Android江湖上一直流传着一部秘籍——Android无障碍宝典。传闻练成这部宝典，可在Android无障碍模式下，飞檐走壁，能人所不能。宝典分为三篇，分别是入门、进阶和高级，由浅入深，全面展示无障碍的基本方法及扩展应用。

Android应用无障碍化，目的是为视觉障碍或其他有障碍的用户提供更好的服务。在无障碍模式下，用户的操作方式与平常不同，比如：

选择（Hover）一个元素：单击   
点击（Click）一个元素：双击   
滚动：双指往上、下、左、右   
选择上或下一个项目：单指往上、下、左、右   
快速回到主画面：单指上滑+左滑   
返回键：单指下滑+左滑   
最近画面键：单指左滑+上滑   
通知栏：单指右滑+下滑

此外，还需要理解在无障碍模式下“无障碍焦点”这个概念。如图1所示，界面上以绿色方框来表示目前获得无障碍焦点的View。拥有无障碍焦点的View，会被TalkBack服务识别，TalkBack会从View中取出相关的无障碍内容，然后提示给用户。

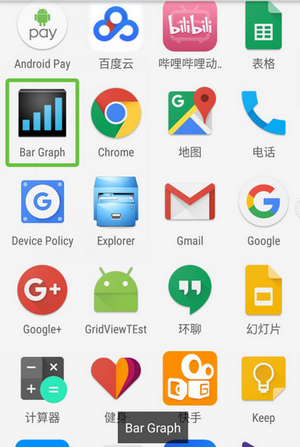


图1 无障碍焦点

有了对无障碍模式初步的了解，就可以正式开始学习如何为应用无障碍化。

入门篇

**为View添加ContentDescription**

UI上的可操作元素都应该添加上ContentDescription, 当此元素获得无障碍焦点时，TalkBack服务就取出View的提示语（contentDescription），并朗读出来。

添加ContentDescription有两种方法，第一种是通过在XML布局中设置android:contentDescripton属性，如：

<button android:id="”@+id/pause\_button”" android:src="”@drawable/pause”" android:contentdescription="”@string/pause”/"></button>

但是很多情况下，View的内容描述会根据不同情景需要而改变，比如CheckBox按钮是否被选中，以及ListView中item的内容描述等。这种则需要在代码中使用setContentDescription方法，如：

String contentDescription = "已选中 " + strValues[position];

label.setContentDescription(contentDescription);

**设置无障碍焦点**

UI上的元素，有的默认带有无障碍焦点，如Button、CheckBox等标准控件，有的如果不设置contentDescription是默认没有无障碍焦点。在开发应用过程中，还会遇到一些UI元素，是不希望它获取无障碍焦点的。以下方法可以改变元素的无障碍焦点：

public void setAccessibilityFocusable(View view, boolean focused){

if(android.os.Build.VERSION.SDK\_INT >= 16){

if(focused){

ViewCompat.setImportantForAccessibility(view, ViewCompat.IMPORTANT\_FOR\_ACCESSIBILITY\_YES);

}else{

ViewCompat.setImportantForAccessibility(view, ViewCompat.IMPORTANT\_FOR\_ACCESSIBILITY\_NO);

}

}

}

IMPORTANT\_FOR\_ACCESSIBILITY\_YES表示这个元素应该有无障碍焦点，会被TalkBack服务读出描述内容；IMPORTANT\_FOR\_ACCESSIBILITY\_NO表示屏蔽元素的无障碍焦点，手指滑动遍历及触摸此元素，都不会获得无障碍焦点，TalkBack服务也不会读出其描述内容。

**发出无障碍事件**

IMPORTANT\_FOR\_ACCESSIBILITY\_YES表示这个元素应该有无障碍焦点，会被TalkBack服务读出描述内容；IMPORTANT\_FOR\_ACCESSIBILITY\_NO表示屏蔽元素的无障碍焦点，手指滑动遍历及触摸此元素，都不会获得无障碍焦点，TalkBack服务也不会读出其描述内容。

view.postDelayed(new Runnable() {

@Override

public void run() {

if(android.os.Build.VERSION.SDK\_INT >= 14){

view.sendAccessibilityEvent(AccessibilityEvent.TYPE\_VIEW\_HOVER\_ENTER);

}

}

},100);

