Flutter编程之BoxDecoration用法详解

原创 CoderCyl 于 2020-08-06 09:13:52 发布 9498 收藏 18 版权

分类专栏： Flutter 文章标签： Flutter Widget

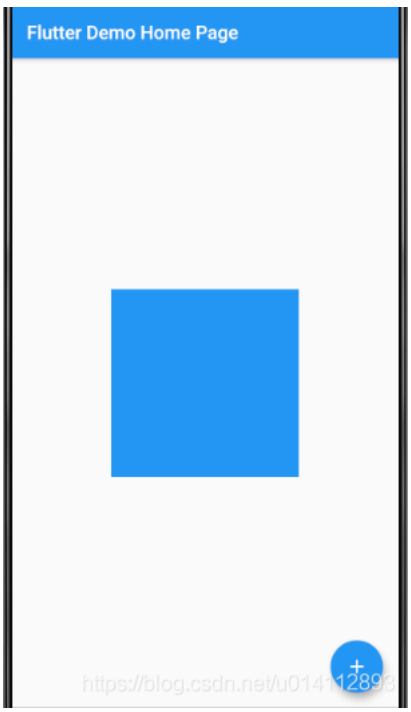
`Widget` 的装饰，使其改变其显示形式。 `Container` 的 `decoration` 用 `BoxDecoration` 来设置。 `BoxDecoration` 的参数如下：

属性	解释	类型
color	颜色背景	Color
image	图片背景	DecorationImage
border	边界	BoxBorder
borderRadius	圆角边界半径	BorderRadiusGeometry
boxShadow	阴影	List<BoxShadow>
gradient	渐变色	Gradient
backgroundBlendMode	背景混合模式	BlendMode
shape	形状	BoxShape

color

设置背景颜色， `decoration` 中的 `color` 不可与 `Container` 的 `color` 属性同时设置，设置了 `decoration` 后， `Container` 的 `color` 必须去掉。定义方式与 `Container` 的 `color` 定义方式一致。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
    ),
  ),
),
```



image

设置图片背景，设置 image 需要一个 DecorationImage 类，其中 image 属性是必须的，其类型是 ImageProvider ，因为 ImageProvider 是抽象类，因此需要用 ImageProvider 的子类来实现。

这里我们加载一个网络图片来作为图片背景，所以就使用 NetworkImage 了， NetworkImage 是 ImageProvider 的子类，正好合适。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(image: NetworkImage("http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")))
  ),
),
```

图片并没有充满整个背景，用 fit 属性来设置一下吧

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        fit: BoxFit.cover,
        image: DecorationImage(image: NetworkImage("http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")))
  ),
),
```

border

设置 Widget 的边框样式，设置方式为 Border.xxx ， Border 为 BoxBorder 类型。

- border.all(Color color, double width,BorderStyle style)

  同时设置上下左右4条边框的样式， color 设置颜色， width 设置边框宽度， style 设置边框样式， style 为 BorderStyle 枚举类型，只有 none 和 solid 两个值。 none 表示无边框， solid 表示实线边框。

  border.all() 的默认颜色是黑色，边框宽度为1.0，样式为 BorderStyle.solid 。

把颜色设置为红色，宽度设置为5.0来看看效果

```
body: Center(
  child: Container(
    width: 200,
```

```
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
                "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
        border: Border.all(
            color: Colors.red,
            width: 5.0,
        )),
  ),
),
```

)

- `Border.symmetric(BorderSide vertical, BorderSide horizontal)`

  从参数可以看出，是分别设置垂直方向的边框和水平方向的边框，即 `vertical` 设置上边和下边的边框， `horizontal` 设置左边和右边的边框。

  参数是 `BorderSide` ，而 `BorderSide` 的参数与 `border.all` 的参数一样，都是 `Color color, double width,BorderStyle style` ，所以设置方式应该是下面这样的：

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
      image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(
              "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
      border: Border.symmetric(
          vertical: BorderSide( // 垂直方向（上、下）
            color: Colors.red,
            width: 5.0,
          ),
          horizontal: BorderSide( // 水平方向（左、右）
            color: Colors.blue,
            width: 10.0,
          )),
    ),
  ),
),
```

