

HGUI 操作指南

目录

[第一章 功能介绍](#)

[第二章 环境配置和资源打包设置](#)

第三章 UI 使用

[3.1 UIElement](#)

[3.2 HCanvas](#)

[3.3 HImage](#)

[3.4 HText](#)

第四章 UI 事件

[4.1 UserAction](#)

[4.2 UserEvent](#)

[4.3 TextSelect](#)

[4.4 TextInput](#)

[4.5 GestureEvent](#)

第五章 复合 UI

[5.1 Composite](#)

[5.2 ScrollContent](#)

[5.3 ScrollX, ScrollY, GridScroll](#)

[5.4 ScrollYExtand](#)

[5.5 UIContainer](#)

[5.6 TreeView](#)

[5.7 UIDate](#)

[5.8 DataGrid](#)

[5.9 Paint](#)

[5.10 TabelControl](#)

[5.11 UIRocker 和其它简单介绍](#)

第六章 数据介绍

6.1 DataBuffer

6.2 ElementAsset

6.3 FakeStruct

6.4 FakeStructArray

6.5 GameobjectBuffer

6.6 LoopBuffer

HGUI 操作指南

6.7 SwapBuffer

6.8 Container

6.9 SpriteData

6.10 QueueBuffer

第七章 页面管理

7.1 UIBase

7.2 UIPage

7.3 UIWindow

第八章 数据通信

8.1 数据封包

8.2 KCP 封包

8.3 TCP UDP KCP Server

8.4 Remote Log

8.5 LinkTread

8.6 LZ4 的使用

第九章 线程池

9.1 线程委托

9.2 线程池原理

9.3 利用多线程 GIF 解码

第十章 UI 数据存取

10.1 TransformLoader

10.2 UIElementLoader

10.3 HGraphicsLoader

10.4 HImageLoader

10.5 HtextLoader

10.6 如何设计你的数据存取器

第十一章 UI 设计过程

11.1 UI 的事件派发过程

11.2 UI 的网格处理过程

11.3 UI 的合批过程

HGUI 操作指南

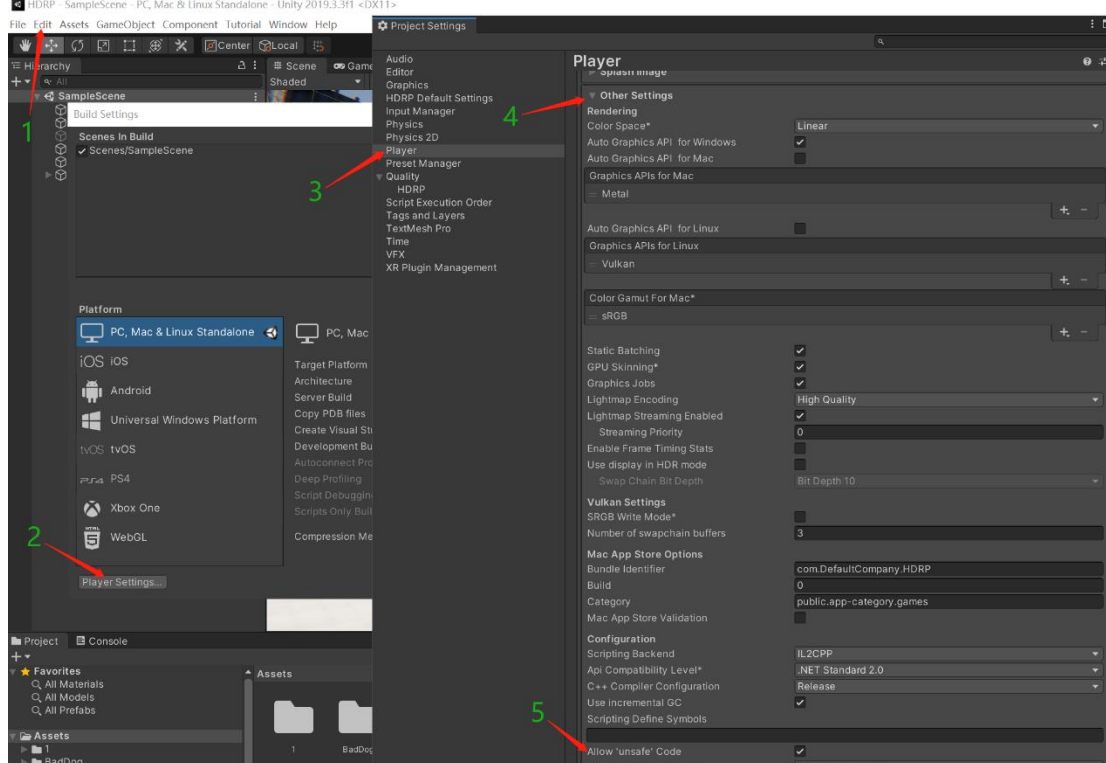
第一章 功能介绍

本插件的开发主要是用来简化 UI 的设计与操作，页面的跳转易于开发者维护，数据与页面分开管理

关于输入法，在 Win 平台和安卓平台可以很好的支持表情符输入，IOS 平台据说已在 2019.3.6f1 中修复

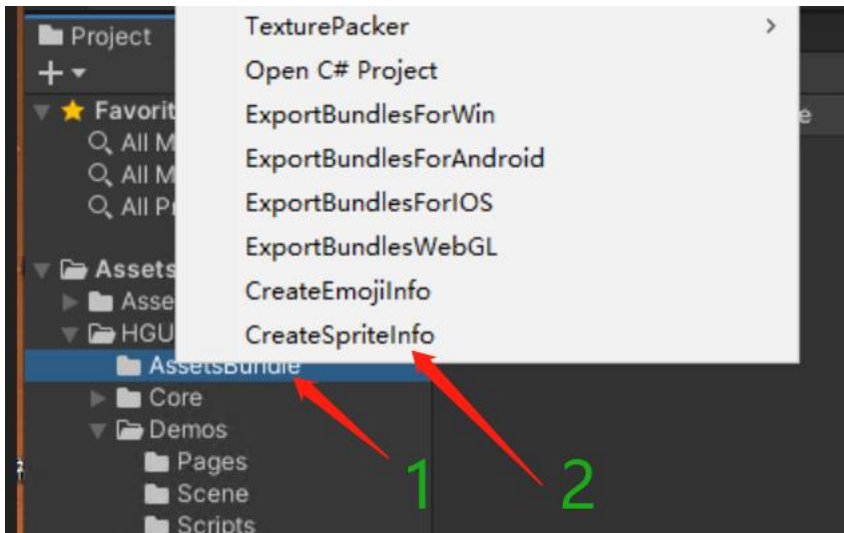
第二章 环境配置和资源打包配置

1. 关于插件导入注意事项，首先本 UI 框架多处用到了指针类的数据，导入插件包后需要开启 unsafe 模式。否则会编译报错，打开方式如下图所示：

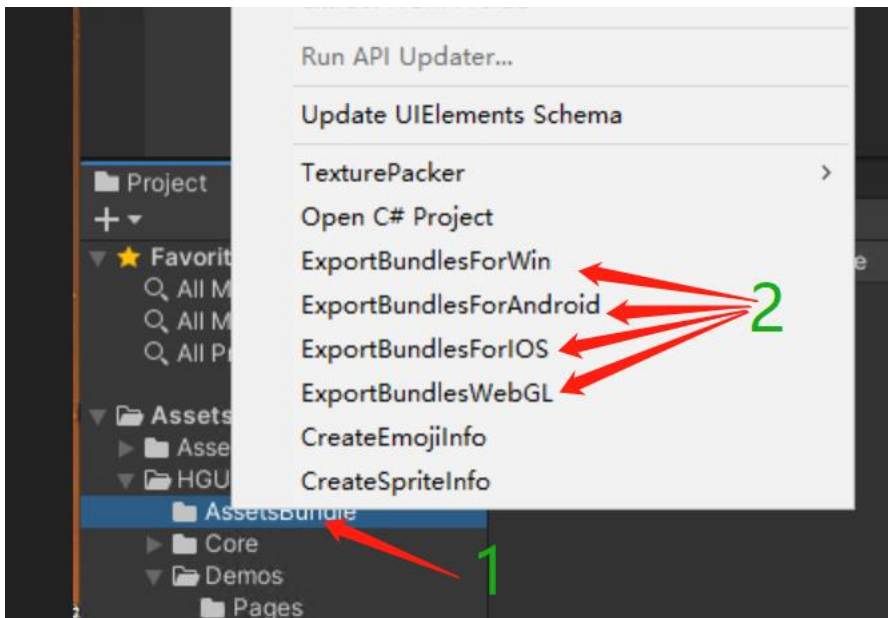


2. 将需要打包的资源最好放入 AssetsBundle 文件夹中，如果你的工程中用到了 SpriteInfo 数据，那么请先创建 SpriteInfo 数据，后缀名注意命名为 bytes，具体操作方式为，找到 Project 面板，选中你需要创建数据的文件夹，按下右键，然后单击 CreateSpriteInfo 按钮，如下图所示：

HGUI 操作指南



3. 创建你的 AssetsBundle，将你的 Assets 的文件后缀名命名为 unity3d，并将其保存在 StreamingAssets 文件夹中



4. 打包你的 UI 布局数据，UI 的布局数据可以和 AssetsBundle 分开，也可以和代码逻辑分开，打包过程如下

如果你的 UI 图片或资源数据依赖于 SpriteInfo 和 AssetsBundle 包，请先依次打包 SpriteInfo 和 AssetsBundle

然后你需要将 TestPageHelper 脚本挂到 HCanvas 对象中，也可以是你自己编写的脚本，但是需要继承 TestPageHelper 脚本. 具体示例可以打开场景 HGUI/Demos/Pages

