UI绘制--常用widget

- 容器类按照child数目进行Widget分类
  - 只有一个child
    - Container
    - Padding
    - Align
    - Center
    - FittedBox
    - AspectRatio
    - ConstraineBox
    - Baseline
    - FracionallySizeBox
    - IntrinsWidth
    - LimitedBox
    - Offstage
    - OverflowboX
    - SizeBox
    - SizedOverflowBox
    - Transform
    - CustomSingleChileLayout
  - 可以有多个child
    - Row
    - Column
    - Stack
    - IndexedStack
    - GridView
    - Flow
    - Table
    - Wrap
    - ListBody
    - ListView
    - CustomMultiChildLayout
- 基础
  - Text
  - Image
  - Icon
  - RaseButton
- Material Components(https://flutterchina.club/widgets/material/)
  - MaterialApp
  - Scaffold
  - AppBar
  - BottomNavigationBar
  - TabBar
  - TabBarView
  - WidgetsApp
  - Drawer
  - ...

- MaterialApp
  - Material风格UI，可以作为构建一个应用的基础widget

- Material Components Widgets 中的一个widget，它封装了应用程序实现Material Design所需要的一些widget
- https://flutterchina.club/widgets/material/

字段	类型
navigatorKey（导航键）	GlobalKey<NavigatorState>
home（主页）	Widget
routes（路由）	Map<String, WidgetBuilder>
initialRoute（初始路由）	String
onGenerateRoute（生成路由）	RouteFactory
onUnknownRoute（未知路由）	RouteFactory
navigatorObservers（导航观察器）	List<NavigatorObserver>
builder（建造者）	TransitionBuilder
title（标题）	String
onGenerateTitle（生成标题）	GenerateAppTitle
color（颜色）	Color
theme（主题）	ThemeData
locale(地点)	Locale
localizationsDelegates（本地化委托）	Iterable<LocalizationsDelegate<dynamic
localeResolutionCallback（区域分辨回调）	LocaleResolutionCallback
supportedLocales（支持区域）	Iterable<Locale>
debugShowMaterialGrid（调试显示材质网格）	bool
showPerformanceOverlay（显示性能叠加）	bool
checkerboardRasterCacheImages（棋盘格光栅缓存图像）	bool
checkerboardOffscreenLayers（棋盘格层）	bool
showSemanticsDebugger（显示语义调试器）	bool
debugShowCheckedModeBanner（调试显示检查模式横幅）	bool

- 重点参数
  - title 显示在任务管理器上的title名称
  - theme 主题
  - home 进入应用的主页，如果不赋值会运行异常
  - routes 键值对页面路由，定义页面以及页面名称对应关系，定义在routes中的页面，可以通过

Navigator.of(context).pushNamed方法进行页面跳转， 定义为'/'，或

Navigator.defaultRouteName 默认为主页

- 路由（Route） 、 导航（Navigatir）
  - 路由
    - 注册所有页面
  - 导航
    - 控制页面跳转
  - 跳转页面
    - 通过创建MaterialPageRoute（）构建一个页面路由

```
Navigator.push(
  context,
  new MaterialPageRoute(builder: (context) => new FirstRoute()),
);

onPressed: () {
  Navigator.pop(context); // 退出当前页面
}
```

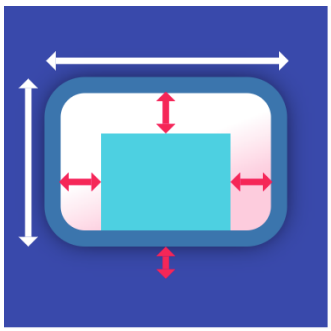
  - 自定义页面路由效果
    - 使用PageRouteBuilder实现页面切换动画效果

```
/**
* 自定义页面过渡效果
*/
void _navigateToPage(BuildContext context) {
  Navigator.push(context,
      new PageRouteBuilder(pageBuilder: (BuildContext context, Animation<double> animation, Animation<double>
secondaryAnimation) => new FirstRoute(),
        transitionDuration: const Duration(milliseconds: 1000),
        transitionsBuilder: (BuildContext context, Animation<double> animation, _, Widget child) =>
        new FadeTransition(opacity: animation,
          child: new RotationTransition(
            turns: new Tween<double>(begin: 0.0, end: 1.0).animate(animation),
            child: child,),),
      ));
}
```

