1. **新版UI系统**

**3.1 Cocos2D-X的UI系统**

**3.1.1 Cocos2D-x原版UI系统介绍**

Cocos2D-x本身是一个巨大的代码库，整合了2D游戏常有的所有功能。当然自身也包含了简单的UI库，UI库基本结构如图3.1所示。

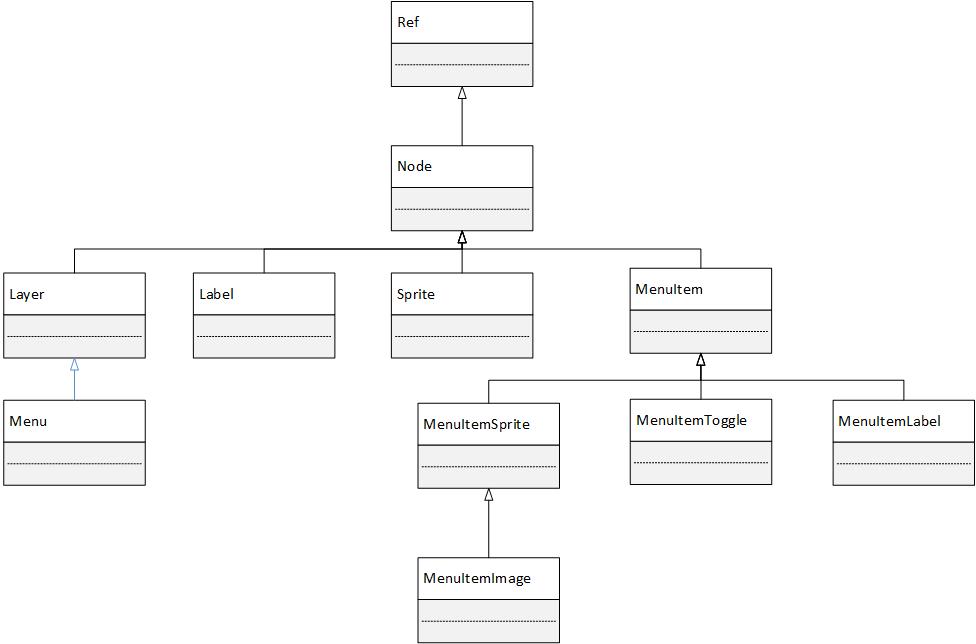


图3.1 cocos2D-x的UI系统

在Cocos2D-x中，所有的UI组件全部继承于Node,Node提供了在2D游戏中最基本的布局信息，比如说位置position，锚点achorPoint，UI大小contentSize等。Node类有一父类Ref，所以所有UI组件存储全部受Cocos2D-x的内存管理系统管理。引擎提供了最基本的UI组件，层(Layer),字体(Label),精灵(Sprite)，按钮(MenuItem)。2D游戏中，场景中大部分的UI元素是由四边形纹理构成，每一个四边形纹理我们使用Sprite类来加载。Cocos2D-x自身提供多样化的字体系统，根据使用方法的不同分为，LabelTTF TTF文本标签控件，LabelAtlas自定义字体文本标签控件，LabelBMFont位图文本标签空间。或者使用他们统一的父类Label。在Label中可以统一选择想要的字体使用方式，完成Label的使用过程。游戏中的所有按钮有统一的父类MenuItem，根据使用按钮的样式不同分为MenuItemSprite,MenuItemImage,MenuItemLabel。其中MenuItemSprite,MenuItemImage按钮由纹理构成，MenuItemLabel由字体构成。MenuItemToggle在内部拥有一个MenuItem数组，在触摸过程中可以展示不同的按钮状态。Menu类负责挂载MenuItem，自身含有触发器，可以触发游戏中的触摸事件。

**3.1.2 存在的局限性**

Cocos2D-x的UI系统功能简单，一般的小型游戏开发起来没有问题。但是对于大型游戏来说，未必有些太简单了，并且Cocos2D-x原版引擎只是一个代码库，并不含有编辑器，所以如果直接使用引擎，势必需要在游戏中用代码硬编译的方式存入UI位置，尺寸等信息，势必造成开发的不变。

后期官方提供了Cocos Studio和Creator等编辑器，但在编辑器中，只提供最基本的游戏UI组件。诸如菜单，进度条等大型组件还需要通过复杂的方式手动配置，扩展性不强，无法实现资源和代码的二次利用。且编译器目前功能不完全，并且存在Bug。