这个方法是让View来自动发出被单击选中的无障碍事件，发出后，UI上的无障碍焦点则会马上赋给这个View，从而达到抢无障碍焦点的效果。再比如：

if(android.os.Build.VERSION.SDK\_INT >= 16){

AccessibilityEvent event = AccessibilityEvent.obtain(AccessibilityEvent.TYPE\_ANNOUNCEMENT);

event.setPackageName(view.getContext().getPackageName());

event.setClassName(view.getClass().getName());

event.setSource(view);

event.getText().add(desc);

view.getParent().requestSendAccessibilityEvent(view, event);

}

AccessibilityEvent.TYPE\_ANNOUNCEMENT是代表元素需要TalkBack服务来读出描述内容。其中desc是描述内容，将它放到event的getText()中，然后请求View的父类来发出事件。

进阶篇

**介绍AccessibilityDelegate**

Android中View含有AccessibilityDelegate这个子类，它可被注册进View中，主要作用是为了增强对无障碍化的支持。

查看View的源码可发现，注册Accessibility Delegate方法很简单：

Public void setAccessibilityDelegate(AccessibilityDelegate delegate) {

mAccessibilityDelegate = delegate;

}

注册后，View对无障碍的处理，则会交给AccessibilityDelegate，如：

public void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info) {

if (mAccessibilityDelegate != null) {

mAccessibilityDelegate.onInitializeAccessibilityNodeInfo(this, info);

} else {

onInitializeAccessibilityNodeInfoInternal(info);

}

}

onInitializeAccessibilityNodeInfo是View源码中初始化无障碍节点信息的方法，从上面代码看出，当mAccessibilityDelegate是开发注册的AccessiblityDelegate时，则会执行AccessiblityDelegate中的onInitializeAccessibilityNodeInfo方法。再看看AccessibilityDelegate类中的onInitializeAccessibilityNodeInfo：

public void onInitializeAccessibilityNodeInfo(View host, AccessibilityNodeInfo info) {

host.onInitializeAccessibilityNodeInfoInternal(info);

}

Host是被注册AccessibilityDelegate的View，onInitializeAccessibilityNodeInfoInternal是View中真正初始化无障碍节点信息的方法。即是说，注册了AccessibilityDelegate并没有改变View原来对无障碍的操作，而是在这个操作之后增加了处理。

**AccessibilityDelegate的应用**

下面介绍下AccessibilityDelegate可以提供哪些无障碍应用，注册AccessibilityDelegate是在API 14以上才开放的接口，API 14以下如需使用，可以接入support v4包中的AccessibilityDelegateCompt。注册方法如下：

if (Build.VERSION.SDK\_INT >= 14) {

View view = findViewById(R.id.view\_id);

view.setAccessibilityDelegate(new AccessibilityDelegate() {

public void onInitializeAccessibilityNodeInfo(View host,

AccessibilityNodeInfo info) {

super.onInitializeAccessibilityNodeInfo(host, info);

// 对info做出扩展性支持

});

}

要对info做出扩展支持，还得先了解AccessibilityNodeInfo这个类。Android开发都知道，UI上的元素是通过View来实现，而AccessibilityNodeInfo则是存储View的无障碍信息（如contentDescription）及无障碍状态（如focusable、visiable、clickable等），同时它还肩负着TalkBack服务和View之间通讯的桥梁作用。如果要修改View的无障碍提示，比如修改View的类型提示，可以这样做：

if(android.os.Build.VERSION.SDK\_INT >= 14){

view.setAccessibilityDelegate(new AccessibilityDelegate(){

@Override

public void onInitializeAccessibilityNodeInfo(View host, AccessibilityNodeInfo info) {

super.onInitializeAccessibilityNodeInfo(host, info);

if(contentDesc != null) {

info.setContentDescription(contentDesc);

}

info.setClassName(className);

}

});

