Top ES6/ES2015 Features

Section 4: Babel

Azat Mardan @azat_co

Node University

What is Babel

Problem: Old browsers (IE9) will never support ES6. How do you know or enforec that all your users use modern browsers?

Most of the times you cannot.

Write for both?

Babel converts ES6 code into ES5-compatible code.

How to use Babel

- Babel command-line interface (CLI) tool
- Node.js or browser JavaScript script (API approach)
- Build tool

Babel RELP

2 lines turn into 7: http://bit.ly/2iLvVW4

```
const a = 1

let f = (r={}) => r
```

Using Babel CLI

Let's use Babel CLI

Firsly, ensure Node v6.2.0, and npm v3.8.9:

```
node -v
npm -v
```

Create a new folder `es6-test`: `mkdir es6-test`

Create `package.json` inside of your new folder with `npm init -y`

Define Babel presets in `package.json`:

```
{
  ...
  "babel": {
    "presets": ["es2015"]
  }
}
```

or `.babelrc`:

```
{
  "presets": ["es2015"]
}
```

Optionally, fill `package.json` with information such as project name, license, GitHub repository, etc.

Install Babel CLI and React preset locally using `npm i babel-cli@6.9.0 babel-preset-es2015@6.5.0 --save-dev` to save these dependencies in `devDependencies` in `package.json`

Test run to see you have Babel CLI:

```
./node_modules/.bin/babel --version
```

(You should see v6.9.0. babel --version might also work depending on your npm configuration.)

If you use React and JSX, you can also add React preset to convert JSX to React:

```
npm i babel-preset-react@6.5.0 --save-dev
```

And add to presets configuration next to `es2015`:

```
{
"presets": ["react", "es2015"]
}
```

After installation, you issue a command to process your `src/script.js` ES6 into `js/script.js` ES5 JavaScript:

```
./node_modules/.bin/babel src/script.js -o js/script.js
```

Optionally, create an npm script `npm run build`. Just open `package.json` with your editor and add this line to `scripts`:

```
{
  ...
  "scripts": [
    "build": "./node_modules/.bin/babel src/script.js -o js/script.js"
  ],
  ...
}
```

You can automate this command with the watch option (`-w` or `--watch`):

```
./node_modules/.bin/babel src/script.js -o js/script.js -w
```

The Babel command to watch for any changes in `src/script.js` and compile it to `js/script.js` when you save the updated ES6 file. When it happens, terminal/command prompt will display:

```
change src/script.js
```

When you start having more JSX files, simply use the command with `-d` (`--out-dir`) and folder names to compile each ES6 source files (`src`) into many regular JS files (`build`):

```
./node_modules/.bin/babel src --d build
```

Often times, having just a single file to load is better for the performance of a front-end app than loading many files. This is because each request add a delay. We can compile all files in the source directory into a single regular JS file with -o (--out-file):

```
./node_modules/.bin/babel src -o script-compiled.js
```

Bonus 

Remember stages 0-4 in section 1?

You can start using these features NOW!

Stage 0 is THE cutting edge.

In terminal/command prompt (replace 0 with 0-3):

```
npm install --save-dev babel-preset-stage-0
```

In `.babelrc`:

```
{
  "presets": ["stage-0"]
}
```

Or instead of `.babelrc` in `package.json`:

```
{
  ...
  "babel": {
    "presets": ["stage-0"]
  }
}
```

More Presets

» es2015-lose

» es2016

» es2017

» latest

» react

» stage-0, 1, 2, 3

https://babeljs.io/docs/plugins/#presets

The End of Section 4 😃