

Index

GettingStart

暑期實訓 其之零

前言

此次實訓為校企合作，主要範圍關係鴻蒙北向應用開發，涉及到的知識點包括

- 由HuaWei公司主推的HarmonyOS生態運作方式
- 由HuaWei公司主推的ArkTS語言使用方式
- 網路通訊協定與網路安全
- 分散式DataBase及其CURD
- 分散式後端開發及流處理

大致瞭解過內容后，某人認為該實訓雖和 實踐 及 學習 均沾邊，但究其本源還只是課程性質罷了

該條目下將記錄為期4週的實訓內容，僅作為成果歸檔和個人報告的一些參考，所有言論代表個人觀點，不針對任何組織

實訓安排

按照目前給出的消息，大致安排如下：

實訓時間		實訓項目名稱	《鴻蒙北向應用開發》項目	
		實訓進度	實訓內容	涉及知識點
第1天	上午	開班典禮 (實訓啟動會)	1、開班典禮儀式、校企方老師講話 2、實訓講師介紹實訓教學計劃 3、鴻蒙生態介紹 4、實訓項目介紹	1、鴻蒙生態
	下午	認識HarmonyOS 並搭建項目開發環境	1、HarmonyOS系統定義 2、HarmonyOS技術特性 3、HarmonyOS系統安全	1、HarmonyOS特色 2、HarmonyOS技術架構 3、開發環境

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
			4、HarmonyOS关键技术 5、开发环境搭建	4、工程结构
第2天	全天	北向应用开发基础 常用组件实验操作 (一)	1、老师讲解并演示部分组件实验 2、学生参照实验指导手册完成实验	1、ArkTS语言介绍和使用 2、容器组件和基础组件布局实验 3、用户登录、注册实验 4、List、Swiper、Dialog、Refresh、Badge、Image、Select、Progress、Slider、Menu、Search、Camera、Video、Canvas等组件基本使用
第3天	全天	北向应用开发基础 常用组件实验操作 (二)	1、老师讲解并演示部分组件实验 2、学生参照实验指导手册完成实验	1、Animation关键帧动画 2、Animate方法快速构建动画 3、API接口创建和运行动画 4、Ability启动原理和日志打印 5、页面路由和系统弹窗 6、设置窗口状态栏与导航栏 7、定时器、剪贴板
第4天	全天	北向应用开发基础 常用组件实验操作 (三)	1、老师讲解并演示部分组件实验 2、学生参照实验指导手册完成实验	1、自定义组件 2、父子组件数据传递 3、\$watch 感知数据改变

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
				4、自定义事件 5、生命周期 6、computed 计算属性 7、系统通知
第5天	全天	北向应用开发基础 常用组件实验操作 (四)	1、老师讲解并演示部分组件实验 2、学生参照实验指导手册完成实验	1、Grid网格组件开发 2、轻量级存储 3、文件交互和文件管理 4、Http Get请求和Post请求 5、WebSocket数据通信 6、手机状态管理
第6天	全天	北向应用开发基础 常用组件实验操作 (五)	1、老师讲解并演示部分组件实验 2、学生参照实验指导手册完成实验	1、Service Ability开发 2、关系型数据库增删改查操作 3、Data Ability开发 4、原子化服务 5、分布式开发基础案例
第7天	上午	项目代码review	1、小组内成员完成互相进行代码review，互相串讲，对不合格的代码提交进行回退整改。	1、git使用 2、编程规范

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
			2、老师选择学生优秀代码提交案例和劣质代码提交案例进行讲解，着重讲解编程规范与编程思想。	
	下午	常用设计组件详解 轮播图	1、首页页面布局分解 2、滚动banner实现 3、菜单项布局实现 4、班级与课程列表实现	1、Swiper组件 2、Flex布局特性 3、循环渲染
第8天	全天	常用设计组件详解 底部导航栏开发	1、底部导航栏布局实现 2、底部导航栏实现子页面切换 3、封装可以复用的自定义导航栏	1、自定义组件
第9天	全天	常用设计组件详解 通讯录	1、标题栏实现 2、tab菜单栏实现 3、课程列表实现 4、课程列表下拉刷新与上拉加载	1、tab组件 2、list列表
第10天	全天	网络请求框架封装 与登录会话保持	1、完成网络请求与UI的解耦设计，封装自定义网路请求框架 2、完成登录会话保持设计	1、编程进阶，解耦设计与框架封装思想 2、轻量级存储 3、token认证设计

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
			1、3、完成所有其他api的对 接与页面路由	
第 11 天	全 天	数据库持久化详解 MyBatis	1、MyBatis快速入门	1、 MyBaits环境搭建 2、 动态SQL处理 3、 resultMap高级映射
第 12 天	全 天	后端接口开发 SpringBoot	2、Maven 快速开发 3、SpringBoot操作 4、RESTFul API	1、 Maven 环境搭建 2、 Maven 项目构建 3、 SpringBoot 搭建后台服务 4、 Controller详解 5、 RESTFul API 请求接口设计 6、 框架整合
第 13 天	全 天	rest api接口对接	1、rest api接口文档讲解 2、使用postman调试接口 3、登录接口（post）对接 课程列表接口（get）对接	1、rest api接口文档编写与阅读 2、接口测试 3、网络请求
第 14 天	全 天	完成视频的分布式 流转与个性化卡片 开发	1、完成视频的分布式流转开 发 2、根据个人喜好开发定制个 性化服务卡片	1、分布式流转 2、服务卡片
第 15	全 天	项目迭代开发	1、晨会：各组汇报设计任务 进	1、项目功能实现

