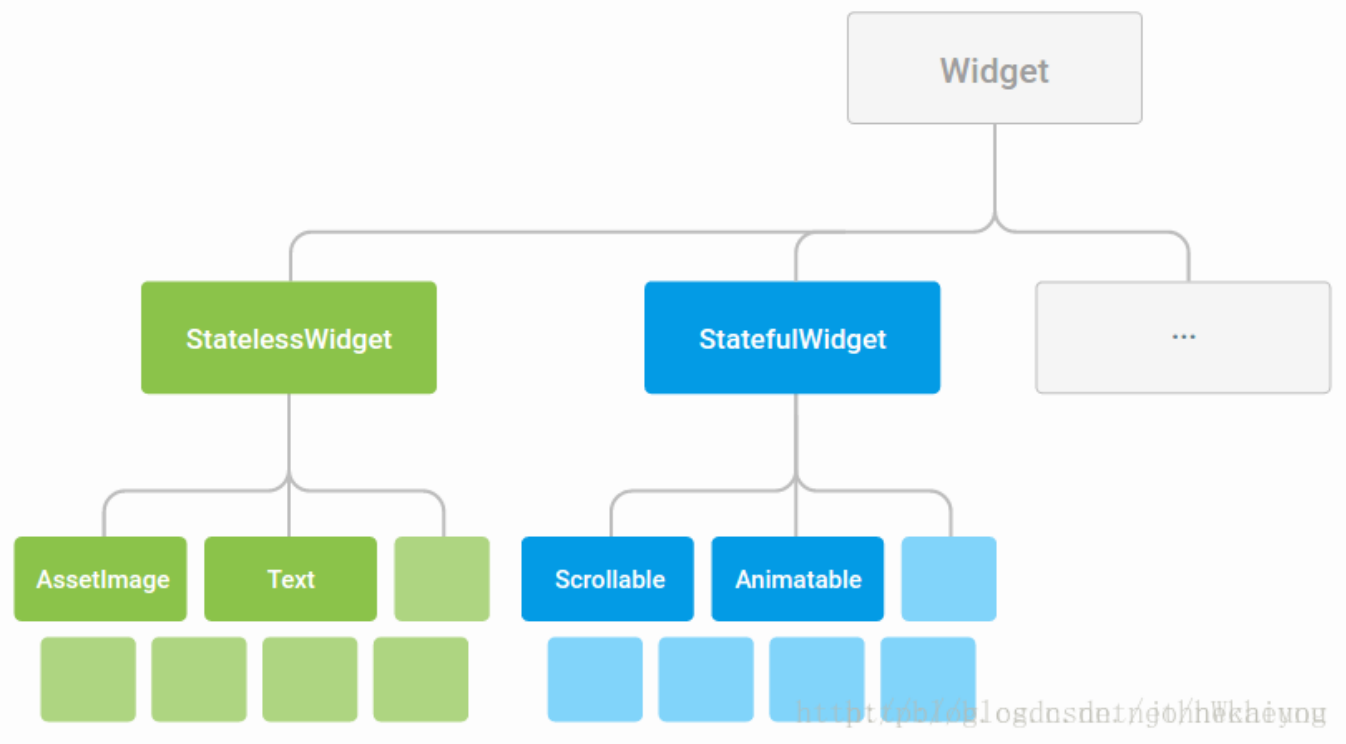
Flutter基本控件介绍

Flutter控件本身通常由许多小型、单用途的控件组成，结合起来产生强大的效果，例如，Container是一种常用的控件，由负责布局、绘画、定位和大小调整的几个控件组成，具体来说，Container是由LimitedBox、ConstrainedBox、 Align、Padding、DecoratedBox和Transform控件组成，而不是将Container子类化来产生自定义效果，您可以用这种新颖的方式组合这些以及其他简单的控件。

类的层次结构是扁平的，以最大化可能的组合数量。



在写应用程序时，经常会使用StatelessWidget和StatefulWidget编写新控件，两者的差别在于你是否要管理控件的状态。一个控件的主要任务是实现build函数，定义控件中其他较低层次的控件。build函数将依次构建这些控件，直到底层渲染对象。

