《嵌入式应用开发》实验 2
组件字典实验指导手册

版本：V 1.0

目录

（一）实验目的

1. 了解常用基础组件
2. 了解容器组件

（二）实验涉及知识点

1. 数组的使用
2. ForEach 用法
3. 数据源使用
4. List 容器
5. Tabs 用法
6. 组件封装

（三）实验准备

1. 技能要求：

操作此实验需要具备基本的 TS 语法知识；

2. 实验环境要求：

基于 Windows10 或者 MacOS 操作系统，安装了 DevEco Studio 开发工具。

(四) 实验内容及课时分配

序号	实验内容	实验课时	对应核心知识点	对应实验目标点
1	常用基础组件	0.5	1、2、3、4、5、6	1
2	容器组件	0.5	1、2、3、4、5、6	2
	总计	2		

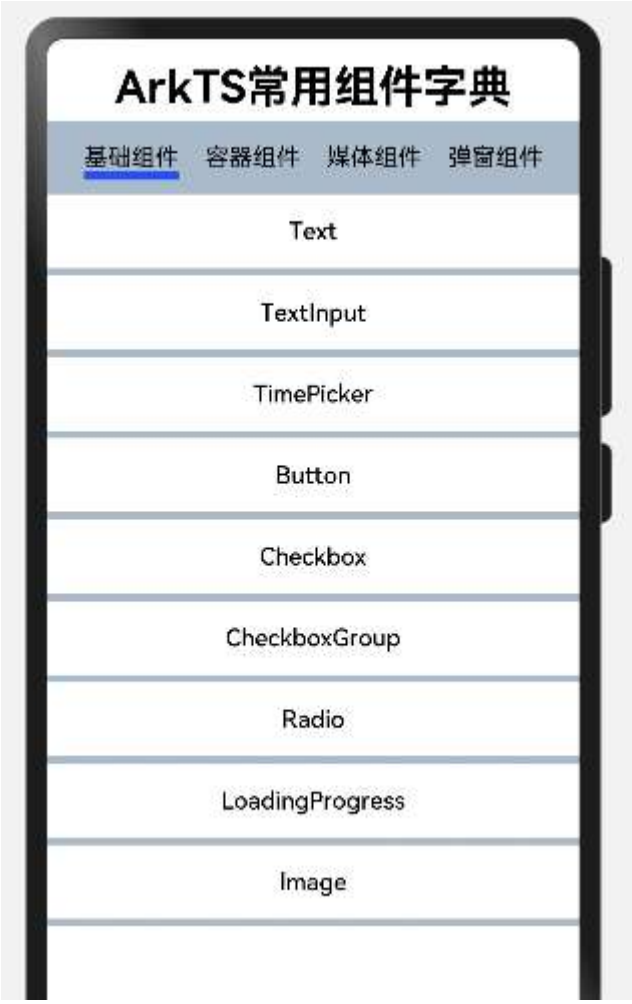
（五）实验过程

1. 实验功能描述

完成功能：

开发一款 UI 组件字典，方便展示各类组件及其 UI 效果，组件数据模型包括 4 个属性：组件分类 ID，组件分类名称、组件 ID、组件名称

首页效果展示：



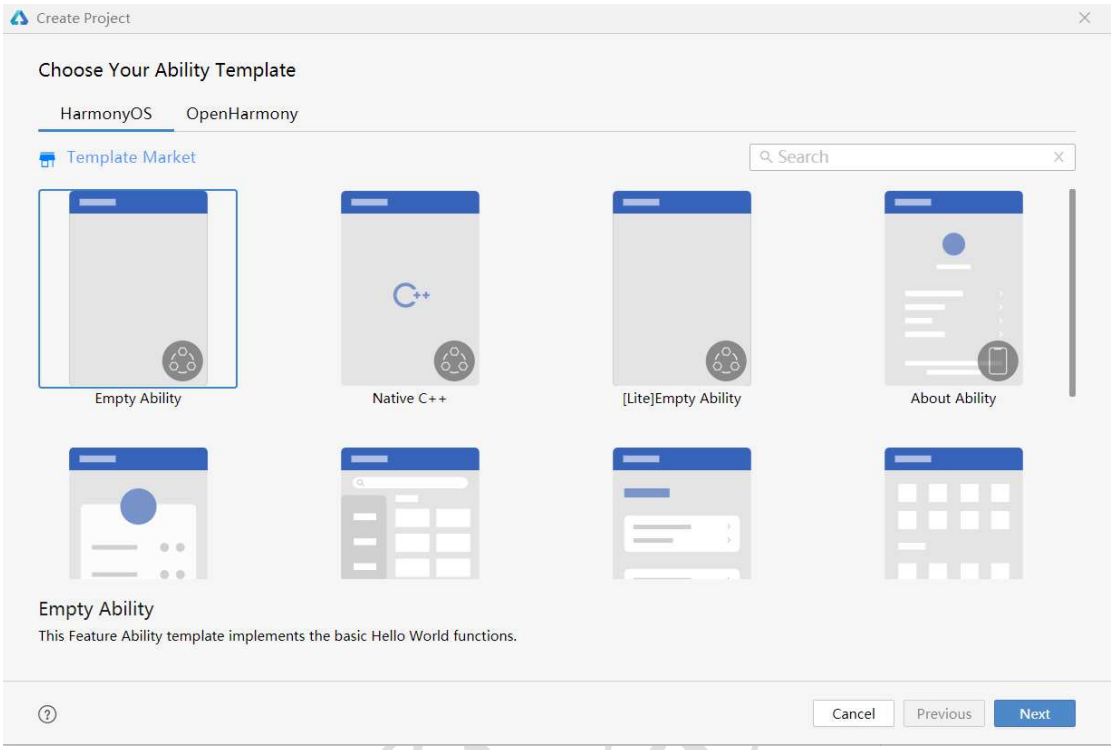
点击 Text 组件：

点击 TextInput 组件：

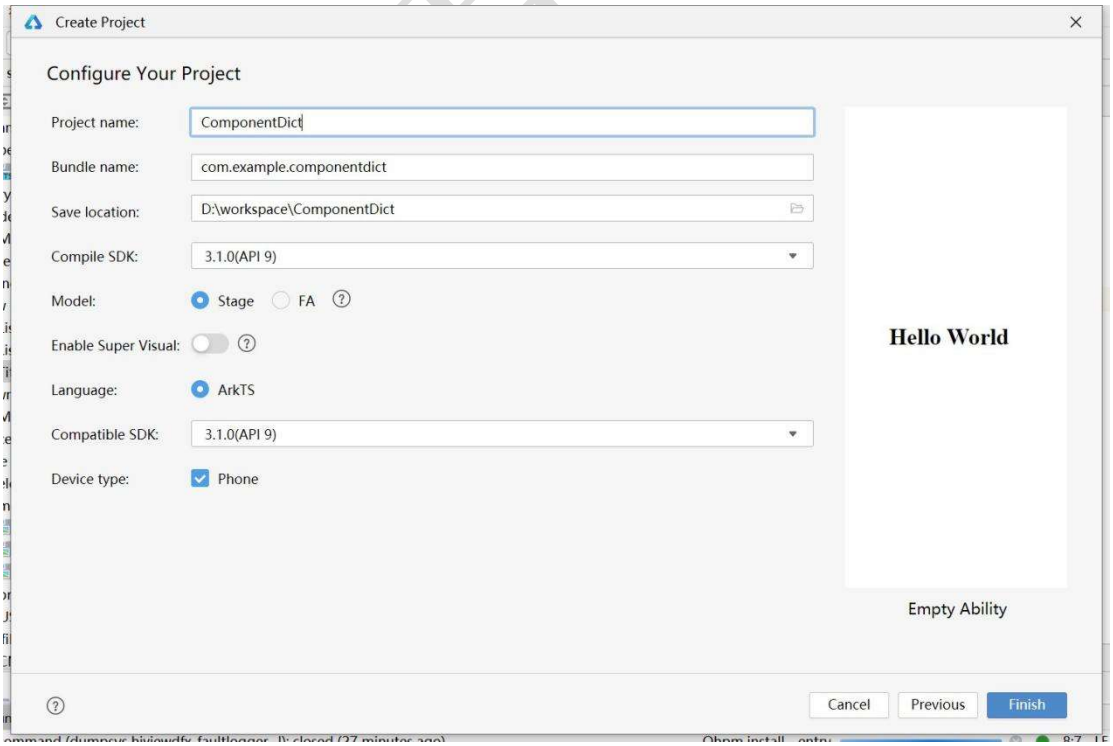
2. 新建项目

1) 创建项目

从 file>new>create project 菜单下新建项目：



点击【Next】按钮，输入项目配置信息：



点击【Finish】按钮完成项目创建。

2) 创建文件夹结构

在 etc 下创建一个 model 文件夹存储数据模型，创建 common/components 两级目录存储组件代码，文件夹结构如下图所示。

3. 建立组件数据模型

1) 在 model 下新建 ComponentModel.ets 用于模拟组件原始信息

```
// 组件基础信息
/*
  catId: 分类 ID
  catName: 分类名
  id: 组件 ID
  compName: 组件名
  此处仅模拟 20 个常用组件
 */

export const ComponentData: any[] =[
  {
    'catId': 1,
    'catName':'基础组件',
    'id': 1,
    'compName':'Text'
  },
  {
    'catId': 1,
    'catName':'基础组件',
    'id': 2,
    'compName':'TextInput'
