<TeachMeSkills/>

курс

Android разработчик

Создание собственных View

Агенда занятия:

Понятие CustomView

Жизненный цикл View

Invalidate/requestLayout

Canvas

- **Понятие CustomView**
  Жизненный цикл View
  Invalidate/requestLayout
  Canvas

Что такое CustomView

CustomView — это пользовательский компонент, унаследованный от View или ViewGroup, с кастомной отрисовкой и/или поведением.

Примеры: круглый прогресс-бар, график, индикатор батареи.

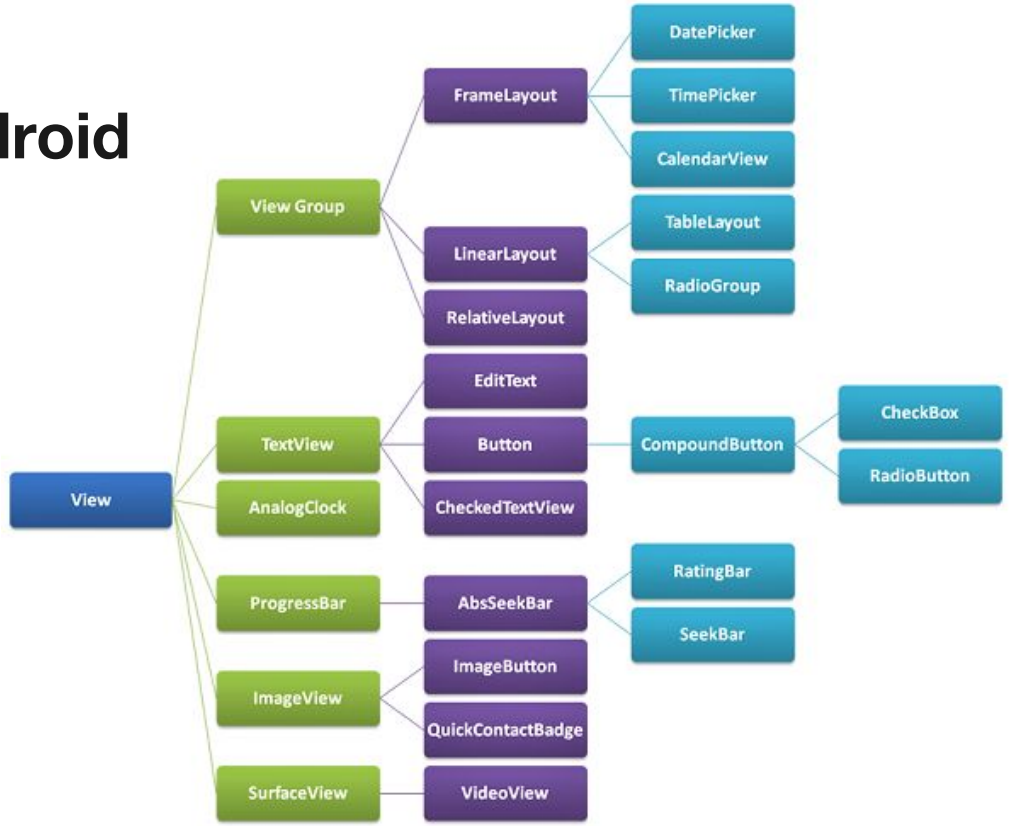
Когда использовать CustomView

Если стандартных компонентов недостаточно.

Для оптимизации или создания уникального UI.

```
class CustomView(context: Context?) : View(context) {

    override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec)
    }

    override fun onLayout(changed: Boolean, left: Int, top: Int, right: Int, bottom: Int) {
        super.onLayout(changed, left, top, right, bottom)
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
    }
}
```

Иерархия View в Android



Конструкторы View

Создание View начинается с конструктора с различными параметрами: Context, AttributeSet, defStyleAttr и defStyleRes. View имеет четыре конструктора, и вам нужно будет переопределить хотя бы один из них.

Конструкторы View

1. Конструктор с одним параметром — **Context.**
Используется только в том случае, если View мы хотим создавать из кода, а не из XML.

2. Конструктор с двумя параметрами — **Context и AttributeSet.**
Используется для создания View с использованием XML-макета. В этом конструкторе можно получить значения атрибутов View, указанных в XML-разметке, и использовать их для настройки свойств вашего Custom View.

Конструкторы View

3. Конструктор с тремя параметрами — **Context, AttributeSet и defStyleAttr.**
Вызывается при создании View с помощью XML-разметки и задании значения стиля (defStyleAttr) из темы.

