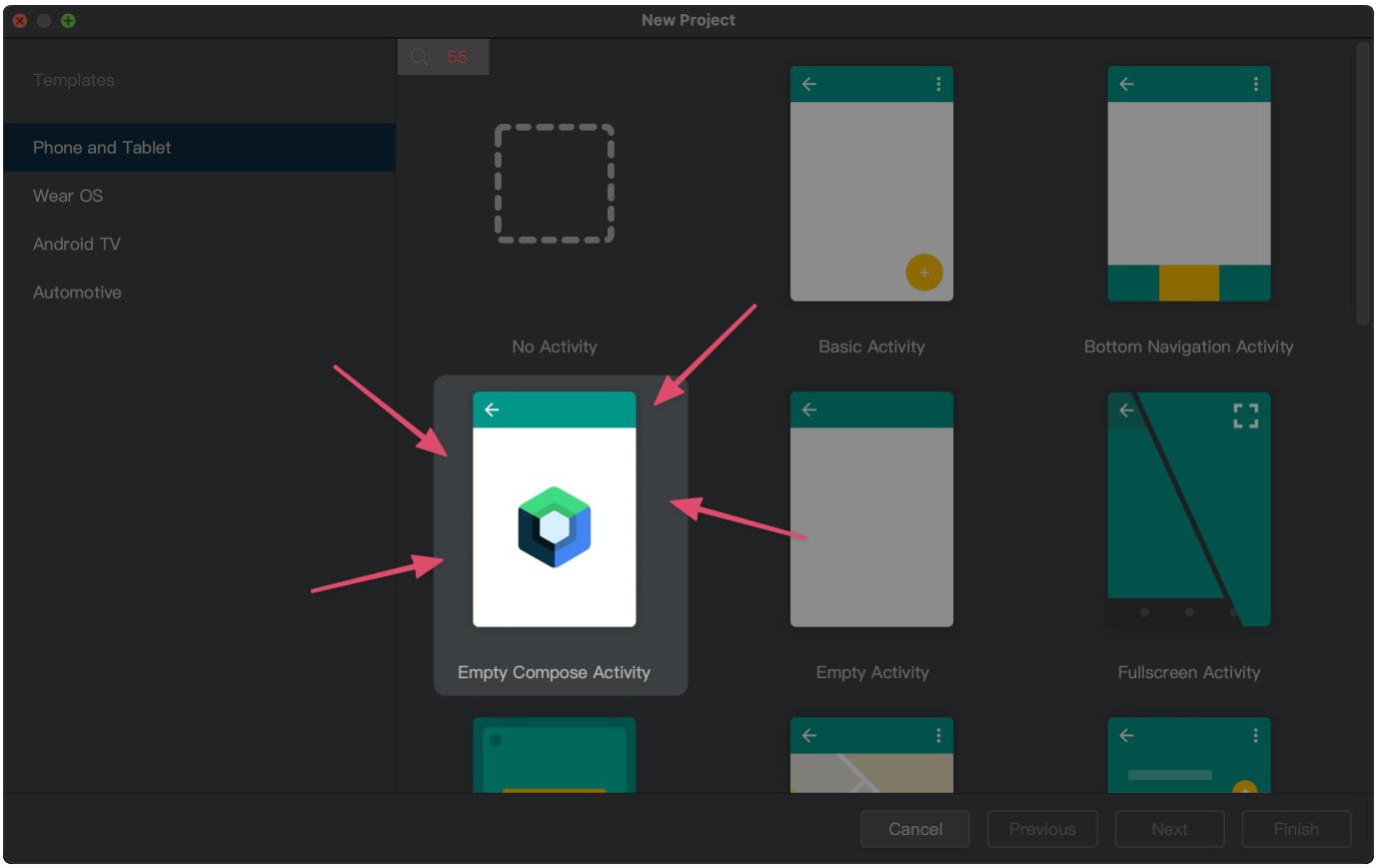
Compose 11.08 直播讲义

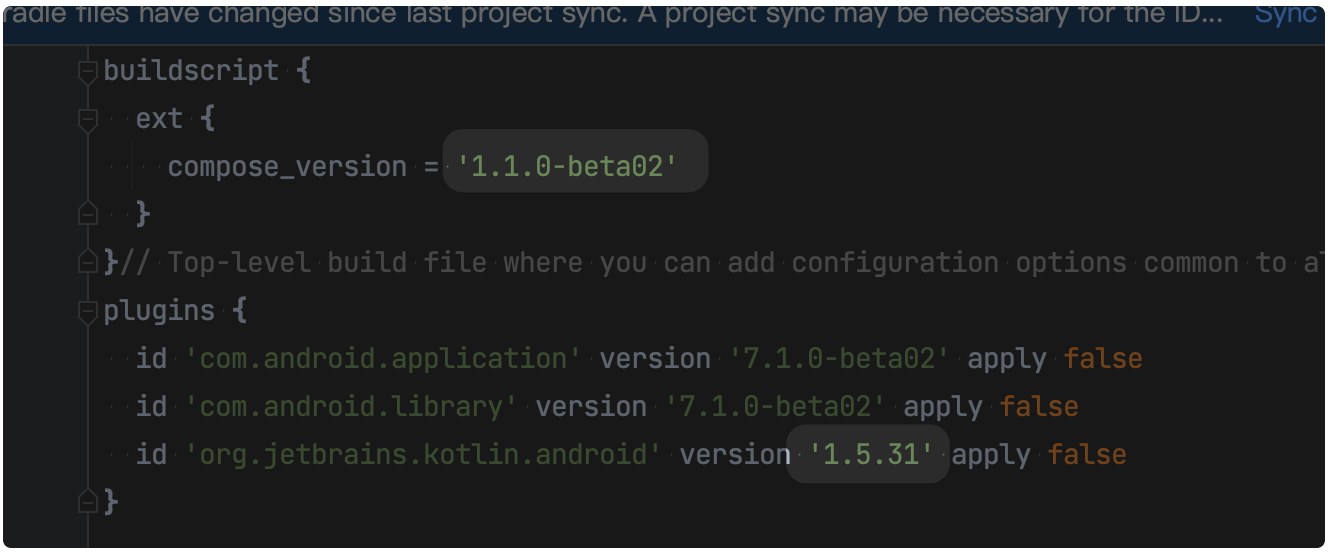
创建项目

选择 Empty Compose Activity：



（可选）改一下 Compose 和 Kotlin 插件的版本号：

```
buildscript {
    ext {
        compose_version = '1.0.1'
    }
}// Top-level build file where you can add configuration options common to
plugins {
    id 'com.android.application' version '7.1.0-beta02' apply false
    id 'com.android.library' version '7.1.0-beta02' apply false
    id 'org.jetbrains.kotlin.android' version '1.5.21' apply false
}
```



Compose 新创建项目的模板代码

`setContent()` 是什么