```

```
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 3,
    'compName':'TimePicker'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 4,
    'compName':'Button'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 5,
    'compName':'Checkbox'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 6,
    'compName':'CheckboxGroup'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 7,
    'compName':'Radio'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 8,
    'compName':'LoadingProgress'
},
{
    'catId': 1,
    'catName':'基础组件',
    'id': 9,
    'compName':'Image'
},
{
```

```
    'catId': 1,
    'catName':'基础组件',
    'id': 10,
    'compName':'Progress'
  },
  {
    'catId': 1,
    'catName':'基础组件',
    'id': 11,
    'compName':'Select'
  },
  {
    'catId': 1,
    'catName':'基础组件',
    'id': 12,
    'compName':'Slider'
  },
  {
    'catId': 2,
    'catName':'容器组件',
    'id': 13,
    'compName':'Column'
  },
  {
    'catId': 2,
    'catName':'容器组件',
    'id': 14,
    'compName':'Row'
  },
  {
    'catId': 2,
    'catName':'容器组件',
    'id': 15,
    'compName':'Panel'
  },
  {
    'catId': 2,
    'catName':'容器组件',
    'id': 16,
    'compName':'swiper'
  },
  {
    'catId': 3,
    'catName':'媒体组件',
```

```
    'id': 17,
    'compName':'Video'
  },
  {
    'catId': 4,
    'catName':'弹窗',
    'id': 18,
    'compName':'AlertDialog'
  }
  ,
  {
    'catId': 4,
    'catName':'弹窗',
    'id': 19,
    'compName':'DatePickerDialog'
  }
  ,
  {
    'catId': 4,
    'catName':'弹窗',
    'id': 20,
    'compName':'TextPickerDialog'
  }
]
```

2) 封装组件数据模型

继续在 ComponentModel.ets 文件中定义类和构造方法：

```
export class ComponentBean {
  catId: number; //组件分类 ID
  catName: string; //组件分类名
  id: number; //组件 id
  compName: string ; //组件名

  //构造方法
  constructor(catId: number, catName: string, id: number, compName: string) {
    this.catId = catId
    this.catName = catName
    this.id = id
    this.compName = compName
  }
}
```

3) 初始化数据

定义初始化方法，返回组件数组模拟外部数据加载，继续在

ComponentModel.ets 中编码

```
//加载组件基本数据
export function initComponentsData():Array<ComponentBean> {

  let componentArray: Array<ComponentBean> = []
  //迭代原始数据封装成 bean 并存储到数组中，然后返回
  ComponentData.forEach((item)=>{
    componentArray.push(new ComponentBean(item.catId,item.catName,item.id,item.compName))
  })

  return componentArray;

}
```

