Webpack面试题

1.loader和plugins的区别

1.webpack可以打包一切文件，但是原生的webpack只能解析js，所以需要loader来解析其他类型的文件。
2.plugin是webpack执行过程中各个阶段会广播出许多时间，webpack可以监听这些事件，在合适的时间改变结果。

2.webpack的构建流程

1.初始化配置参数，配置文件和shell语句中的参数得到最终的配置文件
2.开始编译初始化Compiler对象，加载所有的插件，执行run方法
3.根据配置文件的入口，确定所有的入口文件
4.从入口文件出发调用其所有匹配的loader,找到模块依赖的模块，然后递归本步骤让所有的入口依赖文件都经过本步骤的处理。
5.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,将chunk转化为单独的文件转化到输出列表
6.确定好输出内容，根据配置的输出路劲，生产文件。

3.常见的webpack插件和loaders有哪些。

loader:

css-loader: 处理background:(url)还有@import这些语法。让webpack能够正确的对其路劲进行模块化处理
postcss-loader:加入css前缀
style-loader: 创建style标签,插入到HTML页面中，一般用于开发环境
url-loader:文件较小的时候用base64引入
sass-loader:把Sass/SCSS文件编译成CSS
babel-loader:将es6转化为es5
plugin:
htmlwebpackplugin：定义html模板配置生成html文件
defineplugin:定义全局变量的
terserWebpackPlugin 压缩js
WatchMissingNodeModulesPlugin 允许安装新的库不用重新打包webpack
HotModuleReplacementPlugin 启用模块热替换

4.source map是什么？生产环境怎么用

source map是为了解决生产环境和开发环境代码不一致，帮助我们degug原始代码的技术

- 开发环境使用：cheap-module-eval-source-map
- 生产环境使用：cheap-module-source-map

类型	含义
source-map	原始代码 最好的sourcemap质量有完整的结果，但是会很慢
eval-source-map	原始代码 同样道理，但是最高的质量和最低的性能
cheap-module-eval-source-map	原始代码（只有行内） 同样道理，但是更高的质量和更低的性能
cheap-eval-source-map	转换代码（行内） 每个模块被eval执行，并且sourcemap作为eval的一个dataurl
eval	生成代码 每个模块都被eval执行，并且存在@sourceURL,带eval的构建模式能cache SourceMap
cheap-source-map	转换代码（行内） 生成的sourcemap没有列映射，从loaders生成的sourcemap没有被使用
cheap-module-source-map	原始代码（只有行内） 与上面一样除了每行特点的从loader中进行映射

看似配置项很多， 其实只是五个关键字eval、source-map、cheap、module和inline的任意组合

关键字	含义
eval	使用eval包裹模块代码
source-map	产生.map文件
cheap	不包含列信息（关于列信息的解释下面会有详细介绍)也不包含loader的sourcemap
module	包含loader的sourcemap（比如jsx to js ， babel的sourcemap）,否则无法定义源文件
inline	将.map作为DataURI嵌入，不单独生成.map文件

5.如何利用webpack优化前端性能

1.压缩JS

```
optimization: {     minimize: true,
  minimizer: [
    //压缩JS
```

```
+ new TerserPlugin({}) ]
  },
```

2.压缩css

```
 optimization: {      minimize: true,
    minimizer: [
      //压缩CSS
+ new OptimizeCSSAssetsPlugin({}), ]
  },
```

3.开启Tree-shaking

```
+ mode:'production',
+ devtool:false,
```

4.代码分割

1.入口点进行分割
2.懒加载会自动进行分割

react实现懒加载:

```
const Title=react.lazy(()=>import('./title'))
<Suspense fallback={<Loading/>}>
<Title/>
</Suspense>
```

3.使用spiltchunks,提取公共代码

```
//配置如何优化
 optimization: {
//设置代码分隔的方案
 splitChunks: {
//要分割哪些代码 initial是同步 async是异步
 chunks: 'all',
 name: true,//设置代码块打包后的名称，默认名称是分隔符~分割开的原始代码块
 automaticNameDelimiter: "~",
 maxAsyncRequests: 5,//同一个入口分割出来的最大异步请求数
 maxInitialRequests: 3,//同一个入口分割出来最大的同步请求数
//缓存组 设置不同的缓存组来抽取满足不同规则的chunk
//webpack中还有默认的缓存组，它的优先级是0
 cacheGroups: {
 venders: {
 test: /node_modules/,//条件
 priority: -10//数字越大，优先级越高
 },
 commons: {
 minChunks: 2,//
 minSize: 0,//被提取代码块的最小尺寸,默认是30K
 priority: -20//数字越大，优先级越高
 }
 }
 },
//runtimeChunk:true//提取公共的代码块
 },
```

4.使用preload,和preload预先加载资源

preload用于预先需要加载的资源，提高权重，加入webpack注释之后他会提升

```
<link rel="preload"as="script" href="utils.js">
import( `./utils.js`/* webpackPreload: true *//* webpackChunkName: "utils" */ )
```

而prefetch是浏览器空闲的时候去加载资源

5.使用CDN

可以使用pubilcPath，把打包的文件加入域名的前缀，减少文件的大小。

```
 output: {
 path: path.resolve(__dirname, "dist"),
 filename: "[name].[hash].js",
 chunkFilename: "[name].[hash].chunk.js",
publicPath: "http://img.xxx.cn/", },
```

