FGUI

- 参考资料

编辑器相关

包内资源

- **包内的每个资源都有一个是否导出的属性，一个包只能使用其他包设置为已导出的资源，而不设置为导出的资源是不可访问的。同时，只有设置为导出的组件才可以使用代码动态创建。**已导出的资源在资源库显示时，图标右下角有一个小红点。
- 每个包里都有一个package.xml文件，它是包的数据库文件。如果这个文件被破坏，那么包的内容将无法读取。在多人协作的情况下，如果在拉取package.xml时出现冲突，请先处理好冲突，再在编辑器内刷新包
- 包发布后可以得到一个描述文件和一张或多张纹理集，**不同平台的文件数量和打包方式有差别**

包的依赖

- FairyGUI是不处理包之间的依赖关系的，如果B包导出了一个元件B1，而A包的A1元件使用了元件B1，那么在创建A1之前，必须保证B包已经被载入，否则A1里的B1不能正确显示（但不会影响程序正常运行）。**这个载入需要由开发者手动调用，FairyGUI不会自动载入**
  - 在代码里，可以通过以下API查询包之间的依赖关系

```
var dependencies = UIPackage.dependencies;
foreach(var kv in dependencies)
{
    Debug.Log(kv["id"]); --依赖包的id
    Debug.Log(kv["name"]); --依赖包的名称
}
```

划分包的原则

- 就是不要建立交叉的引用关系，例如避免A包使用B包的资源，B包使用C包的资源这类情况
- 一般建立一个公共包，频繁用到的资源，组件放在这个包里，其它包去依赖这个公共包，而其它包之间就不要相互依赖了
- 具体的划分规则就还是视情况而定，有点像AB包的划分

分支

- **它和代码仓库中的分支概念不一样。UI分支不包含主干的资源，它只放置与主干有差别的内容**
- 同名内容会有映射关系
- 代码中使用分支

  `UIPackage.branch = "分支名称";`
- 设置分支名称可以在AddPackage之后，但应该在创建任何UI之前

常用API

GRoot

- GRoot是2D UI的根容器。当我们通过UIPackage.CreateObject创建出顶级UI界面后，将它添加到GRoot下。例如游戏的登录界面、主界面等，这类界面的特点是在游戏的底层，且比较固定
- GRoot这个根节点始终是占满屏幕的

```
--屏幕宽度
GRoot.inst.width;
--屏幕高度
GRoot.inst.height;
--调整全局声音音量，这个包括按钮声音和动效播放的声音
GRoot.inst.soundVolume = float;
--开启音效
GRoot.inst.EnabledSound();
--关闭音效
GRoot.inst.DisableSound();
--关闭弹出框
GRoot.inst.HidePopup();
--当前点击对象
GRoot.inst.touchTarget;
--物理屏幕坐标转换为逻辑屏幕坐标，还有其它转化
GRoot.inst.GlobalToLocal(screenPos);
--设置适配，跟挂载上UIContentScaler组件的效果是一样的
GRoot.inst.SetContentScaleFactor(1136, 640);
```

GObject对象基类

- 显示对象的基类
- 类似UGUI中的`GameObject`

URL地址

- 在FairyGUI中，每一个资源都有一个URL地址。选中一个资源，右键菜单，选择“复制URL”，就可以得到资源的URL地址
- 或者还有一种URL格式：`ui://包名/资源名`

```
--对象的URL地址
aObject.resourceURL;

--对象在资源库中的名称
aObject.packageItem.name;

--对象所在包的名称
aObject.packageItem.owner.name;

--根据URL获得资源名称
UIPackage.GetItemByURL(resourceURL).name;
```

常用属性

```
--保存数据
aObject.data=object;
--名称
aObject.name=string;
--源高度
aObject.sourceHeight=int;
--源宽度
aObject.sourceWidth=int;
--初始高度
aObject.initHeight=int;
--初始宽度
aObject.initWidth=int;
--设置关联
aObject.relations=Relations;
--拖拽范围
aObject.dragBounds=Rect;
--限制大小
minWidth、maxWidth、minHeight、maxHeight=int;
--设置可见
aObject.visible = bool;
--设置交互
aObject.touchable = bool;
--设置变灰
aObject.grayed = bool;
--设置alpha值
aObject.alpha=float;
--设置激活
aObject.enabled = bool;
--设置旋转，还有rotationx,rotationy
aObject.rotation=float;
--设置对象实用透视模拟
aObject.displayObject.perspective = true;
--可以设置相机距离
aObject.displayObject.focalLength = 2000;
--还有x,y,xy,position等位置属性
--设置或者获取组
aObject.group=GGroup;
--设置过滤器
aObject.filter=IFilter;

--------------------------readonly------------------------------

--对象是否在舞台上
(bool)aObject.onStage;
//root是所有元件组件的最上层节点
(GRoot)aObject.root;
--父组件
(GComponent)aObject.parent;
--可以获得这个对象身上的各个元件实例，例如按钮就是asButton,列表asList，组件asCom
(G+各元件)aObject.(as+元件);
--获取原生对象
(DisplayObject)aObject.displayObject;
```

