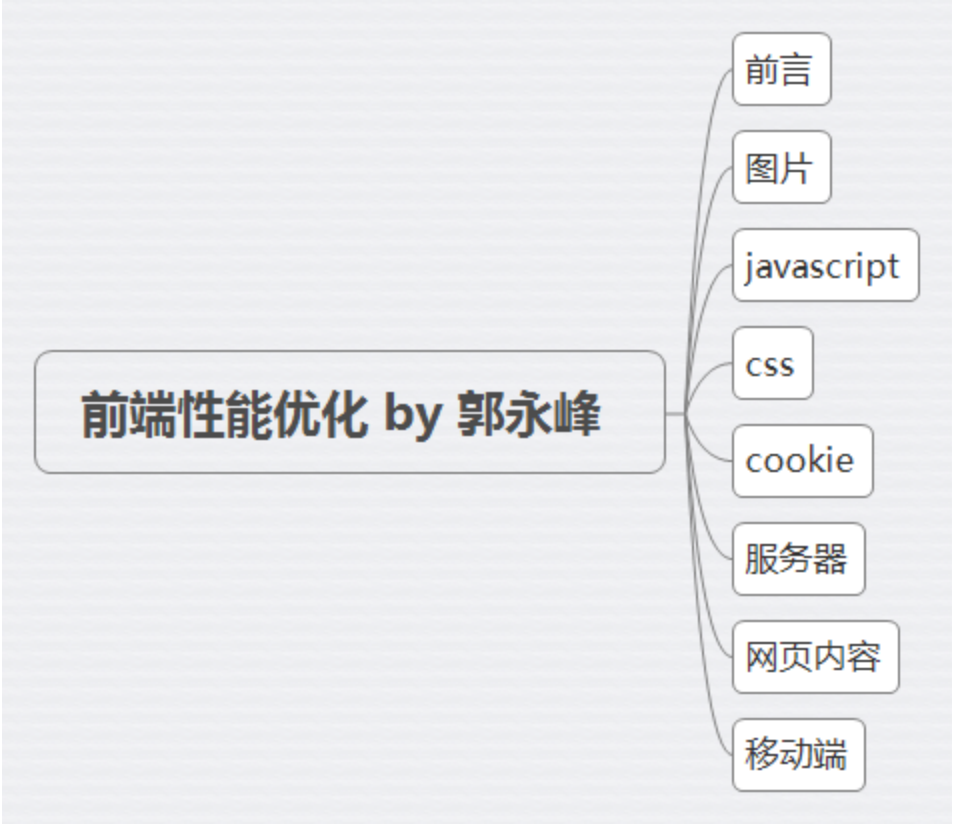
前端性能优化 by 郭永峰

前端性能优化 by 郭永峰.....1
1. 前言.....5
1.1.
前端性能优化不完全总结，性能优化任重而道远。但应该切记的是，不可为了优化而优化，性能优化和开发效率、用户体验以及实现成本等是相结合的。5
1.2.
性能优化，其实现手段多样，关注领域复杂，从前到后，自上而下，方方面面都是性能优化的地方。.....5
2. 图片.....6
2.1. 优化图像.....6
2.1.1. 检查gif图片图像颜色数量和调色板规格是否一致.....6
2.1.2. 有gif的情况考虑换成png.....7
2.1.3. 色值较少的图片切成jpg.....7
2.2. Inline images.....7
2.2.1. 通过编码的字符串将图片直接内嵌在网页文本中.....7
2.2.2. base64编码.....7
2.3. image Maps.....7
2.3.1. 将多张图拼在一起，通过坐标来控制显示导航.....7
2.4. 优化css sptite，合成雪碧图.....7
2.4.1. 可以选用单独的工具来生成雪碧图.....8
2.4.2. 使用构建工具来分析产出雪碧图.....8
2.5. 不在html中缩放图片.....8
2.6. 使用小且可缓存defavicon.ico.....8
2.6.1. 浏览器总会去请求这个图标.....8
2.6.2. 确保图标存在，否则报错.....8
2.6.3. 文件尽量小，最好小于1K.....8
2.6.4. 设置一个长的过期时间.....8
2.7. todo.....8
2.7.1. 规范.....9
2.7.2. 工具.....9
3. javascript.....9
3.1. js脚本文件置底.....9
3.2. 现在主流浏览器都支持defer关键字，可以指定脚本在文档加载后执行.....9
3.3. 使用外链js和css文件.....9
3.4. 精简代码.....9

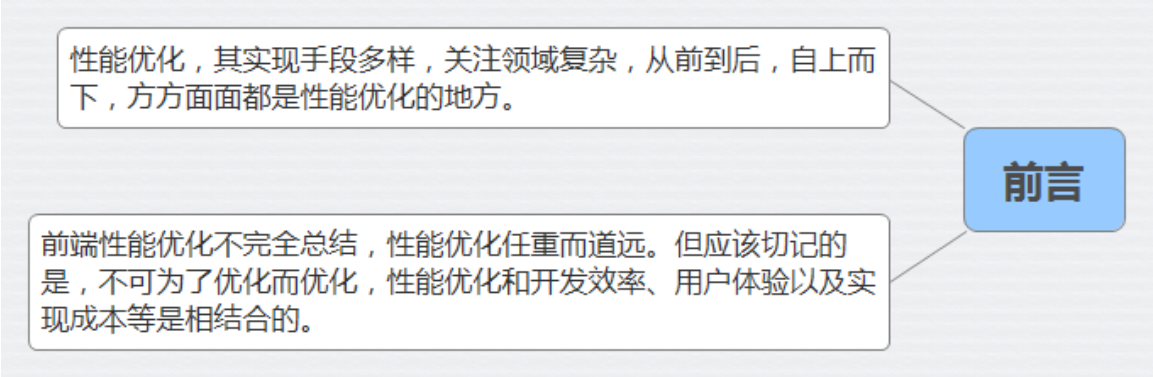
3.5. 减少DOM访问和查询 10
3.5.1. 换成已经访问过的元素 10
3.5.2. 优化dom的选择器 10
3.5.3. 避免通过js修复layout 10
3.5.4. 优化逻辑减少dom操作 10
3.5.5. 双向数据绑定 10
3.5.6. dom diff 10
3.5.7. dom拼接 11
3.6. 减少或避免引起页面repaint或reflow的代码逻辑 11
3.7. 使用智能事件处理方式 11
3.7.1. 事件代理 11
3.8. 语言基础 11
3.9. todo 11
3.9.1. 形成规范进行培训 11
3.9.2. 代码检查工具 11
3.9.3. 性能监测工具 12
3.9.4. 代码走查 12
3.9.5. 将现有问题形成开发任务 12
4. css 12
4.1. 样式表置顶 12
4.2. 避免css表达式 12
4.3. 用link代替@import 13
4.4. 减少层级 13
4.5. 避免filters 13
4.6. 避免冗余 13
4.7. css文件合理管理 13
4.8. todo 13
4.8.1. 代码规则检查工具 13
4.8.2. 代码规范培训 13
4.8.3. 子主题 3 13
5. cookie 13
5.1. 减少cookie大小 14
5.1.1. cookie用来做认证和个性化设置，在请求中包含在http报文头 14
5.1.2. 如果网页不需要用cookie，就完全禁掉 14
5.1.3. 去除没必要的cookie 14
5.1.4. cookie的domain不可影响到子domain 14
5.1.5. 设置合适的过期时间 14
5.2. 页面内容使用无cookie域名 14

