**Loaders**

Apply loaders to the minimal number of modules necessary. Instead of:

module.exports = {

//...

module: {

rules: [

{

test: /\.js$/,

loader: 'babel-loader'

}

]

}

};

Use the include field to only apply the loader modules that actually need to be transformed by it:

module.exports = {

//...

module: {

rules: [

{

test: /\.js$/,

**include: path.resolve(\_\_dirname, 'src'),**

loader: 'babel-loader'

}

]

}

};

### Bootstrap

Each additional loader/plugin has a bootup time. Try to use as few tools as possible.

### Smaller = Faster

Decrease the total size of the compilation to increase build performance. Try to keep chunks small.

* Use fewer/smaller libraries.
* Use the CommonsChunkPlugin in Multi-Page Applications.
* Use the CommonsChunkPlugin in async mode in Multi-Page Applications.
* Remove unused code.
* Only compile the part of the code you are currently developing on.

### Persistent cache

Enable persistent caching with the cache-loader. Clear cache directory on "postinstall" in package.json.

### Incremental Builds

Use webpack's watch mode. Don't use other tools to watch your files and invoke webpack. The built-in watch mode will keep track of timestamps and passes this information to the compilation for cache invalidation.

In some setups, watching falls back to polling mode. With many watched files, this can cause a lot of CPU load. In these cases, you can increase the polling interval with watchOptions.poll.

**Compile in Memory**

The following utilities improve performance by compiling and serving assets in memory rather than writing to disk:

* webpack-dev-server
* webpack-hot-middleware
* webpack-dev-middleware

### Output Without Path Info

webpack has the ability to generate path info in the output bundle. However, this puts garbage collection pressure on projects that bundle thousands of modules. Turn this off in the options.output.pathinfo setting:

module.exports = {

// ...

output: {

pathinfo: false

}

};