

# 实验十 ArkTS 及 ArkUI 综合

## 一、实验目的

1. 了解 DevEco Studio 的使用
2. 学习 ArkTS 语言
3. 编写代码
4. 编译运行
5. 在模拟器上运行

## 二、实验原理

1. 鸿蒙开发原理
2. ArkTS, ArkUI 开发原理
3. 鸿蒙应用运行原理

## 三、实验仪器材料

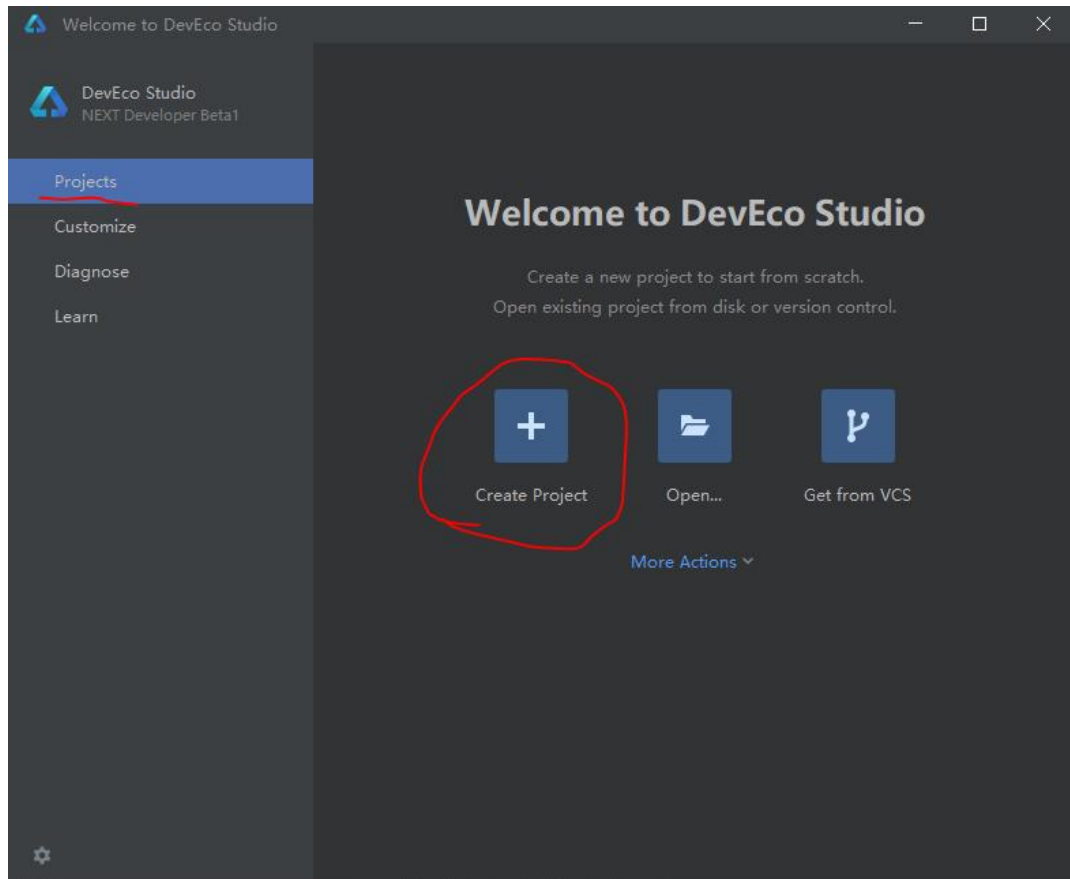
1. 计算机实训室电脑一台
2. DevEco Studio 开发环境及鸿蒙手机模拟器

## 四、实验步骤

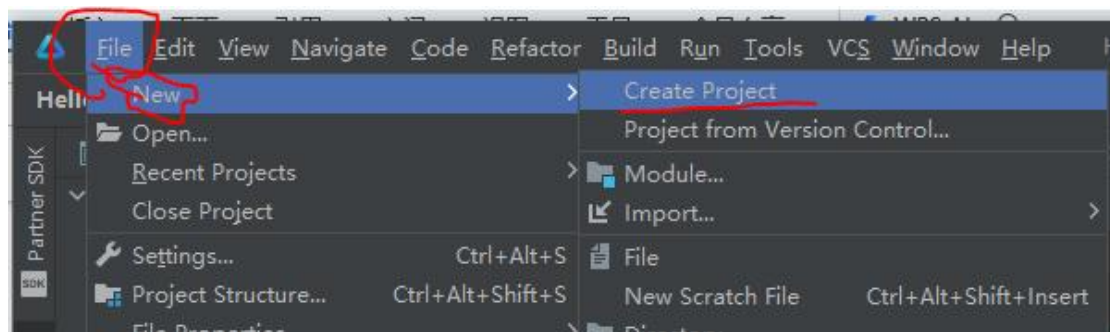
参考：

<https://developer.huawei.com/consumer/cn/training/course/slightMooc/C101717496870909384>

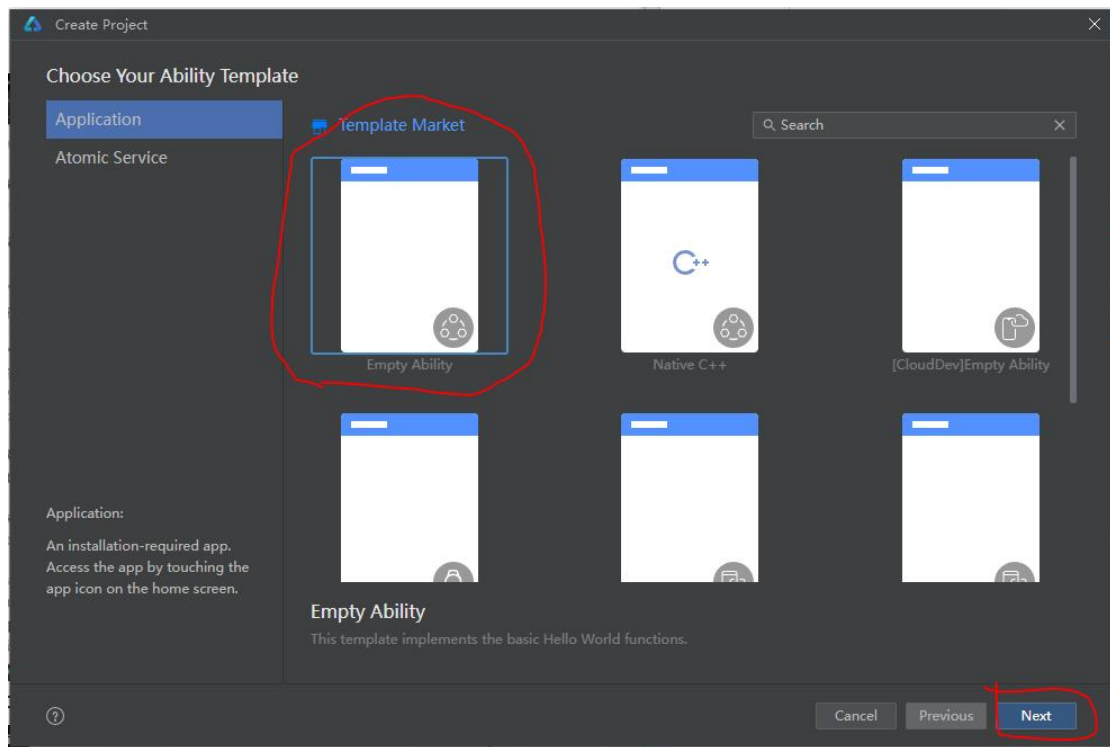
1. 打开 DevEco Studio, 点击 Create Project 创建工程。



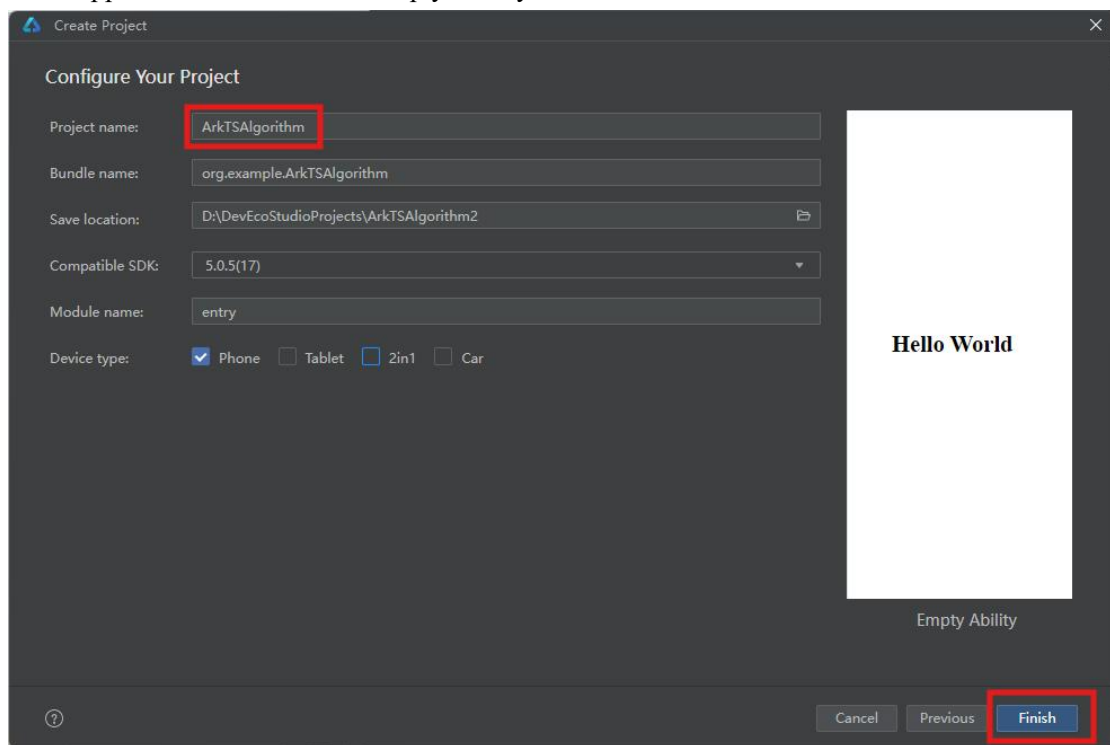
如果已经打开了一个工程，请在菜单栏选择 File > New > Create Project 来创建一个新工程。