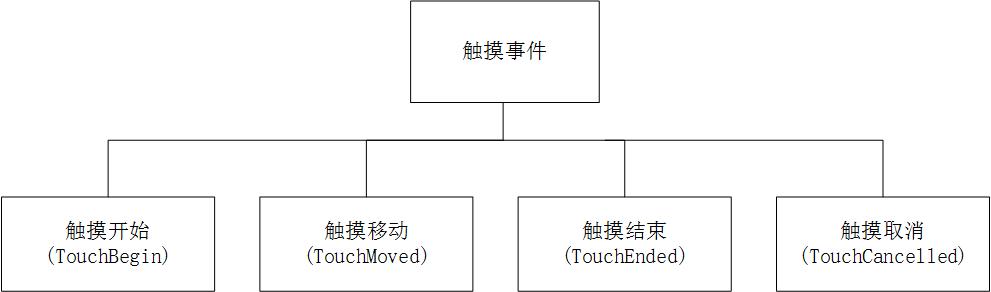
手游开发中，不同手机有着不同尺寸，并且手机和平板之间的尺寸比例和分辨率差别过大，如把游戏分为手机和平板两个版本开发，势必增加游戏开发耗时及在后期的版本更新过程中出现问题，如果使用同一套UI系统，在UI空间屏幕适配的过程中容易出现问题。原版UI系统中，我们只能通过位置和锚点等基础信息来调整UI位置，从而完成UI布局。这种方法不同于其他应用的布置方法，容易出现错误。

**3.2 UI系统需求分析**

**3.2.1 手游UI系统需求**

早期的手机不待用触摸功能，多为带键盘手机，玩家通过键盘的操作来控制游戏，玩家按下键盘直接监听键盘按下的消息，并调用特殊的逻辑对相应的UI进行修改。但近十年，随着智能手机的发展，电容屏手机已基本实现了普及，手机游戏多通过触摸屏幕和虚拟遥感的方式控制，触摸几乎成为了操作手机的唯一方式。

但触摸功能的引用，造成了UI系统设计的越来越复杂，因为触摸控制分为单点触控和多点触控，并且游戏中UI实体很有可能会出现重合，在重合应设定相应的处理方法以获取到正确的UI。触摸功能需要采用一种完全不同于PC平台的输入事件接收方式，我们通常把一条触摸事件分为四条处理TouchBegin，TouchMoved，TouchEnded,TouchCancelled处理，并且配合适当的架构，才能解决触摸事件分发问题。

图3.2 手机游戏的触摸事件分解

以当前流行的网络游戏为例，UI系统已经成为不可缺少的一部分，尤其对于2D游戏来说，本质上view层全部是由UI元素组成。在实际生产过程中，我们需要提供一套完整并接快速的工作流程，这套工作流程中可以通过可视化的方式调整UI布局，并且在游戏中以相同的方式去显示，即使是在不同分辨率的设备上，也需要通过实现设定好的布局显示。可以快速为特定游戏设计出指定的UI实体，并且在游戏的另外一个地方重用这个UI实体。同时在游戏中，系统会统一管理UI配置过程中的配置文件，适当的优化配置文件的体积大小，以至配置文件不会占用过大的空间，自动管理加载配置文件中所需要的纹理等资源。

**3.2.2 新版UI系统需求**

在分析了引擎原版UI系统的局限性和手游UI系统需求之后，我们为新版UI系统提出了以下的需求:

1. UI系统对游戏UI配置文件的管理

在开发过程中，UI编辑器和游戏属于不同的两个程序，需要设计一套配置文件以便在两个程序中读取和编辑。在UI编辑器中可以通过读取特定的配置文件，看到该页面在游戏中加载以后的效果。同时在安卓或是IOS上读取该配置文件以显示对应的UI界面。使用配置文件配置UI方法可以减少我们重编译客户端代码的次数，如果我们修改了UI界面配置文件，只需要重新该界面。

