搬运工客户端手册

# 引言

## 概述

搬运工客户端是搬运工平台的客户展现端，是Phi客户端的进化版本，支持在Windows, Android, iOS, 浏览器(WebAssembly)上跨平台运行。

该框架本身与具体业务无关，它提供一套用于构建业务系统的基础技术实现和规范集，内置一套复用性很强的客户端功能控件，利用这些预定义的控件可直接快速的开发出符合常规要求的业务系统。同时，该框架具有高度的开放性和扩展性，开发人员可根据规范在不同层面、不同粒度下扩展框架，从而满足特定的业务需求。

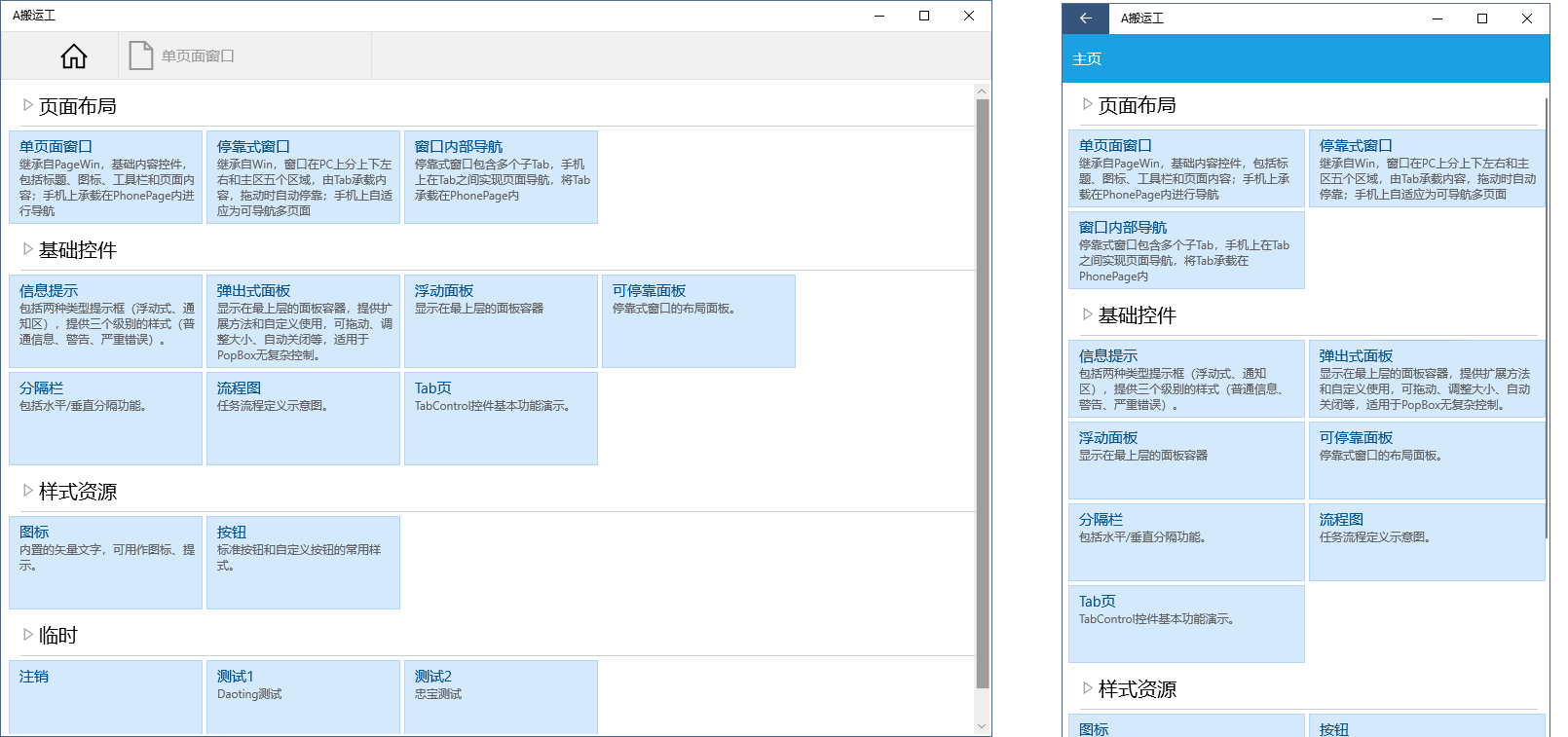
## 名词解释

* Dt：Data Transfer的缩写，搬运工平台的简称，“我们是数据的搬运工”。
* XAML：eXtensible Application Markup Language的缩写，发音为"Zamel"，是微软为构建应用程序用户界面而创建的一种新的描述性语言。在WPF, Silverlight, uwp框架中使用。
* Xamarin：一个跨平台开发框架，支持iOS, Android, uwp, Mac App应用开发。
* Uno：第三方开源项目，用于uwp和iOS, Android, WebAssembly的桥接器。
* wasm：WebAssembly的缩写，能够运行在浏览器中的一种字节码标准。

# 界面框架

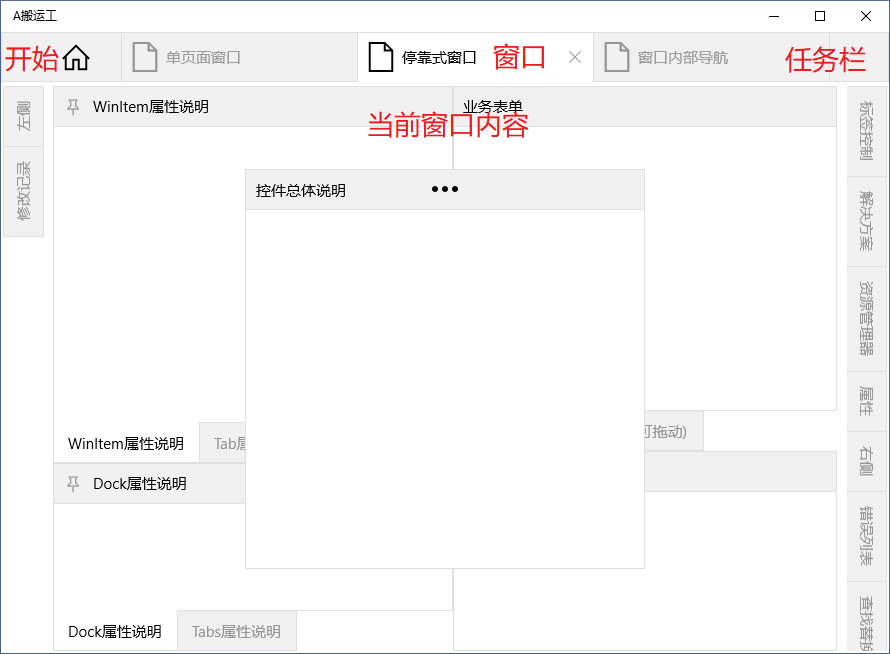
## 界面模式

内置两种界面模式：**windows模式和Phone模式**，iOS, Android上是Phone模式；uwp和wasm上支持两种模式的自动切换，当应用界面的实际宽度足够时按照windows模式显示，宽度较小时自动切换到Phone模式。程序中通过AtSys. IsPhoneUI判断当前模式。



### Windows模式

客户端运行在界面大小富裕的平台上的模式，如windows上的uwp、mac上的浏览器wasm等，该模式模拟windows操作系统，支持多窗口显示，对应关系如下：



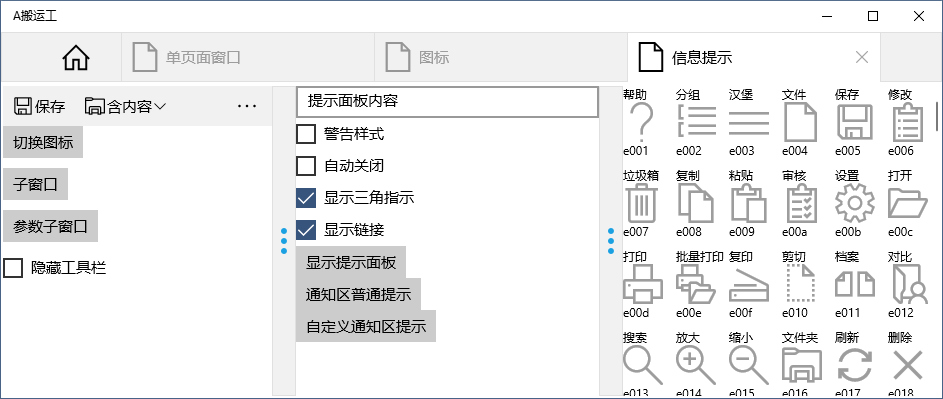
**任务栏**



任务栏按钮支持右键菜单和水平拖拽调序

右键功能

* 设置自启动后，再次启动应用时自动打开当前窗口，只支持设置一个自启动，可以取消自启动。
* 关闭其他即关闭除当前窗口外的所有其他窗口。
* 停靠在两侧的功能模拟多窗口平铺的情况。



