

Flutter 布局 —— 理解 BoxConstraint（盒约束）布局 模型

前面讲了 Flutter 一些 UI Widget 的使用，但实际开发中，我们还需要将这些 Widget 按照一定的布局组合起来，Flutter 的布局和 Android、iOS、WEB 的布局有相似之处，也有不同之处。

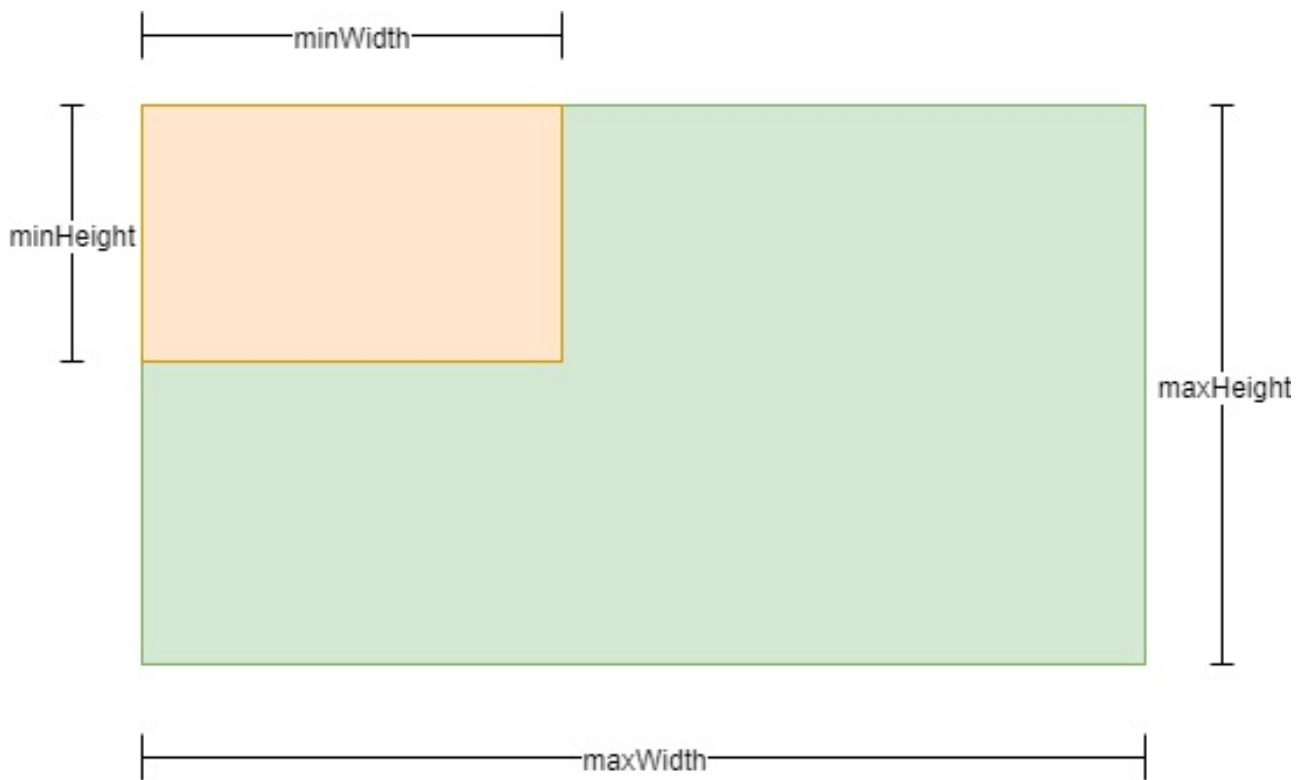
所以这节，主要讲一下 Flutter 的布局原理：BoxConstraints（盒约束）。

什么是 BoxConstraints（盒约束）？

BoxConstraints 翻译过来是 盒约束，用于指定 Widget 大小的约束。

Flutter 的 Widget 都是通过 BoxConstraints 来约束大小的。

BoxConstraints 如何来约束Widget的大小？



如上图，以左上角为原点，在 X轴 和 Y轴 上设置最小值和最大值,就对应了 BoxConstraints 的四个属性：

1. minWidth
2. maxWidth
3. minHeight
4. maxHeight

这四个属性可以确定如下的关系：

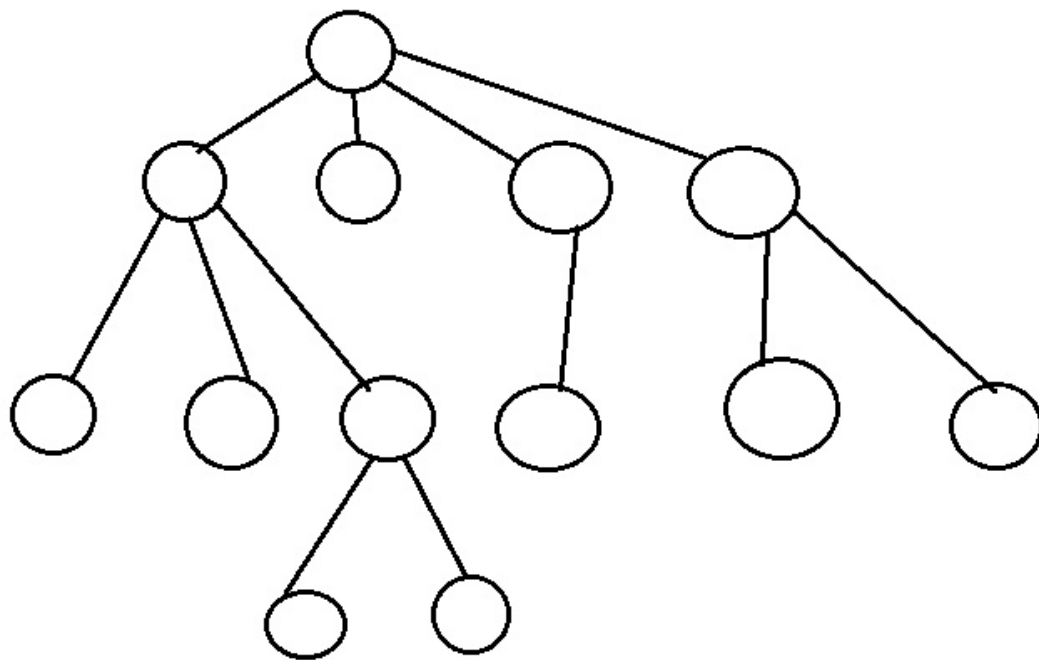
$$0.0 \leq \text{minWidth} \leq \text{Widget宽度的实际大小} \leq \text{maxWidth} \leq \text{double.infinity}$$
$$0.0 \leq \text{minHeight} \leq \text{Widget高度的实际大小} \leq \text{maxHeight} \leq \text{double.infinity}$$