常用方法

```
--设置混合选项
```

```
BlendModeUtils.Override(BlendMode.Add,UnityEngine.Rendering.BlendMode.XX,
UnityEngine.Rendering.BlendMode.XX);

--设置坐标
aObject.SetXY();
aObject.SetPosition();
--设置尺寸，忽略轴心的影响，即在设置了轴心的情况下，改变大小也不会同时改变坐标。
aObject.SetSize(100,100,true);
--设置缩放
aObject.SetScale(x,y);
--设置轴心
aObject.SetPivot(0.5f, 0.5f);
--设置轴心，并同时作为锚点
aObject.SetPivot(0.5f, 0.5f, true);
--将点从全局（阶段）坐标转换为局部坐标系，还有其它各种转化函数
aObject.GlobalToLocal();
--从父对象移除
aObject.RemoveFromParent();
--添加关联
aObject.AddRelation();
--移除关联
aObject.RemoveRelation();
--设置一个父Transform
aObject.SetHome();
--还有一些设置其它属性的方法
--销毁
aObject.Dispose();
--还有一些补间函数Tween
--回收资源
UIPackage.RemovePackage;
```

事件

```
--单击。冒泡事件。事件数据为InputEvent对象
GObject.onClick;
--右键单击。冒泡事件。事件数据为InputEvent对象
GObject.onRightClick;
--鼠标或手指按下。冒泡事件。事件数据为InputEvent对象
GObject.onTouchBegin;
--鼠标或手指释放。冒泡事件。事件数据为InputEvent对象
GObject.onTouchEnd;
--鼠标移入元件。事件数据为InputEvent对象
GObject.onRollOver;
--鼠标移出元件。事件数据为InputEvent对象
GObject.onRollOut;
--元件被添加到舞台。无事件数据
GObject.onAddedToStage;
--元件从舞台移出。无事件数据
GObject.onRemovedFromStage;
--元件接收到按键事件。只有获得焦点的情况下才能接收按键事件。冒泡事件。事件数据为InputEvent对
象。
GObject.onKeyDown;
--文本中的链接被点击。事件数据为href值，字符串类型
GObject.onClickLink;
--元件的位置改变。无事件数据
GObject.onPositionChanged;
--元件的大小改变。无事件数据
```

```
GObject.onSizeChanged;
--拖动是指玩家按住元件拖动然后释放的过程。注意只有设置了GObject.draggable属性的元件才会触发拖动事件。拖动过程中可以获得两个通知：开始和结束。当onDragStart中，调用EventContext.PreventDefault()可以立刻取消拖动。无事件数据。
GObject.onDragStart/GObject.onDragEnd;
```

GImage图片

```
--创建
GImage aImage = UIPackage.CreateObject("包名","图片名").asImage;
--填充属性
aImage.fillAmount = float; //0~1
--填充方式
aImage.fillMethod = FillMethod;
--设置翻转
aImage.flip=FlipType;
--将Texture2D对象赋予给GImage，必须注意GImage不管理外部对象的生命周期，不会主动销毁your_Texture2D
aImage.texture = new NTexture(your_Texture2D);
--设置颜色
aImage.color=Color;
--设置shader
aImage.shader = yourShader;
--设置材质球
aImage.material = yourMaterial;
--从资源中创建
aImage.ConstructFromResource();
```

GMovieClip动画

```
--创建
GMovieClip aMovie = UIPackage.CreateObject("包名","动画名").asMovieClip;
--切换播放和停止状态
aMovie.playing = bool;
--如果动画处于停止状态，可以设置停止在第几帧
aMovie.frame = int;
--播放倍速
aMovie.timeScale=float;
--对动画进行循环播放的设置，例如从第几帧播放到第几帧，循环播放多少次等
aMovie.SetPlaySettings(0, -1, 0, -1);
--返回播放头部
((MovieClip)aMovie.displayObject).Rewind();
--动画播放完的回调
aMovie.onPlayEnd.Add(...);
```

GGraph图形

```
--动态创建图形
GGraph holder = new GGraph();
--设置大小
holder.SetSize(100, 100);
--设置渲染层
graph.sortingOrder = int;
```

```
--绘制矩形，当然还有绘制其它图形的方法，方法名都是（Draw+图形）
holder.DrawRect(...);
//Unity平台没有矢量绘制引擎，因此是通过生成网格来模拟的。以下是Unity里运用GGraph对象的一些技巧
--传入多边形的各个顶点绘制多边形，注意点必须是顺时针方向传入！！
aGraph.shape.DrawPolygon(new Vector2[] { ...} }, new Color[] { ... };
--绘制一个渐变色的矩形
aGraph.shape.DrawRect(0, new Color[] { ... });


--其它创建图形的方法，创建一格Mesh,传入各个点
PolygonMesh trapezoid = shape.graphics.GetMeshFactory<PolygonMesh>();
        trapezoid.usePercentPositions = true;
        trapezoid.points.Clear();
        trapezoid.points.Add(new Vector2(0f, 1f));
        trapezoid.points.Add(new Vector2(0.3f, 0));
        trapezoid.points.Add(new Vector2(0.7f, 0));
        trapezoid.points.Add(new Vector2(1f, 1f));
        trapezoid.texcoords.Clear();
        trapezoid.texcoords.AddRange(VertexBuffer.NormalizedUV);
        shape.graphics.SetMeshDirty();
        shape.graphics.texture = (NTexture)UIPackage.GetItemAsset("Basics",
"chang
```

