Proceedings of the 2nd International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/131/2024.20796

ModuleAnalyst: A Small But Useful Tool

Haochen Liu

Tianjin University, Tianjin, China

liuhaochen_@tju.edu.cn

Abstract. Many coding beginners of JavaScript, which has a bunch of third-party modules, do not know well about the structure of that module and have no idea how to use it. This paper presents a useful tool called Module-analyst, which can extract from a module and then represent its interface. Module-analyst based on existing functions such as "object.getOwnPropertyNames" in JavaScript, etc. And this tool managed to make the output of those functions more understandable and readable

Keywords: JavaScript modules, File system, Utilities

1. Introduction

There are various practical modules in Javascript; people can install what they want and have a better coding experience [1]. But first, we need to figure out how many interfaces are there in this library, their names, and the required parameters. If there is an overview of these interfaces, that would be great. That inspired my interest in developing a tool to fulfill those needs. In fact, there are already some functions that can describe modules' basic information. Like "getOwnProperty Names" in JavaScript and "dir()" in Python[2-3], but the former can only show names of interfaces; the latter can show names and return types of interfaces with an unspecified object. In that case, The work draws on the development ideas of other Javascript tools[4-9] and optimizes existing functions, which are mentioned before, to make the result clearer. First, I want to show the type of the interface so people can have a more accurate understanding of them; then, I indicate the meaning of each field in the results; they are name, signature, and type, respectively. Finally, in order to evaluate this work, I tested several popular modules of Javascript respectively to see whether they work well

2. Literature Review

2.1. Design and Implementation of JavaScript API Automation Testing Solution Management

Brief description: This article developed a testing program based on the Robotium framework, which captures the Logcat information of the tested program, then parses and writes the Logcat, and finally writes the solution and tests it[10]. In the application, the JavaScript API serves as a bridge between the program and the web page, expanding the ability of JavaScript embedded in the web page and bringing a better experience to the web application.

Comment: In this article, the author's team tested 26 APIs to demonstrate the effectiveness of the study. Like my tool, our core idea is to automate the processing of JS interfaces while using specific instances to prove that our results are truly effective.

DOI: 10.54254/2755-2721/131/2024.20796

2.2. Research on Implementing JavaScript Code Unit Testing Using Console Statements

Brief description: Error messages, Ajax calls, performance analysis results, and command-line execution results during JavaScript code execution will all be displayed on the console interface [11]. The author used Firebug on the Firefox browser to output information from running JavaScript code to the Firebug console. By flexibly using console statements, unit testing of JavaScript code can be easily achieved.

In this study, the author printed the process of executing JS statements on the terminal console and debugged the program better based on the results, which inspired me that extract the JS module structure and print the result out may offer a better way to understand the module structure

3. Running Example: Doing Something

An Example that shows how the tool works and the result. I will demonstrate the operation and results of my software. The first thing is to copy Modulus_analyst.exe to the same directory as the module folder you want to analyze (usually named node_modules). Then click Modulus_analyst.exe to execute it. You will see a terminal like this:

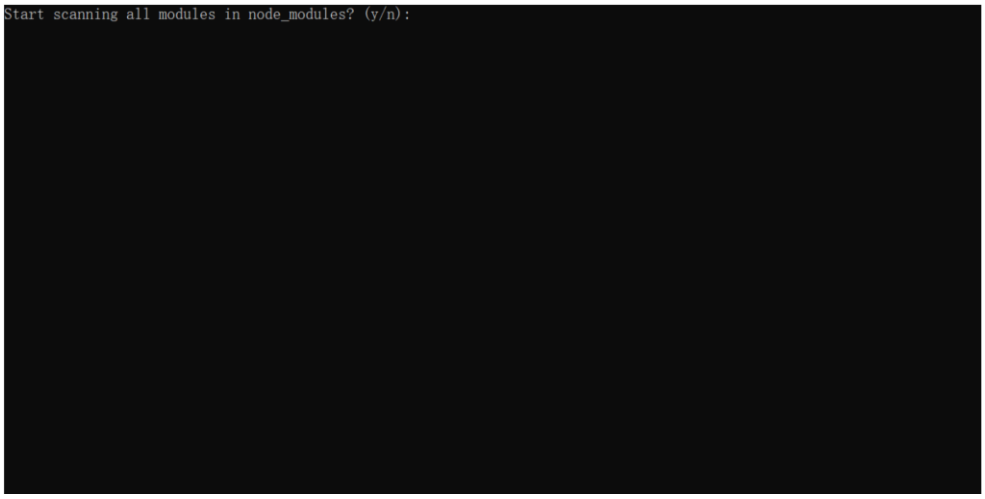

Figure 1. Start using

If y is entered, the software will automatically scan the modules installed in the node_modules directory. Otherwise, if n is entered, the program will exit. Here is the part output of the scanning:

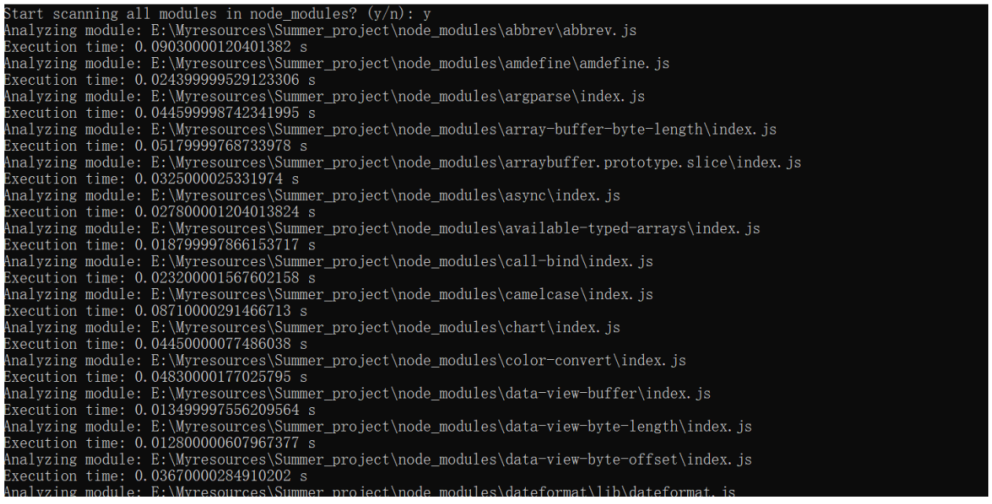

Figure 2. Scanning result

DOI: 10.54254/2755-2721/131/2024.20796

So this is the first step of using this tool. Next, there is a question to inquire whether to output all scanned modules' results; similarly, enter y to see the full result while entering n to do something else. It should be noted that whether to show full results, the module list will be offered in order to maintain readability and usability.

Figure 3. The full results

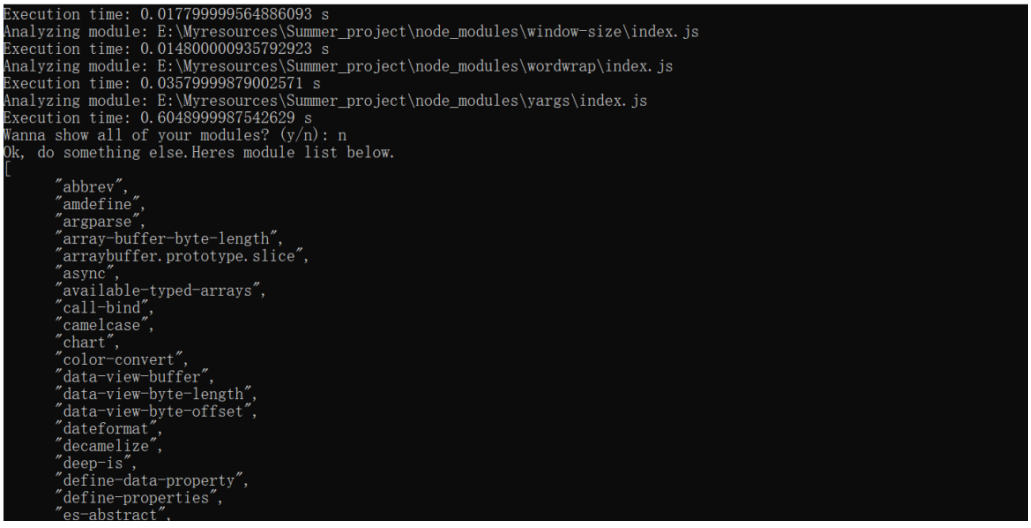

Figure 4. Hiding the full results

Then finally here is the main function of this tool. Select the module to be analyzed from the scanned module list and output its specific information. Here are the results of two popular modules, date format and word wrap as examples:

DOI: 10.54254/2755-2721/131/2024.20796

Figure 5. Results of two modules

4. Background Why develop this tool?

According to Octoverse: The state of open source and rise of AI in 2023, which was published by the GitHub community, In 2023, there will be 6 million Javascript users. Plus, There is a notable rising in this language's users. What matters most is that Javascript is the top popular language all over the world. In that case, there will be more and more coders who want to learn this language, thus, this tool may help them.

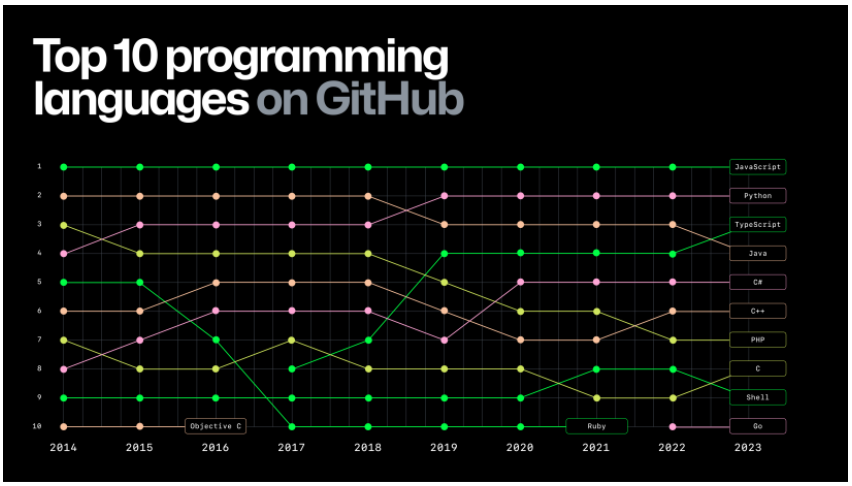

Figure 6. Statistical chart of the number of users

5. Design and Methods

This tool is based on Javascript, and it is quite simple, with five functions and one main function. The main working principle of this software is to scan module files in the node_- modules folder and save the results, and output analysis results for different modules according to users' needs. Here will present the source code of five functions and their main features, along with a brief explanation:

This tool is based on Javascript, and it is quite simple, with five functions and one main function. The main working principle of this software is to scan module files in the node_- modules folder and save the results, and output analysis results for different modules according to users' needs.

Here will present the source code of five functions and their main features, along with a brief explanation :

DOI: 10.54254/2755-2721/131/2024.20796

getSignature():

```
// Get the signatures (parameters) of the module
function getSignature(func) {
    if (func) {
        const funcString = func.toString();
        const signature = funcString.substring(funcString.indexOf('('),
            funcString.indexOf(')') + 1);
        return signature;
```

This function is mainly used to convert the name of the function to a string and then slice the parameter section; if it is not a function, then the result would be undefined.

Analyze():

```
// Get each property and its type from the moduleObj
// and get the signature by executing getSignature function
function Analyze(moduleObj) {
    const Property_Names = Object.getOwnPropertyNames(moduleObj);
    const moduleInterface = {};
    Property_Names.forEach(propertyName => {
        const property = moduleObj[propertyName];
        moduleInterface[propertyName] = {
            name: propertyName,
            signature: getSignature(property),
            type: typeof property
        };
    });
    Allmodule.push(moduleInterface);
```

This function gets all the properties of the module object and output an object containing the property name, type, and (if the property is a function) its signature as a result. Then, add this object to the Allmodule which saves all modules' results. If the property is not a function, its signature will be an empty string or undefined.

