1.

webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)。当 webpack 处理应用程序时，它从配置所指定的入口开始(entry),它会递归地构建一个依赖关系图(dependency graph)，其中包含应用程序需要的每个模块，然后将所有这些模块打包成一个或多个bundle。

Webpack的四大核心概念

**入口（entry）**:指示 webpack 应该使用哪个模块，来作为构建其内部依赖图的开始。进入入口起点后，webpack 会找出有哪些模块和库是入口起点（直接和间接）依赖的。每个依赖项随即被处理，最后输出到称之为 bundles 的文件中

例如在webpack.config.js中：

module.exports = {

entry: ‘./path/to/my/entry/file.js'

}

**出口(output):**

output 属性告诉 webpack 在哪里输出它所创建的 bundles，以及如何命名这些文件，默认值为 ./dist。基本上，整个应用程序结构，都会被编译到你指定的输出路径的文件夹中。

例如在webpack.config.js中：

module.exports = {

entry: './path/to/my/entry/file.js',

output: {

path: path.resolve(\_\_dirname, 'dist'),

filename: 'my-first-webpack.bundle.js'

}

};

在上面代码中，通过执行ouput.filename以及output.path属性，配置webpack bundle讲来输出bundle的名称以及bundle将输出到那个文件夹路径。

**加载器（loader）:**

loader 让 webpack 能够去处理那些非 JavaScript 文件（webpack 自身只理解 JavaScript，例如txt文件，css文件等）。loader 可以将所有类型的文件转换为 webpack 能够处理的有效[模块](https://www.webpackjs.com/concepts/modules)，然后你就可以利用 webpack 的打包能力，对它们进行处理。

本质上，webpack loader 将所有类型的文件，转换为应用程序的依赖图（和最终的 bundle）可以直接引用的模块。

在更高层面，在 webpack 的配置中 loader 有两个目标：

test 属性，用于标识出哪些文件应该被转换。

use 属性，表示对这些标识出的文件转换时，应该使用哪个 loader。

例如在webpack.config.js中：

const path = require('path');

const config = {

output: {

filename: 'my-first-webpack.bundle.js'

},

module: {

rules: [

{test: /\.txt$/, use: 'raw-loader'}

]

}

};

module.exports = config;

以上配置中，对一个单独的 module 对象定义了 rules 属性，里面包含两个必须属性：test 和 use。其中test标识出对所有.txt后缀的文件，use属性说明了对这些表识的txt文件采用raw-loader。

**插件（plugins）**

loader 被用于转换某些类型的模块，而插件则可以用于执行范围更广的任务。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](https://www.webpackjs.com/api/plugins)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，然后把它添加到 plugins 数组中。多数插件可以通过选项(option)自定义。你也可以在一个配置文件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。

例如在webpack.config.js中：

const HtmlWebpackPlugin = require('html-webpack-plugin'); // 通过 npm 安装

const webpack = require('webpack'); // 用于访问内置插件

const config = {

module: {

rules: [

{test: /\.txt$/, use: 'raw-loader'}

]

},

plugins: [

new HtmlWebpackPlugin({template: './src/index.html'})

]

};

module.exports = config;

**模式(mode)：**

通过选择 development 或 production 之中的一个，来设置 mode 参数，你可以启用相应模式下的 webpack 内置的优化.默认值是production

module.exports = {

mode: 'production'

};

**Externals:**

如果我们想引用一个库，但是又不想让webpack打包，并且又不影响我们在程序中以CMD、AMD或者window/global全局等方式进行使用，那就可以通过配置externals。

**额外一些重要的配置(Webpack-dev-server配置)**：

module.exports = {

………

devServer: {

contentBase: \_\_dirname + "/public/"

}

}

devServer字段，用于配置webpack-dev-server

--devServer中的contentBase, 它指定了服务器资源的根目录，如果不写入contentBase的值，那么contentBase默认是项目的目录。

webpack-dev-server将从这个目录寻找index.html文件，如果不存在则报错Cannot GET /

--报错Failed to load resource: the server responded with a stauts of 404 (NotFound),这是由于html中无法加载webpack打包的bundle.js文件。解决方法为检查html文件引入bundle.js的路径是否正确。

--webpack打包和webpack-dev-server开启服务的区别

Webpack命令输出真实的文件，而webpack-dev-server命令打包输出的文件只存在于内存中,**不输出真实的文件**

--webpack-dev-server启动时候，通过制定—configure选项来使用webpack.config.js后，我们在项目中修改文件，webpack-dev-server会自动应用最新的更新，不需要重启webpack-dev-server.

devServer: {

port:7000

}

设置端口为7000而非默认的8080端口。

host设置服务器的主机号。

devServer: {

host:‘0.0.0.0’

}

devServer: {

contentBase: path.join(\_\_dirname, "dist"),

historyApiFallback: {

rewrites: [

{from:/./, to:'/404.html'}

]

}

}

}

这个配置属性是用来应对返回404页面时定向到特定页面

module.exports = {

/\*这里省略entry和output，参照上面写的内容\*/

devServer: {

contentBase: path.join(\_\_dirname, "dist"),

overlay: true

}

}

这个配置属性用来在编译出错的时候，是否在浏览器页面上显示错误，默认是false，可设置为true显示错误。

devServer: {

contentBase: path.join(\_\_dirname, "dist"),

stats: "errors-only"

}

这个配置属性用来控制编译的时候shell上的输出内容, 默认情况下会有许多信息，当我们设置为errors-only以后只打印错误。

-🡪错误信息：webpack Dev Server Invalid Options， 由于devServer中出现了配置意外的key字段引起，例如stats：“read-only”写成了states:”read-only”

2.

执行webpack 不含任何参数，则webpack会根据命令行当前目录下的webpack.config.js来配置，并输出bundle文件。

3.

webpack-dev-server是一个小型的Node.js Express服务器。使用它可以为webpack打包生成的资源文件提供web服务。默认的发布端口为8080。

webpack-dev-server 主要提供两个功能：

* 为静态文件提供服务

自动刷新和热替换(HMR)。

自动刷新是指当我们对业务代码做了一些修改然后保存后（command+s），页面会自动刷新，我们所做的修改会直接同步到页面上，而不需要我们刷新页面，或重新开启服务。

模块热替换是指在热替换（HMR）机制里，不是重载整个页面，HMR程序会只加载被更新的那一部分模块，然后将其注入到运行中的APP中。热替换以后要刷新才会显示不同的页面，所以一般模块热替换和自动刷新一起使用。

4.

我们可以在CLI启动webpack-dev-server中配置webpack-dev-server的许多方面。比较推荐的做法是，采用webpack-config.js的方法进行配置，然后使用—config的方法进行读取配置。

例如：

"webpack-dev-server --config webpack.config.dev.js"

（使用配置文件webpack.config.dev.js配置来启动webpack-dev-server服务器）

"webpack-dev-server --inline"

（采用inline模式启动webpack-dev-server服务器）

webpack-dev-server --port 8000

（启动webpack-dev-server服务器并且端口为8000）

5.

Npm + webpack或者webpack-dev-server连用是常用的模式，例如在package.json中配置：

"scripts": {

"start": "webpack-dev-server"

}

那么当npm run start以后，我们实际上是执行webpack-dev-server，然后webpack-dev-server进行打包并且建立一个web服务器。

6.