这样就会形成一个矩形的范围：不小于黄色的矩形，不大于绿色的矩形，这样就确定了一个 Widget 的大小范围，

Flutter 采用 BoxConstraints 是为了更好的适配多种平台。

Flutter 如何确定 Widget 的大小?

前面讲的 BoxConstraints，只能确定 Widget 大小的一个范围，并不能知道 Widget 的确切大小，这样是无法绘制的，那么 Flutter 是如何确定 Widget 大小的呢？



这是一个Widget树的结构，为了确定 Widget 的大小：

1. 父Widget 会将自己的 BoxConstraints 传递给自己的子Widget，直到传到树的叶节点（没有子节点了），因此树的叶节点会拥有所有父节点的 BoxConstraints，这些 BoxConstraints 会相互影响。
2. 叶节点的Widget 会根据这些 BoxConstraints 和自己的 BoxConstraints 计算出自己的大小，并将自己的大小返回给父 Widget。
3. 父Widget 得到 子Widget 的大小后，根据 子Widget 的大小、前面 父Widget 们传递的 BoxConstraints 和自己的 BoxConstraints，同样可以计算出自己的大小，这样依次往上

传递至 根Widget。

所以 Widget 的大小是受其 父Widget 的约束和自己 子Widget 的大小共同影响的

BoxConstraints 的种类

对 BoxConstraints 的四个属性赋不同的值，会有不同的约束效果，为了更直观的了解这些约束效果，写了一个简单的 demo：

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

void main() => runApp(MyApp("Hello World"));

class MyApp extends StatefulWidget {
  // This widget is the root of your application.

  String content;

  MyApp(this.content);

  @override
  State<StatefulWidget> createState() {
    // TODO: implement createState
    return MyAppState();
  }
}

class MyAppState extends State<MyApp> {
  void increment() {
    setState(() {
      widget.content += "d";
    });
  }
}
```

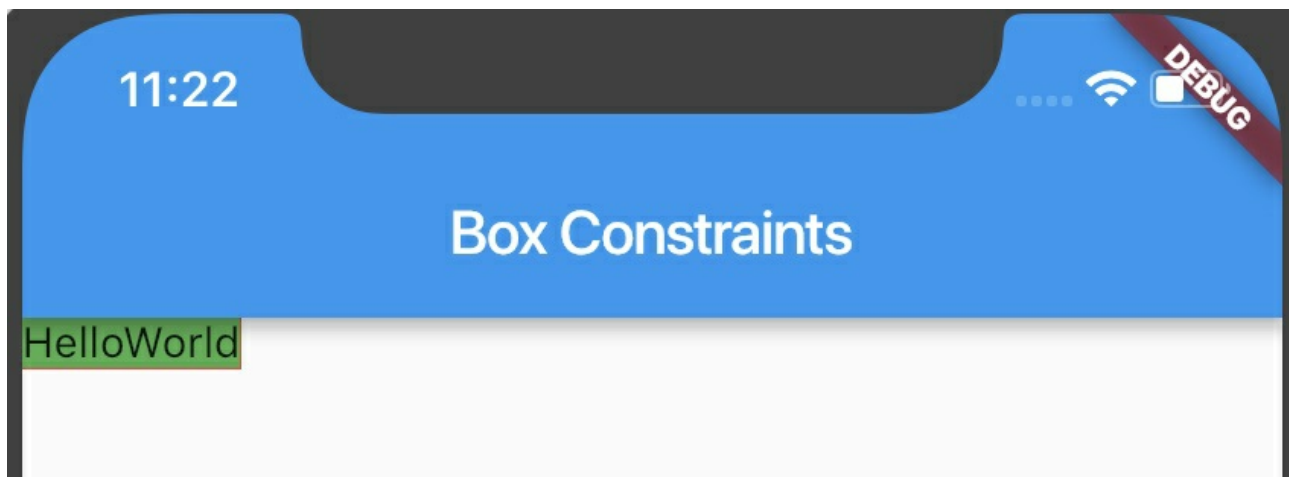
```