### Phone模式

iOS, Android上的显示模式，宽度较小的uwp和wasm应用也使用该模式。



除首页外，其他页面都包含返回按钮，普通窗口的标题显示两行，第一行大字为当前Tab.Title，第二行小字为Win.Title，点击标题部分可以设置或取消自启动，设置自启动后，再次启动应用时自动打开当前窗口的首页但并不是当前页面！其他自启动功能和windows模式相同。右侧工具栏最多显示两个按钮，可以是字符图标或文字，多余的按钮在更多下拉面板中。

**为uno节省可视树级数，TextBox在级数多时崩溃！！！**

## 框架结构

平台真正实现一套代码多种界面模式运行，将大小屏幕无缝融合，内部定义了**四层结构**：桌面层/页面层、对话框层、提示信息层、浮动层。



### 桌面层/页面层

该层是最底层的主要内容区域，在Windows模式时根节点为Desktop，可以容纳多个IWin窗口，在phone模式时根节点为Frame，主页作为首页，可以导航到任意页面。窗口PageWin、Win、NaviWin适用于两种模式，PageWin为页面窗口，windows模式为独立窗口，phone模式为单页面；Win是支持可停靠的多区域普通窗口，windows模式为独立窗口，phone模式多个页面，页面之间的导航关系可在运行时自定义；NaviWin是导航列表和窗口的组合，用来处理动态多窗口切换的情况。

### 对话框层

对话框层始终显示在桌面层/页面层的上面，可以同时显示多个对话框，根据焦点自动调整对话框显示次序，Dlg自适应两种模式，支持多种显示位置，支持异步等待到对话框关闭，默认情况点击对话框外部时自动关闭。

### 提示信息层

提示层在对话框层的上面，主要显示非交互类信息，通过AtKit.Msg, AtKit.Warn方法显示，windows模式显示在右下侧，phone模式显示在顶部，支持同时显示多个、自动关闭、单个操作按钮等功能。

### 浮动层

浮动层采用系统的Popup，主要用于显示下拉菜单、需要交互的提示信息等，始终有全屏遮罩，使用场景较少。

## 基本元素

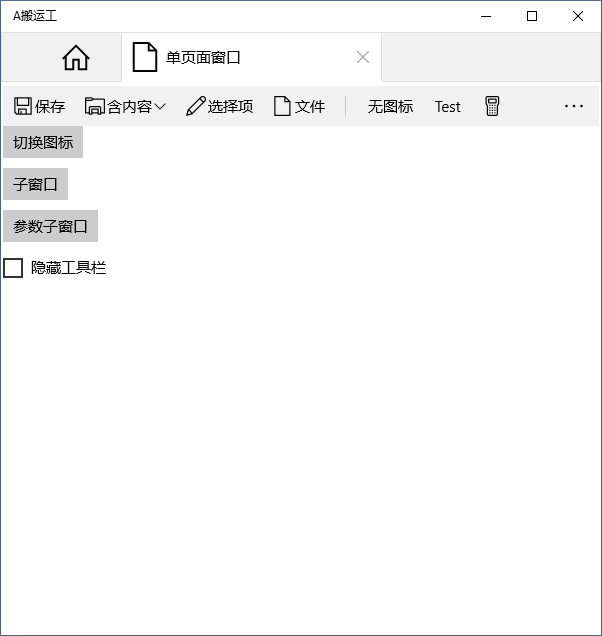
平台基于**两种界面模式和四层结构**，内置多个基本界面元素，包括页面窗口、普通窗口、导航窗口、对话框、提示信息、浮动面板、Tab页、通用PhonePage、桌面等。

### 窗口

窗口是框架结构的基本单位，Windows模式下通过Desktop管理任务栏和窗口内容的联动，窗口是实现IWin接口的任意界面元素，根据窗口内容的不同，内置三种常用类型：页面窗口(PageWin)、普通窗口(Win)、导航窗口(NaviWin)。

#### 页面窗口PageWin

不包含内部导航，整个窗口等同于Phone模式的一个页面，用来处理简单功能，使用场景较少。PageWin继承自Control，包含标题、图标、工具栏、返回按钮、内容，Windows模式下只显示工具栏和内容，Phone模式下显示完整结构。

#### 普通窗口Win

在windows模式下采用传统的可停靠布局方式，窗口内部分为多个区域，每个区域包含多个分类内容，内容之间可能存在逻辑上的父子关系或无直接关系，内容的承载者为Tab，各区域之间可以调整大小，支持浮动子窗口、停靠到四周、拖拽自动组合、恢复默认布局等功能，可视树结构为：Win->WinItem->Tabs->Tab->内容，浮动、停靠、拖拽组合时以Tab为最小单位进行。Windows模式下无返回按钮，在Tab中同样位置提供停靠按钮，phone模式下除主页外，其他页面都显示返回按钮。结构如下图所示：

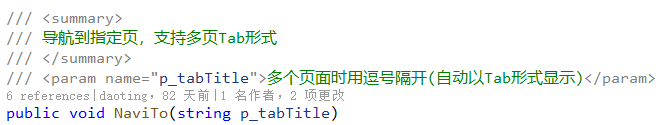
****

在Phone模式下一个普通窗口包含多个页面，多个页面之间可以通过导航方式或标签页方式进行组织，实现窗口所有内容的展示，Tab为页面内容的承载者，是页面导航或标签页的最小组织单位，这种展现窗口所有内容的方式称为**窗口内部导航**。Win继承自ItemsControl，在Phone模式下Win不作为界面直接显示，可以把它看作所有Tab的集合，容器WinItem, Tabs也无需布局管理。

** **

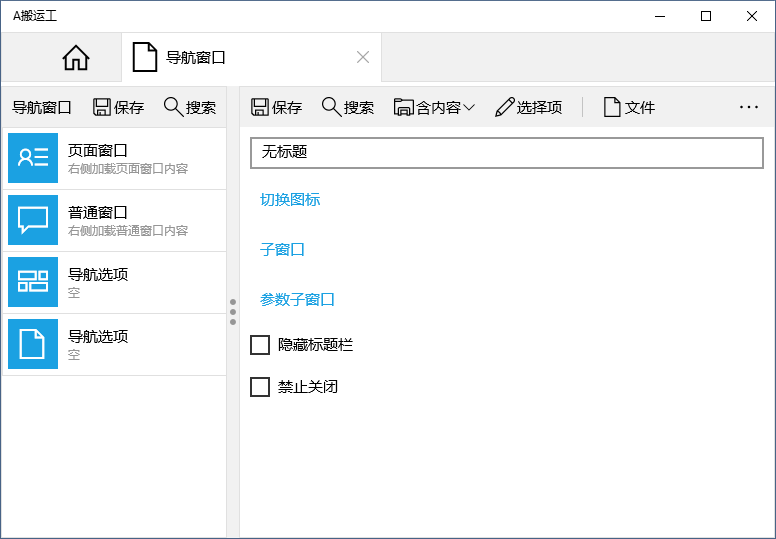
**窗口内部导航**

Win中所有Tab的Title不能重名！在Phone模式下Win提供两种类型的导航：导航到单个Tab，导航到多个Tab。导航过程可能为向前导航或向后导航(可跨多级)，Win中的导航方法定义如下：

