Webpack - Pros and Cons

Overview 1

Advantages 3

Disadvantages 7

Sneak Peek at Webpack Internals 7

Sample Configuration for Webpack 10

Alternatives 11

Conclusion 12

Overview

Webpack is a static module bundler for Javascript applications. It comprises an event driven plug-in based compiler.

It features an extensible loader system which allows to pre-process and bundle the non Javascript assets such as CSS, Images, HTML etc.

The ultimate goal of webpack is to unify all these different sources and module types in a way that's possible to import everything in our JavaScript code, and finally produce a shippable output.

Webpack is extremely modular. What makes Webpack great is that it lets plugins inject themselves into more places in the build process when compared to alternatives like browserify and requirejs. Many things that may seem built into the core are just plugins that are loaded by default and can be overridden (i.e. the CommonJS require() parser).

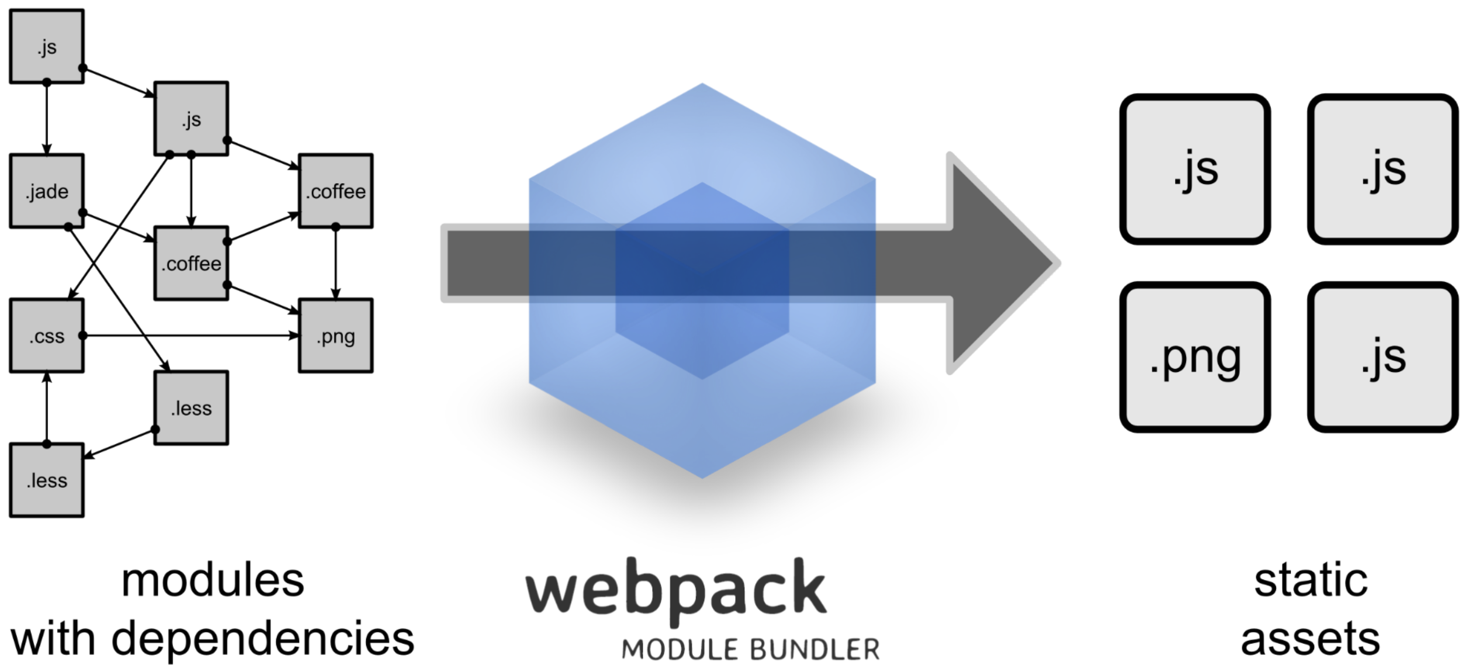
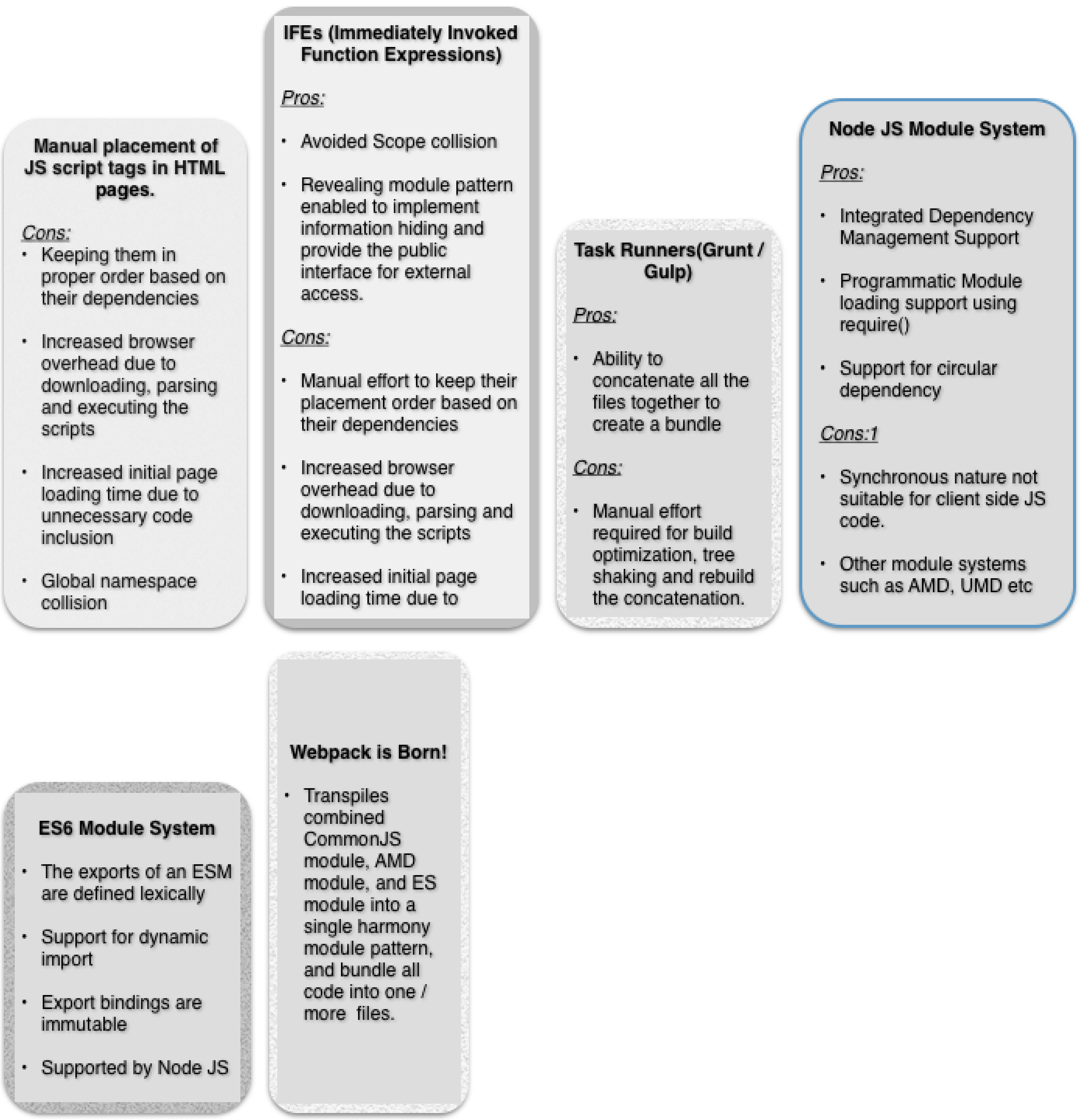