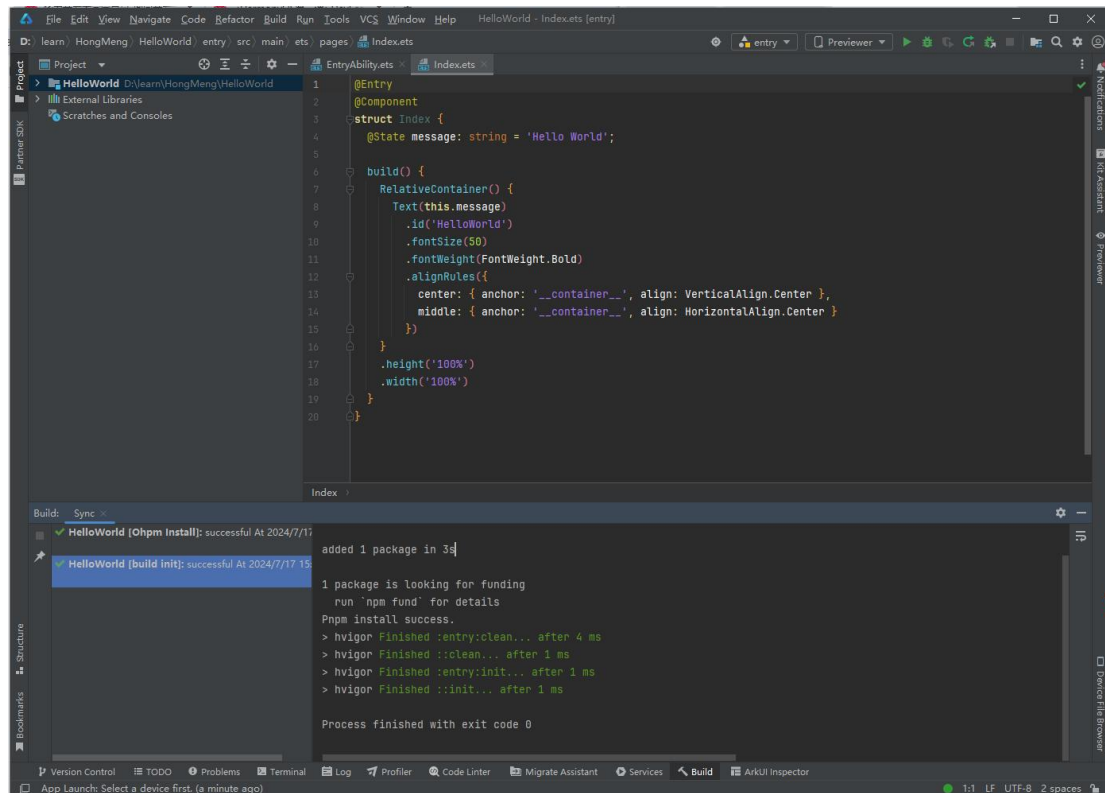
点击“Create Project”，进入：



选择 Application，然后选择“Empty Ability”，点击 Next 按钮，进入：



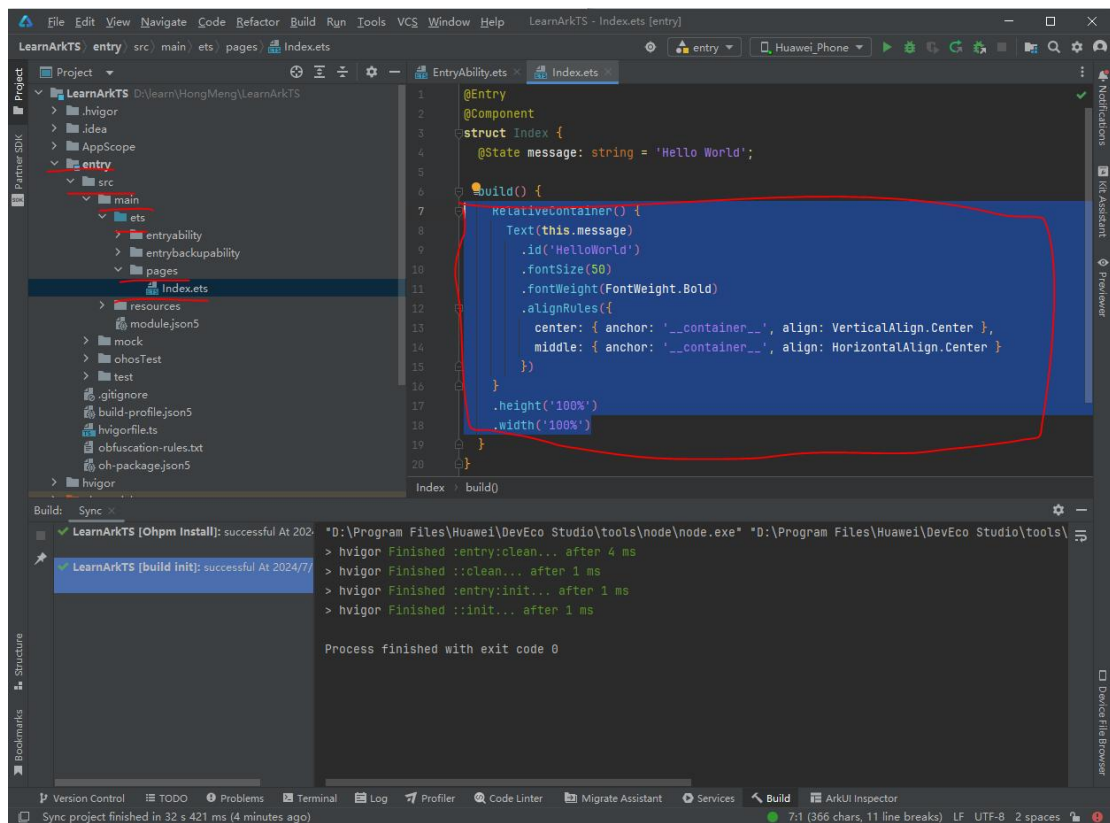
配置好项目名称（如 **ArkTSAAlgorithm**），存放位置（上图是放在 D 盘某个目录下），设备类型（上图除了 Car 全选了）等，然后点击 Finish 按钮，进入到开发界面：



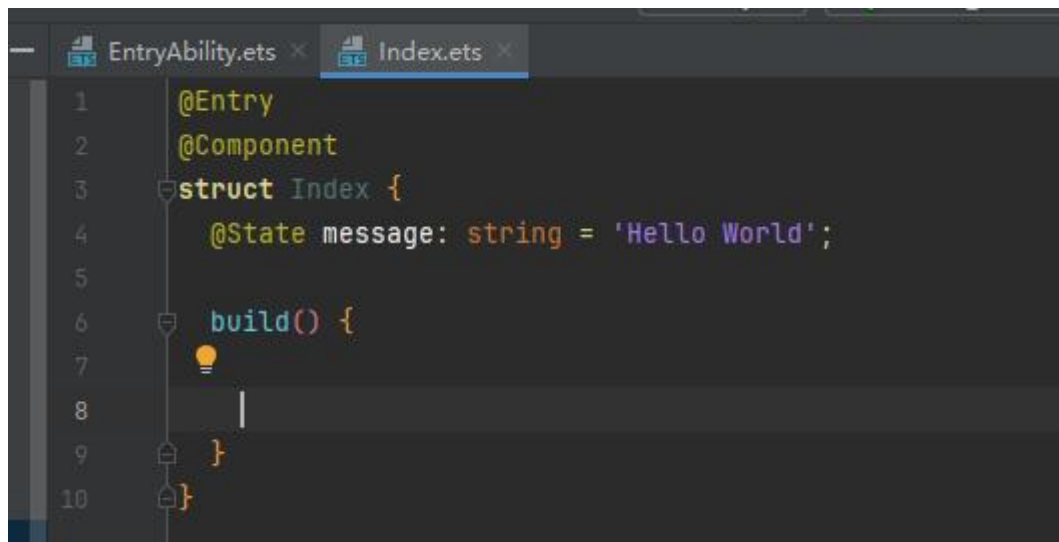
项目创建成功。

## 2. 清理代码

找到 `entry > src > main > ets > pages` 里面的 `Index.ets` 文件，将 `build() {}` 的 `{}` 里面的代码清空。

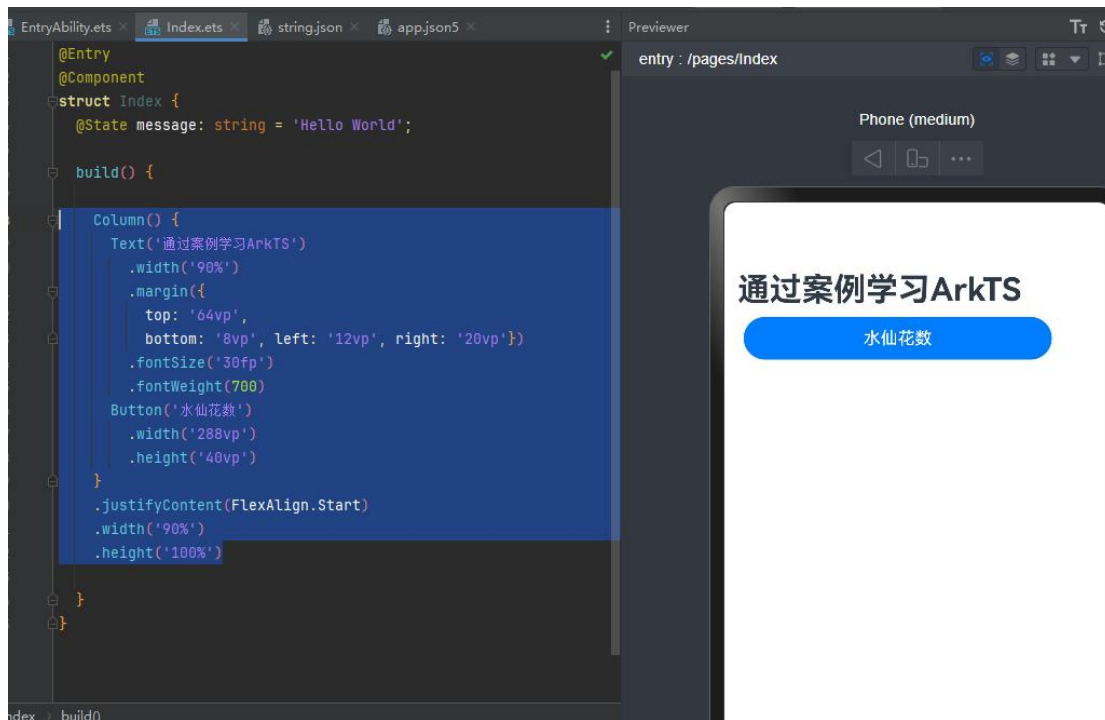


清空之后：



3. 添加一个按钮，点击按钮打开一个 Dialog 窗口，以计算 1000 以内的水仙花数为例

1) 在 Index.ets 的 build() 中加入代码：



代码片段：

```
Column() {
```

```
Text('通过案例学习 ArkTS')
```

```
.width('90%')
```

```
.margin({
```

```
top: '64vp',
```

```
bottom: '8vp', left: '12vp', right: '20vp'})
```

```
.fontSize('30fp')
```

```
.fontWeight(700)
```

```
Button('水仙花数')
```

```
.width('288vp')
```

```
.height('40vp')
```

```
}
```

```
.justifyContent(FlexAlign.Start)
```

