前段时间阅读了饿了么的 PWA 升级实践一文，受益匪浅。其中构建时使用 Vue 预渲染骨架屏一节，为开发者提供了减少白屏时间，提升用户感知体验的新思路。本文将借鉴这一思路，尝试为 Vue 项目添加骨架屏。

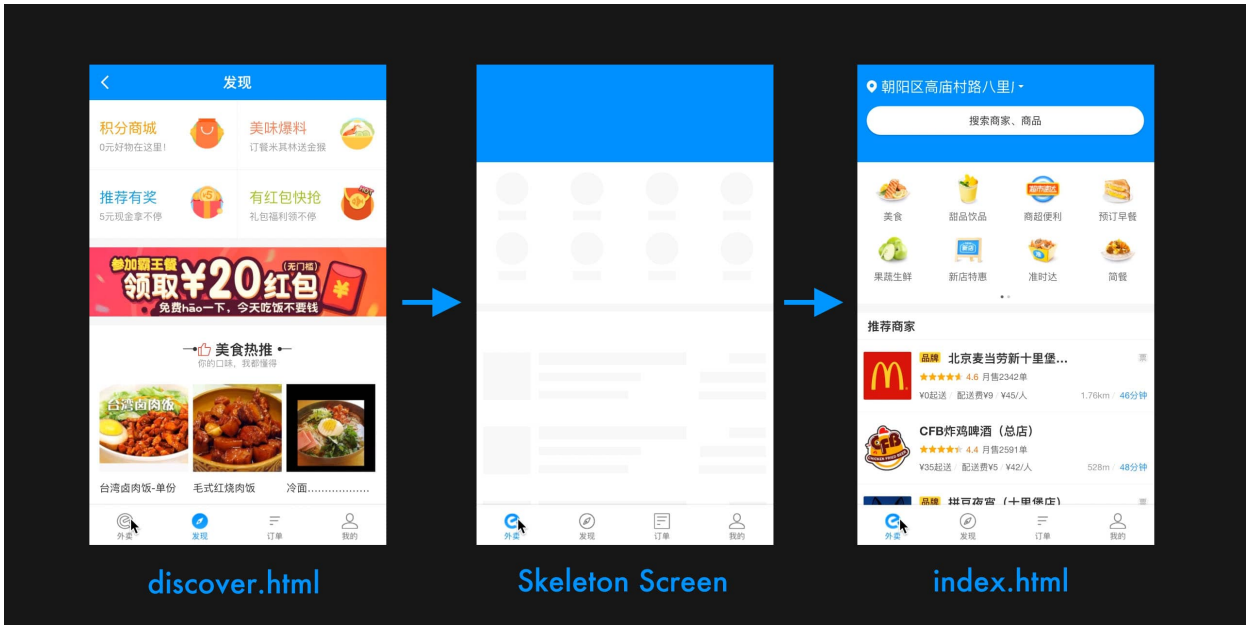
骨架屏是什么？

在 Google 提出的以用户为中心的四个页面性能衡量指标中，FP/FCP(首屏渲染)可能是开发者最熟悉的了。下图来自原文：

关于尽快渲染出首屏，减少白屏时间，我能想到的优化方式大致有以下几种：

- 优化 Critical Rendering Path(关键渲染路径)，尽可能减少阻塞渲染的 JavaScript 和 CSS。常见做法包括使用 async/defer 让浏览器下载 JavaScript 的同时不阻塞 HTML 解析，内联页面关键部分的样式到 HTML 中等。
- 使用 Service Worker 缓存 AppShell，加快后续访问速度。
- 使用 HTTP/2 Server Push，帮助浏览器尽早发现静态资源，减少请求数。浅谈 HTTP/2 Server Push一文介绍了 Ele.me 在这方面的实践，推送 API 请求而非静态资源。

骨架屏充分利用了前两点。下图来自原文为感知体验奋斗一节。从图中的 Skeleton Screen (骨架屏)中可以看出，在页面完全渲染完成之前，用户会看到一个样式简单，描绘了当前页面的大致框架，感知到页面正在逐步加载，最终骨架屏中各个占位部分被完全替换，体验良好。



骨架屏可以看成一个简单的关键渲染路径，由于只是页面的大致框架，样式不会太复杂，内联在 HTML 中体积很小。使用 Service Worker 缓存包含骨架屏的 HTML 页面之后，从缓存中取出展示速度更快。

