# **NGUI优化—基于委托的锚点更新机制**

作者：[冷梦无邪](http://gad.qq.com/user/index?id=183642)

链接：<http://gad.qq.com/article/detail/46851>

## **NGUI原有的更新方式**

NGUI自带的Anchor更新对应的方式包括：

public enum AnchorUpdate

｛

OnEnable,

OnUpdate,

OnStart,

｝

为了节约性能，一般希望使用OnEnable进行锚点的更新，再看UIRect.cs中OnEnable的实现：

protected virtual void OnEnable ()

｛

#if UNITY\_EDITOR

mEnabled = true;

#endif

mUpdateFrame = -1;

if (updateAnchors == AnchorUpdate.OnEnable)

｛

mAnchorsCached = false;

mUpdateAnchors = true;

｝

if (mStarted) OnInit();

mUpdateFrame = -1;

｝

可以看到仅仅是将mUpdateAnchors标记为true，而真正的锚点更新是发生在Update()中的：

public void Update ()

｛

if (!mAnchorsCached) ResetAnchors();

int frame = Time.frameCount;

#if UNITY\_EDITOR

if (mUpdateFrame != frame || !Application.isPlaying)

#else

if (mUpdateFrame != frame)

#endif

｛

#if UNITY\_EDITOR

if (updateAnchors == AnchorUpdate.OnUpdate || mUpdateAnchors || !Application.isPlaying)

#else

if (updateAnchors == AnchorUpdate.OnUpdate || mUpdateAnchors)

#endif

UpdateAnchorsInternal(frame);

// Continue with the update

OnUpdate();

｝

｝

问题：

实际的锚点更新是在Update中，而Unity中Update的更新顺序是不可预知的。

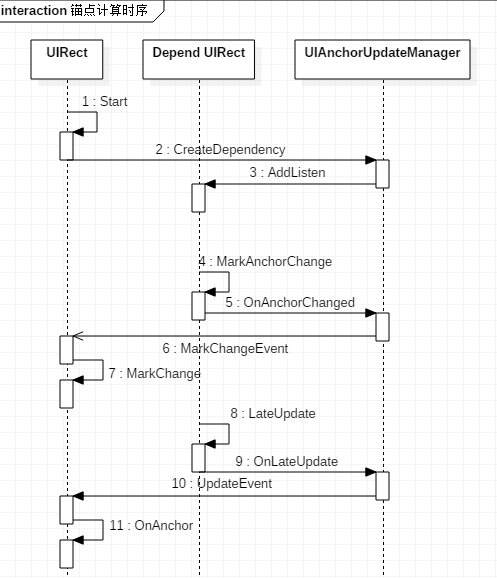
也就是对于更新类型为OnEnable的两个UIRect A和B，若A的锚点依赖与B，但是A的Update更新顺序优先于B，就会出现A先更新，然后B才更新，使得A的锚点计算是错误，为此，NGUI中只能使用的办法是：把A的锚点更新方式改为OnUpdate；

再回头看上面的代码，当updateAnchors == AnchorUpdate.OnUpdate时，UpdateAnchorsInternal(frame)函数会在每帧都被执行（通知所依赖的UIRect去更新刷新锚点，当依赖关系很复杂庞大时，这也将是一棵巨大的调用树），而大部分情况下，我们都希望UI界面只在刚出来那几帧得到正确的锚点即可，后面大部分时间锚点都不会发生变化，因此使用AnchorUpdate.OnUpdate就会带来额外的开销，当UI界面变得复杂的时候，这部分的额外消耗就变得不可忽略。

## **基于委托的更新机制**

分析了NGUI的更新机制，可以知道其实不得已使用OnUpdate更新方式的主要原因是因为Update的更新顺序不可控，也就是说，没办法确保所依赖的对象在自己之前先更新。

为了解决这个问题，想到的一个的方法就是通过委托建立起一棵锚点更新的依赖树，也就是当叶子节点发生变化时，通知父节点去更新，这样就确保了更新顺序。



直接去掉了原有的更新方式，使用监听依赖节点变化的方式来实现更新，虽然实在LateUpdate中检测更新，但是在子节点无变化的情况下基本没有消耗；

会触发MarkAnchorChange事件的地方包括控件size的变换，Transform的变化等；

此外需要注意的是table和Grid这两个容器类，因为它们的子节点是动态增加的，并且自身的锚点信息会受容器中节点的变化影响，因此需要在它们的AddChild函数中去创建依赖关系，并且每个节点都需要在创建依赖时去查找自身所处的容器对象，添加监听。

这个方法已在项目中使用，目前还没有发现有什么大的问题，并且性能也得到了提升，因此发帖谈论下。