你的项目有什么难点？

通过路由懒加载、图片懒加载、静态资源压缩、CDN 缓存静态资源、webpack 配置分包优化首屏加载。

通过虚拟列表优化渲染数据量过大的问题。

通过防抖减少用户频繁发送请求，通过并通过手机号唯一标识防止第三方请求工具恶意测试接口。

路由懒加载

1. 将需要进行懒加载的子模块打包成独立的文件

   可以通过在 import 中通过 webpackChunkName 字段指定按需加载的文件打包后生成的文件名
   `import(/* webpackChunkName: "con" */ './con.js')`

   并且在 webpack 配置中的 output 属性添加上 chukFilename 以 '[name].bundle.js' 的格式，这样在 import 中通过 `/* webpackChunkName:名字 */` 就可以将懒加载的子模块打包成独立的文件了。

   ```
   ```js
   const path = require("path");
   module.exports = {
     entry: "./src/main.js", //入口文件
     output: {
       path: path.resolve(__dirname, "dist"),
       chunkFilename: "[name].bundle.js",
       filename: "bundle.js"
     }
   };
   ```
   ```

2. 借助函数实现延迟执行子模块的加载代码

   ES6 提供了 import() 函数，可以实现运行时的异步加载，与 cjs 的 require 运行时同步加载不同。import 函数帮助我们将需要懒加载的子模块打包成独立的文件，并且可以将模块的引入，用函数的形式实现函数调用时加载子模块代码。

   我们知道 js 的工作过程分为解释和执行，在解释阶段便确定了作用域，并且在执行阶段创建执行上下文，而函数上下文的创建仅在函数调用时，因此函数只要不调用就不会创建相应的执行上下文也就不会执行函数体里的代码。

   语法：
   `() => import(路径)` 、 `resolve => require([路径], resolve)`

图片懒加载

图片懒加载主要是利用图片标签有 src 属性会就发送请求，如果没有 src 属性就不会发送请求的特性。

在图片标签中添加一个 data-src 属性，将图片的 url 放在 data-src 中，然后在图片快要进入可视区前或者已经在可视区时将 url 取出放入到 src 属性中。

1. 判断元素是否在可视区可以通过 元素到文档顶部的距离(固定)-浏览器窗口与文档顶部之间的距离，滚动条滚动距离(可变)是否小于屏幕的可视区高度(不变)来判断，小于就是在可视区内，大于就是不在可视区内，

```
Element.offsetTop - document.documentElement.scrollTop <
document.documentElement.clientHeight;
Element.offsetTop - document.documentElement.scrollTop < window.innerHeight;
```

2. 也可以通过 getBoundingClientRect 获取元素的大小和位置来判断是否在可视区内。当元素到可视窗口顶部的距离小于可视区高度时说明在可视区内。

   在监听滚动条滚动事件来更改图片的 url 的值时采用节流，防止频繁操作 DOM。

```
function isInSight(el) {
  const bound = Element.getBoundingClientRect();
  const eTop = bound.top,
    eBottom = bound.bottom;
  const clientHegiht = window.innerHeight;
  return eTop <= clientHeight;
}
function load(el) {
  if (isInSight(el)) {
    if (!el.src) {
      const src = el.dataset.src; // 通过 dataset 拿到所有 data- 自定义属性
      el.src = src;
      el.removeAttribute("data-src");
    }
  }
}
function throttle(fn, delay = 300) {
  let timer = null;
  return function (...args) {
    if (!timer) {
      timer = setTimeout(() => {
        timer = null; // 清除定时器
        fn.apply(this, args);
      }, delay);
    }
  };
}
function listenerScroll(el) {
  const handler = throttle(load, 300);
  load(el); // 首次渲染时
  window.addEventListener(
    "scroll",
    function () {
      handler(el);
    },
    false
  );
}
```

