05-前端工程化-常见面试题

1. Node基础

1.1 说说Node是什么？和浏览器有什么区别和关系？

- Node.js是一个基于V8 JavaScript引擎的JavaScript运行时环境
  - V8可以嵌入到任何C ++应用程序中，无论是Chrome还是Node.js，事实上都是嵌入了V8引擎来执行JavaScript代码；
  - 但是在Chrome浏览器中，还需要解析、渲染HTML、CSS等相关渲染引擎，另外还需要提供支持浏览器操作的API、浏览器自己的事件循环等
  - 在Node.js中我们也需要进行一些额外的操作，比如文件系统读/写、网络IO、加密、压缩解压文件等操作
- 在浏览器中，全局变量都是在window上的，比如有document、setInterval、setTimeout、alert、console等
  - 在浏览器中执行的JavaScript代码，如果我们在顶级范围内通过var定义的一个属性，默认会被添加到window对象上。
- 在Node中，我们也有一个global属性，并且看起来它里面有很多其他对象
  - 在node中，我们通过var定义一个变量，它只是在当前模块中有一个变量，不会放到全局中

1.2 说说你对模块化开发的理解

- 模块化开发最终的目的是将程序划分成一个个小的结构(模块)
- 这个结构中编写属于自己的逻辑代码，有自己的作用域，定义变量名词时不会影响到其他的结构
- 可导出变量,函数,对象等给其他模块用，也可导入其他模块中的变量,函数,对象
- 按照这种结构划分开发程序的过程，就是模块化开发的过程
- 早期是没有模块化, 带来一些问题
  - 命名冲突 -> 立即执行函数 -> 自定义模块 -> 没有规范
  - 社区中模块化规范: CommonJS/AMD/CMD
  - ES6中推出模块化: ES Module

1.3 什么是 yarn 和 npm？为什么要用 yarn 代替 npm 呢?

- npm 是与 Node.js 自带的默认包管理器，它有一个大型的公共库和私有库，存储在 npm registry 的数据库中（官方默认中心库 http://registry.npmjs.org/，国内淘宝镜像 http://registry.npm.taobao.org/），用户可以通过 npm 命令行访问该仓库。在 npm 的帮助下，用户可以轻松管理项目中的依赖项。

- yarn 也是一个包管理器，是由Facebook、Google、Exponent 和 Tilde 联合推出了一个新的 JS 包管理工具，它有如下的优点：
    - 速度快，支持并行安装。无论 npm 还是 Yarn 在执行包的安装时，都会执行一系列任务。npm 是按照队列执行每个 package，也就是说必须要等到当前 package 安装完成之后，才能继续后面的安装。而 Yarn 是同步执行所有任务，提高了性能。
    - 离线模式，如果之前已经安装过一个软件包，用Yarn再次安装时之间从缓存中获取，就不用像npm那样再从网络下载了。
- yarn 是为了弥补 早期npm 的一些缺陷而出现的，因为早期的npm存在很多的缺陷，比如安装依赖速度很慢、版本依赖混乱等等一系列的问题。虽然从npm5版本开始，进行了很多的升级和改进，但是依然很多人喜欢使用yarn。

1.4 说出npm install的安装过程

执行 npm install 会检测是有package-lock.json文件：

- 没有lock文件
    - 分析依赖关系，这是因为我们可能包会依赖其他的包，并且多个包之间会产生相同依赖的情况；
    - 从registry仓库中下载压缩包（如果我们设置了镜像，那么会从镜像服务器下载压缩包）；
    - 获取到压缩包后会对压缩包进行缓存（从npm5开始有的）；
    - 将压缩包解压到项目的node_modules文件夹中（前面我们讲过，require的查找顺序会在该包下面查找）
- 有lock文件
    - 检测lock中包的版本是否和package.json中一致（会按照semver版本规范检测）
    - 不一致，那么会重新构建依赖关系，直接会走顶层的流程
    - 一致的情况下，会去优先查找缓存
        - 没有找到，会从registry仓库下载，直接走顶层流程
        - 查找到，会获取缓存中的压缩文件，并且将压缩文件解压到node_modules文件夹中

1.5 什么是pnpm？为什么说pnpm高性能?

