Vue SEO优化之路

2019.11.21

杨梨梨

目录

预渲染的用途

SPA为什么不利于SEO

- 1.搜索引擎的基础爬虫的原理就是抓取你的url，然后获取你的html源代码并解析。 而你的页面通常用了vue等js的数据绑定机制来展示页面数据，爬虫获取到的html是你的模型页面而不是最终数据的渲染页面，所以说用js来渲染数据对seo并不友好。
- 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。但一般来说搜索引擎是不回去执行请求到的js的。也就是说，如果一个单页应用，html在服务器端还没有渲染部分数据数据，在浏览器才渲染出数据，而搜索引擎请求到的html是没有渲染数据的。 这样就很不利于内容被搜索引擎搜索到。 所以服务端渲染就是尽量在服务器发送到浏览器前 页面上就是有数据的。

SPA为什么不利于SEO

- vue线上项目的DOC展开图，内容区域趋于空，搜索引擎能找的有效信息太少了。

```
<!DOCTYPE html>
<html lang=en>
    <head>
        <meta charset=UTF-8>
        <meta name=viewport content="">
        <meta http-equiv=X-UA-Compatible content="ie=edge">
        <title>青少年爱挑战</title>
        <meta name=keywords content=青少年,爱挑战,素质教育,活动平台,大赛>
        <meta name=description content=爱挑战大赛旨在发展青少年综合素质，让更多的学生接轨信息化教育，推动青少年素质教育。打造素质教育和互联网+信息教育的品牌。>
        <link rel="shortcut icon" href=/fav.png type=image/x-icon>
        <link href=/css/chunk-112b16fb.02418ca0.css rel=prefetch>
        <link href=/css/chunk-1a7895a1.8cda87da.css rel=prefetch>
        <link href=/css/chunk-27925f17.2a39e588.css rel=prefetch>
        <link href=/css/chunk-29a33146.59f6ef95.css rel=prefetch>
        <link href=/css/chunk-29b3462c.7ecf61c2.css rel=prefetch>
        <link href=/css/chunk-2c58cc9f.15c9c431.css rel=prefetch>
        <link href=/css/chunk-36222c52.5c32066b.css rel=prefetch>
        <link href=/css/chunk-40ca78fe.53911c50.css rel=prefetch>
        <link href=/css/chunk-431875c7.15c9c431.css rel=prefetch>
        <link href=/css/chunk-435d30fc.75b04f8f.css rel=prefetch>
        <link href=/css/chunk-45bb2cab.051512b1.css rel=prefetch>
        <link href=/css/chunk-6bf49a67.de2a3661.css rel=prefetch>
        <link href=/css/chunk-72594d86.38a195ec.css rel=prefetch>
        <link href=/css/chunk-767df4c6.8d7d9261.css rel=prefetch>
        <link href=/css/chunk-a5306d92.7663f3a2.css rel=prefetch>
        <link href=/css/chunk-a58e3892.71d1cf10.css rel=prefetch>
        <link href=/css/chunk-bf8b7aa6.69bad7a8.css rel=prefetch>
        <link href=/css/chunk-df41a074.e8a5e20b.css rel=prefetch>
        <link href=/css/chunk-e59f2c04.1580efe2.css rel=prefetch>
        <link href=/css/chunk-f3e12444.15c9c431.css rel=prefetch>
        <link href=/js/chunk-112b16fb.990597e7.js rel=prefetch>
        <link href=/js/chunk-1a7895a1.97c2ff3d.js rel=prefetch>
        <link href=/js/chunk-27925f17.9b2ddfd7.js rel=prefetch>
        <link href=/js/chunk-29a33146.eec3a50b.js rel=prefetch>
        <link href=/js/chunk-29b3462c.3078eb66.js rel=prefetch>
        <link href=/js/chunk-2c58cc9f.f0957f69.js rel=prefetch>
        <link href=/js/chunk-2d217dac.35a4272e.js rel=prefetch>
        <link href=/js/chunk-36222c52.f15a0b6f.js rel=prefetch>
        <link href=/js/chunk-40ca78fe.6b5a4ade.js rel=prefetch>
        <link href=/js/chunk-431875c7.f2f4a764.js rel=prefetch>
        <link href=/js/chunk-435d30fc.b6eedec1.js rel=prefetch>
        <link href=/js/chunk-45bb2cab.a7e5bcf7.js rel=prefetch>
        <link href=/js/chunk-6bf49a67.653fcb08.js rel=prefetch>