GLoader装载器

- 装载器的用途是动态载入资源

```
--创建装载器
GLoader aLoader = new GLoader();
--设置大小
aLoader.SetSize(100,100);
--使用包内资源填入装载器
aLoader.url = "ui:--包名/图片名";
--使用包外资源填入装载器
//Unity, 这里加载的是路径为Assets/Resources/demo/aimage的一个贴图
aLoader.url = “demo/aimage”;
--自动大小
aLoader.autoSize=bool;
```

自定义装载器

- 拓展装载器的功能
- 你希望从AssetBundle中获取资源，或者你需要加入缓存机制（这是有必要的，如果需要重复加载，建议做缓存），或者需要控制素材的生命期（这也是必要的，因为GLoader不会销毁外部载入的资源），那么你需要扩展GLoader

1. 首先编写你的Loader类，有两个重点方法需要重写

```
class MyGLoader  : GLoader
{
    override protected function LoadExternal()
    {
        /*
        开始外部载入，地址在url属性
```

```
        载入完成后调用OnExternalLoadSuccess
        载入失败调用OnExternalLoadFailed

        注意：如果是外部载入，在载入结束后，调用OnExternalLoadSuccess或
OnExternalLoadFailed前，
        比较严谨的做法是先检查url属性是否已经和这个载入的内容不相符。
        如果不相符，表示loader已经被修改了。
        这种情况下应该放弃调用OnExternalLoadSuccess或OnExternalLoadFailed。
        */
    }

    override protected function FreeExternal(NTexture texture)
    {
        --释放外部载入的资源
    }
}
```

2. 注册我们要使用的Loader类。注册完成后，游戏中**所有装载器**都变成由MyGLoader实例化产生

```
UIObjectFactory.SetLoaderExtension(typeof(MyGLoader));
```

- 在Unity平台中，如果在某些特殊场合需要将Texture2D对象赋予给GLoader，例如一个视频贴图

```
--必须注意GLoader不管理外部对象的生命周期，不会主动销毁your_Texture2D
aLoader.texture = new NTexture(your_Texture2D);
```

GTextField文本与GTextInput

```
--若只作为输出文本，则对象视GTextFiled
--创建文本
GTextField aTextField = new GTextField();
--设置大小
aTextField.SetSize(100,100);
--文本赋值
aTextField.text = string;
--自动大小
aTextField.autoSize=AutoSizeType;
--开启支持UBB语法
aTextField.UBBEnabled=bool;
--动态改变文本的样式（字体大小、颜色等）
--如果要设置文本的字体为位图字体，字体名称直接使用字体的url就可以了，例如‘ui:--包名/字体名’
TextFormat tf = aTextField.textFormat;
tf.color = ...;
tf.size = ...;
aTextField.textFormat = tf;

--如果文本勾选为“输入”，则运行中的实例对象为GTextInput
--主动设置焦点
aTextInput.RequestFocus();
--输入后回调
aTextInput.onChanged.Add(onChanged);
--获得焦点的通知事件
aTextInput.onFocusIn.Add(onFocusIn);
--失去焦点时的通知事件
aTextInput.onFocusOut.Add(onFocusOut);
```

```
--如果输入文本设置了单行，则当用户按回车键时会派发一个事件（注意，仅在PC上有效。在手机上的键盘按Done不属于支持范围内，引擎一般没有提供和手机键盘交互的接口）
aTextInput.onSubmit.Add(onSubmit);
--可输入的emoji表情
aTextInput.emojies = emojies;
```

- UBB语法与富文本类似，只不过使用的时`[]`而不是`<>`

文本模板

```
--文本模板
--在编辑器放置一个文本控件，文本为“我的元宝：{jin=100}金{yin=200}银”，然后勾选“使用文本模板”即可在代码中赋值
aTextField.SetVar("jin", "500").SetVar("yin", "500").FlushVars();
--批量赋值
Dictionary<string, string> values;
values["jin"] = "500";
values["yin"] = "500";
--注意，这种方式不需要再调用FlushVars
aTextField.templateVars = values;
--如果运行时要动态启用文本模板功能，不需要设置什么开关，只需要直接调用SetVar即可。
--如果运行时要动态关闭文本模板功能，只需要把templateVars设置为null即可
aTextField.templateVars = null;
```

GRichTextField富文本

- 与文本类似，只不过支持`HTML`语法

```
--创建
GRichTextField aRichTextField = new GRichTextField();
--设置emoji表情列表
aRichTextField.emojies=Dictionary<uint, Emoji>;
--侦听富文本中链接点击的方法是（这个事件是冒泡的，也就是你可以不在富文本上侦听，在它的父元件或者祖父元件上侦听都是可以的）
//Unity/Cry/MonoGame, EventContext里的data就是href值。
aRichTextField.onClickLink.Add(onClickLink);
```

