

```
1  /**
2   * 网站分析服务
3   * 用于向目标网站发送请求并获取性能指标和位置信息
4   */
5
6  import axios from 'axios';
7
8  // 获取当前环境的基础URL
9  function getBaseUrl() {
10   // 在生产环境中，API和前端在同一域下运行
11   // 在开发环境中，API在localhost:3000上运行
12   const isProd = import.meta.env.PROD;
13   return isProd ? '/api' : 'http://localhost:3000/api';
14 }
15
16 // 代理服务器URL
17 const API_URL = getBaseUrl();
18
19 /**
20 * 解析URL获取完整域名
21 * @param {string} url - 输入的URL或域名
22 * @returns {string} 处理后的完整URL
23 */
24 function parseUrl(url) {
25   if (!url) return '';
26
27   // 如果没有协议，添加https://
28   if (!url.startsWith('http://') && !url.startsWith('https://')) {
29     url = 'https://' + url;
30   }
31
32   try {
33     const parsedUrl = new URL(url);
34     return parsedUrl.href;
35   } catch (error) {
36     console.error('URL解析错误:', error);
37     return '';
38   }
39 }
40
41 /**
42 * 获取网站服务器位置
43 * @param {string} domain - 目标域名
44 * @returns {Promise<Object>} 服务器位置信息
45 */
46 export async function getServerLocation(domain) {
47   try {
48     const response = await axios.get(`${API_URL}/location`, {
49       params: { domain }
50     });
51     return response.data;
52   } catch (error) {
53     console.error('获取服务器位置失败:', error);
```

```
54     // 失败时返回错误信息
55     return {
56         error: '无法获取服务器位置信息',
57         details: error.message,
58         measurable: false
59     };
60 }
61 }
62
63 /**
64  * 获取网站HTTP响应头信息
65  * @param {string} url - 目标URL
66  * @returns {Promise<Object>} HTTP响应头信息
67  */
68 export async function getHttpHeaders(url) {
69     try {
70         const fullUrl = parseUrl(url);
71         const response = await axios.get(`${API_URL}/headers`, {
72             params: { url: fullUrl }
73         });
74         return response.data;
75     } catch (error) {
76         console.error('获取HTTP头信息失败:', error);
77         return {
78             error: '无法获取HTTP头信息',
79             details: error.message,
80             measurable: false
81         };
82     }
83 }
84
85 /**
86  * 测量网站页面大小
87  * @param {string} url - 目标URL
88  * @returns {Promise<number>} 页面大小 (KB)
89  */
90 export async function measurePageSize(url) {
91     try {
92         const fullUrl = parseUrl(url);
93         const response = await axios.get(`${API_URL}/size`, {
94             params: { url: fullUrl }
95         });
96         return response.data;
97     } catch (error) {
98         console.error('测量页面大小失败:', error);
99         // 失败时返回错误信息
100         return {
101             error: '无法测量页面大小',
102             details: error.message,
103             measurable: false
104         };
105     }
106 }
107
108 /**
```

```

109  * 测量网站性能指标
110  * @param {string} url - 目标URL
111  * @param {string} [browser='auto'] - 使用的浏览器, 可选值: 'auto'(自
    动)、'chrome'、'edge'
112  * @returns {Promise<Object>} 性能指标
113  */
114  export async function measurePerformance(url, browser = 'auto') {
115      let retries = 0;
116      const maxRetries = 2;
117      const retryDelay = 3000; // 重试间隔3秒
118
119      const attemptMeasure = async () => {
120          try {
121              const fullUrl = parseUrl(url);
122              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}): ${fullUrl}`);
123
124              const response = await axios.get(`${API_URL}/performance`, {
125                  params: {
126                      url: fullUrl,
127                      browser: browser // 将浏览器参数传递给后端
128                  },
129                  timeout: 120000 // 增加超时时间到120秒, 因为Lighthouse分析可能需要较长时间
130              });
131
132              console.log(`性能指标测量成功 (使用${response.data.measuredBy} || '未知方
    法'):`), response.data);
133              return response.data;
134          } catch (error) {
135              if (axios.isAxiosError(error)) {
136                  console.error(`性能测量失败 (尝试 ${retries + 1}/${maxRetries + 1}):`,
    error.message);
137
138                  if (error.response) {
139                      // 服务器返回了错误状态码
140                      console.error('服务器错误:', error.response.status,
    error.response.data);
141                      return {
142                          error: `性能测量失败: ${error.response.data.message} || '服务器错
    误'`,
143                          details: error.response.data,
144                          measurable: false
145                      };
146                  } else if (error.request) {
147                      // 请求已经发出, 但没有收到响应
148                      console.error('未收到响应:', error.request);
149                      if (retries < maxRetries) {
150                          retries++;
151                          console.log(`等待 ${retryDelay}ms 后重试 ...`);
152                          await new Promise(resolve => setTimeout(resolve, retryDelay));
153                          return attemptMeasure();
154                      }
155                      return {
156                          error: '性能测量超时, 请检查网络连接或网站可访问性',
157                          details: '请求已发出但未收到响应',

