```javascript
/**
 * 网站分析服务
 * 用于向目标网站发送请求并获取性能指标和位置信息
 */

import axios from 'axios';

// 获取当前环境的基础URL
function getBaseUrl() {
  // 在生产环境中，API和前端在同一域下运行
  // 在开发环境中，API在localhost:3000上运行
  const isProd = import.meta.env.PROD;
  return isProd ? '/api' : 'http://localhost:3000/api';
}

// 代理服务器URL
const API_URL = getBaseUrl();

/**
 * 解析URL获取完整域名
 * @param {string} url - 输入的URL或域名
 * @returns {string} 处理后的完整URL
 */
function parseUrl(url) {
  if (!url) return '';

  // 如果没有协议，添加https://
  if (!url.startsWith('http://') && !url.startsWith('https://')) {
    url = 'https://' + url;
  }

  try {
    const parsedUrl = new URL(url);
    return parsedUrl.href;
  } catch (error) {
    console.error('URL解析错误:', error);
    return '';
  }
}

/**
 * 获取网站服务器位置
 * @param {string} domain - 目标域名
 * @returns {Promise<Object>} 服务器位置信息
 */
export async function getServerLocation(domain) {
  try {
    const response = await axios.get(`${API_URL}/location`, {
      params: { domain }
    });
    return response.data;
  } catch (error) {
    console.error('获取服务器位置失败:', error);
```

```
    // 失败时返回错误信息
    return {
      error: '无法获取服务器位置信息',
      details: error.message,
      measurable: false
    };
  }
}

/**
 * 获取网站HTTP响应头信息
 * @param {string} url - 目标URL
 * @returns {Promise<Object>} HTTP响应头信息
 */
export async function getHttpHeaders(url) {
  try {
    const fullUrl = parseUrl(url);
    const response = await axios.get(`${API_URL}/headers`, {
      params: { url: fullUrl }
    });
    return response.data;
  } catch (error) {
    console.error('获取HTTP头信息失败:', error);
    return {
      error: '无法获取HTTP头信息',
      details: error.message,
      measurable: false
    };
  }
}

/**
 * 测量网站页面大小
 * @param {string} url - 目标URL
 * @returns {Promise<number>} 页面大小（KB）
 */
export async function measurePageSize(url) {
  try {
    const fullUrl = parseUrl(url);
    const response = await axios.get(`${API_URL}/size`, {
      params: { url: fullUrl }
    });
    return response.data;
  } catch (error) {
    console.error('测量页面大小失败:', error);
    // 失败时返回错误信息
    return {
      error: '无法测量页面大小',
      details: error.message,
      measurable: false
    };
  }
}

/**
```

```
 * 测量网站性能指标
 * @param {string} url - 目标URL
 * @param {string} [browser='auto'] - 使用的浏览器，可选值：'auto'(自动)、'chrome'、'edge'
 * @returns {Promise<Object>} 性能指标
 */
export async function measurePerformance(url, browser = 'auto') {
  let retries = 0;
  const maxRetries = 2;
  const retryDelay = 3000; // 重试间隔3秒

  const attemptMeasure = async () => {
    try {
      const fullUrl = parseUrl(url);
      console.log(`尝试测量性能指标 (使用${browser === 'auto' ? '自动选择浏览器' : browser}) (尝试 ${retries + 1}/${maxRetries + 1}): ${fullUrl}`);

      const response = await axios.get(`${API_URL}/performance`, {
        params: {
          url: fullUrl,
          browser: browser // 将浏览器参数传递给后端
        },
        timeout: 120000 // 增加超时时间到120秒，因为Lighthouse分析可能需要较长时间
      });

      console.log(`性能指标测量成功 (使用${response.data.measuredBy || '未知方法'}):`, response.data);
      return response.data;
    } catch (error) {
      if (axios.isAxiosError(error)) {
        console.error(`性能测量失败 (尝试 ${retries + 1}/${maxRetries + 1}):`, error.message);

        if (error.response) {
          // 服务器返回了错误状态码
          console.error('服务器错误:', error.response.status, error.response.data);
          return {
            error: `性能测量失败: ${error.response.data.message || '服务器错误'}`,
            details: error.response.data,
            measurable: false
          };
        } else if (error.request) {
          // 请求已经发出，但没有收到响应
          console.error('未收到响应:', error.request);
          if (retries < maxRetries) {
            retries++;
            console.log(`等待 ${retryDelay}ms 后重试...`);
            await new Promise(resolve => setTimeout(resolve, retryDelay));
            return attemptMeasure();
          }
          return {
            error: '性能测量超时，请检查网络连接或网站可访问性',
            details: '请求已发出但未收到响应',
```

```
            measurable: false
          };
        } else {
          // 设置请求时发生了错误
          if (retries < maxRetries) {
            retries++;
            console.log(`等待 ${retryDelay}ms 后重试...`);
            await new Promise(resolve => setTimeout(resolve, retryDelay));
            return attemptMeasure();
          }
          return {
            error: '无法测量性能指标',