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
天			度, 疑难问题, 完成问题汇总 2、APP首页等Story功能实现	2、问题跟踪
第16天	全天	项目迭代开发	1、晨会: 各组汇报设计任务进 度, 疑难问题, 完成问题汇总 2、APP课程列表等Story功能实现	1、项目功能实现 2、问题跟踪
第17天	全天	项目迭代开发	1、晨会: 各组汇报设计任务进 度, 疑难问题, 完成问题汇总 2、APP课程详情等Story功能实现	1、项目功能实现 2、问题跟踪
第18天	全天	项目迭代开发	1、晨会: 各组汇报设计任务进 度, 疑难问题, 完成问题汇总 2、完成APP班级、学习计划管理Story功能实现	1、项目功能实现 2、问题跟踪

实训时间		实训项目名称	《鸿蒙北向应用开发》项目	
第19天	全天	项目迭代开发	1、项目部署、发布 2、功能完整度自检、测试 3、Bug集中修复	1、功能测试 2、部署发布
第20天	全天	综合答辩	1、组织小组答辩，完成个人、组综合评分 2、讲师做实训过程总结	考核+团队协作能力提升

預先學習

從此次實訓的內容安排不難看出，其基於 **校企合作** 戰略，參考了本校所教授知識體系設計內容，基本上為課程知識點的進一步擴展，非常適合大部分計劃考研及考公的學生

由於某人掌握的技術棧與需求相差甚遠，故需提前投入一定的成本來補充前置技能

概念理解

針對安排中出現的陌生詞匯，逐一瞭解概念

- **HarmonyOS**

鴻蒙作業系統是由華為公司自主研發（較大爭議）的分散式作業系統

在較早的 **HarmonyOS** 時期，由於存在較大的使用 Android 開源專案（AOSP）進行開發甚至是直接套用安卓以及調用相應的API等疑似欺詐行為的可能性，曾一度引起質疑和聲討（概括自[wiki](#)）

而在較新的 **HarmonyOS NEXT** 中，由於去掉了 Linux 核心及 AOSP 等，導致 Android 應用無法繼續被支持。

本次實訓特地採用了ArkTS進行開發，預計使用的大概率是 NEXT

- **北向應用**

在鴻蒙生態中，給硬件終端寫代碼叫做 **南向開發**，給應用軟件寫代碼叫做 **北向開發**

（某人吐槽：這命名絕對是從上北下南、底層、上游這些概念發散出來的，太冷笑話了）

- **ArkTS**（存疑，待修改）

ArkTS 是華為推出的聲明式編程語言，本質上是 TypeScript 的超集，用於在鴻蒙應用中設計 ArkUI，進行聲明式UI應用程式開發

目前支持 HarmonyOS、Linux、Windows、macOS、iOS 和 Android 等多個平臺，兼容舊版本的 eTS（擴展 TypeScript）

- 框架封裝思想

就是常說的封裝思想，只不過用在了框架上，於是特地換了一個名字罷了

- MyBatis

Java 生態中的一個數據持久化框架（SQL 映射框架），目前某人將其簡單理解為 Java 版的 Dapper 或者 EFCore（不過這是 ORM 框架）

MyBatis 通過配置文件（通常是 XML）進行映射支持，使用注解來添加特性，使得 SQL 語句實現到 Java 對象上

- resultMap高級映射

高級映射是爲了將數據庫中的表結構映射為對象模型

MyBatis 通過 resultMap 實現高級映射，需要在 mapper（XML 配置文件）中手動指定映射關係

由於這種方式是直接將各個屬性的映射關係直接固定在配置中了，因此比起 EFCore 中使用 Fluent API 的方式會帶來更直觀，更自由的優勢

- SpringBoot

SpringBoot 是個基於 Spring Framework 的框架，類似於基於 ASP.NET Core 的ABP框架

環境搭建

由於內容重合，該部分將整合到[DAY01](#)

個人總結（吐槽）

涉足一個完全陌生的領域是一件叫人很沒底的事情🤔

等結束後來補充一下最終看法吧

暑期實訓 其之一

前言

接[上文](#)

此為實訓第一天內容，首先總結一下今日事項，主要概括為以下幾方面

- 項目介紹與宣講
- 環境搭建等前置技能
- 熟悉基本ArkUI開發

主要内容

項目介紹與宣講

這部分主要就是針對接下來的安排與發生的調整做出說明，沒什麼值得記錄的點

環境搭建等前置技能

集成開發環境

此次實訓采用 Deveco Studio 3.10.501版本進行鴻蒙應用的開發，若采用其他版本，某人無法確保該條目下所有方法可行🙏

1. 安裝Deveco Studio

Deveco Studio 是一款用於開發鴻蒙應用的 IDE，支持 ArkTS 編寫（此後簡稱deveco）

deveco 可以從華為官網下載，注意該 IDE 可能需要使用華為賬號來獲取，因此請先注冊賬號

2. 安裝 node.js 和 ohpm

OHPM（OpenHarmony Package Manager）是一个用于管理 OpenHarmony 项目依赖的包管理工具

安裝 deveco 后會提示導入或者安裝這兩個工具，如果您忘了設置，可以在 **Setting > Build,Extention,Deployment** 中找到相應的修改選項

