DisplayCutout in Android 8.1 (eng)

> This is guide page related to the method of using DisplayCutout supported from Android 8.1 (O_MR1).

INDEX

Purpose and range

Google is providing DisplayCutout as an integrated solution regarding Notch screen in Android 9.0.

But it is not provided officially in versions below 9.0.

DisplayCutout has been applied for supporting Notch screen in 8.1 as well and, this is the guide for the method of using DisplayCutout API in Android 8.1 in this page.

Google Android 9.0 DisplayCutout : https://developer.android.com/about/versions/pie/android-9.0#cutout

DisplayCutout API

Major API list

- DisplayCutout & WindowInsets

DisplayCutout Class	
List<Rect>	getBoundingRects() Returns a list of Rects, each of which is the bounding rectangle for a non-functional area on the display.
int	getSafeInsetBottom() Returns the inset from the bottom which avoids the display cutout in pixels.
int	getSafeInsetLeft() Returns the inset from the left which avoids the display cutout in pixels.
int	getSafeInsetRight() Returns the inset from the right which avoids the display cutout in pixels.
int	getSafeInsetTop() Returns the inset from the top which avoids the display cutout in pixels.
WindowInsets Class	
DisplayCutout	getDisplayCutout() Returns the display cutout if there is one.
WindowInsets	consumeDisplayCutout() Returns a copy of this WindowInsets with the cutout fully consumed.

- WindowManager
  https://developer.android.com/reference/android/view/WindowManager.LayoutParams#layoutInDisplayCutoutMode

Application of DisplayCutout (Notch area extension)

The method of applying DisplayCutout follows the guide of DisplayCutout of AOSP 9.0.

Refer to https://developer.android.com/guide/topics/display-cutout/

The method of applying DisplayCutout follows the guide of DisplayCutout of AOSP 9.0.

There are two ways to apply it to xml and code. It is the same in 9.0.

1. xml is available in both 9.0 and 8.1.
2. code should use reflection. Only available in 8.1.

Below is an example of how to apply.

Method of applying DisplayCutout through xml : windowLayoutInDisplayCutoutMode

- default : This is the default behavior, as described above. Content renders into the cutout area while in portrait mode, but content is letterboxed while in landscape mode.
- shortEdges : Content renders into the cutout area in both portrait and landscape modes.
- never : Content never renders into the cutout area.

1) Declaration of windowLayoutInDisplayCutoutMode through styles

(Sample) res/values/styles.xml

```
<style name="DisplayCutoutDefault" parent="AppTheme">
    <item name="android:windowLayoutInDisplayCutoutMode">default</item>
</style>

<style name="ShortEdges" parent="AppTheme">
    <item name="android:windowLayoutInDisplayCutoutMode">shortEdges</item>
</style>
```

2) Assigning theme to Application or Activity

(Sample) AndroidManifest.xml

```
<activity
 android:name=".DefaultActivity"
 android:theme="@style/DisplayCutoutDefault">
</activity>

<activity
 android:name=".ShortEdgesActivity"
 android:theme="@style/ShortEdges">
</activity>
```

Method of applying DisplayCutout through Code : layoutInDisplayCutoutMode

- LAYOUT_IN_DISPLAY_CUTOUT_MODE_DEFAULT : This is the default behavior, as described above. Content renders into the cutout area while in portrait mode, but content is letterboxed while in landscape mode.
- LAYOUT_IN_DISPLAY_CUTOUT_MODE_SHORT_EDGES : Content renders into the cutout area in both portrait and landscape modes.
- LAYOUT_IN_DISPLAY_CUTOUT_MODE_NEVER : Content never renders into the cutout area.

1) Obtains the instance of WindowManager.LayoutParams from Window.

(Sample) getAttributes

```
Window window = getWindow();
window.addFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
window.addFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN);
WindowManager.LayoutParams lp = window.getAttributes();
```

2) Obtains the field of layoutInDisplayCutoutMode from the instance of WindowManager.LayoutParams.