GGroup组

```
--高级组可以在运行时通过代码访问。但要注意的是，组不是容器，它并没有维护一个组内元件的列表。如果你需要遍历组内的所有元件，你需要遍历容器组件的所有孩子，测试他们group属性
GGroup aGroup = gcom.GetChild("groupName").asGroup;
--组内元件个数
(int)aGroup.numChildren;
--更新组的包围
aGroup.SetBoundsChangedFlag();
```

遍历组内元件

```
int cnt = aGroup.numChildren;
for(int i=0;i<cnt;i++)
{
    if(gcom.GetChildAt(i).group==aGroup)
        Debug.Log("get result");
}
```

GComponent 组件

```
--创建
GComponent gCom = new GComponent();
--或
GComponent gCom = gcom.GetChild("groupName").asCom;
--或
GComponent gCom = UIPackage.CreateObject("包名","组件名").asCom;

----------------------属性-------------------------------
--设置组件点击不穿透。
gcom.opaque = true;
--设置子对象的渲染顺序，默认是升序的，就是从小到大渲染，大序号的在最前面
aComponent.childrenRenderOrder = ChildrenRenderOrder.Ascent;
--子对象数量
(int)gcom.numChildren;
--如果有滚动对象则访问滚动对象
(ScrollPane)gcom.scrollPane;

----------------------方法-------------------------------
--设置尺寸，下面这两个参数可以设置组件全屏
gcom.SetSize(GRoot.inst.width, GRoot.inst.height);
--添加子物体到最后
gcom.AddChild();
--添加子物体到指定位置
gcom.AddChildAt();
--移除一个子物体
gcom.RemoveChild();
--移除指定序号的子物体
gcom.RemoveChildAt();
--移除全部的子物体
gcom.RemoveChildren();
--移除不是销毁，只是移出本组件容器，销毁还是要调用对应的dispos方法
--通过名称获得元件
gcom.GetChild();
--通过索引获得元件
gcom.GetChildAt();
--获得元件在组件内的索引
gcom.GetChildIndex();
--设置元件在组件内的索引
gcom.SetChildIndex();
```

拓展类

- 其实就是将一个类绑定在一整个组件上，这样就相当于该组件有了一个属于自己的功能，就比如按钮如果是组件那他自身是有绑定一个按钮类的

```
public class MyComponent : GComponent
```

```
{
    GObject msgObj;

    --如果你有需要访问容器内容的初始化工作，必须在这个方法里，而不是在构造函数里。各个SDK的函数原型的参数可能略有差别，请以代码提示为准。在Cocos2dx/CocosCreator里，方法的名字是onConstruct，且不带参数
    override protected void ConstructFromXML(XML xml)
    {
        base.ConstructFromXML(xml);

        --在这里继续你的初始化
        msgObj = GetChild("msg");
    }

    public void ShowMessage(string msg)
    {
        msgObj.text = msg;
    }
}
```

- 然后注册你的扩展类。注意，**必须在组件构建前注册**，如果你使用的是UIPanel，那么在Start里注册是不够早的，必须在Awake里，总之，如果注册不成功，90%可能都是注册晚于创建，10%可能是URL错误，这可以通过打印URL排查

```
UIObjectFactory.SetPackageItemExtension("ui:--包名/组件A", typeof(MyComponent));
```

- 这样就为组件A绑定了一个实现类MyComponent 。以后所有组件A创建出来的对象（包括在编辑器里使用的组件A）都是MyComponent类型。然后我们就可以为MyComponent添加API，用更加面向对象的方式操作组件A

```
MyComponent gcom = (MyComponent)UIPackage.CreateObject("包名",  "组件A");
gcom.ShowMessage("Hello world");
```

- 如果组件A只是一个普通的组件，没有定义"扩展"，那么基类是GComponent，如上例所示；如果组件A的扩展是按钮，那么MyComponent的基类应该为GButton，如果扩展是进度条，那么基类应该为GProgressBar，等等。这个千万不能弄错，否则会出现报错。

ScrollPane滚动条

```
--创建
ScrollPane scrollPane =  aComponent.scrollPane;
--视口的高度宽度
scrollPane.viewWidth/viewHeight=float;
--内容的高度宽度，如果为页面模式（内容高宽度/视口高宽度=页面页数）
(float)scrollPane.contentWidth/contentHeight;
--设置滚动位置为100像素或者获得当前滚动位置
scrollPane.posX = 100;
--获得或设置当前滚动的位置，以百分比来计算，取值范围是0-1
scrollPane.percX/scrollPane.percY=float;
--调整滚动位置，使指定的元件出现在视口内
scrollPane.scrollToView();
--滚动到中间位置，带动画过程，还有其它设置滚动位置的方法
scrollPane.SetPercX(0.5f, true);
--滚动一格的距离，常用于带箭头的滚动条，或者鼠标滚轮滚一次
```

```
scrollPane.scrollStep=float;
--打开关闭页面回弹效果
scrollPane.bounceBackEffect=bool;
--向指定方向滚动 N*scrollStep
scrollPane.ScrollLeft()/ScrollRight()/ScrollUp()/ScrollDown();

--若滚动设置为页面模式
--设置或获得当前页面
scrollPane.currentPageX/currentPageY=float;
--设置当前页面
scrollPane.setCurrentPageX()/setCurrentPageY();

--当滚动面板处于拖拽滚动状态或即将进入拖拽状态时，可以调用此方法停止或禁止本次拖拽
scrollPane.CancelDragging();
--事件监听
scrollPane.onScroll.Add(onScroll);
--惯性滚动结束后回调
scrollPane.onScrollEnd.Add(onScrollEnd);
--下拉刷新回调
scrollPane.onPullDownRelease.Add(onPullDownRelease);
--上拉刷新回调
scrollPane.onPullUpRelease.Add(onPullUpRelease);
```

