前端开发核心知识进阶：50 讲从夯实基础到突破瓶颈
来自 Lucas ... · 盐选专栏

查看详情 >

性能优化问题，老司机如何解决（上）

一直以来，性能优化是前端的重要课题，不仅实实在在影响产品性能，在面试环节也会被反复提及。无论应聘者是初入前端的新手，还是工作经验丰富的老司机，面试官都能在性能方面找到合适的切入点，对候选人进行考察。作为程序员，应该如何在平时学习、工作中积累性能优化方面的经验，保障产品顺畅体验？作为面试者，如何在面试流程中出色地回答性能相关问题？

前端性能是一个太过宽泛的话题，脱离场景和需求谈性能往往毫无意义。我相信很少有面试官会直接把：「前端如何优化性能？」——这样一个空架子问题抛出的。也不会有技术经理直接丢给你「把产品性能提升一些」这样的项目。毕竟这样的问题过大，根本让人无处下手。我们还需要针对具体场景和瓶颈来分析。

但是，如果真的有面试官这么问了呢？

如果是我，我也许会这样回答：

前端性能涉及方方面面，优化角度切入点都有所不同。我认为，主要可以分为：页面工程优化和代码细节优化两大方向。

页面工程优化从页面请求开始，涉及网络协议、资源配置、浏览器性能、缓存等；**代码细节优化**上相对零散，比如 JavaScript 对 DOM 操作，宿主环境的单线程相关内容等。