在游戏的不同平台中，我们同时需要构建一套配置文件加载系统，可以在游戏开始时加载所有的配置文件，在读取到某一配置文件时，在去特定位置加载该配置文件所指定的资源。

2.UI系统可以和屏幕更好的适配

在手游的UI系统中，屏幕的适配是一个很重要的问题。在安卓平台上，不同厂家的机型有着不同的分辨率和屏幕尺寸。在游戏一类的应用中，图片资源的大小占整个包体很大的比重，如果针对不同分辨率配置多套资源，势必加载包体的大小以及整体开发的难度。所以一般通过只配置一套资源并附加适当的屏幕适配方法的自动适应屏幕。

3.UI控件的管理

对于一个UI系统最重要的就是对控件的管理。其中包括UI的加载，卸载，删除等管理。同时根据具体UI不同，我们还需要提供一些配置功能。比如说调整位置，显示和隐藏，配置响应函数，比例等信息。甚至可以预先根据控件想要表现的功能挂载一些逻辑代码甚至shader。在UI编辑器中通过修改配置文件以判断是否触发这段逻辑代码或者shader。

同时在UI管理系统中，我们也需要重新构建个UI树用来管理这些UI控件，每一个UI控件都是一个节点，以便和cocos2D-x原有的渲染系统交互，保证UI控件的遍历不会在中途出现问题。并且把加载相关的操作流程做到最简单，涉及到配置文件的地方，只需要加载即可，其他工作全由UI系统自己完成。

4.系统级UI控件的实现

针对与手机游戏，我们在Cocos2D-x原有控件的基础上，重新开发一套新的UI控件。这套控件为游戏底层控件，需要满足Cocos2D-x所能实现的各方面功能。

为此，我们设计了如表3. 所示的几种基本的游戏控件。

|  |  |  |
| --- | --- | --- |
| 类名 | 控件名 | 功能描述 |
| Image | 静态图片 | 以一张静态图片的方式 |
| Button | 按钮 | 游戏中的按钮实体，点击按钮可以触发相应的工作，包括三种状态未触摸，触摸开始，触摸结束，针对这三种状态可以显示不同的静态图片或者文字 |
| Label | 文字 | 游戏中的字体控件，封装了Cocos2D-X所有的字体控件，在配置文件中可以调整不同的配置，达到显示不同信息的目的 |
| Toggle | 按钮触发器 | 由多个按钮组成，在不同按钮之间可以进行切换 |
| SliderBar | 拖拽条 | 游戏中的拖拽条实体，通过滑动拖拽条的方式，调整某一值的大小 |
| InputEdit | 文本输入框 | 文字信息输入 |

1. 游戏级UI控件的实现

随着游戏的开发，我们需要越来越多的UI控件，有大量的游戏UI是卡牌类型游戏所必须的，为了游戏的扩，展性，我们对所开发的游戏的UI进行总结，统计出常用的大型的UI控件，比如卡牌显示菜单，推拽条等等，如表3.所示。使用配置表去管理他们的布局，只需要在配置表中添加少量代码即可完成新控件的添加。

|  |  |  |
| --- | --- | --- |
| 类名 | 控件名 | 功能描述 |
| 菜单实体 | MenuEntity |  |
| 卡片头像 | AvatarEntity |  |
| 切换编队按钮 | FleetButton |  |
| 倒计时器 | TimerEntity |  |
|  |  |  |

**3.3 新版UI系统总体设计**

作者总结了市面上其他的UI系统，以及我们的需求，最后设计出新版的UI系统，新版UI系统的整体结构图如图3. 所示。



图3. 新版UI系统

新版UI主要分为5个大部分，配置文件管理模块，核心模块，系统级控件，游戏级控件和相对应UI编辑器。

配置文件管理模块：配置文件管理模块主要负责游戏中的UI配置文件的加载和释放，控件属性的加载，以及如何管理配置文件相关联的资源。

核心模块:核心模块是新版UI系统最重要模块，其中包括了UI的组建方法，布局管理等。

系统级控件:UI系统最基本的几种控件。

游戏级控件:根据具体游戏需求，扩展出来的其他控件。

UI编辑器:提供一种可以可视化查看以及修改配置文件的方法。

**3.4 配置文件管理模块设计与实现**

配置文件我们选用xml。为了保证游戏中的配置文件加载速度，游戏中使用了RapidXml加载xml。RapidXml相较于其他XML解析库。他的速度能快十倍以上，并且设计简单，在实际使用时，并不占用过多的内存空间，其中XMLMgr负责统一管理配置文件，XMLParser封装了解析XML常用的API，因为XML占内存容量小，在游戏的初始化过程中，我们加载所有的配置文件，统一放在XMLMgr的内存池中，每一个配置文件为一个XMLData，以加载过程中的路径作为索引ID。