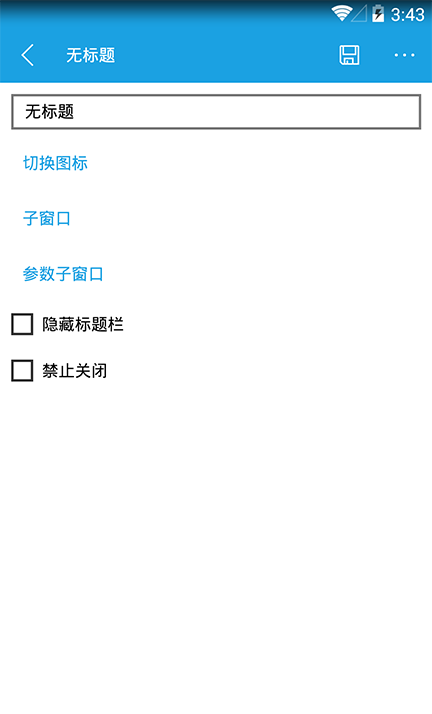


#### 导航窗口NaviWin

NaviWin是导航列表和窗口的组合，用来处理动态多窗口切换的情况。Windows模式采用左右结构，左侧为导航列表，右侧显示当前窗口(PageWin或Win)。如下图所示：



Phone模式时首页显示导航列表，通过导航列表再导航到窗口首页。

** **

#### 打开窗口

共有四种方法：

* 通过菜单打开，AtUI中方法定义

/// <summary>

/// 根据菜单id打开菜单项窗口

/// </summary>

/// <param name="p\_menuID">菜单ID</param>

/// <returns>true 表示打开或激活成功 false表示未找到菜单或视图</returns>

public static bool OpenMenu(string p\_menuID)

/// <summary>

/// 打开菜单项窗口，可以由点击菜单项或直接代码构造Menu的方式调用

/// </summary>

/// <param name="p\_menu">OmMenu实例</param>

/// <returns>true 打开成功</returns>

public static bool OpenMenu(OmMenu p\_menu)

* 通过视图名称打开，AtUI中方法定义

/// <summary>

/// 根据视图名称激活旧窗口或打开新窗口

/// </summary>

/// <param name="p\_viewName">窗口视图名称</param>

/// <param name="p\_title">标题</param>

/// <param name="p\_icon">图标</param>

/// <param name="p\_params">启动参数</param>

/// <returns>true 打开成功</returns>

public static bool OpenView(

string p\_viewName,

string p\_title = null,

Icons p\_icon = Icons.None,

string p\_params = null)

* 根据窗口类型打开，AtUI中方法定义

/// <summary>

/// 根据窗口类型和参数激活旧窗口或打开新窗口

/// </summary>

/// <param name="p\_type">窗口类型</param>

/// <param name="p\_title">标题</param>

/// <param name="p\_icon">图标</param>

/// <param name="p\_params">初始参数</param>

public static IWin OpenWin(

Type p\_type,

string p\_title = null,

Icons p\_icon = Icons.None,

string p\_params = null)

* 通过IWin实例的扩展方法打开

/// <summary>

/// 激活旧窗口或打开新窗口

/// </summary>

/// <param name="p\_win">窗口对象</param>

public static void Open(this IWin p\_win)

### 对话框

Dlg对话框是显示在窗口上层的基本容器，含有标题、工具栏、关闭按钮、内容区域，可指定显示位置，如居中、全屏、相对于窗口中的某元素等，默认情况下点击对话框外部会自动关闭，可以设置IsPinned属性固定对话框，支持异步等待到对话框关闭，提供对话框Closing, Closed事件，Closing事件可控制禁止关闭对话框，Phone模式支持返回键关闭对话框。

显示、关闭对话框方法：

/// <summary>

/// 显示对话框

/// </summary>

public void Show()

/// <summary>

/// 显示对话框，可异步等待到关闭

/// </summary>

/// <returns></returns>

public Task ShowAsync()

/// <summary>

/// 关闭对话框

/// </summary>

public void Close()

**显示位置**定义如下：

/// <summary>

/// 对话框的显示位置

/// </summary>

public enum DlgPlacement

{

/// <summary>

/// 居中显示，windows模式默认

/// </summary>

CenterScreen,

/// <summary>

/// 最大化显示，phone模式默认

/// </summary>

Maximized,

/// <summary>

/// 从左侧弹出

/// </summary>

FromLeft,

/// <summary>

/// 从上面弹出

/// </summary>

FromTop,

/// <summary>

/// 从右侧弹出

/// </summary>

FromRight,

/// <summary>

/// 从下面弹出

/// </summary>

FromBottom,

/// <summary>

/// 左上对齐（对话框的左上角与目标元素的左上角重叠）

/// </summary>

TargetTopLeft,

/// <summary>

/// 右上对齐（对话框的左上角与目标元素的右上角重叠）

/// </summary>

TargetTopRight,

/// <summary>

/// 中心对齐（对话框的中心与目标元素的中心重叠）

/// </summary>

TargetCenter,

/// <summary>

/// 左下对齐（默认，对话框的左上角与目标元素的左下角重叠）

/// </summary>

TargetBottomLeft,

/// <summary>

/// 右下对齐（对话框的左上角与目标元素的右下角重叠）

/// </summary>

TargetBottomRight,

/// <summary>

/// 对话框的右上角与目标元素的左上角重叠

/// </summary>

TargetOuterLeftTop,

/// <summary>

/// 对话框的左下角与目标元素的左上角重叠

/// </summary>

TargetOuterTop

}

### 提示信息

提示面板显示在通知区，windows模式下在右下方，phone模式下在顶部，支持同时显示多个提示信息，可设置自动关闭或手动关闭，分为普通提示信息和警告信息，内容中可包含操作按钮。错误信息不使用提示面板，以对话框方式显示。AtKit中方法定义：

/// <summary>

/// 发布消息提示

/// </summary>

/// <param name="p\_content">显示内容</param>

/// <param name="p\_autoClose">是否自动关闭，默认true</param>

public static void Msg(string p\_content, bool p\_autoClose = true)

/// <summary>

/// 警告提示

/// </summary>

/// <param name="p\_content">显示内容</param>

/// <param name="p\_autoClose">是否自动关闭，默认false</param>

public static void Warn(string p\_content, bool p\_autoClose = false)

/// <summary>

/// 显示错误对话框

/// </summary>

/// <param name="p\_content"></param>

public static async void Error(string p\_content)

### 浮动面板

浮动面板Fly继承自Flyout，用于显示轻量的信息或需要用户交互的内容，内部自动控制浮动面板的最大长宽，因显示位置是Flyout内部处理，可控性差，使用场景较少。

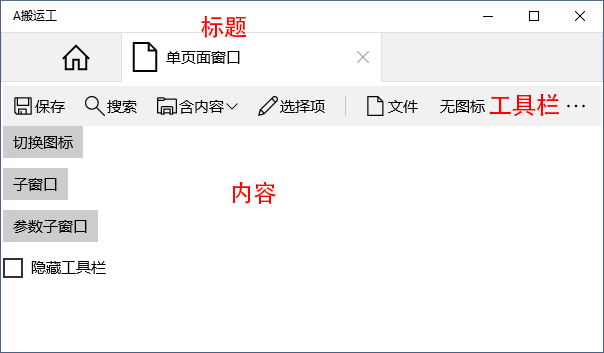
### PhonePage

PhonePage为Phone模式下的内部页面，只作为容器使用，页面内容动态加载，内容类型包括PageWin, Tab, PhoneTabs三种，PageWin为页面窗口，在windows模式下作为窗口，在Phone模式下作为页面内容，提供两种不同的样式，Tab为单个页面内容，PhoneTabs将多个Tab以标签页形式显示。PhonePage不对外公开，只作为页面容器使用。

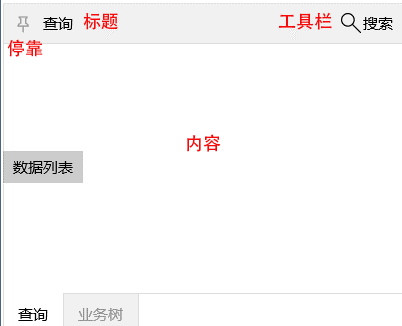
### 内容容器

内容容器有两种：PageWin和Tab，在windows模式下是最小组织单位，在Phone模式下是功能界面和页面容器的桥梁，这两个容器在两种界面模式下分别提供不同的样式。

