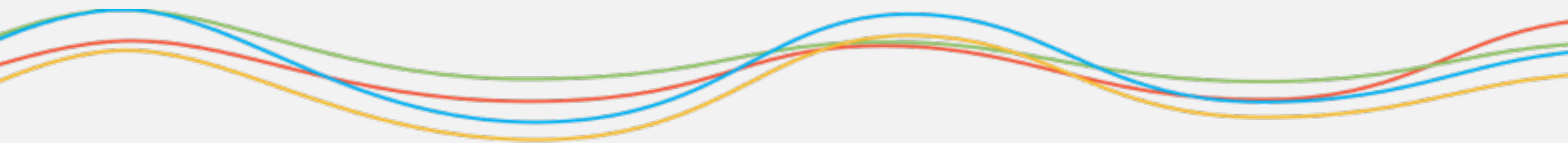


基于Vue的单页面性能优化实践

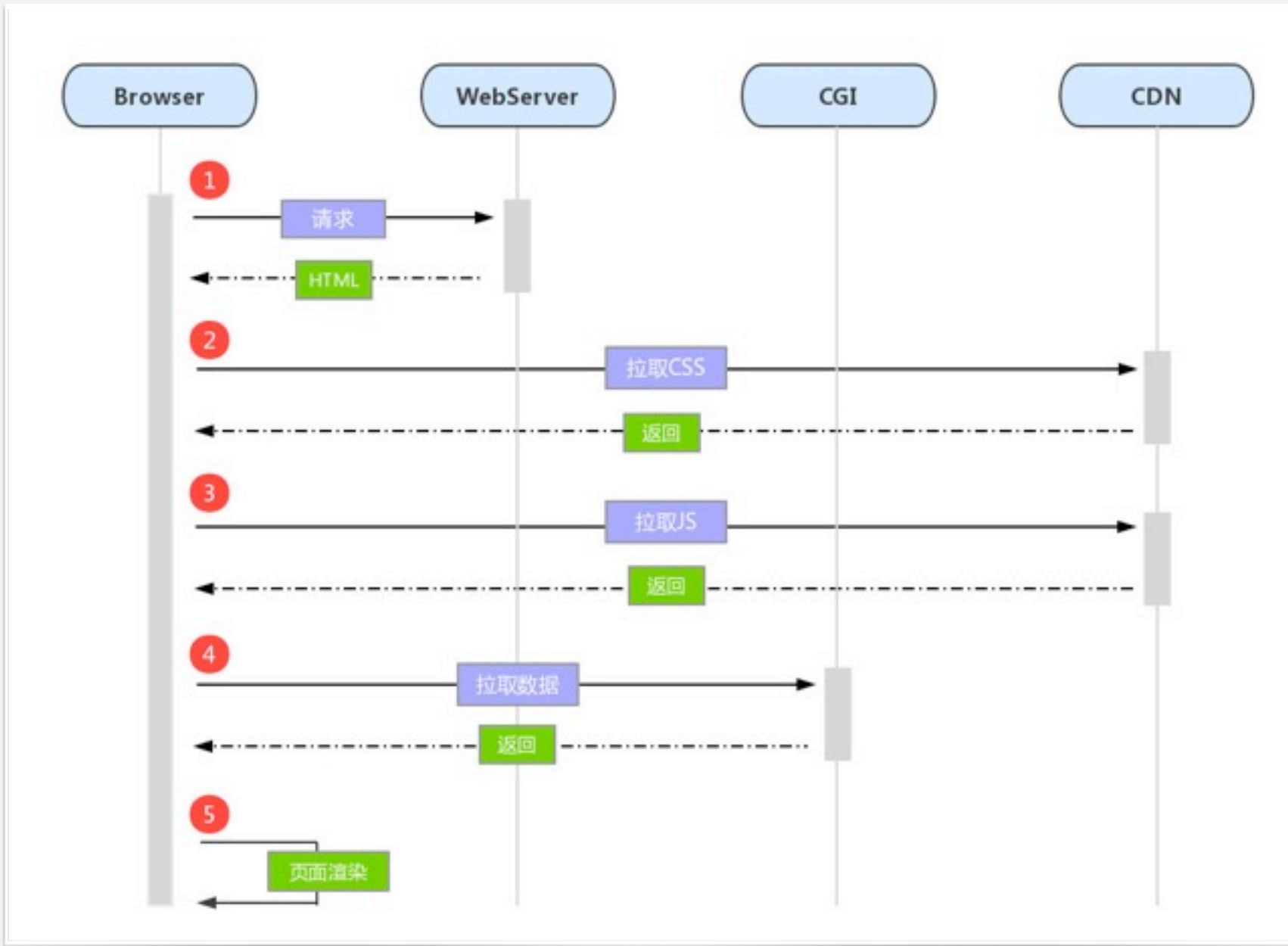
汽车之家 车服务 - 前端团队 王诗扬



王诗扬

- 2013年加入汽车之家，现任车服务 前端负责人
- 致力于公司内外Node和vue的推广和布道
- 乐于分享，喜欢研究新技术

- 用户感受
- 一秒法则

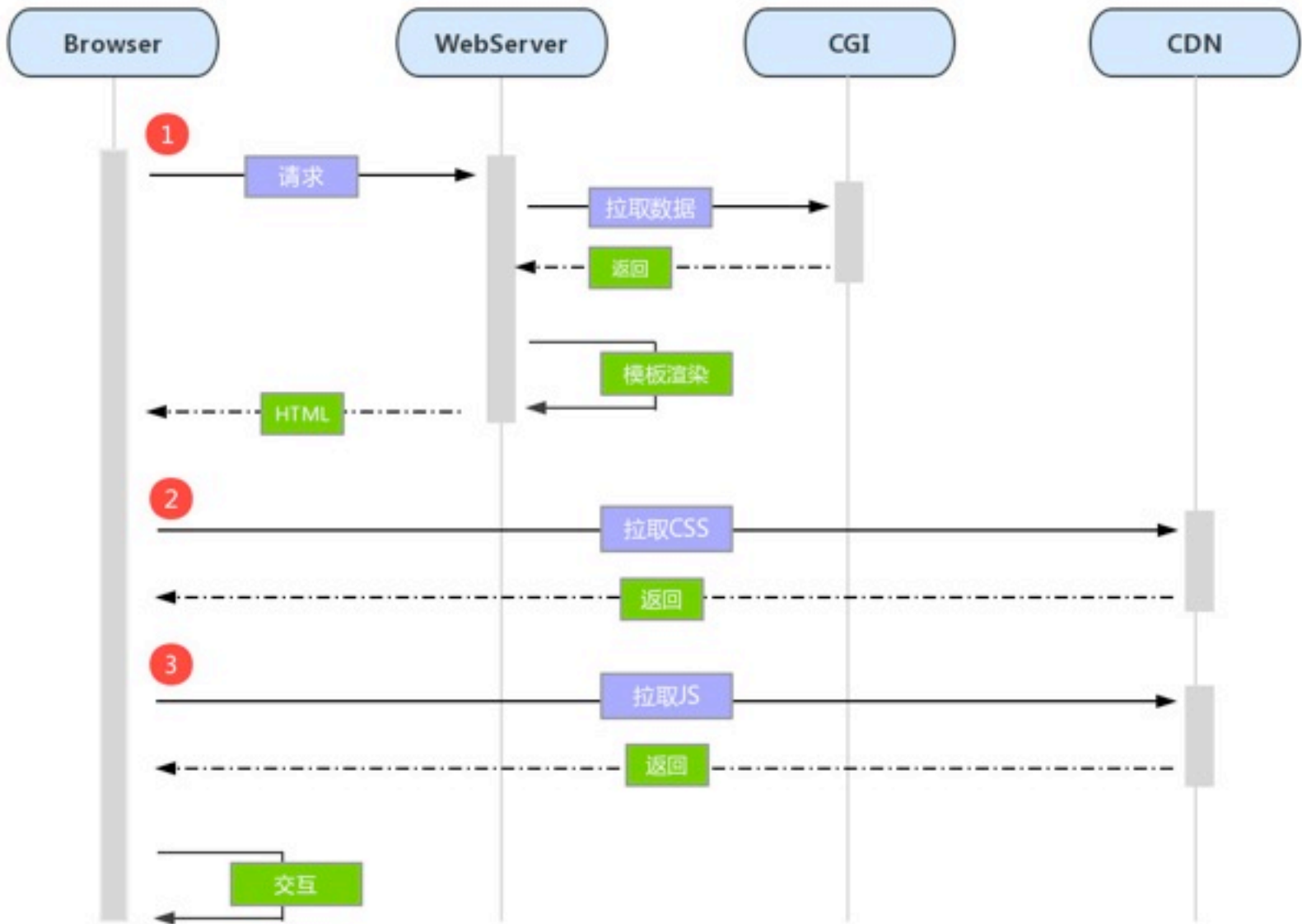


优点：

- 减轻服务端压力
- 共用一套后端程序，适配多端

缺点：

- 首屏加载较慢
- SEO

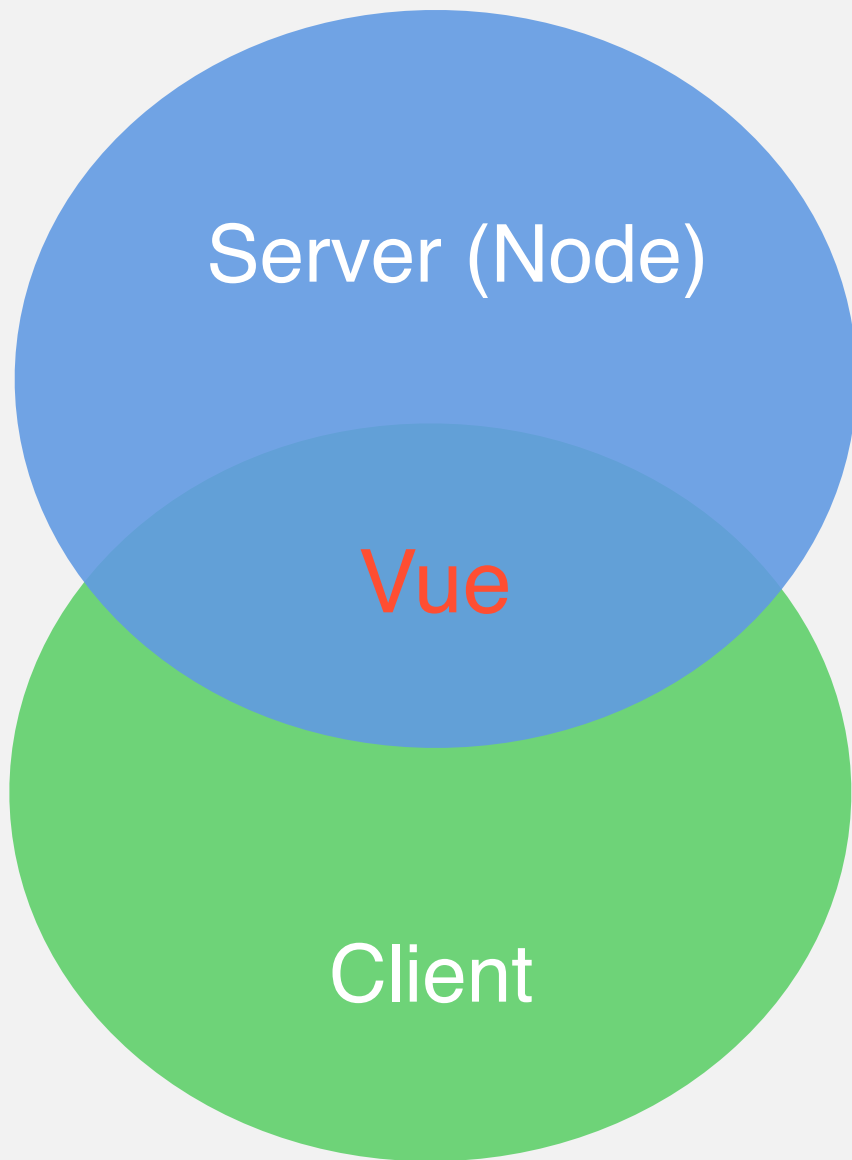


优点：

- 加快首屏渲染时间
- 解决SEO问题

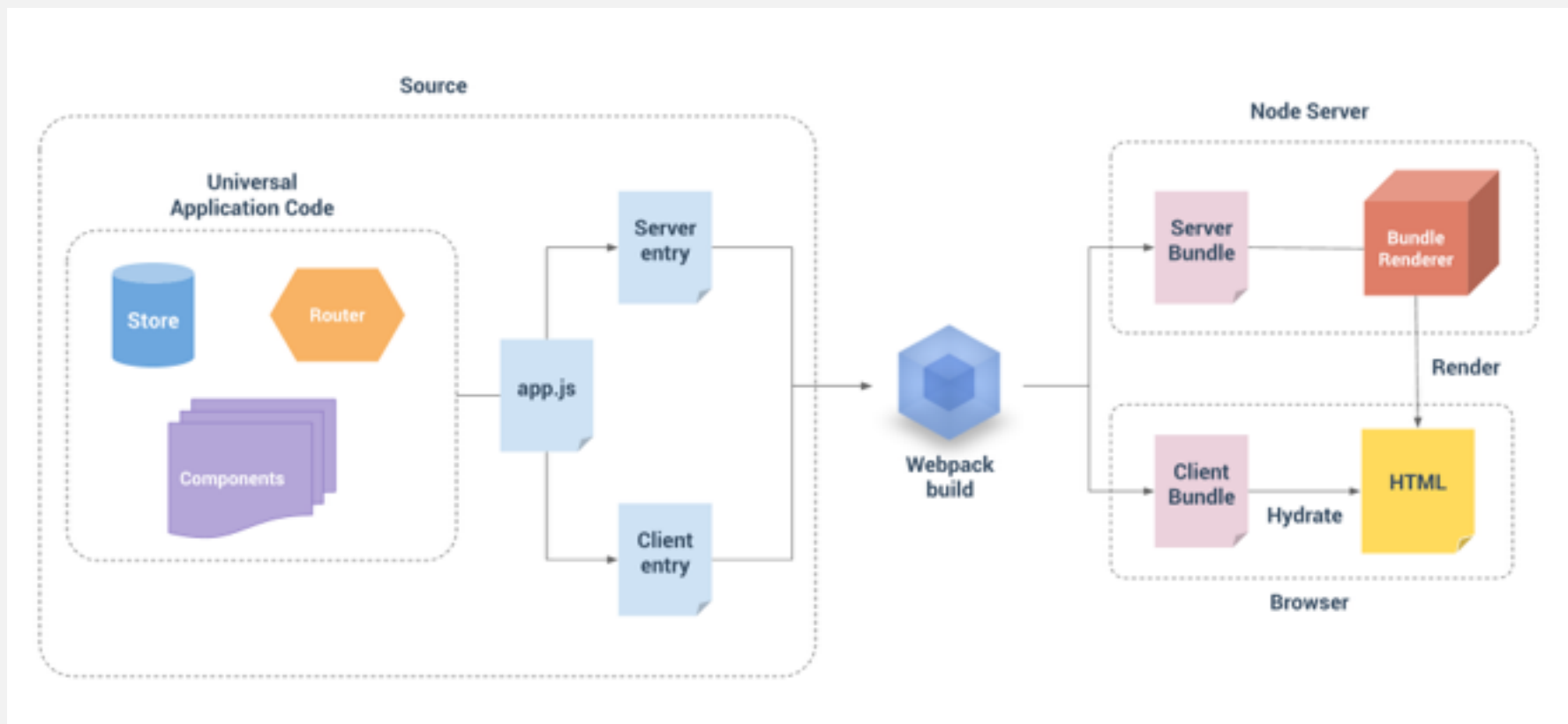
缺点：

- 前后端耦合性较高
- 对服务端压力较大

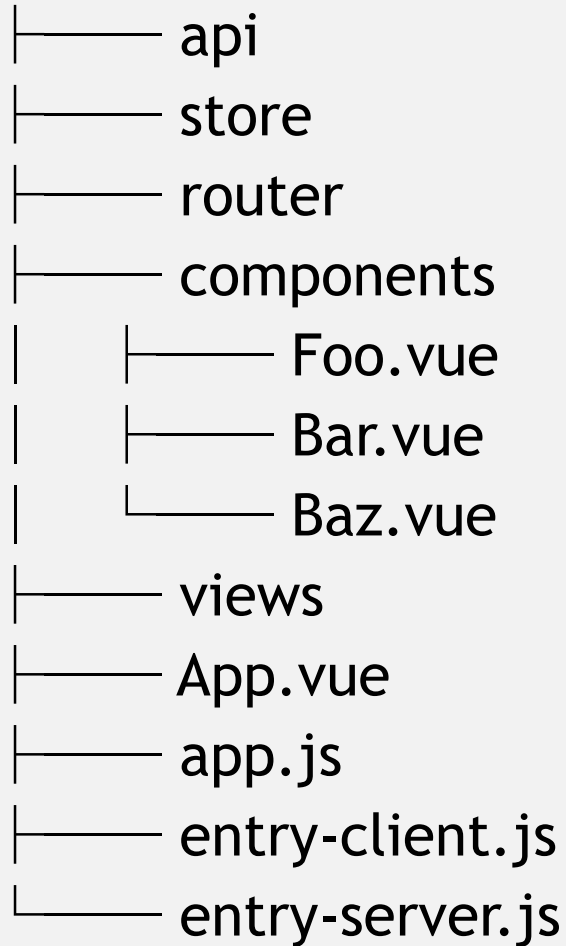


- 前后端复用一套代码
- 良好的用户体验
- 加快首屏的渲染
- 解决SEO问题

Vue SSR 流程图



src



通用入口文件

仅运行于浏览器

仅运行于服务器

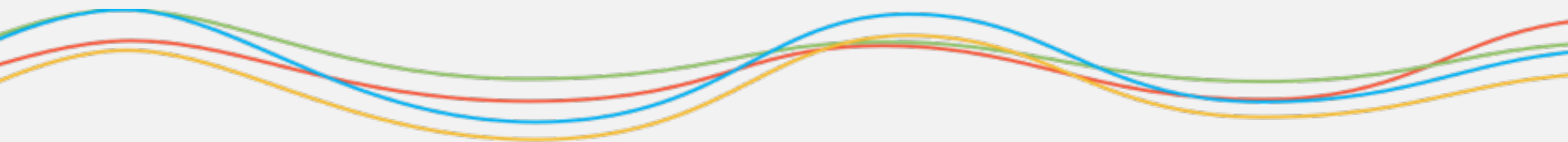
- 跟传统的模板引擎存在性能差异
- 首屏加载文件过大

	总请求数	线程数	循环次数	平均响应时长
Node	50000	500	100	747ms
