Cheat Sheet for comprensive Webpack

Comprehensive Webpack Cheat Sheet

1. Introduction to Webpack

What is Webpack?

- **Module Bundler**: Combines multiple modules into a single file.
- **Static Assets Management**: Handles images, CSS, and other assets.
- **Code Splitting**: Allows splitting code into multiple bundles.
- **Hot Module Replacement (HMR)**: Updates modules in real-time without a full reload.

Installation

```
npm install webpack webpack-cli --save-dev
```

Basic Usage

```
npx webpack
```

---

2. Basic Configuration

Webpack Configuration File (`webpack.config.js`)

```
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
  mode: 'development',
  module: {
    rules: [
      // Loaders configuration
    ],
  },
```

```
  plugins: [
    // Plugins configuration
  ],
};
```

Entry Points

- **Single Entry**:

```
  entry: './src/index.js',
```

- **Multiple Entries**:

```
  entry: {
    app: './src/app.js',
    admin: './src/admin.js',
  },
```

Output

- **Single Output**:

```
  output: {
    filename: 'bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
```

- **Multiple Outputs**:

```
  output: {
    filename: '[name].bundle.js',
    path: path.resolve(__dirname, 'dist'),
  },
```

Mode

- **Development**: `mode: 'development'`
- **Production**: `mode: 'production'`
- **None**: `mode: 'none'`

---

3. Loaders

What are Loaders?

- **Transform Files**: Convert files into modules.

- **Common Loaders**: `babel-loader`, `css-loader`, `style-loader`, `file-loader`, `url-loader`.

Common Loaders Configuration

- **Babel Loader**:

```
module: {
  rules: [
    {
      test: /\.js$/,
      exclude: /node_modules/,
      use: {
        loader: 'babel-loader',
      },
    },
  ],
},
```

- **CSS Loader**:

```
module: {
  rules: [
    {
      test: /\.css$/,
      use: ['style-loader', 'css-loader'],
    },
  ],
},
```

- **File Loader**:

```
module: {
  rules: [
    {
      test: /\.(png|jpg|gif)$/,
      use: [
        {
          loader: 'file-loader',
          options: {},
        },
      ],
```

```
    },
  ],
},
```

Loader Options

- **Options Object**:

```
use: {
  loader: 'babel-loader',
  options: {
    presets: ['@babel/preset-env'],
  },
},
```

---

4. Plugins

What are Plugins?

- **Extend Webpack's Functionality**: Perform tasks like optimization, asset management, and more.

- **Common Plugins**: `HtmlWebpackPlugin`, `MiniCssExtractPlugin`, `CleanWebpackPlugin`.

Common Plugins Configuration

- **HtmlWebpackPlugin**:

```
const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
  plugins: [
    new HtmlWebpackPlugin({
      template: './src/index.html',
    }),
  ],
};
```

- **MiniCssExtractPlugin**:

```
const MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {
```

```
  plugins: [
    new MiniCssExtractPlugin({
      filename: '[name].css',
    }),
  ],
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [MiniCssExtractPlugin.loader, 'css-loader'],
      },
    ],
  },
};
```

- **CleanWebpackPlugin**:

```
const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {
  plugins: [
    new CleanWebpackPlugin(),
  ],
};
```

---

5. Optimization

Code Splitting

- **Dynamic Imports**:

```
import('./module').then(module => {
  // Use module
});
```

- **SplitChunksPlugin**:

```
optimization: {
  splitChunks: {
    chunks: 'all',
  },
},
```

Minification

- **Production Mode**: Automatically minifies code.

- **TerserPlugin**:

```
const TerserPlugin = require('terser-webpack-plugin');

module.exports = {
  optimization: {
    minimize: true,
    minimizer: [new TerserPlugin()],
  },
};
```

Tree Shaking

- **Remove Unused Code**: Automatically removes unused exports.

- **Enable in Production Mode**: `mode: 'production'`.

---

6. Development Tools

DevServer

- **Live Reloading**:

```
devServer: {
  contentBase: './dist',
  hot: true,
},
```

- **Command**:

```
npx webpack serve
```

Source Maps

- **Enable Source Maps**:

```
devtool: 'inline-source-map',
```

- **Types**: `source-map`, `inline-source-map`, `cheap-source-map`, `eval-source-map`.

Hot Module Replacement (HMR)

- **Enable HMR**:

```
devServer: {
  hot: true,
},
```

- **React HMR**:

```
if (module.hot) {
  module.hot.accept();
}
```

---

7. Advanced Techniques

Environment Variables

- **DefinePlugin**:

```
const webpack = require('webpack');

module.exports = {
  plugins: [
    new webpack.DefinePlugin({
      'process.env.NODE_ENV': JSON.stringify('production'),
    }),
  ],
};
```

Multiple Configurations

- **Array of Configurations**:

```
module.exports = [
  {
    entry: './src/app.js',
    output: {
      filename: 'app.bundle.js',
    },
  },
  {
    entry: './src/admin.js',
```

```
    output: {
      filename: 'admin.bundle.js',
    },
  },
];
```

Custom Loaders and Plugins

- **Custom Loader**:

```
module.exports = function(source) {
  return source.replace('foo', 'bar');
};
```

- **Custom Plugin**:

```
class MyPlugin {
  apply(compiler) {
    compiler.hooks.done.tap('MyPlugin', () => {
      console.log('Build completed!');
    });
  }
}

module.exports = {
  plugins: [
    new MyPlugin(),
  ],
};
```

---

8. Common Issues and Solutions

Issue: Module Not Found

- **Solution**: Check the path and ensure the module is installed.

Issue: CSS Not Loading

- **Solution**: Ensure `style-loader` and `css-loader` are correctly configured.

Issue: HMR Not Working

- **Solution**: Ensure `hot: true` is set in `devServer` and HMR code is added in the entry file.

Issue: Build Takes Too Long

- **Solution**: Optimize loaders and plugins, use `thread-loader` for parallel processing.

---

9. Tips and Tricks

Use `webpack-merge` for Configuration

- **Combine Configurations**:

```
  const { merge } = require('webpack-merge');
  const commonConfig = require('./webpack.common.js');

  module.exports = merge(commonConfig, {
    mode: 'development',
    devtool: 'inline-source-map',
  });
```

Use `webpack-bundle-analyzer`

- **Analyze Bundle Size**:

```
  npm install --save-dev webpack-bundle-analyzer
```

```
  const BundleAnalyzerPlugin = require('webpack-bundle-
analyzer').BundleAnalyzerPlugin;

  module.exports = {
    plugins: [
      new BundleAnalyzerPlugin(),
    ],
  };
```

Use `thread-loader` for Parallel Processing

- **Speed Up Builds**:

```
  module: {
    rules: [
      {
        test: /\.js$/,
        use: [
          {
            loader: 'thread-loader',
            options: {
```

```
            workers: 2,
          },
        },
        'babel-loader',
      ],
    },
  ],
},
```

Use `cache-loader` for Caching

- **Improve Build Performance**:

```
module: {
  rules: [
    {
      test: /\.js$/,
      use: [
        'cache-loader',
        'babel-loader',
      ],
    },
  ],
},
```

This cheat sheet provides a comprehensive overview of Webpack, covering essential features, configuration, optimization, and advanced techniques. Use this as a reference to streamline your Webpack setup and improve your development workflow.

By Ahmed Baheeg Khorshid

ver 1.0