# 一、你前端是怎么学的？

面试官问怎么学前端的，这种问题考察什么？总结四个字：学习能力

面试官想知道以下内容：

1.1 学习能力

12 总结能力

1.3 动手能力(这个也是面试官最想知道的，毕竟学以致用)

可以这样回答：
一开始学习前端，主要靠视频以及书籍，主要是总结很多知识点，但是发现学习过程中理论大于实操，所以自己把知识点总结后就开始找项目实操，这样一步一步成长起来的，那么在公司中也会碰到很多问题值得学习，一般都是自己查找原因或者和同事交流进行解决，然后自己总结出文档，方便日后查看和积攒经验。所以我感觉我的学习都是实操加总结。

# 二、移动端兼容问题

面试官问到：在项目有有哪些移动端兼容问题
可以这样回答：哦～兼容问题有很多，可能我一时半会也想不到那么多，我就说一下，我能想起来的吧。【然后就开始说下面的】。

**一、在 ios 键盘中首字母大写的问题?**

```
<input type="text" autocapitalize='off'>
```

**二、html5 标签在低版本浏览器如何兼容？**

使用：html5shiv.js
代码：body 之后,布局之前加入

```
<!-- [if lt ie 9]>
<script src="shiv.js"></script>
<![endif]-->
```

**三、h5 底部输入框被键盘遮挡问题**

h5 页面有个很蛋疼的问题就是，当输入框在最底部，点击软键盘后输入框会被遮挡。可采用如下方式解决

var oHeight = $(document).height(); // 浏览器当前的高度 $(window).resize(function(){ if($(document).height() < oHeight){ $("#footer").css("position","static"); }else{ $("#footer").css("position","absolute"); } });

**四、在移动端使用 click 事件有 300ms 延迟的问题**

解决方式：

1. 禁止双击缩放===》meta:user-scalabel=no

2. fastclick.js

```
<script type="text/javascript" src='fastclick.js'></script>

if ('addEventListener' in document) {
    document.addEventListener('DOMContentLoaded', function() {
        FastClick.attach(document.body);
    }, false);
}
```

**五、移动端 touch 事件有穿透（点透）的问题，怎么解决？**

解决方式：

1. 阻止默认行为

```
e.preventDefault();
```

2. fastclick.js

2.1 fastclick.js 文件
2.2 写入代码

```
if ('addEventListener' in document) {
    document.addEventListener('DOMContentLoaded', function() {
        FastClick.attach(document.body);
    }, false);
}
```

**六、懒加载问题**

如果懒加载插件在某个盒子中执行，但是这个盒子有滚动效果，那么就需要加入属性：

```
$("img.lazy").lazyload({
    container: $("section")       //加入 container 他的值是滚动的父盒子
```

});

**七、在移动端中，如果给元素设置一个 1px 的像素边框的话，那么在手机上看起来是会比一个像素粗的。**

解决方法:使用伪类元素模拟边框使用 transform 缩放。

```
@media screen and (-webkit-min-device-pixel-ratio: 2)
{
    .button:after
    {
        border-radius: calc(5px * 2);
        width: calc(100% * 2);
        height: calc(100% * 2);
        transform: scale(calc(1/2));
    }
}
@media screen and (-webkit-min-device-pixel-ratio: 3)
{
    .button:after
    {
        border-radius: calc(5px * 3);
        width: calc(100% * 3);
        height: calc(100% * 3);
        transform: scale(calc(1/3));
    }
}
@media screen and (-webkit-min-device-pixel-ratio: 4)
{
    .button:after
    {
        border-radius: calc(5px * 4);
        width: calc(100% * 4);
        height: calc(100% * 4);
        transform: scale(calc(1/4));
    }
}
```

**八、消除 transition 闪屏**

-webkit-transform-style: preserve-3d; /*设置内嵌的元素在 3D 空间如何呈现：保留 3D*/
-webkit-backface-visibility: hidden; /*(设置进行转换的元素的背面在面对用户时是否可见：隐藏)*/

**九、禁止 ios 和 android 用户选中文字**

-webkit-user-select:none

**十、在 ios 和 andriod 中,audio 元素和 video 元素在无法自动播放**

应对方案：触屏即播

```
$('html').one('touchstart',function(){
    audio.play()
})
```

**十一、android 下取消输入语音按钮**

input::-webkit-input-speech-button {display: none}

**十二、fixed 定位缺陷**

ios 下 fixed 元素容易定位出错，软键盘弹出时，影响 fixed 元素定位
android 下 fixed 表现要比 iOS 更好，软键盘弹出时，不会影响 fixed 元素定位
ios4 下不支持 position:fixed
解决方案： 可用 iScroll 插件解决这个问题

