一、按子Widget数量分类

1、单子Widget布局

即布局中只含有一个子Widget，通常使用child属性来表示

例如：

```
Center（
child:Text('Hello World'),
）
```

2、多子Widget布局

即布局中会含有多个子Widget,通常用children来表示

例如：Column、Row

```
Column(
 children:<Widget>[
 Text(''),
 Text(''),
 ...
 ]
)
```

二、按照布局方式分类

按照子元素排布的方式分为：

- 弹性布局Widget
- 线性布局Widget
- 流式布局Widget
- 层叠布局Widget

1、弹性布局Flex

弹性布局Flex有主轴和交叉轴，子widget默认按照主轴排序。之所以称为弹性，是因为结合Expanded可以实现Widget按照一定的比例来分类父容器空间。

（1）Flex的构造方法

```
class Flex extends MultiChildRenderObjectWidget {
 Flex({
 Key key,
 @required this.direction,
 this.mainAxisAlignment = MainAxisAlignment.start,
 this.mainAxisSize = MainAxisSize.max,
 this.crossAxisAlignment = CrossAxisAlignment.center,
```

```
8   this.textDirection,
9   this.verticalDirection = VerticalDirection.down,
10   this.textBaseline,
11   List<Widget> children = const <Widget>[],
12   }) : assert(direction != null),
13   assert(mainAxisAlignment != null),
14   assert(mainAxisSize != null),
15   assert(crossAxisAlignment != null),
16   assert(verticalDirection != null),
17   assert(crossAxisAlignment != CrossAxisAlignment.baseline || textBaselin
e != null),
18   super(key: key, children: children);
19   ...
20  }
```

参数名字	参数类型	意义	必选 or 可选
key	Key	Widget 的标识	可选
direction	Axis	主轴的方向	必选
mainAxisAlignment	MainAxisAlignment	表示 子Widget 在主轴的对齐方式	可选
mainAxisSize	MainAxisSize	表示主轴应该占用多大的空间	可选
crossAxisAlignment	CrossAxisAlignment	表示 子Widget 在交叉轴的对齐方式	可选
textDirection	TextDirection	表示 子Widget 在主轴方向上的布局顺序	可选
verticalDirection	VerticalDirection	表示 子Widget 在交叉轴方向上的布局顺序	可选
textBaseline	TextBaseline	排列 子Widget 时使用哪个基线	可选
children	List<Widget>	Flex布局 里排列的内容	可选

- direction

主轴的方向，类型为Axis,可以为垂直和水平

Axis 的值	含义
Axis.horizontal	主轴方向为水平方向，那么 子Widget 就会沿水平方向排列，交叉轴就是垂直方向。
Axis.vertical	主轴方向为垂直方向，那么 子Widget 就会沿垂直方向排列，交叉轴就是水平方向。