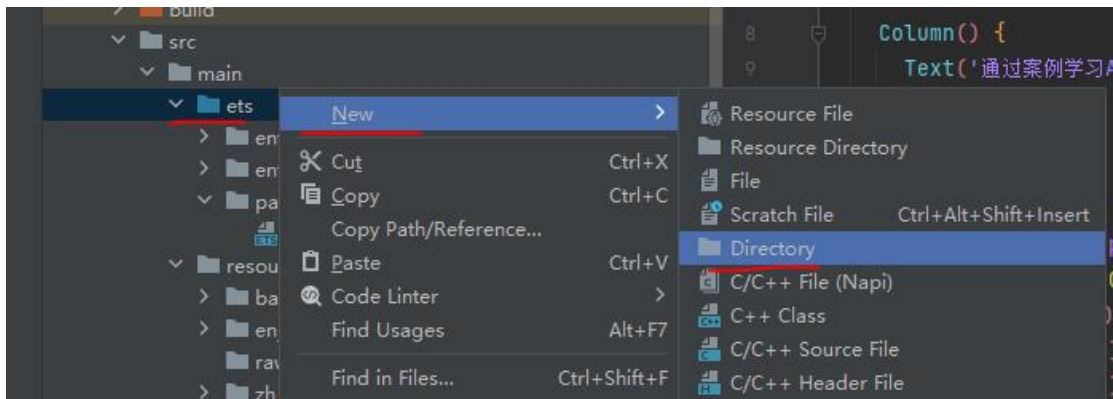
```
.width('100%')
```

```
.height('100%')
```

可以看到 Previewer 中的效果，现在有了一个标题和一个按钮。

## 2) 添加一个 Dialog 对话框

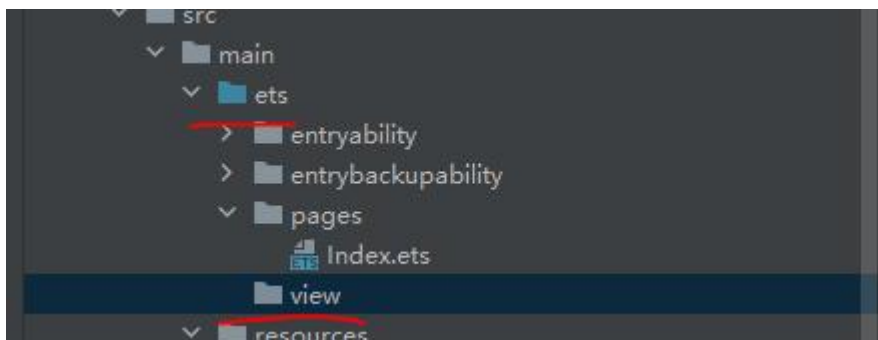
创建一个新的目录，在 `ets` 目录上点击鼠标右键，选择 `New > Directory`:



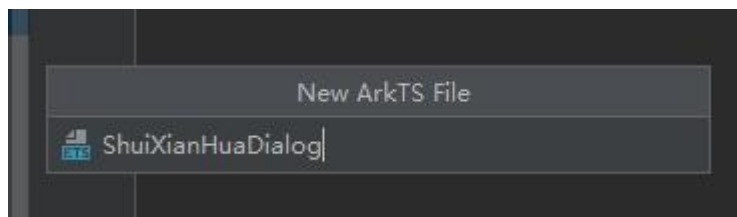
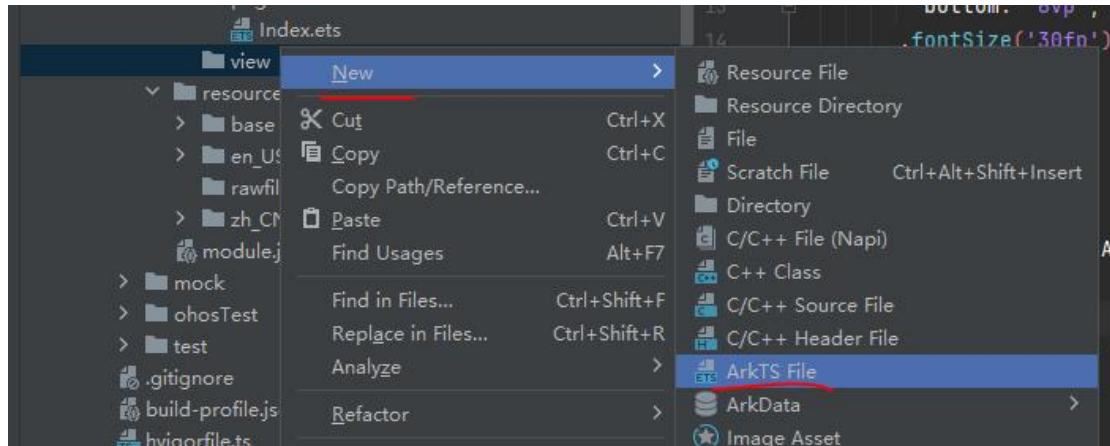
目录命名为 `view`:



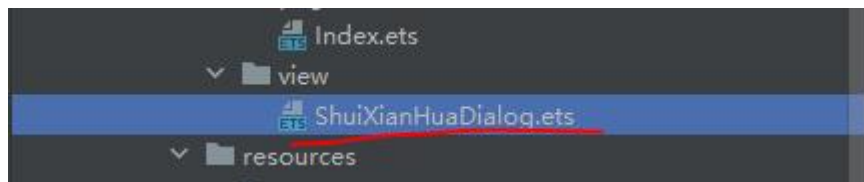
回车后，可以看到新的 `view` 目录创建成功:



在 view 目录下，新建一个文件（点击鼠标右键，New > ArkTS File），命名为 ShuiXianHuaDialog.ets：



回车后，文件创建成功：



在文件中添加代码：



```
EntryAbility.ets x Index.ets x ShuiXianHuaDialog.ets x
1  @Preview
2  @CustomDialog
3  export struct ShuiXianHuaDialog {
4      IsPalindromicStringCustomDialogController?: CustomDialogController;
5
6      build() {
7          Column() {
8              Column() {
9                  Text('ArkTS实例')
10                     .height('40vp')
11                     .font({ size: $r('sys.float.ohos_id_text_size_headline8') })
12                     .fontColor($r('sys.color.ohos_id_color_text_primary'))
13                     .margin({ top: '8vp' })
14                  Text('1000以内的水仙花数判断')
15                     .font({ size: $r('sys.float.ohos_id_text_size_body2') })
16                     .fontColor($r('sys.color.ohos_id_color_text_secondary'))
17                     .margin({ left: '10vp' })
18              }
19              .alignItems(HorizontalAlign.Center)
20              .width('100%')
21              .height('72vp')
22
23              Text('10, 300, 400, 700')
24                 .font({ size: $r('sys.float.ohos_id_text_size_body1') })
25                 .fontColor($r('sys.color.ohos_id_color_text_primary'))
26                 .margin({ top: '24vp' })
27          }
28          .alignItems(HorizontalAlign.Center)
29          .padding({
30              left: '24vp',
31              right: '24vp',
32              bottom: '24vp'
33          })
34      }
35  }
36 }
```

代码片段：

@Preview

@CustomDialog

export struct ShuiXianHuaDialog {

IsPalindromicStringCustomDialogController?: CustomDialogController;

```
build() {
```

```
  Column() {
```

```
    Column() {
```

```
      Text('ArkTS 实例')
```

```
        .height('40vp')
```

```
        .font({ size: $r('sys.float.ohos_id_text_size_headline8') })
```

```
        .fontColor($r('sys.color.ohos_id_color_text_primary'))
```

```
        .margin({ top: '8vp' })
```

```
      Text('1000 以内的水仙花数判断')
```

```
        .font({ size: $r('sys.float.ohos_id_text_size_body2') })
```

```
        .fontColor($r('sys.color.ohos_id_color_text_secondary'))
```

```
        .margin({ left: '10vp' })
```

```
    }
```

```
    .alignItems(HorizontalAlign.Center)
```

```
    .width('100%')
```

```
    .height('72vp')
```

```
    Text('10, 300, 400, 700')
```

```
      .font({ size: $r('sys.float.ohos_id_text_size_body1') })
```

```
      .fontColor($r('sys.color.ohos_id_color_text_primary'))
```

```
      .margin({ top: '24vp' })
```

```

    }

    .alignItems(HorizontalAlign.Center)

    .padding({

        left: '24vp',

        right: '24vp',

        bottom: '24vp'

    })

}

}

```

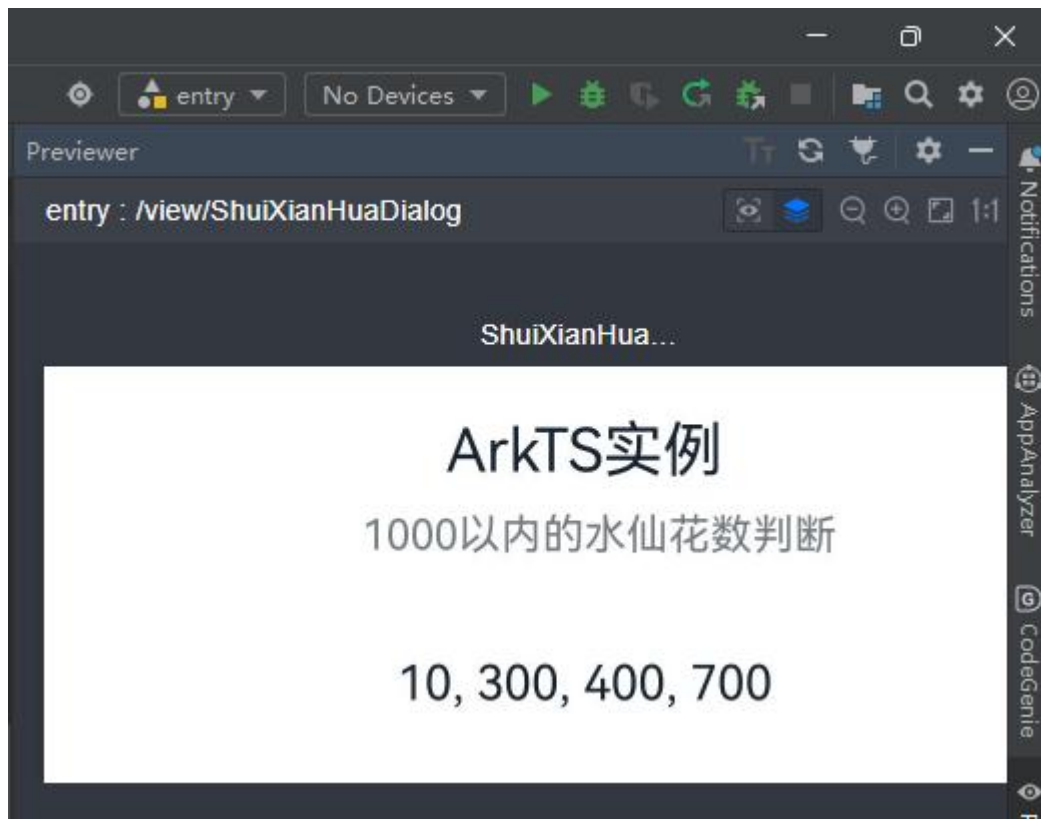
其中几点关键信息如下：

**@CustomDialog:** 装饰器，声明这是一个自定义对话框

**@Preview:** 预览装饰器，可在开发环境中预览效果

**CustomDialogController:** 对话框控制器，用于控制对话框的显示和隐藏

注意，此时我们显示的数字‘10,300,400,700’是硬编码的模拟数据，主要目的是为了设计界面时候占位用，方便我们看界面上现实的效果。此时看 **Previewer**，界面应该是这样的：



