1. 了解UnityUI分析器。
2. Unity Profiler：用来分析游戏运行时的时间消耗，可以通过开启/禁用UI来观察性能的影响，找到最消耗性能的部分。分析ui性能，追踪Canvas.BuildBatch和Canvas.SendWillRenderCanvas,为了更容易看到ui性能差异，通常CPU分析器禁用除了scripts/rendering以外的其他跟踪类别。
3. UnityFrameDebugger:用来查看UI元素drawcalls情况，Canvas的renderMode影响DrawMesh出现的位置
4. ScreenSpace-Overlay，出现在Canvas.RenderOverlays 组
5. ScreenSpace-Camera，出现在Camera.Render 组的子组Render.TransparentGeometry
6. WorldSpace，出现在可见camera的Camera.render组的子组Render.TransparentGeometry下
7. 影响UGUI性能的地方：
8. CPU:主要负责帧率
9. Drawcalls:UI渲染的时候，CPU开始计算逐层读取纹理的顶点数据，当读到下一个纹理时，如果是同一个图集，那么他就会add到序列中，进行批处理，如果不是同一张就会处理当前批，重新准备定点数据，增加drawcalls。

解决方法：a.处理好child的order，调整好子对象的层级，相同图集的对象之间尽量不要穿插其他图集的元素，减少批处理被打断的问题。（此处，text label容易被忽视）

1. Rebuild，Canvas的重建过程发生在CanvasUpdateRegistry.class，当Canvas调用WillRenderCanvases时会唤醒PerformUpdate方法。WillRenderCanvases每帧调用一次，完成一下操作：
2. 如果Layout标记dirty，则通过ICanvasElement.Rebuild重新构造layout
3. 任何注册的剪辑组件如（mask）重新执行CLippingRegistry.cull
4. 所有的脏Graphic（发生变化）组件也会重新构建

如上，应该尽量减少Rebuild的发生，解决方法如下：

1. 对于厚重的canvas最好拆分成两份，同时发生变化的元素放到同一个canvas上，例如一个计时器及相对应的进度条，尽量将动态元素与静态元素分割到不同canvas，这样可以减轻rebuild的压力，但是不能减少drawcalls所以，需要完美平衡两者的关系
2. 排除不可见UI的影响，UI打开关闭时会触发Rebuild因此，对于禁用UI，使用CanvasRender组件的canvasRenderer.cull = true，后者利用CanvasGroup.cs的alpha属性这样只是不绘制，他们还是长治驻留内存，canvas不会丢掉vbo数据，重新打开时不会重新收集顶点数据。然而这种做法不会禁用monobehavior的方法调用，尝试在禁用ui的根目录上做一个回掉管理器，这个管理器在ui显示或者隐藏时收到通知，确保生命周期的开始与停止
3. 优化raycast，关闭不需要进行事件输入的对象的raycasttarget
4. 组件优化，scrollview滚动视图，使用对象池，知生成可见对象。Button注意label的影响，可能造成drawcalls增加
5. 关于布局：简单的布局尽量使用recttransform的anchors

2.GPU:主要负责像素相关

（1）避免过度绘制（overdraw）减少gpu填充率过高的问题。由于大量重叠的ui会造成很严重的overdraw，可以采用以下方法减少影响：

a.禁用不可见ui，使用上面提到的Canvasrender组件禁用，配合回调管理器。

b.将静态的装饰类的元素做到背景图上，Gpu在渲染的时候，会逐层采样，例如在一个包括背景按钮文字的ui上，gpu必须采样背景纹理，然后是按钮纹理，然后是文字纹理，如果ui的装饰增加，那么所采样的次数增加，增大gpu压力，所以，可以选择将不变的装饰和背景做成同一张纹理

c.UI预知制作过程中尽量扁平化。

（2）适时禁用3d摄像机的渲染，对于canvas的rendermode设置为screenspace-overlay，当出现全屏UI，看不到3d场景中的东西，可以禁用3d场景的渲染camera，或者如果可以看到少部分3d场景，可以巧妙利用render texture，对3d场景进行缓存，以减轻GPU压力