Image courtesy: https://devdocs.prestashop.com/1.7/themes/getting-started/asset-management/webpack/

Evolution of JS module system led to the birth of Module bundlers:



Manual placement of JS script tags in HTML pages.

Cons:

- Keeping them in proper order based on their dependencies
- Increased browser overhead due to downloading, parsing and executing the scripts
- Increased initial page loading time due to unnecessary code inclusion
- Global namespace collision

IFEs (Immediately Invoked Function Expressions)

Pros:

- Avoided Scope collision
- Revealing module pattern enabled to implement information hiding and provide the public interface for external access.

Cons:

- Manual effort to keep their placement order based on their dependencies
- Increased browser overhead due to downloading, parsing and executing the scripts
- Increased initial page loading time due to

Task Runners(Grunt / Gulp)

Pros:

- Ability to concatenate all the files together to create a bundle

Cons:

- Manual effort required for build optimization, tree shaking and rebuild the concatenation.

Node JS Module System

Pros:

- Integrated Dependency Management Support
- Programmatic Module loading support using require()
- Support for circular dependency

Cons:1

- Synchronous nature not suitable for client side JS code.
- Other module systems such as AMD, UMD etc

ES6 Module System

- The exports of an ESM are defined lexically
- Support for dynamic import
- Export bindings are immutable
- Supported by Node JS

Webpack is Born!

- Transpiles combined CommonJS module, AMD module, and ES module into a single harmony module pattern, and bundle all code into one / more files.

Advantages

1.Vibrant ecosystem of Plugins / modules(loaders)

Plugins allow to intercept webpack's execution through hooks. Webpack itself has been implemented as a collection of plugins. Underneath it relies on a tappable plugin interface that allows webpack to apply plugins in different ways.

Plugins are usually installed using npm and then added to the webpack configuration file so they can be used as a part of webpack’s bundling process.

Loaders allow you determine what should happen when webpack's module resolution mechanism encounters a file.
A loader definition consists of conditions based on which to match and actions that should be performed when a match happens.

2.Support Flexible Module systems (ES6, CommonJS, AMD, UMD etc)

3. Ability to recompile only the module which gets updated

4. Provision for CLI utility access

5. Support for Module federation (a new method of sharing code between frontend applications)

Starting from webpack 5, there's built-in functionality to develop micro frontends. Module federation and gives you enough functionality to tackle the workflow required by the micro frontend approach.

6. Hot module replacement

Webpack uses express middleware to reload the updated module in real-time in the browser.
By enabling the HMR feature, refreshing the browser in order to download a new bundle becomes unnecessary. The HMR runtime will accept incoming requests from the HMR server that contains the manifest file and chunks of code that will replace the current one in the browser.