        <link href=/js/chunk-72594d86.9c91a7d0.js rel=prefetch>
        <link href=/js/chunk-767df4c6.862a0873.js rel=prefetch>
        <link href=/js/chunk-a5306d92.b44c290c.js rel=prefetch>
        <link href=/js/chunk-a58e3892.7dd42f96.js rel=prefetch>
        <link href=/js/chunk-bf8b7aa6.6a30b07f.js rel=prefetch>
        <link href=/js/chunk-df41a074.553d774f.js rel=prefetch>
        <link href=/js/chunk-e59f2c04.a33f6ab2.js rel=prefetch>
        <link href=/js/chunk-f3e12444.27c95e68.js rel=prefetch>
        <link href=/fonts/iconfont.1cee0e1d.ttf rel=preload as=font crossorigin="">
        <link href=/fonts/iconfont.8d729ed1.woff rel=preload as=font crossorigin="">
        <link href=/fonts/iconfont.9ac2cc5a.ttf rel=preload as=font crossorigin="">
        <link href=/fonts/iconfont.bf0fc2ec.woff rel=preload as=font crossorigin="">
        <link href=/fonts/ionicons.143146fa.woff2 rel=preload as=font crossorigin="">
        <link href=/fonts/ionicons.99ac3308.woff rel=preload as=font crossorigin="">
        <link href=/fonts/ionicons.d535a25a.ttf rel=preload as=font crossorigin="">
        <link href=/css/app.7f9f4273.css rel=stylesheet>
    </head>
    <body>
        <noscript>
            <strong>抱歉，爱挑战活动大赛无法运行在关闭JavaScript支持的浏览器中</strong>
        </noscript>
        <!--[if IE]>
            <div
                style="
            width: 100%; height: 30px; line-height: 30px;
            text-align: center; background: #ffd600; font-size: 14px;
            color: black;
        "
            >
                为了获得更好的体验，请使用Chrome浏览器，火狐浏览器，IE11及以上，以及其他浏览器的极速模式
            </div>
        <![endif]-->
        <div id=wdc-app></div>
        <script>
            window.isActive = true;
            window.PASSPORT_HOST = "passport.wdyedu.com";
            window.PASSPORT_ACTIVE_HOST = location.host;
            var _hmt = _hmt || [];
            (function() {
                var hm = document.createElement("script");
                hm.src = "https://hm.baidu.com/hm.js?388a00be71a53c6d16027f30dff802e2";
                var s = document.getElementsByTagName("script")[0];
                s.parentNode.insertBefore(hm, s);
            }
            )();
        </script>
        <script src=//17dayup-passport-dev.bj.bcebos.com/passport.js></script>
        <script src=/lib/third-party/aliyun-upload-sdk/es6-promise.min.js></script>
        <script src=/lib/third-party/aliyun-upload-sdk/aliyun-oss-sdk-5.3.1.min.js></script>
        <script src=/lib/third-party/aliyun-upload-sdk/aliyun-upload-sdk-1.4.0.min.js></script>
        <script src=/js/chunk-vendors.24616d79.js></script>
        <script src=/js/app.027f7de3.js></script>
    </body>
</html>
```

关于vue seo，技术们做过的尝试。

三种解决方案

- 页面预渲染： prerender-spa-plugin
- 服务端渲染： nuxt
- 路由采用h5 history模式

服务器端渲染 *vs* 预渲染

- 预渲染： 无需使用 web 服务器实时动态编译 HTML，而是使用预渲染方式，在构建时(build time)简单地生成针对特定路由的静态 HTML 文件。优点是设置预渲染更简单，并可以将你的前端作为一个完全静态的站点。

vue-spa-prerender

pu

该插件怎么实现功能的

- 3.X版本基于 puppeteer;
- 2.X版本基于PhantomJS;