LoadNAnalyze():

```
// Load the modules and save their names, execution times
async function LoadNAnalyze(modulePath) {
    try {
        const parts = path.resolve(modulePath).split(path.sep);
        // Get the name of modules(consider different situations)
        let modulename = parts[parts.length - 2];
        if(modulename == 'lib'){
            modulename=parts[parts.length - 1].replace('.js', '');
        }
        //Save the name into 'Allname'
        Allname.push(modulename)
        const moduleObj = require(modulePath);
        console.log(`Analyzing module: ${modulePath}`);
        // Calculate the running time
        const startTime=performance.now()
        Analyze(moduleObj);
        const endTime=performance.now()
        console.log(`Execution time: ${endTime - startTime} s`);
    } catch (error) {
        console.error(`Failed to load module from
        ${modulePath}:`, error.message);
        Allmodule.push("Null")
    }
}
```

This function first loads the module from the given module path, then analyzes the properties and methods of the module object, identifies the name, type, and method signature (parameter list) of each property, and stores this information in an object.

After that, this object is added to the Allmodule to display all results to the user. If the module cannot be loaded, an error message will be output and a null will be added to Allmodule

scanModules():

```
// Read the contents of the module directory
async function scanModules() {
    //Get the absolute path of the node_comodules directory
    const nodeModulesPath = path.resolve(process.cwd(), 'node_modules');
    const folders = await fs.readdir(nodeModulesPath);
    // Search in every folder
    for (const folder of folders) {
        const modulePath = path.join(nodeModulesPath, folder);
        //Get the status of every path and judge whether it is a directory
        const stats = await fs.stat(modulePath);
        if (stats.isDirectory()) {
            // consider different situations of modules' different saving files
            const potentialMainFile = path.join(modulePath,'index.js');
            if (await fs.access(potentialMainFile)
                .then(() => true).catch(() => false)) {
                await LoadNAnalyze(potentialMainFile);
            } else {
                const packageJsonPath = path.join(modulePath,'package.json');
                if (await fs.access(packageJsonPath)
                    .then(() => true).catch(() => false)) {
                    const data = await fs.readFile(packageJsonPath,'utf8');
                    try {
                        const packageJson = JSON.parse(data);
                        const mainFile = packageJson.main || 'index.js';
                        const mainFilePath = path.join(modulePath,mainFile);
                        if (await fs.access(mainFilePath)
                            .then(() => true).catch(() => false)) {
```

DOI: 10.54254/2755-2721/131/2024.20796

```
            await LoadNAnalyze(mainFilePath );

        } else {
            const libPath = path.join(modulePath ,' lib '); if (await fs.access(libPath)
                .then (() => true).catch (() => false)) { const files = await fs.readdir(libPath ); const jsFiles = files.
                filter(file =>file.endsWith('.js ')); if (jsFiles.length > 0) {
                    //Since generally ,
                    // the first file would be index.js const libFilePath = path. join(libPath , jsFiles [0]);
                    await LoadNAnalyze(libFilePath );
                }
            }
        }
    } catch (AnalyzeError) {
        console.log( ' Failed to analyze package.json: '
            , AnalyzeError.message );
```

The scanModules function searches for the Node.js folder and loads all available modules. It first reads all subdirectories under the node_modules directory and then scans each subdirectory to discover if there are any entry files (such as index.js or the main.js file specified in package_json). If a valid entry file is found, it will call the LoadNAnalyze function just explained to load and analyze the module. If the module directory contains a lib folder and there are .js files in the lib folder, it will also attempt to load and analyze these files. However, for modules with unique structures, this method may fail and report an error that the module cannot be found.

This function may be the most difficult to implement, as the structure of each module is different, so the properties and methods that can be called will be saved to different files in different paths. The only thing that could be done is to consider multiple situations here to ensure availability, that is, to ensure the tool could analyze the structure of most modules.

