1. 下面的模块导出了什么结果?

```
exports.a = 'a';
module.exports.b = 'b';
this.c = 'c';
module.exports = {
  d: 'd'
}
```

> 参考答案:
>
> ```
> { d: 'd' }
> ```

2. 说一下你对前端工程化，模块化，组件化的理解?

> 参考答案:
>
> 这三者中，模块化是基础，没有模块化，就没有组件化和工程化
>
> 模块化的出现，解决了困扰前端的两大难题：全局污染问题和依赖混乱问题，从而让精细的拆分前端工程成为了可能。
>
> 工程化的出现，解决了前端开发环境和生产环境要求不一致的矛盾。在开发环境中，我们希望代码使用尽可能的细分，代码格式尽可能的统一和规范，而在生产环境中，我们希望代码尽可能的被压缩、混淆，尽可能的优化体积。工程化的出现，就是为了解决这一矛盾，它可以让我们舒服的在开发环境中书写代码，然后经过打包，生成最合适的生产环境代码，这样就解放了开发者的精力，让开发者把更多的注意力集中在开发环境上即可。
>
> 组件化开发是一些前端框架带来的概念，它把一个网页，或者一个站点，甚至一个完整的产品线，划分为多个小的组件，组件是一个可以复用的单元，它包含了一个某个区域的完整功能。这样一来，前端便具备了开发复杂应用的能力。

3. webpack 和 gulp 的区别是什么?

> 直播讲解

> 参考答案:
>
> webpack 是基于模块化的构建工具，gulp 是基于工作流的构建工具。
>
> webpack 是一个打包器，它以一个入口为起点，构建出整个项目的依赖关系图，然后进行打包合并，生成打包结果。
>
> gulp 是一个过程管理器，每一步做什么完全看开发人员如何配置，把每一个步骤连接起来形成一个完整的构建流水线。

> 这两者并不矛盾，完全可以在一个工程中同时使用 webpack 和 gulp，将 webpack 作为 gulp 流水线中的一环。

4. webpack 中的 loader 属性和 plugins 属性的区别是什么?

> 参考答案:
>
> 它们都是 webpack 功能的扩展点。
>
> loader 是加载器，主要用于代码转换，比如 JS 代码降级，CSS 预编译、模块化等
>
> plugins 是插件，webpack 打包流程中每个环节都提供了钩子函数，可以利用这些钩子函数参与到打包生命周期中，更改或增加 webpack 的某些功能，比如生成页面和 css 文件、压缩打包结果等

5. webpack 的核心概念都有哪些?

> 参考答案:
>
> - loader
>   加载器，主要用于代码转换，比如 JS 代码降级，CSS 预编译、模块化等
> - plugin
>   插件，webpack 打包流程中每个环节都提供了钩子函数，可以利用这些钩子函数参与到打包生命周期中，更改或增加 webpack 的某些功能，比如生成页面和 css 文件、压缩打包结果等
> - module
>   模块。webpack 将所有依赖均视为模块，无论是 js、css、html、图片，统统都是模块
> - entry
>   入口。打包过程中的概念，webpack 以一个或多个文件作为入口点，分析整个依赖关系。
> - chunk
>   打包过程中的概念，一个 chunk 是一个相对独立的打包过程，以一个或多个文件为入口，分析整个依赖关系，最终完成打包合并
> - bundle
>   webpack 打包结果
> - tree shaking
>   树摇优化。在打包结果中，去掉没有用到的代码。
> - HMR
>   热更新。是指在运行期间，遇到代码更改后，无须重启整个项目，只更新变动的那一部分代码。
> - dev server
>   开发服务器。在开发环境中搭建的临时服务器，用于承载对打包结果的访问

6. commonjs 和 es6 模块的区别是什么?

> 参考答案:
>
> 1. CMJ 是社区标准，ESM 是官方标准
> 2. CMJ 是使用 API 实现的模块化，ESM 是使用新语法实现的模块化
> 3. CMJ 仅在 node 环境中支持，ESM 各种环境均支持
> 4. CMJ 是动态的依赖，ESM 既支持动态，也支持静态
> 5. ESM 导入时有符号绑定，CMJ 只是普通函数调用和赋值

7. ES6 中如何实现模块化的异步加载?

> 参考答案:
>
> 使用动态导入即可，导入后，得到的是一个 Promise，完成后，得到一个模块对象，其中包含了所有的导出结果。

8. 说一下 webpack 中的几种 hash 的实现原理是什么?

