iOS UIView绘制（三）从Layout到Display

发表于 2019-08-27 | 分类于 技术 | Valine： 0

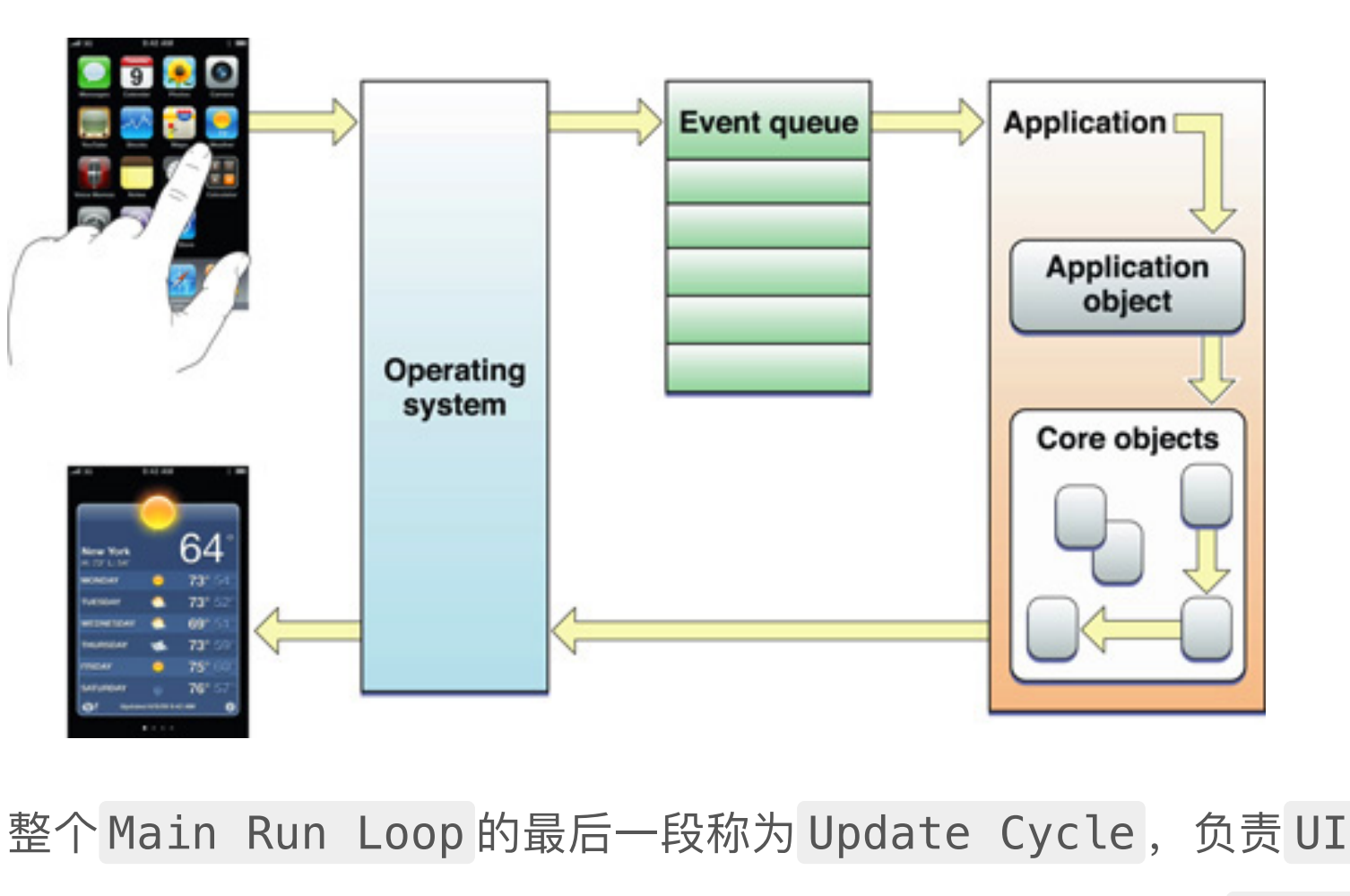
上文主要围绕`UIView`的初始化，探讨了`UIView`从创建到构建`View Hierarchy`的过程。视图初始化之后，下一步就是确定布局，即位置和大小，布局和绘制的相关环节非常相似，就一并介绍了。本文从相关方法入手，介绍一个`UIView`在布局和绘制时经历了哪些过程。