(Sample) getField("layoutInDisplayCutoutMode")

```
try {
    Field field = lp.getClass().getField("layoutInDisplayCutoutMode");
```

3) Sets the value of mode which you want to apply in Field and, sets to attribute of Window.

(Sample) setInt(lp, mode)

```
    field.setAccessible(true);
    // LAYOUT_IN_DISPLAY_CUTOUT_MODE_DEFAULT     0
    // LAYOUT_IN_DISPLAY_CUTOUT_MODE_SHORT_EDGES 1
    // LAYOUT_IN_DISPLAY_CUTOUT_MODE_NEVER       2
    field.setInt(lp, mode);
    window.setAttributes(lp);
} catch (Exception e) {
    e.printStackTrace();
}
```

Method of obtaining DisplayCutout info (SafeInset, BoudingRects)

APIs which are there in DisplayCutout main API list can be used in 8.1's DisplayCutout by using Reflection.

※ Obtaining SafeInset or BoundingRect (also using getSafeInsetTop, getSafeInsetBottom, getSafeInsetLeft, getSafeInsetRight, getBoundingRects)

1) First of all for obtaining DisplayCutout's SafeInset area or BoundingRect, DisplayCutout's instance must be obtained by using reflection of WindowInsets's getDisplayCutout.

(Sample) Reflection 1

```
Method method = WindowInsets.class.getDeclaredMethod("getDisplayCutout");
Object displayCutoutInstance = method.invoke(windowInsets);
```

2) If the instance of DisplayCutout has been obtained, you can do Reflection of getSafeInsetTop, getSafeInsetBottom, getSafeInsetLeft, getSafeInsetRight, getBoundingRects method by using instance.

(Sample) Reflection 2

```
Rect safeInsets = new Rect();
List<Rect> boundingRects = new ArrayList<>();

Class cls = displayCutoutInstance.getClass();

int top = (int)cls.getDeclaredMethod("getSafeInsetTop").invoke
(displayCutoutInstance);
int bottom = (int)cls.getDeclaredMethod("getSafeInsetBottom").invoke
(displayCutoutInstance);
int left = (int)cls.getDeclaredMethod("getSafeInsetLeft").invoke
(displayCutoutInstance);
int right = (int)cls.getDeclaredMethod("getSafeInsetRight").invoke
(displayCutoutInstance);
safeInsets.set(left, top, right, bottom);

boundingRects.addAll((List<Rect>)cls.getDeclaredMethod("getBoundingRects").
invoke(displayCutoutInstance));
```

※ Obtaining Consumed DisplayCutout(WindowInsets) (using consumeDisplayCutout)

- For obtaining DisplayCutout's Consumed DisplayCutout(WindowInsets), it can be ontained by reflection of consumeDisplayCutout of WindowInsets.

(Sample) Reflection 3

```
Method method = WindowInsets.class.getDeclaredMethod
("consumeDisplayCutout");
WindowInsets insets = (WindowInsets)method.invoke(mInner);
```

Sample code

Below is the sample code of using by wrapping DisplayCutout and WindowInsests by using reflection.

DisplayCutoutWrapper class obtains DisplayCutout instance by reflection of WindowInset and, is emulating DisplayCutout as the class which saves after obtaining each SafeInsets and BoundingRect through Reflection.

Since it is DisplayCutoutWrapper class, it can be used for calling DisplayCutout class's getSafeInsetTop, getSafeInsetBottom, getSafeInsetLeft, getSafeInsetRight, getBoundingRects.