}

@override
Widget build(BuildContext context) {
  Paint paint = Paint();
  paint.color = Colors.green;
  return MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: Scaffold(
      appBar: AppBar(
        title: Text("Box Constraints"),
      ),
      body: Container(
        color: Colors.red,
        child: Text(
          "HelloWorld",
          style: TextStyle(background:
paint),
        ),
      ),
    ),
  );
}
}

```

运行的效果如下：



这个例子里有两个 Widget: Container 和 Text, 绿色代表的是 Text 所占的位置, Text 下面还有一层红色, 是 Container 所在的位置, 这里不太明显, 因为 Container 的大小和 Text 的大小是一样的。

下面对 Text 和 Container 设置不同的 BoxConstraints 来研究 BoxConstraints:

- Tightly Constraints (严格约束)
- Loose Constraints (松散约束)
- Bounded Constraints (有界约束)
- Unbounded Constraints (无界约束)
- Infinite Constraints (无限约束)

Tightly Constraints (严格约束)

当某一轴上的最小值和最大值相同时, 那么这个轴上的值就确定了, 就是 Tightly Constraints (严格约束)。

- 当 $\text{minWidth} = \text{maxWidth}$ 时, Widget 的宽就确定了
- 当 $\text{minHeight} = \text{maxHeight}$ 时, Widget 的高度就确定了
- 当 $\text{minWidth} = \text{maxWidth}$ 且 $\text{minHeight} = \text{maxHeight}$, Widget 的宽高就都确定了。

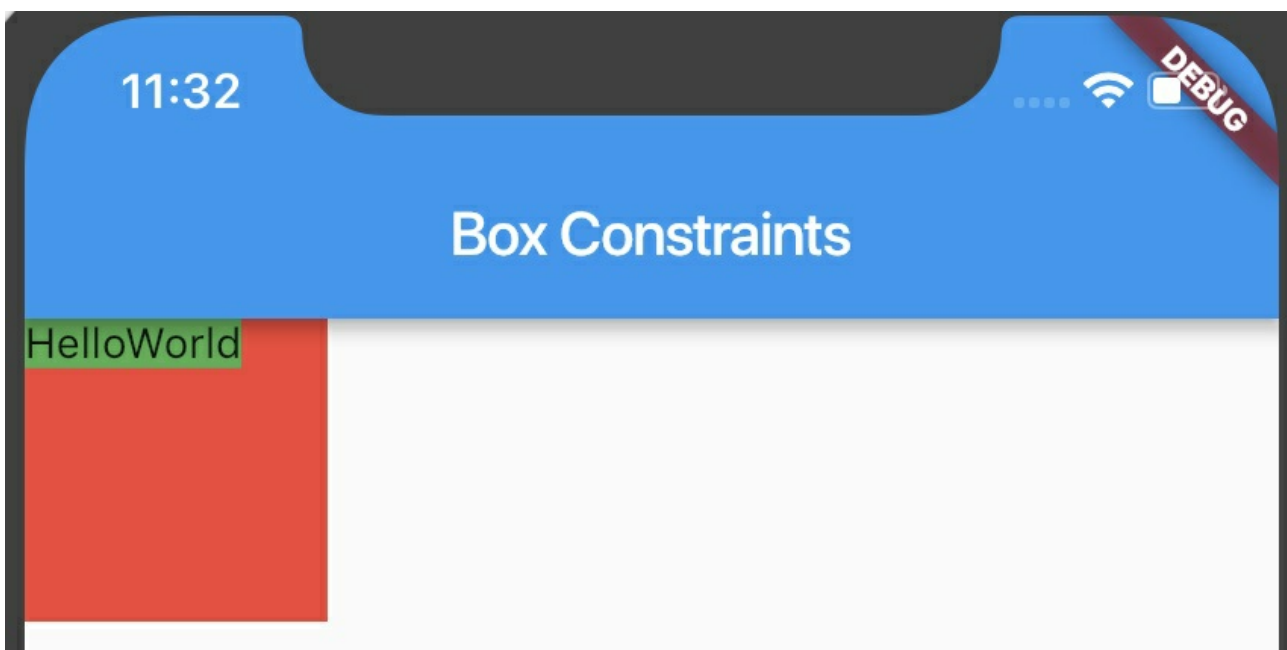
新建一个 Tightly Constraints, 使用如下的方法:

```
BoxConstraints.tight(Size(width, height))
```

给 Container 加一个 Tightly Constraints:

```
body: Container(  
  constraints: BoxConstraints.tight(Size(100,  
100)), //添加 Tightly Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background: paint),  
  )),
```

运行结果如下:



代表 Container 的红色就是固定大小。

当 Widget 是 Tightly Constraints 时, 它的宽或高是固定的。

Loose Constraints (松散约束)

当某一轴上的最小值为 0 时，就是 Loose Constraints（松散约束）。

这时候就有两种情况：

1. 最大值是确定的值

- Container 的 X 轴最小值是 0，假设最大值是 100，那么你想一下，红色矩形的宽度会比绿色的宽吗？

答案是不会，因为 Container 的宽度最小为 0，最大为 100，如果子Widget 的宽度大于 100，那么 Container 的宽度最高只能为 100，如果子Widget 的宽度小于 100，那么 Container 的宽度就和子Widget 的宽度一样。

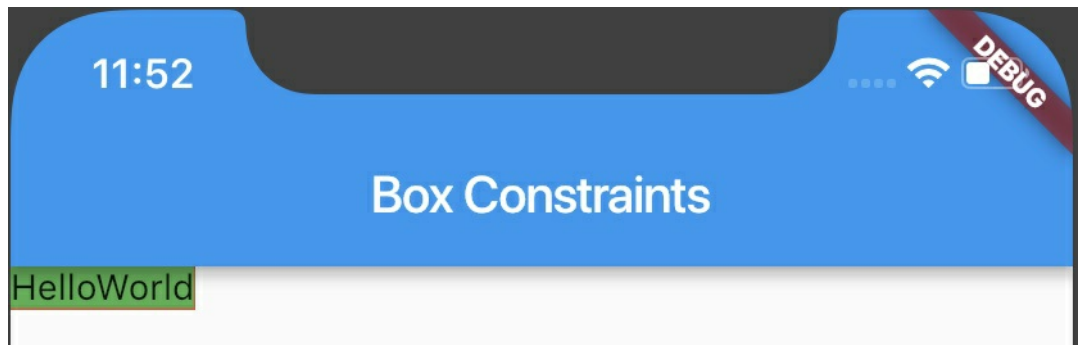
为了验证结果，新建一个最大值是确定值的松散约束：