小插曲 在本次實訓中，某人攜帶了一臺長期閒置的筆電，由於曾經使用過 node.js，於是希望引用本地的 node。然而 deveco 中對文件路徑有著近乎變態的嚴格要求，導致某人最終只能選擇按照其傻瓜式操作一鍵安裝

3. 安裝鴻蒙 OS 的 SDK

目前使用到兩個主要的 SDK

- hmscore
- openharmony

相關配置選項可以在 [Setting > SDK](#) 中找到

4. 安裝模擬器

完成以上步驟之後，應該可以在 deveco 中按照項目模板進行創建。若無法完成該動作，請檢查此前步驟，有問題請優先查閱華為官方提供的文檔以及相關社區內容

創建項目后無法直接運行得到直觀的結果，此時在運行鍵左側點開下拉框，選擇 Device Manager，根據需求安裝對應的模擬器

版本控制工具

在示例中采用 TortoiseGit 作為版本控制工具

由於已安裝 GitHub Desktop，故未采用該方式

鑒於目擊多起 git 工具使用失敗案例，下面給出大致操作步驟供參考

1. 安裝 git

無論是TortoiseGit 還是 GitHub Desktop，其實都可以算是 git 的魔改，推薦極簡主義者安裝 git，推薦無腦主義者安裝 GitHub Desktop，搜索關鍵字參考“{您使用的軟體} + 使用步驟 + 教你學會 + 倉庫拉取”

2. 獲取本機密鑰

如果您是第一次使用 git，請通過命令生成專屬於本機的密鑰，搜索關鍵字參考“[rsa + ssh + git](#) | [ssh密鑰](#)”

(Windows中) 正常得到密鑰后應該在[C://Users/{您的用戶名}/.ssh](#)路徑下找到兩個文件

- [id_rsa](#)
- [id_rsa.pub](#)

3. 上傳密鑰

來到您倉庫所在位置，不論是常見的GitHub，Gitee還是自建的GitLab之類的，都會提供ssh密鑰上傳的功能，通常位於倉庫設置、倉庫屬性等叫法的地方

找到通常名為[ssh公鑰](#)或[ssh密鑰](#)等條目下，添加一條記錄，形式為 標題-值 的鍵值對，值為上一步中 [id_rsa.pub](#)裏的內容，請確保以 [rsa](#) 開頭

注意，上傳[id_rsa.pub](#)的作用是讓倉庫知道，用您手上這臺電腦操作倉庫和您賬號操作是等價的，這樣才能無感使用git工具（這一步忽略者較多，當然用賬號密碼登錄也行，但是使用git時好像會直接回顯權限錯誤）

4. 拉取倉庫

在目標文件夾位置運行

```
git clone {目標倉庫}
```

這行命令一般會在倉庫醒目位置標出

熟悉基本ArkUI開發（分析說明之類的，有時間再補充）

在以上步驟完成之後，可通過deveco創建一個空項目，選項均保持默認即可

某人創建了一個名為**PracticalTraining**的項目，創建後將默認打開文件Index.ets

關於文件目錄的詳細信息請參照華為相關文檔或等待某人後文的補充

HelloWorld

在Index.ets中的代碼是一個默認的 HelloWorld，其中包含注解

```
@Entry  
@Component
```

這說明了將結構體（暫且這麼叫）作為一個組件來使用，此處當然是根組件了

在結構體裏面，可以定義變量，使用如下句型

```
@State label:string = '用户名称';  
@State name:string = '张三';
```

採用了神似Vue的build()，基本佈局可以放在裏面

常用了Row和Column，常用屬性有width()和height()等，使用方法參考華為提供的文檔

子組件 定義

前面所操作的内容均為 Index.ets文件中的代碼，此時將其作為一個根組件使用

子組件則是將某個組件引用到其他組件中，從而實現以組件為單位的代碼復用

1. 創建新組件

在 ets 檔案夾中創建新的檔案夾，我將其命名為 components（和pages同級）

在 components 檔案夾下新增 ets 文件，我將其命名為Test1.ets

2. 編輯代碼

```
@Styles function rowStyle(){
    .padding(10).width("100%").border({width:{bottom:1},color:"#EEE"})
}

@Extend(Row) function rowBlock(){
    .justifyContent(FlexAlign.SpaceBetween)
}

@Component
export default struct Test1{
    build() {
        Column() {
            Column() {
                Row() {
                    Text("名字")
                    Text("GZ4nna")
                }.rowBlock().rowStyle()

                Row() {
                    Text("账户")
                    Text("GZ4nna")
                }.rowBlock().rowStyle()

                Row() {
                    Text("年齡")
                    Text("" + 123)
                }.rowBlock().rowStyle()

            }.backgroundColor("#FFF")
        }.width("100%").height("100%").backgroundColor("#AAA")
        .justifyContent(FlexAlign.Center)
    }
}
```

3. 作為子組件引用

```
import Test from "../components/Test1"

@Entry
@Component
struct Index {
    build() {
        Column() {
```

```

        Test()
    }.width("100%).height("100%")
}
}

