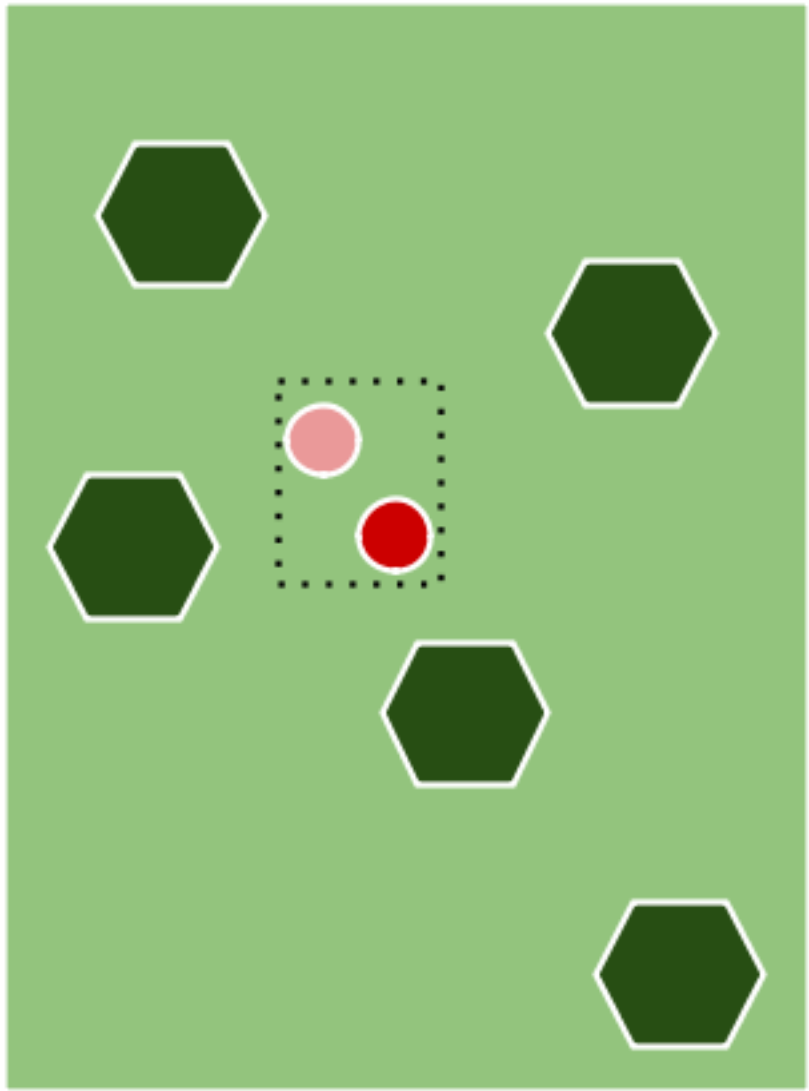
- 最简单的编码在每次响应移动事件时调用 invalidate()