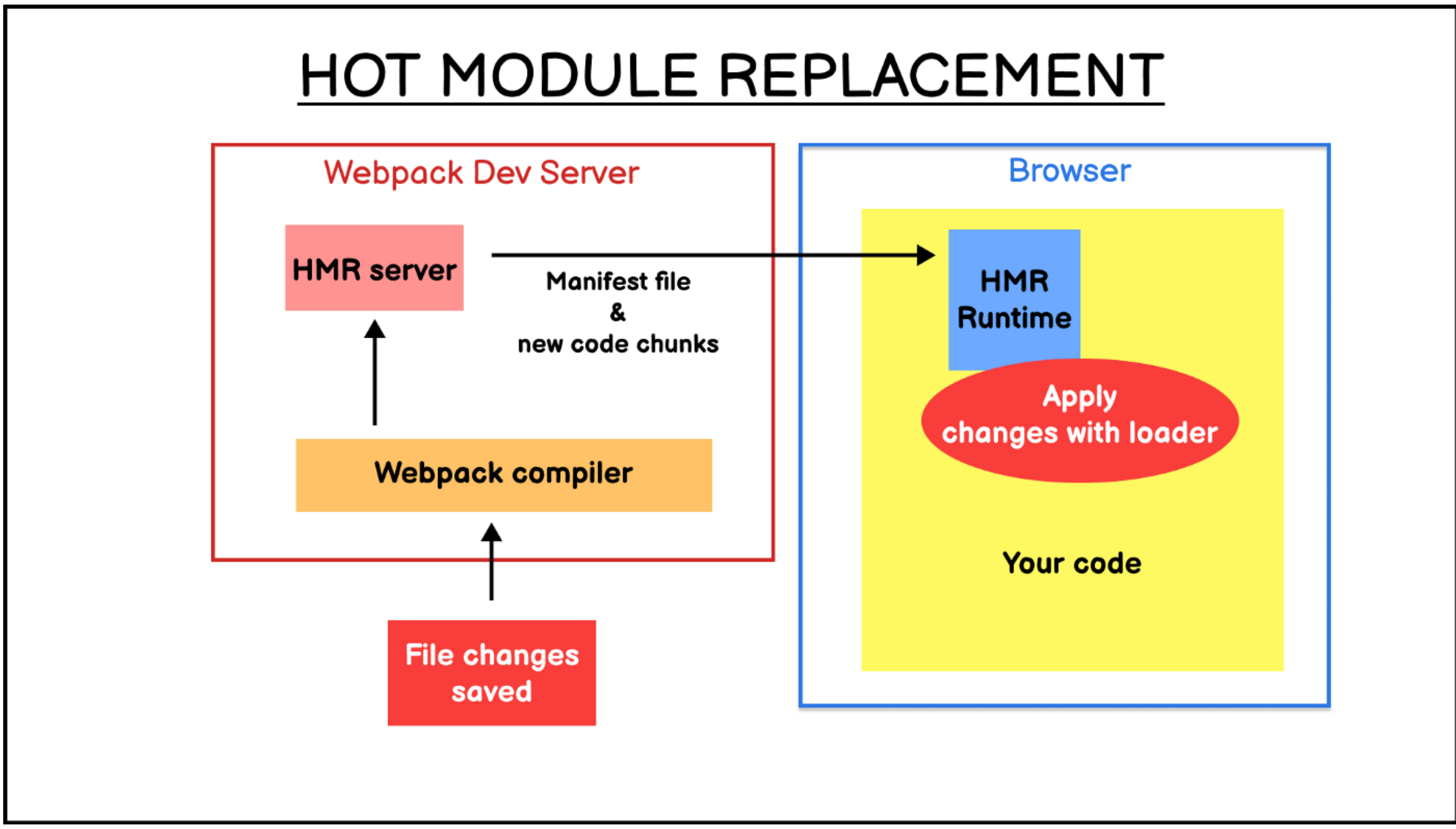


Image courtesy: https://blog.bitsrc.io/webpacks-hot-module-replacement-feature-explained-43c13b169986

7. Tree Shaking

Tree shaking is a feature enabled by the ES2015 module definition. The idea is that given it's possible to analyse the module definition statically without running it, webpack can tell which parts of the code are being used and which are not. It's possible to verify this behaviour by expanding the application and adding code there that should be eliminated.

8. Code splitting

It enables us to load more code as the user enters a new view of the application. We can also tie loading to a specific action like scrolling or clicking a button. We could also try to predict what the user is trying to do next and load code based on our guess. This way, the functionality would be already there as the user tries to access it.

9. Multi Entry Support

We could define all the different entries to generate many output files. It is great to control exactly the generated bundles by viewing it directly in our entry definition.

Starting from webpack 5, it's possible to define bundle splitting using entries.

10. Dev Server

webpack-dev-server tool — A Node.js server (need to install it separately)

11. Scope hoisting

Since webpack 4, it applies scope hoisting in production mode by default. It hoists all modules to a single scope instead of writing a separate closure for each. Doing this slows down the build but gives you bundles that are faster to execute.

12. Source maps

Source maps solve this problem by providing a mapping between the original and the transformed source code. In addition to source compiling to JavaScript, this works for styling as well.

Webpack can generate both inline or separate source map files. The inline ones are included to the emitted bundles and are valuable during development due to better performance. The separate files are handy for production usage as then loading source maps is optional.

13. Minifying Javascript

Since webpack 4, the production output gets minified using terser by default. Terser is an ES2015+ compatible JavaScript-minifier.

14. Server-Side Rendering and Pre-Rendering

Webpack provides require.resolveWeak for implementing SSR. It's a specific feature used by solutions such as react-universal-component underneath.

Prerendering is another technique similar to SSR that is easier to implement. The point is to use a headless browser to render the initial HTML markup of the page and then serve that to the crawlers. The caveat is that the approach won't work well with highly dynamic data.

The following solutions exist for webpack:

prerender-spa-plugin uses Puppeteer underneath.

prerender-loader integrates with html-webpack-plugin but also works without it against HTML files. The loader is flexible and can be customised to fit your use case (i.e. React or other framework).

Disadvantages

1. ***Webpack's approach comes with a learning curve as it relies on configuration and looks very different from other tools.***

2. ***Slower Development Mode***

   Webpack has to bundle all modules when you start the development server. Because of this, it can be very slow, taking from 2 to 30 seconds typically, or even as much as 150, to finish when you start the dev server.

3. ***Complex and customised configuration for different environments(dev/prod)***

Webpack is famous for how much we have to learn just to be able to configure it. We need plugins for doing simple things, like loading css, and the config files are extremely complicated.

4. ***Duplication of Transitive dependencies***

Webpack is not going to check whether 2 different versions of the transitive dependency files are exactly the same. it will treat them as unique files and bundle them together. Need to carefully analyse the bundle and eliminate the duplication with the suitable plugin.

5. ***Complexity associated with Plugins and Loaders development with major breakthrough changes with Webpack.***

Sneak Peek at Webpack Internals

Dependency Graph

When a file depends on another file, it recursively builds a graph with all dependencies needed to generate the bundle.