基本组件

Container

容器，一个常用的控件，由基本的绘制、位置和大小控件组成。负责创建矩形的可视元素，可以用BoxDecoration来设计样式，比如背景、边框和阴影，Container也有边距、填充和大小限制，另外，还可以在三维空间利用矩阵进行变换。

没有子控件的容器尽可能大，除非传入的大小约束是无限的，在这种情况下，它们尽可能小。有子控件的容器将自己的尺寸给他们的孩子。我们可以通过width、height和 constraints属性控制size。

```
new Container(
  constraints: new BoxConstraints.expand(
    height: Theme.of(context).textTheme.display1.fontSize * 1.1 + 200.0,
  ),
  padding: const EdgeInsets.all(8.0),
  color: Colors.teal.shade700,
  alignment: Alignment.center,
  child: new Text('Hello World', style: Theme.of(context).textTheme.display
  foregroundDecoration: new BoxDecoration(
    image: new DecorationImage(
      image: new NetworkImage('https://www.example.com/images/frame.png'),
      centerSlice: new Rect.fromLTRB(270.0, 180.0, 1360.0, 730.0),
    ),
  ),
  transform: new Matrix4.rotationZ(0.1),
)
```

Row

flex水平布局控件，能够将子控件水平排列，是基于Web的flexbox的布局模式设计的。

Row子控件有灵活与不灵活的两种，Row首先列出不灵活的子控件，减去它们的总宽度，计算还有多少可用的空间。然后Row按照Flexible.flex属性确定的比例在可用空间中列出灵活的子控件。要控制灵活子控件，需要使用Expanded控件。

注意该控件不支持滑动，如果子控件超过剩余空间，会报错，如果想支持水平滑动，考虑使用ListView。

如果只有一个子控件，可以使用 Align or Center控件定义该子控件位置。

```
new Row(
  children: <Widget>[
    new Expanded(
      child: new Text('Deliver features faster', textAlign: TextAlign.cente
    ),
    new Expanded(
      child: new Text('Craft beautiful UIs', textAlign: TextAlign.center),
    ),
    new Expanded(
      child: new FittedBox(
        fit: BoxFit.contain, // otherwise the logo will be tiny
        child: const FlutterLogo(),
      ),
    ),
  ],
)
```

Column

flex垂直布局控件，能够将子控件垂直排列。

用法与Row控件一样。

```
new Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  mainAxisSize: MainAxisSize.min,
  children: <Widget>[
    new Text('We move under cover and we move as one'),
    new Text('Through the night, we have one shot to live another day'),
    new Text('We cannot let a stray gunshot give us away'),
    new Text('We will fight up close, seize the moment and stay in it'),
    new Text('It’s either that or meet the business end of a bayonet'),
    new Text('The code word is ‘Rochambeau,’ dig me?'),
    new Text('Rochambeau!', style: DefaultTextStyle.of(context).style.apply
  ],
)
```

Image

显示图像的控件，Image控件有多种构造函数：

- new Image，用于从ImageProvider获取图像。
- new Image.asset，用于使用key从AssetBundle获取图像。
- new Image.network，用于从URL地址获取图像。
- new Image.file，用于从File获取图像。

为了自动执行像素密度感知资源分辨率，使用AssetImage指定图像，需要确保在控件树中的图片控件上方存在MaterialApp、WidgetsApp和MediaQuery控件。

不同的手机有不同的像素比率，这时就需要根据手机的像素比率来加载不同图片，做法很简单，只需要在图片同级目录下创建2.0x/…和3.0x/…的目录就可以了。

我们在pubspec.yaml这个文件里指定本地图片路径

```
# To add assets to your application, add an assets section, like this:
# assets:
# - images/a_dot_burr.jpeg
# - images/a_dot_ham.jpeg
```

