# View

**当前的动画，通过startAnimation或setAnimation设置。在ViewGroup中，如果子View可见或者Animation不为空，会绘制此View。**

**protected** Animation mCurrentAnimation = **null**;

**通过onMeasure调用setMeasureDimension设置View在View系统中的实际的大小。**

**int** mMeasuredWidth;

**int** mMeasuredHeight;

**protected** ViewParent mParent;

**View在ViewGroup中的布局的信息**

**protected** ViewGroup.LayoutParams mLayoutParams;

**static** **class** TransformationInfo {

**float** mRotationY = 0f;

**float** mRotationX = 0f;

**float** mRotation = 0f;

**float** mTranslationX = 0f;

**float** mScaleX = 1f;

**float** mScaleY = 1f;

**float** mAlpha = 1f;

**float** mTransitionAlpha = 1f;

}

**通过Transformation修改DisplayList**

TransformationInfo mTransformationInfo;

DisplayList mDisplayList;

**通过layout方法调用setFrame设置的View在View系统中的位置信息。mMeasuredWidth= mRight- mLeft。**

**protected** **int** mLeft;

**protected** **int** mRight;

**protected** **int** mTop;

**protected** **int** mBottom;

**通过scrollTo或者scrollBy修改Canvas的位移值。在调用boolean draw(Canvas canvas, ViewGroup parent, long drawingTime) 的时候，会通过mScrollX和mScrollY修改Canvas。**

**mScrollX和mScrollY表示的是canvas的原点,为负值包含的子View才像右下移动。**

**protected** **int** mScrollX;

**protected** **int** mScrollY;

**View内部的Padding信息**

**protected** **int** mPaddingBottom;

**protected** **int** mPaddingTop;

**protected** **int** mPaddingRight = 0;

**protected** **int** mPaddingLeft = 0;

**获取和设置X的位置，原理是mLeft保持不变，修改mTranslationX**

**public** **float** getX() {

**return** mLeft + (mTransformationInfo != **null** ? mTransformationInfo.mTranslationX : 0);

}

**public** **void** setX(**float** x) {

setTranslationX(x - mLeft);

}

# TouchEvent

如果子窗口没有处理onTouchEvent，那么父窗口可以继续处理onTouchEvent

父窗口在onInterceptEvent做了截获处理，子窗口将不能处理onTouchEvent，将留给父窗口

ViewGroup requestDisallowInterceptTouchEvent

ViewGroup dispatchTouchEvent {

ViewGroup onInterceptTouchEvent

ViewGroup dispatchTransformedTouchEvent {

View dispatchTouchEvent {

1，是否有mOnTouchListener

2，View onTouchEvent

}

}

}

## Activity

### public boolean dispatchTouchEvent(MotionEvent ev) {

**if** (ev.getAction() == MotionEvent.*ACTION\_DOWN*) {

onUserInteraction();

}

**if** (getWindow().superDispatchTouchEvent(ev)) {

**return** **true**;

}

**return** onTouchEvent(ev);

}

如果窗口没有处理时间，交给Activity处理。

## View

/\*\*

\* Pass the touch screen motion event down to the target view, or this

\* view if it is the target.

\*

\* **@param** event The motion event to be dispatched.

\* **@return** True if the event was handled by the view, false otherwise.

\*/

### public boolean dispatchTouchEvent(MotionEvent event) {

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onTouchEvent(event, 0);

}

// 如果mOnTouchListener处理了消息，不传递给onTouchEvent

**if** (onFilterTouchEventForSecurity(event)) {

//noinspection SimplifiableIfStatement

ListenerInfo li = mListenerInfo;

**if** (li != **null** && li.mOnTouchListener != **null** && (mViewFlags & *ENABLED\_MASK*) == *ENABLED*

&& li.mOnTouchListener.onTouch(**this**, event)) {

**return** **true**;

}

//如果onTouchEvent处理了事件，返回True

**if** (onTouchEvent(event)) {

**return** **true**;

}

}

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onUnhandledEvent(event, 0);

}

**return** **false**;

}

/\*\*

\* Implement this method to handle touch screen motion events.

\*

\* **@param** event The motion event.

\* **@return** True if the event was handled, false otherwise.

\*/

### public boolean onTouchEvent(MotionEvent event) {

**final** **int** viewFlags = mViewFlags;

//如果是DISABLED状态

**if** ((viewFlags & *ENABLED\_MASK*) == *DISABLED*) {

**if** (event.getAction() == MotionEvent.*ACTION\_UP* && (mPrivateFlags & *PFLAG\_PRESSED*) != 0) {

setPressed(**false**);

}

// A disabled view that is clickable still consumes the touch

// events, it just doesn't respond to them.

**return** (((viewFlags & *CLICKABLE*) == *CLICKABLE* ||

(viewFlags & *LONG\_CLICKABLE*) == *LONG\_CLICKABLE*));

}

**if** (((viewFlags & *CLICKABLE*) == *CLICKABLE* ||

(viewFlags & *LONG\_CLICKABLE*) == *LONG\_CLICKABLE*)) {

**switch** (event.getAction()) {

**case** MotionEvent.*ACTION\_UP*:

… …

**if** (!mHasPerformedLongPress) {

// This is a tap, so remove the longpress check

removeLongPressCallback();

// Only perform take click actions if we were in the pressed state

**if** (!focusTaken) {

// Use a Runnable and post this rather than calling

// performClick directly. This lets other visual state

// of the view update before click actions start.

**if** (mPerformClick == **null**) {

mPerformClick = **new** PerformClick();

}

//触发点击事件

**if** (!post(mPerformClick)) {

performClick();

}

}

}

… …

removeTapCallback();

}

**break**;

**case** MotionEvent.*ACTION\_DOWN*:

mHasPerformedLongPress = **false**;

**if** (performButtonActionOnTouchDown(event)) {

**break**;

}

// Walk up the hierarchy to determine if we're inside a scrolling container.

// 遍历是否在可滚动的容器内，如果是检测是否为点击

**boolean** isInScrollingContainer = isInScrollingContainer();

// For views inside a scrolling container, delay the pressed feedback for

// a short period in case this is a scroll.

