GettingStart

前言

T-SQL 主要分爲三種類型的陳述式

- DDL
- DML
- DCL

DML

DML(Data Manipulation Language)

數據操作語言,用於操作和處理數據庫中的數據,主要涉及增刪改查等操作

常見用法

select

```
SELECT
        *
FROM
        Employees
```

如果有 `where` 條件,放在 `group by` 或 `order by` 之前

如果有複雜的查詢條件,在使用 EF Core 時,我傾向與分步執行后在内存中整合結果,在程序層處理數據

但是在sql中更加常用的方式是,將結果的查詢放在最外層,然後通過子查詢或公共表達式(Common Table Expressions)來不斷降低每一步的條件複雜度,也就是説將數據處理放在了數據庫層

子查詢示例

```
SELECT
        *
FROM
        Employees
WHERE
        EmployeeID IN (
            SELECT
                    EmployeeID
            FROM
                    Orders
            WHERE
```

```
                OrderDate > '2023-01-01'
        )
```

公共表達式示例

```
WITH RecentOrders AS (
        SELECT
                EmployeeID,
                COUNT(*) AS OrderCount
        FROM
                Orders
        WHERE
                OrderDate > '2023-01-01'
        GROUP BY
                EmployeeID
)

SELECT
        e.FirstName,
        e.LastName,
        r.OrderCount
FROM
        Employees e
        JOIN
        RecentOrders r
        ON
                e.EmployeeID = r.EmployeeID
```

有關 select 的更多使用範例可以查看微軟文檔

insert

```
INSERT INTO
        Employees (FirstName, Department)
VALUES
        ('Jane', 'Marketing');
```

插入表的時候可以選擇只插入部分列

例如上面例子中,Employees 表中可以有不只兩個列的屬性,但是在顯式指定了插入數據是屬於具體的哪些列后,完全少插入一些數據(在不違反完整性唯一性等等條件下)

注意

儘管有以下定義

```
-- Syntax for Azure Synapse Analytics and Parallel Data Warehouse and Microsoft Fabric

INSERT [INTO] { database_name.schema_name.table_name | schema_name.table_name | table_name }
    [ ( column_name [ ,...n ] ) ]
    {
      VALUES ( { NULL | expression } )
      | SELECT <select_criteria>
    }
    [ OPTION ( <query_option> [ ,...n ] ) ]
[;]
```

但事實上,不同的數據庫產品可能會有自己的 SQL 方言,所以强制添加而不是省略引數 into 是更好的做法.
後續示例也都基於這個規則

有關 insert 的更多使用範例可以查看微軟文檔

update

```
UPDATE
        Employees
SET
        Department = 'Marketing',
        FirstName = 'Jane'
WHERE
        LastName = 'Doe'
```

delete

```
DELETE FROM
        Employees
WHERE
        LastName = 'Doe'
```

DDL

DDL(Data Definition Language)

數據定義語言,用於定義和管理數據庫中的結構,主要在修改表結構而不是內容時使用

常見用法

create

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName NVARCHAR(50),
    LastName NVARCHAR(50),
    Department NVARCHAR(50)
)
```

alter

```
ALTER TABLE
        Employees
ADD
        Email NVARCHAR(100)


ALTER TABLE
        Employees
ALTER COLUMN
        LastName NVARCHAR(100)


ALTER TABLE
        Employees
DROP COLUMN
        Department
```

drop

```
DROP TABLE
        Employees
```

DCL

DCL(Data Control Language) 數據控製語言,用於控制對數據庫的訪問權限,主要涉及用戶權限等

其他操作

若非一般的操作,基本上我偏向於在程序代碼中完成,但是也有必須使用SQL語句的情況

merge

merge 是用於將數據插入,更新或刪除到目標表中的一種語句

常常會涉及源表和目標表的概念,并且可以在一個操作中完成複數個操作的功能

```
MERGE
INTO
        Employees AS Target
USING
        NewEmployees AS Source
ON
        Target.EmployeeID = Source.EmployeeID
WHEN
MATCHED
THEN
    UPDATE
    SET
            Target.FirstName = Source.FirstName,
            Target.LastName = Source.LastName,
            Target.Department = Source.Department
WHEN
NOT MATCHED
THEN
    INSERT
            (EmployeeID, FirstName, LastName, Department)
    VALUES
            (Source.EmployeeID, Source.FirstName, Source.LastName, Source.Department)
WHEN
NOT MATCHED BY SOURCE
THEN
DELETE
```

而在 EF Core 中,類似的功能實現如下

```
// 1. 查询所有匹配的记录
var matchedEmployees = context.Employees
    .Where(e => context.NewEmployees.Any(ne => ne.EmployeeID == e.EmployeeID))
    .ToList();

// 2. 更新匹配的记录
foreach (var employee in matchedEmployees)
{
    var newEmployee = context.NewEmployees
        .FirstOrDefault(ne => ne.EmployeeID == employee.EmployeeID);

    if (newEmployee != null)
```

```
    {
        employee.FirstName = newEmployee.FirstName;
        employee.LastName = newEmployee.LastName;
        employee.Department = newEmployee.Department;
    }
}

context.SaveChanges();

// 3. 插入未匹配的记录
var unmatchedNewEmployees = context.NewEmployees
    .Where(ne => !context.Employees.Any(e => e.EmployeeID == ne.EmployeeID))
    .ToList();

context.Employees.AddRange(unmatchedNewEmployees);
context.SaveChanges();

// 4. 删除不再匹配的记录
var employeesToDelete = context.Employees
    .Where(e => !context.NewEmployees.Any(ne => ne.EmployeeID == e.EmployeeID))
    .ToList();

context.Employees.RemoveRange(employeesToDelete);
context.SaveChanges();
```

索引

若數據表的規模較大,在查詢的時候需要花費的時間可能會漫長到無法等待,在列上建立索引可以加快查詢到速度,並有效避免遍歷全部記錄才能夠找到目標

數據表中的索引就相當於在原有的一列記錄上疊加了一個值從而達到優化目的

聚集索引

在表中設置主鍵后,將會自動創建一個聚集索引

聚集索引將會決定表中數據的物流存儲順序,并且只能存在一個聚集索引

除了主鍵,範圍查詢也適合使用聚集索引

```
CREATE CLUSTERED INDEX
        IX_Employees_EmployeeID
ON
        Employees(EmployeeID)
```

非聚集索引

非聚集索引像是對列數據創建了一張跳轉表,適合排序和分組

```
CREATE NONCLUSTERED INDEX
        IX_Employees_LastName
ON
        Employees(LastName)
```

唯一索引

唯一索引僅確保數據唯一,可以是聚集索引也可以是非聚集索引

```
CREATE UNIQUE INDEX
        IX_Employees_Email
ON
        Employees(Email);
```

全文索引

XML索引

空間索引

過濾索引

事務

函數

安裝使用

項目中也許會使用不同的數據庫,甚至可能同時涉及多個不同的數據庫.不管是開發還是維護,都有必要瞭解常見產品的基本使用

- SQL server
-

在 Web API 中使用 Swagger

创建一个新的 ASP.NET Core WebAPI 项目时，会同步使用 Swagger 作为 API 文档，我们可以通过在管线中自定义添加 Swagger 中间件时的实现，来达到对 Swagger 内容或执行逻辑的一些定制化支持

Swagger 默认会添加在 Program.cs 中，自动生成的相关代码如下

```
builder.Services.AddSwaggerGen();

...

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
        app.UseSwaggerUI();
}
```

通过修改服务的添加来将默认的 Swagger 改为我们想要的样子，首先将原本的 AddSwaggerGen 封装在自己的方法中

```
public static void InitSwagger(this IServiceCollection services)
{
    services.AddSwaggerGen();
}
```

根据 AddSwaggerGen 方法的定义，我们需要添加 SwaggerGenOptions 到方法中

```
public static IServiceCollection AddSwaggerGen(
    this IServiceCollection services,
    Action<SwaggerGenOptions> setupAction = null)
```

接下来，以不同版本的 API 接口为例。创建一个枚举类来标示版本

```
public enum APIVersionEnum
{
    Version_0_0 = 1,
    Version_1_0 = 2,
}
```

在 SwaggerGenOptions 中，需要根据不同的枚举值做出不同的操作

```
services.AddSwaggerGen(options =>
{
        typeof(APIVersionEnum).GetEnumNames().ToList().ForEach(version =>
        {
                // 处理逻辑
        });
});
```

额外地，需要在项目生成属性中勾选输出 API 文档，并在程序中反射到该文档

```
public static void InitSwagger(this IServiceCollection services)
{
    services.AddSwaggerGen(options =>
    {
        typeof(APIVersionEnum).GetEnumNames().ToList().ForEach(version =>
        {
            options.SwaggerDoc(version, version switch
            {
                "Version_0_0" => new OpenApiInfo()
                {
                    Title = "Version_0_0",
                    Version = "v0.0",
                    Description = "description",
                    Contact = new OpenApiContact()
                    {
                        Name = "author",
                        Url = new Uri("http://gz4nna.github.io")
                    }
                },
                "Version_1_0" => new OpenApiInfo()
                {
                    Title = "Version_1_0",
                    Version = "v1.0",
                    Description = "description",
                    Contact = new OpenApiContact()
                    {
                        Name = "author",
                        Url = new Uri("http://gz4nna.github.io")
                    }
                },
                _ => new OpenApiInfo()
                {
                    Title = "title",
                    Version = "v0.0",
                    Description = "description",
```

```
                    Contact = new OpenApiContact()
                    {
                        Name = "author",
                        Url = new Uri("http://gz4nna.github.io")
                    }
                }
            });
        });

        var xmlFileName = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml";
        options.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory,
xmlFileName), true);
    });
}
```

以上为对服务的修改，SwaggerDoc 定义如下

```
public static void SwaggerDoc(this SwaggerGenOptions swaggerGenOptions, string name,
OpenApiInfo info)
{
    swaggerGenOptions.SwaggerGeneratorOptions.SwaggerDocs.Add(name, info);
}
```

我们用 APIVersionEnum 中的枚举值将原本一整个 Swagger 文档分开，每部分展示的内容除了对 API 本身的手动划分外，还与指定的 OpenApiInfo 有关，常用的属性诸如 `Title`, `Description`, `Version`, `TermsOfService`, `Contact`, `License` 等等

接下来在路由中添加终结点

```
public static void InitSwagger(this WebApplication app)
{
    app.UseSwagger();
    app.UseSwaggerUI(options =>
    {
        typeof(APIVersionEnum).GetEnumNames().ToList().ForEach(version =>
        {
            options.SwaggerEndpoint($"/swagger/{version}/swagger.json", $"{version}");
        });
    });
}
```

最后将两处变更同步到 Program.cs 中，原本的代码变成了

```
builder.Services.InitSwagger();

...

if (app.Environment.IsDevelopment())
{
    app.InitSwagger();
}
```

现在对于一个新的 Controller，可以指定所属版本并在文档中体现出来

```
[ApiController]
[Route("api/[controller]/[action]")]
[ApiExplorerSettings(GroupName = nameof(APIVersionEnum.Version_0_0))]
```

如何封装一个 NuGet 包

创建一个类库,包含接口和实现,将需要的功能在方法中暴露,将使用的工具隐藏在方法的实现中 在 .csproj 文件中添加 NuGet 包的元数据

```
<PropertyGroup>
        <!-- 编译目标框架 -->
        <TargetFramework>net6.0</TargetFramework>

        <!-- 包唯一标识符 -->
        <PackageId>MyCustomLibrary</PackageId>

        <!-- 包版本 -->
        <Version>1.0.0</Version>

        <!-- 包作者 -->
        <Authors>YourNameOrCompany</Authors>

        <!-- 包描述 -->
        <Description>This is a library.</Description>

        <!-- 包版权信息 -->
        <Copyright>Copyright © 2024 YourCompany</Copyright>

        <!-- 包标签（关键词） -->
        <PackageTags>EFCore;Redis;Caching;Repository</PackageTags>