6.webpack中几个hash的区别

hash:一次编译一个hash
chunkhash:一般用于代码块，一个代码块一个hash
contenthash:内容hash，更具内容来设置hash，一般用于css

7.如果对bundle体积进行监控

webpack-bundle-analyzer

8.如何提高webpack的构建速度

1.缩小查找范围

resolve:{
extensions:[".js",".jsx",'.json','css']
}

resolve:{
alias:{
components: 'hzero-front/lib/components/',
}
}
modules
用来配置查找直接申明的模块
resolve: {
modules: ['node_modules'],
}
mainFields
配置package.json文件的查找规则默认为main
mainFileIds:["module","main"]
5.resolveLoader
配置loader的查找位置

```
module.exports = {
  resolveLoader: {
    modules: [ 'node_modules' ],
    extensions: [ '.js', '.json' ],
    mainFields: [ 'loader', 'main' ]
  }
};
```

2.noParse

可以配置一些库不需要分析依赖

3.配置IgonrePlugin

4.HardSourceWebpackPlugin <https://github.com/mzgoddard/hard-source-webpack-plugin> 为模块提供中间缓存，

，默认缓存的目录是node_modules/.cache/hard-source

- 配置 hard-source-webpack-plugin后，首次构建时间并不会有太大的变化，但是从第二次开始，构建时间大约可以减少 80%左右
- webpack5中会内置`hard-source-webpack-plugin`

5.使用 oneof

module:{
rules:[
{

}
]
}

- 对于每个文件rules都会遍历一遍，利用oneof可以匹配到一个就退出

6.thread-loader

把这个loader放置其他loader之前开启多进程打包

```
    rules: [
      {
        test: /\.js/,
        include: path.resolve(__dirname, "src"),
        use: [
+          {
+            loader:'thread-loader',
+            options:{
+              workers:3
+            }
+          },
          {
            loader: "babel-loader",
            options: {
              presets: ["@babel/preset-env", "@babel/preset-react"],
            },
          },
        ],
      },
```

9.Loader的执行顺序是怎么样的

loader先执行前置，后执行正常，在执行内联，最后执行后置

use:{

enfore:'pre'

},

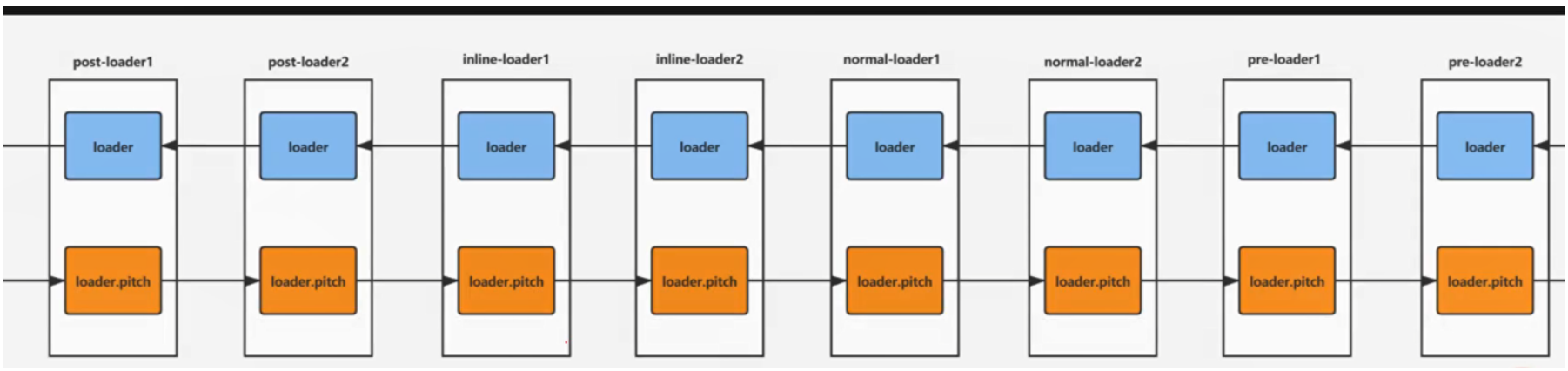
{

enfore:'post'

},

内联样式是写在require里面的，-!不要前置和普通 ，!!只要内联 !不要普通。

内部执行顺序是先执行，loader.pitch然后执行，如果有返回值则跳过剩下的loader，直接往回走loader



10.是否写过loader，写loader的思路是什么

一般是接收源代码，转化成直接想要的内容返回，比如babel-loader，是将es6转化成es5,调用babel.transform

11.loader如何编写异步的

```
const callback = this.async(); // 声明一下异步操作
setTimeout(() => {
  const result = source.replace('xiaochengzi', options.name);
  callback(null, result); // 在回调里返回结果
}, 1000)
```

12.是否写过plugin，plugin的思路是什么

写一个class方法，绑定apply方法，在该方法取注册compiler和Compilation暴露的方法，tap,tapSync之类的。

13.TreeShaking了解过吗

原理是将一行的导入变为多个导入，具体的组件比如antd/button,es6模块只能定义在最前面静态导入，不能写在判断中，可以导出多个。