Кастомные контейнеры компоновки

Создание модификаторов компоновки

Каждый раз, когда вызывается родительский компонуемый объект, он отвечает за управление размером и расположением всех своих дочерних элементов. Положение дочернего элемента определяется с использованием координат x и y относительно положения родителя. Что касается размера, родительский элемент накладывает ограничения, которые определяют максимально и минимально допустимые размеры высоты и ширины дочернего элемента. В зависимости от конфигурации размер родительского компонента может быть либо фиксированным (например, с помощью модификатора size()), либо рассчитан на основе размера и расположения его дочерних элементов. Все встроенные контейнеры - Box, Row и Column содержат логику, которая измеряет каждого дочернего компонента и вычисляет, как расположить каждый из них. И мы также можем использовать для создания собственных контейнеров все те же методы, которые применяются встроенными контейнерами.

Кастомные контейнеры компоновки можно разделить на две категории:

- В самой базовой форме пользовательский контейнер может быть реализован как модификатор компоновк, который можно применить к одному элементу пользовательского интерфейса (что-то похожее на стандартный модификатор padding())
- Второй способ представляет создание своего объекта Layout, который будет применяться ко всем дочерним компонентам (аналогично Box, Column и Row)

Создание кастомного модификатора компоновки

Общий синтаксис реализации кастомного модификатора компоновки выглядит следующим образом:

```
fun Modifier.имя_модификатора (необязательные_параметры) = layout { measurable,
constraints ->

    // код настройки позиции и размера компонента
}
```

Концевой лямбде передаются два параметра с именами measurable и constraints. Параметр measurable представляет дочерний компонент, для которого был вызван модификатор, а параметр constraints содержит максимальную и минимальную ширину и высоту, разрешенные для дочернего компонента.

Когда модификатор размещает дочерний компонент, ему необходимо знать размеры этого компонента, чтобы убедиться, что он соответствует ограничениям constraints, переданным в лямбда-выражение. Эти значения можно получить, вызвав у объекта measurable метод measure(), в который передается параметр constraints:

```
val placeable = measurable.measure(constraints)
```

Этот вызов вернет объект класса Placeable, который в свойствах width и height содержит значения высоты и ширины. Тип Placeable предоставляет ряд методов для установки новой позиции компонента в контейнере:

- place()
- placeRelative()
- placeRelativeWithLayer()

Эти методы имеют множество различных версий, но все они помещают компонент на определенную позицию внутри контейнера.

Например определим модификатор компоновки, который помещает компонент на определенную позицию с координатами X и Y относительно позиции по умолчанию:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.runtime.Composable
import androidx.compose.ui.layout.layout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            Box(Modifier.fillMaxSize()) {
                MyBox(Modifier.positionLayout(100, 50).background(Color.DarkGray))
            }
        }
    }
}

fun Modifier.positionLayout(x: Int, y: Int) = layout { measurable, constraints ->
    val placeable = measurable.measure(constraints)
    layout(placeable.width, placeable.height) {
        placeable.placeRelative(x, y)
    }
}

@Composable
fun MyBox(modifier: Modifier) {
```

```
    Box(Modifier.size(200.dp).then(modifier))
}
```

Здесь определен кастомный модификатор компоновки positionLayout (название произвольное). Пусть в качестве параметров он принимает координаты верхнего левого угла компонента относительно позиции по умолчанию (если компонент один, то это будет позиция верхнего левого угла контейнера)- параметры x и y.

```
fun Modifier.positionLayout(x: Int, y: Int) = layout { measurable, constraints ->
    val placeable = measurable.measure(constraints)
    layout(placeable.width, placeable.height) {
        placeable.placeRelative(x, y)
    }
}
```

В концевой лямбде получаем объект Placeable

```
val placeable = measurable.measure(constraints)
```

Затем вызывается функция layout():

```
layout(placeable.width, placeable.height) {
    placeable.placeRelative(x, y)
}
```

Эта функция получает высоту и ширину объекта Placeable. В качестве последнего параметра в функцию layout передается лямбда-выражение, которое собственно и выполняет позиционирование компонента. В частности, для этого мы используем метод placeable.placeRelative(), который принимает координаты x и y, по которым надо поместить компонент (координаты относительно верхнего левого угла контейнера).

Для примера здесь определяется кастомный компонент MyBox, который по сути является оберткой над Box:

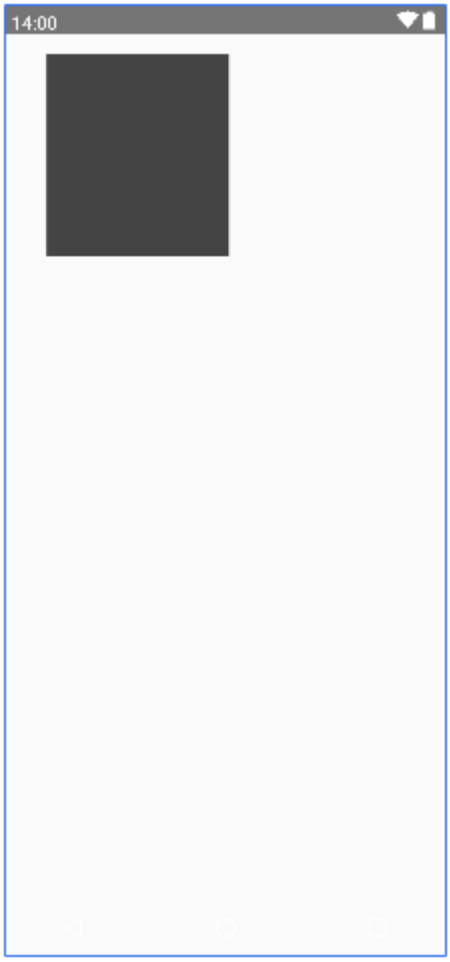
```
@Composable
fun MyBox(modifier: Modifier) {
    Box(Modifier.size(200.dp).then(modifier))
}
```

К Box применяется модификатор, который устанавливает размер прямоугольной области в 200 пикселей и добавляет функции модификаторов, которые передаются в компонент MyBox через единственный параметр.

При вызове метода setContent() применяем наш модификатор positionLayout:

```
MyBox(Modifier.positionLayout(100, 50).background(Color.DarkGray))
```

ТО есть в данном случае компонент будет позиционироваться на координату с x=100 и y=50. И кроме того, к нему применяется темно-серый фон:



Причем опять же подчеркну, что позиция устанавливается относительно точки по умолчанию (в данном случае верхний левый угол контейнера). И если наш контейнер MyBox помещается в другой контейнер, то к MyBox применяются также отступы родительского контейнера. Например:

```
Box(Modifier.padding(10.dp, 40.dp)) {
    MyBox(Modifier.positionLayout(100, 50).background(Color.DarkGray))
}
```

В данном случае реальная позиция MyBox будет на точке x=(10+100)=110 и y=(40+50)=90.

Также при разработке своих макетов компоновки следует помнить, что дочерний компонент необходимо измерять только один раз при каждом вызове модификатора. Это так называемое однократное измерение (single-pass measurement) необходимо для обеспечения быстрой и эффективной визуализации иерархий дерева пользовательского интерфейса.

Другой пример - кастомный модификатор margin(), который устанавливает внешние отступы, но его мезаника будет аналогична:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.runtime.Composable
import androidx.compose.ui.layout.layout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent{
            Box(Modifier.fillMaxSize()) {
                MyBox(Modifier.margin(200).background(Color.DarkGray))
            }
        }
    }
}

fun Modifier.margin(all:Int) = layout { measurable, constraints ->
    val placeable = measurable.measure(constraints)
    layout(placeable.width, placeable.height) {
        placeable.placeRelative(all, all)
    }
}

@Composable
fun MyBox(modifier: Modifier) {
    Box(Modifier.size(200.dp).then(modifier))
}
```

Создание контейнеров компоновки

Jetpack Compose позволяет создавать собственные компоненты компоновки с полным контролем над размером и расположением всех дочерних компонентов. Для создания контейнеров компоновки применяется встроенная функция Layout, которая предоставляет способ измерения и позиционирования дочерних компонентов. Функция Layout имеет ряд версий, которые похожи. Возьмем одну из них:

```
@UiComposable
@Composable
inline fun Layout(
    content: @Composable @UiComposable () -> Unit,
    modifier: Modifier = Modifier,
    measurePolicy: MeasurePolicy
): Unit
```

В общем случае функция-компонент Layout принимает три параметра:

- content: содержимое
- modifier: функции-модификаторы
- measurePolicy: политика компоновки вложенного содержимого в виде объекта MeasurePolicy

В начале рассмотрим общий принцип создания контейнера компоновки. Для демонстрации возьмем самое просто определение контейнера:

```
@Composable
fun MyLayout(modifier: Modifier = Modifier, content: @Composable () -> Unit){
    Layout(modifier = modifier, content = content) {
        measurables, constraints ->
            val placeables = measurables.map { measurable ->
                // измеряем каждый дочерний компонент
                measurable.measure(constraints)
            }
            layout(constraints.maxWidth, constraints.maxHeight) {
                placeables.forEach { placeable ->
                    placeable.placeRelative(x = 0, y = 0)
                }
            }
    }
}
```

В данном случае контейнер называется MyLayout. Он принимает два параметра - модификаторы и функцию установки содержимого. Это наиболее часто применяемые параметры, но естественно при необходимости можно определять и другие параметры.

Далее функция MyLayout вызывает компонент Layout, в который передаются модификаторы и содержимое. Третий параметр функции Layout в данном случае представляет концевую лямбду, которая принимает два параметра: measurables и constraints. Параметр measurables содержит все дочерние компоненты, которые передаются в Layout через content (список объектов Measurable). А параметр constraints содержит максимальную и минимальную ширину и высоту, допустимую для дочерних компонентов.

```
Layout(modifier = modifier, content = content) {
   measurables, constraints ->
   ..................
```

Далее выполняется измерение дочерних компонентов, а результаты помещаются в список объектов Placeable:

```
val placeables = measurables.map { measurable ->
    // измеряем каждый дочерний компонент
    measurable.measure(constraints)
}
```

Здесь метод map() проходит по всем дочерним компонентам (объектам Measurable) и для каждого из них выполняет метод measure(), который, в свою очередь, измеряет дочерний компонент. Результатом является список экземпляров Placeable (по одному для каждого дочернего компонента).

