从工程实践出发，深入剖析Android自定义控件、第三方组件、应用架构等内容，并以一个知识问答与分享应用案例展示Android应用开发实践。

Broadview®
www.broadview.com.cn

Android 应用开发进阶

范磊 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Android
应用开发进阶

范磊 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书为 Android 应用开发进阶的参考用书，旨在帮助 Android 开发者能够快速有效地掌握 Andriod 应用开发进阶相关的知识点。本书内容为 Android 应用开发热门的话题，包括自定义控件、第三方组件的使用和实现原理、Android 应用架构等。本书提供了一个知识问答与分享的实战案例，包括 Java 版本和 Kotlin 版本，方便读者完成从 Java 到 Kotlin 的平滑过渡，读者朋友可根据该案例搭建属于自己的应用架构。

希望本书能够帮助读者朋友在工作中解决实际问题，提升 Android 应用开发水平和能力。

本书适用于具有 Android 开发经验的读者或工程师。对于初级开发者，本书可作为进阶的参考用书，对于高级开发者，本书也具有一定的参考价值。

图书在版编目（CIP）数据

Android 应用开发进阶 / 范磊著. —北京：电子工业出版社，2018.5
ISBN 978-7-121-33958-5

Ⅰ. ①A… Ⅱ. ①范… Ⅲ. ①移动终端－应用程序－程序设计 Ⅳ. ①TN929.53

中国版本图书馆 CIP 数据核字（2018）第 065501 号

责任编辑：陈晓猛
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：29.25　　字数：559.2 千字
版　　次：2018 年 5 月第 1 版
印　　次：2018 年 5 月第 1 次印刷
定　　价：89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819，faq@phei.com.cn。

前言

记得刚从事 Android 开发工作时，市面上鲜有 Android 开发书籍可供参考，很多时候遇到问题只能自己去阅读源码，慢慢地探索。经过多年的发展和积累，国内涌现出了很多优秀的开发者和分享者，有很多参考价值很高的入门和进阶的书籍，这些书籍推动了国内 Android 开发者技术水平的提升，本书也希望能够为此贡献一点力量。

本书并不试图介绍 Android 开发的各个方面，因为有些专题甚至可以单独成书了，这里只涉及 Android 开发者经常会遇到的热门话题，如果能够给 Android 开发者的工作提供一些帮助或启发，就是本书存在的价值。

本书的主要内容

第 1 章介绍自定义控件的基本原理，包括 View 的绘制流程和触摸事件的处理；总结自定义控件中常用的一些方法；通过案例来展示如何完成一个自定义控件；最后介绍自定义控件中性能优化的问题。

第 2 章介绍在实际开发中常用轮子的使用方法及其实现原理。这些轮子包括 Butter Knife、EventBus、Dagger、OkHttp、Retrofit、Volley、RxJava。

第 3 章通过 Google 官方的架构示例项目 ToDo，详细介绍 Android 的应用架构，包括 MVP、MVP-Clean、MVP-Dagger、MVP-RxJava、MVVM-DataBinding 和 MVVM-Live。

第 4 章介绍实战项目 AndroidPlus，AndroidPlus 是一个专注于 Andriod 领域的知识问答与分享平台，遵循 Material Design，使用 MVP-Dagger 架构，提供 Java 版本和 Kotlin 版本的实现。

第 5 章介绍 Android 开发中的一些实践，包括 Android Studio 中的 Git 操作、Android 屏幕适配的解决方案、常用视频播放器的使用，以及声网直播的实践。

本书适合的对象

本书适用于具有 Android 开发经验的读者或工程师。对于初级开发者，本书可作为进阶的

参考用书，对于高级开发者，本书也具有一定的参考价值。

勘误与互动

因个人水平有限，书中难免存在错误或不准确之处，还望读者朋友批评指正。如果对本书有较好的建议或者对书中内容有所疑惑，可通过 QQ 或微信公众号与我联系，届时也会定期在微信公众号上公布勘误内容。最后，欢迎大家关注我的微信公众号、简书、掘金，获得更多的文章更新。