askQuestion() And Main():

```
1 //Ask users questions
2 async function askQuestion(query) {
  3 return new Promise(resolve => {
        readline.question(query , answer => {
            resolve(answer );
        });
    });
8 }
9 async function Main() {
10      const scanAnswer = await askQuestion
11      ( ' Start scanning all modules in node_modules? (y/n): ');
12      if (scanAnswer.trim (). toLowerCase () === 'y ') {
13          await scanModules ();
14      } else {
15          console.log(' Exited successfully . ');
16          readline.close ();
17          process.exit (0);
18      } 19
20      const showModulesAnswer = await askQuestion
21      ( ' Wanna show all of your modules? (y/n): ');
22      if (showModulesAnswer.trim (). toLowerCase () === 'y ') {
23          console.log(Allmodule );
24          console.log( ' Heres module list below. ');
25          console.log(JSON.stringify(Allname , null , 6));
26      } else {
27          console.log( ' Ok, do something else.Heres module list below. ');
28          console.log(JSON.stringify(Allname , null , 6));
29      }
30      while (1){
31          const analyzeModuleAnswer = await askQuestion
32          ( ' Which module would you like to analyze? Press q to quit: '); 33
34          if (analyzeModuleAnswer.trim (). toLowerCase () === 'q ') {
35              console.log(' Exited successfully . ');
36              readline.close ();
37              process.exit (0);
38          } else {
39              const index = Allname.indexOf(analyzeModuleAnswer );
40              if (index !== -1) {
41                  console.log(Allmodule[index ]);
42              } else {
43                  console.log( ' Module not found. ');
44              }
45          } 46
47      }
```

The two functions here implement interaction with the user. For convenience, I have encapsulated the function read-line. question to ask the user questions and wait for input.

Pack the code to a .exe file: Finally, use pkg to package the source code with pkg and automatically generate an exe file. The command line is as follows: pkg-twin

....js –output ..._analyst.exe:

6. Results and Discussion

Since it's a tool based on existing methods and has optimized them, there should be an evaluation method here to assess whether these optimizations are effective [12-15].

1, Correctness: Whether the result of program execution correctly includes all interfaces and their characteristics

2, Usability: If the tool is easy to use and its results are straightforward and intuitive.

3, Generality: This tool could be used in any module successfully

DOI: 10.54254/2755-2721/131/2024.20796

4, Speed: The results will be quickly returned after processing

So What are the performances?

Correctness: This tool can scan all the properties and methods of each module. However, as mentioned in section A, the structure of different modules is different, so some modules may not be recognized correctly. This is also an area that the tool will focus on improving in the future

Usability: As shown in the section 'Design and Implementation,' users only need to set the .exe file and the folder to be analyzed (which must be named the default Node.js) in the same directory, start the software, and enter the corresponding command to analyze. The drawback is that having a visual interface (such as developing one with Vue) would be even better.

Generality: This tool can scan most of the modules in- stalled by the user, and the software will also remove or add the analysis results of the corresponding modules after the user installs and deletes them.

Speed: The program only takes about dozens to 100 milliseconds to run and can provide real-time answers. Here are five samples and their running times.

7. Conclusion

This article describes a module analysis tool written in Javascript language, mainly based on the file structure and traversal of common Javascript modules, and discusses the ideas of classification. The built-in module analysis tool in JavaScript has limited functionality and cannot scan and output the modules installed by the user. And there are drawbacks such as unclear output results and the need to open an IDE to write code when using it.

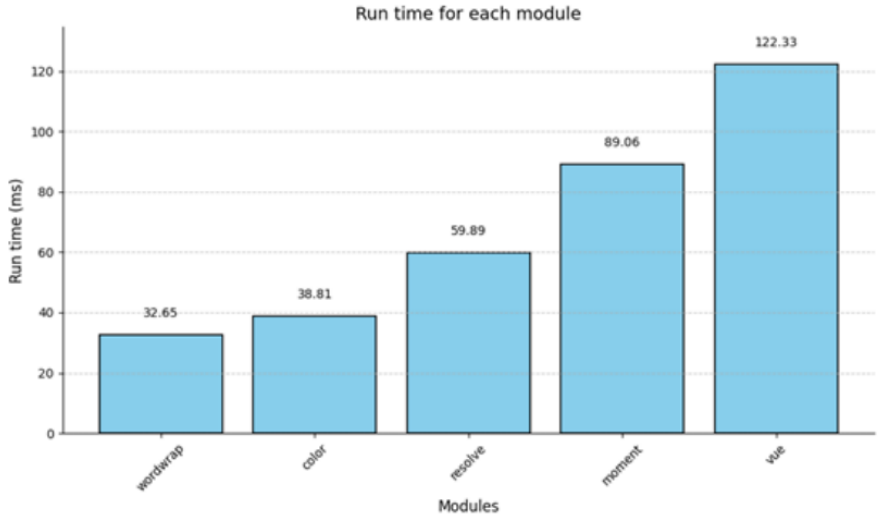

Figure 7. Different run time for different modules

The advantage of this analysis tool is that it can scan all the modules installed by users and output the module names for users' reference, and it is easy to use, just follow the prompts to operate. Can ensure that novice programmers with limited programming experience can use it.

