link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=220 sentences=343, words=2405

1. 为什么要关注Web性能

- Web性能可以直接影响业务指标，例如转化率和用户满意度

2. Lighthouse

- lighthouse (https://www.npmjs.com/package/lighthouse)是一个开源的自动化工具，用于改进网络应用的质量
- Lighthouse报告分析了加载页面生命周期中的各种性能指标
- webpagetest (https://www.webpagetest.org/)是一个免费的网站性能测试工具

```
lighthouse https:
```

3. Lighthouse性能指标

3.1 FP和FCP

- First Paint(首次渲染)表示了浏览器从开始请求网站到屏幕渲染第一个像素点的时间
- First Contentful Paint(首次内容渲染) (https://web.dev/first-contentful-paint)表示浏览器渲染出第一个内容的时间，这个内容可以是文本、图片或SVG元素等，不包括iframe和白色背景的canvas元素

3.1.1 记录FP和FCP

- First_paint (https://developer.mozilla.org/zh-CN/docs/Glossary/First_paint)
- First_contentful_paint (https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint)
- document.readyState (https://developer.mozilla.org/zh-CN/docs/Web/API/Document/readyState)
- PerformanceObserver (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceObserver/PerformanceObserver)
- PerformanceObserver.observe (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceObserver/observe)
- PerformanceEntry.entryType (https://developer.mozilla.org/zh-CN/docs/Web/API/PerformanceEntry/entryType)
- PerformanceObserverEntryList (https://developer.mozilla.org/en-US/docs/Web/API/PerformanceObserverEntryList)

3.1.1.1 安装依赖

```
npm install express morgan compression --save
```

3.1.1.2 site\index.js

site\index.js

```
const express = require('express');
const logger = require('morgan');
const delayConfig = require('./delayConfig');
const app = express();
app.use(logger('dev'));
app.use((req, res, next) => {
    let url = req.url;
    let delay = delayConfig[url];
    if (delay) {
        setTimeout(next, delay);
    } else {
        next();
    }
});
app.use(express.static('public'));
app.listen(80, () => console.log(`server started at 80`));
```

3.1.1.3 delayConfig.js

site\delayConfig.js

```
const delayConfig = {
    "/index.html": 100
}
module.exports = delayConfig;
```

3.1.1.4 index.html

site\public\index.html

```

<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>lighthousetitle>
head>

<body>
    <canvas style="width:100%;height:500px;">canvas>
    <div>hellohellohellohellohellohellohellohellohellohellohellohellohellohellohellohellodiv>
    <script src="/perf.js">script>
body>

html>
```

3.1.1.5 perf.js

site\public\perf.js

```
(function (ready) {
    if (document.readyState === "complete" || document.readyState === "interactive") {
        ready();
    } else {
        document.addEventListener("readystatechange", function () {
            if (document.readyState === "complete") {
                ready();
            }
        });
    }
})(function perf() {
    var data = {
        url: window.location.href,
        FP: 0,
        FCP: 0
    };
    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.name === "first-paint") {
                data.FP = entry.startTime;
                console.log("记录FP: " + data.FP);
            } else if (entry.name === "first-contentful-paint") {
                data.FCP = entry.startTime;
                console.log("记录FCP: " + data.FCP);
            }
        });
    }).observe({ type: "paint", buffered: true });
});
```

3.1.2 改进FP和FCP

- 加快服务器响应速度
    - 升级服务器配置
    - 合理设置缓存
    - 优化数据库索引
- 加大服务器带宽
- 服务器开启 gzip压缩
- 开启服务器缓存(redis)
- 避免重定向操作
- 使用 dns-prefetch进行DNS进行预解析
- 采用域名分片技术突破同域6个TCP连接限制或者采用HTTP2
- 使用CDN减少网络跳转
- 压缩JS和CSS和图片等资源
    - TerserWebpackPlugin (https://webpack.docschina.org/plugins/terser-webpack-plugin/)
    - purgecss-webpack-plugin (https://www.npmjs.com/package/purgecss-webpack-plugin)
- 减少HTTP请求，合并JS和CSS，合理内嵌JS和CSS

3.1.2.1 site\index.js

site\index.js

```
const express = require('express');
const logger = require('morgan');
+const compression = require('compression')
const delayConfig = require('./delayConfig');
const app = express();
app.use(logger('dev'));

app.use((req, res, next) => {
    let url = req.url;
    let delay = delayConfig[url];
    if (delay) {
        setTimeout(next, delay);
    } else {
        next();
    }
});
+app.use(compression());
app.use(express.static('public', {
    setHeaders
}));
function setHeaders(res, path) {
    res.setHeader('cache-control', 'no-cache')
}
app.listen(80, () => console.log(`server started at 80`));
```