> 参考答案:
>
> - hash
>   hash 是跟整个项目的构建相关，只要项目里有文件更改，整个项目构建的 hash 值都会更改，并且全部文件都共用相同的 hash 值
> - chunkhash
>   每个打包过程单独的 hash 值，如果一个项目有多个 entry，则每个 entry 维护自己的 chunkhash。
> - contenthash
>   每个文件内容单独的 hash 值，它和打包结果文件内容有关，只要文件内容不变，contenthash 不变。

9. webpack 如果使用了 hash 命名，那是每次都会重新生成 hash 吗?

> 参考答案:
>
> 不会。它跟关联的内容是否有变化有关系，如果没有变化，hash 就不会变。具体来说，contenthash 和具体的打包文件内容有关，chunkhash 和某一 entry 为起点的打包过程中涉及的内容有关，hash 和整个工程所有模块内容有关。

10. webpack 中是如何处理图片的? （抖音直播）

> 参考答案:
>
> webpack 本身不处理图片，它会把图片内容仍然当做 JS 代码来解析，结果就是报错，打包失败。如果要处理图片，需要通过 loader 来处理。其中，url-loader 会把图片转换为 base64 编码，然后得到一个 dataurl，file-loader 则会将图片生成到打包目录中，然后得到一个资源路

> 径。但无论是哪一种 loader，它们的核心功能，都是把图片内容转换成 JS 代码，因为只有转换成 JS 代码，webpack 才能识别。

11. webpack 打包出来的 html 为什么 style 放在头部 script 放在底部?

> 说明：这道题的表述是有问题的，webpack 本身并不打包 html，相反，它如果遇到 html 代码会直接打包失败，因为 webpack 本身只能识别 JS。之所以能够打包出 html 文件，是因为插件或 loader 的作用，其中，比较常见的插件是 html-webpack-plugin。所以这道题的正确表述应该是：「html-webpack-plugin 打包出来的 html 为什么 style 放在头部 script 放在底部？」

> 参考答案：
>
> 浏览器在解析 HTML 时是从上到下进行解析的，当遇到样式和 JS 时，都会停止 HTML 解析，转而解析样式和执行 JS。而我们往往希望，页面的样式解析完成后再解析 HTML，这样可以避免页面闪烁，基于此，样式应该放到顶部；而相反的，我们希望在解析完 HTML 后再执行 JS，这样可以让用户尽快的看到页面，同时也让 JS 执行时能够拿到完整的 DOM 树，基于此，JS 代码应该放到底部。
>
> 不过，在 HTML5 中出现了 async 和 defer 属性，使用该属性可以更好的解决 JS 的问题，我们可以把 script 放到顶部，让浏览器尽快下载，但延迟执行。实际上，在新版本的 html-webpack-plugin 中，它已经这样做了。

12. webpack 配置如何实现开发环境不使用 cdn、生产环境使用 cdn?

> 要配置 CDN，有两个步骤：
>
> 1. 在 html 模板中直接加入 cdn 引用
> 2. 在 webpack 配置中，加入 `externals` 配置，告诉 webpack 不要打包其中的模块，转而使用全局变量
>
> 若要在开发环境中不使用 CDN，只需根据环境变量判断不同的环境，进行不同的打包处理即可。
>
> 1. 在 html 模板中使用 ejs 模板语法进行判断，只有在生产环境中引入 CDN
> 2. 在 webpack 配置中，可以根据 `process.env` 中的环境变量进行判断是否使用 `externals` 配置
> 3. 在 `package.json` 脚本中设置不同的环境变量完成打包或开发启动。

13. 介绍一下 webpack4 中的 tree-shaking 的工作流程?

> 推荐阅读：https://tsejx.github.io/webpack-guidebook/principle-analysis/operational-principle/tree-shaking

> 参考答案：

> tree-shaking 仅支持 ESM 的静态导入语法，对于 CMJ 或者 ESM 中的动态导入不支持 tree shaking。
>
> 具体流程主要分为两步：标记和删除
>
> 1. 标记
>   webpack 在分析依赖时，会使用注释的方式对导入和导出进行标记，对于模块中没有被其他模块用到的导出标记为 unused harmony export
> 2. 删除
>   之后在 Uglifyjs (或者其他类似的工具) 步骤进行代码精简，把标记为无用的代码删除。

14. 说一下 webpack loader 的作用是什么?

