2.9. Графические ресурсы, стили и темы Android приложения. Ресурсы для адаптивных макетов

Сайт:	Samsung Innovation Campus	Напечатано::	Murad Rezvan
Курс:	Мобильная разработка на Kotlin	Дата:	понедельник, 3 июня 2024, 17:47
Книга:	2.9. Графические ресурсы, стили и темы Android приложения. Ресурсы для адаптивных макетов		

Оглавление

2.9.1. Графические ресурсы Android-приложений

Введение

Для статических изображений в приложениях можно задействовать класс Drawable и его подклассы для. `Drawable` - это общая абстракция для рисования. Различные его подклассы помогают в определенных ситуациях. Их можно расширить, чтобы задать собственные объекты рисования, которые ведут себя специальным образом.

Существует два способа создания экземпляра класса `Drawable` (кроме использования конструкторов):

- Обратиться к ресурсу изображения в проекте.
- Использовать ресурс XML, который определяет свойства для рисования.

Кроме растровых изображений возможно использование векторного рисования, что позволяет масштабировать рисунки для разных размеров без потери качества.

Создание изображений из ресурсов

В Android-приложения возможно добавить графику в ваше приложение, ссылаясь на файл изображения из ресурсов проекта. При этом поддерживаются следующие типы:

- PNG (предпочтительно);
- JPG (приемлемо);
- GIF (не рекомендуется).

Таким образом удобно хранить иконки приложений, логотипы и т.п. Чтобы задействовать изображения из ресурсов, нужно добавить картинку в каталог *res/drawable/*, а затем в коде приложения можно ссылаться на графический ресурс файла без расширения. Если рисунок назывался *test.png*, то код может выглядеть следующим образом:

```
val myImage: Drawable = ResourcesCompat.getDrawable(context.resources, R.drawable.test, null)
```

Следующий фрагмент кода демонстрирует, как создать `ImageView`, который использует изображение из ресурсов и размещает его в разметку:

```
private lateinit var constraintLayout: ConstraintLayout

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Создание экземпляра ImageView и определение его свойств
    val i = ImageView(this).apply {
        setImageResource(R.drawabletest)
        contentDescription = resources.getString(R.string.test_desc)

        //устанавливаем границы ImageView
        adjustViewBounds = true
        layoutParams = ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.WRAP_CONTENT,
                ViewGroup.LayoutParams.WRAP_CONTENT)
    }

    // Создаем ConstraintLayout, в который нужно добавить ImageView
    constraintLayout = ConstraintLayout(this).apply {

        // Добавляем ImageView в разметку.
        addView(i)
    }

    // Устанавливаем разметку
    setContentView(constraintLayout)
}
```

Далее приведен фрагмент XML-кода, демонстрирующий добавление ресурса ImageView в разметку XML:

```
<ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/test"
        android:contentDescription="@string/test_desc" />
```

При использовании графических ресурсов следует убедиться, что изображения имеют подходящий размер в пикселях. Для избежания подобного рода проблем следует ознакомится со следующими материалами.

Более подробную информацию об использовании графических ресурсов можно найти в официальной документации. Кроме указанного выше примера в приложениях возможно рисунков из XML-ресурсов, использование форм для рисунков (Shape Drawables), использование растровых изображений, которое вы можете использовать в качестве фона представления (NinePatch drawables) и др.

2.9.2. Стили и темы Android приложения

Стили и темы в Android позволяют определить дизайн приложения и отделить детали дизайна от его логики и пользовательского интерфейса. **Стиль** - это набор атрибутов, которые определяют внешний вид представлений. Можно задавать такие атрибуты, как цвет шрифта, размер шрифта, цвет фона и многое другое. **Тема** приложения - это стиль, который применяется ко всему приложению, а не только к отдельному элементу пользовательского интерфейса. Стили и темы объявляются в файле ресурсов стилей в *res/values /*, обычно называемом *styles.xml*.

Далее на рисунке представлены две типичные темы для одной активности: `Theme.AppCompat` (слева) и `Theme.AppCompat.Light` (справа)

рисунок

Создание и применение стилей

Чтобы создать собственный стиль или тему, необходимо открыть файл *res/values/styles.xm*. Далее следует выполнить следующие действия:

- добавьте элемент `<style>` с именем, однозначно определяющим стиль;
- добавьте элемент `<item>` для каждого атрибута стиля, который вы хотите определить.

