```
/**
 * GreenWeb后端代理服务器
 * 用于处理跨域请求和获取网站性能数据
 */

const express = require('express');
const axios = require('axios');
const cors = require('cors');
const dns = require('dns').promises;
const { performance } = require('perf_hooks');
const { URL } = require('url');
const geoip = require('geoip-lite');
const whois = require('whois-json');
const useragent = require('express-useragent');
const path = require('path');
const https = require('https');
const os = require('os');
const { execSync } = require('child_process');
const puppeteer = require('puppeteer');
const fetch = require('node-fetch'); // 用于API请求

// 声明chromeLauncher变量，稍后通过动态导入获取
let chromeLauncher;

// 异步初始化函数用于导入ESM模块
async function initializeESModules() {
  try {
    // 动态导入chrome-launcher
    chromeLauncher = await import('chrome-launcher');
    console.log('Chrome Launcher模块已成功导入');
  } catch (err) {
    console.error('导入ESM模块失败:', err);
  }
}

// 执行初始化
initializeESModules().catch(err => {
  console.error('初始化ESM模块时出错:', err);
});

const app = express();
const PORT = process.env.PORT || 3000;

// 中间件
app.use(cors());
app.use(express.json());
app.use(useragent.express());
app.use(express.static(path.join(__dirname, '../dist')));

// 创建不验证SSL证书的axios实例
const axiosInstance = axios.create({
  httpsAgent: new https.Agent({
    rejectUnauthorized: false
```

```
  }),
  timeout: 10000
});

// 全局常量 - 用于碳排放计算
const GLOBAL_CONSTANTS = {
  // 能源消耗常量
  averageEnergyConsumption: 1.805, // kWh/GB
  greenEnergyCarbonIntensity: 50, // gCO2e/kWh
  averageCarbonIntensity: 475, // gCO2e/kWh

  // 数据中心效率
  averagePUE: 1.67, // 电能使用效率
  bestPUE: 1.1,

  // 传输和设备能耗
  averageTransmissionPerGB: 0.06, // kWh/GB
  averageDevicePerGB: 0.08, // kWh/GB

  // 缓存效率
  cachingEfficiency: 0.2, // 20%的流量被缓存

  // 访问量估算
  averageMonthlyVisits: 10000,

  // 碳中和阈值
  greenEnergyThreshold: 80, // 80%以上可再生能源视为碳中和

  // 每年一棵树可吸收二氧化碳量（kg）
  treeCO2PerYear: 25
};

// 国家碳强度数据（gCO2e/kWh）
const COUNTRY_CARBON_INTENSITY = {
  'US': 383,
  'CN': 554,
  'IN': 739,
  'JP': 478,
  'DE': 344,
  'GB': 231,
  'FR': 56,
  'IT': 331,
  'BR': 87,
  'CA': 135,
  'KR': 415,
  'RU': 351,
  'AU': 656,
  'ES': 200,
  'MX': 428,
  'ID': 736,
  'NL': 358,
  'SA': 523,
  'CH': 24,
  'TR': 461,
  'SE': 13,
```

```
  'PL': 751,
  'BE': 161,
  'TH': 471,
  'AT': 109,
  'IE': 291,
  'SG': 418,
  'IL': 529,
  'DK': 135,
  'FI': 89,
  'NO': 8,
  // 默认值将使用全球平均值
};

// 主要云服务提供商信息
const PROVIDER_INFO = {
  'aws': {
    name: 'Amazon Web Services',
    renewableRange: [60, 85],
    renewableChance: 0.8,
    pueRange: [1.1, 1.4],
    serverEfficiency: 1.2,
    regions: ['us-east', 'us-west', 'eu-west', 'eu-central', 'ap-northeast', 'ap-southeast', 'sa-east']
  },
  'azure': {
    name: 'Microsoft Azure',
    renewableRange: [65, 90],
    renewableChance: 0.85,
    pueRange: [1.1, 1.35],
    serverEfficiency: 1.25,
    regions: ['us-east', 'us-west', 'eu-west', 'eu-north', 'asia-east', 'asia-southeast', 'australia-east']
  },
  'google': {
    name: 'Google Cloud',
    renewableRange: [80, 95],
    renewableChance: 0.9,
    pueRange: [1.1, 1.2],
    serverEfficiency: 1.3,
    regions: ['us-central', 'us-east', 'us-west', 'europe-west', 'europe-north', 'asia-east', 'asia-south']
  },
  'cloudflare': {
    name: 'Cloudflare',
    renewableRange: [70, 90],
    renewableChance: 0.8,
    pueRange: [1.1, 1.3],
    serverEfficiency: 1.2,
    regions: ['global-edge']
  },
  'alibaba': {
    name: 'Alibaba Cloud',
    renewableRange: [45, 70],
    renewableChance: 0.6,
    pueRange: [1.3, 1.6],
```

```
    serverEfficiency: 1.1,
    regions: ['cn-hangzhou', 'cn-shanghai', 'cn-beijing', 'us-west', 'eu-central', 'ap-southeast']
  },
  'tencent': {
    name: 'Tencent Cloud',
    renewableRange: [40, 65],
    renewableChance: 0.5,
    pueRange: [1.35, 1.65],
    serverEfficiency: 1.0,
    regions: ['ap-guangzhou', 'ap-shanghai', 'ap-beijing', 'na-siliconvalley', 'eu-frankfurt', 'ap-singapore']
  },
  'other': {
    name: 'Unknown Provider',
    renewableRange: [30, 60],
    renewableChance: 0.4,
    pueRange: [1.5, 1.9],
    serverEfficiency: 0.9,
    regions: ['unknown']
  }
};

/**
 * 健康检查端点
 */
app.get('/api/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

/**
 * 获取服务器位置信息
 */
app.get('/api/location', async (req, res) => {
  try {
    const { domain } = req.query;

    if (!domain) {
      return res.status(400).json({ error: '缺少域名参数' });
    }

    // 使用DNS查询获取IP地址
    const addresses = await dns.lookup(domain, { all: true });
    const ip = addresses[0]?.address;

    if (!ip) {
      return res.status(404).json({ error: '无法解析域名' });
    }

    // 使用GeoIP获取位置信息
    const geo = geoip.lookup(ip);

    // 尝试使用WHOIS获取额外信息
    let whoisData = {};
    try {
```

```
      whoisData = await whois(domain);
    } catch (whoisError) {
      console.error('WHOIS查询失败:', whoisError);
    }

    res.json({
      ip,
      country: geo?.country || null,
      region: geo?.region || null,
      city: geo?.city || null,
      timezone: geo?.timezone || null,
      coordinates: geo?.ll || null,
      org: geo?.org || null,
      registrar: whoisData.registrar || null,
      registrantCountry: whoisData.registrantCountry || null
    });
  } catch (error) {
    console.error('位置查询错误:', error);
    res.status(500).json({ error: '位置查询失败', message: error.message });
  }
});

/**
 * 获取HTTP响应头信息
 */
app.get('/api/headers', async (req, res) => {
  try {
    const { url } = req.query;

    if (!url) {
      return res.status(400).json({ error: '缺少URL参数' });
    }

    const startTime = performance.now();
    const response = await axiosInstance.head(url, {
      maxRedirects: 5,
      validateStatus: null
    });
    const endTime = performance.now();

    res.json({
      status: response.status,
      statusText: response.statusText,
      headers: response.headers,
      responseTime: endTime - startTime
    });
  } catch (error) {