4.1 属性设置说明

//你已经打包好的 UI 数据，可以用来克隆

```
public TextAsset bytesUI;
```

//你要保存的数据包名

```
public string AssetName = "baseUI";
```

HGUI 操作指南

//你要保存的数据路径，默认路径为Assets/AssetsBundle

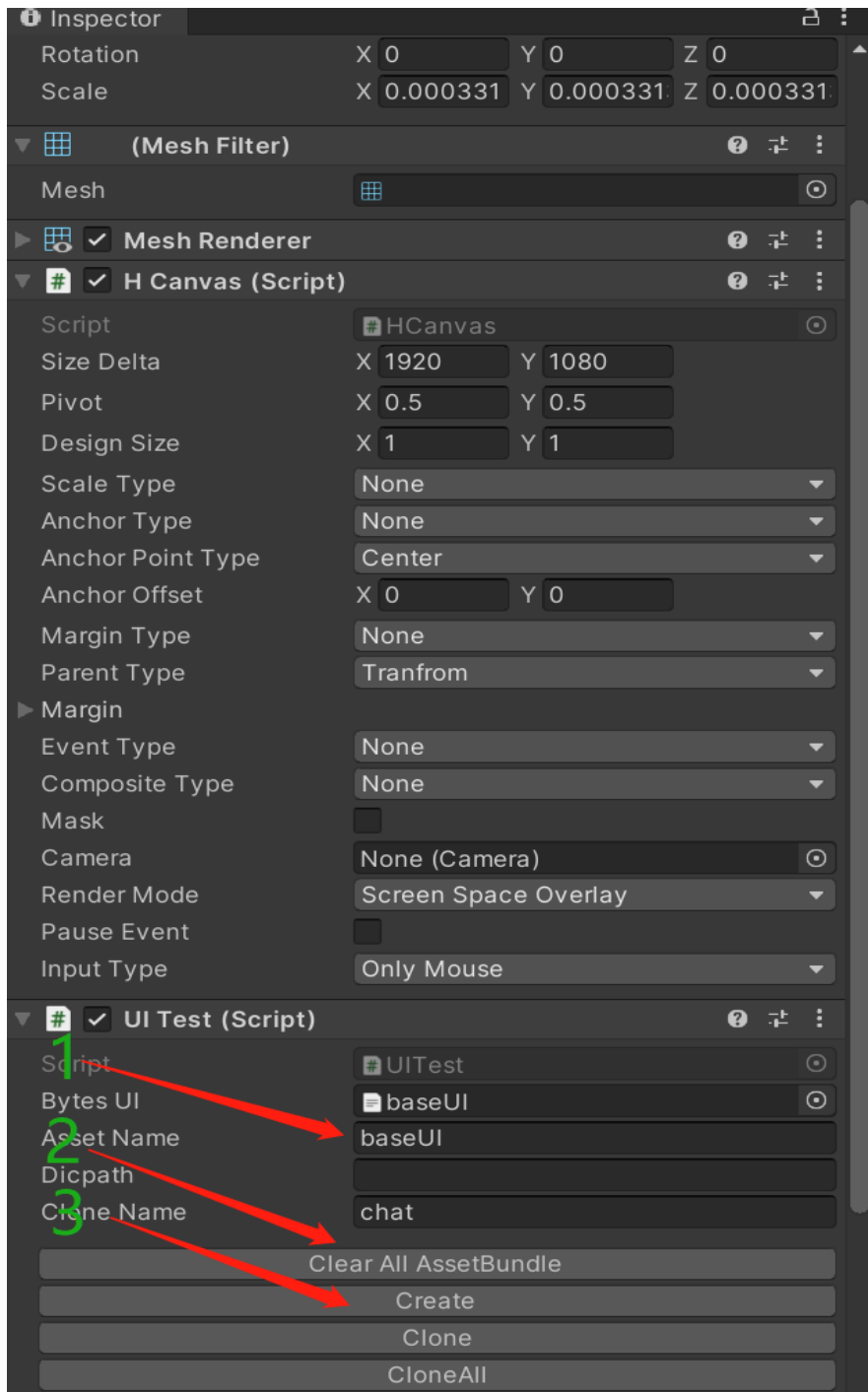
```
public string dicpath;
```

//克隆一个对象到当前HCanvas下面， 如果bytesUI数据包中存在此UI

```
public string CloneName;
```

4.2 生成你的 UI 数据

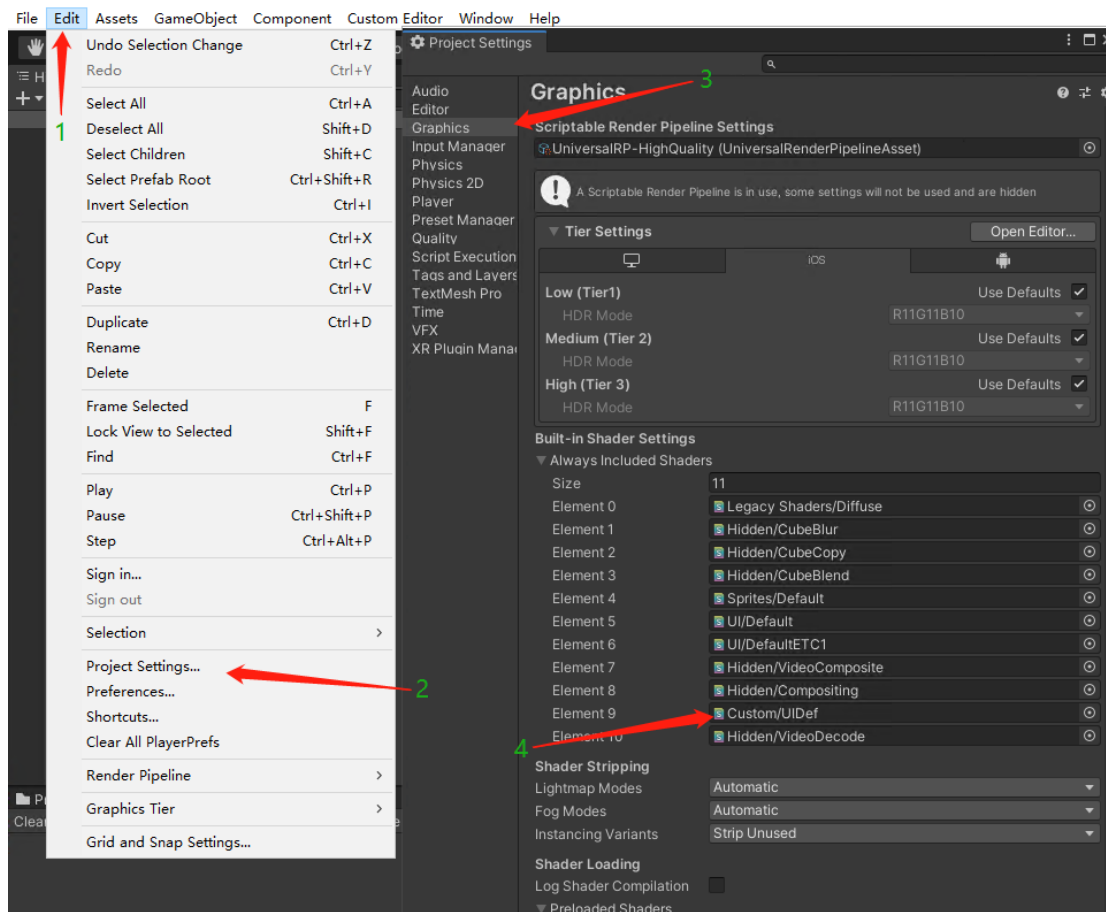
首先需要清除内存中的 AssetsBundle，防止重复加载报错，然后设置好你要存储的包名，最后点击 Create 按钮生成 UI 布局数据包。具体操作如下图所示：