  垂直方向上的边框颜色为红色，宽度为5.0，水平方向上的边框颜色为蓝色，宽度为10.0。

- `Border.fromBorderSide(BorderSide side)`

  字面上的意思是从 `BorderSide` 中获取样式设置，设置方式与 `Border.symmetric` 一致，因为都是 `BorderSide` 类型，设置效果与 `Border.all` 一致，因为都是同时设置4条边框的样式。

如：

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
 "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
        border: Border.fromBorderSide(BorderSide(
          color: Colors.red,
          width: 10,
        ))),
  ),
),
```

- `Border({BorderSide top, BorderSide right, BorderSide bottom, BorderSide left})`

  四条边框分别设置，可以设置不同的样式。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
      image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(
              "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
      border: Border(
          top: BorderSide(color: Colors.blue, width: 3), // 上边边框
          right: BorderSide(color: Colors.red, width: 4), // 右侧边框
          bottom: BorderSide(color: Colors.yellow, width: 5), // 底部边框
          left: BorderSide(color: Colors.cyan, width: 6)), // 左侧边框
    ),
  ),
),
```

borderRadius

设置边框圆角，类型为 `BorderRadiusGeometry` ，设置方式为 `BorderRadius.xxx`

- `BorderRadius.all(Radius radius)`

  同时设置4个圆角，参数是 `Radius` ，设置正圆角，则用 `Radius.circular(double radius)` 。，等同于 `BorderRadius.circular(double radius)` ；设置椭圆角，则用 `Radius.elliptical(double x, double y)` 。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
                "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
        border: Border.all(color: Colors.cyan, width: 5), // 设置边框
        borderRadius: BorderRadius.all(Radius.circular(10))), // 设置正圆角
  ),
),
```

再试试设置椭圆角是什么效果

```
borderRadius: BorderRadius.all(Radius.elliptical(10, 100))), // 设置椭圆角
```

试验了很多次，貌似只有在4条边框都设置成一致时，圆角与边框的效果才能同时出现。

- `BorderRadius.zero`

  没有圆角效果

- `BorderRadius.horizontal({Radius left, Radius right})`

  设置水平方向上的圆角， `left` 设置左上和左下两个圆角， `right` 设置右上和右下两个圆角。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
      image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(
              "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
      border: Border.all(color: Colors.cyan, width: 5),
      borderRadius: BorderRadius.horizontal(
          left: Radius.circular(50), right: Radius.circular(20)), // 设置左上和左下圆角半径为50，右上和右下圆角半径为20，也可
    ),
  ),
),
```

- `BorderRadius.vertical({Radius top, Radius bottom})`

  设置垂直方向上的圆角， `top` 设置左上和右上两个圆角， `bottom` 设置左下和右下两个圆角。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
      image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(
              "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
      border: Border.all(color: Colors.cyan, width: 5),
      borderRadius: BorderRadius.vertical(
          top: Radius.circular(50), bottom: Radius.circular(20)), // 设置左上和右上圆角半径为50，左下和右下圆角半径为20，也可
    ),
  ),
),
```

- `BorderRadius.only({Radius topLeft, Radius topRight, Radius bottomLeft, Radius bottomRight})`

  单独设置每个角的圆角效果。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.blue,
      image: DecorationImage(
          fit: BoxFit.cover,
          image: NetworkImage(
              "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
      border: Border.all(color: Colors.cyan, width: 5),
      borderRadius: BorderRadius.only(
          topLeft: Radius.circular(10),
          topRight: Radius.circular(20),
          bottomLeft: Radius.circular(30),
          bottomRight: Radius.circular(40)),
    ),
  ),
),
```

boxShadow

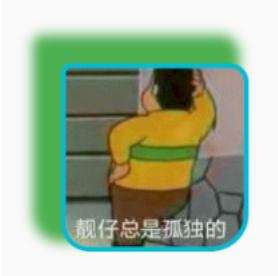
设置阴影效果。参数类型是 `List<BoxShadow>`，是一个集合，可以看出应该是多个 `BoxShadow` 叠加后的效果。 `BoxShadow` 参数如下：