如图3.所示，该图为游戏中新版UI系统的设计图。其中暴露给开发者的是UIMgr这个类。这个类加载UI控件，负责管理所有UI控件，根据配置文件中设定的UI控件的id找到相应的UI控件。在加载场景时，我们通过这个类的API进行加载，下面描述加载的全过程。

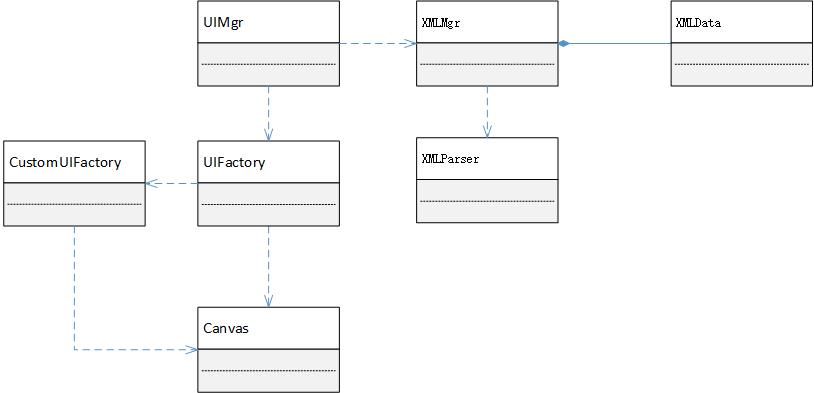


图3. 新版UI系统类图

配置文件管理模块为了加大模块的扩展性，UI的创建使用了工厂模式。其中所有的系统级控件使用UIFactory类创建，所有的游戏级控件使用CustomUIFactory创建。工厂类根据从xml中解析出来的文本标签来判断该UI控件属于何种类型，创建相应组件的构造和初始化函数。

如图3. 所示，该图为UI加载过程中的流程图。首先我们在XMLMgr中调用加载函数API，遍历XMLMgr中的内存池，如果在池中发现了对应的XMLData，返回当前XMLData，如果池中不存在，则根据当前路径从文件中加载XMLData。判断当前节点节点是否解析成功。如果不成功直接返回。当前节点节点解析成功，皆解析出当前节点的标识符，把当前标识符以字符串的形式传入工厂类中。工厂类分为UIFactory和CustomUIFactory，UIFactory负责系统级控件的构造。CustomUIFactory负责游戏级控件的构造，这两个类通过责任链设计模式连接。控件创建成功以后，下来解析出当前文本标签所带有的所有属性名和属性值，以两个不同的队列传入控件的Parse函数中，在Parse函数中分析两个队列的属性并且完成属性值的解析，调用适合的属性修改函数。完整这步以后，一个控件配置完成，但在一个UI框架是由一个UI树组成，所以我们还需要以相同方式处理配置文件的下一个节点，直到整个UI树解析完毕。