5. 添加 Shader UDef

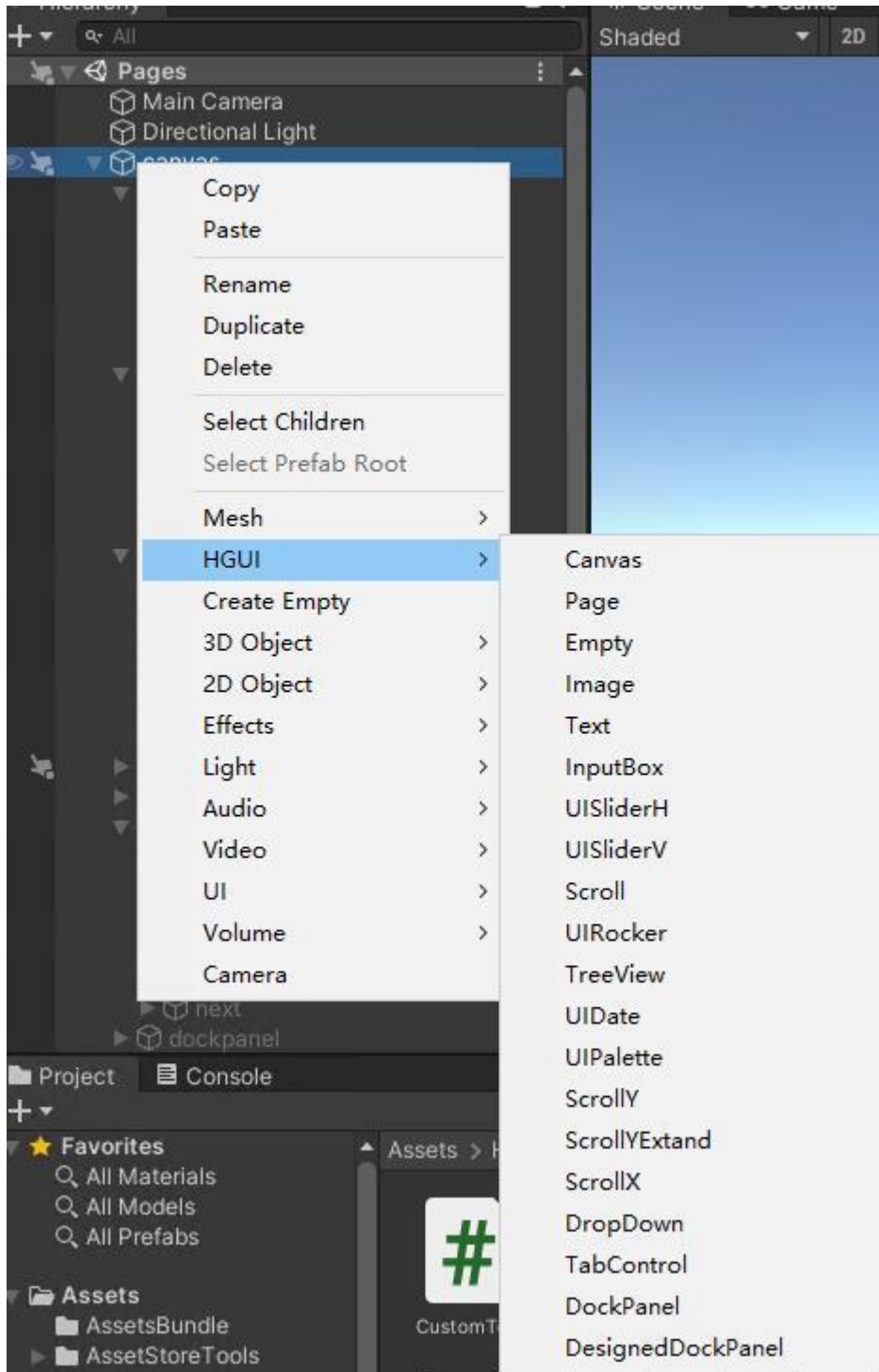
HGUI 操作指南

操作步骤如下图所示



6. UI 模型的快捷创建，如下图所示

HGUI 操作指南



HGUI 操作指南

第三章 UI 介绍

3.1 UIElement

UI的基本元素类, 其基本属性介绍



```
public ScaleType scaleType;//UI的缩放类型
public AnchorType anchorType;//UI的停靠类型
public AnchorPointType anchorPointType;//UI的停靠位置
public Vector2 anchorOffset;//UI停靠位置的偏移量
public MarginType marginType;//UI的边缘类型
public ParentType parentType;//边缘的参考对象
public Margin margin;//边缘
public EventType eventType;//事件类型，默认不注册事件
public CompositeType compositeType;//复合型UI类型，默认无
public bool Mask;//是否开启遮罩，常用于滚动框
```


HGUI 操作指南

3.2 HCanvas

3.2.1 Hcanvas 基本介绍

此类继承于 UIElement，会对其所有子物体进行网格和纹理合批处理，并且派发用户操作

Camera 设置目标相机，此对象会渲染到该摄像机的前方，默认为 MainCamera

RenderMode 设置 UI 在摄像机前的呈现模式

PauseEvent 暂停所有用户事件

InputType 用户的事件输入类型

3.2.2 Hcanvas 渲染流程

```
Update();  
UserAction.Update(); //更新用户输入事件  
Keyboard.InfoCollection(); //更新用户键盘输入  
DispatchUserAction(); //派发用户事件  
UIPage.CurrentPage.Update(UserAction.TimeSlice); //更新当前UI页面  
TextInput.Dispatch(); //派发输入法事件  
InputCaret.UpdateCaret(); //更新输入框光标  
CheckSize(); //检测当前屏幕大小是否改变  
ThreadMission.ExtcuteMain(); //执行来自分线程的任务  
LateUpdate();  
TxtCollector.Clear(); //清除文字纹理收集器的缓存  
Collection(transform, -1, 0); //收集子物体数据  
for (int i = 0; i < max; i++)  
    scripts[i].MainUpdate(); //更新UI数据  
TxtCollector.GenerateTexture(); //生成文字纹理  
UpdateMesh(); //更新子物体网格  
ClearMesh(); //清除网格缓存  
HBatch.Batch(this, PipeLine); //子物体网格合批处理  
ApplyMeshRenderrer(); //应用到MeshRenderrer  
ApplyToCamera(); //坐标转换，跟随摄像机
```

3.3 HImage

3.3.1 HImage基本属性介绍

继承于HGraphics

功能基本与UGUI的Image相同，直接填入纹理则与RawImage相同

3.4 HText

3.4.1 HText基本属性介绍

继承于HGraphics

功能基本与UGUI的Text相同，但是支持带Emoji的Unicode编码字符串

HGUI 操作指南

第四章 UI 事件

4.1 UserAction

此类主要做当前用户的鼠标或者触控事件的收集，记录当前鼠标或触控的移动速率，及焦点事件

4.2 UserEvent

用户的基础事件, 事件的注册方式为使用委托, 常用的事件包含

```
public Action<UserEvent, UserAction> PointerDown;
public Action<UserEvent, UserAction> PointerUp;
public Action<UserEvent, UserAction> Click;
public Action<UserEvent, UserAction> PointerEntry;
public Action<UserEvent, UserAction> PointerMove;
public Action<UserEvent, UserAction> PointerLeave;
public Action<UserEvent, UserAction> PointerHover;
public Action<UserEvent, UserAction> MouseWheel;
public Action<UserEvent, UserAction, Vector2> Drag;
public Action<UserEvent, UserAction, Vector2> DragEnd;
public Action<UserEvent, Vector2> Scrolling;
public Action<UserEvent> ScrollEndX;
public Action<UserEvent> ScrollEndY;
public Action<UserEvent, UserAction> LostFocus;
```

基础属性有

```
public Vector3 GlobalScale = Vector3.one; // 相对于HCanvas的全局缩放
public Vector3 GlobalPosition; // 相对于HCanvas的全局坐标
public Quaternion GlobalRotation; // 相对于HCanvas的全局旋转
public float DecayRateX = 0.998f; // 速率X的衰减率, 越接近1, 则越慢
public float DecayRateY = 0.998f; // 速率Y的衰减率, 越接近1, 则越慢
public bool forbid; // 禁用事件
public bool CutRect = false; // 开启此项, 范围外不会把事件传给子组件
public bool ForceEvent = false; // 事件不被子组件拦截
public bool Penetrate = false; // 允许事件穿透, 事件继续向下传递
public bool IsCircular = false; // 圆形按钮
public Vector2 RawPosition { get; protected set; } // 第一次触发事件的位置
public int HoverTime { get; protected set; } // 悬停时间
public long PressTime { get; internal set; } // 按压时间
public long EntryTime { get; protected set; } // 进入的时间
public long StayTime { get; protected set; } // 静止不动的时间
public bool Pressed { get; internal set; } // 按压状态
public Vector2 BoxAdjuvant; // 当物体的实际尺寸很小, 很难点中时, 是用此字段扩大
```

点击范围, 也可以是负数, 缩小范围

```
public bool AutoColor = true; // 是否自动给UI上色
public object DataContext; // 数据联系上下文
```

实际使用代码示例:

HGUI 操作指南

4.3 TextSelect

文本选择器，此类继承于UserEvent，主要用来选择和复制文本

```
public Color32 PointColor;//单击时的光标颜色
public Color32 SelectionColor;//选择文本时的光标颜色
public string GetSelectString();//获取当前选中的字符串
实际使用代码示例：
```

4.4 TextInput

文本输入框，此类继承于TextSelect

用户的基础事件，事件的注册方式为使用委托

```
public Func<TextInput, int, char, char> ValidateChar;
public Action<TextInput> OnValueChanged;
public Action<TextInput> OnSubmit;
public Action<TextInput> OnDone;
public Action<TextInput, UserAction> OnSelectChanged;
public Action<TextInput, UserAction> OnSelectEnd;
基本属性
public Color TipColor;//当输入内容为空时的提示文字
public InputType inputType;
public LineType lineType;
public ContentType contentType;
public int CharacterLimit;//输入文字的数量限制，默认为不限制
public bool Editing;//编辑状态
实际使用代码示例：
```

4.5 GestureEvent

手势输入，多点操作，此类继承于 UserEvent

基础事件

```
public Action<GestureEvent> TowFingerPressd;
public Action<GestureEvent> ThreeFingerPressd;
public Action<GestureEvent> FourFingerPressd;
public Action<GestureEvent> FiveFingerPressd;
public Action<GestureEvent> TowFingerMove;
public Action<GestureEvent> ThreeFingerMove;
public Action<GestureEvent> FourFingerMove;
public Action<GestureEvent> FiveFingerMove;
public Action<GestureEvent> TowFingerUp;
public Action<GestureEvent> ThreeFingerUp;
public Action<GestureEvent> FourFingerUp;
public Action<GestureEvent> FiveFingerUp;
public Action<GestureEvent> TowFingerClick;
public Action<GestureEvent> ThreeFingerClick;
```

HGUI 操作指南

```
public Action<GestureEvent> FourFingerClick;
public Action<GestureEvent> FiveFingerClick;
```

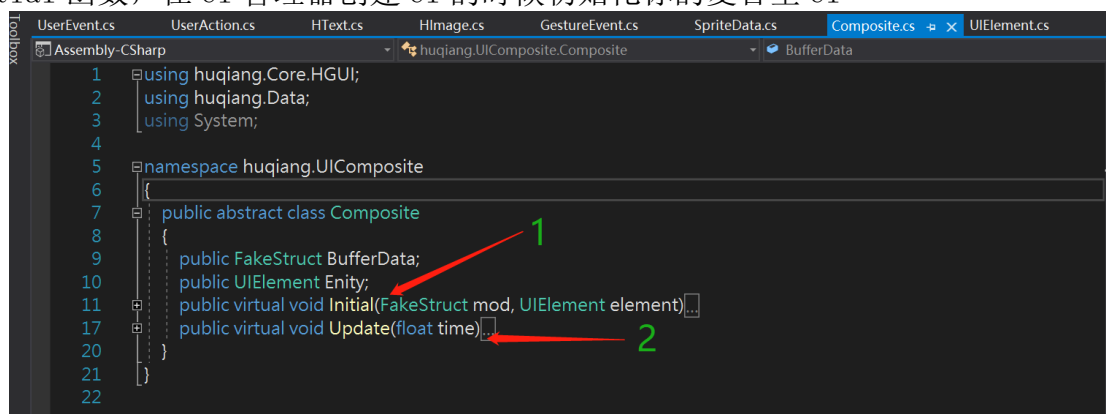
基本属性

```
public float CurScale { get; private set; } //当前双指操作的缩放
public float DeltaScale { get; private set; } //当前帧双指操作的缩放
public float DirPix { get; private set; } //双指操作移动的总计像素
public float DeltaPix { get; private set; } //双指操作当前帧移动像素
public float DirAngle { get; private set; } //双指操作总计旋转角度
public float DeltaAngle { get; private set; } //双指操作当前帧旋转角度
```