```

子組件 單向傳值

子組件光是作為固定的視圖使用的話，就像是一塊死物放在那裏做展示，我們希望組件也是可以參與交互過程的

從創建子組件的過程中可以看出來，組件本質上也是一個對象罷了，那麼在一個類中引用其他類的對象，對其屬性做出操作，不就可以使子組件有辦法響應父組件的行爲了麼

1. 創建子組件

在 components 檔案夾下新增 ets 文件，我將其命名為Test2.ets

2. 編輯代碼

```

@Component
struct Person{
    label:string
    value:string

    build(){
        Row(){
            Text(this.label)
            Text(this.value)
        }.padding(10).width("100%).justifyContent(FlexAlign.SpaceBetween)
        .border({width:{bottom:1},color:"#EEE"})
    }
}

```

```

@Component
export default struct Test2{
    build() {
        Column() {
            Person({label:"名字",value:"张三"})
        }.width("100%).height("100%")
    }
}

```

3. 在父組件中傳值

修改引用

```
import Test from "../components/Test2"
```

子組件 雙向傳值

在現代的應用程式中，講究視圖和數據的響應式，比如MVM模式和MVU模式等，而實現這些現代化模式的基礎就是實現了數據的雙向綁定

對於子組件來說，能夠實現雙向傳值是必要的，這決定了父組件能否將散發后的行為匯聚起來，當然，目前討論的重點在於確實將一個值在父組件和子組件間傳遞

1. 創建子組件
2. 編輯代碼

```
@Component
struct Person{
  @Link username:string
  @Link age:number

  build(){
    Column(){
      Row(){
        Text("子组件")
      }.width("100%").padding(10)
      .justifyContent(FlexAlign.Center).backgroundColor("#b1ae65")

      Column(){
        Row(){
          Text("姓名")
          TextInput({text:this.username}).onChange((val)=>{
            this.username = val
          }).flexBasis(220).backgroundColor("#FFF")
        }
        .width("100%").padding({left:10,right:10,top:10,bottom:3})
        .justifyContent(FlexAlign.SpaceBetween)
        .border({width:{bottom:1},color:"#EEE"})

        Row(){
          Text("年齡")
          TextInput({text:this.age + ""}).onChange((val)=>{
            this.age = parseInt(val)
          }).flexBasis(220).backgroundColor("#FFF").type(InputType.Number)
        }.width("100%").padding({left:10,right:10,top:10,bottom:3})
        .justifyContent(FlexAlign.SpaceBetween).border({width:{bottom:1},color:"#EEE"})
      }.flexGrow(1)
    }
  }
}
```



```

        .width(300).height(150).border({width:1})
    }
}

```

1. 編輯父組件

@Component

export default struct Test3{

@State username:string = "张三"

@State age:number = 23

@State sex:string = "张三"

build(){

Column(){

Column(){

Row(){

Text("父组件")

}.width("100%).padding(10)

.justifyContent(FlexAlign.Center).backgroundColor("#0F0")

Column(){

Row(){

Text("姓名")

TextInput({text:this.username}).onChange((val)=>{

this.username = val

}).flexBasis(220).backgroundColor("#FFF")

}.width("100%).padding({left:10,right:10,top:10,bottom:3})

.justifyContent(FlexAlign.SpaceBetween).border({width:{bottom:1},color:"#EEE"})

Row(){

Text("年龄")

TextInput({text:this.age + ""}).onChange((val)=>{

this.age = parseInt(val)

}).flexBasis(220).backgroundColor("#FFF").type(InputType.Number)

}.width("100%).padding({left:10,right:10,top:10,bottom:3})

.justifyContent(FlexAlign.SpaceBetween).border({width:{bottom:1},color:"#EEE"})

}.flexGrow(1)

}.width(300).height(150).border({width:1})

Person({username:\$username,age:\$age})

Person({username:\$username,age:\$age})

}.width("100%).height("100%).justifyContent(FlexAlign.SpaceEvenly)

}

}

後面沒好沒好。。。。

父组件直接点出属性，可以连续添加 比如

```
Row(){  
  
}.padding().width().balabala.....
```

通过扩展方法实现大量重复属性的继承

```
@Extend(Row){  
    .padding().width().balabala.....  
}
```

在Extend注解里面是需要被扩展的组件类型，将点出来的属性放在内容中

個人總結（吐槽）

暑期實訓 其之二

前言

接[上文](#)

此頁記錄實訓第二天內容，首先總結一下今日事項，主要概括為以下幾方面

- 補充第一天未講解的部分
- 熟悉各種組件及基本使用方法

主要内容

由於基本動作皆為在組件目錄中創建一個新組件來進行一輪嘗試，因此接下來僅展示部分主要代碼

基本使用

外部類使用

組件內容：

```
import Person from "../models/Person"

@Extend(Row) function rowBlock(){
    .width("100%").padding({left:10,right:10,top:10,bottom:3})
    .justifyContent(FlexAlign.SpaceBetween).border({width:{bottom:1},color:"#AAA"})
}

@Component
struct Item{
    @Link person : Person

    build(){
        Column(){
            Row(){
                Text("子组件")
            }.width("100%").padding(10)
            .justifyContent(FlexAlign.Center).backgroundColor("#ff65b17e")

            Column(){
                Row(){
                    Text("姓名")
                    TextInput({text:this.person.username}).onChange((val)=>{
                        this.person.username = val
                    }).flexBasis(220).backgroundColor("#FFF")
                }
            }
        }
    }
}
```

```

    }.rowBlock()

    Row(){
        Text("年龄")
        TextInput({text:this.person.age + ""}).onChange((val)=>{
            this.person.age = parseInt(val)
        }).flexBasis(220).backgroundColor("#FFF").type(InputType.Number)
    }.rowBlock()
    }.flexGrow(1)
}
.width(300).height(150).border({width:1})
}
}