Например, можно определить следующий стиль:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="RedText" parent="TextAppearance.AppCompat">
        <item name="android:textColor">#FF0000</item>
    </style>
</resources>
```

Теперь можно применить стиль к представлению следующим образом:

```
<Button
    style="@style/RedText"
    ... />
```

Атрибуты, указанные в стиле, применяется к представлениям, если это возможно.

Настройка стилей

При создании собственных стилей необходимо расширять существующие чтобы поддерживать совместимость со стилями пользовательского интерфейса платформы. Для расширения стиля укажите стиль, который вы хотите расширить, с помощью родительского атрибута. Затем можно переопределить унаследованные атрибуты стиля и добавить новые.

Например, можно унаследовать внешний вид текста платформы Android по умолчанию и изменить его следующим образом:

```
<style name="RedText" parent="@android:style/TextAppearance">
    <item name="android:textColor">#FF0000</item>
</style>
```

Ваши приложения могут наследовать стили из стандартной библиотеки Android. Это стало возможно начиная с версии Android 4.0 и выше. Например следующим образом.

```
<style name="RedText" parent="TextAppearance.AppCompat">
    <item name="android:textColor">#FF0000</item>
</style>
```

Можно расширять собственные стили. Например, следующий стиль перенимает все настройки собственного стиля RedText при этом увеличивая размер текста:

```
<style name="GreenText.Large">
    <item name="android:textSize">22dp</item>
</style>
```

Таким образом можно строить менять свои стили, строя из них своеобразную "цепочку".

Применение стилей как тему

В Android так же как и стили, можно создавать и темы. Отличие заключается в том, что тема будет применена не к отдельному представлению а к целому приложению или отдельно взятой Активности. Тогда, в случае если необходимо применить темную тему для всего приложения в файле `AndroidManifest.xml.` следует прописать следующее:

```
<manifest ... >
    <application android:theme="@style/Theme.AppCompat" ... >
    </application>
</manifest>
```

А если светлую:

```
<manifest ... >
    <application ... >
        <activity android:theme="@style/Theme.AppCompat.Light" ... >
        </activity>
    </application>
</manifest>
```

Теперь каждое представление в приложении или Активности будет в едином стиле. Полную информацию по выбору стилей можно посмотреть на сайте developer.android.com/

2.9.3. Ресурсы для адаптивных макетов

Введение

В системе Android экраны устройств в простейшем случае классифицируются по размеру и плотности. При разработке приложений вы должны учесть тот факт, что ваше приложение будет запускаться на разнообразных устройствах с различными экранами. Что бы избежать размытых изображений, "пиксельных" картинок и других проблем, в приложении должны находится альтернативные ресурсы, которые оптимизирующие интерфейс вашего приложения для экранов различных размеров и плотности.

Существует четыре обобщенных размера: маленький (small), нормальный (normal), большой (large), очень большой (xlarge) и четыре обобщенные плотности: низкая (ldpi), средняя (mdpi), высокач (hdpi), сверхвысокая (xhdpi).

Для использования различных изображений для разных экранов необходимо поместить эти изображения в альтернативные ресурсы, которые располагаются в отдельных каталогах (подобно тому, как вы мы использовали различные строки для разной локализации приложения).

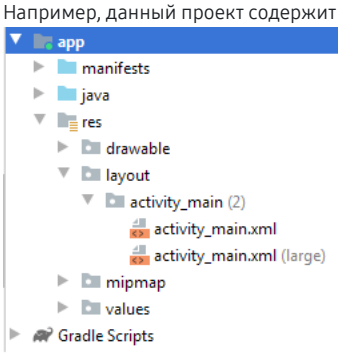
Кроме этого необходимо иметь ввиду, что ориентация экрана (альбомная или книжная) - это изменение размера экрана, поэтому многие приложения должны использовать адаптивные ресурсы при смене ориентации для оптимизации работы пользователей.

Создание различных макетов

С целью оптимизации работы пользователей с различными размерами экрана, необходимо создать уникальный XML-файл макета для каждого размера, который вы хотите поддерживать. Каждый макет должен быть сохранен в папке соответствующих ресурсов, названный с суффиксом размера экрана. Например, макет для больших экранов должен быть сохранен в папке res/layout-large/.

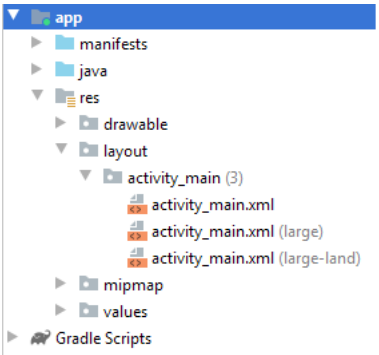
Android автоматически масштабирует ваш макет. Таким образом, не нужно беспокоиться об абсолютных размерах элементов пользовательского интерфейса, Вместо этого целесообразно сосредоточится на структуре макета (размере или положении элементов интерфейса по отношению друг к другу).

Например, данный проект содержит в себе макет по умолчанию и альтернативный макет экран больших размеров:



При этом имена файлов должны быть такие же, но их содержание отличается в целях обеспечения оптимизации интерфейса для соответствующего экрана. В зависимости от экрана система сама выберет макет, который нужно использовать. Более подробную информацию о том, как в выбирается соответствующий ресурс можно найти здесь.

Для того, чтобы обеспечить специальный макет для альбомной ориентации, в том числе и для больших экранов, то необходимо использовать спецификаторы large и land :



Начиная с версии Android 3.2, поддерживается усовершенствованный метод идентификации размеров экрана. Это позволяет указать ресурсы для размеров экрана на основе минимальной ширины и высоты в независимых от плотности пикселях. Подробнее можно прочитать по ссылке.

Создание различных растровых изображений

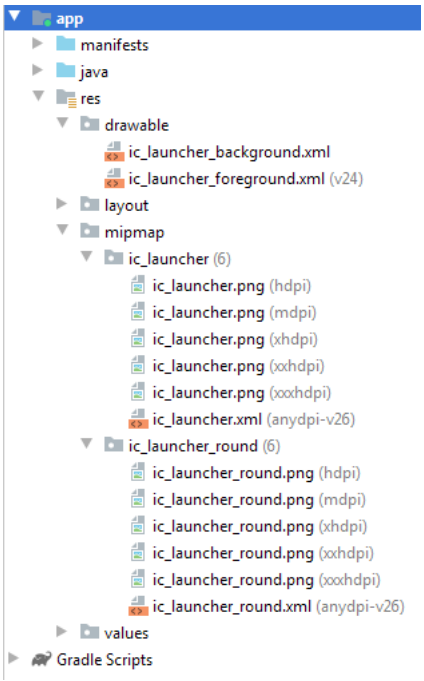
Вы всегда должны предоставлять растровые ресурсы, которые корректно масштабируются для каждой из обобщенных плотностей: низкой, средней, высокой и сверхвысокой плотности. Это поможет вам достичь хорошего графического качества и производительности на всех плотностях экрана. Для создания этих изображений, вы должны начать с ресурса в векторном формате и генерировать изображения для каждой плотности, используя следующую шкалу размеров:

- xhdpi: 2.0

- hdpi: 1.5
- mdpi: 1.0 (базовый)
- ldpi: 0.75

Это означает, что если вы создаете изображение 200×200 для xhdpi устройства, необходимо генерировать тот же ресурс в 150×150 для hdpi, 100×100 для mdpi и 75×75 для ldpi устройств.

Затем поместите файлы в соответствующий каталог рисуемых ресурсов:



Каждый раз, когда вы ссылаетесь @drawable/awesomeimage, система выбирает соответствующее растровое изображение на основе плотности экрана.

Ресурсы низкой плотности (ldpi) не всегда необходимы. Когда вы предоставляете hdpi наборы, система масштабирует их на половину, чтоюы соответствовать должным образом экранам ldpi.

Упражнение 2.9

Разработаем приложение с адаптивной разметкой. В приложении будет присутствовать два файла *activity_main.xml* для портретной и ландшафтной ориентации. В каждом файле разметки есть `ImageView` с ShapeDrawable, заданной через xml (*ic_rounded_rectangle.xml*). Положение элементов на экране будет разным в зависимости от ориентации.

Сначала в папке `drawable` создадим файл *ic_rounded_rectangle.xml*. Его код будет выглядеть следующим образом.

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <size
        android:width="100dp"
        android:height="100dp" />
    <solid android:color="#00AA00"/>
    <corners android:radius="10dp"/>

</shape>
```

В итоге получится зеленый квадрат с закругленными углами.