3.1.2.2 index.html

site\public\index.html

```
+
    lighthouse

+
```

3.2 .SI

- Speed Index(速度指数) (https://web.dev/speed-index/?utm_source=lighthouse&utm_medium=cli)表明了网页内容的可见填充速度
- 速度指数衡量页面加载期间内容的视觉显示速度

3.2.1 如何改进SI

3.2.1.1 最小化主线程工作

- 最小化主线程工作 (https://web.dev/mainthread-work-breakdown/)
- 脚本
    - 优化第三方的JS脚本 (https://web.dev/fast/#optimize-your-third-party-resources)
    - 对输入进行防抖处理 (https://developers.google.com/web/fundamentals/performance/rendering/debounce-your-input-handlers)

        - 使用 web workers (https://web.dev/off-main-thread/)
    - 样式和布局
        - 缩小样式计算的范围并降低其复杂性 (https://developers.google.com/web/fundamentals/performance/rendering/reduce-the-scope-and-complexity-of-style-calculations)
        - 避免复杂的布局和 布局抖动 (https://developers.google.com/web/fundamentals/performance/rendering/avoid-large-complex-layouts-and-layout-thrashing)
    - 渲染
        - 坚持仅合成器的属性和管理层计数 (https://developers.google.com/web/fundamentals/performance/rendering/stick-to-compositor-only-properties-and-manage-layer-count)
        - 简化绘制的复杂度、减小绘制区域 (https://developers.google.com/web/fundamentals/performance/rendering/simplify-paint-complexity-and-reduce-paint-areas)
    - 解析HTML和CSS
        - 提取关键CSS (https://web.dev/extract-critical-css/)
        - 压缩CSS (https://web.dev/minify-css/)
        - 延迟加载非关键的CSS (https://web.dev/defer-non-critical-css/)
    - 脚本解析和编译
        - 通过代码拆分减少JS的负载 (https://web.dev/reduce-javascript-payloads-with-code-splitting/)
        - 删除未使用的JS (https://web.dev/remove-unused-code/)
    - 垃圾收集
        - 监控网页的总内存使用情况 (https://web.dev/monitor-total-page-memory-usage/)

3.2.1.2 减少 JavaScript 执行时间

- 减少 JavaScript 执行时间 (https://web.dev/bootup-time/)
- 通过代码分割仅发送用户需要的代码 (https://web.dev/reduce-javascript-payloads-with-code-splitting/)
- 压缩代码 (https://web.dev/reduce-network-payloads-using-text-compression/)
- 删除未使用代码 (https://web.dev/remove-unused-code/)
- 使用PRPL模式缓存你的代码来减少网络开销 (https://web.dev/apply-instant-loading-with-prpl/)

3.2.1.3 确保文本在 webfont 加载期间保持可见

- 确保文本在 webfont 加载期间保持可见 (https://web.dev/font-display/)
- 字体通常是需要一段时间才能加载的大文件。一些浏览器在字体加载之前隐藏文本，导致不可见文本 (FOIT) 闪烁。
- 通过包含font-display: swap在您的@font-face风格中，您可以在大多数现代浏览器中避免 FOIT

```
@font-face {
  font-family: 'Pacifico';
  font-style: normal;
  font-weight: 400;
  src: local('Pacifico Regular'), local('Pacifico-Regular'), url(https://fonts.gstatic.com/s/pacifico/v12/FwZY7-Qmy14u9lezJ-6H6MmBp0u-.woff2) format('woff2');
  font-display: swap;
}
```

3.2.1.4 web workers

site\public\worker.html

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>webworkertitle>
head>

<body>
    <script>
        function start() {

            const worker = new Worker('/worker.js');
            worker.postMessage(100000000);
            worker.addEventListener('message', function (event) {
                console.log('sum:', event.data);
            });
        }
        function sum() {
            let total = 0;
            for (let i = 0; i < 100000000; i++) {
                total += i;
            }
            console.log('sum:', total);
        }

