1，webpack打包的配置，常见的loaders和plugins？

**2，webpack的基本配置有哪些？**

Entry 配置模块的入口；

Output 配置如何输出最终想要的代码；

Module 配置处理模块的规则；

Resolve 配置寻找模块的规则；

Plugins 配置扩展插件；

DevServer 配置 DevServer；

**3，webpack 打包按需加载？**

在 webpack 的构建环境中，要按需加载代码模块很简单，遵循 ES 标准的动态加载语法 dynamic-import 来编写代码即可，webpack 会自动处理使用该语法编写的模块：

需要我们在 webpack 配置中添加一个 output.chunkFilename 的配置：

output: {

  path: path.resolve(\_\_dirname, 'dist'),

  filename: '[name].[hash:8].js',

  chunkFilename: '[name].[hash:8].js' // 指定分离出来的代码文件的名称

}

**3，简单介绍下webpack的工作原理？**

Webpack 的运行流程是一个串行的过程，从启动到结束会依次执行以下流程：

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；

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；

确定入口：根据配置中的 entry 找出所有的入口文件；

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；

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；

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；

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。

<https://whjin.github.io/full-stack-development/posts/>工作原理概括.html

**4，webpack配置开发环境和生产环境的区别？**

日常的前端开发工作中，一般都会有两套构建环境：一套开发时使用，构建结果用于本地开发调试，不进行代码压缩，打印 debug 信息，包含 sourcemap 文件；另外一套构建后的结果是直接应用于线上的，即代码都是压缩后，运行时不打印 debug 信息，静态文件不包括 sourcemap 的。有的时候可能还需要多一套测试环境，在运行时直接进行请求 mock 等工作。

webpack 4.x版本在webpack配置中有mode选项可以直接配置production 或 development

webpack 3.x 一般是通过node命令传递环境变量，来控制不同环境下的构建行为

如：

{

  "scripts": {

    "build": "NODE\_ENV=production webpack",

    "dev": "NODE\_ENV=development webpack-dev-server"

  }

}

然后在 webpack.config.js 文件中可以通过 process.env.NODE\_ENV 来获取命令传入的环境变量：

**常见的环境差异配置**

生产环境可能需要分离 CSS 成单独的文件，以便多个页面共享同一个 CSS 文件

生产环境需要压缩 HTML/CSS/JS 代码

生产环境需要压缩图片

开发环境需要生成 sourcemap 文件

开发环境需要打印 debug 信息

开发环境需要 live reload 或者 hot reload 的功能

以上是常见的构建环境需求差异，可能更加复杂的项目中会有更多的构建需求（如划分静态域名等），但是我们都可以通过判断环境变量来实现这些有环境差异的构建需求。

webpack 4.x 的 mode 已经提供了上述差异配置的大部分功能，mode 为 production 时默认使用 JS 代码压缩，而 mode 为 development 时默认启用 hot reload，等等。这样让我们的配置更为简洁，我们只需要针对特别使用的 loader 和 plugin 做区分配置就可以了。

webpack 3.x 版本还是只能自己动手修改配置来满足大部分环境差异需求。

**5，webpack热加载的原理， 介绍一下HMR？**

HMR 全称是 Hot Module Replacement，即模块热替换。在这个概念出来之前，我们使用过 Hot Reloading，当代码变更时通知浏览器刷新页面，以避免频繁手动刷新浏览器页面。HMR 可以理解为增强版的 Hot Reloading，但不用整个页面刷新，而是局部替换掉部分模块代码并且使其生效，可以看到代码变更后的效果。所以，HMR 既避免了频繁手动刷新页面，也减少了页面刷新时的等待，可以极大地提高前端页面开发效率.

在webpack中用法很简单

HMR 是 webpack 提供的非常有用的一个功能，跟我们之前提到的一样，安装好 webpack-dev-server， 添加一些简单的配置，即在 webpack 的配置文件中添加启用 HMR 需要的两个插件：

const webpack = require('webpack')

module.exports = {

  // ...

  devServer: {

    hot: true // dev server 的配置要启动 hot，或者在命令行中带参数开启

  },

  plugins: [

    // ...