> 参考答案:
>
> 用于转换代码。有时是因为 webpack 无法识别某些内容，比如图片、css 等，需要由 loader 将其转换为 JS 代码。有时是因为某些代码需要被特殊处理，比如 JS 兼容性的处理，需要由 loader 将其进一步转换。不管是什么情况，loader 的作用只有一个，就是转换代码。

15. 在开发过程中如果需要对已有模块进行扩展，如何进行开发保证调用方不受影响?

> 参考答案:
>
> 实际上就是一个版本管理的问题。
>
> 如果此次模块升级只是修复了某一些 bug，作为补丁版本升级即可，不影响主版本和次版本号
>
> 如果此次模块升级会新增一些内容，完全兼容之前的 API，作为次版本升级即可
>
> 如果此次模块升级会修改之前的 API，则作为主版本升级
>
> 在开发项目时，让项目依赖模块的主版本，因此，当模块更新时，只要不是主版本更新，项目都可以非常方便的升级模块版本，无须改动任何代码。但若涉及主版本更新，项目可以完全无视此次版本更新，仍然使用之前的旧版本，无须改动任何代码；当然也可以升级主版本，但就会涉及代码的改动，这就好比跟将 vue2 升级到 vue3 会涉及大量改动一样。
>
> 而在开发模块时，在一开始就要精心设计 API，尽量保证 API 的接口稳定，不要经常变动主版本号。如果实在要更新主版本，就需要在一段时间内同时维护两个版本（新的主版本，旧的主版本），给予其他项目一定的升级时间。

16. export 和 export default 的区别是什么?

> 参考答案:
>
> export 为普通导出，又叫做具名导出，顾名思义，它导出的数据必须带有命名，比如变量定义、函数定义这种带有命名的语句。在导出的模块对象中，命名即为模块对象的属性名。在一

> 个模块中可以有多个具名导出
>
> export default 为默认导出，在模块对象中名称固定为 default，因此无须命名，通常导出一个表达式或字面量。在一个模块中只能有一个默认导出。

17. webpack 打包原理是什么?

> 参考答案:
>
> 1. **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
> 2. **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
> 3. **确定入口**：根据配置中的 `entry` 找出所有的入口文件
> 4. **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，再把翻译后的内容转换成 AST，通过对 AST 的分析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
> 5. **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 `依赖关系图`
> 6. **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
> 7. **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

18. webpack 热更新原理是什么?

> 参考答案:
>
> 当开启热更新后，页面中会植入一段 websocket 脚本，同时，开发服务器也会和客户端建立 websocket 通信，当源码发生变动时，webpack 会进行以下处理:
>
> 1. webpack 重新打包
> 2. webpack-dev-server 检测到模块的变化，于是通过 webscoket 告知客户端变化已经发生
> 3. 客户端收到消息后，通过 ajax 发送请求到开发服务器，以过去打包的 hash 值请求服务器的一个 json 文件
> 4. 服务器告诉客户端哪些模块发生了变动，同时告诉客户端这次打包产生的新 hash 值
> 5. 客户端再次用过去的 hash 值，以 JSONP 的方式请求变动的模块
> 6. 服务器响应一个函数调用，用于更新模块的代码
> 7. 此时，模块代码已经完成更新。客户端按照之前的监听配置，执行相应模块变动后的回调函数。

19. 如何优化 webpack 的打包速度?

> 参考答案：
>
> 1. noParse
>   很多第三方库本身就是已经打包好的代码，对于这种代码无须再进行解析，可以使用 noParse 配置排除掉这些第三方库
> 2. externals
>   对于一些知名的第三方库可以使用 CDN，这部分库可以通过 externals 配置不进行打包
> 3. 限制 loader 的范围
>   在使用 loader 的时候，可以通过 exclude 排除掉一些不必要的编译，比如 babel-loader 对于那些已经完成打包的第三方库没有必要再降级一次，可以排除掉
> 4. 开启 loader 缓存
>   可以利用 `cache-loader` 缓存 loader 的编译结果，避免在源码没有变动时反复编译
> 5. 开启多线程编译
>   可以利用 `thread-loader` 开启多线程编译，提升编译效率
> 6. 动态链接库
>   对于某些需要打包的第三方库，可以使用 dll 的方式单独对其打包，然后 DLLPlugin 将其整合到当前项目中，这样就避免了在开发中频繁去打包这些库

20. webpack 如何实现动态导入?