**十三、input 的 placeholder 会出现文本位置偏上的情况**

input 的 placeholder 会出现文本位置偏上的情况：PC 端设置 line-height 等于 height 能够对齐，而移动端仍然是偏上，解决是设置 line-height：normal

**十四、android 里 line-height 不居中**

把字号内外边距等设置为需求大小的 2 倍，使用 transform 进行缩放。（不适用）
把字号内外边距等设置为需求大小的 2 倍，使用 zoom 进行缩放，可以完美解决。
把 line-height 设置为 0，使用 padding 值把元素撑开，说是可以解决（不适用）。

**十五、安卓部分版本 input 的 placeholder 偏上**

```
input{
```

```
    line-height:normal;
}
```

**十六、ios 日期转换 NAN 问题**

具体就是，new Date('2020-11-12 00:00:00')在 ios 中会为 NAN

解决方案：用 new Date('2020/11/12 00:00:00')的日期格式，或者写个正则转换

**十七、禁止数字识别为电话号码**

```
<meta name = "format-detection" content = "telephone=no">
```

# 三、前端性能优化

### 3.1 Web 性能优化辅助工具

3.11 Lighthouse

详细的内容，可以去参考 git：https://github.com/GoogleChrome/lighthouse

3.1.2 测试网站

https://www.webpagetest.org/

### 3.2 具体优化内容有：

**1. 加载优化**

1. 压缩合并
2. 代码分割(code spliting)，可以基于路由或动态加载
3. 第三方模块放在 CDN
4. 大模块异步加载，例如: Echarts，可以使用 require.ensure，在加载成功后，在显示对应图表
5. 小模块适度合并，将一些零散的小模块合并一起加载，速度较快
6. 可以使用 pefetch 预加载，在分步场景中非常适合

**2. 图片优化**

1. 小图使用雪碧图，iconFont，base64 内联
2. 图片使用懒加载

3. webp 代替其他格式

4. 图片一定要压缩

5. 可以使用的 img 的 srcset，根据不同分辨率显示不同尺寸图片，这样既保证显示效果，又能节省带宽，提高加载速度

### 3. css 优化

1. css 写在头部

2. 避免 css 表达式

3. 移除空置的 css 规则

4. 避免行内 style 样式

### 4. js 优化

1. js 写在 body 底部

2. js 用 defer 放在头部，提前加载时间，又不阻塞 dom 解析

3. script 标签添加 crossorigin，方便错误收集

### 5. 渲染优化

1. 尽量减少 reflow 和 repaint

涉及到样式，尺寸，节点增减的操作，都会触发 reflow 和 repaint。

1.1 用变量缓存 dom 样式，不要频繁读取

1.2 通过 DocumentFragment 或 innerHTML 批量操作 dom

1.3 dom 隐藏，或复制到内存中，类似 virtual dom，进行修改，完成后再替换回去

1.4 动画元素一定要 absolute，脱离文档流，不影响其他元素。动画不要用 left，top 等操作，要使用 transform 和 opacity，同时开启渲染层(will-change 或 translate3d(0,0,0))

1.5 动画尽量用 requestAnimationFrame，不要用定时器

1.6 移动端硬件加速，触发 GPU 渲染，还是 translate3d(0,0,0)

### 6. 首屏优化

原则：显示快，滚动流畅，懒加载，懒执行，渐进展现

1. 代码分离，将首屏不需要的代码分离出去

2. 服务端渲染或预渲染，加载完 html 直接渲染，减少白屏时间

3. DNS prefetch，使用 dns-prefetch 减少 dns 查询时间，PC 端域名发散，移动端域名收敛

4. 减少关键路径 css，可以将关键的 css 内联，这样可以减少加载和渲染时间

### 7. 打包优化（主要是 webpack 优化）

1. 拆包 externals dllPlugin
2. 提取公共包 commonChunkPlugin 或 splitChunks
3. 缩小范围 各种 loader 配置 include 和 exclude，noParse 跳过文件
4. 开启缓存 各种 loader 开启 cache
5. 多线程加速 happypack 或 thead-loader
6. tree-shaking ES 模块分析，移除死代码
7. Scope Hoisting ES6 模块分析，将多个模块合并到一个函数里，减少内存占用，减小体积，提示运行速度