}

className是类型名称，这里如果是Button.class.getName()，则TalkBack会对这个View读“XXX 按钮”，“XXX”是contentDescription，而“按钮”则是TalkBack服务添加的（如果是英文环境，则是“XXX button”）。使用上面的方法，可以为非按钮控件加上“按钮”的提示，方便无障碍用户识别UI上元素的作用，同时又不必把“按钮”提示强加入contentDescription中。除了修改AccessibilityNodeInfo外，使用AccessibilityDelegate还可以影响无障碍事件，如：

if (android.os.Build.VERSION.SDK\_INT >= 14) {

view.setAccessibilityDelegate(new AccessibilityDelegate() {

@Override

public void sendAccessibilityEvent(View host, int eventType) {

// 弹出Popup后，不自动读各项内容

if (eventType != AccessibilityEvent.TYPE\_WINDOW\_STATE\_CHANGED) {

super.sendAccessibilityEvent(host, eventType);

}

}

});

}

无障碍模式下，弹出Dialog，则会把Dialog中的所有元素都读一遍。这个方法可以把弹起窗口的无障碍事件拦截，Dialog弹起就不会再自动读各项内容。

高级篇

有了AccessibilityDelegate这把利器之后，开发可以轻松对应Android中大部分的无障碍化，但是如果想要做到游刃有余，还得深造更高级的功夫——自定义View无障碍化。

应用开发过程中，总会需要自定义View来实现特殊的UI效果，当一个自定义View中包含多种UI元素时，无障碍模式下并不能区分包含的多种UI元素，而只为自定义View添加一个大无障碍焦点。如图2所示。

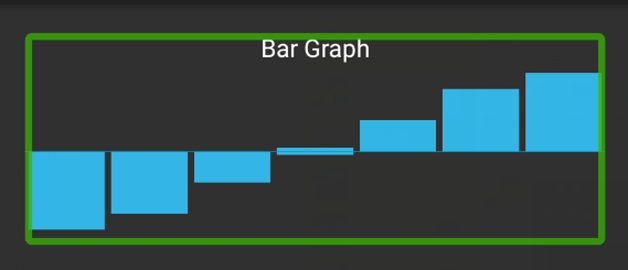


图2 自定义View只有一个大无障碍焦点

图中只有一个大无障碍焦点，因为这是一个View，里面的文字及蓝色的矩形都是绘制出来的。

@Override

public void onDraw(Canvas c) {

super.onDraw(c);

if (mTitle != null) {

drawTitle(c);

}

for (int i = 0; i < mSize; i++) {

drawBarAtIndex(c, i);

}

drawAxisY(c);

}

代码中绘制出来的元素不可被TalkBack识别出来，所以开发需要多做一步，对自定义View无障碍化。这里将介绍如何通过官方提供的ExploreByTouchHelper来实现。

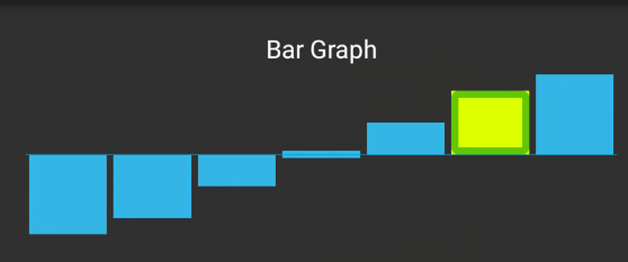


图3 自定义View内元素获得无障碍焦点

图3是使用ExploreByTouchHelper实现的最终效果，每一个小矩形都能获取到无障碍焦点，且可以进行选中高亮。实现ExploreByTouchHelper需要五步。   
第一步，委托处理无障碍。

public class BarGraphView extends View {

private final BarGraphAccessHelper mBarGraphAccessHelper;

public BarGraphView(Context context, AttributeSet attrs, int defStyle) {

super(context, attrs, defStyle);

...

mBarGraphAccessHelper = new BarGraphAccessHelper(this);

ViewCompat.setAccessibilityDelegate(this, mBarGraphAccessHelper);

}

@Override

public boolean dispatchHoverEvent(MotionEvent event) {

if ((mBarGraphAccessHelper != null)

&& mBarGraphAccessHelper.dispatchHoverEvent(event)) {

return true;

}

return super.dispatchHoverEvent(event);

}

}

mBarGraphAccessHelper继承ExploreBy TouchHelper，可通过注册AccessibilityDelegate的方式来注册给自定义BarGraphView，同时让mBarGraphAccessHelper来处理Hover事件（无障碍模式下的点击）的分发。

