常用 Plugin 有哪些，该如何配置?

Plugin 简介

对于 loader，它就是一个转换器，将 A 文件进行编译形成 B 文件，这里操作的是文件，比如将 A.scss 或 A.less 转变为 B.css，单纯的文件转换过程;

对于 plugin，它就是一个扩展器，它丰富了 wepack 本身，针对是 loader 结束后，webpack 打包的整个过程，它并不直接操作文件，而是基于事件机制工作，会监听 webpack 打包过程中的某些节点，执行广泛的任务。

loader 被用于转换某些类型的模块，而插件则可以用于执行广泛的任务。插件的范围包括，从打包优化和压缩，一直到重新定义环境中的变量。插件接口功能极其强大，可以用来处理各种各样的任务。

想要使用一个插件，你只需要 require()它，然后将它添加到 plugins 数组中。多数插件可以通过选项自定义。你也可以在一个配置中因为不同目的而多次使用同一个插件，这时需要通过使用 new 操作符来创建它的一个实例。

剖析：

webpack 插件是一个具有 apply 属性的 JavaScript 对象。apply 属性会被 webpack compiler 调用，并且 compiler 对象可在整个编译周期访问。

用法：

由于插件可以携带参数/选项，你必须在 webpack 配置中，想 plugins 属性传入 new 实例。

例如:

```
module.exports = {
        //...
        plugins: [
                new HtmlWebpackPlugin({
            template: './src/templet.html',
            filename: 'pageA.html',
            title: 'pageA',
            chunks: ['pageA'],
            hash: true,
            minify: {
                removeAttributeQuotes: true
            }
        })
        ],
   //...
```

```
}
```

常用 Plugin 介绍以及配置（适用 Webpack4.0+）

1.mini-css-extract-plugin

css-提取，看名字就懂提取 css 用的。在这之前我们可能会使用 extract-text-webpack-plugin 比较多一些，两者相比它有什么优势呢？extract-text-webpack-plugin 它对 css 的提取，最终是根据你创建的实例、或者配置多个入口 chunk 来的，比如你配置了一个入口文件，最终所有的 css 都会提取在一个样式文件里，如果你创建了多个 extract-text-webpack-plugin 实例，则会生成多个 css 的文件，而 mini-css-extract-plugin，它默认就会对你的样式进行模块化拆分，嗯，有点跟 output 里的配置一个意思，异步按需加载，不再仅仅是 js 的特权;

```
const MiniCssExtractPlugin = require("mini-css-extract-plugin");
module.exports = {
  plugins: [
    new MiniCssExtractPlugin({
      filename: "[name].css",
      chunkFilename: "[id].css"
    })
  ],
  module: {
    rules: [
      {
        test: /\.css$/,
        use: [
          {
            loader: MiniCssExtractPlugin.loader,
            options: {
              publicPath: '../'
            }
          },
          "css-loader"
        ]
      }
    ]
  }
}
```

2.optimize-css-assets-webpack-plugin

它将在 Webpack 构建期间搜索 CSS 资源，并将优化\最小化 CSS

```
var OptimizeCssAssetsPlugin = require('optimize-css-assets-webpack-plugin');
module.exports = {
  module: {
    rules: [
      {
        test: /\.css$/,
        loader: ExtractTextPlugin.extract('style-loader', 'css-loader')
      }
    ]
  },
  plugins: [
    new ExtractTextPlugin('styles.css'),
    new OptimizeCssAssetsPlugin({
      assetNameRegExp: /\.optimize\.css$/g,
      cssProcessor: require('cssnano'),
      cssProcessorPluginOptions: {
        preset: ['default', { discardComments: { removeAll: true } }],
      },
      canPrint: true
    })
  ]
};
```

配置介绍

- assetNameRegExp：默认是全部的 css 都会压缩，该字段可以进行指定某些要处理的文件，
- cssProcessor：指定一个优化 css 的处理器，默认 cssnano，
- cssProcessorPluginOptions：cssProcessor 后面可以跟一个 process 方法，会返回一个 promise 对象，而 cssProcessorPluginOptions 就是一个 options 参数选项！
- canPrint：布尔，是否要将编译的消息显示在控制台，没发现有什么用！
- 坑点 ：建议使用高版本的包，之前低版本有遇到样式丢失把各浏览器前缀干掉的问题，

3.html-webpack-plugin

- 为 html 文件中引入的外部资源如 script、link 动态添加每次 compile 后的 hash，防止引用缓存的外部文件问题
- 可以生成创建 html 入口文件，比如单页面可以生成一个 html 文件入口，配置 N 个 html-webpack-plugin 可以生成 N 个页面入口

```
new HtmlWebPackPlugin({
  filename: path.resolve(__dirname, '../assets/index.html'),
  template: path.resolve(__dirname,"../views/temp.html"),
  minify:{ //压缩 HTML 文件
    removeComments:true,
    collapseWhitespace:true
    },
  inlineSource:  '.(js|css)',
  inject: false,
    chunks: ['vendors', 'index'], //首席填坑官·苏南的专栏
    hash:true,
    favicon、meta、title 等都可以配置，页面内使用「<%=
htmlWebpackPlugin.options.title %>」即可
    ……
})
```