**if** (isInScrollingContainer) {

mPrivateFlags |= *PFLAG\_PREPRESSED*;

**if** (mPendingCheckForTap == **null**) {

mPendingCheckForTap = **new** CheckForTap();

}

postDelayed(mPendingCheckForTap, ViewConfiguration.*getTapTimeout*());

} **else** {

// Not inside a scrolling container, so show the feedback right away

setPressed(**true**);

checkForLongClick(0);

}

**break**;

**case** MotionEvent.*ACTION\_CANCEL*:

setPressed(**false**);

removeTapCallback();

removeLongPressCallback();

**break**;

**case** MotionEvent.*ACTION\_MOVE*:

**final** **int** x = (**int**) event.getX();

**final** **int** y = (**int**) event.getY();

// Be lenient about moving outside of buttons

**if** (!pointInView(x, y, mTouchSlop)) {

// Outside button

removeTapCallback();

**if** ((mPrivateFlags & *PFLAG\_PRESSED*) != 0) {

// Remove any future long press/tap checks

removeLongPressCallback();

setPressed(**false**);

}

}

**break**;

}

// 处理所有的消息

**return** **true**;

}

**return** **false**;

}

## ViewGroup

@Override

### public boolean dispatchTouchEvent(MotionEvent ev) {

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onTouchEvent(ev, 1);

}

**boolean** handled = **false**;

**if** (onFilterTouchEventForSecurity(ev)) {

**final** **int** action = ev.getAction();

**final** **int** actionMasked = action & MotionEvent.*ACTION\_MASK*;

// Handle an initial down.

**if** (actionMasked == MotionEvent.*ACTION\_DOWN*) {

// Throw away all previous state when starting a new touch gesture.

// The framework may have dropped the up or cancel event for the previous gesture

// due to an app switch, ANR, or some other state change.

cancelAndClearTouchTargets(ev);

resetTouchState();

}

// Check for interception.

**final** **boolean** intercepted;

**if** (actionMasked == MotionEvent.*ACTION\_DOWN*

|| mFirstTouchTarget != **null**) {

**final** **boolean** disallowIntercept = (mGroupFlags & *FLAG\_DISALLOW\_INTERCEPT*) != 0;

**if** (!disallowIntercept) {

// 判断是否截获消息

intercepted = onInterceptTouchEvent(ev);

ev.setAction(action); // restore action in case it was changed

} **else** {

intercepted = **false**;

}

} **else** {

// There are no touch targets and this action is not an initial down

// so this view group continues to intercept touches.

intercepted = **true**;

}

// Check for cancelation.

**final** **boolean** canceled = *resetCancelNextUpFlag*(**this**)

|| actionMasked == MotionEvent.*ACTION\_CANCEL*;

// Update list of touch targets for pointer down, if needed.

**final** **boolean** split = (mGroupFlags & *FLAG\_SPLIT\_MOTION\_EVENTS*) != 0;

TouchTarget newTouchTarget = **null**;

**boolean** alreadyDispatchedToNewTouchTarget = **false**;

// 如果消息没有被截获，判断子窗口

**if** (!canceled && !intercepted) {

**if** (actionMasked == MotionEvent.*ACTION\_DOWN*

|| (split && actionMasked == MotionEvent.*ACTION\_POINTER\_DOWN*)

|| actionMasked == MotionEvent.*ACTION\_HOVER\_MOVE*) {

**final** **int** actionIndex = ev.getActionIndex(); // always 0 for down

**final** **int** idBitsToAssign = split ? 1 << ev.getPointerId(actionIndex)

: TouchTarget.*ALL\_POINTER\_IDS*;

// Clean up earlier touch targets for this pointer id in case they

// have become out of sync.

removePointersFromTouchTargets(idBitsToAssign);

**final** **int** childrenCount = mChildrenCount;

**if** (childrenCount != 0) {

// Find a child that can receive the event.

// Scan children from front to back.

**final** View[] children = mChildren;

**final** **float** x = ev.getX(actionIndex);

**final** **float** y = ev.getY(actionIndex);

**final** **boolean** customOrder = isChildrenDrawingOrderEnabled();

**for** (**int** i = childrenCount - 1; i >= 0; i--) {

**final** **int** childIndex = customOrder ?

getChildDrawingOrder(childrenCount, i) : i;

**final** View child = children[childIndex];

// 判断子窗口的位置是否能接受消息

**if** (!*canViewReceivePointerEvents*(child)

|| !isTransformedTouchPointInView(x, y, child, **null**)) {

**continue**;

}

newTouchTarget = getTouchTarget(child);

**if** (newTouchTarget != **null**) {

// Child is already receiving touch within its bounds.

// Give it the new pointer in addition to the ones it is handling.

newTouchTarget.pointerIdBits |= idBitsToAssign;

**break**;

}

*resetCancelNextUpFlag*(child);

// 子窗口已经处理了此消息

**if** (dispatchTransformedTouchEvent(ev, **false**, child, idBitsToAssign)) {

// Child wants to receive touch within its bounds.

mLastTouchDownTime = ev.getDownTime();

mLastTouchDownIndex = childIndex;

mLastTouchDownX = ev.getX();

mLastTouchDownY = ev.getY();

newTouchTarget = addTouchTarget(child, idBitsToAssign);

alreadyDispatchedToNewTouchTarget = **true**;

**break**;

}

}

}

**if** (newTouchTarget == **null** && mFirstTouchTarget != **null**) {

// Did not find a child to receive the event.

// Assign the pointer to the least recently added target.

newTouchTarget = mFirstTouchTarget;

**while** (newTouchTarget.next != **null**) {

newTouchTarget = newTouchTarget.next;

}

newTouchTarget.pointerIdBits |= idBitsToAssign;

}

}

}

// Dispatch to touch targets.

**if** (mFirstTouchTarget == **null**) {

// No touch targets so treat this as an ordinary view.

// 没有子窗口，自身处理此消息

handled = dispatchTransformedTouchEvent(ev, canceled, **null**,

TouchTarget.*ALL\_POINTER\_IDS*);

} **else** {

// Dispatch to touch targets, excluding the new touch target if we already

// dispatched to it. Cancel touch targets if necessary.

