任何的现代Gui基本上都有控件的概念，有了控件就有了基于控件的核心调度，事件能够正确的通过特定的流程，被转发到控件的回调中去响应并处理，是设计的必须，同样的，控件处于一个控件树中，它有着特殊的相对关联关系，通常控件是以面向对象的设计思想去构建的，但是C语言本身语法不支持面向对象编程，所以有很多面向对象的要求，便根据自身的需要去合理的考虑即可，同样的嵌入式Gui的对象理论上也不需要有太复杂的继承关系，你可以考虑简单的一集继承（从基类到派生类），复杂的多级继承则根据实际需要考虑即可（派生类到派生类）

**参考设计属性（控件：基类）：**

控件类型

控件状态

控件事件响应表

控件父节点关联

控件孩子列表关联

控件的画布

控件管理区域

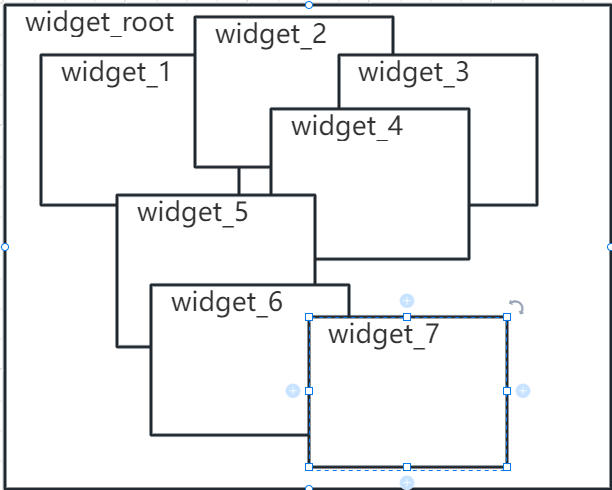
控件一般具有父子管理表，当前控件既可以向上迭代到父亲，又可以向下迭代到孩子，控件树在拓扑图中是一颗多叉树，一个控件有0个或多个孩子，有一个父亲，父子关系是双向关联的，那么控件在响应事件时，则根据事件特性从上至下的深度优先，或者从下至上的回溯递归，都可以

控件的画布便是控件的绘制上下文，它可以是根控件画布的一个引用，也可以为自己额外私开一块小画布管理从它自己的绘制，最终目标是尽可能的使用硬件加速去进行处理，将复杂的图形缓存到画布中，将最终的绘制变成一个贴图的过程，以提升性能

控件可以有一些基础的描述，如控件的背景色或者背景图，复杂的Gui框架如lvgl的基础控件还支持绘制圆角，边框，外部线，阴影等信息，通常来说嵌入式Gui不需要搞这么复杂，因为如上述的介绍是不能硬件加速的，而且很少有控件需要同时绘制这么多的底部信息，通常有一个背景色已经完全足够了

控件一般有一个区域坐标，描述控件在控件树中的空间布局关系，表明控件所管理的一块区域，子控件区域超出父控件区域要进行裁剪，这一部分我们也可以简单的理解为控件的剪切域，但是与绘制不一样的是，这块剪切域是虚拟的，作用目标不一致，控件剪切域主要承担描述的作用，而绘制剪切域则承担裁剪的作用，它们之间可以相互转换，不过这部分内容通常较为复杂，目前还没有一个简单的描述

例(控件树的调度流程):



假设我现在有一个以上的控件树:  
 每一个框代表一个控件，它们有一个自己的区域，我们看到，按编号大小，前一个控件都被后一个给盖住了一部分，我们分析一下如果要进行Gui的控件绘制流程时，步骤应该是如何的。首先我们需要先绘制背景，也就是widget\_root，然后按照widget\_1到widget\_7的顺序从前向后绘制，这样子每绘制后一个控件就会覆盖前面绘制的一部分，保证我们看到的效果是完整的。很显然这个逻辑就清晰下来了，那么这个流程是否可以递归的进行下去呢？假如说widget\_7里面又有几个孩子，或者说这个控件树的子控件还有孩子？答案是显然的，这样子我们就得到了一个唯一的绘制流程，这也符合ui工程师的使用，将控件看作层级，则根控件就是最下面的一层，根控件的子控件们就是相对根控件上一层。