`setContentView()` 的 Compose 等价物，负责「进入Compose 模式」。——实质上就是让你写出的 Compose 代码可以被 Android 认识。

——Android 本身是不知道什么叫 Compose 的，Compose 要想被 Android 运行，就得通过某种翻译器来让 Android 认识。所谓「翻译器」，实际上就是 Compose 的整个编译和运行的框架，具体来说，就是你需要设置一个新的函数，然后在这个函数的内部去写 Compose 代码，并由这个函数来翻译成 Android 认识的样子。

这个「翻译器函数」，就是 `setContent()`。

`setContent()` 内部的其他函数

它里面的 `ComposeTodoTheme` 以及更里面的 `Surface()`、`Greeting()`，就是所谓的「Compose 格式」的代码了。它用一层层大括号，把 UI 逻辑包进了一个个的 Lambda 表达式里，用 DSL（Domain Specific Language）的方式写出了「长得像配置文件」的可执行代码。

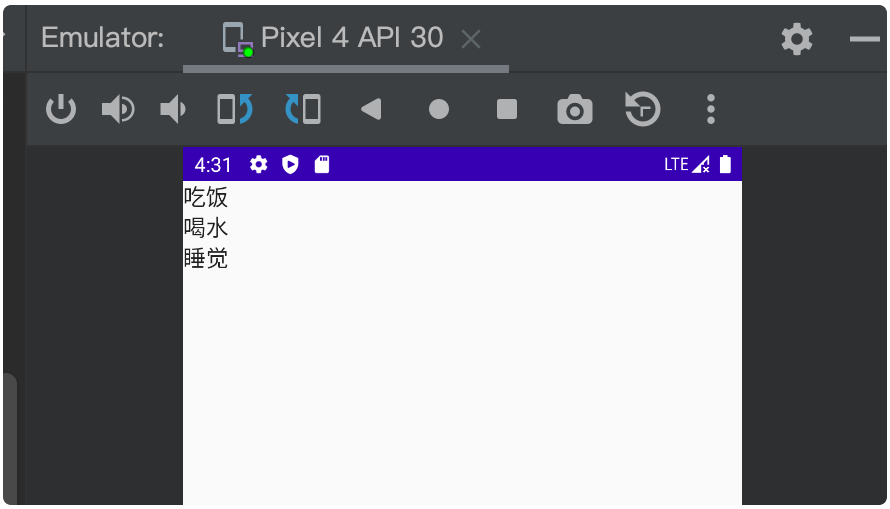
关键：大括号是 Lambda，而 Lambda 代码的执行完全依赖于调用它的函数，绝不是「从外到内顺序执行」。