- pnpm是一个快速、节省磁盘空间的软件包管理器，特点是：快速、高效，并支持monorepo等等。
- 当使用 npm 或 Yarn 时，如果你有 100 个项目，并且所有项目都有一个相同的依赖包，那么， 你在硬盘上就需要保存 100 份该相同依赖包的副本。为了解决上面的问题，就出现了pnpm，使用pnpm安装的依赖包将被存放在一个统一的位置。
    - 当安装软件包时， 其包含的所有文件都会硬链接到此位置，而不会占用额外的硬盘空间
    - pinia是软连接和硬链接相结合，方便在项目之间共享相同版本的依赖包

2. webpack基础

2.1 什么是webpack，说说你对webpack的理解

- webpack是一个静态的模块化打包工具，它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。
- 对webpack的理解
  - 打包工具：webpack可以将帮助我们进行模块打包，所以它是一个打包工具。
  - 生成静态资源：这样表述的原因是我们最终可以将代码打包成最终的静态资源（部署到静态服务器）
  - 支持模块化：webpack默认支持各种模块化开发，ES Module、CommonJS、AMD等

2.2 有哪些常见的Loader？你用过哪些Loader

`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 (处理图片和字体)

`url-loader`：与 file-loader 类似，区别是用户可以设置一个阈值，大于阈值会交给 file-loader 处理，小于阈值时返回文件 base64 形式编码 (处理图片和字体)

`sass-loader`：将SCSS/SASS代码转换成CSS

`less-loader`：将Less码转换成CSS

`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

`postcss-loader`：扩展 CSS 语法，使用下一代 CSS，可以配合 autoprefixer 插件自动补齐 CSS3 前缀

2.3 有哪些常见的Plugin？你用过哪些Plugin?

- `define-plugin`：定义环境变量 (Webpack4 之后指定 mode 会自动配置)
- `html-webpack-plugin`：简化 HTML 文件创建 (依赖于 html-loader)
- `clean-webpack-plugin`: 目录清理
- `webpack-bundle-analyzer`: 可视化 Webpack 输出文件的体积 (业务组件、依赖第三方模块)

2.4 说一说Loader和Plugin的区别?

- `Loader` 本质就是一个函数，在该函数中对接收到的内容进行转换，返回转换后的结果。 因为 Webpack 只认识 JavaScript，所以 Loader 就成了翻译官，对其他类型的资源进行转译的预处理工作。
- `Plugin` 就是插件，基于事件流框架 `Tapable`，插件可以扩展 Webpack 的功能，在 Webpack 运行的生命周期中会广播出许多事件，Plugin 可以监听这些事件，在合适的时机通过 Webpack 提供的 API 改变输出结果。
- `Loader` 在 module.rules 中配置，作为模块的解析规则，类型为数组。每一项都是一个 Object，内部包含了 test(类型文件)、loader、options (参数)等属性。
- `Plugin` 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。

2.5 source map是什么?

`source map` 是将编译、打包、压缩后的代码映射回源代码的过程。打包压缩后的代码不具备良好的可读性，想要调试源码就需要 soucre map。

map文件只要不打开开发者工具，浏览器是不会加载的。

线上环境一般有三种处理方案：

- `hidden-source-map`：借助第三方错误监控平台 Sentry 使用
- `nosources-source-map`：只会显示具体行数以及查看源代码的错误栈。安全性比 sourcemap 高
- `sourcemap`：通过 nginx 设置将 .map 文件只对白名单开放(公司内网)

注意：避免在生产中使用 `inline-` 和 `eval-`，因为它们会增加 bundle 体积大小，并降低整体性能。

2.6 什么是babel？babel在开发中是什么作用?

- Babel是一个JavaScript的编译工具，常用于编译JavaScript代码。
- 比如：可以将ECMAScript 2015+代码转换为向后兼容版本的JavaScript (即ES6以上的代码转成ES5代码)，但Babel在进行代码转换时是需要依赖对应的插件来转换。

2.7 vue-loader是什么？使用它的用途有哪些?

- Vue Loader 是一个 webpack 的 loader，它允许你以一种名为单文件组件 (SFC)的格式撰写 Vue 组件。
- Vue Loader 还提供了很多酷炫的特性：
  - 允许为 Vue 组件的每个部分使用其它的 webpack loader，例如在 `<style>` 的部分使用 Sass 和在 `<template>` 的部分使用 Pug；
  - 允许在一个 `.vue` 文件中使用自定义块，并对其运用自定义的 loader 链；
  - 使用 webpack loader 将 `<style>` 和 `<template>` 中引用的资源当作模块依赖来处理；
  - 为每个组件模拟出 scoped CSS；
  - 在开发过程中使用热重载来保持状态。
- 简而言之，webpack 和 Vue Loader 的结合为你提供了一个现代、灵活且极其强大的前端工作流，来帮助撰写 Vue.js 应用。

3. Git

3.1 集中式版本控制和分布式版本控制有什么区别?