- `Color color`： 阴影颜色
- `Offset offset`： 偏移量
- `double blurRadius`： 模糊半径，半径越大越模糊
- `double spreadRadius`： 延伸范围半径，半径越大，阴影范围越广

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
                "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
        border: Border.all(color: Colors.cyan, width: 5),
        borderRadius: BorderRadius.only(
            topLeft: Radius.circular(10),
            topRight: Radius.circular(20),
            bottomLeft: Radius.circular(30),
            bottomRight: Radius.circular(40)),
        boxShadow: [
          BoxShadow( // 设置第一个阴影效果
              color: Colors.green, // 阴影颜色为绿色
              offset: Offset(-20, -20),
              blurRadius: 10, // 模糊半径为10
              spreadRadius: 10), // 延伸半径为10
        ]),
  ),
),
```

上图是没有设置 `Offset` 偏移量，我们让它向左上方各偏移20看看是什么效果

```
BoxShadow( // 设置第一个阴影效果
    color: Colors.green, // 阴影颜色为绿色
    offset: Offset(-20, -20), // X轴左移20，Y轴上移20
    blurRadius: 10, // 模糊半径为10
    spreadRadius: 10), // 延伸半径为10
```



再看看设置两个 BoxShadow 叠加是什么效果

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
        color: Colors.blue,
        image: DecorationImage(
            fit: BoxFit.cover,
            image: NetworkImage(
                "http://wx4.sinaimg.cn/mw690/6a04b428gy1fyrldlsv4yg204r05i3yt.gif")),
        border: Border.all(color: Colors.cyan, width: 5),
        borderRadius: BorderRadius.only(
            topLeft: Radius.circular(10),
            topRight: Radius.circular(20),
            bottomLeft: Radius.circular(30),
            bottomRight: Radius.circular(40)),
        boxShadow: [
          BoxShadow( // 第一个阴影
              color: Colors.green,
              offset: Offset(-20, -20),
              blurRadius: 10,
              spreadRadius: 10),
          BoxShadow( // 增加第二个阴影
              color: Colors.yellow, // 阴影颜色为黄色
              offset: Offset(10, 10), // 右下方偏移10
              blurRadius: 20,
              spreadRadius: 20),
        ]),
  ),
),
```

可以看出，多个 BoxShadow 确实是叠加的效果，并且后面的 BoxShadow 会在之前的 BoxShadow 的上方。

对比一下两个 BoxShadow 交换位置的效果

黄色阴影在上	绿色阴影在上

黄色阴影在上	绿色阴影在上
	
	

gradient

设置渐变背景色，类型是 `Gradient` ， `Gradient` 是抽象类，它有3个实现类：

- `LinearGradient` ： 线性渐变

  `LinearGradient(AlignmentGeometry begin, AlignmentGeometry end, List<Color> colors, List<double> stops, TileMode tileMode, GradientTransform transform)`