写一个 Todo App（待办事项）流程回顾

列表：`LazyColumn()`——`RecyclerView` 的等价物

Kotlin 复制代码

```kotlin
setContent {
  ComposeTodoAppTheme {
    LazyColumn {
      val todos = listOf("吃饭", "喝水", "睡觉")
      items(todos) { todo ->
        Text(todo)
      }
    }
  }
}
```



预览功能：`@Preview`

Kotlin 复制代码

```kotlin
@Composable
private fun TodoList() {
  LazyColumn {
    val todos = listOf("吃饭", "喝水", "睡觉")
    items(todos) { todo ->
      Text(todo)
    }
  }
}
```

Kotlin 复制代码

```kotlin
@Preview
@Composable
fun TodoListPreview() {
  TodoList()
}
```

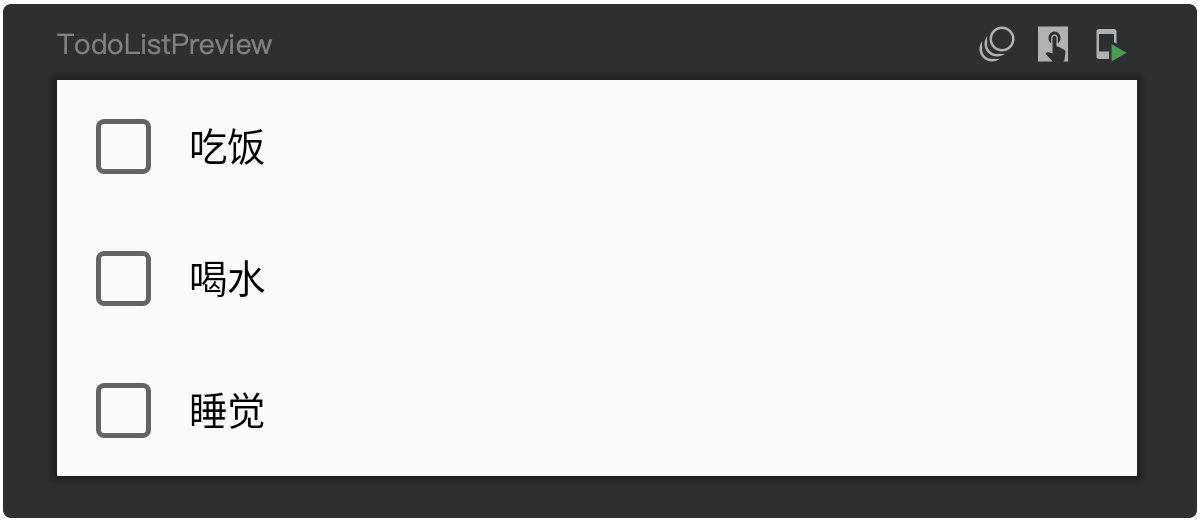
Kotlin 复制代码

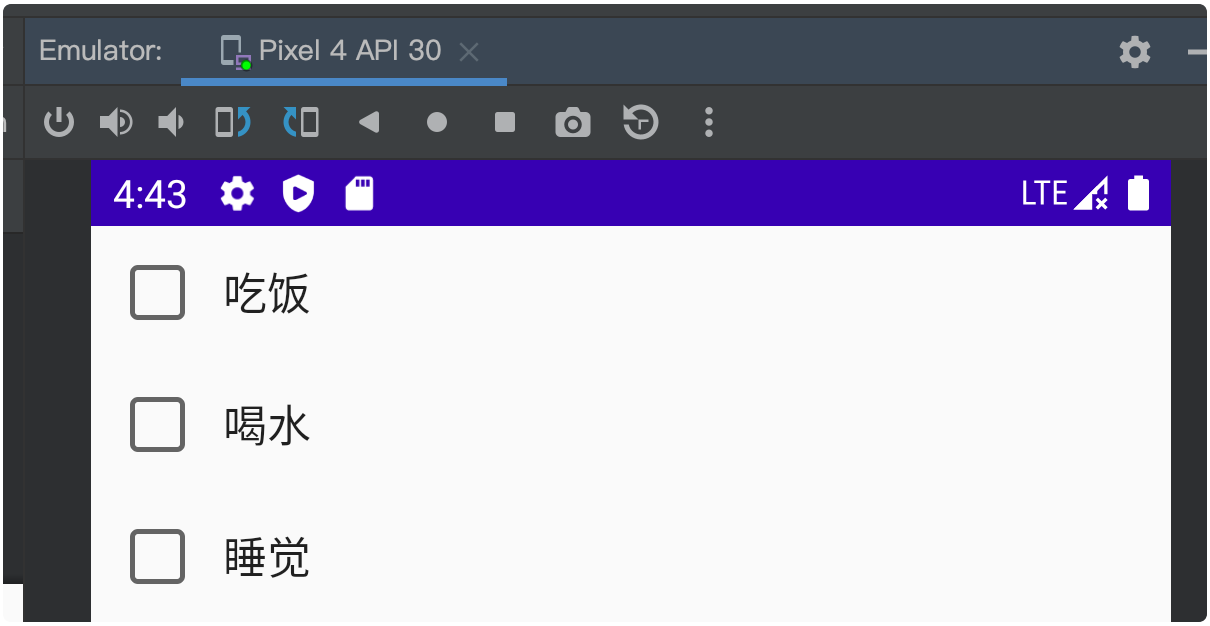
```kotlin
@Preview(showBackground = true)
@Composable
fun TodoListPreview() {
  TodoList()
}
```