Entry Point

It is the file that webpack uses to start resolving the dependency graph.

Output

The name of the file or files and the path where webpack will generate the bundles containing the dependency graph.

Loader

These modules are responsible for processing different kinds of files. They are transformed into JavaScript modules that can be understood by webpack.

Later, the modules can be added to the bundle file.

Plugins

Plugins are modules that can perform many tasks, some examples are,

- Extract text (CSS) from your bundles into a separate file.
- Replace just the module changed instead of refreshing the whole page (Enable Hot Module Replacement).

Tapable

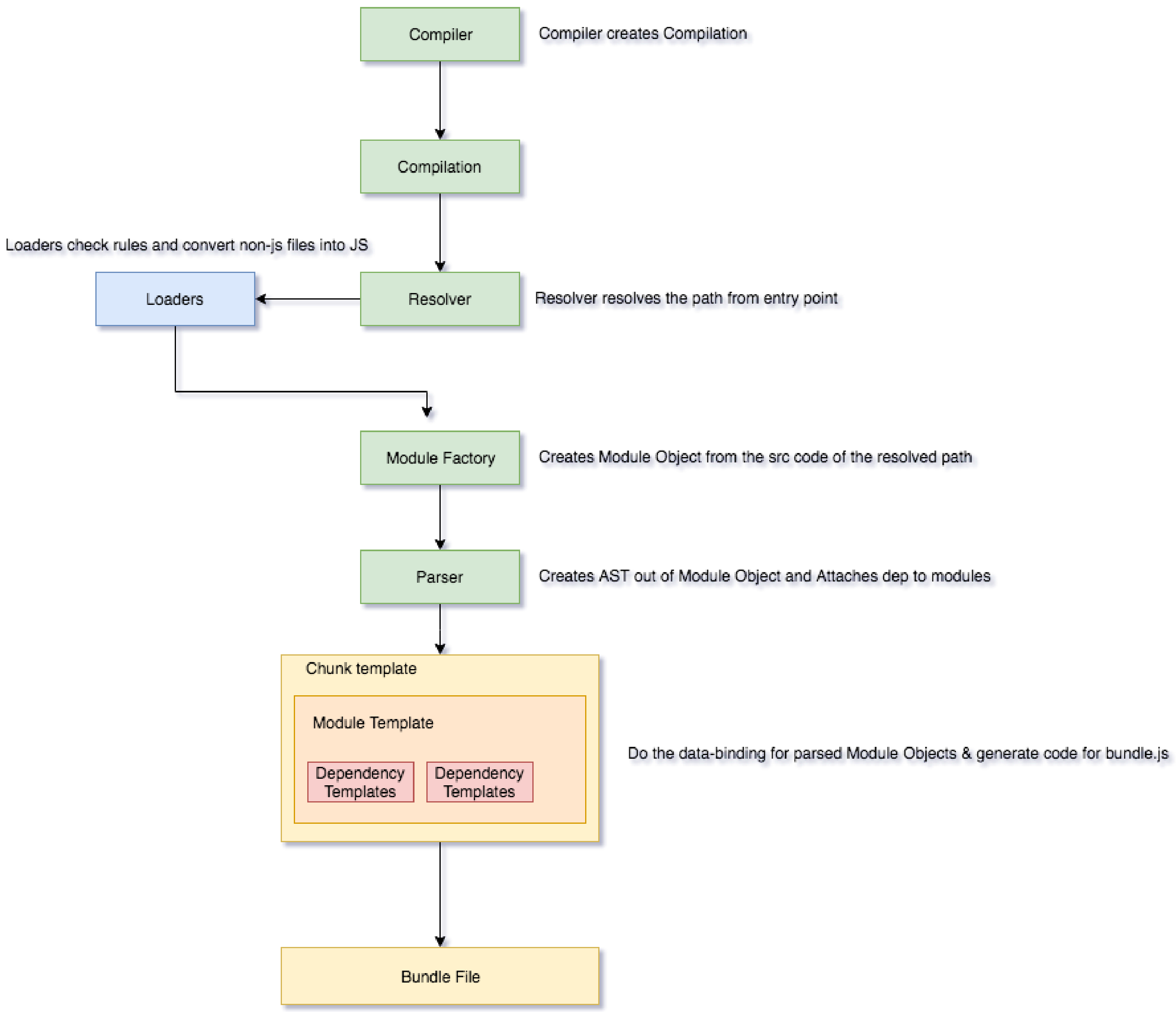
Tapable is a library that allows us to add plugins to a module. This is possible because every Class and Object that extends Tapable emits events that plugins can hook into to add functionality.

