1. webpack3和webpack4的区别

1.1. mode/–mode参数 新增了mode/--mode参数来表示是开发还是生产（development/production） production 侧重于打包后的文件大小，development侧重于goujiansud 1.2. 移除loaders，必须使用rules（在3版本的时候loaders和rules 是共存的但是到4的时候只允许使用rules） 1.3. 移除了CommonsChunkPlugin (提取公共代码)，用optimization.splitChunks和optimization.runtimeChunk来代替 1.4. 支持es6的方式导入JSON文件，并且可以过滤无用的代码

```
let jsonData = require('./data.json')



import jsonData from './data.json'



import { first } from './data.json' // 打包时只会把first相关的打进去
```

1.5. 升级happypack插件（happypack可以进行多线程加速打包） 1.6. ExtractTextWebpackPlugin调整，建议选用新的CSS文件提取kiii插件mini-css-extract-plugin，production模式，增加 minimizer

2. loader 和 plugin 不同

2.1. loader是使wenbpack拥有加载和解析非js文件的能力 2.2. plugin 可以扩展webpack的功能，使得webpack更加灵活。可以在构建的过程中通过webpack的api改变输出的结果

3. webpack构建流程

3.1. 初始化参数，从配置文件和shell语句中读到的参数合并，得到最后的参数 3.2. 开始编译：用合并得到的参数初始化complier对象，加载是所有配置的插件，执行run方法开始编译 3.3. 确定入口，通过entry找到入口文件 3.4. 编译模块，从入口文件出发，调用所有配置的loader对模块进行解析翻译，在找到该模块依赖的模块进行处理 3.5. 完成模块编译，得到每个模块被翻译之后的最终的内容和依赖关系 3.6. 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，在把每个chunk转换成一个单独的文件加载到输出列表 3.7. 输出完成，确定输出的路径和文件名，把内容写到文件系统中 **在以上过程中，webpack会在特定的时间点广播出特定的事件，插件在舰艇感兴趣的事件后会执行特定的逻辑，改变webpack的运行结果**

4. webpack 热加载执行原理

？？？？

5. 如何利用webpack来优化前端性能

5.1. 压缩代码。uglifyJsPlugin 压缩js代码， mini-css-extract-plugin 压缩css代码 5.2. 利用CDN加速，将引用的静态资源修改为CDN上对应的路径，可以利用webpack对于output参数和loader的publicpath参数来修改资源路径 5.3. 删除死代码（tree shaking），css需要使用Purify-CSS 5.4. 提取公共代码。webpack4移除了CommonsChunkPlugin (提取公共代码)，用optimization.splitChunks和optimization.runtimeChunk来代替

6. 什么是bundle,什么是chunk，什么是module?

bundle:有webpack打包出来的文件 chunk：webpack在进行模块的依赖分析的时候，代码分割出来的代码块 module:开发中的单个模块

7. webpack-dev-server和http服务器如nginx有什么区别?

webpack-dev-server使用内存来存储webpack开发环境下的打包文件，并且可以使用模块热更新，他比传统的http服务对开发更加简单高效。

1. 对webpack的了解

本质上，webpack 是一个现代 JavaScript 应用程序的静态模块打包器(module bundler)，将项目当作一个整体，通过一个给定的的主文件，webpack将从这个文件开始找到你的项目的所有依赖文件，使用loaders处理它们，最后打包成一个或多个浏览器可识别的js文件

核心概念：

- 入口(entry)

入口起点(entry point)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

可以通过在 webpack 配置中配置 entry 属性，来指定一个入口起点（或多个入口起点）

```
module.exports = {
  entry: './path/to/my/entry/file.js'
};
```

- 输出(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

- loader

loader 让 webpack 能够去处理那些非 JavaScript 文件（ webpack 自身只理解 JavaScript ）

- 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

- 模式

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

```
module.exports = {
  mode: 'production'
};
```

