babel-loader

> *This README is for babel-loader v8 + Babel v7 Check the 7.x branch for docs with Babel v6*

This package allows transpiling JavaScript files using Babel and webpack.

Note: Issues with the output should be reported on the Babel Issues tracker.

Install

> *webpack 4.x | babel-loader 8.x | babel 7.x*

```
npm install -D babel-loader @babel/core @babel/preset-env webpack
```

Usage

webpack documentation: Loaders

Within your webpack configuration object, you'll need to add the babel-loader to the list of modules, like so:

```
module: {
  rules: [
    {
      test: /\.m?js$/,
      exclude: /(node_modules|bower_components)/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['@babel/preset-env']
        }
      }
    }
  ]
}
```

Options

See the `babel` options.

You can pass options to the loader by using the `options` property:

```
module: {
  rules: [
    {
      test: /\.m?js$/,
      exclude: /(node_modules|bower_components)/,
      use: {
        loader: 'babel-loader',
        options: {
          presets: ['@babel/preset-env'],
          plugins: ['@babel/plugin-proposal-object-rest-spread']
        }
      }
    }
  ]
}
```

This loader also supports the following loader-specific option:

- `cacheDirectory` : Default `false` . When set, the given directory will be used to cache the results of the loader. Future webpack builds will attempt to read from the cache to avoid needing to run the potentially expensive Babel recompilation process on each run. If the value is set to `true` in options (`{cacheDirectory: true}`), the loader will use the default cache directory in `node_modules/.cache/babel-loader` or fallback to the default OS temporary file directory if no `node_modules` folder could be found in any root directory.
- `cacheIdentifier` : Default is a string composed by the `@babel/core` 's version, the `babel-loader` 's version, the contents of `.babelrc` file if it exists, and the value of the environment variable `BABEL_ENV` with a fallback to the `NODE_ENV` environment variable. This can be set to a custom value to force cache busting if the identifier changes.
- `cacheCompression` : Default `true` . When set, each Babel transform output will be compressed with Gzip. If you want to opt-out of cache compression, set it to `false` -- your project may benefit from this if it transpiles thousands of files.
- `customize` : Default `null` . The path of a module that exports a `custom` callback like the one that you'd pass to `.custom()` . Since you already have to make a new file to use this, it is recommended that you instead use `.custom` to create a wrapper loader. Only use this if you *must* continue using `babel-loader` directly, but still want to customize.

Troubleshooting

babel-loader is slow!

Make sure you are transforming as few files as possible. Because you are probably matching `/\.m?js$/`, you might be transforming the `node_modules` folder or other unwanted source.

To exclude `node_modules`, see the `exclude` option in the `loaders` config as documented above.

You can also speed up babel-loader by as much as 2x by using the `cacheDirectory` option. This will cache transformations to the filesystem.

Babel is injecting helpers into each file and bloating my code!

Babel uses very small helpers for common functions such as `_extend`. By default, this will be added to every file that requires it.

You can instead require the Babel runtime as a separate module to avoid the duplication.

The following configuration disables automatic per-file runtime injection in Babel, requiring `@babel/plugin-transform-runtime` instead and making all helper references use it.

See the docs for more information.

NOTE: You must run `npm install -D @babel/plugin-transform-runtime` to include this in your project and `@babel/runtime` itself as a dependency with `npm install @babel/runtime`.

```
rules: [
  // the 'transform-runtime' plugin tells Babel to
  // require the runtime instead of inlining it.
  {
    test: /\.m?js$/,
    exclude: /(node_modules|bower_components)/,
    use: {
      loader: 'babel-loader',
      options: {
        presets: ['@babel/preset-env'],
        plugins: ['@babel/plugin-transform-runtime']
      }
    }
  }
]
```

NOTE: transform-runtime & custom polyfills (e.g. Promise library)

Since @babel/plugin-transform-runtime includes a polyfill that includes a custom regenerator-runtime and core-js, the following usual shimming method using `webpack.ProvidePlugin` will not work:

```
// ...
        new webpack.ProvidePlugin({
            'Promise': 'bluebird'
        }),
// ...
```

The following approach will not work either:

```
require('@babel/runtime/core-js/promise').default = require('bluebird');

var promise = new Promise;
```

which outputs to (using `runtime`):

```
'use strict';

var _Promise = require('@babel/runtime/core-js/promise')['default'];

require('@babel/runtime/core-js/promise')['default'] = require('bluebird');

var promise = new _Promise();
```

The previous `Promise` library is referenced and used before it is overridden.

