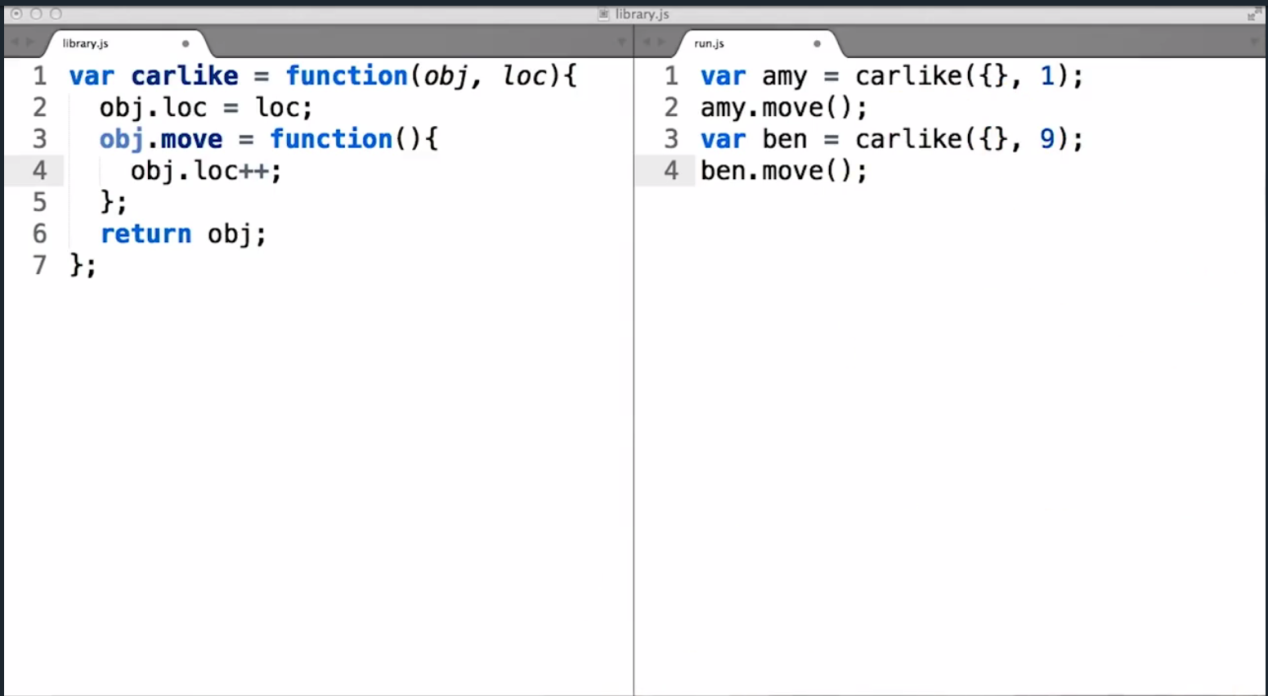
**第一周**

闭包：

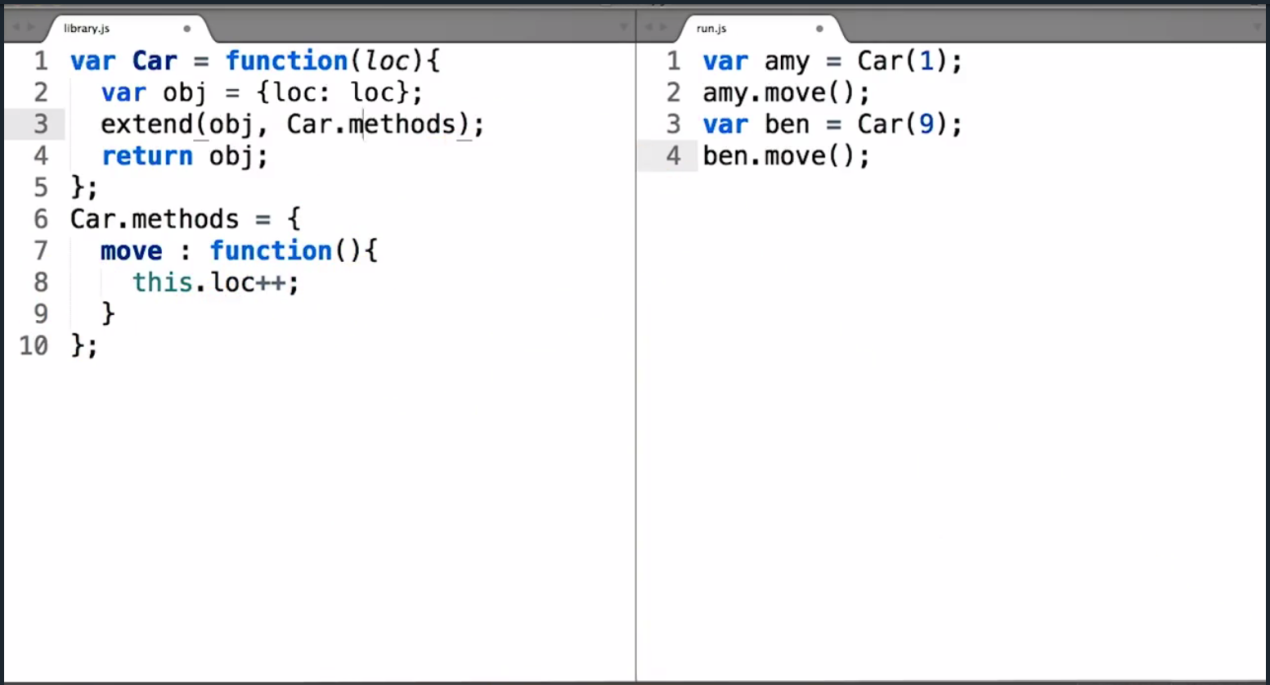
闭包是指一些函数通过某些方式，可以随时被访问

**课程5 对象修饰模式**



每次调用carlike都会生成move的function 占用了内存

**课程6 函数类**



**课程7 原型类**



提高性能的方法

不适合extend 来复制方法 而是使用原型对象来存储所有共享方法，并使实例委托到这个共享原型对象

Car.prototype 原型对象

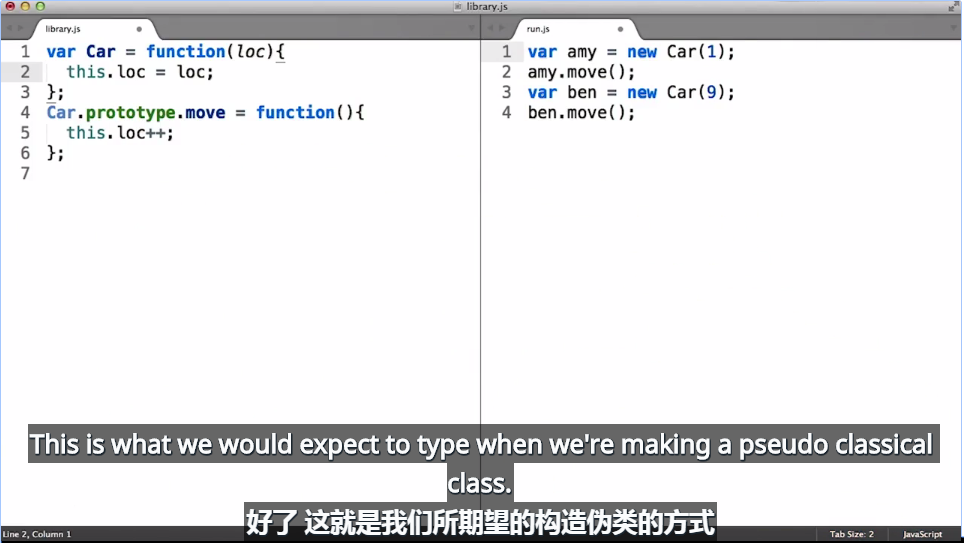
**使用Object.create()实现委托**

Car.prototype.move 与Car.move 没有什么不同，对内存模式没有影响

instanceof 用于检查右侧运算对象的 .prototype对象是否存在左侧运算对象的原型链中

**课程8 伪类模式**

通过new 实例出来实现了伪类方式

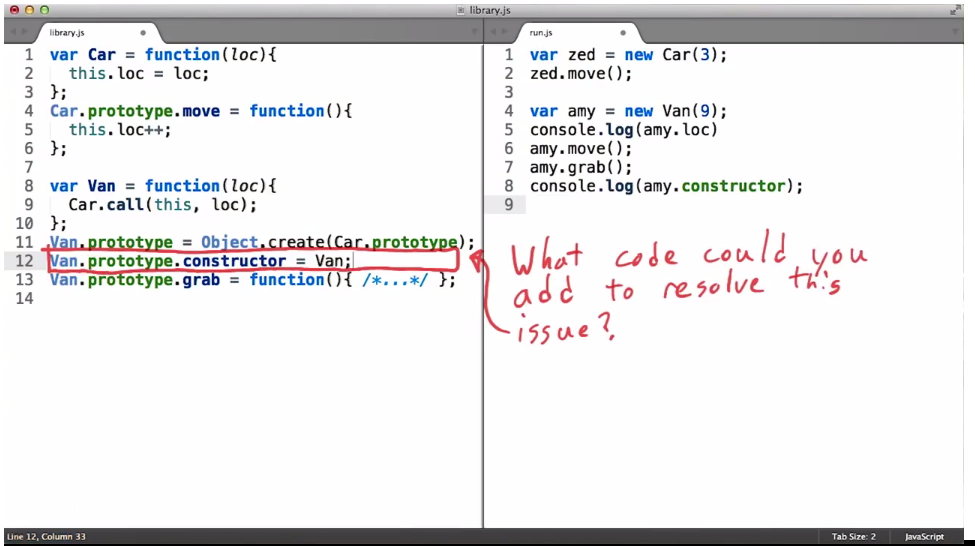


**课程9 超类**



Car是van和cop的父类（超类）

**课程10 伪类子类**



子类原型委托

Van.prototype = Object.create(Car.prototype) 使用Object.create()实现

Van.prototype = new Car() 不要使用new的方式实现委托 这样做的话，每次构造一个Van的子类或者Car的子类都会调用Car这个函数作为整个过程的一部分

子类构造函数委托