第二步，标记无障碍虚拟节点ID。

private class BarGraphAccessHelper extends ExploreByTouchHelper {

private final Rect mTempParentBounds = new Rect();

public BarGraphAccessHelper(View parentView) {

super(parentView);

}

@Override

protected int getVirtualViewIdAt(float x, float y) {

final int index = getBarIndexAt(x, y);

if (index >= 0) {

return index;

}

return ExploreByTouchHelper.INVALID\_ID;

}

@Override

protected void getVisibleVirtualViewIds(List<integer> virtualViewIds) {

final int count = getBarCount();

for (int index = 0; index < count; index++) {

virtualViewIds.add(index);

}

}

}

getVirtualViewIdAt和getVisibleVirtualViewIds都是ExploreByTouchHelper类需要实现的方法，分别代表获取虚拟无障碍节点的id以及设置虚拟无障碍节点的id。由于自定义View里的元素非继承于View，如要在无障碍模式下被识别，则需要构造一个虚拟无障碍节点。构造方法已封装到ExploreByTouchHelper里，开发只需要告诉ExploreByTouchHelper有哪些虚拟无障碍节点的id即可。无障碍节点id需要满足以下条件：id是一个接一个的，稳定且为非负整数。设置好无障碍虚拟节点id后，根据用户操作UI上的xy坐标，取得对应的无障碍虚拟节点id，通过getVirtualViewIdAt方法告诉ExploreByTouchHelper类。

第三步，填充无障碍节点的属性。

private class BarGraphAccessHelper extends ExploreByTouchHelper {

...

private CharSequence getDescriptionForIndex(int index) {

final int value = getBarValue(index);

final int templateRes = ((mHighlightedIndex == index) ?

R.string.bar\_desc\_highlight : R.string.bar\_desc);

return getContext().getString(templateRes, index, value);

}

@Override

protected void populateEventForVirtualViewId(int virtualViewId, AccessibilityEvent event) {

final CharSequence desc = getDescriptionForIndex(virtualViewId);

event.setContentDescription(desc);

}

@Override

protected void populateNodeForVirtualViewId(

int virtualViewId, AccessibilityNodeInfoCompat node) {

final CharSequence desc = getDescriptionForIndex(virtualViewId);

node.setContentDescription(desc);

node.addAction(AccessibilityNodeInfoCompat.ACTION\_CLICK);

final Rect bounds = getBoundsForIndex(virtualViewId, mTempParentBounds);

node.setBoundsInParent(bounds);

}

}

构造了虚拟无障碍节点后，便可以往节点里塞无障碍信息。populateEventForVirtualViewId是将无障碍信息填入无障碍事件中。populateNodeForVirtualViewId是初始化每个虚拟无障碍节点，设置contentDescription，注册所需要处理的Action，以及设置无障碍焦点的边框。setBoundsInParent一定要设置有效的边框，否则会导致虚拟无障碍节点无法获取无障碍焦点。

第四步，提供用户无障碍交互支持。

private class BarGraphAccessHelper extends ExploreByTouchHelper {

...

@Override

protected boolean performActionForVirtualViewId(

int virtualViewId, int action, Bundle arguments) {

switch (action) {

case AccessibilityNodeInfoCompat.ACTION\_CLICK:

onBarClicked(virtualViewId);

return true;

}

return false;

}

}

private void onBarClicked(int index) {

setSelection(index);

if (mBarGraphAccessHelper != null) {

mBarGraphAccessHelper.sendEventForVirtualViewId(

index, AccessibilityEvent.TYPE\_VIEW\_CLICKED);

}

}

小矩形点击后是会被选中且高亮的，在performActionForVirtualViewId中实现对应的点击事件处理。经过这四步，自定义View就可以完美支持无障碍化了！

可以看出，ExploreByTouchHelper简化了虚拟节点层次结构的构造，封装AccessibilityNodeProvider的实现，更完善的控制Hover事件、无障碍事件。有了它，Android无障碍化再也不是难题。

Android无障碍化宝典的内容就介绍到此，在实际开发中，遇到的无障碍化问题都比较细小和琐碎，希望以上介绍能提供一点帮助。很多Android开发以为无障碍化就是为控件加上ContentDescription，其实还有空描述、混乱焦点、焦点顺序、描述准确性等地方需要注意和优化。只有用心、持续地改进和优化，才能做出真正无障碍的产品。