Vue SSR	50000	500	100	3172ms

- string-based 基于字符串的拼接
- virtual-dom-based 基于虚拟DOM对象

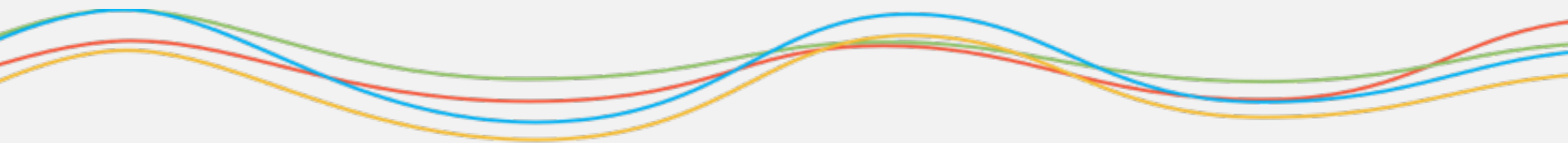
相差 4.24 倍

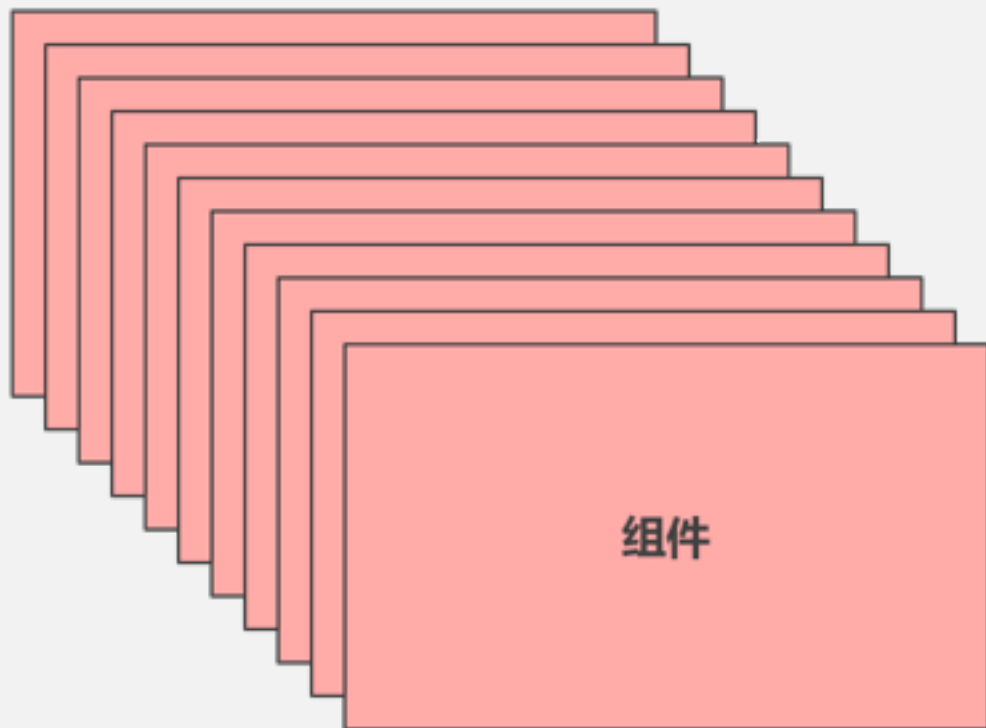
优化策略



- 滚屏加载
- 延迟加载
- 请求数优化
- 图片压缩、base64
- Iconfont
- CDN加速
- Gzip压缩

资源加载





随着组件的增加，JS文件越来越大

只加载页面所需的代码。提升页面的渲染速度

- 异步组件
- 代码分割

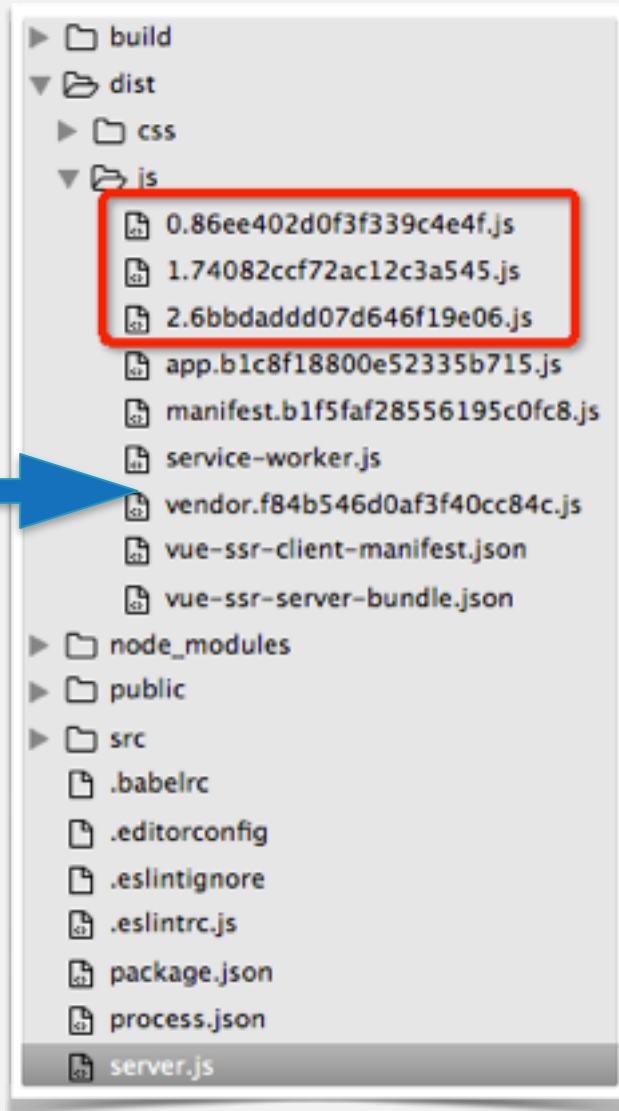
按需分块加载

```
// router.js
import Vue from 'vue';
import VueRouter from 'vue-router';

Vue.use(VueRouter);

const IndexView = () => import('../views/index');
const PostView = () => import('../views/post');
const TagView = () => import('../views/tag');

export function createRouter () {
  return new VueRouter({
    routes: [
      {