        start();
    script>
body>

html>
```

site\public\worker.js

```
self.addEventListener('message', function (event) {
    let total = 0;
    for (let i = 0; i < event.data; i++) {
        total += i;
    }
    self.postMessage(total);
});
```

3.2.1.5 避免强制同步布局和布局抖动

强制同步布局

- 改变一个元素的特性或者修改其内容有时不仅影响该元素，有时候会导致级联的变化，可能影响该元素的子节点、兄弟节点、父节点的改变，所以每次进行修改时，浏览器都必须重新计算这些改变的影响
- 如果我们编写的代码不能让浏览器有足够的时间和空间来进行优化，强制浏览器执行大量重新计算，就会造成布局抖动

接口对象 属性名 Element clientHeight, clientLeft, clientTop, clientWidth, focus, getBoundingClientRect, getClientRects, innerText, offsetHeight, offsetLeft, offsetParent, offsetTop, offsetTop, offsetWidth, outerText, scrollByLines, scrollByPages, scrollLeft, scrollHeight, scrollIntoView, scrollIntoViewIfNeeded, scrollTop, scrollWidth MouseEvent layerX layerY offsetX offsetY Window getComputedStyle scrollBy scrollTo scroll scrollY Frame,Document,Image height width

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>layouttitle>
head>

<body>
    <div id="root">div>
    <script>
        function reflow() {
            let container = document.getElementById('root');
            let div1 = document.createElement('div');
            div1.innerHTML = 'hello';
            container.appendChild(div1);
            console.log(container.offsetHeight);
            let div2 = document.createElement('div');
            div2.innerHTML = 'world';
            container.appendChild(div2);
            requestAnimationFrame(reflow);
        }
        requestAnimationFrame(reflow);
    script>
body>

html>
```

- 每次修改DOM，浏览器必须在读取任何布局信息之前先重新计算布局，对性能的损耗十分巨大
- 避免布局抖动的一种方法就是使用不会导致浏览器重排的方式编写代码 比如批量的读取和写入等

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>layouttitle>
    <style>
        .box {
            width: 100px;
            border: lpx solid green;
        }
    style>
head>

<body>
    <div class="box">box1div>
    <div class="box">box2div>
    <div class="box">box3div>
    <div class="box">box4div>
    <div class="box">box5div>
    <script src="https://cdn.bootcdn.net/ajax/libs/fastdom/1.0.10/fastdom.js">script>
    <script>
        let boxes = document.querySelectorAll('.box');
        for (let i = 0; i < boxes.length; i++) {
            const box = boxes[i];
            fastdom.measure(() => {
                const offsetWidth = box.offsetWidth;
                fastdom.mutate(() => {
                    box.style.width = offsetWidth + 5 + 'px';
                });
            });
        }
    script>
body>
html>
```

3.3 LCP

- Largest Contentful Paint(最大内容绘制) (https://web.dev/lighthouse-largest-contentful-paint)标记了渲染出最大文本或图片的时间
- 最大内容绘制 (LCP) 是测量感知加载速度的一个以用户为中心的重要指标

3.3.1 记录LCP

3.3.1.1 perf.js

site\public\perf.js

```
(function (ready) {
    if (document.readyState
        ready();
    } else {
        document.addEventListener("readystatechange", function () {
            if (document.readyState
                ready();
            }
        });
    }
})(function perf() {
    var data = {
        url: window.location.href,
        FP: 0,
        FCP: 0,
+       LCP: 0
    };
    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.name
                console.log("记录FP: " + (data.FP = entry.startTime));
            } else if (entry.name
                console.log("记录FCP: " + (data.FCP = entry.startTime));
            }
        });
    }).observe({ type: "paint", buffered: true });

+   new PerformanceObserver(function (entryList) {
+       var entries = entryList.getEntries() || [];
+       entries.forEach(function (entry) {
+           if (entry.startTime > data.LCP) {
+               console.log("记录LCP: " + (data.LCP = entry.startTime));
+           }
+       });
+   }).observe({ type: "largest-contentful-paint", buffered: true });
});
```

3.3.2 改进LCP

- 改进LCP (https://web.dev/lcp/#how-to-improve-largest-contentful-paint-on-your-site)
- 使用 PRPL 模式做到即时加载 (https://web.dev/apply-instant-loading-with-prpl/)
    - 推送(或预加载)最重要的资源 Preload是一个声明性的获取请求，它告诉浏览器尽快请求资源
    - 尽快渲染初始路线 内联首屏JS和CSS推荐其余部分
    - 预缓存剩余资源 Service Worker
    - 延迟加载其他路由和非关键资源
- 优化关键渲染路径 (https://developers.google.com/web/fundamentals/performance/critical-rendering-path/)
- 优化您的 CSS (https://web.dev/fast/#optimize-your-css)
- 优化您的图像 (https://web.dev/fast/#optimize-your-images)
- 优化网页字体 (https://web.dev/fast/#optimize-web-fonts)
- 优化您的JavaScript (https://web.dev/fast/#optimize-your-javascript)