4. Конструктор с четырьмя параметрами — **Context, AttributeSet, defStyleAttr и defStyleRes.**
Также используется для создания View с использованием XML-макета, со стилем из темы и/или с ресурсом стиля.

Понятие CustomView

▶ **Жизненный цикл View**

Invalidate/requestLayout

Canvas

Жизненный цикл View

Особенности:

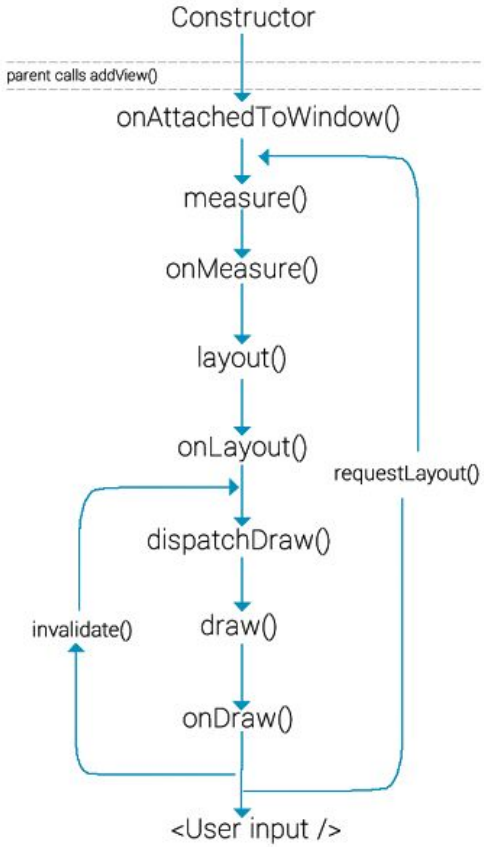
onMeasure() – расчёт размера компонента.

onLayout() – размещение дочерних View (в ViewGroup).

onDraw() – отрисовка на Canvas.

Важно:

Не забывать вызывать super в большинстве переопределённых методов.



onMeasure()

Конечная цель метода **onMeasure()** — определить размер и расположение вашего **View** на экране.

В качестве параметров он принимает две переменные **widthMeasureSpec** и **heightMeasureSpec**, которые в свою очередь представляют собой требования измерения ширины и высоты вашего **View**.

При переопределении данного метода, необходимо указать ширину и высоту вашего **View** самостоятельно, используя метод **setMeasuredDimension()**.

onMeasure()

MeasureSpec — это класс, используемый для определения размеров View в Android. Когда View помещается на экран, ей нужно знать, какое место ей предоставлено, чтобы правильно расположиться и отобразиться. MeasureSpec состоит из двух основных компонентов: размера и режима измерения.

```
class CustomView(context: Context?) : View(context) {

    override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec)
        val width = MeasureSpec.getSize(widthMeasureSpec)
        val height = MeasureSpec.getSize(heightMeasureSpec)
        setMeasuredDimension(width, height)
    }
```

onMeasure()

Режим измерения может быть одним из трех типов:

1. **EXACTLY (точно)** — размер View должен быть задан точно (например, в dp или px). Это может быть указано в макете View с атрибутом android:layout_width или android:layout_height со значением фиксированной ширины или высоты.

onMeasure()

2. **AT_MOST (не больше)** — View может быть любого размера, который не превышает указанный максимальный размер, например, layout_width="wrap_content". Это означает, что View может иметь любой размер, пока он не превышает размер родительского контейнера.

3. **UNSPECIFIED (неопределенный)** — размер View может быть любым, не ограниченным размером родителя.

onMeasure()

```
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    val desiredWidth = 100 // Предполагаемая ширина View
    val desiredHeight = 100 // Предполагаемая высота View

    val widthMode = MeasureSpec.getMode(widthMeasureSpec)
    val widthSize = MeasureSpec.getSize(widthMeasureSpec)
    val heightMode = MeasureSpec.getMode(heightMeasureSpec)
    val heightSize = MeasureSpec.getSize(heightMeasureSpec)

    val width = when (widthMode) {
        MeasureSpec.EXACTLY -> widthSize // Задан конкретный размер для ширины
        MeasureSpec.AT_MOST -> min(desiredWidth, widthSize) // Размер не должен превышать заданный размер
        else -> desiredWidth // Задать предпочтительный размер, если точного или максимального размера не задано
    }

    val height = when (heightMode) {
        MeasureSpec.EXACTLY -> heightSize // Задан конкретный размер для высоты
        MeasureSpec.AT_MOST -> min(desiredHeight, heightSize) // Размер не должен превышать заданный размер
        else -> desiredHeight // Задать предпочтительный размер, если точного или максимального размера не задано
    }

    setMeasuredDimension(width, height) // Устанавливаем фактический размер View
}
```

onLayout()