## 8. webpack 长缓存优化

1. js 文件使用 chunkhash，不使用 hash
2. css 文件使用 contenthash，不使用 chunkhash，不受 js 变化影响
3. 提取 vendor，公共库不受业务模块变化影响
4. 内联 webpack runtime 到页面，chunkId 变化不影响 vendor
5. 保证 module Id 稳定，不使用数字作为模块 id，改用文件内容的 hash 值，使用 HashedModuleIdsPlugin，模块的新增或删除，会导致其后面的所有模块 id 重新排序，为避免这个问题
6. 保证 chunkhash 稳定，使用 webpack-chunk-hash，替代 webpack 自己的 hash 算法。webpack 自己的 hash 算法，对于同一个文件，在不同开发环境下，会计算出不用的 hash 值，不能满足跨平台需求。

## 8. vue 优化

1. 路由懒加载组件
2. keep-alive 缓存组件，保持原显示状态
3. 列表项添加 key，保证唯一
4. 列表项绑定事件，使用事件代理(v-for)
5. v-if 和 v-for 不要用在一个标签上，它会在每个项上进行 v-if 判断

## 9. react 优化

1. 路由组件懒加载，使用 react-loadable
2. 类组件添加 shouldComponent 或 PureComponent
3. 函数组件添加 React.memo
4. 列表项添加 key，保证唯一
5. 函数组件使用 hook 优化，useMemo，useCallback

## 10. SEO 优化

1. 添加各种 meta 信息
2. 预渲染
3. 服务端渲染

# 四、你还有什么想问的吗？

### 4.1 避免坑：

第一个错误是说没有什么要问的，如果你什么都没有了解的话，对方会觉得你对工作没什么想法，认为自己能找到一个工作就不错了，说明你价值也低。

那应该问啥呢？关于公司的能在网上搜索到的信息不用问，说明你提前有了解，对自己人生负责，检索能力不错。

### 4.2 可以问：

入职的部门现状
人员配置、开发的项目
使用的技术栈
公司发展现状
试用期、福利待遇

# 五、前端部分算法

### 5.1 深度平铺数组

使用递归。 通过空数组([]) 使用 Array.concat() ，结合 展开运算符( ... ) 来平铺数组。 递归平铺每个数组元素。

```
const deepFlatten = arr => [].concat(...arr.map(v => (Array.isArray(v) ? deepFlatten(v) : v)));

console.log(    deepFlatten([1,2,3,[4,5,[6,7]]]) )
```

### 5.2 数组去重

使用 ES6 的 Set 和 ...rest 操作符剔除重复的值。

```
const distinctValuesOfArray = arr => [...new Set(arr)];
distinctValuesOfArray([1, 2, 2, 3, 4, 4, 5]); // [1,2,3,4,5]
```

### 5.3 将数字转化为千分位格式

使用 toLocaleString() 将浮点数转换为 Decimal mark 格式。 将整数部分转化为用逗号分隔的字符串。

```
const toDecimalMark = num => num.toLocaleString('en-US');
toDecimalMark(12305030388.9087); // "12,305,030,388.909"
```

### 5.4 计算函数执行所花费的时间

使用 console.time() 和 console.timeEnd() 来测量开始和结束时间之间的差，以确定回调执行的时间。

```
const timeTaken = callback => {
  console.time('timeTaken');
  const r = callback();
  console.timeEnd('timeTaken');
  return r;
};
timeTaken(() => Math.pow(2, 10)); // 1024, (logged): timeTaken: 0.02099609375ms
```

### 5.5 解析网址参数

通过适当的正则表达式，使用 String.match() 来获得所有的键值对，Array.reduce() 来映射和组合成一个单一的对象。将 location.search 作为参数传递给当前 url。

```
const getURLParameters = url =>
  url
    .match(/([^?=&]+)(=([^&]*))/g)
    .reduce((a, v) => ((a[v.slice(0, v.indexOf('='))] = v.slice(v.indexOf('=') + 1)), a), {});
getURLParameters('http://url.com/page?name=Adam&surname=Smith'); // {name: 'Adam', surname: 'Smith'}
```

