PREV
15. Testing

NEXT
17. Architectural Patterns

Chapter 16. Build Tools

Webpack

Webpack is like browserify, it bundles up your javascript into a single package.

Webpack can also:

- bundle css, images and other assets into the same package.
- pre-process files before bundling (less, coffee, jsx, etc).
- split your bundle into multiple files based on entry locations.
- support feature flags for development.
- perform “hot” module replacement.
- support asynchronous loading.

As a result, webpack can do the job of browserify and other build tools like grunt or gulp.

Webpack is a module system, it support adding and replacing plugins. By default it comes with a commonjs parser plugin enabled.

This section isn’t going to go into detail every aspect of webpack. Just the basics and what you need to get webpack to work with React.js.

WEBPACK AND REACT.JS

React.js helps you to create components for you application. Web pack helps you bundle not only javascript but all your other assets required for you application. Allowing you to create components that include all their asset dependencies. Making your components more portable, as they can bring their own dependencies along with them. As an added advantage, as your application grows

and changes, when you remove components, their asset dependencies are also removed. No more dead css or orphaned images.

Let's have a look at what a what a React.js component looks like when it is requiring it's own asset dependencies.

```
//logo.js
require('./logo.css');

var React = require('react');

var Logo = React.createClass({
    render: function () {
        return <img className="Logo" src={require('./logo.png')} />
    }
});

module.exports = Logo;
```

To bundle this component up we need to have any entry point for our application.

```
//app.js
var React = require('react');
var Logo = require('./logo.js');

React.renderComponent(<Logo/>, document.body);
```

We will now create a web pack config file to tell webpack what loaders to use for the different file types. What your applications entry point is and where to place all the bundled files.

```
//webpack.config.js
module.exports = {
    //starting point of your application
  entry: './app.js',
  output: {
        //location for all bundled assets
    path: './public/build',

        //prefix for all url-loader assets
    publicPath: './build/',

        //resulting name of bundled application
    filename: 'bundle.js'
  },
  module: {
    loaders: [
      {
                //regex for file type supported by the loader
                test: /\.(js)$/,

                //type of loader to be used
                //loaders can accept parameters as a query string
                loader: 'jsx-loader?harmony'
```

```
                test: /\.(css)$/,

                //multiple loaders are chained together with an "!";
                loader: 'style-loader!css-loader'
            },
            {
                test: /\.(png|jpg)$/,

                //url-loader supports base64 encoding for inline assets
                loader: 'url-loader?size=8192'
            }
        ]
    }
};
```

Now you want to install webpack and all the loaders. You can either use **npm** via command line or package.json.

Make sure you install the loaders locally, not globally (-g).

```
npm install webpack react
npm install url-loader jsx-loader style-loader css-loader
```

Once everything is installed, you can run webpack.

```
//build once for development
webpack

//build with source maps
webpack -d

//build for production, minified, uglify dead-code removal
webpack -p

//fast incremental building, can be use with other options
webpack --watch
```

Browserify

Browserify is a JavaScript packaging tool that allows for node.js style `require()` calls in the browser. Without getting into too much detail and getting lost in the woods, browserify packages all the requires into a bundled js file suitable for running in the browser environment. Running browserify on any js file with `require` statements will automatically bundle the necessary js.

While powerful, Browserify is only for JavaScript, unlike bower, webpack, or other bundling solutions.

Setup a Browserify Project

terimnal. This will create a `package.json` file with the necessary assets.

```
npm init
# ... answer questions as necessary to complete init
npm install --save-dev browserify reactify react uglify-js
```

Add the following build script to the bottom of your `package.json` file so it will look similar to this:

```
  ...
  "devDependencies": {
    "browserify": "^5.11.2",
    "reactify": "^0.14.0",
    "react": "^0.11.1",
    "uglify-js": "^2.4.15"
  },
  "scripts": {
    "build": "browserify --debug index.js > bundle.js",
    "build-dist": "NODE_ENV=production browserify index.js | uglifyjs -m > bundle.min.js
  },
  "browserify": {
    "transform": ["reactify"]
  }
}
```

The default build task can be run with `npm run build`, and will create a bundle with source maps. This lets you inspect error messages and place breakpoints as if each file was included individually. You'll also see the original code with JSX rather than the compiled output.

For production builds we need to specify that the environment is production. React uses a transform called envify, which combined with a minifier like uglify, it allows debug code and detailed error messages to be removed for better performance and smaller file sizes.

If you would like some es6 features like arrow functions and class syntax, you can change the transform line to this:

```
    "transform": [["reactify", {"harmony": true}]]
```

Now you're ready to write some React and package it.

Add some React content

Create the following React + JSX javascript file as `index.js`.

```
/** @jsx React.DOM */
var React = require('react');
```

And add a basic `index.html` file

```
<html>
  <head>
    <title>React + Browserify Demo</title>
  </head>
  <body>
    This text should not appear in the browser
    <script src="bundle.js"></script>
  </body>
</html>
```

Now your project will have the following files and folders inside its root directory:

- index.html
- index.js
- node_modules/
- package.json

If you try to load the index.html file now it will fail to render the js since we have yet to build the final package. Run `npm run build` and reload the page to see your hello world.

Watchify

You may choose to add a watch task, which is good for development. Watchify is a wrapper around browserify which rebuilds your bundle when you change a file. It uses caching to speed up the rebuilds.

```
npm install --save-dev watchify
```

Add this to your scripts object in package.json.

```
"watch": "watchify --debug index.js -o bundle.js"
```

Now instead of `npm run build`, you can `npm run watch` for a smoother development experience.

Build

Now simply run the build script to package your React + JSX file as a bundled script for the browser.

```
npm run-script build
```

You should see a new bundle.js file appear. If you open bundle.js you'll notice some minified js at the top, followed by your transformed React + JSX code. This file now contains all the supporting code along with your index.js file, ready to run in the browser, and loading index.html will now succeed.