# 工程化

## Babel

* babel 原理

[Babel原理 - 云+社区 - 腾讯云 (tencent.com)](https://cloud.tencent.com/developer/article/1592034)

* babel 中 plugin 和 presets 区别
* babel 中 plugin 和 presets 执行顺序

[Babel：plugin、preset的区别与使用 - 蓝天fly - 博客园 (cnblogs.com)](https://www.cnblogs.com/dapengFly/p/9876915.html)

* babel 中 polyfill 作用

[【JavaScript】深入理解Babel原理及其使用 - 简书 (jianshu.com)](https://www.jianshu.com/p/e9b94b2d52e2)

* babel 和 webpack 区别

## Webpack

* webpack 介绍

[面试官：说说你对webpack的理解？解决了什么问题？ · Issue #121 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/121)

|  |
| --- |
| 现代前端开发已经变得十分的复杂，所以我们开发过程中会遇到如下的问题：   * 需要通过模块化的方式来开发 * 使用一些高级的特性来加快我们的开发效率或者安全性，比如通过ES6+、TypeScript开发脚本逻辑，通过sass、less等方式来编写css样式代码 * 监听文件的变化来并且反映到浏览器上，提高开发的效率 * JavaScript 代码需要模块化，HTML 和 CSS 这些资源文件也会面临需要被模块化的问题 * 开发完成后我们还需要将代码进行压缩、合并以及其他相关的优化   而webpack恰巧可以解决以上问题。 |
| webpack 是一个用于现代JavaScript应用程序的静态模块打包工具   * 静态模块   这里的静态模块指的是开发阶段，可以被 webpack 直接引用的资源（可以直接被获取打包进bundle.js的资源）  当 webpack 处理应用程序时，它会在内部构建一个依赖图，此依赖图对应映射到项目所需的每个模块（不再局限js文件），并生成一个或多个 bundle |

* webpack 构建流程

单看比较复杂，还是多看原文章，有机会自己用一下应该比较好理解

[webpack打包原理 ? 看完这篇你就懂了 ! - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/101541041" \l ":~:text=webpack %E6%9E%84%E5%BB%BA%E6%B5%81%E7%A8%8B Webpack %E7%9A%84%E8%BF%90%E8%A1%8C%E6%B5%81%E7%A8%8B%E6%98%AF%E4%B8%80%E4%B8%AA%E4%B8%B2%E8%A1%8C%E7%9A%84%E8%BF%87%E7%A8%8B,%E4%BB%8E%E5%90%AF%E5%8A%A8%E5%88%B0%E7%BB%93%E6%9D%9F%E4%BC%9A%E4%BE%9D%E6%AC%A1%E6%89%A7%E8%A1%8C%E4%BB%A5%E4%B8%8B%E6%B5%81%E7%A8%8B : 1 %E5%88%9D%E5%A7%8B%E5%8C%96%E5%8F%82%E6%95%B0%EF%BC%9A%E4%BB%8E%E9%85%8D%E7%BD%AE%E6%96%87%E4%BB%B6%E5%92%8C Shell,%E7%BF%BB%E8%AF%91%E5%AE%8C%E6%89%80%E6%9C%89 ... 6 %E8%BE%93%E5%87%BA%E8%B5%84%E6%BA%90%EF%BC%9A%E6%A0%B9%E6%8D%AE%E5%85%A5%E5%8F%A3%E5%92%8C%E6%A8%A1%E5%9D%97%E4%B9%8B%E9%97%B4%E7%9A%84%E4%BE%9D%E8%B5%96%E5%85%B3%E7%B3%BB,%E7%BB%84%E8%A3%85%E6%88%90%E4%B8%80%E4%B8%AA%E4%B8%AA%E5%8C%85%E5%90%AB%E5%A4%9A%E4%B8%AA ... 7 %E8%BE%93%E5%87%BA%E5%AE%8C%E6%88%90%EF%BC%9A%E5%9C%A8%E7%A1%AE%E5%AE%9A%E5%A5%BD%E8%BE%93%E5%87%BA%E5%86%85%E5%AE%B9%E5%90%8E,%E6%A0%B9%E6%8D%AE%E9%85%8D%E7%BD%AE%E7%A1%AE%E5%AE%9A%E8%BE%93%E5%87%BA%E7%9A%84%E8%B7%AF%E5%BE%84%E5%92%8C%E6%96%87%E4%BB%B6%E5%90%8D )