```
<link rel="preload" as="style" href="css/style.css">
```

3.4. TTI

- Time to Interactive(可交互时间) (https://web.dev/tti/)指标测量页面从开始加载到主要子资源完成渲染，并能够快速、可靠地响应用户输入所需的时间
- webpagetest (https://www.webpagetest.org/)
- 虽然 TTI 可以在实际情况下进行测量，但我们不建议这样做，因为用户交互会影响您网页的 TTI，从而导致您的报告中出现大量差异。如需了解页面在实际情况中的交互性，您应该测量First Input Delay 首次输入延迟 (FID)

3.4.1 改进TTI

- 缩小JavaScript (https://web.dev/unminified-javascript/)
- 预连接到所需的来源 (https://web.dev/uses-rel-preconnect/)
- 预加载关键请求 (https://web.dev/uses-rel-preload/)
- 减少第三方代码的影响 (https://web.dev/third-party-summary/)
- 最小化关键请求深度 (https://web.dev/critical-request-chains/)
- 减少 JavaScript 执行时间 (https://web.dev/bootup-time/)
- 最小化主线程工作 (https://web.dev/mainthread-work-breakdown/)
- 保持较低的请求数和较小的传输大小 (https://web.dev/resource-summary/)

3.5 TBT

- total Blocking Time(总阻塞时间) (https://web.dev/tbt/)指标测量First Contentful Paint 首次内容绘制 (FCP)与Time to Interactive 可交互时间 (TTI)之间的总时间，这期间，主线程被阻塞的时间过长，无法作出输入响应
- 虽然 TBT 可以在实际情况下进行测量，但我们不建议这样做，因为用户交互会影响您网页的 TBT，从而导致您的报告中出现大量差异。如需了解页面在实际情况中的交互性，您应该测量First Input Delay 首次输入延迟 (FID)

3.5.1 如何改进TBT

- 减少第三方代码的影响 (https://web.dev/third-party-summary/)
- 减少 JavaScript 执行时间 (https://web.dev/bootup-time/)
- 最小化主线程工作 (https://web.dev/mainthread-work-breakdown/)
- 保持较低的请求数和较小的传输大小 (https://web.dev/resource-summary/)

3.6 FID

- 首次输入延迟 (FID) (https://web.dev/fid/)是测量加载响应度的一个以用户为中心的重要指标
- 因为该项指标将用户尝试与无响应页面进行交互时的体验进行了量化，低 FID 有助于让用户确信页面是有效的
- 首次输入延迟 (FID) 指标有助于衡量您的用户对网站交互性和响应度的第一印象
- FID 测量从用户第一次与页面交互（例如当他们单击链接、点按按钮或使用由 JavaScript 驱动的自定义控件）直到浏览器对交互作出响应，并实际能够开始处理事件处理程序所经过的时间

3.6.1 测试FID

3.6.1.1 perf.js

site\public\perf.js

```
(function (ready) {
    if (document.readyState
        ready();
    } else {
        document.addEventListener("readystatechange", function () {
            if (document.readyState
                ready();
            }
        });
    }
})(function perf() {
    var data = {
        url: window.location.href,
        FP: 0,
        FCP: 0,
        LCP: 0,
+       FID: 0
    };
    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.name
                console.log("记录FP: " + (data.FP = entry.startTime));
            } else if (entry.name
                console.log("记录FCP: " + (data.FCP = entry.startTime));
            }
        });
    }).observe({ type: "paint", buffered: true });

    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.startTime > data.LCP) {
                console.log("记录LCP: " + (data.LCP = entry.startTime));
            }
        });
    }).observe({ type: "largest-contentful-paint", buffered: true });

+   new PerformanceObserver((entryList) => {
+       for (const entry of entryList.getEntries()) {
+           const FID = entry.processingStart - entry.startTime;
+           console.log('FID:', FID, entry);
+       }
+   }).observe({ type: 'first-input', buffered: true });
});
```

3.6.2 改进FID

- 减少第三方代码的影响 (https://web.dev/third-party-summary/)
- 减少 JavaScript 执行时间 (https://web.dev/bootup-time/)
- 最小化主线程工作 (https://web.dev/mainthread-work-breakdown/)
- 保持较低的请求数和较小的传输大小 (https://web.dev/resource-summary/)

3.7 CLS

- [https://web.dev/cls/](Cumulative Layout Shift(累积布局偏移))是测量视觉稳定性的一个以用户为中心的重要指标
- CLS 测量整个页面生命周期内发生的所有意外布局偏移中最大一连串的布局偏移分数

3.7.1 如何计算CLS

3.7.1.1 perf.js

site\public\perf.js