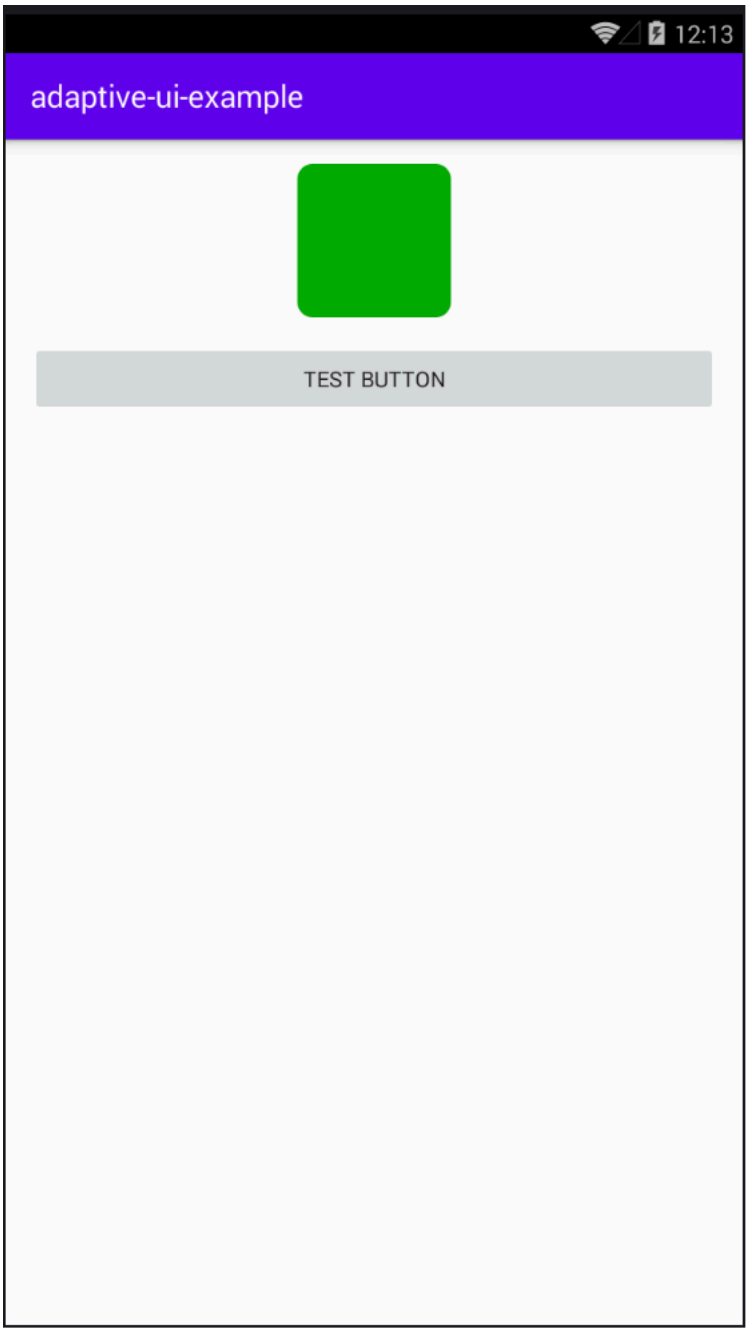
Теперь необходимо создать файлы разметки. Для файла *activity_main.xml* код будет следующим:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imgShape"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_rounded_rect"
        android:layout_margin="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:text="Test button"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/imgShape"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