配置介绍：

- filename：输出文件名；
- template：模板文件，不局限于 html 后缀哦；
- removeComments：移除 HTML 中的注释；
- collapseWhitespace：删除空白符与换行符，整个文件会压成一行；
- inlineSource：插入到 html 的 css、js 文件都要内联，即不是以 link、script 的形式引入；
- inject：是否能注入内容到 输出 的页面去；
- chunks：指定插入某些模块；
- hash：每次会在插入的文件后面加上 hash ，用于处理缓存，如：；

4.uglifyjs-webpack-plugin

js 代码压缩,默认会使用 optimization.minimizer

```
/默认：
optimization:{
    minimizer:true
};
```

```
//自定义
minimizer: [
  new UglifyJsPlugin({
    cache: true,
    // cache: "assets",
    parallel: true, //也可以指定 Number ,即最多并行运行数量
    sourceMap: true,
    uglifyOptions: {
      output: {
        comments: false,
        …… //首席填坑官·苏南的专栏，QQ:912594095
      },
      compress: {
          warnings: false,
          drop_console:true,
          ……
        }
    },
  }),
],
```

配置介绍：

- cache: Boolean/String ,字符串即是缓存文件存放的路径；
- test：正则表达式、字符串、数组都可以，用于只匹配某些文件，如：/.js(\?.*)?$/i;
- parallel ：启用多线程并行运行来提高编译速度，经常编译的时候听到电脑跑的呼呼响，可能就是它干的，哈哈～；
- output.comments ： 删除所有注释，
- compress.warnings ：插件在进行删除一些无用代码的时候，不提示警告，
- compress.drop_console：喜欢打 console 的同学，它能自动帮你过滤掉，再也不用担心线上还打印日志了；

5.preload-webpack-plugin

预加载是一种 Web 标准，旨在提高性能和粒度负载。这是一个声明性提取，可以告诉浏览器开始提取源，因为开发人员知道很快就会需要资源。

在简单的 Web 应用程序中，可以直接指定要预加载的脚本的静态路径 - 特别是如果它们的名称或位置不太可能发生变化。在更复杂的应用程序中，可以使用动态名称将

JavaScript 拆分为“块”（表示路径或组件）。这些名称可以包括可以随每个构建而更改的哈希，数字和其他属性。

例如，chunk.31132ae6680e598f8879.js。

为了更容易为延迟加载连接异步块，这个插件提供了一种使用它们连接它们的简单方法 <link rel='preload'>。

//注意点 1：请把配置一定写在 HtmlWebPackPlugin 插件之后，否则会报 `HtmlWebpackPlugin.getHooks is not a function`错误，
//注意点 2：webpack4 之后，请使用最新版本 npm install --save-dev preload-webpack-plugin@next，

```
new PreloadWebpackPlugin({
  rel: 'prefetch',
  as: 'script',
  // as(entry) {
  //   if (/\.css$/.test(entry)) return 'style';
  //   return 'script';//首席填坑官·苏南的专栏，QQ:912594095
  // },
  include: 'asyncChunks',
  // fileBlacklist: ["index.css"]
  fileBlacklist: [/\index.css|index.js|vendors.js/, /\.whatever/]
})
```

配置介绍：

- 编译完成后，你可以（指定某些/全部）文件动态插入到 HtmlWebPackPlugin 配置输出的文件内，
- as： 表示你预加载的资源类型；可以有有先多：script、font、image、style、video 等等，更多详细请查看 API，它还可以返回 function；
- include：要插入，进行预加载的文件，它有：allChunks、initial、asyncChunks、数组等选项，数组即表示指定插入某些文件
- fileBlacklist:即文件黑名单，可以指定某个文件，也可以使用正则来匹配；

6.HotModuleReplacementPlugin

- 热更新替换，在不刷新重载页面的情况下更换编辑修改后的代码：
- 它只会更新改动过的内容，所以速度很快，几乎在自己刚改完，切到浏览器窗口内容就已经更新完了；
- 使用 HotModuleReplacementPlugin 插件后，它会暴露一个 module.hot 对象。

开启 webpack-dev-server 的 HMR
因为 Hot-Module-Replacement 的热更新是依赖于 webpack-dev-server，后者是在打包文件改变时更新打包文件或者 reload 刷新整个页面，HRM 是只更新修改的部分。

配置 webpakc-dev-server

```
devServer:{
    port:8080,
    contentBase:path.join(__dirname,'./dist'),//设置 url 的根目录，如果不设置，则默认是指向项目根目录
    historyApiFallback : true,//让所有 404 的页面定位到 index.html
    hotOnly:true
},
```

hotOnly:true 表示只会对可以热更新的部分进行热更新

安装对应的插件

```
plugins: [
new webpack.NamedModulesPlugin(), //用于启动 HMR 时可以显示模块的相对路径
new webpack.HotModuleReplacementPlugin(), //hot module replacement 启动模块热替换的插件
]
```

入口文件配置接受热更新

```
if (module.hot) {
    module.hot.accept(() => {
    const NextApp = require('component/App/App').default;
    renderWithHotReload(App);
    });
}
```

以上就是为大家整理的=项目中常用的插件。对于 Webpack 中的插件是非常多的，但是常用的就是这些。对于插件，是为了我们更好的编程。所以使用插件也要适度。