  - `colors`

    渐变色数组，不可为空，如 `[Colors.red, Colors.yellow, Colors.green]`

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [Colors.red, Colors.yellow, Colors.green],
      ),
    ),
  ),
),
```

  - `stops`

渐变色的终止百分比位置的数组，可为空。如 `[0.1, 0.3, 0.5]` ，当 `stops` 不为空时，其长度必须和 `colors` 一致。

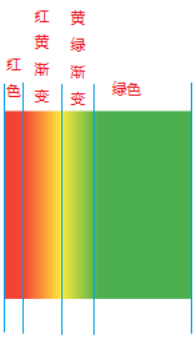
```
    body: Center(
      child: Container(
        width: 200,
        height: 200,
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [Colors.red, Colors.yellow, Colors.green],
            stops: [0.1, 0.3, 0.5],
          ),
        ),
      ),
    ),
```

登录后复制

从图中可以不太容易看出来 `stops` 是什么意思。其实 `stops` 中的每个元素的取值范围是 `0.0 ~ 1.0` ，把 `Widget` 从 `begin` 开始位置到 `end` 结束位置平均分为10份，每个元素之间表示一个范围值，因此一般 `stops` 里的元素都是递增的，如果设置成 `[0.1, 0.3, 0.1]` 这种递增再递减就没有实际的意义了。

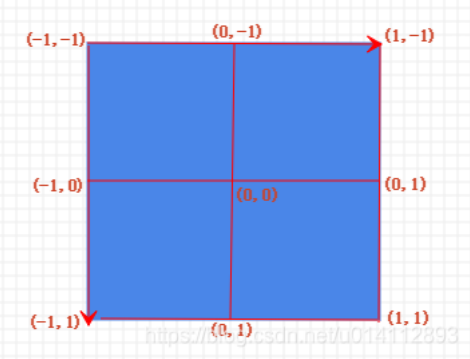
对于上面例子来说，因为没指定 `begin` 和 `end` 的值（这两个值的含义下面会讲到），那么就是默认值 `Alignment.centerLeft` 和 `Alignment.centerRight` ，表示从左侧中间往右侧中间水平渐变。 `[Colors.red, Colors.yellow, Colors.green]` 和 `[0.1, 0.3, 0.5]` 这两个数组表示：

```
0.0 ~ 0.1: 红色
0.1 ~ 0.3: 红色到黄色渐变
0.3 ~ 0.5: 黄色到绿色渐变
0.5 ~ 1.0: 绿色
```



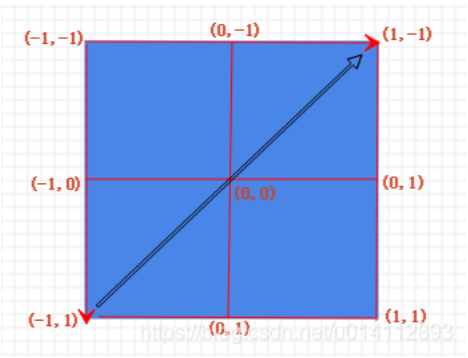
- `begin` `end`

  渐变颜色的起止位置，表示渐变色从 `begin` 位置开始，到 `end` 位置结束。她们都是 `AlignmentGeometry` 类型，表示对齐方式。和 `Container` 的对齐方式 `alignment` 一样，将一个 `Widget` 分为了9个点(参考Container的对齐方式)：



默认的对齐方式是从左侧中间到右侧中间渐变，即从 `(-1, 0)` 到 `(0 1)` 。

如果将上例中的 `begin` 指定为 `Alignment.bottomLeft` ， `end` 指定为 `Alignment.topRight` ，则渐变方向变成了从 左下角 往 右上角 渐变。



- `tileMode`

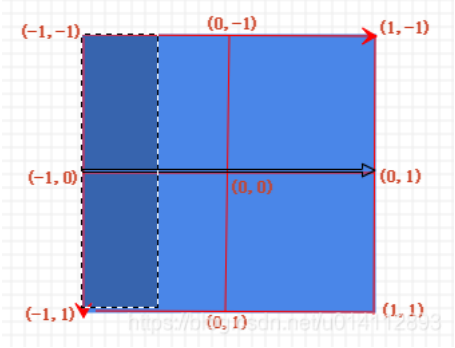
  平铺模式，共有三种模式， `TileMode.clamp` `TileMode.repeated` `TileMode.mirror` ，默认模式是 `TileMode.clamp` 。

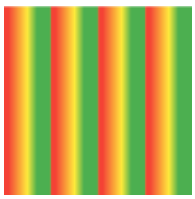
  如果在上例的基础上直接设置 `tileMode` 是看不到任何效果的，因为默认情况下始末位置就是占满整个 `Widget` 。

  我们现在把渐变区域设置为整个区域的1/4，水平向右渐变。

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      gradient: LinearGradient(
          begin: Alignment(-1, 0),
          end: Alignment(-0.5, 0),
          colors: [Colors.red, Colors.yellow, Colors.green],
          stops: [0.1, 0.5, 0.7],
          tileMode: TileMode.xxx),
    ),
  ),
),
```

