什么是前端工程化

前端工程化: 在企业级的前端项目开发中，把前端开发所需要的工具，技术，流程，经验等进行规范化，标准化。

- 模块化 （js的模块化,css的模块化,资源的模块化）
- 组件化（复用现有的UI结构，样式，行为）
- 规范化（目录结构的划分，编码规范化，接口规范化，文档规范化，Git分支管理）
- 自动化（自动化构建，自动化部署，自动化测试）

前端工程化的解决方案

早期的前端工程化解决方案：grunt ,gulp

目前主流的前端工程化解决方案: webpack

webpack 的基本使用

1.什么是webpack

概念 webpack 是前端项目工程化的具体解决方案。

主要功能：它提供了友好的前端模块化开发支持，以及代码压缩混淆，处理浏览器端javascript兼容性，性能优化等强大的功能。

优点: 让程序员把工作的中心放在具体功能的实现上，提高了前端开发效率和项目的可维护性。

注意：在vue和react等前端项目，基本上都是基于webpack进行工程化开发的。

2.创建新项目

1)、新建项目空白目录 ，并运行npm init -y 命令，初始化包管理配置文件 package.json

2)、新建src -> index.html 首页和 src -> index.js 脚本文件

3)、初始化首页基本的结构

4)、运行 npm install jquery -s 命令 ，安装jQuery （-s 即 --save 保存到packge.json依赖中，开发和运行都需要）

3.安装webpack

在终端运行如下的命令，安装webpack相关的两个包

npm install webpack@5.42.1 webpack-cli@4.9.0 -D （表示记录到 dev里面,只在开发阶段用到）
-D 是-save-dev的简写

4. 配置webpack

1）在项目根目录下创建 名为 webpack.config.js 的 webpack 配置文件，并初始化基本配置:

```
  module.exports = {
    mode:'development'//mode用来指定结构模式，可选值development 和 production
  }
```

2）在 pakage.json 的 script 节点下，新增dev脚本如下：

```
  "scripts":{
    "dev":"webpack"   //script 节点下的脚本，可以通过npm run 执行，例如 npm run dev
  }
```

3）在终端运行 npm run dev 命令，启动webpack进行项目的打包构建

4.1 mode的可选值

mode节点的可选值有两个，分别是：
1、development

- 开发环境
- 不会对打包生成的文件进行代码压缩和性能优化
- 打包速度快，适合在开发阶段使用

2、production

- 生产环境
- 会对打包生成的文件进行代码压缩和性能优化
- 打包速度很慢，仅适合在项目发布阶段使用

4.2 webpack中的默认约定

在webpack 4.x和5.x版本中，有如下默认的约定：
1、默认的打包入口文件位 src->index.js
2、默认的输出文件路径为 dist-main.js

注意： 可以在webpack.config.js中修改打包的默认约定

4.3 自定义打包的入口与输出

在 webpack.config.js 的配置文件中，通过 entry 节点指定打包的入口。通过 output 节点指定打包的输出。

示例如下：

```
  const path = require('path')  //导入node.js 中专门操作路径的模块

  module.exports = {
    entry: path.join(__dirname,'./src/index.js'),//打包入口文件路径
    output:{
      path:path.join(__dirname,'./dist'),  //输出文件存放路径
      filename:'bundle.js'   //输出文件的名称
    }
  }
```

webpack 中的插件

1. webpack插件的作用

通过安装和配置第三方的插件，可以拓展 webpack 的能力，从而让 webpack 用起来更加方便。最常用的 webpack 插件有如下两个：

1)、webpack-dev-server

- 类似于 node.js阶段用到的nodemon工具
- 每当修改了源代码，webpack 会自动进行项目的打包和构建

2)、html-webpack-plugin

- webpack中的HTML插件（类似一个模板引擎插件）
- 可以通过此插件自定制index.html页面的内容

1.1 安装webpack-dev-server

运行如下命令，即可在项目中安装此插件

```
  npm install webpack-dev-server@3.11.2 -D
```

1.2 配置webpack-dev-server

1） 修改 package.json -> scripts 中的dev命令如下：

```
"scripts":{
 "dev":"webpack serve", //script 节点下的脚本，可以通过npm run执行
}
```

2）再次运行 npm run dev 命令，重新进行项目的打包
3）在浏览器中访问 http://localhost:8080地址，查看自动打包效果

注意：webpack-dev-server 会启动一个实时打包的http服务器

将 index.html 页面的 js link 路径改成 /bundle.js，因为启动这个服务后用的是内存中的 bundle.js，而不是 dist 下的 bundle.js 这样才能监听到修改后的代码。