        path: '/',
        name: 'index',
        component: IndexView
      },
      {
        path: '/article/:pathName',
        name: 'article',
        component: PostView
      },
      {
        path: '/tags',
        name: 'tags',
        component: TagView
      }
    ]
  });
};
```



```
<head>
  <title>车服务前端团队</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width,initial-scale=1,minimum-scale=1,maximum-scale=1,user-scalable=no">
  <link rel="preload" href="/dist/manifest.430d7683ad2487f69d53.js" as="script">
  <link rel="preload" href="/dist/vendor.f84b546d0af3f40cc84c.js" as="script">
  <link rel="preload" href="/dist/app.b1c8f18800e52335b715.js" as="script">
  <link rel="preload" href="/dist/common.b1c8f18800e52335b715.css" as="style">
  <link rel="preload" href="/dist/2.6bbdadd07d646f19e06.js" as="script">
  <link rel="prefetch" href="/dist/5.3f721442a28346e98c2a.js" as="script">
  <link rel="prefetch" href="/dist/0.86ee402d0f3f339c4e4f.js" as="script">
  <link rel="prefetch" href="/dist/3.bf1ffeb3b767631028df.js" as="script">
  <link rel="prefetch" href="/dist/4.de9c6655bb7636f288df.js" as="script">
  <link rel="prefetch" href="/dist/1.74082ccf72ac12c3a545.js" as="script">
  <link rel="prefetch" href="/dist/6.a32dd3eb73a7818b8d46.js" as="script">
  <link rel="stylesheet" href="/dist/common.b1c8f18800e52335b715.css">
</head>
```

`<link rel="" href="1.js" as="script">`

preload

预加载当前页所需的资源

prefetch

预加载下一页所需的资源

```
<html>
  <head>
    <!-- 用于当前渲染的 chunk 会被资源预加载(preload) -->
    <link rel="preload" href="/manifest.js" as="script">
    <link rel="preload" href="/main.js" as="script">
    <link rel="preload" href="/0.js" as="script">
    <!-- 未用到的异步 chunk 会被数据预取(preload) (次要优先级) -->
    <link rel="prefetch" href="/1.js" as="script">
  </head>
  <body>
    <!-- 应用程序内容 -->
    <div data-server-rendered="true"><div>async</div></div>
    <!-- manifest chunk 优先 -->
    <script src="/manifest.js"></script>
    <!-- 在主 chunk 之前注入异步 chunk -->
    <script src="/0.js"></script>
    <script src="/main.js"></script>
  </body>
</html>
```