4. 展示基本组件

在 Index.ets 中完成列表界面开发。

1) 打开 pages 文件夹下的 Index.ets 文件，清理代码：

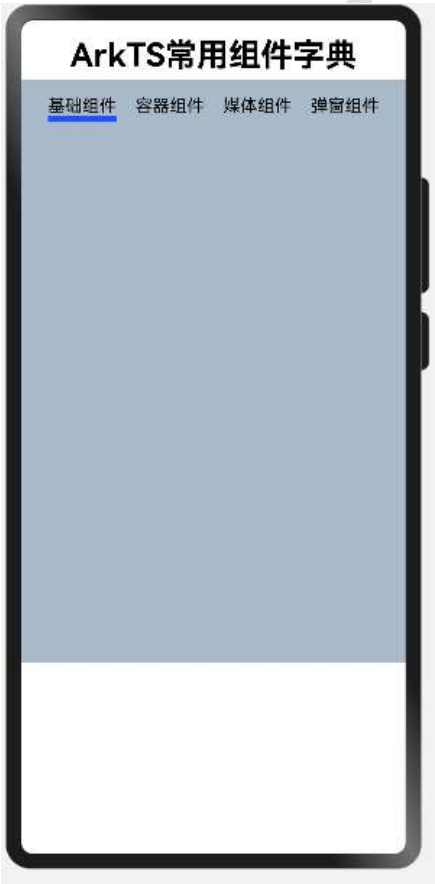
```
@Entry
@Component
struct Index {


  build() {
    Column(){

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

2) 布局 Tabs

```
@Entry
@Component
struct Index {

  build() {
    Column(){
      Text('ArkTS 常用组件字典')
        .fontSize(30)
        .fontWeight(700)
        .margin({top:15,bottom:5})
      Tabs(){
        TabContent(){
```

```
            }.tabBar('基础组件')
            TabContent(){

            }.tabBar('容器组件')
            TabContent(){

            }.tabBar('媒体组件')
            TabContent(){

            }.tabBar('弹窗组件')
        }
        .barWidth('100%')
        .barHeight(50)
        .scrollable(true)
        .height('70%')
        .backgroundColor('#ffa9bacb')
    }.width('100%')
    .height('100%')
  }
}
```

效果预览：



3) 定义基础组件列表

继续在 Index.ets 中进行编码，定义基础组件列表，定义组件需要使用 @Component 装饰器，并编写 build 函数。

```
@Component
struct BasicComponentList {
  private listArray: Array<ComponentBean> = []

  aboutToAppear() {
    var newList = initComponentsData()
    this.listArray = newList.slice(0, 12)
  }

  build() {
    Column(){
      List({ space: 5 }) {
        ForEach(this.listArray,item =>{
          ListItem(){
            Button(item.compName)
              .type(ButtonType.Normal)
              .width('100%')
              .height(50)
              .fontColor(0x000000)
              .backgroundColor(0xffffff)
          }
        },item=>item.id.toString())
      }
    }