从Main Run Loop说起

每当谈到循环，我都会想起大一时用命令行窗口实现的贪吃蛇。游戏本身就是一个死循环，蛇的每一步都要进行很多判断：是否吃到食物、是否撞墙等等，这一系列事件的执行称为一次循环，循环退出的条件就是蛇死掉了：

```
while (snake.alive) {
  snake.walk();
  if (snake.eatFood()) {
    snake.length ++;
  }
  if (snake.hitWall()) {
    snake.alive = false;
  }
}
```

对于iOS应用来说，`Main Run Loop`就是这个死循环，每次循环都要处理用户的输入交互，并触发响应。用户的每一个交互，都被看作一个事件加入了事件队列，`Application`对象不断地从队列中获取事件，并分发给合适的目标进行处理。这个过程可以用下图表示：



整个`Main Run Loop`的最后一段称为`Update Cycle`，负责`UIView`的布局和绘制。当所有事件都被处理之后，就开始更新布局或者重绘。如果我们改变了`UIView`的属性，系统会帮我们做一个“需要重新布局”或者“需要重绘”的标记，在下一次循环到`Update Cycle`时进行相应的操作。每次循环非常快，所以用户察觉不到交互和视图更新的间隙，iOS应用一般能够达到60fps，即循环一次只需要1/60秒。如果在视图布局或重绘时，我们编写了一些耗时的代码，循环的时间可能会加长，最终导致卡顿。

Auto Layout中视图的布局和绘制分为三步：第一步是更新约束，第二步是布局，第三步是绘制，每一个Update Cycle中都遵循这个顺序，并且每个步骤内部的流程和彼此非常相似，下面通过介绍各方法的调用来了解视图经历的过程。

Constraints

从Android开发转行的我当初非常不适应，原先布局的思考模式几乎都是`ConstraintLayout`或者`RelativeLayout`，所有控件的位置和大小都依赖于其他控件，即使是`LinearLayout`，也可以按照顺序依次排列，另外对`match_parent`和`wrap_content`尤其怀念。现在iOS中创建View大多使用了`initWithFrame`，需要直接指定绝对位置和大小，幸而约束救我一命。

updateConstraints()

这个方法用于计算并更新View的所有约束，每当约束有更新时，系统就会调用这个方法，但是**永远不要手动或主动地调用这个方法**，系统提供了设置需要更新约束的标记。另外要注意的是，很多静态约束是不会改变的，例如某个View占满屏幕等等，这些约束应该在Interface Builder中创建，或者在View初始化时添加，亦或者在`ViewDidLoad()`中创建。继承的`updateConstraints()`方法中不应该频繁地更新这些约束，而是应该根据以上约束的变化来进行相关操作，相当于是一个监听约束变化的回调。

setNeedsUpdateConstraints()

上面说到不应该主动调用更新约束的方法，正确的做法是通过`setNeedsUpdateConstraints()`方法标记某个View为需要更新约束，下一次`Update Cycle`中系统会遍历所有View，其中设置了标记的就调用`updateConstraints()`，在这之前可以通过`needsUpdateConstraints()`来获取是否需要更新约束。

updateConstraintsIfNeeded()

