

先去Asset Store import StrangeIoC 然后创建四个文件夹保存在Script下分别是Model,View,Command,Service

创建新的空的gameobject（ContextView），挂在一个脚本Demo1ContextView 使其继承ContextView（在StrangeIOC里有），这是用来启动MVCSContext的

再创建一个Demo1Context的脚本继承自MVCS，不能使用空的构造方法

对于Demo1ContextView：创建一个Demo1Context对象，用来启动UI框架

对于Demo1Context：利用mapBindings进行绑定映射

绑定分为四类（model service command mediator）

在command文件下创建脚本StartCommand继承自Command，添加Execute()方法

一个命令和一个枚举类型进行绑定，出发绑定就是触发该枚举类型的标识

Eg：

commandBinder.Bind(ContextEvent.START).To<StartCommand>().Once();

在model文件下创建一个ScoreModel类（数据模型，只需要承载数据），不需要继承任何类，创建一个方法来获取分数

创建service，分两步

1. 创建service的接口（可多次实现，方便更换实现）

在service文件下创建IScoreService接口（类型为interface），不需要继承任何类

定义接口方法void RequestScore(string url);//请求分数和void OnReceiveScore();//收到服务器端的分数 ，UpdateScore（string url，int score）//更新分数

1. 创建实现这个接口的类ScoreService，实现接口

创建一个cube挂上CubeView继承自View，定义Init方法做初始化，提供一个修改分数（例子）的方法，在其他地方调用

在View下创建一个CubeMediator脚本继承自Mediator，实现两个方法OnRegister和OnRemove，对CubeView的注入进行调用eg：[Inject] public CubeView cubeView {get; set;}

在Demo1Context下把CubeView和CubeMediator绑定

在command下创建一个RequestScoreCommand脚本

**通过dispatcher来进行模块与模块之间的绑定和触发，注入Dispatcher**

创建一个类，枚举类型的，用于保存命令触发的事件，并且在Demo1Context进行绑定

Eg: commandBinder.Bind(Demo1CommandEvent.RequestScore).To<RequestScoreCommand>();

在mediator发起请求：在CubeMediator下的OnRegiter下通过dispatcher发起请求

Eg:dispatcher.Dispatch(Demo1CommandEvent.RequestScore);因为绑定了Command所以会触发它的Execute方法然后在调用ScoreService的方法，ScoreService和IScoreService绑定了回调用其中的Execute方法，从而发起命令请求分数

要想引用service，需要注入绑定，在所需引用Service的那个脚本注入，在context下进行绑定

Eg:injectionBinder.Bind<IScoreService>().To<ScoreService>().ToSingleton();//表示只在工程生成一次

（调用谁就要赋予Event事件，并且绑定）

下面是回调分数请求命令

在RequestScoreCommand下创建一个OnComplete方法

在IScore Service定义一个接口IEventDispatcher dispatcher { get; set; }//这是为了引用scoreService

在RequestScoreCommand的execute方法调用

scoreService.dispatcher.AddListener(Demo1ServiceEvent.RequestScore,OnComolete);

执行这句就是绑定枚举类型Demo1ServiceEvent.RequestScore和ScoreService，就是有请求分数就回调OnComplete

这需要之前定义一个枚举类型作为key，value为触发的函数（OnComplete），在service下创建一个Demo1ServiceEvent枚举类（用来表明事件）

在ScoreService下的OnRegister下

dispatcher.Dispatch(Demo1ServiceEvent.RequestScore,score);

为了让CubeMediator得到分数，那么创建一个事件的枚举类型，在CubeMediator通过监听得到分数：dispatcher.AddListener(Demo1MediatorEvent.ScoreChange, OnScoreChaged);//当ScoreChange被调用就会调用OnScoreChanged方法

在OnComplete方法：dispatcher.Dispatch(Demo1MediatorEvent.ScoreChange, evt.data);//这个会调用ScoreChange并把参数传过去，在OnScoreChanged：cubeView.UpdateScore((int)evt.data);//更新分数

利用command去更新model来保存数据（分数）

在Context下进行自身绑定

在RequestScoreCommand：public ScoreModel scoreModel { get; set; }

在OnComplete下保存分数 scoreModel.Score = (int)evt.data;

在CubeView下要调用CubeMediator，通过Dispatcher：

public IEventDispatcher dispatcher { get; set; }

通过这个dispatcher来绑定CubeMediator和加分方法OnClickDown

也就是在CubeMediator下：

cubeView.dispatcher.AddListener(Demo1MediatorEvent.ClickDown, OnClickDown);

下面这个就是触发dispatcher.Dispatch(Demo1MediatorEvent.ClickDown);

在OnClickDown方法要更新分数到Service所以在Command下创建一个UpdateScore Command的脚本并且重写execute方法，记得在Context下的Command的去绑定：

在OnClickDown下

dispatcher.Dispatch(Demo1CommandEvent.UpdateScore);//发出更新分数的命令

因为绑定了命令，所以会在UpdateScoreCommand执行Execute方法更新分数

在UpdateScoreCommand中的Execute方法

dispatcher.Dispatch(Demo1MediatorEvent.ScoreChange, scoreModel.Score);可以更新CubeMediator的分数

这会在CubeMediator的ScoreChanged更新分数

cubeView.UpdateScore((int)evt.data);