更新请求

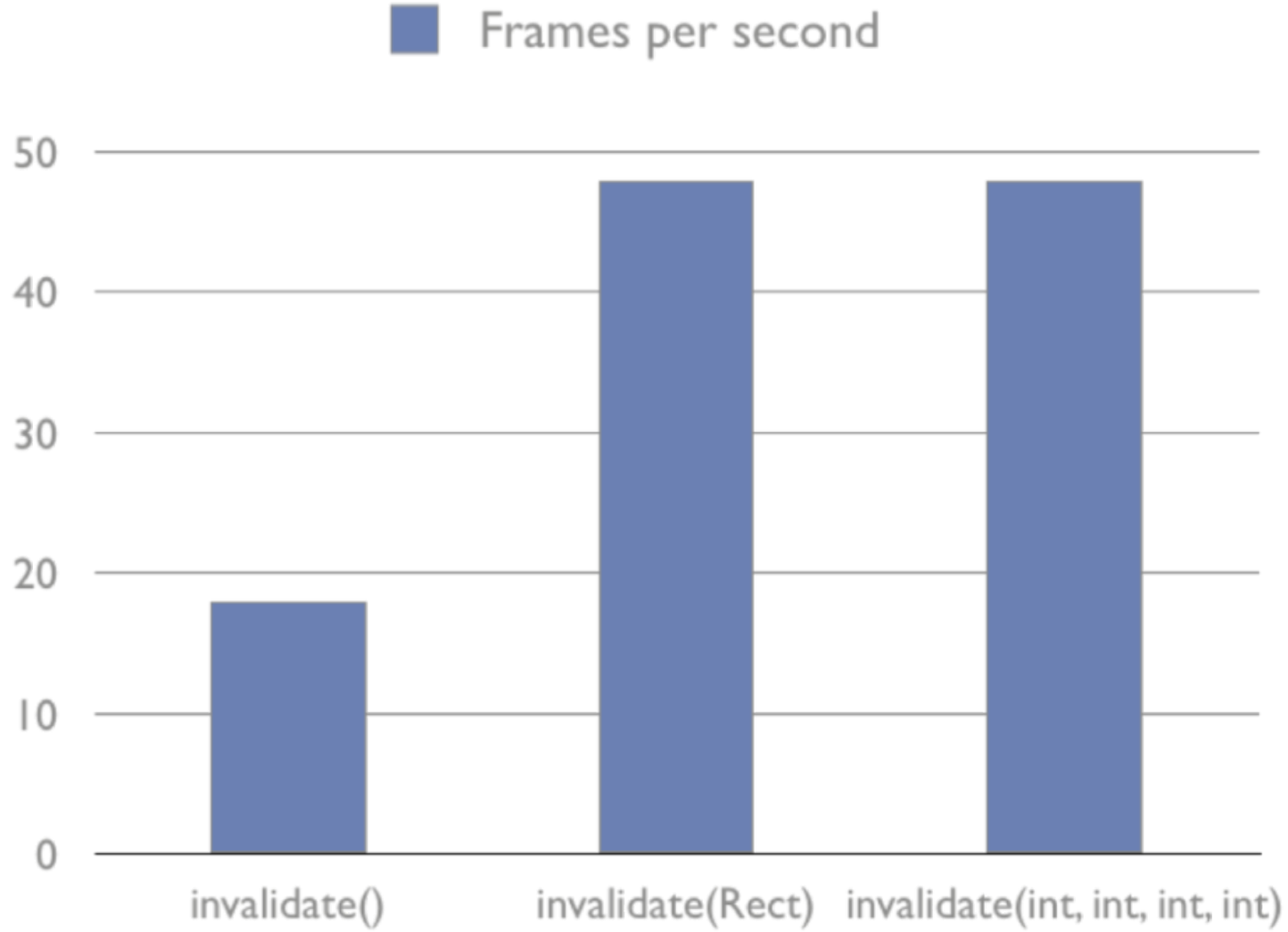
应用实例：在屏幕上触摸移动小图标

- 更有效的执行方法是只更新需要更新的区域



更新请求

更新率比较


Frames per second
50
40
30
20
10
0
invalidate()
invalidate(Rect)
invalidate(int, int, int, int)

技巧和设计模式

- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

视图和布局

越简单越好

- 如果一个窗口包含很多视图
  - 启动时间长
  - 测量时间长
  - 布局时间长
  - 绘制时间长
- 如果视图树深度太深
  - StackOverflowException
  - 用户界面反应速度很慢