调整格式

Kotlin 复制代码

```kotlin
@Composable
private fun TodoList() {
  LazyColumn(Modifier.fillMaxWidth()) { // 1️⃣ Modifier.fillMaxWidth() 横向填充
    val todos = listOf("吃饭", "喝水", "睡觉")
    items(todos) { todo ->
      Row(verticalAlignment = Alignment.CenterVertically) { // 3️⃣ 垂直居中
        Checkbox(checked = false, onCheckedChange = {}) // 2️⃣ 增加 Checkbox()
        Text(todo)
      }
    }
  }
}
```

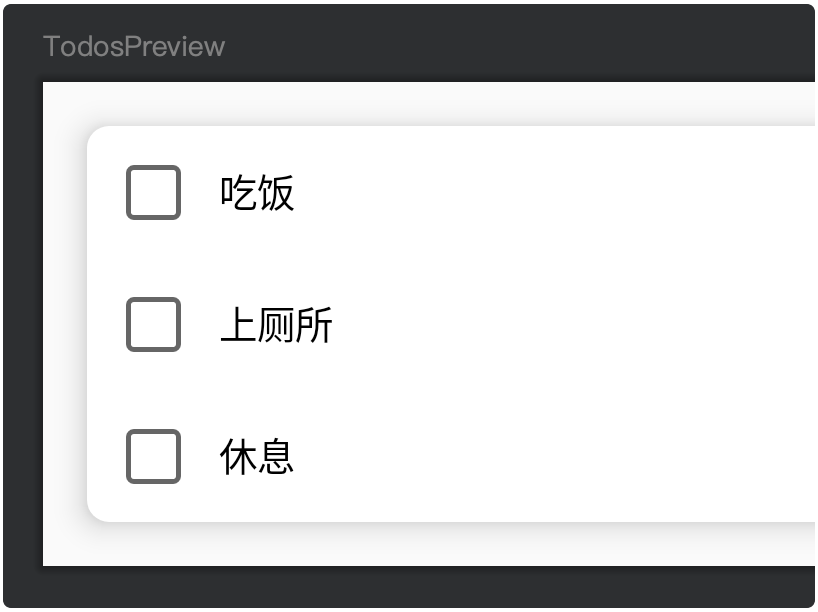




增加阴影和圆角：`Surface()`

Kotlin | 复制代码

```kotlin
@Composable
private fun TodoList() {
  // 包进 Surface，并加上 padding 和 elevation 以及 shape
  Surface(Modifier.padding(16.dp), elevation = 8.dp,
      shape = RoundedCornerShape(8.dp)) {
    LazyColumn(Modifier.fillMaxWidth()) {
      ...
    }
  }
}
```



> 阴影怎么来的?靠的就是前面说的「Compose 环境」——所有组件都由 Compose 统一渲染，而不是各自独立渲染，这样就能让每个组件很方便地渲染它的「外部」区域，于是阴影效果这种传统 View 写法很难做的事，到了 Compose 里变得很简单。