要计算 1000 以内的水仙花数，我们可以在 `ShuiXianHuaDialog.ets` 中添加一个函数：

```
EntryAbility.ets x Index.ets x ShuiXianHuaDialog.ets x
24 .font({ size: $r('sys.float.ohos_id_text_size_body1') })
25 .fontColor($r('sys.color.ohos_id_color_text_primary'))
26 .margin({ top: '24vp' })
27
28 }
29 .alignItems(HorizontalAlign.Center)
30 .padding({
31   left: '24vp',
32   right: '24vp',
33   bottom: '24vp'
34 })
35 }
36 }
37
38 function shuiXianHuaNumber(): number[] {
39   let result: number[] = [];
40   for (let i = 100; i < 1000; i++) {
41     let unitsDigit: number = i % 10;
42     let tenthsDigit: number = Math.floor(i / 10) - Math.floor(i / 100) * 10;
43     let hundredthsDigit: number = Math.floor(i / 100);
44     if (i === unitsDigit * unitsDigit * unitsDigit + tenthsDigit * tenthsDigit *
45         hundredthsDigit * hundredthsDigit * hundredthsDigit) {
46       result.push(i);
47     }
48   }
49   return result;
50 }
```

代码片段：

```
function shuiXianHuaNumber(): number[] {
```

```
  let result: number[] = [];
```

```
  for (let i = 100; i < 1000; i++) {
```

```
    let unitsDigit: number = i % 10;
```

```
    let tenthsDigit: number = Math.floor(i / 10) - Math.floor(i / 100) * 10;
```

```
    let hundredthsDigit: number = Math.floor(i / 100);
```

```
    if (i === unitsDigit * unitsDigit * unitsDigit + tenthsDigit * tenthsDigit *
```

```
        tenthsDigit +
```

```

        hundredthsDigit * hundredthsDigit * hundredthsDigit) {

    result.push(i);

}

}

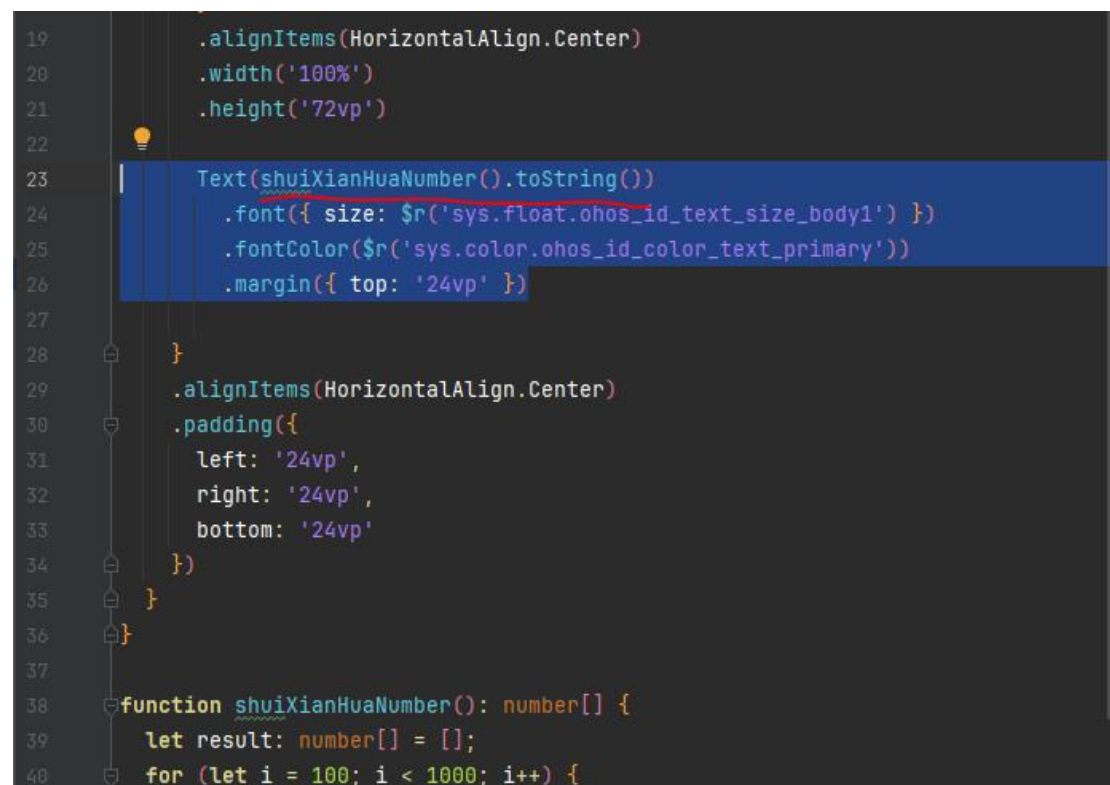
return result;

}

```

注意，这里用到了函数，数组，分支，循环等 ArkTS 的语法。

然后，我们在显示数字那里调用这个函数，显示出正确的水仙花数：



```

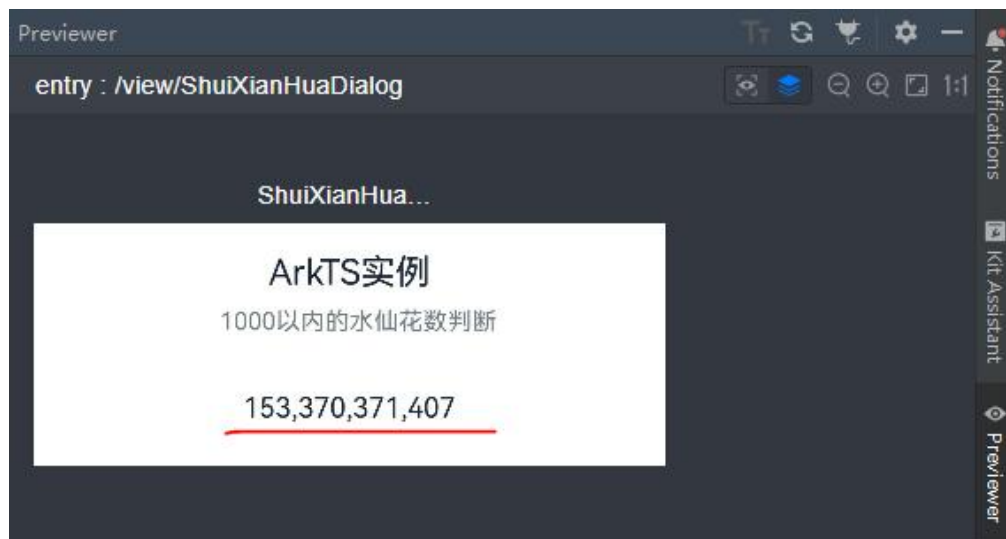
19      .alignItems(HorizontalAlign.Center)
20      .width('100%')
21      .height('72vp')
22
23      Text(shuiXianHuaNumber().toString())
24          .font({ size: $r('sys.float.ohos_id_text_size_body1') })
25          .fontColor($r('sys.color.ohos_id_color_text_primary'))
26          .margin({ top: '24vp' })
27
28  }
29  .alignItems(HorizontalAlign.Center)
30  .padding({
31      left: '24vp',
32      right: '24vp',
33      bottom: '24vp'
34  })
35  }
36  }
37
38  function shuiXianHuaNumber(): number[] {
39      let result: number[] = [];
40      for (let i = 100; i < 1000; i++) {

```

也就是说，将 `Text('10, 300, 400, 700')` 代码改为：

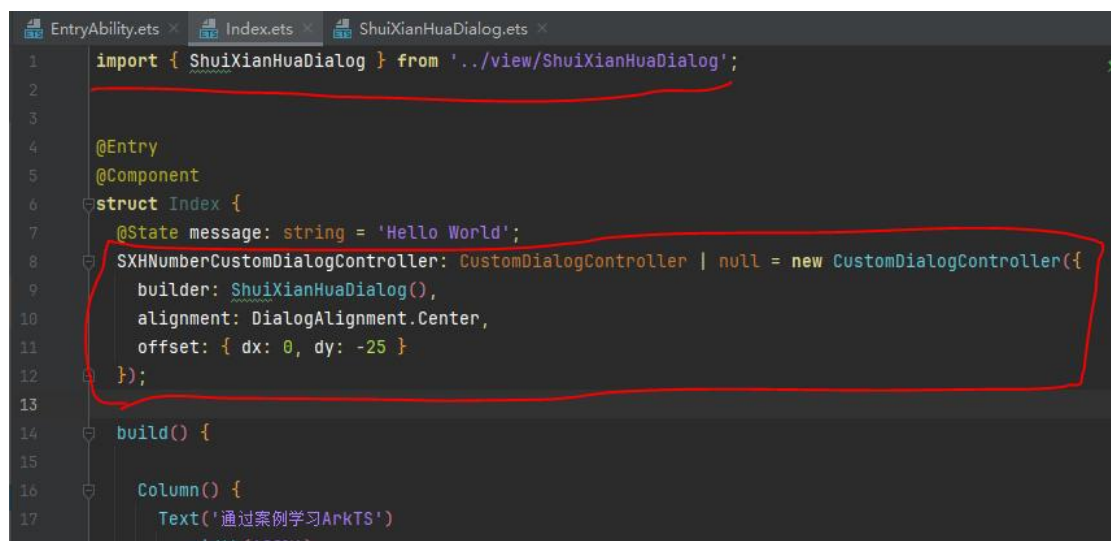
```
Text(shuiXianHuaNumber().toString())
```

此时，看 Previewer 那里，显示出正确的水仙花数：



3) 加入代码去实现：点击按钮，弹出 ShuiXianHuaDialog 对话框的效果

修改 index.ets，加入这两段代码：



第一句 import 是导入我们创建的 ShuiXianHuaDialog 对话框：

```
import { ShuiXianHuaDialog } from '../view/ShuiXianHuaDialog';
```

第二段是为 ShuiXianHuaDialog 创建一个 CustomDialogController，用于在点击事件的时候响应并打开对话框。代码片段：

```
SXHNumberCustomDialogController: CustomDialogController | null = new  
CustomDialogController({  
  
    builder: ShuiXianHuaDialog(),  
  
    alignment: DialogAlignment.Center,  
  
    offset: { dx: 0, dy: -25 }  
  
});
```

此时，我们再添加按钮的 onClick 事件，打开对话框：



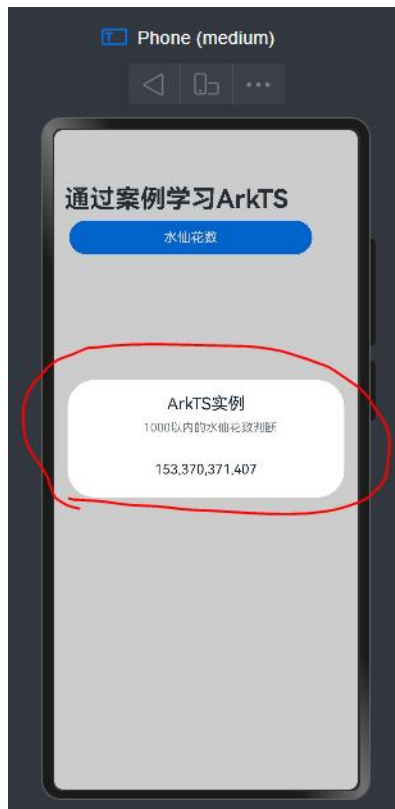
```
build() {  
    Column() {  
        Text('通过案例学习ArkTS')  
            .width('90%')  
            .margin({  
                top: '64vp',  
                bottom: '8vp', left: '12vp', right: '20vp'})  
            .fontSize('30fp')  
            .fontWeight(700)  
        Button('水仙花数')  
            .width('288vp')  
            .height('40vp')  
            .onClick(() => {  
                this.SXHNumberCustomDialogController?.open();  
            })  
    }  
    .justifyContent(FlexAlign.Start)  
    .width('90%')  
    .height('100%')  
}
```