    console.error('头信息获取错误:', error);
    res.status(500).json({ error: '头信息获取失败', message: error.message });
  }
});

/**
 * 测量页面大小
 */
```

```
app.get('/api/size', async (req, res) => {
  try {
    const { url } = req.query;

    if (!url) {
      return res.status(400).json({ error: '缺少URL参数' });
    }

    const startTime = performance.now();
    const response = await axiosInstance.get(url, {
      maxRedirects: 5,
      validateStatus: null,
      responseType: 'arraybuffer' // 获取二进制响应以精确计算大小
    });
    const endTime = performance.now();

    // 计算页面大小（KB）
    const contentLength = response.headers['content-length'] ||
response.data.length;
    const sizeInKB = Math.round(contentLength / 1024);

    res.json({
      size: sizeInKB,
      contentType: response.headers['content-type'],
      responseTime: endTime - startTime
    });
  } catch (error) {
    console.error('页面大小测量错误:', error);
    res.status(500).json({ error: '页面大小测量失败', message: error.message
});
  }
});

/**
 * 分析服务提供商
 */
app.get('/api/provider', async (req, res) => {
  try {
    const { domain } = req.query;

    if (!domain) {
      return res.status(400).json({ error: '缺少域名参数' });
    }

    // 使用DNS和WHOIS信息尝试确定服务提供商
    const addresses = await dns.lookup(domain, { all: true });
    const ip = addresses[0]?.address;

    if (!ip) {
      return res.status(404).json({ error: '无法解析域名' });
    }

    // 使用GeoIP获取组织信息
    const geo = geoip.lookup(ip);
    const org = geo?.org || '';
```

```
322
323     // 尝试确定服务提供商
324     let provider = 'other';
325
326     if (org.includes('AMAZON') || org.includes('AWS')) {
327       provider = 'aws';
328     } else if (org.includes('MICROSOFT') || org.includes('MSFT')) {
329       provider = 'azure';
330     } else if (org.includes('GOOGLE')) {
331       provider = 'google';
332     } else if (org.includes('ALIBABA') || org.includes('ALIYUN')) {
333       provider = 'alibaba';
334     } else if (org.includes('TENCENT')) {
335       provider = 'tencent';
336     } else if (org.includes('CLOUDFLARE')) {
337       provider = 'cloudflare';
338     }
339
340     // 获取提供商详细信息
341     const providerInfo = PROVIDER_INFO[provider] || PROVIDER_INFO.other;
342
343     // 估计可再生能源使用比例
344     const [minRenewable, maxRenewable] = providerInfo.renewableRange;
345     const renewablePercentage = Math.floor(Math.random() * (maxRenewable -
    minRenewable + 1)) + minRenewable;
346
347     // 计算数据中心PUE（电能使用效率）
348     const [minPUE, maxPUE] = providerInfo.pueRange;
349     const dataCenterPUE = parseFloat(safeToFixed(Math.random() * (maxPUE -
    minPUE) + minPUE, 2));
350
351     res.json({
352       provider,
353       providerName: providerInfo.name,
354       ip,
355       org,
356       renewablePercentage,
357       pue: dataCenterPUE,
358       serverEfficiency: providerInfo.serverEfficiency
359     });
360   } catch (error) {
361     console.error('提供商分析错误:', error);
362     res.status(500).json({ error: '提供商分析失败', message: error.message });
363   }
364 });
365 /**
366  * 修改性能测量API端点，添加WebPageTest选项
367  */
368 app.get('/api/performance', async (req, res) => {
369   try {
370     const url = req.query.url;
371     const requestedMethod = req.query.browser || 'auto';
372
373     if (!url) {
374       return res.status(400).json({ error: '请提供URL' });
```

```
    }

    console.log(`接收到性能分析请求: ${url}, 方法: ${requestedMethod}`);

    let result;
    let allErrors = [];

    // 尝试基础HTTP分析
    try {
      if (requestedMethod === 'basic' || requestedMethod === 'basic-http' || requestedMethod === 'auto') {
        console.log('尝试使用基础HTTP方法分析...');
        result = await measurePerformanceBasic(url);

        if (result.success) {
          console.log('基础HTTP分析成功');
          // 发起碳排放计算请求
          try {
            const carbonParams = new URLSearchParams({
              pageSize: result.performance.pageSize,
              requestCount: result.performance.requestCount,
              domainCount: result.performance.domainCount,
              responseTime: result.performance.responseTime,
              hasCompression: result.headers.supportsCompression,
              resourceStats: JSON.stringify(result.performance.resourceStats)
            });

            const carbonResult = await axios.get(`http://localhost:${PORT}/api/carbon?${carbonParams}`);

            if (carbonResult.data && carbonResult.data.measurable) {
              result.carbonEmission = carbonResult.data;
            }
          } catch (carbonError) {
            console.error('碳排放计算错误:', carbonError);
          }

          // 清除任何非真实测量的数据
          for (const metric in result.performance) {
            if (result.performance[metric] === null || result.performance[metric] === undefined) {
              delete result.performance[metric];
            }
          }

          // 只返回真实测量的数据
          return res.json({
            success: true,
            performance: result.performance,
            headers: result.headers,
            carbonEmission: result.carbonEmission || { measurable: false },
            measurementMethod: result.measurementMethod,
            measuredBy: 'basic-http',
            allDataIsReal: true // 标记所有数据都是真实的
```

```
426             });
427           } else {
428             console.log('基础HTTP分析失败:', result.error);
429             allErrors.push(result.error || '基础HTTP分析失败，无具体错误信息');
430           }
431         }
432       } catch (basicError) {
433         console.error('基础HTTP分析抛出异常:', basicError);
434         allErrors.push(`基础HTTP分析异常: ${basicError.message}`);
435       }
436 
437       // 如果所有方法都失败，返回基础HTTP分析结果或错误信息
438       return res.status(500).json({
439         success: false,
440         error: '所有分析方法都失败',
441         details: allErrors,
442         measurable: false
443       });
444     } catch (error) {
445       console.error('性能测量API错误:', error);
446       res.status(500).json({
447         success: false,