  }
}
```

在 Tabs 中加载基础组件列表：

```
Tabs() {
        TabContent() {
          BasicComponentList()
        }.tabBar('基础组件')
          ...
```

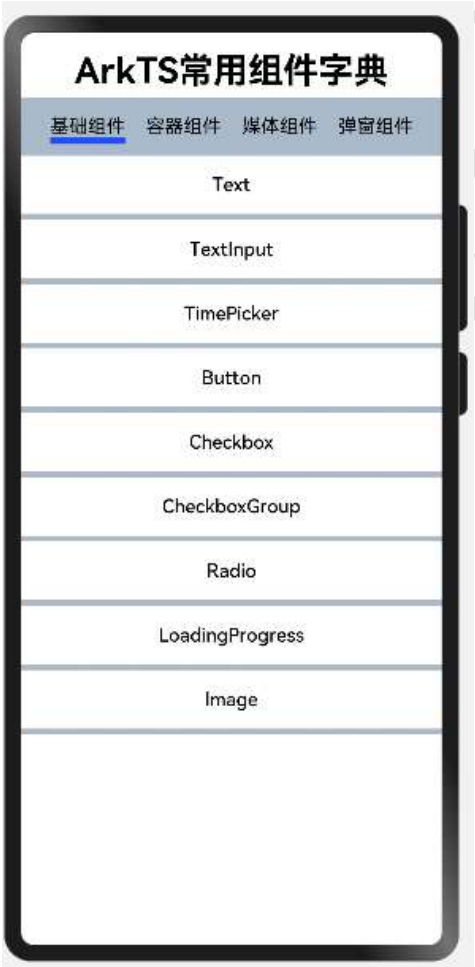
Index.ets 当前代码：

```
import { initComponentsData } from '../model/ComponentModel';
import { ComponentBean } from '../model/ComponentModel';

@Entry
```

```
@Component
struct Index {
  build() {
    Column() {
      Text('ArkTS 常用组件字典')
        .fontSize(30)
        .fontWeight(700)
        .margin({ top: 15, bottom: 5 })
      Tabs() {
        TabContent() {
          BasicComponentList()
        }.tabBar('基础组件')

        TabContent() {

        }.tabBar('容器组件')

        TabContent() {

        }.tabBar('媒体组件')

        TabContent() {

        }.tabBar('弹窗组件')
      }
      .barWidth('100%')
      .barHeight(50)
      .scrollable(true)
      .height('70%')
      .backgroundColor('#ffa9bacb')
    }.width('100%')
    .height('100%')
  }
}

@Component
struct BasicComponentList {
  private listArray: Array<ComponentBean> = []

  aboutToAppear() {
    var newList = initComponentsData()
    this.listArray = newList.slice(0, 12)
  }
```

```
  build() {
    Column(){
      List({ space: 5 }) {
        ForEach(this.listArray,item =>{
          ListItem(){
            Button(item.compName)
              .type(ButtonType.Normal)
              .width('100%')
              .height(50)
              .fontColor(0x000000)
              .backgroundColor(0xffffff)
          }
        },item=>item.id.toString())
      }
    }

  }
}
```

预览效果：



5. 展示容器组件

和前面的代码类似，只是数组切割不同：

```
//容器组件列表
@Component
struct ContainerComponentList {
  private listArray: Array<ComponentBean> = []

  aboutToAppear() {
    var newList = initComponentsData()
    this.listArray = newList.slice(12, 16)
  }

  build() {
    Column(){
      List({ space: 5 }) {
        ForEach(this.listArray,item =>{
          ListItem(){
            Button(item.compName)
              .type(ButtonType.Normal)
              .width('100%')
              .height(50)
              .fontColor(0x000000)
              .backgroundColor(0xffffff)
          }
        },item=>item.id.toString())
      }
    }

  }
}
```

在入口 tabs 中调用：

```
TabContent() {
  ContainerComponentList()
}.tabBar('容器组件')
```

效果展示：

6. 展示媒体组件

组件定义：

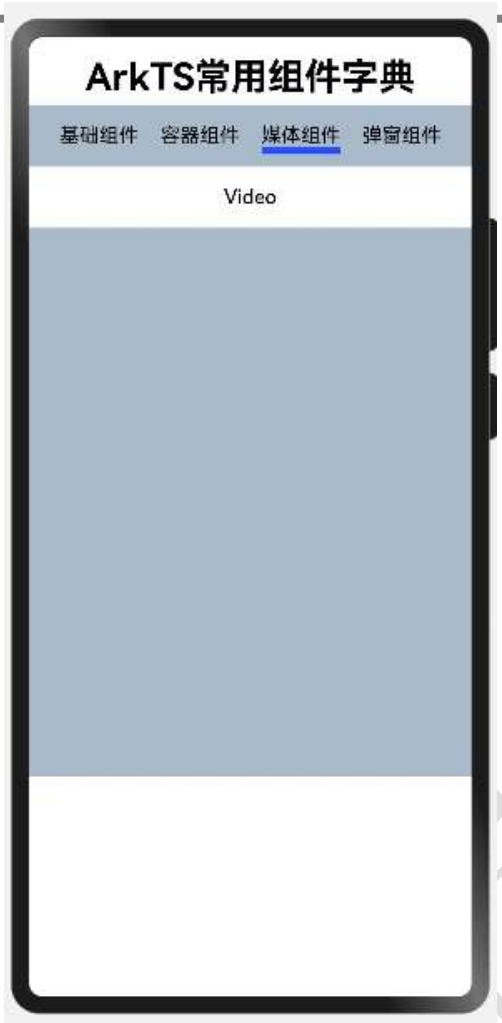
```
//媒体组件列表
@Component
struct MediaComponentList {
  private listArray: Array<ComponentBean> = []