PageWin的不同样式：



Tab的不同样式：



### 工具栏

工具栏和标题、返回按钮占有同一行，属于内容容器的标准组成部分，工具栏子项类型包含三种：普通按钮、选择按钮、分割线。按钮可以设置图标和文字，工具栏在Phone模式下第一级最多显示两个按钮，只显示图标或文字，通过ShowTextInPhone="True"控制只显示文字。在xaml中可直接设置工具栏内容：

<a:Tab.Bar>

<a:Bar>

<a:Btn Icon="搜索" Text="搜索" />

</a:Bar>

</a:Tab.Bar>

<a:PageWin.Bar>

<a:Bar>

<a:Btn Icon="搜索" Text="搜索" ShowTextInPhone="True"/>

</a:Bar>

</a:PageWin.Bar>



### 标题

Windows模式下窗口标题在任务栏按钮中显示，phone模式下标题在顶部紧靠返回按钮，iOS推荐标题居中显示，因Tab需要二级标题，iOS中未按照此设计，iOS, Android使用统一格式。Xaml中设置标题：

<a:Tab Title="业务树" Icon="饼图">

<a:PageWin Title="业务树" Icon="饼图">

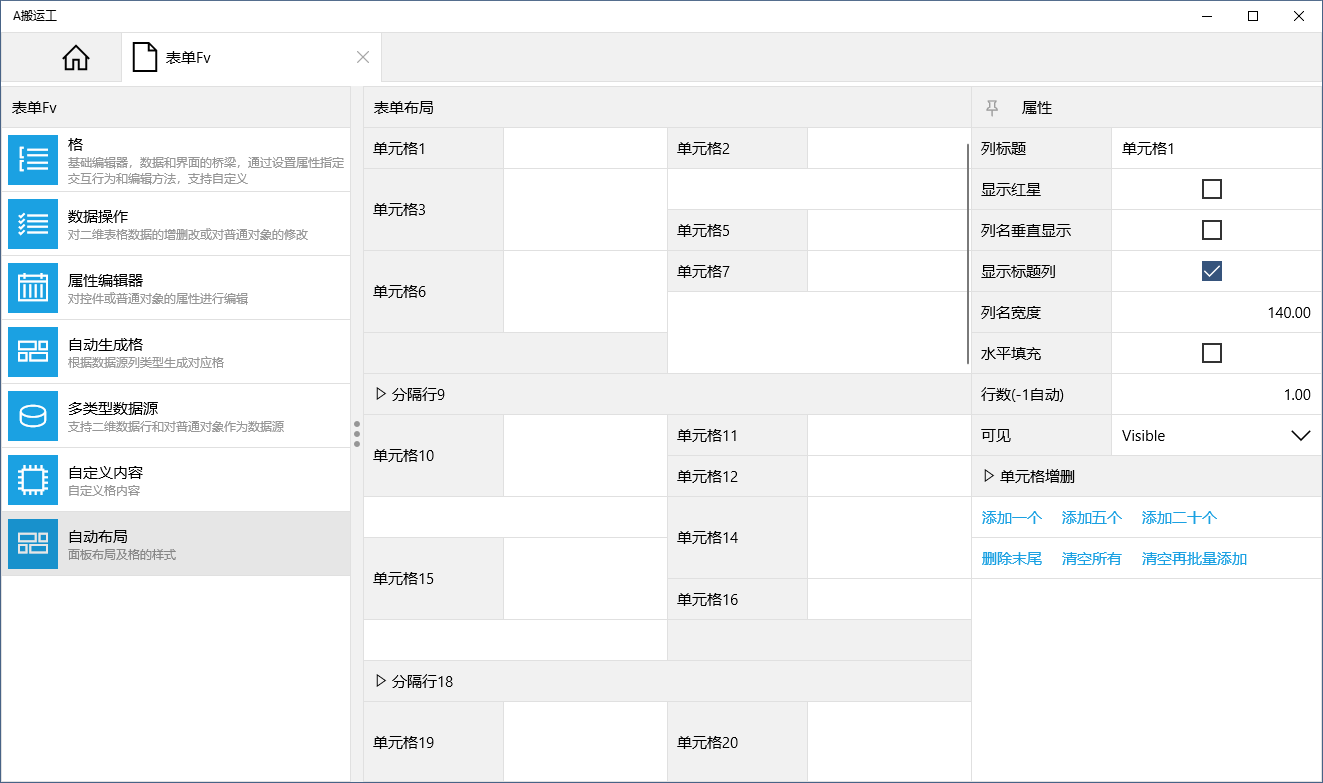
# 数据控件

## Fv

### 布局

为统一两模式下的布局方式，采用自定义布局面板FormPanel，单元格的宽度根据实际区域自动调整，在296-456px之间，高度按行数计算，每行高度41px，内容高度实际40px，通过指定RowSpan来设置行数，当RowSpan为-1时，根据内容高度自动计算所占行数。通过IsHorStretch设置单元格是否水平填充，水平填充时在多列时也独占一行。

布局算法为：根据表单实际区域大小和单元格数量决定每个单元格的位置，优先按列排；当单元格的数量按照一列摆放，未排满高度的80%时，或表单总宽度无法容纳两列时(<296 X 2)时，只摆放一列，phone模式基本一列；能够摆放两列，按照一列摆放超过80%或无法容纳三列时，摆放两列；能够摆放三列，按照两列摆放超过80%或无法容纳四列时，摆放三列；暂时最多支持摆放四列。当表单区域无法摆放所有单元格时，显示垂直滚动栏，始终无水平滚动。



Fv可以作为普通的布局控件使用，为获得单元格样式的控制，通常将普通控件放在CFree内，达到和普通单元格相同的控制，可以设置标题、标题宽度、是否显示标题、是否显示红星、所占行数等，但内嵌的普通控件不参与数据的绑定、编辑等功能，当内嵌控件实现IFreeCell接口后，成为自定义格，作为布局控件使用时如：

<a:Fv>

<a:CBar Title="设置数据源" />

<a:CFree Title="自动行高" RowSpan="-1">

<TextBox AcceptsReturn="True" MinHeight="130" />

</a:CFree>

<Button />

<TextBlock />

</a:Fv>

### 数据源

数据源可以为Row或普通对象，Row时将Cell值绑定到单元格，普通对象时将属性绑定到单元格，支持多级属性，根据属性是否可写控制单元格能否编辑，因此Fv还可作为属性编辑器独立使用。针对不同的数据类型内部提供多种通用格，实现数据编辑。

通过Fv.Data设置数据源，根据单元格ID将对应的数据项绑定到单元格，编辑后新值再同步到数据源，完成数据修改功能。

Fv支持根据数据源自动生成单元格的功能，通过Fv.AutoCreateCell控制，内部根据数据项的数据类型生成对应格，外部无法动态指定，使用场景较少。

在切换数据源时，Fv只处理含ID的单元格，其它单元格怕有影响未设置DataContext！

上级元素或当前元素设置DataContext时，所有绑定中未指明数据源的都使用继承的DataContext，当绑定模式为TwoWay或OneWay时，DataContext必须实现INotifyPropertyChanged接口才能将修改值传递到绑定的UI，DataContext继承DependencyObject的依赖属性也无效，但直接在绑定中指定数据源时依赖属性有效！

数据源为null时，隐藏所有含ID的单元格编辑器，避免误操作。

数据源为普通对象时，已实现和Row相同的功能，包括Fv.Changed事件、FvCell.Changed事件、淡黄色修改状态样式、支持撤销等功能，对普通对象没有特殊要求，内部采用包装方式进行功能扩展(ObjectView,PropertyView)。

列值变化事件Fv.Changed，使用场景较少，Row的所有列值变化都触发，不只是FvCell绑定的列，普通对象只有UI对应的属性变化时触发：

/// <summary>

/// 数据源的列值/属性值修改后事件

/// </summary>

public event EventHandler<ICell> Changed;