3. 也可以使用 IntersectionObserver 这个 API ，监测每一个图片元素 intersectionRatio 这个属性变化来加载图片，这个属性的含义是可见比例，处于 0-1 时表示元素可见。并且在每次元素 onload 完后取消对它的监听。

```
const io = new IntersectionObsever(io => {
  const el = io.target; // 获取目标元素
  const intersectionRatio = io.itersectionRatio;
  if (itersectionRatio > 0 && itersectionRatio <= 1) {
    if (!el.src) {
      const src = el.dataset.src; // 通过 dataset 拿到所有 data- 自定义属性
      el.src = src;
      el.removeAttribute("data-src");
    }
  }
  el.onload = el.onerror = () => io.unobserve(el);
});
const eleList = document.querySelectorAll("img");
eleList.forEach(el => {
  io.observe(el);
});
```

createDocumentFragment + requestAnimationFrame

因为 requestAnimationFrame 总是在下一次重绘之前，更新动画帧所调用的函数，并且会为该回调函数传入一个当前微秒级的时间戳。可以通过 requestAnimationFrame 替代 setTimeout 减少回流的次数，因为所有动画帧都会挤在一次重绘中执行，在每次要添加新的列表项数据时，先通过一个文档片 documentFragment 装载，再装载完列表项数据后一次性插入到容器尾部，减少回流的次数(如果是一个一个插入，每次插入都会引起回流)。

另外也可以不采用 setTimeout 和 requestAnimationFrame ，通过懒加载的形式添加列表项数据，基本原理与图片懒加载相似，可以在每次添加完列表项后记录最后一个列表项的下标，监听这个下标的列表项元素是否快到达可视区内，如果是就执行一次装载并更新最后一个列表项的下标。

缺点：装载完的元素始终保存在 DOM 树中，占用浏览器内存，解决方法就是通过虚拟列表进行优化。

虚拟列表

维护两个下标 startIndex 和 endIndex ，通过 列表容器滚动距离 动态计算可视区列表的 startIndex `开始下标` 和 endIndex `结束下标` ，可视区列表中只存储 `开始下标` 到 `结束下标` 中的列表项。开始下标通过 `列表容器向上滚动距离` / `列表项高度` 计算， `结束下标` 通过 `开始下标` + `列表容器可视区域能存储的列表项数` * 2 计算，乘以 2 主要是为了预加载一部分内容，防止滚动过快出现白屏， `列表容器可视区能存储的列表项数` 通过 `列表容器的高度` / `列表项的高度` 得到。并且在列表项外包裹一个扩高的容器实现列表的滚动，扩高容器通过 padding 实现。

webpack 配置分包

减少接口恶意测试

在每次接收到发送手机验证码的请求时，在 redis 中保存键名为手机号码，键值为该手机对应的验证码的键值对，并设置存活时间为 1 min ，表明该手机号码 1min 内已经发送过验证码。并且在每次发送手机验证码之前，都会去依据手机号码作为键，去 redis 中检测该手机号码是否在 1min 内发送过验证码了，如果是就返回一个内容为频繁发送验证码的服务端错误响应，并不做任何处理。

权限认证

整个博客系统分为三种角色，游客、普通用户、管理员用户。权限分为三级权限，一级权限是应用访问权限，本博客中应用分为主站和用户个人中心也就是后台，所有角色都具有访问主站的权限、后台访问权限只有管理员或在本站中注册的用户具有；二级权限是菜单访问权限以及按钮的访问权限，例如后台中管理员会比普通用户多一个用户管理的菜单栏，因为管理员具有管理用户的权限(增删改用户角色以及增删改用户权限)、以及文章底部的评论区只有具有评论权限的用户才可以发布评论；三级权限是数据的控制权限，例如具有文章管理权限的用户可以实现文章的发布、修改和删除。

博客系统中游客仅有访问主站中各个页面的权限，普通用户具有访问后台、博客管理、评论等权限，管理员用户具有站点中所有权限。

在前端通过全局路由守卫 beforeEach 对访问后台的用户进行拦截校验权限，从 Vuex 中或者 localStorage 中获取用户的信息，游客默认是不具有用户信息的，因此在每次访问后台时并不放行而是跳转到登录页，而用户只要登录后后端就会返回用户基本的信息以及用户的权限列表，默认用户是具有访问后台的权限的，因此可以进入后台。用户管理的菜单权限通过动态挂载路由实现，给后台的路由项添加一个 beforeEnter 路由守卫，并校验该用户是否具有用户管理的权限，如果具有就会添加一项用户管理这一菜单栏的子路由项。

如果用户手动修改了在 localStorage 中的权限列表，虽然可以进入后台或者是显示用户管理的菜单栏，但实际上不会具有执行该操作的权限，在后端每次接收到一些涉及权限的请求时，会从数据库中查找出该用户的权限信息，如果查询到用户不具有该接口的权限，就不会执行用户的操作并且还会返回一个响应让用户跳转回登录页面并清空 localStorage 中的信息。

用户评论的 xss 防护

通过 rem、响应式布局 作移动端适配