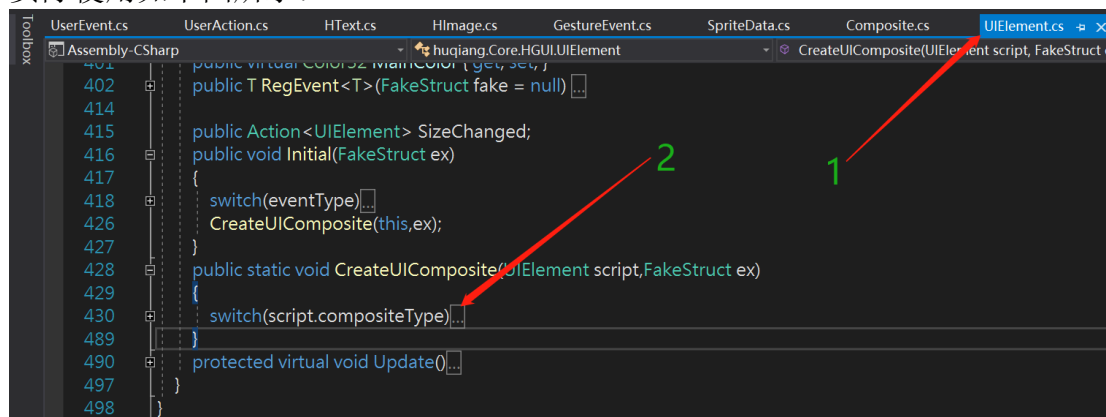
第五章 复合 UI

5.1 Composite

实际开发当中，只有图片和文字满足不了开发者要求，所有需要将其组合起来，增强其体验效果，composite 为一个抽象类，当你需要定制你的复合型 UI 时，可已继承此类，重写 Initial 函数，在 UI 管理器创建 UI 的时候初始化你的复合型 UI



重写好的你的类之后，你需要在 UIElement 添加你的初始化方法，它将在创建 UI 时被调用，实际使用如下图所示：



5.2 ScrollContent

此类继承于Composite，它是一个滚动框的内容管理器，支持的数据大致类型有三种

HGUI 操作指南

IList, Array, FakeArray

属性介绍

```
public FakeStruct ItemMod;//项目内容的模型，默认名称为Item
public Vector2 ActualSize;//根据内容的长度计算出的实际尺寸
public Vector2 ItemSize;//项目内容的 UI 尺寸
public object BindingData;//绑定的数据
public Vector2 ItemOffset;//项目内容的起始偏移位置
public virtual UISlider Slider { get; set; }//滑块条
```

委托方法

```
public Action<ScrollItem> ItemRecycle;// 当某个 ui 超出 Mask 边界，被回收时调用
```

核心使用方法

因为其回收重复利用的机制，只会显示当前可展示范围的 UI，当发生滚动时 UI 需要更新新展示的内容，因此可以使用下面的方法来更新你的每一项项目内容

```
public void SetItemUpdate<T,U>(Action<T,U, int> action,bool reflect =
true)where T:class,new()
```

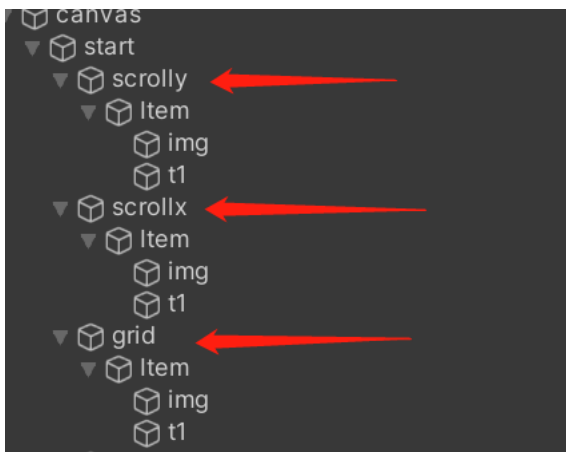
其具体使用方法在下一节介绍

5.3 ScrollX, ScrollY, GridScroll

5.3.1 demo 示例请找到 Demos 中的 Pages

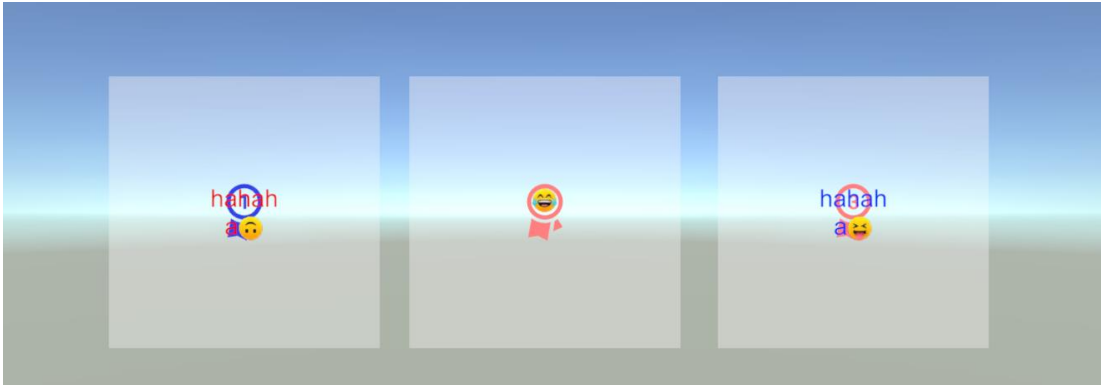
首先我们在 UI 中编辑三一样的模板，分别命名为

scrolly, scrollx, grid, 注意不要重命名，构造器是根据名称进行反射的



场景中显示如下

HGUI 操作指南



5.3.2 创建你的页面，声明你要反射 UI 的类

```
class View
{
    public ScrollY scolly;
    public ScrollX scrollx;
    public GridScroll grid;
}
View view;
```

5.3.3 声明你要反射的子项 UI 类

```
class ItemView
{
    public UserEvent img;
    public HText t1;
}
```

5.3.4 初始化并且载入你的 UI 界面

```
base.Initial(parent, dat);
view = LoadUI<View>("baseUI", "start");
```

5.3.5 准备好你要绑定的数据，数据可绑定的类型包括 IList, Array, FakeArray

```
List<string> data = new List<string>();
for (int i = 1000; i < 1200; i++)
    data.Add(i.ToString()+"😊");
```

5.3.6 设置你的子项 UI 更新方法

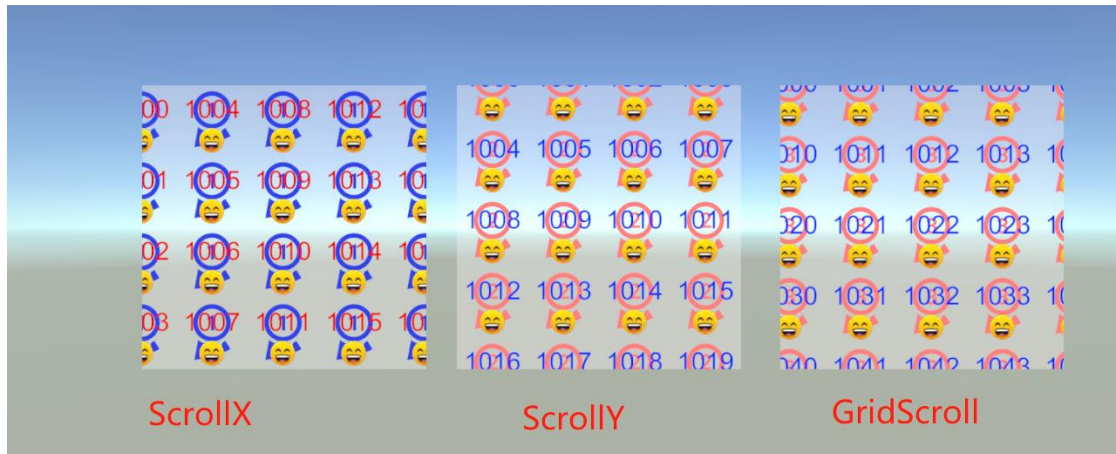
```
view.scolly.SetItemUpdate<ItemView,
string>(ItemUpdate);
view.scrollx.SetItemUpdate<ItemView,
string>(ItemUpdate);
view.grid.SetItemUpdate<ItemView, string>(ItemUpdate);
void ItemUpdate(ItemView item, string dat, int index)
{
    item.t1.Text = dat;
    item.img.DataContext = index;
    item.img.Click = ItemClick;
}
```

HGUI 操作指南

5.3.7 绑定数据，并刷新 UI

```
view.scrolly.BindingData = data;  
view.scrolly.Refresh();  
view.scrollx.BindingData = data;  
view.scrollx.Refresh();  
view.grid.BindingData = data;  
view.grid.Refresh();
```