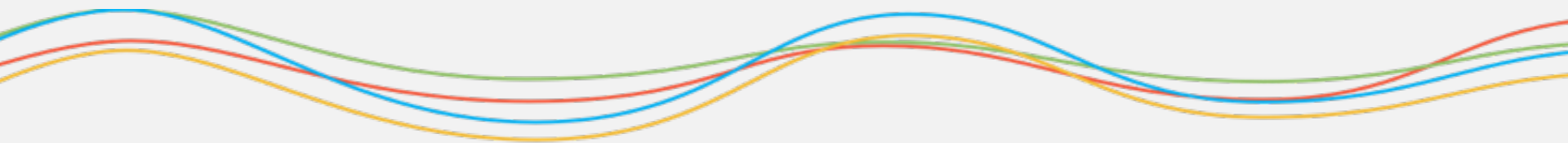
优点：

- 简单好用
- 能有效提升性能

潜藏问题：

- 不能确保加载完毕
- 不能确保缓存的持久

缓存



- 在serviceWorker 进程中运行
- 缓存必要的前端资源和请求
- 大幅度减少CDN的压力
- 资源是持久性存储

```
npm install --save-dev sw-precache-webpack-plugin
```

```
new SWPrecachePlugin({
  cacheId: 'yc',
  filename: 'service-worker.js',
  minify: true,
  mergeStaticsConfig: true,
  staticFileGlobs: [
    path.join(__dirname, '../dist/static/*.*)
  ],
  stripPrefixMulti: {
    [path.join(__dirname, '../dist/static')]: '/static'
  },
  // add hash to path
  dontCacheBustUrlsMatching: false,
  staticFileGlobsIgnorePatterns: [
    /index\.html$/,
    /\.map$/,
    /\.css$/,
    /\.svg$/,
    /\.eot$/
  ]
})
```


serviceWorker 的使用

```
// server.js  
app.use('/service-worker.js', serve('./dist/service-worker.js'));
```

```
// client-entry.js  
if (window.location.protocol === 'https:' && navigator.serviceWorker)  
  navigator.serviceWorker.register('/service-worker.js');  
}
```