6. 服务器 14
6.1. 使用cdn分发资源 15
6.1.1. CDN通过部署在不同地区的服务器来提高客户的下载速度 15
6.1.2. 可将站点大量的静态内容放上cdn 15
6.2. 添加expires或cache-control报文头 15
6.2.1. 静态内容添加expires，将静态内容设置为永不过期 16
6.2.2. 动态内容适合cache-control，让浏览器根据条件来发送请求 16
6.3. gzip压缩 16
6.4. 配置etags 16
6.5. flush输出 16
6.6. get请求 16
6.7. 避免空的图片src 16
6.7.1.
空的图片src仍然会使浏览器发送请求到服务器，这样完全是浪费时间，而且浪费服务器的资源 16
7. 网页内容 16
7.1. 减少http请求次数 17
7.1.1. combo合并多个文件请求为一个 18
7.1.2. 打包合并文件，打包策略很重要 18
7.2. 减少dns查询次数 18
7.2.1.
愿意在于网页中包含不同domain的内容时，如嵌入广告或引用了外部图片或脚本导致 18
7.2.2.
dns查询结果可缓存在本地和浏览器一段时间，所以一般在首次访问时消耗流量 18
7.3. 避免页面跳转 18
7.4. 缓存ajax请求 18
7.5. 延迟加载 18
7.5.1.
确定页面初始加载需要的最小内容集，剩下内容推到延迟加载的集合中 19
7.6. 提前加载 19
7.6.1. 有条件提前加载 19
7.6.1.1. 用户的输入推断需要加载的内容，如智能搜索框 19
7.6.2. 无条件提前加载 19

7.6.2.1.
当前网页加载完成后，马上去下载一些其他的内容，这些内容是必须会用到的或是公共依赖的 20
7.7. 减少DOM元素数量 20
7.8. 根据域名划分内容 20
7.9. 减少iframe数量 20
7.9.1. 优点 20
7.9.1.1. 用来加载速度较慢的广告 20
7.9.1.2. 安全沙箱保护 20
7.9.1.3. 脚本可以并行下载 20
7.9.2. 缺点 20
7.9.2.1. 即使iframe内容为空也消耗加载时间 21
7.9.2.2. 会阻止页面的加载 21
7.10. 避免404 21
8. 移动端 21
8.1. localStorage本地存储与优化 21
8.1.1.
大数据量交互，数据不怎么更新的，含版本控制机制，一次请求，之后高枕无忧 22
8.1.2. 充分利用其资源 22
8.2. 保持单个内容小于25K 22
8.2.1. 这是因为iphone的限制，它只能缓存解压后小于25K的资源 22
8.2.2. 所以单纯gzip不一定够用，精简文件工具 22
8.3. 打包文档 22
8.4. quikling方案 22