448         error: '性能测量API发生错误',
449         message: error.message,
450         measurable: false
451       });
452     }
453 });
454 
455 /**
456  * 碳排放计算 - 优化使用实际测量数据
457  */
458 app.get('/api/carbon', async (req, res) => {
459   try {
460     const {
461       pageSize,
462       country,
463       renewablePercentage,
464       pue,
465       requestCount,
466       domainCount,
467       resourceStats,
468       responseTime,
469       hasCompression
470     } = req.query;
471 
472     // 检查必须的真实测量数据
473     if (!pageSize || parseInt(pageSize) <= 0) {
474       return res.status(400).json({
475         error: '缺少页面大小参数或值为零',
476         measurable: false,
477         message: '无法获取页面大小数据，无法计算能源消耗'
478       });
479     }
480 
```

```
    // 检查其他必要的真实数据
    if (!requestCount || !domainCount) {
      return res.status(400).json({
        error: '缺少网络请求数据',
        measurable: false,
        message: '无法获取网络请求和域名数量，无法准确计算碳排放'
      });
    }

    // 使用实际测量值，不进行估算
    const pageSizeKB = parseInt(pageSize);
    const countryCode = country || null; // 不提供默认值，如果没有则返回错误
    const renewable = parseFloat(renewablePercentage);
    const dataCenterPUE = parseFloat(safeToFixed(pue, 2));
    const actualRequestCount = parseInt(requestCount);
    const actualDomainCount = parseInt(domainCount);
    const actualResponseTime = parseInt(responseTime) || 0;
    const isCompressed = hasCompression === 'true' || hasCompression === true;

    // 检查国家信息
    if (!countryCode) {
      return res.status(400).json({
        error: '缺少国家/地区信息',
        measurable: false,
        message: '无法获取服务器地理位置，无法准确计算碳排放'
      });
    }

    // 检查可再生能源信息
    if (isNaN(renewable)) {
      return res.status(400).json({
        error: '缺少可再生能源使用比例',
        measurable: false,
        message: '无法获取服务器使用的可再生能源比例，无法准确计算碳排放'
      });
    }

    // 检查PUE信息
    if (isNaN(dataCenterPUE)) {
      return res.status(400).json({
        error: '缺少数据中心PUE',
        measurable: false,
        message: '无法获取数据中心PUE，无法准确计算碳排放'
      });
    }

    // 解析资源统计数据 - 只使用真实值
    let resourceStatsData = {};
    if (resourceStats) {
      try {
        resourceStatsData = typeof resourceStats === 'string' ? JSON.parse(resourceStats) : resourceStats;
      } catch (e) {
        console.error('解析资源统计数据失败:', e);
```

```
534           return res.status(400).json({
535             error: '资源统计数据无效',
536             measurable: false,
537             message: '无法解析资源统计数据，无法准确计算碳排放'
538           });
539         }
540       } else {
541         return res.status(400).json({
542           error: '缺少资源统计数据',
543           measurable: false,
544           message: '无法获取资源统计数据，无法准确计算碳排放'
545         });
546       }
547
548       // 使用真实数据计算而不是估算
549       const pageSizeInGB = pageSizeKB / 1024 / 1024;
550
551       // 基于实际测量的缓存效率，不使用估算值
552       const cacheControlHeader = req.query.cacheControl;
553       let measuredCacheEfficiency = 0;
554
555       // 只有当存在真实的Cache-Control头时才使用缓存效率
556       if (cacheControlHeader) {
557         const maxAge = /max-age=(\d+)/.exec(cacheControlHeader);
558         if (maxAge && maxAge[1]) {
559           const maxAgeValue = parseInt(maxAge[1]);
560           // 基于max-age值计算缓存效率
561           if (maxAgeValue > 86400) { // 1天以上
562             measuredCacheEfficiency = 0.6;
563           } else if (maxAgeValue > 3600) { // 1小时以上
564             measuredCacheEfficiency = 0.4;
565           } else if (maxAgeValue > 0) {
566             measuredCacheEfficiency = 0.2;
567           }
568         }
569       }
570
571       // 压缩效率 - 根据是否启用压缩（这是真实测量的）
572       const compressionEfficiency = isCompressed ? 0.7 : 1.0;
573
574       // 能源强度计算 - 使用国际能源署的真实数据
575       const countryCarbonValue = COUNTRY_CARBON_INTENSITY[countryCode] ||
    null;
576       if (!countryCarbonValue) {
577         return res.status(400).json({
578           error: '无法获取国家电网碳强度',
579           measurable: false,
580           message: `无法获取 ${countryCode} 的电网碳强度数据，无法准确计算碳排放`
581         });
582       }
583
584       // 基于真实测量值计算能源消耗
585       const baseEnergyIntensity = GLOBAL_CONSTANTS.averageEnergyConsumption;
586       const energyIntensity = baseEnergyIntensity * dataCenterPUE;
587
```

```
    // 计算能源消耗 - 确保即使页面很小也有最小值
    // 使用Math.max确保能源消耗有一个最小基准值
    const pageSizeInGBSafe = Math.max(pageSizeInGB, 0.0001); // 确保至少有0.1MB
    const dataCenterEnergy = pageSizeInGBSafe * (energyIntensity / dataCenterPUE);
    const transmissionEnergy = pageSizeInGBSafe * GLOBAL_CONSTANTS.averageTransmissionPerGB;
    const deviceEnergy = pageSizeInGBSafe * GLOBAL_CONSTANTS.averageDevicePerGB;

    // 数据中心碳排放 - 考虑可再生能源比例
    const dataTransferCarbon = dataCenterEnergy * (
      (renewable / 100) * GLOBAL_CONSTANTS.greenEnergyCarbonIntensity +
      ((100 - renewable) / 100) * countryCarbonValue
    );

    // 网络传输和客户设备碳排放
    const networkCarbon = transmissionEnergy * countryCarbonValue;
    const clientCarbon = deviceEnergy * countryCarbonValue;

    // 计算总碳排放量（单位：g CO2e）
    const totalCarbonEmission = dataTransferCarbon + networkCarbon + clientCarbon;

    // 估计月访问量 - 基于固定值，但明确标记这不是测量值
    const monthlyCarbonEmission = (totalCarbonEmission * GLOBAL_CONSTANTS.averageMonthlyVisits) / 1000; // 单位：kg CO2e
    const annualCarbonEmission = monthlyCarbonEmission * 12;

    // 计算碳中和所需的树木数量
    const treesNeeded = Math.ceil(annualCarbonEmission / GLOBAL_CONSTANTS.treeCO2PerYear);

    // 碳足迹和能源效率评分 - 基于真实测量值计算
    const carbonFootprintScore = Math.min(100, Math.max(1,
      (totalCarbonEmission / 0.5) * 20 + // 基础碳排放
      (dataCenterPUE / GLOBAL_CONSTANTS.bestPUE - 1) * 20 + // 数据中心效率
      ((100 - renewable) / 100) * 40 // 可再生能源使用
    ));

    const energyEfficiencyScore = Math.min(100, Math.max(1,
      100 - (totalCarbonEmission / 3) * 20 - // 基础能源效率
      (actualDomainCount - 1) * 3 // 域名数量惩罚（实际测量）
    ));

    res.json({
      measurable: true,
      pageSize: pageSizeKB,
      energyIntensity,