代码片段：

```
.onClick() => {  
  
    this.SXHNumberCustomDialogController?.open();  
  
})
```

此时，到 Previewer 中查看，点击按钮“水仙花数”：





可以看到对话框正确弹出，点击空白处对话框消失。

#### 4. 打印九九乘法表的对话框

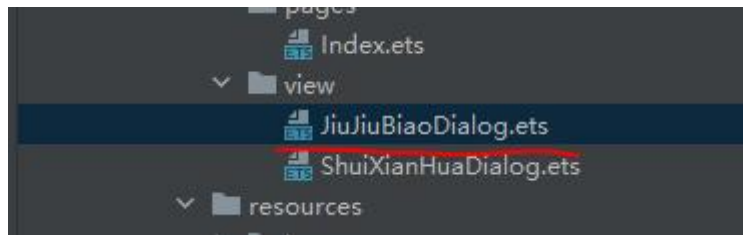
基于上面的经验，再做一个按钮，一个对话框，打印出来一个九九乘法表。

##### 1) 添加按钮

```
Button('打印九九乘法表')  
  
    .width('288vp')  
  
    .height('40vp')  
  
    .margin({ top: '10vp' })  
  
    .onClick() => {  
  
    }  
}
```

## 2) 增加对话框

在 view 目录下创建一个新的文件 JiuJiuBiaoDialog.ets



在 JiuJiuBiaoDialog.ets 中添加代码:

```
@Preview
```

```
@CustomDialog
```

```
export struct JiuJiuBiaoDialog {
```

```
  IsPalindromicStringCustomDialogController?: CustomDialogController;
```

```
  build() {
```

```
    Column() {
```

```
      Column() {
```

```
        Text('ArkTS 实例')
```

```
        .height('40vp')
```

```
        .font({ size: $r('sys.float.ohos_id_text_size_headline8') })
```

```
        .fontColor($r('sys.color.ohos_id_color_text_primary'))
```

```
        .fontColor($r('sys.color.ohos_id_color_text_primary'))
```

```
        .margin({ top: '8vp' })
```

```
        Text('九九乘方表')
```

```
        .font({ size: $r('sys.float.ohos_id_text_size_body2') })
```

```

        .fontColor($r('sys.color.ohos_id_color_text_secondary'))

        .margin({ left: '10vp' })

    }

    .alignItems(HorizontalAlign.Center)

    .width('100%')

    .height('72vp')

    Text('请查看 Log 中打印出来的结果')

    .font({ size: $r('sys.float.ohos_id_text_size_body1' )})

    .fontColor($r('sys.color.ohos_id_color_text_primary'))

    .margin({ top: '24vp' })

    }

    .alignItems(HorizontalAlign.Center)

    .padding({

        left: '24vp',

        right: '24vp',

        bottom: '24vp'

    })

    }

}

```

在 Previewer 中看到的是这样的：



### 3) 点击按钮打开对话框

首先，在 Index.ets 中添加一个计算九九乘方表的函数 multiplicationTable()：

```
50     }
51     .justifyContent(FlexAlign.Start)
52     .width('90%')
53     .height('100%')
54
55 }
56
57
58
59 function multiplicationTable(): string[][] {
60     let result: string[][] = [];
61     for (let i = 1; i <= 9; i++) {
62         let index: string[] = [];
63         for (let j = 1; j <= i; j++) {
64             let temp: string = j + ' * ' + i + ' = ' + i * j;
65             index.push(temp);
66         }
67         result.push(index);
68     }
69     return result;
70 }
```

代码片段：

```
function multiplicationTable(): string[][] {
```

```

let result: string[][] = [];

for (let i = 1; i <= 9; i++) {

    let index: string[] = [];

    for (let j = 1; j <= i; j++) {

        let temp: string = j + '*' + i + '=' + i * j;

        index.push(temp);

    }

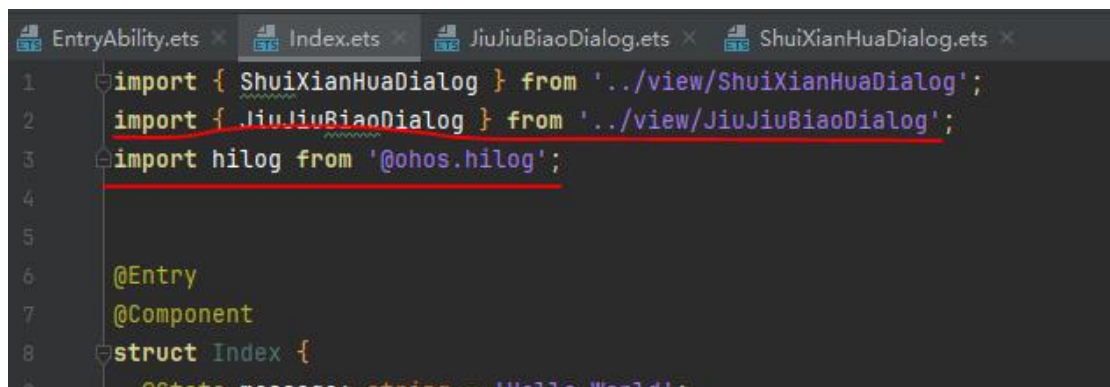
    result.push(index);

}

return result;
}

```

然后，导入依赖，一个是新创建的对话框，一个是 `hilog`，用来在 Log 中输出：



代码片段：

```

import { JiuJiuBiaoDialog } from '../view/JiuJiuBiaoDialog';

import hilog from '@ohos.hilog';

```

同样，创建一个对话框的 Controller：

```

5
6   @Entry
7   @Component
8   struct Index {
9       @State message: string = 'Hello World';
10      SXHNumberCustomDialogController: CustomDialogController | null = new CustomDialogController({
11          builder: ShuiXianHuaDialog(),
12          alignment: DialogAlignment.Center,
13          offset: { dx: 0, dy: -25 }
14      });
15
16      JiuJiuBiaoDialogController: CustomDialogController | null = new CustomDialogController({
17          builder: JiuJiuBiaoDialog(),
18          alignment: DialogAlignment.Center,
19          offset: { dx: 0, dy: -25 }
20      });
21
22      build() {
23
24          Column() {
25              Text('通过案例学习ArkTS')
26                  .width('90%')

```

代码片段：

```

JiuJiuBiaoDialogController: CustomDialogController | null = new
CustomDialogController({
    builder: JiuJiuBiaoDialog(),
    alignment: DialogAlignment.Center,
    offset: { dx: 0, dy: -25 }
});

```

现在，增加按钮的点击事件：

```
        .fontWeight(700)
        Button('水仙花数')
            .width('288vp')
            .height('40vp')
            .onClick(() => {
                this.SXHNumberCustomDialogController?.open();
            })
        Button('打印九九乘法表')
            .width('288vp')
            .height('40vp')
            .margin({ top: '10vp' })
            .onClick(() => {
                this.JiuJiuBiaoDialogController?.open();
                let result = multiplicationTable();
                hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);
                for (let index = 0; index < result.length; index++) {
                    hilog.info(0xFF00, "testTag", result[index].toString());
                }
            })
    }
    .justifyContent(FlexAlign.Start)
    .width('90%')
    .height('100%')
```

代码片段：

```
.onClick() => {

    this.JiuJiuBiaoDialogController?.open();

    let result = multiplicationTable();

    hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);

    for (let index = 0; index < result.length; index++) {

        hilog.info(0xFF00, "testTag", result[index].toString());

    }

}
```

此时，刷新 Previewer，点击打印九九乘法表按钮，可以看到对话框弹出：



Log 中打印结果:

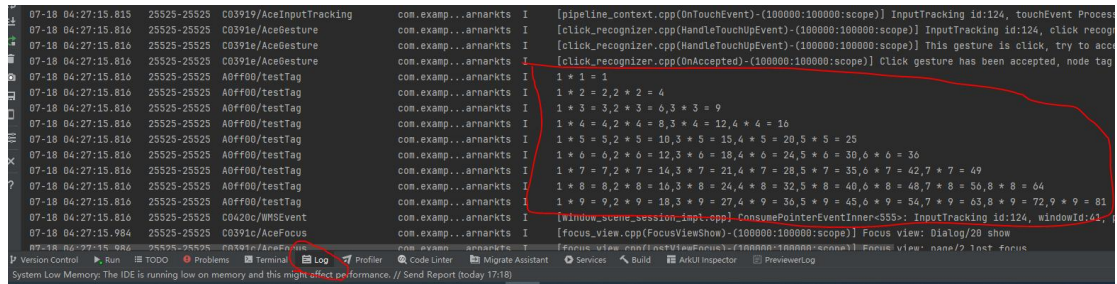
```
I    1 * 1 = 1
I    1 * 2 = 2,2 * 2 = 4
I    1 * 3 = 3,2 * 3 = 6,3 * 3 = 9
I    1 * 4 = 4,2 * 4 = 8,3 * 4 = 12,4 * 4 = 16
I    1 * 5 = 5,2 * 5 = 10,3 * 5 = 15,4 * 5 = 20,5 * 5 = 25
I    1 * 6 = 6,2 * 6 = 12,3 * 6 = 18,4 * 6 = 24,5 * 6 = 30,6 * 6 = 36
I    1 * 7 = 7,2 * 7 = 14,3 * 7 = 21,4 * 7 = 28,5 * 7 = 35,6 * 7 = 42,7 * 7 = 49
I    1 * 8 = 8,2 * 8 = 16,3 * 8 = 24,4 * 8 = 32,5 * 8 = 40,6 * 8 = 48,7 * 8 = 56,8 * 8 = 64
I    1 * 9 = 9,2 * 9 = 18,3 * 9 = 27,4 * 9 = 36,5 * 9 = 45,6 * 9 = 54,7 * 9 = 63,8 * 9 = 72,9 * 9 = 81
```

我们可以通过在模拟器中运行来查看:





此时再查看 Log，可以看到九九乘方表被正确打印了：

A screenshot of the Android Studio interface showing the Logcat window. The log contains several lines of output from the application, including timestamps, log levels, and messages. A red box highlights a section of the log where the 9x9 multiplication table is printed, showing rows from 1\*1 to 9\*9. The messages are from the 'com.examp...arnarkts' package and include log levels like 'I' (Info) and 'E' (Error). The bottom of the screenshot shows the Android Studio toolbar with icons for Version Control, Run, Problems, Terminal, Log, Profiler, Code Linter, Migrate Assistant, Services, Build, APK Inspector, and Preview/Log.