```
(function (ready) {
    if (document.readyState
        ready();
    } else {
        document.addEventListener("readystatechange", function () {
            if (document.readyState
                ready();
            }
        });
    }
})(function perf() {
    var data = {
        url: window.location.href,
        FP: 0,
        FCP: 0,
        LCP: 0,
        FID: 0,
        CLS: 0
    };
    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.name
                console.log("记录FP: " + (data.FP = entry.startTime));
            } else if (entry.name
                console.log("记录FCP: " + (data.FCP = entry.startTime));
            }
        });
    }).observe({ type: "paint", buffered: true });

    new PerformanceObserver(function (entryList) {
        var entries = entryList.getEntries() || [];
        entries.forEach(function (entry) {
            if (entry.startTime > data.LCP) {
                console.log("记录LCP: " + (data.LCP = entry.startTime));
            }
        });
    }).observe({ type: "largest-contentful-paint", buffered: true });

    new PerformanceObserver((entryList) => {
        for (const entry of entryList.getEntries()) {
            const FID = entry.processingStart - entry.startTime;
            console.log('FID:', FID, entry);
        }
    }).observe({ type: 'first-input', buffered: true });

+   new PerformanceObserver((entryList) => {
+       var entries = entryList.getEntries() || [];
+       entries.forEach(function (entry) {
+           console.log('entry', entry);
+           if (!entry.hadRecentInput) {
+               data.CLS += entry.value;
+               console.log("CLS: " + data.CLS);
+           }
+       });
+   }).observe({ type: 'layout-shift', buffered: true });
});
```

3.7.2 如何改进CLS

- 始终在您的图像和视频元素上包含尺寸属性
- 首选转换动画，而不是触发布局偏移的属性动画
- 除非是对用户交互做出响应，否则切勿在现有内容的上方插入内容

3.7.2.1 perf.js

site\public\perf.js

```
    lighthouse
+   </span>
<span class="hljs-addition">+       @keyframes grow {</span>
<span class="hljs-addition">+           from {</span>
<span class="hljs-addition">+               /**transform: scaleY(0);**/</span>
<span class="hljs-addition">+               height: 200px;</span>
<span class="hljs-addition">+           }</span>
<span class="hljs-addition">+           to {</span>
<span class="hljs-addition">+               /**transform: scaleY(2);**/</span>
<span class="hljs-addition">+               height: 500px;</span>
<span class="hljs-addition">+           }</span>
<span class="hljs-addition">+       }</span>
<span class="hljs-addition">+       #banner {</span>
<span class="hljs-addition">+           animation: grow 3s infinite;</span>
<span class="hljs-addition">+       }</span>
<span class="hljs-addition">+

+
+
```

4. performance面板

4.1 面板说明

4.2 面板说明

4.3 核心流程

4.3.1 导航阶段

事件 含义 beforeunload 事件触发于 window、document 和它们的资源即将卸载时 navigationstart 相同的浏览环境下卸载前一个文档结束之时 pagehide 当浏览器在显示与会话历史记录不同的页面的过程中隐藏当前页面时, pagehide(页面隐藏)事件会被发送到一个Window visibilitychange 当浏览器的某个标签页切换到后台，或从后台切换到前台时就会触发该消息 unload 当文档或一个子资源正在被卸载时, 触发 unload事件 unloadEventEnd 事件处理程序结束之时 send request 发送请求 receive data 接收响应 commitNavigationEnd 提交本次导航结束 domLoading 解析器开始工作时

4.3.2 解析HTML阶段

事件 含义 receive data 接收数据 complete loading 完成加载 parseHTML 解析HTML recalculateStyle 重新计算样式 layout 布局 update layer tree 更新图层树 paint 绘制 raster GPU光栅化 compositor 复合图层 display 显示 dominteractive 主文档的解析器结束工作时 readystatechange interactive（可交互） domContentLoadedEventStart 所有的需要被运行的脚本已经被解析之时 DOMContentLoaded 当初始的 HTML 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发，而无需等待样式表、图像和子框架的完全加载 domContentLoadedEventEnd 这个时刻为所有需要尽早执行的脚本不管是否按顺序，都已经执行完毕 domComplete 主文档的解析器结束工作 readystatechange complete（完成） loadEventStart load事件被现在的文档触发之时 load 当整个页面及所有依赖资源如样式表和图片都已完成加载时，将触发load事件 loadEventEnd load事件处理程序被终止 pageshow 当一条会话历史记录被执行的时候将会触发页面显示(pageshow)事件

main.html