- QQ：526247082
- 微信公众号：Android 应用开发进阶
- 简书：https://www.jianshu.com/u/fe8153fbd5de
- 掘金：https://juejin.im/user/5674fb4360b2298f1221f103
- GitHub：https://github.com/uncleleonfan

致谢

感恩父母对我的培养和教育，感恩一起共事过的同事和朋友，感恩生活，感恩祖国。另外，特别感谢陈晓猛编辑耐心的指导、审稿和编辑，因而才有了本书最后的诞生。

范磊

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在 下载资源 处下载。
- **提交勘误**：您对书中内容的修改意见可在 提交勘误 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方 读者评论 处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/33958

目录

第1章 自定义控件

Android 应用与用户进行交互的界面一般由控件组成，Android SDK 和 Support 包提供了非常丰富的控件供开发者使用。但需求是千变万化的，官方提供的控件远远满足不了各种应用场景及特效，这时就需要开发者自己定制控件，即所谓的自定义控件。自定义控件类型有很多种，主要分为：

（1）继承已有成熟控件，扩展其功能。

（2）将已有的控件组合起来，放在一个布局中，整体当作一个模块，即组合式控件。

（3）完全自定义控件，自主完成控件的绘制布局及事件处理。

不管是哪种类型的自定义控件，都需要对 Android 中控件的实现原理有深入了解，才能应用自如。原理主要包括两个方面，一是控件如何在界面绘制出来，即 View 的绘制流程，二是这个控件如何与用户交互，即触摸事件的传递和处理。

1.1 View 的绘制流程

1.1.1 View 和 ViewGroup

View 是用户界面基本的构建块，它占据一个矩形的区域，职责是绘制和事件处理，View 是 Android 里面所有控件的基类，它的继承层级如图 1-1 所示。

View 直接继承自 Object，又派生出很多的子类，图中圈出的是一些常用的控件：ProgressBar、ImageView、ImageButton、TextView、Button、EditText。我们所说的自定义控件实质上就是继