- mainAxisAlignment

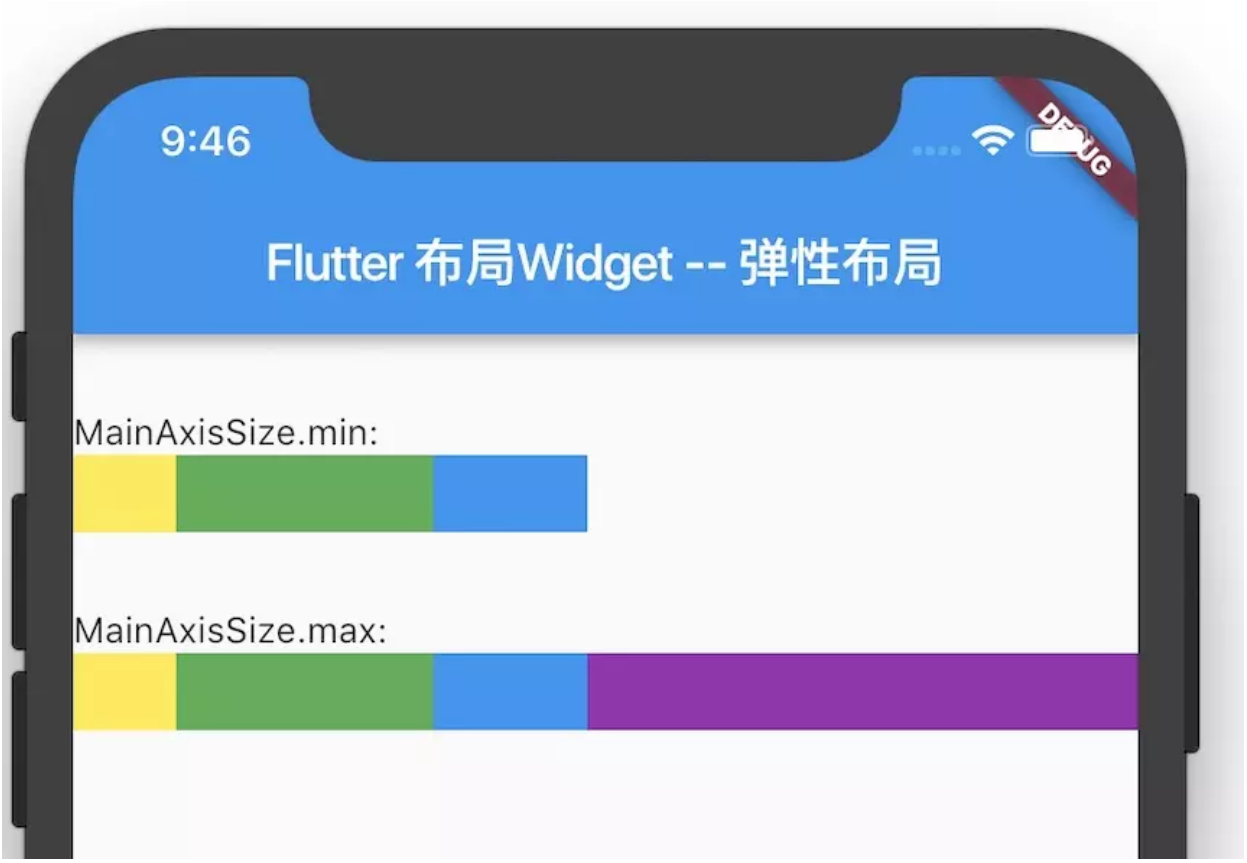
子Widget在主轴的对齐方式，类型是：MainAxisAlignment

MainAxisAlignment 的值	含义
MainAxisAlignment.start	沿着主轴的起点对齐 textDirection 必须有值，以确定是从左边开始的还是从右边开始的
MainAxisAlignment.end	沿着主轴的终点对齐 textDirection 必须有值，以确定是在左边结束的还是在右边结束的
MainAxisAlignment.center	在主轴上居中对齐
MainAxisAlignment.spaceBetween	在主轴上，两端对齐，项目之间的间隔都相等。
MainAxisAlignment.spaceAround	在主轴上，将多余的控件均匀分布给 子Widget 之间，而且第一个 子Widget 和 最后一个子Widget 距边框的距离是 两个 子Widget 距离的一半
MainAxisAlignment.spaceEvenly	在主轴上，将多余的控件均匀分布给 子Widget 之间，而且第一个 子Widget 和 最后一个子Widget 距边框的距离和 子Widget 之间的距离一样

- mainAxissize

表示主轴占用多大空间，类型是MainAxisSize

MainAxisSize 的值含义MainAxisSize.min主轴的大小是能显示完 子Widget 的最小大小，主轴的大小就是 子Widget 的大小MainAxisSize.max主轴能显示的最大的大小，根据约束来判断



- crossAxisAlignment

表示子Widget在交叉轴的对齐方式，类型是：CrossAxisAlignment

CrossAxisAlignment 的值	含义
CrossAxisAlignment.start	沿着交叉轴的起点对齐 verticalDirection 必须有值，以确定是从左边开始的还是从右边开始的
CrossAxisAlignment.end	沿着主轴的终点对齐 verticalDirection 必须有值，以确定是在左边结束的还是在右边结束的
CrossAxisAlignment.center	在交叉轴上居中对齐
CrossAxisAlignment.stretch	要求 子Widget 在交叉轴上填满
CrossAxisAlignment.baseline	要求 子Widget 的基线在交叉轴上对齐

- textDirection

表示子Widget在主轴的布局顺序，类型是TextDirection

TextDirection 的值	含义
TextDirection.rtl	表示从右到左
TextDirection.ltr	表示从左到右

- VerticalDirection

表示子Widget在交叉轴上的布局顺序。类型是VerticalDirection

VerticalDirection 的值	含义
VerticalDirection.up	表示从下到上
VerticalDirection.down	表示从上到下

该属性在主轴为水平时使用

2、Flexible与Expanded

当Flex里内容过长时，超过主轴的大小，此时会抛出layout错误：

例如：

```
Flex(
 direction: Axis.horizontal,
 mainAxisAlignment: MainAxisAlignment.start,
 children: <Widget>[
 Text('Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!')
 ],
)
```

```
A RenderFlex overflowed by 267 pixels on the right.
```

在界面上会看到黑黄的条。

为了避免子Widget在Row、Column、Flex中超界，就可以使用Flexible与Expanded。它们可以让Row、Column、Flex中具有弹性

例如：

```
Flexible(
child: Text(
'Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!'),
)

Expended(
child: Text(
'Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!Hello Flutter!'),
)
```

包裹子Widget后，当子Widget超过主轴大小时，会自动换行。

(1) Flexible

1> 构造函数

```
class Flexible extends ParentDataWidget<Flex> {
 const Flexible({
 Key key,
 this.flex = 1,
 this.fit = FlexFit.loose,
 @required Widget child,
 }) : super(key: key, child: child);
 ...
}
```

参数名字	参数类型	意义	必选 or 可选
key	Key	Widget 的标识	可选
flex	int	此 Widget 的弹性因子	可选
fit	FlexFit	如何分配 弹性Widget 在可用空间里的大小	可选
child	Widget	要显示的 Widget	必选