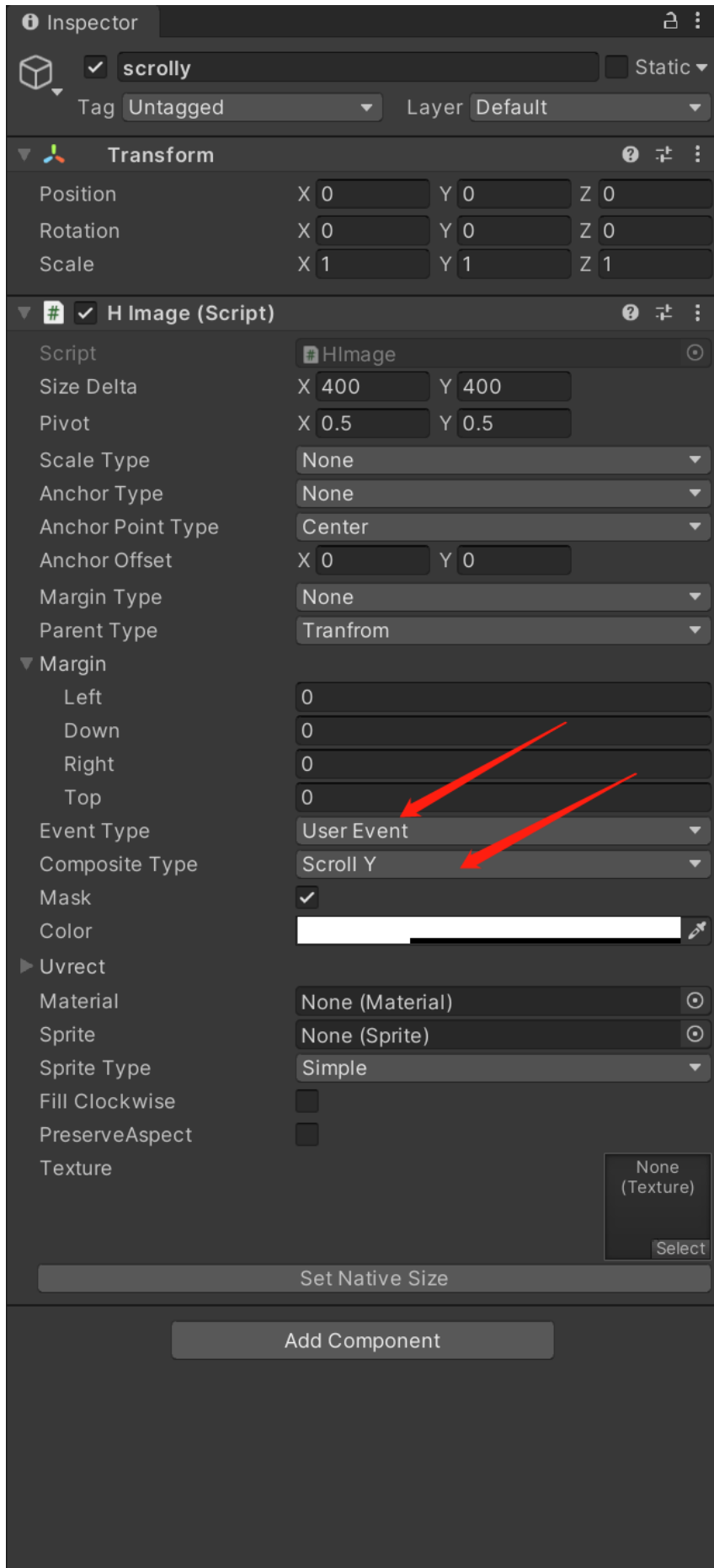
5.3.8 运行后效果如下



5.3.9 注意事项

当你需要反射你的 UI 时，最好在 Inspector 面板中设置好类型，否则 `UIInitializer` 将会使用 `Activator.CreateInstance` 创建实例，损失部分性能
设置如下图所示

HGUI 操作指南



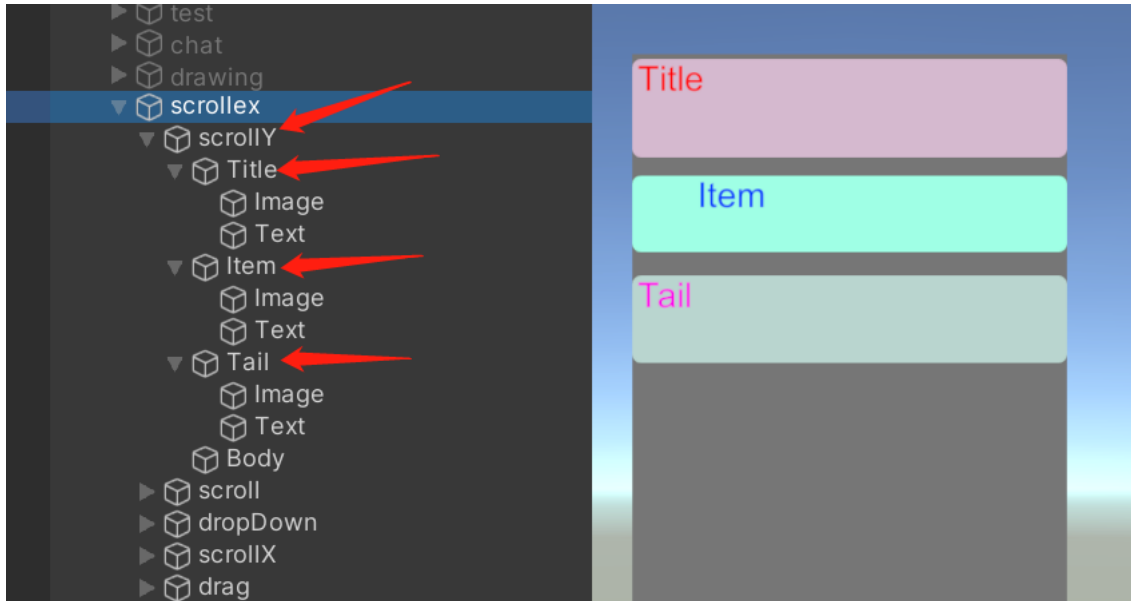
HGUI 操作指南

5.3.10 关于 `scrollx` 和 `scrolly` 与滑块条的配合使用，可以参考 Apps 中 `scrollex` 页面

5.4 ScrollYExtand

5.4.1 ScrollYExtand 主要用来开发带有标头和标尾的滚动框，它和 ScrollY 用法基本相同

首先我们来定制 UI 模板，注意 Title ,Item, Tail 这三个模板名称正确，Tail 模板可以为空



5.4.2 与普通滚动框绑定数据不同，这里绑定的数据必须

`List<ScrollYExtand.DataTemplate>`

其中

`ScrollYExtand.DataTemplate.Title` 为标头所绑定的数据

`ScrollYExtand.DataTemplate.Tail` 为标尾所绑定的数据

但是 `ScrollYExtand.DataTemplate.Data` 中数据可绑定的类型包括 `IList`, `Array`, `FakeArray`

5.4.3 下面是数据绑定实例，可以在 Demo 的 ScrollPage 中查看

```
List<ScrollYExtand.DataTemplate> datas = new
```

```
List<ScrollYExtand.DataTemplate>();
```

```
ScrollYExtand.DataTemplate tmp = new
```

```
ScrollYExtand.DataTemplate();
```

```
tmp.Title = "test1";
```

```
tmp.Tail = "over1";
```

```
List<string> list = new List<string>();
```

```
for (int i = 0; i < 22; i++)
```

```
    list.Add("tttt" + i.ToString());
```

```
tmp.Hide = true;
```

HGUI 操作指南

```
tmp.Data = list;
datas.Add(tmp);

tmp = new ScrollYExtand.DataTemplate();
tmp.Title = "test2";
tmp.Tail = "over2";
list = new List<string>();
for (int i = 0; i < 11; i++)
    list.Add("tttt" + i.ToString());
tmp.Hide = true;
tmp.Data = list;
datas.Add(tmp);
```

```
tmp = new ScrollYExtand.DataTemplate();
tmp.Title = "test3";
tmp.Tail = "over3";
list = new List<string>();
for (int i = 0; i < 7; i++)
    list.Add("tttt" + i.ToString());
tmp.Hide = true;
tmp.Data = list;
datas.Add(tmp);
```

```
ScrollYExtand scrollY = view.scrollY;
scrollY.BindingData = datas;
```

5.4.4 设置 UI 的刷新方法，并刷新 UI

```
scrollY.SetTitleUpdate<TitleItem,
    ScrollYExtand.DataTemplate>(TitleUpdate);
    scrollY.SetItemUpdate<SubItem,
string>(ItemUpdate);
scrollY.SetTailUpdate<TitleItem,
    ScrollYExtand.DataTemplate>(TailUpdate);
scrollY.Refresh();
void TitleUpdate(TitleItem title,
ScrollYExtand.DataTemplate data, int index)
{
    title.Text.Text = data.Title as string;
    title.Image.DataContext = data;
    title.Image.Click = (o, e) => {
        var dt = o.DataContext as
ScrollYExtand.DataTemplate;
        if (dt.Hide)
        {
```

HGUI 操作指南

```
view.scrollY.OpenSection(dt);//展开选中栏
if (current != dt)
{
view.scrollY.HideSection(current);//收缩当前栏
}
current = dt;
}
else
{
view.scrollY.HideSection(dt);
if (dt == current)
current = null;
}
};
}
void TailUpdate(TitleItem title,
ScrollYExtand.DataTemplate data, int index)
{
title.Text.Text = data.Tail as string;
}
void ItemUpdate(SubItem item, string data, int
index)
{
item.Text.Text = data;
}
```

