**ПОНИМАНИЕ 2D РИСОВАНИЯ В ANDROID**

Android поставляется вместе с мощными библиотеками API с открытым исходным кодом, которые поддерживают настраиваемую 2D и 3D графику в дополнение к анимации.

API-интерфейсы платформы Android также предоставляют набор API-интерфейсов для 2D-рисования, которые дают вам возможность настраивать графику на холсте или изменять текущие представления для изменения их внешнего вида и внешнего вида.

При рисовании 2D-графики вы будете делать это двумя способами. API делает доступными двухмерные API рисования для простой анимации, которая не нуждается в изменениях ключевых изменений. Эти два способа сделать это с помощью API:

**Рисовать в виде**

**Рисовать на холсте**

**РИСОВАНИЕ КРУГА ДЛЯ ПРОСМОТРА**

Рисование для просмотра является предпочтительным вариантом, когда ваш пользовательский интерфейс не требует динамических изменений в приложении. Наиболее подходящим аспектом для этого является то, что платформа Android предоставит вам предопределенный Canvas, на который вы будете выполнять вызовы рисования.

Это может быть выполнено просто путем расширения категории View и определения метода обратного вызова onDraw ().

Внутри onDraw () вашего компонента View используйте Canvas, предлагаемый вам для каждого рисования, используйте множество методов Canvas.draw… () (например: canvas.drawCircle (x / 2, y / 2, radius, покрасить);). onDraw () - это метод обратного вызова, вызываемый при первом отображении представления.

**РИСОВАНИЕ НА ХОЛСТЕ**

Это предпочтительный вариант, когда ваше приложение требует постоянно перерисовывать себя. Такие приложения, как видеоигры, должны рисовать на холсте самостоятельно. Хотя есть и другие способы достижения этого.

Пример, приведенный ниже, наглядно иллюстрирует код, показывающий, как нарисовать круг, как упоминалось ранее.

**import** android.app.Activity;

**import** android.content.Context;

**import** android.graphics.Canvas;

**import** android.graphics.Color;

**import** android.graphics.Paint;

**import** android.os.Bundle;

**import** android.view.View;

**public** **class** **MainActivity** **extends** **Activity** {

/\*\* Called when the activity is first created. \*/

@Override

**public** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(**new** MyView(**this**));

}

**public** **class** **MyView** **extends** **View** {

**public** **MyView**(Context context) {

**super**(context);

// TODO Auto-generated constructor stub

}

@Override

**protected** **void** **onDraw**(Canvas canvas) {

// TODO Auto-generated method stub

**super**.onDraw(canvas);

**int** x = getWidth();

**int** y = getHeight();

**int** radius;

radius = 100;

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

paint.setStyle(Paint.Style.FILL);

paint.setColor(Color.WHITE);

canvas.drawPaint(paint);

// Use Color.parseColor to define HTML colors

paint.setColor(Color.parseColor("#CD5C5C"));

canvas.drawCircle(x / 2, y / 2, radius, paint);

}

}

}

**КАК РИСОВАТЬ 2D-ОБЪЕКТЫ НА ХОЛСТЕ**

Чтобы рисовать 2D-графику в том месте, где требуется приложение, для которого необходимо постоянно перерисовывать себя, лучшим вариантом для вас является рисование на холсте. Canvas функционирует как интерфейс к реальной поверхности, на которой будет нарисована ваша графика.

Если вам необходимо создать новый Canvas, вам следует указать растровое изображение, на котором будет отображаться рисунок. Растровое изображение всегда необходимо для холста.

Код ниже иллюстрирует способ рисования прямоугольника на холсте, как указано выше.

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

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

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

android:id="@+id/rect">

</**LinearLayout**>

**import** android.app.Activity;

**import** android.graphics.Bitmap;

**import** android.graphics.Canvas;

**import** android.graphics.Color;

**import** android.graphics.Paint;

**import** android.graphics.drawable.BitmapDrawable;

**import** android.os.Bundle;

**import** android.view.Menu;

**import** android.widget.LinearLayout;

**public** **class** **MainActivity** **extends** **Activity** {

@Override

**public** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

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

paint.setColor(Color.parseColor("#CD5C5C"));

Bitmap bg = Bitmap.createBitmap(480, 800, Bitmap.Config.ARGB\_8888);

Canvas canvas = **new** Canvas(bg);

canvas.drawRect(50, 50, 200, 200, paint);

LinearLayout ll = (LinearLayout) findViewById(R.id.rect);

ll.setBackgroundDrawable(**new** BitmapDrawable(bg));

}

@Override

**public** **boolean** **onCreateOptionsMenu**(Menu menu) {

// Inflate the menu; this adds items to the action bar if it is present.

getMenuInflater().inflate(R.menu.activity\_main, menu);

**return** **true**;

}

}

**ВВОДИМЫЙ КОЭФФИЦИЕНТ**

Android предоставляет настраиваемые файлы 2D-графики для рисования фигур и изображений. Файл android.graphics.drawable - это место, где можно найти обычные категории, используемые для рисования в двух измерениях.