Van.prototype.constructor = Van

canvas

创建web服务

1 使用python方法



windows 下载python ,安装时勾选左下角python语言安装选项

打开cmd 命令工具， 进入需要打开服务的文件位置， 使用python –version 查看python 的版本

有版本的情况下，

python 2 输入python –m SimpleHTTPServer 8000 命令

python 3 输入python –m http.server 8000 命令

浏览器打开<http://localhost:8000/>访问文件

2 使用node 方法 npm

package.json

**编写Readme 文档**

开始： 标题和说明

install

1 哪些步骤是必要的 将环境配置好并成功运行需要哪几步

2 需要预先安装哪些配置

3 哪些会是读者难以理解的部分

使用markdown来编写readme

**网站优化**

chrome浏览器扩展PageSpeed-Insights 插件

使用chrome canary

手机安装chrome浏览器，直接usb连接到电脑，中间不能接usb集线器，打开开发者模式，USB调试。

电脑网址输入chrome://inspect 进入Devtol 检测出手机后，手机打开谷歌浏览器，电脑在open输入栏，输入随意网址，手机将会打开输入的网址。电脑设置了代理或者能翻墙，才能使用inspect功能

当浏览器必须重新计算文档中元素的位置和几何结构以便重新渲染它时，我们称之为**回流**。

**关键渲染路径**

render 绘制，呈现

DOM