> 参考答案：
>
> 当遇到代码中包含动态导入语句时，webpack 会将导入的模块及其依赖分配到单独的一个 chunk 中进行打包，形成单独的打包结果。而动态导入的语句会被编译成一个普通的函数调用，该函数在执行时，会使用 JSONP 的方式动态的把分离出去的包加载到模块集合中。

21. 说一下 webpack 有哪几种文件指纹

> 参考答案：
>
> - hash
>   hash 是跟整个项目的构建相关，只要项目里有文件更改，整个项目构建的 hash 值都会更改，并且全部文件都共用相同的 hash 值
> - chunkhash
>   每个打包过程单独的 hash 值，如果一个项目有多个 entry，则每个 entry 维护自己的 chunkhash。
> - contenthash
>   每个文件内容单独的 hash 值，它和打包结果文件内容有关，只要文件内容不变，contenthash 不变。

22. 常用的 webpack Loader 都有哪些?

> 参考答案：

- cache-loader：启用编译缓存
- thread-loader：启用多线程编译
- css-loader：编译 css 代码为 js
- file-loader：保存文件到输出目录，将文件内容转换成文件路径
- postcss-loader：将 css 代码使用 postcss 进行编译
- url-loader：将文件内容转换成 dataurl
- less-loader：将 less 代码转换成 css 代码
- sass-loader：将 sass 代码转换成 css 代码
- vue-loader：编译单文件组件
- babel-loader：对 JS 代码进行降级处理

23. 说一下 webpack 常用插件都有哪些?

参考答案：

- clean-webpack-plugin：清除输出目录
- copy-webpack-plugin：复制文件到输出目录
- html-webpack-plugin：生成 HTML 文件
- mini-css-extract-plugin：将 css 打包成单独文件的插件
- HotModuleReplacementPlugin：热更新的插件
- purifycss-webpack：去除无用的 css 代码
- optimize-css-assets-webpack-plugin：优化 css 打包体积
- uglify-js-plugin：对 JS 代码进行压缩、混淆
- compression-webpack-plugin：gzip 压缩
- webpack-bundle-analyzer：分析打包结果

24. 使用 babel-loader 会有哪些问题，可以怎样优化?

参考答案：

1. 如果不做特殊处理，babel-loader 会对所有匹配的模块进行降级，这对于那些已经处理好兼容性问题的第三方库显得多此一举，因此可以使用 exclude 配置排除掉这些第三方库
2. 在旧版本的 babel-loader 中，默认开启了对 ESM 的转换，这样会导致 webpack 的 tree shaking 失效，因为 tree shaking 是需要保留 ESM 语法的，所以需要关闭 babel-loader 的 ESM 转换，在其新版本中已经默认关闭了。

25. babel 是如何对 class 进行编译的?

参考答案：

本质上就是把 class 语法转换成普通构造函数定义，并做了以下处理：

1. 增加了对 this 指向的检测

2. 将原型方法和静态方法变为不可枚举
3. 将整个代码放到了立即执行函数中，运行后返回构造函数本身

26. 解释一下 babel-polyfill 的作用是什么?

说明:

babel-polyfill 已经是一个非常古老的项目了，babel 从 7.4 版本开始已不再支持它，转而使用更加强大的 core-js，此题也适用于问「core-js 的作用是什么」

参考答案:

默认情况下，babel 本身只转换新的语法，而不对新的 API 进行处理。由于旧的环境中无法支持新的 API，比如 IE6 无法支持 fetch api，这就需要一个补丁，用旧语言的特性实现一遍新的 API，babel-polyfill 就是用来做这件事的。

27. 解释一下 less 的&的操作符是做什么用的?

参考答案:

&符号后面的内容会和父级选择器合并书写，即中间不加入空格字符

28. 在前端工程化中，可以进行哪些方面的优化?

