1. 建立dist文件
2. 建立src文件 ：

2.1、在src目录下件css文件

2.2、建js文件

2.3、建images文件

建立主页面 index.html

建立main.js引入 相关的js文件

1. wepack的安装配置 ：

首先进行初始化 ： npm init -y

局部安装webpack： npm install --save-dev [webpack@3.6.0](mailto:webpack@3.6.0)

执行webpack命令：node\_modules\.bin\webpack app\main.js public\bundle.js

在根目录下创建webpack.config.js文件： 配置出口和入口文件

module.exports = {  
 entry: \_\_dirname + "/app/main.js",*//已多次提及的唯一入口文件* output: {  
 path: \_\_dirname + "/public",*//打包后的文件存放的地方* filename: "bundle.js"*//打包后输出文件的文件名* }  
}

此时直接运行webpack命令即可。

1. 创建webpack-dev-server热加载工具 ：

指定命令 ： npm install [webpack@2.9.3](mailto:webpack@2.9.3) --save-dev

在packjson.json中配置如下信息 ：添加 dev字段

"scripts": {  
 "test": "echo \"Error: no test specified\" && exit 1",  
 "dev":"webpack-dev-server"  
},

在命令行运行 npm run dev 即可执行该工具。当文件中有变化时，会自动进行编译

由于其将代码部署到本地服务器进行运行，所以需要引用服务器中的bundle.js

**<script src="http://localhost:8080/bundle.js"></script>**

Webpack-dev-server 打包的bundle.js并没有放到实际的物理磁盘中，而是放到了电脑内存中，默认在根目录下有一个虚拟的bundle.js

在dev中加入参数 ： --open 启动时直接打开浏览器 --hot 有更改时进行自动刷新

--port 3000 改变端口号

"dev":"webpack-dev-server --open --hot"

1. 下载html-webpack-plugin 插件。可以自动将我们执行的模板页面加载到内存中，并且帮助我们自动引入bundle.js文件。

下载 ： npm install html-webpack-plugin --save-dev

在webpack.config.js 中进行插件配置 ：

引入插件

const htmlWebpackPlugin = require('html-webpack-plugin')

进行配置 ：在mudel.exports中

plugins: [  
 new htmlWebpackPlugin({

*// 创建一个在内存中生成HTML页面的插件* template: path.join(\_\_dirname,'./public/index.html'), *// 指定要加载到内存中的模板页面* filename:'index.html' *// 指定模板页面名称* })  
],

1. 引入css样式：

在main.js中直接进行配置导入css文件

Import css文件路径

下载loader：

Npm install style-loader --save-dev

Npm install css-loader --save-dev

在webpack.config.js中进行配置 ：

module: { *// 用于配置第三方模块加载器* rules: [  
 {  
 test : /\.css$/, *// 匹配以css结尾的文件* use:['style-loader','css-loader'] *// 需要的加载器* }  
 ]  
}

1. 对于图片路径进行设置打包 ：

下载loader ： url-loader file-loader

Npm install url-loader file-loader --save-dev

在webpack.config.js中进行配置 ：

{  
 test:/\.jpg|png|gif$/,  
 use:'url-loader?limit=8192’  
}

参数limit进行限制 ： 当图片小于8kb时，使用Base64位进行编码

use:'url-loader?limit=8192&name=[hash:8]-[name].[ext]'

上述图片命名方式为 ： 8位哈希值 + 图片原名

1. 对于部分高级语法需要使用相应的loader进行转化

*1、下载相关的插件和包  
\* babel-core babel-loader babel-plugin-transform-runtime  
\* babel-preset-env babel-preset-stage-0  
\* 2、在webpack.config.js文件rules添加加载器  
\* test:/\.js$/, use:'babel-loader',exclude:/node\_modules/  
\* 3、在根目录下创建.babelrc 文件，这个文件时是 JSON 格式，不能有注释  
\* {  
\* presets:["env","stage-0"],  
\* plugins:["transform-runtime"],  
\* }  
\**

1. webpack构建webpack项目 ：

下载vue : npm i vue -s

在main.js中引入vue.js文件。

import Vue from 'vue' *// 此时倒入的不是最全的vue.js文件，只是运行时的vue.js，需要在webpack中进行配置  
/\*import Vue from '../node\_modules/vue/dist/vue.js' \*/ // 该种方式倒入的是最全的vue*

在webpack.config.js中配置相关的引入全包问题 ：

resolve: {  
 alias:{  
 "vue$":"vue/dist/vue.js" *// 以vue结尾的去改路径下引入文件* }  
}

1. webpack通过render构建组件的方式 ：

下载两个安装包：npm install vue-loader vue-template-compiler --save-dev

创建一个以.vue结尾的文件，代表一个组件，组件内容有三部分

<template> html代码 </template>

<script> 业务 </scrit>

<style> 样式 </style>

在webpack.config.js中引入loader，和组件

const VueLoaderPlugin = require('vue-loader/lib/plugin');

plugins: [  
 new VueLoaderPlugin()  
],

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

在main.js中：

import login from './login.vue'  
var vm = new Vue({  
 el:"#app",  
 data:{   
 },  
 render: function(createElements) {  
 return createElements(login)  
 }  
})

1. 设置组件的样式 ：

**<style scoped>  
 div{  
 background-color: red;  
 }  
</style>**

需要添加scoped 的属性，只将当前的组件进行样式更改，若不加该属性，则将进行全局的更改。