Daily_Learning(2020-3-17_webpack)

1.有哪些常见的Plugin？你用过哪些Plugin?

- define-plugin：定义环境变量 (Webpack4 之后指定 mode 会自动配置)
- ignore-plugin：忽略部分文件
- html-webpack-plugin：简化 HTML 文件创建 (依赖于 html-loader)
- web-webpack-plugin：可方便地为单页应用输出 HTML，比 html-webpack-plugin 好用
- uglifyjs-webpack-plugin：不支持 ES6 压缩 (Webpack4 以前)
- terser-webpack-plugin: 支持压缩 ES6 (Webpack4)
- webpack-parallel-uglify-plugin: 多进程执行代码压缩，提升构建速度
- mini-css-extract-plugin: 分离样式文件，CSS 提取为独立文件，支持按需加载 (替代 extract-text-webpack-plugin)
- serviceworker-webpack-plugin：为网页应用增加离线缓存功能
- clean-webpack-plugin: 目录清理
- ModuleConcatenationPlugin: 开启 Scope Hoisting
- speed-measure-webpack-plugin: 可以看到每个 Loader 和 Plugin 执行耗时 (整个打包耗时、每个 Plugin 和 Loader 耗时)
- webpack-bundle-analyzer: 可视化 Webpack 输出文件的体积 (业务组件、依赖第三方模块)

官网：https://webpack.docschina.org/loaders/

说一说Loader和Plugin的区别?

- Loader 本质就是一个函数，在该函数中对接收到的内容进行转换，返回转换后的结果。因为 Webpack 只认识 JavaScript，所以 Loader 就成了翻译官，对其他类型的资源进行转译的预处理工作。

- Plugin 就是插件，基于事件流框架 Tapable，插件可以扩展 Webpack 的功能，在 Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。

- Loader 在 module.rules 中配置，作为模块的解析规则，类型为数组。每一项都是一个 Object，内部包含了 test(类型文件)、loader、options (参数)等属性。

- Plugin 在 plugins 中单独配置，类型为数组，每一项是一个 Plugin 的实例，参数都通过构造函数传入。

Webpack构建流程简单说一下

Webpack 的运行流程是一个串行的过程，从启动到结束会依次执行以下流程：

- 初始化参数：从配置文件和 Shell 语句中读取与合并参数，得出最终的参数
- 开始编译：用上一步得到的参数初始化 Compiler 对象，加载所有配置的插件，执行对象的 run 方法开始执行编译
- 确定入口：根据配置中的 entry 找出所有的入口文件
- 编译模块：从入口文件出发，调用所有配置的 Loader 对模块进行翻译，再找出该模块依赖的模块，再递归本步骤直到所有入口依赖的文件都经过了本步骤的处理
- 完成模块编译：在经过第4步使用 Loader 翻译完所有模块后，得到了每个模块被翻译后的最终内容以及它们之间的依赖关系
- 输出资源：根据入口和模块之间的依赖关系，组装成一个个包含多个模块的 Chunk，再把每个 Chunk 转换成一个单独的文件加入到输出列表，这步是可以修改输出内容的最后机会
- 输出完成：在确定好输出内容后，根据配置确定输出的路径和文件名，把文件内容写入到文件系统

在以上过程中，Webpack 会在特定的时间点广播出特定的事件，插件在监听到感兴趣的事件后会执行特定的逻辑，并且插件可以调用 Webpack 提供的 API 改变 Webpack 的运行结果。

简单说

- 初始化：启动构建，读取与合并配置参数，加载 Plugin，实例化 Compiler
- 编译：从 Entry 出发，针对每个 Module 串行调用对应的 Loader 去翻译文件的内容，再找到该 Module 依赖的 Module，递归地进行编译处理
- 输出：将编译后的 Module 组合成 Chunk，将 Chunk 转换成文件，输出到文件系统中

文件监听原理

在发现源码发生变化时，自动重新构建出新的输出文件。

Webpack开启监听模式，有两种方式：

- 启动 webpack 命令时，带上 --watch 参数
- 在配置 webpack.config.js 中设置 watch:true

缺点：每次需要手动刷新浏览器

原理：轮询判断文件的最后编辑时间是否变化，如果某个文件发生了变化，并不会立刻告诉监听者，而是先缓存起来，等 aggregateTimeout 后再执行。

```
module.export = {
// 默认false,也就是不开启
watch: true,
// 只有开启监听模式时，watchOptions才有意义
watchOptions: {
// 默认为空，不监听的文件或者文件夹，支持正则匹配
ignored: /node_modules/,
// 监听到变化发生后会等300ms再去执行，默认300ms
aggregateTimeout:300,
// 判断文件是否发生变化是通过不停询问系统指定文件有没有变化实现的，默认每秒问1000次
poll:1000
}
}

```

Webpack 的热更新原理

Webpack 的热更新又称热替换（Hot Module Replacement），缩写为 HMR。 这个机制可以做到不用刷新浏览器而将新变更的模块替换掉旧的模块。

HMR的核心就是客户端从服务端拉去更新后的文件，准确的说是 chunk diff（chunk 需要更新的部分），实际上 WDS 与浏览器之间维护了一个 Websocket，当本地资源发生变化时，WDS 会向浏览器推送更新，并带上构建时的 hash，让客户端与上一次资源进行对比。客户端对比出差异后会向 WDS 发起 Ajax 请求来获取更改内容(文件列表、hash)，这样客户端就可以再借助这些信息继续向 WDS 发起 jsonp 请求获取该chunk的增量更新。

后续的部分(拿到增量更新之后如何处理？哪些状态该保留？哪些又需要更新？)由 HotModulePlugin 来完成，提供了相关 API 以供开发者针对自身场景进行处理，像react-hot-loader 和 vue-loader 都是借助这些 API 实现 HMR。
详细解析

Babel原理

大多数JavaScript Parser遵循 estree 规范，Babel 最初基于 acorn 项目(轻量级现代 JavaScript 解析器) Babel大概分为三大部分：

解析：将代码转换成 AST

- 词法分析：将代码(字符串)分割为token流，即语法单元成的数组
- 语法分析：分析token流(上面生成的数组)并生成 AST

转换：访问 AST 的节点进行变换操作生产新的 AST

- Taro就是利用 babel 完成的小程序语法转换

生成：以新的 AST 为基础生成代码