参考答案:

1. 对传输性能的优化
   - 压缩和混淆
     使用 Uglifyjs 或其他类似工具对打包结果进行压缩、混淆，可以有效的减少包体积
   - tree shaking
     项目中尽量使用 ESM，可以有效利用 tree shaking 优化，降低包体积
   - 抽离公共模块
     将一些公共代码单独打包，这样可以充分利用浏览器缓存，其他代码变动后，不影响公共代码，浏览器可以直接从缓存中找到公共代码。
     具体方式有多种，比如 dll、splitChunks
   - 异步加载
     对一些可以延迟执行的模块可以使用动态导入的方式异步加载它们，这样在打包结果中，它们会形成单独的包，同时，在页面一开始解析时并不需要加载它们，而是页面解析完成后，执行 JS 的过程中去加载它们。
     这样可以显著提高页面的响应速度，在单页应用中尤其有用。
   - CDN
     对一些知名的库使用 CDN，不仅可以节省打包时间，还可以显著提升库的加载速度
   - gzip
     目前浏览器普遍支持 gzip 格式，因此可以将静态文件均使用 gzip 进行压缩

- 环境适配

  有些打包结果中包含了大量兼容性处理的代码，但在新版本浏览器中这些代码毫无意义。因此，可以把浏览器分为多个层次，为不同层次的浏览器给予不同的打包结果。

2. 对打包过程的优化
   - noParse

     很多第三方库本身就是已经打包好的代码，对于这种代码无须再进行解析，可以使用 noParse 配置排除掉这些第三方库
   - externals

     对于一些知名的第三方库可以使用 CDN，这部分库可以通过 externals 配置不进行打包
   - 限制 loader 的范围

     在使用 loader 的时候，可以通过 exclude 排除掉一些不必要的编译，比如 babel-loader 对于那些已经完成打包的第三方库没有必要再降级一次，可以排除掉
   - 开启 loader 缓存

     可以利用 `cache-loader` 缓存 loader 的编译结果，避免在源码没有变动时反复编译
   - 开启多线程编译

     可以利用 `thread-loader` 开启多线程编译，提升编译效率
   - 动态链接库

     对于某些需要打包的第三方库，可以使用 dll 的方式单独对其打包，然后 DLLPlugin 将其整合到当前项目中，这样就避免了在开发中频繁去打包这些库
3. 对开发体验的优化
   - lint

     使用 eslint、stylelint 等工具保证团队代码风格一致
   - HMR

     使用热替换避免页面刷新导致的状态丢失，提升开发体验

29. 如果有一个工程打包特别大-如何进行优化?

参考答案:

1. CDN

   如果工程中使用了一些知名的第三方库，可以考虑使用 CDN，而不进行打包
2. 抽离公共模块

   如果工程中用到了一些大的公共库，可以考虑将其分割出来单独打包
3. 异步加载

   对于那些不需要在一开始就执行的模块，可以考虑使用动态导入的方式异步加载它们，以尽量减少主包的体积
4. 压缩、混淆
5. tree shaking

   尽量使用 ESM 语法进行导入导出，充分利用 tree shaking 去除无用代码

6. gzip
   开启 gzip 压缩，进一步减少包体积
7. 环境适配
   有些打包结果中包含了大量兼容性处理的代码，但在新版本浏览器中这些代码毫无意义。因此，可以把浏览器分为多个层次，为不同层次的浏览器给予不同的打包结果。

30. webpack 怎么进行首屏加载的优化?

参考答案:

1. CDN
   如果工程中使用了一些知名的第三方库，可以考虑使用 CDN，而不进行打包
2. 抽离公共模块
   如果工程中用到了一些大的公共库，可以考虑将其分割出来单独打包
3. 异步加载
   对于那些不需要在一开始就执行的模块，可以考虑使用动态导入的方式异步加载它们，以尽量减少主包的体积
4. 压缩、混淆
5. tree shaking
   尽量使用 ESM 语法进行导入导出，充分利用 tree shaking 去除无用代码
6. gzip
   开启 gzip 压缩，进一步减少包体积
7. 环境适配
   有些打包结果中包含了大量兼容性处理的代码，但在新版本浏览器中这些代码毫无意义。因此，可以把浏览器分为多个层次，为不同层次的浏览器给予不同的打包结果。

31. 介绍一下 webpack scope hoisting?