是什么 文档对象模型（DOM）是HTML和XML文档的编程接口，DOM将文档解析为一个由结点和对象组成的结构集合

DOM tree 构建过程

字节—>字符—>令牌（Tokens）—>节点（Nodes）—>对象模型（DOM树）

如何使用时间轴工具

<https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool>

<http://www.jianshu.com/p/4da0f0bda768>

CSS tree 构建过程

字节—>字符—>令牌（Tokens）—>节点（Nodes）—>对象模型（CSSOM树）

CSS 被视为阻塞渲染的资源

CSSOM 树和 DOM 树合并成渲染树，然后用于计算每个可见元素的布局，并输出给绘制流程，将像素渲染到屏幕上。优化上述每一个步骤对实现最佳渲染性能至关重要

**注意“阻塞渲染”仅是指浏览器是否需要暂停网页的首次渲染**

浏览器完成的步骤：

1.处理 HTML 标记并构建 DOM 树。

2.处理 CSS 标记并构建 CSSOM 树。

3.将 DOM 与 CSSOM 合并成一个渲染树。

4.根据渲染树来布局，以计算每个节点的几何信息。

5.将各个节点绘制到屏幕上。

**优化关键渲染路径就是指最大限度缩短执行上述第 1 步至第 5 步耗费的总时间。**

**关键渲染路径优化**

1 **优化DOM**

资源最小化 <https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#minification-preprocessing--context-specific-optimizations>