Мы предоставили здесь основы использования Drawable объектов для рисования графики и как использовать несколько подклассов категории Drawable.

Drawable - это обычная абстракция для «того, что можно нарисовать». Вы обнаружите, что категория Drawable расширяется, чтобы определить множество конкретных форм рисованной графики, которая состоит из BitmapDrawable, ShapeDrawable, PictureDrawable, LayerDrawable и многих других. Вы также можете расширить их, чтобы указать свои собственные настраиваемые объекты Drawable, которые действуют определенным образом.

Существует три способа указать и инициировать Drawable: с помощью изображения, сохраненного в ресурсах вашего проекта; посредством использования файла XML, который определяет черты Drawable; или конструкторы стандартной категории.

**ГЕНЕРАЦИЯ ИЗ ИЗОБРАЖЕНИЙ РЕСУРСОВ**

Простой способ добавить графику в ваше приложение - обратиться к файлу изображения из ресурсов вашего проекта.

Поддерживаются следующие типы файлов: PNG (наиболее предпочтительный вариант), JPG (приемлемый вариант) и GIF (не следует использовать вообще). Этот метод явно предпочтителен для значков приложений, логотипов или другой графики, подобной тем, которые используются в игре.

Чтобы использовать ресурс изображения, вам просто нужно включить ваш файл в каталог res / drawable / вашего проекта.

Вы можете ссылаться на него из своего кода или вашего XML-макета. Какой бы вариант вы ни выбрали, это называется использованием идентификатора ресурса, то есть имени файла без расширения с расширением типа файла, такого как my\_image.png, которое называется my\_image.

**ПРИМЕР КОДА**

Приведенный ниже код иллюстрирует способ создания представления изображения, которое использует изображение из доступных для рисования ресурсов и включает его в макет.

LinearLayout mLinearLayout;

**protected** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

// Create a LinearLayout in which to add the ImageView

mLinearLayout = **new** LinearLayout(**this**);

// Instantiate an ImageView and define its properties

ImageView i = **new** ImageView(**this**);

i.setImageResource(R.drawable.my\_image);

i.setAdjustViewBounds(**true**); // set the ImageView bounds to match the Drawable's dimension

i.setLayoutParams(**new** Gallery.LayoutParams(LayoutParams.WRAP\_CONTENT,

LayoutParams.WRAP\_CONTENT));

// Add the ImageView to the layout and set the layout as the content view

mLinearLayout.addView(i);

setContentView(mLinearLayout);

}

В других сценариях вы можете позаботиться о своем ресурсе изображения как объект Drawable. Чтобы достичь этого, создайте Drawable из ресурса, такого как:

**Resources** res = mContext.getResources();

**Drawable** myImage = res.getDrawable(R.drawable.my\_image);

Каждый отдельный ресурс в вашем проекте может поддерживать только уникальное состояние, независимо от количества различных объектов, которые вы можете инициировать для него.

Например, если вы инициируете два объекта Drawable из эквивалентного ресурса изображения, а затем измените свойство (например, альфа) для одного из Drawables, то это также повлияет на другой.

Таким образом, каждый раз, когда вы обрабатываете множество примеров ресурса изображения, вместо того, чтобы неуклонно менять Drawable, вы должны выполнять анимацию анимации.

**Пример XML**

Приведенный ниже код XML иллюстрирует, как добавить ресурс Drawable в ImageView в макете XML (с небольшим красным оттенком, просто для развлечения).

<**ImageView**

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:tint="#55ff0000"

android:src="@drawable/my\_image"/>

**СОЗДАНИЕ ИЗ РЕСУРСА XML**

На этом этапе вы должны иметь возможность создавать пользовательский интерфейс. Поэтому вы должны знать силу и гибкость, присущие указанию объектов в XML.

Это переносится из представлений в Drawables. Если есть объект Drawable, который вы хотели бы создать, который вначале не зависит от переменных, указанных в коде вашего приложения или при взаимодействии с пользователем, то указание Drawable в XML является отличным вариантом.

Даже если вы ожидаете, что ваш Drawable изменит свои функции во время работы пользователя с вашим приложением, вы должны принять во внимание спецификацию объекта в XML, поскольку вы всегда можете изменять свойства сразу же после его запуска.

Сразу же вы указали свой Drawable в XML, сохраните файл в каталоге res / drawable / вашего проекта и после этого получите и инициируйте объект, вызвав Resources.getDrawable (), передав ему идентификатор ресурса вашего XML-файла.

Любая подкатегория Drawable, поддерживающая метод inflate (), может быть указана в XML и запущена вашим приложением. Каждый Drawable, который поддерживает XML-инфляцию, использует определенные XML-характеристики, которые помогают вам определять свойства объекта. См. Документацию по категориям для каждой подкатегории Drawable для получения информации о том, как определить ее в XML.

