JS/HTML/CSS优化方案

**减小JS体积**

* Code Spliting 代码拆分，按需加载
* Tree Shaking 代码减重，有一些用不到的代码就不需要打包，减小JS文件体积。

**减少主线程工作量**

* 避免长任务，长任务会占用主线程，导致阻塞时间过久
* 避免超过1kb的行间脚本，浏览器无法对行间脚本进行有效优化，行间脚本越大，解析消耗时间会过长。
* 使用rAF和rIC进行时间调度

**函数优化**

V8引擎会对函数进行一个懒解析（只解析声明），当函数真正被调用的时候，才会解析这个函数体。

我们可以进行eager parsing（饥饿解析），在函数体旁边加一对括号。

**对象优化**

* 以相同顺序初始化对象成员，避免隐藏类的调整。
  + V8引擎会自己推断一个对象的类型，我们称为hidden class（隐藏类型），如果按照同样的顺序初始化对象，这些隐藏类可以被V8引擎复用
* 实例化后避免添加新属性
  + const car1 = {color:’red’} //In-object属性
  + car1.seats = 4 //Normal/Fast属性，存储在property store里，需要通过描述数组间接查找。
* 尽量使用Array代替Array-like对象，因为V8引擎会对Array对象进行优化。 Array-like对象即使转换成数组再遍历，效率也会高。
* 尽量避免数组越界
  + 会造成undefined和数字的比较，转型开销
  + 会沿着原型链查找
* 避免数组元素类型转换
  + 数组允许掺杂不同类型的元素，但是最好同一类型，这样会造成额外开销。元素类型越具体，引擎能做的优化越好。

**HTML优化**

* 减小iframes的使用
  + 会延缓父文档的加载，iframes加载不完成，父文档onLoad事件无法触发
  + 也可以延缓加载，先定义iframe但是不赋予’src’属性，等到onload事件触发之后再使用js赋值’src’属性
* 压缩空白符
  + 空白符也是占空间的
* 避免深层次嵌套，遍历层次越多，消耗越大。
* 避免table布局
  + 造成开销比较大
* 删除注释
* CSS & JS尽量外链（除了一些首屏优化的解决方案），因为会造成文档体积变大。
* 删除元素默认属性

CSS优化

* 降低CSS对渲染的阻塞
  + 只加载必要的CSS，减少首屏渲染时间
  + 后续CSS延迟加载
* 利用GPU完成动画（为动画单独分一个图层）
* Contain:layout 可以进行优化，告诉浏览器这个元素的子元素的变化不会影响其他元素，其他元素不会收到影响，减少计算样式和布局的时间。