- Scaffold

```
const Scaffold({
  Key key,
  this.appBar, // 头部水平栏
  this.body, // 内部view
  this.floatingActionButton,
  this.floatingActionButtonLocation,
  this.floatingActionButtonAnimator,
  this.persistentFooterButtons, // body顶部bottomBar顶部
  this.drawer,
  this.endDrawer,
  this.bottomNavigationBar, // 底部导航栏
  this.backgroundColor, // 背景色
  this.resizeToAvoidBottomPadding: true,
  this.primary: true, 状态栏AppBar高度
})
```

- Container
  - 理解为矩形容器，类似layout，可以控制大小、位置、背景、边框、阴影。内部只有一个childview



```
// 可设置的属性
Container({
  Key key, // id
  this.alignment, // 内部child对其位置
  this.padding,
  Color color,
  Decoration decoration, // 背景
  this.foregroundDecoration, // 前景
  double width,
  double height,
  BoxConstraints constraints, // child约束
  this.margin,
  this.transform, // 位置变换
  this.child,
}
```

Container自身尺寸的调节几种情况：

- Container在没有子view的时候，会满屏。
- 有子view时
  - 如果想要实现包裹住子view
    - constraints 不设置
    - width 无限制
    - height 无限制
    - alignment 空
  - 如果要实现宽度包裹高度最大
    - constraints 不设置
    - alignment 空
    - width 无限大 double.infinity
- 含有子view，但是没有width、height、constraints以及alignment，Container会将父节点的constraints传递给child，并且根据child调整自身
- 如果container设置了宽高
- 如果不设置宽高，则包裹内容，根据constrains的限制去调整整体大小 constraints: BoxConstraints(maxHeight: 100.0) 此时container高度为100，宽度为屏幕最大
- 如果设置了宽高， constraints: BoxConstraints(maxHeight: 100.0, maxWidth: 100.0) 宽高大于字view最大高度，则以onstraint为主
- 总结
  - 如果Container的约束是有限制的，那么没有子部件的Container会尝试尽可能大，如果Container的约束是没有限制(即未进行定义)，它就会尽可能小（无alignment时）。
  - 有子部件的Container，根据子部件确定自己的大小。

Container的组成如下：

- 最里层的是child元素；
- child元素首先会被padding包着；
- 然后添加额外的constraints限制；
- 最后添加margin。

Container的绘制的过程如下：

- 设置padding
- 添加子view的约束
- 绘制transform效果
- 绘制decoration
- 绘制child
- 最后绘制foregroundDecoration

使用场景

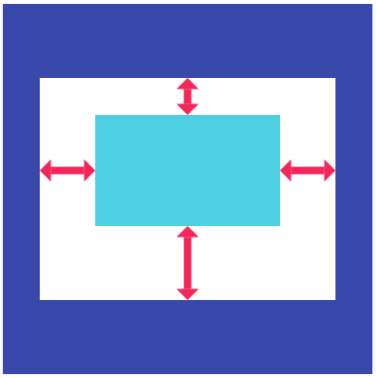
- 需要设置间隔、背景、圆角、对齐效果时使用

GestureDetector 手势widget

- 继承了StatelessWidget，内部chile作为承载点击事件的ui
- 主要方法
  - OnTap() 点击完成
  - OnTapDown() 按下
  - OnTapCancle() 取消
  - onTapUp() 抬起

```
new GestureDetector(
    onTap: () {
      print("onTap");
    },
    onTapDown: (TapDownDetails details) {
      print("onTapDown");

    },
    onTapUp: (TapUpDetails details) {
      print("onTapUp");
    },
    onTapCancel: () {
      print("onTapCancel");
    },
    child: new Container(
      decoration: new BoxDecoration(
        color: currentColor,
        borderRadius:
        new BorderRadius.all(new Radius.circular(widget.height / 2.0)),
      ),
      height: widget.height,
      width: widget.width,
      alignment: Alignment.center,
      child: new Text(
        widget.text,
      style: new TextStyle(
        fontSize: 18.0,
        color: currentTextColor,
      ),
    ),
    ),
  );
}
```