TouchTarget predecessor = **null**;

TouchTarget target = mFirstTouchTarget;

**while** (target != **null**) {

**final** TouchTarget next = target.next;

**if** (alreadyDispatchedToNewTouchTarget && target == newTouchTarget) {

handled = **true**;

} **else** {

**final** **boolean** cancelChild = *resetCancelNextUpFlag*(target.child)

|| intercepted;

**if** (dispatchTransformedTouchEvent(ev, cancelChild,

target.child, target.pointerIdBits)) {

handled = **true**;

}

**if** (cancelChild) {

**if** (predecessor == **null**) {

mFirstTouchTarget = next;

} **else** {

predecessor.next = next;

}

target.recycle();

target = next;

**continue**;

}

}

predecessor = target;

target = next;

}

}

// Update list of touch targets for pointer up or cancel, if needed.

**if** (canceled

|| actionMasked == MotionEvent.*ACTION\_UP*

|| actionMasked == MotionEvent.*ACTION\_HOVER\_MOVE*) {

resetTouchState();

} **else** **if** (split && actionMasked == MotionEvent.*ACTION\_POINTER\_UP*) {

**final** **int** actionIndex = ev.getActionIndex();

**final** **int** idBitsToRemove = 1 << ev.getPointerId(actionIndex);

removePointersFromTouchTargets(idBitsToRemove);

}

}

**if** (!handled && mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onUnhandledEvent(ev, 1);

}

**return** handled;

}

### public boolean onInterceptTouchEvent(MotionEvent ev) {

**return** **false**;

}

### public void requestDisallowInterceptTouchEvent(boolean disallowIntercept) {

**if** (disallowIntercept == ((mGroupFlags & *FLAG\_DISALLOW\_INTERCEPT*) != 0)) {

// We're already in this state, assume our ancestors are too

**return**;

}

**if** (disallowIntercept) {

mGroupFlags |= *FLAG\_DISALLOW\_INTERCEPT*;

} **else** {

mGroupFlags &= ~*FLAG\_DISALLOW\_INTERCEPT*;

}

// Pass it up to our parent

**if** (mParent != **null**) {

mParent.requestDisallowInterceptTouchEvent(disallowIntercept);

}

}

/\*\*

\* Transforms a motion event into the coordinate space of a particular child view,

\* filters out irrelevant pointer ids, and overrides its action if necessary.

\* If child is null, assumes the MotionEvent will be sent to this ViewGroup instead.

\*/

### private boolean dispatchTransformedTouchEvent(MotionEvent event, boolean cancel,

View child, **int** desiredPointerIdBits) {

**final** **boolean** handled;

// Canceling motions is a special case. We don't need to perform any transformations

// or filtering. The important part is the action, not the contents.

**final** **int** oldAction = event.getAction();

**if** (cancel || oldAction == MotionEvent.*ACTION\_CANCEL*) {

event.setAction(MotionEvent.*ACTION\_CANCEL*);

**if** (child == **null**) {

handled = **super**.dispatchTouchEvent(event);

} **else** {

handled = child.dispatchTouchEvent(event);

}

event.setAction(oldAction);

**return** handled;

}

// Calculate the number of pointers to deliver.

**final** **int** oldPointerIdBits = event.getPointerIdBits();

**final** **int** newPointerIdBits = oldPointerIdBits & desiredPointerIdBits;

// If for some reason we ended up in an inconsistent state where it looks like we

// might produce a motion event with no pointers in it, then drop the event.

**if** (newPointerIdBits == 0) {

**return** **false**;

}

// If the number of pointers is the same and we don't need to perform any fancy

// irreversible transformations, then we can reuse the motion event for this

// dispatch as long as we are careful to revert any changes we make.

// Otherwise we need to make a copy.

**final** MotionEvent transformedEvent;

**if** (newPointerIdBits == oldPointerIdBits) {

**if** (child == **null** || child.hasIdentityMatrix()) {

**if** (child == **null**) {

handled = **super**.dispatchTouchEvent(event);

} **else** {

**final** **float** offsetX = mScrollX - child.mLeft;

**final** **float** offsetY = mScrollY - child.mTop;

event.offsetLocation(offsetX, offsetY);

handled = child.dispatchTouchEvent(event);

event.offsetLocation(-offsetX, -offsetY);

}

**return** handled;

}

transformedEvent = MotionEvent.*obtain*(event);

} **else** {

transformedEvent = event.split(newPointerIdBits);

}

// Perform any necessary transformations and dispatch.

**if** (child == **null**) {

handled = **super**.dispatchTouchEvent(transformedEvent);

} **else** {

**final** **float** offsetX = mScrollX - child.mLeft;

**final** **float** offsetY = mScrollY - child.mTop;

transformedEvent.offsetLocation(offsetX, offsetY);

**if** (! child.hasIdentityMatrix()) {

transformedEvent.transform(child.getInverseMatrix());

}

handled = child.dispatchTouchEvent(transformedEvent);

}

// Done.

transformedEvent.recycle();

**return** handled;

}

onInterceptTouchEvent在Down事件时返回false：

1. 如果子View不处理此事件,则交给本ViewGroup的onTouchEvent来处理。

onTouchEvent返回false，则不再有dispatchTouchEvent发送消息。

onTouchEvent返回true，就会有其他事件，但是不再调用onInterceptTouchEvent。

2，如果子view处理了事件，那么下一次dispatchTouchEvent时还会调用onInterceptTouchEvent，直到onInterceptTouchEvent返回true，此后消息不再发送给子View，而是交给ViewGroup本身的onTouchEvent来处理。

onInterceptTouchEvent根据截获情况返回mIsBeingDragged。在Down事件中记录初始位置，如果子View没有处理消息，那么交给onTouchEvent来判断是否达到Drag条件。如果子View处理了消息，onInterceptTouchEvent需要判断是否达到了Drag的条件。

onTouchEvent一般都返回true。

# KeyEvent

## Activity

### public boolean dispatchKeyEvent(KeyEvent event) {

onUserInteraction();

Window win = getWindow();

**if** (win.superDispatchKeyEvent(event)) {

**return** **true**;

}

View decor = mDecor;

**if** (decor == **null**) decor = win.getDecorView();

**return** **event.dispatch**(**this**, decor != **null**

? decor.getKeyDispatcherState() : **null**, **this**);

}

如果View系统没有处理key事件，返回给activity处理。

## View

### public boolean dispatchKeyEvent(KeyEvent event) {

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onKeyEvent(event, 0);

}

// Give any attached key listener a first crack at the event.

