Navigation

1. 工程配置文件module.json5中配置 {"routerMap": "$profile:route_map"}

2. route_map中 必须要有一个entry的page

```
{
  "name": "pageOne",//页面name 用于跳转
  "pageSourceFile": "src/main/ets/pages/Page1.ets",//页面位置
  "buildFunction": "Page1Builder",//声明的全局的方法创建页面
  "data": {
    "description": "this is pageOne"//传递的参数
  }
}
```

3.

跳转和携带参数

```
this.pageInfos.pushPath({name:'pageOne',param:'this is page1'})

```

获取参数

```
console.log('--------------------')
console.log('获取栈中所有NavDestination页面的名称',
JSON.stringify(this.pageInfos.getAllPathName()))
console.log('获取index指定的NavDestination页面的参数信息',
  JSON.stringify(this.pageInfos.getParamByIndex(0)))
console.log('获取全部名为name的NavDestination页面的位置索引',
  JSON.stringify(this.pageInfos.getIndexByName('pageOne')))
console.log('获取栈大小', JSON.stringify(this.pageInfos.size()))
```

```
pageInfos: NavPathStack = new NavPathStack()
```

route的info可以通过下面获取打印

```
context.getConfigInRouteMap()
console.log("current page config info is " +
JSON.stringify(context.getConfigInRouteMap()))
```

1.线性布局

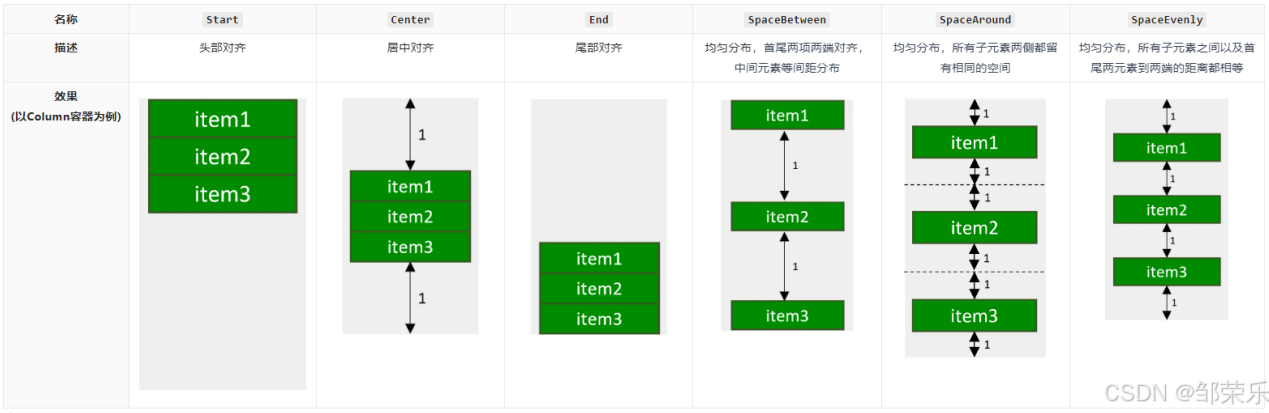
线性布局（LinearLayout）是开发中最常用的布局，通过线性容器Row和Column构建。线性布局是其他布局的基础，其子元素在线性方向上（水平方向和垂直方向）依次排列。线性布局的排列方向由所选容器组件决定，Column容器内子元素按照垂直方向排列，Row容器内子元素按照水平方向排列。根据不同的排列方向，开发者可选择使用Row或Column容器创建线性布局。