参考答案:

scope hoisting 是 webpack 的内置优化，它是针对模块的优化，在生产环境打包时会自动开启。

在未开启scope hoisting时，webpack 会将每个模块的代码放置在一个独立的函数环境中，这样是为了保证模块的作用域互不干扰。

而 scope hoisting 的作用恰恰相反，是把多个模块的代码合并到一个函数环境中执行。在这一过程中，webpack 会按照顺序正确的合并模块代码，同时对涉及的标识符做适当处理以避免重名。

这样做的好处是减少了函数调用，对运行效率有一定提升，同时也降低了打包体积。

但 scope hoisting 的启用是有前提的，如果遇到某些模块多次被其他模块引用，或者使用了动态导入的模块，或者是非 ESM 的模块，都不会有 scope hoisting。

32. webpack proxy 工作原理，为什么能解决跨域?

> 说明:
>
> 严格来说，webpack 只是一个打包工具，它并没有 proxy 的功能，甚至连服务器的功能都没有。之所以能够在 webpack 中使用 proxy 配置，是因为它的一个插件，即 webpack-dev-server 的能力。
>
> 所以，此题应该问做：「webpack-dev-server 工作原理，为什么能解决跨域？」

> 参考答案:
>
> 首先，proxy 配置是针对开发环境的，对生产环境没有任何意义。
>
> 当我们通过 webpack-dev-server 启动开发服务器后，所有的打包资源都可以通过访问开发服务器获得。
>
> 与此同时，我们又配置了 proxy，当我们向开发服务器请求特定的地址时，开发服务器会将其代理到目标地址。因此，后续对代理地址的请求，就可以变为直接请求开发服务器。
>
> 这样一来，我们请求静态页面的域和请求代理地址的域就同源了，因为都是请求开发服务器，所以就不会产生跨域问题。

33. 组件发布的是不是所有依赖这个组件库的项目都需要升级?

> 参考答案:
>
> 实际上就是一个版本管理的问题。
>
> 如果此次模块升级只是修复了某一些 bug，作为补丁版本升级即可，不影响主版本和次版本号
>
> 如果此次模块升级会新增一些内容，完全兼容之前的 API，作为次版本升级即可
>
> 如果此次模块升级会修改之前的 API，则作为主版本升级
>
> 在开发项目时，让项目依赖模块的主版本，因此，当模块更新时，只要不是主版本更新，项目都可以非常方便的升级模块版本，无须改动任何代码。但若涉及主版本更新，项目可以完全无视此次版本更新，仍然使用之前的旧版本，无须改动任何代码；当然也可以升级主版本，但就会涉及代码的改动，这就好比跟将 vue2 升级到 vue3 会涉及大量改动一样。
>
> 而在开发模块时，在一开始就要精心设计 API，尽量保证 API 的接口稳定，不要经常变动主版本号。如果实在要更新主版本，就需要在一段时间内同时维护两个版本（新的主版本，旧的主版本），给予其他项目一定的升级时间。

34. 开发过程中，如何进行公共组件的设计？（字节跳动）

参考答案：

1. 确定使用场景
   明确这个公共组件的需求是怎么产生的，它目前的使用场景有哪些，将来还可能出现哪些使用场景。
   明确使用场景至关重要，它决定了这个组件的使用边界在哪，通用到什么程度，从而决定了这个组件的开发难度
2. 设计组件功能
   根据其使用场景，设计出组件的属性、事件、使用说明文档
3. 测试用例
   根据使用说明文档编写组件测试用例
4. 完成开发
   根据使用说明文档、测试用例完成开发

35. 说一下项目里有做过哪些 webpack 上的优化（字节跳动）

参考答案：

1. 对传输性能的优化
   - 压缩和混淆
     使用 Uglifyjs 或其他类似工具对打包结果进行压缩、混淆，可以有效的减少包体积
   - tree shaking
     项目中尽量使用 ESM，可以有效利用 tree shaking 优化，降低包体积
   - 抽离公共模块
     将一些公共代码单独打包，这样可以充分利用浏览器缓存，其他代码变动后，不影响公共代码，浏览器可以直接从缓存中找到公共代码。
     具体方式有多种，比如 dll、splitChunks
   - 异步加载
     对一些可以延迟执行的模块可以使用动态导入的方式异步加载它们，这样在打包结果中，它们会形成单独的包，同时，在页面一开始解析时并不需要加载它们，而是页面解析完成后，执行 JS 的过程中去加载它们。
     这样可以显著提高页面的响应速度，在单页应用中尤其有用。
   - CDN
     对一些知名的库使用 CDN，不仅可以节省打包时间，还可以显著提升库的加载速度
   - gzip
     目前浏览器普遍支持 gzip 格式，因此可以将静态文件均使用 gzip 进行压缩
   - 环境适配
     有些打包结果中包含了大量兼容性处理的代码，但在新版本浏览器中这些代码毫无意义。因此，可以把浏览器分为多个层次，为不同层次的浏览器给予不同的打包结果。
2. 对打包过程的优化