  `begin` 是 `(-1, 0)` ， `end` 是 `(-0.5, 0)` ，刚好占据1/4,。



TileMode.clamp	TileMode.repeated	TileMode.mirror
		

- `RadialGradient`： 放射渐变

  `RadialGradient({AlignmentGeometry center, double radius, List<Color> colors, List<double> stops, TileMode tileMode, AlignmentGeometry focal, double focalRadius})`

  其中 `colors` `stops` 和线性渐变一样，接下来说说剩下的几个参数：

  - `center`

    扫描区域的中心位置，也就是圆心的位置，默认是 `Alignment.center` ，即 `Widget` 中心的位置。

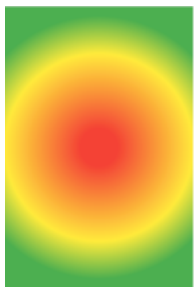
```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      gradient: RadialGradient(
        colors: [Colors.red, Colors.yellow, Colors.green],
        stops: [0.1, 0.5, 0.7],
      ),
    ),
  ),
),
```

将 `center` 的位置设置到右上角

  - `radius`

    扫描渐变区域半径比例，默认为 `0.5` ，即矩形 `Widget` 较短边的一半。

将 `radius` 设置成1，



- tileMode
- `focal` `focalRadius`

  还没理解清楚这是什么

- `SweepGradient`： 扫描渐变

  `SweepGradient({AlignmentGeometry center, double startAngle, double endAngle, List<Color> colors, List<double> stops, TileMode tileMode, GradientTransform transform})`

  除了 `startAngle` `endAngle` 之外，其他的参数都和上面的一样。

  - `startAngle`

    开始渐变的弧度，觉得这个地方的命名并不合适，应该叫 `startRadians` 才对，意思是开始弧度， `2π` 为一周360度，默认值是0，为右侧中间的位置。

  - `endAngle`

    结束渐变的弧度，默认值是 `2π` ，还有一点就是 `endAngle` 必须大于 `startAngle` ，否则不绘制。

```
import 'dart:math'; // 需要引入math包

body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      gradient: SweepGradient(
          colors: [Colors.red, Colors.blue, Colors.yellow],
          startAngle: pi,
          endAngle: pi * 2),
    ),
  ),
),
```

渐变范围从弧度为 π 到 `2π` ，所以真正的扫描渐变范围在上面一半，下面一半红色显示的是开始颜色红色。

现在我们把扫描范围设置成从 `0` 到 `3 * π/ 4` ，再看看效果

从上面2个效果图中可以看出来，在默认混合模式 `TileMode.clamp` 中，当扫描范围不满一周时，接近0弧度的区域填充的是 `colors` 中设置的起始颜色，越接近 `2π` 弧度的区域填充的是 `colors` 中设置的结束颜色。

最后来对比一下最后一个例子中在不同混合模式下的效果

```
body: Center(
  child: Container(
    width: 200,
    height: 200,
    decoration: BoxDecoration(
      gradient: SweepGradient(
        colors: [Colors.red, Colors.blue, Colors.yellow],
        startAngle: 0,
        endAngle: 3 * pi / 4,
        tileMode: xxx,
      ),
    ),
  ),
),
```

TileMode.clamp	TileMode.repeated	TileMode.mirror