Text

用来显示文本的控件

下面的实例有7个不同样式的文本控件：

```
import 'package:flutter/material.dart';
class TextDemo extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text('文本控件'),
```

```
    ),
    body: new Column(
      children: <Widget>[
        new Text(
          '红色+黑色删除线+25号',
          style: new TextStyle(
            color: const Color(0xffff0000),
            decoration: TextDecoration.lineThrough,
            decorationColor: const Color(0xff000000),
            fontSize: 25.0,
          ),
        ),
        new Text(
          '橙色+下划线+24号',
          style: new TextStyle(
            color: const Color(0xffff9900),
            decoration: TextDecoration.underline,
            fontSize: 24.0,
          ),
        ),
        new Text(
          '虚线上划线+23号+倾斜',
          style: new TextStyle(
            decoration: TextDecoration.overline,
            decorationStyle: TextDecorationStyle.dashed,
            fontSize: 23.0,
            fontStyle: FontStyle.italic,
          ),
        ),
        new Text(
          'serif字体+24号',
          style: new TextStyle(
            fontFamily: 'serif',
            fontSize: 26.0,
          ),
        ),
        new Text(
          'monospace字体+24号+加粗',
          style: new TextStyle(
            fontFamily: 'monospace',
            fontSize: 24.0,
            fontWeight: FontWeight.bold,
          ),
        ),
        new Text(
```

```
            '天蓝色+25号+2行跨度',
            style: new TextStyle(
              color: const Color(0xff4a86e8),
              fontSize: 25.0,
              height: 2.0,
            ),
          ),
          new Text(
            '24号+2个字母间隔',
            style: new TextStyle(
              fontSize: 24.0,
              letterSpacing: 2.0,
            ),
          ),
        ]
      ),
    );
  }
}
void main() {
  runApp(
    new MaterialApp(
      title: 'Flutter教程',
      home: new TextDemo(),
    ),
  );
}
```

Icon

图标控件，按照IconData中所描述的规则绘制，如Material中预定义的IconDatas。

该控件不可交互，要实现可交互的图标，可以考虑使用Material中的IconButton。

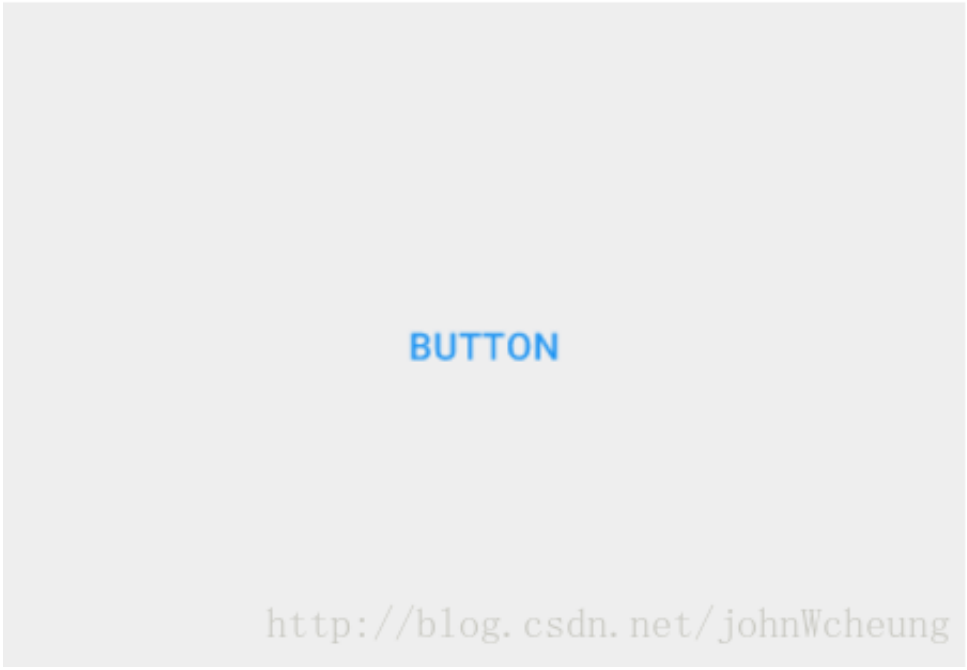
该控件必须在 Directionality控件里使用，通常这是由WidgetsApp或MaterialApp自动引入的。