也正如上所答，本节课程也会基于以下两个大方向的相关知识进行梳理：

页面工程优化

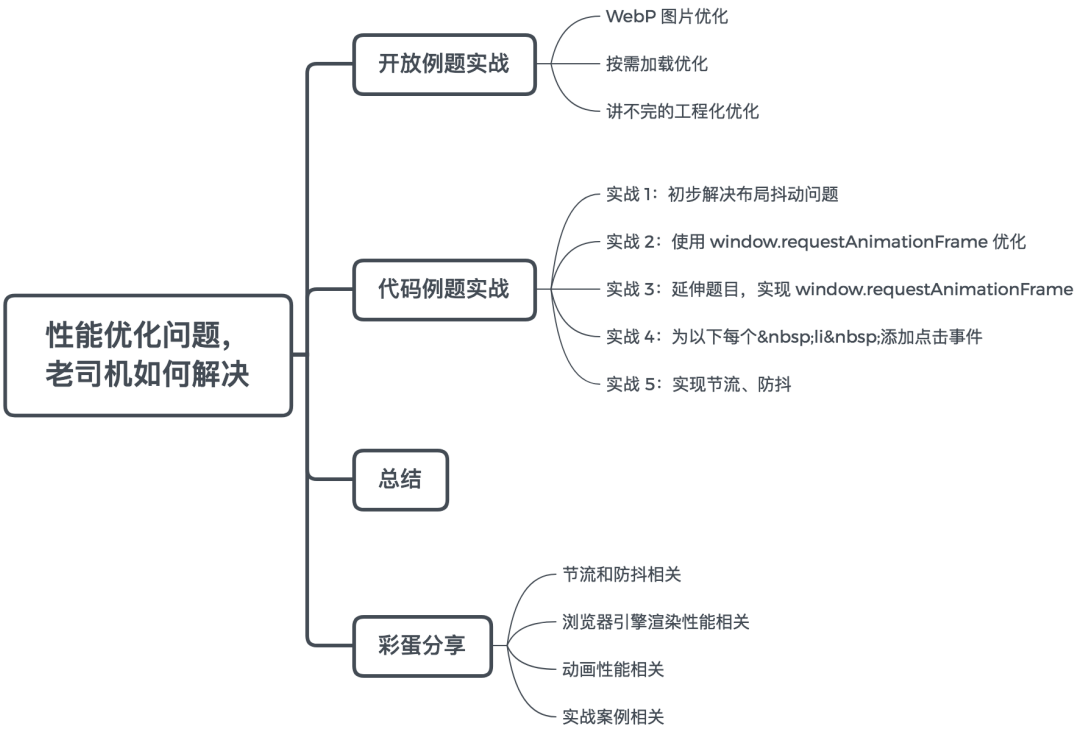
代码细节优化

为了更好地还原真实场景，这两方面我都将配合两类面试题目来解析：

开放例题实战

代码例题实战

这个主题的知识点如下：



接下来，我们通过 2 节内容来学习这个主题。

开放例题实战

如上分析，面试官往往会根据面试者的实际经验或者性能的某一细分方向，进行深度提问，以了解候选人的知识储备以及在以往项目中的表现。

作为一个面试者，包括在蚂蚁金服、阿里淘宝某团队、头条、美团以及其他公司的多次面试流程中，我曾经被问到过：**「在平时工作中做过哪些性能优化方面的项目？」**

我是这样回答的：

因为我服务的是有亿级流量的 To C 型产品，因此平时工作中，在性能优化方面一直持续进行探索和迭代。除了代码细节方面外，较大型工程优化主要有 WebP 图片格式替换、资源打包和逆向代码拆分（按需加载）等。

WebP 图片优化

因为并不知道面试官需要考察的程度，以上回答可以避免自己「侃侃而谈」浪费时间的尴尬。在这样的面试场景中，我往往会把主动权交给面试官。大部分面试官会继续追问，比如他对 WebP 图片格式优化项目感兴趣，**那我会从项目的立项、实施、收益的角度进行解答，表现作为一个项目负责人对优化项目的理解**：

我们的产品页面中，往往存在大量的图片内容，因此图片的性能优化是瓶颈和重点。除了传统的图片懒加载手段以外，我调研并实施了 WebP 图片格式的替换。由于可能会有潜在兼容性的问题，因而具体做法是先进行兼容性嗅探。这一手段借鉴了社区一贯做法，利用 `img` 标签加载一张 base64 的 WebP 图片，并将结果存入 `localStorage` 中防止重复判断。如果该终端支持，则再对图片格式进行替换。这个兼容性嗅探过程，也封装成 promise 化的通用接口。

相关代码片段如下：

```
const supportWebp = () => new Promise(resolve => {
   const image = new Image()
   image.onerror = () => resolve(false)
   image.onload = () => resolve(image.width === 1)
   image.src =
'
```

```
oBAAEAAwA0JaQAA3AA/vuUAAA='
}).catch(() => false)
```

这时候，面试官往往会进一步关心项目收益情况。这就需要面试者根据实情作答，仍然以这个项目为例：

在具体上线时，我对 10% 的流量进行了分组切分。5% 为对照组，仍然采用传统格式；另外 5% 为实验组，进行 WebP 格式试验。 **最终结果显示收益非常有限**。 为此我进行分析：认为出现近似零收益的原因是图片服务的缓存问题。新转换的一批 WebP 格式图片由于没有缓存，因而在性能上打了折扣。为了验证猜想，我决定继续进行扩量试验并观察结果。果然，后续排除缓存问题之后，收益提升 25%~30% 左右。

这里，我们就涉及了工程优化中的一个重要环节：网络请求和缓存。这些内容我们会在后续课程网络环节《第 8-1 课：缓存谁都懂，一问都哑口》中具体展开。

通过以上回答，我如实讲述了出现的非预期 case，并说明遇见问题时，如何进行分析进而解决问题的一系列过程。这样的回答能明确表现出我确实做过该项目并进行了思考分析，最终落地。这类思路也更容易被面试官所接受。

更多关于 WebP 的好文章：

> WebP Support - It's More Than You Think
>
> 从零开始带你认识最新的图片格式 WebP
>
> 把网站的图片升级到 WebP 格式吧

也由此看出，性能优化其实并不难做，重要的是解决问题的思路，以及解决过程中对于项目的把控和推荐。这些内容我们称之为「软素质」。

按需加载优化

如果面试官围绕着刚才列举的「资源打包和逆向拆分（按需加载）」方向提问，我仍然会采用同样的「思路」进行回答：

我接手项目之后，发现历史原有的资源打包配置并不合理，严重影响了性能表现。因此借助构建工具，对资源进行合并打包。但是，需要注意的是，我的策略并不是大刀阔斧地进行资源合并，因为这样会让 bundle.js 的 size 越来越大，所以也进行了逆向过程。