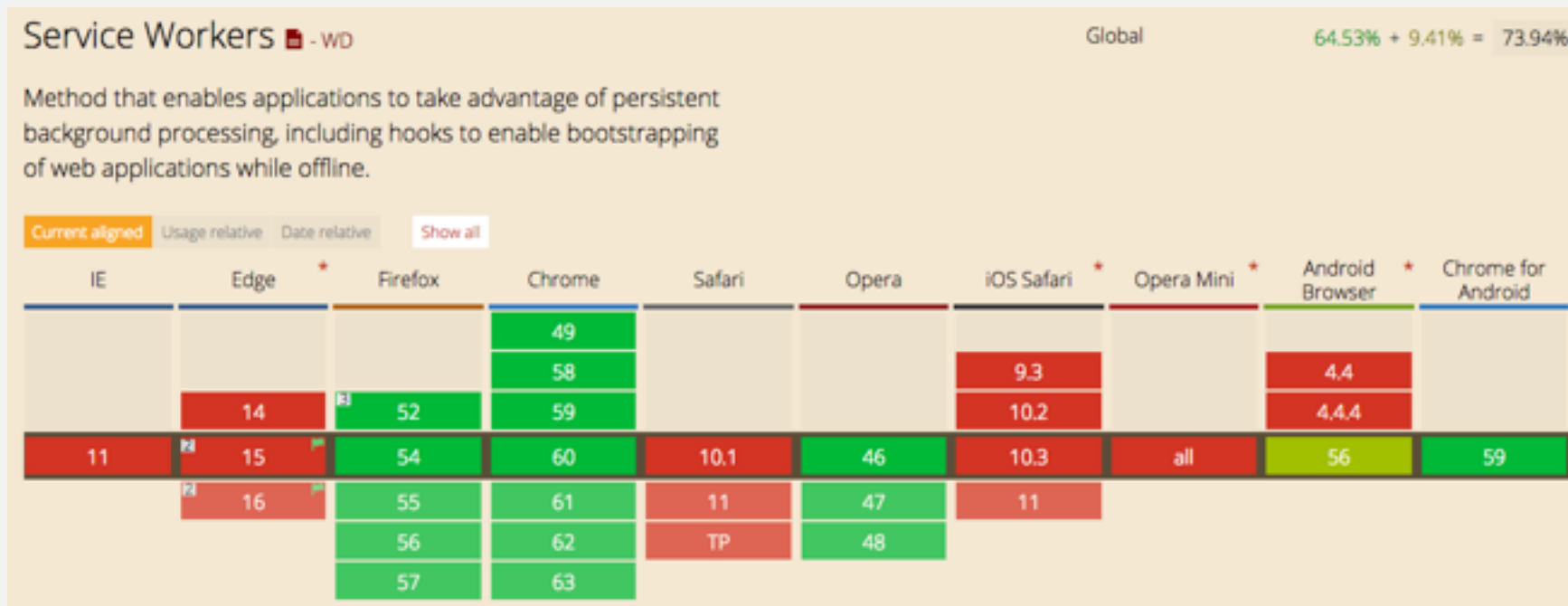
serviceWorker 的存储空间

manifest.430c7683ad2487f69d53.js	200	script	autocfe.cn:7	(from ServiceWorker)	4 ms
vendor.f94b548d0af3f40cc84c.js	200	script	autocfe.cn:7	(from ServiceWorker)	6 ms
app.b1c8f18800e52335b715.js	200	script	autocfe.cn:7	(from ServiceWorker)	5 ms
common.b1c8f18800e52335b715.css	200	stylesheet	autocfe.cn:7	(from ServiceWorker)	5 ms
2.6bbdadd07c646f19e06.js	200	script	autocfe.cn:7	(from ServiceWorker)	6 ms
5.3f721442a28346e98c2a.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	3 ms
0.86ee402d0f3f339c4e4f.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	4 ms
3.bf1f5eb3b767631026df.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	5 ms
4.de9c6655bb7636f288df.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	6 ms
1.74082ccf72ac12c3a545.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	7 ms
data:image/png;base...	200	png	Other	(from memory cache)	0 ms
data:image/png;base...	200	png	Other	(from memory cache)	0 ms
data:image/peg.bas...	200	jpeg	Other	(from memory cache)	0 ms
6.a32dd3eb73a7818b8d46.js	200	javascript	autocfe.cn:7	(from ServiceWorker)	4 ms
font_qd5eg5m9yvgkqpvj.woff?1=1496328533253	200	font	autocfe.cn:71	5.4 KB	748 ms
@service-worker.js	200	javascript	service-worker.js	0 B	293 ms

和indexDB、LocalStorage共用，存储空间约有 **50M**

- 仅缓存必要的js、css、iconfont
- 对不需要即时更新的接口数据进行缓存

serviceWorker 的兼容性



- android 53
- chrome、firefox
- uc、qq、百度

未来随着所有浏览器对serviceWorker的支持

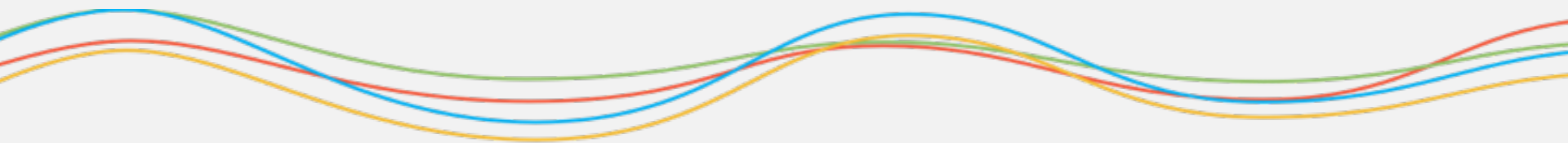


```
const axios = require('axios');
const md5 = require('md5');
const cached = require('lru-cache')({
  max: 1000, // 最大数量
  maxAge: 1000 * 60 * 15 // 缓存15分钟
});
export default {
  get: function(target, params) {
    const key = md5(target + JSON.stringify(params));
    if (cached && cached.has(key)) { // 命中缓存
      return Promise.resolve(cached.get(key)); // 返回缓存
    }
    return new Promise((resolve, reject) => {
      axios({
        url: target,
        method: 'get',
        params,
        timeout: 10000
      }).then(res => {
        if (cached && params.cache) {
          cached.set(key, res.data); // 保存缓存数据
        }
        resolve(res.data);
      }).catch((error) => {
        reject(error);
      });
    });
  }
};
```

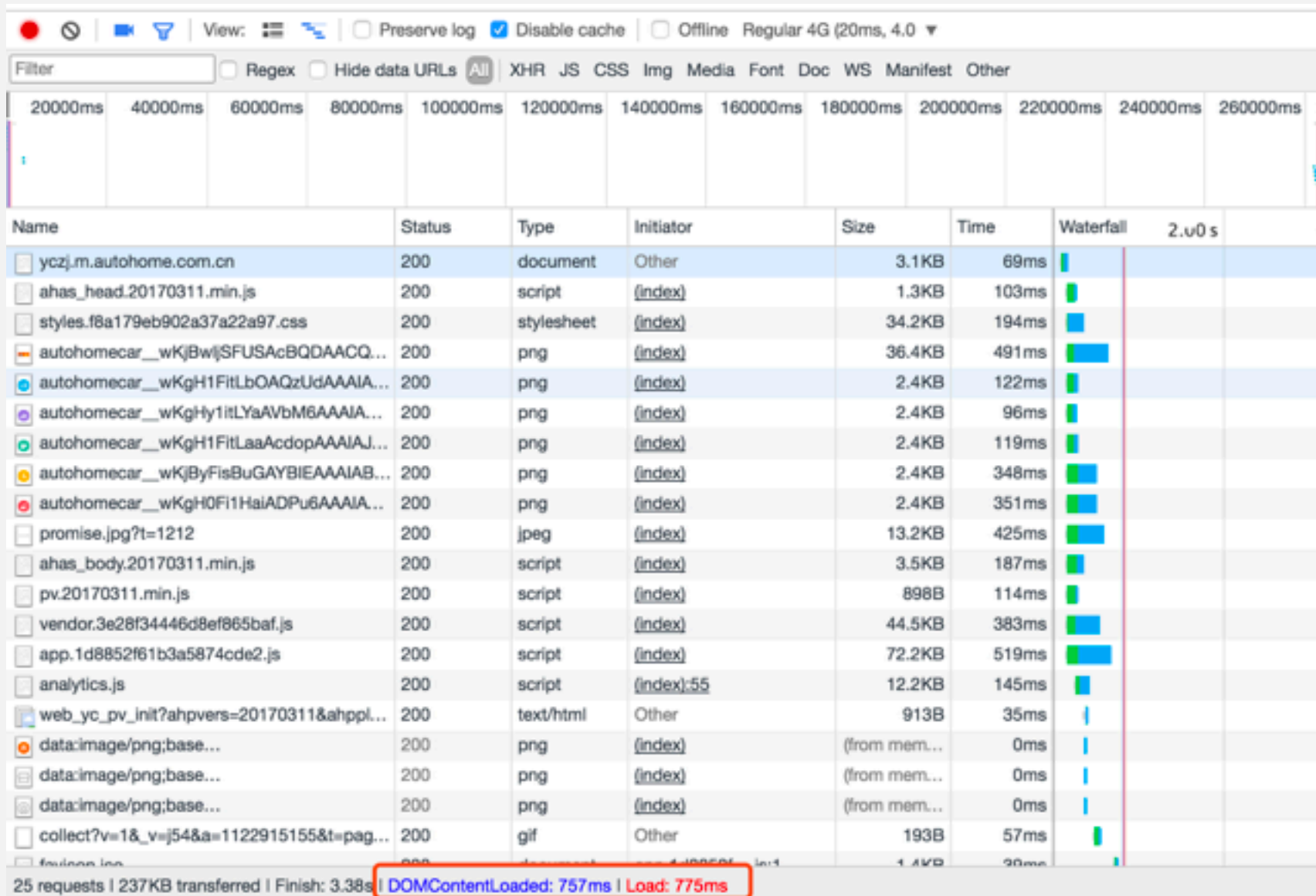
```
return require('vue-server-renderer').createBundleRenderer(bundle, {
  cache: require('lru-cache')({
    max: 1000, // 缓存最大数量
    maxAge: 1000 * 60 * 15 // 15分钟缓存
  })
});
```

```
export default {
  name: 'server-list',
  props: {
    article: {
      type: Object,
      required: true
    },
    supportWebp: Boolean
  },
  serverCacheKey: props => {
    return `${props.article.pathName}::${props.article.updatedAt}`
  }
}
```