      cachingEfficiency: measuredCacheEfficiency,
      compressionEfficiency: isCompressed ? 0.3 : 0, // 压缩节省百分比（实际测量）
      dataCenterEnergy,
      transmissionEnergy,
```

```
      deviceEnergy,
      dataTransferCarbon,
      networkCarbon,
      clientCarbon,
      totalCarbonEmission,
      monthlyCarbonEmission,
      annualCarbonEmission,
      treesNeeded,
      isGreen: renewable ≥ GLOBAL_CONSTANTS.greenEnergyThreshold,
      carbonFootprintScore,
      energyEfficiencyScore,
      resourceBreakdown: resourceStatsData,
      requestCount: actualRequestCount,
      domainCount: actualDomainCount,
      dataSourceInfo: {
        allRealMeasurements: true,
        estimatedValues: ['monthlyCarbonEmission', 'annualCarbonEmission', 'treesNeeded'],
        explanation: '月度和年度碳排放基于固定的月访问量估算，树木数量基于年均CO2吸收量估算。所有其他数据基于实际测量。'
      }
    });
  } catch (error) {
    console.error('碳排放计算错误:', error);
    res.status(500).json({
      error: '碳排放计算失败',
      message: error.message,
      measurable: false
    });
  }
});

/**
 * 生成优化建议
 */
app.get('/api/suggestions', async (req, res) ⇒ {
  try {
    const {
      pageSize,
      performance,
      provider,
      renewablePercentage,
      pue,
      totalCarbonEmission,
      carbonFootprintScore,
      energyEfficiencyScore,
      treesNeeded,
      requestCount,
      domainCount
    } = req.query;

    // 提取性能数据
    const performanceData = typeof performance === 'string' ? JSON.parse(performance) : (performance || {});

```

```
    // 准备建议生成的数据
    const data = {
      pageSize: parseInt(pageSize) || 0,
      performance: {
        fcp: parseFloat(performanceData.fcp) || 1.5,
        lcp: parseFloat(performanceData.lcp) || 2.5,
        cls: parseFloat(performanceData.cls) || 0.05,
        fid: parseFloat(performanceData.fid) || 100,
        ttfb: parseFloat(performanceData.ttfb) || 200
      },
      provider: provider || 'other',
      renewablePercentage: parseFloat(renewablePercentage) || 50,
      pue: parseFloat(pue) || 1.67,
      totalCarbonEmission: parseFloat(totalCarbonEmission) || 0,
      carbonFootprintScore: parseFloat(carbonFootprintScore) || 0,
      energyEfficiencyScore: parseFloat(energyEfficiencyScore) || 0,
      treesNeeded: parseInt(treesNeeded) || 0,
      requestCount: parseInt(requestCount) || 1,
      domainCount: parseInt(domainCount) || 1
    };

    // 生成建议
    const suggestions = generateOptimizationSuggestions(data);

    res.json({ suggestions });
  } catch (error) {
    console.error('建议生成错误:', error);
    res.status(500).json({ error: '建议生成失败', message: error.message });
  }
});

/**
 * 综合网站分析
 */
app.get('/api/analyze', async (req, res) => {
  try {
    const { domain, browser = 'auto' } = req.query;

    if (!domain) {
      return res.status(400).json({ error: '缺少域名参数' });
    }

    // 确保域名格式正确
    let url;
    try {
      if (domain.startsWith('http')) {
        url = new URL(domain).href;
      } else {
        url = `https://${domain}`;
      }
    } catch (error) {
      return res.status(400).json({ error: '无效的域名格式' });
    }

    // 并行执行所有分析任务
```

```
    let [locationData, headerData, sizeData, providerData] = [null, null, null, null];
    let performanceData = null;

    try {
      [locationData, headerData, providerData] = await Promise.all([
        axios.get(`http://localhost:${PORT}/api/location?domain=${domain}`).then(resp => resp.data),
        axios.get(`http://localhost:${PORT}/api/headers?url=${encodeURIComponent(url)}`).then(resp => resp.data),
        axios.get(`http://localhost:${PORT}/api/provider?domain=${domain}`).then(resp => resp.data)
      ]);
    } catch (error) {
      console.error('基础数据获取错误:', error);
    }

    // 单独执行性能测量，因为它可能需要更长时间
    try {
      performanceData = await axios.get(`http://localhost:${PORT}/api/performance?url=${encodeURIComponent(url)}&browser=${browser}`)
        .then(resp => resp.data);

      // 如果有性能数据，使用它的页面大小
      if (performanceData && performanceData.pageSize) {
        sizeData = { size: performanceData.pageSize };
      }
    } catch (error) {
      console.error('性能测量错误:', error);
      performanceData = { measurable: false, error: error.message };
    }

    // 如果没有页面大小数据，尝试单独获取
    if (!sizeData) {
      try {
        sizeData = await axios.get(`http://localhost:${PORT}/api/size?url=${encodeURIComponent(url)}`)
          .then(resp => resp.data);
      } catch (error) {
        console.error('页面大小测量错误:', error);
        sizeData = { measurable: false, error: error.message };
      }
    }

    // 尝试计算碳排放（即使部分数据缺失）
    let carbonData = null;
    try {
      const pageSize = sizeData?.size || 0;
      const country = locationData?.country || null;
      const renewablePercentage = providerData?.renewablePercentage || 50;
      const pue = providerData?.pue || GLOBAL_CONSTANTS.averagePUE;

      // 获取性能数据中的请求数和域名数
      const requestCount = performanceData?.requestCount || 1;
```

```
      const domainCount = performanceData?.domainCount || 1;

      if (pageSize > 0) {
        carbonData = await axios.get(`http://localhost:${PORT}/api/carbon`, {
          params: {
            pageSize,
            country,
            renewablePercentage,
            pue,
            requestCount,
            domainCount
          }
        }).then(resp => resp.data);
      } else {
        carbonData = { measurable: false, error: '缺少页面大小数据' };
      }
    } catch (error) {
      console.error('碳排放计算错误:', error);
      carbonData = { measurable: false, error: error.message };
    }

    // 只有当性能数据可用时才生成优化建议
    let suggestionsData = { suggestions: [] };
    if (performanceData && !performanceData.error) {
      try {
        suggestionsData = await axios.get(`http://localhost:${PORT}/api/suggestions`, {
          params: {
            pageSize: sizeData?.size || 0,
            performance: JSON.stringify(performanceData),
            provider: providerData?.provider || 'other',