该方法首先检查View是否需要更新约束，如果需要，则马上调用`updateConstraints()`，而不会等到`Update Cycle`。

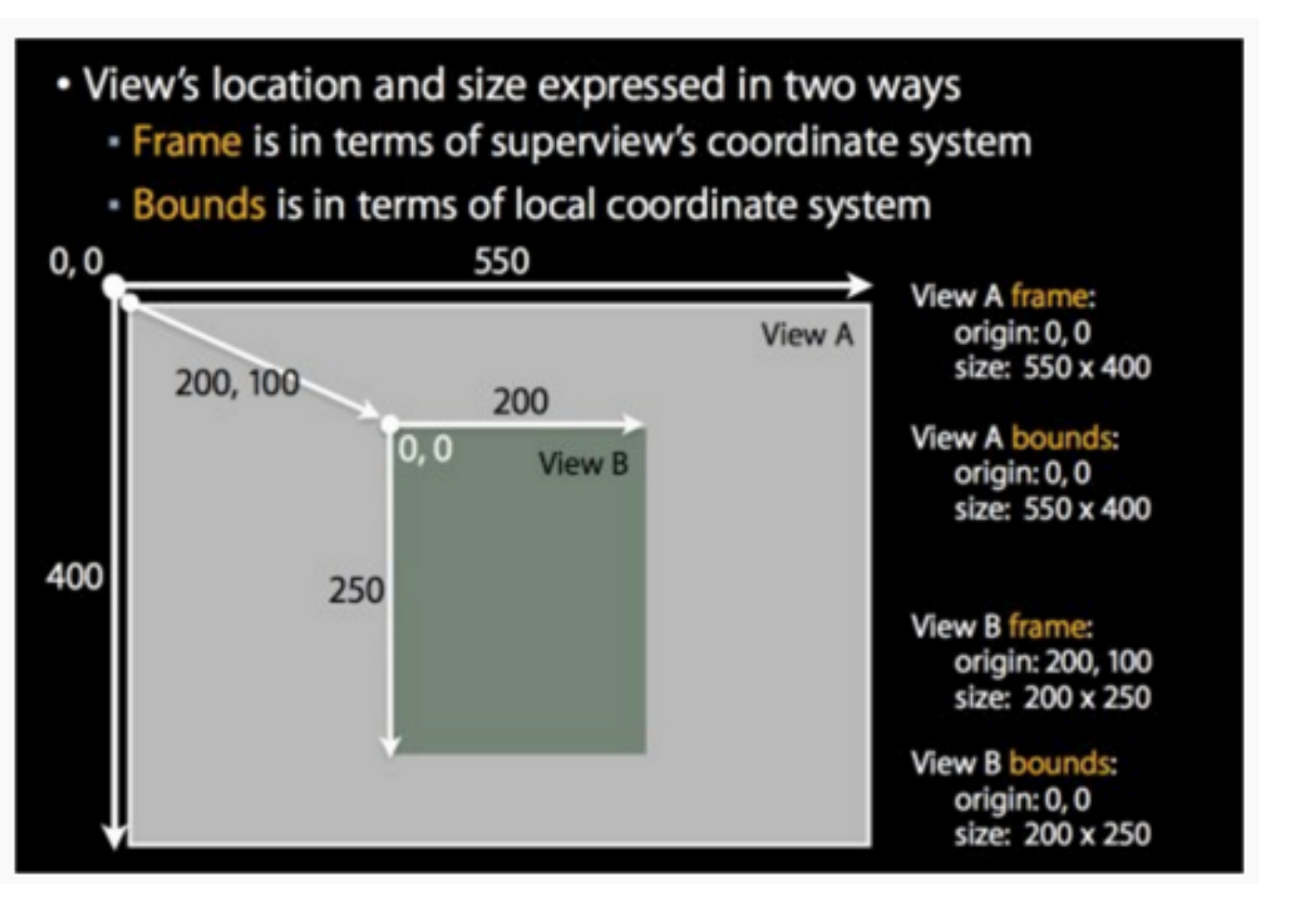
invalidateIntrinsicContentSize()

有些View会有一个`intrinsicContentSize`属性，这个属性称为固有大小，它制定了View的自然大小。一般来说，这个大小会根据控件的约束计算出来，不过当约束有冲突、或者没有相关约束的时候，View就会根据这个属性来确定自己的大小。而`invalidateIntrinsicContentSize()`方法给View设置了一个标记，表明View需要更新这个属性，在下一步Layout过程中需要重新计算。

站在View的角度考虑，仅仅知道约束是不能确定布局的，例如B紧贴A的右边，宽高都是A的一半，这时确定B布局的另一个条件就是要知道A的位置和大小，所以计算完这些约束，下一步就是根据约束确定具体的布局。

Layout

Layout，布局表示视图的位置和大小，`UIView`通过`frame`或者`bounds`属性来描述这些信息，其中`frame`表示以父视图为参考系时的位置和大小，`bounds`表示以自己为参考系时的位置和大小，具体可以参考下图：



layoutSubviews()