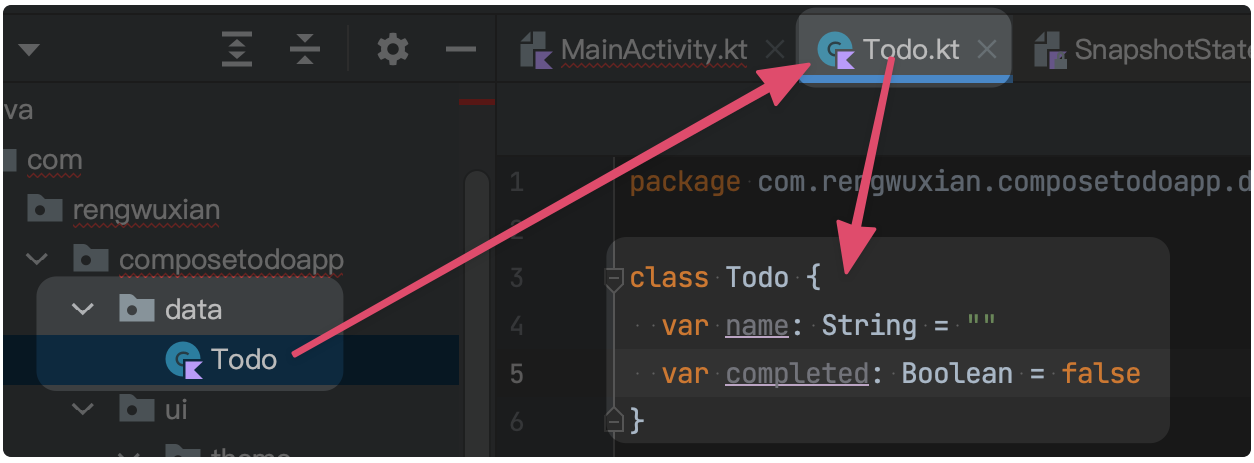
把任务 List 用实体 `Todo` 对象来显示

把待办数据加进 `Todo()` 的参数

把待办由 `String` 抽出单独的类 `Todo`：

Kotlin 复制代码

```kotlin
class Todo {
  var name: String = ""
  var completed: Boolean = false
}
```



修改 `TodoList()` 的代码：

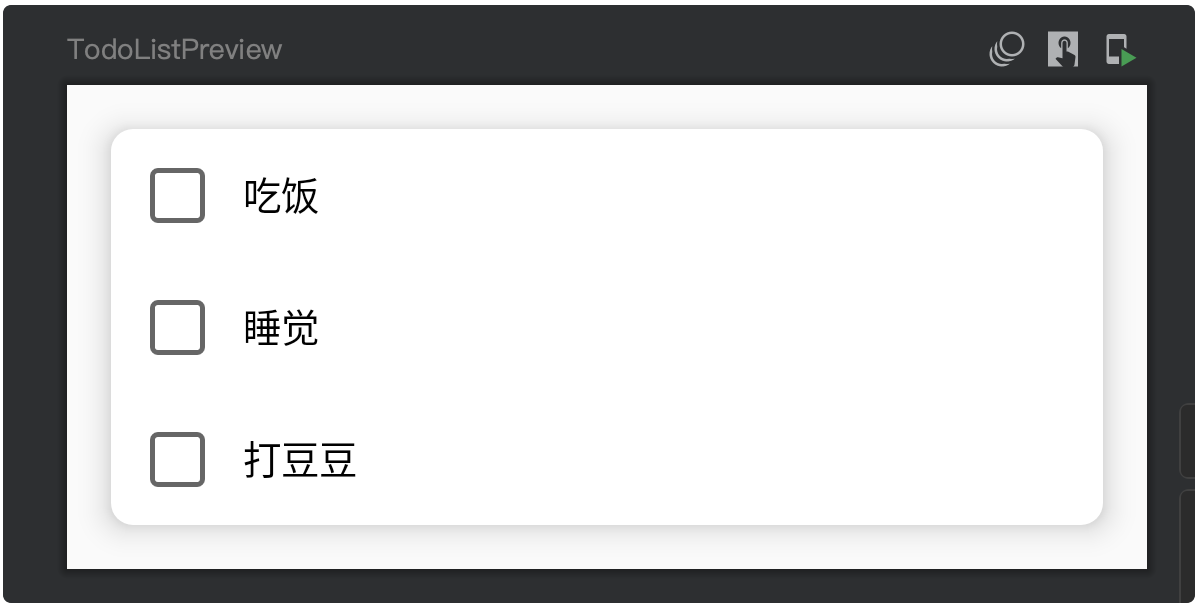
Kotlin 复制代码

```kotlin
@Composable
private fun TodoList(todos: List<Todo>) { // 1️⃣ 参数：todo: List<Todo>
  Surface(Modifier.padding(16.dp), elevation = 8.dp, shape =
RoundedCornerShape(8.dp)) {
    LazyColumn(Modifier.fillMaxWidth()) {
      items(todos) { todo ->
        Row(verticalAlignment = Alignment.CenterVertically) {
          Checkbox(checked = todo.completed, onCheckedChange = {}) // 3️⃣
todo.completed
          Text(todo.name) // 2️⃣ todo 改为 todo.name
        }
      }
    }
  }
}
```

