自动布局原理

自动布局本质上就是封装了各种更加便捷的相对布局属性，结合失效验证机制(../../../../extension/EUI/autoLayout/FailureToVerify/index.html)，在合适的触发条件下(如尺寸发生改变时)，自动设置相关对象的 `x` ， `y` ， `width` ， `height` 等属性。所以无论过程是怎么样的，最终结果都是直接体现在 `x` ， `y` ， `width` ， `height` 这些最原始的属性上，并没有脱离显示对象原始的API。

在上一节内容中，简单介绍了跟布局有关的两个方法： `measure()` 和 `updateDisplayList()` 。本节详细说明这个两个方法：

`updateDisplayList()` 方法负责布局子项(即设置子项的 `x` ， `y` ， `width` ， `height`),或做一些矢量重绘操作。 `measure()` 自动测量出当前适合子项的相关属性(`width` ， `height` 等)。

举个例子：有一个容器(Group)，里面放一个单行文本(Label)，想要文本始终在容器中水平居中(horizotalCenter = 0)。不需要显式设置文本的 `width` ，这时 `measure()` 会在文本内容改变时，自动测量出当前合适的 `width` ，父级就会根据这个 `width` ，重新布局它的 `x` ， `y` 。

EUI 里所有的组件都是这样：如果不显式设置宽高，就调用 `measure()` 方法测量出一个最佳尺寸，在没有被父级强制布局情况下，这个值最终赋值给 `width` 和 `height` 属性。反之，如果显式设置了组件的 `width` 或 `height` 属性， `width` 或 `height` 就使用显式设置的值。若组件同时被设置了 `width` 和 `height` ， `measure()` 方法将不会被调用。至于何为测量的”最佳尺寸”？不同的组件有各自的测量方式，容器是能刚好放下所有子项的尺寸，文本是能完整显式文本内容的尺寸，而Image则是内部要显示的位图素材的尺寸。还有其他的组件，具体在各自的 `measure()` 方法里实现。多个组件都需要测量的时候，会按照显式列表深度，从内向外测量。而布局阶段正好相反，从外向内。具体细节参考失效验证机制(../../../../extension/EUI/autoLayout/FailureToVerify/index.html)的内容。

总之，如果希望自定义的组件像框架里的标准组件一样，能加入自动布局体系，就必须要同时重写 `measure()` 和 `updateDisplayList()` 方法。

`Group` 是容器基类。它不负责具体的布局规则，而是做了一个解耦处理。增加 `layout` 属性，类型为 `LayoutBase` 。 `Group` 中的 `measure()` 和 `updateDisplayList()` 方法的内容为：

```
    protected measure():void {
        if (!this.$layout) {
            this.setMeasuredSize(0, 0);
            return;
        }
        this.$layout.measure();
    }
    protected updateDisplayList(unscaledWidth:number, unscaledHeight:number):void {
        if (this.$layout) {
            this.$layout.updateDisplayList(unscaledWidth, unscaledHeight);
        }
        this.updateScrollRect();
    }
```

`Group` 把自己的 `measure()` 方法交给 `layout.measure()` 实现， `updateDisplayList()` 交给 `layout.updateDisplayList()` 实现。也就是把具体的布局方式解耦出来，形成独立的 `LayoutBase` 类。这样所有的容器都可以采用 `Group+LayoutBase` 的组合的方式，来设置任意的布局。