#9 Widget开发

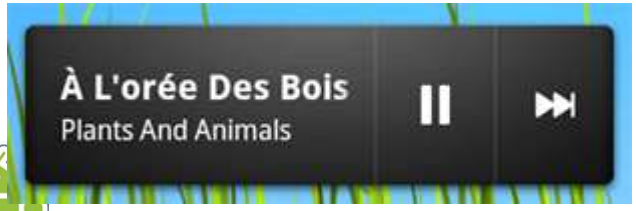
Widget

Category

1. Widget概述
2. Widget的开发
3. Widget的开发实例
4. Widget事件处理
5. 从配置活动更新Widget
6. Android Studio Debug
7. 使用 Logcat 写入和查看日志
8. Android开发中的常见异常

Widget的概述（1）

- Widget是微型应用程序视图，可以嵌入其他应用程序（如主屏幕）并接收定期更新。这些视图在用户界面中称为小部件。
- 标准的Android系统映像包含了一些示例widgets包括指针时钟、音乐播放器和其他工具如Google搜索栏
- Widget和标准的Apps相比没有太大的区别，更多的是在UI上的处理，逻辑执行设计成服务，具备更稳定和更高的可靠性



Widget的概述（2）

- 官方文档：
  https://developer.android.com/guide/topics/appwidgets/index.html

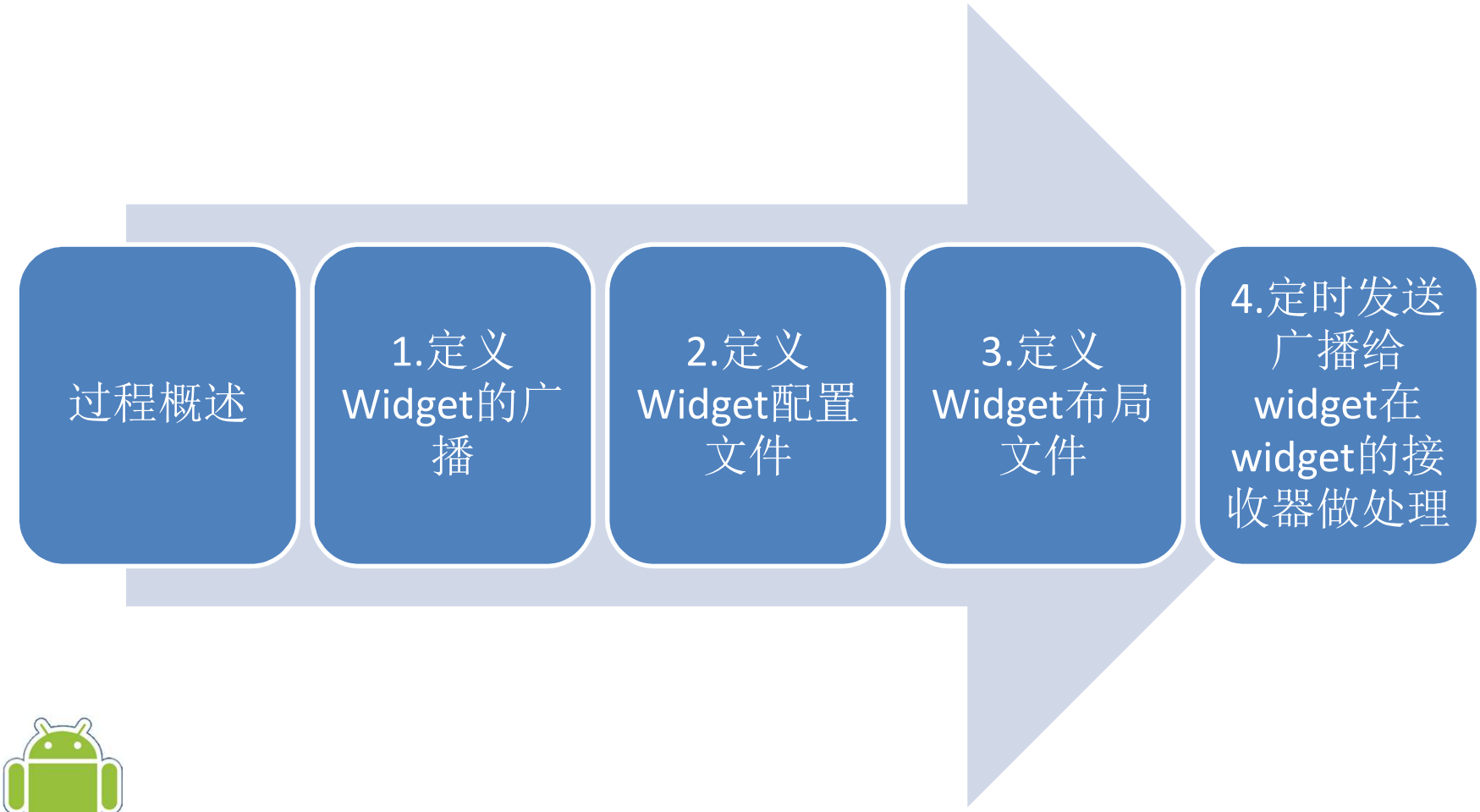
Widget的概述（3）

- 直接显示到桌面上的小控件，定期更新
- 每个Widget就是一个广播接收器
- 显示的内容封装成RemoteViews对象

Widget的概述（4）

- Widget 不是运行在自己进程里，而是宿主进程，所以交互需要处理AppWidget 广播。AppWidgetProvider 只接收和这个App Widget 相关的事件广播，比如这个App Widget 被更新，删除，启用，以及禁用。
- 每个Widget就是一个BroadcastReceiver，它们用XML metadata来描述Widget细节。AppWidget framework通过intent 和Widget通信，Widget更新使用RemotesViews来发送。RemotesViews被包装成一个layout和特定的内容来显示到桌面上。

Widget的开发（1）


过程概述
1.定义Widget的广播
2.定义Widget配置文件
3.定义Widget布局文件
4.定时发送广播给widget在widget的接收器做处理

Widget的开发（2）

- **AppWidgetProviderInfo object** 定义一个对象（**XML)**
  - Describes the metadata for an App Widget, such as the App Widget's layout, update frequency, and the AppWidgetProvider class. This should be defined in XML.

- **AppWidgetProvider class implementation** 实现一个类**(java)**
  - Defines the basic methods that allow you to programmatically interface with the App Widget, basedon broadcast events. Through it, you will receive broadcasts when the App Widget is updated,enabled, disabled and deleted.

Widget的开发（3）

- **View layout** 定义初始布局**(XML)**
  - Defines the initial layout for the App Widget, defined in XML.
  - Additionally, you can implement an App Widget configuration Activity. This is an optional Activity thatlaunches when the user adds your App Widget and allows him or her to modify App Widget settingsat create-time.