            renewablePercentage: providerData?.renewablePercentage || 50,
            pue: providerData?.pue || GLOBAL_CONSTANTS.averagePUE,
            totalCarbonEmission: carbonData?.totalCarbonEmission || 0,
            carbonFootprintScore: carbonData?.carbonFootprintScore || 0,
            energyEfficiencyScore: carbonData?.energyEfficiencyScore || 0,
            treesNeeded: carbonData?.treesNeeded || 0,
            requestCount: performanceData?.requestCount || 1,
            domainCount: performanceData?.domainCount || 1
          }
        }).then(resp => resp.data);
      } catch (error) {
        console.error('建议生成错误:', error);
        suggestionsData = {
          suggestions: ['由于性能数据获取失败，无法生成针对性优化建议。'],
          measurable: false,
          error: error.message
        };
      }
    } else {
      suggestionsData = {
        suggestions: ['由于性能数据获取失败，无法生成针对性优化建议。'],
        measurable: false
      };
```

```
    }

    // 整合所有数据
    const result = {
      domain,
      url,
      browser,
      timestamp: new Date().toISOString(),

      // 位置信息
      location: locationData || { measurable: false },

      // 页面信息
      pageSize: sizeData?.size || 0,
      contentType: sizeData?.contentType || null,
      headers: headerData?.headers || null,

      // 服务提供商信息
      provider: providerData?.provider || 'unknown',
      providerName: providerData?.providerName || 'Unknown Provider',
      renewablePercentage: providerData?.renewablePercentage || null,
      pue: providerData?.pue || null,

      // 性能指标
      performance: performanceData || { measurable: false },

      // 碳排放信息
      energyIntensity: carbonData?.energyIntensity || null,
      dataCenterEnergy: carbonData?.dataCenterEnergy || null,
      transmissionEnergy: carbonData?.transmissionEnergy || null,
      deviceEnergy: carbonData?.deviceEnergy || null,
      dataTransferCarbon: carbonData?.dataTransferCarbon || null,
      networkCarbon: carbonData?.networkCarbon || null,
      clientCarbon: carbonData?.clientCarbon || null,
      totalCarbonEmission: carbonData?.totalCarbonEmission || null,
      monthlyCarbonEmission: carbonData?.monthlyCarbonEmission || null,
      annualCarbonEmission: carbonData?.annualCarbonEmission || null,
      treesNeeded: carbonData?.treesNeeded || null,
      isGreen: carbonData?.isGreen || false,

      // 优化建议
      suggestions: suggestionsData.suggestions || []
    };

    res.json(result);
  } catch (error) {
    console.error('综合分析错误:', error);
    res.status(500).json({
      error: '综合分析失败',
      message: error.message,
      stack: process.env.NODE_ENV === 'development' ? error.stack :
undefined
    });
  }
});
```

```
897
898 /**
899  * 生成智能优化建议
900  * @param {Object} data - 性能和碳排放数据
901  * @returns {Array} 优化建议列表
902  */
903 function generateOptimizationSuggestions(data) {
904   // 如果没有性能数据，返回通用建议
905   if (!data.performance || !data.performance.lcp) {
906     return [
907       '无法获取性能指标，请确保网站可访问并且服务器配置正确',
908       '考虑使用CDN分发静态资源，减少数据传输距离和能耗',
909       '实施高效的HTTP缓存策略，延长缓存有效期减少重复请求',
910       '优化图片资源，考虑使用WebP或AVIF等新一代图片格式'
911     ];
912   }
913
914   const suggestions = [];
915
916   // 基于页面大小的建议
917   if (data.pageSize > 4000) {
918     suggestions.push('大幅压缩图片资源，当前页面大小过大('+data.pageSize+'KB)，严重影响加载速度和能源消耗');
919     suggestions.push('使用WebP或AVIF等新一代图片格式，可减少50-90%的图片大小');
920     suggestions.push('实施延迟加载(Lazy Loading)技术，仅加载可视区域内容');
921   } else if (data.pageSize > 2500) {
922     suggestions.push('压缩图片和媒体资源，减少页面大小('+data.pageSize+'KB)和传输量');
923     suggestions.push('优化JavaScript和CSS文件，减少不必要的代码');
924   } else if (data.pageSize > 1500) {
925     suggestions.push('考虑进一步优化资源大小('+data.pageSize+'KB)，提高页面加载速度');
926   }
927
928   // 基于域名数量的建议
929   if (data.domainCount > 10) {
930     suggestions.push(`网站加载资源来自过多域名(${data.domainCount}个)，建议合并资源来源减少DNS查询和连接建立开销`);
931   } else if (data.domainCount > 5) {
932     suggestions.push(`考虑减少加载资源的域名数量(${data.domainCount}个)，以降低DNS查询时间`);
933   }
934
935   // 基于请求数量的建议
936   if (data.requestCount > 100) {
937     suggestions.push(`网页请求数过多(${data.requestCount}个)，严重影响加载速度，建议合并资源并减少不必要的API调用`);
938   } else if (data.requestCount > 50) {
939     suggestions.push(`网页请求数较多(${data.requestCount}个)，建议通过合并小文件减少HTTP请求`);
940   }
941
942   // 基于性能指标的建议
943   if (data.performance.lcp > 2.5) {
```

```
944       suggestions.push(`优化最大内容绘制(LCP=${safeToFixed(data.performance.lcp, 2)}s)，重点优化主要内容元素的加载时间`);
945       if (data.performance.lcp > 5) {
946         suggestions.push('LCP值严重超标，建议使用预加载(preload)关键资源并优化服务器响应速度');
947       }
948     }
949
950     if (data.performance.cls > 0.1) {
951       suggestions.push(`减少累积布局偏移(CLS=${safeToFixed(data.performance.cls, 3)})，预先设置图片和元素尺寸`);
952       if (data.performance.cls > 0.25) {
953         suggestions.push('CLS值严重超标，检查是否有动态注入内容导致布局偏移，为所有图片和嵌入元素设置明确尺寸');
954       }
955     }
956
957     if (data.performance.ttfb > 300) {
958       suggestions.push(`优化服务器响应时间(TTFB=${Math.round(data.performance.ttfb)}ms)，考虑使用边缘CDN或优化后端处理`);
959       if (data.performance.ttfb > 1000) {
960         suggestions.push('服务器响应时间过长，建议使用服务端缓存，优化数据库查询，或升级服务器配置');
961       }
962     }
963
964     if (data.performance.fid > 130) {
965       suggestions.push(`提高首次输入延迟(FID=${Math.round(data.performance.fid)}ms)，减少主线程阻塞的JavaScript执行`);
966       if (data.performance.fid > 300) {
967         suggestions.push('输入延迟严重，拆分长任务为较小任务，并延迟加载非关键JavaScript');
968       }
969     }
970
971     // 基于碳排放的建议
972     if (data.carbonFootprintScore > 70) {
973       suggestions.push(`当前网站碳足迹评分较高(${Math.round(data.carbonFootprintScore)}分)，建议全面优化能源使用效率`);
974     }
975
976     if (data.energyEfficiencyScore < 50) {
977       suggestions.push(`能源效率评分较低(${Math.round(data.energyEfficiencyScore)}分)，建议采用更现代的Web技术和优化策略`);
978     }
979
980     if (data.renewablePercentage < GLOBAL_CONSTANTS.greenEnergyThreshold) {
981       suggestions.push(`当前服务器使用的可再生能源比例(${data.renewablePercentage}%)偏低，建议迁移至更环保的数据中心`);