        <!-- 许可证表达式或自定义许可证文件 -->
        <PackageLicenseExpression>MIT</PackageLicenseExpression>
        <!-- <PackageLicenseFile>LICENSE.txt</PackageLicenseFile> -->
        <!-- 可选，使用自定义文件代替许可证表达式 -->

        <!-- 代码仓库 URL -->
        <RepositoryUrl>https://url</RepositoryUrl>

        <!-- 包图标（需添加到项目并设置“在输出目录中复制”） -->
        <Icon>icon.png</Icon>

        <!-- 发布日志 -->
        <ReleaseNotes>Basic repository and caching support.</ReleaseNotes>

        <!-- 项目主页或文档 URL -->
        <ProjectUrl>https://yourprojecthomepage.com</ProjectUrl>

        <!-- 是否要求用户在安装时接受许可证 -->
```

```
        <RequireLicenseAcceptance>true</RequireLicenseAcceptance>

        <!-- 简短描述 -->
        <Summary>A lightweight library for caching and data access.</Summary>

        <!-- 主要编程语言 -->
        <Language>en-US</Language>

        <!-- 包显示名称 -->
        <Title>My Custom Library</Title>

        <!-- 构建时生成 NuGet 包 -->
        <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
    </PropertyGroup>
```

如何将项目做成 Docker 镜像

在项目目录下制作 Dockerfile 文件,注意检查使用的端口

```
# 使用官方 .NET 8 SDK 镜像来构建应用程序
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

# 使用官方 .NET 8 SDK 镜像来进行构建
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

# 将项目文件复制到 Docker 容器中
COPY ["YourProjectName.csproj", "./"]

# 还原依赖项
RUN dotnet restore "./YourProjectName.csproj"

# 将项目的所有文件复制到容器中并进行构建
COPY . .
RUN dotnet publish "./YourProjectName.csproj" -c Release -o /app/publish

# 创建最终镜像并从生成的发布输出运行应用程序
FROM base AS final
WORKDIR /app
COPY --from=build /app/publish .
```

```
ENTRYPOINT ["dotnet", "YourProjectName.dll"]
```

曾经碰到情况为引用了类库,但是不在同一目录下,解决方法

1. 放弃使用类库,转为在该项目中实现
2. 在父级目录下创建 Dockerfile,拷贝使用的所有目录,并使用 .sln 文件来还原依赖

```
# 使用官方 .NET 8 SDK 镜像来构建应用程序
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src

# 将整个解决方案文件夹复制到容器中
COPY . .

# 还原整个解决方案的依赖项，以便处理所有项目的引用
RUN dotnet restore "SolutionFile.sln"

# 使用解决方案文件来构建项目并发布
RUN dotnet publish "MainProject/MainProject.csproj" -c Release -o /app/publish

# 使用运行时镜像来执行已发布的应用
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS final
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "MainProject.dll"]
```

在当前目录下运行 cmd 命令

```
docker build -t yourimagename .
```

最后创建容器,注意检查端口是否冲突

```
docker run -d -p 8080:80 --name yourcontainername yourimagename
```

使用 Redis

<折光编年史:reflaction>开发总结

十月底 TapTap 举办了 聚光灯GameJam 活动,我应小新邀请,在活动中作为一名程序参与制作了 <折光编年史:reflaction>

<折光> 是一款卡通风塔防游戏,玩家需要放置防御塔来保护基地不受敌人的攻击.防御塔主要使用光线对敌人攻击,另外设有辅助性质的塔,可提供反射和折射等功能

前期我主要负责了编写敌人相关的逻辑,后期负责整个项目的代码维护和功能添加.现在活动告一段落,遂回想一下这次经历所带来的收获

程序方面

从自己代码中总结

优雅致死,追求完美可能会损失效率

举个例子,在刚开始的阶段,我负责的是所有敌人的相关逻辑编写.出于 cs 的编码思维,我很自然地一开始就使用一个接口来描述所有敌人的基本属性和基本行为,然后继承自该接口,实现了一个敌人的基本类,再着手开始各种敌人的开发

在整个开发的后期,我这样的做法确实带来了很大的便利性.得益于我在设计阶段就考虑得很充分,后续不管增加什么样的敌人,大部分都只需要继承已有的类,然后重写几行特定的行为就可以了.甚至有些敌人可以在不修改代码而只是调整预制体参数设置的情况下完成

但是在开发前期,由于我需要把精力放在思考"如何设计才是更合理的"这个问题上面,导致另一位程序完成许多功能时我还依旧在各种方案之间取舍,结果最后被评价了"过于模式化"┬┬﹏┬┬但其实我完全可以每个类单独开发,重复的代码 CV 一下也花不了几秒钟,对性能也没有什么影响,至于优化完全可以在 PostJam 的阶段完成,那么为什么不呢?完全只是因为我觉得不优雅😭

上面的情况反映出来的**起步阶段的效率丢失**是一方面,另一方面是,GameJam 本身是一种生命周期极短的活动,一个游戏需要在几天之内完成,完成后也不一定会继续开发.就比如这次的<折光>,在策划案中的内容明显比实际开发的内容更加丰富,并且直到发布在 TapTap 上的前夕,成员们都还有继续开发后续内容的打算.然而现实是,直到现在也没有着手开始对游戏进行完善.于是乎,<折光>彻底变成了一上线就"死掉"的项目,我彻彻底底地做了一次**过度设计**

我明显地感觉到,自己学习的东西越多,越有一种想要在一开始就达成"完美设计"的强烈冲动,但是必须得考虑是否值得.短期且不需要重写的东西还是适当将重心放在追求速度上吧~

使用 StartCoroutine 开启协程达到延时效果

在编码时碰到一个需求:我要在指定时间间隔后执行一些逻辑

在 .NET 程序中,一般会采用 Task.Delay 方法结合 await/async 来实现这个功能,但是 Task 实际上是采用线程实现的,而在 Unity 中,尽管存在对 Task 的支持,但是在非主线程上获取物体是会出问题的,所以这时候需

要换用协程来处理

例如以下代码:

```
public bool Attackable
{
    get => attackable;
    set
    {
        attackable = value;
        if(attackable == false) StartCoroutine(ResetAttackableAfterDelay(AttackInterval));
    }
}

private IEnumerator ResetAttackableAfterDelay(float delay)
{
    yield return new WaitForSeconds(delay);
    attackable = true;
}
```

使用 `StartCoroutine` **方法开启一个协程**,在 `Unity` 运行到这个地方时,会在每一帧检查迭代器 `ResetAttackableAfterDelay` 中的 `WaitForSeconds` 方法是否完成,完成后才会执行语句 `attackable = true;`, 从而达到本处延时但不影响其他部分的正常运行(不会阻塞主线程)

对应的 `.NET` 版本应该是这样

```
private Task resetTask;

public bool Attackable
{
    get => attackable;
    set
    {
        attackable = value;
        if (attackable == false) resetTask = ResetAttackableAfterDelay(AttackInterval);
    }
}

private async Task ResetAttackableAfterDelay(int delay)
{
    await Task.Delay(delay);
    attackable = true;
}
```

使用 PlayerPrefs 存储临时数据

PlayerPrefs 是 Unity 提供的一个轻量级数据存储系统,适合用来**保存简单的数据**.

我一开始并不知道有它的存在,于是在单个场景中使用单例实现数据的临时存放.但是后续出现了要在多个场景之间进行变量传递的需求,此时我发现 PlayerPrefs 是一个很好的工具,就像一个随时可用的字典

- 存储数据

```
PlayerPrefs.SetString(string key, string value);
PlayerPrefs.Save();
```

- 获取数据

```
PlayerPrefs.GetString(string key, string defaultValue = "");
```

- 删除数据

```
PlayerPrefs.DeleteKey(string key);
```

甚至它可以在应用退出的时候自动保存数据,所以条件有限的时候还可以勉强用来做持久化.更多内容参考官方文档的描述

使用 JSON 来保存数据

和"困觉"的小伙伴们一起开发的时候,曾经尝试过将数据存放在 csv 中,而这次我选择了使用 JSON 来存放关卡信息.这样的选择主要考虑了几方面:

- 希望可以在游戏发布后方便地修改数据(自带老金)
- 没有尝试过在 Unity 中使用 JSON

首先,我将关卡信息放在 Unity 默认存放数据用的 Assets/Resources 文件夹下,这样可以确保使用 Resources.Load 方法获取文件

接下来,将持久化文件放在各个平台对应的默认存放位置,这个位置可以使用 Application.persistentDataPath 获取到

至此,完成了数据在目标平台的存放,读取也很简单

首先使用 File.ReadAllText 方法读取文件,再使用 JsonUtility.FromJson<T> 方法转换成相应的类型,这部分的操作和 .NET 程序中是完全一样的

总体上来说,在 Unity 中使用 JSON 来存放信息的**操作没有特殊的变化**.需要注意的是,虽然我在使用的时候一切正常,但是 Resources.Load 这个方法被描述为"不支持直接加载 JSON"(我使用的泛型类型为 TextAsset,估计把它当作 txt 去读了)

更加推荐的方式是将文件放入 Assets/StreamingAssets 文件夹,然后这样使用

```
string path = System.IO.Path.Combine(Application.streamingAssetsPath, "文件名");

UnityWebRequest request = UnityWebRequest.Get(path);
yield return request.SendWebRequest();

string jsonContent = request.downloadHandler.text;
// 解析
```

UnityWebRequest 是异步操作,需要用上文提到的方式去开启协程来调用.这种方式效率较低,一般用于**需要跨平台并确保发布时不易更改的静态文件**,比如这次的关卡信息就很合适

从他人代码中学习

使用 LineRender 制作射线

防御塔的主要攻击手段是发射光线到敌人身上,这部分的实现小伙伴采用了 LineRender 这个类型来绘制光线,这恰好是我不了解的知识

LineRender 作为组件挂载到 GameObject 上,简单的绘制激光主要需要设置:

- 线条宽度

```
lineRenderer.startWidth = 0.1f;
lineRenderer.endWidth = 0.1f;
```

- 线条颜色

```
lineRenderer.material = new Material(Shader.Find("Unlit/Color"));
lineRenderer.material.color = laserColor;
```

- 起点和终点

```
lineRenderer.SetPosition(0, laserOrigin.position);
lineRenderer.SetPosition(1, hit.point);
```

可以参考官方文档中对此的相关描述

安装 Aseprite

`Aseprite` 支持用户自行编译使用, 仓库位于这里

INSTALL.md 中有基本的安装指南, 以下是在win10上的实践

平台

我使用的是 Windows 10, 编译的 `Aseprite` 版本为 1.3.9

依赖

使用 Windows 编译 `Aseprite` 需要:

- Visual Studio VisualStudio 不是必须的, 使用 VisualStudio 是为了选择 `Visual Studio installer` 中的工作负荷 `使用c++的桌面开发`, 这个工作负荷会帮助安装好
  - 3.16以上版本的 CMake
  - Ninja

  这两个才是必要的, 可以自行安装, 如果使用 VS 的话, 可以在安装目录的 `Common7\IDE\CommonExtensions\Microsoft\CMake` 这个路径下找到这两个工具
- 编译后的 Skia 库 去看仓库中的 README 文件里推荐使用哪个 branch, 我在安装时使用了 m102

  下载发行版可以省去编译的环节,注意需要和平台对应, 我选择的是x64

确保拥有以上内容后, 可以在系统环境变量中将 cmake 和 ninja 添加到 Path 中, 遇到添加后不生效的情况可以关闭所有 cmd 后使用以下命令来刷新验证是否成功添加(或者重启来刷新):

```
echo %PATH%
```

编译

`Visual Studio` 的编译器可能无法直接被找到, 因此, 需要使用 `Developer Command Prompt for VS` 运行编译的指令

> **注意**

在开始菜单中的 `Developer Command Prompt for VS` 默认是32位, 如果编译目标为64位, 需要使用 `Common7\Tools\VsDevCmd.bat` 这个批处理

```
call "VsDevCmd.bat" -arch=x64
```

在 cmd 中运行以上命令后, 便进入了 `Developer Command Prompt for VS` 工具

由于 `Aseprite` 中 CMakeList 文件使用的编译器名称和 VS 中不同, 此时还需要进一步设定来帮助 CMake 指定编译器:

```
set CC=cl
set CXX=cl
```

现在, 可以在想要的输出目录下运行以下命令(根据自己的变量名替换):

```
cmake -DCMAKE_BUILD_TYPE=RelWithDebInfo -DLAF_BACKEND=skia -DSKIA_DIR=C:/deps/skia -
DSKIA_LIBRARY_DIR=C:/deps/skia/out/Release-x64 -DSKIA_LIBRARY=C:/deps/skia/out/Release-
x64/skia.lib -G Ninja ..
```

成功后, 运行:

```
ninja aseprite
```

生成的可执行文件在输出目录的 `\bin` 下

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，某人認爲該實訓雖和 實践 及 學習 均沾邊，但究其本源還只是課程性質罷了

該條目下將記錄爲期4週的實訓內容，僅作爲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：

实训时间		**实训项目名称**	**《鸿蒙北向应用开发》项目**	
		实训进度	**实训内容**	**涉及知识点**
第1天	上午	开班典礼 （实训启动会）	1、开班典礼仪式、校企方老师讲话 2、实训讲师介绍实训教学计划 3、鸿蒙生态介绍 4、实训项目介绍	1、鸿蒙生态
	下午	认识HarmonyOS 并搭建项目开发环境	1、HarmonyOS系统定义 2、HarmonyOS技术特性 3、HarmonyOS系统安全	1、HarmonyOS特色 2、HarmonyOS技术架构 3、开发环境

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爲公司自主研發（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等疑似欺詐行爲的可能性，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。

  本次實訓特地采用了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 是華爲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

  就是常説的封裝思想，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爲 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 Framwork 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

此爲實訓第一天內容，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

主要內容

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説明，沒什麽值得記錄的點

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 可以從華爲官網下載，注意該 IDE 可能需要使用華爲賬號來獲取，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，某人携帶了一臺長期閑置的筆電，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爲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 作爲版本控制工具

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

   找到通常名爲`ssh公鑰`或`ssh密鑰`等條目下，添加一條記錄，形式爲 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（分析説明之類的，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

某人創建了一個名爲**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爲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
```

這説明了將結構體（暫且這麽叫）作爲一個組件來使用，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';
```

采用了神似Vue的build()，基本佈局可以放在裏面

常用了Row和Column，常用屬性有wicth()和height()等，使用方法參考華爲提供的文檔

子組件 定義

前面所操作的内容均爲 Index.ets文件中的代碼，此時將其作爲一個根組件使用

子組件則是將某個組件引用到其他組件中，從而實現以組件爲單位的代碼復用

1. 創建新組件

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

   在 components 檔案夾下新增 ets 文件，我將其命名爲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("年龄")
          Text("" + 123)
        }.rowBlock().rowStyle()

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

3. 作爲子組件引用

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

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

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

子組件 單向傳值

子組件光是作爲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，那麽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麽

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MVVM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爲匯聚起來，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閒傳遞

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})
  }
}
```

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

主要內容

由於基本動作皆爲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)
  }
}
```

類内容:

```
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

滚動條組件

```
@Component
export default struct Test4{
  @State array : Array<string> =[姓名列表，已去除内容]
  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("年龄")
        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

某人認爲更合適的學習路綫

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

主要內容

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%")
 }
}
```

下拉框組件

```
@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%")
  }
}
```

搜索組件

```
@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.scrollEdge(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屬性被設爲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，在搜索執行結束后對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來説是無效的，因爲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同時作爲被修改值用來記錄當前選定值，又作爲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，那麽完全可以替換掉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

暑期實訓 其之四

前言

接上文

此頁記錄實訓第四天內容，除了和先前一樣的熟悉組件用法之外，還涉及了一些佈局相關的基礎

此外，講師提及接下來即將安排進行網絡授課學習，準備考取某認證

主要內容

首先延續先前內容，介紹了最後兩個常見的功能性控件元素的使用方法：列表組件和列表分組組件

功能性控件

列表組件

```
@Component
export default struct Test16{

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

  @Builder
  editRowInfo(item:string, index:number){
    Row(){
      Search({value:item, icon:""}).onSubmit((val) => {
        this.studentArray[index] = val
      })
    }.padding(10).height(50)

  }

  build(){
    Column(){
      Row(){
        Text("人员列表").fontColor("#FFF")

      }.backgroundColor("#F283").width("100%").padding(10).justifyContent(FlexAlign.Center)

      List(){
        ForEach(this.studentArray, (item, index) => {
          ListItem(){
            Row(){
              Text(index.toString())
```

```
            Text(item)
          }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
          .border({width:{bottom:1},color:"#BBB"}).height(50)
        }.swipeAction({end:this.editRowInfo(item,index)})
      })
    }.height("95%")
  }
 }
}
```

其中，由於在搜索框內引用了網絡資源，所以需要將應用的網絡訪問權限打開

在 module.json5 文件中為 module 添加如下內容

```
"requestPermissions": [
  {
    "name": "ohos.permission.INTERNET"
  }
],
```

列表分組組件

```
class GroupList{
  groupName:string
  groupArray:Array<string>
}

@Component
export default struct Test17{

  @State
  student:Array<GroupList> = [
      {
        groupName:"1",
        groupArray:["1a","1b","1c","1d","1e","1f"]
      },
      {
        groupName:"2",
        groupArray:["2a","2b","2c","2d","2e","2f"]
      },
      {
        groupName:"3",
        groupArray:["3a","3b","3c","3d","3e","3f"]
      }
```

```
    ]

  @State
  openArray:Array<Boolean> = [true, true, true]

  @Builder
  groupBar(item:GroupList, index:number){
    Row(){
      Text(item.groupName)

}.backgroundColor("#FDBB").width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
    .margin({bottom:10}).onClick(() => {
      this.openArray[index] = !this.openArray[index]
    })
  }


  build(){
    Column(){
      Row(){
        Text("分组统计").fontColor("#FFF").fontSize(20)
      }.backgroundColor("#F661").width("100%").padding(10).justifyContent(FlexAlign.Center)

      List(){
        ForEach(this.student, (groupItem, groupIndex) => {
          ListItemGroup({header:this.groupBar(groupItem,groupIndex)}){
            if(this.openArray[groupIndex]){
              ForEach(groupItem.groupArray, (item, index) => {
                ListItem(){
                  Row(){
                    Text(index.toString())
                    Text(item)
                  }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)
                  // .border({width:{bottom:1},color:"#AAA"})
                }
              })
            }
          }// .divider({strokeWidth:1, color:"#AAA"})
        })
      }.height("95%")

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

佈局控件

層疊佈局組件

```
@Component
export default struct Test18{
  build(){
    Column(){
      Row(){
        Text("层叠布局").fontSize(20).fontColor("#FFF")
      }.backgroundColor("#FA11").width("100%").padding(10).justifyContent(FlexAlign.Center)

      Scroll(){

        Column(){
          Stack(){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.Start}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.End}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.Top}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.Bottom}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.TopStart}){
```

```
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.TopEnd}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.BottomStart}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }

          Stack({alignContent:Alignment.BottomEnd}){
            Circle().width(150).height(150).colorBlend("#F00")
            Circle().width(100).height(100).colorBlend("#fbea")
            Circle().width(50).height(50).colorBlend("#f59a")
          }
        }      }.height("95%")
    }.width("100%").height("100%")
  }
}
```

這一段可謂是對stack的使用列出了非常詳盡的展示啊😂

元素重疊位置可以參考Alignment后屬性名的字面意思

網格佈局組件

```
class StudentInfo{
  id:string
  name:string
  email:string
}
@Component
export default struct Test19{
  @State
  studentArray:Array<StudentInfo> = [
    {id:" 1 ",name:" zhangsan1 ",email:" zhangsan1@qq.com "},
    {id:" 2 ",name:" zhangsan2 ",email:" zhangsan2@qq.com "},
    {id:" 3 ",name:" zhangsan3 ",email:" zhangsan3@qq.com "},
    {id:" 4 ",name:" zhangsan4 ",email:" zhangsan4@qq.com "},
```

```
    {id:" 5 ",name:" zhangsan5 ",email:" zhangsan5@qq.com "},
    {id:" 6 ",name:" zhangsan6 ",email:" zhangsan6@qq.com "},
    {id:" 7 ",name:" zhangsan7 ",email:" zhangsan7@qq.com "},
    {id:" 8 ",name:" zhangsan8 ",email:" zhangsan8@qq.com "},
    {id:" 9 ",name:" zhangsan9 ",email:" zhangsan9@qq.com "},
  ]

  build(){
    Column(){
      Row(){
        Text("网格布局").fontColor("#fff").fontSize(20)
      }.width("100%").justifyContent(FlexAlign.Center).backgroundColor("#f9c0").padding(10)

      Search({placeholder:"请输入条件"}).padding(10)

      Grid(){
        ForEach(this.studentArray, (item:StudentInfo, index) => {
          GridItem(){
            Column(){
              Row(){
                Text("编号")
                Text(item.id)
              }.width(130).justifyContent(FlexAlign.SpaceBetween).padding(10)

              Row(){
                Text("姓名")
                Text(item.name)
              }.width(130).justifyContent(FlexAlign.SpaceBetween).padding(10)

              Row(){
                Text("邮箱")
                Text(item.email).fontSize(7)
              }.width(130).justifyContent(FlexAlign.SpaceBetween).padding(10)

            }.width(130).height(140).border({width:1}).margin({bottom:10})
            .justifyContent(FlexAlign.SpaceAround)
          }
        })
      }.height("90%").columnsTemplate("1fr 1fr 1fr")
    }.width("100%").height("100%")
  }
}
```

對於某人來説，網格佈局是個人開發時最愛用的，這是因爲grid不僅能夠快速劃分出大致定位區域，還很適合進行響應式佈局

不過在ArkUI中，單純的Grid和GridItem不太能快速做出類似WPF一樣靈活的分塊操作

例如，Grid對不同格的劃分是以“fr”為單位，從IDE自帶API Reference中推測，似乎只能做到根據數值等分，無法限定其中一部分然後令其他按比例劃分

側面工具條

```
@Component
export default struct Test20{
  build(){
    Column(){
      Row(){
        Text("侧面工具栏").fontSize(20).fontColor("#fff")
      }.backgroundColor("#f182").width("100%").padding(10).justifyContent(FlexAlign.Center)

      SideBarContainer(SideBarContainerType.Overlay){
        Column(){
          Text("用户管理")
          Text("角色管理")
          Text("配置管理")
          Text("日志管理")
        }.backgroundColor(Color.Gray).justifyContent(FlexAlign.SpaceEvenly)

        Column(){
          Text("内容显示 内容显示 内容显示 内容显示 内容显示 内容显示")
        }.height("100%").justifyContent(FlexAlign.Center)
      }.height("100%").minSideBarWidth(50).sideBarWidth(90).maxSideBarWidth(150)
      .showSideBar(false)
      .controlButton({top:10,left:10})
      .sideBarPosition(SideBarPosition.End)
      .autoHide(false)

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

路由表組件

```
import router from '@ohos.router'
@Component
export default struct Test21{
  build(){
    Column(){
      Row(){
        Text("路由表").fontColor("#fff").fontSize(20)
```

```
      }.backgroundColor("#f290").width("100%").padding(10).justifyContent(FlexAlign.Center)

      Row(){
        Navigator({target:"pages/UserPage"}){
          Text("用户管理")
        }

        Navigator({target:"pages/RolePage",type:NavigationType.Replace}){
          Text("角色管理")
        }
      }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

      Row(){
        Button("用户管理").onClick(() => {
          router.pushUrl({url:"pages/UserPage"})
        })

        Button("角色管理").onClick(() => {
          router.replaceUrl({url:"pages/RolePage"})
        })
      }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

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

頁面跳轉是非常重要的功能，該示例展示了 Navigator 的用法

可以看到，Navigator 實現跳轉的邏輯還是使用一個棧來管理路由中的頁面，默認就是將 target 所指的頁面直接壓入棧，這一點做法和 Android 以及 xamarin 采用的策略都是基本一致的