这个方法负责重新计算自己和所有子View的大小，以及重新摆放自己和所有子View的位置。这是一个递归执行的方法，它会调用所有子View的`layoutSubviews()`方法，所以比较耗时。系统会在需要的时候自动调用这个方法，我们可以继承这个方法，在方法内部进行特殊操作以达到想要的布局。

`layoutSubviews()`完成之后，view所在的ViewController会触发`viewDidLayoutSubviews()`，所以重新布局完成之后的操作应该在这个方法内执行。

setNeedsLayout()

上面说到如果需要重新加载视图，不应该直接调用`layoutSubviews()`，而是应该通过`setNeedsLayout()`告诉系统这个View的布局需要重新设置。设置需要重新布局的标记之后，系统会在下一次`Update Cycle`时调用`layoutSubviews()`，这是触发`layoutSubviews()`最节省资源的做法。并且设置标记的过程非常快，即`setNeedsLayout()`会马上返回，所以用户并不会有卡顿的感觉。

layoutIfNeeded()

如果通过`setNeedsLayout()`或者系统本身标记了需要重新布局，此时调用`layoutIfNeeded()`，则View的`layoutSubviews()`会马上触发。但如果视图不需要重新布局，调用此方法就没有任何效果。如果对视图进行了标记，并在同一次Run Loop循环中调用了两次`layoutIfNeeded()`，且这两次之间视图并没有更新，则第二次调用并不会触发`layoutSubviews()`。

和只调用`setNeedsLayout()`相比，这种方式的好处就是布局会在`layoutIfNeeded()`返回之前就重新加载完成，所以如果在下一次Update Cycle之前就需要新的布局结果的话，这种方式就非常适用。例如，通过动画更新约束时，在block调用前调用一次`layoutIfNeeded()`，以确保动画开始之前所有视图处于理想状态，在block中设置新的约束之后，再次调用`layoutIfNeeded()`通过动画更新到新的状态。

上面说到，View重新布局的标记不一定是我们手动设置的，某些事件触发系统为视图设置标记：

- 改变View大小，只改变位置不会触发
- 添加子View
- 滑动`UIScrollView`
- 旋转屏幕，会触发当前ViewController根View的重新布局
- 更新约束

Display

视图的显示包括颜色、文字、图片以及`Core Graphics`绘制等内容，触发机制和Layout非常相似，具体的实现方法名也比较类似。

draw(_ rect: CGRect)

该方法负责视图的绘制，我们也不应该直接调用`draw`方法。但是注意有一点和`layoutSubviews`不同：`draw`方法不是递归调用的，即并不会触发子视图的`draw`方法。

setNeedsDisplay()

这个方法类似于`setNeedsLayout()`，会给有内容更新的视图设置一个标记，方法立即返回，在下一个`Update Cycle`中，系统会遍历所有有标记的视图，并调用它们的`draw`方法。如果只需要重绘一部分视图，可以调用`setNeedsDisplay(_ rect: CGRect)`方法指定需要重绘的范围。

大多数情况下，改变UI控件的相关属性时会自动添加上需要更新的标志，例如设置宽高等等，这时不调用`setNeedsDisplay`方法也能更新视图。但是有些属性是没有和UI控件绑定的，而这些属性更新时又需要进行视图更新，解决方案是在属性的`didSet`中调用`setNeedsDisplay`方法。