```
BoxConstraints.loose(Size(width, height))
```

代码是：

```
body: Container(  
  constraints:  
    BoxConstraints.loose(Size(100, 100)), //  
    添加 Loose Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background:  
      paint),  
  )),  
));
```

运行结果如下：

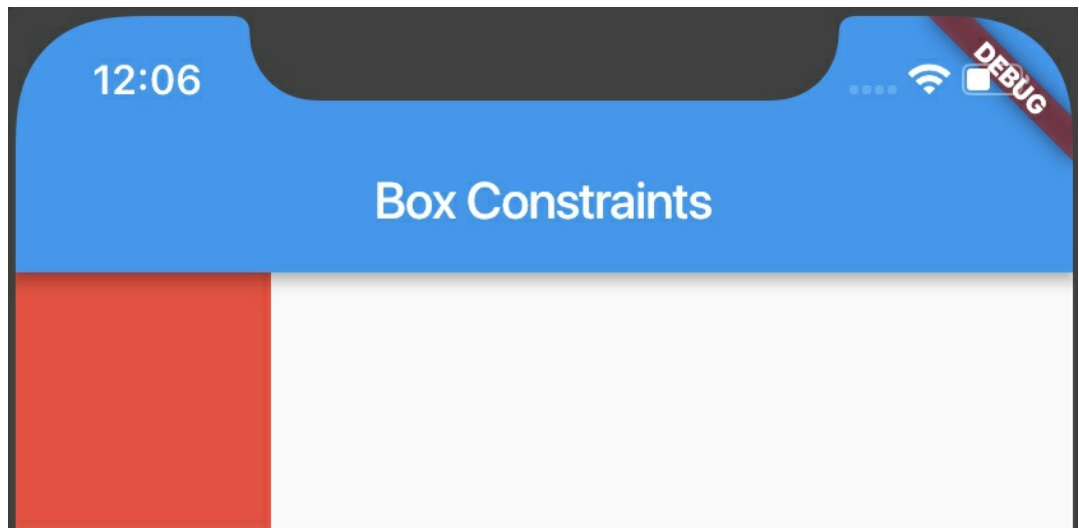


- 接着上一个假设，给 Container 设定了最大值是确定值的松散约束，但是 Container 没有子Widget，红色矩形又是怎么显示的呢？

代码如下：

```
body: Container(  
  constraints:  
    BoxConstraints.loose(Size(100, 100)), //  
    添加 Loose Constraints  
  color: Colors.red,));
```

运行效果如下：



这时候 Container 显示的是最大宽度。

2. 最大值是 Infinite(无限值)

- Container 的 X 轴最小值是 0，假设最大值是 Infinite(无限值)，红色矩形和绿色矩形又是什么关系呢？

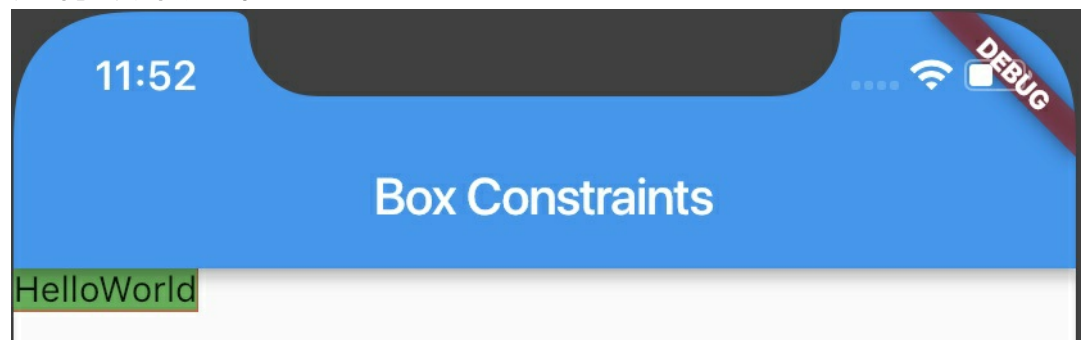
新建一个最大值是无限值的松散约束：

```
BoxConstraints.tightFor()
```

代码如下：

```
body: Container(  
  constraints:  
    BoxConstraints.tightFor(), //添加 Loose  
    Constraints  
    color: Colors.red,  
    child: Text(  
      "HelloWorld",  
      style: TextStyle(background:  
paint),  
    )),  
  ));
```

运行效果如下：



- 接着上一个，如果 Container 没有 子Widget 呢？

代码如下：

```
body: Container(  
  constraints:  
    BoxConstraints.tightFor(), //添加 Loose  
    Constraints  
    color: Colors.red,)))
```

运行效果如下：

12:03



Box Constraints



Container 铺满了全屏

当 Widget 是 Loose Constraints 时，无论最大值是确定值还是无限制，如果有子Widget，那么 Widget 的大小就是子Widget 的大小，相当于 wrap_content，如果没有子Widget，那么 Widget 就是最大值，如果这个最大值是 Infinite,就相当于 match_parent

Bounded Constraints (有界约束)

当某一轴上的最大值是确定的值时，就是 Bounded Constraints (有界约束)。

新建一个 Bounded Constraints,maxWidth和maxHeight必须为固定值：

```
constraints:  
BoxConstraints(minWidth,maxWidth,minHeight,maxHeight), //添加 Bounded Constraints
```

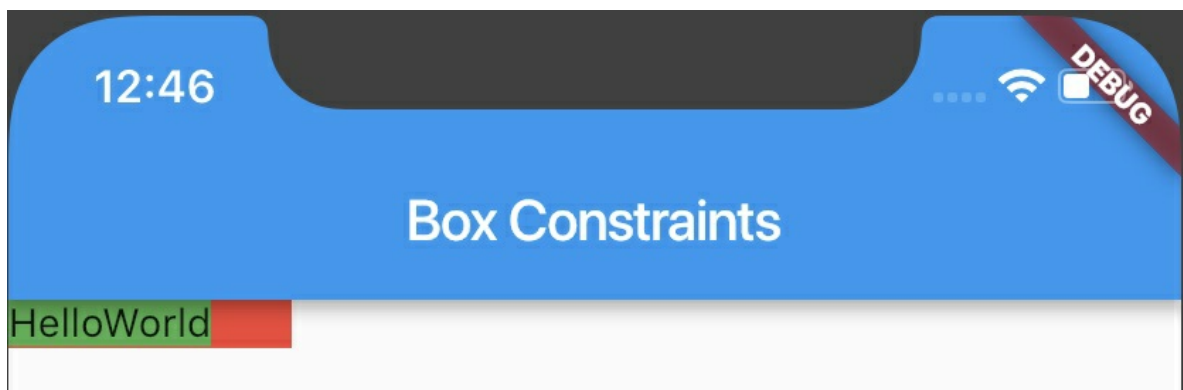
然后考虑如下的情形：

- 如果 minWidth 大于子 Widget 的宽

如下代码：