Widget的开发（4）

- **AppWidgetProvider** ONE Class

  继承自**BroadcastRecevier**，这些广播事件发生时,**AppWidgetProvider**将通过自己的方法来处理，这些方法包括：**update**、**enable**、**disable**和**delete**时接收通知。其中，**onUpdate**、**onReceive**是最常用到的方法。

➡ **onReceive**

接收到每个广播时都会被调用，而且在上面的回调函数之前。

➡ **onUpdate**

间隔性更新App Widget，间隔时间在AppWidgetProviderInfo里的updatePeriodMillis属性定义。该方法也会在添加App Widget时被调用，进行widget配置。

Widget的开发（5）

- **onDisabled**

  当App Widget的最后一个实例被从宿主中删除时被调用。例如在onDisabled中做一些清理工作，比如关掉后台服务。

- **onDeleted**

  当App Widget从宿主中删除时被调用。

- **onEnabled**

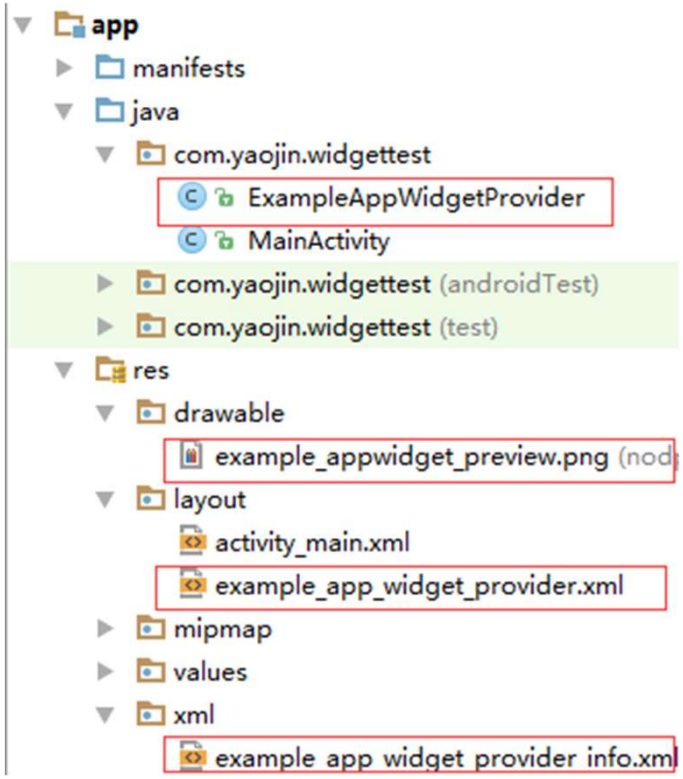
  当Widget实例第一次创建时被调用。若用户添加两个同一个App Widget实例，只在第一次被调用。适用于需要打开一个新的数据库或者执行其他对于所有的App Widget实例只需要发生一次的处理

- **onAppWidgetOptionsChange**

  第一次放置小部件和调整小部件的大小被调用这个。您可以使用此回调来根据窗口小部件的大小范围显示或隐藏内容

Widget的开发实例（1）

- 典型的Android Widget有三个主要组件，一个边框、一个框架和图形控件以及其他元素。在Android Studio中创建Widget类后，会直接生成相关文件。创建之后文件如图：



Widget的开发实例（2）

1、Widget内容提供者文件example_app_widget_provider_info.xml，初始代码如下，可以根据情况进行对应修改

```
<?xml version="1.0" encoding="utf-8"?>
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialKeyguardLayout="@layout/example_app_widget_provider"
    android:initialLayout="@layout/example_app_widget_provider"
    android:minHeight="160dp"
    android:minWidth="160dp"
    android:previewImage="@drawable/example_appwidget_preview"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="86400000"
    android:widgetCategory="home_screen"
    android:configure="com.yaojin.widgettest.MainActivity">
</appwidget-provider>
```

Widget的开发实例（3）

- <appwidget-provider> 属性介绍:

① initialKeyguardLayout属性指向定义App Widget显示在键盘锁界面的布局资源。

② initialLayout属性指向定义App Widget显示在主屏幕的布局资源。

③ previewImage属性指定在配置应用程序小部件后，用户在选择应用程序小部件时看到的内容。如果没有提供，用户会看到您的应用程序的启动器图标。

④ updatePeriodMillis属性定义自动更新的时间间隔。
注意：updatePeriodMillis请求的更新每30分钟内最多一次。

⑤ configure属性定义当用户添加App Widget时要启动的Activity，以便配置App Widget属性。

Widget的开发实例（4）

⑥ widgetCategory属性确定此窗口小部件是否可以显示在主屏幕，键盘保护或两者上。

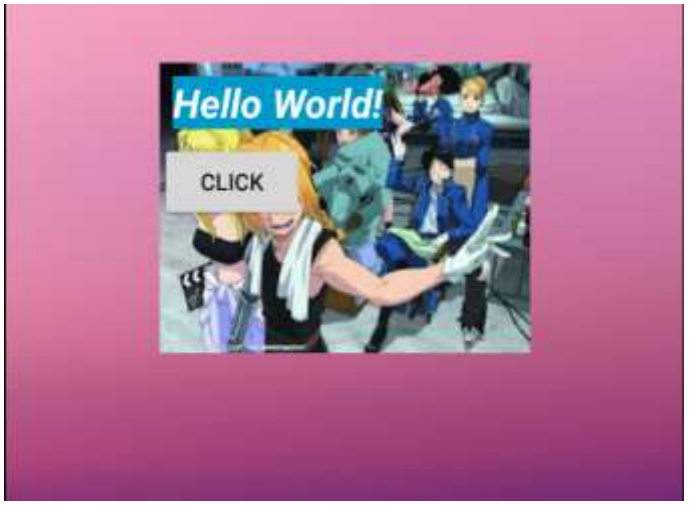
⑦ minWidth和minHeight属性的值指定App Widget默认消耗的最小空间量。如果应用程序小部件的最小宽度或高度的值与屏幕单元格的尺寸不匹配，则应用程序小部件的尺寸将舍入到最近的单元格大小。

⑧ minResizeWidth和minResizeHeight属性指定App Widget的绝对最小大小,允许用户将窗口小部件的大小调整为可能小于由minWidth和minHeight属性定义的默认窗口大小的大小 。

Widget的开发实例（5）

2、布局文件example_app_widget_provider.xml，默认情况下仅有1个textView，本实例中添加了一个button按钮并添加了背景，效果如图:

```
<TextView
    android:id="@+id/appwidget_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:background="#09C"
    android:contentDescription="@string/appwidget_text"
    android:text="@string/appwidget_text"
    android:textColor="#ffffff"
    android:textSize="24sp"
    android:textStyle="bold|italic" />
<Button
    android:id="@+id/appwidget_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/Click"
    android:layout_below="@id/appwidget_text"/>
```



Widget的开发实例（6）

3、添加到桌面

问题：如何处理*Widget*事件？

Working with RemoteViews

- 主屏幕实际上是一个系统程序，因为安全原因，程序员不容易直接修改其运行代码
- Android提供给用户程序访问主屏幕和修改特定区域内容的方法： RemoteViews架构
- RemoteViews架构允许用户程序更新主屏幕的View

➡ 点击Widget激活点击事件

➡ Android会将其转发给用户程序，由AppWidgetProviders类处理

➡ 用户程序可更新主屏幕Widget.