1. 前言

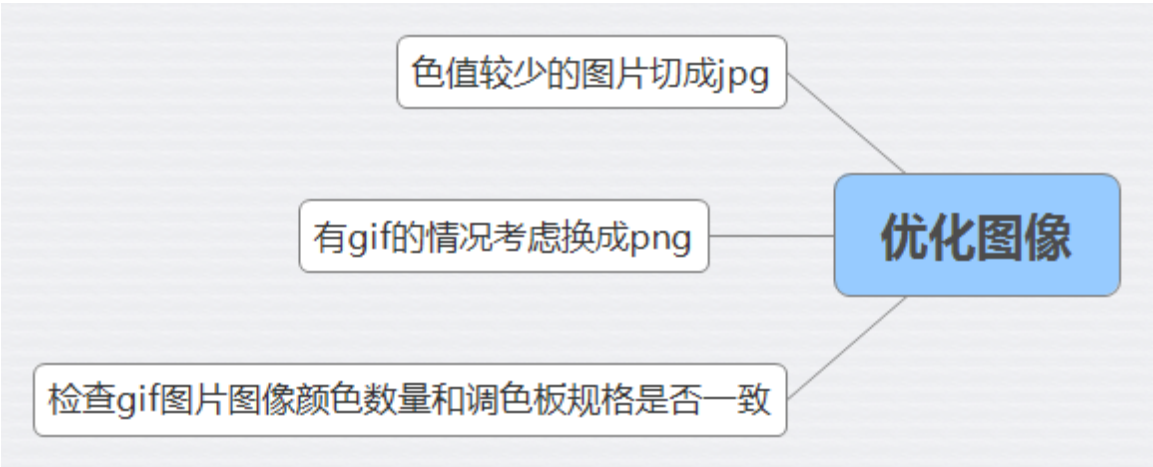


1.1. 前端性能优化不完全总结，性能优化任重而道远。但应该切记的是，不可为了优化而优化，性能优化和开发效率、用户体验以及实现成本等是相结合的。

1.2. 性能优化，其实现手段多样，关注领域复杂，从前到后，自上而下，方方面面都是性能优化的地方。

2. 图片

2.1. 优化图像

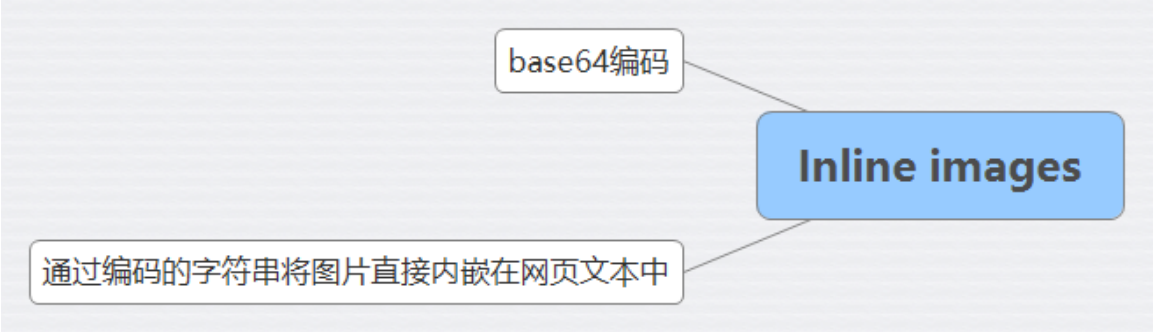


2.1.1. 检查gif图片图像颜色数量和调色板规格是否一致

2.1.2. 有gif的情况考虑换成png

2.1.3. 色值较少的图片切成jpg

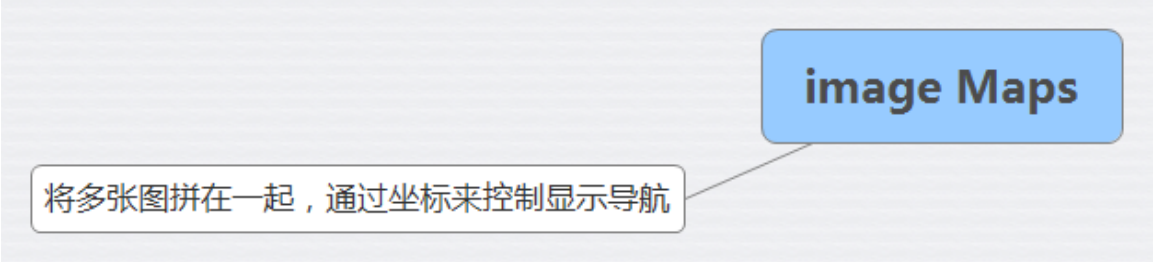
2.2. Inline images



2.2.1. 通过编码的字符串将图片直接内嵌在网页文本中

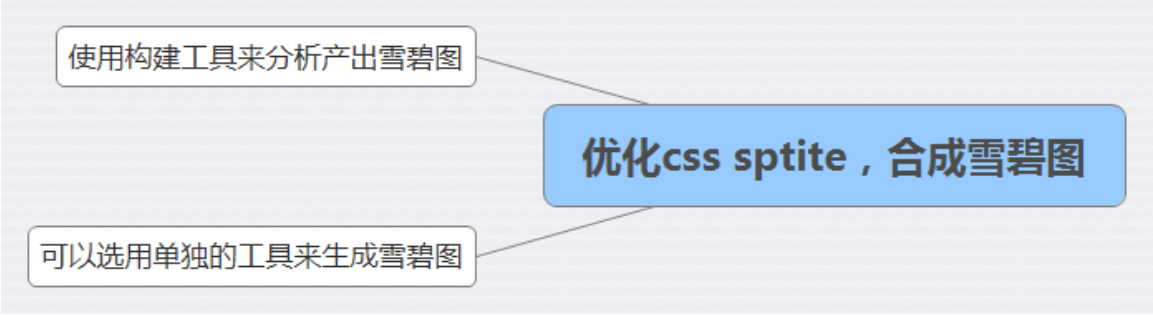
2.2.2. base64编码

2.3. image Maps



2.3.1. 将多张图拼在一起，通过坐标来控制显示导航

2.4. 优化css sptite，合成雪碧图



2.4.1. 可以选用单独的工具来生成雪碧图

2.4.2. 使用构建工具来分析产出雪碧图

2.5. 不在html中缩放图片