以实际页面为例：

如上，当点击左图播放按钮之后，页面出现视频列表浮层（右侧所示，仍为同一页面，类似单页应用）。视频列表浮层包含了滚动处理、视频播放等多项复杂逻辑，因此关于这个浮层的脚本我并没有进行打包合并，而是单独进行拆分。当用户点击浮层触发按钮后，再执行对这一部分脚本的请求。

工程化性能优化不仅需要做，还要用数据证明做法的合理性：

同时，我对用户点击触发按钮的概率进行统计，发现进入页面的用户只有 10% 左右会点击按钮，从而触发视频列表浮层。也就是说，大部分用户（90%）并不会看到这一浮层，延迟按需加载是有统计数据支持的。

通过这个案例，我们发现性能优化其实是一个开放式问题，非常依赖实践。读者可根据上面的例子，结合自己的项目进行回答。

虽然没有涉及代码实施，但是建立起项目意识和方向意识，这在工程化性能上非常难能可贵。上面提到的借助构建工具进行「按需加载」等内容，前面章节如《webpack 工程师 > 前端工程师》都会具体从代码角度给出示例。

当然上述举例的按需加载，我们并不是使用成熟的 webpack 工具链，而是采用公司内部封装的工程化工具。在几年前，这样的方案并不成熟，因此我写了一些按需加载的差距，配合自己的工程化工具使用。这一方面内容，很多面试官会很感兴趣，话题也可以延伸到 FIS 和 webpack 的比较，工程化工具的设计等话题。

讲不完的工程化优化

此外，工程优化方向还包括：

图片懒加载

雪碧图

合理设置缓存策略

使用 prefetch / preload 预加载等新特性

以 tree shaking 手段为主的代码瘦身

这里不再一一举例，欢迎订阅此课程的同学结合自己的经历在评论区留言讨论或者直接向我提问，我会尽量拿出时间跟大家一起进行项目分析、描述。

以上是**工程化**性能的实际题目，总结一下：工程师需要对日常项目进行深入总结，结合产品角度、研发角度描述。在面试前就需要做到「心中有数，胸有成

竹」。

另外，在具体的实现方向上，关于性能优化的切入点也有很多。比如：

动画性能方向

操作 DOM 方向

浏览器加载、渲染性能方向

性能测量、监控方向

这些方向并不是相互独立的，它们彼此依存，比如**动画性能方向与浏览器渲染性能息息相关**。这里我用一道经典的面试题来分析：**「如果发现页面动画效果卡顿，你会从哪些角度解决问题？」**

首先从动画实现入手：

一般 CSS3 动画会比基于 JavaScript 实现的动画效率要高，因此优先使用 CSS3 实现效果（这一点并不绝对）

在使用 CSS3 实现动画时，考虑开启 GPU 加速（这一点也并不总是正向效果）

优先使用消耗最低的 transform 和 opacity 两个属性

使用 will-change 属性

独立合成层，减少绘制区域

对于只能使用 JavaScript 实现动画效果的情况，考虑 requestAnimationFrame、requestIdleCallback API

批量进行样式变换，减少布局抖动

事实上，上面每一点的背后都包含着很多知识点，例如：

如何理解 requestAnimationFrame 和 60 fps

如何实现 requestAnimationFrame polyfill

哪些操作会触发浏览器 reflow（重排）或者 repaint（重绘）

对于给出的代码，如何进行优化

如何实现滚动时的节流、防抖函数

这些问题，我们会拿出其中几个在下一节分析。同时，订阅课程的朋友们，都可以在下节课末尾彩蛋区获得我在这些主题方向上收藏的文章。

总结

工程化优化是一个太大的话题了，本节课我们只是在「大面上」进行方向指引，以面试为角度抛砖引玉，更多具体代码细节实例，下节课我们会继续探讨。

另外，正因为这个话题的特殊性，一千个项目，就有一千个优化场景，也欢迎大家拿出来项目中的具体情况，跟我讨论。

点击查看下一节

性能优化问题，老司机如何解决（下）