1、space属性

Column和Row容器的参数类型为{space?: string | number}，开发者可通过space属性调整子元素在主轴方向上的间距

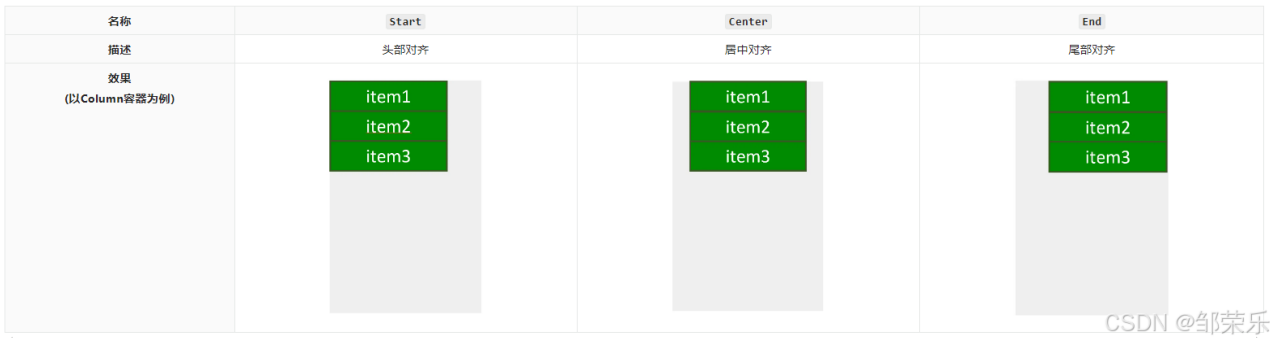
2、justifyContent属性

子元素沿主轴方向的排列方式可以通过justifyContent()方法进行设置，其参数类型为枚举类型FlexAlign，可选的枚举值有



3、alignItems属性

子元素沿交叉轴方向的对齐方式可以通过alignItems()方法进行设置，其参数类型，对于Column容器来讲是HorizontalAlign，对于Row容器来讲是VerticalAlign，两者都是枚举类型，可选择枚举值也都相同，具体内容如下



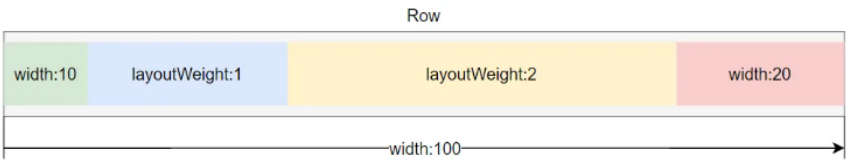
实用技巧

1、Blank组件的使用

Blank可作为Column和Row容器的子组件，该组件不显示任何内容，并且会始终充满容器主轴方向上的剩余空间，效果如下

2、layoutWeight属性的使用

layoutWeight属性可用于Column和Row容器的子组件，其作用是配置子组件在主轴方向上的尺寸权重。配置该属性后，子组件沿主轴方向的尺寸设置（width或height）将失效，具体尺寸根据权重进行计算，计算规则如下图所示：



Stack

层叠布局（StackLayout）用于在屏幕上预留一块区域来显示组件中的元素，提供元素可以重叠的布局。层叠布局通过Stack容器组件实现位置的固定定位与层叠，容器中的子元素（子组件）依次入栈，后一个子元素覆盖前一个子元素，子元素可以叠加，也可以设置位置。

Stack作为容器，容器内的子元素（子组件）的顺序为Item1->Item2->Item3。

Z序控制

Stack容器中兄弟组件显示层级关系可以通过Z序控制的zIndex属性改变。zIndex值越大，显示层级越高，即zIndex值大的组件会覆盖在zIndex值小的组件上方

对齐方式

Stack组件通过alignContent参数实现位置的相对移动。支持九种对齐方式。

弹性布局 (Flex)

FlexDirection.Row（默认值）：主轴为水平方向，子元素从起始端沿着水平方向开始排布

FlexDirection.RowReverse：主轴为水平方向，子元素从终点端沿着FlexDirection. Row相反的方向开始排布。

FlexDirection.Column：主轴为垂直方向，子元素从起始端沿着垂直方向开始排布。

FlexDirection.ColumnReverse：主轴为垂直方向，子元素从终点端沿着FlexDirection. Column相反的方向开始排布。

```
Flex({ direction: FlexDirection.Row }) {
  Text('1').width('33%').height(50).backgroundColor(0xF5DEB3)
  Text('2').width('33%').height(50).backgroundColor(0xD2B48C)
  Text('3').width('33%').height(50).backgroundColor(0xF5DEB3)
}
```