2.6. 使用小且可缓存defavicon.ico

2.6.1. 浏览器总会去请求这个图标

2.6.2. 确保图标存在，否则报错

2.6.3. 文件尽量小，最好小于1K

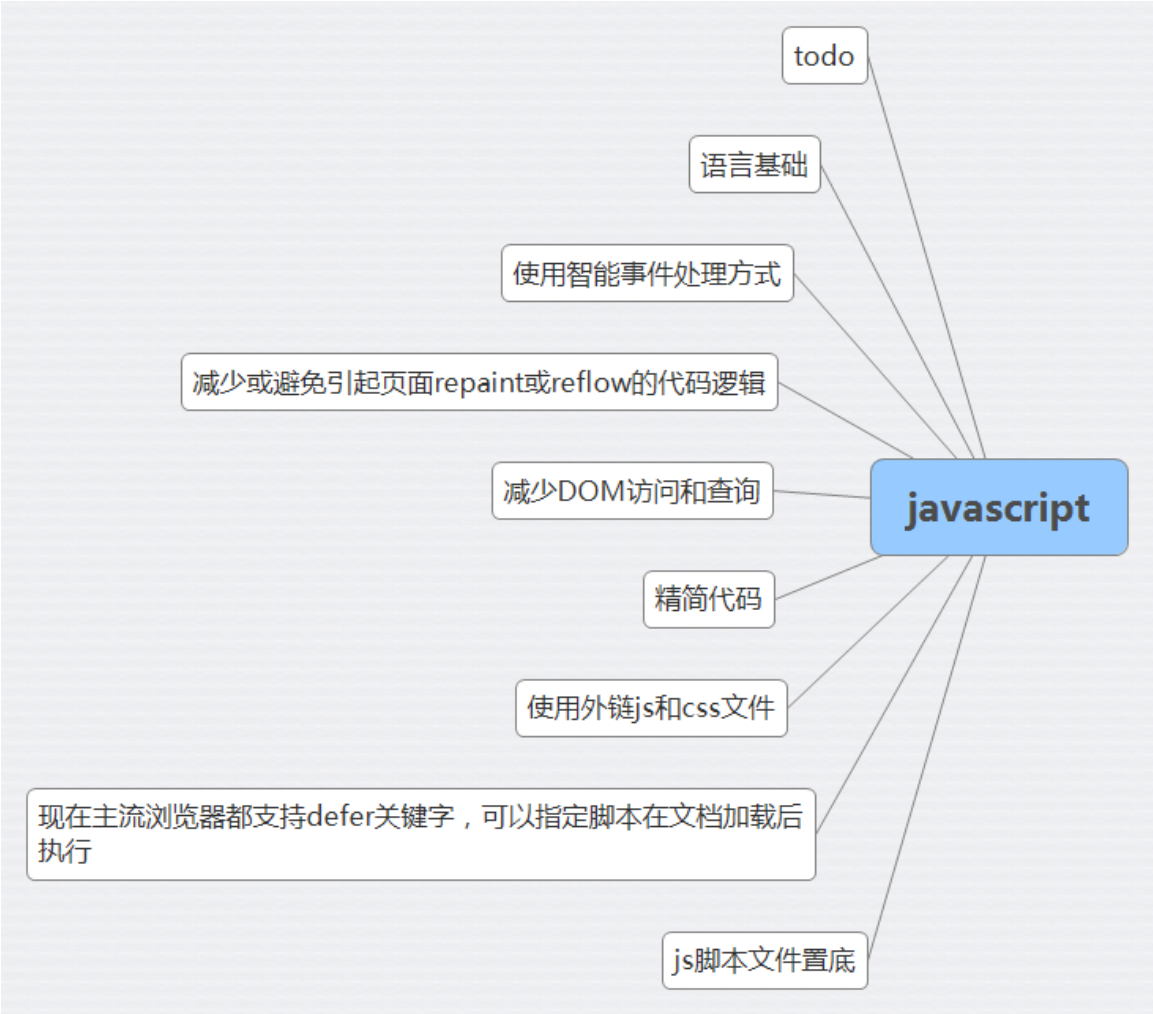
2.6.4. 设置一个长的过期时间

2.7. todo

2.7.1. 规范

2.7.2. 工具

3. javascript



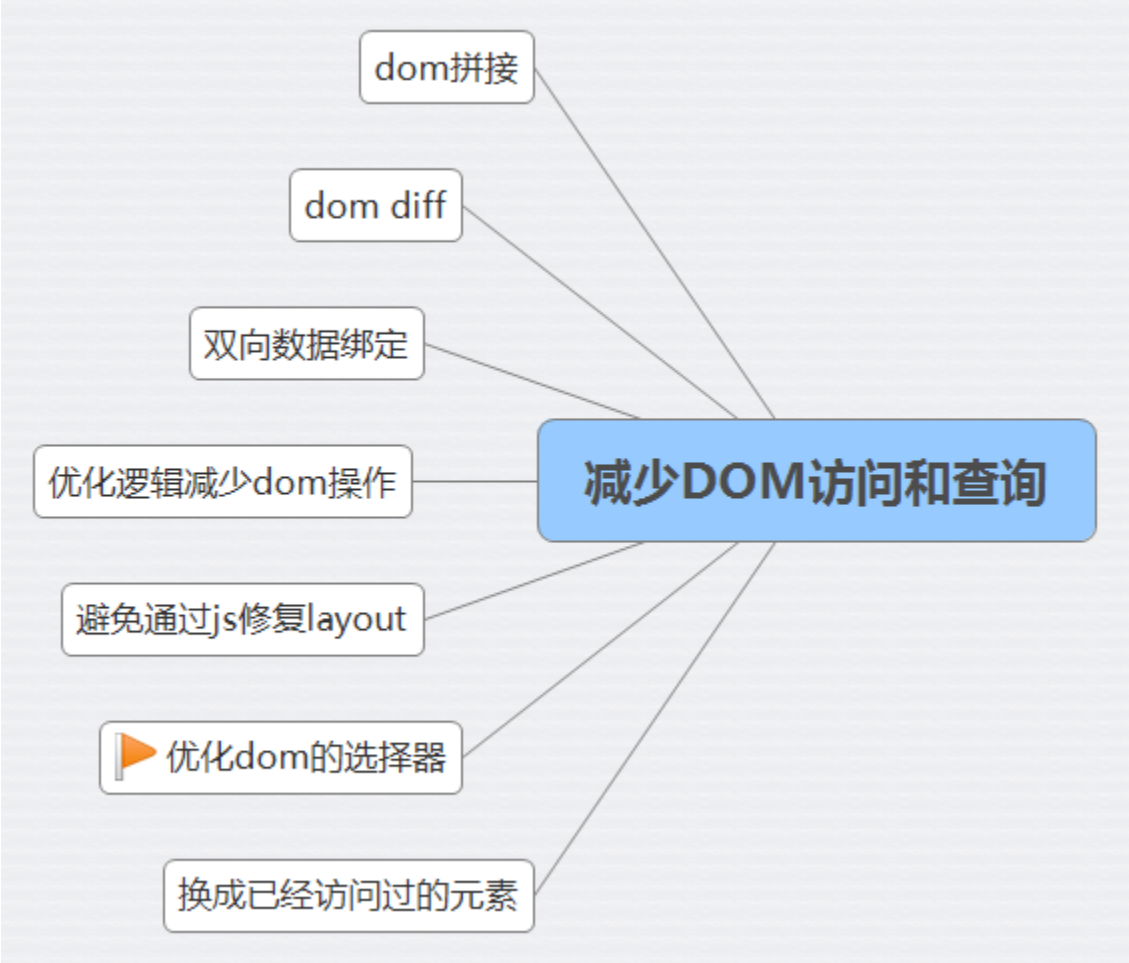
3.1. js脚本文件置底

3.2. 现在主流浏览器都支持defer关键字，可以指定脚本在文档加载后执行

3.3. 使用外链js和css文件

3.4. 精简代码

3.5. 减少DOM访问和查询



3.5.1. 换成已经访问过的元素

3.5.2. 优化dom的选择器

3.5.3. 避免通过js修复layout

3.5.4. 优化逻辑减少dom操作

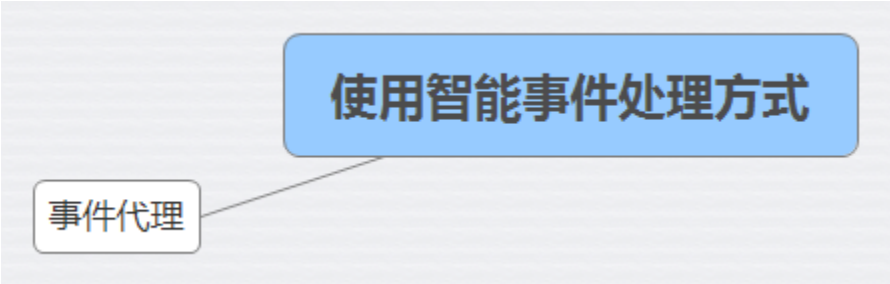
3.5.5. 双向数据绑定

3.5.6. dom diff

3.5.7. dom拼接

3.6. 减少或避免引起页面repaint或reflow的代码逻辑

3.7. 使用智能事件处理方式



3.7.1. 事件代理

3.8. 语言基础

3.9. todo

3.9.1. 形成规范进行培训