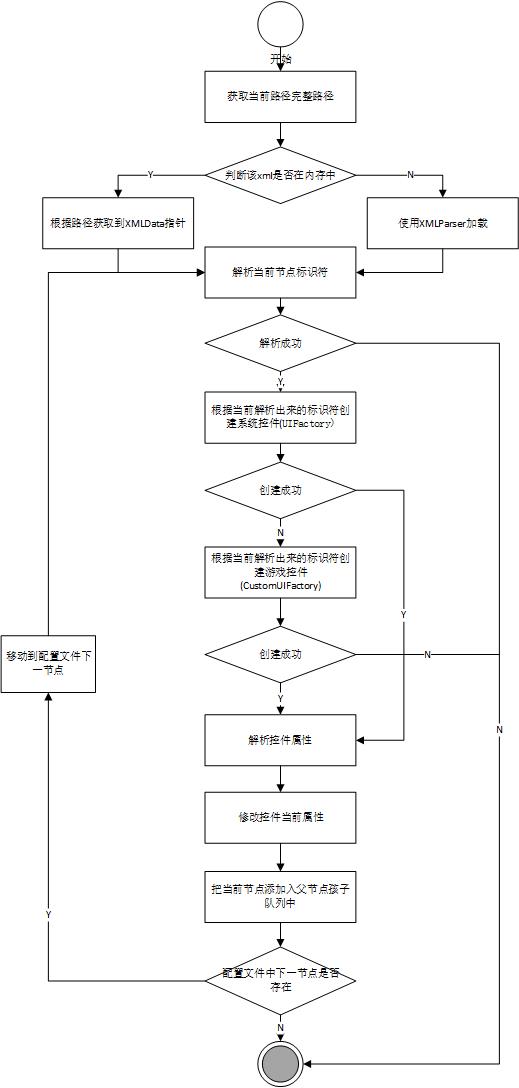


图3. UI控件加载过程流程图

**3.5 核心模块的设计与实现**

为了方便UI系统的布局，简化开发难度，我们重新设计了UI系统中布局的方式。我们借鉴了其他UI系统中的布局方式，提出了新式的UI布局方式。在新版的UI系统中，我们定义了四个变量Left,Right,Top，Bottom。分别代表着当前UI控件距离父UI控件边界的距离。在不同机器型号分辨率适配过程中，减少了UI冲突可能性，只需要设定UI控件之间的距离就可以完成布局。为了照顾使用时的特殊情况，仍然保留原先的利用位置和锚点定位的方式。

Canvas为基础类，相当于原版Cocos2D-x中的Node。Canvas直接继承于Node，提供了UI控件基础上的进一步的功能。同时Canvas内部保存着一颗UI树，与原本UI树为不同的两颗树,但两棵树保持同步。

表3. Canvas属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Id | String | 控件名称索引 |
| Width | Float | 控件宽度 |
| Height | Float | 控件高度 |
| ContentSize | Bool | 是否使用宽度和高度约束控件大小 |
| Position | Vec2 | 控件位置 |
| Z | Float | 控件的GlobalOrder大小 |
| Left | Float | 控件距离父控件左边界距离 |
| Right | Float | 控件距离父控件右边界距离 |
| Top | Float | 控件距离父控件上边界距离 |
| Bottom | Float | 控件距离父控件下边界距离 |
| Anchor | Vec2 | 锚点位置 |
| IgnoreAnchor | Bool | 是否使用锚点，不使用情况下使用默认锚点(0.5,0.5) |
| Visible | Bool | 控件初始化时是否可见 |
| Touchabled | Bool | 控件是否响应触摸事件 |
| Scale | Vec2 | 控件缩放尺寸比例 |
| SwallowTouch | bool | 控件的子控件是否响应触摸事件 |
| TouchExtend | Int | 触摸区域是否向外延展 |

(1)原先cocos2D-x变量

Position指代UI的位置，Anchor和指代锚点的位置， IgnoreAnchor指代是否忽略锚点属性，如果为True则忽略，锚点选用默认的(0.5,0.5)。这套方法是原先UI系统中布局UI的方式，如果在配置文件中， Left,Right,Bottom,Top任意一个属性存在，则忽略原先Position,Anchor的配置。Visible使用方法与原先UI系统一样。

(2)ID

通过配置文件提前设置ID的方式，作为UI元素的映射。如在UI系统中想要寻找到某一控件，无需提前保持该控件的指针，直接调用UIMgr中的FindUIByID函数，该函数会遍历当前UI树，并且匹配每个UI控件的ID值。如果发现某个UI控件的ID值与想要找的控件ID值相同，则返回该控件。

(3)Left,Right,Top,Bottom

该四个变量为新版UI系统的核心所在，Left指代当前控件距离父控件左边界距离，Right指代当前控件距离父控件左边界距离，Top指代当前控件距离父控件左边界距离，Bottom指代当前控件距离父控件左边界距离。如果当前UI控件是整个UI树最顶层控件，则四个属性分别指代屏幕的四个边界。

(4)Width,Height,ContentSize

Width和Height,来控制当前控件大小，如果Width,Height设定的值比当前控件要大，则当前控件做居中处理，剩下边界做透明处理。我们Width和Height可以制作一个空白控件，方便他的子控件布局。ContentSize属性负责Width和Height属性是否有效，默认为True。如果设成False，则Width和Height的大小不会影响控件的大小。