考虑一个这样子的控件树：

Widget\_r

Widget\_l1

Widget\_l1\_l1

Widget\_l1\_l2

Widget\_l2

Widget\_l2\_l1

Widget\_l2\_l2

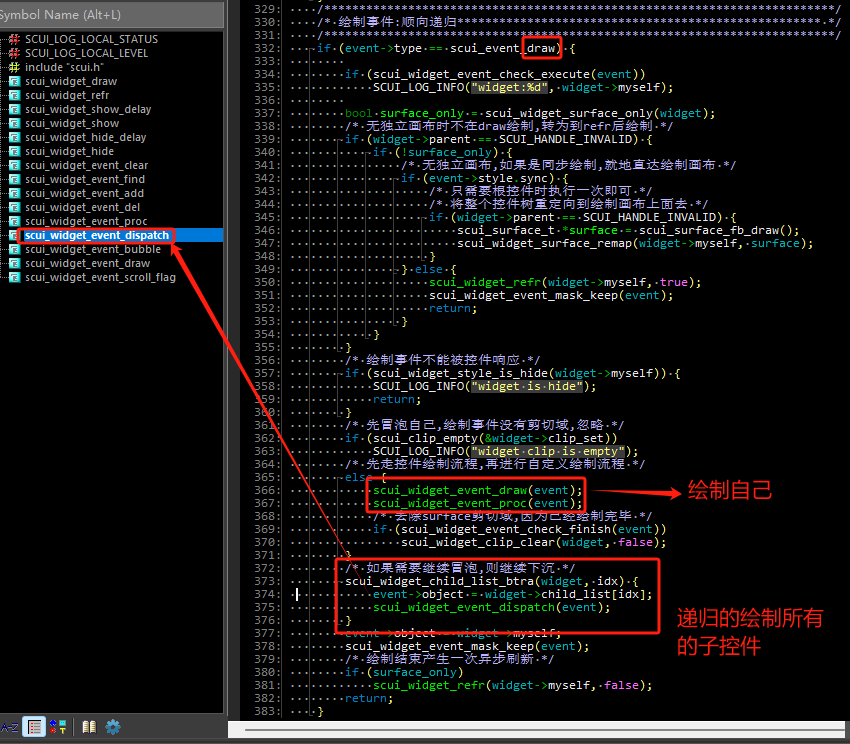
假如说, Widget\_l1被Widget\_l2覆盖住了, 那么Wigdet\_l1的孩子是否可以被Widget\_l2覆盖呢？假设它们绘制区域存在交集的话？很显然是可以的，在控件树的空间布局中，父子关系是一种约束，意味着父控件被覆盖了，那么子控件也会被覆盖，因为不存在父控件完全被盖住的情况下子控件还能显示，子控件永远只能显示在父控件所在的区域范围内部，这个语义生成的结果是完备的

所以简单总结一下，控件树响应绘制事件的处理流程：

先绘制自己

然后按子控件的先后顺序遍历

递归的绘制子控件



大道至简，Gui的控件树的绘制流程实际上就是 数据结构 中的 深度优先搜索的 原理，我们将其中的范围替换成绘制动作，就得到了一个通用的控件树的绘制流程，与这个相对于的流程便是逆向回溯递归，那么响应ptr事件则就需要使用到这种流程去处理，因为你点击是直接以命中屏幕上显示的控件为目标的

同样的所有的Gui的控件树的调度逻辑，基本上都可以被分为 尾递归(深度优先搜索) 和 回溯递归 这俩大类别。