```
body: Container(  
  constraints: BoxConstraints(minWidth:  
100,maxWidth: 300,minHeight: 0,maxHeight:  
300), //添加 Bounded Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background: paint),  
  ));
```

运行效果如下：



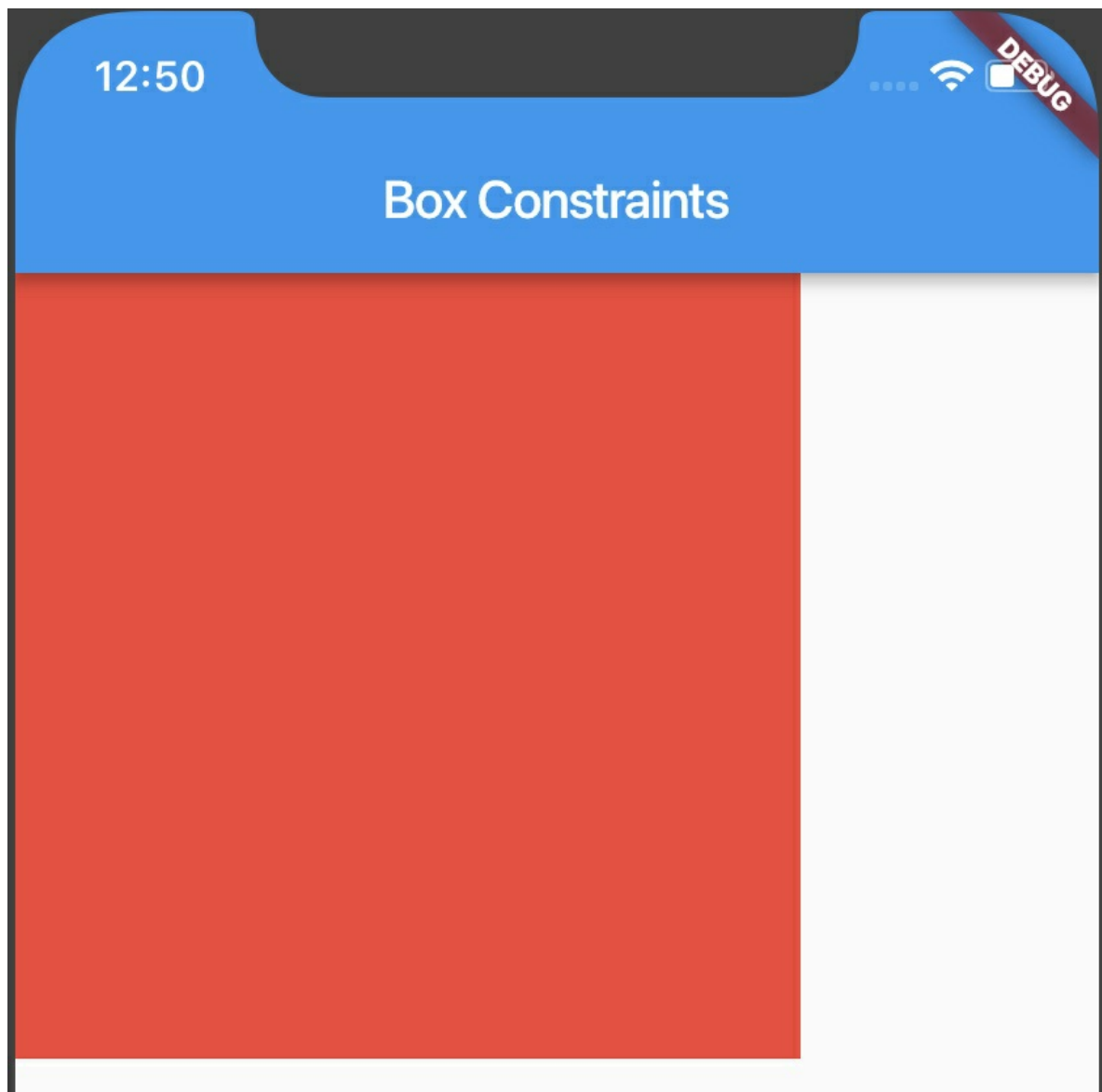
Container 的宽度取的是最小值 100，而不是最大值 300

- 如果 Container 没有 子Widget 呢？

如下的代码：

```
body: Container(  
  constraints: BoxConstraints(minWidth:  
100,maxWidth: 300,minHeight: 0,maxHeight:  
300), //添加 Bounded Constraints  
  color: Colors.red,));
```

运行效果如下：



这时候 Container 显示的是最大值。

当 Widget 是 Bounded Constraints 时,如果有 子 Widget, 子Widget 的大小小于有界约束的最小值, 则显示的是有界约束的最小值, 而当 子Widget 的大小大于有界约束的最小值, 小于有界约束的最大值, 则显示的是 子Widget 的大小, 否则显示的是有界约束的最大值; 如果没有 子Widget 就显示有界约束的最大值

Unbounded Constraints (无界约束)

当某一轴上的最大值是 Infinite(无限值) 时, 就是 Unbounded Constraints (无界约束)。

新建一个 Unbounded Constraints,maxWidth 和 maxHeight 必须为 double.infinity:

```
constraints:  
BoxConstraints(minWidth,maxWidth,minHeight,maxHeight), //添加 Bounded Constraints
```

然后考虑如下的情形:

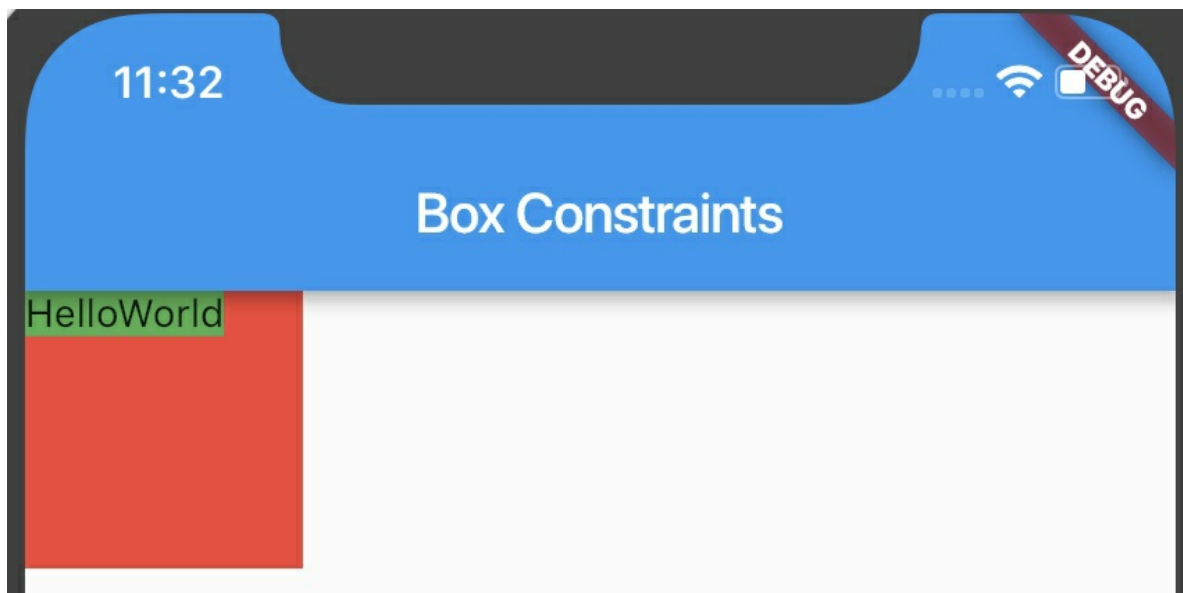
- 当子Widget的宽小于 minWidth

红色矩形显示的宽度肯定是 minWidth。

代码如下:

```
body: Container(  
  constraints: BoxConstraints(minWidth:  
100,maxWidth: double.infinity,minHeight:  
100,maxHeight: double.infinity), //添加  
Unbounded Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background: paint),  
  )),);
```

运行效果如下:



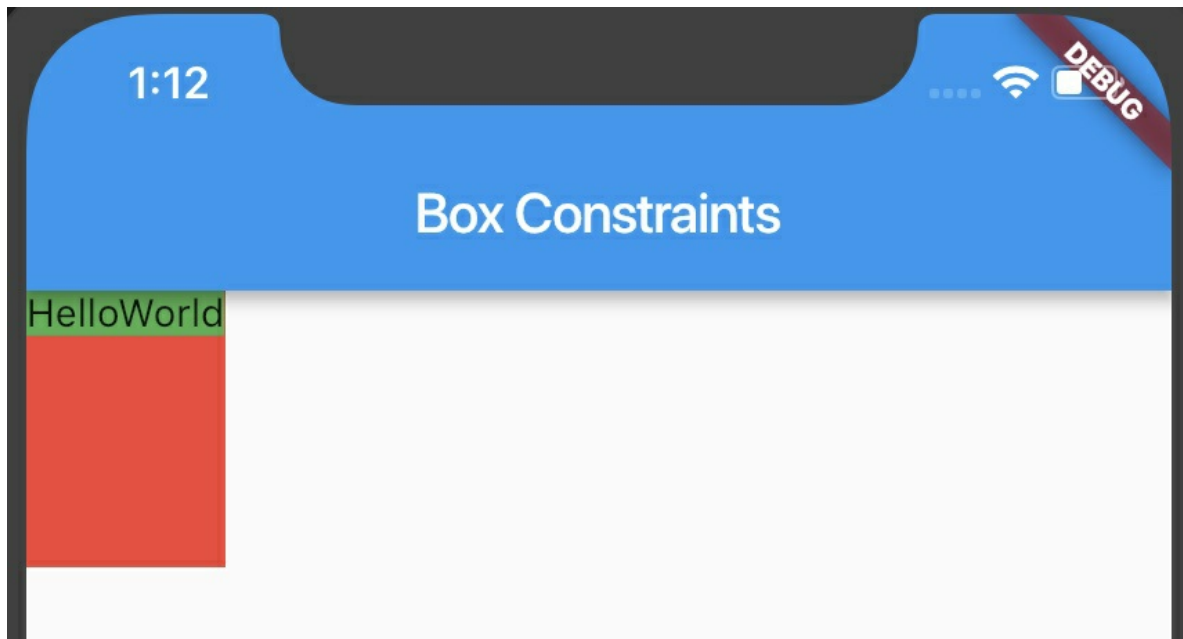
- 当子Widget的宽大于 minWidth

红色矩形显示的宽度肯定是子Widget的宽度。

代码如下：