布局换行

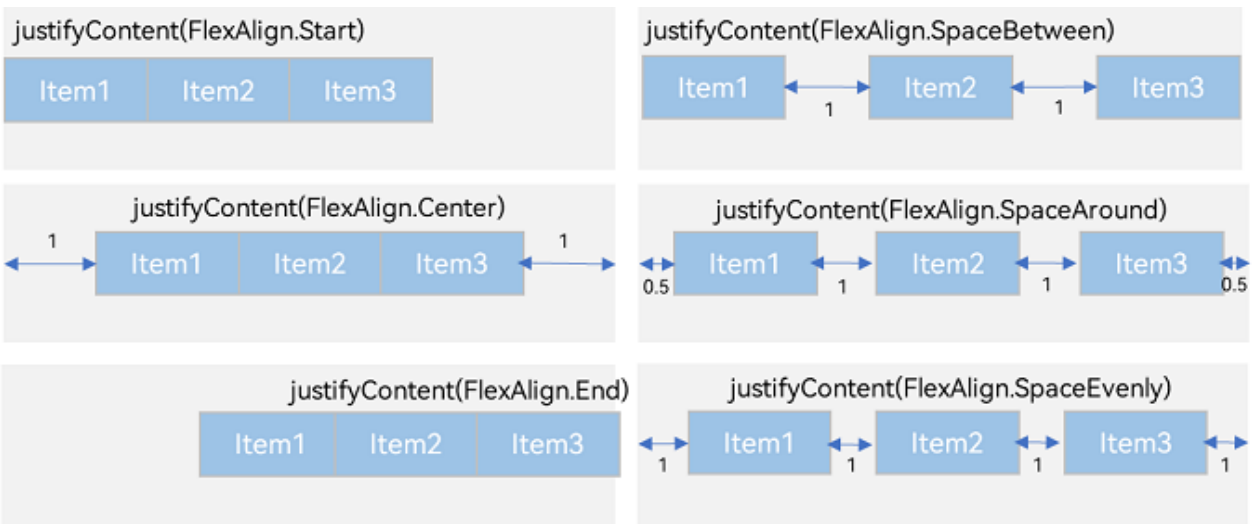
FlexWrap. NoWrap（默认值）：不换行。如果子元素的宽度总和大于父元素的宽度，则子元素会被压缩宽度。

FlexWrap. Wrap：换行，每一行子元素按照主轴方向排列。

```
Flex({ direction: FlexDirection.RowReverse }) {
  Text('1').width('33%').height(50).backgroundColor(0xF5DEB3)
  Text('2').width('33%').height(50).backgroundColor(0xD2B48C)
  Text('3').width('33%').height(50).backgroundColor(0xF5DEB3)
}
.height(70)
.width('90%')
.padding(10)
.backgroundColor(0xAFEEEE)
```

主轴对齐方式

通过justifyContent参数设置子元素在主轴方向的对齐方式。FlexAlign



List组件

List是很常用的滚动类容器组件，一般和子组件ListItem一起使用，List列表中的每一个列表项对应一个ListItem组件。

在List组件中按垂直或者水平方向线性排列子组件ListItemGroup或ListItem

ListItemGroup用于列表数据的分组展示，其子组件也是ListItem。ListItem表示单个列表项，可以包含单个子组件。List组件的sticky属性配合ListItemGroup组件使用，用于设置ListItemGroup中的头部组件是否呈现吸顶效果或者尾部组件是否呈现吸底效果。

List组件里面的列表项默认是按垂直方向排列的，如果您想让列表沿水平方向排列，您可以将List组件的listDirection属性设置为Axis.Horizontal。

布局

List除了提供垂直和水平布局能力、超出屏幕时可以滚动的自适应延伸能力之外，还提供了自适应交叉轴方向上排列个数的布局能力

Grid和WaterFlow也可以实现单列、多列布局，如果布局每列等宽，且不需要跨行跨列布局，相比Gird和WaterFlow，则更推荐使用List。

设置List排列方向

①Vertical（默认值）：子组件ListItem在List容器组件中呈纵向排列。

②Horizontal：子组件ListItem在List容器组件中呈横向排列。

使用ForEach渲染列表

列表往往由多个列表项组成，所以我们需要在List组件中使用多个ListItem组件来构建列表，这就会导致代码的冗余。使用循环渲染（ForEach）遍历数组的方式构建列表，可以减少重复代码

设置列表分割线

List组件子组件ListItem之间默认是没有分割线的，部分场景子组件ListItem间需要设置分割线，这时候您可以使用List组件的divider属性。divider属性包含四个参数：

strokeWidth: 分割线的线宽。

color: 分割线的颜色。

startMargin：分割线距离列表侧边起始端的距离。

endMargin: 分割线距离列表侧边结束端的距离。

List列表滚动事件监听

List组件提供了一系列事件方法用来监听列表的滚动，您可以根据需要，监听这些事件来做一些操作：

onScroll：列表滑动时触发，返回值scrollOffset为滑动偏移量，scrollState为当前滑动状态。

onScrollIndex：列表滑动时触发，返回值分别为滑动起始位置索引值与滑动结束位置索引值。

onReachStart：列表到达起始位置时触发。

onReachEnd：列表到底末尾位置时触发。

onScrollStop：列表滑动停止时触发。

长列表的处理/LazyForEach

循环渲染适用于短列表，当构建具有大量列表项的长列表时，如果直接采用循环渲染方式，会一次性加载所有的列表元素，会导致页面启动时间过长，影响用户体验。因此，推荐使用数据懒加载（LazyForEach）方式实现按需迭代加载数据，从而提升列表性能。