5.4.5 运行后效果如下，本人没有美术功底，实际效果并不美观

HGUI 操作指南

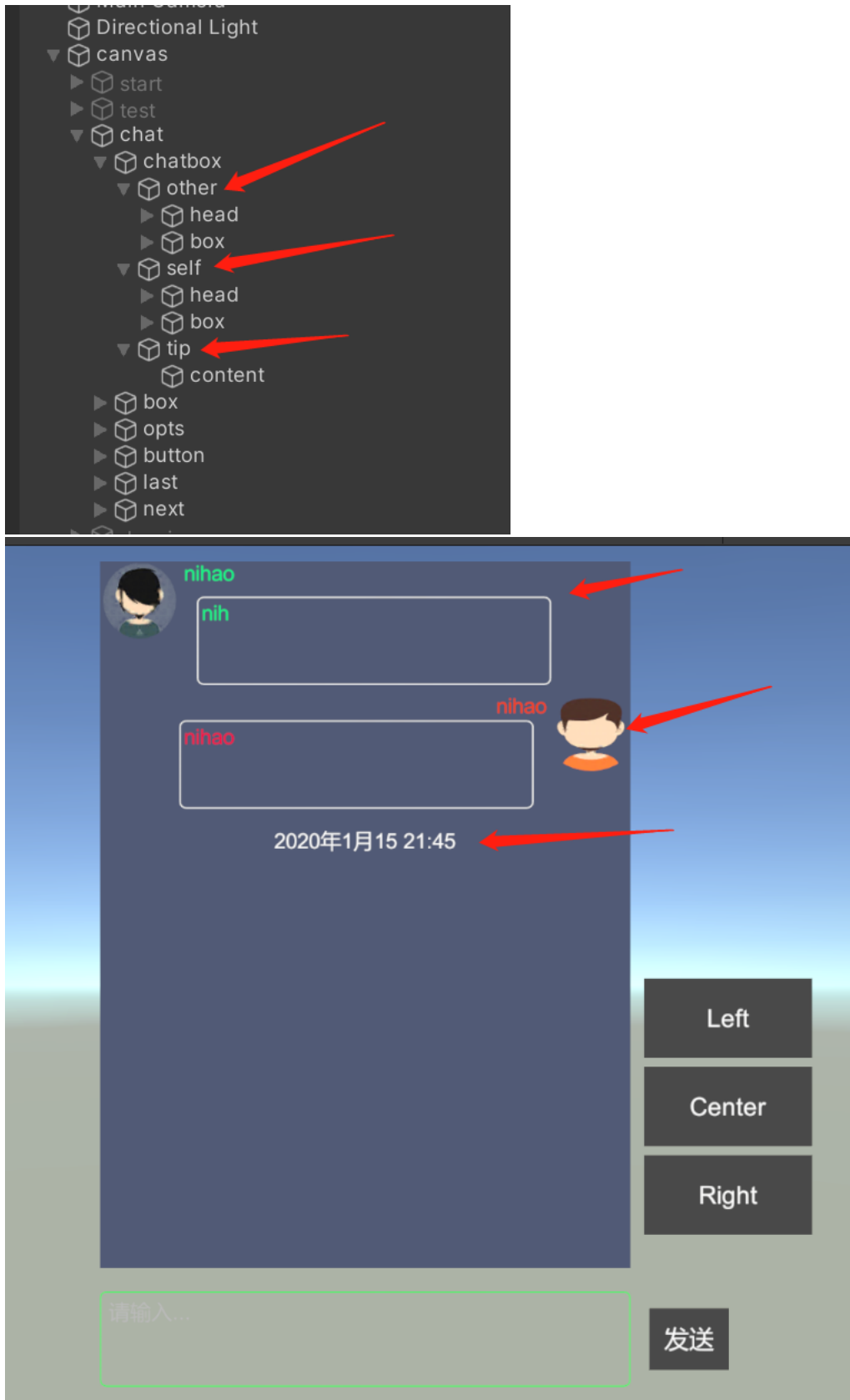


5.5 UIContainer

5.5.1 UIContainer 可以绑定 UI 模型没有限制，主要用来实现聊天框，模型的尺寸也不是固定的，文字的高度需要用户自己计算

首先第一步还是制作 UI 模板，一下示例添加了三个模板，分别是别人的消息，自己的消息和日期，如下图所示

HGUI 操作指南



5.2 如上图所示还增加了三个的按钮，分别用于发送三种消息

5.3 向 UIContainer 中注册模板

```
container = view.chatbox.composite as UIContainer;
```

HGUI 操作指南

```
other = container.RegLinker<ChatItem,
ChatData>("other");
self = container.RegLinker<ChatItem, ChatData>("self");
tip = container.RegLinker<TipItem, TipData>("tip");
5.4 设置内容尺寸计算方法，日期的内容尺寸不会变动所以不予计算
other.CalcItemHigh = GetContentSize;
self.CalcItemHigh = GetContentSize;
float GetContentSize(ChatItem chat, ChatData data)
{
    if (data.conSize == Vector2.zero)
    {
        Vector2 size = new Vector2(360, 60);
        chat.content.GetPreferredSize(ref size,
data.content);
        size.x += 8;
        size.y += 8;
        data.conSize = size;
    }
    chat.content.SizeDelta = data.conSize;
    var s = data.conSize;
    s.x += 10;
    s.y += 10;
    chat.box.SizeDelta = s;
    var ui =
chat.box.transform.parent.GetComponent<UIElement>();
    s.y += 60;
    s.x = 600;
    ui.SizeDelta = s;
    UIElement.ResizeChild(ui);
    return s.y;
}
5.5 设置模板更新内容的方法
other.ItemUpdate = ItemUpdate;
self.CalcItemHigh = GetContentSize;
tip.ItemUpdate = TipItemUpdate;
void ItemUpdate(ChatItem chat, ChatData data, int
index)
{
    chat.content.Text = data.content;
    chat.name.Text = data.name;
```

HGUI 操作指南

```
    }  
    void TipItemUpdate(TipItem tip, TipData data, int  
index)  
    {  
        tip.content.Text = data.content;  
    }  
}
```

5.6 设置单选按钮，让发送的消息匹配选中项

```
option = new OptionGroup();  
option.AddEvent(view.left.userEvent);  
option.AddEvent(view.right.userEvent);  
option.AddEvent(view.center.userEvent);  
option.SelectChanged = SelectChanged;  
option.Selecet = view.right.userEvent;  
  
void SelectChanged(OptionGroup option, UserAction  
action)  
{  
    var ue = option.LastSelect;  
    if(ue!=null)  
    {  
        var trans = ue.Context.transform;  
  
trans.GetComponentInChildren<HImage>().MainColor =  
0x5E5E5EFF.ToColor();  
  
trans.GetComponentInChildren<HText>().MainColor =  
Color.white;  
    }  
    ue = option.Selecet;  
    if(ue!=null)  
    {  
        opt = ue.Context.name;  
        var trans = ue.Context.transform;  
  
trans.GetComponentInChildren<HImage>().MainColor =  
Color.blue;  
  
trans.GetComponentInChildren<HText>().MainColor = Color.red;  
    }  
}
```

HGUI 操作指南

5.7 设置发送消息的方法，根据单选框发送输入框中的消息

```
input.OnSubmit = OnSubmit;
view.send.userEvent.Click = (o, e) => {OnSubmit(input); };
```

```
void OnSubmit(TextInput input)
{
    string str = input.InputString;
    if (str == "")
        return;
    switch (opt)
    {
        case "left":
            ChatData chat = new ChatData();
            chat.name = "江海胡";
            chat.content = str;
            other.AddAndMove(chat);
            break;
        case "center":
            TipData t = new TipData();
            str = DateTime.Now.ToString();
            t.content = str;
            tip.AddAndMove(t);
            break;
        case "right":
            chat = new ChatData();
            chat.name = "胡强";
            chat.content = str;
            self.AddAndMove(chat);
            break;
    }
    input.InputString = "";
}
```

5.8 运行后，实际效果如下图所示，输入法支持直接输入表情符

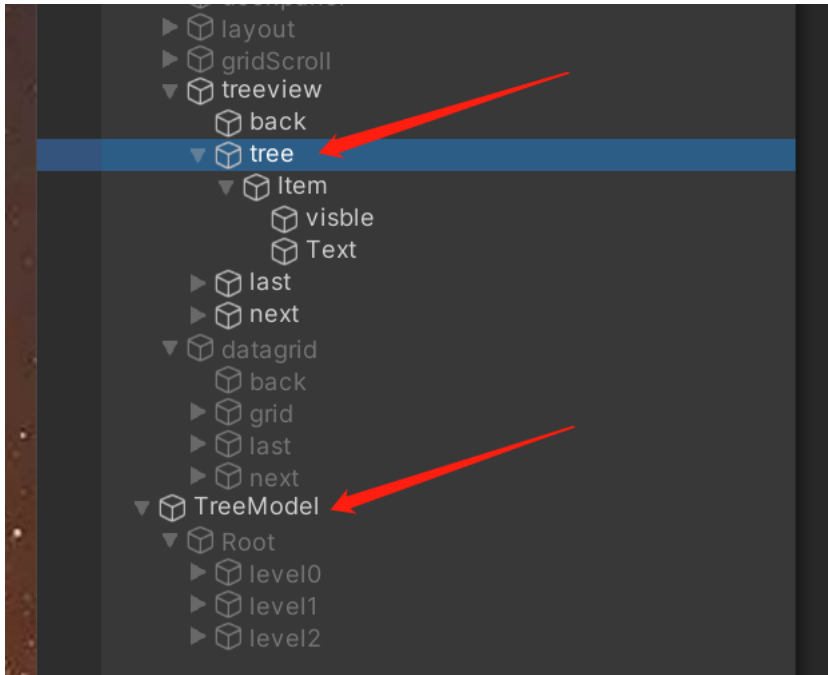
HGUI 操作指南



5.6 TreeView

5.6.1 同样的我们先在 UI 界面中定制模板，为了更好的实际效果，我在 demo 中关联了实际模型，在 Hierarchy 中的 TreeModel，用来做选中，显示和隐藏操作
模板如下图所示

HGUI 操作指南



5.6.2 收集并关联 TreeModel 中 GameObject

```
view.tree.nodes = CreateNodeChild(root);  
TreeViewNodeEx CreateNodeChild(Transform part)  
{  
    TreeViewNodeEx node = new TreeViewNodeEx();  
    node.context = part;  
}
```

HGUI 操作指南

```
node.content = part.name;
node.game = part.gameObject;
node.active = part.gameObject.activeSelf;
int c = part.childCount;
for (int i = 0; i < c; i++)
    node.Add(CreateNodeChild(part.GetChild(i)));
return node;
}
```

5.6.3 设置 TreeView 的子项更新方法

注意绑定的数据类型需要继承于 TreeViewNode

```
view.tree.SetItemUpdate<TreeViewItemEx,
TreeViewNodeEx>(TreeItemUpdate);
void TreeItemUpdate(TreeViewItemEx item,
TreeViewNodeEx node)
{
    item.Item.DataContext = item;
    item.Item.AutoColor = false;
    item.Item.Click = view.tree.DefaultItemClick;
    item.Text.Text = node.content;
    item.visble.DataContext = item;
    item.visble.Click = VisbleClick;
    item.visble.AutoColor = false;
    if (node.active)
    {
        item.visble.Context.MainColor =
0xFFFF41FFF.ToColor();
    }
    else
    {
        item.visble.Context.MainColor = Color.gray;
    }
    if (node == view.tree.SelectNode)
    {
        item.Item.Context.MainColor = new
Color32(128, 164, 255, 255);
    }
    else
    {
        item.Item.Context.MainColor = new Color32(0,
0, 0, 0);
    }
}
```

HGUI 操作指南

```
    }  
}  
//当可视按钮被单击后的效果  
void VisbleClick(UserEvent callBack, UserAction  
action)  
{  
    var item = callBack.DataContext as TreeViewItemEx;  
    var node = item.node as TreeViewNodeEx;  
    node.active = !node.active;  
    if (node.active)  
    {  
        item.visble.Context.MainColor =  
0xFFFF41FFF.ToColor();  
        node.game.SetActive(true);  
    }  
    else  
    {  
        item.visble.Context.MainColor = Color.gray;  
        node.game.SetActive(false);  
    }  
}
```

