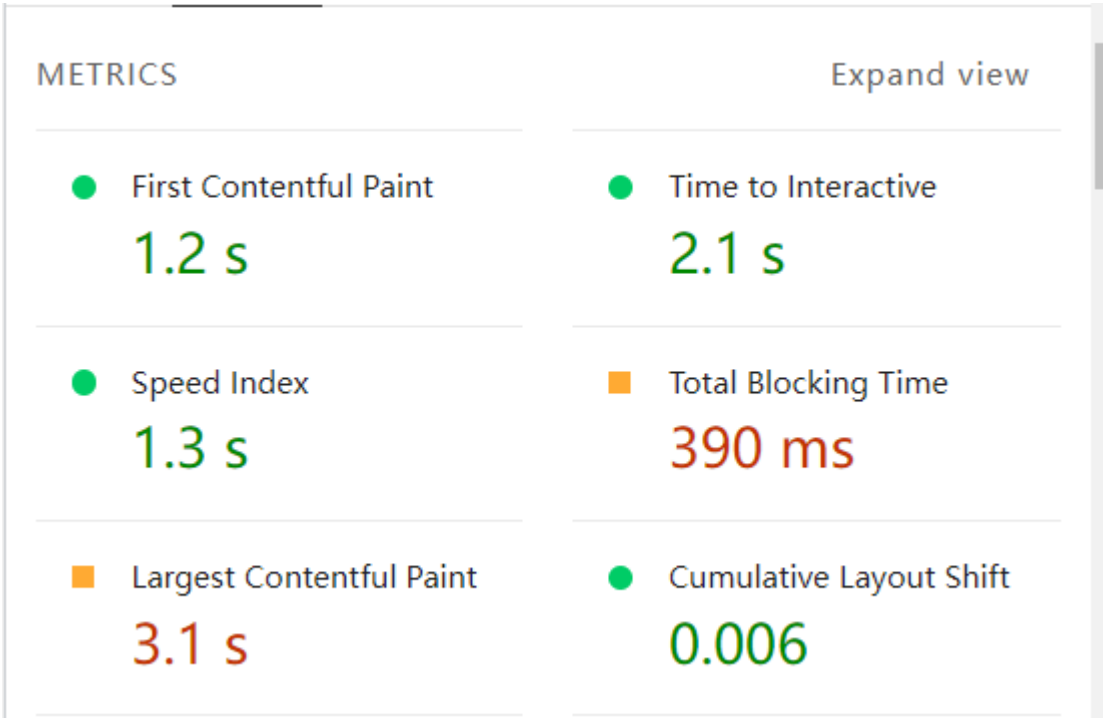
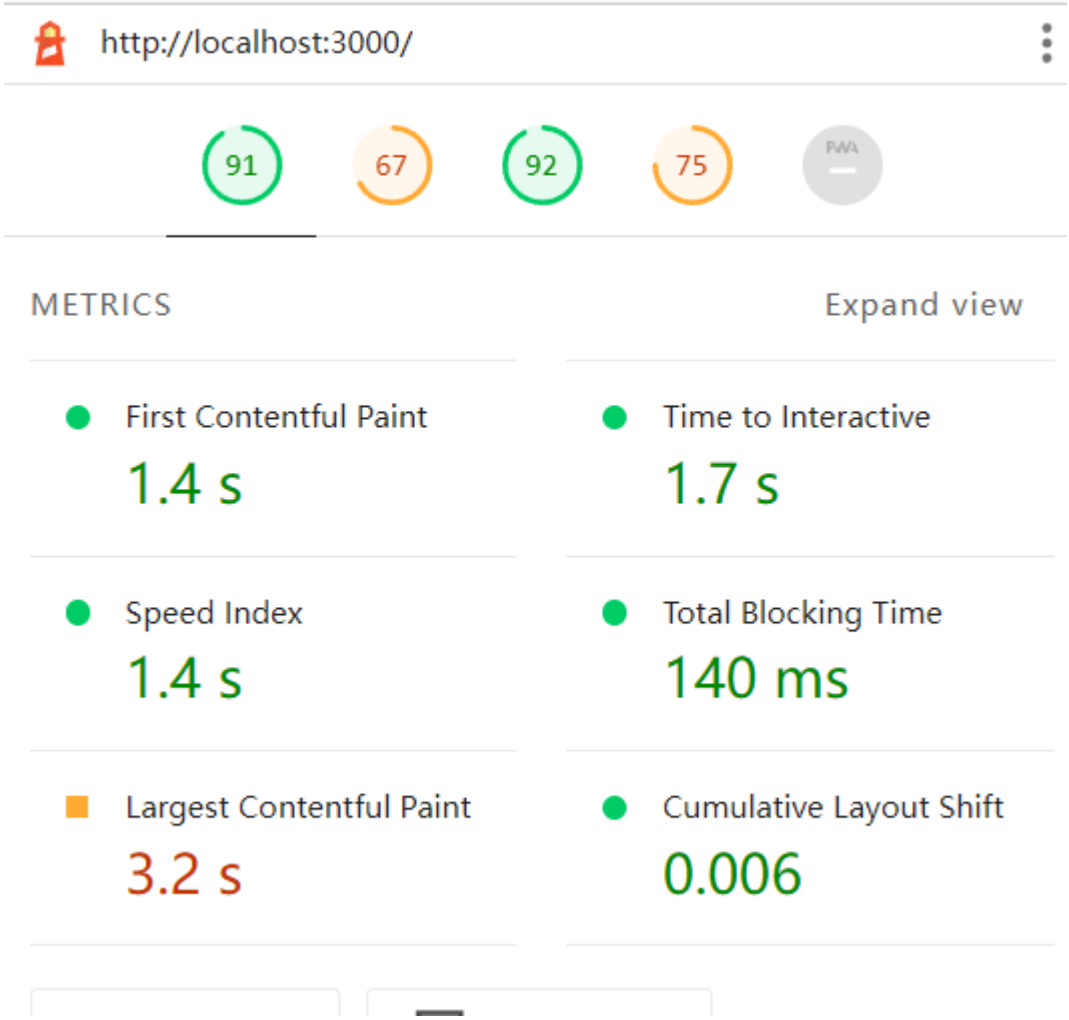
验证分数 谷歌浏览器 Lighthouse mobile 分数:
不用 vite 框架直接加载静态网页:



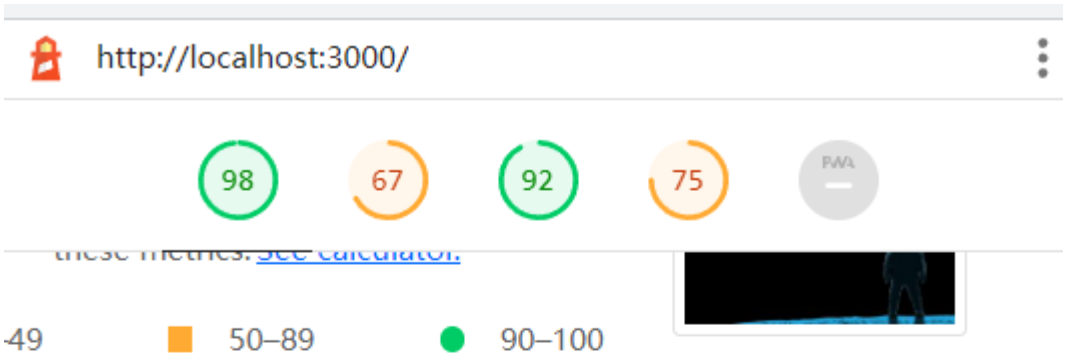
Vite 框架初始状态:

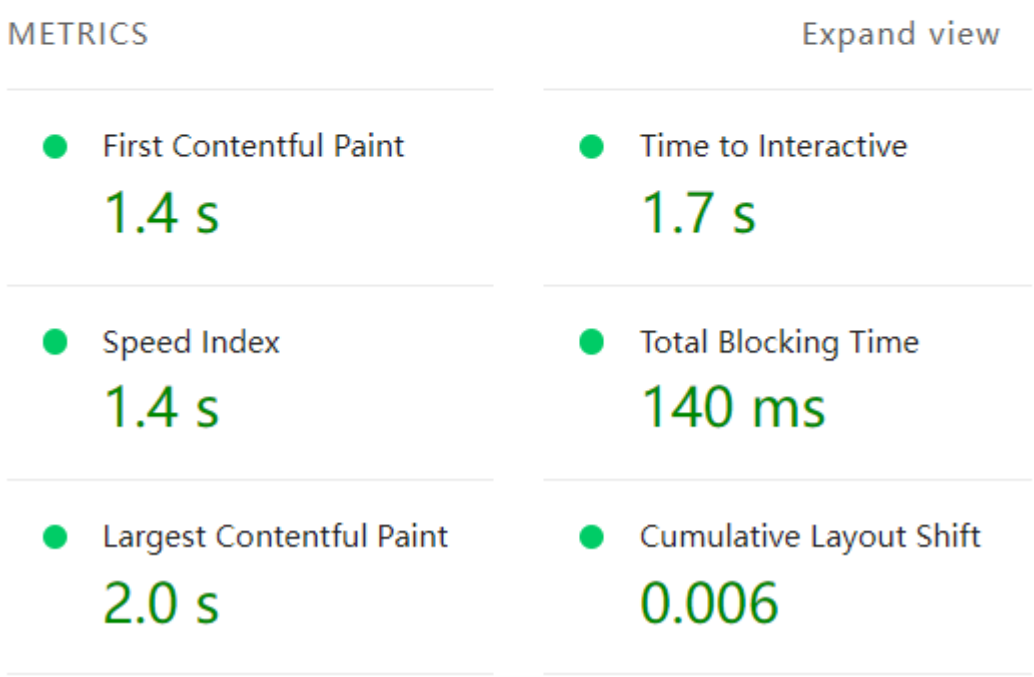


部分指标略低于静态网页，猜测是由于内容较少，架构的冗余加载（封装层次更高）带来的负担大于 vite 框架的优势。

使用 tinypng 压缩 png 图片：

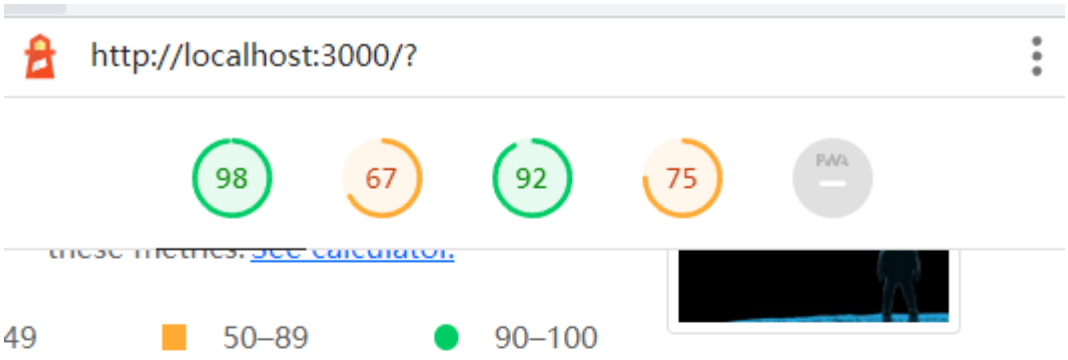
将背景图丢入 tinypng 后

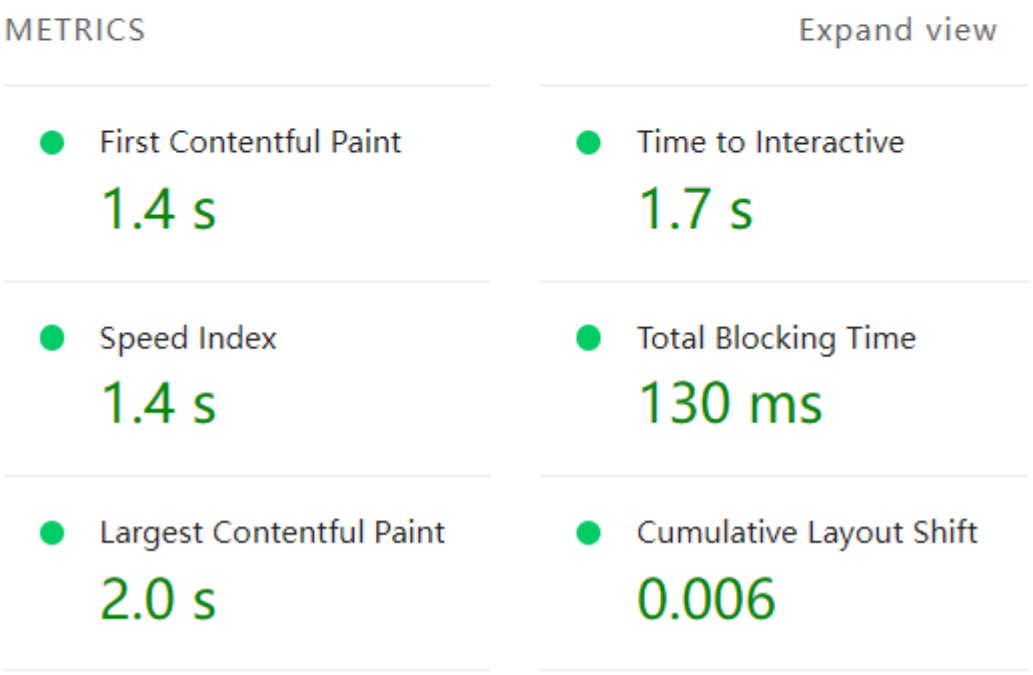




（网页总共就一张背景图片）

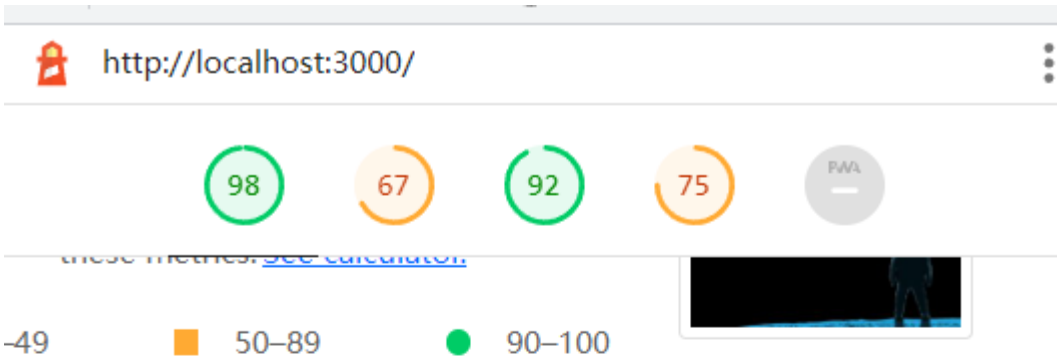
将 tinypng 转换后的图片转为 webp 格式：

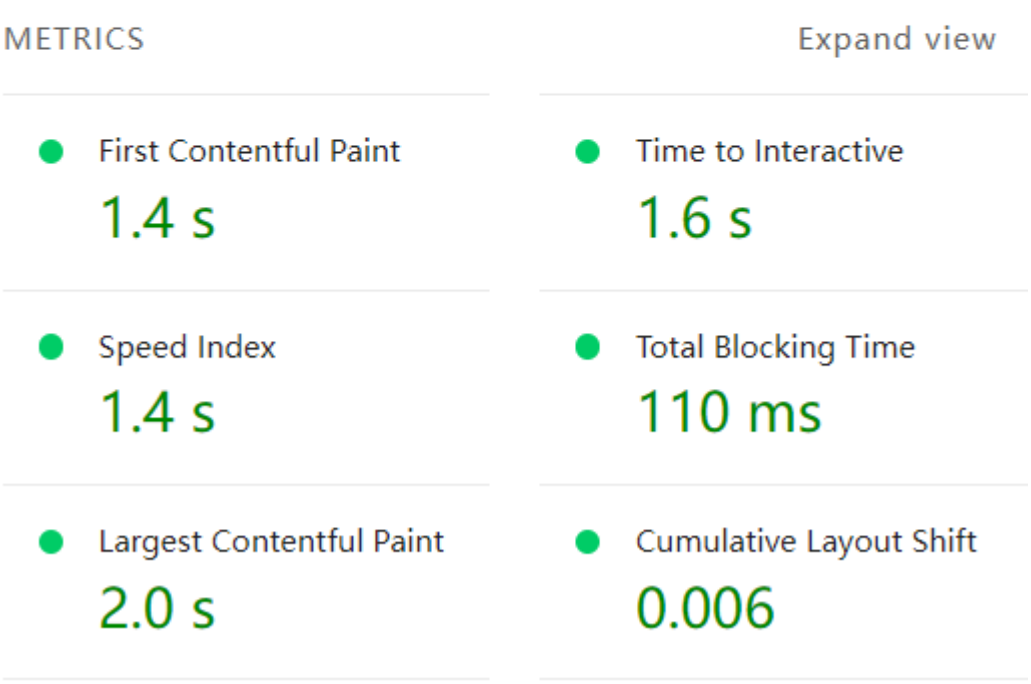




可见和 tinypng 的性能差不多，应该是经过转换后的 png 图片大小和 webp 大小差不多，心性能主要取决于图片大小而不是 webp 格式。(而且好像同一个代码，每次跑都稍微有些差别。。所以参考价值也不太大)
以下在 webp 格式下继续优化:

使用 img 的 loading lazy:





TBT 少了 20ms，有点用

使用 transform 代替 position:

```css
.img_rot{
    animation: rotating 20s linear infinite;
}

@keyframes rotating{
    0% {
        transform: rotate(0);
    }

    50%{
        transform: rotate(180deg);
    }

    100%{
        transform: rotate(360deg);
    }
}
```

由于之前没有使用旋转图标，性能得分已没有参考价值，没有跑。

使用防抖函数:

为了便于 debug，将 button 设置为 type="button"防止页面重绘

```html
<button id="btn1" type="button"></button>
<script>
    var obtn=document.getElementById('btn1');
    function debounce(fn,delay){
        var timer=null;//利用闭包存入内存
        return function f(){
            clearTimeout(timer);
            timer=setTimeout((function () {
                fn();
            }),delay)
        }
    }
    obtn.onclick=debounce(function () {
        console.log('你提交了一次!');
    },600);
</script>
```

可见，类似于 python 的装饰器，闭包把要执行的函数套一层 settimeout 然后返回给 onclick 函数，这样每次在设置的 600ms 中点击都会出发 onclick 的函数，而每次点击都会把通过闭包在内存中的唯一变量 timer cleartimeout 并重新设置时间为 600ms，即重新计时，从而减少服务器端的请求数。

总结:

实现了一些基本的性能优化方式，而本静态页面图片和内容有点少，预计当大型网页的时候

这些优化手段的效果应该更明显。