Controller控制器

```
Controller c1 = aComponent.GetController("c1");

--通过索引设置控制器的活动页面或者获得控制器当前的活动页面
c1.selectedIndex = int;

--如果希望改变控制器时不触发Change事件
c1.SetSelectedIndex();

--也可以使用页面的名称设置
c1.selectedPage = string;

--控制器改变时会有通知事件
c1.onChanged.Add(onChanged);

--改变控制器页面时，与之连接的属性控制可能带有缓动，如果你要获得缓动结束的通知，可以侦听
GearStop事件
aObject.OnGearStop.Add(OnGearStop);

--如果你正在做界面的初始化，可能不希望出现任何缓动
--禁止所有控制器引起的缓动
GearBase.disableAllTweenEffect = true;
c1.selectedIndex = 1;
--记住要复原
GearBase.disableAllTweenEffect = false;
```

Relations关联

- 定义两个元件之间关于位置，大小的关系

```
--获得关联
GObject.relations=Relations;
--添加关联
GObject.AddRelation();
--删除关联
GObject.RemoveRelation();
```

GLabel标签

```
--设置标签的标题或者图标，你不需要强制对象为GLabel的类型，直接用GObject提供的接口就可以
GObject obj = gcom.GetChild(name);
obj.text = string;
obj.icon = url;
--修改标题颜色可以这样
GLabel label = gcom.GetChild(name).asLabel;
lable.titleColor = Color;
```

GButton按钮

```
--设置按钮的标题或者图标，你甚至不需要强制对象为GButton的类型，直接用GObject提供的接口就可以
GObject obj = gcom.GetChild(name);
obj.text = string;
obj.icon = url;
--如果是单选或者多选按钮，下面的方法设置是否选中
button.selected = bool;
----关闭后你只能通过改变selected属性修改按钮状态，用户点击不会改变状态。
button.changeStateOnClick = bool;

--按钮全局声音设置，Unity版本要求一个AudioClip对象，如果是使用库里面的资源，那么可以使用：
UIConfig.buttonSound = (NAudioClip)UIPackage.GetItemAssetByURL("ui:--包名/声音
名");
--全局音量
UIConfig.buttonSoundVolumeScale = 1f;
--这个设置只能在创建任何UI前设置。如果要动态控制全局声音（包括动态里的音效）的开关或音量，Unity
平台可以用以下的方式
GRoot.inst.EnabledSound();
GRoot.inst.DisableSound();
--调整全局声音音量，这个包括按钮声音和动效播放的声音
GRoot.inst.soundVolume = 0.5f;

--监听普通按钮点击的方式
button.onClick.Add(onClick);
--点击事件不只是按钮有，任何支持触摸的元件都有，例如普通组件、装载器、图形等，他们的点击事件注册
方式和按钮是相同的
--模拟触发点击，只会有一个触发的表现，以及改变按钮状态，不会触发侦听按钮的点击事件。
button.FireClick(true);
--如果同时要触发点击事件，需要额外调用
button.onClick.Call();
--单选和多选按钮状态改变时有通知事件
button.onChanged.Add(onChanged);
```

GComboBox下拉列表

```
--我们可以在编辑器编辑下拉列表的项目，也可以用代码动态设置
GComboBox combo = gcom.GetChild(name).asComboBox;

//items是列表项目标题的数组。
combo.items = new string[] { "Item 1", "Item 2", ...};

//values是可选的，代表每个列表项目的value。
combo.values = new string[] { "value1", "value2", ...};

--获得当前选中项的索引或设置当前索引
combo.selectedIndex=int;

--获得当前选中项的value或设置选中项通过数据
combo.value=data;

--下拉框选择改变时有通知事件
combo.onChanged.Add(onChanged);

--点击空白处后弹出框会自动关闭，如果要获得这个关闭的通知，可以监听移出舞台的事件
combo.dropdown.onRemoveFromStage.Add(onPopupClosed);

--如果要手工关闭弹出框
GRoot.inst.HidePopup();
```

GProgressBar进度条

```
GProgressBar pb = gcom.GetChild(name).asProgress;
--设置当前值
pb.value = double;
--设置或获取最大值
pb.max = double;
--是否反向
pb.reverse=bool;
--如果想改变进度值有一个动态的过程
pb.TweenValue();
```

GSlider滑动条

```
--获得或创建
GSlider slider = gcom.GetChild(name).asSlider;
--设置/获得值
slider.value = double;
--设置/获得最大值
slider.max=double;
--滑动条进度改变时有通知事件
slider.onChanged.Add();
--当触摸结束时通知事件
slider.onGripTouchEnd();
```