```

@Component

```

export default struct Test4{
    @State person : Person = new Person()

    aboutToAppear(){
        this.person.username = "张三"
        this.person.age = 23
    }

    build(){
        Column(){
            Column(){
                Row(){
                    Text("父组件")
                }.width("100%").padding(10)
                .justifyContent(FlexAlign.Center).backgroundColor("#b1ae65")

                Column(){
                    Row(){
                        Text("姓名")
                        TextInput({text:this.person.username}).onChange((val)=>{
                            this.person.username = val
                        }).flexBasis(220).backgroundColor("#FFF")
                    }.rowBlock()

                    Row(){
                        Text("年龄")
                        TextInput({text:this.person.age + ""}).onChange((val)=>{
                            this.person.age = parseInt(val)
                        }).flexBasis(220).backgroundColor("#FFF").type(InputType.Number)
                    }.rowBlock()
                }.flexGrow(1)
            }
        }
    }
}

```

```

    }
    .width(300).height(150).border({width:1})

    Item({person:$person})

    Item({person:$person})

    Item({person:$person})
  }.width("100%").height("100%")
  .backgroundColor("#EEE").justifyContent(FlexAlign.SpaceEvenly)
}
}

```

類內容：

```

export default class Person{
  username : string
  age : number
  email : string

  constructor() {
    this.username = ""
    this.age = 0
    this.email = ""
  }
}

```

其中，我們將Person類的定義單獨放在了一個新文件中，通過語句

```
import Person from "../models/Person"
```

對其進行引用

熟悉各種組件

滾動條組件

```

@Component
export default struct Test4{
  @State array : Array<string> =[姓名列表，已去除內容]
  build(){
    Column(){
      Row(){
        Text("滚动条")

```

```

        }.width("100%").padding(10).justifyContent(FlexAlign.Center)
    Scroll(){
        Column(){
            ForEach(this.array, (item, index) => {
                Row(){
                    Text(item)
                }.width("100%").padding(10).margin(5)
                .backgroundColor("#EEE").justifyContent(FlexAlign.Center)
            })
        }
    }.scrollable(ScrollDirection.Vertical)
    .onScroll((x, y) =>{
        console.info(`position x:${x} y:${y}`)
    }).flexBasis(600).scrollBar(BarState.On)
}.width("100%").height("100%")
}
}

```

Scroll為滾動組件，可將其中內容進行滾動

默認進行豎向滾動，可通過屬性scrollable修改

滾動時觸發事件為onScroll，接受一個含雙參方法，輸入橫縱坐標

```

(method) ScrollAttribute.onScroll(event: (xOffset: number, yOffset: number) =>
void): ScrollAttribute

```

文本選擇器

```

@Component
export default struct Test6{
    @State selectArray : Array<string> = ["张飞", "关羽", "刘备", "赵云"]
    @State value : string = "张飞"
    @State selectArray1 : Array<string> = ["21", "22", "23", "24"]
    @State value1 : string = "21"

    build(){
        Column(){
            Row(){
                Text("三国人物")
                TextPicker({range:this.selectArray,value:this.value})
                    .onChange((val) => {
                        this.value = val
                    })
            }
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
    }
}

```

```

Row(){
    Text("选取值")
    Text(this.value)
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Row(){
    Text("年齡")
    TextPicker({range:this.selectArray1,value:this.value1})
        .onChange((val) => {
            this.value1 = val
        })
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Row(){
    Text("选取值")
    Text(this.value1)
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
}.width("100%").height("100%")
}
}

```

TextPicker為文本選擇器，可選中範圍內的文本，接受一個選項，常見取值範圍

```

const TextPicker: TextPickerInterface
(options?: TextPickerOptions) => TextPickerAttribute

```

所選內容發生變化時觸發onChange，接受含單參方法，傳入當前選擇內容

時間選擇器

```

@Component
export default struct Test7{
    @State hour:number = 0
    @State minute:number = 0

    date:Date = new Date("2024-7-9 14:00:00")

    build(){
        Column(){
            Row(){
                Text("时间选择器").fontSize(20).fontWeight(700)
            }.padding(10).width("100%").backgroundColor("#FF0")
            .justifyContent(FlexAlign.Center)

```

```

Row(){
    Text("出发时间")
    Text(`2024-7-9 ${this.hour}:${this.minute}:00`)
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Row(){
    Text("12小时制")

    TimePicker({selected:this.date}).width(200)
        .onChange((date) => {
            this.date = new Date(`2024-7-9 ${date.hour}:${date.minute}:00`)
        })
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Row(){
    Text("24小时制")

    TimePicker({selected:new Date(`2024-7-9 ${this.hour}:${this.minute}`)}).width(200)
        .onChange((date) => {
            this.hour = date.hour
            this.minute = date.minute
        }).useMilitaryTime(true)
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
}.width("100%").height("100%")
}
}

```

TimePicker時間選擇器，接受一個選項，與文本選擇器不同，一般直接確定的時間，可以在範圍內取值

```

const TimePicker: TimePickerInterface
(options?: TimePickerOptions) => TimePickerAttribute