```
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <html>
    <head>
        <title>hellotitle>
        <style>
            #posts {
                width: 300px;
                height: 300px;
                background-color: green;
            }

            .post {
                width: 300px;
                height: 100px;
                background-color: red;
            }
        style>
    head>
<body>
    <div id="posts">div>
    <script>
        [illegible]
    script>
body>
html>
head>
body>
html>
```

```
    document.addEventListener('readystatechange', (event) => {
        console.log(event, document.readyState);
    });
```

4. Lighthouse优化

4.1 减少未使用的 JavaScript

- Remove unused JavaScript (https://web.dev/unused-javascript/?utm_source=lighthouse&utm_medium=cli)
- 请减少未使用的 JavaScript，并等到需要使用时再加载脚本，以减少网络活动耗用的字节数**4.2 采用新一代格式提供图片#**
- Serve images in modern formats (https://web.dev/uses-webp-images/?utm_source=lighthouse&utm_medium=cli)
- WebP 和 AVIF 等图片格式的压缩效果通常优于 PNG 或 JPEG，因而下载速度更快，消耗的数据流量更少

** 4.3 适当调整图片大小 #**

- Properly size images (https://web.dev/uses-responsive-images/?utm_source=lighthouse&utm_medium=cli)
- 提供大小合适的图片可节省移动数据网络流量并缩短加载用时

** 4.4 推迟加载屏幕外图片 #**

- Defer offscreen images (https://web.dev/offscreen-images/?utm_source=lighthouse&utm_medium=cli)
- 建议您在所有关键资源加载完毕后推迟加载屏幕外图片和处于隐藏状态的图片，从而缩短可交互前的耗时

** 4.5 移除阻塞渲染的资源 #**

- Eliminate render-blocking resources (https://web.dev/render-blocking-resources/?utm_source=lighthouse&utm_medium=cli)
- 资源阻止了系统对您网页的首次渲染。建议以内嵌方式提供关键的 JS/CSS，并推迟提供所有非关键的 JS/样式

** 4.6 减少未使用的 CSS #**

- Remove unused CSS (https://web.dev/unused-css-rules/?utm_source=lighthouse&utm_medium=cli)
- 请从样式表中减少未使用的规则，并延迟加载首屏内容未用到的 CSS，以减少网络活动耗用的字节数

** 4.7 使用视频格式提供动画内容 #**

- Use video formats for animated content (https://web.dev/efficient-animated-content/?utm_source=lighthouse&utm_medium=cli)
- 使用大型 GIF 提供动画内容会导致效率低下。建议您改用 MPEG4/WebM 视频（来提供动画）和 PNG/WebP（来提供静态图片）以减少网络活动消耗的字节数

** 4.8 预先连接到必要的来源 #**

- Preconnect to required origins (https://web.dev/uses-rel-preconnect/?utm_source=lighthouse&utm_medium=cli)
- 建议添加 `preconnect` 或 `dns-prefetch` 资源提示，以尽早与重要的第三方来源建立连接

** 4.9 应避免向新型浏览器提供旧版JavaScript #**

- Deploying ES2015+ Code in Production Today (https://philipwalton.com/articles/deploying-es2015-code-in-production-today/)
- Polyfill 和 transform 让旧版浏览器能够使用新的 JavaScript 功能。不过，其中的很多函数对新型浏览器而言并非必需。对于打包的 JavaScript，请采用现代脚本部署策略，以便利用 module/nomodule 功能检测机制来减少传送到新型浏览器的代码量，同时保留对旧版浏览器的支持

** 4.10 确保文本在网页字体加载期间保持可见状态 #**

- Ensure text remains visible during webfont load (https://web.dev/font-display/?utm_source=lighthouse&utm_medium=cli)
- 利用 font-display 这项 CSS 功能，确保文本在网页字体加载期间始终对用户可见

** 4.11 未使用被动式监听器来提高滚动性能 #**

- Use passive listeners to improve scrolling performance (https://web.dev/uses-passive-event-listeners/?utm_source=lighthouse&utm_medium=cli)
- 建议您将触摸和滚轮事件监听器标记为 `passive`，以提高页面的滚动性能, `passive`不会对事件的默认行为说 no