Widget事件处理(1)

- 配置文件添加:

```
<receiver android:name=".ExampleAppWidgetProvider">
    <intent-filter>
        <action android:name="com.yaojin.widgettest.CLICK"/>
        <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
    </intent-filter>

    <meta-data
        android:name="android.appwidget.provider"
        android:resource="@xml/example_app_widget_provider_info" />
</receiver>
```

Widget事件处理(2)

appWidgetManager

RemoteViews ComponentName

PendingIntent

Intent

- 在ExampleAppWidgetProvider.class的onUpdate添加代码，定义并发送事件

```
public void onUpdate(Context context, AppWidgetManager appWidgetManager, int[] appWidgetIds) {
    RemoteViews updateViews=new RemoteViews(context.getPackageName(),
       R.layout.example_app_widget_provider);//实例化RemoteView,其对应相应的Widget布局
    Intent i=new Intent("com.yaojin.widgettest.CLICK");
    PendingIntent pi=PendingIntent.getBroadcast(context,0,i,PendingIntent.FLAG_UPDATE_CURRENT);
    //PendingIntent pi=PendingIntent.getActivity(context,0,i,0);
    updateViews.setOnClickPendingIntent(R.id.appwidget_button,pi);//RemoteView上的Button设置按钮事件
    ComponentName me=new ComponentName(context,ExampleAppWidgetProvider.class);
    appWidgetManager.updateAppWidget(me,updateViews);
}
```

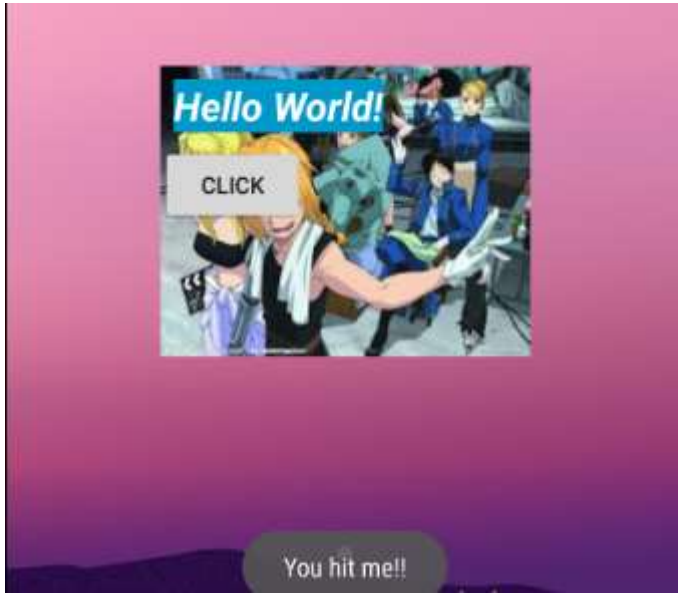
- 添加onReceive()处理事件

```
public void onReceive(Context context,Intent intent)
{
    super.onReceive(context,intent);
    if(intent.getAction().equals("com.yaojin.widgettest.CLICK"))
    {
        Toast.makeText(context,"You hit me!!",Toast.LENGTH_LONG).show();
    }
}
```

Widget事件处理(3)

- 最终效果。

  点击CLICK按钮后Toast显示：



Configuration Activity(1)

- 从配置活动更新应用程序小部件:

1. 当应用程序小部件使用配置活动时，配置完成后，更新应用程序小部件是活动的责任。您可以直接从AppWidgetManager请求更新。

2. configure属性定义当用户添加App Widget时要启动的Activity，以便他或她配置App Widget属性。

Configuration Activity(2)

1、<appwidget-provider>添加configure属性

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
    android:initialKeyguardLayout="@layout/example_app_widget_provider"
    android:initialLayout="@layout/example_app_widget_provider"
    android:minHeight="160dp"
    android:minWidth="160dp"
    android:previewImage="@drawable/example_appwidget_preview"
    android:resizeMode="horizontal|vertical"
    android:updatePeriodMillis="86400000"
    android:widgetCategory="home_screen"
    android:configure="com.yaojin.widgettest.MainActivity">

</appwidget-provider>
```

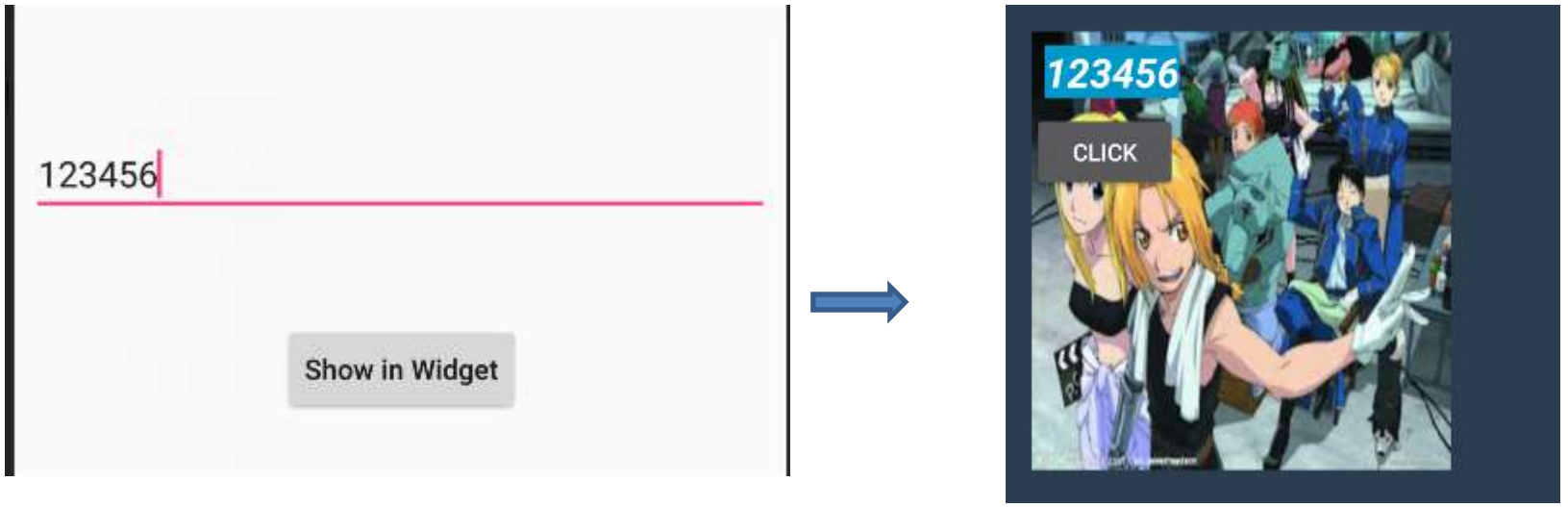
Configuration Activity(3)

2、在configuration activity对widget进行配置:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    mAppWidgetId = extras.getInt(
            AppWidgetManager.EXTRA_APPWIDGET_ID,
            AppWidgetManager.INVALID_APPWIDGET_ID);
}//获取APP Widget ID
AppWidgetManager appWidgetManager =
AppWidgetManager.getInstance(MainActivity.this);//获取AppWidgetManager实例
RemoteViews views = new RemoteViews(getPackageName(),
        R.layout.example_app_widget_provider);
views.setTextViewText(R.id.appwidget_text, editText.getText());//设置显示内容
appWidgetManager.updateAppWidget(mAppWidgetId, views);//更新App Widget
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);//设置完成
finish();
```