GList列表

```
--获得
GList list = gcom.GetChild(name).asList;
--设置项目资源，即item的对应组件，就是item的样子
list.defaultItem=url;
--增加一个项目
list.AddChild();
--在指定的位置增加一个项目
AddChildAt();
--删除一个项目
RemoveChild();
--删除一个指定位置的项目
RemoveChildAt();
--删除一个范围内的项目，或者全部删除
RemoveChildren();
--对象池的使用，从池里创建就归还给池
--从池里取出（如果有）或者新建一个对象，添加到列表中。如果不使用参数，则使用列表的“项目资源”的设置；也可以指定一个URL，创建指定的对象
AddItemFromPool();
--从池里取出（如果有）或者新建一个对象
GetFromPool();
--将对象返回池里
ReturnToPool();
--删除一个item，并将对象返回池里
RemoveChildToPool();
--删除一个指定位置的item，并将对象返回池里
RemoveChildToPoolAt();
--删除一个范围内的item，或者全部删除，并将删除的对象都返回池里
RemoveChildrenToPool();

    --以上都需要通过循环来一个个添加到列表中，所以一般不用于添加列表内容，只做对当个对象操作时
-----------------------------------------------------
    --自动循环添加列表
--添加一个回调，回调里对单项进行数据操作，比如对元件赋值什么的，回调最好不要用匿名函数，当要更新某个item的时候，调用该回调，传入索引和对应参数即可
list.itemRenderer=(int,GObject){};
--设置列表个数
aList.numItems = int;

-----------------------------------------------------
--点击item的时候触发的事件
list.onClickItem.Add();
```

虚拟列表

- 创建item要使用`list.itemRenderer`回调创建
- 开启滚动，必须开启滚动
- 设置好item的项目资源，就是子项样式
- 设置好项目个数`numItems`

```
--开启虚拟列表
list.SetVirtual();
--虚拟列表中item个数和显示对象个数是不一样的，item个数是实际数据个数，显示对象个数是存在的item实体个数

--获得item个数
list.numItems=int;
```

```
--项目个数
(int)asCom.numChildren;
--转换项目索引为显示对象索引。
list.ItemIndexToChildIndex(int);
--转换显示对象索引为项目索引。
list.ChildIndexToItemIndex(int);
--设置/获得item索引，可以跳转到某一个索引位置
list.selectedIndex=int;
--跳转到某一个索引位置
list.AddSelection(int,bool);
--跳转到项目最顶item
list.scrollPane.ScrollTop();
--跳到最底
list.scrollPane.ScrollBottom();
--当更新子项个数或者内容时,刷新虚拟列表
list.RefreshVirtualList();
```

- 虚拟列表不允许使用`AddChild`/`RemoveChild`等方法增添子项，只通过设置`numItems`
- 可以支持可变大小的item，在`itemRenderer`回调里可以去动态修改item的大小
- 列表支持不同的item混合

```
--混合item使用
--根据索引的不同，返回不同的资源URL
string GetListItemResource(int index)
{
    Message msg = _messages[index];
    if (msg.fromMe)
        return "ui://Emoji/chatRight";
    else
        return "ui://Emoji/chatLeft";
}
--然后设置这个函数为列表的item提供者
aList.itemProvider = GetListItemResource;
```

循环列表

- 与虚拟列表一致，只是打开方法不一样

```
list.SetVirtualAndLoop()。
```

TreeView树

```
--创建/获得实例对象,这里得到的是个空树，也就是空列表，枝干树叶都没有
treeView = new TreeView(com.GetChild(name).asList);
--将节点添加至树对象，添加给树对象的节点就是第一层文件，而添加给页节点的就是第二层第三层
treeView.root.AddChild();
--当TreeNode需要更新时回调，更列表的那个回调类似
treeView.treeNodeRender = (TreeNode){};
--点击节点时触发的回调
treeView.onClickNode.Add();
-- 当TreeNode即将展开或者收缩时回调。可以在回调中动态增加子节点
treeView.treeNodeWillExpand=(TreeNode,bool){}

--创建节点，true表示文件夹节点，false表示叶节点
```

```
TreeNode node = new TreeNode(bool);
--设置数据
node.data = data;
--节点添加子对象
node.AddChild();
--是否是文件夹
(bool)node.isFolder;
--设置/获取文件夹打开/关闭状态
node.expanded=bool;
```

PopupMenu弹出页面

- 在UI系统中我们经常需要弹出一些组件，这些组件在用户点击空白地方的情况下就会自动消失。FairyGUI内置了这个功能

```
--也可以带一个参数，指定这个菜单使用的菜单组件资源
PopupMenu menu = new PopupMenu();
--如果要修改菜单的宽度。
menu.contentPane.width = 300;
--添加一个菜单item，并注册点击回调函数
GButton item=menu.AddItem(string, MenuItemCallback);
--设置子项名字
item.name = string;
--添加分隔条
menu.AddSeperator();
--设置菜单项变灰
menu.SetItemGrayed("item1", true);
--显示菜单（菜单组件，是否是向下展开的）
menu.Show(GObject, bool);
```