承自 View 或 View 的子类，完成特定的绘制和交互功能。在 iOS SDK 中也有类似的类，叫作 UIView。

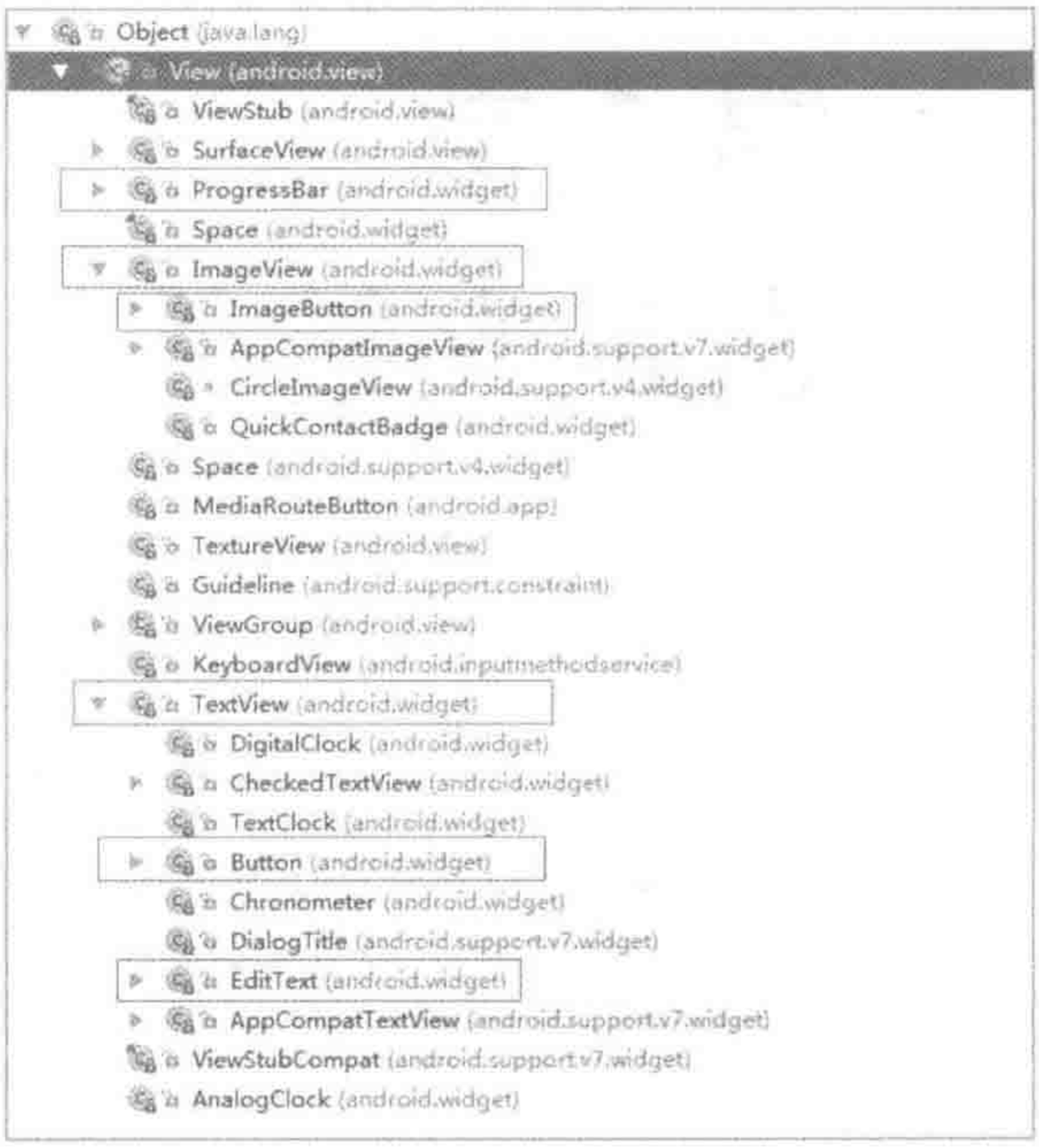

图 1-1　View 的继承层级

那 ViewGroup 又是什么呢？ViewGroup 继承自 View，所以它本质上是一个 View，但它是一个特殊的 View，特殊在它能够包含其他的 View，这些被 ViewGroup 包含的 View 我们称之为 ViewGroup 的“孩子”或子 View，ViewGroup 自己可以称之为父容器。ViewGroup 是 Android 中布局和容器的基类，继承关系如图 1-2 所示。

图中圈出的是一些基本的布局：LinearLayout、FrameLayout、AbsoluteLayout、RelativeLayout。另外，还有支持包中常用的组件，比如 Toolbar、ViewPager、ConstraintLayout、SwipeRefreshLayout 等，它们都继承自 ViewGroup。

一个 ViewGroup 还定义其内部“孩子”的布局参数（布局文件中带 layout 前缀的属性）。比如一个 TextView 在布局文件中配置属性 layout_centerInParent，这个属性只有在其父容器是 RelativeLayout 时才起作用，因为只有 RelativeLayout 才会处理这个属性，将这个 TextView 摆放在 RelativeLayout 的中心。