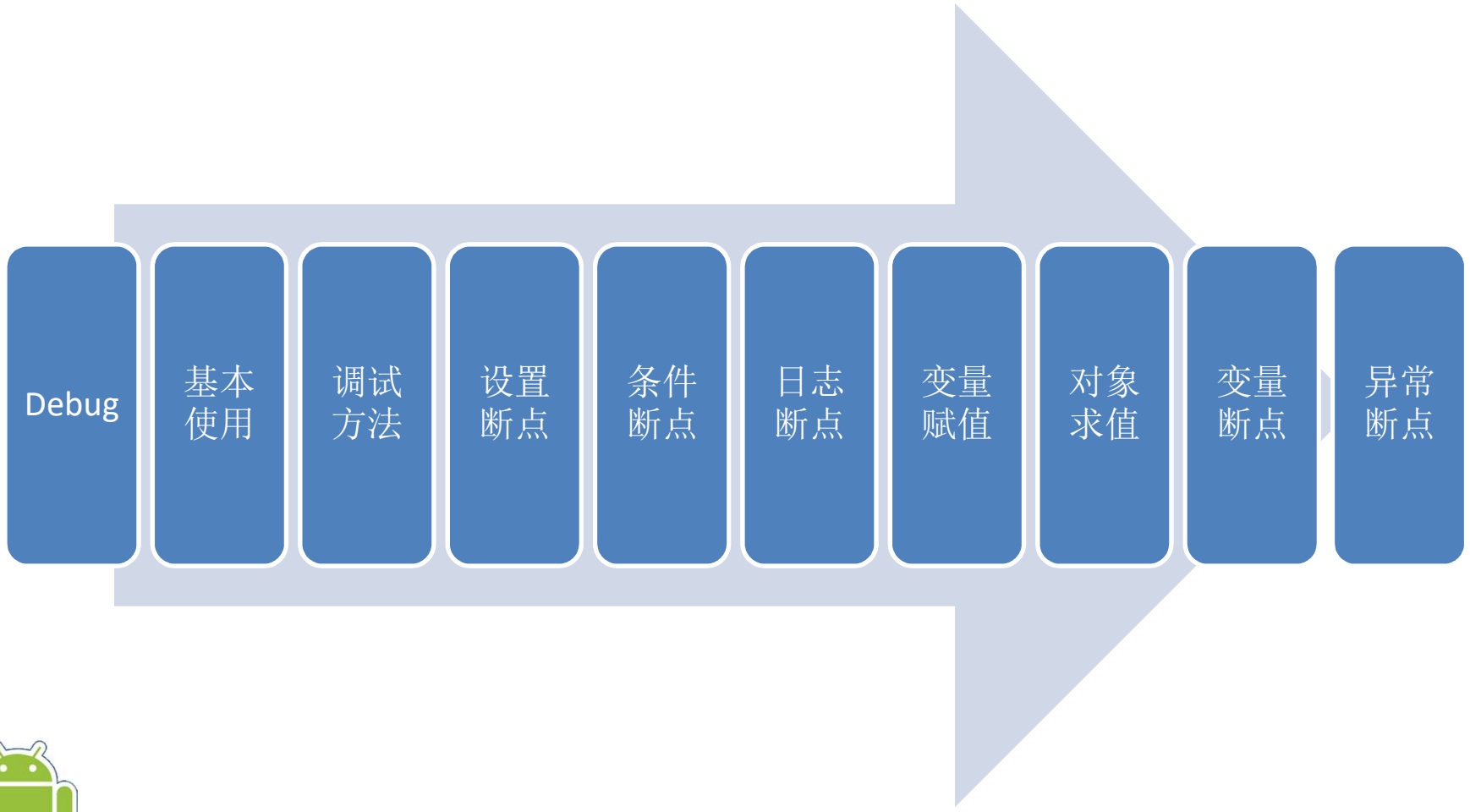
Configuration Activity(4)

3. 最终效果:



Note：如果没有在Configuration Activity完成对Widget的配置，Widget将不会显示在屏幕上。

Android Studio Debug

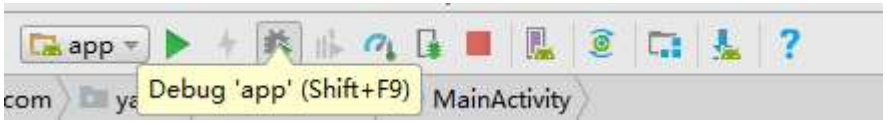


Android Studio Debug(1)

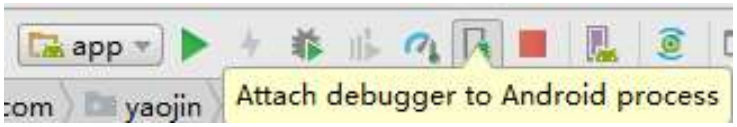
- 基本使用

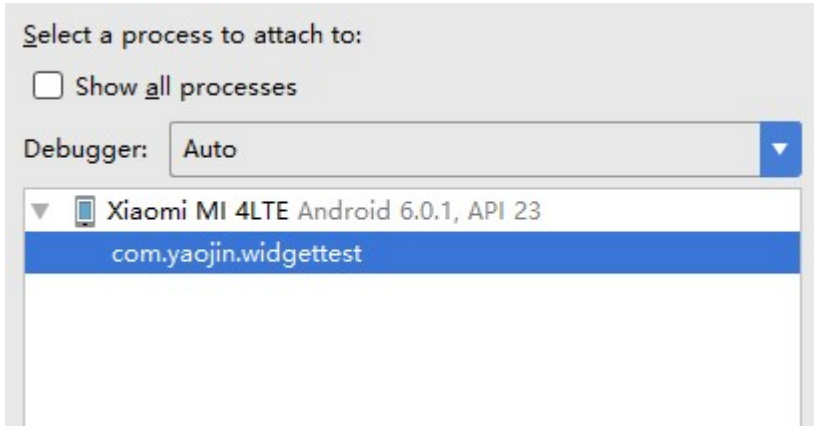
  Debug App有两种途径

  1. 直接点击运行按钮右侧的小虫状图标



  2. 调试当前已经处于运行状态下的App，即Attach debugger to Android process 。





Android Studio Debug(2)

- 调试方法介绍:

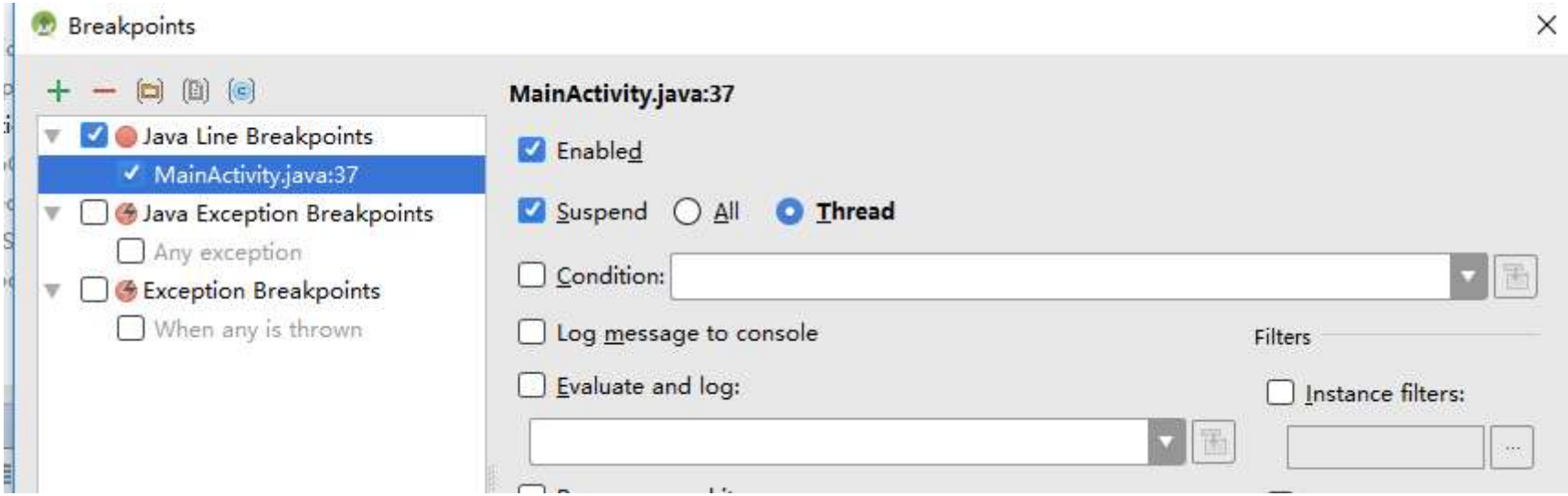
step into:单步执行，遇到子函数就进入并且继续单步执行.
step over:在单步执行时，在函数内遇到子函数时不会进入子函数内单步执行，把子函数整个作为一步。
step out:单步执行到子函数内时，执行完子函数余下部分，并返回到上一层函 数。

Run to cursor:执行到光标处。

Android Studio Debug(3)

- 设置断点：单击目标代码行的行号右侧空白处。

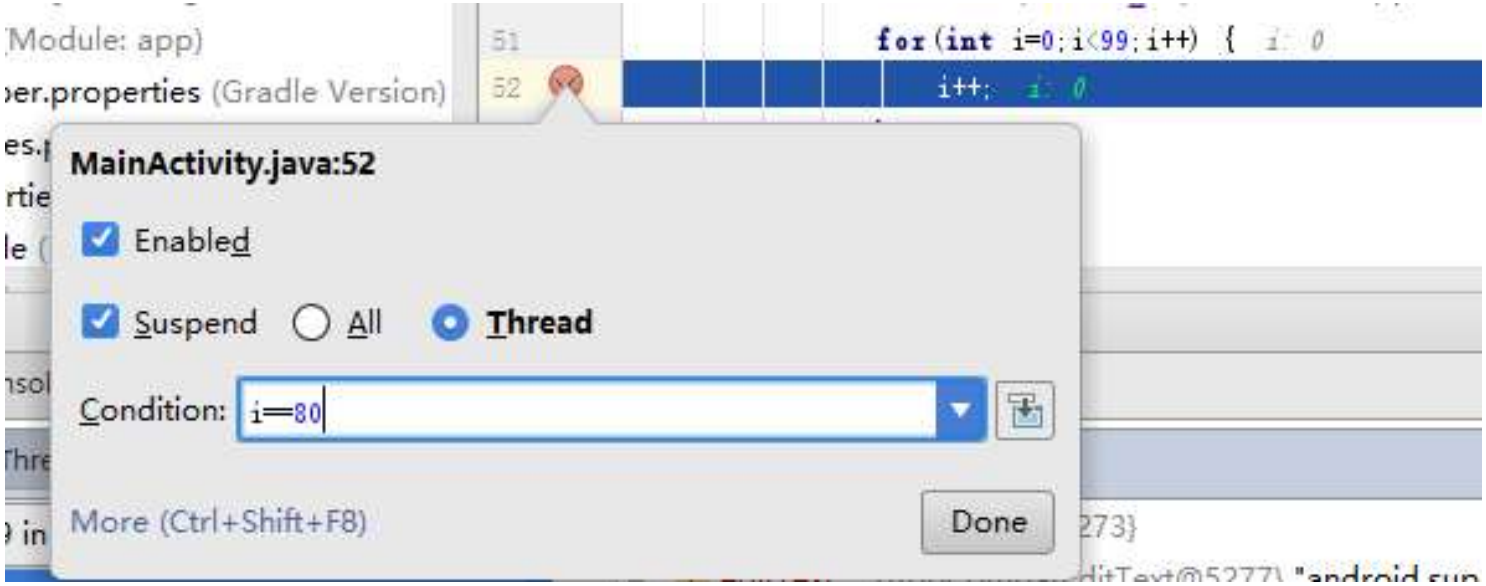
  Debug窗口左侧有个断点浏览按钮View Breakpoints，可以浏览Project中的所有断点，同时可以添加删除断点：



Android Studio Debug(4)-条件断点(2)

- 条件断点

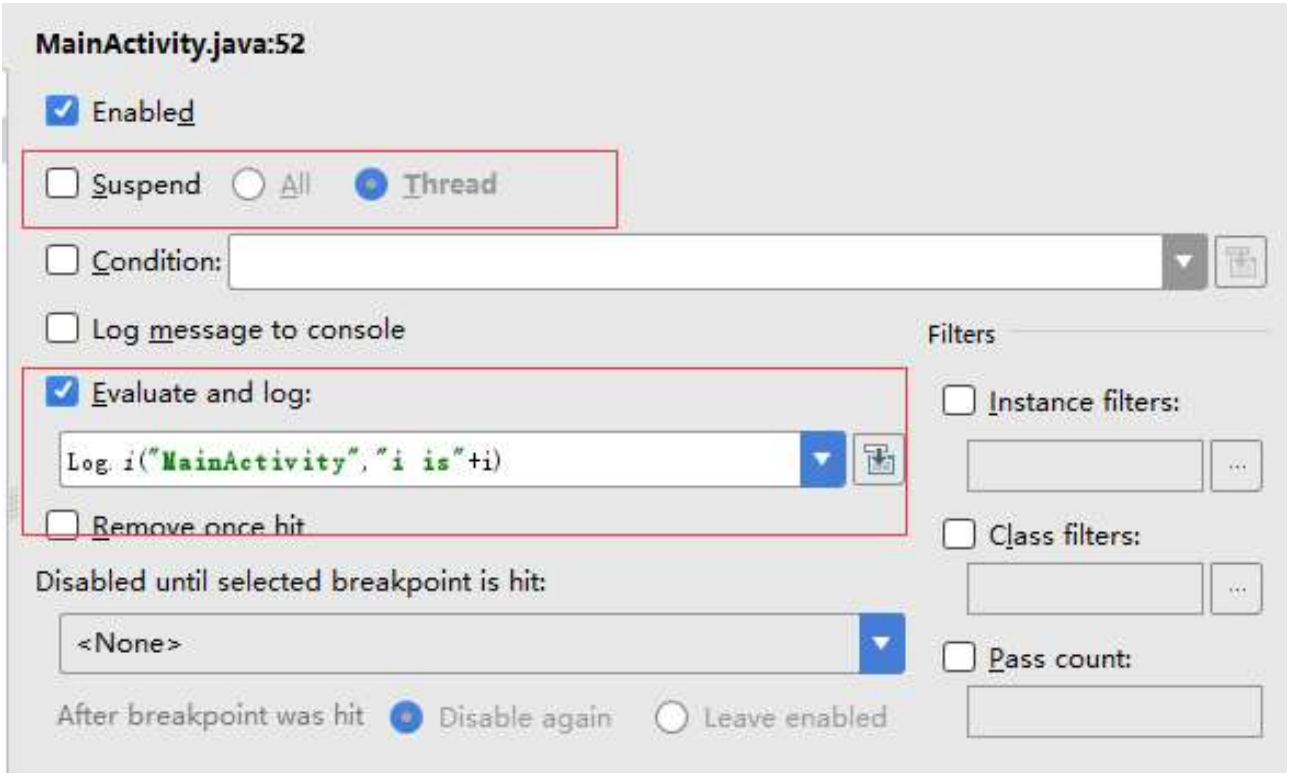
  当断点在循环体内时，右键点击断点，在弹出的窗口中输入Condiction条件，点击Done按钮.程序执行到循环体时，会在满足条件的一次循环中停下来，供我们调试：