### 5.6 合并对象

使用 Array.reduce() 与 Object.keys(obj) 结合来遍历所有对象和键。 使用 hasOwnProperty() 和 Array.concat() 为存在与多个对象中的键添加值。

```
onst merge = (...objs) =>
  [...objs].reduce(
    (acc, obj) =>
      Object.keys(obj).reduce((a, k) => {
        acc[k] = acc.hasOwnProperty(k) ? [].concat(acc[k]).concat(obj[k]) : obj[k];
        return acc;
```

```
            }, {}),
        {}
    );
const object = {
    a: [{ x: 2 }, { y: 4 }],
    b: 1
};
const other = {
    a: { z: 3 },
    b: [2, 3],
    c: 'foo'
};
merge(object, other); // { a: [ { x: 2 }, { y: 4 }, { z: 3 } ], b: [ 1, 2, 3 ], c:
```

### 5.7 对象转化为键值对

使用 Object.keys() 和 Array.map() 遍历对象的键并生成一个包含键值对的数组。

```
const objectToPairs = obj => Object.keys(obj).map(k => [k, obj[k]]);
objectToPairs({ a: 1, b: 2 }); // [['a',1],['b',2]]
```

### 5.8 散列算法

使用 String.split('') 和 Array.reduce() 创建输入字符串的散列，利用位移。

```
const sdbm = str => {
    let arr = str.split('');
    return arr.reduce(
        (hashCode, currentVal) =>
            (hashCode = currentVal.charCodeAt(0) + (hashCode << 6) + (hashCode << 16) - hashCode),
        0
    );
};
sdbm('name'); // -3521204949
```

### 5.9 幂集

使用 Array.reduce() 与 Array.map() 结合来遍历元素，并将其组合成一个包含所有排列组合的数组。

```
const powerset = arr => arr.reduce((a, v) => a.concat(a.map(r => [v].concat(r))), [[]]);
```

```
powerset([1, 2]); // [[], [1], [2], [2,1]]
```

### 5.10 最大公约数

内部的 _gcd 函数使用递归。基本情况是，当 y 等于 0 的情况下，返回 x 。否则，返回 y 的最大公约数和 x / y 的其余数。

```
const gcd = (...arr) => {
  const _gcd = (x, y) => (!y ? x : gcd(y, x % y));
  return [...arr].reduce((a, b) => _gcd(a, b));
};
gcd(8, 36); // 4
gcd(...[12,8,32]); // 4
```

### 5.11 生成斐波纳契数组

创建一个指定长度的空数组，初始化前两个值( 0 和 1 )。使用 Array.reduce() 向数组中添加值，该值是最后两个值的和，前两个值除外。

```
const fibonacci = n =>
  Array.from({ length: n }).reduce(
    (acc, val, i) => acc.concat(i > 1 ? acc[i - 1] + acc[i - 2] : i),
    []
  );
fibonacci(6); // [0, 1, 1, 2, 3, 5]
```

### 5.12 等级评分算法

使用 Elo 评分系统 计算两个或两个以上对手之间的新评分。它需要一个预定义数组，并返回一个包含事后评级的数组。 数组应该从最高评分到最低评分排序（赢家 -> 失败者）。

使用指数 ** 操作符和数学运算符来计算预期分数(获胜几率)，并计算每个对手的新评级。对每个对手计算新的评分。 循环评分，使用每个排列组合，以成对方式计算每个玩家的 Elo 评分。 忽略第二个参数，使用默认的 k-factor 为 32。

```
const elo = ([...ratings], kFactor = 32, selfRating) => {
  const [a, b] = ratings;
  const expectedScore = (self, opponent) => 1 / (1 + 10 ** ((opponent - self) / 400));
  const newRating = (rating, i) =>
    (selfRating || rating) + kFactor * (i - expectedScore(i ? a : b, i ? b : a));
  if (ratings.length === 2) {
    return [newRating(a, 1), newRating(b, 0)];
```

```
    } else {
      for (let i = 0; i < ratings.length; i++) {
        let j = i;
        while (j < ratings.length - 1) {
          [ratings[i], ratings[j + 1]] = elo([ratings[i], ratings[j + 1]], kFactor);
          j++;
        }
      }
    }
    return ratings;
  };
  elo([1200, 1200]);
  elo([1200, 1200], 64);
  elo([1200, 1200, 1200, 1200]).map(Math.round);
```