```

內容更新時觸發onChange

可通過useMilitaryTime修改顯示時間格式

日期選擇器

```

@Component
export default struct Test8{
    @State date:DatePickerResult = {}
    @State selectedDate:string = ""

    aboutToAppear(){

```



```

this.selectedDate = "2024-7-9 00:00:00"
var now = new Date(this.selectedDate)
this.date.year = now.getFullYear()
this.date.month = now.getMonth()
this.date.day = now.getDay()
}

build(){
  Column(){
    Row(){
      Text("日期选择器").fontColor("#FFF").fontWeight(700).fontSize(20)
    }.width("100%").padding(10).justifyContent(FlexAlign.Center)
    .backgroundColor("#ff106922")

    Text(`${this.date.year}-${this.date.month}-${this.date.day} 00:00:00`)
    Text(new Date(`${this.date.year}-${this.date.month}-${this.date.day}
00:00:00`).toString())

    Row(){
      Text("出生年月")
      DatePicker({selected:new Date(this.selectedDate)})
        .width(200).onChange((date) => {
          this.date.year = date.year
          this.date.month = date.month + 1
          this.date.day = date.day
          this.selectedDate = `${date.year}-${date.month + 1}-${date.day} 00:00:00`
        console.log(`date year: ${date.year}`)
        })
    }.width("100%").justifyContent(FlexAlign.SpaceBetween).padding({ left:10,right:10 })
    .backgroundColor("#ff989898")

    Row(){
      Text("出生年月农历")
      DatePicker({selected:new Date(this.selectedDate)})
        .width(200).onChange((date) => {
          this.date.year = date.year
          this.date.month = date.month + 1
          this.date.day = date.day
          this.selectedDate = `${date.year}-${date.month + 1}-${date.day}
00:00:00`
        }).lunar(true)
    }.width("100%").justifyContent(FlexAlign.SpaceBetween).padding({ left:10,right:10 })
    .backgroundColor("#ff989898")

  }.width("100%").height("100%")
}
}

```

DatePicker日期選擇器，接受一個選項，常見默認選擇日期

```
const DatePicker: DatePickerInterface
(options?: DatePickerOptions) => DatePickerAttribute
```

可通過lunar指定顯示模式為陰曆，此時內部表示日期的值依然是陽曆

單選框

```
@Component
export default struct Test9{
  @State roleArray:Array<string> = ["管理员","研发人员","资料人员","测试人员","考勤员"]
  @State roleSelect:Array<boolean> = []
  @State selectIndex:number = 2

  aboutAppear(){
    this.roleArray.forEach(() => {
      this.roleSelect.push(false)
    })
    this.roleSelect[this.selectIndex] = true
  }

  build(){
    Column(){
      Row(){
        Text("角色列表").fontColor("#FFF")
      }.backgroundColor("#f200").width("100%").justifyContent(FlexAlign.Center).padding(10)

      Column(){
        ForEach(this.roleArray, (item, index) =>{
          Row(){
            Radio({group:"roleManage1",value:item}).margin({right:20})
              .checked(this.roleSelect[index])
              .onChange((val) => {
                this.roleSelect[index] = val
                if(val){
                  this.selectIndex = index
                }
              })
            Text(item)
          }.width(100)
        })
      }.height(200).justifyContent(FlexAlign.SpaceEvenly)

      Column(){
```

```

ForEach(this.roleArray, (item, index) =>{
  Row(){
    Radio({group:"roleManage2",value:item}).margin({right:20})
      .checked(this.roleSelect[index])
      .onChange((val) => {
        this.roleSelect[index] = val
        if(val){
          this.selectIndex = index
        }
      })
    Text(item)
  }.width(100)
})
}.height(200).justifyContent(FlexAlign.SpaceEvenly)

Row(){
  Text("选中角色")
  Text(this.roleArray[this.selectIndex])
}.width("100%").justifyContent(FlexAlign.SpaceBetween).padding(10)
}.width("100%").height("100%")
}
}

```

Radio為單選框，接受一個選項，不僅需要設定值，還要劃分組別

```

const Radio: RadioInterface
(options: RadioOptions) => RadioAttribute

```

checked在被選中時執行，onChange在選擇狀態改變時執行

暑期實訓 其之三

前言

接[上文](#)

此頁記錄實訓第三天內容，今日全程熟悉組件用法

首先給出結論，前三天的所有內容不出一個上午就可以全部掌握，之後應該也會繼續花幾天時間消耗在試用功能這種瑣事上，所以“名稱-使用示例”這樣的呈現方式可能還得持續些篇幅

某人認為更合適的學習路線

1. 基礎用法和概念直接查看[文檔](#)
2. 詳細信息去翻[項目](#)

另外，雖說鴻蒙開發和安卓開發名義上劃清界限了，但是按照目前看來至少界面的設計還是很有相通之處的。再加上縫合了多種現代語言的特點，所以對於有開發基礎的人來說，上手只會更快