在目錄中新建兩個 page 如下

```
import router from '@ohos.router'
@Entry
@Component
struct RolePage {
  @State message: string = '角色管理'

  build() {
    Column(){
      Row(){
        Row(){
          Text("角色管理").fontSize(20).fontColor("#fff").margin({right:20})
            .onClick(() => {
```

```
            router.back()
          })
        }.flexGrow(1).justifyContent(FlexAlign.Center)


}.backgroundColor("#ff0").width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

      Text(this.message).fontSize(50).fontWeight("100%")
    }
  }}
```

```
import router from '@ohos.router'
@Entry
@Component
struct UserPage {
  @State message: string = '用户管理'

  build() {
    Column(){
      Row(){
        Row(){
          Text("用户管理").fontSize(20).fontColor("#fff").margin({right:20})
            .onClick(() => {
              router.back()
            })
        }.flexGrow(1).justifyContent(FlexAlign.Center)


}.backgroundColor("#ff0").width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

      Text(this.message).fontSize(50).fontWeight("100%")
    }
  }}
```

此時查看 entry/src/main/resources/base/profile 中的 main_pages.json 内容

```
{
  "src": [
    "pages/Index",
    "pages/UserPage",
    "pages/RolePage"
  ]
}
```

可以看到新頁面的 URI 已經被自動添加到 src 下了，這裏就是 target 需要指向的路徑了

個人總結（吐槽）

將近一周的開發，差不多也認識了ArkUI的基本使用邏輯了

不管某人是否認同，但是ArkUI確實有便捷之處

可以很明顯地感覺到，在使用ArkUI的時候，很多的控件的屬性和方法都具有較高的共性，甚至某人原本第一反應是使用模型綁定加自寫方法實現的邏輯也可以找到封裝好的基本方法。不是説明其豐富性高，某人猜測可能是將UI都以一塊在熒幕上渲染的圖形為基礎，而不是將其作爲類的結果。這樣的優勢也非常明顯——就算某人現在認識的控件不多，也可以把一個基本方法套用在很多不認識或者功能看上去完全沒關係的控件上（就算實在不行了，那不還有擴展方法和模型綁定麼😂）

暑期實訓 其之五

前言

接上文

此頁記錄實訓第五天內容，這一次反而又開始試用控件了

大體流程和前些天一樣，給出示例后復現，然後下一個控件，沒什麽值得記錄的地方

主要内容

路由传参

```
import router from '@ohos.router'
@Component
export default struct Test22{

  @State
  userName:string = ""


  build(){
    Column(){
      Row(){
        Text("路由传参").fontColor("#fff").fontSize(20)
      }.backgroundColor("#f065").width("100%").padding(10).justifyContent(FlexAlign.Center)

      Divider()

      Row(){
        Text("参数")
        TextInput({placeholder:"请输入参数", text:this.userName}).onChange((val) => {
          this.userName = val
        })
      }

      Divider()

      Button("提交").onClick(() => {
        var reg = new RegExp("^.+$")
        if(this.userName == null || this.userName == "" || !reg.test(this.userName)){
          AlertDialog.show({message:"请输入用户名称"})
          return
        }
        router.pushUrl({url:"pages/ParamerPage",params:{"userName":this.userName}})
```

```
      })

      Button("提交1").onClick(() => {
        router.pushUrl({url:"pages/ParamerPage"})
      })
    }
  }}
```

在DAY04中曾經嘗試過了直接使用的路由跳轉功能，但是在實際使用中，跳轉前後的兩個頁面基本上都是需要有數據交換的

此時可以通過在RouterOptions裏添加所傳參數

```
interface RouterOptions {
        url: string;
        params?: Object
}
```

作爲接受跳轉的頁面

```
import router from '@ohos.router'
@Entry
@Component
struct ParamerPage {
  @State
  message: string = '接收参数'

  @State
  userName:string = ""

  aboutToAppear(){
    var param = router.getParams()["userName"]
    if(param){
      this.userName = router.getParams()["userName"]
    }
  }
  build() {
    Column(){
      Row(){

        Row(){
          Text("接收参数").fontSize(20).fontColor("#fff")
            .onClick(() => {
              router.back()
```

```
            })
        }.flexGrow(1).justifyContent(FlexAlign.Center)

      }.backgroundColor("#f170").width("100%").padding(10)

      Row(){
        Text("用户名")
        Text(this.userName)
      }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

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

可以通過 router.getParams()[參數鍵名] 來獲取傳入的內容

轮播组件

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

@Component
export default struct Test23{
  build(){
    Stack({alignContent:Alignment.BottomEnd}){
      Swiper(){
        Text("1")
        Text("2")
        Text("3")
        Text("4")
        Text("5")
      }.indicatorStyle({size:25, color:"#f00", selectedColor:"#0f0"})
      .autoPlay(true).onAnimationEnd((index) => {
        if(index == 3){
          router.replaceUrl({url: "pages/RolePage"})
        }
      })

      Text("跳转应用").fontColor("#0f0").padding(10).fontSize(20)
        .onClick(() => {
          router.replaceUrl({url: "pages/RolePage"})
        })
    }.width("100%").height("100%")
  }
}
```

Swiper 可以輪播其中元素

通過 indicatorStyle 可以指定輪播時底部指示條的樣式，onAnimationEnd 可以指定在一次切換動畫結束時執行的動作

```
@Component
export default struct Test24{

  control:SwiperController = new SwiperController()


  build(){
    Column(){
      Stack(){
        Swiper(this.control){
          Text("1")
          Text("2")
          Text("3")
          Text("4")
          Text("5")
        }.indicatorStyle({size:25, color:"#f00", selectedColor:"#0f0"})
        .autoPlay(true)

        Row(){
          Text("上一个").fontColor("#0f0").fontWeight(800)
            .onClick(() => {
              this.control.showPrevious()
            })
          Text("下一个").fontColor("#0f0").fontWeight(800)
            .onClick(() => {
              this.control.showNext()
            })
        }.width("100%").padding(10).justifyContent(FlexAlign.SpaceBetween)

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

tabs组件

```
@Component
export default struct Test25{

  @State
```

```
  selectIndex:number = 0

  control:TabsController = new TabsController()

  @Builder
  tarBar(text:string, index:number){
    Column(){
      if(this.selectIndex === index) {
        Text(text).fontSize(20).fontColor("#f00").fontWeight(600)
      }
      else{
        Text(text).fontSize(20).fontWeight(600)
      }
    }.onClick(() => {
      this.control.changeIndex(index)
    })
  }


  build(){
    Tabs({barPosition:BarPosition.End, index:this.selectIndex}) {

      TabContent() {
        Text("主页").fontSize(40)
      }.tabBar(this.tarBar("主页", 0))

      TabContent() {
        Text("查找").fontSize(40)
      }.tabBar(this.tarBar("查找", 1))

      TabContent() {
        Text("通讯录").fontSize(40)
      }.tabBar(this.tarBar("通讯录", 2))

      TabContent() {
        Text("我的").fontSize(40)
      }.tabBar(this.tarBar("我的", 3))

    }.onChange((index) => {
      this.selectIndex = index
    })
  }
}
```

tabs 控件常常用作單個頁面中的內容切換，也可以結合路由實現導航的功能

走马灯组件

```
const contents:string = "阿斯顿法国红酒看来去微软推哦怕自行车v不那么去啊自我实现二的吃软饭v提供必要和奴家拍咯咯密集怒吼吧应该v她非常弱的学而思欺骗我欸如天涯螺丝扣搭街坊换个名字那些"

@Component
export default struct Test26{

  @State
  start:boolean = true

  @State
  from:boolean = true

  @State
  step:number = 6


  build(){
    Column(){
      Row(){
        Text("走马灯").fontColor("#fff").fontSize(20)
      }.width("100%").padding(10).justifyContent(FlexAlign.Center).backgroundColor("#f055")

      Column(){
        Marquee({start:this.start,src:contents,fromStart:this.from,step:this.step})

        Row(){
          Button("开始").onClick(() => {
            this.start = true
          })

          Button("停止").onClick(() => {
            this.start = false
          })
        }.justifyContent(FlexAlign.SpaceBetween).width("100%")

        Row(){
          Button("正向").onClick(() => {
            this.from = true
          })

          Button("反向").onClick(() => {
            this.from = false
          })
        }.justifyContent(FlexAlign.SpaceBetween).width("100%")
```

```
        Row(){
          Button("加速").onClick(() => {
            this.step += 6
          })

          Button("减速").onClick(() => {
            this.step -= 6
          })
        }.justifyContent(FlexAlign.SpaceBetween).width("100%")

      }.flexGrow(1).justifyContent(FlexAlign.SpaceAround)

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

屏幕监听

```
import mediaquery from '@ohos.mediaquery'
@Component
export default struct Test27{

  @State
  horizontalFlag:boolean = true

  listener:mediaquery.MediaQueryListener = mediaquery.matchMediaSync("
(orientation:landscape)")

  aboutToAppear(){
    this.listener.on("change",(result:mediaquery.MediaQueryResult) => {
      this.horizontalFlag = result.matches
    })
  }


  build(){
    if(this.horizontalFlag){
      Row(){
        Column(){
          Text("横屏").fontSize(30)
        }.flexBasis(50).padding(10)
        .border({width:{right:1}}).height("100%").justifyContent(FlexAlign.Center)

        Column(){
```

```
          Text("通过不会被大风吹v哦空么大出风头应该会")
        }.flexGrow(1).justifyContent(FlexAlign.Center)

      }.width("100%").height("100%")
    }
    else{
      Column(){
        Row(){
          Text("竖屏").fontSize(30)
        }.width("100%").padding(10).border({width:
{bottom:1}}).justifyContent(FlexAlign.Center)

        Column(){
          Text("去啊是的乳房v关于何炅i看楼主下次发给")
        }.flexGrow(1).justifyContent(FlexAlign.Center)

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

名稱為屏幕監聽，實際上示例中只給出了設備橫竪屏狀態監聽，有關監聽器的功能應該還有其他的，待查看文檔后補充

对话框组件

```
import promptAction from '@ohos.promptAction'

const button_confirm:number = 0
const button_cancel:number = 1

@Component
export default struct Test28{

  @State
  message:string = ""


  build(){
    Column(){
      Row(){
        Text("弹出框").fontSize(20).fontColor("#fff")
      }.width("100%").padding(10).justifyContent(FlexAlign.Center).backgroundColor("#f274")

      Column(){
```

```
Button("弹出框").onClick(() => {
  promptAction.showToast({message:"好好学习"})
})

Button("弹出框1").onClick(() => {
  promptAction.showToast({message:"好好学习1", duration:2000, bottom:400})
})

Button("对话框").onClick(() => {
  promptAction.showDialog({message:"登陆成功"})
})

Text("点击了" + this.message)

Button("对话框1").onClick(() => {
  promptAction.showDialog({message:"登陆成功1",buttons:[
    {text:"确认",color:"#00f"},
    {text:"取消",color:"#00f"},
    {text:"复制",color:"#00f"}
  ]}).then((event) => {
    console.info("prompAction button:click" + JSON.stringify(event))
    if(event.index === button_confirm){
      this.message = "对话框-确认按钮"
    }
    else if(event.index === button_cancel){
      this.message = "对话框-取消按钮"
    }
  })
})

Button("alert对话框").onClick(() => {
  AlertDialog.show({message:"登陆失败"})
})

Button("alert对话框1").onClick(() => {
  AlertDialog.show({message:"认证成功",confirm:{
    value:"确认",action:() => {
      this.message = "alert对话框-确认"
    }
  }})
})

Button("alert对话框1").onClick(() => {
  AlertDialog.show({
    message:"认证成功",
    primaryButton:{
```

```
            value:"确认",
            action:() => {
              this.message = "alert对话框-primaryButton"
            }
          },
          secondaryButton:{
            value:"取消",
            action:() => {
              this.message = "alert对话框-secondaryButton"
            }
          }          })
      })