实现思路

参考原文中在构建时使用 Vue 预渲染骨架屏一节介绍的思路，我将骨架屏也看成路由组件，在构建时使用 Vue 预渲染功能，将骨架屏组件的渲染结果 HTML 片段插入 HTML 页面模版的挂载点中，将样式内联到 head 标签中。这样等前端渲染完成时，Vue 将使用客户端混合，把挂载点中的骨架屏内容替换成真正的页面内容。

有了以上思路，让我们看看如何为一个简单的 Vue 应用添加骨架屏。

具体实现

为此我开发了一个 webpack 插件：vue-skeleton-webpack-plugin。下面将从以下三方面介绍部分实现细节：

- 使用 Vue 预渲染骨架屏
- 将骨架屏渲染结果插入 HTML 模版中

- 开发模式下插入各个骨架屏路由

使用 Vue 预渲染骨架屏

我们使用 Vue 的预渲染功能渲染骨架屏组件，不熟悉的同学可以先阅读官方文档中的基本用法一节。

首先需要创建一个仅使用骨架屏组件的入口文件：

```
// src/entry-skeleton.jsimport Skeleton from'./Skeleton.vue';// 创建一个骨架屏 Vue 实例exportdefaultnewVue({    components:{        Skeleton    },    template:''});
```

接下来创建一个用于服务端渲染的 webpack 配置对象，将刚创建的入口文件指定为 entry 依赖入口：

```
// webpack.skeleton.conf.js{    target:'node',// 区别默认的 'web'    entry:resolve('./src/entry-skeleton.js'),// 多页传入对象    output:{        libraryTarget:'commonjs2'},    externals:nodeExternals({        whitelist:/\.css$/}),    plugins:[]}
```

> 这里只展示单页应用的情况，在多页应用中，指定 entry 为包含各个页面入口的对象即可。关于多页中的 webpack 配置对象示例，可参考插件的多页测试用例或者Lavas MPA 模版。

然后我们将这个 webpack 配置对象通过参数传入骨架屏插件中。

```
// webpack.dev.conf.jsplugins:[newSkeletonWebpackPlugin({// 我们编写的插件        webpackConfig:require('./webpack.skeleton.conf')})]
```

骨架屏插件运行时会使用 webpack 编译这个传入的配置对象，得到骨架屏的 bundle 文件。接下来只需要使用这个 bundle 文件内容创建一个 renderer，调用renderToString()方法就可以得到字符串形式的 HTML 渲染结

果了。由于我们不需要将过程中的文件产物保存在硬盘中，使用内存文件系统memory-fs即可。

```
// vue-skeleton-webpack-plugin/src/ssr.jsconst createBundleRenderer =require('vue-server-renderer').createBundleRenderer;// 从内存文件系统中读取 bundle 文件let bundle = mfs.readFileSync(outputPath,'utf-8');// 创建 rendererlet renderer =createBundleRenderer(bundle);// 渲染得到 HTMLrenderer.renderToString({},(err, skeletonHtml)=>{});
```

默认情况下，webpack 模块引用的样式内容是内嵌在 JavaScript bundle 中的。官方插件 ExtractTextPlugin可以进行样式分离。我们也使用这个插件，将骨架屏样式内容输出到单独的 CSS 文件中。 关于插件更多用法，可参考官方文档或者 Vue 基于 webpack 的模版。

```
// vue-skeleton-webpack-plugin/src/ssr.js// 加入 ExtractTextPlugin 插件到 webpack 配置对象插件列表中
serverWebpackConfig.plugins.push(newExtractTextPlugin({    filename: outputCssBasename // 样式文件名}));
```

至此，我们已经得到了骨架屏的渲染结果 HTML 和样式内容，接下来需要关心如何将结果注入 HTML 页面模版中。

注入渲染结果

Vue webpack 模版项目使用了 HTML Webpack Plugin生成 HTML 文件。参考该插件的 事件说明，我们选择监听html-webpack-plugin-before-html-processing事件，在事件的回调函数中，插件会传入当前待处理的 HTML 内容供我们进一步修改。