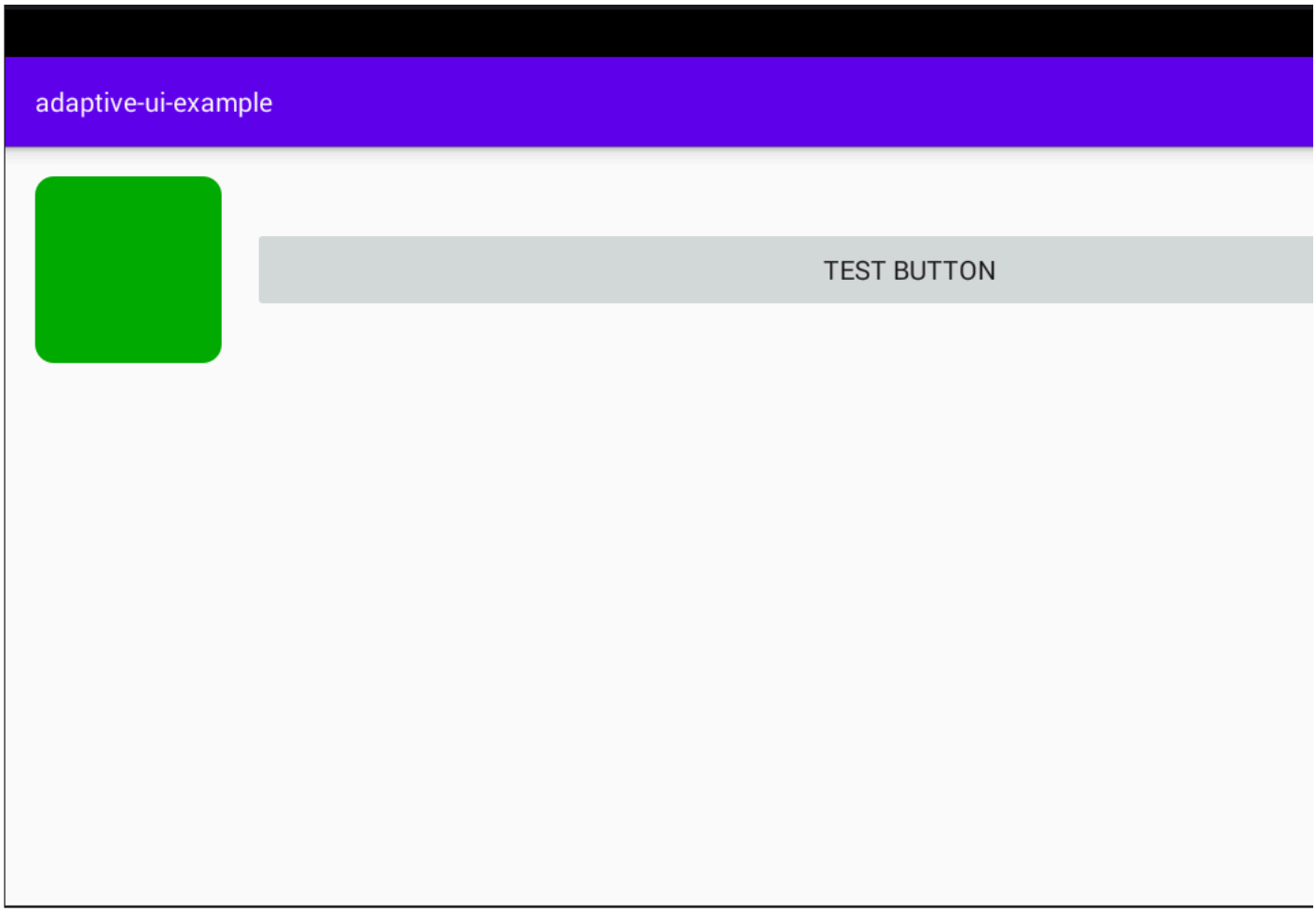
Создадим файл разметки для ландшафтной ориентации *activity_main.xml (land)* и заполним следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imgShape"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:src="@drawable/ic_rounded_rect"
        android:layout_margin="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:text="Test button"
        app:layout_constraintStart_toEndOf="@id/imgShape"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="@id/imgShape"
        app:layout_constraintBottom_toBottomOf="@id/imgShape"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

После запуска приложения в горизонтальной ориентации экран будет иметь следующий вид:



В ландшафтной ориентации расположение элементов будет другим:



Таким образом в созданном приложении виджеты адаптируются к ориентации экрана.

Начать тур для пользователя на этой странице