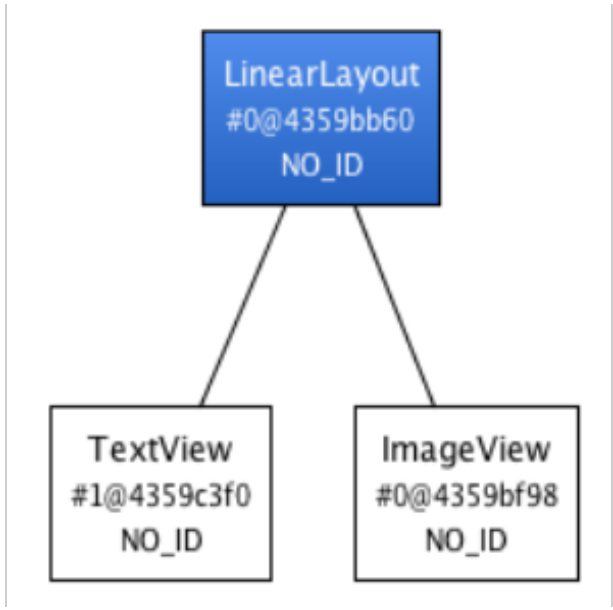
视图和布局

解决方法

- 使用 TextView 的复合 drawables 减少层次
- 使用 ViewStub 延迟展开视图
- 使用 <merge> 合并中间视图
- 使用 RelativeLayout 减少层次
- 使用自定义视图
- 使用自定义布局

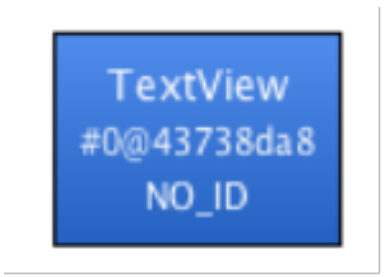
视图和布局

使用 TextView 的复合 drawables 减少层次



视图和布局

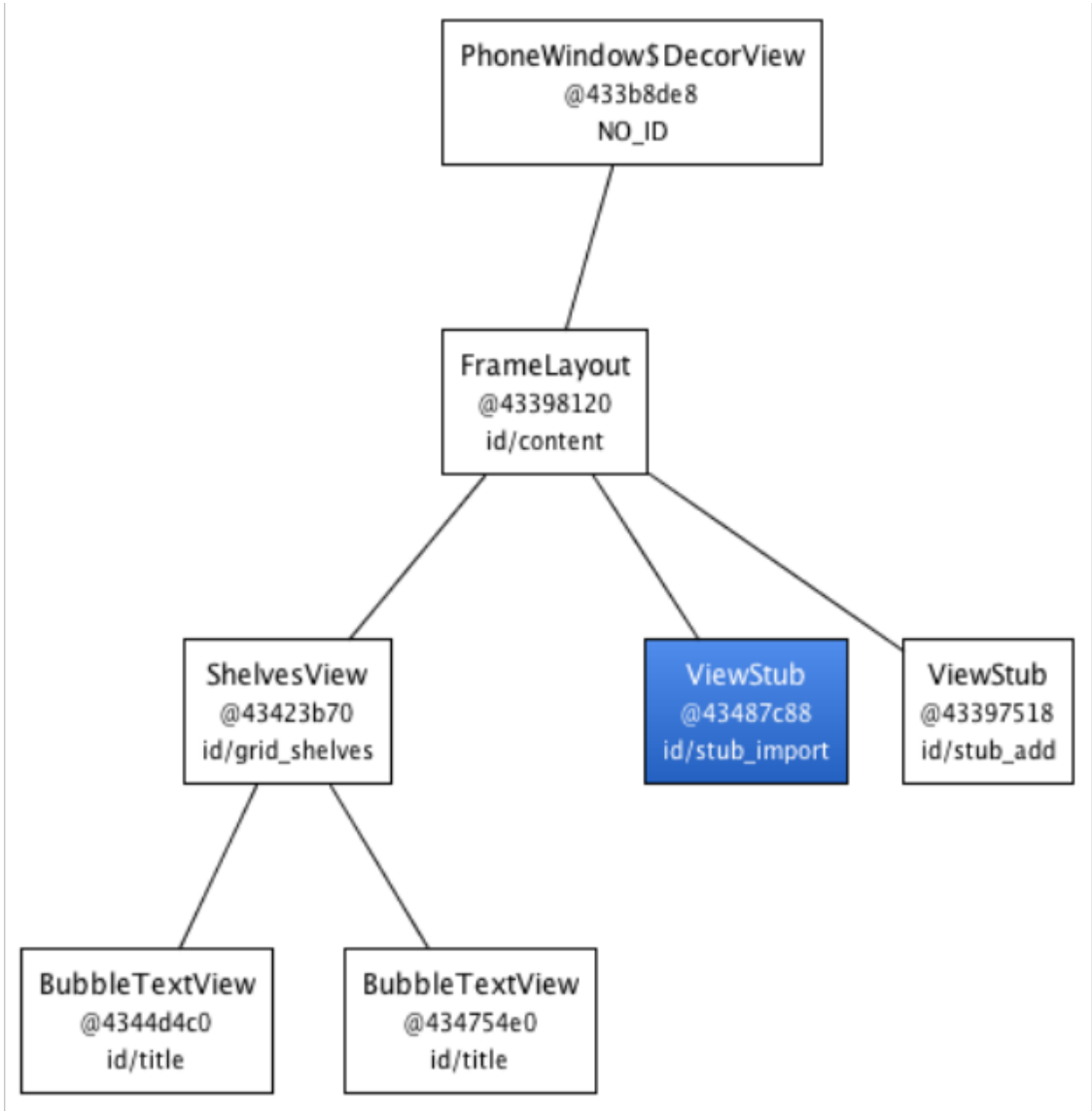
使用 TextView 的复合 drawables 减少层次




```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:drawableLeft="@drawable/icon"/>
```

