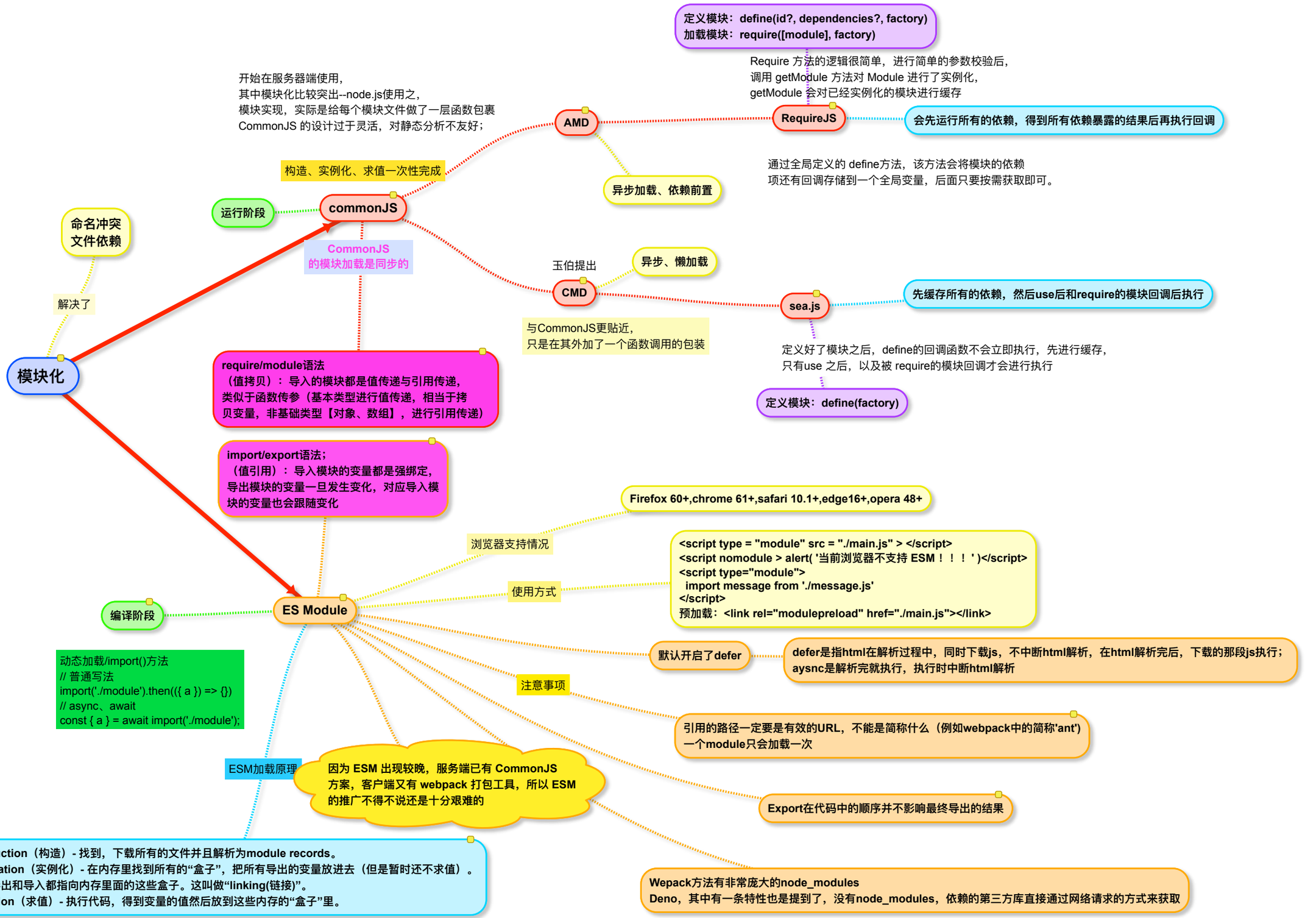
模块化
命名冲突
文件依赖
解决了
运行阶段
commonJS
构造、实例化、求值一次性完成
开始在服务器端使用，
其中模块化比较突出--node.js使用之，
模块实现，实际是给每个模块文件做了一层函数包裹
CommonJS 的设计过于灵活，对静态分析不友好；
CommonJS
的模块加载是同步的
require/module语法
（值拷贝）：导入的模块都是值传递与引用传递，
类似于函数传参（基本类型进行值传递，相当于拷贝变量，非基础类型【对象、数组】，进行引用传递）
AMD
定义模块：define(id?, dependencies?, factory)
加载模块：require([module], factory)
RequireJS
Require 方法的逻辑很简单，进行简单的参数校验后，
调用 getModule 方法对 Module 进行了实例化，
getModule 会对已经实例化的模块进行缓存
会先运行所有的依赖，得到所有依赖暴露的结果后再执行回调
异步加载、依赖前置
通过全局定义的 define方法，该方法会将模块的依赖
项还有回调存储到一个全局变量，后面只要按需获取即可。
CMD
玉伯提出
异步、懒加载
与CommonJS更贴近，
只是在其外加了一个函数调用的包装
sea.js
先缓存所有的依赖，然后use后和require的模块回调后执行
定义好了模块之后，define的回调函数不会立即执行，先进行缓存，
只有use 之后，以及被 require的模块回调才会进行执行
定义模块：define(factory)
import/export语法；
（值引用）：导入模块的变量都是强绑定，
导出模块的变量一旦发生变化，对应导入模块的变量也会跟随变化
ES Module
编译阶段
浏览器支持情况
Firefox 60+,chrome 61+,safari 10.1+,edge16+,opera 48+
使用方式
<script type = "module" src = "./main.js" > </script>
<script nomodule > alert('当前浏览器不支持 ESM ！ ！ ！ ')</script>
<script type="module">
import message from './message.js'
</script>
预加载：<link rel="modulepreload" href="./main.js"></link>
默认开启了defer
defer是指html在解析过程中，同时下载js，不中断html解析，在html解析完后，下载的那段js执行；
aysnc是解析完就执行，执行时中断html解析
注意事项
引用的路径一定要是有效的URL，不能是简称什么（例如webpack中的简称'ant'）
一个module只会加载一次
Export在代码中的顺序并不影响最终导出的结果
Wepack方法有非常庞大的node_modules
Deno，其中有一条特性也是提到了，没有node_modules，依赖的第三方库直接通过网络请求的方式来获取
动态加载/import()方法
// 普通写法
import('./module').then(({ a }) => {})
// async、await
const { a } = await import('./module');
ESM加载原理
因为 ESM 出现较晚，服务端已有 CommonJS 方案，客户端又有 webpack 打包工具，所以 ESM 的推广不得不说还是十分艰难的
1: Construction（构造）- 找到，下载所有的文件并且解析为module records。
2: Instantiation（实例化）- 在内存里找到所有的“盒子”，把所有导出的变量放进去（但是暂时还不求值）。
然后，让导出和导入都指向内存里面的这些盒子。这叫做“linking(链接)”。
3: Evaluation（求值）- 执行代码，得到变量的值然后放到这些内存的“盒子”里。