弹出窗口/页面项

```
--创建一个页面或者窗口
GComponent com=UIPackage.CreateObject("Basics", "Component12").asCom;
Window aWindow=new Window();
--弹出在当前鼠标位置
GRoot.inst.ShowPopup(com/aWindow);
--弹出在aButton的下方
GRoot.inst.ShowPopup(com/aWindow, aButton);
--弹出在自定义的位置
GRoot.inst.ShowPopup(com/aWindow);
com/aWindow.SetXY(100, 100);
--通过ShowPopup打开的窗口和使用aWindow.Show显示窗口的唯一区别就是多了点击空白关闭的功能，其它用法没有任何区别
--点击空白弹出框会关闭，获得这个关闭的通知，可监听移出舞台的事件
com.onRemoveFromStage.Add();
```

Drag&Drop拖拽

```
--设置元件的可拖拽属性
aObject.draggable = bool;
--可以设置一个拖拽范围，没设置就是可以全屏拖拽，注意这里的矩形范围使用的是舞台上的坐标，不是元件的本地坐标。
aObject.dragBounds = new Rect(100,100,200,200);
--拖拽开始事件
aObject.onDragStart.Add(onDragStart);
--拖拽过程中的事件
aObject.onDragMove.Add(onDragMove);
--拖拽结束事件
aObject.onDragEnd.Add(onDragEnd);
```

转换拖动（设置元件的某个区域拖拽）

- 当不希望整个元件所有地方都是一个可拖拽的状态时就可以设置

```
--设置拖动区域为可拖动，然后侦听拖动开始事件
_dragArea.draggable = true;
--添加回调
_dragArea.onDragStart.Add(onDragStart);
--回调
void onDragStart(EventContext context)
{
    --取消掉源拖动，也就是_dragArea不会被实际拖动
    context.PreventDefault();
    --设置窗口处于拖动状态。context.data是手指的id。
    this.StartDrag((int)context.data);
}
--通过以上的方式，实现当_dragArea被尝试拖动时，会转换为Window自身的拖动
```

替身拖动（类似赋装备操作）

```
aObject.draggable = true;
aObject.onDragStart.Add(onDragStart);
void onDragStart(EventContext context)
{
    --取消掉源拖动
    context.PreventDefault();
    //icon是这个对象的替身图片，userData可以是任意数据。context.data是手指的id。
    DragDropManager.inst.StartDrag(null, icon, userData, (int)context.data);
}

--使用了替身拖动后，如果要检测拖动结束，不能再监听原来的对象，而是使用以下这个
DragDropManager.inst.dragAgent.onDragEnd.Add(onDragEnd);

//DragDropManager还提供了常用的拖->放功能，如果一个组件需要接收其他元件拖动到它里面并释放的事件，可以使用
aComponent.onDrop.Add(onDrop);
void onDrop(EventContext context)
{
    --这里context.data就是StartDrag里传入的userData
    Debug.Log(context.data);
}
```

```
//DragDropManager使用了一个图片资源表达替身，这个图片是用装载器显示的。你可以调整它的参数以适应实际项目需求
DragDropManager.inst.dragAgent=GLoader;

--如果你的替身不是一个图片那么简单，比如你需要用一个组件作为替身，那么你可以定义自己的DragDropManager
```

Window窗口

```
--创建窗口，窗口类需要继承Window类自己重写，根据窗口里面的内容定制自己的需求
MyWindow myWin = new MyWindow();
--显示窗口
myWin.Show();
--隐藏窗口，不是销毁
myWin.Hide();
--设置模态窗口，回阻止用户点击模态窗口后面的内容
myWin.modal=bool;
--获取窗口是否显示
(bool)myWin.isShowing;
--锁定窗口
myWin.ShowModalWait();
--取消窗口的锁定
myWin.CloseModalWait();
--设置自己的显示页面是什么
myWin.contentPane = UIPackage.CreateObject("Bag", "BagWin").asCom;

--可以重写当窗口打开或关闭时的效果
DoShowAnimation()/DoHideAnimation();
```

窗口管理

- `GRoot`里提供里一些窗口管理的方法

```
--把窗口提到所有窗口的最前面
BringToFront();
--隐藏所有窗口。注意不是销毁
CloseAllWindows();
--隐藏所有非模态窗口
CloseAllExceptModals();
--返回当前显示在最上面的窗口
GetTopWindow();
--当前是否有模态窗口在显示
hasModalWindow();
```

窗口自动排序

- Window是具有点击自动排序功能的，也就是说，你点击一个窗口，系统会自动把窗口提到所有窗口的最前面，这也是所有窗口系统的规范。但你可以关闭这个功能

`UIConfig.bringWindowToFrontOnClick = false;`

Transition动效

```
--获得动效
```

```
Transition trans = aComponent.GetTransition("peng");
--播放动效，播放动效有重载，可以传入回调或者指定播放范围
trans.Play();
--倒放
trans.playReverse();
--暂停和恢复播放动效
trans.setPaused(bool);
--中途停止播放动效
trans.Stop();
//Stop方法也可以带参数
--参数说明：setToComplete表示是否将组件的状态设置到播放完成的状态，如果否，组件的状态就会停留在当前时间；processCallback是否调用Play方法传入的回调函数
trans.Stop(bool, bool);
--修改某帧的属性值
trans.SetValue(name, 属性对应参数);
--运行到某帧的时候触发回调
trans.SetHook(帧标签, callback);
--可以修改动效某个标签对应动效片段的目标对象，但必须注意，要在动效停止状态下调用
trans.SetTarget(帧标签, newTarget);
--在Unity中，动效的播放速度默认是不受Time.timeScale影响的，但你也可以设定它受影响
trans.ignoreEngineTimeScale = bool;
--可以单独设置动效的timeScale
trans.timeScale = float;
```

