Setting up Babel 6

Posted Oct 31, 2015 by James Kyle

Babel 6 is fresh off the press and we're still getting a lot sorted out. In the past two days we've seen more activity on GitHub and Slack than ever before. Everyone who has been helping out has been doing a great job.

However, the documentation is still lacking at this point, I'm currently going through and creating an entirely new "Getting Started" section of the site.

This blog post will cover most of the basics until that's completed.

Installing Babel

The `babel` package is no more. Previously, it was the entire compiler and all the transforms plus a bunch of CLI tools, but this lead to unnecessarily large downloads and was a bit confusing. Now we've split it up into two separate packages: `babel-cli` and `babel-core`.

Shell

```
$ npm install --global babel-cli
# or
$ npm install --save-dev babel-core
```

If you want to use Babel from the CLI you can install `babel-cli` or if you want to use the Node API you can install `babel-core`.

Right now, most integrations haven't been updated for Babel 6, but that will be changing over the coming days.

Adding Plugins and Presets

Babel 6 ships without any default transforms, so when you run Babel on a file it will just print it back out to you without changing anything.

In order to start compiling various features you need to install plugins. For example if you want to use arrow functions:

First install the arrow functions plugin:

Shell

```
$ npm install --save-dev babel-plugin-transform-es2015-arrow-functions
```

Then add it to your `.babelrc` file like this:

JavaScript

```
{
  "plugins": ["transform-es2015-arrow-functions"]
}
```

Now if you run Babel on a file with arrow functions they will be compiled. Easy right? Not so fast.

Babel plugins are designed to compile down incrementally. Say if you wanted to use an ES2019 feature, it would be compiled down to ES2018 then ES2017 and so on. This ensures that people can use native implementation if they want to.

This even happens within a single specification, for example ES2015 constants will be compiled to ES2015 `let` variables. So if you want it to go all the way down to ES5 you need to compile that as well.

Shell

```
$ npm install --save-dev babel-plugin-check-es2015-constants
$ npm install --save-dev babel-plugin-transform-es2015-block-scoping
```

JavaScript

```
{
  "plugins": [
    "check-es2015-constants",
    "transform-es2015-block-scoping"
  ]
}
```

These dependencies aren't easy to keep track and if you don't want to manually specify them all you may install a preset instead.

Shell

```
$ npm install --save-dev babel-preset-es2015
```

JavaScript

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

To Include all Javascript versions:

Shell

```
$ npm install --save-dev babel-preset-env
```

JavaScript

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

React also has it's own preset:

Shell

```
$ npm install --save-dev babel-preset-react
```

JavaScript

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

This is also what the `stage` option has been replaced with:

Shell

```
$ npm install --save-dev babel-preset-stage-2
```

JavaScript

```
{
  "presets": ["stage-2"]
}
```

> Note: Stage presets include all the stages above them automatically (ie. stage 1 includes 2 and 3)

Tweet (https://twitter.com/share)

Community Discussion

2 replies
Dec '16

elgs

The beauty of Javascript is its simplicity. I like all the features in ES6 exception the class and extends. I think the ES6 designers are too eager to make Javascript look like Java. They might think Javascript was inferior without those shiny Java keywords. Actually it is the other way around.

5 Jan

That-Random-Guy

How come there's no ES6+ way to define propTypes in the component class in ReactJS?

For instance, component classes can now be written as such:

```
class Video extends React.Component {
  static defaultProps = {
    autoPlay: false,
    maxLoops: 10,
  }
  static propTypes = {
    autoPlay: React.PropTypes.bool.isRequired,
    maxLoops: React.PropTypes.number.isRequired,
    posterFrameSrc: React.PropTypes.string.isRequired,
    videoSrc: React.PropTypes.string.isRequired,
  }
  state = {
    loopsRemaining: this.props.maxLoops,
  }
}
```

The guide on this page demonstrates that defining the propTypes property is still inherently only plausible via the old convention. Why can't that also be updated to fit the newer syntax?

Why can't the the propType definition be its own class definition? Isn't that what it's doing behind the scenes anyway?