3.9.2. 代码检查工具

3.9.3. 性能监测工具

3.9.4. 代码走查

3.9.5. 将现有问题形成开发任务

4. css

4.1. 样式表置顶

4.2. 避免css表达式

4.3. 用link代替@import

4.4. 减少层级

4.5. 避免filters

4.6. 避免冗余

4.7. css文件合理管理

4.8. todo

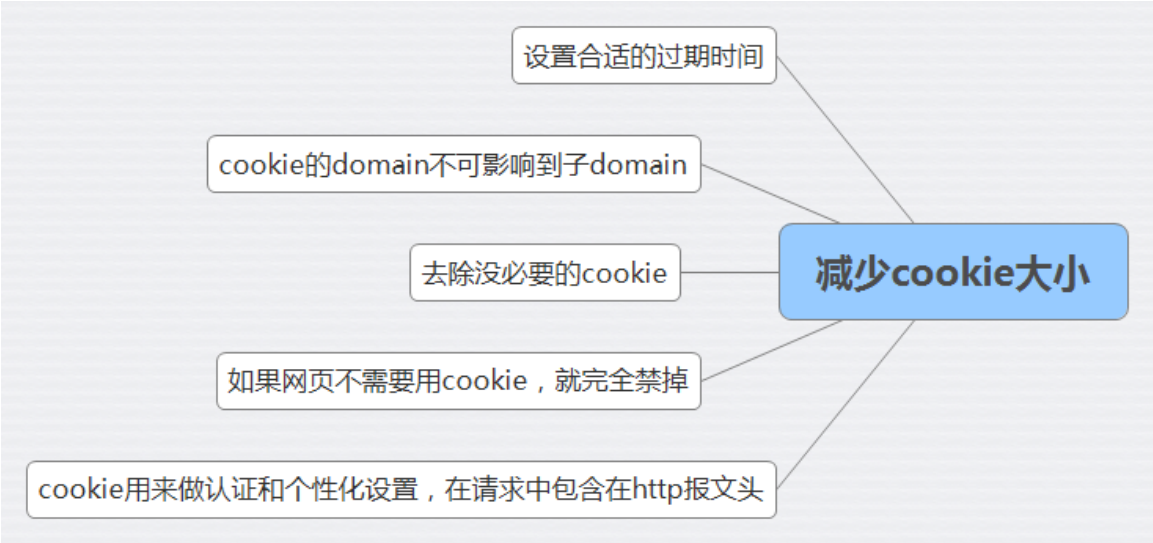
4.8.1. 代码规则检查工具

4.8.2. 代码规范培训

4.8.3. 子主题 3

5. cookie

5.1. 减少cookie大小



5.1.1. cookie用来做认证和个性化设置，在请求中包含在http报文头

5.1.2. 如果网页不需要用cookie，就完全禁掉

5.1.3. 去除没必要的cookie

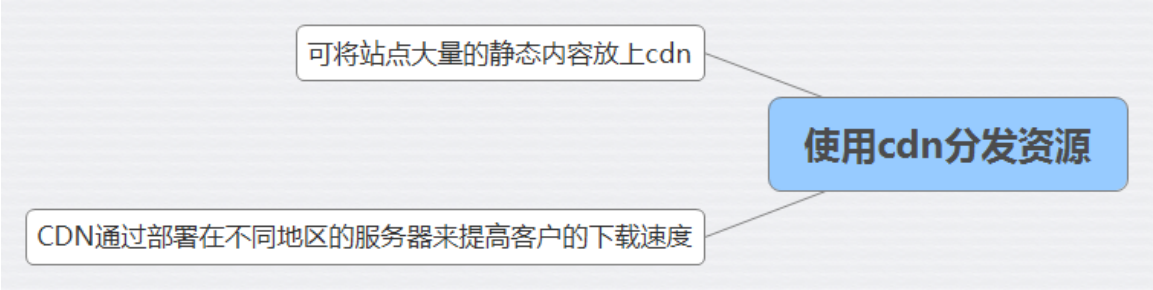
5.1.4. cookie的domain不可影响到子domain

5.1.5. 设置合适的过期时间

5.2. 页面内容使用无cookie域名

6. 服务器

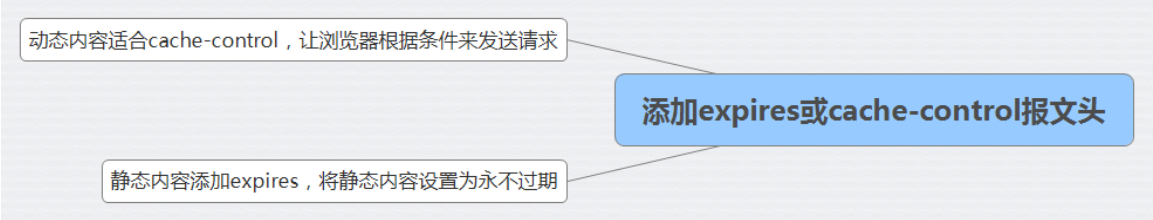
6.1. 使用cdn分发资源



6.1.1. CDN通过部署在不同地区的服务器来提高客户的下载速度

