## 1.

webpack是一个打包模块化javaScript的工具，它会从main.js出发，识别出源码中的模块化导入语句，递归地
找出出入口文件的所有依赖，将入口和其所有依赖打包到一个单独的文件中。

# 常用配置

```
* Module 配置处理模块的规则
* Resolve 寻找模块的规则
* Plugins 扩展插件
* Entry 配置模块的入口
* Output 配置如何输出
* DevServer
```

## 2.解析loader

```
loaders告诉webpack 在遇到哪些文件时使用 哪些Loader去加载和转换。
三种方式：
1.条件匹配:通过test,include,exclude,来选中loader要应用规则的文件
2.应用规则：对选中的文件通过use 配置项来应用 Loader,
test: 匹配文件,可以是数组
include: 包含某文件
exclude: 排除某文件
use: use是每一个rule的属性，指定要用什么loader

例子：
在遇到.css 的结尾的文件时，先使用 css-load 读取css文件，再由style-loader将css的内容注入JavaScript。
注意：
* use 属性的值需要是一个由Loader 名称组成的数组，Loader 的执行顺序是由后到前的。
* 每个loader都可以通过 URL querystring 的方式传入参数。
```

webpack.config.js

```
const path = require('path');

module.exports = {
  // JS 执行入口文件
  entry: './main.js',
  output: {
    // 把所有依赖的模块合并输出到一个 bundle.js 文件
    filename: 'bundle.js',
    // 输出文件都放到 dist 目录下
    path: path.resolve(__dirname, './dist'),
  },
  module: {
    rules: [
      {
        // 用正则去匹配要用该 loader 转换的 css 文件
        test: /\.css$/,
```

```
        loaders: ['style-loader', 'css-loader'],
      }
    ]
  }
};
```

2.noParse:忽略对部分没采用模块化的文件的递归解析和处理。提高构建性能。
一些库如jq,chartJS大而没采用模块化标准让webpack 解析耗时又没意义

3.parser:细粒度地配置哪些模块被哪些模块解析

```
module:{
    rules:[
        {
            test://,
            noParse:'',
            parser:{

            }
    },
        {}
    ]
}
```

## resolve 配置webpack 如何寻找模块所对应地文件。

webpack 内置js模块化语法解析功能，也可以自定义规则：

**1.alias:通过别名来将导入路径映射成一个新的导入路径**

**2.mainFields**

**3.extensions:当没有文件后缀，webpack配置在尝试过程中用到地后缀列表：**

```
    resolve: {
        extensions: [".vue", ".js", ".json"],
        alias: {
            'com': resolve('src/components'),
            'mod': resolve('src/modules'),
            'util': resolve('src/util'),
            '@': resolve('src')
        }
    },
```

```
extensions:['.js','.json']
```

# 2.Entry

类型:

```
string  './app'
array  ['.app/entry3','./app/entry2']
object {a:'app/entry3',b:'app/'}
```

实例:

```
entry: {
    app: ["babel-polyfill", "./src/index.js"]
},
```

# 3.output

```
output: {
    filename: "bundle.js",
    path: __dirname + "/dist",
    publicPath: process.env.BUILD_ENV === 'production'
    ? config.build.assetsPublicPath
    : config.dev.assetsPublicPath
},
```

3.1 output.filename.配置输出文件的名称，string 类型
3.2 path
3.3 publicPath 配置发布到线上资源url 前缀，（在复杂的项目可能会有一些构建出的资源需要异步加载）

## Pulgin:扩展Webpack的功能

接收一个数组，数组里的每一项都是一个要使用的Pulgin的实例，Pulgin需要的参数通过构造函数传入

```
module.exports = {
    pulgins:[]
}
```

## 其他属性