Amazon Host IP Address Updater

Amazon Host IP Address Updater是一个Node.js应用程序，它可以通过读取配置文件、执行网络请求和解析响应来获取Amazon主机的IP地址，并将这些信息更新到README文件中，其中包含最新的IP地址列表和更新时间，以便用户能够获取最新的IP地址列表。它Amazon主机的IP地址。该应用程序能够处理多个URL，同时支持并发请求，并在必要时进行重试。

package.json

```json
{
  "name": "Amazon Host IP Address Updater",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.6.7",
    "ping": "^0.4.4",
    "winston": "^3.11.0"
  },
  "nodemonConfig": {
    "ignore": [
      "dist/*",
      "README.md",
      "error.log",
      "amazon_urls.txt"
    ]
  },
  "devDependencies": {
    "@types/node": "^16.11.7"
  },
  "engines": {
    "node": ">=14.0.0"
  }
}
```

index.js

```javascript
// Import necessary libraries
const winston = require('winston');
const { format, transports } = winston;
const { combine, timestamp, printf } = format;
const fs = require('fs').promises;
const axios = require('axios');
const ping = require('ping');
const { promisify } = require('util');
const path = require('path');

const SCRIPT_DIR = __dirname;
const README_PATH = path.join(SCRIPT_DIR, 'README.md');
const README_TEMPLATE_PATH = path.join(SCRIPT_DIR, 'README_template.md');
const HOSTS_PATH = path.join(SCRIPT_DIR, '/dist/hosts');
const HOSTS_JSON_PATH = path.join(SCRIPT_DIR, '/dist/hosts.json');
const AMAZON_URLS_FILE = path.join(SCRIPT_DIR, 'amazon_urls.txt');

const PING_TIMEOUT = 2000; // in milliseconds
const MAX_ATTEMPTS = 5;
const CONCURRENT_REQUESTS = 5;

// Configure logging
const logger = winston.createLogger({
  level: 'info',
  format: combine(
      timestamp(),
      printf(({ level, message, timestamp }) => {
          return `${timestamp} ${level}: ${message}`;
      })
  ),
  transports: [
      new transports.Console(),
      new transports.File({ filename: 'error.log', level: 'error' }),
  ],
});

async function readAmazonUrls() {
  const fileContent = await fs.readFile(AMAZON_URLS_FILE, 'utf-8');
  return fileContent
    .split('\n')
    .map((line) => line.trim())
    .filter(Boolean);
}
```

```
async function writeReadmeFile(hostsContent, updateTime) {
  const outputDocFilePath = README_PATH;
  const templatePath = README_TEMPLATE_PATH;
  await writeHostFile(hostsContent);

  try {
    await fs.access(outputDocFilePath);
    const oldContent = await fs.readFile(outputDocFilePath, 'utf-8');
    if (oldContent) {
        const oldHosts = oldContent.split('```bash')[1].split('```')[0].trim();
        const oldUpdateTime = oldContent.split('# Update time:')[1].trim().split('\n')[0];
        const hostsContentHosts = hostsContent.split('# Update time:')[0].trim();

        if (oldHosts === hostsContentHosts && oldUpdateTime === updateTime) {
            logger.info('Host not changed');
            return false;
        }
    }
} catch (error) {
    // File doesn't exist or error reading file
}

  const templateStr = await fs.readFile(templatePath, 'utf-8');
  const updatedHostsContent = templateStr.replace('{hosts_str}', hostsContent).replace('{update

  await fs.writeFile(outputDocFilePath, updatedHostsContent, 'utf-8');
  return true;
}

async function writeHostFile(hostsContent) {
  const outputFilePath = HOSTS_PATH;
  await fs.writeFile(outputFilePath, hostsContent, 'utf-8');
}

async function writeJsonFile(hostsList) {
  const outputFilePath = HOSTS_JSON_PATH;
  await fs.writeFile(outputFilePath, JSON.stringify(hostsList), 'utf-8');
}

async function retryIfResultNone(fn) {
  let result = await fn();
  let attempt = 1;

  while (result === null && attempt <= MAX_ATTEMPTS) {