6.1.2. 可将站点大量的静态内容放上cdn

6.2. 添加expires或cache-control报文头



6.2.1. 静态内容添加expires，将静态内容设置为永不过期

6.2.2. 动态内容适合cache-control，让浏览器根据条件来发送请求

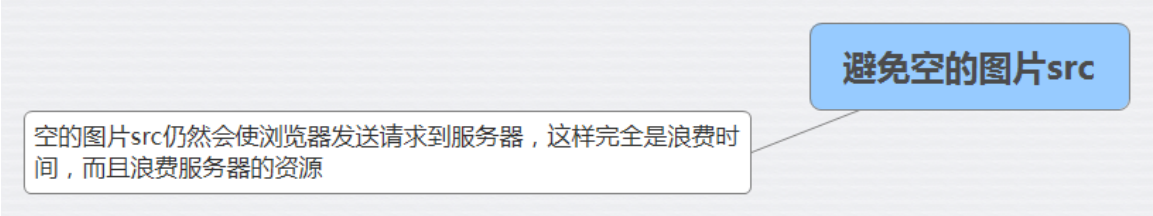
6.3. gzip压缩

6.4. 配置etags

6.5. flush输出

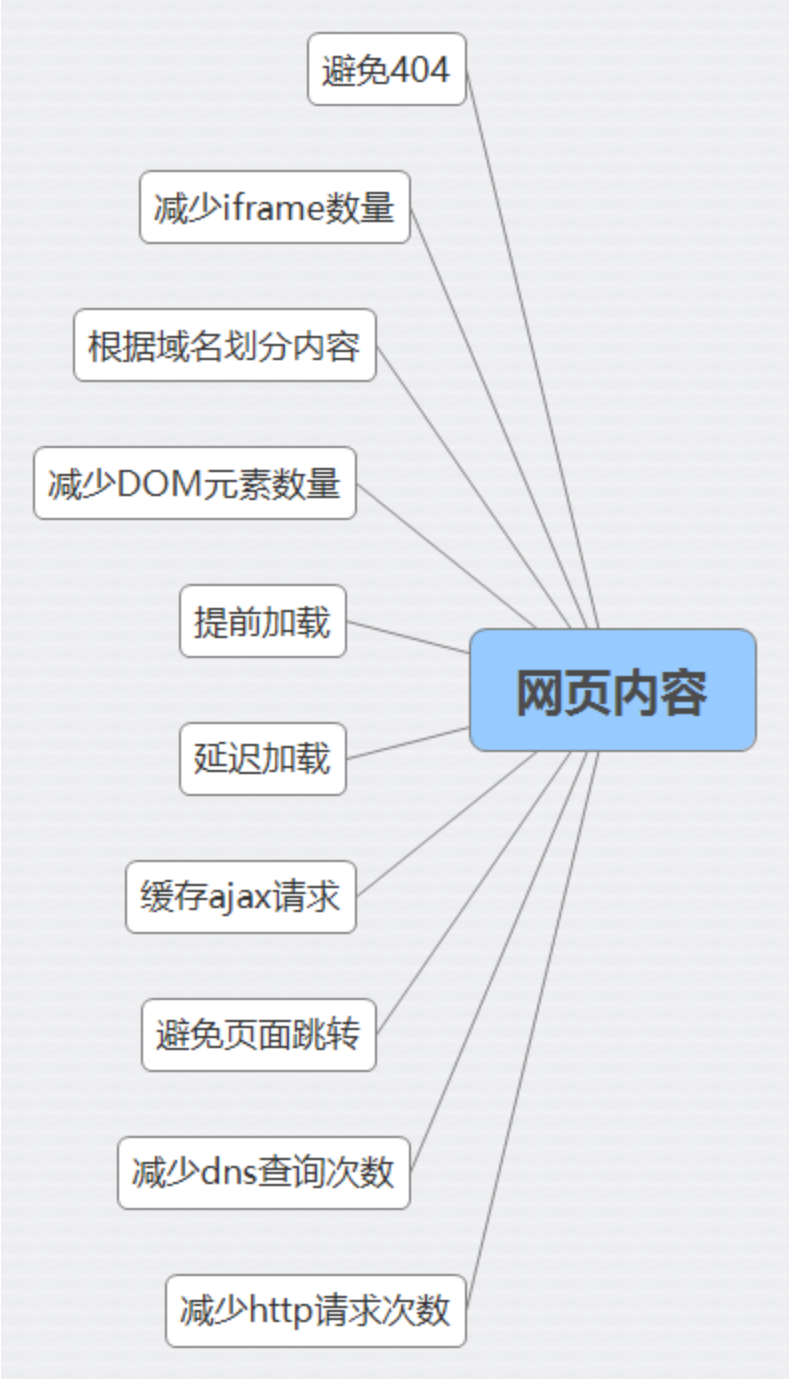
6.6. get请求

6.7. 避免空的图片src

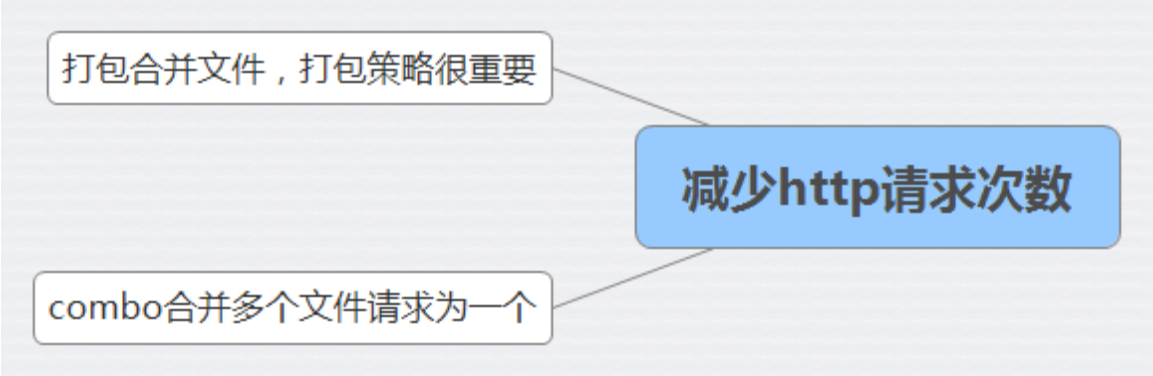


6.7.1. 空的图片src仍然会使浏览器发送请求到服务器，这样完全是浪费时间，而且浪费服务器的资源

7. 网页内容



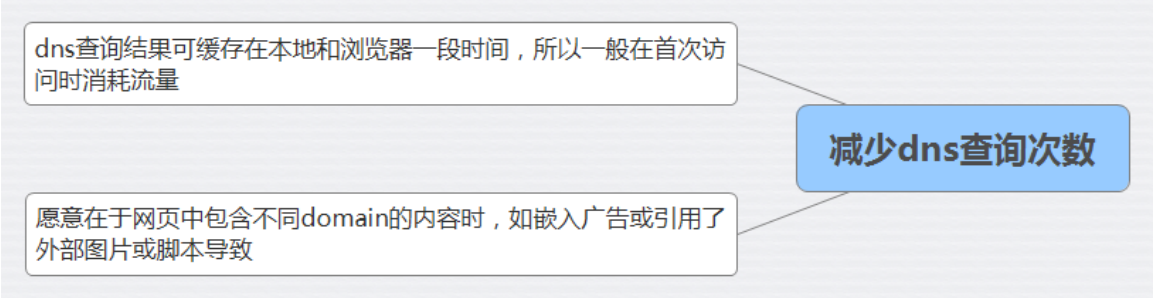
7.1. 减少http请求次数



7.1.1. combo合并多个文件请求为一个

7.1.2. 打包合并文件，打包策略很重要

7.2. 减少dns查询次数



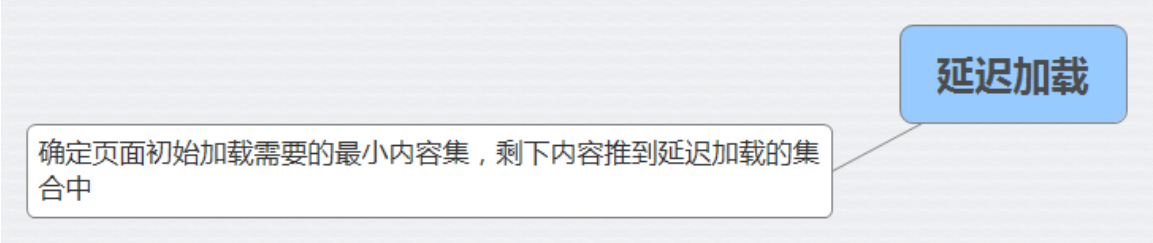
7.2.1. 愿意在于网页中包含不同domain的内容时，如嵌入广告或引用了外部图片或脚本导致