(5)Touchabled,SwallowTouch,TouchExtend

Touchabled,SwallowTouch,TouchExtend负责控件的触摸管理，Canvas相较于Node，自身携带触摸事件的监听器，能够监听触摸事件的分发，UI控件分发的顺序为渲染UI控件的相反顺序。Touchabled负责开关控制触摸，默认Touchabled为False，如果为True则该控件接收触摸事件的分发。在实际使用时，有可能多个UI控件会叠加在一起，通过设定SwallowTouch，我们可以组织，触摸事件分发到当前UI控件下面的那个控件，组织触摸事件的进一步传递。TouchExtend是为了延伸触摸区域大小，触摸区域可以比控件实际渲染区域更大。方便了小控件的使用。但实际使用时，要注意UI控件延伸区域的重叠。

**3.6 系统级控件的设计与实现**

**3.6.1 Image**

Image为游戏中最基本的控件，提供了一张静态图片的显示。

表3. Image属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| SourcePath | String | 图片资源路径 |
| Rect | Vec2 | 图片显示范围 |

(1).SourcePath

SourcePath为加载图片资源路径，在UI配置文件时，并不进行图片纹理资源的加载，而是等待加载改UI控件时，通过Cocos2D-x默认的资源管理类统一加载，所以该纹理资源的内存管理不由XMLMgr负责，由资源管理类负责。

(2)Rect

在实际使用过程中，需要对特殊图片裁剪使用。对此我们使用Rect来设定图片显示范围。

**3.6.2 Button**

Button为游戏中的按钮控件，是游戏中最常用的控件之一。按钮控件一共有三种状态，未被触摸状态(Up)，被触摸状态（Down）,以及移出响应区域状态(Disabled)。三种状态可以挂载三种不同的控件，当处于特定状态时显示当前状态的控件。

Button由三个Canvas组成，分别指代三种不同状态下的Canvas。Button重构了OnTouchBegin和OnTouchEnded来达到设计的功能。实际使用中，首先通过FindWidgetID找到该Button，之后SetCallBack为该Button添加特定的回调函数，根据按钮触摸情况，分发事件至回调函数中。

表3. Button属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| ScaleMode | Bool | 是否开启按钮缩放模式，如果开启点击按钮之后，按钮会缩放固定倍率 |
| ScaleModeFactor | Float | 按钮缩放倍率，需要提起开启按钮缩放模式 |
| OriginScale | Float | 按钮原始倍率 |
| Up | Value | 按钮未被触摸时，所显示的控件，可以为字体控件或者图片控件等 |
| Down | Value | 按钮被触摸时，所显示的控件，可以为字体控件或者图片控件等 |
| Disabled | Value | 按钮触摸以后，移出响应区域，所显示的控件，可以为字体控件或者图片控件等 |

ScaleMode,ScaleModeFactor,OriginScale

（1）使用ScaleMode来开发按钮缩放模式，当ScaleMode为TRUE时，触摸按钮以后，按钮会缩放固定倍数比例，缩放事件固定不变。缩放倍率由ScaleModeFactor属性设置，而原有控件缩放倍率为OrginScale。

(2)Up,Down,Disabled

指代按钮三种状态下所显示的控件，该控件可以为文字，静态图片等。在静态文件存储时，挂载在按钮的子节点中，当解析一个按钮控件，需要查看他的子节点是否为Up，Down，Disabled标签，三个标签所指代的控件为Button控件三种不同的状态。

**3.6.3 Label**

Label是游戏中基本的组件，指代游戏中的字体控件。本Label是在原先UI系统的Label上封装了一层，并且提供了利用配置文件设定Label基本属性的功能。

表3. Label属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Text | String | 字体控件默认字符串 |
| FontName | String | 字体控件字体名称 |
| FontSize | Int | 字体控件字体大小 |
| FontColor | Vec4 | 字体控件字体颜色 |
| Valign | String | 字体垂直对齐方式，可以居上对齐或是下对齐 |
| HAlign | String | 字体水平对齐方式，可以左边对齐或是右对齐 |
| OutLine | Int | 字体外发光尺寸，默认为0 |
| OutLineColor | Vec4 | 自体外发光颜色，默认为白色 |