    logger.error(`Attempt ${attempt} failed. Retrying...`);
```

```
    await sleep(1000); // wait for 1 second before retrying
    result = await fn();
    attempt++;
  }

  return result;
}

async function getIpFromApi(amazonUrl, headers) {
  let trueIp = null;

  for (let attempt = 1; attempt <= MAX_ATTEMPTS; attempt++) {
    try {
      const response = await axios.get(`http://ip-api.com/json/${amazonUrl}?lang=zh-CN`, {
        headers,
        timeout: 5000,
      });

      const data = response.data;

      if (
        data.status === 'success' &&
        data.query.match(/\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b/) &&
        data.query.match(/\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b/).length === 1
      ) {
        trueIp = data.query;
        break;
      }
    } catch (error) {
      logger.error(`Error querying ${amazonUrl}: ${error.message}`);
    }
  }

  return trueIp;
}

async function queryIpApi(amazonUrl) {
  try {
    const amazonIps = await promisify(require('dns').resolve)(amazonUrl);
    return amazonIps;
  } catch (error) {
    logger.error(`Error querying ${amazonUrl} by Socket. Trying to get IP from API: ${error.mes
    try {
      const cAmazonIps = await retryIfResultNone(() =>
        getIpFromApi(amazonUrl, {
          'user-agent':
```

```
          'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Ch
        Host: 'ip-api.com',
      })
    );
    return cAmazonIps;
  } catch (error) {
    logger.error(`Error querying ${amazonUrl} by API: ${error.message}`);
    return null;
  }
 }
}

async function getBestIp(ipList) {
  let bestIp = '';
  let minMs = PING_TIMEOUT;

  for (const ip of ipList) {
    const pingResult = await ping.promise.probe(ip, { timeout: PING_TIMEOUT });

    if (pingResult.time === PING_TIMEOUT) {
      // Timeout, consider IP invalid
      continue;
    } else {
      if (pingResult.time < minMs) {
        minMs = pingResult.time;
        bestIp = ip;
      }
    }
  }

  return bestIp;
}

async function getIp(amazonUrl) {
  return await queryIpApi(amazonUrl);
}

async function processUrl(amazonUrl, verbose) {
  try {
    const ip = await getIp(amazonUrl); // Await the result here
    const bestIp = await getBestIp(ip); // Await the result here

    if (verbose) {
      logger.info(`Processed ${amazonUrl}`);
    }
    return [bestIp || '#'.padEnd(30), amazonUrl];
  } catch (error) {
    logger.error(`Error processing ${amazonUrl}: ${error.message}`);
    return ['', amazonUrl];
  }
}

async function main(verbose = false) {
  const start_time = Date.now();
  const amazonUrls = await readAmazonUrls();
  const contentList = await processUrls(amazonUrls, verbose);
  const end_time = Date.now();

  if (!contentList.length) {
    logger.warn('No valid content obtained.');
    return;
  }

  const content = contentList
    .map(([ip, url]) => `${ip}${ip !== '#' ? ' '.repeat(30 - ip.length) : ''}${url}`)
    .join('\n');

  const update_time = new Date().toISOString();

  const hasChange = await writeReadmeFile(content, update_time);

  if (hasChange) {
    await writeJsonFile(contentList);
  }

  if (verbose) {
    logger.info(content);
    logger.info(`End script. Time taken: ${(end_time - start_time) / 1000} seconds.`);
  }
}

async function processUrls(urls, verbose) {
  const contentList = [];
  const tasks = urls.map((url) => processUrl(url, verbose));

  for (const result of await Promise.all(tasks)) {
    if (result) {
      contentList.push(result);
    }
  }
  return contentList;
}

main(true);
```