视图和布局

使用 ViewStub 延迟展开视图



视图和布局

使用 ViewStub 延迟展开视图

- 首先在 XML 布局文件中定义 ViewStub

```
<ViewStub android:id = "@+id/stub_import"
          android:inflatedId="@+id/panel_import"
          android:layout="@layout/progress_overlay"
          android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:layout_gravity="bottom"/>
```

- 在需要展开视图时

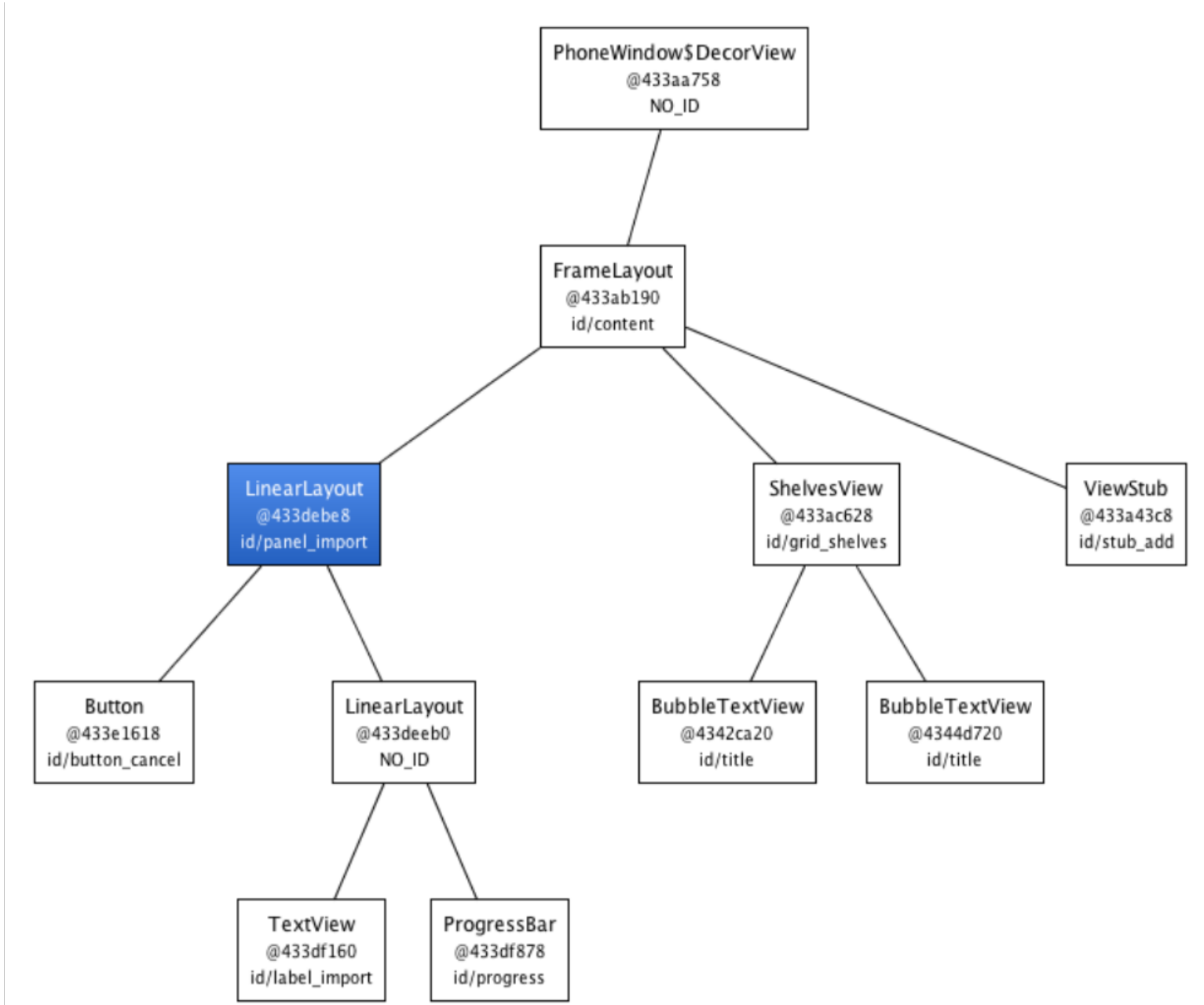
```
findViewById(R.id.stub_import).setVisibility(View.VISIBLE);

// 或者

View importPanel = ((ViewStub)
        findViewById(R.id.stub_import)).inflate();
```