  aboutToAppear() {
    var newList = initComponentsData()
    this.listArray = newList.slice(16, 17)
  }

  build() {
```

```
      Column(){
        List({ space: 5 }) {
          ForEach(this.listArray,item =>{
            ListItem(){
              Button(item.compName)
                .type(ButtonType.Normal)
                .width('100%')
                .height(50)
                .fontColor(0x000000)
                .backgroundColor(0xffffff)
            }
          },item=>item.id.toString())
        }
      }

  }
}
```

组件调用：

```
TabContent() {
  MediaComponentList()
}.tabBar('媒体组件')
```

效果展示：



7. 展示弹窗组件

组件定义：

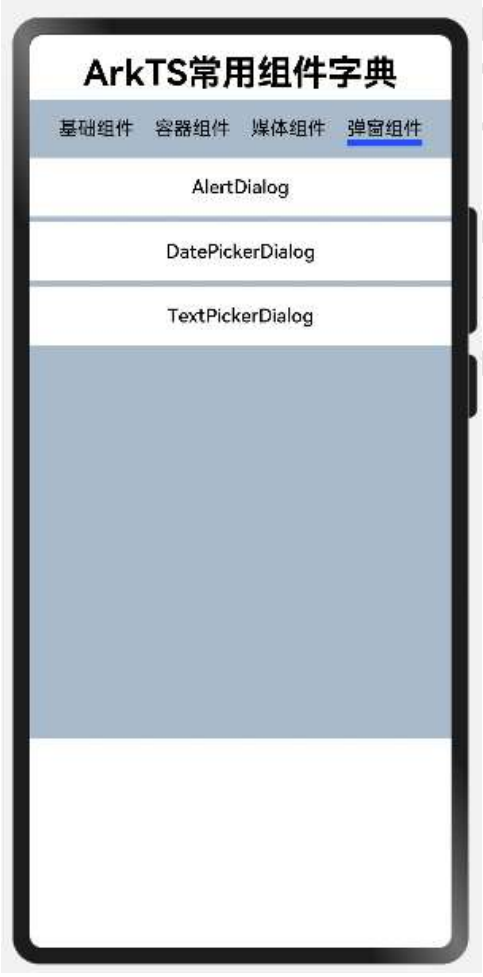
```
//弹窗组件列表
@Component
struct DailogComponentList {
  private listArray: Array<ComponentBean> = []

  aboutToAppear() {
    var newList = initComponentsData()
    this.listArray = newList.slice(17, 20)
  }

  build() {
    Column(){
      List({ space: 5 }) {
        ForEach(this.listArray,item =>{
          ListItem(){
            Button(item.compName)
```

```
                .type(ButtonType.Normal)
                .width('100%')
                .height(50)
                .fontColor(0x000000)
                .backgroundColor(0xffffff)
            }
        },item=>item.id.toString())
      }
    }

  }
}
```

组件调用：

```
TabContent() {
  DailogComponentList()
}.tabBar('弹窗组件')
```

运行效果：



8. 组件详情页面

1) 在 Pages 目录下创建 Page，命名为 ComponentDetails.ets，通过条件渲染加载各组件：