FvCell.Changed事件使用比较多，如格之间的联动、引起的动态校验等：

/// <summary>

/// 单元格值修改后事件，参数为新值

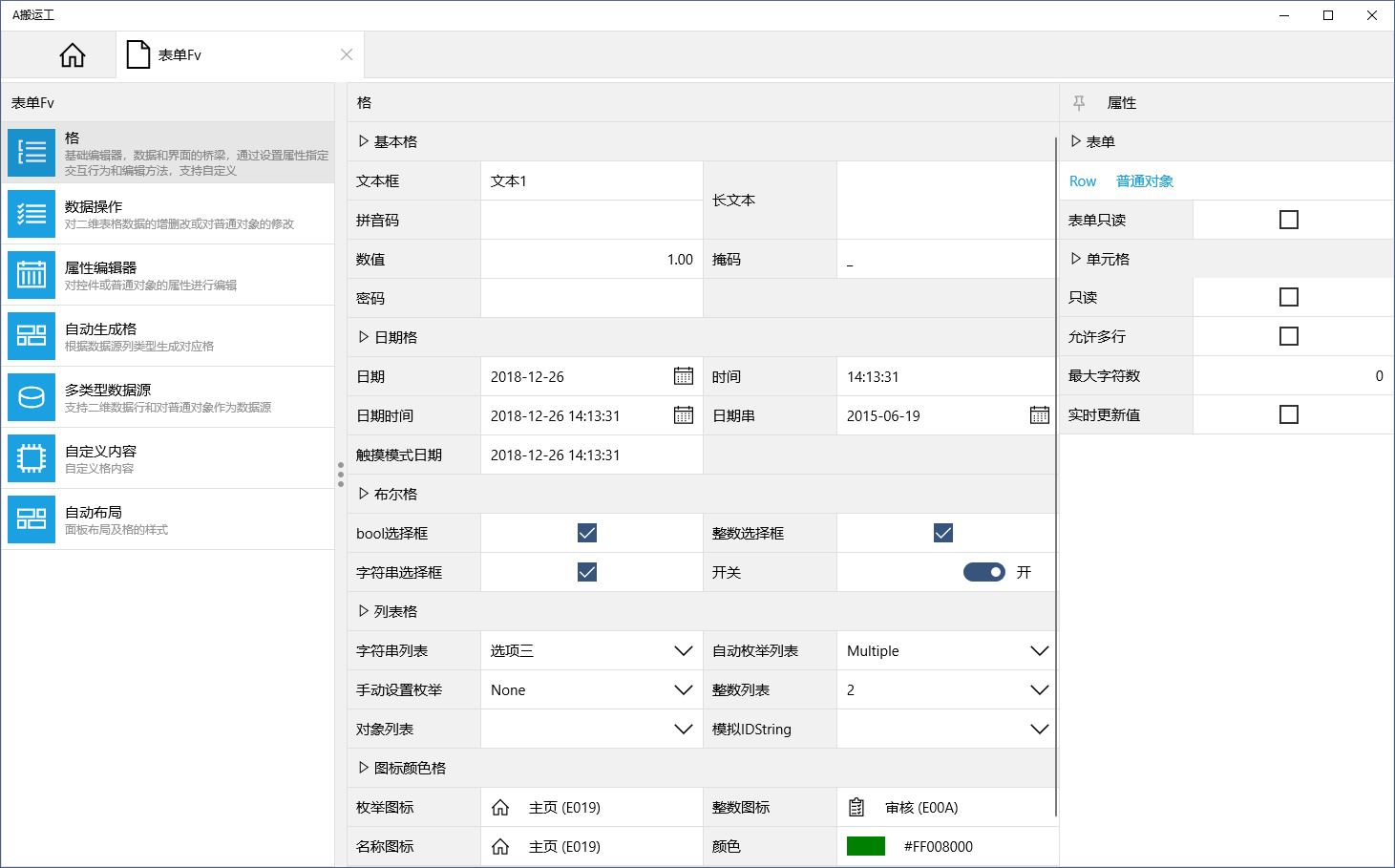
/// </summary>

public event EventHandler<object> Changed;

数据源列值的初始值，主要有以下几种情况：普通默认值、获取服务端的序列值、CList格自动取第一行、内置表达式、单元格cookie值等。

### 格

格是表单编辑的最小单位，负责数据的显示、交互编辑、新值同步到数据源等功能，格之间可能存在联动关系，内部提供多种格：普通文本格、数值格、掩码格、布尔格、下拉选择格、时间格、图标选择格、颜色选择格、文件格、图像格、自定义格等，实现常见的编辑功能。如下图：



格的基类为FvCell，实现公共样式和功能，常用属性如下：

/// <summary>

/// 获取设置列名(字段名或属性名)

/// </summary>

public string ID

/// <summary>

/// 获取设置列标题

/// </summary>

public string Title

/// <summary>

/// 获取设置列名的宽度

/// </summary>

public double TitleWidth

/// <summary>

/// 获取设置是否显示标题列

/// </summary>

public bool ShowTitle

/// <summary>

/// 获取设置是否显示红星

/// </summary>

public bool ShowStar

/// <summary>

/// 获取设置是否垂直显示标题

/// </summary>

public bool IsVerticalTitle

/// <summary>

/// 获取设置单元格是否水平填充，默认false

/// </summary>

public bool IsHorStretch

/// <summary>

/// 获取设置占用的行数，默认1行，-1时自动行高

/// </summary>

public int RowSpan

/// <summary>

/// 获取设置列是否只读，最终是否只读由ReadOnlyBinding确定！

/// </summary>

public bool IsReadOnly

/// <summary>

/// 获取设置是否自动保存单元格最后一次编辑值，默认False

/// </summary>

public bool AutoCookie

/// <summary>

/// 获取设置占位符文本

/// </summary>

public string Placeholder

每种类型的格有自己的行为控制，格变量通常写在xaml中，

## Lv

Lv为列表控件，无编辑功能，常用属性有：



### 视图类型

Lv.ViewMode提供三种视图类型：表格、列表、磁贴，默认为List。



Lv.View为行/项目模板，支持DataTemplate、DataTemplateSelector 或 Cols列定义，是Lv的默认xaml内容。

表格的定义：

<a:Lv x:Name="\_lv" ViewMode="Table">

<a:Cols>

<a:Col ID="xm" Title="姓名" Width="120" />

<a:Col ID="bh" Title="编号" Width="80" />

<a:Col ID="chushengrq" Title="出生日期" Width="120" />

</a:Cols>

</a:Lv>

其中，Cols, Col属性包括：





列表的行模板定义：

<a:Lv x:Name="\_lv">

<DataTemplate>

<StackPanel Orientation="Horizontal">

<a:Dot ID="xb" />

<a:Dot ID="chushengrq" Margin="20,0,0,0" />

</StackPanel>

</DataTemplate>

</a:Lv>

模板内的Dot元素继承自ContentPresenter，和Col相同，ID用于指定列名或属性名，可以设置Dot的Foreground,FontWeight,FontStyle,FontSize属性控制显示内容的样式，优先级高于外部的视图扩展。

当ViewMode为List且提供Cols定义时显示为表单列表。

磁贴项模板和列表行模板相同，主要设置ViewMode为Tile即可。

### 数据源

通过Lv.Data属性获取设置数据源对象，类型为IList，内部特殊处理Table类型的数据源；也可通过Lv.PageData设置分页数据源对象。其数据显示过程为：Data(源) -> ILvDataView(排序、过滤、分组处理) -> ViewItem(数据+Style+UI) -> LvRow (内容布局、数据绑定)。所有数据源类型都支持动态设置排序、过滤、分组操作，其属性如下：

/// <summary>

/// 获取设置过滤回调

/// </summary>

public Predicate<object> Filter

/// <summary>

/// 获取设置分组列名

/// </summary>

public string GroupName

/// <summary>

/// 获取设置排序条件

/// </summary>

public SortDescription SortDesc

Lv通过设置AutoCreateCol属性能够根据数据源自动生成列，使用场景较少。

Lv.PageData分页数据源类型为PageData，通过NextPage回调方法或Sql获取下一页数据，获取新页数据的优先级为：NextPage > Sql > SqlStr， InsertTop属性控制新数据是否插入头部(默认插入尾部)，调用PageData.LoadPageData()方法加载新页数据，如：