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

暑期實訓 其之六

前言

接上文

第六天

主要內容

自定義對話框

```
@CustomDialog
struct DialogUi{
  controller:CustomDialogController
  build(){
    Column(){
      Row(){
        Text("角色分配")
      }

      Divider().margin(10)

      Column(){
        Row(){
          Checkbox()
          Text("管理员")
        }.width(150)

        Row(){
          Checkbox()
          Text("开发人员")
        }.width(150)

        Row(){
          Checkbox()
          Text("调试人员")
        }.width(150)
      }

      Divider().margin(10)

      Row(){
        Button("确定").onClick(() => {
          AlertDialog.show({message:"登录成功"})
        })
```

```
        Button("取消").onClick(() => {
          this.controller.close()
        })
      }.width(200).justifyContent(FlexAlign.SpaceBetween)
    }.padding(10)
  }
}

@Component
export default struct Test29{

  controller:CustomDialogController = new CustomDialogController({
    builder:DialogUi()
  })
  build(){
    Button("打开对话框").onClick(() => {
      this.controller.open()
    })
  }
}
```

屬性動畫

```
@Component
export default struct Test30{

  @State
  cWidth:number = 80

  @State
  angle:number = 0

  build(){
    Column(){
      Row(){
        Text("属性动画").fontColor("#fff").fontSize(20)
      }.width("100%").padding(10).justifyContent(FlexAlign.Center).backgroundColor("#f671")

      Column(){

        Row().width(this.cWidth).height(this.cWidth).backgroundColor("#f00").margin(20)
          .borderRadius(this.cWidth).onClick(() => {
          if(this.cWidth === 80){
            this.cWidth *= 4
          }
```

```
          else{
            this.cWidth = 80
          }
        }).animation({
          duration:2000,
          playMode:PlayMode.Alternate,
          curve:Curve.EaseInOut,
          iterations:-1
        })

        Row().width(100).height(10).backgroundColor("#f0f").rotate({angle:this.angle})
          .onClick(() => {
            this.angle += 180
          })
          .animation({
            duration:2000,
            playMode:PlayMode.Normal,
            curve:Curve.EaseInOut,
            iterations:-1
          })


Row().width(100).height(10).backgroundColor("#f182").rotate({angle:this.angle,y:200})
          .onClick(() => {
            this.angle += 180
          })
          .animation({
            duration:2000,
            playMode:PlayMode.Normal,
            curve:Curve.EaseInOut,
            iterations:-1
          })


Row().width(100).height(10).backgroundColor("#fba1").rotate({angle:this.angle,x:200})
          .onClick(() => {
            this.angle += 180
          })
          .animation({
            duration:2000,
            playMode:PlayMode.Normal,
            curve:Curve.EaseInOut,
            iterations:-1
          })

      }.flexGrow(1).justifyContent(FlexAlign.SpaceEvenly)
```

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

顯示動畫

```
@Component
export default struct Test31{
  @State diceImage:Resource = $r("app.media.app_icon")
  @State imageArray:Array<Resource> =
[$r("app.media.app_icon"),$r("app.media.app_icon"),$r("app.media.app_icon"),$r("app.media.ap
p_icon")]
  @State angle:number = 0
  @State baArray:Array<Resource> =
[$r("app.media.app_icon"),$r("app.media.app_icon"),$r("app.media.app_icon"),$r("app.media.ap
p_icon")]
  @State baImage:Resource =$r("app.media.app_icon")
  @State baIndex:number = 0
  @State marginRight:number=0
  random(){
    setTimeout(()=>{
      this.random()
      var random = Math.random()*6
      var index = parseInt(random.toString())
      animateTo({playMode:PlayMode.Normal,
        iterations:index+1,
        curve:Curve.EaseInOut,
        duration:1000},()=>{
        this.diceImage = this.imageArray[index]
        this.marginRight += 100
      })
    },1000)
  }
  ba(){
    setTimeout(()=>{
      this.ba()
      this.baIndex++
      if (this.baIndex>8) {
        this.baIndex = 0
      }
      animateTo({
        playMode:PlayMode.Normal,
        iterations:this.baIndex+1,
        curve:Curve.EaseInOut,
```

```
        duration:500
      },()=>{
        this.baImage=this.baArray[this.baIndex]
        this.marginRight += 5
        if (this.marginRight>200) {
          this.marginRight = 0
        }
      })
    },500)
  }
  build(){
    Column(){
      Row(){
        Text("显示动画").fontSize(20).fontColor("#FFF")
      }.width("100%").padding(10).justifyContent(FlexAlign.Center)
      .backgroundColor("#ff173c86")
      Row(){
        Image(this.diceImage).width(50).rotate({angle:this.angle,y:200})
      }.flexBasis(100).width("100%").padding(10).justifyContent(FlexAlign.Center)
      Row(){
        Button("摇骰子").onClick(()=>{
          this.random()
        })
      }
      Row(){
        Image(this.baImage).width(150).margin({right:this.marginRight})
      }.flexBasis(200).width("100%").padding(10).justifyContent(FlexAlign.End)
      .backgroundColor("#f00")
      Row(){
        Button("孙悟空").onClick(()=>{
          this.ba()
        })
      }
    }.width("100%").height("100%")
  }
}
```

轉場動畫

```
import router from '@ohos.router'
@Component
export default struct Test32{
  @State myOpacity:number = 1

  pageTransition(){
```

```
    PageTransitionEnter({
      duration:500,
      curve:Curve.EaseIn
    }).onEnter((type: RouteType, progress: number) => {
      this.myOpacity = progress
    })
    PageTransitionExit({
      duration:500,
      curve:Curve.EaseOut
    }).onExit((type: RouteType, progress: number) => {
      this.myOpacity = 1 - progress
    })
  }

  build(){
    Column(){
      Row(){
        Text("转场动画").fontColor("#FFF").fontSize(20).fontWeight(700)

}.backgroundColor("#ff257e38").width("100%").padding(10).justifyContent(FlexAlign.Center)

      Column(){
        Button("汽车之家").onClick(()=>{
          router.pushUrl({url:"pages/CarHomePage"})
        })

        Button("教师之家").onClick(()=>{
          router.pushUrl({url:"pages/TeacherHomePage"})
        })
      }.flexGrow(1).justifyContent(FlexAlign.SpaceEvenly).opacity(this.myOpacity)
    }.width("100%").height("100%")
  }
}
```

插槽注解

```
import StudentInfo from "../models/StudentInfo"
@Component
struct ListInfo{
  @Link list:Array<StudentInfo>
  @BuilderParam title:()=>void
  @BuilderParam showListItem:(data:StudentInfo,index?:number)=>void
  build(){
    Column() {
      Row(){
```

```
        this.title()
      }.width("100%").padding(10)
      List() {
        ForEach(this.list, (item: StudentInfo, index) => {
          ListItem() {
            this.showListItem(item,index)
          }
        })
      }
    }  }}

@Component
export default struct Test33{
  @State
  list:Array<StudentInfo> = [
    {name:"张三",mobile:"123564678"},
    {name:"李四",mobile:"156786464"},
    {name:"王五",mobile:"194564634"},
  ]

  @Builder
  titleShow(){
    Row(){
      Text("标题").fontColor("#FFF").fontSize(20).fontWeight(700)

}.backgroundColor("#ff4def12").width("100%").padding(10).justifyContent(FlexAlign.Center)
  }

  @Builder
  LItemShow(item:StudentInfo,index:number){
    if(index % 2 === 0){
      Row(){
        Text(index.toString())
        Text(item.name).fontSize(20)
        Text(item.mobile).fontSize(20)

}.width("100%").justifyContent(FlexAlign.SpaceBetween).backgroundColor("#EEE").padding(10)
    }else{
      Row(){
        Text(index.toString())
        Text(item.name).fontSize(20)
        Text(item.mobile).fontSize(20)

}.width("100%").justifyContent(FlexAlign.SpaceBetween).backgroundColor("#EE0").padding(10)
    }
  }
```

```
  build(){
    Column(){
      Row(){
        Text("插槽注解").fontColor("#FFF").fontSize(20).fontWeight(700)

}.backgroundColor("#ff12deef").width("100%").padding(10).justifyContent(FlexAlign.Center)

      ListInfo({list:$list,title:this.titleShow,showListItem:this.LItemShow})
    }.width("100%").height("100%")
  }
}
```

插槽對話框

```
@CustomDialog
struct DialogUI{
  @Prop title:string
  @BuilderParam body:()=>void
  confirmAction:()=>void
  dialogControl:CustomDialogController
  build(){
    Column(){
      Row(){
        Text(this.title).fontSize(20).fontColor("#FFF")

}.backgroundColor("#fff30dba").width("100%").padding(10).justifyContent(FlexAlign.Center)
      this.body()
      Row(){
        Button("确认").onClick(()=>{
          this.confirmAction()
        })
        Button("取消").onClick(()=>{
          this.dialogControl.close()
        })
      }.width(200).padding(10).justifyContent(FlexAlign.SpaceBetween)
    }.padding(10)
  }
}

@Component
export default struct Test34{
  dialogControl:CustomDialogController = new CustomDialogController({
    builder:DialogUI({
      title:"分配角色",
```

```
      body:this.dialogBody,
      confirmAction:()=>{
        AlertDialog.show({message:"分配角色失败"})
      }
    })
  })

  @Builder
  dialogBody(){
    Column(){
      Row(){
        Checkbox()
        Text("用户管理")
      }

      Row(){
        Checkbox()
        Text("角色管理")
      }

      Row(){
        Checkbox()
        Text("配置管理")
      }
    }  }
  build(){
    Column(){
      Row(){
        Text("尾随闭合函数").fontColor("#FFF").fontSize(20).fontWeight(700)

}.backgroundColor("#fff30dba").width("100%").padding(10).justifyContent(FlexAlign.Center)

      Column(){
        Button("打开").onClick(()=>{
          this.dialogControl.open()
        })
      }.flexGrow(1)
    }.width("100%").height("100%")
  }
}
```

暑期實訓 其之十一

前言

涉及的想法僅為個人觀點，甚至日後會修改也不一定，請務必批判而不是認同

從第三周開始，需要運用此前通過示例學會的組件知識

機構提供了一個學習平臺，並同步提供一系列api供調用，現在需要開發一個應用對接該平臺，稱作雲課堂項目

需要擁有的story如下

Story编号	Story模块	Story名称	story描述
1	用户模块	用户登录	用户名、密码、验证滑动
2	用户模块	用户注册	邮箱、验证码、用户密码、确认密码
3	用户模块	忘记密码	邮箱、验证码、用户密码、确认密码
4	课程模块	全部课程	公开课程和定制课程(学生和班级、课程相关联)
5	课程模块	公开课程	公开课程
6	课程模块	定制课程	定制课程(学生和班级、课程关联)
7	课程模块	课程目录	章节和计划，计划下可以关联视频、文档
8	课程模块	课程评价	5星评分、点赞、留言
9	课程模块	我的笔记	课程、学生相关联，留言

Story编号	Story模块	Story名称	story描述
10	课程模块	视频播放	全屏、开始、暂停、进度条
11	课程模块	文档预览	文档预览
12	班级模块	全部班级	公开班级、我的班级((学生和班级相关联)
13	班级模块	我的班级	我的班级((学生和班级相关联)
14	班级模块	班级详情	班主任、学员数量、班级简介、班级名称、加入班级
15	我的模块	个人列表	个人图标、用户名、班级名称、二维码、学习计划、测评记录、学习记录、证书、机构注册与服务协议、班级信息处置规范、隐私政策、常见问题
16	我的模块	修改人个信息	修改 用户名称、邮箱、电话、昵称

主要内容

總體設計

應用結構

某人打算沿用此前使用MAUI進行安卓開發時候的思路，使用MVVM模式進行前端頁面開發，并在必要時候通過發送http請求來獲取後端web api提供的各種數據來豐富展示數據

想要結構合乎MVVM，除了最顯而易見的對視圖和視圖模型進行數據綁定外，需要合理設計模型。通過分析story要求，認爲所有動作均是圍繞用戶，課程，班級三者發出的，因此首先設計了`User`、`Course`、`Class`這三個模型，但是暫時不定義任何內容，僅作爲占位