```
import { MyText } from '../common/components/MyText';
@Entry
@Component
struct ComponentDetails {
  @State compnentName: string = 'Text'

  build() {
    Column(){
      if(this.compnentName == 'Text'){
        MyText()
      }
    }
  }
}
```

2) 在 common/components 文件夹下完成 MyText 组件开发，此处仅简单演示：

```
@Component
export struct MyText {
  build() {
    Flex({ direction: FlexDirection.Row,
      alignItems: ItemAlign.Start,
      justifyContent: FlexAlign.SpaceBetween }){
      //单行文本
      Text('单行文本').fontSize(20).borderWidth(1)
      //多行文本
      Text('多行文本多行文本多行文本多行文本多行文本多行文本多行文本')
        .fontSize(20).textAlign(TextAlign.Center).borderWidth(1)
    }
  }
}
```

详情页预览效果：

3) TextInput 组件展示

在 ComponentDetails.ets 组件中继续添加条件渲染：

```
if(this.compnentName == 'TextInput'){
  MyTextInput()
}
```

同时开发对应的 MyTextInput 组件，在 components 目录下新建 ArkTS，命名为 MyTextInput.ets，内容如下：

```
@Component
export struct MyTextInput {
  @State message:string =''
  build(){
    Column(){
      TextInput({placeholder:'请输入你想说的话！'}).height(50)
        .onChange((value)=>{
          this.message = value;
        }
        )
      Text(`你想说：${this.message}`).fontSize(30)
    }

  }

}
```

注意：需要在 ComponentDetails.ets 中引入 MyTextInput 组件

效果预览：

9. 跳转到详情页

1) 编辑 Index.ets 页面找到对应基础组件列表，为按钮添加单击事件进行跳转。

导入路由：

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

在 BasicComponentList 中跳转：

```
Button(item.compName)
  .type(ButtonType.Normal)
  .width('100%')
  .height(50)
  .fontColor(0x000000)
  .backgroundColor(0xffffff)
  .onClick(()=>{
    router.pushUrl({
      url:'pages/ComponentDetails',
      params:{
        name:item.compName
      }
```