[面试官：说说webpack的构建流程? · Issue #122 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/122)

|  |
| --- |
| 从启动到结束会依次执行以下三大步骤：   * 初始化流程：从配置文件和 Shell 语句中读取与合并参数，并初始化需要使用的插件和配置插件等执行环境所需要的参数 * 编译构建流程：从 Entry 发出，针对每个 Module 串行调用对应的 Loader 去翻译文件内容，再找到该 Module 依赖的 Module，递归地进行编译处理 * 输出流程：对编译后的 Module 组合成 Chunk，把 Chunk 转换成文件，输出到文件系统 |
|  |

* webpack 中 编译 生命周期

|  |
| --- |
| 关于整个编译生命周期钩子，有如下：   * entry-option ：初始化 option * run * compile： 真正开始的编译，在创建 compilation 对象之前 * compilation ：生成好了 compilation 对象 * make 从 entry 开始递归分析依赖，准备对每个模块进行 build * after-compile： 编译 build 过程结束 * emit ：在将内存中 assets 内容写到磁盘文件夹之前 * after-emit ：在将内存中 assets 内容写到磁盘文件夹之后 * done： 完成所有的编译过程 * failed： 编译失败的时候 |

* webpack 中 loader

[面试官：说说webpack中常见的Loader？解决了什么问题？ · Issue #123 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/123)

|  |
| --- |
| loader 用于对模块的"源代码"进行转换，在 import 或"加载"模块时预处理文件  webpack做的事情，仅仅是分析出各种模块的依赖关系，然后形成资源列表，最终打包生成到指定的文件中。如下图所示：  在webpack内部中，任何文件都是模块，不仅仅只是js文件  默认情况下，在遇到import或者load加载模块的时候，webpack只支持对js文件打包  像css、sass、png等这些类型的文件的时候，webpack则无能为力，这时候就需要配置对应的loader进行文件内容的解析  在加载模块的时候，执行顺序如下：  [IMG_257](https://camo.githubusercontent.com/0f03b1163bc8559e534a0c51c9fa85cb55def06d16802a3d718b63a774669ffb/68747470733a2f2f7374617469632e7675652d6a732e636f6d2f39633263343362302d613666662d313165622d383566362d3666616337376330633962332e706e67)  当 webpack 碰到不识别的模块的时候，webpack 会在配置的中查找该文件解析规则  关于配置loader的方式有三种：   * 配置方式（推荐）：在 webpack.config.js文件中指定 loader * 内联方式：在每个 import 语句中显式指定 loader * CLI 方式：在 shell 命令中指定它们 |

* webpack 中 loader 配置与使用

|  |
| --- |
| **配置方式**  关于loader的配置，我们是写在module.rules属性中，属性介绍如下：  rules是一个数组的形式，因此我们可以配置很多个loader  每一个loader对应一个对象的形式，对象属性test 为匹配的规则，一般情况为正则表达式  属性use针对匹配到文件类型，调用对应的 loader 进行处理  代码编写，如下形式：  module.exports = {  module: {  rules: [  {  test: /\.css$/,  use: [  { loader: 'style-loader' },  {  loader: 'css-loader',  options: {  modules: true  }  },  { loader: 'sass-loader' }  ]  }  ]  }}; |
| 从上述代码可以看到，在处理css模块的时候，use属性中配置了三个loader分别处理css文件  因为loader 支持链式调用，链中的每个loader会处理之前已处理过的资源，最终变为js代码。顺序为相反的顺序执行，即上述执行方式为sass-loader、css-loader、style-loader  除此之外，loader的特性还有如下：   * loader 可以是同步的，也可以是异步的 * loader 运行在 Node.js 中，并且能够执行任何操作 * 除了常见的通过 package.json 的 main 来将一个 npm 模块导出为 loader，还可以在 module.rules 中使用 loader 字段直接引用一个模块 * 插件(plugin)可以为 loader 带来更多特性 * loader 能够产生额外的任意文件   可以通过 loader 的预处理函数，为 JavaScript 生态系统提供更多能力。用户现在可以更加灵活地引入细粒度逻辑，例如：压缩、打包、语言翻译和更多其他特性 |

* webpack 中常用 loader

|  |
| --- |
| 常见的loader如下：   * style-loader: 将css添加到DOM的内联样式标签style里 * css-loader :允许将css文件通过require的方式引入，并返回css代码 * less-loader: 处理less * sass-loader: 处理sass * postcss-loader: 用postcss来处理CSS * autoprefixer-loader: 处理CSS3属性前缀，已被弃用，建议直接使用postcss * file-loader: 分发文件到output目录并返回相对路径 * url-loader: 和file-loader类似，但是当文件小于设定的limit时可以返回一个Data Url * html-minify-loader: 压缩HTML * babel-loader :用babel来转换ES6文件到ES5 |