(Sample Code) DisplayCutoutWrapper.java

```
public class DisplayCutoutWrapper {

    private static final String GET_DISPLAY_CUTOUT = "getDisplayCutout";
    private static final String GET_SAFE_INSET_TOP = "getSafeInsetTop";
    private static final String GET_SAFE_INSET_BOTTOM =
"getSafeInsetBottom";
    private static final String GET_SAFE_INSET_LEFT = "getSafeInsetLeft";
```

```
    private static final String GET_SAFE_INSET_RIGHT = "getSafeInsetRight";
    private static final String GET_BOUNDING_RECTS = "getBoundingRects";

    private final Rect mSafeInsets = new Rect();
    private final List<Rect> mBoundingRects = new ArrayList<>();
    public DisplayCutoutWrapper(WindowInsets windowInsets) {
        try {
            Method method = WindowInsets.class.getDeclaredMethod
(GET_DISPLAY_CUTOUT);
            Object displayCutoutInstance = method.invoke(windowInsets);

            Class cls = displayCutoutInstance.getClass();

            int top = (int)cls.getDeclaredMethod(GET_SAFE_INSET_TOP).invoke
(displayCutoutInstance);
            int bottom = (int)cls.getDeclaredMethod(GET_SAFE_INSET_BOTTOM).
invoke(displayCutoutInstance);
            int left = (int)cls.getDeclaredMethod(GET_SAFE_INSET_LEFT).
invoke(displayCutoutInstance);
            int right = (int)cls.getDeclaredMethod(GET_SAFE_INSET_RIGHT).
invoke(displayCutoutInstance);
            mSafeInsets.set(left, top, right, bottom);

            mBoundingRects.addAll((List<Rect>)cls.getDeclaredMethod
(GET_BOUNDING_RECTS).invoke(displayCutoutInstance));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /** Returns the inset from the top which avoids the display cutout in
pixels. */
 public int getSafeInsetTop() {
        return mSafeInsets.top;
    }

    /** Returns the inset from the bottom which avoids the display cutout
in pixels. */
 public int getSafeInsetBottom() {
        return mSafeInsets.bottom;
    }

    /** Returns the inset from the left which avoids the display cutout in
pixels. */
 public int getSafeInsetLeft() {
        return mSafeInsets.left;
    }

    /** Returns the inset from the right which avoids the display cutout
in pixels. */
 public int getSafeInsetRight() {
        return mSafeInsets.right;
    }
```

```
    /**
 * Returns a list of {@code Rect}s, each of which is the bounding
rectangle for a non-functional
 * area on the display.
 *
 * There will be at most one non-functional area per short edge of the
device, and none on
 * the long edges.
 *
 * @return a list of bounding {@code Rect}s, one for each display cutout
area.
 */
 public List<Rect> getBoundingRects() {
        return mBoundingRects;
    }

    @Override
 public String toString() {
        StringBuilder sb = new StringBuilder(32);
        sb.append("DisplayCutout{");
        sb.append("safeInsets=" + mSafeInsets);
        sb.append(", boundingRect=");
        if (mBoundingRects.isEmpty()) {
            sb.append("Empty");
        }
        for (int i = 0; i < mBoundingRects.size(); ++i) {
            sb.append(mBoundingRects.get(i));
        }
        sb.append("}");
        return sb.toString();
    }
}
```

WindowInsetsWrapper class is carrying WindowInset instance as inner and, imports DisplayCutoutWrapper but, DisplayCutout consumed WindowInsets can be imported by calling consumeDisplayCutout.

(Sample Code) WindowInsetsWrapper.java

```
public class WindowInsetsWrapper {

    private static final String CONSUME_DISPLAY_CUTOUT =
"consumeDisplayCutout";

    private WindowInsets mInner;
    private DisplayCutoutWrapper mDisplayCutoutWrapper;
    public WindowInsetsWrapper(WindowInsets windowInsets) {
        mInner = windowInsets;
        mDisplayCutoutWrapper = new DisplayCutoutWrapper(mInner);
    }

    /**
 * Returns the display cutout if there is one.
 *
 * @return the display cutout or null if there is none
 * @see DisplayCutout
 */
 @Nullable
 public DisplayCutoutWrapper getDisplayCutoutWrapper() {
        return mDisplayCutoutWrapper;
    }

    /**
 * Returns a copy of this WindowInsets with the cutout fully consumed.
 *
 * @return A modified copy of this WindowInsets
 */
 public WindowInsets consumeDisplayCutout() {
        WindowInsets insets = null;
        try {
            Method method = WindowInsets.class.getDeclaredMethod
(CONSUME_DISPLAY_CUTOUT);
            insets = (WindowInsets)method.invoke(mInner);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return insets;
    }
}
```