\_lv.PageData = new PageData { NextPage = OnNextPage, InsertTop = true };

void OnNextPage(PageData p\_pd)

{

IList ls = SampleData.CreatePersonsTbl(p\_pd.PageSize);

p\_pd.LoadPageData(ls);

}

PageData属性：

/// <summary>

/// 获取设置sql字典键值，语句中不需包含分页功能

/// </summary>

public string Sql

/// <summary>

/// 获取设置sql查询语句，语句中不需包含分页功能

/// </summary>

public string SqlStr

/// <summary>

/// 获取设置sql查询参数值字典

/// </summary>

public Dict SqlParams

/// <summary>

/// 获取设置进入下一页面的回调方法

/// </summary>

public Action<PageData> NextPage

/// <summary>

/// 获取设置页面行数

/// </summary>

public int PageSize

/// <summary>

/// 获取设置是否还有未加载的页面

/// </summary>

public bool HasMorePages

/// <summary>

/// 获取设置是否将新页面数据插入到头部

/// </summary>

public bool InsertTop

/// <summary>

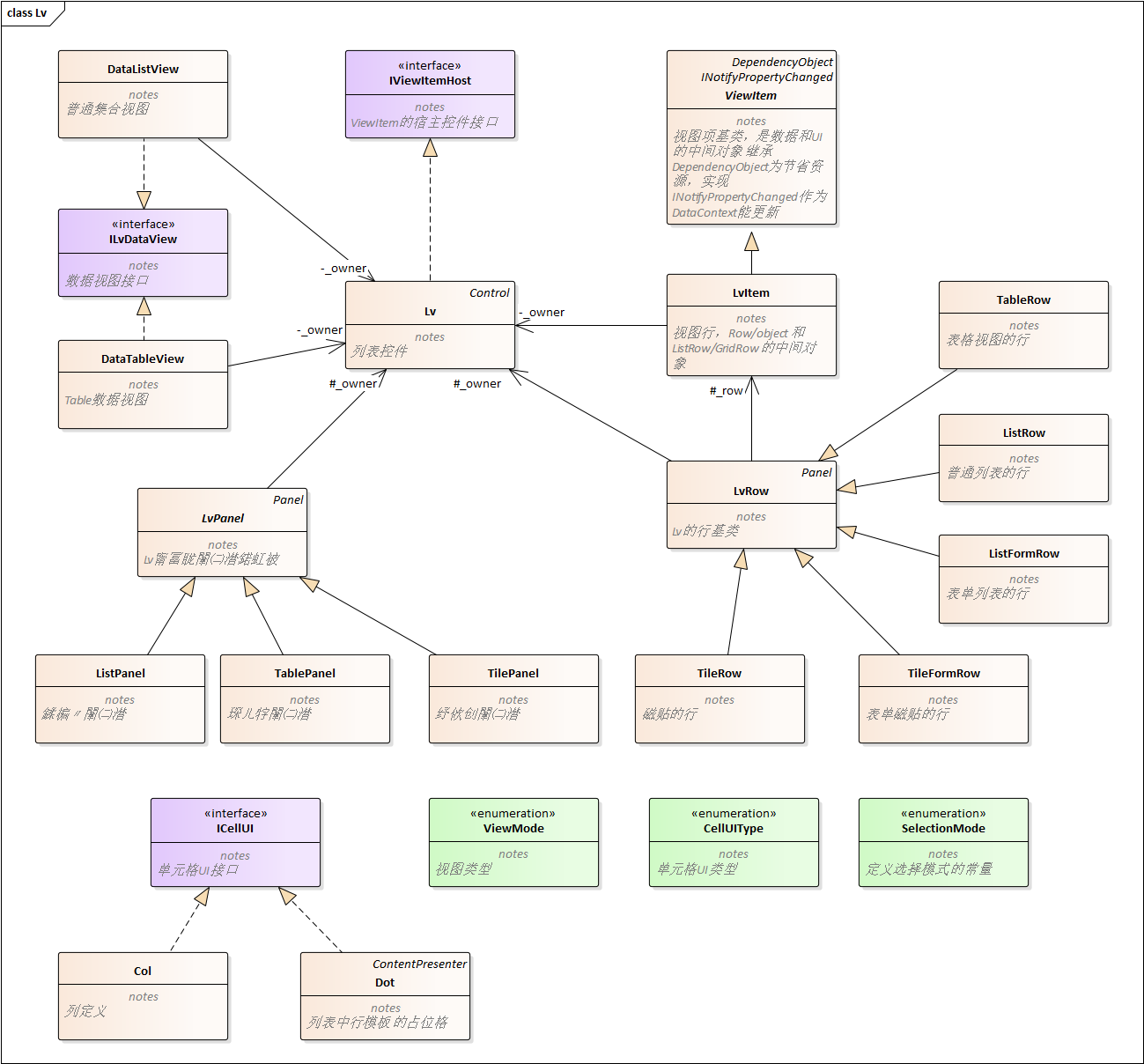
/// 获取当前页序号

/// </summary>

public int PageNo

### 定义UI

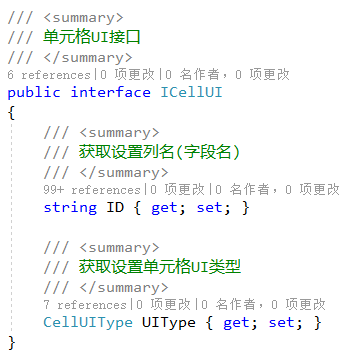
Lv可视树结构：Lv -> LvPanel -> LvRow -> ContentPresenter/Dot -> 单元格UI，LvPanel 负责LvRow的生成和布局，LvRow负责单元格UI的生成和布局，针对不同的视图类型有不同的实现方式，单元格UI作为Lv最小粒度的UI可以通过以下两种方式控制。



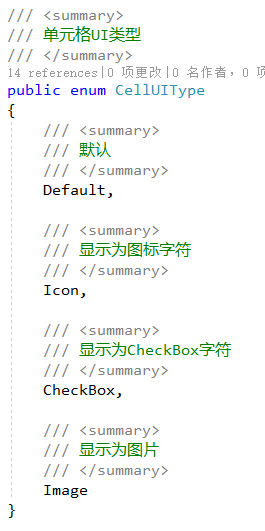
#### 直接定义

直接定义一般用来指定某列显示的固定UI类型，如照片列统一显示为图片、已婚列以checkbox方式显示、图标列以内置字符显示等等，它只是数据到UI类型的固定转换，无法实现根据数据内容的不同而显示不同UI类型或样式，该需求可通过视图扩展实现。

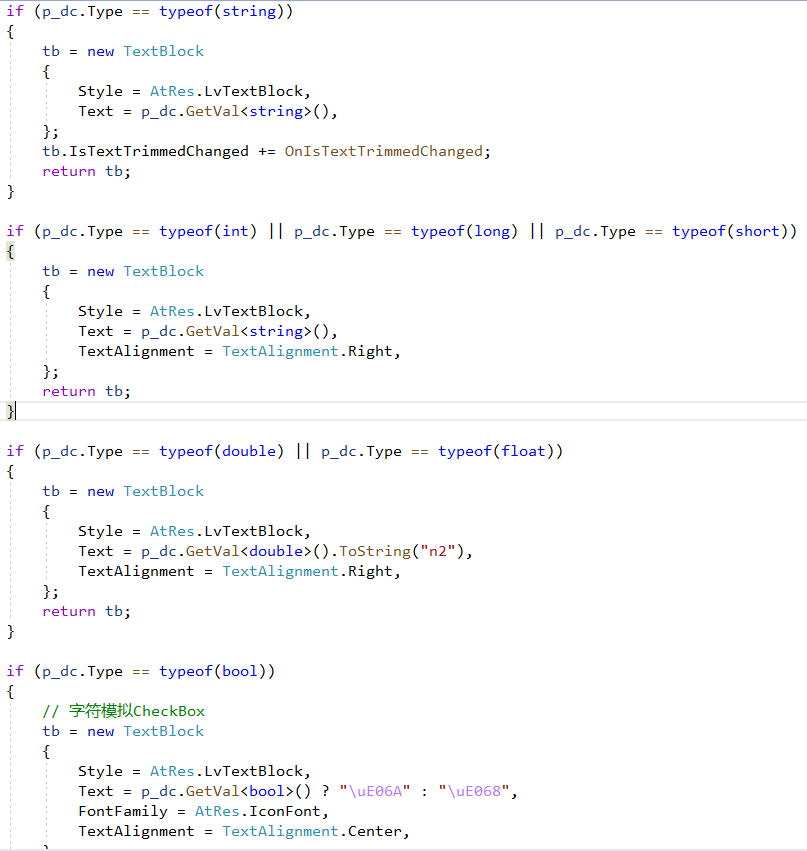
直接定义在表格模式时通过Col定义，列表和磁贴模式通过Dot定义，Col和Dot都实现接口ICellUI，除了指定列名外还可以设置单元格UI类型，定义如下：