982     }
983
984     if (data.pue > 1.5) {
985       suggestions.push(`当前数据中心PUE值(${safeToFixed(data.pue, 2)})较高，选择更高能效的服务提供商可降低碳排放`);
```

```
  }

  if (data.totalCarbonEmission > 1.5) {
    suggestions.push(`当前页面单次访问碳排放(${safeToFixed(data.totalCarbonEmission, 2)}gCO2e)偏高，建议全面优化页面资源`);
    if (data.totalCarbonEmission > 3) {
      suggestions.push('碳排放量远高于平均水平，建议对页面进行全面性能审计并减少不必要的资源加载');
    }
  }

  // 加入碳中和相关建议
  if (data.treesNeeded > 5) {
    suggestions.push(`抵消网站年度碳排放需要种植约${data.treesNeeded}棵树，建议考虑参与碳抵消项目或减少能源消耗`);
  }

  // 通用绿色建议
  if (suggestions.length < 3) {
    suggestions.push('实施高效的HTTP缓存策略，延长缓存有效期减少重复请求');
    suggestions.push('使用CDN分发静态资源，减少数据传输距离和能耗');
    suggestions.push('考虑使用绿色主机服务，选择使用可再生能源的数据中心');
  }

  // 确保建议数量不会太多
  return suggestions.slice(0, 10);
}

/**
 * 检查系统环境
 */
function checkEnvironment() {
  console.log('=== 系统环境信息 ===');
  console.log(`操作系统: ${os.platform()} ${os.release()}`);
  console.log(`Node.js 版本: ${process.version}`);
  console.log(`CPU 架构: ${os.arch()}`);
  console.log(`可用内存: ${Math.round(os.freemem() / 1024 / 1024)} MB / ${Math.round(os.totalmem() / 1024 / 1024)} MB`);
  console.log('=== 环境信息结束 ===');
}

/**
 * 基础版网页分析 - 不依赖浏览器，使用纯HTTP请求
 * @param {string} url - 目标URL
 * @returns {Promise<Object>} 基础性能指标
 */
async function measurePerformanceBasic(url) {
  try {
    const startTime = Date.now();

    // 使用HEAD请求测量TTFB和响应时间
    const headStartTime = Date.now();
    const headResponse = await axiosInstance.head(url);
    const headEndTime = Date.now();
    const ttfb = headEndTime - headStartTime;
```

```
1037
1038      // 获取HTTP头信息
1039      const headers = headResponse.headers;
1040      const contentLength = headers['content-length'];
1041      const contentType = headers['content-type'] || '';
1042      const serverType = headers['server'] || 'unknown';
1043
1044      // 使用GET请求获取完整内容
1045      const getStartTime = Date.now();
1046      const response = await axiosInstance.get(url);
1047      const getEndTime = Date.now();
1048      const totalTime = getEndTime - getStartTime;
1049
1050      // 计算实际页面大小
1051      let pageSize = 0;
1052      if (contentLength) {
1053        pageSize = parseInt(contentLength);
1054      } else {
1055        // 如果没有content-length头，使用响应数据长度
1056        pageSize = Buffer.byteLength(response.data);
1057      }
1058
1059      // 更精确地估算FCP和LCP
1060      // FCP通常是TTFB + DOM解析时间 + 关键资源加载时间
1061      const fcpEstimate = ttfb + Math.min(500, totalTime * 0.3);
1062
1063      // LCP基于总加载时间，但通常早于完全加载完成
1064      const lcpEstimate = Math.min(totalTime, ttfb + totalTime * 0.7);
1065
1066      // CLS估计 - 由于无法精确测量布局稳定性，使用基于HTML大小的启发式方法
1067      let clsEstimate = 0.02; // 默认值
1068      if (pageSize > 500000) clsEstimate = 0.25; // 大型页面可能有更多的布局偏移
1069      else if (pageSize > 200000) clsEstimate = 0.15;
1070      else if (pageSize > 100000) clsEstimate = 0.08;
1071
1072      // FID估计 - 由于无法精确测量交互延迟，使用基于响应时间的启发式方法
1073      let fidEstimate = 50; // 默认值（毫秒）
1074      if (totalTime > 3000) fidEstimate = 200;
1075      else if (totalTime > 1500) fidEstimate = 100;
1076
1077      // 分析HTML提取资源URL
1078      const resourceAnalysis = extractResourceUrls(response.data, url);
1079      const { $, resourceUrls, uniqueDomains } = resourceAnalysis;
1080
1081      // 统计不同类型的资源
1082      const resourceStats = countResourceTypes(resourceUrls);
1083
1084      // 估计总资源大小
1085      const totalResourceSize = estimateResourceSize(resourceStats, pageSize);
1086
1087      // 检查页面是否使用CDN
1088      const cdnInfo = checkCdnUsage(uniqueDomains, headers);
1089
1090      // 计算安全分数
1091      const securityInfo = calculateSecurityScore(headers);
```

```
1092
1093      // 检查页面是否支持HTTPS
1094      const supportsHTTPS = url.startsWith('https://');
1095
1096      // 检查页面是否使用压缩
1097      const supportsCompression = headers['content-encoding'] &&
1098                                  (headers['content-encoding'].includes('gzip')
    ||
1099                                   headers['content-encoding'].includes('br')
    ||
1100                                   headers['content-
    encoding'].includes('deflate'));
1101
1102      // 检查是否使用缓存控制
1103      const cacheControl = headers['cache-control'] || '';
1104      const supportsCaching = cacheControl !== '' &&
1105                              (cacheControl.includes('max-age') ||
1106                               cacheControl.includes('s-maxage') ||
1107                               cacheControl.includes('public'));
1108
1109      // 提取HTML内容质量信息
1110      const contentQuality = analyzeContentQuality($);
1111
1112      // 构建测量结果对象
1113      const performance = {
1114        fcp: fcpEstimate,
1115        lcp: lcpEstimate,
1116        cls: clsEstimate,
1117        fid: fidEstimate,
1118        ttfb: ttfb,
1119        pageSize: pageSize,
1120        totalResourceSize: totalResourceSize,
1121        requestCount: resourceUrls.length + 1, // 加1是因为主HTML请求
1122        domainCount: uniqueDomains.size,
1123        responseTime: totalTime,
1124        resourceStats: resourceStats,
1125        usesHttps: supportsHTTPS,
1126        serverType: serverType,
1127        supportsCompression: supportsCompression,
1128        supportsCaching: supportsCaching,
1129        usesCdn: cdnInfo.usesCdn,
1130        cdnProvider: cdnInfo.cdnProvider,
1131        contentQuality: contentQuality
1132      };
1133
1134      const securityHeaders = {
1135        score: securityInfo.score,
1136        details: securityInfo.details
1137      };
1138
1139      return {
1140        success: true,
1141        performance,
1142        headers: {
1143          supportsCompression,
```

```
        supportsCaching,
        securityHeaders,
        serverType,
        supportsHTTPS
      },
      measurementMethod: 'basic-http'
    };
  } catch (error) {
    console.error('基础性能测量错误:', error);
    return {
      success: false,