5.6.4 设置 TreeView 的选中项被更改的方法

```
view.tree.SelectChanged = SelectChanged;  
void SelectChanged(TreeView tv, TreeViewItem item)  
{  
    var items = tv.swap;  
    int len = items.Length;  
    for (int i = 0; i < len; i++)  
    {  
        var it = items[i];  
        it.Item.Context.MainColor = new Color32(0, 0,  
0, 0);  
    }  
    item.Item.Context.MainColor =  
0x4B75FFff.ToColor();  
    var node = tv.SelectNode;  
    if (node != null)  
    {  
  
    }  
}
```

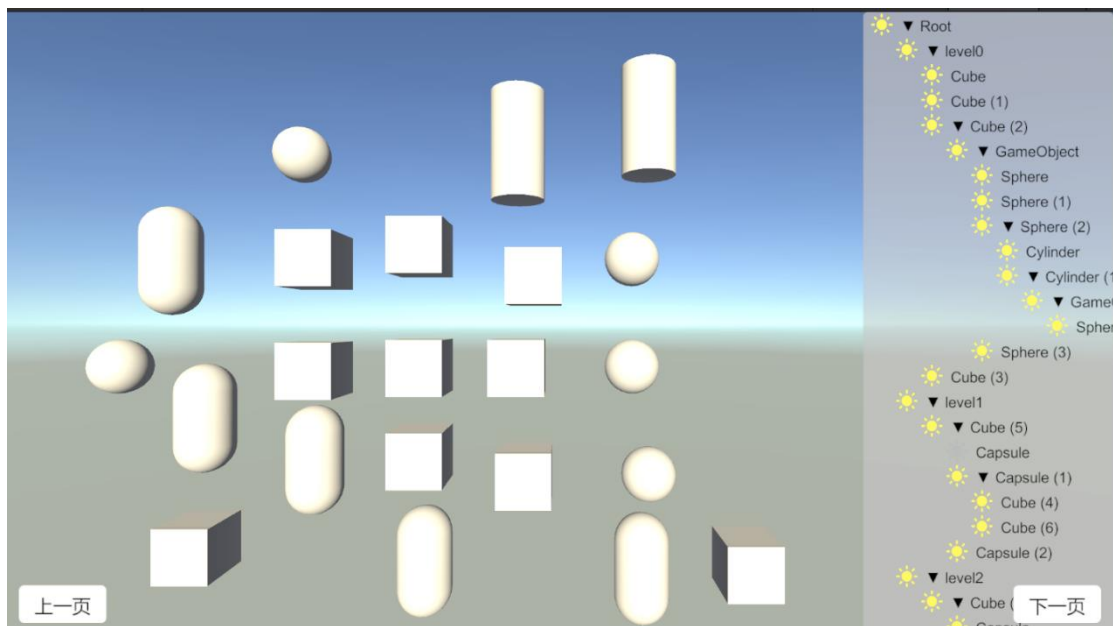
HGUI 操作指南

```
    }
}
6.5 设置单击模型区域时的射线检测
view.back.Click = RayCast;
void RayCast(UserEvent user, UserAction action)
{
    var ray =
Camera.main.ScreenPointToRay(action.Position);
    RaycastHit hit;
    if (UnityEngine.Physics.Raycast(ray, out hit,
10000))
    {
        var level =
FindLevel(hit.transform, TreeModel.transform.GetChild(0));
        int l = level.Length - 1;
        int[] buf = new int[1];
        for (int i = 0; i < l; i++)
            buf[i] = level[i + 1];
        var nod = view.tree.nodes.Find(buf);
        if (nod != null) //展开相应层级，并设置为选中
项
        {
            nod.Expand();
            view.tree.SelectNode = nod;
            view.tree.Refresh();
        }
    }
}
//层级关系查询
static int[] FindLevel(Transform transform, Transform
root)
{
    List<int> list = new List<int>();
    for (int i = 0; i < 256; i++)
    {
        list.Add(transform.GetSiblingIndex());
        if (transform == root)
        {
            break;
        }
        transform = transform.parent;
    }
}
```

HGUI 操作指南

```
}  
var array = list.ToArray();  
int len = array.Length;  
int c = len / 2;  
len--;  
for (int i = 0; i < c; i++)  
{  
    int a = array[i];  
    array[i] = array[len];  
    array[len] = a;  
    len--;  
}  
return array;  
}
```

5.6.6 实际效果展示如下



5.7 UIDate

UIDate 组件基本算是一个封闭组件，不需要定制，如果你觉得不美观，也可以自行修改

运行后直接通过字段获取结果，效果如下

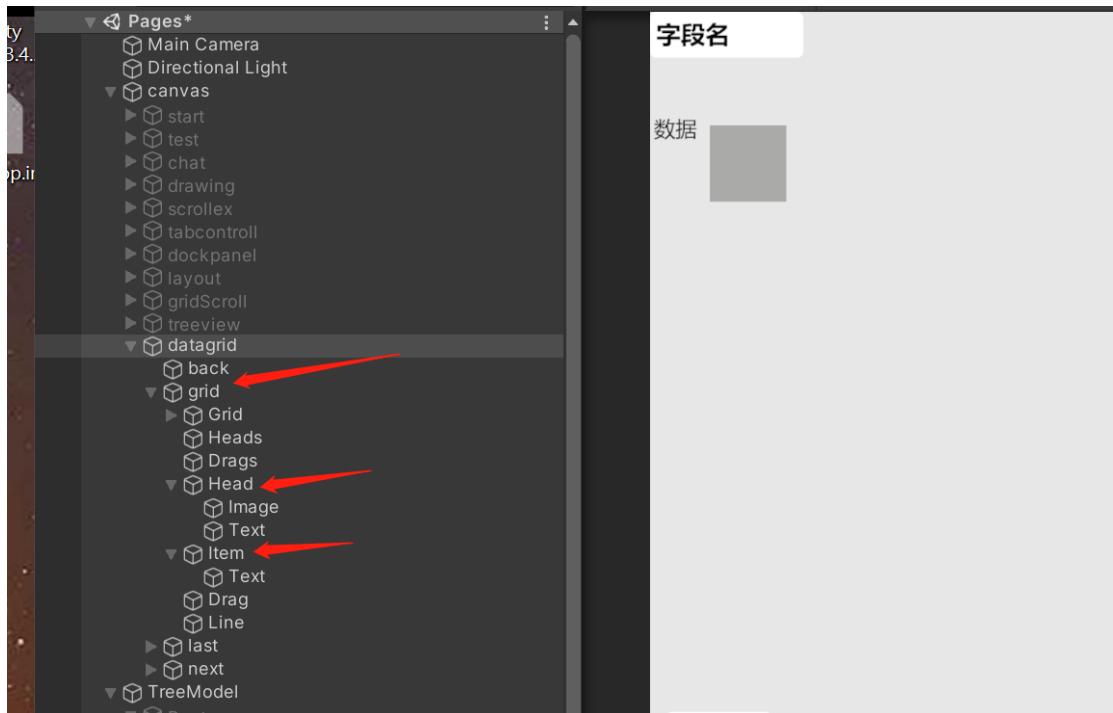
```
public int year;  
public int month = 1;  
public int day = 1;
```

HGUI 操作指南



5.8 DataGrid

5.8.1 首先还是定制 UI 模型，请不要对 Grid, Heads, Drags Head, Item, Drag, Line 的名称做更改



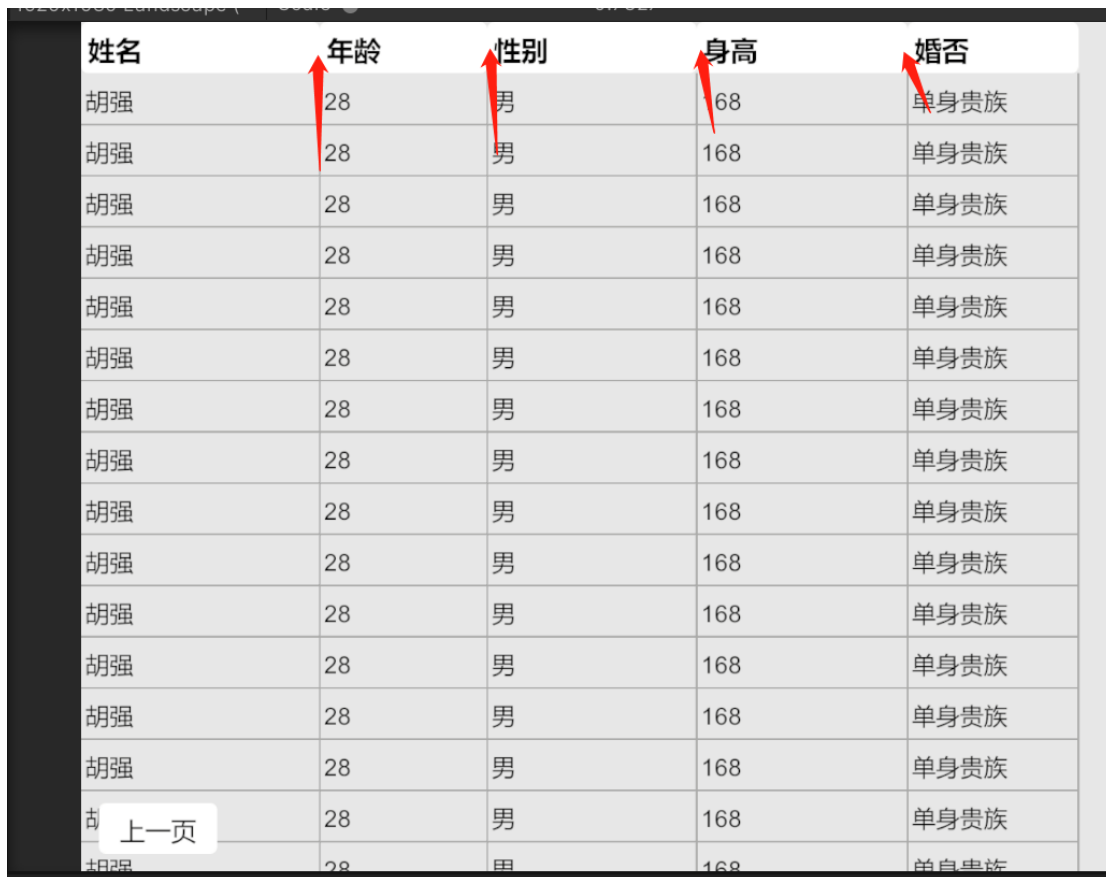
5.8.2 设置事件及绑定数据

```
DataGridColumn column = new DataGridColumn();  
column.Head = "姓名";  
view.grid.AddColumn(column);  
column = new DataGridColumn();  
column.Head = "年龄";  
view.grid.AddColumn(column);  
column = new DataGridColumn();  
column.Head = "性别";  
view.grid.AddColumn(column);  
column = new DataGridColumn();  
column.Head = "身高";
```