//noinspection SimplifiableIfStatement

ListenerInfo li = mListenerInfo;

**if** (li != **null** && li.mOnKeyListener != **null** && (mViewFlags & *ENABLED\_MASK*) == *ENABLED*

&& li.mOnKeyListener.onKey(**this**, event.getKeyCode(), event)) {

**return** **true**;

}

**// 处理分发事件**

**if** (**event.dispatch**(**this**, mAttachInfo != **null**

? mAttachInfo.mKeyDispatchState : **null**, **this**)) {

**return** **true**;

}

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onUnhandledEvent(event, 0);

}

**return** **false**;

}

## ViewGroup

### public boolean dispatchKeyEvent(KeyEvent event) {

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onKeyEvent(event, 1);

}

**// 如果ViewGroup有焦点，自己处理，不交给子View**

**if** ((mPrivateFlags & (PFLAG\_FOCUSED | PFLAG\_HAS\_BOUNDS))

== (PFLAG\_FOCUSED | PFLAG\_HAS\_BOUNDS)) {

**if** (**super**.dispatchKeyEvent(event)) {

**return** **true**;

}

**// 把事件交给有焦点的子View处理**

} **else** **if** (mFocused != **null** && (mFocused.mPrivateFlags & PFLAG\_HAS\_BOUNDS)

== PFLAG\_HAS\_BOUNDS) {

**if** (mFocused.dispatchKeyEvent(event)) {

**return** **true**;

}

}

**if** (mInputEventConsistencyVerifier != **null**) {

mInputEventConsistencyVerifier.onUnhandledEvent(event, 1);

}

**return** **false**;

}

# Measure

|  |  |  |
| --- | --- | --- |
| **Measure的结果** | **父View AT\_MOST + parentSize** | **父View EXACTLY + parentSize** |
| 子View match\_parent | AT\_MOST + parentSize | EXACTLY + parentSize |
| 子View wrap\_content | AT\_MOST + parentSize | AT\_MOST + parentSize |
| 子View 10dp childSize | EXACTLY + childSize | EXACTLY + childSize |

ViewRootImpl performTraversals {

ViewRootImpl measureHierarchy {

ViewRootImpl performMeasure {

View measure {

// 只有在view的measure中调用**onMeasure**

**// 需要在具体的布局的onMeasure中实现**

**Framelayout onMeasure {**

**setMeasuredDimension // 设置自身**

// 调用ViewGroup处理子View的measure

ViewGroup measureChildWithMargins {

View.measure {

… …

}

}

}

}

}

}

}

}

## View

### public final void measure(int widthMeasureSpec, int heightMeasureSpec) {

… …

onMeasure(widthMeasureSpec, heightMeasureSpec);

mPrivateFlags |= *PFLAG\_LAYOUT\_REQUIRED*;

}

mOldWidthMeasureSpec = widthMeasureSpec;

mOldHeightMeasureSpec = heightMeasureSpec;

}

### protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

setMeasuredDimension(*getDefaultSize*(getSuggestedMinimumWidth(), widthMeasureSpec),

*getDefaultSize*(getSuggestedMinimumHeight(), heightMeasureSpec));

}

## ViewGroup

### protected void measureChildWithMargins(View child,

**int** parentWidthMeasureSpec, **int** widthUsed,

**int** parentHeightMeasureSpec, **int** heightUsed) {

**final** MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();

**final** **int** childWidthMeasureSpec = *getChildMeasureSpec*(parentWidthMeasureSpec,

mPaddingLeft + mPaddingRight + lp.leftMargin + lp.rightMargin

+ widthUsed, lp.width);

**final** **int** childHeightMeasureSpec = *getChildMeasureSpec*(parentHeightMeasureSpec,

mPaddingTop + mPaddingBottom + lp.topMargin + lp.bottomMargin

+ heightUsed, lp.height);

child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

}

## FrameLayout

### protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {

**int** count = getChildCount();

**final** **boolean** measureMatchParentChildren =

MeasureSpec.getMode(widthMeasureSpec) != MeasureSpec.EXACTLY ||

MeasureSpec.getMode(heightMeasureSpec) != MeasureSpec.EXACTLY;

mMatchParentChildren.clear();

**int** maxHeight = 0;

**int** maxWidth = 0;

**int** childState = 0;

**for** (**int** i = 0; i < count; i++) {

**final** View child = getChildAt(i);

**if** (mMeasureAllChildren || child.getVisibility() != GONE) {

**measureChildWithMargins**(child, widthMeasureSpec, 0, heightMeasureSpec, 0);

**final** LayoutParams lp = (LayoutParams) child.getLayoutParams();

maxWidth = Math.max(maxWidth,

child.getMeasuredWidth() + lp.leftMargin + lp.rightMargin);

maxHeight = Math.max(maxHeight,

child.getMeasuredHeight() + lp.topMargin + lp.bottomMargin);

childState = combineMeasuredStates(childState, child.getMeasuredState());

**if** (measureMatchParentChildren) {

**if** (lp.width == LayoutParams.MATCH\_PARENT ||

lp.height == LayoutParams.MATCH\_PARENT) {

mMatchParentChildren.add(child);

}

}

}

}

// Account for padding too

maxWidth += getPaddingLeftWithForeground() + getPaddingRightWithForeground();

maxHeight += getPaddingTopWithForeground() + getPaddingBottomWithForeground();

// Check against our minimum height and width

maxHeight = Math.max(maxHeight, getSuggestedMinimumHeight());

maxWidth = Math.max(maxWidth, getSuggestedMinimumWidth());

// Check against our foreground's minimum height and width

**final** Drawable drawable = getForeground();

**if** (drawable != **null**) {

maxHeight = Math.max(maxHeight, drawable.getMinimumHeight());

maxWidth = Math.max(maxWidth, drawable.getMinimumWidth());

}

**setMeasuredDimension**(resolveSizeAndState(maxWidth, widthMeasureSpec, childState),

resolveSizeAndState(maxHeight, heightMeasureSpec,

childState << MEASURED\_HEIGHT\_STATE\_SHIFT));

count = mMatchParentChildren.size();

**if** (count > 1) {

**for** (**int** i = 0; i < count; i++) {

**final** View child = mMatchParentChildren.get(i);

**final** MarginLayoutParams lp = (MarginLayoutParams) child.getLayoutParams();

**int** childWidthMeasureSpec;

**int** childHeightMeasureSpec;

**if** (lp.width == LayoutParams.MATCH\_PARENT) {

childWidthMeasureSpec = MeasureSpec.makeMeasureSpec(getMeasuredWidth() -

getPaddingLeftWithForeground() - getPaddingRightWithForeground() -

lp.leftMargin - lp.rightMargin,

MeasureSpec.EXACTLY);