# 六、Vue 性能优化

**1. 利用 Object.freeze()提升性能，Vue 在遇到像 Object.freeze() 这样被设置为不可配置之后的对象属性时，不会为对象加上 setter getter 等数据劫持的方法。**

代码：

```
export default{
    data () {
        return {
            list:[]
        }
    },
    async created(){
        let list = await this.axios.get('/api/list');
        this.list = Object.freeze(list);
    }
}
```

**2. v-if 和 v-show 区分使用场景**

v-if 是 真正 的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建；也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

v-show 就简单得多， 不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 的 display 属性进行切换。

所以，v-if 适用于在运行时很少改变条件，不需要频繁切换条件的场景；v-show 则适用于需要非常频繁切换条件的场景。

**3. computed 和 watch 区分使用场景**

computed： 是计算属性，依赖其它属性值，并且 computed 的值有缓存，只有它依赖的属性值发生改变，下一次获取 computed 的值时才会重新计算 computed 的值；

watch： 更多的是「观察」的作用，类似于某些数据的监听回调 ，每当监听的数据变化时都会执行回调进行后续操作；

运用场景：

当我们需要进行数值计算，并且依赖于其它数据时，应该使用 computed，因为可以利用 computed 的缓存特性，避免每次获取值时，都要重新计算；

当我们需要在数据变化时执行异步或开销较大的操作时，应该使用 watch，使用 watch 选项允许我们执行异步操作 ( 访问一个 API )，限制我们执行该操作的频率，并在我们得到最终结果前，设置中间状态。这些都是计算属性无法做到的。

**4. v-for 遍历必须为 item 添加 key，且避免同时使用 v-if**

（1）v-for 遍历必须为 item 添加 key

在列表数据进行遍历渲染时，需要为每一项 item 设置唯一 key 值，方便 Vue.js 内部机制精准找到该条列表数据。当 state 更新时，新的状态值和旧的状态值对比，较快地定位到 diff 。

（2）v-for 遍历避免同时使用 v-if

v-for 比 v-if 优先级高，如果每一次都需要遍历整个数组，将会影响速度，尤其是当之需要渲染很小一部分的时候，必要情况下应该替换成 computed 属性。

推荐：

```
<ul>
  <li
    v-for="user in activeUsers"
    :key="user.id">
    {{ user.name }}
  </li>
</ul>
computed: {
  activeUsers: function () {
```

```
            return this.users.filter(function (user) {
          return user.isActive
            })
        }
    }
```

不推荐：

```
    <ul>
      <li
        v-for="user in users"
        v-if="user.isActive"
        :key="user.id">
        {{ user.name }}
      </li>
    </ul>
```

**5. 长列表性能优化**

Vue 会通过 Object.defineProperty 对数据进行劫持，来实现视图响应数据的变化，然而有些时候我们的组件就是纯粹的数据展示，不会有任何改变，我们就不需要 Vue 来劫持我们的数据，在大量数据展示的情况下，这能够很明显的减少组件初始化的时间，那如何禁止 Vue 劫持我们的数据呢？可以通过 Object.freeze 方法来冻结一个对象，一旦被冻结的对象就再也不能被修改了。

```
export default {
  data: () => ({
    users: {}
  }),
  async created() {
    const users = await axios.get("/api/users");
    this.users = Object.freeze(users);
  }
};
```

**6. 事件的销毁**

Vue 组件销毁时，会自动清理它与其它实例的连接，解绑它的全部指令及事件监听器，但是仅限于组件本身的事件。如果在 js 内

```
created() {
  addEventListener('click', this.click, false)
},
beforeDestroy() {
  removeEventListener('click', this.click, false)
```

}

**7. 图片资源懒加载**

对于图片过多的页面，为了加速页面加载速度，所以很多时候我们需要将页面内未出现在可视区域内的图片先不做加载， 等到滚动到可视区域后再去加载。这样对于页面加载性能上会有很大的提升，也提高了用户体验。我们在项目中使用 Vue 的 vue-lazyload 插件：

（1）安装插件

```
npm install vue-lazyload --save-dev
```

（2）在入口文件 man.js 中引入并使用

```
import VueLazyload from 'vue-lazyload'
```

然后再 vue 中直接使用

```
Vue.use(VueLazyload)
```

或者添加自定义选项

```
Vue.use(VueLazyload, {
preLoad: 1.3,
error: 'dist/error.png',
loading: 'dist/loading.gif',
attempt: 1
})
```