The disadvantage of this tool is that it lacks a visual interactive interface and cannot scan and recognize modules with different structures, which will become a focus of future research

Module-analyst's implementation are all open source and available for download: https://github.com/lhc2002/Module- analyst.

DOI: 10.54254/2755-2721/131/2024.20796

References

[1] Kang, S (Kang, Seonghoon) ; Ryu, S (Ryu, Sukyoung) Formal Specification of a JavaScript Module System [M].ACM International Conference on Object Oriented Programming Systems Languages and Applications

[2] Fan Xiang. Analysis of Web Development Techniques in Software Engineering [J]. Information and Computer (Theoretical Edition), 2022, 34 (21): 139-141

[3] Dr. Anupam Sharma; Archit Jain; Ayush Bahuguna; Deeksha Dinkar, A Simple Comparison Between Java Python and Nodejs in Web Development | [J] Journal of Research in Science and Engineering. Volume 3 , Issue 8 . 2021

[4] Wang Tingting, Shen Qijie JavaScript debugger soft- ware architecture [J]. Journal of Natural Sciences, Hunan Normal University, 2014, 37 (06): 88-92

[5] KONG W; JING Y; MA N; LV Q JavaScript file testing method, involves obtaining pre-established testing tool, compiling testing tool to obtain execution file corresponding to test tool, and calling execution file to obtain to-be-tested JavaScript file:CHN,CN114371996-A [P] 19 Apr 2022

[6] Pan Congxiang, Jiang Letian. JavaScript API Design for TurtleBot Robot Based on Node.js [J]. Information Technology, 2018, (03): 89-91. DOI: 10.13274/j.cnki. hdzj. 2018.03.018

[7] Liu Shengjian, Luo Lin, Du Jian Research on Efficient Distributed Real time Applications of Node.js [J]. Software Guide, 2014,13 (12): 25-28

[8] Chen Y., NodeSRT: A Selective Regression Testing Tool for Node.js Application Journal | [J] Proceedings - International Conference on Software Engineering. Volume, Issue. 2021. PP 126-128

[9] Chen Binbin, Li Xuesong, Li Ziyang, et al. Design of Grass Livestock Balance Data Middleware Based on Node.js [J]. Geospatial Information, 2024, 22 (03): 83-86

[10] Zhang Lingfen Management Design and Implementation of JavaScript API Automation Testing Scheme [J]. Electronic Design Engineering, 2016,24 (02): 35-37+41. DOI: 10.14022/j.cnki.dzsjgc.2016.02.011

[11] Dong Ning, Wang Bo. Research on Implementing JavaScript Code Unit Testing Using Console Statements [J]. Software Guide, 2017, 16 (02): 13-15

[12] Janthong, P (Janthong, Pannawat); Suwannasart, T (Suwannasart, Taratip) Tool for Generating Test Module for JavaScript Based on Statement Coverage Criteria [M].2014 11TH INTERNATIONAL JOINT CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING (JCSSE)331- 336

[13] Xiao Dongping, Li Linglin. Design and Implementation of Performance Testing Scheme in Software Systems [J]. Enterprise Technology Development, 2014, 33 (13): 21-22. DOI: 10.14165/j.cnki.Hunansci.2014.13.029

[14] Chen Jiangyong, Xu Li, Zhang Hui, etc Design and Implementation of Web Automation Testing Framework [J]. Journal of Fujian Normal University (Natural Science Edition), 2013, 29 (04): 39-45

[15] Dai Jingjing, Dong Shujian Exploration of JavaScript Local Call Methods [J]. Computer Programming Techniques and Maintenance, 2023, (10): 33-34+43. DOI: 10.16184/j.cnki.comprg.2023.10.048