В конце вызывается функция layout(), которой передаются максимальные значения высоты и ширины, разрешенные родительским элементом:

```
layout(constraints.maxWidth, constraints.maxHeight) {
    placeables.forEach { placeable ->
        placeable.placeRelative(x = 0, y = 0)
    }
}
```

Концевая лямбда перебирает все дочерние компоненты (которые теперь представляют объекты Placeable) с помощью функции placeables.forEach() и позиционирует их относительно позиции по умолчанию, назначенной родительским элементом. В данном случае для простоты позиционирование производится на точку с координатами x=0 и y=0.

Рассмотрим применение своего контейнера компоновки:

```
package com.example.helloapp

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.size
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.unit.dp
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.runtime.Composable
import androidx.compose.ui.layout.Layout

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
        setContent{
            Box(Modifier.fillMaxSize()) {
                MyLayout(spacing = 20){
                    Box(Modifier.size(100.dp).background(Color.DarkGray))
                    Box(Modifier.size(100.dp).background(Color.Red))
                    Box(Modifier.size(100.dp).background(Color.Blue))
                    Box(Modifier.size(100.dp).background(Color.Magenta))
                    Box(Modifier.size(100.dp).background(Color.Green))
                }
            }
        }
    }
}

@Composable
fun MyLayout(modifier: Modifier = Modifier, spacing: Int = 0, content: @Composable
() -> Unit){
    Layout(modifier = modifier,content = content) {
        measurables, constraints ->
            val placeables = measurables.map { measurable ->
                measurable.measure(constraints)
            }
            layout(constraints.maxWidth, constraints.maxHeight) {
                var yCoord = 0
                var xCoord = 0
                placeables.forEach { placeable ->
                    // размещаем текущий дочерний компонент
                    placeable.placeRelative(x = xCoord, y = yCoord)
                    // устанавливаем x-координату для следующего компонента
                    xCoord += placeable.width + spacing
                    // если дошли до края контейнера
                    if((xCoord + placeable.width) > constraints.maxWidth) {
                        // увеличиваем y-координату (переход на следующую строку)
                        yCoord += placeable.height + spacing
                        // и сбрасываем x-координату
                        xCoord = 0
                    }
                }
            }
    }
}
```

Здесь наш кастомный контейнер MyLayout принимает три параметра:

```
@Composable
fun MyLayout(modifier: Modifier = Modifier, spacing: Int = 0, content: @Composable
() -> Unit){
```

Кроме стандартных параметров типа модификатора и содержимого контейнер также принимает параметр spacing - отступ между вложенными компонентами. По умолчанию он равен 0, соответственно параметр является необязательным для установки.

Внутри MyLayout в компоненте Layout выполняем размещение каждого вложенного компонента с помощью следующего кода:

```
layout(constraints.maxWidth, constraints.maxHeight) {
    var yCoord = 0
    var xCoord = 0
    placeables.forEach { placeable ->
        placeable.placeRelative(x = xCoord, y = yCoord)
        xCoord += placeable.width + spacing
        if((xCoord + placeable.width) > constraints.maxWidth) {
            yCoord += placeable.height + spacing
            xCoord = 0
        }
    }
}
```

Для установки координат x и y здесь определены дополнительные переменные xCoord и yCoord соответственно. ПРи переборе каждого вложенного компонента вначале устанавливаем его позицию на основании переменных xCoord и yCoord:

```
placeable.placeRelative(x = xCoord, y = yCoord)
```

Затем увеличиваем значение координаты x на ширину размещенного компонента плюс отступ:

```
xCoord += placeable.width + spacing
```

Если координата x вышла за диапазон допустимой ширины, то сбрасываем переменную xCoord и увеличиваем переменную yCoord на высоту размещенного компонента плюс отступ. ТО есть таким образом мы перейдем на следующую строку:

```
if((xCoord + placeable.width) > constraints.maxWidth) {
    yCoord += placeable.height + spacing
    xCoord = 0
}
```

Для установки следующего компонента, таким образом, будут применяться уже измененные значения xCoord и yCoord.

Применение нашего контейнера компоновки аналогично другим компонентам. В данном случае устанавливаем параметр spacing и размещаем внутри несколько типовых компонентов:

```
MyLayout(spacing = 20){
    Box(Modifier.size(100.dp).background(Color.DarkGray))
    Box(Modifier.size(100.dp).background(Color.Red))
    Box(Modifier.size(100.dp).background(Color.Blue))
    Box(Modifier.size(100.dp).background(Color.Magenta))
    Box(Modifier.size(100.dp).background(Color.Green))
}
```