详见：https://docs.flutter.io/flutter/widgets/Icon-class.html

RaisedButton

Material Design 风格的浮动按钮，以方形纸片样式悬停在界面上，点击后会产生墨水扩散效果。

避免在dialog和card控件里使用，一般弹出式的控件建议使用扁平化按钮，减少布局层次叠加。



使用时，要实现onPressed回调方法，否则按钮处于禁用状态，默认显示disabledColor样式的扁平化按钮，并且此时更改按钮的颜色不会生效。

注意该控件的父控件必须是Material控件。

如果你只需要点击后产生墨水扩散效果，但不想使用按钮，请考虑直接使用InkWell控件。

如有必要，该按钮将拉伸以适应子控件大小。

详见：https://docs.flutter.io/flutter/material/RaisedButton-class.html

Scaffold

Scaffold 实现了基本的Material Design布局结构。也就是说， MaterialApp 的 child 是 Scaffold Widget。

在Material设计中定义的单个界面上的各种布局元素，在 Scaffold 中都有支持，比如 左边栏（Drawers）、snack bars、以及 bottom sheets。

Scaffold 有下面几个主要属性：

- appBar：显示在界面顶部的一个 AppBar，也就是 Android 中的 ActionBar 、Toolbar
- body：当前界面所显示的主要内容 Widget
- floatingActionButton：Material设计中所定义的 FAB，界面的主要功能按钮
- persistentFooterButtons：固定在下方显示的按钮，比如对话框下方的确定、取消按钮
- drawer：侧边栏控件
- backgroundColor： 内容的背景颜色，默认使用的是 ThemeData.scaffoldBackgroundColor 的值
- bottomNavigationBar： 显示在页面底部的导航栏
- resizeToAvoidBottomPadding：类似于 Android 中的 android:windowSoftInputMode=”adjustResize”，控制界面内容 body 是否重新布局来避免底部被覆盖了，比如当键盘显示的时候，重新布局避免被键盘盖住内容。默认值为 true。

显示 snackbar 或者 bottom sheet 的时候，需要使用当前的 BuildContext 参数调用 Scaffold.of 函数来获取 ScaffoldState 对象，然后使用 ScaffoldState.showSnackBar 和 ScaffoldState.showBottomSheet 函数来显示。

要特别注意 Scaffold.of 的参数 BuildContext， 如果包含该 BuildContext 的

Widget 是 Scaffold 的父 Widget，则 Scaffold.of 是无法查找到对应的 ScaffoldState 对象的，Scaffold.of 返回的是父对象中最近的 Scaffold 中的 ScaffoldState 对象。 比如，如果在 Scaffold 的 build 函数中，使用 build 的 BuildContext 参数是可以的：

```
@override
Widget build(BuildContext context) {
  return new RaisedButton(
    child: new Text('SHOW A SNACKBAR'),
    onPressed: () {
      Scaffold.of(context).showSnackBar(new SnackBar(
        content: new Text('Hello!'),
      ));
    },
  );
}
```

如果 build 函数返回一个 Scaffold 对象，则由于 Scaffold 对象是这个 Widget 的子对象，所以使用这个 build 的 BuildContext 参数是不能查找到 ScaffoldState 对象的，这个时候，通过在 Scaffold 中使用一个 Builder 来提供一个新的 BuildConext ：

```
@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text('Demo')
    ),
    body: new Builder(
      // Create an inner BuildContext so that the onPressed methods
      // can refer to the Scaffold with Scaffold.of().
      builder: (BuildContext context) {
        return new Center(
          child: new RaisedButton(
            child: new Text('SHOW A SNACKBAR'),
            onPressed: () {
              Scaffold.of(context).showSnackBar(new SnackBar(
                content: new Text('Hello!'),
              ));
            },
```

```
            ),
          );
        },
      ),
    );
}
```

另外还可以把 build 函数中的 Widget 分别创建，分别引入新的 BuildContext 来获取 Scaffold。

Appbar

AppBar 和 SliverAppBar 是Material Design中的 App Bar，也就是 Android 中的 Toolbar，关于 Toolbar 的设计指南请参考Material Design中 Toolbar 的内容。

AppBar 和 SliverAppBar 都是继承StatefulWidget 类，都代表 Toobar，二者的区别在于 AppBar 位置的固定的应用最上面的；而 SliverAppBar 是可以跟随内容滚动的。