Android Studio Debug(5)

- 日志断点

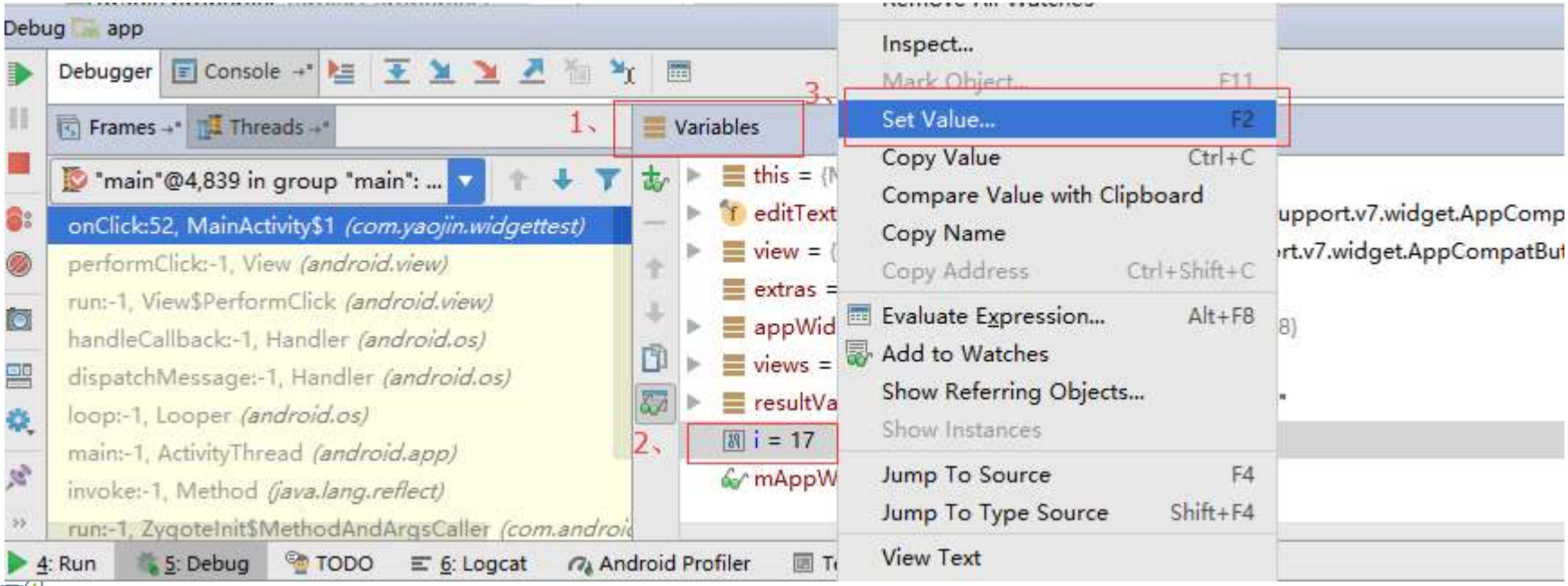
  可在断点处输出日志。右键点击断点，在弹出的窗口中取消勾选Suspend复选框，然后勾选Evaluate and log:，并输入打印语句：



Android Studio Debug(6)

- 变量赋值

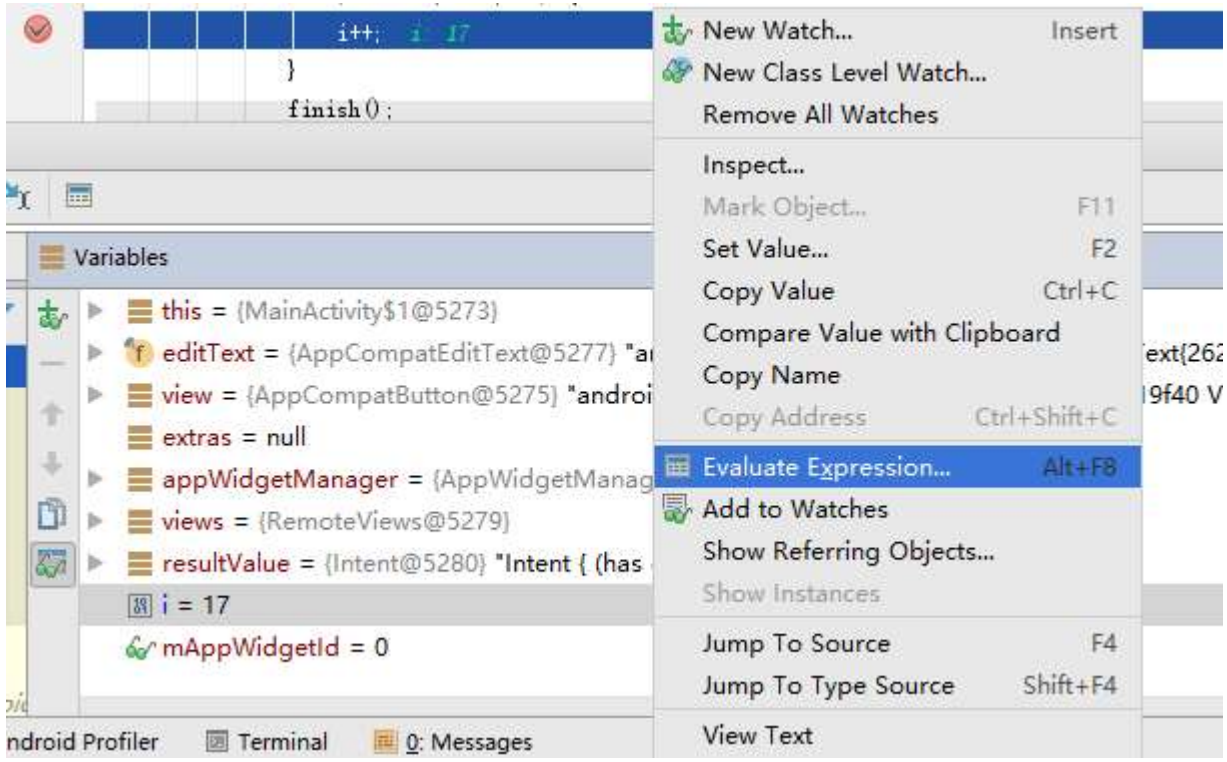
  可以在Variables窗口找到对应的变量，右键点击，修改变量值。



Android Studio Debug(7)