7.2.2. dns查询结果可缓存在本地和浏览器一段时间，所以一般在首次访问时消耗流量

7.3. 避免页面跳转

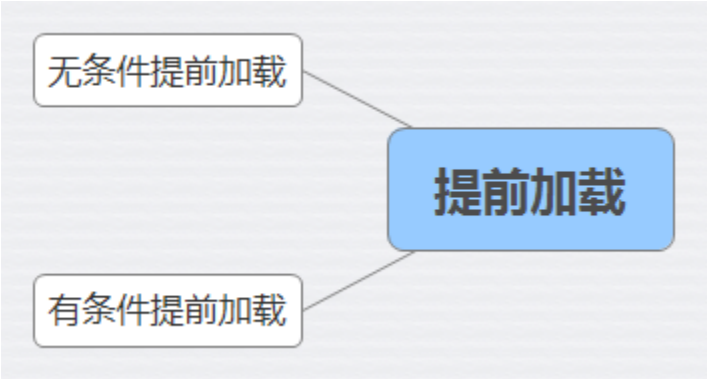
7.4. 缓存ajax请求

7.5. 延迟加载

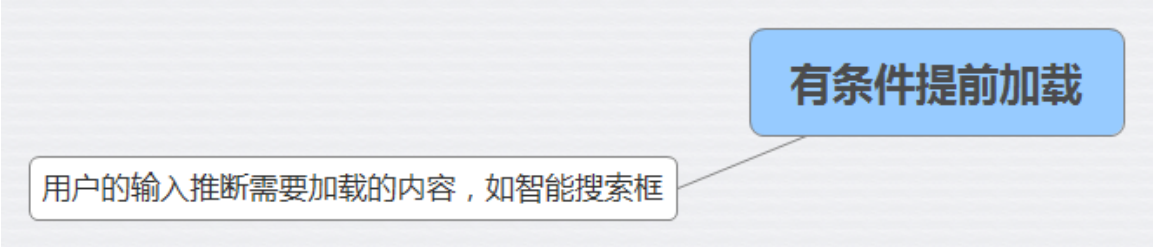


7.5.1. 确定页面初始加载需要的最小内容集，剩下内容推到延迟加载的集合中

7.6. 提前加载

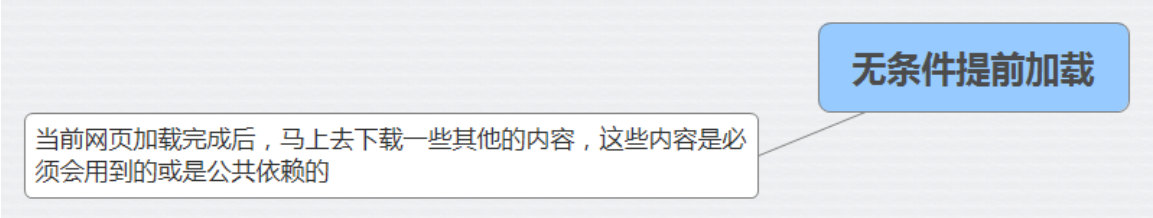


7.6.1. 有条件提前加载



7.6.1.1. 用户的输入推断需要加载的内容，如智能搜索框

7.6.2. 无条件提前加载

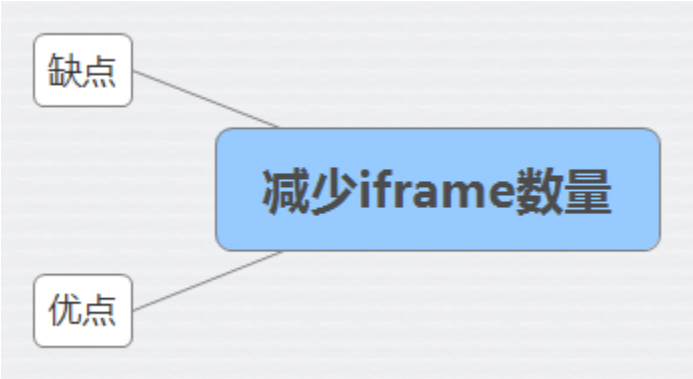


7.6.2.1. 当前网页加载完成后，马上去下载一些其他的内容，这些内容是必须会用到的或是公共依赖的

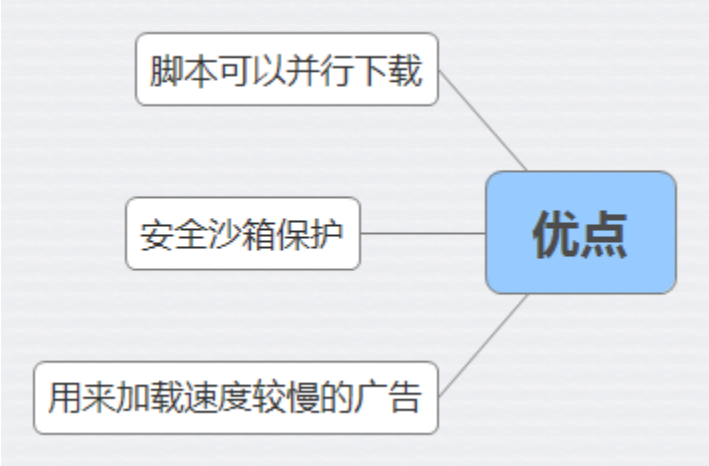
7.7. 减少DOM元素数量

7.8. 根据域名划分内容

7.9. 减少iframe数量



7.9.1. 优点