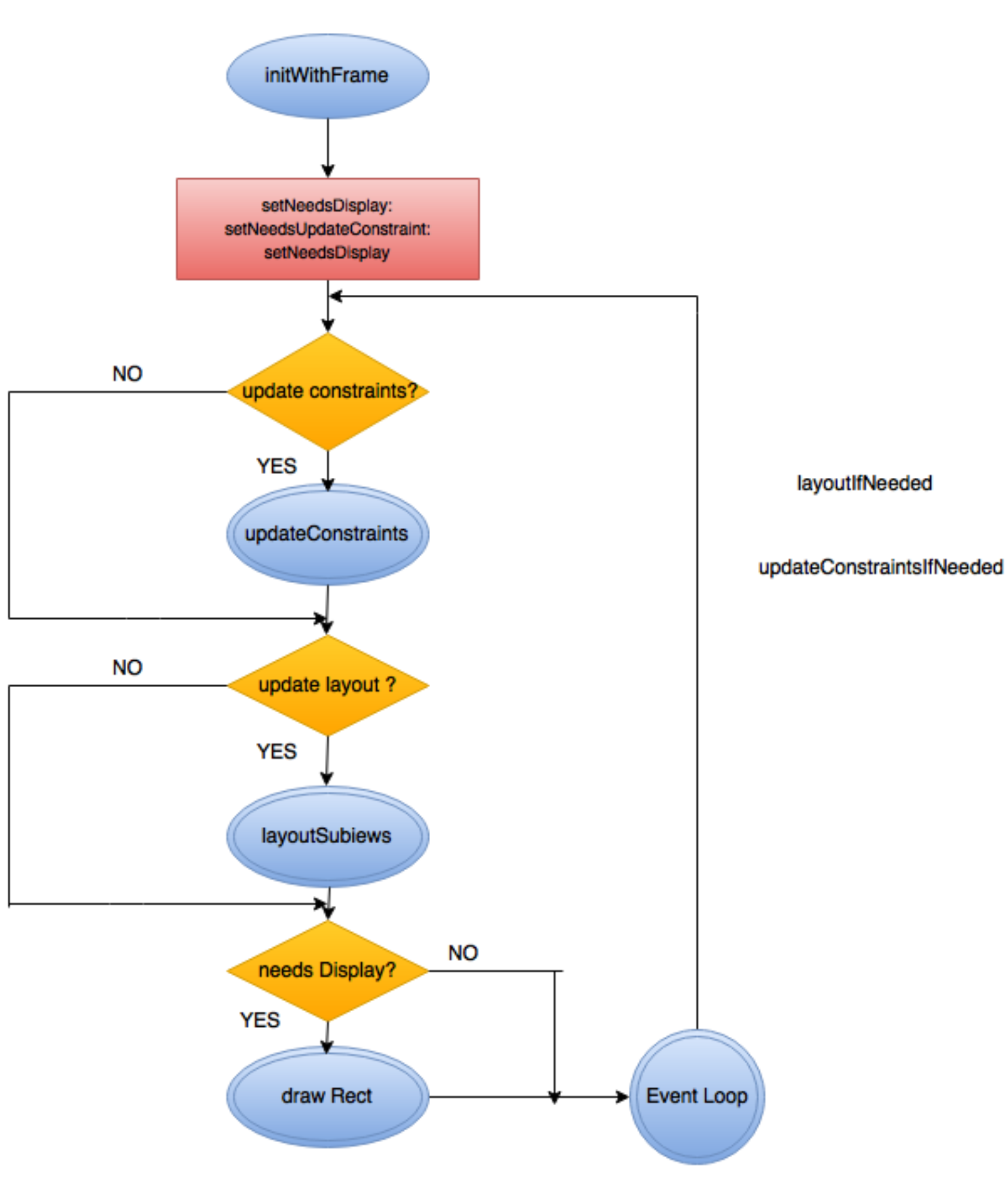
Display和Layout还有一点不同是，`layoutIfNeeded`可以让视图在需要重新布局时马上进行布局，不需要等到下一次`Update Cycle`，但是重绘不能提前，只能在`Update Cycle`中进行。

总结

Constraints、Layout、Display都遵循相似的设计模式，例如它们的更新方式和标记方式，以及跟Run Loop的关系。

行为	Constraints	Layout	Display
实现更新	`updateConstraints()`	`layoutSubviews()`	`draw(_ rect: CGRect)`
标记需要更新	`setNeedsUpdateConstraints()`	`setNeedsLayout()`	`setNeedsDisplay()`
按需更新	`updateConstraintsIfNeeded()`	`layoutIfNeeded()`	无，只能被动更新

对应的，Event Loop和Update Cycle的交互也可以用流程图也解释：



> 参考：
>
> Demystifying iOS Layout

技术　# iOS　# View

‹ iOS UIView绘制（二）View Hierarchy　　抖音实习周记（六）THE END ›

昵称　邮箱

欢迎批评指正

提交

来发评论吧~

↑ 87%