- 对象求值

  针对Variables视图中的变量对象，我们可以输入任何计算语句，实时查看表达式计算结果。具体操作为，右键Variables视图中的变量对象，选择Evaluate Expression：



Android Studio Debug(8)

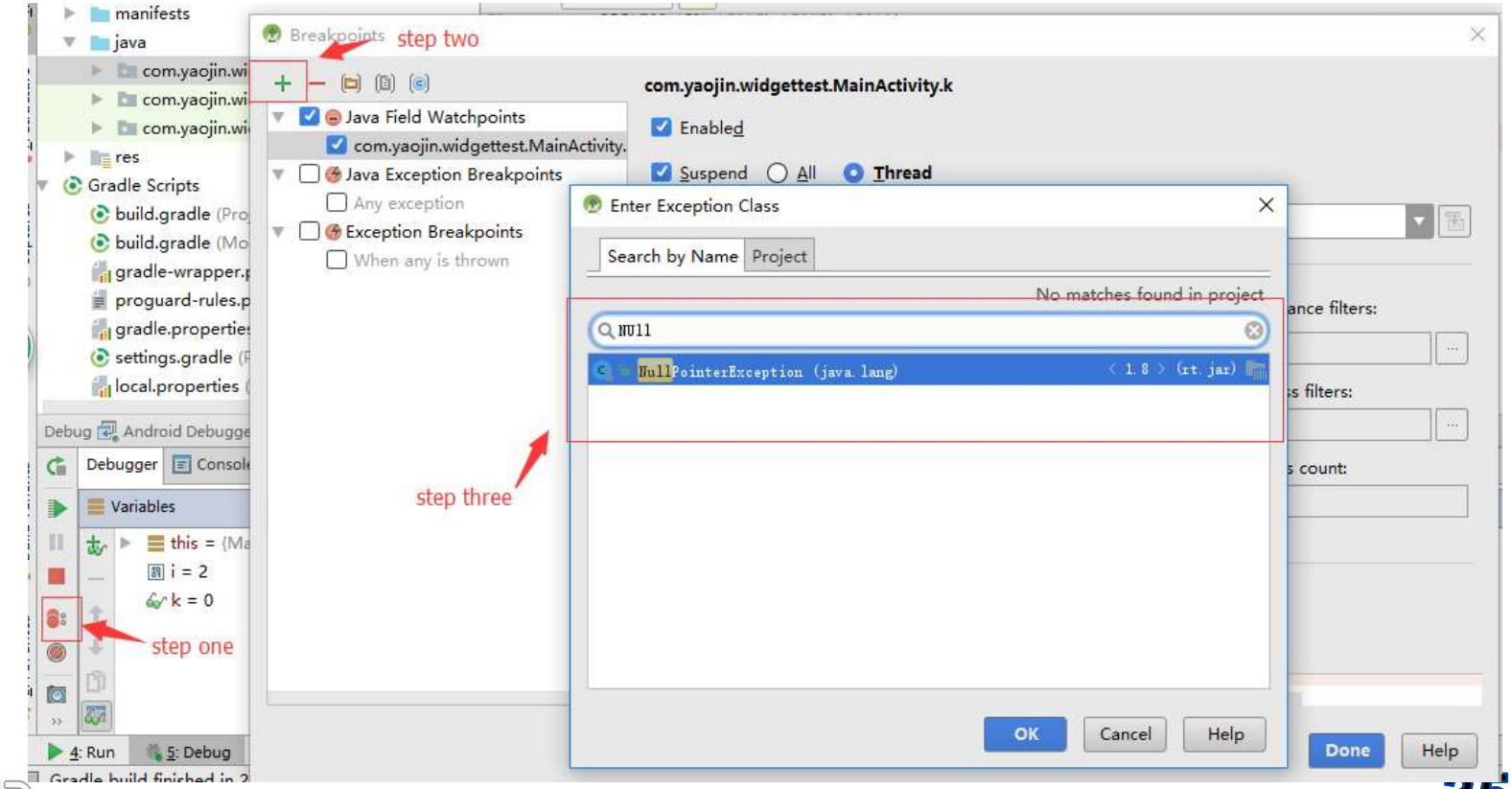
- 变量断点

在程序执行的过程中，如果该变量的值发生改变，程序会自动停下来，并定位在改变变量值的地方，供开发者调试：

Android Studio Debug(9)

- 异常断点

  能在发生异常的时候第一时间让程序停下来，并定位到异常出现的地方。



使用 Logcat 写入和查看日志

Android Monitor 包含一个可以显示调试消息的 logcat 监视器

logcat 监视器有如下功能:

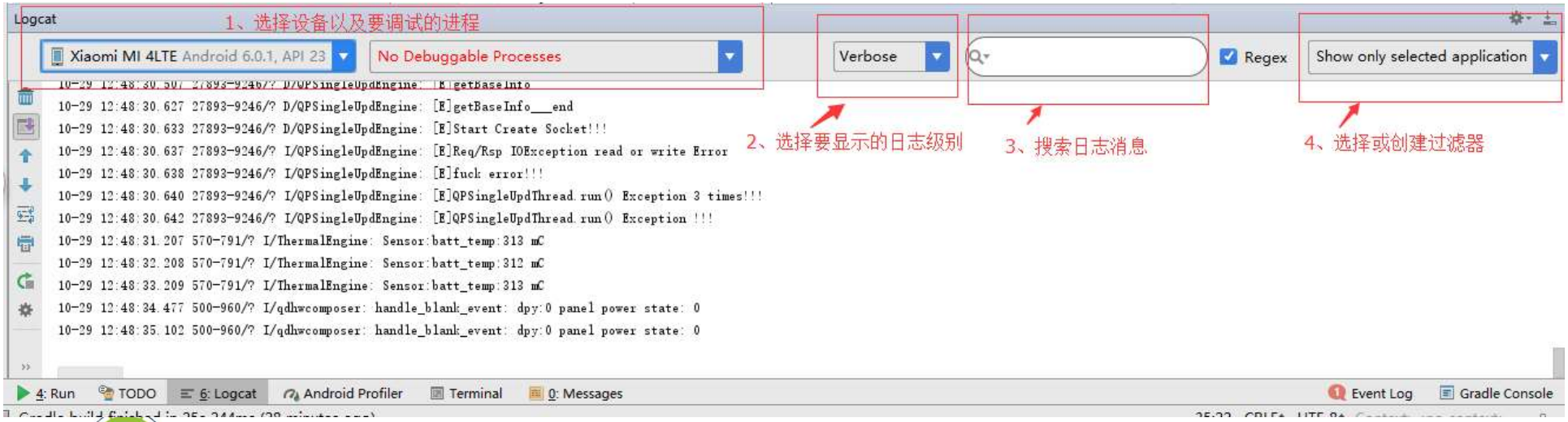
1. 可以显示系统消息（例如何时发生了垃圾回收）。
2. 能显示您使用 Log 类添加到应用中的消息。
3. 此监视器可以实时显示消息，也可以保留历史记录，以便您查看较旧的消息。

