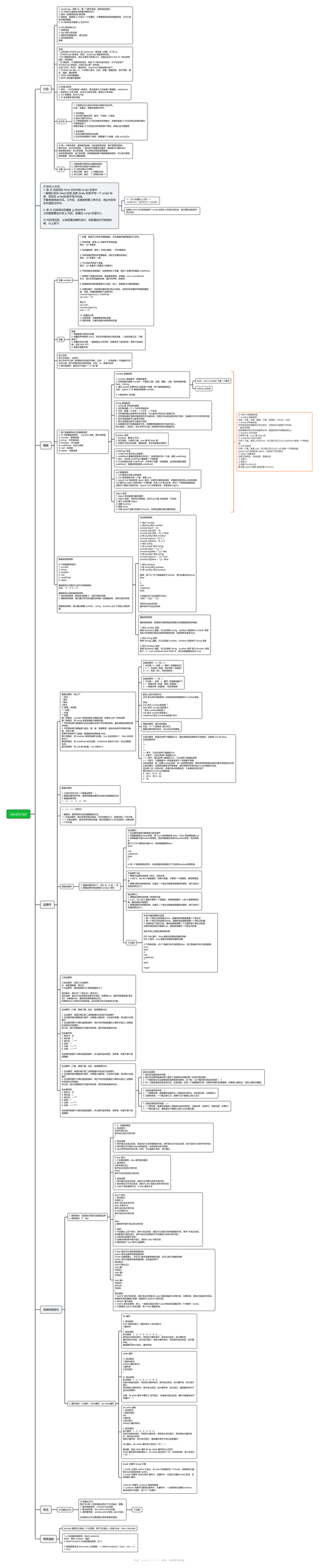
JavaScript
介绍
1. JavaScript，简称 JS，是一门基本语法、面向对象的语言。
2. 浏览器内核分为渲染引擎和JS引擎。
3. 前端一般指浏览器端的应用。
4. 浏览器端是 B 平台上一个重要的，不要轻视浏览器内部的复杂性，它可以调动底层的能力。
5. 代码用作脚本 js 的文件中
6. 可以用作脚本的文件
.mjs
.cjs
.js（默认）
7. 脚本文件的扩展名
.mjs 模块脚本
.cjs 脚本
.js 模块脚本
8. 脚本文件的编码
utf-8
9. 脚本文件的大小
2 GB（理论上限）
10. 在浏览器中的代码
JS 代码可以通过多种方式引入到网页中，这些方式各有优缺点，以下是详细说明：
1. 内联脚本（Inline Script）
直接在 HTML 文件中写入 JS 代码，适合小型脚本或测试。
2. 外部脚本（External Script）
通过 .js 文件引入，适合大型项目，便于维护和复用。
3. 模块脚本（Module Script）
使用 import/export 语法，支持模块化开发，适合现代项目。
4. 动态脚本（Dynamic Script）
通过 DOM 操作动态创建脚本，适合按需加载。
5. 浏览器内核
JS 代码运行的环境，不同内核对 JS 的支持略有差异。
6. 引擎
JS 代码的执行者，负责解析、编译和执行代码。
7. 渲染引擎
负责解析 HTML 和 CSS，与 JS 引擎协作。
8. 内存管理
JS 引擎自动管理内存，但需注意内存泄漏。
9. 性能
合理使用内存和优化代码可提升性能。
10. 安全
浏览器对 JS 有沙箱限制，防止恶意代码。
11. 兼容性
不同浏览器对 JS 的支持可能有差异，需测试。
12. 调试
使用开发者工具（如 Chrome DevTools）调试 JS 代码。
13. 现代特性
ES6+ 引入了许多新特性，如箭头函数、解构、模块等。
14. 工具链
使用 Babel、Webpack 等工具优化开发流程。
15. 未来
关注 ECMAScript 提案，了解语言的发展趋势。
16. 学习资源
MDN、官方文档、社区教程等是学习的好资源。
17. 实践
通过项目实践来巩固知识，解决实际问题。
18. 工具
使用 ESLint、Prettier 等工具保持代码质量。
19. 社区
参与开源项目，与开发者交流。
20. 总结
JS 是一门强大的语言，掌握其特性和生态，能显著提升开发效率。
JS 的引入方式：
1. 内联 JS（Inline Script）
直接在 HTML 文件中写入 JS 代码，适合小型脚本或测试。
2. 外部 JS（External Script）
通过 .js 文件引入，适合大型项目，便于维护和复用。
3. 模块 JS（Module Script）
使用 import/export 语法，支持模块化开发，适合现代项目。
4. 动态 JS（Dynamic Script）
通过 DOM 操作动态创建脚本，适合按需加载。
5. 浏览器内核