主要内容

複選框組件

```
@Component
export default struct Test10{
    @State
    studentArray:Array<string> = ["小学","中学","高中","一本","二本","三本","研究生"]

    @State
    @Watch("countStudentName")
    checkArray:Array<boolean> = []

    @State
    selectIndex1:Array<number> = [2,3]

    @State
    selectIndex:Array<number> = []

    @State
    checkStudentName:string = ""

    countStudentName(){
        this.checkStudentName = ""
        this.selectIndex.splice(0, this.selectIndex.length)
```

```

this.checkArray.forEach((item, index) => {
    if(item){
        this.checkStudentName += this.studentArray[index] + "|"
        this.selectIndex.push(index)
    }
})
}

aboutToAppear(){
    this.studentArray.forEach(() => {
        this.checkArray.push(false)
    })

    this.selectIndex1.forEach((item) =>{
        this.checkArray[item] = true
    })
}

build(){
    Column() {
        Row() {
            Text("学历管理").fontColor("#FFF").fontSize(20).fontWeight(700)
        }.backgroundColor("#FF0").width("100%").padding(10).justifyContent(FlexAlign.Center)

        Column(){
            ForEach(this.studentArray, (item, index) => {
                Row() {
                    Checkbox({ group: "student", name: item })
                        .margin({ right: 40 }).select(this.checkArray[index])
                        .onChange((val) => {
                            this.checkArray[index] = val
                        })
                    Text(item)
                }.width(100)
            })
        }.height(200).justifyContent(FlexAlign.SpaceEvenly)

        Divider()

        Column(){
            ForEach(this.studentArray, (item, index) => {
                Row() {
                    Checkbox({ group: "student", name: item })
                        .margin({ right: 40 }).select(this.checkArray[index])
                        .onChange((val) => {
                            this.checkArray[index] = val

```

```

        })
        Text(item)
    }.width(100)
    })
}.height(200).justifyContent(FlexAlign.SpaceEvenly)

Divider()

Row() {
    Text("学历")
    Text(this.checkStudentName)
}.width("100%").padding(10).justifyContent(FlexAlign.Center)

Row() {
    Text("索引")
    Text(JSON.stringify(this.selectIndex))
}.width("100%").padding(10).justifyContent(FlexAlign.Center)

}.width("100%").height("100%")
}
}

```

下拉框组件

```

@Component
export default struct Test11{
    @State
    sexArray:Array<SelectOption> = [
        {value:"请选择"},
        {value:"男"},
        {value:"女"}
    ]

    @State
    sexValue:string = "请选择"

    @State
    runArray:Array<SelectOption> = [
        {value:"请选择"},
        {value:"足球"},
        {value:"篮球"},
        {value:"网球"},
        {value:"排球"}
    ]
}

```

```

@State
runIndex:number = 1

build(){
    Column(){
        Row(){
            Text("下拉列表")
        }.backgroundColor("#f466").width("100%").padding(10).justifyContent(FlexAlign.Center)

        Row(){
            Text("性别")
            Select(this.sexArray).value(this.sexValue)
                .onSelect((index,item) => {
                    this.sexValue = item
                })
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

        Row(){
            Text("性别值")
            Text(this.sexValue)
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

        Divider()

        Row(){
            Text("运动")
            Select(this.runArray).selected(this.runIndex)
                .value(this.runArray[this.runIndex].value.toString())
                .onSelect((index) => {
                    this.runIndex = index
                })
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

        Divider()

        Row(){
            Text("运动索引")
            Text(this.runIndex + "")
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

    }.width("100%").height("100%")
}
}

```

滑動組件

@Component

export default struct Test12{

@State

startFlag:boolean = true

@State

sendFlag:boolean = false

@State

stopFlag:boolean = false

build(){

Column(){

Row(){

Text("滑动组件").fontColor("#FFF").padding(20)

}.backgroundColor("#F330").width("100%").padding(10).justifyContent(FlexAlign.Center)

Row(){

Text("是否启动")

Toggle({type:ToggleType.Switch,isOn:this.startFlag}).onChange((val) => {
this.startFlag = val

})

}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()

Row(){

Text("是否启动")

Toggle({type:ToggleType.Switch,isOn:this.startFlag}).onChange((val) => {
this.startFlag = val

})

}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()

Row(){

Text("是否启动")

Text(this.startFlag + "")

}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()


```

Row(){
    Text("发送消息")
    Toggle({type:ToggleType.Checkbox,isOn:this.sendFlag}).onChange((val) => {
        this.sendFlag = val
    })
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()

Row(){
    Text("发送消息")
    Text(this.sendFlag + "")
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Row(){
    Text("停止运行")
    Toggle({type:ToggleType.Button,isOn:this.stopFlag}).onChange((val) => {
        this.stopFlag = val
    }).width(70)
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()

Row(){
    Text("停止运行")
    Toggle({type:ToggleType.Button,isOn:this.stopFlag}).onChange((val) => {
        this.stopFlag = val
    }).width(70).backgroundColor("#AAA").selectedColor("#0F0")
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

Divider()

Row(){
    Text("停止运行")
    Text(this.stopFlag + "")
}.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

}.width("100%").height("100%")
}
}

```

進度條組件

```

@Component
export default struct Test13{

    @State
    value:number = 0

    @State
    total:number = 250

    addProgress(){
        setTimeout(() => {
            this.addProgress()
            this.value += 10
            if(this.value > this.total + 20){
                this.value = 0
            }
        }, 1000)
    }

    aboutToAppear(){
        this.addProgress()
    }

    build(){
        Column(){
            Row(){
                Text("进度条").fontSize(20).fontColor("#FFF")
            }.backgroundColor("#FF0").width("100%").padding(10).justifyContent(FlexAlign.Center)

            Scroll(){
                Column(){
                    Row(){
                        Text("进度")
                        Text(this.value + "")
                    }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

                    Progress({value:this.value, total:this.total})

                    Progress({value:this.value, total:this.total, type:ProgressType.Ring}).width(200)
                        .rotate({angle:180}).color("#F00").backgroundColor("#00F")

                    Progress({value:this.value, total:this.total,
type:ProgressType.ScaleRing}).width(200)
                        .color("#FF0").backgroundColor("#F00")