** 4.12 图片元素没有明确的width和height #**

- 请为图片元素设置明确的宽度值和高度值，以减少布局偏移并改善 CLS

** 4.13 注册"unload"事件监听器 #**

- Legacy lifecycle APIs to avoid (https://developers.google.com/web/updates/2018/07/page-lifecycle-api?utm_source=lighthouse&utm_medium=cli#the-unload-event)
- `unload`事件不会可靠地触发，而且监听该事件可能会妨碍系统实施"往返缓存"之类的浏览器优化策略。建议您改用 `pagehide`或 `visibilitychange`事件

** 4.14 最大限度地减少主线程工作 #**

- Minimize main thread work (https://web.dev/mainthread-work-breakdown/?utm_source=lighthouse&utm_medium=cli)
- 建议您减少为解析、编译和执行 JS 而花费的时间。您可能会发现，提供较小的 JS 负载有助于实现此目标

** 4.15 采用高效的缓存策略提供静态资源 #**

- Serve static assets with an efficient cache policy (https://web.dev/uses-long-cache-ttl/?utm_source=lighthouse&utm_medium=cli)
- 延长缓存期限可加快重访您网页的速度

** 4.16 缩短 JavaScript 执行用时 #**

- Reduce JavaScript execution time (https://web.dev/bootup-time/?utm_source=lighthouse&utm_medium=cli)
- 建议您减少为解析、编译和执行 JS 而花费的时间。您可能会发现，提供较小的 JS 负载有助于实现此目标

** 4.17 避免链接关键请求 #**

- Avoid chaining critical requests (https://web.dev/critical-request-chains/?utm_source=lighthouse&utm_medium=cli)
- 下面的关键请求链显示了以高优先级加载的资源。请考虑缩短链长、缩减资源的下载文件大小，或者推迟下载不必要的资源，从而提高网页加载速度

** 4.18 请保持较低的请求数量和较小的传输大小 #**

- Use Lighthouse for performance budgets (https://web.dev/use-lighthouse-for-performance-budgets/?utm_source=lighthouse&utm_medium=cli)
- performancebudget (https://www.performancebudget.io/)
- 若要设置页面资源数量和大小的预算，请添加 budget.json 文件

** 4.19 最大内容渲染时间元素 #**

- Largest Contentful Paint (https://web.dev/lighthouse-largest-contentful-paint/?utm_source=lighthouse&utm_medium=cli)
- 这是此视口内渲染的最大内容元素

** 4.20 请避免出现大幅度的布局偏移 #**

- 这些 DOM 元素对该网页的 CLS 影响最大

** 4.21 应避免出现长时间运行的主线程任务 #**

- Are long JavaScript tasks delaying your Time to Interactive? (https://web.dev/long-tasks-devtools/?utm_source=lighthouse&utm_medium=cli)
- 列出了主线程中运行时间最长的任务，有助于识别出导致输入延迟的最主要原因

** 4.22 避免使用未合成的动画 #**

- Avoid non-composited animations (https://web.dev/non-composited-animations/?utm_source=lighthouse&utm_medium=cli)
- 未合成的动画可能会卡顿并增加 CLS

** 4.23 缩减 CSS #**

- Minify CSS (https://web.dev/unminified-css/?utm_source=lighthouse&utm_medium=cli)
- 缩减 CSS 文件大小可缩减网络负载规模

** 4.24 缩减 JavaScript #**

- Minify JavaScript (https://web.dev/unminified-javascript/?utm_source=lighthouse&utm_medium=cli)
- 如果缩减 JavaScript 文件大小，则既能缩减负载规模，又能缩短脚本解析用时

** 4.25 对图片进行高效编码 #**

- Efficiently encode images (https://web.dev/uses-optimized-images/?utm_source=lighthouse&utm_medium=cli)
- 如果图片经过了优化，则加载速度会更快，且消耗的移动数据网络流量会更少

** 4.26 启用文本压缩 #**

- Enable text compression (https://web.dev/uses-text-compression/?utm_source=lighthouse&utm_medium=cli)
- 对于文本资源，应先压缩（gzip、deflate 或 brotli），然后再提供，以最大限度地减少网络活动消耗的字节总数

** 4.27 初始服务器响应用时较短 #**

- Reduce server response times (TTFB) (https://web.dev/time-to-first-byte/?utm_source=lighthouse&utm_medium=cli)
- 请确保服务器响应主文档的用时较短，因为这会影响到所有其他请求的响应时间

** 4.28 避免多次网页重定向 #**

- Avoid multiple page redirects (https://web.dev/redirects/?utm_source=lighthouse&utm_medium=cli)
- 重定向会在网页可加载前引入更多延迟