他们的主要属性如下：

- leading：在标题前面显示的一个控件，在首页通常显示应用的 logo；在其他界面通常显示为返回按钮
- title： Toolbar 中主要内容，通常显示为当前界面的标题文字
- actions：一个 Widget 列表，代表 Toolbar 中所显示的菜单，对于常用的菜单，通常使用 IconButton 来表示；对于不常用的菜单通常使用 PopupMenuButton 来显示为三个点，点击后弹出二级菜单
- bottom：一个 AppBarBottomWidget 对象，通常是 TabBar。用来在 Toolbar 标题下面显示一个 Tab 导航栏
- elevation：纸墨设计中控件的 z 坐标顺序，默认值为 4，对于可滚动的 SliverAppBar，当 SliverAppBar 和内容同级的时候，该值为 0， 当内容滚动 SliverAppBar 变为 Toolbar 的时候，修改 elevation 的值

flexibleSpace：一个显示在 AppBar 下方的控件，高度和 AppBar 高度一样，可以实现一些特殊的效果，该属性通常在 SliverAppBar 中使用

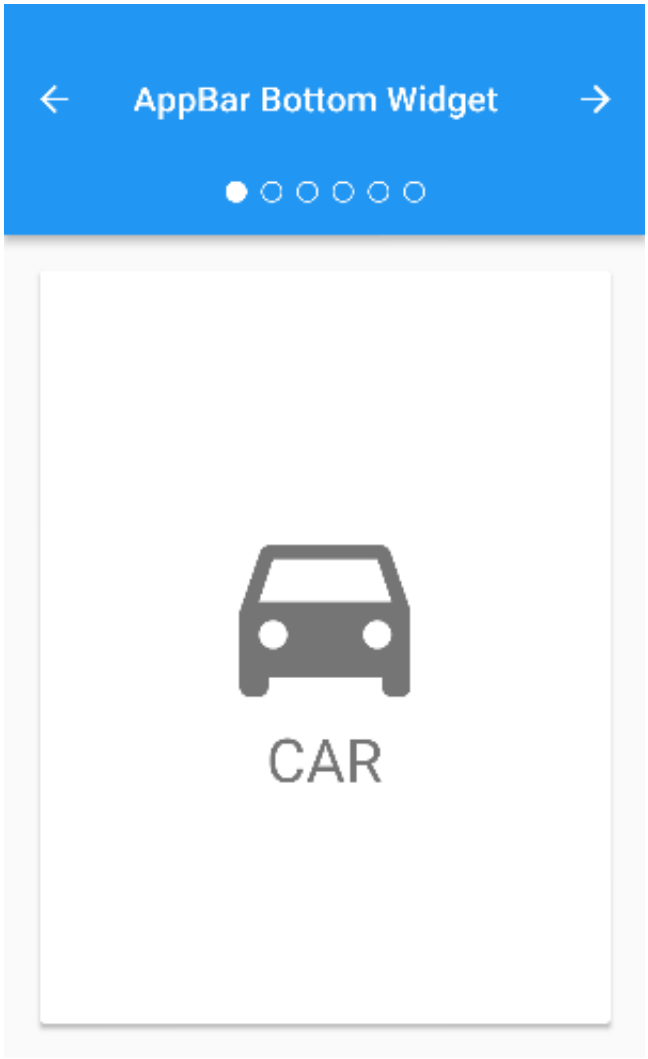
- backgroundColor：APP bar 的颜色，默认值为 ThemeData.primaryColor。改值通常和下面的三个属性一起使用
- brightness：App bar 的亮度，有白色和黑色两种主题，默认值为 ThemeData.primaryColorBrightness
- iconTheme：App bar 上图标的颜色、透明度、和尺寸信息。默认值为 ThemeData.primaryIconTheme
- textTheme： App bar 上的文字样式。默认值为 ThemeData.primaryTextTheme
- centerTitle： 标题是否居中显示，默认值根据不同的操作系统，显示方式不一样

```
import 'package:flutter/material.dart';

class AppBarBottomSample extends StatefulWidget {
  @override
  _AppBarBottomSampleState createState() => new _AppBarBottomSampleState();
}

class _AppBarBottomSampleState extends State<AppBarBottomSample> with Singl
  TabController _tabController;

  @override
  void initState() {
    super.initState();
    _tabController = new TabController(vsync: this, length: choices.length)
  }

  @override
  void dispose() {
    _tabController.dispose();
    super.dispose();
  }
```

```
void _nextPage(int delta) {
  final int newIndex = _tabController.index + delta;
  if (newIndex < 0 || newIndex >= _tabController.length)
    return;
  _tabController.animateTo(newIndex);
}

@override
Widget build(BuildContext context) {
  return new MaterialApp(
    home: new Scaffold(
      appBar: new AppBar(
        title: const Text('AppBar Bottom Widget'),
        leading: new IconButton(
          tooltip: 'Previous choice',
          icon: const Icon(Icons.arrow_back),
          onPressed: () { _nextPage(-1); },
        ),
        actions: <Widget>[
          new IconButton(
            icon: const Icon(Icons.arrow_forward),
            tooltip: 'Next choice',
            onPressed: () { _nextPage(1); },
          ),
        ],
        bottom: new PreferredSize(
          preferredSize: const Size.fromHeight(48.0),
          child: new Theme(
            data: Theme.of(context).copyWith(accentColor: Colors.white),
            child: new Container(
              height: 48.0,
              alignment: Alignment.center,
              child: new TabPageSelector(controller: _tabController),
            ),
          ),
        ),
      ),
      body: new TabBarView(
        controller: _tabController,
        children: choices.map((Choice choice) {
          return new Padding(
            padding: const EdgeInsets.all(16.0),
            child: new ChoiceCard(choice: choice),
          );
        }).toList(),
```

```
          ),
        ),
      );
    }
  }

  class Choice {
    const Choice({ this.title, this.icon });
    final String title;
    final IconData icon;
  }

  const List<Choice> choices = const <Choice>[
    const Choice(title: 'CAR', icon: Icons.directions_car),
    const Choice(title: 'BICYCLE', icon: Icons.directions_bike),
    const Choice(title: 'BOAT', icon: Icons.directions_boat),
    const Choice(title: 'BUS', icon: Icons.directions_bus),
    const Choice(title: 'TRAIN', icon: Icons.directions_railway),
    const Choice(title: 'WALK', icon: Icons.directions_walk),
  ];

  class ChoiceCard extends StatelessWidget {
    const ChoiceCard({ Key key, this.choice }) : super(key: key);

    final Choice choice;

    @override
    Widget build(BuildContext context) {
      final TextStyle textStyle = Theme.of(context).textTheme.display1;
      return new Card(
        color: Colors.white,
        child: new Center(
          child: new Column(
            mainAxisSize: MainAxisSize.min,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
              new Icon(choice.icon, size: 128.0, color: textStyle.color),
              new Text(choice.title, style: textStyle),
            ],
          ),
        ),
      );
    }
  }
```

```
void main() {
  runApp(new AppBarBottomSample());
}
```