以及 `TodoListPreview()`：

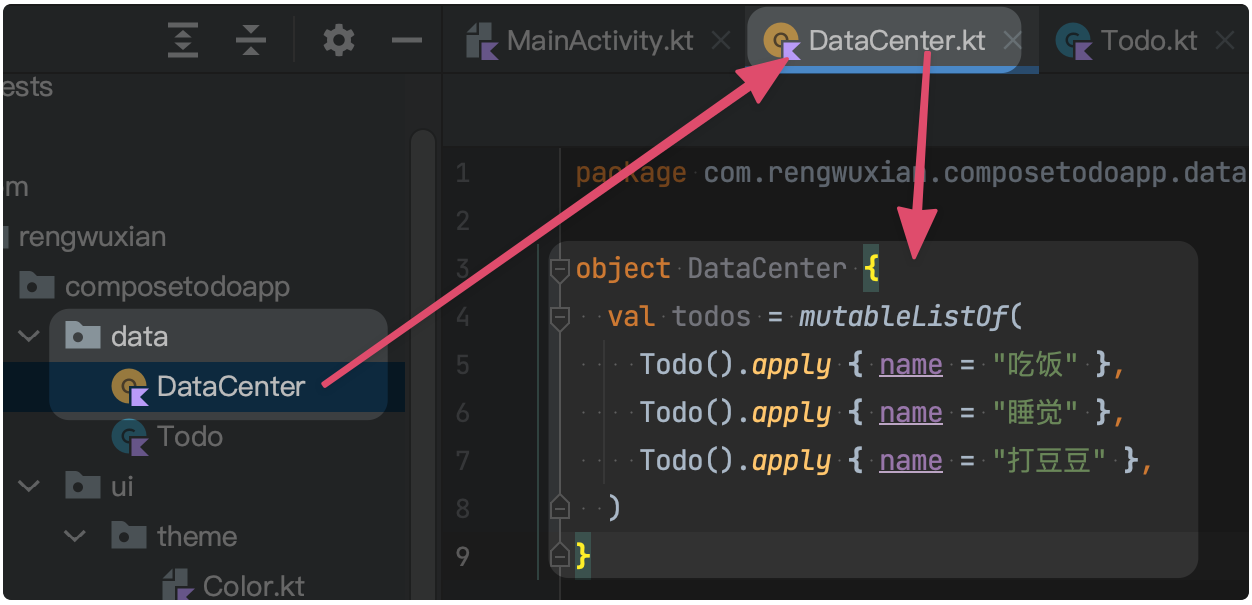
Kotlin 复制代码

```kotlin
@Preview(showBackground = true)
@Composable
fun TodoListPreview() {
  val todos = listOf(
    Todo().apply { name = "吃饭" },
    Todo().apply { name = "睡觉" },
    Todo().apply { name = "打豆豆" },
  )
  TodoList(todos)
}
```



从「数据库」加载

创建「真实」外部数据：



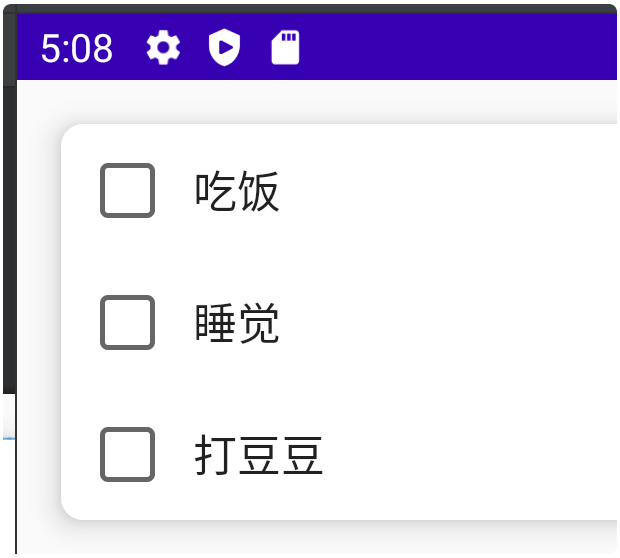
Kotlin 复制代码

```kotlin
object DataCenter {
  val todos = mutableListOf(
    Todo().apply { name = "吃饭" },
    Todo().apply { name = "睡觉" },
    Todo().apply { name = "打豆豆" },
  )
}
```

然后把它用到 `setContent()` 去：

Kotlin 复制代码

```kotlin
setContent {
  ComposeTodoAppTheme {
    TodoList(DataCenter.todos) // 解封，并且填入 DataCenter.todos
  }
}
```



点击 Checkbox 来完成待办

设置 `Checkbox()` 的监听

直接点击 `Checkbox()`，它不会改变选中状态，这是 Compose 的机制决定的。Compose 整体上就被设计为了数据和表现始终一致的数据模型，这个在 Compose 的官方文档称作是「Unidirectional Data Flow」，单向数据流。