* webpack 中 plugin / 配置

[面试官：说说webpack中常见的Plugin？解决了什么问题？ · Issue #124 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/124)

|  |
| --- |
| webpack中的plugin也是如此，plugin赋予其各种灵活的功能，例如打包优化、资源管理、环境变量注入等，它们会运行在 webpack 的不同阶段（钩子 / 生命周期），贯穿了webpack整个编译周期  [IMG_256](https://camo.githubusercontent.com/8906dd3044545b1226482931c15071f36991ca0ef9d8889f8d55f9b02fbbe727/68747470733a2f2f7374617469632e7675652d6a732e636f6d2f39613034656334302d613763322d313165622d616239302d6439616538313462323430642e706e67)  目的在于解决loader 无法实现的其他事 **配置方式** 这里讲述文件的配置方式，一般情况，通过配置文件导出对象中plugins属性传入new实例对象。如下所示：  const HtmlWebpackPlugin = require('html-webpack-plugin'); // 通过 npm 安装  const webpack = require('webpack'); // 访问内置的插件  module.exports = {  ...  plugins: [  new webpack.ProgressPlugin(),  new HtmlWebpackPlugin({ template: './src/index.html' }),  ],}; |

* webpack 中 plugin 本质调用

|  |
| --- |
| 其本质是一个具有apply方法javascript对象  apply 方法会被 webpack compiler 调用，并且在整个编译生命周期都可以访问 compiler 对象  const pluginName = 'ConsoleLogOnBuildWebpackPlugin';  class ConsoleLogOnBuildWebpackPlugin {  apply(compiler) {  compiler.hooks.run.tap(pluginName, (compilation) => {  console.log('webpack 构建过程开始！');  });  }}  module.exports = ConsoleLogOnBuildWebpackPlugin;  compiler hook 的 tap 方法的第一个参数，应是驼峰式命名的插件名称 |

* webpack 中常用 plugin

[webpack常用plugins - 掘金 (juejin.cn)](https://juejin.cn/post/6844903935917080590)

|  |
| --- |
| autoprefixer 是个前端都能懂，就是css自动加浏览器前缀，配合postcss使用，如果想更加详细的postcss配置，请移步移动端适配。  html-webpack-plugin 这个插件就是为你生成一个已经自动注入打包后的js的html文件  case-sensitive-paths-webpack-plugin 这个插件就是防止不同的系统下对于大小写的问题导致路径出错  InterpolateHtmlPlugin 这个插件是配合html-webpack-plugin一起使用的,允许你在index.html中使用变量 |

* webpack 中 loader 和 plugin 区别

[webpack中loader 和 plugin 的主要区别 - 简书 (jianshu.com)](https://www.jianshu.com/p/28b42927d4af)

|  |
| --- |
| 1、loader 用于加载某些资源文件。因为 webpack 只能理解 JavaScript 和 JSON 文件，对于其他资源例如 css，图片，或者其他的语法集，比如 jsx， coffee，是没有办法加载的。 这就需要对应的loader将资源转化，加载进来。从字面意思也能看出，loader是用于加载的，它作用于一个个文件上。  2、plugin 用于扩展webpack的功能。目的在于解决loader无法实现的其他事,它直接作用于 webpack，扩展了它的功能。当然loader也是变相的扩展了 webpack ，但是它只专注于转化文件（transform）这一个领域。而plugin的功能更加的丰富，而不仅局限于资源的加载。  可以看到，两者在运行时机上的区别：   * loader 运行在打包文件之前 * plugins 在整个编译周期都起作用 |

* 如何编写 loader 和 plugin

[面试官：说说Loader和Plugin的区别？编写Loader，Plugin的思路？ · Issue #125 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/125)

* webpack 的热更新

[面试官：说说webpack的热更新是如何做到的？原理是什么？ · Issue #126 · febobo/web-interview (github.com)](https://github.com/febobo/web-interview/issues/126)

[Webpack HMR 原理解析 - 知乎 (zhihu.com)](https://zhuanlan.zhihu.com/p/30669007)

|  |
| --- |
| 当你对代码进行修改并保存后，webpack 将对代码重新打包，并将新的模块发送到浏览器端，浏览器通过新的模块替换老的模块，这样在不刷新浏览器的前提下就能够对应用进行更新。  例如，在开发 Web 页面过程中，当你点击按钮，出现一个弹窗的时候，发现弹窗标题没有对齐，这时候你修改 CSS 样式，然后保存，在浏览器没有刷新的前提下，标题样式发生了改变。感觉就像在 Chrome 的开发者工具中直接修改元素样式一样。 |
| 关于webpack热模块更新的总结如下：   * 通过webpack-dev-server创建两个服务器：提供静态资源的服务（express）和Socket服务 * express server 负责直接提供静态资源的服务（打包后的资源直接被浏览器请求和解析） * socket server 是一个 websocket 的长连接，双方可以通信 * 当 socket server 监听到对应的模块发生变化时，会生成两个文件.json（manifest文件）和.js文件（update chunk） * 通过长连接，socket server 可以直接将这两个文件主动发送给客户端（浏览器） * 浏览器拿到两个新的文件后，通过HMR runtime机制，加载这两个文件，并且针对修改的模块进行更新 |

