88. Estados (State) en Compose

(Seguir en Android Developer AD)

El estado de una app es cualquier valor que puede cambiar con el paso del tiempo.

88.1 Estado y composición

Compose es declarativo y, por lo tanto, la única manera de actualizarlo es llamar al mismo elemento que admite composición con argumentos nuevos. Estos argumentos son representaciones del estado de la IU. Cada vez que se actualiza un estado, se produce una **recomposición**

88.2 El estado de elementos componibles

Las funciones de compatibilidad pueden usar la API de `remember` para almacenar un objeto en la memoria. Un valor calculado por remember se almacena en la composición durante la composición inicial, y el valor almacenado se muestra durante la recomposición. Se puede usar remember para almacenar tanto objetos mutables como inmutables.

mutableStateOf crea un MutableState observable, que es un tipo observable integrado en el entorno de ejecución de Compose.

```
interface MutableState<T> : State<T> {
    override var value: T
}
```

Existen tres maneras de declarar un objeto MutableState en un elemento que admite composición:

```
val mutableState = remember { mutableStateOf(default) }
var value by remember { mutableStateOf(default) }
val (value, setValue) = remember { mutableStateOf(default) }
```

La sintaxis del delegado by requiere las siguientes importaciones:

```
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
```

Aunque remember te ayuda a retener el estado entre recomposiciones, el estado no se retiene entre cambios de configuración. Para ello, debes usar rememberSaveable. rememberSaveable almacena automáticamente cada valor que se puede guardar en un Bundle. Para otros valores, puedes pasar un objeto Saver personalizado.

88.3 Otros tipos de estados compatibles

Antes de leer otro tipo observable en Jetpack Compose, debes convertirlo en un State para que Jetpack Compose pueda recomponer automáticamente cuando cambie el estado.

Compose cuenta con funciones para crear State a partir de tipos observables comunes utilizados en apps para Android: Antes de usar estas integraciones, agrega los artefactos adecuados según se describe a continuación:

- Flow: collectAsStateWithLifecycle()

collectAsStateWithLifecycle() recopila valores de un Flow de manera optimizada para ciclos de vida, lo que permite que tu app guarde los recursos innecesarios. Representa el último valor emitido a través de Compose State. Usa esta API como la forma recomendada de recopilar flujos en apps para Android.

88.4 Elevación de Estado

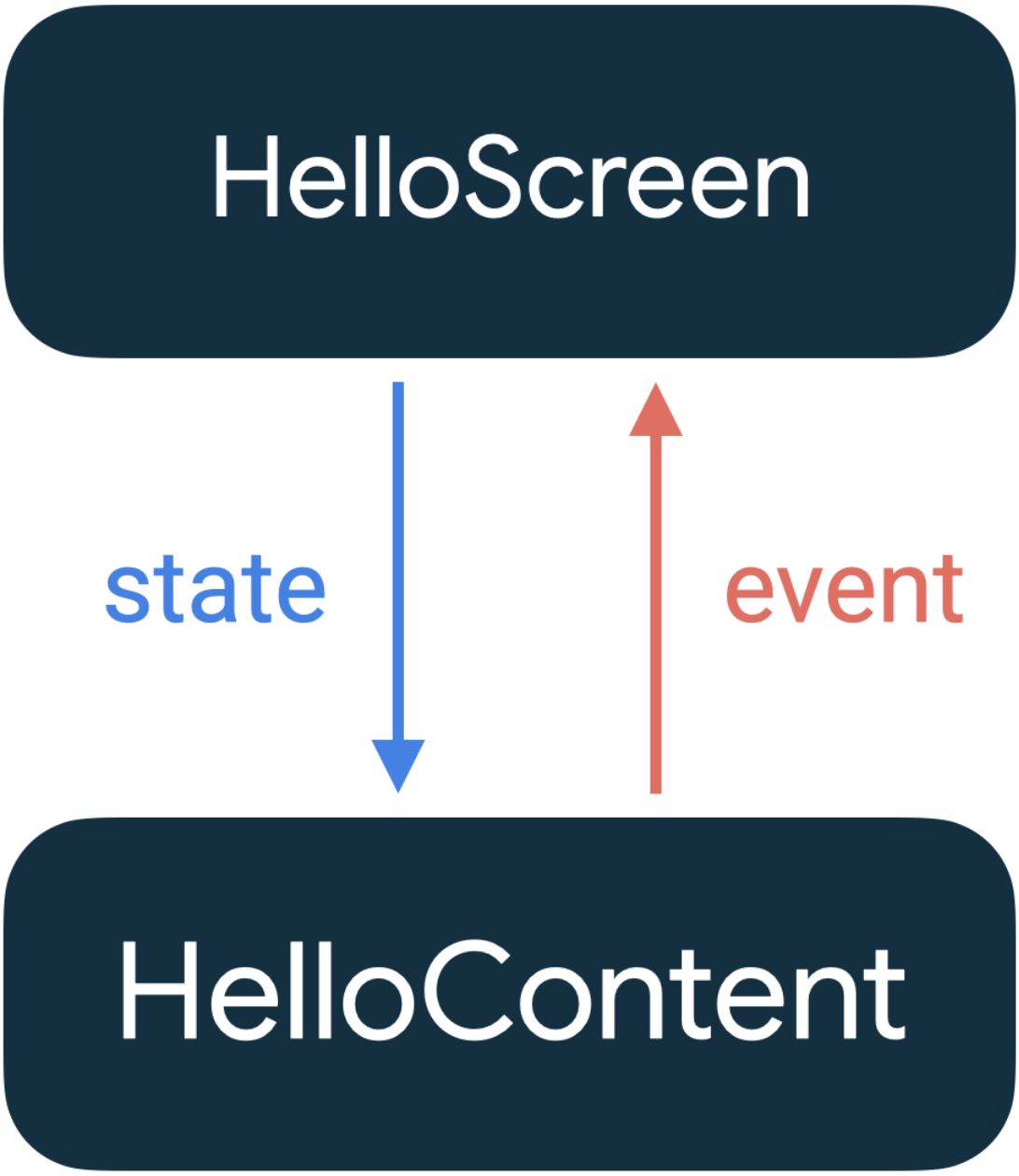
La elevación de estado se refiere al hecho de tratar los estados en las ventanas superiores (que contienen otras ventanas) y hacer que las ventanas hijas sean sin estado.
Para ello utilizaremos parámetros de entrada de tipo funciones lambda.

```
@Composable
fun HelloScreen() {
    var name by rememberSaveable { mutableStateOf("") }

    HelloContent(name = name, onNameChange = { name = it })
}

@Composable
fun HelloContent(name: String, onNameChange: (String) -> Unit) {
    Column(modifier = Modifier.padding(16.dp)) {
        Text(
            text = "Hello, $name",
            modifier = Modifier.padding(bottom = 8.dp),
            style = MaterialTheme.typography.h5
        )
        OutlinedTextField(
            value = name,
            onValueChange = onNameChange,
            label = { Text("Name") }
        )
    }
}
```

La variable `name` reside en `HelloScreen` y se pasa a `HelloContent` con dos parametros: `name` y `onNameChange`

Generalizando, hemos sustituido la variable de estado local por dos parámetros: * name: de tipo * onNameChange: de tipo (T)->Unit

Se sigue un patrón `flujo unidireccional de datos` : los estadis bajan y los eventos suben.



88.5 Apendice

Versión 0.5 (20-12-23)

¿Fue útil esta página?

☺ ☹