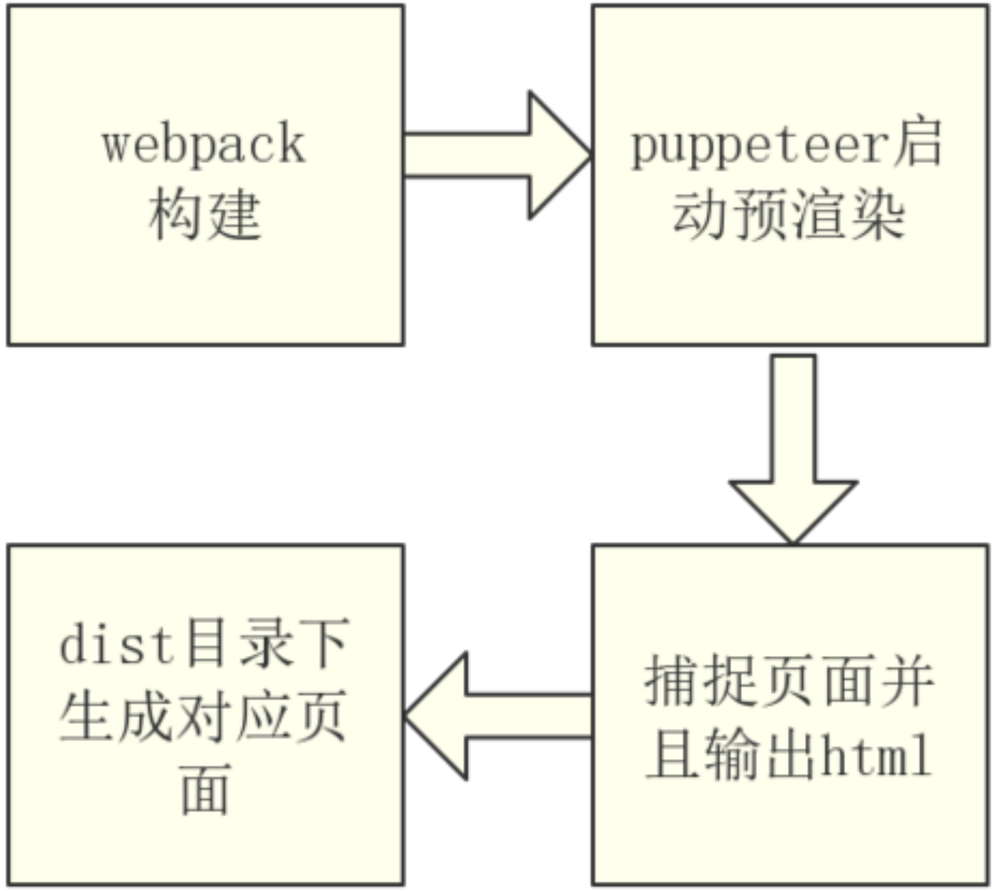
Puppeteer

Puppeteer 是一个node库，他提供了一组用来操纵Chrome的API, 通俗来说就是一个 headless chrome浏览器 (当然你也可以配置成有UI的，默认是没有的)。既然是浏览器，那么我们手工可以在浏览器上做的事情 Puppeteer 都能胜任, 另外，Puppeteer 翻译成中文是”木偶”意思

生成网页截图或者 PDF；高级爬虫，可以爬取大量异步渲染内容的网页；模拟键盘输入、表单自动提交、登录网页等，实现 UI 自动化测试;捕获站点的时间线，以便追踪你的网站，帮助分析网站性能问题