```

```

        Progress({value:this.value, total:this.total,
type:ProgressType.Eclipse}).width(200)

        Progress({value:this.value, total:this.total,
type:ProgressType.Capsule}).width(200)

        }.justifyContent(FlexAlign.SpaceEvenly)
    }.flexGrow(1)
}.width("100%").height("100%")
}
}

```

搜索组件

```

@Component
export default struct Test14{

    @State
    searchVal:string = ""

    @State
    srcArray:Array<string> = [****]

    @State
    descArray:Array<string> = []

    @State
    scrollHeight:number = 0

    scroll:Scroller = new Scroller()

    aboutToAppear(){
        this.descArray = [...this.srcArray]
        this.scrollHeight = this.descArray.length * 40
    }

    build() {
        Column(){
            Row(){
                Text("搜索组件").fontColor("#FFF").fontSize(20)

            }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween).backgroundColor("#F511")

            Search({placeholder:"请输入条件", value:this.searchVal}).onSubmit((val) => {
                this.searchVal = val
            })
        }
    }
}

```

```

    this.descArray.splice(0, this.descArray.length)
    if(val === ""){
        this.descArray = [...this.srcArray]
    }
    else{
        this.srcArray.forEach((item) => {
            if(item.indexOf(val) !== -1){
                this.descArray.push(item)
            }
        })
    }
    this.scroll.scrollTo(Edge.Top)
    this.scrollHeight = this.descArray.length * 40
})

Scroll(this.scroll){
    Column(){
        ForEach(this.descArray, (item, index) => {
            Row(){
                Text(index + "")
                Text(item)
            }.width("100%").height(40).justifyContent(FlexAlign.SpaceBetween)
        })
    }
    }.flexGrow(1).backgroundColor("#F00").height(this.scrollHeight)

    }.width("100%")
}
}

```

這裏某人畫蛇添足了許多內容，起因是示例中發生了一個小問題，情況復現如下：

1. 未進行搜索時，顯示所有記錄，由於記錄較多，所以很自然地充斥整個界面
2. 執行搜索，示例中的邏輯為根據輸入關鍵詞進行局部匹配
3. 當篩選結束時，若匹配項較少，則這些項目會在豎直方向上居中顯示

根據講師對多個區域著色嘗試定位每個組件區域推測，應該不是示例原意，但是講師嘗試未果后放棄更正

某人排查后發現，是Scroll的父元素Column的height屬性被設為100%導致高度撐開了整個屏幕，當內部元素高度不夠時，被自動置中了，因此去掉該屬性賦值即可

期間嘗試使用option來創建scroll，在搜索執行結束後對Scroller類對象執行scrollEdge方法使其滾動到邊界，然而這對於自身高度小於父組件高度的scroll來說是無效的，因為顯示時已然為滾動至邊界狀態

此外還嘗試了固定元素的外觀如高度之類的屬性，雖然也沒有效果，但是期間某人才意識到是父元素的問題

該問題的排查暴露了某人在前端基礎上的不足，若是熟知常見的佈局模型，應該能少走彎路

面板組件

```
@Component
export default struct Test15{

  @State
  show:boolean = true

  @State
  birthday:DatePickerResult = {year:1980, month:1, day:4}

  birthdayValue:string = ""

  aboutToAppear(){
    this.birthdayValue = `${this.birthday.year}-${this.birthday.month}-${this.birthday.day}`
  }

  build(){
    Column(){
      Row(){
        Text("面板组件").fontColor("#FFF")
      }.width("100%").padding(10).justifyContent(FlexAlign.Center).backgroundColor("#F800")

      Row(){
        Text("出生年月")
        Text(`${this.birthday.year}-${this.birthday.month}-${this.birthday.day}`)
      }.width("100%").padding(10).justifyContent(FlexAlign.Center)
      .onClick(() => {
        this.show = !this.show
      })

      Divider()

      Panel(this.show){
        Row(){
          Text("编译出生日期")
        }.padding(10)

        Divider()

        DatePicker({selected:new Date(`${this.birthdayValue}
```

```

01:01:01`)).width(200).onChange((date)=>{
    this.birthday.year = date.year
    this.birthday.month = date.month + 1
    this.birthday.day = date.day
    this.birthdayValue =
`${this.birthday.year}-${this.birthday.month}-${this.birthday.day}`
    console.info("date year:" + JSON.stringify(date))
})

    Button("确认").width("95%).margin(10).onClick(() => {
        this.show = false
    })
}.miniHeight(100).mode(PanelMode.Mini)

}.width("100%).height("100%)
}
}

```

此處用到了DatePicker，和前一次使用時一樣，示例中將birthday同時作為被修改值用來記錄當前選定值，又作為選擇器打開時的選擇初值，這在使用IDE提供的模擬器運行時會出現選擇值跳動問題，並無法滾動到指定的內容

解決思路也是一樣的，既然selected屬性值只在一開始作用，而變值只在onChange方法中進行改變，那麼完全可以替換掉selected中對birthday的引用。option的具體生效方式可以去項目倉庫中搜索