所以，我应该去修改 `Checkbox()` 所绑定的数据的值：

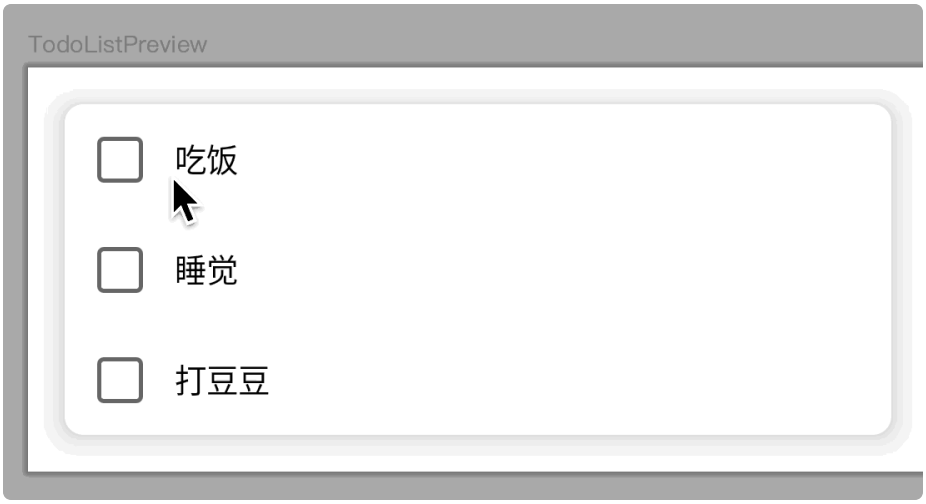
Kotlin 复制代码

```kotlin
Checkbox(checked = todo.completed, onCheckedChange = {
  todo.completed = it // 修改它
})
```

然后，还需要把 `Todo` 里的变量右边由 `=` 号赋值，改为 by `mutableStateOf()`，来让它的读写被 `MutableState` 对象代理，从而实现可被订阅：

Kotlin 复制代码

```kotlin
class Todo {
  // 包上 mutableStateOf()
  var name by mutableStateOf("")
  var completed by mutableStateOf(false)
}
```



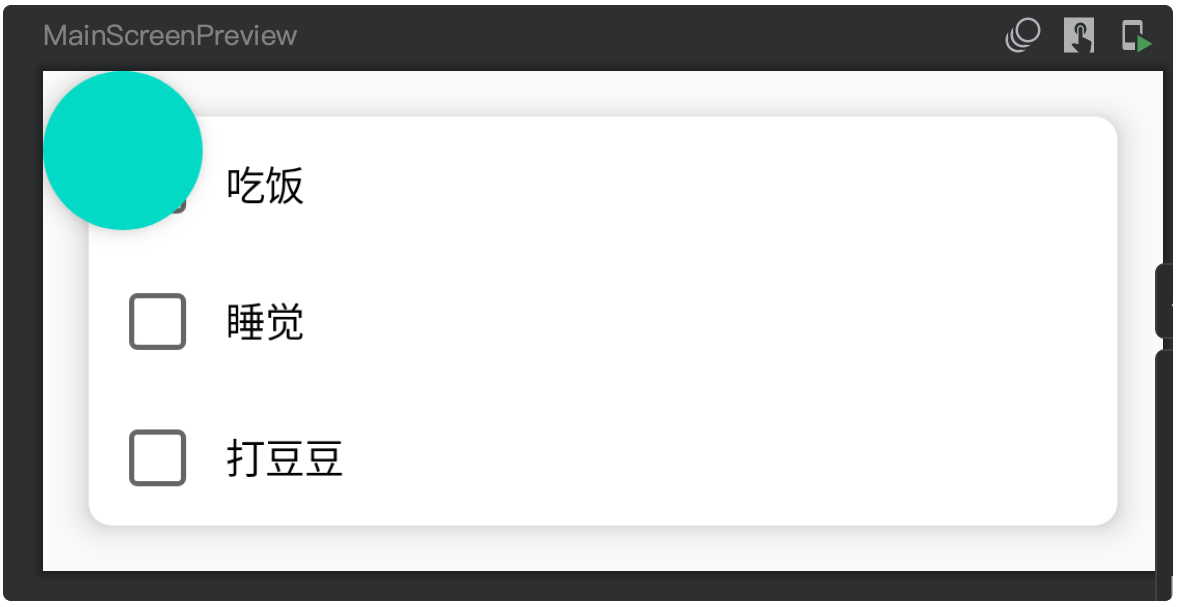
点击 + 号按钮来创建待办

把 + 号做出来

创建按钮：`FloatingActionButton()`——又是 Material Design：

Kotlin 复制代码

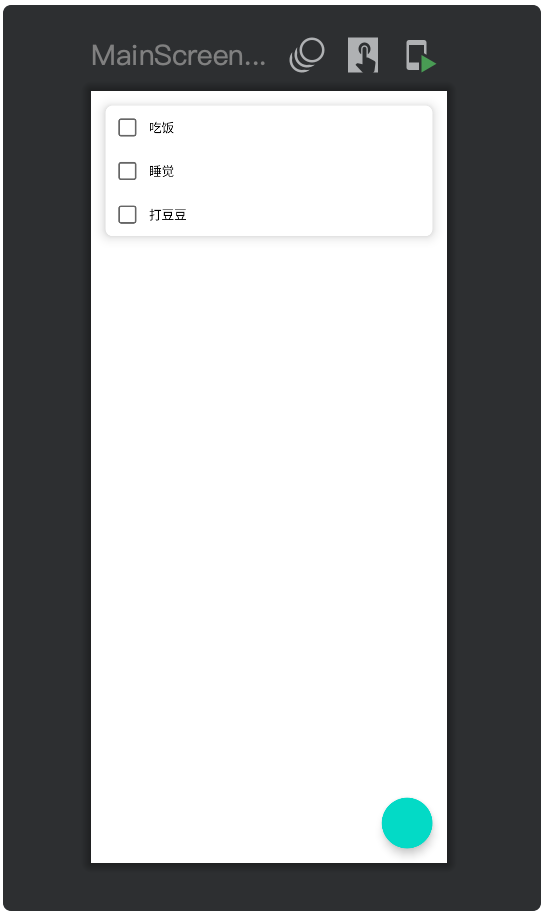
```kotlin
setContent {
  ComposeTodoAppTheme {
    TodoList(DataCenter.todos)
    FloatingActionButton(onClick = { /*TODO*/ }) { // 这个 FAB

    }
  }
}
```