关于prerender-spa-plugin

流程图：



关于prerender-spa-plugin/快速开始

快速开始：

- 步骤一：npm install prerender-spa-plugin --save

  安装的过程有点长，100多M的下载。

  如遇 Chromium revision is not downloaded，试试 npm rebuild puppeteer

  注意，保证你本地搭好了梯子。

- 步骤二：修改vue 项目的配置。修改vue.config.js与main.js

  代码如下图所示：

关于prerender-spa-plugin/快速开始

修改vue.config.js

```
plugins: [
    new PrerenderSPAPlugin({
        // 生成文件的路径，也可以与webpakc打包的一致。
        // 下面这句话非常重要！！！
        // 这个目录只能有一级，如果目录层次大于一级，在生成的时候不会有任何错误提示，在预渲染的时候只会卡着不动。
        staticDir: path.join(__dirname, 'dist'),
        // 对应自己的路由文件，比如a有参数，就需要写成 /a/param1。
        routes: [
            '/',
            '/guide',
            '/works/talent',
        ],
        // 这个很重要，如果没有配置这段，也不会进行预编译
        renderer: new Renderer({
            inject: { // window.__PRERENDER_INJECTED 判断是否是预渲染
                foo: 'bar',
            },
            headless: true,
            renderAfterElementExists: '#wdc-app',
            // 在 main.js 中 document.dispatchEvent(new Event('render-event'))，两者的事件名称要对应上。
            renderAfterDocumentEvent: 'render-event',
        }),
    }),
],
```