FlutterLogo

定义flutter应用的logo，该控件受IconTheme约束。

```
import 'package:flutter/material.dart';

void main() {
  runApp(new FadeAppTest());
}

class FadeAppTest extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
```

```
    return new MaterialApp(
      title: 'Fade Demo',
      theme: new ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: new MyFadeTest(title: 'Fade Demo'),
    );
  }
}

class MyFadeTest extends StatefulWidget {
  MyFadeTest({Key key, this.title}) : super(key: key);
  final String title;
  @override
  _MyFadeTest createState() => new _MyFadeTest();
}

class _MyFadeTest extends State<MyFadeTest> with TickerProviderStateMixin {
  AnimationController controller;
  CurvedAnimation curve;

  @override
  void initState() {
    controller = new AnimationController(duration: const Duration(milliseco
    curve = new CurvedAnimation(parent: controller, curve: Curves.easeIn);
  }

  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text(widget.title),
      ),
      body: new Center(
          child: new Container(
              child: new FadeTransition(
                  opacity: curve,
                  child: new FlutterLogo(
                    size: 100.0,
                  )))),
      floatingActionButton: new FloatingActionButton(
        tooltip: 'Fade',
        child: new Icon(Icons.brush),
        onPressed: () {
          controller.forward();
```

```
          },
        ),
      );
    }
}
```

Placeholder

占位控件，该控件绘制一个框，表示将来会在该位置添加其他控件。

这个控件在开发过程中很有用，可提示该处接口还没完成。

默认情况下，控件的大小自适应其容器。如果该控件处于无界空间，它将根据给定的fallbackWidth和fallbackHeight自行调整大小。

详见：https://docs.flutter.io/flutter/widgets/Placeholder-class.html