**пример**

Ниже приведены несколько XML, которые определяют TransitionDrawable:

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

<**item** android:drawable="@drawable/image\_expand">

<**item** android:drawable="@drawable/image\_collapse">

</**transition**>

С этим XML-файлом, сохраненным в файле res / drawable / expand\_collapse.xml, код запускает TransitionDrawable и устанавливает его как содержимое ImageView:

Resources res = mContext.getResources();

TransitionDrawable transition = (TransitionDrawable)

res.getDrawable(R.drawable.expand\_collapse);

ImageView image = (ImageView) findViewById(R.id.toggle\_image);

image.setImageDrawable(transition);

At **this** point the transition can be run forward (**for** 1 second) **with**:

transition.startTransition(1000);

**ФОРМА DRAWABLE**

Каждый раз, когда вы собираетесь рисовать несколько 2D-графики динамически, вам может понадобиться объект ShapeDrawable, чтобы достичь этого.

ShapeDrawable, позволяет вам рисовать как программу первозданные фигуры и проектировать их любым удобным для вас способом.

ShapeDrawable - это расширение Drawable, которое позволяет использовать его везде, где должен использоваться Drawable, как для фона вида, установленного с помощью setBackgroundDrawable ().

Конечно, вы также можете нарисовать свою фигуру в виде собственного настраиваемого вида, который будет включен в ваш макет, независимо от того, как вам нравится.

В связи с тем, что ShapeDrawable обладает своей собственной техникой draw (), вы можете создать подкатегорию View, которая рисует ShapeDrawable во время техники View.onDraw ().

Ниже приведено основное расширение категории «Вид», которая рисует ShapeDrawable в качестве представления:

**public** **class** **CustomDrawableView** **extends** **View** {

**private** ShapeDrawable mDrawable;

**public** **CustomDrawableView**(Context context) {

**super**(context);

**int** x = 10;

**int** y = 10;

**int** width = 300;

**int** height = 50;

mDrawable = **new** ShapeDrawable(**new** OvalShape());

mDrawable.getPaint().setColor(0xff74AC23);

mDrawable.setBounds(x, y, x + width, y + height);

}

**protected** **void** **onDraw**(Canvas canvas) {

mDrawable.draw(canvas);

}

}

В конструкторе ShapeDrawable указывается как OvalShape. Затем предлагается цвет и пределы формы устанавливаются. Если вы не установите ограничения, то форма не будет нарисована, в то время как если вам не удастся установить цвет, он по умолчанию изменится на черный.

С указанным настроенным видом, его можно нарисовать в любой форме, которая вам нравится. На примере выше мы можем нарисовать фигуру в виде программы в Activity:

CustomDrawableView mCustomDrawableView;

**protected** **void** **onCreate**(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

mCustomDrawableView = **new** CustomDrawableView(**this**);

setContentView(mCustomDrawableView);

}

Если вы хотите нарисовать этот настраиваемый объект рисования из макета XML, а не из Activity, то категория CustomDrawable должна переопределить конструктор View (Context, Character Set), который вызывается во время запуска View через инфляцию из XML. После этого включите CustomDrawable фактор в XML, такой как:

<**com.example.shapedrawable.CustomDrawableView**

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"/>

Категория ShapeDrawable, например, множество других типов Drawable в пакете android.graphics.draw + able, позволяет указывать множество свойств объекта Drawable с помощью общедоступных методов.

Вот некоторые свойства, которые вы можете изменить: альфа-прозрачность, цветовой фильтр, сглаживание, непрозрачность и цвет.

Вы также можете указать первичные нарисованные формы с использованием XML.

**NINE-PATCHDRAWABLE ГРАФИКА**

Графический объект NinePatchDrawable - это растровое изображение, которое можно растянуть, и которое Android будет регулярно корректировать, чтобы оно содержало содержимое представления, в которое вы поместили его в качестве фона.

Одним из примеров, который показывает использование NinePatch, являются фоны, используемые типичными кнопками Android - кнопки должны растягиваться, чтобы содержать строки различной длины.

Инструмент Draw 9-patch представляет исключительно практичный способ создания изображений NinePatch с использованием графического редактора WYSIWYG. Он даже увеличивает предупреждения, если область, указанная для расширяемой области, подвержена риску создания артефактов рисования из-за дублирования пикселей.

Пример XML

Ниже приведен пример экземпляра XML-макета, в котором показано, как добавить изображение NinePatch в группу кнопок. Изображение NinePatch хранится в форме res / drawable / my\_button\_background.9.png

<Button **id**="@+id/tiny"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentTop="true"

android:layout\_centerInParent="true"

android:text="Tiny"

android:textSize="8sp"

android:background="@drawable/my\_button\_background"/>

<Button **id**="@+id/big"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:layout\_alignParentBottom="true"

android:layout\_centerInParent="true"

android:text="Biiiiiiig text!"

android:textSize="30sp"

android:background="@drawable/my\_button\_background"/>

Обратите внимание, что ширина и высота установлены на «wrap\_content», чтобы убедиться, что кнопка точно соответствует тексту.