框架会根据滚动容器可视区域按需创建组件，当组件滑出可视区域外时，框架会进行组件销毁回收以降低内存占用。

LazyForEach必须和@Reusable装饰器一起使用才能触发节点复用。使用方法：将@Reusable装饰在LazyForEach列表的组件上，见使用规则。

数据源封装

数据源需要自己封装成一个类并实现IDataSource接口，重写必备的四个方法。

```
export class datamodel implements IDataSource{
  totalCount(): number { //数据源数据个数
    throw new Error("Method not implemented.");
  }

  getData(index: number) {//根据下标获取数据
    throw new Error("Method not implemented.");
  }

  registerDataChangeListener(listener: DataChangeListener): void {//注册数据改变监听
    throw new Error("Method not implemented.");
  }

  unregisterDataChangeListener(listener: DataChangeListener): void {//解绑数据改变监听
    throw new Error("Method not implemented.");
  }

}

```

LazyForEach必须在容器组件内使用，仅有List、Grid、Swiper以及WaterFlow组件支持数据懒加载缓存前后3个组件

```
List() {
  // ...
}.cachedCount(3)
```

Tabs组件

ArkUI开发框架提供了一种页签容器组件Tabs，开发者通过Tabs组件可以很容易的实现内容视图的切换。页签容器Tabs的形式多种多样，不同的页面设计页签不一样，可以把页签设置在底部、顶部或者侧边。

Tabs组件的页面组成包含两个部分，分别是TabContent和TabBar。TabContent是内容页

Tabs的布局模式有Fixed（默认）和Scrollable两种：

- BarMode.Fixed：所有TabBar平均分配barWidth宽度（纵向时平均分配barHeight高度）,页签不可滚动，效果图如下：
- BarMode.Scrollable：每一个TabBar均使用实际布局宽度，超过总长度（横向Tabs的barWidth，纵向Tabs的barHeight）后可滑动。当页签比较多的时候，可以滑动页签

自定义tabbar

切换至指定页签/内容页和页签联动

Tabs提供的onChange事件方法，监听索引index的变化，并将当前活跃的index值传递给currentIndex从而实现页签的切换

也可以使用TabsController，TabsController是Tabs组件的控制器，用于控制Tabs组件进行内容页切换。通过TabsController的changeIndex方法来实现跳转至指定索引值对应的TabContent内容。

开发者可以通过Tabs组件的onContentWillChange接口，设置自定义拦截回调函数。拦截回调函数在下一个页面即将展示时被调用，如果回调返回true，新页面可以展示；如果回调返回false，新页面不会展示，仍显示原来页面。

SubTabBarStyle

子页签样式。

```
SubTabBarStyle.of('开始【单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断单行省略号截断】结束')
          .labelStyle({
            overflow: TextOverflow.Ellipsis,
            maxLines: 1,
            minFontSize: 10,
            heightAdaptivePolicy: TextHeightAdaptivePolicy.MAX_LINES_FIRST,
            font: { size: 20 }
          })
```

创建网格 (Grid/GridItem)

设置行列数量与占比

通过设置行列数量与尺寸占比可以确定网格布局的整体排列方式。Grid组件提供了rowsTemplate和columnsTemplate属性用于设置网格布局行列数量与尺寸占比。

rowGap) 设置分组的行间距

rowsTemplate和columnsTemplate属性用于设置网格布局行列数量与尺寸占比。

在网格中，可以通过onGetRectByIndex返回的[rowStart,columnStart,rowSpan,columnSpan]来实现跨行跨列布局，其中rowStart和columnStart属性表示指定当前元素起始行号和起始列号，rowSpan和columnSpan属性表示指定当前元素的占用行数和占用列数。

设置主轴方向

使用Grid构建网格布局时，若没有设置行列数量与占比，可以通过layoutDirection设置网格布局的主轴方向，决定子组件的排列方式。此时可以结合minCount和maxCount属性来约束主轴方向上的网格数量。

构建可滚动的网格布局

可滚动的网格布局常用在文件管理、购物或视频列表等页面中，如下图所示。在设置Grid的行列数量与占比时，如果仅设置行、列数量与占比中的一个，即仅设置rowsTemplate或仅设置columnsTemplate属性，网格单元按照设置的方向排列，超出Grid显示区域后，Grid拥有可滚动能力。

控制滚动位置

Grid组件初始化时，可以绑定一个Scroller对象，用于进行滚动控制，例如通过Scroller对象的scrollPage方法进行翻页。