Margin 与 Padding



开发过程中，经常用到margin与padding去设置widget之间的间距，Flutter中设置Padding需要使用Padding控件。

- 使用

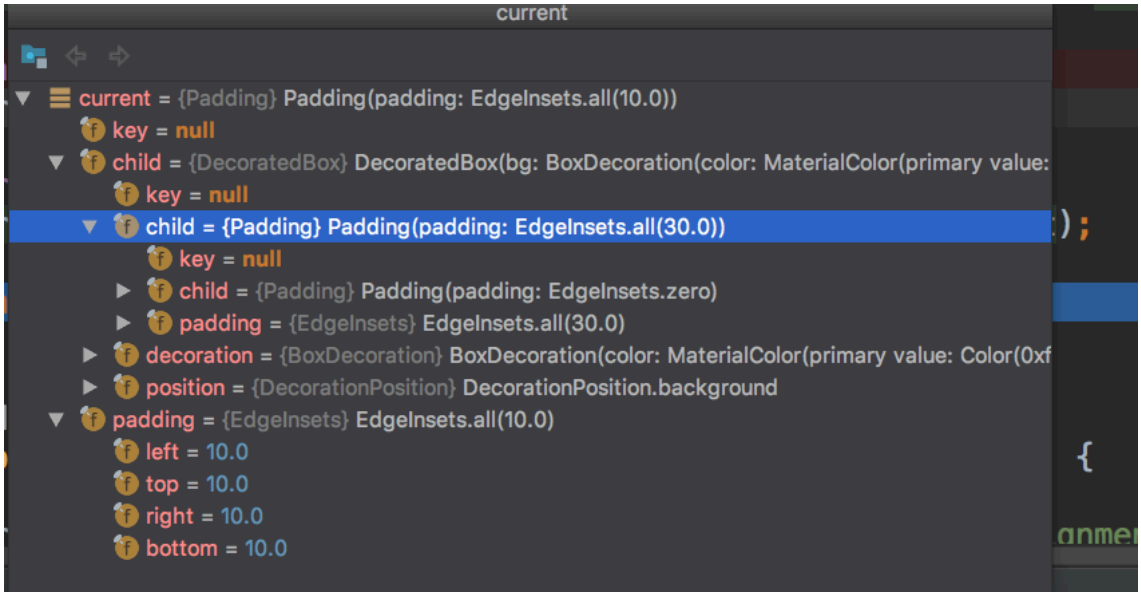
```
new Padding(
  padding: new EdgeInsets.all(8.0), // 为card设置宽度为8的内边距
  child: const Card(child: const Text('Hello World!')),
)
```

而margin在flutter中并不是以控件形式存在，而是作为widget的属性存在的，比如

```
Container({
  Key key,
  this.alignment,
  this.padding,
  this.margin, // margin属性
  this.child,
})

@override
Widget build(BuildContext context) {
  Widget current = child;

  if (child == null && (constraints == null || !constraints.isTight)) {
    current = new LimitedBox(
      maxWidth: 0.0,
      maxHeight: 0.0,
      child: new ConstrainedBox(constraints: const BoxConstraints.expand())
    );
  }

  if (alignment != null)
    current = new Align(alignment: alignment, child: current);

  final EdgeInsetsGeometry effectivePadding = _paddingIncludingDecoration;
  if (effectivePadding != null)
    current = new Padding(padding: effectivePadding, child: current);

  if (decoration != null)
    current = new DecoratedBox(decoration: decoration, child: current);

  if (foregroundDecoration != null) {
    current = new DecoratedBox(
      decoration: foregroundDecoration,
      position: DecorationPosition.foreground,
      child: current
    );
  }

  if (constraints != null)
    current = new ConstrainedBox(constraints: constraints, child: current);

  if (margin != null)
    current = new Padding(padding: margin, child: current); // 仍然使用padding实现
```

```
  if (transform != null)
    current = new Transform(transform: transform, child: current);

  return current;
}
```

从container的build源码中可以看出，margin实现同样是以padding实现。
为container同时设置margin和padding， 断点结果如下



从断点结果看出，最后的container被封装成一个padding，padding内部再嵌套一个padding。

- padding属性
  padding类型为 EdgeInsetsGeometry，EdgeInsetsGeometry是EdgeInsets以及EdgeInsetsDirectional的基类
- EdgeInsets与方向无关
  - EdgeInsets.Only()，这个方法可以只设置一个参数
  - EdgetInsets.all()，设置后左、上、右、下都会留有同等间距

```
const EdgeInsets.only({
  this.left: 0.0,
  this.top: 0.0,
  this.right: 0.0,
  this.bottom: 0.0
});
```