## 5. 优化代码结构

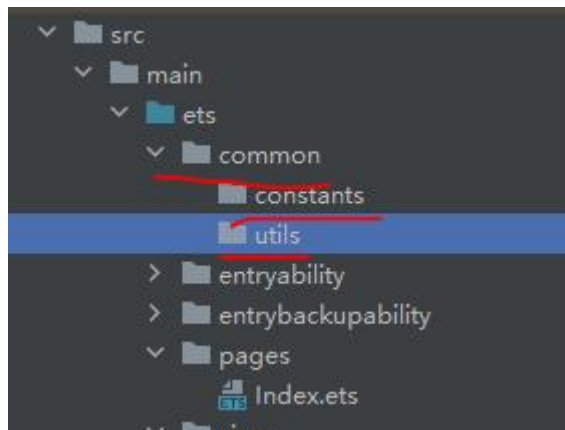
现在回头看，代码是不是比较乱？在 Index.ets 中加了一个函数计算九九乘法表，在 ShuiXianShuDialog.ets 中也加了一个函数计算水仙数。能否整合一下？

另外，如果后续的例子不断添加，按钮多了，对话框在中间会遮挡住按钮，不好看。界面需要优化一下。还有，很多界面控制的参数我们现在都是写死的数值，而且在多处出现，最好是在一个地方定义，那样的话，如需改动，只需要在一个地方改动一次即可。

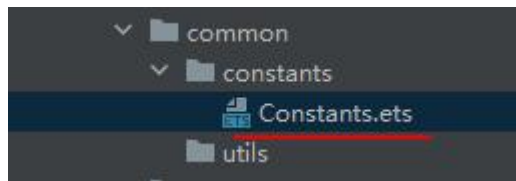
现在我们来优化一下代码。

### 常量代码优化

首先，在 main > ets 目录下，建一个新的目录 common，然后在 common 目录下建两个子目录：constants（用来存放一些常量）和 utils（用来放一些公用的函数）



在 constants 目录下，创建一个 Constants.ets 文件，用来存放变量：



在这个 Constants.ets 中加入代码：

```
export class CommonConstants {  
  
    static readonly OFFSET_X: number = 0;  
  
    static readonly OFFSET_Y: number = -25;  
  
    static readonly PERCENT_NINETY: string = '90%';  
  
    static readonly PERCENT_FULL: string = '100%';  
  
    static readonly FONT_WEIGHT_SEVEN_HUNDRED: number = 700;  
  
}
```

这样我们就定义并导出了一个类，里面包含了几个只读静态变量，然后我们到 Index.ets 中，先导入这个类（在最上面）：

```
import { CommonConstants } from '../common/constants/Constants'
```

然后我们修改 struct Index 里面的代码，注意红色字体部分是替换后的新的代码：

```
@Entry  
  
@Component  
  
struct Index {  
  
    @State message: string = 'Hello World';  
  
    SXHNumberCustomDialogController: CustomDialogController | null = new  
CustomDialogController({  
  
        builder: ShuiXianHuaDialog(),
```

```
        alignment: DialogAlignment.Center,

        offset: {
            dx: CommonConstants.OFFSET_X,
            dy: CommonConstants.OFFSET_Y }

    });
```

```
    JiuJiuBiaoDialogController: CustomDialogController | null = new
CustomDialogController({

    builder: JiuJiuBiaoDialog(),

    alignment: DialogAlignment.Center,

    offset: {
        dx: CommonConstants.OFFSET_X,
        dy: CommonConstants.OFFSET_Y }

    });
```

```
    build() {

        Column() {

            Text('通过案例学习 ArkTS')

                .width(CommonConstants.PERCENT_NINETY)

                .margin({

                    top: '64vp',

                    bottom: '8vp', left: '12vp', right: '20vp'})

                .fontSize('30fp')
```

```
.fontWeight(CommonConstants.FONT_WEIGHT_SEVEN_HUNDRED)
```

```
Button('水仙花数')
```

```
.width('288vp')
```

```
.height('40vp')
```

```
.onClick() => {
```

```
    this.SXHNumberCustomDialogController?.open();
```

```
}}
```

```
Button('打印九九乘法表')
```

```
.width('288vp')
```

```
.height('40vp')
```

```
.margin({ top: '10vp' })
```

```
.onClick() => {
```

```
    this.JiuJiuBiaoDialogController?.open();
```

```
    let result = multiplicationTable();
```

```
    hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);
```

```
    for (let index = 0; index < result.length; index++) {
```

```
        hilog.info(0xFF00, "testTag", result[index].toString());
```

```
    }
```

```
}}
```

```
}
```

```
.justifyContent(FlexAlign.Start)
```

```
.width(CommonConstants.PERCENT_FULL)
```

```

        .height(CommonConstants.PERCENT_FULL)
    }
}

```

可以看到，我们是把一些本来写为固定数字或字符串的常量用导入的 CommonConstants 里面的静态变量替换掉了。这样的好处是，如果将来需要有变化，只需要改动 Constants.ets 文件中的值即可。

此时，刷新 Previewer，没有变化。

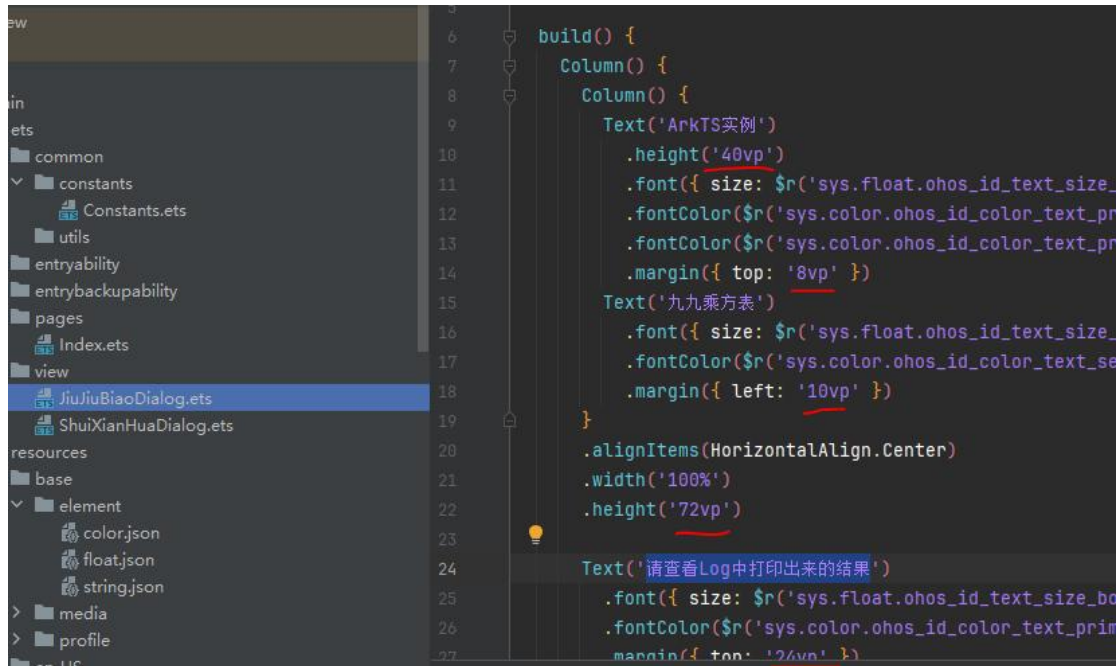
对于另外一些常量，如下图标出的数值，也可以用一些预定义的常量来表示：

```

column() {
    Text('通过案例学习ArkTS')
        .width(CommonConstants.PERCENT_NINETY)
        .margin({
            top: '64vp',
            bottom: '8vp', left: '12vp', right: '20vp'})
        .fontSize('30fp')
        .fontWeight(CommonConstants.FONT_WEIGHT_SEVEN_HUNDRED)
    Button('水仙花数')
        .width('288vp')
        .height('40vp')
        .onClick(() => {
            this.SXHNNumberCustomDialogController?.open();
        })
    Button('打印九九乘法表')
        .width('288vp')
        .height('40vp')
        .margin({ top: '10vp' })
        .onClick(() => {
            this.JiuJiuBiaoDialogController?.open();
        })
}

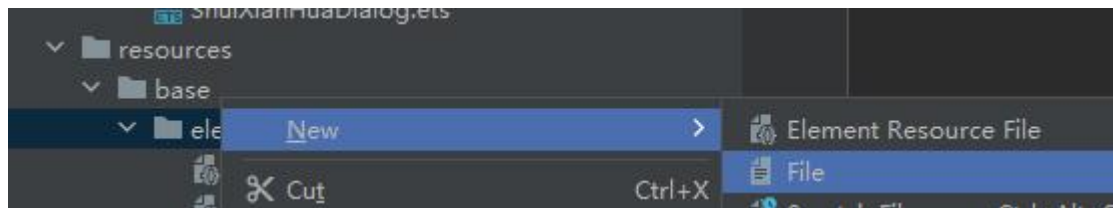
```

上图是 Index.ets 文件。

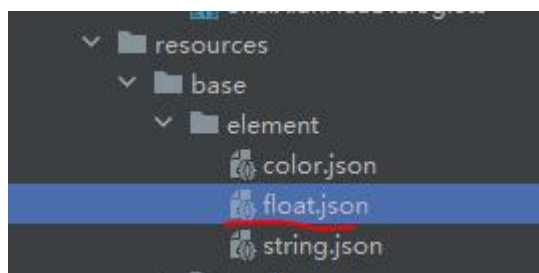


上图是 JiuJiuBiaoDialog.ets 文件。

通常的做法是，我们先在 resources > base > element 下面创建一个 json 文件：



输入窗口中直接输入 float.json:



然后我们在这里定义一个对象，里面是一个对象数组（为了方便，我把本次实验要用到的所有的值都放在这里了）：

```
{
  "float": [
    {
```

```
"name": "text_margin_top",
```

```
"value": "64vp"
```

```
},
```

```
{
```

```
"name": "text_margin_bottom",
```

```
"value": "8vp"
```

```
},
```

```
{
```

```
"name": "text_margin_left",
```

```
"value": "12vp"
```

```
},
```

```
{
```

```
"name": "text_margin_right",
```

```
"value": "20vp"
```

```
},
```

```
{
```

```
"name": "text_font_size",
```

```
"value": "30fp"
```

```
},
```

```
{
```

```
"name": "button_width",
```

```
"value": "288vp"
```



```
},
```

```
{
```

```
  "name": "button_height",
```

```
  "value": "40vp"
```

```
},
```

```
{
```

```
  "name": "button_margin_top",
```

```
  "value": "12vp"
```

```
},
```

```
{
```

```
  "name": "button_margin_bottom",
```

```
  "value": "16vp"
```

```
},
```

```
{
```

```
  "name": "dialog_text_margin_left",
```

```
  "value": "10vp"
```

```
},
```

```
{
```

```
  "name": "dialog_text_height",
```

```
  "value": "72vp"
```

```
},
```

```
{
```

```

    "name": "dialog_text_margin_top",

    "value": "8vp"

  },

  {

    "name": "dialog_padding",

    "value": "24vp"

  },

  {

    "name": "dialog_button_width",

    "value": "296vp"

  },

  {

    "name": "dialog_textInput_height",

    "value": "48vp"

  }

]

}

```

以这样的格式放在这里的好处是，后面的这些值都可以通过使用`$r('app.float.xxx')`访问到，也就是说，比如使用`$r('app.float.dialog_textInput_height')`，就得到“48vp”。