} **else** {

childWidthMeasureSpec = getChildMeasureSpec(widthMeasureSpec,

getPaddingLeftWithForeground() + getPaddingRightWithForeground() +

lp.leftMargin + lp.rightMargin,

lp.width);

}

**if** (lp.height == LayoutParams.MATCH\_PARENT) {

childHeightMeasureSpec = MeasureSpec.makeMeasureSpec(getMeasuredHeight() -

getPaddingTopWithForeground() - getPaddingBottomWithForeground() -

lp.topMargin - lp.bottomMargin,

MeasureSpec.EXACTLY);

} **else** {

childHeightMeasureSpec = getChildMeasureSpec(heightMeasureSpec,

getPaddingTopWithForeground() + getPaddingBottomWithForeground() +

lp.topMargin + lp.bottomMargin,

lp.height);

}

child.measure(childWidthMeasureSpec, childHeightMeasureSpec);

}

}

}

# layout

ViewRootImpl performTraversals {

ViewRootImpl performLayout {

ViewGroup layout {

View layout {

**1，setFrame 设置位置(所有的view)**

2，只有在view的layout中调用**onLayout**

**// 需要在具体的布局的onLayout中实现**

**FrameLayout onLayout {**

**layoutChildren {**

// 处理子View的layout

child = getChildAt(i);

child.layout {

ViewGroup layout {

View layout {

… …

}

}

}

**}**

**}**

}

}

}

}

}

}

## View

### public void layout(int l, int t, int r, int b) {

**int** oldL = mLeft;

**int** oldT = mTop;

**int** oldB = mBottom;

**int** oldR = mRight;

**boolean** changed = setFrame(l, t, r, b);

**if** (changed || (mPrivateFlags & *PFLAG\_LAYOUT\_REQUIRED*) == *PFLAG\_LAYOUT\_REQUIRED*) {

**onLayout**(changed, l, t, r, b);

mPrivateFlags &= ~*PFLAG\_LAYOUT\_REQUIRED*;

ListenerInfo li = mListenerInfo;

**if** (li != **null** && li.mOnLayoutChangeListeners != **null**) {

ArrayList<OnLayoutChangeListener> listenersCopy =

(ArrayList<OnLayoutChangeListener>)li.mOnLayoutChangeListeners.clone();

**int** numListeners = listenersCopy.size();

**for** (**int** i = 0; i < numListeners; ++i) {

listenersCopy.get(i).onLayoutChange(**this**, l, t, r, b, oldL, oldT, oldR, oldB);

}

}

}

mPrivateFlags &= ~*PFLAG\_FORCE\_LAYOUT*;

}

### protected void onLayout(boolean changed, int left, int top, int right, int bottom) {

**// 空的**

}

## ViewGroup

### public final void layout(int l, int t, int r, int b) {

**if** (mTransition == **null** || !mTransition.isChangingLayout()) {

**if** (mTransition != **null**) {

mTransition.layoutChange(**this**);

}

**super**.layout(l, t, r, b);

} **else** {

// record the fact that we noop'd it; request layout when transition finishes

mLayoutSuppressed = **true**;

}

}

### protected abstract void onLayout(boolean changed,

**int** l, **int** t, **int** r, **int** b); **// 空的**

## FrameLayout

### protected void onLayout(boolean changed, int left, int top, int right, int bottom) {

**final** **int** count = getChildCount();

**final** **int** parentLeft = getPaddingLeftWithForeground();

**final** **int** parentRight = right - left - getPaddingRightWithForeground();

**final** **int** parentTop = getPaddingTopWithForeground();

**final** **int** parentBottom = bottom - top - getPaddingBottomWithForeground();

mForegroundBoundsChanged = **true**;

**for** (**int** i = 0; i < count; i++) {

**final** View child = getChildAt(i);

**if** (child.getVisibility() != GONE) {

**final** LayoutParams lp = (LayoutParams) child.getLayoutParams();

**final** **int** width = child.getMeasuredWidth();

**final** **int** height = child.getMeasuredHeight();

**int** childLeft;

**int** childTop;

**int** gravity = lp.gravity;

**if** (gravity == -1) {

gravity = DEFAULT\_CHILD\_GRAVITY;

}

**final** **int** layoutDirection = getLayoutDirection();

**final** **int** absoluteGravity = Gravity.getAbsoluteGravity(gravity, layoutDirection);

**final** **int** verticalGravity = gravity & Gravity.VERTICAL\_GRAVITY\_MASK;

**switch** (absoluteGravity & Gravity.HORIZONTAL\_GRAVITY\_MASK) {

**case** Gravity.LEFT:

childLeft = parentLeft + lp.leftMargin;

**break**;

**case** Gravity.CENTER\_HORIZONTAL:

childLeft = parentLeft + (parentRight - parentLeft - width) / 2 +

lp.leftMargin - lp.rightMargin;

**break**;

**case** Gravity.RIGHT:

childLeft = parentRight - width - lp.rightMargin;

**break**;

**default**:

childLeft = parentLeft + lp.leftMargin;

}

**switch** (verticalGravity) {

**case** Gravity.TOP:

childTop = parentTop + lp.topMargin;

**break**;

**case** Gravity.CENTER\_VERTICAL:

childTop = parentTop + (parentBottom - parentTop - height) / 2 +

lp.topMargin - lp.bottomMargin;

**break**;

**case** Gravity.BOTTOM:

childTop = parentBottom - height - lp.bottomMargin;

**break**;

**default**:

childTop = parentTop + lp.topMargin;

}

child.**layout**(childLeft, childTop, childLeft + width, childTop + height);