            details: error.message,
            measurable: false
          };
        }
      } else {
        // 非Axios错误
        console.error('性能测量过程中发生非网络错误:', error);
        return {
          error: '无法测量性能指标',
          details: error.message,
          measurable: false
        };
      }
    }
  };

  return attemptMeasure();
}

/**
 * 分析网站服务器提供商
 * @param {string} domain - 目标域名
 * @returns {Promise<Object>} 服务提供商信息
 */
export async function analyzeProvider(domain) {
  try {
    const response = await axios.get(`${API_URL}/provider`, {
      params: { domain }
    });
    return response.data;
  } catch (error) {
    console.error('分析服务提供商失败:', error);
    return {
      error: '无法分析服务提供商',
      details: error.message,
      measurable: false
    };
  }
}

/**
 * 综合分析网站（使用后端的单一接口）
 * @param {string} domain - 目标域名
```

```
 * @param {string} [browser='auto'] - 使用的浏览器，可选值：'auto'(自动)、'chrome'、'edge'
 * @returns {Promise<Object>} 分析结果
 */
export async function analyzeWebsite(domain, browser = 'auto') {
  try {
    // 使用后端的综合分析端点
    const response = await axios.get(`${API_URL}/analyze`, {
      params: {
        domain,
        browser // 添加浏览器参数
      },
      timeout: 120000 // 增加超时时间到120秒
    });

    return response.data;
  } catch (error) {
    console.error('网站分析失败:', error);
    return {
      error: '网站分析失败',
      details: error.message,
      measurable: false
    };
  }
}

/**
 * 单独调用，不使用综合接口（用于测试和调试）
 * @param {string} domain - 目标域名
 * @param {string} [browser='auto'] - 使用的浏览器，可选值：'auto'(自动)、'chrome'、'edge'
 * @returns {Promise<Object>} 分析结果
 */
export async function analyzeWebsiteSeparately(domain, browser = 'auto') {
  try {
    // 准备完整URL
    const url = parseUrl(domain);
    if (!url) throw new Error('无效的域名');

    // 并行执行所有请求以提高性能
    const [locationData, headersData, sizeData, performanceData, providerData] = await Promise.all([
      getServerLocation(domain),
      getHttpHeaders(url),
      measurePageSize(url),
      measurePerformance(url, browser), // 传递浏览器参数
      analyzeProvider(domain)
    ]);

    // 检查是否有任何请求失败
    if (locationData.error || headersData.error || sizeData.error ||
        performanceData.error || providerData.error) {
      const errors = [];
      if (locationData.error) errors.push(locationData.error);
      if (headersData.error) errors.push(headersData.error);
```

```
      if (sizeData.error) errors.push(sizeData.error);
      if (performanceData.error) errors.push(performanceData.error);
      if (providerData.error) errors.push(providerData.error);

      return {
        url,
        domain,
        browser, // 添加使用的浏览器信息
        error: '部分数据获取失败',
        details: errors.join('; '),
        measurable: false,
        location: locationData,
        provider: providerData.provider || 'unknown',
        pageSize: sizeData.size,
        performance: performanceData,
        headers: headersData.headers
      };
    }

    return {
      url,
      domain,
      browser, // 添加使用的浏览器信息
      provider: providerData.provider,
      location: locationData,
      pageSize: sizeData.size,
      performance: performanceData,
      headers: headersData.headers
    };
  } catch (error) {
    console.error('网站分析失败:', error);
    return {
      error: '网站分析失败',
      details: error.message,
      measurable: false
    };
  }
}

/**
 * 分析网站碳排放
 * @param {Object} params - 碳排放分析参数
 * @param {number} params.pageSize - 页面大小(KB)
 * @param {string} params.country - 服务器所在国家代码
 * @param {number} params.requestCount - 请求数量
 * @param {number} params.domainCount - 域名数量
 * @param {number} [params.renewablePercentage] - 可再生能源使用百分比
 * @param {number} [params.pue] - 数据中心PUE
 * @returns {Promise<Object>} 碳排放分析结果
 */
export async function analyzeCarbonEmission(params) {
  try {
    const response = await axios.get(`${API_URL}/carbon`, { params });
    return response.data;
  } catch (error) {
```

```
    console.error('碳排放分析失败:', error);
    return {
      error: '无法分析碳排放',
      details: error.message,
      measurable: false,
      totalCarbonEmission: 0,
      monthlyCarbonEmission: 0,
      annualCarbonEmission: 0,
      isGreen: false
    };
  }
}

export default {
  analyzeWebsite,
  analyzeWebsiteSeparately,
  getServerLocation,
  measurePageSize,
  measurePerformance,
  analyzeProvider,
  getHttpHeaders,
  analyzeCarbonEmission
};
```