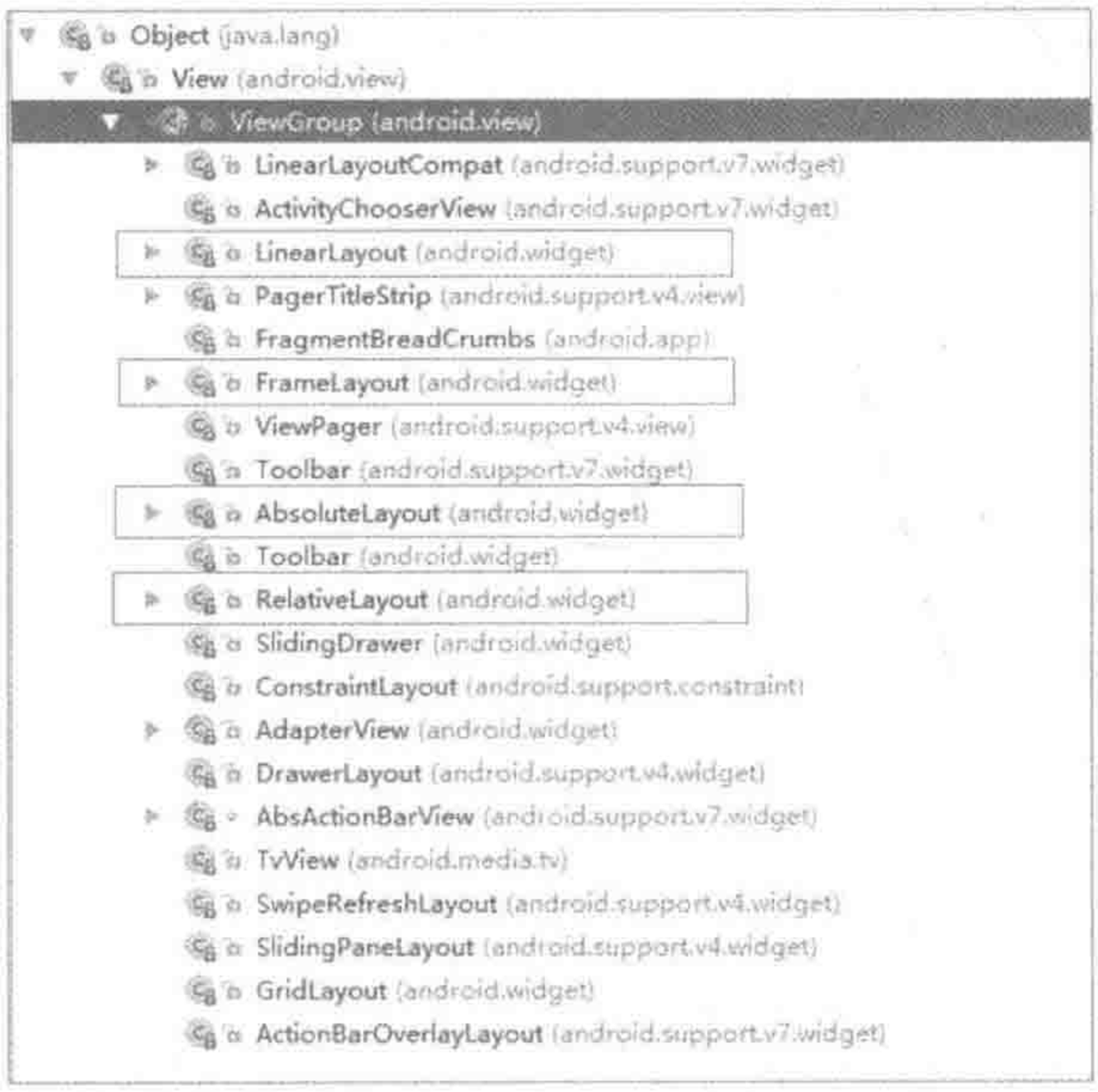

图 1-2 ViewGroup 的继承层级

由上可知，ViewGroup 和 View 存在两种关系：继承关系和嵌套关系。而 Android 用户界面实际上就是使用 ViewGroup 和 View 进行嵌套构建而成的。举个简单的例子：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
              android:layout_width="match_parent"
              android:layout_height="match_parent">
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="250dp"/>
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"/>
    </RelativeLayout>
</LinearLayout>
```

这个布局里面的 LinearLayout 和 RelativeLayout 是 ViewGroup，ImageView 和 TextView 是

普通的 View，不包含其他的“孩子”。LinearLayout 里嵌套了 ImageView 和 RelativeLayout，RelativeLayout 里又嵌套了 TextView。正是这种嵌套关系，构建了我们所看到的千变万化的用户界面。

1.1.2 View 的绘制流程

View 的绘制流程，即 View 在界面上绘制出来会经历哪些过程。我们平时写布局时只要写好布局文件就能完成界面展示，好像不需要什么绘制流程。真的是这么简单吗？我们从 Activity 里 onCreate 方法内调用的 setContentView 设置布局文件入手，调用 setContentView(int resID)要传入一个布局资源文件的 ID，那接下来会发生什么呢？setContentView 的代码如下：