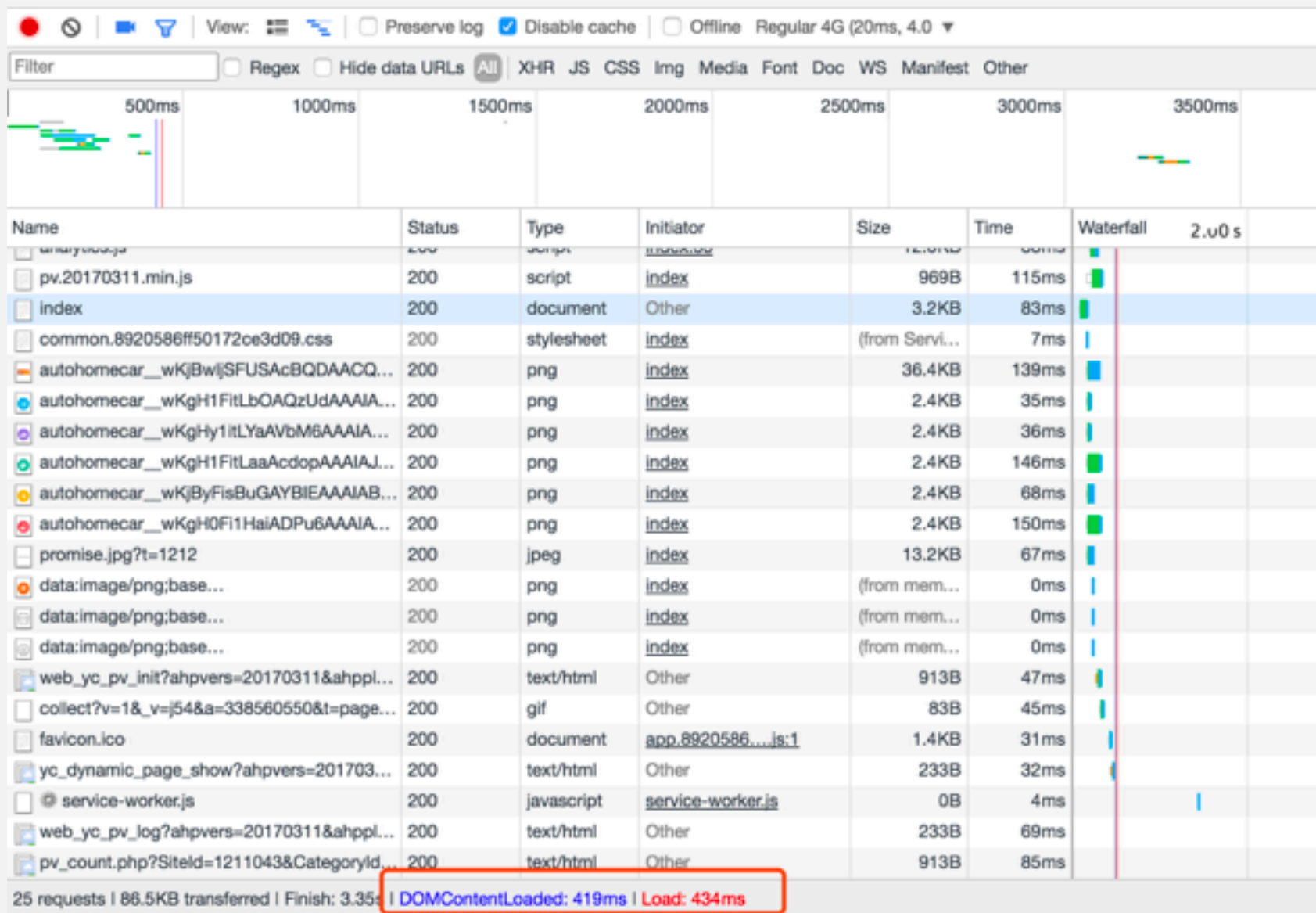
优化结果



首页优化前



首页优化后



- 渲染方案的对比
- 优化策略
- 优化结果

THANK YOU