Метод **onLayout()** вызывается при каждом изменении размера и позиции View, в том числе при его создании и перерисовке. Обычно этот метод переопределяется в Custom View только в том случае, когда в нем есть дочерние View, которые нужно разместить в определенном порядке.

onDraw()

Основной метод при разработке собственной View.
При переопределении метода onDraw() используется объект Canvas (2D-холст), на котором можно рисовать графические элементы. Также в этом методе можно использовать объекты Paint и Path, которые определяют стиль и форму рисуемых элементов.

Понятие CustomView

Жизненный цикл View

- **Canvas**

Invalidate/requestLayout

Canvas

Canvas предоставляет нам методы для рисования фигур, линий, текста и других элементов на экране.
Например:

- **drawColor(color: Int)** заливает всю область цветом, указанным в аргументе;
- **drawLine(startX: Float, startY: Float, stopX: Float, stopY: Float, paint: Paint)** рисует линию, заданную двумя точками;

Canvas

- **drawRect(left: Float, top: Float, right: Float, bottom: Float, paint: Paint)** рисует прямоугольник, заданный координатами левого верхнего и правого нижнего углов;
- **drawCircle(cx: Float, cy: Float, radius: Float, paint: Paint)** рисует круг, заданный координатами центра и радиусом;
- **drawText(text: String, x: Float, y: Float, paint: Paint)** рисует текст, заданный строкой и координатами базовой точки.

Paint

Объект Paint представляет собой кисть, с помощью которой мы рисуем на Canvas.

Примеры методов для val paint = Paint():

- **color (цвет рисования)** — paint.color = Color.RED
- **strokeWidth (ширина линии рисования)** — paint.strokeWidth = 10f
- **style (стиль рисования)** — paint.style = Paint.Style.FILL. Принимает в качестве параметра одно из значений класса Paint.Style: FILL, STROKE или FILL_AND_STROKE.
- **textSize (размер шрифта текста)** — paint.textSize = 30f

Paint и Canvas

```
class MyCustomView(context: Context, attrs: AttributeSet?) : View(context, attrs) {

    private val paint = Paint()

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)

        // Устанавливаем цвет и стиль для Paint
        paint.color = Color.RED
        paint.style = Paint.Style.FILL

        // Рисуем круг на Canvas
        canvas.drawCircle( cx: width / 2f,  cy: height / 2f,  radius: 100f, paint)

        // Устанавливаем цвет и стиль для Paint
        paint.color = Color.BLUE
        paint.style = Paint.Style.STROKE
        paint.strokeWidth = 10f

        // Рисуем прямоугольник на Canvas
        canvas.drawRect( left: 50f,  top: 50f,  right: 200f,  bottom: 200f, paint)
    }
}
```

Paint и Canvas

Создание объектов в onDraw() может привести к лишним затратам памяти и ухудшению производительности приложения.
Метод onDraw() вызывается при каждой перерисовке View, поэтому слишком частое создание новых объектов Paint может вызвать нагрузку на сборщик мусора. Вместо этого рекомендуется создать объект Paint в конструкторе класса или в другом подходящем методе и переиспользовать его в методе onDraw().

Понятие CustomView

Жизненный цикл View

Canvas

- **Invalidate/requestLayout**

Обновление View

Когда данные, используемые внутри View, изменяются, требуется обновить View, чтобы отобразить новые значения. Из диаграммы жизненного цикла видно, что существуют два метода, которые заставляют View перерисовываться:

- **invalidate()** — используется, когда нужно только перерисовать ваш элемент;
- **requestLayout()** — используется, когда нужно изменить размеры вашего View.