UIConfig全局设置

- 这里保存的是设置全局默认参数的一些设置，如设置模态窗口的资源，默认音乐，默认滑动条样式，默认字体，默认字体大小等等，具体可以去类里查看

Stage舞台

- 对舞台的设置，舞台就跟UGUI中的Canvas类似，可以做监听事件，设置层级等，主要做监听事件

```
--是否点击在UI上
(bool)Stage.isTouchOnUI;
--这是鼠标的位置，或者最后一个手指的位置
(Vector2)Stage.inst.touchPosition;
--获取指定手指的位置，参数是手指id
Stage.inst.GetTouchPosition(int);
--这是当前按下的手指的数量
(int)Stage.inst.touchCount;
--获得当前所有按下的手指id
int[] touchIDs = Stage.inst.GetAllTouch(null);
```

AB包加载

- FGUI加载AB包中的内容有几个步骤
    1. `UnityWebReqyest`加载出AB包
    2. `UIPackage.AddPackage(bundle);`添加包
    3. 然后就和其它创建页面一样了

```
IEnumerator LoadUIPackage()
{
    --创建AB包路径
```

```
    string url = Application.streamingAssetsPath.Replace("\\", "/") +
"/fairygui-examples/bundleusage.ab";
    if (Application.platform != RuntimePlatform.Android)
        url = "file:--/" + url;

    --加载主AB包
    UnityWebRequest www = UnityWebRequestAssetBundle.GetAssetBundle(url);

    yield return www.SendWebRequest();

    --是否有加载错误
    if (!www.isNetworkError && !www.isHttpError)
    {
        --创建需要的AB包
        AssetBundle bundle = DownloadHandlerAssetBundle.GetContent(www);

        if (bundle == null)
        {
            Debug.LogWarning("Run Window->Build FairyGUI example Bundles
first.");
            yield return 0;
        }
        --添加包
        UIPackage.AddPackage(bundle);

        _mainView = UIPackage.CreateObject("BundleUsage", "Main").asCom;
        _mainView.fairyBatching = true;
        _mainView.SetSize(GRoot.inst.width, GRoot.inst.height);
        _mainView.AddRelation(GRoot.inst, RelationType.Size);

        GRoot.inst.AddChild(_mainView);
        _mainView.GetTransition("t0").Play();
    }
    else
        Debug.LogError(www.error);
}
```

SDK

UIPanel

- 类似Canvas的组件，必须通过这个来显示UI面板，还有一个是UIPartner，具体参数查看教程

动态创建UI

```
GComponent view = UIPackage.CreateObject("包名", "组件名").asCom;

--以下几种方式都可以将view显示出来:

//1, 直接加到GRoot显示出来
GRoot.inst.AddChild(view);

//2, 使用窗口方式显示
aWindow.contentPane = view;
aWindow.Show();

//3, 加到其他组件里
aComponnent.AddChild(view);
```

- 如果界面内容过多，创建时可能引起卡顿，FairyGUI提供了异步创建UI的方式，异步创建方式下，每帧消耗的CPU时间将受到控制，但创建时间也会比同步创建稍长一点

```
UIPackage.CreateObjectAsync("包名","组件名", MyCreateObjectCallback);

void MyCreateObjectCallback(GObject obj)
{
}
```

- 动态创建的界面不会自动销毁，例如一个背包窗口，你并不需要在每次过场景都销毁。如果要销毁界面，需要手工调用Dispose方法

使用UIPanel和UIPackage.CreateObject的场合和注意事项

- UIPanel最常用的地方就是3D UI。他可以方便地将UI挂到任意GameObject上。当然，UIPanel在2D UI中也可以使用，他的优点是可以直接摆放在场景中，符合Unity的ECS架构。缺点是这种用法对UI管理带来很多麻烦，特别是对中大型游戏。
- 使用UIPackage.CreateObject可以使用代码创建任何界面，可以应用在传统的设计模式中，Lua支持也十分方便。不过必须要小心处理生成的对象的生命周期，因为它需要手动显式销毁，并且永远不要将使用CreateObject创建出来的对象挂到其他一些普通GameObject上，否则那些GameObject销毁时会一并销毁这个UI里的GameObject，但这个UI又还处于正常使用状态，就会出现空引用错误

多点触摸

- FairyGUI支持多点触摸的处理。每个手指都会按照TouchBegin->TouchMove->TouchEnd流程派发事件

```
--区分手指ID
EventContext.inputEvent.touchId=int;
--这是当前按下的手指的数量
int touchCount = Stage.inst.touchCount;
--获得当前所有按下的手指id
int[] touchIDs = Stage.inst.GetAllTouch(null);
--关闭多点触摸
Input.multiTouchEnabled = false;
```

事件

- 事件机制