Actual coding example using DisplayCutoutWrapper and WindowInsetsWrapper

(Sample Code) BaseActivity.java

```
@Override
public WindowInsets onApplyWindowInsets(View view, WindowInsets insets) {
    WindowInsetsWrapper wrapper = new WindowInsetsWrapper(insets);
    mLastDisplayCutoutWrapper = wrapper.getDisplayCutoutWrapper();

    Log.i(TAG, mLastDisplayCutoutWrapper.toString());
    Log.i(TAG, "Top=" + mLastDisplayCutoutWrapper.getSafeInsetTop());
    Log.i(TAG, "Bottom=" + mLastDisplayCutoutWrapper.getSafeInsetBottom());
    Log.i(TAG, "Left=" + mLastDisplayCutoutWrapper.getSafeInsetLeft());
    Log.i(TAG, "Right=" + mLastDisplayCutoutWrapper.getSafeInsetRight());
    List<Rect> boundingRects = mLastDisplayCutoutWrapper.
getBoundingRects();
    for (int i = 0; i < boundingRects.size(); ++i) {
        Log.i(TAG, "BoundingRects=" + boundingRects.get(i));
    }

    TextView txtSafeInset = findViewById(R.id.txtSafeInset);
    txtSafeInset.setText(mLastDisplayCutoutWrapper.toString());
    return insets;
}
```

Note

1. How to classify whether Notch (DisplayCutout) API can be used?

   It's not official, resource("config_mainBuiltInDisplayCutout") contains cutout spec.

   Check that the string of config_mainBuiltInDisplayCutout is empty.

   e.g. If you have a spec, it's a cutout model.

   How to classify whether Notch

   ```
               try {
                   final Resources res = context.getResources();
                   final int resId = res.getIdentifier
    ("config_mainBuiltInDisplayCutout", "string", "android");
                   final String spec = resId > 0 ? res.getString(resId)
    : null;
                   mHasDisplayCutout = spec != null && !TextUtils.isEmpty
    (spec);
               } catch (Exception e) {
                   Log.w(mLogTag, "Can not update hasDisplayCutout. " +
    e.toString());
               }
   ```

2. How can the WindowInset can be received which is bringing DisplayCutout?

1. Implements View.OnApplyWindowInsetsListener.

(Sample Code) implements

```
public class BaseActivity extends AppCompatActivity implements
View.OnApplyWindowInsetsListener {


@Override
public WindowInsets onApplyWindowInsets(View view, WindowInsets
insets) {
        // insert your code.
 return insets;
}
```

2. Registers Listener (setOnApplyWindowInsetsListener).

(Sample Code) set listener

```
mDecor = getWindow().getDecorView();
mDecor.setOnApplyWindowInsetsListener(this);
```

3. Have called getDisplayCutout. However, a null value was returned. Why?
   If app's windows do not overlap with cutouts, safe inset is zero. It will return null.
   This is the AOSP logic. Therefore, 9.0 is the same.
4. When does Letterbox appear? (Below is the AOSP logic. Therefore, 9.0 is the same.)
    1. In never mode, it appears in both portrait and landscape.
    2. In the default mode, it always appears in landscape mode.
       In portrait, it may not appear depending on app's implementation.
       e.g. has not FLAG_FULLSCREEN, has FLAG_LAYOUT_IN_SCREEN, has FLAG_LAYOUT_INSET_DECOR => The letterbox does not appear. app can stretch to the cutout.
       e.g. has FLAG_FULLSCREEN => The letterbox does appear.
    3. In short edge mode, The letterbox does not appear.
5. When does Letterbox not appear? (Below is the AOSP logic. Therefore, 9.0 is the same.)
    1. Letterbox only appears when app have an activity(AppWindowToken).
    2. If the style below is declared, Letterbox will not appear.
        1. <item name="windowIsTranslucent">true</item>
        2. <item name="windowIsFloating">true</item>
        3. <item name="windowSwipeToDismiss">true</item>
6. What happens in short edge mode?
    1. If there is no implementation, the background of the window is visible.