回到 `Index.ets`，再次改动 `build()` 里面的部分代码（红色字体）：

```

build() {

```

```
Column() {
```

```
Text('通过案例学习 ArkTS')
```

```
.width(CommonConstants.PERCENT_NINETY)
```

```
.margin({
```

```
top: $r('app.float.text_margin_top'),
```

```
bottom: $r('app.float.text_margin_bottom'),
```

```
left: $r('app.float.text_margin_left'),
```

```
right: $r('app.float.text_margin_right')
```

```
})
```

```
.fontSize($r('app.float.text_font_size'))
```

```
.fontWeight(CommonConstants.FONT_WEIGHT_SEVEN_HUNDRED)
```

```
Button('水仙花数')
```

```
.width($r('app.float.button_width'))
```

```
.height($r('app.float.button_height'))
```

```
.onClick() => {
```

```
this.SXHNumberCustomDialogController?.open();
```

```
})
```

```
Button('打印九九乘法表')
```

```
.width($r('app.float.button_width'))
```

```
.height($r('app.float.button_height'))
```

```
.margin({ top: $r('app.float.button_margin_top') })
```

```
.onClick() => {
```

```
this.JiuJiuBiaoDialogController?.open();
```

```
let result = multiplicationTable();
```

```
hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);
```

```
for (let index = 0; index < result.length; index++) {
```

```
    hilog.info(0xFF00, "testTag", result[index].toString());
```

```
}
```

```
}}
```

```
}
```

```
.justifyContent(FlexAlign.Start)
```

```
.width(CommonConstants.PERCENT_FULL)
```

```
.height(CommonConstants.PERCENT_FULL)
```

```
}
```

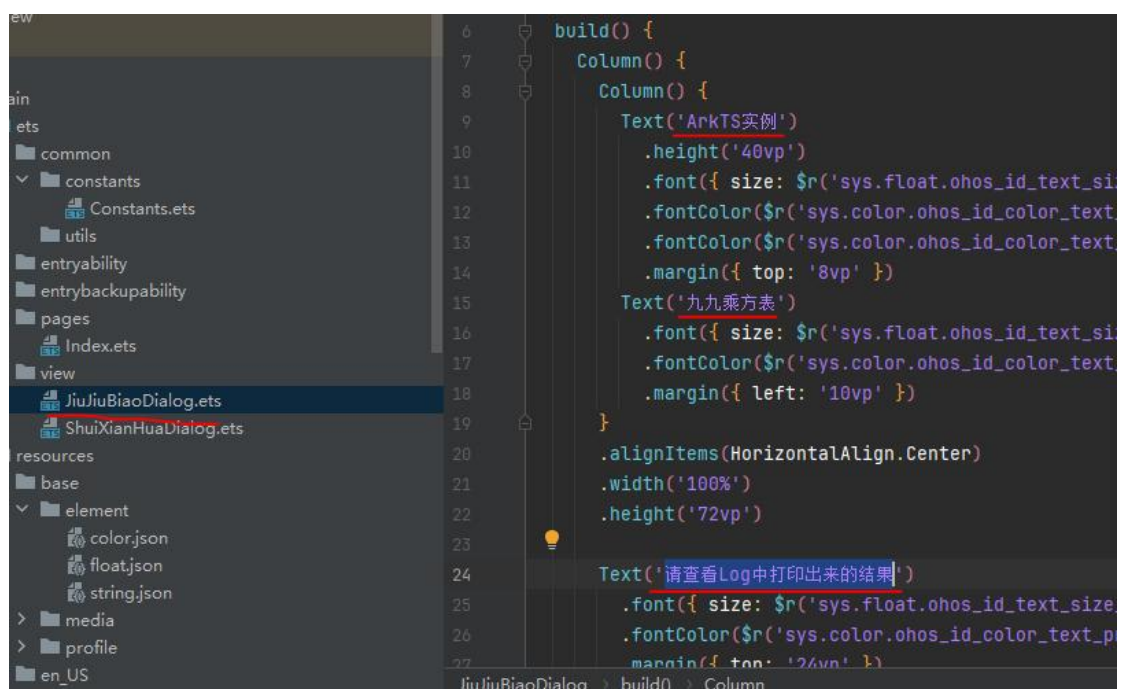
刷新 **Previewer**，没有变化。这样做的好处就是更规范，然后改动一处即可。

同样，对于这些字符串，最好也是定义在 **resources** 里面，这样一旦需要改动，直接去 **resources** 下面的 **string.json** 文件里面改动，而不必非要到代码中去查找。

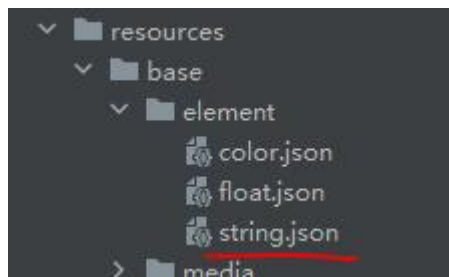
```

Column() {
  Text('通过案例学习ArkTS')
    .width(CommonConstants.PERCENT_NINETY)
    .margin({
      top: $r('app.float.text_margin_top'),
      bottom: $r('app.float.text_margin_bottom'),
      left: $r('app.float.text_margin_left'),
      right: $r('app.float.text_margin_right')
    })
    .fontSize($r('app.float.text_font_size'))
    .fontWeight(CommonConstants.FONT_WEIGHT_SEVEN_HUNDRE
  Button('水仙花数')
    .width($r('app.float.button_width'))
    .height($r('app.float.button_height'))
    .onClick(() => {
      this.SXHNumberCustomDialogController?.open();
    })
  Button('打印九九乘法表')
    .width($r('app.float.button_width'))
    .height($r('app.float.button_height'))
    .margin({ top: $r('app.float.button_margin_top') })
    .onClick(() => {
      this.JiuJiuBiaoDialogController?.open();
      let result = multiplicationTable();
    })
}

```



在小的项目中可能查找无所谓，但是在大的项目，需要养成习惯，把字符串相关的定义都放在这个 `string.json` 文件中，然后通过使用 `$r('app.string.xxx')` 来获取 `xxx` 对应的字符串常量：



我们把 `string.json` 里面的代码替换为：（同样，为了方便，把所有需要用到的都放在这里了）

```
{  
  "string": [  
    {  
      "name": "module_desc",  
      "value": "模块描述"  
    },  
    {  
      "name": "EntryAbility_desc",  
      "value": "description"  
    },  
    {  
      "name": "EntryAbility_label",  
      "value": "ArkTS 开发案例"  
    },  
    {  
      "name": "ArkTS_Development_Case",
```

```
"value": "ArkTS 开发案例"
```

```
},
```

```
{
```

```
"name": "DaffodilsNumber",
```

```
"value": "打印水仙花数"
```

```
},
```

```
{
```

```
"name": "MultiplicationTable",
```

```
"value": "打印九九乘法表"
```

```
},
```

```
{
```

```
"name": "IsPalindromicString",
```

```
"value": "判断字符串是否为回文字符串"
```

```
},
```

```
{
```

```
"name": "StringReversal",
```

```
"value": "字符串反转"
```

```
},
```

```
{
```

```
"name": "IsLeapYear",
```

```
"value": "判断是否为闰年"
```

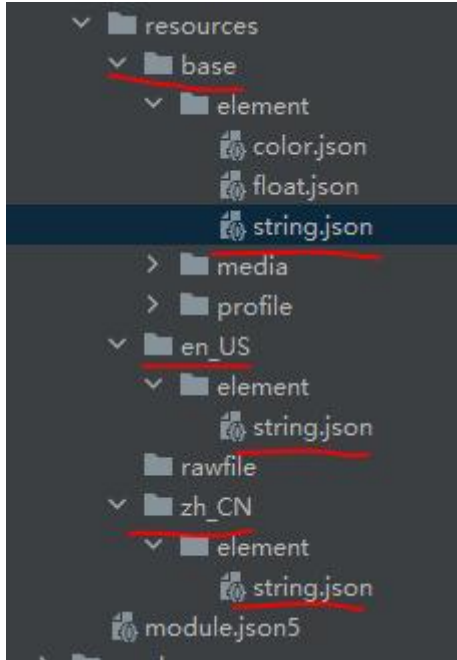
```
},
```

```
{  
  "name": "DaffodilsNumberTitle",  
  "value": "水仙花数"  
},  
  
{  
  "name": "Judgment_of_leap_year",  
  "value": "闰年判断"  
},  
  
{  
  "name": "Judgment_of_palindromic_string",  
  "value": "回文字符串判断"  
},  
  
{  
  "name": "Multiplication_Table_Title",  
  "value": "九九乘法表"  
},  
  
{  
  "name": "check_info_in_log",  
  "value": "请在日志打印中查看"  
}  
]  
}
```



比如，使用`$r('app.string.check_info_in_log')`即可获得值“请在日志打印中查看”。

可能有同学注意到，在 `resources` 下面，还有两个目录：`en_US` 和 `zh_CN`，而这两个目录下面都有 `element` 子目录，里面有 `string.json` 文件：



这个实际上是为了适应国际化的需求而做的。从名字上可以看出，`en_US` 针对的是英语市场，而 `zh_CN` 是针对中国市场。如果用户的手机是英文的鸿蒙系统，那么使用 `app.string.xxx` 的时候，就会去 `en_US > element > string.json` 中查找。而如果既不是 `en_US`，也不是 `zh_CN`，就到 `base > element` 下面去找 `string.json`。可以理解为这里的是缺省值。

所以，我们也贴一下英文的：

```
{
  "string": [
    {
      "name": "module_desc",
      "value": "module description"
    },
    {
```

```
"name": "EntryAbility_desc",
```

```
"value": "description"
```

```
},
```

```
{
```

```
"name": "EntryAbility_label",
```

```
"value": "ArkTS Development Case"
```

```
},
```

```
{
```

```
"name": "ArkTS_Development_Case",
```

```
"value": "ArkTS Development Case"
```

```
},
```

```
{
```

```
"name": "DaffodilsNumber",
```

```
"value": "Print the number of daffodils within 1000"
```

```
},
```

```
{
```

```
"name": "MultiplicationTable",
```

```
"value": "Print Multiplication Tables"
```

```
},
```

```
{
```

```
"name": "IsPalindromicString",
```

```
"value": "Determines whether a string is a palindromic string"
```

```
},
```

```
{
```

```
  "name": "StringReversal",
```

```
  "value": "String reversal"
```

```
},
```

```
{
```

```
  "name": "IsLeapYear",
```

```
  "value": "Determine whether it is a leap year"
```

```
},
```

```
{
```

```
  "name": "DaffodilsNumberTitle",
```

```
  "value": "Daffodils Number"
```

```
},
```

```
{
```

```
  "name": "Judgment_of_leap_year",
```

```
  "value": "Judgment of leap year"
```

```
},
```

```
{
```

```
  "name": "Judgment_of_palindromic_string",
```

```
  "value": "Judgment of palindromic string"
```

```
},
```

```
{
```

```

        "name": "Multiplication_Table_Title",

        "value": "Multiplication Tables"

    },

    {

        "name": "check_info_in_log",

        "value": "Please check in the log print"

    }

]

}