- 集中式版本控制(简称 CVCS） **比如 CVS和SVN**

    - 主要特点是单一的集中管理的服务器，保存所有文件的修订版本
    - 协同开发人员通过客户端连接到这台服务器，取出最新的文件或者提交更新
    - 集中式版本控制也有一个核心的问题：**中央服务 器不能出现故障**
- 分布式版本控制系统 (简 称 DVCS) **比如git**
    - 客户端并不只提取最新版本的文件快照， 而是把代码仓库完整地镜像下 来，包括完整的历史记录
    - 这么一来，任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复
    - 因为每一次的克隆操作，实际上都是一次**对代码仓库的完整备份**
- Git和SVN的区别

Git	SVN
1. Git是一个分布式的版本控制工具	1. SVN 是集中版本控制工具
2.它属于第3代版本控制工具	2.它属于第2代版本控制工具
3.客户端可以在其本地系统上克隆整个存储库	3.版本历史记录存储在服务器端存储库中
4.即使离线也可以提交	4.只允许在线提交
5.Push/pull 操作更快	5.Push/pull 操作较慢
6.工程可以用 commit 自动共享	6.没有任何东西自动共享

3.2 工作中常见的Git命令有哪些?

```
#创建版本库:
$ git clone <url>  #克隆远程版本库
$ git init #初始化本地版本库

#修改和提交:
$ git status #查看状态
$ git add <file> #跟踪指定的文件
$ git add .  #将文件添加到暂存区中
$ git commit -m #将暂存区的文件进行提交更新

#查看提交历史:
$ git log #查看提交的历史
$ git log -p <file> #查看指定文件的提交历史

#对分支和标签的操作:
#分支:
$ git branch #查看本地所有分支
$ git checkout <branch> #切换到指定的分支
$ git branch <new-branch> #创建新的分支
$ git branch -d <branch> #删除本地分支

#标签:
```

```
$ git tag <newtag> #基于最新提交创建标签
$ git tag #查看所有的本地标签
$ git tag -d <tagname> #删除指定的标签
$ git push --tags #上传所有标签到远程仓库

#分支合并:
$ git merge <branch> #合并指定分支到当前分支
$ git rebase <branch> #衍合指定分支到当前分支

#远程操作:

$ git remote add <remote> <url> #关联远程仓库
$ git remote -v #查看所关联的远程版本库信息
$ git fetch <remote> #从指定的远程仓库中获取代码、信息
$ git pull <remote> <branch> #获取远程仓库的代码并且合并
$ git push <remote> <branch> #上传本地库的代码至远程仓库并且合并


#远程分支的操作:
#分享一个分支，并且将其推送到有写入权限的仓库上:
$ git push <remote> <branch>


#跟踪远程分支:
#克隆一个仓库时，默认会创建一个跟踪origin/master的分支
#自己设置跟踪其他分支:
$ git checkout --track <remote>/<branch>

#删除远程分支:
$ git push origin --delete <branch>


#获取别人更新的远程分支信息:
$ git fetch <remote> <branch>
```

3.3 Git分支管理（Git Flow）

- Production分支

  也就是我们经常使用的Master分支，这个分支包含最近发布到生产环境的代码，最近发布的Release， 这个分支只能从其他分支合并，不能在这个分支直接修改

- Develop分支

  这个分支是我们的主开发分支，包含所有要发布到下一个Release的代码，这个主要合并于其他分支，比如Feature分支

- Feature分支

这个分支主要是用来开发一个新的功能，一旦开发完成，我们合并回Develop分支，并进入下一个Release

- Release分支

  当你需要发布一个新Release的时候，我们基于Develop分支创建一个Release分支，完成Release后，我们合并到Master和Develop分支

- Hotfix分支

  当我们在Production发现新的Bug时候，我们需要创建一个Hotfix, 完成Hotfix后，我们合并回Master和Develop分支，所以Hotfix的改动会进入下一个Release