      error: `基础性能测量失败: ${error.message}`
    };
  }
}

/**
 * 分析HTML内容质量
 * @param {CheerioStatic} $ - Cheerio对象
 * @returns {Object} 内容质量分析结果
 */
function analyzeContentQuality($) {
  if (!$) return { score: 0 };

  try {
    // 计算文本内容量
    const bodyText = $('body').text().trim();
    const textLength = bodyText.length;

    // 计算图像数量和是否有alt属性
    const images = $('img');
    const imageCount = images.length;
    let imagesWithAlt = 0;

    images.each((i, img) => {
      if ($(img).attr('alt')) imagesWithAlt++;
    });

    // 检查标题层次结构
    const h1Count = $('h1').length;
    const h2Count = $('h2').length;
    const h3Count = $('h3').length;

    // 检查链接质量
    const links = $('a');
    const linkCount = links.length;
    let linksWithText = 0;

    links.each((i, link) => {
      if ($(link).text().trim().length > 0) linksWithText++;
    });

    // 检查元数据
    const hasTitle = $('title').length > 0;
    const hasDescription = $('meta[name="description"]').length > 0;
```

```
1199      const hasKeywords = $('meta[name="keywords"]').length > 0;
1200
1201      // 计算内容质量分数 (0-100)
1202      let score = 50; // 基础分数
1203
1204      // 文本内容加分
1205      if (textLength > 2000) score += 15;
1206      else if (textLength > 1000) score += 10;
1207      else if (textLength > 500) score += 5;
1208
1209      // 图像质量加分
1210      if (imageCount > 0 && imagesWithAlt / imageCount > 0.8) score += 10;
1211      else if (imageCount > 0 && imagesWithAlt / imageCount > 0.5) score += 5;
1212
1213      // 标题结构加分
1214      if (h1Count === 1) score += 5; // 最佳实践是只有一个h1
1215      if (h2Count > 0) score += 5;
1216      if (h3Count > 0) score += 3;
1217
1218      // 链接质量加分
1219      if (linkCount > 0 && linksWithText / linkCount > 0.9) score += 7;
1220
1221      // 元数据加分
1222      if (hasTitle) score += 5;
1223      if (hasDescription) score += 5;
1224      if (hasKeywords) score += 3;
1225
1226      // 确保分数范围在0-100之间
1227      score = Math.min(100, Math.max(0, score));
1228
1229      return {
1230        score,
1231        textLength,
1232        imageCount,
1233        imagesWithAlt,
1234        headingStructure: {
1235          h1Count,
1236          h2Count,
1237          h3Count
1238        },
1239        linkCount,
1240        linksWithText,
1241        metadata: {
1242          hasTitle,
1243          hasDescription,
1244          hasKeywords
1245        }
1246      };
1247    } catch (e) {
1248      console.error('内容质量分析错误:', e);
1249      return { score: 0 };
1250    }
1251  }
1252
1253  /**
```

```
 * 估计资源大小
 * @param {Object} resourceStats - 资源统计
 * @param {number} pageSize - HTML页面大小
 * @returns {number} 估计的总资源大小（字节）
 */
function estimateResourceSize(resourceStats, pageSize) {
  // 不同资源类型的平均大小估计（字节）
  const averageSizes = {
    css: 20000,  // 平均CSS文件约20KB
    js: 80000,   // 平均JS文件约80KB
    images: 200000, // 平均图片约200KB
    fonts: 30000,   // 平均字体约30KB
    other: 10000    // 其他资源约10KB
  };

  let totalSize = pageSize || 0; // HTML大小

  // 计算各类资源的估计总大小
  for (const [type, count] of Object.entries(resourceStats)) {
    if (averageSizes[type]) {
      totalSize += count * averageSizes[type];
    }
  }

  return totalSize;
}

/**
 * HTTP头部分析 - 检查与性能相关的HTTP头
 * @param {string} url - 目标URL
 * @returns {Promise<Object>} 基于头部的指标
 */
async function analyzeHttpHeaders(url) {
  console.log('分析HTTP头部信息...');
  try {
    const response = await axiosInstance.head(url, {
      maxRedirects: 5,
      validateStatus: null
    });

    const headers = response.headers;

    // 检查性能相关的HTTP头
    const hasCompression = headers['content-encoding'] &&
      (headers['content-encoding'].includes('gzip') ||
       headers['content-encoding'].includes('br') ||
       headers['content-encoding'].includes('deflate'));

    const hasCaching = headers['cache-control'] || headers['expires'];

    const securityHeaders = [
      'strict-transport-security',
      'content-security-policy',
      'x-content-type-options',
      'x-frame-options',
```

```
      'x-xss-protection'
    ];

    // 检测存在的安全头
    const presentSecurityHeaders = [];
    securityHeaders.forEach(header => {
      if (headers[header]) {
        presentSecurityHeaders.push(header);
      }
    });

    const securityScore = (presentSecurityHeaders.length / securityHeaders.length) * 100;

    // 分析CDN使用情况
    let usingCDN = false;
    let cdnProvider = null;

    // 常见CDN标识头
    if (headers['server'] &&
        (headers['server'].includes('cloudflare') ||
         headers['server'].includes('akamai') ||
         headers['server'].includes('fastly'))) {
      usingCDN = true;
      cdnProvider = headers['server'].split(' ')[0];
    }

    // 检查CDN特有头
    if (headers['cf-ray'] || headers['cf-cache-status']) {
      usingCDN = true;
      cdnProvider = 'Cloudflare';
    } else if (headers['x-cache'] && headers['x-served-by']) {
      usingCDN = true;
      cdnProvider = 'Fastly/Varnish';
    } else if (headers['x-amz-cf-id']) {
      usingCDN = true;
      cdnProvider = 'Amazon CloudFront';
    } else if (headers['x-azure-ref']) {
      usingCDN = true;
      cdnProvider = 'Azure CDN';
    }

    return {
      hasCompression,
      hasCaching,
      usingCDN,
      cdnProvider,
      securityScore,
      securityHeaders: presentSecurityHeaders,
      server: headers['server'] || 'unknown',
      headers: headers
    };
  } catch (error) {
    console.error('HTTP头分析失败:', error);
    throw error;
```

```
  }
}

// 所有其他路由返回前端应用
const serveIndex = (req, res) => {
  res.sendFile(path.join(__dirname, '../dist/index.html'));
};

app.get('*', serveIndex);

// 启动服务器
app.listen(PORT, async () => {
  console.log(`GreenWeb服务器运行在端口 ${PORT}`);
  console.log(`访问 http://localhost:${PORT} 以使用应用`);
});

/**
 * 从HTML中提取资源URL
 * @param {string} html - HTML内容
 * @param {string} baseUrl - 基础URL
 * @returns {Object} 提取的资源URL和分析结果
 */
function extractResourceUrls(html, baseUrl) {
  try {
    const cheerio = require('cheerio');