設計思想

應前端後端化趨勢，這裏嘗試通過story描述功能對所有動作進行領域劃分

但是認爲在此處使用領域思想不夠合適，因爲各個部分雖然在代碼上可以視作獨立的塊，但是在邏輯上關係緊密。比如：不管是課程還是班級，總得依賴於用戶才行

最後我認爲合適的方式是使用一個單例來存放必要資源，所有模塊各自代碼獨立，通過到該統一資源存放點自取所需必要內容來支持自己的運行，這種方式在當前場景下兼顧了效率和便捷性

前期工作

決定好了總體的方向后，開始著手準備前期的工作

資源管理容器

這裏的資源指的是運行過程中在內存中產生的實例，通過簡單的查閱后，沒有發現 ArkTS 提供 DI 容器，這對於 MVVM 來說是非常不利的。剛好此前也提到了有統一管理資源的需求，所以嘗試構造以下類型：

```
export default class ServiceContainHelper {
  private static instances: Map<string, any> = new Map();

  static register<T>(key: string, instance: T) {
    this.instances.set(key, instance);
  }

  static resolve<T>(key: string): T {
    return this.instances.get(key);
  }

  static contain(key: string): boolean {
    return this.instances.has(key);
  }

  static remove(key: string) {
    this.instances.delete(key)
  }
}
```

以上類型提供了一個容器，可以通過鍵值來存放任意實例，并可以手動注冊、獲取、刪除

此後的開發中，不管是依賴還是邏輯上的單例，都可以通過這個容器來進行管理

弊端也很明顯，超級容易越放越多，只適合小項目這樣不嚴謹地玩，此後需要在開發時注意這個問題

網絡請求幫助

由於所有數據需要從平臺提供的api獲取，所以首先實現一個幫助類，爲我們的請求提供支持：

```
import http from '@ohos.net.http'


// cloud class address
```

```
const BASE_ADDRESS = "http://123.60.25.147:8090"

// entity of ajax format request
class AjaxEntity{
  // path of api, don't add base url and port
  url: string

  // params in http request
  extraData: Array<ExtraDataEntity>

  // function used when request success
  success: (data: Object) => void

  // function used when request fail
  fail: (data: Object) => void


  // necessary need url and extraData
  constructor(url: string,extraData:
    Array<ExtraDataEntity>,success:(data:Object)=>void,fail:(data:Object)=>void) {
    this.url = url;
    this.extraData = extraData;
    this.success = success;
    this.fail = fail;
  }
}

// entity of extra data
// when you send a http request, should take some key-value pair as ExtraDataEntity format
// always used like {name: "key", value: value} to set a entity in other constructor
class ExtraDataEntity{
  name: string
  value: string|number|boolean

  constructor(name: string, value: string) {
    this.name = name
    this.value = value
  }
}

// http request helper
export default class HttpClientHelper{

  static send(value:AjaxEntity):Promise<void>{
    return new Promise((resolve,reject) => {
      HttpClientHelper.sendHttpMsg(value)
```

```
  })
}

// emulate send action
// return a  http response
static sendHttpMsg(value: AjaxEntity) {
  let httpClient = http.createHttp()

  httpClient.on('headersReceive', (header) => {
    console.info('header: ' + JSON.stringify(header));
  });

  httpClient.request(
    BASE_ADDRESS + value.url,
    {
      method: http.RequestMethod.POST,
      header: {
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      extraData: HttpClientHelper.changeWWWParam(value.extraData),
      expectDataType: http.HttpDataType.STRING,
      usingCache: true,
      priority: 1,
      connectTimeout: 60000,
      readTimeout: 60000,
      usingProtocol: http.HttpProtocol.HTTP1_1,
    }, (err, data) => {
      if (!err) {
        if(data.result){
          try{
            var rst:string = data.result.toString();
            value.success(JSON.parse(rst))
          }catch(error){
            console.info("error:" + JSON.stringify(error));
          }
        }          else{
          value.success(null)
        }
      } else {
        value.fail(err)
      }
    httpClient.off('headersReceive');
    httpClient.destroy();
    }
  )
}
```

```
    // convert ExtraDataEntity type to http request param
    static changeWWWParam(dataArray:Array<ExtraDataEntity>){
      var rst:string = '';

      dataArray.forEach((item)=>{
        rst+=item.name + "=" + item.value + "&"
      })

      // remove last "&"
      if(rst !== ''){
        rst = rst.substring(0, rst.length - 1)
      }

      console.info("changeExtraDataWWW:" + rst)
      return rst
    }
  }
```

ExtraDataEntity:

ExtraDataEntity 類是爲了存放置於請求中的報文參數

AjaxEntity:

AjaxEntity 類模擬了一個ajax請求體

在最初的設計中，不僅對於所有成員，還提供了請求類型、報頭等等屬性的自定義設置，但是在長時間的對 HttpClientHelper 類的調試中最終捨棄了這些內容

HttpClientHelper:

HttpClientHelper 類型是負責 http 請求的幫助類

在 HttpClientHelper 中提供了一個 send 方法，接受一個 AjaxEntity 類型參數，並提供對 sendHttpMsg 方法的包裝調用。這是因爲 sendHttpMsg 方法在執行時由於其異步性質會導致調試時很難看到執行結果，封裝后可以通過 async/await 來方便地等待執行結束后進行下一步的調試

> 一開始某人認爲可以將 ExtraDataEntity 和 AjaxEntity 獨立放在其他文件夾中進行進一步的結構管理，但是很遺憾的是會出現衆多不可名狀的錯誤，最後發現使用單文件可以一勞永逸杜絕各種問題

開始開發

測試部分省略后，接下來快速開發出模板，作爲後續所有開發的依據

model 模板

```
import IUser from "./IUser"
import UserInfo from './UserInfo'

@Observed
export default class User implements IUser{
  userUuid:string
  userInfo:UserInfo
}
```

按照下面規則編寫

- 在接口中確保只聲明最重要的內容
- 在主要 model 類中按照最直觀的邏輯設計
- 能封裝成類就不要直接使用字段，這利好後續變更

viewmodel 模板

```
import router from '@ohos.router';
import HttpClientHelper from '../../helpers/HttpClientHelper';
import ServiceContainHelper from '../../helpers/ServiceContainHelper';
import ForgetPasswordMsg from '../../misc/ForgetPasswordMsg';
import LoginMsg from '../../misc/LoginMsg';
import RegisterUserMsg from '../../misc/RegisterUserMsg';
import User from '../models/User';
import UserInfo from '../models/UserInfo';


@Observed
export default class LoginPageViewModel{

  /*
   * status flags
   * */

  // user login status, false is not login yet
  // must put instance into service container before change to true
  // must route into index page after change to true
  statusFlagUserLoginStatus: boolean = false
  public get StatusFlagUserLoginStatus(): boolean {
    return this.statusFlagUserLoginStatus;
  }
  public set StatusFlagUserLoginStatus(value:boolean) {
    ServiceContainHelper.register("user",this.user)
```

```
  ServiceContainHelper.register("userInfo",this.userInfo)
  this.statusFlagUserLoginStatus = value
  router.back()
}

/*
 * request params
 * */

// user name for user login  requestParamUserLoginFunctionParamUserName: string = ""
public get RequestParamUserLoginFunctionParamUserName(): string {
  return this.requestParamUserLoginFunctionParamUserName
}
public set RequestParamUserLoginFunctionParamUserName(value: string) {
  this.requestParamUserLoginFunctionParamUserName = value
}

// user password for user login
requestParamUserLoginFunctionParamPassword: string = ""
public get RequestParamUserLoginFunctionParamPassword(): string {
  return this.requestParamUserLoginFunctionParamPassword
}
public set RequestParamUserLoginFunctionParamPassword(value: string) {
  this.requestParamUserLoginFunctionParamPassword = value
}

// email for user register
requestParamUserRegisterFunctionParamEmail: string = ""
public get RequestParamUserRegisterFunctionParamEmail(): string {
  return this.requestParamUserRegisterFunctionParamEmail
}
public set RequestParamUserRegisterFunctionParamEmail(value: string) {
  this.requestParamUserRegisterFunctionParamEmail = value
}

// password for user register
requestParamUserRegisterFunctionParamPassword: string = ""
public get RequestParamUserRegisterFunctionParamPassword(): string {
  return this.requestParamUserRegisterFunctionParamPassword
}
public set RequestParamUserRegisterFunctionParamPassword(value: string) {
  this.requestParamUserRegisterFunctionParamPassword = value
}

// email for forget password
requestParamForgetPasswordFunctionParamEmail: string = ""
```

```
public get RequestParamForgetPasswordFunctionParamEmail(): string {
  return this.requestParamForgetPasswordFunctionParamEmail
}
public set RequestParamForgetPasswordFunctionParamEmail(value: string) {
  this.requestParamForgetPasswordFunctionParamEmail = value
}

// password for forget password
requestParamForgetPasswordFunctionParamPassword: string = ""
public get RequestParamForgetPasswordFunctionParamPassword(): string {
  return this.requestParamForgetPasswordFunctionParamPassword
}
public set RequestParamForgetPasswordFunctionParamPassword(value: string) {
  this.requestParamForgetPasswordFunctionParamPassword = value
}

/*
 * response messages
 * */

// user login request's result
// see misc/LoginMsg for detailed structure
responseMessageLoginMessage: LoginMsg = new LoginMsg()
public get ResponseMessageLoginMessage(): LoginMsg {
  return this.responseMessageLoginMessage
}
public set ResponseMessageLoginMessage(value: LoginMsg) {
  this.responseMessageLoginMessage = value
}

// user register request's result
// see misc/RegisterUserMsg for detailed structure
responseMessageRegisterMessage: RegisterUserMsg = new RegisterUserMsg()
public get ResponseMessageRegisterMessage(): RegisterUserMsg {
  return this.responseMessageRegisterMessage
}
public set ResponseMessageRegisterMessage(value: RegisterUserMsg) {
  this.responseMessageRegisterMessage = value
}

// forget password request's result
// see misc/ForgetPasswordMsg for detailed structure
responseMessageForgetPasswordMessage: ForgetPasswordMsg = new ForgetPasswordMsg()
public get ResponseMessageForgetPasswordMessage(): ForgetPasswordMsg {
  return this.responseMessageForgetPasswordMessage
}
```

```
public set ResponseMessageForgetPasswordMessage(value: ForgetPasswordMsg) {
  this.responseMessageForgetPasswordMessage = value
}

//

/*
* models
* */

user: User = new User()
userInfo: UserInfo = new UserInfo()


// login function
async UserLoginFunction(){
  await HttpClientHelper.send({
    url:"/api/user/login",
    extraData:[
      {name:"userName",value:this.RequestParamUserLoginFunctionParamUserName},
      {name:"password",value:this.RequestParamUserLoginFunctionParamPassword}
    ],
    success:(data:LoginMsg) => {
      this.ResponseMessageLoginMessage = data
    },
    fail:() => {}
  })
}

async UserRegisterFunction(){
  await HttpClientHelper.send({
    url:"/api/user/registerUserByEmail",
    extraData:[
      {name:"email",value:this.RequestParamUserRegisterFunctionParamEmail},
      {name:"password",value:this.RequestParamUserRegisterFunctionParamPassword}
    ],
    success:(data:RegisterUserMsg) => {
      this.ResponseMessageRegisterMessage = data
    },
    fail:() => {}
  })
}

async ForgetPassword(){
  await HttpClientHelper.send({
    url:"/api/user/forgetPasswdByEmail",
```

```
      extraData:[
        {name:"email",value:this.RequestParamForgetPasswordFunctionParamEmail},
        {name:"password",value:this.RequestParamForgetPasswordFunctionParamPassword}
      ],
      success:(data:RegisterUserMsg) => {
        this.ResponseMessageForgetPasswordMessage = data
      },
      fail:() => {}
    })
  }

}
```