    new webpack.NamedModulesPlugin(), // 用于启动 HMR 时可以显示模块的相对路径

    new webpack.HotModuleReplacementPlugin(), // Hot Module Replacement 的插件

  ],

}

**5，webpack提取公共模块？**

webpack 3.x 以前的版本是使用 CommonsChunkPlugin 来做代码分离的，而 webpack 4.x 则是把相关的功能包到了 optimize.splitChunks 中，直接使用该配置就可以实现代码分离。

webpack4

显式配置共享类库可以这么操作：

module.exports = {

  entry: {

    vendor: ["react", "lodash", "angular", ...], // 指定公共使用的第三方类库

  },

  optimization: {

    splitChunks: {

      cacheGroups: {

        vendor: {

          chunks: "initial",

          test: "vendor",

          name: "vendor", // 使用 vendor 入口作为公共部分

          enforce: true,

        },

      },

    },

  },

  // ... 其他配置

}

8，webpack优化

9， svn与git的区别？

git是分布式的，svn不是。

git跟svn一样有自己的集中式版本库或服务器。但git更倾向于被使用于分布式模式，克隆版本库后即使没有网络也能够commit文件，查看历史版本记录，创建项目分支等，等网络再次连接上Push到服务器端。

git把内容按元数据方式存储，而svn是按文件。

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。

git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

git没有一个全局的版本号，svn有。

git的内容完整性优于svn。

因为git的内容存储使用的是SHA-1哈希算法。

git可以有无限个版本库，svn只能有一个指定中央版本库。

当svn中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成。

每一个git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）有问题，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

1、svn git上传合并项目怎么做？

4、怎么确保代码质量？

5、gulp是怎么合并文件的

6、移动端怎么测试的

7、抓包工具用过吧

8、平时用git的话有什么优点缺点

9、遇到冲突是怎么解决的？

10、glup 功能？

11、gulp做压缩还是打包

14、grunt, gulp, webpack三者的区别

17、用node发过包吗，步骤是什么

19、版本的管理工具，是用通讯化工具，还是直接敲的命令行

20、grunt和gulp？

接口测试工具用过什么

代码检测用什么

什么是css预处理器，什么是css后处理器

webpack常用配置

46，web开发中会话跟踪的方法有哪些

cookie

session

url参数传递

隐藏input

ip地址

48，svn与git的区别

git是分布式的，svn不是。

git跟svn一样有自己的集中式版本库或服务器。但git更倾向于被使用于分布式模式，克隆版本库后即使没有网络也能够commit文件，查看历史版本记录，创建项目分支等，等网络再次连接上Push到服务器端。

git把内容按元数据方式存储，而svn是按文件。

所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn,.cvs等的文件夹里。

git目录是处于你的机器上的一个克隆版的版本库，它拥有中心版本库上所有的东西，例如标签，分支，版本记录等。

git没有一个全局的版本号，svn有。

git的内容完整性优于svn。

因为git的内容存储使用的是SHA-1哈希算法。

git可以有无限个版本库，svn只能有一个指定中央版本库。

当svn中央版本库有问题时，所有工作成员都一起瘫痪直到版本库维修完毕或者新的版本库设立完成。

每一个git都是一个版本库，区别是它们是否拥有活跃目录（Git Working Tree）。如果主要版本库（例如：置於GitHub的版本库）有问题，工作成员仍然可以在自己的本地版本库（local repository）提交，等待主要版本库恢复即可。工作成员也可以提交到其他的版本库！

51，Less常用特性

变量(@color = #fff)

混合(Mixin)

内置函数(颜色，字符串，类型判断，数学)

循环

嵌套

运算

导入(@import)

href是Hypertext Reference的缩写，指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果我们在文档中添加<link href="common.css" rel="stylesheet"/>那么浏览器会识别该文档为css文件，就会并行下载资源并且不会停止对当前文档的处理。这也是为什么建议使用link方式来加载css，而不是使用@import方式。

Href 超文本链接,指向网络资源所在位置,如果我们在文档添加<link href="common.css" rel="stylesheet"/>浏览器会下载资源并且不会停止对当前文档的处理。

54.线程与进程的区别?

一个程序至少有一个进程,一个进程至少有一个线程;

多进程拥有独立的内存,多线程共享内存,所以多线程提高了运行效率;

多线程的重要意义在于一个应用程序中,有多个执行程序能够同时执行,但是系统并没有将多线程看成多个独立的应用

55，对前端工程师这个职位你是怎么样理解的？

与用户打交道最近的地方,负责用户所能看到的一切的构建,在满足基本功能的同时,添加炫酷的特效,排版等,提升用户体验感,加强产品粘性。

前端是最贴近用户的程序员，前端的能力就是能让产品从 90 分进化到 100 分，甚至更好

参与项目，快速高质量完成实现效果图，精确到 1px；

与团队成员，UI 设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理 hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

56.平时如何管理你的项目？

58，浅拷贝vs深拷贝

拷贝其实就是对象复制，为了解决对象复制是产生的引用类型问题

浅拷贝：利用迭代器，循环对象将对象中的所有可枚举属性复制到另一个对象上，但是浅拷贝的有一个问题就是只是拷贝了对象的一级，其他级还如果是引用类型的值的话依旧解决不了

深拷贝：深拷贝解决了浅拷贝的问题，利用递归的形势便利对象的每一级，实现起来较为复杂，得判断值是数组还是对象；

22.。