```

```

158         measurable: false
159     };
160 } else {
161     // 设置请求时发生了错误
162     if (retries < maxRetries) {
163         retries++;
164         console.log(`等待 ${retryDelay}ms 后重试 ...`);
165         await new Promise(resolve => setTimeout(resolve, retryDelay));
166         return attemptMeasure();
167     }
168     return {
169         error: '无法测量性能指标',
170         details: error.message,
171         measurable: false
172     };
173 }
174 } else {
175     // 非Axios错误
176     console.error('性能测量过程中发生非网络错误:', error);
177     return {
178         error: '无法测量性能指标',
179         details: error.message,
180         measurable: false
181     };
182 }
183 }
184 };
185
186 return attemptMeasure();
187 }
188
189 /**
190  * 分析网站服务器提供商
191  * @param {string} domain - 目标域名
192  * @returns {Promise<Object>} 服务提供商信息
193  */
194 export async function analyzeProvider(domain) {
195     try {
196         const response = await axios.get(`${API_URL}/provider`, {
197             params: { domain }
198         });
199         return response.data;
200     } catch (error) {
201         console.error('分析服务提供商失败:', error);
202         return {
203             error: '无法分析服务提供商',
204             details: error.message,
205             measurable: false
206         };
207     }
208 }
209
210 /**
211  * 综合分析网站（使用后端的单一接口）
212  * @param {string} domain - 目标域名

```

```
213 * @param {string} [browser='auto'] - 使用的浏览器，可选值：'auto'(自
动)、'chrome'、'edge'
214 * @returns {Promise<Object>} 分析结果
215 */
216 export async function analyzeWebsite(domain, browser = 'auto') {
217   try {
218     // 使用后端的综合分析端点
219     const response = await axios.get(`${API_URL}/analyze`, {
220       params: {
221         domain,
222         browser // 添加浏览器参数
223       },
224       timeout: 120000 // 增加超时时间到120秒
225     });
226
227     return response.data;
228   } catch (error) {
229     console.error('网站分析失败:', error);
230     return {
231       error: '网站分析失败',
232       details: error.message,
233       measurable: false
234     };
235   }
236 }
237
238 /**
239 * 单独调用，不使用综合接口（用于测试和调试）
240 * @param {string} domain - 目标域名
241 * @param {string} [browser='auto'] - 使用的浏览器，可选值：'auto'(自
动)、'chrome'、'edge'
242 * @returns {Promise<Object>} 分析结果
243 */
244 export async function analyzeWebsiteSeparately(domain, browser = 'auto') {
245   try {
246     // 准备完整URL
247     const url = parseUrl(domain);
248     if (!url) throw new Error('无效的域名');
249
250     // 并行执行所有请求以提高性能
251     const [locationData, headersData, sizeData, performanceData,
providerData] = await Promise.all([
252       getServerLocation(domain),
253       getHttpHeaders(url),
254       measurePageSize(url),
255       measurePerformance(url, browser), // 传递浏览器参数
256       analyzeProvider(domain)
257     ]);
258
259     // 检查是否有任何请求失败
260     if (locationData.error || headersData.error || sizeData.error ||
performanceData.error || providerData.error) {
261       const errors = [];
262       if (locationData.error) errors.push(locationData.error);
263       if (headersData.error) errors.push(headersData.error);
264
```

```

265     if (sizeData.error) errors.push(sizeData.error);
266     if (performanceData.error) errors.push(performanceData.error);
267     if (providerData.error) errors.push(providerData.error);
268
269     return {
270         url,
271         domain,
272         browser, // 添加使用的浏览器信息
273         error: '部分数据获取失败',
274         details: errors.join('; '),
275         measurable: false,
276         location: locationData,
277         provider: providerData.provider || 'unknown',
278         pageSize: sizeData.size,
279         performance: performanceData,
280         headers: headersData.headers
281     };
282 }
283
284 return {
285     url,
286     domain,
287     browser, // 添加使用的浏览器信息
288     provider: providerData.provider,
289     location: locationData,
290     pageSize: sizeData.size,
291     performance: performanceData,
292     headers: headersData.headers
293 };
294 } catch (error) {
295     console.error('网站分析失败:', error);
296     return {
297         error: '网站分析失败',
298         details: error.message,
299         measurable: false
300     };
301 }
302 }
303
304 /**
305  * 分析网站碳排放
306  * @param {Object} params - 碳排放分析参数
307  * @param {number} params.pageSize - 页面大小(KB)
308  * @param {string} params.country - 服务器所在国家代码
309  * @param {number} params.requestCount - 请求数量
310  * @param {number} params.domainCount - 域名数量
311  * @param {number} [params.renewablePercentage] - 可再生能源使用百分比
312  * @param {number} [params.pue] - 数据中心PUE
313  * @returns {Promise<Object>} 碳排放分析结果
314  */
315 export async function analyzeCarbonEmission(params) {
316     try {
317         const response = await axios.get(`${API_URL}/carbon`, { params });
318         return response.data;
319     } catch (error) {

```

```
320     console.error('碳排放分析失败:', error);
321     return {
322       error: '无法分析碳排放',
323       details: error.message,
324       measurable: false,
325       totalCarbonEmission: 0,
326       monthlyCarbonEmission: 0,
327       annualCarbonEmission: 0,
328       isGreen: false
329     };
330   }
331 }
332
333 export default {
334   analyzeWebsite,
335   analyzeWebsiteSeparately,
336   getServerLocation,
337   measurePageSize,
338   measurePerformance,
339   analyzeProvider,
340   getHttpHeaders,
341   analyzeCarbonEmission
342 };
```