One approach is to have a "bootstrap" step in your application that would first override the default globals before your application:

```
// bootstrap.js

require('@babel/runtime/core-js/promise').default = require('bluebird');

// ...

require('./app');
```

The Node.js API for `babel` has been moved to `babel-core`.

If you receive this message, it means that you have the npm package `babel` installed and are using the short notation of the loader in the webpack config (which is not valid anymore as of webpack 2.x):

```
  {
    test: /\.m?js$/,
```

```
      loader: 'babel',
    }
```

webpack then tries to load the `babel` package instead of the `babel-loader`.

To fix this, you should uninstall the npm package `babel`, as it is deprecated in Babel v6. (Instead, install `@babel/cli` or `@babel/core`.) In the case one of your dependencies is installing `babel` and you cannot uninstall it yourself, use the complete name of the loader in the webpack config:

```
    {
      test: /\.m?js$/,
      loader: 'babel-loader',
    }
```

Exclude libraries that should not be transpiled

`core-js` and `webpack/buildin` will cause errors if they are transpiled by Babel.

You will need to exclude them form `babel-loader`.

```
{
  "loader": "babel-loader",
  "options": {
    "exclude": [
      // \\ for Windows, \/ for Mac OS and Linux
      /node_modules[\\\/]core-js/,
      /node_modules[\\\/]webpack[\\\/]buildin/,
    ],
    "presets": [
      "@babel/preset-env"
    ]
  }
}
```

Customize config based on webpack target

Webpack supports bundling multiple targets. For cases where you may want different Babel configurations for each target (like `web` *and* `node`), this loader provides a `target` property via Babel's caller API.

For example, to change the environment targets passed to `@babel/preset-env` based on the webpack target:

```
// babel.config.js

module.exports = api => {
  return {
    plugins: [
      "@babel/plugin-proposal-nullish-coalescing-operator",
      "@babel/plugin-proposal-optional-chaining"
    ],
    presets: [
      [
        "@babel/preset-env",
        {
          useBuiltIns: "entry",
          // caller.target will be the same as the target option from webpack
          targets: api.caller(caller => caller && caller.target === "node")
            ? { node: "current" }
            : { chrome: "58", ie: "11" }
        }
      ]
    ]
  }
}
```

Customized Loader

`babel-loader` exposes a loader-builder utility that allows users to add custom handling of Babel's configuration for each file that it processes.

`.custom` accepts a callback that will be called with the loader's instance of `babel` so that tooling can ensure that it using exactly the same `@babel/core` instance as the loader itself.

In cases where you want to customize without actually having a file to call `.custom`, you may also pass the `customize` option with a string pointing at a file that exports your `custom` callback function.

Example

```
// Export from "./my-custom-loader.js" or whatever you want.
module.exports = require("babel-loader").custom(babel => {
  function myPlugin() {
    return {
      visitor: {},
    };
  }

  return {
```

```
      // Passed the loader options.
      customOptions({ opt1, opt2, ...loader }) {
        return {
          // Pull out any custom options that the loader might have.
          custom: { opt1, opt2 },

          // Pass the options back with the two custom options removed.
          loader,
        };
      },

      // Passed Babel's 'PartialConfig' object.
      config(cfg) {
        if (cfg.hasFilesystemConfig()) {
          // Use the normal config
          return cfg.options;
        }

        return {
          ...cfg.options,
          plugins: [
            ...(cfg.options.plugins || []),

            // Include a custom plugin in the options.
            myPlugin,
          ],
        };
      },

      result(result) {
        return {
          ...result,
          code: result.code + "\n// Generated by some custom loader",
        };
      },
    };
  });


  // And in your Webpack config
  module.exports = {
    // ..
    module: {
      rules: [{
        // ...
        loader: path.join(__dirname, 'my-custom-loader.js'),
        // ...
      }]
    }
  };
```

`customOptions(options: Object): { custom: Object, loader: Object }`

Given the loader's options, split custom options out of `babel-loader`'s options.

`config(cfg: PartialConfig): Object`

Given Babel's `PartialConfig` object, return the `options` object that should be passed to `babel.transform`.

`result(result: Result): Result`

Given Babel's result object, allow loaders to make additional tweaks to it.

License

MIT