* webpack 配置单页和多页

[webpack 之 单页面、多页面打包配置篇 - 掘金 (juejin.cn)](https://juejin.cn/post/7077179971772350494)

[webpack 的核心概念和构建流程 - 云+社区 - 腾讯云 (tencent.com)](https://cloud.tencent.com/developer/article/2018884)

|  |
| --- |
| **1.单页应用**  一个单页应用需要配置一个entry指明执行入口，web-webpack-plugin里的WebPlugin可以自动的完成这些工作：webpack会为entry生成一个包含这个入口的所有依赖文件的chunk，但是还需要一个html来加载chunk生成的js，如果还提取出css需要HTML文件中引入提取的css。  一个简单的webpack配置文件例子：  const { WebPlugin } = require('web-webpack-plugin');  module.exports = {  entry: {  app: './src/doc/index.js',  home: './src/doc/home.js'  },  plugins: [  // 一个WebPlugin对应生成一个html文件  new WebPlugin({  //输出的html文件名称  filename: 'index.html',  //这个html依赖的`entry`  requires: ['app','home'],  }),  ],};  复制  说明：require: ['app', 'home']指明这个html依赖哪些entry，entry生成的js和css会自动注入到html中。  还支持配置这些资源注入方式，支持如下属性：   1. \_dist只有在生产环境中才引入的资源； 2. \_dev只有在开发环境中才引入的资源； 3. \_inline把资源的内容潜入到html中； 4. \_ie只有IE浏览器才需要引入的资源。 |
| **一个[项目管理](https://cloud.tencent.com/product/coding-pm?from=10680" \t "https://cloud.tencent.com/developer/article/_blank)多个单页面**  一个项目中会包含多个单页应用，虽然多个单页面应用可以合成一个，但是这样做会导致用户没有访问的部分也加载了，如果项目中有很多的单页应用。为每一个单页应用配置一个entry和WebPlugin？如果又新增，又要新增webpack配置，这样做麻烦，这时候有一个插件web-webpack-plugin里的AutoWebPlugin方法可以解决这些问题。  module.exports = {  plugins: [  // 所有页面的入口目录  new AutoWebPlugin('./src/'),  ]};  复制  ****分析：****   1. AutoWebPlugin会把./src/目录下所有每个文件夹作为一个单页页面的入口，自动为所有的页面入口配置一个 WebPlugin 输出对应的html。 2. 要新增一个页面就在 ./src/ 下新建一个文件夹包含这个单页应用所依赖的代码，AutoWebPlugin 自动生成一个名叫文件夹名称的html文件。 |

* webpack 配置按需加载

|  |
| --- |
| Webpack 内置了强大的分割代码的功能去实现按需加载，实现起来非常简单。  举个例子，现在需要做这样一个进行了按需加载优化的网页：   * 网页首次加载时只加载 main.js 文件，网页会展示一个按钮，main.js 文件中只包含监听按钮事件和加载按需加载的代码。 * 当按钮被点击时才去加载被分割出去的 show.js 文件，加载成功后再执行 show.js 里的函数。   其中 main.js 文件内容如下：  window.document.getElementById('btn').addEventListener('click', function () {  // 当按钮被点击后才去加载 show.js 文件，文件加载成功后执行文件导出的函数  import(/\* webpackChunkName: "show" \*/ './show').then((show) => {  show('Webpack');  })  });  show.js 文件内容如下：  module.exports = function (content) {  window.alert('Hello ' + content);  };  代码中最关键的一句是 import(/\* webpackChunkName: "show" \*/ './show')，Webpack 内置了对 import(\*) 语句的支持，当 Webpack 遇到了类似的语句时会这样处理：   * 以 ./show.js 为入口新生成一个 Chunk； * 当代码执行到 import 所在语句时才会去加载由 Chunk 对应生成的文件。 * import 返回一个 Promise，当文件加载成功时可以在 Promise 的 then 方法中获取到 show.js 导出的内容。 |

* webpack 的 sourceMap 作用
* package.json 字段含义
* 自己发布过 npm 包吗
* wepack 许多配置简单说明介绍

[深入webpack4配置笔记(必备/可选配置 单页/多页配置)\_ 一期一会的博客-CSDN博客](https://blog.csdn.net/qq_34832846/article/details/89154006)

|  |
| --- |
|  |