Метод invalidate()

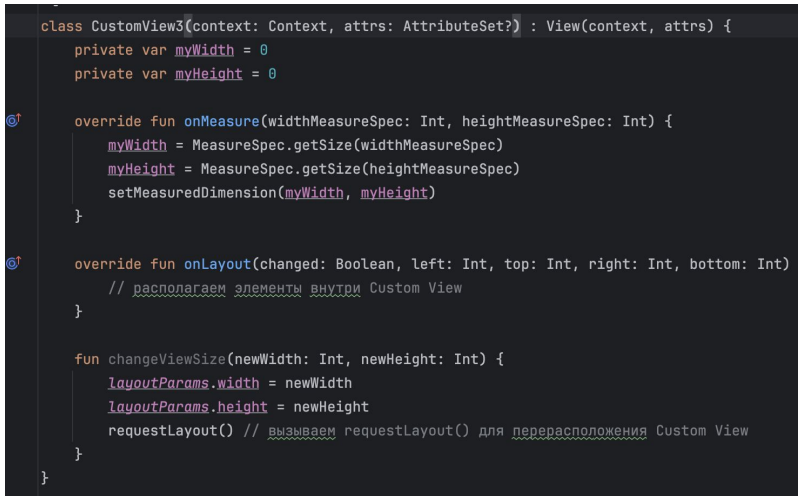
Для обновления визуальной части нашего **View**, используется метод **invalidate()**. Например, когда ваш View-компонент обновляет свой текст, цвет или обрабатывает прикосновение. Это значит, что View-компонент будет вызывать только метод **onDraw()**, чтобы обновить свое состояние.

Метод invalidate()

```
class CustomView2(context: Context, attrs: AttributeSet?) : View(context, attrs) {
    private val paint = Paint()

    init {
        paint.color = Color.RED
    }

    override fun onDraw(canvas: Canvas) {
        canvas.drawCircle( cx: width / 2f,  cy: height / 2f,  radius: width / 4f, paint)
    }

    fun changePaintColor(newColor: Int) {
        paint.color = newColor
        invalidate() // вызываем invalidate() для перерисовки Custom View
    }
}
```

Метод requestLayout()

Если у нашего View были изменены размеры и/или позиция, необходимо вызвать метод requestLayout(), после которого последует
вызов методов согласно жизненному циклу View, т. е. onMeasure() → onLayout() → onDraw().



```
class CustomView3(context: Context, attrs: AttributeSet?) : View(context, attrs) {
    private var myWidth = 0
    private var myHeight = 0

    override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
        myWidth = MeasureSpec.getSize(widthMeasureSpec)
        myHeight = MeasureSpec.getSize(heightMeasureSpec)
        setMeasuredDimension(myWidth, myHeight)
    }

    override fun onLayout(changed: Boolean, left: Int, top: Int, right: Int, bottom: Int)
        // располагаем элементы внутри Custom View
    }

    fun changeViewSize(newWidth: Int, newHeight: Int) {
        layoutParams.width = newWidth
        layoutParams.height = newHeight
        requestLayout() // вызываем requestLayout() для перерасположения Custom View
    }
}
```

Обновление view

Особенность применения этих методов заключаются в том, что частая перерисовка или пересчет размеров View может замедлить работу приложения.
Поэтому лучше использовать эти методы только в случае необходимости. Например, если изменения внешнего вида View могут быть объединены в один вызов invalidate(), то лучше объединить их для уменьшения количества перерисовок.

Метод onTouchEvent()

Метод **onTouchEvent()** — это один из методов обработки событий пользовательского ввода в View. Вызывается при каждом событии касания на View, например при нажатии, перемещении или отпускании пальца.

Метод возвращает значение типа Boolean, которое указывает, было ли событие обработано этим методом. Если метод возвращает true, это означает, что событие было обработано и больше никаких действий не требуется, если он возвращает false, событие продолжит свой путь по иерархии View и будет обработано другими методами.

Метод onTouchEvent()

```
class MyCustomView(context: Context, attrs: AttributeSet) : View(context, attrs) {

    override fun onTouchEvent(event: MotionEvent): Boolean {
        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                // обработка нажатия пальца на экран
                return true
            }
            MotionEvent.ACTION_MOVE -> {
                // обработка перемещения пальца по экрану
                return true
            }
            MotionEvent.ACTION_UP -> {
                // обработка отпускания пальца от экрана
                return true
            }
        }
        return super.onTouchEvent(event)
    }
}
```

Практика

Задача 1. Создание базовой CustomView.

Создайте ColorBoxView, который рисует прямоугольник заданного цвета.

Задача 2.

Добавьте обработку касания. При нажатии на View, цвет меняется случайным образом.

Задача 3. Применение requestLayout()

Сделайте кнопку, при нажатии на которую View увеличивается в 2 раза. Используйте requestLayout() и invalidate().

Задача 4. Рисование текста и фигур

Нарисуйте текст в центре круга (например, "50%") с помощью drawText() и drawCircle().

Домашнее задание

Задача 1. Свой индикатор загрузки

Реализовать любой кастомный индикатор загрузки.

Q&A

Ваши вопросы

Спасибо

<TeachMeSkills/>