我们知道骨架屏组件最终的渲染结果包含 HTML 和样式两部分，样式部分可以直接插入 head 标签内，而 HTML 需要插入挂载点中。插件使用者可以通

过参数设置这个挂载点位置，默认将使用

。

看起来一切都很顺利，但是在多页应用中，情况会变的稍稍复杂。多页项目中通常会引入多个 HTML Webpack Plugin，例如我们在 Lavas MPA 模版中使用的 Multipage Webpack 插件就是如此，这就会导致html-webpack-plugin-before-html-processing事件被多次触发。

在多页应用中，我们传给骨架屏插件的 webpack 配置对象是包含多个入口的：

```
// webpack.skeleton.conf.jsentry:{
page1:resolve('./src/pages/page1/entry-skeleton.js'),
page2:resolve('./src/pages/page2/entry-skeleton.js')}
```

这就意味着每次html-webpack-plugin-before-html-processing事件触发时，骨架屏插件都需要识别出当前正在处理的入口文件，执行 webpack 编译当前页面对应的骨架屏入口文件，渲染对应的骨架屏组件。查找当前处理的入口文件过程如下：

```
// vue-skeleton-webpack-plugin/src/index.js// 当前页面使用的所有chunkslet usedChunks = htmlPluginData.plugin.options.chunks;let entryKey;// chunks 和所有入口文件的交集就是当前待处理的入口文件if(Array.isArray(usedChunks)){    entryKey = Object.keys(skeletonEntries);    entryKey = entryKey.filter(v => usedChunks.indexOf(v)>-1)[0];}// 设置当前的 webpack 配置对象的入口文件和结果输出文件webpackConfig.entry = skeletonEntries[entryKey];webpackConfig.output.filename =`skeleton-${entryKey}.js`;// 使用配置对象进行服务端渲染ssr(webpackConfig).then(({skeletonHtml, skeletonCss})=>{// 注入骨架屏 HTML 和 CSS 到页面 HTML 中});
```

至此，我们已经完成了骨架屏的渲染和注入工作，接下来有一个开发中的小问题需要关注。

开发模式下插入路由

前面说过，由于 Vue 会使用客户端混合，骨架屏内容在前端渲染完成后就会被替换，那么如何在开发时方便的查看调试呢？

使用浏览器开发工具设置断点是一个办法，但如果能在开发模式中向路由文件插入骨架屏组件对应的路由规则，使各个页面的骨架屏能像其他路由组件一样被访问，将使开发调试变得更加方便。向路由文件插入规则代码的工作将在插件的 loader中完成。如果您对 webpack loader 还不了解，可以参阅官方文档。

我们需要向路由文件插入两类代码：引入骨架屏组件的代码和对应的路由规则对象。关于代码插入点，引入组件代码相对简单，放在文件顶部就行了，而路由规则需要插入路由对象数组中，目前我使用的是简单的字符串匹配来查找这个数组的起始位置。例如下面的例子中，需要向 loader 传入routes:[来确定插入路由的位置。

```
// router.jsimport Skeleton from'@/pages/Skeleton.vue'routes:[{// 插入骨架屏路由        path:'/skeleton',        name:'skeleton',component: Skeleton    }// ...其余路由规则]
```

在多页应用中，每个页面对应的骨架屏都需要插入代码，使用者可以通过占位符设置引入骨架屏组件语句和路由规则的模版。loader 在运行时会使用这些模版，用真实的骨架屏名称替换掉占位符。在下面的例子中，假设我们有Page1.skeleton.vue和Page2.skeleton.vue这两个骨架屏，开发模式下可以通过/skeleton-page1和/skeleton-page2访问这两个骨架屏路由。更多参数说明可以参考这里。

```
// webpack.dev.conf.jsmodule:{    rules:[]// 其他规
则.concat(SkeletonWebpackPlugin.loader({
resource:resolve('src/router.js'),// 目标路由文件              options:{
entry:['page1','page2'],                  importTemplate:'import
[nameCap] from \'@/pages/[name]/[nameCap].skeleton.vue\';',
routePathTemplate:'/skeleton-[name]'}}))}
```

多页中的具体应用示例，可参考多页测试用例或者Lavas MPA 模版。

总结

感谢饿了么的 PWA 升级实践一文，提供了解决这个问题的思路。当然文章中包含的远不止骨架屏这方面，相信每个读者都会受益匪浅。

插件使用中遇到任何问题，都欢迎提出 ISSUE。

百度 Lavas是一个基于 Vue 的 PWA 解决方案，帮助开发者轻松搭建 PWA 站点。其中多个模版也使用了这个插件，欢迎大家试用并提出意见。