(1)Text

字体控件默认的字符串，最后显示在窗口中的静态字体

(2)FontName,FontSize,FontColor

字符控件的基础属性设置。

(3)Valign,HAlign

字符控件可以指定其width和height的大小，在控件中字体无法填满控件的情况下。Valign,Halign两个属性可以指定了如何对齐的方法，Valign有两个选项Top和Bottom，分别代表了对齐到上边界和对齐到下边界。Halign有两个选项Left和Right，分别代表了对齐到左边界和对齐到右边界

(4)OutLine,OutLineColor

指代字符控件是否开启外描边以及外描边的颜色。

**3.6.4 Toggle**

Toggle的功能类似于原先UI系统中的MenuItemToggle.Toggle在内部拥有一个Button数组，在内部负责展现了不同的按钮状态，通过触摸事件完成不同状态之间的切换,在切换过程中可以触发相应状态的回调函数。

表3. Toggle属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Size | Int | 所挂载的按钮大小 |
| Button | Button\* | 所挂载的按钮 |

Size指代所携带的Button数量，之后根据Size数量解析固定长度Toggle子节点，该子节点为Button类型控件。指代Toggle控件之间的各个状态.

**3.6.5 SlideBar**

表3. SlideBar属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Id | String | 控件名称索引 |
| Width | Float | 控件宽度 |
| Height | Float | 控件高度 |
| ContentSize | Bool | 是否使用宽度和高度约束控件大小 |
| Position | Vec2 | 控件位置 |
| Z | Float | 控件的GlobalOrder大小 |
| Left | Float | 控件距离父控件左边界距离 |
| Right | Float | 控件距离父控件右边界距离 |

**3.6.6 InputEdit**

InputEdit为游戏中的输入框控件，在登录游戏，修改编队名称这里都需要输入框控件。输入框控件自带监听器，可以响应触摸事件的分发。

表3. InputEdit属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| DefaultText | String | 输入框控件提示字符串 |
| Text | String | 输入框控件输入字符串 |
| DefaultColor | Vec4 | 提示字符串颜色 |
| FontName | String | 输入框控件字体名称 |
| FontSize | Int | 输入框控件字体大小 |
| FontColor | Vec4 | 字体控件字体颜色 |
| MaxNum | Int | 输入框控件可输入最大的字符数 |
| PasswordOpen | Bool | 密码框开关 |

(1)DefaultText,DefaultColor,Text

在字符框为空时，字符框中可有提示字符串。指示玩家该框输入字符串的具体要求，提供交互性良好的界面。DefaultText指代提示字符串，DefaultColor指代提示字符串颜色，Text指代输入字符串，其他系统模块取这个值以完成具体需求。

(2)FontName，FontSize,FontColor

字符串最基本的设置信息

(3)MaxNum

字符框所继承的canvas中的width和height指定字符框大小。在输入过程中，对输入字符串数量有明确的限制， 比如说用户名不能超过某一固定字数，如果超过固定字符，则禁止继续输入。MaxNum这一属性则指定了当前字符框所能输入的最大字符数量。

(4)PasswordOpen

游戏对于密码这类输入，要以‘\*’或‘。’来代替明文显示，PasswordOpen这用来明暗文显示，如果为True则为暗文显示，为False则为明文显示。

**3.7 游戏级控件的设计与实现**

**3.7.1 菜单控件（ListEntity)**

菜单控件用于在游戏中选择和修改物品，如图3.所示。游戏中一共有三个系统需要用到菜单控件，分别为编队系统，修理系统，建造系统。菜单可以实现移入移出功能，在移入和移出过程中屏蔽所有的按键事件，并且附加背景遮罩。菜单用于显示所表示物品的基本信息，并可以根据特定物品属性对物品排序。

菜单类为ListEntity，为了分离各部分功能，实现模块间解耦，菜单外部通信以及模版的内部逻辑，大量使用回调函数。同时在可能新增功能的地方预留了slot[23]。ListEntity继承于CutoverMotion，CutoverMotion提供了实体移入移出功能，需要提前提供实体的原始位置和切入位置，以及所触发的切入事件和切出事件。CutoverMotion类带有LayerCover实体，负责捕捉切出动作的事件并且同时提供了遮罩功能。ListEntity中每一个物品以简要信息的形式展示，称作一个Cell。一个物品占据一行，每一行相当于一个小的UI控件，可以用SetCellCallFunc的方式指定每一行的UI控件。菜单中保存着队列，存储着所有物品的指针，队列中指针的顺序既为控件在菜单中显示的顺序,控件的排序方式需要提前使用SetSortCallfunc提前设定，使用时通过Menu上的Toggle实现。菜单保留着一个模版队列，并且设有脏指针，在每帧中进行判断，如果模版队列与实际队列不相同，则重新加载所有Cell。