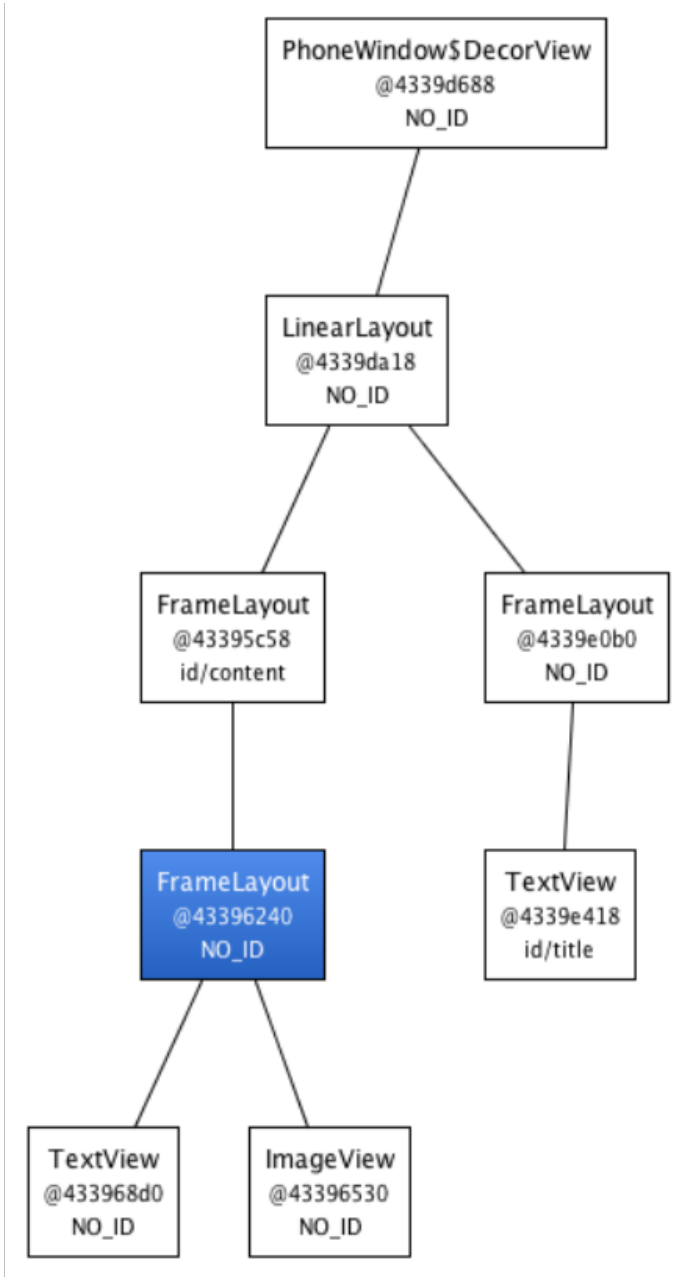
视图和布局

使用 ViewStub 延迟展开视图



视图和布局

使用 <merge> 合并视图



视图和布局

使用 <merge> 合并视图

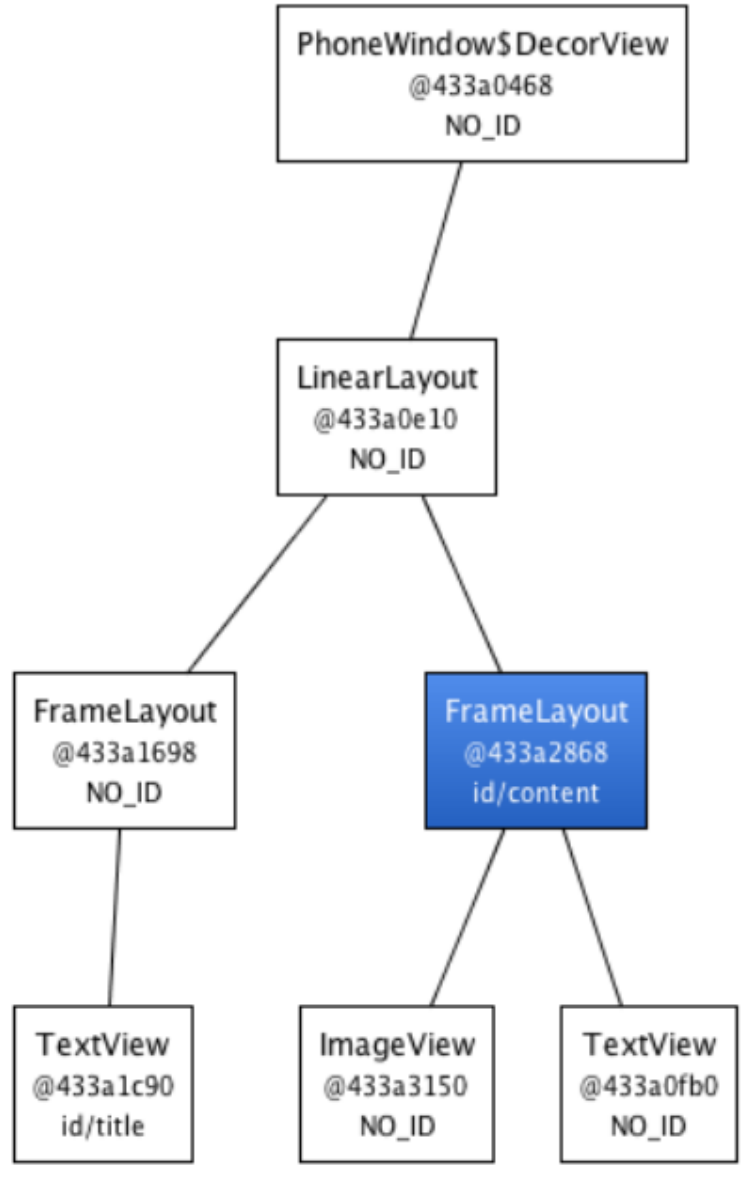
- 默认情况下，布局文件的根作为一个结点加入到父视图中
- 如果使用 <merge> 可以避免根接点

```
<!-- The merge tag must be the root tag -->
<merge xmlns:android =
         "http://schemas.android.com/apk/res/android">
    <! -- Content -->

</merge>
```