单元格UI类型目前包括以下，后续会增加：



当为Default时，系统会根据实际的数据类型生成对应的UI，如bool显示为CheckBox，Icons显示为图标，int右对齐，double显示两位小数并右对齐，Color显示为颜色填充的矩形框等等，一般能满足常用需求，如下：





当有特殊需求时，如数据是字符串类型，为图片的地址，默认方式只会按普通字符串显示，要想显示为图片需指定UIType为Image，CellUIType支持的类型会随着应用逐渐增加。

#### 视图扩展

视图扩展是Lv实现最小粒度自定义UI的唯一方式，需要外部提供视图扩展的类型，在运行时调用其静态方法获取要显示的内容或样式。属性定义如下：



静态方法中除SetStyle外其余方法名和列名对应，SetStyle方法用来控制行样式，如Foreground, Background,FontWeight,FontStyle,FontSize。

所有方法输入参数为ViewItem，返回参数可以为UIElement, object，返回普通对象时输出ToString()后的TextBlock；内部对每个格的UI缓存复用，只第一次显示时或数据改变时构造，不显示或重复加载时不构造。外部扩展类型如：

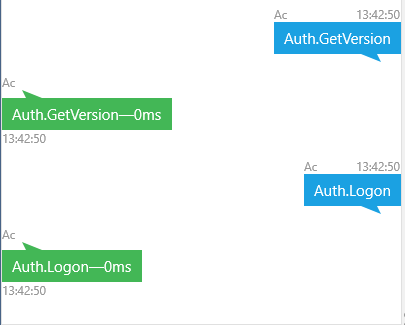


ViewItem视图行是Row/object 和 ListRow/GridRow 的中间对象，是数据和UI的桥梁，常用属性：



### 行模板选择器

当未设置视图类型时，可以设置行模板选择器实现不同条件下的行内容和行样式，是粗粒度的自定义UI，布局时不采用虚拟行，如聊天的对话目录。



/// <summary>

/// 获取设置行模板选择器

/// </summary>

public DataTemplateSelector ViewSelector

public class TraceItemSelector : DataTemplateSelector

{

public DataTemplate Normal { get; set; }

public DataTemplate Call { get; set; }

protected override DataTemplate SelectTemplateCore(object item)

{

switch ((TraceOutType)((Row)item).Int("type"))

{

case TraceOutType.RpcCall:

case TraceOutType.WsCall:

return Call;

default:

return Normal;

}

}

}

### 选择模式

Lv. SelectionMode选择模式共三种：None,Single,Multiple，默认单选Single，可以动态切换，多选状态时行头显示选择框，None时无选择样式。

单选模式时常用ItemClick事件，每次点击行时都触发，通过参数ItemClickArgs.IsChanged判断当前点击行是否和上次点击行相同，如导航列表中常用ItemClick事件，事件定义：

/// <summary>

/// 单击行事件

/// </summary>

public event EventHandler<ItemClickArgs> ItemClick;

/// <summary>

/// 单击行事件参数

/// </summary>

public class ItemClickArgs : EventArgs

{

/// <summary>

/// 当前点击行是否和上次点击行相同

/// </summary>

public bool IsChanged

/// <summary>

/// 当前点击行

/// </summary>

public object Data { get; }

/// <summary>

/// 上次点击行

/// </summary>

public object OldData { get; }

/// <summary>

/// 当前点击Row

/// </summary>

public Row Row

}

/// <summary>

/// 选择变化事件

/// </summary>

public event SelectionChangedEventHandler SelectionChanged;

属性HasSelected判断当前是否有选择行，该属性常用来绑定到有选择行时的操作，如删除：

IsEnabled="{Binding HasSelected,ElementName=\_lv}"

和选择相关的属性包括：

/// <summary>

/// 获取是否有选择行

/// </summary>

public bool HasSelected

/// <summary>

/// 获取当前选择的行列表

/// </summary>

public IList<object> SelectedItems

/// <summary>

/// 获取设置当前选定行，设置null时清空选择，多选时为返回最后选择行

/// </summary>

public object SelectedItem

/// <summary>

/// 获取设置选定行的索引，-1无选定行，设置-1清空选择

/// </summary>

public int SelectedIndex

/// <summary>

/// 获取当前选定行Row，为SelectedItem转型，方便使用

/// </summary>

public Row SelectedRow

Lvx包含Lv的附加依赖项属性，MultiSelect指定双向绑定 Lv.SelectionMode <=> ToggleButton.IsChecked，用法如：<a:ToggleBtn Icon="详细" Text="多选" a:Lvx.MultiSelect="{Binding ElementName=\_lv}" />

### 分组

所有视图类型都支持分组，效果如下图所示，通过ShowGroupHeader属性控制显示分组导航。

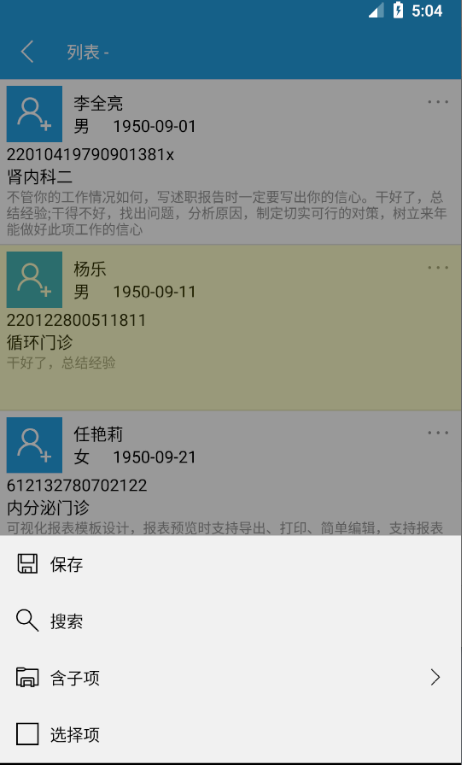


### 上下文菜单

上下文菜单定义如：



触发上下文菜单的方式：windows系统为鼠标右键，phone模式通过点击标识按钮。



### 建议

列表控件Lv采用虚拟行和真实行两种模式进行布局，当未采用行模板选择器且不自动行高时使用虚拟行模式，此方式节省资源、交互流畅，但行高固定(以第一行为标准)、行模板相同，在业务需求允许的情况下尽可能使用虚拟行模式。



LvPanel类包含整个布局算法，windows模式和Phone模式同样支持，分组行始终以真实行绘制，当虚拟行、真实行混排又支持多种视图类型时算法较复杂。

当Lv为列表或磁贴模式时，行模板内尽可能使用Dot显示数据，Dot作为行模板的占位格(继承自ContentPresenter)，具有写法简洁、能根据数据类型生成可视元素、缓存可视元素、支持视图扩展等优势，可参见”定义UI”部分，当无法使用Dot时，可以采用如下的绑定方式：

<Ellipse Visibility="{Binding [unread], Converter={StaticResource CvBoolToVisible}}" Fill="Red" Height="8" Width="8" />