HGUI 操作指南

```
view.grid.AddColumn(column);
column = new DataGridViewColumn();
column.Head = "婚否";
view.grid.AddColumn(column);
for(int i=0;i<20;i++)
view.grid.AddRow(
    new DataGridViewItemContext() { Text="胡强"},
    new DataGridViewItemContext() { Text="28"},
    new DataGridViewItemContext() { Text="男"},
    new DataGridViewItemContext() { Text="168"},
    new DataGridViewItemContext() { Text="单身贵族"}
);
view.grid.Refresh();
```

5.8.3 运行后实际效果如下，并可以拖动调节每一列的宽度



姓名	年龄	性别	身高	婚否
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族
胡强	28	男	168	单身贵族

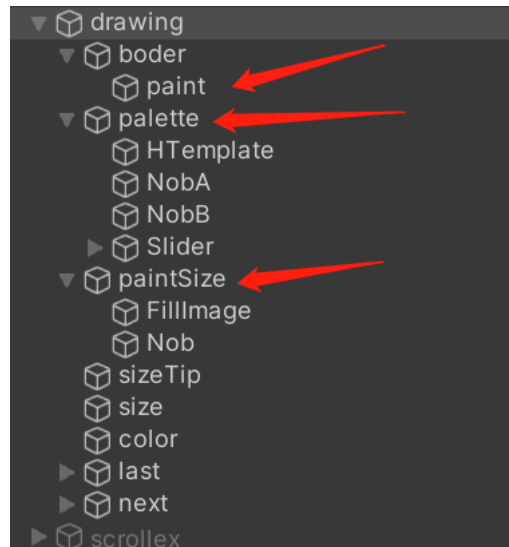
5.9 Paint

5.9.1 关于 Paint 组件的使用

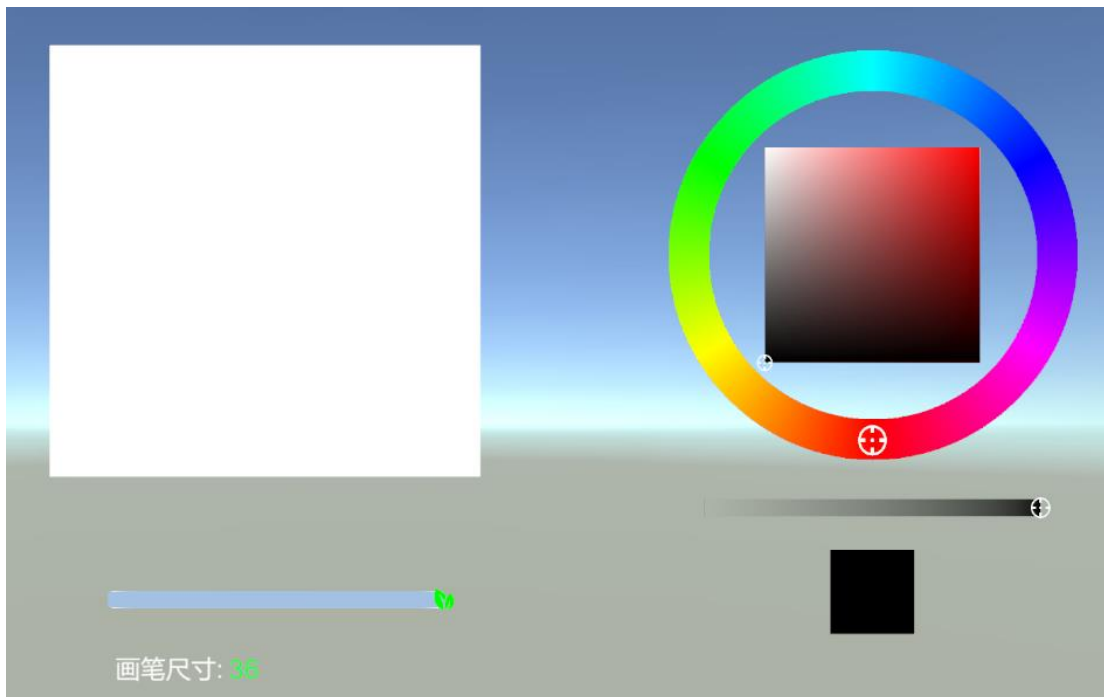
使用 Paint 需要注意的是，Paint 需要设置画笔，大致有两个参数，画笔颜色和尺寸，因此这里创建另外两个组合 UI 作为辅助，UIPalette 和 UISlider

我们来先创建 UI 模型，其中包含 Paint UIPalette 和 UISlider，如下图所示

HGUI 操作指南



Game 场景显示如下



5.9.2 书写代码，关联三个组件

//反射 UI 界面上的物体

```
class View
```

```
{
```

```
    public Paint paint;
```

```
    public UIPalette palette;
```

```
    public UISlider paintSize;
```

```
    public HText size;
```

```
    public HImage color;
```

```
}
```

view = LoadUI<View>("baseUI", "drawing");//"baseUI"创建的
bytes 文件名,"page"为创建的页面名

HGUI 操作指南

```
void InitialUI()
{
    view.paint.BrushColor = Color.black;
    view.paint.BrushSize = 36;
    view.palette.TemplateChanged=
    view.palette.ColorChanged = (o) => {
        view.color.MainColor = o.SelectColor;
        view.paint.BrushColor = o.SelectColor;
    };
    view.paintSize.OnValueChanged = (o) => {
        var v = o.Percentage * 36;
        if (v < 1)
            v = 1;
        view.size.Text = v.ToString();
        view.paint.BrushSize = v;
    };
}
```

5.9.3 实际运行效果如下，用鼠标绘制一个 hello



5.10 TabControl

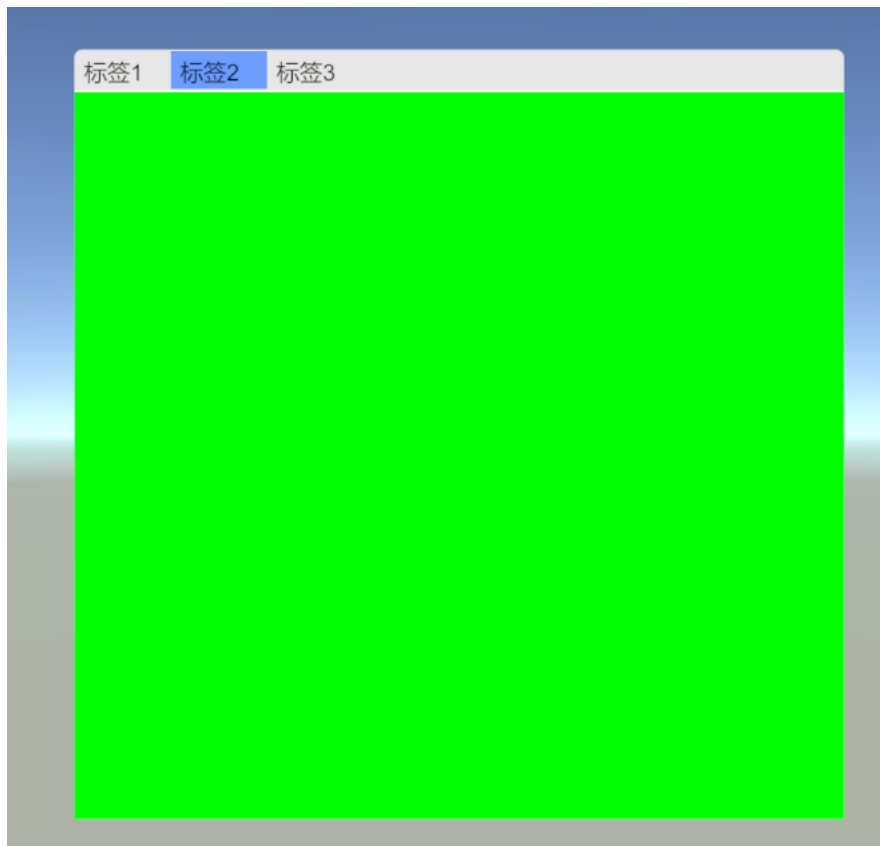
TabControl 使用比较简单，主要是添加一些标签页，UI 模板的创建就不多做介绍了，直接上代码

```
TabControl tab = view.tab;
var img =
UICreator.CreateHImage(Vector3.zero, Vector2.zero, "con1", null);
```

HGUI 操作指南

```
img.marginType = MarginType.Margin;
img.MainColor = Color.red;
tab.AddContent(img, "标签 1");
img = UICreator.CreateHImage(Vector3.zero, Vector2.zero,
"con2", null);
img.marginType = MarginType.Margin;
img.MainColor = Color.green;
tab.AddContent(img, "标签 2");
img = UICreator.CreateHImage(Vector3.zero, Vector2.zero,
"con3", null);
img.marginType = MarginType.Margin;
img.MainColor = Color.blue;
tab.AddContent(img, "标签 3");
```

运行后效果如下



5.11 UIRocker和其它简单介绍

5.11.1 关于 UIRocker 的使用

UIRocker 的创建也很简单，这里就介绍一下其使用方式

```
public Action<UIRocker> Rocking;
public float Angle { get; } //返回当前摇杆的较大
public float Slider { get; } //返回当前握柄滑动的距离范围（0-1）
public Direction direction { get; } //返回摇杆的方向
```

HGUI 操作指南

```
public float Radius{get;set;}//获取或者设置握柄的范围
```

5.11.2 关于 DesignedDockPanel 的使用

详情请见 Demo

关于 DragContent 就不多做介绍了，作用与 UGUI 中的 ScrollContent 相同