视图和布局

使用 <merge> 合并视图



视图和布局

使用 RelativeLayout 减少层次

- 很有功效
- 可以代替 LinearLayout
  - 可以在水平 LinearLayout 中加入垂直 LinearLayout
  - 反过来也可以
- 可以用单层次表达复杂布局

视图和布局

使用 RelativeLayout 减少层次

视图和布局

使用 RelativeLayout 减少层次

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content">
    <ImageView android:id="@+id/icon"
        android:layout_width="48dip" android:layout_height="48dip"
        android:layout_alignParentLeft="true"
        android:layout_centerVertical="true"/>
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/text_line1"
        android:layout_alignParentTop="true"
        android:layout_toRightOf="@id/icon"/>
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:id="@+id/text_line2"
        android:layout_toRightOf="@id/icon"
        android:layout_below="@id/text_line1"/>
    <Checkbox android:id="@+id/star"
        android:layout_width="48dip" android:layout_height="48dip"
        android:layout_alignParentRight="true"
        android:layout_centerVertical="true"/>
</RelativeLayout>
```

视图和布局

自定义视图

视图和布局

自定义视图

```
class CustomView extends View {

    @Override
    protected void onDraw(Canvas canvas) {
        // 加入你的绘图编码
    }
    @Override
    protected void onMeasure(int widthMeasureSpec,
            int heightMeasureSpec) {
        // 计算视图的尺寸
        setMeasuredDimension(widthSpecSize, heightSpecSize);
    }
}
```

视图和布局

自定义布局

视图和布局

自定义布局

```
class GridLayout extends ViewGroup {

    @Override
    protected void onLayout(boolean changed, int l, int t,
            int r, int b) {
        final int count = getChildCount();
        for (int i=0; i < count; i++) {
            final View child = getChildAt(i);
            if (child.getVisibility() != GONE) {
                // 计算子视图的位置
                child.layout(left, top, right, bottom);
            }
        }
    }
}
```

技巧和设计模式

- 如何使用 Adapter
- 背景和图像
- 更新请求
- 视图和布局
- 内存分配

内存分配

不要创建 Java 对象

- 在性能敏感的代码里，尽量避免创建 Java 对象
  - 测量: onMeasure()
  - 布局: onLayout()
  - 绘图: dispatchDraw(), onDraw()
  - 事件处理: dispatchTouchEvent(), onTouchEvent()
  - Adapter: getView(), bindView()
- GC, 垃圾回收
  - 整个程序会暂停
  - 慢 (大约几百个毫秒)

内存分配

强行限制（适用于调试模式）

```
int prevLimit = -1;
try {
    prevLimit = Debug.setAllocationLimit(0);
    // 执行不分配内存的代码
} catch (dalvik.system.AllocationLimitError e) {
    // 如果代码分配内存，Java 虚拟机会抛出错误
    Log.e(LOGTAG, e);
} finally {
    Debug.setAllocationLimit(prevLimit);
}
```

内存分配

管理好对象

- 使用软引用
  - 内存缓存的最佳选择
- 使用弱引用
  - 避免内存泄露

内存分配

内存缓存实例

```
private final HashMap<String, SoftReference<T>> mCache;

public void put(String key, T value) {
    mCache.put(key, new SoftReference<T>(value));
}

public T get(String key, ValueBuilder builder) {
    T value = null;
    SoftReferece<T> reference = mCache.get(key);
    if (reference != null) {
        value = reference.get();
    }
    if (value == null) {
        value = builder.build(key);
        mCache.put(key, new SoftReference<T>(value));
    }
    return value;
}
```

总结

- 用 ViewHolder 实现 Adapter 的 getView
- 为背景选择正好的图像
- 针对无效区做更新请求
- 视图和布局，越简单越好
- 避免在性能敏感路径上创建对象

Learn more at http://code.google.com


Google
Developer
Day2009