（3）在 vue 文件中将 img 标签的 src 属性直接改为 v-lazy ，从而将图片显示方式更改为懒加载显示：

```
<img v-lazy="/static/img/1.png">
```

以上为 vue-lazyload 插件的简单使用，如果要看插件的更多参数选项，可以查看 vue-lazyload 的 github 地址。

**8. 路由懒加载**

Vue 是单页面应用，可能会有很多的路由引入 ，这样使用 webpcak 打包后的文件很大，当进入首页时，加载的资源过多，页面会出现白屏的情况，不利于用户体验。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应的组件，这样就更加高效了。这样会大大提高首屏显示的速度，但是可能其他的页面的速度就会降下来。

路由懒加载：

```
const Foo = () => import('./Foo.vue')
const router = new VueRouter({
  routes: [
    { path: '/foo', component: Foo }
  ]
})
```

**9. 第三方插件的按需引入**

我们在项目中经常会需要引入第三方插件，如果我们直接引入整个插件，会导致项目的体积太大，我们可以借助 babel-plugin-component ，然后可以只引入需要的组件，以达到减小项目体积的目的。以下为项目中引入 element-ui 组件库为例：

（1）首先，安装 babel-plugin-component ：

```
npm install babel-plugin-component -D
```

（2）然后，将 .babelrc 修改为：

```
{
  "presets": [["es2015", { "modules": false }]],
  "plugins": [
    [
      "component",
      {
        "libraryName": "element-ui",
        "styleLibraryName": "theme-chalk"
      }
    ]
  ]
}
```

（3）在 main.js 中引入部分组件：

```
import Vue from 'vue';
import { Button, Select } from 'element-ui';

 Vue.use(Button)
 Vue.use(Select)
```

**10. 优化无限列表性能**

如果你的应用存在非常长或者无限滚动的列表，那么需要采用 窗口化 的技术来优化性能，只需要渲染少部分区域的内容，减少重新渲染组件和创建 dom 节点

的时间。你可以参考以下开源项目 vue-virtual-scroll-list 和 vue-virtual-scroller 来优化这种无限列表的场景的。

### 11. 服务端渲染 SSR or 预渲染

服务端渲染是指 Vue 在客户端将标签渲染成的整个 html 片段的工作在服务端完成，服务端形成的 html 片段直接返回给客户端这个过程就叫做服务端渲染。

（1）服务端渲染的优点：

更好的 SEO：因为 SPA 页面的内容是通过 Ajax 获取，而搜索引擎爬取工具并不会等待 Ajax 异步完成后再抓取页面内容，所以在 SPA 中是抓取不到页面通过 Ajax 获取到的内容；而 SSR 是直接由服务端返回已经渲染好的页面（数据已经包含在页面中），所以搜索引擎爬取工具可以抓取渲染好的页面；

更快的内容到达时间（首屏加载更快）：SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，文件下载等需要一定的时间等，所以首屏渲染需要一定的时间；SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，所以 SSR 有更快的内容到达时间；

（2）服务端渲染的缺点：

更多的开发条件限制：例如服务端渲染只支持 beforCreate 和 created 两个钩子函数，这会导致一些外部扩展库需要特殊处理，才能在服务端渲染应用程序中运行；并且与可以部署在任何静态文件服务器上的完全静态单页面应用程序 SPA 不同，服务端渲染应用程序，需要处于 Node.js server 运行环境；

更多的服务器负载：在 Node.js 中渲染完整的应用程序，显然会比仅仅提供静态文件的 server 更加大量占用 CPU 资源，因此如果你预料在高流量环境下使用，请准备相应的服务器负载，并明智地采用缓存策略。

如果你的项目的 SEO 和 首屏渲染是评价项目的关键指标，那么你的项目就需要服务端渲染来帮助你实现最佳的初始加载性能和 SEO，具体的 Vue SSR 如何实现，可以参考作者的另一篇文章《Vue SSR 踩坑之旅》。如果你的 Vue 项目只需改善少数营销页面（例如 /， /about，/contact 等）的 SEO，那么你可能需要预渲染，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，具体你可以使用 prerender-spa-plugin 就可以轻松地添加预渲染 。

### 12. 使用 keep-alive 缓存页面

可以使用 keep-alive，keep-alive 是 Vue 提供的一个比较抽象的组件，用来对组件进行缓存，从而节省性能。