}

}

}

# Draw

ViewRootImpl performTraversals {

ViewRootImpl performDraw {

ViewRootImpl draw {

ViewRootImpl drawSoftware {

DecorView draw {

FrameLayout draw {

View draw {

1，绘制背景

2，prepare for fading

3，onDraw // 绘制自身

4，ViewGroup dispatchDraw {

ViewGroup drawChild {

**boolean ViewGroup parent, long drawingTime) {**

**getDisplayList() {**

**private DisplayList getDisplayList(DisplayList displayList, boolean isLayer) {**

**canvas.translate(-mScrollX, -mScrollY); // 子窗口的canvas做clip操作**

**boolean draw(Canvas canvas, ViewGroup parent, long drawingTime)**

**draw(canvas); // 子窗口的绘制**

**setDisplayListProperties(displayList);**

**}**

**}**

**}**

}

}

}

5，draw the fading edges

6，onDrawScrollBars

}

}

}

}

}

}

}

}

## View

### boolean draw(Canvas canvas, ViewGroup parent, long drawingTime) {

### public void draw(Canvas canvas) {

**final** **int** privateFlags = mPrivateFlags;

**final** **boolean** dirtyOpaque = (privateFlags & *PFLAG\_DIRTY\_MASK*) == *PFLAG\_DIRTY\_OPAQUE* &&

(mAttachInfo == **null** || !mAttachInfo.mIgnoreDirtyState);

mPrivateFlags = (privateFlags & ~*PFLAG\_DIRTY\_MASK*) | *PFLAG\_DRAWN*;

… …

// Step 3, draw the content

**if** (!dirtyOpaque) **onDraw(canvas);**

// Step 4, draw the children

**dispatchDraw(canvas);**

// Step 6, draw decorations (scrollbars)

onDrawScrollBars(canvas);

}

### protected void onDraw(Canvas canvas) {

**// 空的**

}

## ViewGroup

### protected void dispatchDraw(Canvas canvas) {

**final** **int** count = mChildrenCount;

**final** View[] children = mChildren;

**int** flags = mGroupFlags;

… …

// 如果存在动画或者可见，绘制子View

if ((child.mViewFlags & VISIBILITY\_MASK) == VISIBLE || child.get ation() != null) {

more |= drawChild(canvas, child, drawingTime);

}

… …

}

### protected boolean drawChild(Canvas canvas, View child, long drawingTime) {

**return** child.draw(canvas, **this**, drawingTime);

}

scrollTo scrollBy 不会改变layout和measure,只会改变窗口Canvas的绘制原点。当原点变为负值时，子窗口会向右下移动。

### protected int getChildDrawingOrder(int childCount, int i) {

**return** i;

}

// 更改child的绘制顺序

# invalidate

## View

public void invalidate(Rect dirty) {

if (skipInvalidate()) {

return;

}

if ((mPrivateFlags & (PFLAG\_DRAWN | PFLAG\_HAS\_BOUNDS)) == (PFLAG\_DRAWN | PFLAG\_HAS\_BOUNDS) ||

(mPrivateFlags & PFLAG\_DRAWING\_CACHE\_VALID) == PFLAG\_DRAWING\_CACHE\_VALID ||

(mPrivateFlags & PFLAG\_INVALIDATED) != PFLAG\_INVALIDATED) {

mPrivateFlags &= ~PFLAG\_DRAWING\_CACHE\_VALID;

mPrivateFlags |= PFLAG\_INVALIDATED;

mPrivateFlags |= PFLAG\_DIRTY;

final ViewParent p = mParent;

final AttachInfo ai = mAttachInfo;

//noinspection PointlessBooleanExpression,ConstantConditions

if (!HardwareRenderer.RENDER\_DIRTY\_REGIONS) {

if (p != null && ai != null && ai.mHardwareAccelerated) {

// fast-track for GL-enabled applications; just invalidate the whole hierarchy

// with a null dirty rect, which tells the ViewAncestor to redraw everything

p.invalidateChild(this, null);

return;

}

}

if (p != null && ai != null) {

final int scrollX = mScrollX;

final int scrollY = mScrollY;

final Rect r = ai.mTmpInvalRect;

r.set(dirty.left - scrollX, dirty.top - scrollY,

dirty.right - scrollX, dirty.bottom - scrollY);

mParent.invalidateChild(this, r);

}

}

}

## ViewGroup

public final void invalidateChild(View child, final Rect dirty) {

ViewParent parent = this;

…

parent = parent.invalidateChildInParent(location, dirty);

if (view != null) {

// Account for transform on current parent

Matrix m = view.getMatrix();

if (!m.isIdentity()) {

RectF boundingRect = attachInfo.mTmpTransformRect;

boundingRect.set(dirty);

m.mapRect(boundingRect);

dirty.set((int) (boundingRect.left - 0.5f),

(int) (boundingRect.top - 0.5f),

(int) (boundingRect.right + 0.5f),

(int) (boundingRect.bottom + 0.5f));

}

}

} while (parent != null);

}

public ViewParent invalidateChildInParent(final int[] location, final Rect dirty) {

…

return mParent;

}

## ViewRoot

public ViewParent invalidateChildInParent(int[] location, Rect dirty) {

checkThread();

…

if (!mWillDrawSoon && (intersected || mIsAnimating)) {

scheduleTraversals();

}

return null;

}

# requestLayout

## View

public void requestLayout() {

if (mMeasureCache != null) mMeasureCache.clear();

if (mAttachInfo != null && mAttachInfo.mViewRequestingLayout == null) {

// Only trigger request-during-layout logic if this is the view requesting it,

// not the views in its parent hierarchy

ViewRootImpl viewRoot = getViewRootImpl();

if (viewRoot != null && viewRoot.isInLayout()) {

if (!viewRoot.requestLayoutDuringLayout(this)) {

return;

}

}

mAttachInfo.mViewRequestingLayout = this;

}

mPrivateFlags |= PFLAG\_FORCE\_LAYOUT;

mPrivateFlags |= PFLAG\_INVALIDATED;

if (mParent != null && !mParent.isLayoutRequested()) {

mParent.requestLayout();

}

if (mAttachInfo != null && mAttachInfo.mViewRequestingLayout == this) {

mAttachInfo.mViewRequestingLayout = null;

}

}

## ViewRoot

@Override

public void requestLayout() {

checkThread();

mLayoutRequested = true;

scheduleTraversals();

}

# @TargetApi and @SuppressLint

@TargetApi and @SuppressLint have the same core effect: they suppress the Lint error.

The difference is that with @TargetApi, you declare, via the parameter, what API level you have addressed in your code, so that the error can pop up again if you later modify the method to try referencing something newer than the API level cited in @TargetApi.