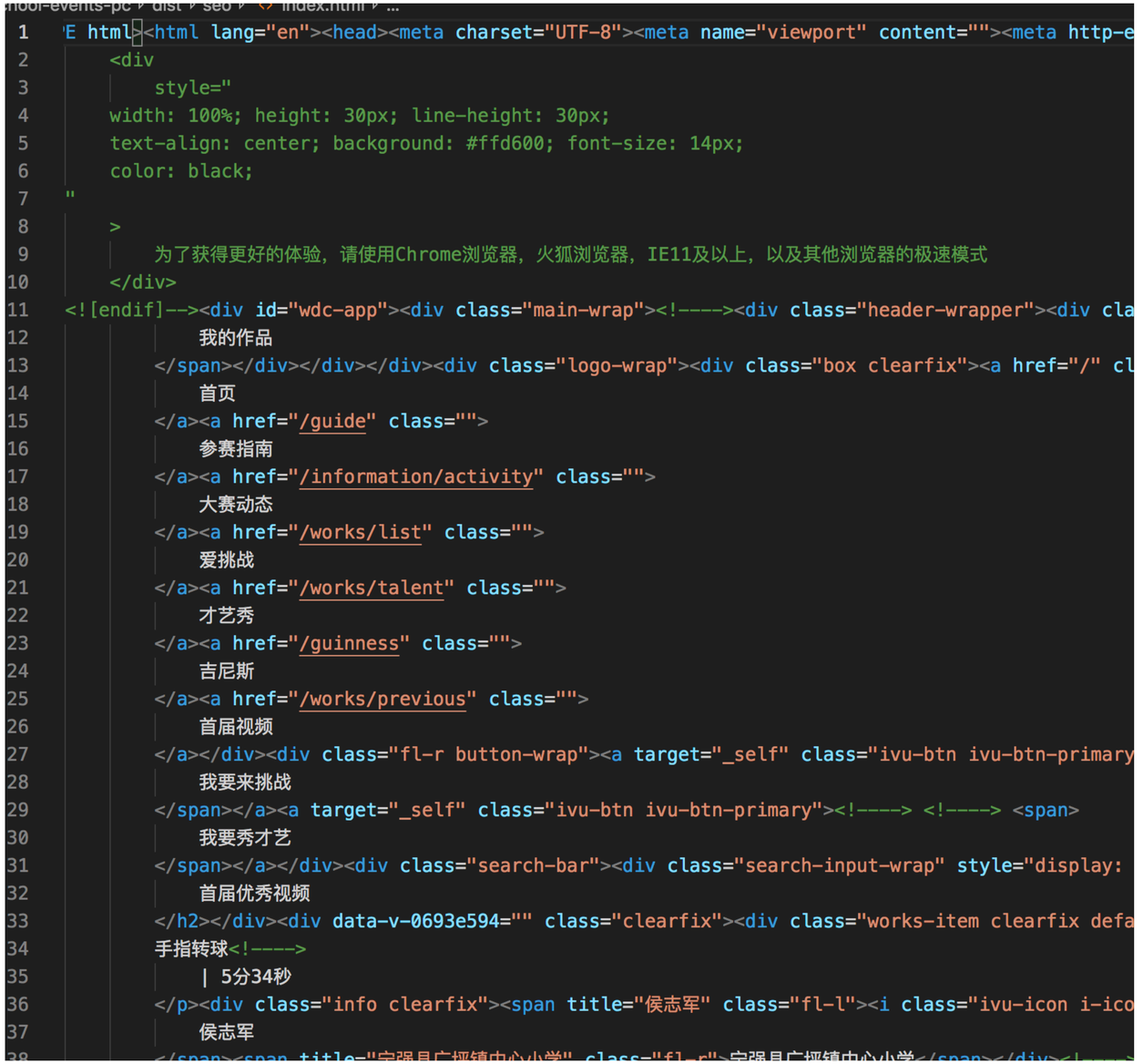
prerender-spa-plugin实践

修改main.js

```
new Vue({
    router,
    store,
    i18n,
    mounted() {
        document.dispatchEvent(new Event('render-event'));
    },
    render: h => h(App),
}).$mount('#wdc-app');
```

prerender-spa-plugin实践

- 步骤三：上述修改完，执行npm run build, 构建完成后，查看dist目录下的文件，新增加了对应路由文件。打开dist/index.html，查看到内容，页面内容基本都出现了。



prerender-spa-plugin报错经历

1.[prerender-spa-plugin] Unable to prerender all routes

In the interest of transparency, there are some use-cases where prerendering might not be a great idea.

- **Tons of routes** - If your site has hundreds or thousands of routes, prerendering will be really slow. Sure you only have to do it once per update, but it could take ages. Most people don't end up with thousands of static routes, but just in-case...
- **Dynamic Content** - If your render routes that have content that's specific to the user viewing it or other dynamic sources, you should make sure you have placeholder components that can display until the dynamic content loads on the client-side. Otherwise it might be a tad weird.

- 官方解释，这就是个缺陷，所以，可以选择的替代方式是，
  1）选择性的生成html页面
  2) 路由params改成query的写法。

prerender-spa-plugin报错经历

2.页面刷新后，闪现首页的骨架图

- 1)配置输出地址，outputDir，不在直接覆盖index.html
- 2.)配置 nginx,修改页面的配置，页面刷新不在闪首页。

3.页面接口请求失败的弹窗，也会打包进去。

window.__PRERENDER_INJECTED用来判断是否为预渲染

prerender-spa-plugin报错经历

4，生成的html页面内部，接口地址错误

- 1)手动配置server 中的proxy，在重新进行构建。

5，页面请求的内容没有生成html

应该等待接口内容响应结束，并且DOM已经挂载成功，再派发预渲染事件。document.dispatch(new Evenr(‘render-enent’))

贴一下修改后的vue.config.js

```
new PrerenderSPAPlugin({
    // 生成文件的路径，也可以与webpakc打包的一致。
    // 下面这句话非常重要！！！
    // 这个目录只能有一级，如果目录层次大于一级，在生成的时候不会有任何错误提示，在预渲染的时候只会卡着不动。
    staticDir: path.join(__dirname, 'dist'),
    outputDir: path.join(__dirname, 'dist/seo'),
    indexPath: path.join(__dirname, 'dist', 'index.html'),
    // 对应自己的路由文件，比如a有参数，就需要写成 /a/param1.
    routes: [
        '/',
        '/guide',
        '/works/talent',
        '/works/list',
        '/works/previous',
        '/guinness',
        '/sample',
        '/query',
    ],
    server: {
        proxy: {
            '/api': {
                target: 'http://aitiaozhan.dev.wdyclass.com:8000/',
                changeOrigin: true,
                pathRewrite: {
                    '^/api': '/api',
                },
            },
        },
    },
    // 这个很重要，如果没有配置这段，也不会进行预编译
    renderer: new Renderer({
        inject: {
            foo: 'bar',
        },
        headless: false,
        renderAfterElementExists: '#wdc-app',
        // 在 main.js 中 document.dispatchEvent(new Event('render-event'))，两者的事件名称要对应上。
        renderAfterDocumentEvent: 'render-event',
    }),
}),
```