## Tv

# 基础控件

## 属性编辑器

# 基础功能

## Table操作

### 创建Table

有以下几种方式创建Table：手动创建、由表名创建、由本地库表名创建、复制表结构、复制行结构。

**手动构造Table**，创建时直接给出列结构，string 类型可以省略，推荐使用初始化器的简单写法：

Table tbl = new Table

{

{ "id" },

{ "bh", typeof(int) },

{ "chushengrq", typeof(DateTime) },

{ "hunfou", typeof(bool) },

{ "shengao", typeof(double) },

{ "bumen", typeof(TargetSystem) },

};

**由表名创建：**

Table tbl = Table.Create("dt\_menu");

**由本地库表名(状态库中)创建：**

Table tbl = Table.CreateLocal("ClientLog");

**复制表结构：**

Table tbl = Table.Create(tgtTbl);

**复制行结构：**

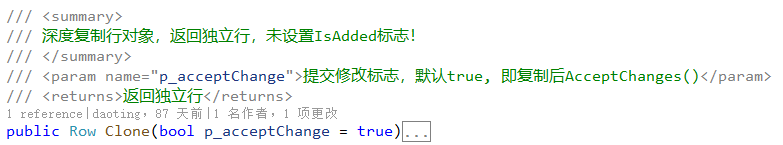
Table tbl = Table.Create(tgtRow);

### Row操作

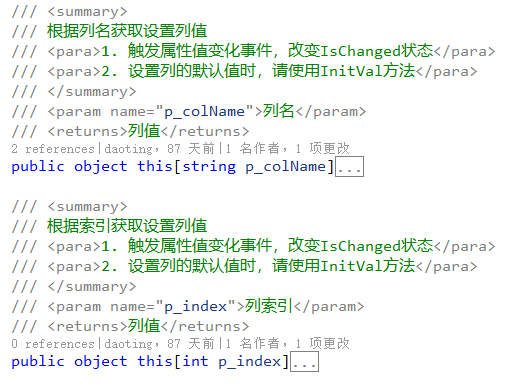
**创建新行**



克隆行



按列名或列索引设置Cell值



另一种设置列默认值：



### Column操作

增加列或删除列时，内部会同步所有Row的单元格，增加列时会为所有行增加新单元格，删除列时同步删除所有行的对应单元格，方法为：



## RPC

RPC（Remote Procedure Call）全称为远程过程调用，是客户端与服务之间进行网络通信的基础。系统采用http2协议、使用Post方法、传输json内容的方式调用服务端的Api，返回json格式的结果，详细描述参见《搬运工服务端手册》。

客户端Rpc代理方法为静态方法，由服务端Api自动生成，主要涉及两种模式的调用：

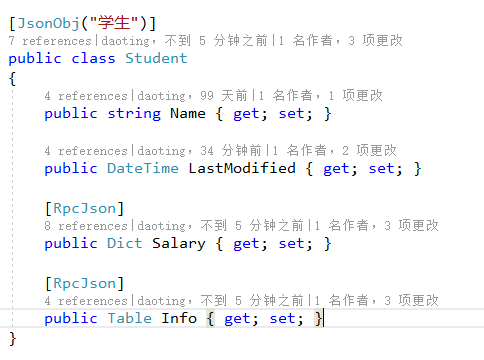


UnaryRpc基于Http2的请求/响应模式的远程调用，ServerStreamRpc客户端发送一个请求，服务端返回数据流响应的远程调用，实现服务器推送功能，替代websocket方式。

基础通信使用HttpClient，不同平台使用不同Handler类，UWP使用默认HttpClientHandler，Android,iOS使用自定义的NativeMessageHandler封装对本地平台的调用，Android内部使用第三方OkHttp3，iOS内部使用原生NSUrlSession，所有服务端证书始终有效。

## 序列化

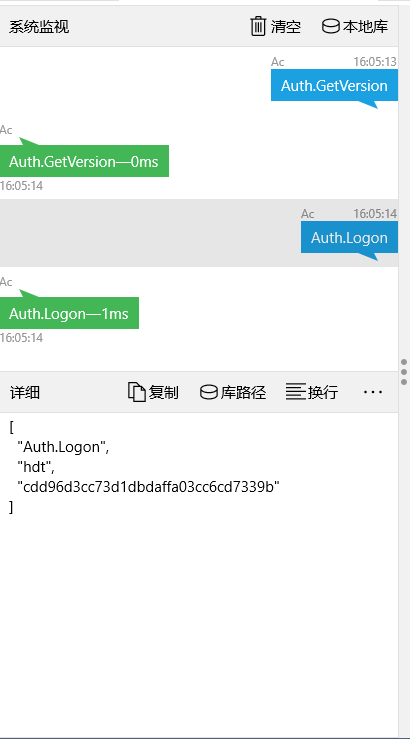
Rpc的内容为json格式，两端支持的序列化类型相同，除支持常用的简单类型外，还包括Table, Row, Dict等类型，以及这些类型的List<T>集合和互相嵌套，基本涵盖常用场景，List<object>和Dict中值的类型为object，虽然json中无法区分数值类型、时间类型、字节数组，但两端在传输时已记录类型，所以两端类型始终相同，同时也支持自定义序列化类型，如：



Json内容的数据结构参见Dt.Core项目的《JsonRpc结构.txt》文件

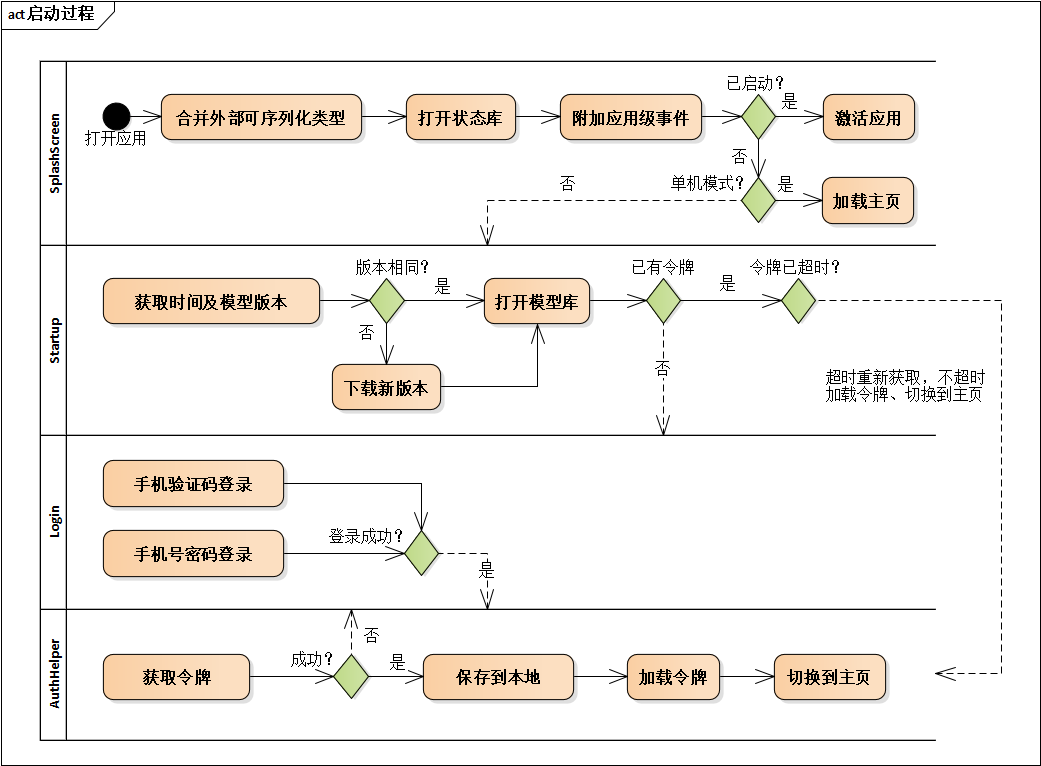
## 系统监控

windows模式快捷键Alt + →或phone模式点击标题栏菜单，调出系统监控：



系统监控功能包括对服务器的Rpc调用json、返回json、本地库操作、常用工具等。

## 启动过程



## 数据访问

## 本地存储

Sqlite

## 参数管理

AtParam

## 当前用户

AtUser