JS 代码运行的环境，不同内核对 JS 的支持略有差异。
6. 引擎
JS 代码的执行者，负责解析、编译和执行代码。
7. 渲染引擎
负责解析 HTML 和 CSS，与 JS 引擎协作。
8. 内存管理
JS 引擎自动管理内存，但需注意内存泄漏。
9. 性能
合理使用内存和优化代码可提升性能。
10. 安全
浏览器对 JS 有沙箱限制，防止恶意代码。
11. 兼容性
不同浏览器对 JS 的支持可能有差异，需测试。
12. 调试
使用开发者工具（如 Chrome DevTools）调试 JS 代码。
13. 现代特性
ES6+ 引入了许多新特性，如箭头函数、解构、模块等。
14. 工具链
使用 Babel、Webpack 等工具优化开发流程。
15. 未来
关注 ECMAScript 提案，了解语言的发展趋势。
16. 学习资源
MDN、官方文档、社区教程等是学习的好资源。
17. 实践
通过项目实践来巩固知识，解决实际问题。
18. 工具
使用 ESLint、Prettier 等工具保持代码质量。
19. 社区
参与开源项目，与开发者交流。
20. 总结
JS 是一门强大的语言，掌握其特性和生态，能显著提升开发效率。
JS 代码可以通过多种方式引入到网页中，这些方式各有优缺点，以下是详细说明：
1. 内联脚本（Inline Script）
直接在 HTML 文件中写入 JS 代码，适合小型脚本或测试。
2. 外部脚本（External Script）
通过 .js 文件引入，适合大型项目，便于维护和复用。
3. 模块脚本（Module Script）
使用 import/export 语法，支持模块化开发，适合现代项目。
4. 动态脚本（Dynamic Script）
通过 DOM 操作动态创建脚本，适合按需加载。
5. 浏览器内核
JS 代码运行的环境，不同内核对 JS 的支持略有差异。
6. 引擎
JS 代码的执行者，负责解析、编译和执行代码。
7. 渲染引擎
负责解析 HTML 和 CSS，与 JS 引擎协作。
8. 内存管理
JS 引擎自动管理内存，但需注意内存泄漏。
9. 性能
合理使用内存和优化代码可提升性能。
10. 安全
浏览器对 JS 有沙箱限制，防止恶意代码。
11. 兼容性
不同浏览器对 JS 的支持可能有差异，需测试。
12. 调试
使用开发者工具（如 Chrome DevTools）调试 JS 代码。
13. 现代特性
ES6+ 引入了许多新特性，如箭头函数、解构、模块等。
14. 工具链
使用 Babel、Webpack 等工具优化开发流程。
15. 未来
关注 ECMAScript 提案，了解语言的发展趋势。
16. 学习资源
MDN、官方文档、社区教程等是学习的好资源。
17. 实践
通过项目实践来巩固知识，解决实际问题。
18. 工具
使用 ESLint、Prettier 等工具保持代码质量。
19. 社区
参与开源项目，与开发者交流。
20. 总结
JS 是一门强大的语言，掌握其特性和生态，能显著提升开发效率。
数据
变量 var/let
常量
数据类型
运算符
流程控制语句
语法
特殊函数