**单页面应用**

（SinglePage Web Application，SPA）

**概念**：单页应用的全称是 single-page application，简称 SPA，它是一种网站应用的模型，它可以动态重写当前的页面来与用户交互，而不需要重新加载整个页面。常用于PC端官网、购物等网站

简单来说——SPA是**只有一张Web页面**的应用，一种从Web服务器加载的富客户端，单页面跳转仅刷新局部资源 ，公共资源(js、css等)仅需加载一次。

**基本元素**：页面路由

**工作原理：**SPA 的一个重要实现就是改变路由时，页面不刷新。实现这个功能，通常有两种方式：使用 window.history 对象或 location.hash。

**背景**：在早期的 Web 应用中，前端的作用很弱，业务逻辑和数据处理都在后端，后端给前端返回一个 HTML 页面，前端只负责展示。在这种模式下，单个 HTML 页面被当做一个功能原件，表单提交，超链接跳转都可以用来重新请求新的 HTML 页面，从而达到给用户展示新的页面的目的。在这种方式中，你可以感觉到页面总是在不停的刷新，一个点击可能都需要等待很久，这是因为，当页面变化时，所有资源都需要重新请求。这些问题导致了用户体验不佳，也浪费了资源。单页应用便很好的解决了这些问题。

**优点**：

1. 流畅性更好：单页应用的流畅性让 Web 应用更像桌面端或 Native 应用了。
2. 前后端分离：相对于传统的 Web 应用，单页应用做到了**前后端分离**，后端只负责处理数据提供接口，页面逻辑和页面渲染都交给了前端。各司其职，不会把前后端的逻辑混杂在一起；
3. 应用广泛：目前时兴的 React、Vue、Angular 等前端框架均采用了 SPA 原则。
4. 单页应用意味着前端掌握了主动权，也让前端代码更复杂和庞大，模块化、组件化以及架构设计都变得越来越重要。
5. 无刷新体验：用户在切换页面过程中不会频繁被“打断”，因为界面框架都在本地，对用户的响应非常及时，因此提升了用户体验；
6. 减轻服务器压力：服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍；
7. API 共享：同一套后端程序代码，不用修改就可以用于Web界面、手机、平板等多种客户端；
8. 完全的前端组件化：前端开发不再以页面为单位，更多地采用组件化的思想，代码结构和组织方式更加规范化，便于修改和调整。

**缺点：**

1. 对 SEO 不太友好，尽管可以通过 Prerender 预渲染优化等技术解决一部分，但是相对还是不容易索引到它；（seo 本质是一个服务器向另一个服务器发起请求，解析请求内容; Prerender 可以对这些使用了前端渲染的 JavaScript 框架做的网站进行良好的 SEO 优化, 可以对 JavaScript 页面进行静态化。）

解决方案：

* 服务端渲染：服务器合成完整的 html 文件再输出到浏览器；
* 页面预渲染；
* 路由采用h5 history模式；

1. 易出错，需要使用程序管理前进、后退、地址栏等信息；
2. 较高的前端开发门槛，对技术能力要求较高，需要对设计模式有一定理解，因为面对不是一个简单的页面，而是一个运行在浏览器环境里面的桌面软件。
3. 首屏加载慢：如果不对路由进行处理，在加载首页的时候，就会将所有组件全部加载，并向服务器请求数据，这必将拖慢加载速度；通过查看Network，发现整个网站加载试讲长达10几秒，加载时间最长的就是js、css文件和媒体文件及图片