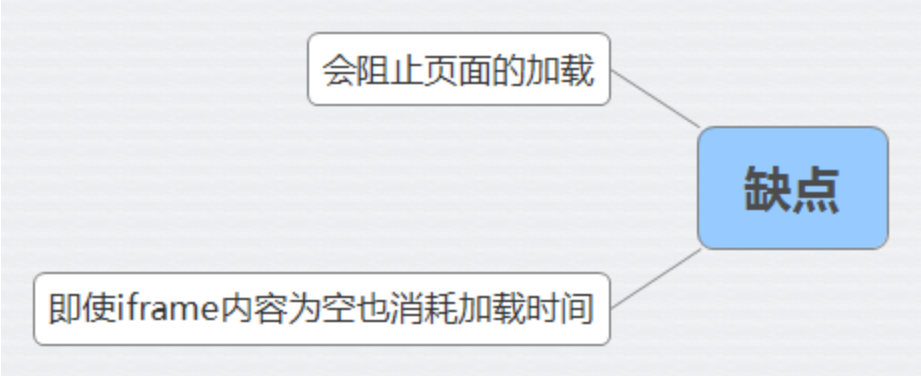


7.9.1.1. 用来加载速度较慢的广告

7.9.1.2. 安全沙箱保护

7.9.1.3. 脚本可以并行下载

7.9.2. 缺点

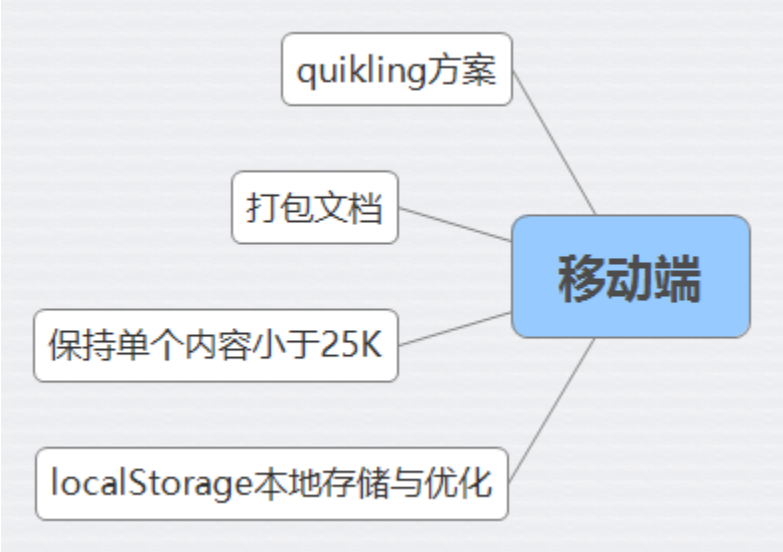


7.9.2.1. 即使iframe内容为空也消耗加载时间

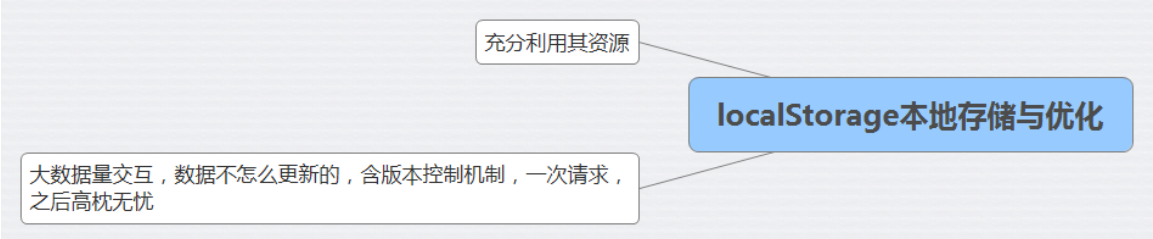
7.9.2.2. 会阻止页面的加载

7.10. 避免404

8. 移动端



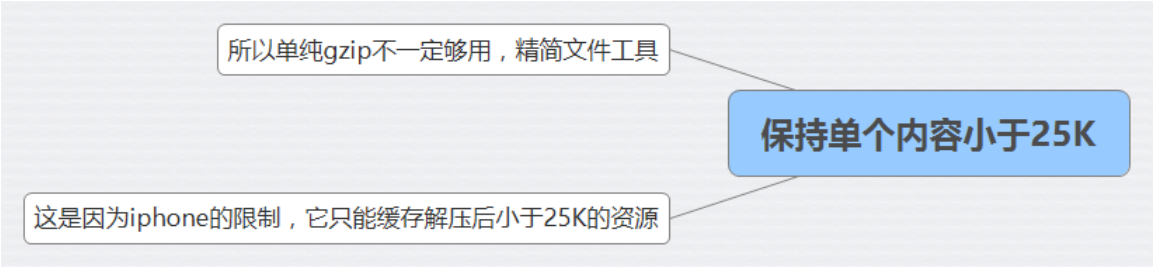
8.1. localStorage本地存储与优化



8.1.1. 大数据量交互，数据不怎么更新的，含版本控制机制，一次请求，之后高枕无忧

8.1.2. 充分利用其资源

8.2. 保持单个内容小于**25K**



8.2.1. 这是因为**iphone**的限制，它只能缓存解压后小于**25K**的资源

8.2.2. 所以单纯**gzip**不一定够用，精简文件工具

8.3. 打包文档

8.4. quikling方案