通过GZip进行的文本压缩

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#text-compression-with-gzip>

HTTP缓存

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

我们唯一要做的就是确保服务器提供必要的 ETag 令牌。检查您的服务器文档中有无必要的配置标志。

HTML5 Boilerplate 项目包含所有最流行服务器的[配置文件样例](https://github.com/h5bp/server-configs)

**2 CSS 优化**

请务必精简您的 CSS，尽快提供它，并利用媒体类型和查询来解除对渲染的阻塞。

通过媒体类型和媒体查询将一些 CSS 资源标记为不阻塞渲染。

**声明您的样式表资产时，请密切注意媒体类型和查询，因为它们将严重影响关键渲染路径的性能。**

**3 JavaScript 优化**

1 向 script 标记添加异步关键字**async**可以指示浏览器在等待脚本可用期间不阻止 DOM 构建，这样可以显著提升性能。

注意 内联js脚本使用async没有作用

2 向脚本标记添加 **defer** 属性，告诉解析程序脚本应该等到文档加载后执行

脚本标记也可以采用 defer 属性，方法与采用 async 属性相同。差别在于对于 defer，脚本需要等到文档解析后执行，而 async 允许脚本在文档解析时位于后台运行

3 使用window.onload 不是简单办法 上面两点是简单的方法

在提高网页加载速度上您可以采取的最有效措施就是，通过优化和压缩其余资源来最大限度减小总下载大小

高性能浏览器联网著作 <https://hpbn.co/>

关键渲染路径 Critical Rendering Path CRP

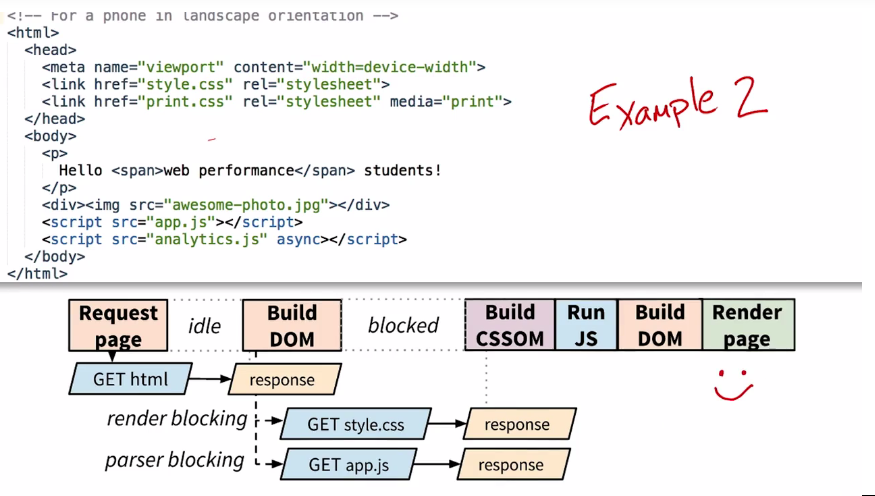
* **关键资源：** 可能阻止网页首次渲染的资源。
* **关键路径长度：** 获取所有关键资源所需的往返次数或总时间。
* **关键字节：** 实现网页首次渲染所需的总字节数，它是所有关键资源传送文件大小的总和。我们包含单个 HTML 页面的第一个示例包含一项关键资源（HTML 文档）；关键路径长度也与 1 次网络往返相等（假设文件较小），而总关键字节数正好是 HTML 文档本身的传送大小。

**CRP优化**

**CSS压缩** [**https://developers.google.com/speed/pagespeed/service/MinifyCSS**](https://developers.google.com/speed/pagespeed/service/MinifyCSS)

**内联和优化“关键CSS”** [**https://developers.google.com/speed/pagespeed/service/PrioritizeCriticalCss**](https://developers.google.com/speed/pagespeed/service/PrioritizeCriticalCss)