图4. 菜单模块界面

表3. ListEntity属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Gap | float | 菜单单元之间间隔，默认为0 |
| Align | String | 对齐方式，提供左对齐和右对齐 |
| PageCapacity | Int | 每一页最大容量，需要提前开启分页模式 |
| Paging | Bool | 是否开启分页模式 |

(1)Gap,Align

Gap指代每个Cell之间的间距，默认为0。Align指代菜单中Cell中对齐方式。一共有两种对齐方式可以选择，左对齐和右对齐。

PageCapacity,Paging

如果该菜单所含物理数量庞大，同一界面未必能全部显示，这是设置Paging为TRUE，同时由PageCapacity设定一页中Cell大小。开启分页模式，分页菜单由ListButton实现。

**3.7.2 卡片头像(AvatarEntity)**

表3. ListEntity属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| Id | String | 控件名称索引 |
| Width | Float | 控件宽度 |
| Height | Float | 控件高度 |
| ContentSize | Bool | 是否使用宽度和高度约束控件大小 |
| Position | Vec2 | 控件位置 |
| Z | Float | 控件的GlobalOrder大小 |
| Left | Float | 控件距离父控件左边界距离 |
| Right | Float | 控件距离父控件右边界距离 |

**3.7.3 列表切换按钮(ListButton)**

列表切换按钮是一系列按钮以一定序列间隔排布，通过在不同按钮之间切换的方式切换状态，同时只能有一个按钮被选中。如图3.所示，是游戏中的编队列表切换按钮。按钮的最大数量和初始按钮位置可以提前设置。切换列表所调用的回调函数通过SetCallTouchCallfunc提前指定。

表3. ListButton属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| MaxButtonSize | Int | 最大可见的按钮数量 |
| StartButtonNum | Int | 初始按钮位置 |
| EnableButtonSize | Int | 已经开启的按钮数量 |
| Up | Value | 编队按钮单元未被选中时，所显示的控件 |
| Down | Value | 编队按钮单元被选中时，所显示的控件 |
| Disable | Value | 编队按钮单元未被开起时，所显示的控件 |

(1)MaxButtonSize，StartButtonNum，EnableButtonSize

MaxButtonSize指代列表序列中最大可见按钮的数量，EnableButtonSize指代序列中已经开启的按钮数量，为开启的按钮以Disable状态显示。StartButtonNum指代列表切换按钮初始化时，当前触发列表中按钮的位置。该值不能大于以开启的按钮的数量。

(2)Up，Down，Disable

Up，Down，Disable是按钮的三种状态，Up为未触发状态，Down为触发状态，Disable为未开启状态。

**3.7.4 倒计时器(****TimerEntity)**

在游戏中，在修理系统和建造系统中，需要倒计时控件，倒计时控件在时间到的时，回调提前注册好的回调函数。通过SetCallFunc这个函数提前指定回调函数。

为了防止游戏切入后台再切回以后时间显示出现问题，提前计算出倒计时完成的时间戳，每次刷新时当前时间戳减倒计时完成时间戳来获得剩余时间。游戏中扩展了时间管理类，当前时间戳由std::chrono::steady\_clock::now()来得到，相比传统的时间戳获得方式，这种通过计算设备CPU运行时间内时间钟数量来获得时间戳的方式，时间戳的值不会因为我们修改了系统时间而改变。

表3. TimeEntity属性表

|  |  |  |
| --- | --- | --- |
| 属性名 | 类型 | 说明 |
| StartTime | String | 开始倒计时间 |
| Multiple | Float | 和现实时间的倍率，默认为1.0 |

(1)StartTime

开始倒计时间以字符串的形式指定，七小时三十分钟剩余时间以”07:30”的形式配置。读取到该字符串以后，通过转换函数转换为时间戳的形式。

(2)Multiple

Multiple属性指代与现实时间流失的倍率，如multiple值为2.0，则现实时间每倒计时2秒钟，倒计时器只走了1秒。

**3.8 UI编辑器**

**3.8.1 UI编辑器的需求**

**3.8.2 UI编辑器的设计实现**