```
    })
})
```

2) 修改组件详情页面接收路由参数

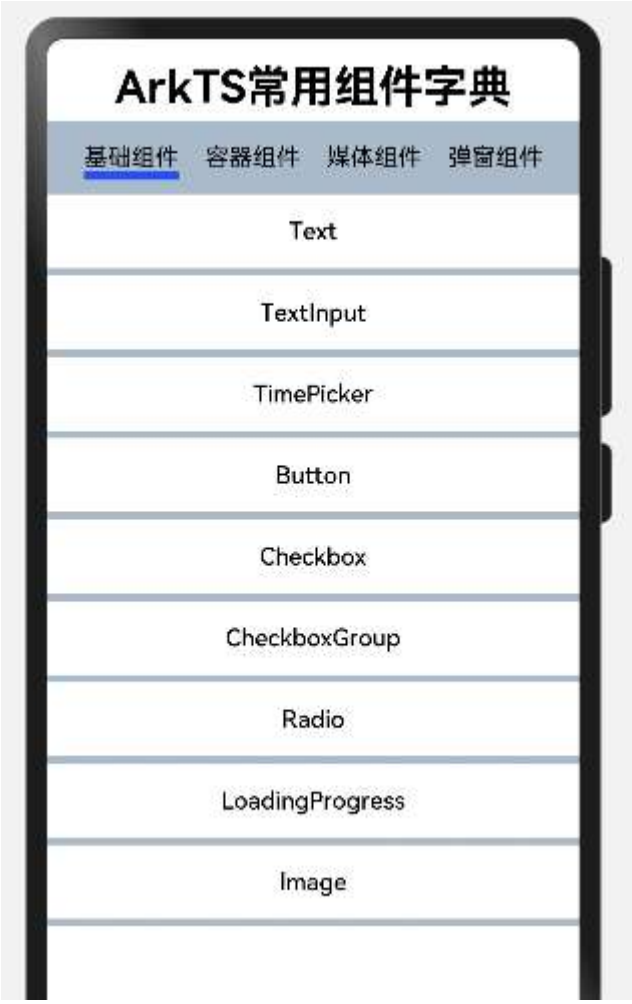
在 ComponentDetails.ets 组件中导入路由：

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

接收参数：

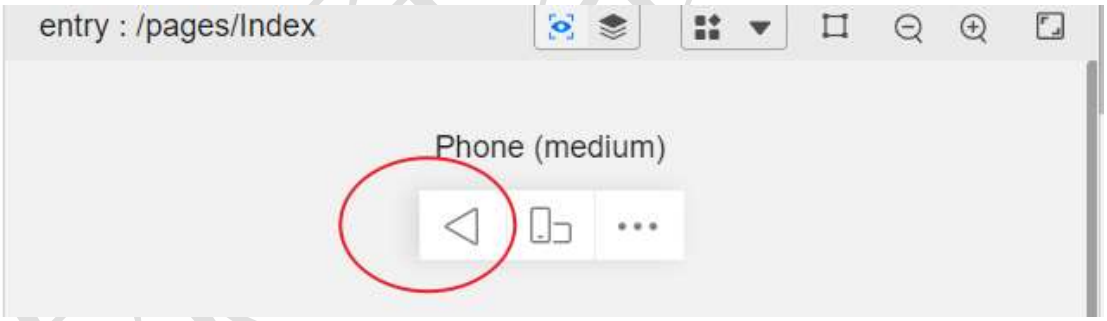
```
@State compnentName: string = router.getParams()['name']
```

3) 从 index 页面进行跳转测试

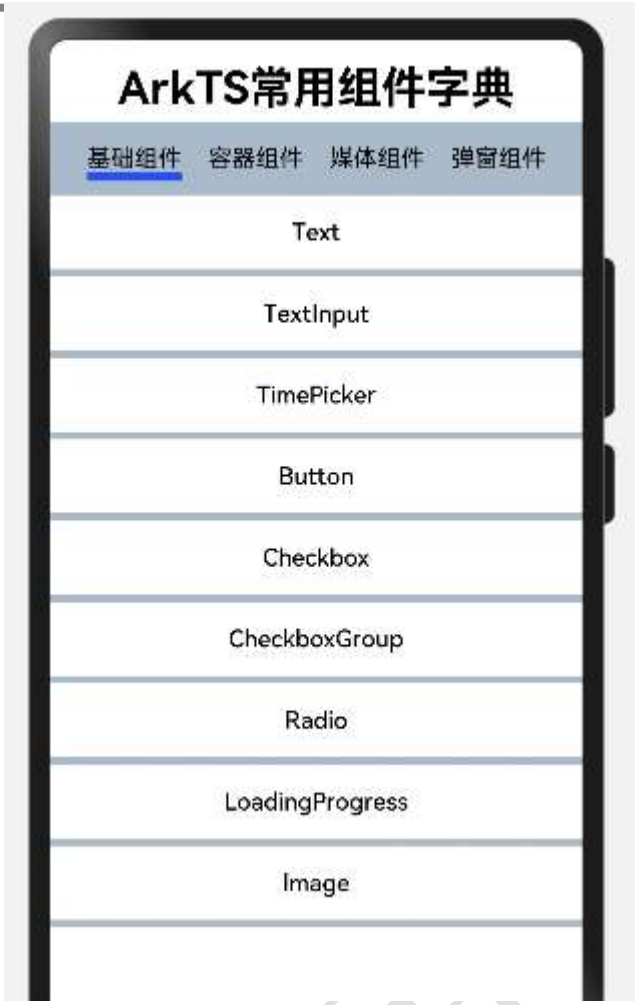


点击 Text 组件：

点击预览器返回按钮：



返回后页面：



点击 TextInput 组件：

10. 继续开发剩余组件

自此流程关键流程已经开发完毕，可以继续开发其他组件，开发过程和前面类似，此处不再赘述。