在Flexible中的flex弹性因子为FlexFit.loose，即当有剩余空间时，只会占用自身大小

(2) Expanded

1> 构造函数

```
class Expanded extends Flexible {
```

```
  /// Creates a widget that expands a child of a [Row], [Column], or [Flex]
  /// expand to fill the available space in the main axis.
  const Expanded({
  Key key,
  int flex = 1,
  @required Widget child,
  }) : super(key: key, flex: flex, fit: FlexFit.tight, child: child);
}
```

参数名字	参数类型	意义	必选 or 可选
key	Key	Widget 的标识	可选
flex	int	此 Widget 的弹性因子	可选
child	Widget	要显示的 Widget	必选

Expanded中的弹性因子是FlexFit.tight，即当有剩余空间时，会占满剩余空间

(3) flex弹性因子

在Flexible和Expanded中有一个重要属性：flex，flex为弹性因子。

弹性因子在布局中是如何计算?

- flex为0或者null，则没有弹性，称为非弹性Widget，非弹性因子的大小即自身的大小

flex大于0则表示有弹性，称为弹性Widget。

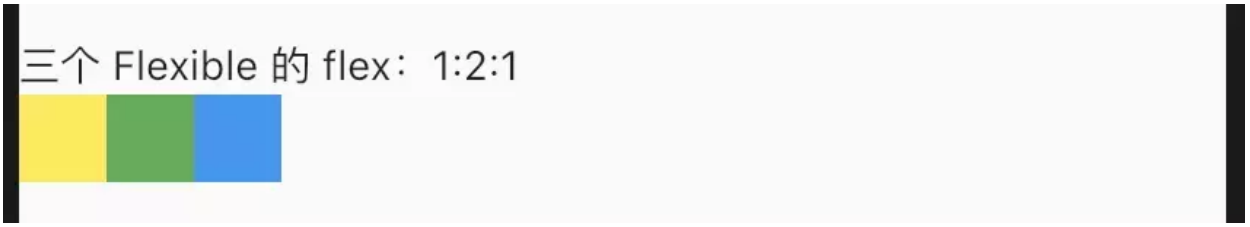
在布局计算中，首先会计算出flex为0的组件，然后按照子Widget中弹性Widet的flex占弹性子Widget的flex总和比例来分割空闲空间

(4) 弹性因子的使用

```
Flex(
 direction: Axis.horizontal,
 mainAxisAlignment: MainAxisAlignment.start,
 children: <Widget>[
 Flexible(
 flex: 1,
 child: Container(
 height: 30.0,
 width: 30.0,
  color: Colors.yellow,
  ),
  ),
```

```
 Flexible(
 flex: 2,
 child: Container(
 height: 30.0,
 width: 30.0,
 color: Colors.green,
 ),
 ),
 Flexible(
 flex: 1,
 child: Container(
 height: 30.0,
 width: 30.0,
 color: Colors.blue,
 ),
 ),
 ],
 ),
```

使用 Flexible 包裹三个宽高都为 30 的色块，并设置 flex 为 1、2、1，效果如下：

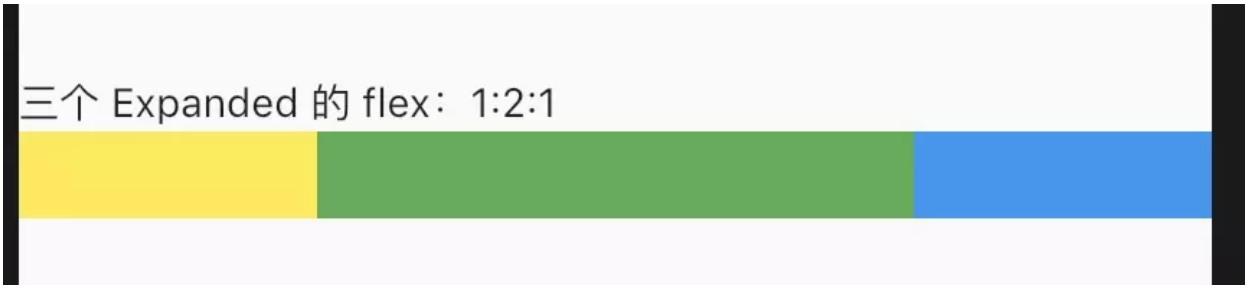


因为 子Widget 的宽度是固定的，所以 Flexible 只会占用本身的大小。

```
Flex(
 direction: Axis.horizontal,
 mainAxisAlignment: MainAxisAlignment.start,
 children: <Widget>[
 Expanded(
 flex: 1,
 child: Container(
 height: 30.0,
 width: 30.0,
 color: Colors.yellow,
 ),
 ),
 Expanded(
 flex: 2,
```

```
child: Container(
height: 30.0,
width: 30.0,
color: Colors.green,
),
),
Expanded(
flex: 1,
child: Container(
height: 30.0,
width: 30.0,
color: Colors.blue,
),
),
],
),
```

使用 Expanded 包裹三个宽高都为 30 的色块，并设置 flex 为 1、2、1，效果如下：



虽然三个色块的宽度是固定的，但是 Expanded 还是按照比例瓜分了剩余的全部空间。