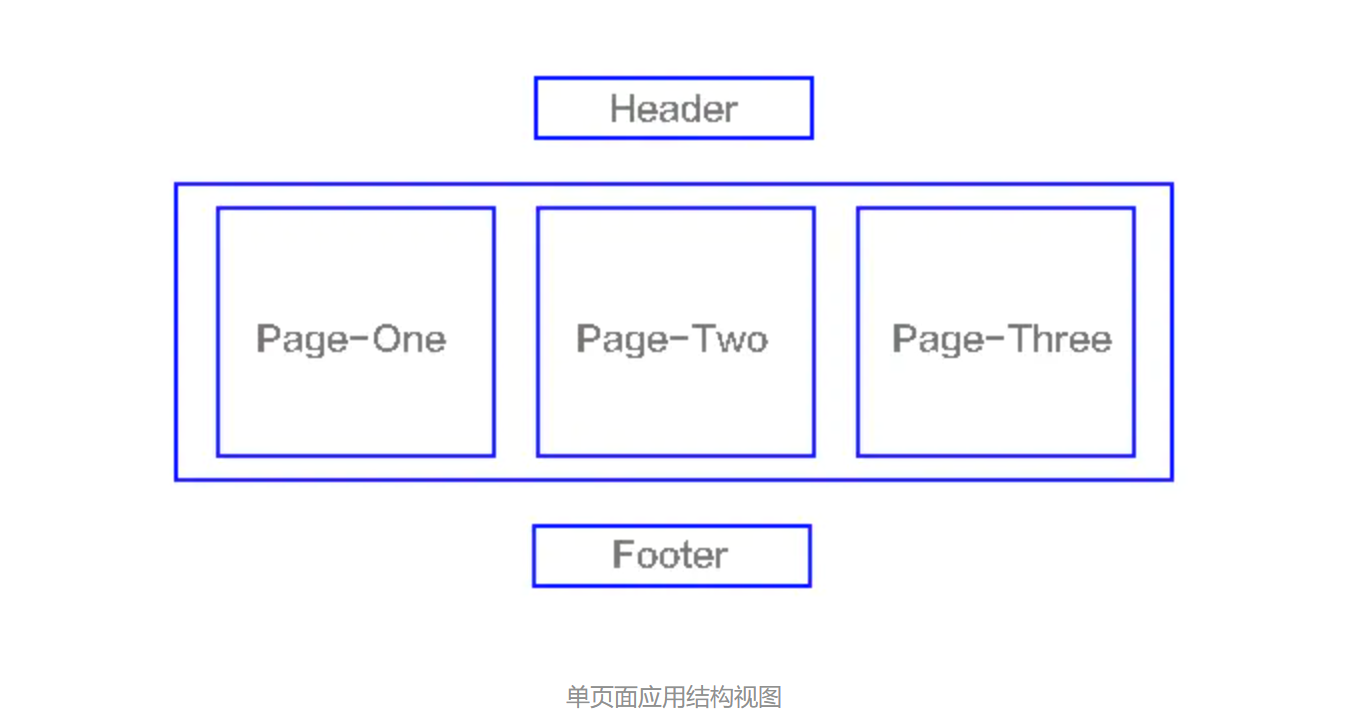
解决办案：

* Vue-router懒加载：Vue-router懒加载就是按需加载组件，只有当路由被访问时才会加载对应的组件，而不是在加载首页的时候就加载，项目越大，对首屏加载的速度提升得越明显。
* 使用CDN加速：在做项目时，我们会用到很多库，采用CDN加载可以加快加载速度。
* 异步加载组件：这里可以参考别人的介绍（<https://segmentfault.com/a/1190000012138052>）
* 服务端渲染：服务端渲染还能对seo优化起到作用，有利于搜索引擎抓取更多有用的信息（如果页面纯前端渲染，搜索引擎抓取到的就只是空页面）

1. 不适合开发大型项目：大型项目中可能会涉及大量的DOM操作、复杂的动画效果，也就不适合使用Vue、react框架进行开发。

**单页面与多页面对比：**

1. 结构示意图



1. 具体对比分析

|  | **单页面应用SPA** | **多页面应用MPA** |
| --- | --- | --- |
| 组成 | 一个外壳页面和多个页面片段组成 | 多个完整页面构成 |
| 资源共用(css,js) | 共用，只需在外壳部分加载 | 不共用，每个页面都需要加载 |
| 刷新方式 | 页面局部刷新或更改 | 整页刷新 |
| url 模式 | a.com/#/pageone a.com/#/pagetwo | a.com/pageone.html a.com/pagetwo.html |
| 用户体验 | 页面片段间的切换快，用户体验良好 | 页面切换加载缓慢，流畅度不够，用户体验比较差 |
| 转场动画 | 容易实现 | 无法实现 |
| 数据传递 | 容易 | 依赖 url传参、或者cookie 、localStorage等 |
| 搜索引擎优化(SEO) | 需要单独方案、实现较为困难、不利于SEO检索 可利用服务器端渲染(SSR)优化 | 实现方法简易 |
| 试用范围 | 高要求的体验度、追求界面流畅的应用 | 适用于追求高度支持搜索引擎的应用 |
| 开发成本 | 较高，常需借助专业的框架 | 较低 ，但页面重复代码多 |
| 维护成本 | 相对容易 | 相对复杂 |

**实现流程：**

**拓展：**

1. vue-router有两种模式，hash模式和history模式

Hash: 这里的hash是指url尾巴后的#号及后面的字符。这里的#和css里的#是一个意思。hash也称作锚点，本身是用来做页面定位的，她可以使对应id的元素显示在可是区域内。由于hash值变化不会导致浏览器向服务器发出请求，而且hash改变会触发hashchange事件，浏览器的进后退也能对其进行控制，所以人们在 html5 的 history 出现前，基本都是使用 hash 来实现前端路由的。

History: HTML5规范提供了history.pushState和history.replaceState来进行路由控制。通过这两个方法可以改变url且不向服务器发送请求。同时不会像hash有一个#,更加的美观。但是history路由需要服务器的支持，并且需将所有的路由重定向倒根页面。

1. history与hash比较

hash 能兼容到IE8， history 只能兼容到 IE10；

首先，hash 本来是拿来做页面定位的，如果拿来做路由的话，原来的锚点功能就不能用了。其次，hash 的传参是基于 url 的，如果要传递复杂的数据，会有体积的限制，而 history 模式不仅可以在url里放参数，还可以将数据存放在一个特定的对象中。vue-router默认hash模式，使用 URL 的 hash 来模拟一个完整的 URL，于是当 URL 改变时，页面不会重新加载。不过这种模式要玩好，还需要后台配置支持。

因为我们的应用是个单页客户端应用，如果后台没有正确的配置。当用户在浏览器直接访问http://oursite.com/user/id 就会返回 404，这就不好看了。所以，你要在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个index.html 页面，这个页面就是你 app 依赖的页面。