    const $ = cheerio.load(html);
    const resourceUrls = [];
    const uniqueDomains = new Set();

    // 获取URL的域名部分
    const getUrlDomain = (url) => {
      try {
        if (!url || typeof url !== 'string') return '';
        const urlObj = new URL(url, baseUrl);
        return urlObj.hostname;
      } catch (e) {
        return '';
      }
    };

    // 提取域名
    const baseDomain = getUrlDomain(baseUrl);

    // 处理URL
    const processUrl = (url) => {
      if (!url) return null;
      try {
        // 处理相对URL
        const absoluteUrl = new URL(url, baseUrl).href;
        const domain = getUrlDomain(absoluteUrl);
        if (domain) {
          uniqueDomains.add(domain);
        }
        return absoluteUrl;
      } catch (e) {
```

```
        return null;
      }
    };

    // 提取CSS链接
    $('link[rel="stylesheet"]').each((_, el) => {
      const url = processUrl($(el).attr('href'));
      if (url) resourceUrls.push({ url, type: 'css' });
    });

    // 提取JavaScript
    $('script[src]').each((_, el) => {
      const url = processUrl($(el).attr('src'));
      if (url) resourceUrls.push({ url, type: 'js' });
    });

    // 提取图片
    $('img[src]').each((_, el) => {
      const url = processUrl($(el).attr('src'));
      if (url) resourceUrls.push({ url, type: 'images' });
    });

    // 提取背景图像从内联样式
    $('[style*="background"]').each((_, el) => {
      const style = $(el).attr('style');
      if (style) {
        const match = style.match(/url\(['"]?([^'")]+)['"]?\)/);
        if (match && match[1]) {
          const url = processUrl(match[1]);
          if (url) resourceUrls.push({ url, type: 'images' });
        }
      }
    });

    // 提取字体
    $('link[rel="preload"][as="font"]').each((_, el) => {
      const url = processUrl($(el).attr('href'));
      if (url) resourceUrls.push({ url, type: 'fonts' });
    });

    return {
      $,
      resourceUrls,
      uniqueDomains
    };
  } catch (error) {
    console.error('提取资源URL错误:', error);
    return {
      $: null,
      resourceUrls: [],
      uniqueDomains: new Set()
    };
  }
}

```

```
/**
 * 统计不同类型的资源
 * @param {Array} resourceUrls - 资源URL数组
 * @returns {Object} 资源类型统计
 */
function countResourceTypes(resourceUrls) {
  const stats = {
    css: 0,
    js: 0,
    images: 0,
    fonts: 0,
    other: 0
  };

  resourceUrls.forEach(resource => {
    if (stats[resource.type] !== undefined) {
      stats[resource.type]++;
    } else {
      stats.other++;
    }
  });

  return stats;
}

/**
 * 检查CDN使用情况
 * @param {Set} uniqueDomains - 唯一域名集合
 * @param {Object} headers - HTTP响应头
 * @returns {Object} CDN使用信息
 */
function checkCdnUsage(uniqueDomains, headers) {
  const cdnProviders = {
    'cloudflare': ['cloudflare', 'cloudflare-nginx', 'cloudfront.net'],
    'akamai': ['akamai', 'akamaiedge.net', 'akamaized.net'],
    'fastly': ['fastly'],
    'cloudfront': ['cloudfront.net'],
    'vercel': ['vercel-edge', 'vercel.app'],
    'netlify': ['netlify', 'netlify.app']
  };

  // 检查响应头中是否包含CDN信息
  let cdnProvider = 'unknown';
  let usesCdn = false;

  // 检查常见CDN响应头
  if (headers['server']) {
    for (const [provider, keywords] of Object.entries(cdnProviders)) {
      if (keywords.some(keyword =>
headers['server'].toLowerCase().includes(keyword))) {
        cdnProvider = provider;
        usesCdn = true;
        break;
      }
    }
```

```
1527    }
1528
1529    // 检查CDN特定头部
1530    if (!usesCdn && headers['cf-ray']) {
1531      cdnProvider = 'cloudflare';
1532      usesCdn = true;
1533    } else if (!usesCdn && headers['x-fastly-request-id']) {
1534      cdnProvider = 'fastly';
1535      usesCdn = true;
1536    } else if (!usesCdn && headers['x-amz-cf-id']) {
1537      cdnProvider = 'cloudfront';
1538      usesCdn = true;
1539    } else if (!usesCdn && headers['x-vercel-cache']) {
1540      cdnProvider = 'vercel';
1541      usesCdn = true;
1542    } else if (!usesCdn && headers['x-nf-request-id']) {
1543      cdnProvider = 'netlify';
1544      usesCdn = true;
1545    }
1546
1547    // 检查域名中是否包含CDN信息
1548    if (!usesCdn) {
1549      const domains = Array.from(uniqueDomains);
1550      for (const domain of domains) {
1551        for (const [provider, keywords] of Object.entries(cdnProviders)) {
1552          if (keywords.some(keyword => domain.includes(keyword))) {
1553            cdnProvider = provider;
1554            usesCdn = true;
1555            break;
1556          }
1557        }
1558        if (usesCdn) break;
1559      }
1560    }
1561
1562    return {
1563      usesCdn,
1564      cdnProvider
1565    };
1566  }
1567
1568  /**
1569   * 计算安全分数
1570   * @param {Object} headers - HTTP响应头
1571   * @returns {Object} 安全分数和详情
1572   */
1573  function calculateSecurityScore(headers) {
1574    let score = 0;
1575    const details = {};
1576
1577    // 检查常见安全响应头
1578    const securityHeaders = {
1579      'strict-transport-security': { score: 20, name: '严格传输安全' },
1580      'content-security-policy': { score: 20, name: '内容安全策略' },
1581      'x-content-type-options': { score: 10, name: '内容类型选项' },
```

```
    'x-frame-options': { score: 10, name: '框架选项' },
    'x-xss-protection': { score: 10, name: 'XSS保护' },
    'referrer-policy': { score: 10, name: '引用策略' },
    'permissions-policy': { score: 10, name: '权限策略' },
    'feature-policy': { score: 10, name: '功能策略' }
  };

  for (const [header, info] of Object.entries(securityHeaders)) {
    if (headers[header]) {
      score += info.score;
      details[header] = {
        present: true,
        value: headers[header],
        name: info.name
      };
    } else {
      details[header] = {
        present: false,
        name: info.name
      };
    }
  }

  return {
    score: Math.min(93, score),
    details
  };
}

// 添加安全的toFixed函数，避免对null/undefined调用toFixed
function safeToFixed(value, digits = 2) {
  if (value === null || value === undefined) return '0.00';
  return Number(value).toFixed(digits);
}
```