摆放 FAB

`Scaffold`——还是 Material Design：

```kotlin
@Composable
private fun MainScreen() {
  Scaffold(floatingActionButton = {
    FloatingActionButton(onClick = { /*TODO*/ }) {

    }
  }) {
    TodoList(DataCenter.todos)
  }
}
```



加入加号

```kotlin
FloatingActionButton(onClick = { /*TODO*/ }) {
  Icon(Icons.Filled.Add, "添加") // 这个
}
```

设置监听

Kotlin 复制代码

```kotlin
FloatingActionButton(onClick = {
  // 弹出输入框
}
```

输入框和确认按钮

Kotlin 复制代码

```kotlin
Box(Modifier.fillMaxSize()) {
  TodoList(DataCenter.todos)
  Row(
    Modifier
      .align(Alignment.BottomStart)
      .fillMaxWidth(),
    verticalAlignment = Alignment.CenterVertically
  ) {
    TextField(value = "", onValueChange = {}, Modifier.weight(1f))
    IconButton(onClick = { /*TODO*/ }) {
      Icon(Icons.Filled.Send, contentDescription = "确认")
    }
  }
}
```

通过点击来显示和隐藏 + 号或输入框

`remember` 和 `mutableStateOf()` 的配合使用：

Kotlin 复制代码

```kotlin
@Composable
private fun MainScreen() {
  // 增加这个 inputing，讲解一下 remember
  var inputing by remember {
    mutableStateOf(false)
  }
  Scaffold(floatingActionButton = {
    // 增加这个 if 判断
    if (!inputing) {
      FloatingActionButton(onClick = {
        // 这个点击监听
        inputing = !inputing
      }) {
        Icon(Icons.Filled.Add, "添加")
      }
    }
  }) {
    Box(Modifier.fillMaxSize()) {
      TodoList(DataCenter.todos)
      // 增加这个 if 判断
      if (inputing) {
        Row(
          Modifier
            .align(Alignment.BottomStart)
            .fillMaxWidth(),
          verticalAlignment = Alignment.CenterVertically
        ) {
          TextField(value = "", onValueChange = {}, Modifier.weight(1f))
          IconButton(onClick = {
            // 这个点击监听
            inputing = !inputing
          }) {
            Icon(Icons.Filled.Send, contentDescription = "确认")
          }
        }
      }
    }
  }
}
```

加上动画

`animateFloatAsState()`：

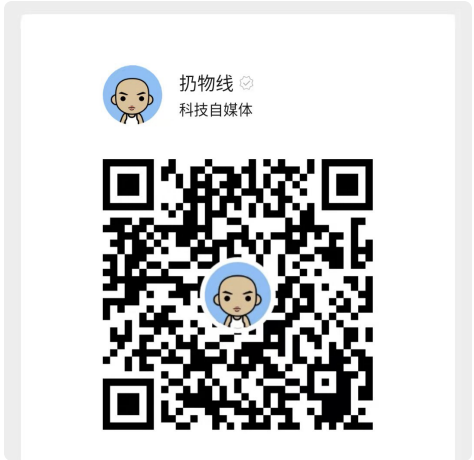
```kotlin
val animatedFabScale by animateFloatAsState(targetValue = if (inputing) 0f else 1f)
val animatedInputScale by animateFloatAsState(targetValue = if (inputing) 1f else 0f)

...

Modifier.alpha(animatedFabScale)

...

Modifier.alpha(animatedInputScale)
```

更多其他分享

欢迎关注我的哔哩哔哩、公众号和视频号，查看我的往期分享，以及第一时间获取我未来的文字、视频分享，还有我随时可能会开的在线直播！

- 官网：https://rengwuxian.com
- 哔哩哔哩：https://space.bilibili.com/27559447
- 微信公众号：

- 微信视频号：



进阶课程

对我们的 Android 全方位进阶课程或 Jetpack Compose 高级课程有兴趣的，请加助教微信咨询详情和了解最新优惠：