** 4.29 预加载关键请求 #**

- Preload key requests (https://web.dev/uses-rel-preload/?utm_source=lighthouse&utm_medium=cli)
- 建议使用 `<link rel="preload">` 来优先提取当前在网页加载后期请求的资源

** 4.30 使用 HTTP/2 #**

- Does not use HTTP/2 for all of its resources (https://web.dev/uses-http2/?utm_source=lighthouse&utm_medium=cli)
- HTTP/2 提供了许多优于 HTTP/1.1 的益处，包括二进制标头和多路复用

** 4.31 请移除 JavaScript 软件包中的重复模块 #**

- 从软件包中移除重复的大型 JavaScript 模块可减少网络传输时不必要的流量消耗

** 4.32 预加载 LCP 元素所用图片 #**

- 优化 Largest Contentful Paint 最大内容绘制 (https://web.dev/optimize-lcp/?utm_source=lighthouse&utm_medium=cli#preload-important-resources)
- 请预加载 Largest Contentful Paint (LCP) 元素所用的图片，以缩短您的 LCP 用时

** 4.33 避免网络负载过大 #**

- Avoid enormous network payloads (https://web.dev/total-byte-weight/?utm_source=lighthouse&utm_medium=cli)
- 网络负载过大不仅会让用户付出真金白银，还极有可能会延长加载用时

** 4.34 避免 DOM 规模过大 #**

- Avoid an excessive DOM size (https://web.dev/dom-size/?utm_source=lighthouse&utm_medium=cli)
- 大型 DOM 会增加内存使用量、导致样式计算用时延长，并产生高昂的布局重排成本

** 4.35 User Timing 标记和测量结果 #**

- User Timing marks and measures (https://web.dev/user-timings/?utm_source=lighthouse&utm_medium=cli)
- 建议使用 User Timing API 检测您的应用，从而衡量应用在关键用户体验中的实际性能

** 4.36 尽量减少第三方使用 #**

- Loading Third-Party JavaScript (https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript/?utm_source=lighthouse&utm_medium=cli)
- 第三方代码可能会显著影响加载性能。请限制冗余第三方提供商的数量，并尝试在页面完成主要加载后再加载第三方代码

** 4.37 使用 Facade 延迟加载第三方资源 #**

- Lazy load third-party resources with facades (https://web.dev/third-party-facades/?utm_source=lighthouse&utm_medium=cli)
- 您可以延迟加载某些第三方嵌入代码。不妨考虑使用 Facade 替换这些代码，直到您需要使用它们为止

** 4.38 Largest Contentful Paint 所对应的图片未被延迟加载 #**

- The performance effects of too much lazy-loading (https://web.dev/lcp-lazy-loading/?utm_source=lighthouse&utm_medium=cli)
- 被延迟加载的首屏图片会在页面生命周期内的较晚时间呈现，这可能会致使系统延迟渲染最大内容元素

** 4.39 请勿使用 document.write() #**

- Uses document.write() (https://web.dev/no-document-write/?utm_source=lighthouse&utm_medium=cli)
- 对于连接速度较慢的用户，通过 `document.write()` 动态注入的外部脚本可将网页加载延迟数十秒

** 4.40 具有包含 width 或 initial-scale 的 标记 #**

- Does not have a tag with width or initial-scale (https://web.dev/viewport/?utm_source=lighthouse&utm_medium=cli)
- `<meta name="viewport">` 不仅会针对移动设备屏幕尺寸优化您的应用，还会阻止系统在响应用户输入前出现 300 毫秒的延迟

5.参考

- web-worker (http://www.ruanyifeng.com/blog/2018/07/web-worker.html)
- https://crux-compare.netlify.app (https://crux-compare.netlify.app)
- https://web.dev (https://web.dev)
- https://developer.mozilla.org/zh-CN/docs/Web/API/performance_property (https://developer.mozilla.org/zh-CN/docs/Web/API/performance_property)
- IntersectionObserver懒加载 (https://www.jianshu.com/p/84a86e41eb2b)
- https://app.requestmetrics.com (https://app.requestmetrics.com)
- performancebudget (https://www.performancebudget.io/)
- readystatechange_event (https://developer.mozilla.org/zh-CN/docs/Web/API/Document/readystatechange_event)
- readyState (https://developer.mozilla.org/zh-CN/docs/Web/API/Document/readyState)