- EdgeInsetsDirectional

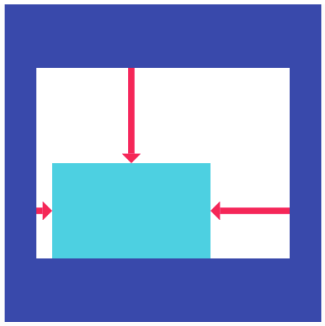
```
const EdgeInsetsDirectional.only({
  this.start: 0.0,
  this.top: 0.0,
  this.end: 0.0,
  this.bottom: 0.0
});
```

使用场景

如果只是要设置间距，使用Padding比Container的成本要小一些，Container里面包含了多个widget。Padding能够实现的，Container都能够实现，而Container更加的复杂

Align

它可以将其子widget对齐，并可以根据子widget的大小自动调整大小



设置child的对齐方式，原生中将Align作为view的一个属性，而Flutter中把Align作为一个控件来使用

```
const Align({
  Key key,
  this.alignment: Alignment.center,
  this.widthFactor, // 至少是1.0 宽度是子view的几倍，不为负
  this.heightFactor, // 至少是1.0
  Widget child
})
```

- Aligment默认值

```
/// The top left corner.
static const Alignment topLeft = const Alignment(-1.0, -1.0);

/// The center point along the top edge.
static const Alignment topCenter = const Alignment(0.0, -1.0);

/// The top right corner.
static const Alignment topRight = const Alignment(1.0, -1.0);

/// The center point along the left edge.
static const Alignment centerLeft = const Alignment(-1.0, 0.0);

/// The center point, both horizontally and vertically.
static const Alignment center = const Alignment(0.0, 0.0);

/// The center point along the right edge.
static const Alignment centerRight = const Alignment(1.0, 0.0);

/// The bottom left corner.
static const Alignment bottomLeft = const Alignment(-1.0, 1.0);

/// The center point along the bottom edge.
static const Alignment bottomCenter = const Alignment(0.0, 1.0);

/// The bottom right corner.
static const Alignment bottomRight = const Alignment(1.0, 1.0);
```

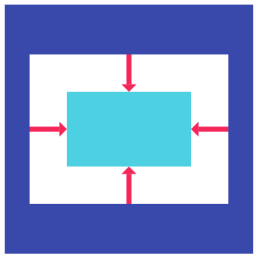
实现源码

```
createRenderObject 方法，使用RenderPositionBox增加child限制
@override
RenderPositionedBox createRenderObject(BuildContext context) {
  return new RenderPositionedBox(
    alignment: alignment,
    widthFactor: widthFactor,
    heightFactor: heightFactor,
    textDirection: Directionality.of(context),
  );
}

RenderPositionedBox 的performLayout方法确定位置

@override
void performLayout() {
  final bool shrinkWrapWidth = _widthFactor != null || constraints.maxWidth == double.infinity;
  final bool shrinkWrapHeight = _heightFactor != null || constraints.maxHeight == double.infinity;
```

```
  if (child != null) {
//  如果child不为null，则根据规则设置Align的宽高，如果需要缩放，则根据_widthFactor是否为null来进行缩放，如果不需要，则尽量扩
    child.layout(constraints.loosen(), parentUsesSize: true);
    size = constraints.constrain(new Size(shrinkWrapWidth ? child.size.width * (_widthFactor ?? 1.0) : double.infinity,
                                          shrinkWrapHeight ? child.size.height * (_heightFactor ?? 1.0) :
double.infinity));
    alignChild();
  } else {
    size = constraints.constrain(new Size(shrinkWrapWidth ? 0.0 : double.infinity,
                                          shrinkWrapHeight ? 0.0 : double.infinity));
  }
}
```

Center

将其子widget居中显示在自身内部的widget，继承了Align



```
class Center extends Align {
  /// Creates a widget that centers its child.
  const Center({ Key key, double widthFactor, double heightFactor, Widget child })
    : super(key: key, widthFactor: widthFactor, heightFactor: heightFactor, child: child);
}
```

默认Aligment.Center