```
@Override
public void setContentView(int resId) {
    ensureSubDecor();
    //找到 ID 为 content 的 ViewGroup，它是一个 Activity 默认就有的布局
    ViewGroup contentParent = (ViewGroup) mSubDecor.findViewById(android.R.id.content);
    //清空 contentParent 内部所有的子 View
    contentParent.removeAllViews();
    //解析 resId 对应的布局文件，将它嵌入到 contentParent 内部
    LayoutInflater.from(mContext).inflate(resId, contentParent);
    mOriginalWindowCallback.onContentChanged();
}
```

根据代码逻辑，resId 对应的布局会被解析出来，然后嵌入到一个 ID 为 content 的 ViewGroup 中，而这个 ViewGroup 是每个 Activity 默认就存在的，所以这也是函数名 setContentView 的由来。解析布局是调用了 LayoutInflater 的 inflate 的方法，大致的实现是通过递归方式，遍历布局中所有的控件，调用 createViewFromTag 方法创建控件对应的 View 对象。可以看出，setContentView 只是根据布局文件创建对应控件的 View 对象，并不涉及怎么去绘制一个 View。也就是说，布局资源 xml 文件只是一个配置文件，与 View 的绘制流程没有任何关系，主要是方便开发者快速高效地完成 UI 布局，并且 IDE 提供布局的预览，方便布局的调试。

那 View 的绘制流程到底是怎么一个过程呢？举个例子，假设一个书法家要在宣纸上写一个字，那在下笔之前心里会估算一下这个字要写多大，然后确定写在哪个位置，最后再下笔写出这个字。这跟我们在手机屏幕上绘制一个控件的过程是一致的，首先我们得知道控件的大小，然后确定这个控件放哪里，最后放好控件之后再去画这个控件。这三个过程对应代码的实现分别是 View 类中的三个 API：measure（测量）、layout（布局）、draw（绘制）。

1. 测量

我们找到测量方法 measure (int widthMeasureSpec, int heightMeasureSpec)，该方法的作用是找出控件的大小，即完成对控件大小的测量。真正的测量工作是由在 measure 方法里面调用的 onMeasure 方法完成的，在 View 类中 onMeasure 的默认实现里调用了 setMeasuredDimension 来设置测量的维度，即宽和高的大小。setMeasuredDimension 方法非常重要，它必须在 onMeasure 方法中被调用，否则会抛出 IllegalStateExeception 的异常。为什么这么重要呢？因为它内部完成了对成员变量 mMeasuredWidth 和 mMeasuredHeight 的赋值，未调用 setMeasuredDimension 方法则这两个值为默认为 0，即控件测量的宽高为 0，那这样没有大小的控件还有什么用呢?

我们写一个最小的自定义控件 MyView，它直接继承自 View，提供两个构造方法，第一个构造方法调用第二个构造方法。第一个构造方法通常在代码中创建对象时使用，第二个构造方法必须要有，因为在解析布局文件时正是调用带有两个参数的构造方法完成 View 对象的创建。MyView 的实现如下：

```
public class MyView extends View {
    public MyView(Context context) {
        this(context, null);
    }
    public MyView(Context context, @Nullable AttributeSet attrs) {
        super(context, attrs);
    }
}
```

在布局中配置 MyView 的宽高:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <com.example.view.MyView
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:background="@android:color/black"/>
</RelativeLayout>
```

在 MyView 类中对测量这一步没有做任何处理，最后运行的结果显示 MyView 的宽高为 200dp，即布局文件中配置的宽高。接下来，在 MyView 中覆写 onMeasure 方法并直接调用

serMeasuredDimension 指定测量的宽高都为 50px。

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    //super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    setMeasuredDimension(50, 50);
}
```

最后运行的结果显示 MyView 的宽高是多少呢？没错就是 50px。在布局文件中配置的 200dp 已经失效，setMeasuredDimension 最终决定了控件测量的宽高。但是如果直接使用 View 类中 onMeasure 的默认实现，即调用 super.onMeasure 方法，代码更改如下，那么结果将会是怎么样呢？

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    //setMeasuredDimension(50, 50);
}
```