按照下面規則編寫

- 按照功能置於不同位置，在顯眼処劃分
- 盡量添加描述性注釋，就算是很直觀的内容
- 命名盡量按照：主要涉及功能+主要涉及方法+主要涉及參數，杜絕簡寫縮寫
- 對所有成員添加訪問器，在 view 中統一通過訪問器訪問

view 模板

```
import LoginPageViewModel from '../mvvm/viewModels/LoginPageViewModel'
@Entry
@Component
struct LoginPage {
  @State
  viewModel: LoginPageViewModel = new LoginPageViewModel()

  build() {
    Column(){
      Column(){
        Text("邮箱")
        TextInput({text: this.viewModel.RequestParamUserRegisterFunctionParamEmail})
      }.justifyContent(FlexAlign.SpaceBetween).padding(10)
      Column(){
        Text("密码")
        TextInput({text: this.viewModel.RequestParamUserRegisterFunctionParamPassword})
      }.justifyContent(FlexAlign.SpaceBetween).padding(10)
      Column(){
        Button("注册账号").onClick(() => {
          this.viewModel.UserRegisterFunction()
        })
      }.justifyContent(FlexAlign.SpaceBetween).padding(10)
```

```
    }
  }}
```

按照下面規則編寫

- 盡量只使用 viewmodel 提供的數據

雖然這是一個 Page 例子，但是實際上其他頁面都是通過 tabs 組件在主頁上切換不同的 component 實現的，考慮到未登錄時既然不可執行其他動作那麼導航部分也應該是不可見的，所以成爲了一個特例

以上的代碼編寫風格大致符合目前個人開發時所習慣性遵守的，由於在該開發環境中難免需要改變一些細節，并且合作時也需要先統一風格等各種因素，於是首先在開發的第一步完成了可以涵蓋大部分開發內容的幾個類型文件的簡單示例編寫。

其他工作

除此之外，主頁的形式也大致確定了

```
import router from '@ohos.router'
import ClassPage from '../Components/ClassPage'
import CoursePage from '../Components/CoursePage'
import UserInfoPage from '../Components/UserInfoPage'
import ServiceContainHelper from '../helpers/ServiceContainHelper'
@Entry
@Component
struct Index {

  @State
  selectIndex:number = 0
  control:TabsController = new TabsController()

  @Builder
  tarBar(text:string, index:number){
    Column(){
      if(this.selectIndex === index) {
        Text(text).fontSize(20).fontColor("#f00").fontWeight(600)
      }
      else{
        Text(text).fontSize(20).fontWeight(600)
      }
    }.onClick(() => {
      this.control.changeIndex(index)
    })
  }
```

```
  build() {
    Column(){
      // logic reverse
      if(!ServiceContainHelper.contain("User")){
        Button("请先登录").onClick(() => {
          router.pushUrl({url:"./LoginPage"})
        })
      }
      else{
        Tabs({barPosition:BarPosition.End}){
          TabContent() {
            CoursePage()
          }.tabBar(this.tarBar("课程", 0))

          TabContent() {
            ClassPage()
          }.tabBar(this.tarBar("班级", 0))

          TabContent() {
            UserInfoPage()
          }.tabBar(this.tarBar("我的", 0))
        }.onChange((index) => {
          this.selectIndex = index
        })
      }
    }.width("100%").height("100%")
    .justifyContent(FlexAlign.Center)
  }
}
```

得益於 tabs 組件的便捷切換，可以先將每個模塊作爲一個 component 設計，通過切換組件模擬路由功能實現跳轉。兼顧了多人開發和全局管理

感想與總結

本日主要是嘗試將常用的開發方式運用到該項目中

在實際的操作時，遇到了較多的問題

首先是數據綁定，在使用自帶的 對象鏈接 或者 本地存儲 功能時會遇到諸如初始化等問題，最後放棄了大部分的綁定，改爲統一存放資源加手動同步的方式，雖然原始且繁瑣，但總算實現了差不多的功能

其次是網絡請求，由於默認的異步請求會導致返回包數據賦值到本地的順序晚於本地調用，導致測試時無法取出數據，但是卻顯示請求成功。當時完全沒想到異步的問題導致消磨了將近一上午的時間

最後總算是都解決了並給出了一個相對可以讓所有合作者看懂的小示例。不過已有代碼沒有經過任何可行性測試，也沒有完善資源管理和模型定義相關內容，這是接下來在模塊開發之前的必要步驟

暑期實訓 其之十二

前言

涉及的想法僅為個人觀點，甚至日後會修改也不一定，請務必批判而不是認同

接上文

今天除了編寫一些簡單的代碼，優化項目結構，最重要的內容是對於團隊協作和組織多人項目的思考

主要內容

完善 LoginViewModel

```
import router from '@ohos.router';
import HttpClientHelper from '../../helpers/HttpClientHelper';
import ServiceContainHelper from '../../helpers/ServiceContainHelper';
import ForgetPasswordMsg from '../../misc/ForgetPasswordMsg';
import LoginMsg, { LoginMsgObj } from '../../misc/LoginMsg';
import RegisterUserMsg from '../../misc/RegisterUserMsg';
import User from '../models/User';
import GetVerificationCodeMsg from '../../misc/GetVerificationCodeMsg';


@Observed
export default class LoginPageViewModel{

  /*
   * status flags
   * */

  // user login status, false is not login yet
  // must put instance into service container before change to true
  // must route into index page after change to true
  statusFlagUserLoginStatus: boolean = false
  public get StatusFlagUserLoginStatus(): boolean {
    return this.statusFlagUserLoginStatus;
  }
  public set StatusFlagUserLoginStatus(value:boolean) {
    ServiceContainHelper.register("User",this.user)
    this.statusFlagUserLoginStatus = value
    router.replaceUrl({
      url:"pages/Index",
      params:{
        status: this.StatusFlagUserLoginStatus
      }
```

```
  })
}

// user verify status
// check it before user register and forget password
statusFlagUserVerifyStatus: boolean = false
public get StatusFlagUserVerifyStatus(): boolean {
  return this.statusFlagUserVerifyStatus;
}
public set StatusFlagUserVerifyStatus(value:boolean) {
  this.statusFlagUserVerifyStatus = value
}

// slide verify status
// check it before user login
statusFlagSlideVerifyStatus: boolean = false
public get StatusFlagSlideVerifyStatus(): boolean {
  return this.statusFlagSlideVerifyStatus;
}
public set StatusFlagSlideVerifyStatus(value:boolean) {
  this.statusFlagSlideVerifyStatus = value
}

/*
 * request params
 * */

// user name for user login
requestParamUserLoginFunctionParamUserName: string = "zsjyvso8u@mozmail.com"
public get RequestParamUserLoginFunctionParamUserName(): string {
  return this.requestParamUserLoginFunctionParamUserName
}
public set RequestParamUserLoginFunctionParamUserName(value: string) {
  this.requestParamUserLoginFunctionParamUserName = value
}

// user password for user login
requestParamUserLoginFunctionParamPassword: string = "123456"
public get RequestParamUserLoginFunctionParamPassword(): string {
  return this.requestParamUserLoginFunctionParamPassword
}
public set RequestParamUserLoginFunctionParamPassword(value: string) {
  this.requestParamUserLoginFunctionParamPassword = value
}

// email for user register
```

```
requestParamUserRegisterFunctionParamEmail: string = "zsjyvso8u@mozmail.com"
public get RequestParamUserRegisterFunctionParamEmail(): string {
  return this.requestParamUserRegisterFunctionParamEmail
}
public set RequestParamUserRegisterFunctionParamEmail(value: string) {
  this.requestParamUserRegisterFunctionParamEmail = value
}

// password for user register
requestParamUserRegisterFunctionParamPassword: string = "123456"
public get RequestParamUserRegisterFunctionParamPassword(): string {
  return this.requestParamUserRegisterFunctionParamPassword
}
public set RequestParamUserRegisterFunctionParamPassword(value: string) {
  this.requestParamUserRegisterFunctionParamPassword = value
}

// email for forget password
requestParamForgetPasswordFunctionParamEmail: string = ""
public get RequestParamForgetPasswordFunctionParamEmail(): string {
  return this.requestParamForgetPasswordFunctionParamEmail
}
public set RequestParamForgetPasswordFunctionParamEmail(value: string) {
  this.requestParamForgetPasswordFunctionParamEmail = value
}

// password for forget password
requestParamForgetPasswordFunctionParamPassword: string = ""
public get RequestParamForgetPasswordFunctionParamPassword(): string {
  return this.requestParamForgetPasswordFunctionParamPassword
}
public set RequestParamForgetPasswordFunctionParamPassword(value: string) {
  this.requestParamForgetPasswordFunctionParamPassword = value
}

// email for get verification code
requestParamGetVerificationCodeFunctionParamEmail: string = ""
public get RequestParamGetVerificationCodeFunctionParamEmail(): string {
  return this.requestParamGetVerificationCodeFunctionParamEmail
}
public set RequestParamGetVerificationCodeFunctionParamEmail(value: string) {
  this.requestParamGetVerificationCodeFunctionParamEmail = value
}

/*
 * response messages
```

```
   * */

  // user login request's result
  // see misc/LoginMsg for detailed structure
  responseMessageLoginMessage: LoginMsgObj = new LoginMsgObj()
  public get ResponseMessageLoginMessage(): LoginMsgObj {
    return this.responseMessageLoginMessage
  }
  public set ResponseMessageLoginMessage(value: LoginMsgObj) {
    this.responseMessageLoginMessage = value
  }

  // user register request's result
  // see misc/RegisterUserMsg for detailed structure
  responseMessageRegisterMessage: RegisterUserMsg = new RegisterUserMsg()
  public get ResponseMessageRegisterMessage(): RegisterUserMsg {
    return this.responseMessageRegisterMessage
  }
  public set ResponseMessageRegisterMessage(value: RegisterUserMsg) {
    this.responseMessageRegisterMessage = value
  }

  // forget password request's result
  // see misc/ForgetPasswordMsg for detailed structure
  responseMessageForgetPasswordMessage: ForgetPasswordMsg = new ForgetPasswordMsg()
  public get ResponseMessageForgetPasswordMessage(): ForgetPasswordMsg {
    return this.responseMessageForgetPasswordMessage
  }
  public set ResponseMessageForgetPasswordMessage(value: ForgetPasswordMsg) {
    this.responseMessageForgetPasswordMessage = value
  }

  // get verification code request's result
  // see misc/ForgetPasswordMsg for detailed structure
  responseMessageGetVerificationCodeMessage: GetVerificationCodeMsg =
new GetVerificationCodeMsg()
  public get ResponseMessageGetVerificationCodeMessage(): GetVerificationCodeMsg {
    return this.responseMessageGetVerificationCodeMessage
  }
  public set ResponseMessageGetVerificationCodeMessage(value: GetVerificationCodeMsg) {
    this.responseMessageGetVerificationCodeMessage = value
  }

  /*
   * user input
   * */
```

```
  userInputVerificationCode: string = ""
  public get UserInputVerificationCode(): string {
    return this.userInputVerificationCode
  }
  public set UserInputVerificationCode(value: string) {
    this.userInputVerificationCode = value
  }

  /*
   * models
   * */

  user: User = new User()


  // login function
  async UserLoginFunction(){
    console.info("12138--into login")
    if(this.StatusFlagSlideVerifyStatus){
      await HttpClientHelper.send({
        url:"/api/user/login",
        extraData:[
          {name:"userName",value:this.RequestParamUserLoginFunctionParamUserName},
          {name:"password",value:this.RequestParamUserLoginFunctionParamPassword}
        ],
        success:(data:LoginMsg) => {
          this.ResponseMessageLoginMessage = data.obj
          console.info("12138--msg:"+data.msg+data.success+data.description)
          if(data.success){
            console.info("12138--
info:"+this.ResponseMessageLoginMessage.userUuid+this.ResponseMessageLoginMessage.email)
            this.user.userUuid = this.ResponseMessageLoginMessage.userUuid
            this.StatusFlagUserLoginStatus = true
          }
        },
        fail:(data:LoginMsg) => {
          console.info("12138--fail")
          console.info("12138--msg:"+data.msg+data.success+data.description)
        }
      })
    }
  }

  async UserRegisterFunction(){
    if(this.StatusFlagUserVerifyStatus){
```

```
      await HttpClientHelper.send({
        url:"/api/user/registerUserByEmail",
        extraData:[
          {name:"email",value:this.RequestParamUserRegisterFunctionParamEmail},
          {name:"password",value:this.RequestParamUserRegisterFunctionParamPassword}
        ],
        success:(data:RegisterUserMsg) => {
          this.ResponseMessageRegisterMessage = data
          console.info("12138--"+data.msg+data.success+data.description)
        },
        fail:(data:RegisterUserMsg) => {
          console.info("12138--fail")
          console.info("12138--"+data.msg+data.success+data.description)
        }
      })
    }
  }

  async ForgetPassword(){
    if(this.StatusFlagUserVerifyStatus){
      await HttpClientHelper.send({
        url:"/api/user/forgetPasswdByEmail",
        extraData:[
          {name:"email",value:this.RequestParamForgetPasswordFunctionParamEmail},
          {name:"password",value:this.RequestParamForgetPasswordFunctionParamPassword}
        ],
        success:(data:RegisterUserMsg) => {
          this.ResponseMessageForgetPasswordMessage = data
        },
        fail:() => {}
      })
    }
  }

  async GetVerifyCode(){
    await HttpClientHelper.send({
      url:"/api/user/getVerifyCode",
      extraData:[
        {name:"email",value:this.RequestParamGetVerificationCodeFunctionParamEmail}
      ],
      success:(data:GetVerificationCodeMsg) => {
        console.info("12138--get verify code success: " + data.msg + data.success +
data.description + data.code)
        this.ResponseMessageGetVerificationCodeMessage = data
      },
      fail:() => {}
```

```
    })
  }

  CheckVerificationCode(){
    if(this.UserInputVerificationCode ==
this.ResponseMessageGetVerificationCodeMessage.code){
      this.StatusFlagUserVerifyStatus = true
    }
  }
}
```