The following Tapable instances are being used in Webpack's bundling process.

- Compiler
    - Top-level API
    - Start / Stop the Webpack
    - Central dispatcher
    - Exposed Via node API
    - Main engine that creates a compilation instance with all the options passed through the CLI or Node API by extending the Tapable class
    - To register and call plugins
- Compilation

    - Module is used by the Compiler to create new compilations (or builds).
    - A compilation instance has access to all modules and their dependencies (most of which are circular references).
    - It is the literal compilation of all the modules in the dependency graph of an application.
- Resolver
    - Takes relative path and find out the file existence using absolute path

- Module factory
    - Responsible for creating module objects from the collected info from resolver.
    - Takes successfully resolved requests and collects the store and Create module object.
    - Module is used by the Compiler to generate dependencies from webpack specific require.context API.
- Parser
    - Takes each module content in string format and converts it into a tree (AST) and to be fed into the module object
    - Finds all require, import, dynamic import and look for other specific statements.
    - Module object passed to the parser and the dependency instances attached to the module itself.
    - Provides a variety of Tapable hooks that can be used by plugin authors to customise the parsing process.
- Template
    - This template instance is for representing data binding in dependency graphs.
    - Creates the code / final output to be seen in bundles


Compiler
Compiler creates Compilation
Compilation
Loaders check rules and convert non-js files into JS
Loaders
Resolver
Resolver resolves the path from entry point
Module Factory
Creates Module Object from the src code of the resolved path
Parser
Creates AST out of Module Object and Attaches dep to modules
Chunk template
Module Template
Dependency Templates
Dependency Templates
Do the data-binding for parsed Module Objects & generate code for bundle.js
Bundle File

Sample Configuration for Webpack

```
module.exports = options => {
  return {
    entry: {
      'bundle': path.resolve(__dirname, './test.js')
    },
    output: {
      filename: '[name].js',
    }
    module: {
      rules: [
        {
          test: /\.css$/,
          use: [ 'style-loader', 'çss-loader']
        }
      ]
    }
    plugins: [
      new customPlugin({
        options: 'nothing'
      })
    ]
  }
}
```

Alternatives

There are plenty of tools available for javascript application bundling. Each one of them is thriving to solve a focused problem.

Here, the list goes

- Rollup
  - It uses the standardised ES module format for code, instead of previous idiosyncratic solutions such as CommonJS and AMD.
- Parcel
  - It's a zero configuration build tool. It provides almost all the features provided by webpack with the lightning speed.
  - As It's compiler and source map implementations are in Rust, it's 10-20x faster than other JavaScript-based tools.
- Esbuild
  - It's a new born baby in the JS bundler's family and written in Go. It supports CJS IFFI, and ESM module systems.
  - Support for Transpilation of JSX to JS
  - As it's still in the experimental stage, it's APIs are unstable.

    - Code-splitting supported only for ESM modules.
- Snowpack
    - A modern JavaScript module bundler unique for its unbundled development idea.
    - The unbundled development pattern involves serving your application unbundled during development. Each file is built once and cached, and when a file changes, Snowpack rebuilds only that file. And the result of this is that we get much faster build time since we are not rebuilding our application with each file change.
- Speedy Web Compiler(SWC)
    - It is a JavaScript/TypeScript compiler focusing on performance written in Rust.
    - Can be used for both compilation and bundling. For compilation, it takes JavaScript / TypeScript files using modern JavaScript features and outputs valid code that is supported by all major browsers.

[1]

Conclusion

Webpack is not limited to simply bundling source files. Because it can support a multitude of plugins, it can perform many additional tasks. Webpack module loaders are able to parse different file types.

It's high potentiality can be utilised with properly customised configuration with required loaders / plugins and a realistic performance budget based on our application size and requirements.

The generated bundles have to be periodically analysed with the available bundle analyser tools to improve the application's performance.

Webpack is a better choice for building more complex Javascript applications with lots of third-party integrations and CommonJs dependencies, which would need extensive code splitting and use lots of static assets.

[1] Rereference

Everything is a plugin! Mastering webpack from the inside out - Sean Larkin
https://www.youtube.com/watch?v=4tQiJaFzuJ8