调整下nginx的location

```
location / {
    root /Users/lilyyanglili/Documents/jishuzhongxin/school-events-pc/dist;
    try_files $uri /seo$uri/index.html $uri/  /index.html;
}
location = / {
    root /Users/lilyyanglili/Documents/jishuzhongxin/school-events-pc/dist;
    try_files /seo/index.html $uri/;
}
```

prerender-spa-plugin的总结

小结

- prerender-spa-plugin 代码侵入性低，可以根据路由渲染出对应的html文件，的确在一定程度上解决了我们对于 SEO 的诉求和页面加载慢的问题。但是它的缺点还是很明显的。
  - 不同的用户看到不同的页面，动态数据页面。
  - 经常发生变化的页面，数据实时性展示
  - 路由过多，构建时间过长
  - 新需求上线，数据请求没有结果。

vue-meta-info

描述：

`vue-meta-info` 是一个基于vue 2.0的插件，它会让你更好的管理你的 app 里面的 meta 信息。你可以直接 在组件内设置 metaInfo 便可以自动挂载到你的页面中，如果数据的有变化，自动更新你的title、meta等信息。这款插件，简单方便使用。

使用方式：

```
import Vue from 'vue'
import MetaInfo from 'vue-meta-info'


Vue.use(MetaInfo)
```

组件内静态使用与异步加载使用

```
  ...
</template>

<script>
  export default {
    metaInfo: {
      title: 'My Example App', // set a title
      meta: [{                 // set meta
        name: 'keyWords',
        content: 'My Example App'
      }]
      link: [{                 // set link
        rel: 'asstes',
        href: 'https://assets-cdn.github.com/'
      }]
    }
  }
```

```
<script>
  export default {
    name: 'async',
    metaInfo () {
      return {
        title: this.pageName
      }
    },
    data () {
      return {
        pageName: 'loading'
      }
    },
    mounted () {
      setTimeout(() => {
        this.pageName = 'async'
      }, 2000)
    }
  }
</script>
```

页面效果

如图上显示，青少年爱挑战的标题已经变化，设置了页面都会跟着切换页面显示不同的标题。

自研方案

整体思路

1.通过puppeteer爬取线上加载完毕的HTML

2.配置Nginx，通过路由优先查找对应HTML文件

3.定时执行爬虫，将爬取的HTML合入dist目录，自动化上线(待实现。)

启动puppeteer

```
const puppeteer = require("puppeteer");

async function getPic() {
    const browser = await puppeteer.launch();
    const page = await browser.newPage();
    await page.goto("https://google.com");
    await page.screenshot({ path: "google.png" });

    await browser.close();
}

getPic();
```

修改nginx配置

```
location / {
    root /Users/lilyyanglili/Documents/jishuzhongxin/school-events-pc/dist;
    try_files $uri /seo$uri/index.html $uri/  /index.html;
}
location = / {
    root /Users/lilyyanglili/Documents/jishuzhongxin/school-events-pc/dist;
    try_files /seo/index.html $uri/;
}
```

我们把抓取的html放到seo目录下，用户访问时，没有抓取的页面直接访问根目录的html，对应的html则会抓取对应的页面。

自研方案小结：

搜索引擎的爬虫得到更为完整的页面；对业务代码0侵入；页面在跳转路由等工作机制没有被改变，因为页面的引入的vue的脚本，会把挂载的根结点元素的内容替换为编译后的内容，这保证了页面内容的真实性。当然这一替换过程也带来了暂短的视觉弹跳。

另外，作为开发者我们写完脚本，后续的一些风险性问题我们依然要关注：

通过puppeteer抓取的页面内容，并不需要页面渲染的时候就出现，比如加载进度条，消息提示框，报错信息等等，这就需要开发者在爬取了页面进行仔细的检查，并删除。

如果网页内容更迭比较快，使用定时抓取并修改对应的html，在保证内容的效果稳定性，依然还需要谨慎处理。

当爬虫判断页面是否加载完成的节点被删除时，会导致爬取失败，

路由变化就需要修改脚本。

参考资源

预渲染的git 地址： https://github.com/chrisvfritz/prerender-spa-plugin

vue的seo问题；链接地址： https://juejin.im/post/5c710626f265da2d9b5e1a94

Vue.js服务端渲染指南；链接地址：https://ssr.vuejs.org/zh/

浅谈SPA ，SEO，SSR；链接地址： https://juejin.im/entry/5bbbf852f265da0aea699497

puppeteer入门实践，链接地址： https://www.jb51.net/article/139808.htm

puppeteer文档，链接地址： https://pptr.dev/

Thank you for listening