最后运行显示 MyView 的宽高都为 200dp，布局文件中设置的宽高又起作用了，这是为什么呢？其实 super.onMeasure 方法内部也是调用了 setMeasuredDimension 来设置宽高的，而设置的宽高又是通过 widthMeasureSpec 和 heighMeasureSpec 计算而来的，很明显可以看出，widthMeasureSpec 和 heightMeasureSpec 同布局文件中配置的宽高是有联系的。

那么 widthMeasureSpec 和 heightMeasureSpec 是从哪里来的呢？这就要看哪里调用了 MyView 的 onMeasure 方法，事实上它们最早是从 MyView 的 measure 方法传过来的，然后传给 onMeasure 方法。它们是父容器（当前示例中就是 RelativeLayout）对 MyView 宽高的限制。

widthMeasureSpec 和 heightMeasureSpec 都是 int 类型，我们研究其中一个就可以了。widthMeasureSpec 是 int 类型，由 32 位二进制组成，前两位为父容器对“孩子”View 限制的模式，后 30 位表示大小值，模式在 MeasureSpec 类中定义为：

```
private static final int MODE_SHIFT = 30;
public static final int UNSPECIFIED = 0 << MODE_SHIFT;
public static final int EXACTLY     = 1 << MODE_SHIFT;
public static final int AT_MOST     = 2 << MODE_SHIFT;
```

- **UNSPECIFIED 模式：**父容器对“孩子”View 没有任何的大小限制，“孩子”想多大就多大；

- EXACTLY 模式：父容器对“孩子”View 有确切的大小要求，大小为后 30 位；
- AT_MOST 模式：父容器对“孩子”View 的最大值有要求，最大值为后 30 位。

我们在代码中拆分出模式和大小：

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    //setMeasuredDimension(50, 50);
    int mode = MeasureSpec.getMode(widthMeasureSpec);//拆出宽度的模式
    int size = MeasureSpec.getSize(widthMeasureSpec);//拆出宽度的大小
    Log.d(TAG, "onMeasure: " + (mode>>30) + " " + size);
}
```

打印出来结果为：D/MyView: onMeasure: 1 400。

由于模式都是向左位移了 30 位，这里为了看得更清楚一点，让它向右位移了 30 位，最后结果是 1，表示是 EXACTLY 模式，size 为 400px，这是由于运行模拟器的分辨率是 xhdpi，1dp=2px。整个结果说明 MyView 的父容器（RelativeLayout）对 MyView 的大小有确切的要求，大小为 400px，而 MyView 中 onMeasure 方法的默认实现就是按照父容器的要求来设置自己的大小。

整个测量过程就可以分析出来了：MyView 在布局文件中配置 layout_width 和 layout_height 分别为 200dp，其父容器 RelativeLayout 通过获取 MyView 的 LayoutParams 对象从而获取 layout_width 和 layout_height 值，然后根据实际情况构建出对 MyView 宽高的限制 widthMeasureSpec 和 heightMeasureSpec（注意 widthMeasureSpec 中后 30 位的大小不一定就是 layout_width 值，heightMeasureSpec 同理），接着 RelativeLayout 调用 MyView 的 measure 方法，传入 widthMeasureSpec 和 heightMeasureSpec 作为参数，进而传给了 onMeasure 方法。一般默认 onMeasure 方法会根据父容器的限制来设置自己的宽高。

2. 布局

View 类中 layout 方法为 layout(int l, int t, int r, int b)，l、t、r、b 分别表示相对于父容器上下左右的位置，如图 1-3 所示。

如果以父容器为坐标系，那么（l，t）可以看作 View 左上角的位置，（r，b）可以看作右下角的位置。只要这个四个位置确定之后，View 的位置就确定了。那么一个控件的 layout 方法是被谁调用的呢？由于四个参数都是相对于父容器的，由此假设得出 layout 的方法是被父容器调用的，父容器来指定子 View 相对于自己上下左右的位置，事实上也是如此。假如每个子 View 都能决定自己在哪个位置，那整个布局不就乱套了吗？