前沿优化解决方案

**痛点：对于移动端来说，几kb的图片都稍微有点大**

SVG移动端图标

传统png：色彩比较好

**IconFont**

我们可以用字体的方式来代替图片。

IconFont

一般是包在css里面，然后使用<i>元素 调用不同的class来使用图标。最终引用的是字体文件。

可以通过font-size来调整大小

浏览器会使用unicode来对字体文件进行寻找。

优点：  
多个图标可以合成一个字体，减少获取时的请求和体积

矢量图，可伸缩

直接通过CSS修改样式大小，像对待字体一样。

缺点：

没办法直接修改一个字体的颜色

SVG

保持了图片能力，支持多色彩。

是个独立的矢量图形，按需使用。

它是XML的语法，方便SEO。

Flexbox 优化布局

Flexbox可以进行性能提升

在父元素设置display:flex

Flex-flow: row wrap按行显示，然后换行

如果使用float实现grid布局，10万个色块大概2800ms。

使用flex实现同样的布局，10万个色块大约2400。

优点：

更高性能的实现方案，

容器有能力决定子元素的大小，顺序，对齐，间隔等

双向布局。（横向纵向）

**优化资源加载顺序**

Priority

浏览器默认安排资源加载优先级。

默认html最高优先级

使用preload和prefetch来调整优先级

用<link>标签，在rel属性里面设置preload，href配合as。

Preload:提前加载较晚出现，但对当前页面非常重要的资源，比如字体。

Preload字体的时候必须设置跨域”crossorigin”属性，一般设置为匿名”anonymous”

Prefetch:提前加载其他页面的东西，但是本页面不需要。一般是在本页面所有资源加载完之后，再预加载这个资源。浏览器会把预加载的页面放到缓存里。

**预渲染**

比如react-snap

React.hydate.

大型的单页应用的性能瓶颈：JS下载+解析+执行

SSR主要问题：牺牲TTFB（至第一字节时间）来补救首屏渲染，实现复杂。

相当于把前端任务转给后端。

预渲染打包时提前渲染页面，没有服务端参与。

内联样式避免页面闪动。

**Windowing**

只显示必要的行，过去的行消失掉。

**React-window库**

加载大列表，大表单的每一行严重影响性能。

Lazy Loading仍然会让DOM变得更大

Windowing只渲染可见的行，渲染和滚动的性能都会提升。

**骨架组件**

React-placeholder

主要用处是占位，减少布局抖动。