1.3 安装html-webpack-plugin

运行如下命令，即可在项目中安装此插件.

```
  npm install html-webpack-plugin@5.3.2 -D
```

1.4 配置html-webpack-plugin

```
//1.导入HTML插件，得到一个构造函数
const HtmlPlugin = require('html-webpack-plugin')

//2.创建HTML 插件的实例对象
const htmlPlugin = new HtmlPlugin({
 template: './src/index.html',  //指定原文件的存放路径
 filename: './index.html',      //指定生成文件的存放路径
})

module.exports = {
 mode: 'development',
 plugins: [htmlPlugin] , //3.通过plugins节点，使htmlPlugin插件生效
}
```

注意：再运行一边 npm run dev，这个复制的页面也会在内存中，不会显示（如果报错请考虑版本原因）

1.5 解惑html-webpack-plugin

1）通过HTML插件复制到项目根目录中的 index.html 页面，也放到了内存中
2）HTML插件在生成的 index.html 页面，自动注入了打包的 bundle.js 文件

1.6 devServer节点

在 webpack.config.js 配置文件中，可以通过 devServer 节点对 webpack-dev-server 插件进行更多的配置，示例代码如下：

```
  devServer: {
    open: true, //初次打包完成后，自动打开浏览器
    host: '127.0.0.1' ,//实是打包所使用的主机地址
    port:80, //实时打包使用的端口号（在http协议中，端口是80则可以省略不显示）
  }
```

注意：凡是修改了webpack.config.js 配置文件，或修改了package.json配置文件，必须重启实时打包的服务器，否则最新的配置文件无法生效

webpack 中的 loader

1. loader概述

在实际开发过程中，webpack 默认只能打包处理以.js后缀名结尾的模块。其他非.js后缀名结尾的模块，webpack 默认处理不了，需要调用 loader 加载器才可以正常打包，否则会报错！

loader 加载器的作用：协助 webpack 打包处理特定的文件模块。比如：

- css-loader 可以打包处理.css相关文件
- less-loader 可以打包处理 .less相关的文件
- babel-loader 可以打包处理 webpack无法处理的高级js语法

webpack 处理流程:
将要被 webpack 打包处理的文件模块 -> 是否为js模块
是js文件 -> 是否包含高级js语法 -> 没包含则直接webpack处理）包含了继续判断 -> 是否配置了babel 是 —> 调用loader处理 不是则报错
不是js文件 —> 是否配配置了对应文件的loader —> 是 调用loader处理 不是则报错

2.打包处理css文件

1）运行 npm i style-loader@3.0.0 css-loader@5.2.6 -D 命令，安装处理css文件的loader
2）在webpack.config.js 的module -> rules 数组中，添加 loader 规则如下：

```
  module:{  //所有第三方模块匹配规则
   rules:[    //文件后缀名的匹配规则
     {test:/\.css/,use:['style-loader','css-loader']}
   ]
  }
```

其中，test 表示匹配的文件类型，use 表示对应要调用的 loader
注意：

- use 数组中指定的 loader 顺序是固定的
- 多个loader的调用顺序是：从后往前调用
  1. webpack 默认只能打包处理.js结尾的文件，处理不了其他后缀的文件
  2. 由于代码中包含了index.css这个文件，因此webpack默认处理不了
  3. 当webpack发现某个文件处理不了，会查找webpack.config.js这个配置文件，看module.rules数组中，是否配置了对应的loader加载器。
  4. webpack把index.css这个文件，先转交给最后一个loader进行处理（先转交给css-loader）
  5. 当css-loader处理完毕之后，会把处理的结果，转交给下一个loader（转交给style-loader）
  6. 当style-loader处理完毕后，发现没有下一个loader了，于是就把处理的结果，转交给了webpack
  7. webpack把style-loader处理的结果，合并到/dist/bundle.js中，最终生成打包好的文件。

3.打包处理less文件

1）运行 npm i less-loader@10.0.1 less@1.1.1 -D 命令
2）在webpack.config.js的 module -> rules 数组中，添加loader规则如下:

```
  module:{ //所有第三方文件模块的匹配规则
   rules:[ //文件后缀名的匹配规则
     {test:/\.less$/,use:['style-loader','css-loader','less-loader']},
   ]
  }
```

可以使用vscode less插件将less自动转为css，则不需要以上配置。

4.base64图片的优缺点

优点：可以减少频繁的发起请求图片的网络请求，效果类似于精灵图（请求一次精灵图，然后通过定位展现不同的图片）
缺点：转成base64之后体积会变大
小图片适合使用base64，大图片不适合

5.打包处理样式表中与url路径相关的文件

1）运行 npm i url-loader@4.1.1 file-loader@6.2.0 -D 命令
2）在webpack.config.js 的module -> rules 数组中，添加loader规则如下：

```
  module:{
   rules:[
    {test:/\.jpg|png|gif$/,use:['url-loader?limit=22229']}
   ]
  }
```