For example, suppose that, instead of blocking the StrictMode complaints about your networking bug, you were trying to work around the issue of AsyncTask being serialized on newer versions of Android. You have a method like this in your code to opt into the thread pool on newer devices and use the default multithread behavior on older devices:

@TargetApi(11)

static public <T> void executeAsyncTask(AsyncTask<T, ?, ?> task,

T... params) {

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

task.executeOnExecutor(AsyncTask.THREAD\_POOL\_EXECUTOR, params);

}

else {

task.execute(params);

}

}

Having @TargetApi(11) means that if Lint detects that I am using something newer than my android:minSdkVersion, but up to API Level 11, Lint will not complain. In this case, that works. If, however, I modified this method to reference something that wasn't added until API Level 14, then the Lint error would appear again, because my @TargetApi(11) annotation says that I only fixed the code to work on API Level 11 and below, not API Level 14 and below.

Using @SuppressLint('NewApi'), I would lose the Lint error for any API level, regardless of what my code references and what my code is set up to handle.

Hence, @TargetApi is the preferred annotation, as it allows you to tell the build tools "OK, I fixed this category of problems" in a more fine-grained fashion.

# 自定义属性

1，定义，在attrs.xml文件中，

<declare-styleable name="SubscribeButtonType">

<attr name="type" format="enum">

<enum name="pic" value="1" />

< enum name="text" value="2" />

< enum name="pictext" value="3" />

</attr>

<attr name="displayOptions" format="flag">

<flag name="useLogo" value="0x1" />

<flag name="showHome" value="0x2" />

</attr>

<attr name="numColumns" format="integer" />

<attr name="horizontalSpacing" format="dimension" />

</declare-styleable>

其中类型可以为boolean, string , integer , dimension , reference , color , enum, flag等。

同一个应用中所有的属性名称不能重复。

2，配置。

xmlns:MyName ="<http://schemas.android.com/apk/res/com.sina.sinavideo>"

或者xmlns: MyName ="http://schemas.android.com/apk/res-auto"

<com.sina.sinavideo.view.SubscribeButton

android:id="@+id/program\_main\_detail\_add"

android:layout\_width="@dimen/dp\_102"

android:layout\_height="@dimen/dp\_94"

android:layout\_alignParentRight="true"

android:layout\_alignParentBottom="true"

MyName:type="text" />

其中MyName可以自定义，用来表示包中自定义的属性。

3，读取。

TypedArray a = context.obtainStyledAttributes(attrs,

R.styleable.SubscribeButtonType, defStyle, 0);

mType = a.getInteger(R.styleable.SubscribeButtonType\_type, TYPE\_PICTEXT);

1. recycle();

# Bitmap BitmapFactory Canvas Paint

## Bitmap

*HONEYCOMB\_MR1*

public final int getByteCount() {

// int result permits bitmaps up to 46,340 x 46,340

return getRowBytes() \* getHeight();

}

// 图像处理，生成一个dstWidth和dstHeight的图像

// 具体是通过Matrix设置比例实现的

// **final** **int** width = src.getWidth();

// **final** **int** height = src.getHeight();

// **final** **float** sx = dstWidth / (**float**)width;

// **final** **float** sy = dstHeight / (**float**)height;

// m.setScale(sx, sy);

**public** **static** Bitmap createScaledBitmap(Bitmap src, **int** dstWidth, **int** dstHeight, **boolean** filter);

// 从图像的x,y位置截图长度为width和height的图像

// 不做任何的缩放处理，仅仅得到长宽为width和height的图像

**public** **static** Bitmap createBitmap(Bitmap source, **int** x, **int** y, **int** width, **int** height);

// 从图像x,y位置截图width和height的图像并做matrix变换

**public** **static** Bitmap createBitmap(Bitmap source, **int** x, **int** y, **int** width, **int** height,

Matrix m, **boolean** filter);

// 创建一个空位图，可以为ARGB\_8888等配置

**public** **static** Bitmap createBitmap(**int** width, **int** height, Config config);

// 获取alpha

**public** Bitmap extractAlpha();

// 压缩图像输出到stream中，可以为JPEG，PNG和WEBP

**public** **boolean** compress(CompressFormat format, **int** quality, OutputStream stream);

## BitmapFactory

主要用来生成位图

// 从文件生成

**public** **static** Bitmap decodeFile(String pathName, Options opts);

// 从资源文件获取

**public** **static** Bitmap decodeResource(Resources res, **int** id, Options opts);

// 从字节数组获取

**public** **static** Bitmap decodeByteArray(**byte**[] data, **int** offset, **int** length, Options opts);

// 从输入流获取

**public** **static** Bitmap decodeStream(InputStream is, Rect outPadding, Options opts);

**BitmapFactory.Options**

inSampleSize长和宽等比例压缩，表示长和宽分别为以前的1/inSampleSize

## Canvas

绘制的画布

// 通过位图构造画布，通过画布修改位图

**public** Canvas(Bitmap bitmap);

// 保存canvas的环境，在绘制时对canvas做过修改后恢复

**public** **native** **int** save();

**public** **native** **void** restore();

**public** **native** **int** getSaveCount();

**public** **native** **void** restoreToCount(**int** saveCount);

// 画布的变换

**public** **native** **void** translate(**float** dx, **float** dy);

**public** **native** **void** scale(**float** sx, **float** sy);

**public** **native** **void** rotate(**float** degrees);

**public** **void** setMatrix(Matrix matrix);

// 画布的绘制

**public** **void** drawRGB(**int** r, **int** g, **int** b);

**public** **void** drawColor(**int** color);

**public** **native** **void** drawPoints(**float**[] pts, **int** offset, **int** count, Paint paint);

**public** **native** **void** drawPoint(**float** x, **float** y, Paint paint);

**public** **void** drawLine(**float** startX, **float** startY, **float** stopX, **float** stopY, Paint paint);

**public** **native** **void** drawLines(**float**[] pts, **int** offset, **int** count, Paint paint);

**public** **void** drawRect(RectF rect, Paint paint);

**public** **void** drawCircle(**float** cx, **float** cy, **float** radius, Paint paint);

**public** **void** drawArc(RectF oval, **float** startAngle, **float** sweepAngle, **boolean** useCenter, Paint paint);

**public** **void** drawRoundRect(RectF rect, **float** rx, **float** ry, Paint paint)

**public** **void** drawPath(Path path, Paint paint);

**public** **void** drawBitmap(Bitmap bitmap, **float** left, **float** top, Paint paint);

**public** **void** drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint);

// y字符baseline位置top-paint.asent()