2. webpack，里面的webpack.config.js怎么配置

let webpack = require('webpack');

module.exports = { entry:'./entry.js', //入口文件

```
output:{
   //node.js中__dirname变量获取当前模块文件所在目录的完整绝对路径
   path:__dirname, //输出位置
   filename:'build.js' //输入文件
},

module:{
   // 关于模块的加载相关，我们就定义在module.loaders中
   // 这里通过正则表达式去匹配不同后缀的文件名，然后给它们定义不同的加载器。
   // 比如说给less文件定义串联的三个加载器（！用来定义级联关系）：
   rules:[
    {
      test:/\.css$/,     //支持正则
      loader:'style-loader!css-loader'
    }
   ]
},

//配置服务
devServer:{
   hot:true, //启用热模块替换
   inline:true
   //此模式支持热模块替换：热模块替换的好处是只替换更新的部分,而不是页面重载.
},

//其他解决方案配置
resolve:{
   extensions:['','.js','.json','.css','.scss']
},

//插件
plugins:[
   new webpack.BannerPlugin('This file is create by baibai')
]
```

}

3. webpack本地开发怎么解决跨域的

下载 webpack-dev-server 插件

配置 webpack.config.js 文件

// webpack.config.js

var WebpackDevServer = require("webpack-dev-server");

module.exports = { ...

```
devServer: {
    ...
    port: '8088', //设置端口号
    // 代理设置
    proxy: {
        '/api': {
            target: 'http://localhost:80/index.php', // 目标代理
            pathRewrite: {'^/api' : ''}, // 重写路径
            secure: false, // 是否接受运行在 HTTPS 上

        }
    }
}
```

}

5. webpack与grunt、gulp的不同

三者都是前端构建工具

`grunt` 和 `gulp` 是基于任务和流的。找到一个（或一类）文件，对其做一系列链式操作，更新流上的数据， 整条链式操作构成了一个任务，多个任务就构成了整个web的构建流程

`webpack` 是基于入口的。`webpack` 会自动地递归解析入口所需要加载的所有资源文件，然后用不同的`Loader` 来处理不同的文件，用 `Plugin` 来扩展 `webpack` 功能

`webpack` 与前者最大的不同就是支持代码分割，模块化（AMD,CommonJ,ES2015），全局分析

6. 有哪些常见的Loader？他们是解决什么问题的

- `css-loader`：加载 `CSS`，支持模块化、压缩、文件导入等特性
- `style-loader`：把 `CSS` 代码注入到 `JavaScript 中`，通过 `DOM` 操作去加载 `CSS`
- `slint-loader`：通过 `SLint` 检查 `JavaScript` 代码
- `babel-loader`：把 `ES6` 转换成 `ES5`
- `file-loader`：把文件输出到一个文件夹中，在代码中通过相对 `URL` 去引用输出的文件
- `url-loader`：和 `file-loader` 类似，但是能在文件很小的情况下以 `base64` 的方式把文件内容注入到代码中去

7. 有哪些常见的Plugin？他们是解决什么问题的

- `define-plugin`：定义环境变量
- `commons-chunk-plugin`：提取公共代码

8. Loader和Plugin的不同

loader 加载器

Webpack

将一切文件视为模块，但是

```
webpack
```

原生是只能解析

```
js
```

文件.

```
Loader
```

的作用是让

```
webpack
```

拥有了加载和解析非

```
JavaScript
```

文件的能力

在 `module.rules` 中配置，也就是说他作为模块的解析规则而存在，类型为数组

- Plugin 插件
- 扩展

  ```
  webpack
  ```

  的功能，让

  ```
  webpack
  ```

  具有更多的灵活性

  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

9. webpack的构建流程是什么

1. 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
2. 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
3. 确定入口：根据配置中的 `entry` 找出所有的入口文件
4. 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

5. 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
6. 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
7. 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

12. 如何利用webpack来优化前端性能

- 压缩代码。删除多余的代码、注释、简化代码的写法等等方式
- 利用 `CDN` 加速。在构建过程中，将引用的静态资源路径修改为 `CDN` 上对应的路径
- 删除死代码 `Tree Shaking )` 。将代码中永远不会走到的片段删除掉
- 优化图片，对于小图可以使用 `base64` 的方式写入文件中
- 按照路由拆分代码，实现按需加载，提取公共代码
- 给打包出来的文件名添加哈希，实现浏览器缓存文件14. 怎么配置单页应用？怎么配置多页应用
    - 单页应用可以理解为 `webpack` 的标准模式，直接在 `entry` 中指定单页应用的入口即可
    - 多页应用的话，可以使用 `webpack` 的 `AutoWebPlugin` 来完成简单自动化的构建，但是前提是项目的目录结构必须遵守他预设的规范