05-前端⼯程化-常⻅⾯试题

# Node基础

## 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

## 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

## 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.tao](http://registry.npm.tao/) bao.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。

## 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

## 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

# webpack基础

## 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

## 有哪些常⻅的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 插件⾃动补⻬ CSS3 前缀

## 有哪些常⻅的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)

## 说⼀说Loader和Plugin的区别？

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。

Loader

就是插件，基于事件流框架 Tapable ，插件可以扩展 Webpack 的功能，在 Webpack

Plugin

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。

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，

Loader

内部包含了 test(类型⽂件)、loader、options (参数)等属性。

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

Plugin

数传⼊。

## source map是什么？

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。

source 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。

## 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。

## 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：

<style>

允许为 Vue 组件的每个部分使⽤其它的 webpack loader，例如在

的部分使⽤

Sass 和在允许在⼀个

<template>

的部分使⽤ Pug；

⽂件中使⽤⾃定义块，并对其运⽤⾃定义的 loader 链；

.vue

<template>

使⽤ webpack loader 将 和为每个组件模拟出 scoped CSS；

<style>

中引⽤的资源当作模块依赖来处理；

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。

# Git

## 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.没有任何东⻄⾃动共享 |

## ⼯作中常⻅的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>

* 1. **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