**public** **void** drawText(String text, **float** x, **float** y, Paint paint);

## Paint

画笔画刷TextPaint

// 设置边缘的宽度

**public** **native** **void** setStrokeWidth(**float** width);

// 设置绘制的样式FILL，STROKE，FILL\_AND\_STROKE，填充或者空心

**public** **void** setStyle(Style style);

// 设置颜色

**public** **native** **void** setColor(**int** color);

// 设置抗锯齿模式

**public** **native** **void** setAntiAlias(**boolean** aa);

**public** **native** **void** setDither(**boolean** dither);

**public** **native** **void** setFilterBitmap(**boolean** filter);

**public native void** setAlpha**(int a);**

**// Canvas canvas = new Canvas(bitmap);**

**// Paint paint = new Paint();**

**// paint.setFilterBitmap(true);**

// 绘制模式，当为SRC\_IN时只绘制在Dest中SRC的部分

**public** Xfermode setXfermode(Xfermode xfermode);

paint.setXfermode(**new** PorterDuffXfermode(Mode.*SRC\_IN*));

// 计算字符串使用Paint绘制占用的长度

**public float measureText(String text);**

// 文字距离baseline的值

**public** **native** **float** ascent();

**public** **native** **float** descent();

**public** **native** **void** setTextSize(**float** textSize);

# Permission

<permission  android:description="string resource"

android:icon="drawable resource"

        android:label="string resource"

        android:name="string"

        android:permissionGroup="string"

        android:protectionLevel=["normal" | "dangerous" |

                                     "signature" | "signatureOrSystem"] />

声明权限：

<**permission**  android:name="com.example.testbutton.RECEIVE"

android:protectionLevel="signature" />

声明权限限制：

<receiver

android:name="com.example.testbutton.TestButtonReceiver"

**android:permission**="com.example.testbutton.RECEIVE" >

  <intent-filter>

     <action android:name="com.test.action" />

    </intent-filter>

</receiver>

使用权限：（如果权限声明时signature，那么需要签名相同的应用才能使用此权限）

<**uses-permission** android:name="com.example.testbutton.RECEIVE" />

# Intent PendingIntent

构造Action：

1. 使用Action；

intent.**setAction**("");

1. 直接使用类的Class；

intent.**setClass**(Context packageContext, Class<?> cls);

内部会构造出一个ComponentName:

**public** Intent setClass(Context packageContext, Class<?> cls) {

mComponent = **new** ComponentName(packageContext, cls);

**return** **this**;

}

1. 使用Component

intent.**setComponent**(new ComponentName(String pkg, String cls));

启动组件：

**public** **abstract** **void** **startActivity**(Intent intent);

**public** **abstract** ComponentName **startService**(Intent service);

**public** **abstract** **boolean** **bindService**(Intent service, ServiceConnection conn, **int** flags);

**public** **abstract** **void** **sendBroadcast**(Intent intent);

**public** **abstract** **void** **sendBroadcast**(Intent intent, String **receiverPermission**);

PendingIntent:

**public** **static** PendingIntent **getService**(Context context, **int** requestCode,

Intent intent, **int** flags)

**public** **static** PendingIntent **getActivity**(Context context, **int** requestCode,

Intent intent, **int** flags)

**public** **static** PendingIntent **getBroadcast**(Context context, **int** requestCode,

Intent intent, **int** flags)

# 混淆:

**1,保证类名不被混淆**

-keep public class \* extends android.app.Activity

**2,保证类中指定的成员变量和方法不被混淆**

-keepclasseswithmembernames class \* {

native <methods>;

}

**3,保证类中所有的方法不被混淆**

-keep class \* implements java.io.Serializable {

\*;

}

**4,保证包中所有的类都不混淆**

-keep class com.sina.sinavideo.sdk.\*\* {

\*;

}

# Animation的原理

1. **父窗口调用子窗口的boolean draw(Canvas canvas, ViewGroup parent, long drawingTime) {}**
2. **如果动画不为空，获取动画的Transformation。**

**final** Animation a = getAnimation();

**if** (a != **null**) {

more = drawAnimation(parent, drawingTime, a, scalingRequired);

concatMatrix = a.willChangeTransformationMatrix();

**if** (concatMatrix) {

mPrivateFlags3 |= *PFLAG3\_VIEW\_IS\_ANIMATING\_TRANSFORM*;

}

transformToApply = parent.getChildTransformation();

}

1. **在drawAnimation中调用动画的getTransformation。**

**private** **boolean** drawAnimation(ViewGroup parent, **long** drawingTime,

Animation a, **boolean** scalingRequired) {

Transformation invalidationTransform;

**final** **int** flags = parent.mGroupFlags;

**final** **boolean** initialized = a.isInitialized();

**if** (!initialized) {

…

onAnimationStart();

}

**final** Transformation t = parent.getChildTransformation();

**boolean** more = a.getTransformation(drawingTime, t, 1f);

…

// 刷新并进行下一次的显示

parent.invalidate(left, top, left + (**int**) (region.width() + .5f),

top + (**int**) (region.height() + .5f));

…

}

1. **在Animation的getTransformation中调用applyTransformation**

**public** **boolean** getTransformation(**long** currentTime, Transformation outTransformation) {

…

**final** **float** interpolatedTime = Interpolator.getInterpolation(normalizedTime);

**applyTransformation**(interpolatedTime, outTransformation);

…

}

1. **通过获取的Transformation修改Canvas**

canvas.translate(-transX, -transY);

canvas.concat(transformToApply.getMatrix());

canvas.translate(transX, transY);

# ShapeDrawable

使用xml来配置矢量的资源，其中shape的类型可以为rectangle，oval，line，ring。

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<shape xmlns:android=*"http://schemas.android.com/apk/res/android"*

android:shape=*"rectangle"* >

<corners android:radius=*"10dp"*/>

<padding android:left=*"10dp"* />

<size android:width=*"10dp"* />

<stroke android:width=*"10dp"* android:color=*"#000000"*/>

<solid android:color=*"#00ff00"* />

<gradient android:startColor=*"#ff0000"*

android:centerColor=*"#00ff00"*

android:endColor=*"#0000ff"*/>

</shape>