編寫 LoginViewModel

```
import HttpClientHelper from '../../helpers/HttpClientHelper'
import ServiceContainHelper from '../../helpers/ServiceContainHelper'
import GetUserInfoMsg from '../../misc/GetUserInfoMsg'
import UpdateUserInfoMsg from '../../misc/UpdateUserInfoMsg'
import User from '../models/User'
import UserInfo from '../models/UserInfo'

@Observed
export default class UserInfoPageViewModel{

  /*
   * status flags
   * */

  // update user info status
  statusFlagUpdateUserInfoStatus: boolean = false
  public get StatusFlagUpdateUserInfoStatus(): boolean {
    return this.statusFlagUpdateUserInfoStatus;
  }
  public set StatusFlagUpdateUserInfoStatus(value:boolean) {
    this.statusFlagUpdateUserInfoStatus = value
  }

  /*
   * request params
   * */

  // user uuid for user info
  requestParamUserInfoFunctionParamUserUuid: string = "9b991a40-47db-11ef-ab56-fa163e0d65b3"
  public get RequestParamUserInfoFunctionParamUserUuid(): string {
    return this.requestParamUserInfoFunctionParamUserUuid
  }
```

```
public set RequestParamUserInfoFunctionParamUserUuid(value: string) {
  this.requestParamUserInfoFunctionParamUserUuid = value
}

// user uuid for update user info
public get RequestParamUpdateUserInfoFunctionParamUserUuid(): string {
  return this.requestParamUserInfoFunctionParamUserUuid
}

// user name for update user info
requestParamUpdateUserInfoFunctionParamUserName: string = ""
public get RequestParamUpdateUserInfoFunctionParamUserName(): string {
  return this.requestParamUpdateUserInfoFunctionParamUserName
}
public set RequestParamUpdateUserInfoFunctionParamUserName(value: string) {
  this.requestParamUpdateUserInfoFunctionParamUserName = value
}

// email for update user info
requestParamUpdateUserInfoFunctionParamEmail: string = ""
public get RequestParamUpdateUserInfoFunctionParamEmail(): string {
  return this.requestParamUpdateUserInfoFunctionParamEmail
}
public set RequestParamUpdateUserInfoFunctionParamEmail(value: string) {
  this.requestParamUpdateUserInfoFunctionParamEmail = value
}

// mobile for update user info
requestParamUpdateUserInfoFunctionParamMobile: string = ""
public get RequestParamUpdateUserInfoFunctionParamMobile(): string {
  return this.requestParamUpdateUserInfoFunctionParamMobile
}
public set RequestParamUpdateUserInfoFunctionParamMobile(value: string) {
  this.requestParamUpdateUserInfoFunctionParamMobile = value
}

// nice name for update user info
requestParamUpdateUserInfoFunctionParamNiceName: string = ""
public get RequestParamUpdateUserInfoFunctionParamNiceName(): string {
  return this.requestParamUpdateUserInfoFunctionParamNiceName
}
public set RequestParamUpdateUserInfoFunctionParamNiceName(value: string) {
  this.requestParamUpdateUserInfoFunctionParamNiceName = value
}

// photo for update user info
```

```
requestParamUpdateUserInfoFunctionParamPhoto: string = ""
public get RequestParamUpdateUserInfoFunctionParamPhoto(): string {
  return this.requestParamUpdateUserInfoFunctionParamPhoto
}
public set RequestParamUpdateUserInfoFunctionParamPhoto(value: string) {
  this.requestParamUpdateUserInfoFunctionParamPhoto = value
}

// dvcUuid for update user info
requestParamUpdateUserInfoFunctionParamDvcUuid: string = ""
public get RequestParamUpdateUserInfoFunctionParamDvcUuid(): string {
  return this.requestParamUpdateUserInfoFunctionParamDvcUuid
}
public set RequestParamUpdateUserInfoFunctionParamDvcUuid(value: string) {
  this.requestParamUpdateUserInfoFunctionParamDvcUuid = value
}

/*
 * response messages
 * */

// get user info request's result
// see misc/GetUserInfoMsg for detailed structure
responseMessageGetUserInfoMessage: GetUserInfoMsg = new GetUserInfoMsg()
public get ResponseMessageGetUserInfoMessage(): GetUserInfoMsg {
  return this.responseMessageGetUserInfoMessage
}
public set ResponseMessageGetUserInfoMessage(value: GetUserInfoMsg) {
  this.responseMessageGetUserInfoMessage = value

  this.userInfo.userName = value.userName
  this.userInfo.email = value.email
  this.userInfo.mobile = value.mobile
  this.userInfo.niceName = value.niceName
  this.userInfo.photo = value.photo
  this.userInfo.dvcUuid = value.dvcUuid

  // register instance after get
  ServiceContainHelper.register("UserInfo", this.userInfo)

  this.RequestParamUpdateUserInfoFunctionParamUserName = value.userName
  this.RequestParamUpdateUserInfoFunctionParamEmail = value.email
  this.RequestParamUpdateUserInfoFunctionParamMobile = value.mobile
  this.RequestParamUpdateUserInfoFunctionParamNiceName = value.niceName
  this.RequestParamUpdateUserInfoFunctionParamPhoto = value.photo
  this.RequestParamUpdateUserInfoFunctionParamDvcUuid = value.dvcUuid
```

```
}

// update user info request's result
// see misc/UpdateUserInfoMsg for detailed structure
responseMessageUpdateUserInfoMessage: UpdateUserInfoMsg = new UpdateUserInfoMsg()
public get ResponseMessageUpdateUserInfoMessage(): UpdateUserInfoMsg {
  return this.responseMessageUpdateUserInfoMessage
}
public set ResponseMessageUpdateUserInfoMessage(value: UpdateUserInfoMsg) {
  this.responseMessageUpdateUserInfoMessage = value
  this.StatusFlagUpdateUserInfoStatus = value.success
  if(!value.success){
    // update fail
    this.GetUserInfo()
  }
}

/*
 * models
 * */

user: User
userInfo: UserInfo


constructor() {
  // di
  this.user = ServiceContainHelper.resolve<User>("User")

  if(ServiceContainHelper.contain("UserInfo")){
    this.userInfo = ServiceContainHelper.resolve("UserInfo")
  }
  else{
    this.userInfo = new UserInfo()
  }

  this.userInfo.userUuid = this.user.userUuid
  this.RequestParamUserInfoFunctionParamUserUuid = this.user.userUuid

  this.GetUserInfo()
}


// get user info
async GetUserInfo(){
  await HttpClientHelper.send({
```

```
      url:"/api/user/queryUserInfoById",
      extraData:[
        {name:"userUuid",value:this.RequestParamUserInfoFunctionParamUserUuid}
      ],
      success:(data:GetUserInfoMsg) => {
        console.info("12138--get verify code success: " + data.email + data.mobile +
data.niceName + data.userName + data.dvcUuid)
        // 验证
        this.ResponseMessageGetUserInfoMessage = data
      },
      fail:() => {}
    })
  }

  // update user info
  async UpdateUserInfo(){
    await HttpClientHelper.send({
      url:"/api/user/updateUserInfo",
      extraData:[
        {name:"userUuid",value:this.RequestParamUpdateUserInfoFunctionParamUserUuid},
        {name:"userName",value:this.RequestParamUpdateUserInfoFunctionParamUserName},
        {name:"email",value:this.RequestParamUpdateUserInfoFunctionParamEmail},
        {name:"mobile",value:this.RequestParamUpdateUserInfoFunctionParamMobile},
        {name:"niceName",value:this.RequestParamUpdateUserInfoFunctionParamNiceName},
        {name:"photo",value:this.RequestParamUpdateUserInfoFunctionParamPhoto},
        {name:"dvcUuid",value:this.RequestParamUpdateUserInfoFunctionParamDvcUuid},
      ],
      success:(data:UpdateUserInfoMsg) => {
        console.info("12138--UpdateUserInfo success: " + data.success + data.msg +
data.description)
        // 验证
        this.ResponseMessageUpdateUserInfoMessage = data
      },
      fail:() => {}
        })
  }
}
```

感想與總結

直到現在，我才終於意識到這次實訓應該帶來的影響究竟是什麼

如果只是爲了學會一門語言，或者進步自己的編碼水平，那完全可以在任何項目上完成。被選爲組長后，我依然將目光放在編碼者的角度，思考如何將項目完成的優美且合理。但是這並不是一個處於領導和管理

地位的角色應該考慮的事情，管理者的目光始終應該放在項目本身而不是具體實現上，這點直到現在我也不太懂

因此，由於前期項目任務沒有分配好，導致我個人的想法實際上只有自己才清楚，并沒有準確傳達到每一個成員那兒。事實上，僅僅是第二天，就出現了各自爲政，閉門造車的情況。此外，認知上的不平衡也爲合作添加了許多難度，我希望重視結構，但是組内也有追求快速實現的想法，甚至還出現了完全不知道 api 文檔是什麽的個例

通過第一天暴露的問題，我意識到對於一個 team 來説，影響力度大的不是個人能力而是合作的能力。如何將不同的能力、不同的想法和不同的風格組織起來，是一件不易的事情，而在個人開發時不會涉及這方面的問題

於是我學到的第一節課就是，在團隊作業中，要學會的是妥協

我當然覺得自己的設計是最完美的，我有結構，我有層次，我甚至盡可能將其他高級後端語言的特性用這個語言去復現。但是對於其他人來説，在他們的考慮方向下，他們的設計也是最完美的。作爲一個組織者，必須要發揮所有人的能力，不可能一個人逞英雄主義（實際上一開始是打算這麽幹的，但是由於客觀因素被打斷了），這時候就必須對成員妥協，**與其將一個任務分配給能將這個任務做的最好的人，不如將這個任務分配給能做的最好的任務是它的那個人**

得出這種想法后，我考慮重新對項目任務重新分配，於是乎，我現在只需要著手于編寫我的 viewmodel 而不用考慮任何前端 ui 的展示邏輯，但它只是徒有虚名，實際上已經淪爲一個數據提供者了，畢竟開發人員永遠不知道其他人會怎麽去使用。對於個人來説，這是不幸的，但是對於整個項目而言，這樣的妥協卻是繼續推動進度的必要

那麽我應該對於這次實訓抱有何種期待呢？就盡可能體驗完全不同角度下的目光看到的項目該是什麽樣子吧