```

然后，我们去修改 Index.ets 中的字符串，改为使用 `$r('app.string.xxx')` 的方式：

```

Text($r('app.string.ArkTS_Development_Case'))

Button($r('app.string.DaffodilsNumber'))

Button($r('app.string.MultiplicationTable'))

```

相应地，请自己修改 view 目录下的 JiuJiuBiaoDialog.ets 和 ShuiXianHuaDialog.ets 中的常量（记住需要导入文件）， float 参数，以及字符串常量。

改完之后，刷新 Previewer，可以看到没有变化，但是现在项目的代码结构更加清晰了。

### Logger 优化

对于日志记录，类似在 Index.ets 中的：



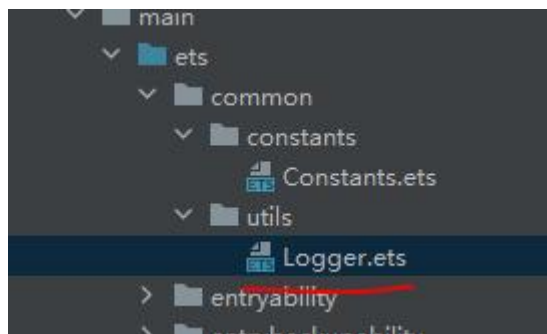
```

.onClick(() => {
    this.JiuJiuBiaoDialogController?.open();
    let result = multiplicationTable();
    hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);
    for (let index = 0; index < result.length; index++) {
        hilog.info(0xFF00, "testTag", result[index].toString());
    }
})

```

如果将来这样的内容多了，要改动标志信息 0xFF00 或 “testTag” 的时候，会有很多的地方需要改动。华为官方在很多的实例中，已经给出了一个推荐的做法。

首先，在 common > utils 目录下，创建一个 Logger.ets 文件：



在 Logger.ets 中加入代码：

```
import hilog from '@ohos.hilog';
```

```
const LOGGER_TAG: string = 'testTag';
```

```
class Logger {
```

```
    private domain: number;
```

```
    private tag: string;
```

```
    // format Indicates the log format string.
```

```
    private format: string = '%{public}s, %{public}s';
```

```
constructor(tag: string = "", domain: number = 0xFF00) {
```

```
    this.tag = tag;
```

```
    this.domain = domain;
```

```
}
```

```
debug(...args: string[]): void {
```

```
    hilog.debug(this.domain, this.tag, this.format, args);
```

```
}
```

```
info(...args: string[]): void {
```

```
    hilog.info(this.domain, this.tag, this.format, args);
```

```
}
```

```
warn(...args: string[]): void {
```

```
    hilog.warn(this.domain, this.tag, this.format, args);
```

```
}
```

```
error(...args: string[]): void {
```

```
    hilog.error(this.domain, this.tag, this.format, args);
```

```
}
```

```
}
```

```
export default new Logger(LOGGER_TAG, 0xFF02);
```

可以看到，我们定义了一个名字为 `Logger` 的类，这个类包含 `domain`，`tag`，`format` 三个成员变量，分别表示：

参数名	类型	必填	说明
<u>domain</u>	number	是	日志对应的领域标识，范围是0x0-0xFFFF。 建议开发者在应用内根据需要自定义划分。
<u>tag</u>	string	是	指定日志标识，可以为任意字符串，建议用于标识调用所在的类或者业务行为。
<u>format</u>	string	是	格式字符串，用于日志的格式化输出。格式字符串中可以设置多个参数，参数需要包含参数类型、隐私标识。 隐私标识分为(public)和(private)，缺省为(private)。标识(public)的内容明文输出，标识(private)的内容以<private>过滤回显。

这些变量其实是与 `hilog` 的函数中的前面三个参数对应的。

最后我们导出的是一个 `Logger` 实例，其中采用的 `LOGGER_TAG` 是“testTag”，而领域标识是 `0xFF02`。

做完这些之后，回到 `Index.ets`，首先我们没必要再自己导入 `hilog` 了，可以从刚才新增的文件中导入，因此，将这条语句：

```
import hilog from '@ohos.hilog';
```

改为：

```
import Logger from '../common/Utils/Logger';
```

然后查找调用了 `hilog` 的两行：

```

    Button($r('app.string.MultiplicationTable'))
        .width($r(288vp))
        .height($r(40vp))
        .margin({ top: $r(12vp) })
        .onClick(() => {
            this.JiuJiuBiaoDialogController?.open();
            let result = multiplicationTable();
            //hilog.isLoggable(0xFF00, "testTag", hilog.LogLevel.INFO);
            for (let index = 0; index < result.length; index++) {
                Logger.info(result[index].toString(), '');
            }
        })
    }

    .justifyContent(FlexAlign.Start)
    .width(CommonConstants.PERCENT_FULL)

```

对于 isLoggable，在 Logger.ets 中没有这个函数，我们可以直接注释掉。这个函数的功能请查一下 API Reference。然后把另外一句：

```
hilog.info(0xFF00, "testTag", result[index].toString());
```

改为：

```
Logger.info(result[index].toString(), "");
```

此时，刷新 Previewer，打印九九乘方表，没有变化。

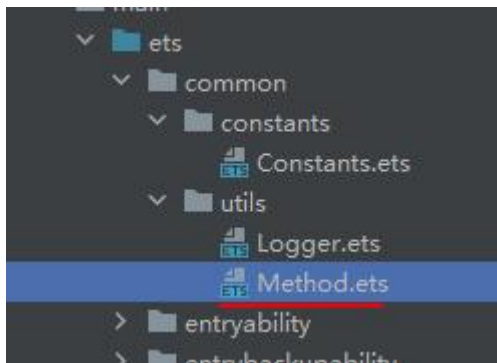
如果你查看华为官方的很多实例，会发现在很多实例中都是用类似的方法处理 Log 日志。

## Method 优化

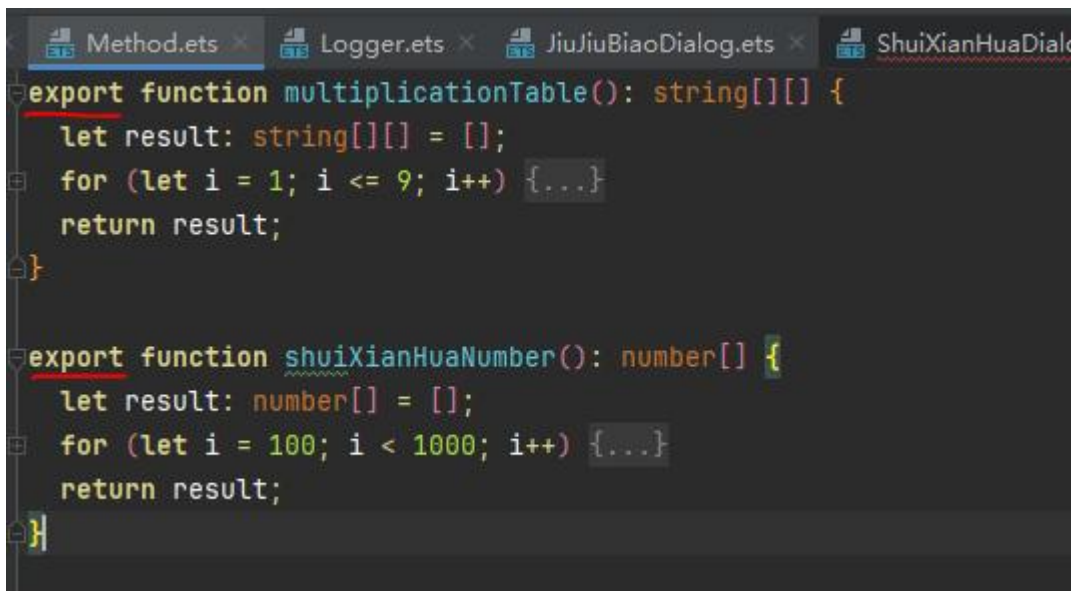
在之前的代码中，对于算法函数，一个是放在 Dialog 文件中，另一个是放在 Index 中，基本上就是在哪里调用就放在哪里。这样的话，算法函数一多，就会比较混乱。

我们还是把所有的算法函数都放在 common > utils 目录下面。创建一个文件 Method.ets：

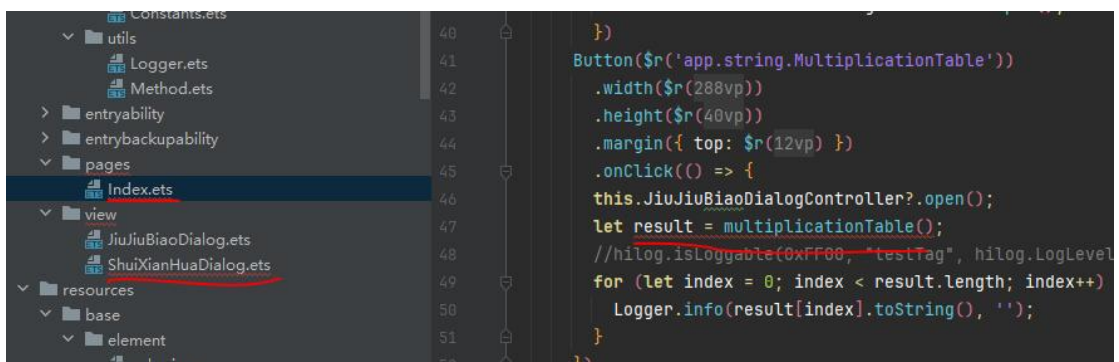




然后把 ShuiXianHuaDialog.ets 中的函数 shuiXianHuaNumber 剪切后粘贴到 Method.ets 中;把 Index.ets 中的函数 multiplicationTable 剪切后粘贴到 Method.ets 中，然后在前面加上 export 进行导出，以便在其他地方可以导入调用。



挪动了之后，可以看到，调用这些函数的地方就报错了：



此时，只需要去对应的地方做一下导入即可。

在 Index.ets 中导入：

```
import { multiplicationTable } from '../common/utils/Method'
```

在 ShuiXianHuaDialog.ets 中导入：

```
import { shuiXianHuaNumber } from '../common/utils/Method'
```

刷新 Previewer，没有变化。但是现在我们清楚相应的功能代码放在什么位置了。

做到这一步，喘口气，休息一下。好像我们做了很多无用功？请思考一下为什么我们要这么做，好处是什么？

## 6. 判断字符串是否为回文

请自己阅读源代码：

<https://developer.huawei.com/consumer/cn/training/course/slightMooc/C101717496870909384>

实现这个功能。注意，这部分又在 common > utils 里面加了一个文件，请自己尝试并理解。

## 7. 字符串反转

请自己阅读源代码：

<https://developer.huawei.com/consumer/cn/training/course/slightMooc/C101717496870909384>

实现这个功能。

## 8. 判断年份是否为闰年

请自己阅读源代码：

<https://developer.huawei.com/consumer/cn/training/course/slightMooc/C101717496870909384>

实现这个功能。

## 9. 调整按钮位置

请自己实现。

## 五、实验注意事项

1. 注意教师的操作演示。
2. 学生机与教师机内网连通，能接收和提交实验结果。
3. 按实验要求输入测试数据，并查看输出结果是否符合实验结果。

## 六、思考题

1. 通过这个实验，你学到了什么？