Comparison Between Webpack and Broswerify

Jiahui Ruan

NetId: jr58

Introduction

At the beginning of this web development class, we use HTML5/CSS/JavaScript to create our web application, which is mostly the same as the beginning of modern web development industry. This method is easy, but it has lots of problems. For example, when we build a form to transfer data back and forth between the client and server, every time the client app calls the server, the server renders a new HTML page. This triggers a page refresh in the browser. (1) There is no doubt that this method becomes slower and slower as the content of the web pages goes larger and larger. To avoid this problem and reduce the time for refreshing, people create a term Single-Page-Applications (SPAs), which are web apps that load a single HTML page and dynamically update that page as the user interacts with the app. SPAs use AJAX and HTML5 to create fluid and responsive Web apps, without constant page reloads. These AJAX calls return data—not markup—usually in JSON format. The app uses the JSON data to update the page dynamically, without reloading the page.

Another problem rises with the popularity of SPAs. They tend to rely on numerous hefty libraries. (2) There are multiple strategies on how to deal with loading them. You could load them all at once. You could also consider loading libraries as you need them. You may think that you can load your modules in a series of <script> tags in the HTML, but it can be problematic for two reasons: 1) It forces you to manage dependencies by ensuring your script tags appear in the proper order, and makes it cumbersome to manage complex dependency graphs. 2) With regular <script> tags, you cannot control [script] loading and executing behavior reliably cross-browser. (3) The first major solution to these problem is called RequireJS, which is the loaders of AMD specification. But AMD modules also has its own problem: the syntax makes people hard to read and it is hard to update the dependencies of modules since you have to specify them as an array of string that results in parameters to a callback function. (4)

So, people invent and develop two strategies to solve the above problem. They are Broswerify and Webpack. In this article, I am going to talk about what and are, their difference and why we prefer Webpack rather than Browserify. In 2015, RequireJS is one of 3 major options on the module loading scene, along with Browserify and Webpack. But in 2016, people are moving past the RequireJS.

What is Browserify?

In order to solve the two major drawbacks of AMD specification and it loaders RequireJS, people use other module syntax like CommonJS and the ES6. One of the biggest difference between these two is that CommonJS use require like Node.js and

ES6 use import. Browserify is an attempt to build a module loader on top of the NPM ecosystem and node modules. It uses CommonJS modules and integrates tightly with NPM. It provides a way to bundle CommonJS modules together. The Browserify ecosystem is composed of a lot of small modules. In this way, Browserify adheres to the Unix philosophy. Browserify is a little easier to adopt than Webpack, and is, in fact, a good alternative to it. For conclusion, the advancement of Browserify are mostly based on RequireJS: 1)Simplifies many preprocessing tasks with its transform system 2) Solves the same problems regarding asynchronous loading addressed by RequireJS 3) Opens the doors to the vast and growing NPM ecosystem

What is Webpack?

If you were a web tools developer, you must noticed that Browserify has some limitation: 1) it does not support ES6 or AMD, if we want to use ES6, we have to use some tools or plugins to translate our code into ES6. 2) It can not handle assets like CSS and preprocessors of them. People began to build a more monolithic tool to solve the problem. Webpack seeks to unify Javascript module syntaxes and provide tools for a full swath of static asset management tasks. It imposes no restrictions on your choice of module syntax, and offers full support for Javascript, CSS, and even image preprocessing. Webpack will traverse through the require statements of your project and will generate the bundles you have defined. The loader mechanism works for CSS as well and @import is supported. There are also plugins for specific tasks, such as minification, localization, hot loading, and so on.

Why we prefer Webpack and conclusion

Browserify and Webpack both provide servers for development, allowing you to instantly integrate changes without a long build process or extra JavaScript files loaded into the browser. RequireJS doesn’t have a great solution for development workflow, requiring you to either do a full build every time, or load a copy of the RequireJS JavaScript file up to your browser where it loads files from the client. This has disadvantages both in terms of reliability (you’ll be running the files in a different way in development than on production, and there will be timing differences due to the need to load the files on the client), and configuration (you’ll need to have separate configurations for development and production that will have to be kept in sync somehow).

Browserify and Webpack also provide a clean syntax for preprocessing things like CoffeeScript or JSX files, using transforms or loaders (respectively). These transforms can be applied to a subset of files with configuration. These methods compare favorably to RequireJS’s plugin system, which require you to manually specify the plugins used each time you load a resource.

Webpack is a much more comprehensive tool: 1) Hot Module Replacement(HMR): refresh the browser automatically as you make changes. 2) Bundle Splitting: Aside from the HMR feature, Webpack's bundling capabilities are extensive. It allows you to split bundles in various ways. You can even load them dynamically as your application gets

executed. 3) Asset Hashing: With Webpack, you can easily inject a hash to each bundle name. This allows you to invalidate bundles on the client side as changes are made. Bundle splitting allows the client to reload only a small part of the data in the ideal case. 4) Loaders and Plugins: If you are missing something, there are loaders and plugins available that allow you to go further. Webpack is a very impressive and inclusive tool. In webpack 2, it even allows the new technology called SystemJS, which pushes package management directly to the browser. All in all, even though Webpack might not be the final solution of the management of bundlers in JS. It it the best solution we can use now. And since the JS technology update so fast, we have to improve ourselves everyday and keep an open mined to new technology in order to make things easier.

Reference:

1. https://msdn.microsoft.com/en-us/magazine/dn463786.aspx
2. http://survivejs.com/webpack/webpack-compared/
3. https://scotch.io/tutorials/getting-started-with-browserify
4. http://benmccormick.org/2015/05/28/moving-past-requirejs/