使用 Logcat 写入日志

- 您可以通过 Log 类创建在 logcat 监视器中显示的日志消息。一般来说，您应使用以下日志方法，这些方法按照优先级从高到低（或者从最简略到最详细）的顺序列示：
  - `Log.e(String, String)`（错误）
  - `Log.w(String, String)`（警告）
  - `Log.i(String, String)`（信息）
  - `Log.d(String, String)`（调试）
  - `Log.v(String, String)`（详细）
- 第一个参数都应是唯一标记，第二个参数是消息。
  - 系统日志消息的标记是一个简短的字符串，其表示消息所源自的系统组件（例如，ActivityManager）。
  - 您的标记可以是您认为有用的任意字符串，例如当前类的名称。

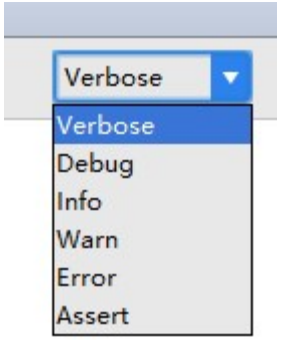
使用 Logcat 查看日志(1)-配置

1. 要仅显示感兴趣的信息，您可以创建过滤器，修改消息中显示的信息量，设置优先级，仅显示通过应用代码生成的消息，以及搜索日志。默认情况下，logcat 监视器仅显示与最近运行的应用相关的日志输出。



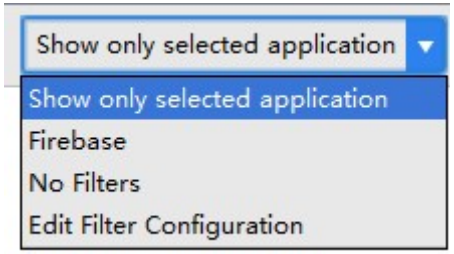
使用 Logcat 查看日志(2)-设置日志级别

- 在 **Log level** 菜单中，选择以下值之一：
  - **Verbose** - 显示所有日志消息（默认值）。
  - **Debug** - 显示仅在开发期间有用的调试日志消息，以及此列表中较低的消息级别。
  - **Info** - 显示常规使用的预计日志消息，以及此列表中较低的消息级别。
  - **Warn** - 显示尚不是错误的潜在问题，以及此列表中较低的消息级别。
  - **Error** - 显示已经引发错误的问题，以及此列表中较低的消息级别。
  - **Assert** - 显示开发者预计绝不会发生的问题。



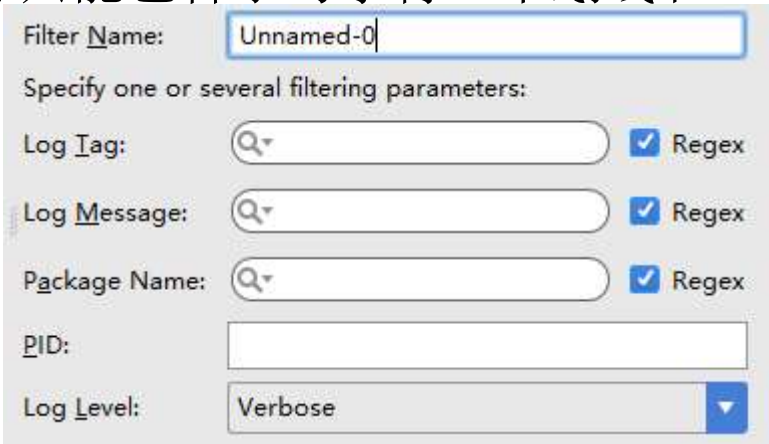
使用 Logcat 查看日志(3)-过滤logcat消息

- 在过滤器菜单中，选择一个过滤选项：
  - **Show only selected application** - 仅显示通过应用代码生成的消息（默认选项）。logcat 监视器将使用活动应用的 PID 过滤日志消息。
  - **No Filters** - 不应用过滤器。无论您选择哪个进程，logcat 监视器都会显示设备中的所有日志消息。
  - **Edit Filter Configuration** - 创建或修改自定义过滤器。例如，您可以创建一个过滤器，以同时查看两个应用中的日志消息。



使用 Logcat 查看日志(4)-创建自定义过滤器

- 在 *Create New Logcat Filter* 对话框中指定过滤器参数：
  - **Filter Name** - 键入您想要定义的过滤器的名称，或者从左侧面板中选择一个以修改现有过滤器。名称只能包含小写字符、下划线和数字。
  - **Log Tag** -指定标记。
  - **Log Message** -指定日志消息文本。
  - **Package Name** -指定软件包名称。
  - **PID** -指定进程 ID。
  - **Log Level** -选择日志级别。
  - **Regex** - 选择此选项可以为相关参数使用正则表达式。



- 以上参数除**Filter Name**皆为可选。

使用 Logcat 查看日志(5)-堆栈跟踪

- 如果应用引发异常，logcat 监视器会显示一条消息，接着显示关联的堆栈跟踪，堆栈跟踪包含指向该代码的链接。此功能可以帮助您修复错误和改进应用操作。
  - 点击“Up the Stack Trace”，移动到日志中相对于当前位置的前一个方法。
  - 点击“Down the Stack Trace”， 可以移动到日志中相对于当前位置的下一个方法。

Android开发中的常见异常(1)

a. **空指针异常** java.lang.NullPointerException

有对象的声明， 却没有给对象在堆中开辟内存，这时当调用对象的属性或方法就会抛出该异常。

b. **强制转换异常**java.lang.ClassCastException

需要转换的对象与向上或者向下转换的实例没有关系的时候，强制转换会报此异常。

c. **下角标越界异常** java.lang.ArrayIndexOutOfBoundsException

d. **转换类异常** java.lang.ClassNotFoundException

转换的目的类不存在就会导致此异常产生

e. **内存不足** java.lang.OutOfMemoryError

当运行一个程序的时候，手机的可用内存不足以让java虚拟机给一个对象内存空间时

Android开发中的常见异常（2）

f. **栈溢出错误** java.lang.StackOverflowError

应用的递归调用的层次太深而导致堆栈溢出时抛出该错误，会直接导致程序崩溃。

g. **算数异常** java.lang.ArithmeticException

调用除法运算法则，除数为零的话，就会报此异常

h. **非法访问异常** java.lang.IllegalAccessException

- 应用通过反射方式创建某个类的实例、访问该类属性、调用该类方法，但是又没有访问类的属性、方法的构造方法定义时。
- 例如，你定义的某个包的类的权限为protected，当其他的包下的类需要访问时，又因为没有这个权限，就会抛出此异常。

Android开发中的常见异常（3）

i. **没有这个方法** java.lang.NoSuchMethodException

调用某个类不存在的方法时，抛出此异常

j. **没有这个属性** java.lang.NoSuchFieldException

调用某个类不存在的属性时，抛出此异常

k. **数字格式异常** java.lang.NumberFormatException

被转换的数据不能满足指定数据类型出现此异常，如(int 转换为字符串,boolean转换为字符串)

Questions?