```
body: Container(  
  constraints: BoxConstraints(minWidth:  
10,maxWidth: double.infinity,minHeight:  
100,maxHeight: double.infinity), //添加  
Unbounded Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background: paint),  
  )),
```

运行效果如下：

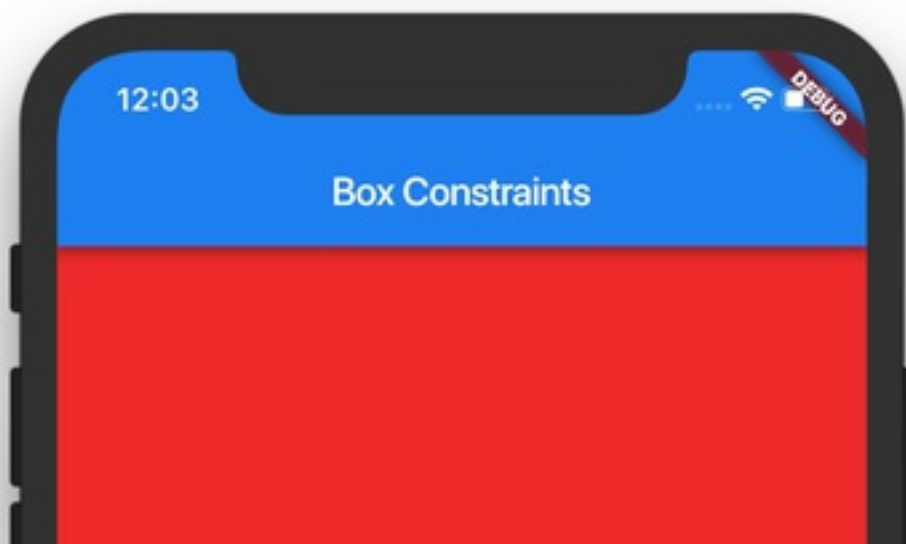


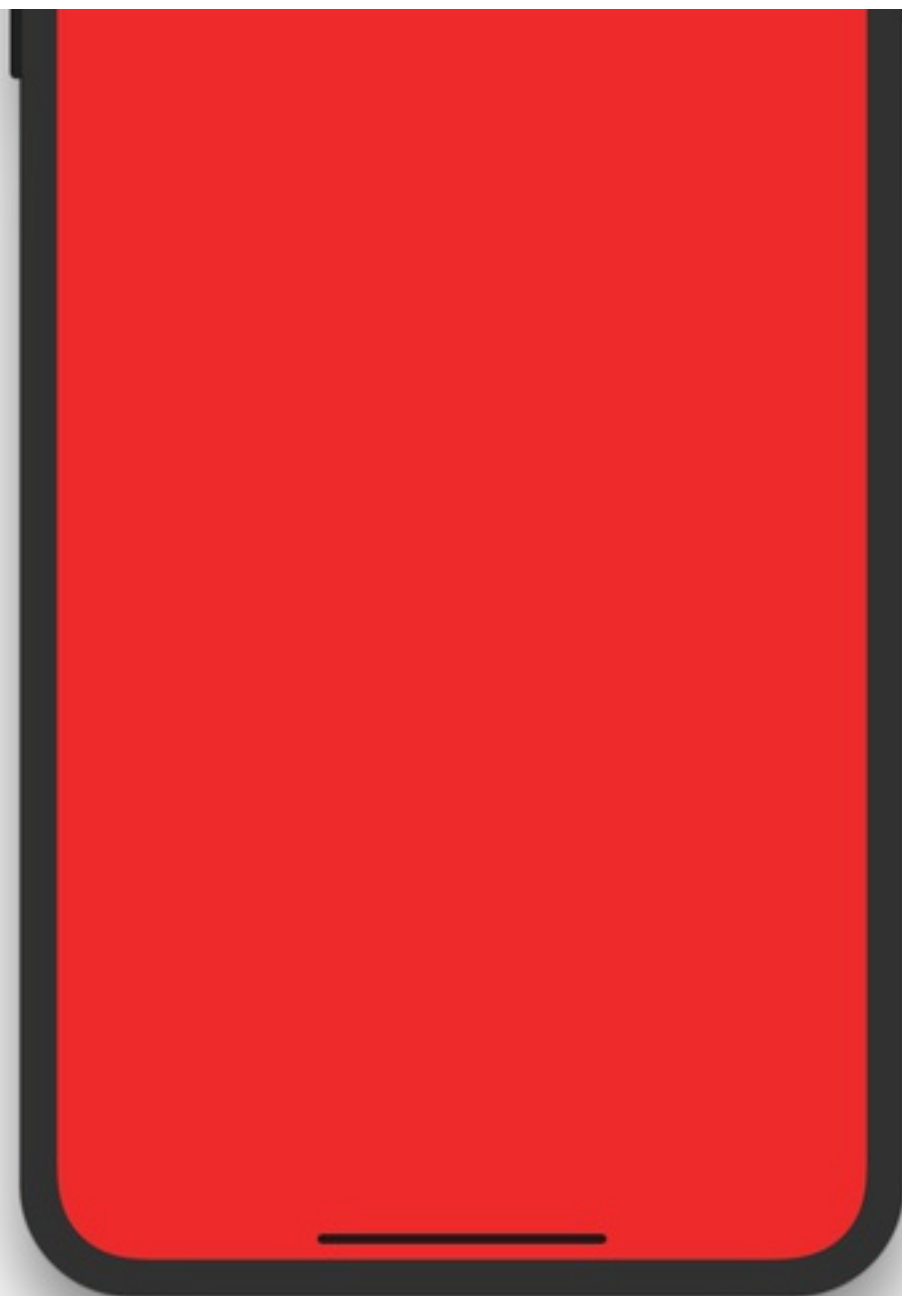
- 如果 Container 没有 子Widget 呢?

如下的代码:

```
body: Container(  
  constraints: BoxConstraints(minWidth:  
10,maxWidth: double.infinity,minHeight:  
100,maxHeight: double.infinity), //添加  
Unbounded Constraints  
  color: Colors.red,))));
```

运行效果如下:





Container 铺满了全屏

其实 Unbounded Constraints 很像 Loose Constraints。当 Unbounded Constraints 有子Widget 时，若子Widget 的大小小于 Unbounded Constraints 的最小值时，则 Widget 显示 Unbounded Constraints 的最小值，若子Widget 的大小大于 Unbounded Constraints 的最小值时，则显示子Widget 的大小；若没有子Widget，就相当于 match_parent

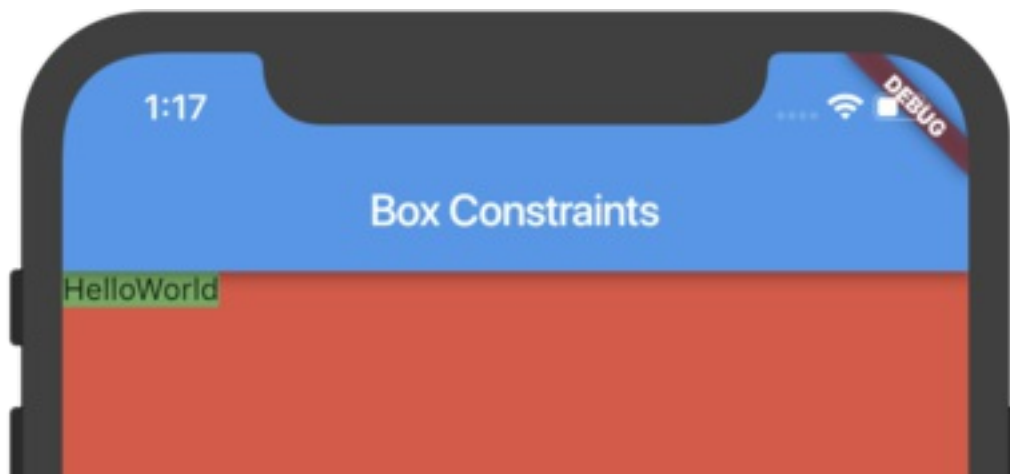
Infinite Constraints（无限约束）

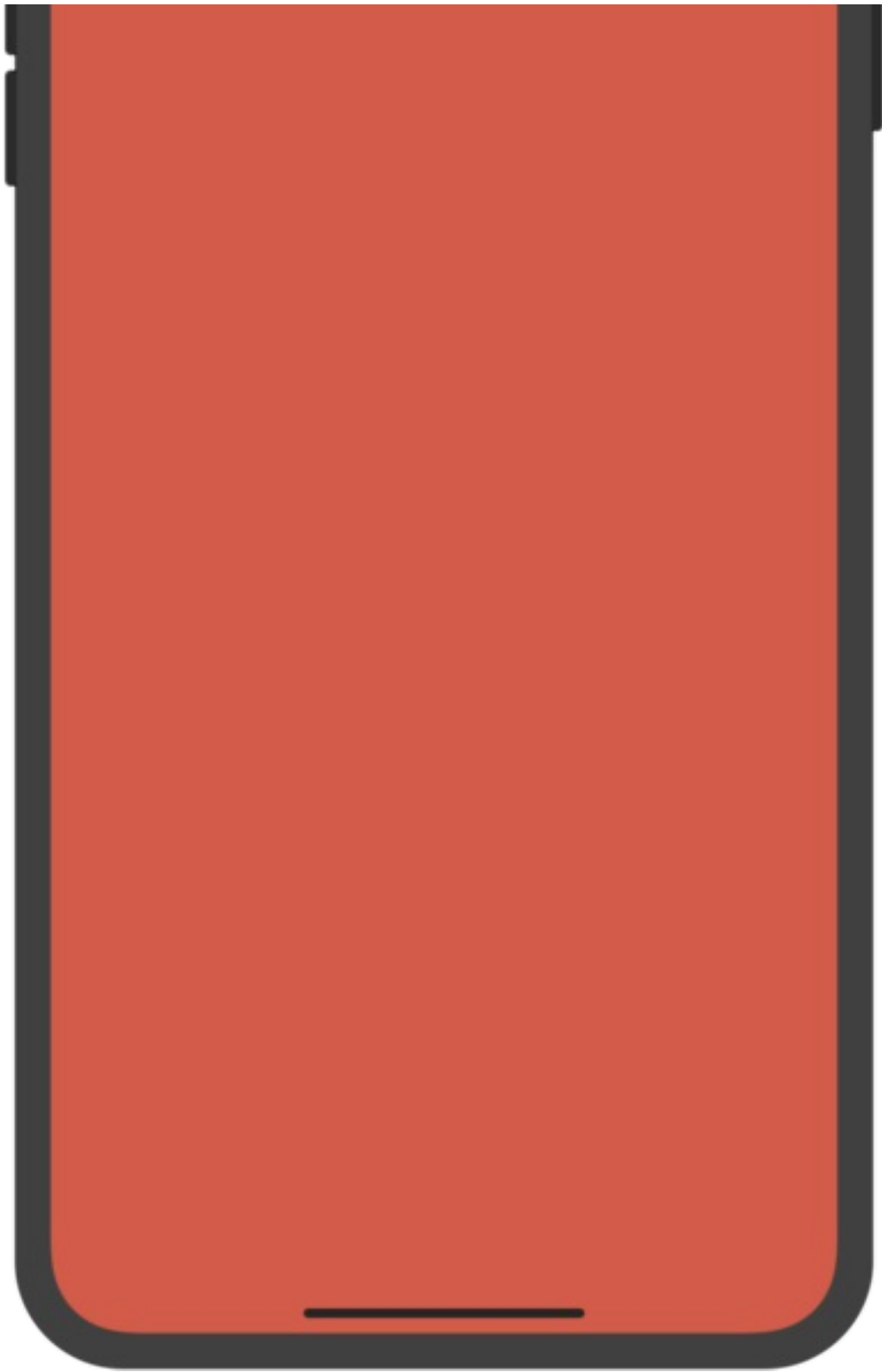
当某一轴上的最大值和最小值都是 Infinite(无限值) 时，就是 Infinite Constraints（无限约束）。

新建一个 Infinite Constraints（无限约束）：

```
body: Container(  
  constraints: BoxConstraints.expand(), //添加  
  Infinite Constraints  
  color: Colors.red,  
  child: Text(  
    "HelloWorld",  
    style: TextStyle(background: paint),  
  ));
```

运行效果如下：





Infinite Constraints 就相当于 match_parent