- noParse
  很多第三方库本身就是已经打包好的代码，对于这种代码无须再进行解析，可以使用 noParse 配置排除掉这些第三方库
- externals
  对于一些知名的第三方库可以使用 CDN，这部分库可以通过 externals 配置不进行打包
- 限制 loader 的范围
  在使用 loader 的时候，可以通过 exclude 排除掉一些不必要的编译，比如 babel-loader 对于那些已经完成打包的第三方库没有必要再降级一次，可以排除掉
- 开启 loader 缓存
  可以利用 `cache-loader` 缓存 loader 的编译结果，避免在源码没有变动时反复编译
- 开启多线程编译
  可以利用 `thread-loader` 开启多线程编译，提升编译效率
- 动态链接库
  对于某些需要打包的第三方库，可以使用 dll 的方式单独对其打包，然后 DLLPlugin 将其整合到当前项目中，这样就避免了在开发中频繁去打包这些库

3. 对开发体验的优化
   - lint
     使用 eslint、stylelint 等工具保证团队代码风格一致
   - HMR
     使用热替换避免页面刷新导致的状态丢失，提升开发体验

36. 具体说一下 splitchunksplugin 的使用场景及使用方法。（字节跳动）

参考答案：

1. 公共模块
   比如某些多页应用会有多个入口，从而形成多个 chunk，而这些 chunk 中用到了一些公共模块，为了减少整体的包体积，可以使用 splitchunksplugin 将公共模块分离出来。
   可以配置 minChunks 来指定被多少个 chunk 引用时进行分包
2. 并行下载
   由于 HTML5 支持 defer 和 async，因此可以同时下载多个 JS 文件以充分利用带宽。如果打包结果是一个很大的文件，就无法利用到这一点。
   可以利用 splitchunks 插件将文件进行拆分，通过配置 maxSize 属性指定包体积达到多大时进行拆分

37. 描述一下 webpack 的构建流程？（CVTE）

参考答案：

1. **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

2. **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
3. **确定入口**：根据配置中的 `entry` 找出所有的入口文件
4. **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，再把翻译后的内容转换成 AST，通过对 AST 的分析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
5. **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 `依赖关系图`
6. **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
7. **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

38. 解释一下 webpack 插件的实现原理？（CVTE）

参考答案：

本质上，webpack 的插件是一个带有 `apply` 函数的对象。当 webpack 创建好 compiler 对象后，会执行注册插件的 apply 函数，同时将 compiler 对象作为参数传入。

在 apply 函数中，开发者可以通过 compiler 对象监听多个钩子函数的执行，不同的钩子函数对应 webpack 编译的不同阶段。当 webpack 进行到一定阶段后，会调用这些监听函数，同时将 compilation 对象传入。开发者可以使用 compilation 对象获取和改变 webpack 的各种信息，从而影响构建过程。

39. 有用过哪些插件做项目的分析吗？（CVTE）

参考答案：

用过 webpack-bundle-analyzer 分析过打包结果，主要用于优化项目打包体积

40. 什么是 babel，有什么作用？

参考答案：

babel 是一个 JS 编译器，主要用于将下一代的 JS 语言代码编译成兼容性更好的代码。

它其实本身做的事情并不多，它负责将 JS 代码编译成为 AST，然后依托其生态中的各种插件对 AST 中的语法和 API 进行处理

41. 解释一下 npm 模块安装机制是什么？

参考答案：

> 1. npm 会检查本地的 node_modules 目录中是否已经安装过该模块，如果已经安装，则不再重新安装
> 2. npm 检查缓存中是否有相同的模块，如果有，直接从缓存中读取安装
> 3. 如果本地和缓存中均不存在，npm 会从 registry 指定的地址下载安装包，然后将其写入到本地的 node_modules 目录中，同时缓存起来。