其中 ? 之后是loader的参数项：

- limit用来指定图片的大小，单位是字节(byte)
- 只有<= limit大小的图片，才会被转为base64格式的图片,否则不会转为base64

6.打包处理js文件中的高级语法

webpack只能打包处理一部分高级的javascript语法。对弈那些webpack无法处理的高级js语法，需要借助于babel-loader进行打包处理。例如webpakc无法处理下面的javascript代码：

```
  //1.定义了名为info的装饰器
  function info(target){
   //2.为目标添加静态属性
   target.info = 'person info'
  }
  //3.为Person类应用info装饰器
  @info
  class Person{}

  //4.打印Person的静态属性info
  console.log(Person.info)
```

6.1 安装babel-loader相关的包

运行如下的命令安装对应的依赖包:
npm i babel-loader@8.2.2 @babel/core@7.14.6 @babel/plugin-proposal-decorators@7.14.5 -D

在webpack.config.js 的module -> rules数组中，添加loader规则如下：

```
//注意:必须使用exclude指定排除项：因为node_modules 目录下的第三方包不需要被打包
{test:/\.js$/,use:'babel-loader',exclude:/node_modules/}
```

6.2 配置 babel-loader

在项目根目录下，创建名为babel.config.js的配置文件，定义Babel的配置项如下:

```
  module.exports = {
   //声明babel可用的插件
   pulgins:[['@babel/plugin-proposal-decorators',{legacy:true}]]
  }
```

webpack 打包发布

1. 配置 webpack的打包发布

在package.json 文件的scripts节点下，新增build命令如下：

```
  "scripts":{
   "build":"webpack --mode production" //项目发布时，运行build命令
  }
```

—mode是一个参数项，用来指定webpack的运行模式。production代表生产环境，会对打包生成的文件进行代码压缩和性能优化。

注意：通过 --mode指定的参数项，会覆盖webpack.config.js 中的mode选项

2.把生成的bundle.js放到js文件夹下

```
output: {
  path: path.join(__dirname, './dist'),
  filename: 'js/bundle.js'   //加一层js/
},
```

3.把图片文件统一生成到image目录中

修改 webpack.config.js 中的 url-loader 配置项，新增 outputPath 选项即可指定图片文件的输出路径:

```
  {
 test:/\.jpg|png|gif$/,
 use:{
  loader:'url-loader',
  options:{
   limit:1000,
   //明确指定把打包生成的图片文件，存储到dist目录下的image文件夹中
   outputPath:'image',
  }
 }
}

//建议使用如下配置
{
  test: /\.png|jpg|gif$/,
  use: ['url-loader?limit=1000&outputPath=imagess']
},
```

4. 自动清理dist目录下的旧文件

为了在每次打包发布时自动清理掉 dist 目录下的旧文件，可以安装并配置 clean-webpack-plugin 插件:

```
  //1.安装清理dist目录下的webpack插件
  npm install clean-webpack-plugin@3.0.0 -D

  //2.按需导入插件、得到插件的构造函数值之后，创建插件的实例对象
  //左侧花括号是解构赋值
  const {CleanWebpackPlugin} = require('clean-webpack-plugin')
  const cleanPlugin = new CleanWebpackPlugin()

  //3.把创建的cleanPlugin 插件实例对象，挂载到plugins节点中
  plugins:[htmlPlugin,cleanPlugin], //挂载插件
```

Source Map

1. 什么是Source Map

Source Map 就是一个信息文件，里面存储着位置信息。也就是说，Source Map 文件中存储着压缩混淆后的代码，所对应的转换前的位置。
有了它，出错的时候，除错工具将直接显示原始代码，而不是转换后的代码，能够极大的方便后期的调试。

2. 解决默认Source Map的问题

开发环境下，推荐在 webpack.config.js 中添加如下的配置，即可保证运行时报错的行数与源代码的行数保持一致:

```
  module.exports = {
   mode: 'development',
   //eval-source-map 仅限在'开发模式'下使用，不建议在'生产模式'下使用。
   //此选项生成的Source Map 能够保证 '运行时报错的行数' 与 '源代码的行数'保持一致
   devtool:'eval-source-map',
   //省略其他的配置项...
  }
```

3. webpack生产环境下的Source Map

在生产环境下，如果省略了 devtool选项，则最终生成的文件中不包含Source Map。这能够防止原始代码通过Source Map形式暴露给别有所图之人。

3.1 只定位行数不暴露源码

在生产环境下，如果只想定位报错的具体行数，且不想暴露源码。此时可以将devtool 的值设置为 nosources-source-map 。

4. Source Map 总结

1)开发环境下：

建议把 devtool 的值设置为 eval-source-map
好处：可以精准定位到具体的错误行

2）生产环境下：
建议关闭Source Map 或者将devtool 的值设置为 nosources-source-map
好处：防止源码泄露，提高网站的安全性