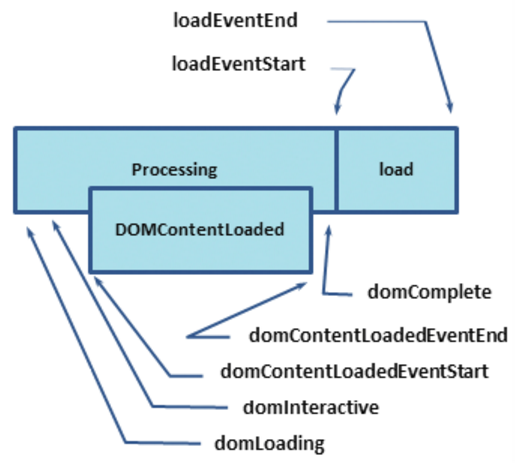
**CRP图表例子**

****

**通过 Navigation Timing 进行检测**

[**https://developers.google.com/web/fundamentals/performance/critical-rendering-path/measure-crp**](https://developers.google.com/web/fundamentals/performance/critical-rendering-path/measure-crp)

**dcl domContentLoaded 一般表示 DOM 和 CSSOM 均准备就绪的时间点。**

****

三大类优化

1 减小数据量

缩小 压缩 和缓存 HTML CSS javascript

2 减少关键资源的数量

使用媒体查询来取消阻止呈现

内联CSS

推迟javascript执行过程

在script中使用async属性

3 缩短关键呈现路径的长度

判断CSS将触发的工作 构建性能良好的网站需要熟悉该网站

<https://csstriggers.com/>

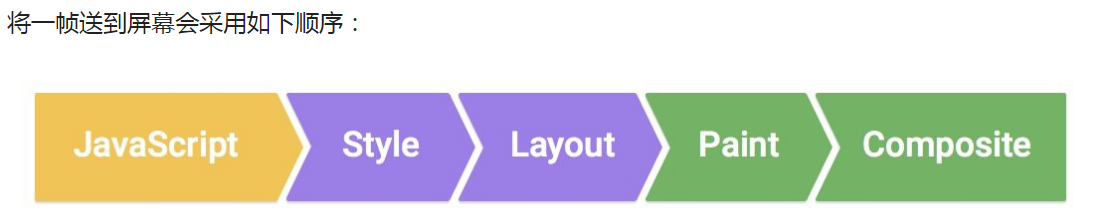
CSS应该尽可能触发最少量的工作 也就是**避免绘制和布局**

不阻塞渲染地加载CSS的方法

**管道**

帧顺序

javaScript >>> Style >>> Layout （布局）>>> Paint（绘制） >>> Composite（图层合成）



合成是指浏览器将网页的单个图层合拼到一起

通过js或者css Animas or web anima API对网页做出外观改变

对于给定的帧，管道通常会经历三种形式

1 通过css或js做出外观更改

浏览器必须重新计算收到影响的元素样式

如果改变了布局属性 例如几何结构、宽度、高度、相对另一个元素的位置

受影响的区域将需要重新绘制

最后绘制的元素将需要合成在一起

下面每一步都需要

javaScript >>> Style >>> Layout >>> Paint >>> Composite

2 只更改了绘制属性

例如背景图片，文本颜色或阴影

不需要进行布局，因为没有更改任何元素的几何结构

是下面的步骤

javaScript >>> Style>>> Paint >>> Composite

3 涉及的更改不需要布局或绘制 只需要合成

javaScript >>> Style>>> Composite

**App生命周期**

浏览器如何通过HTML CSS 和JS来渲染像素理解这一流程是哟花应用性能的关键

RAIL

R response 响应 A animation 动画

I idle 闲置 应用加载后通常处于闲置状态等待用户采取操作这个时候应该抓住机会处理那些为了满足1秒钟加载目标而推迟的工作 通常这些闲置时段的时长约为50毫秒

L load 加载 针对关键路径做出优化

时间顺序是 L I A R

初始化加载操作应该在1秒钟内完成

R response 响应 100ms

A animation 动画 16ms（60帧每秒） 实际需要10-12ms就要完成因为浏览器处理也需要实际

I idle 闲置 50ms

L load 加载 1 second

FLIP 策略

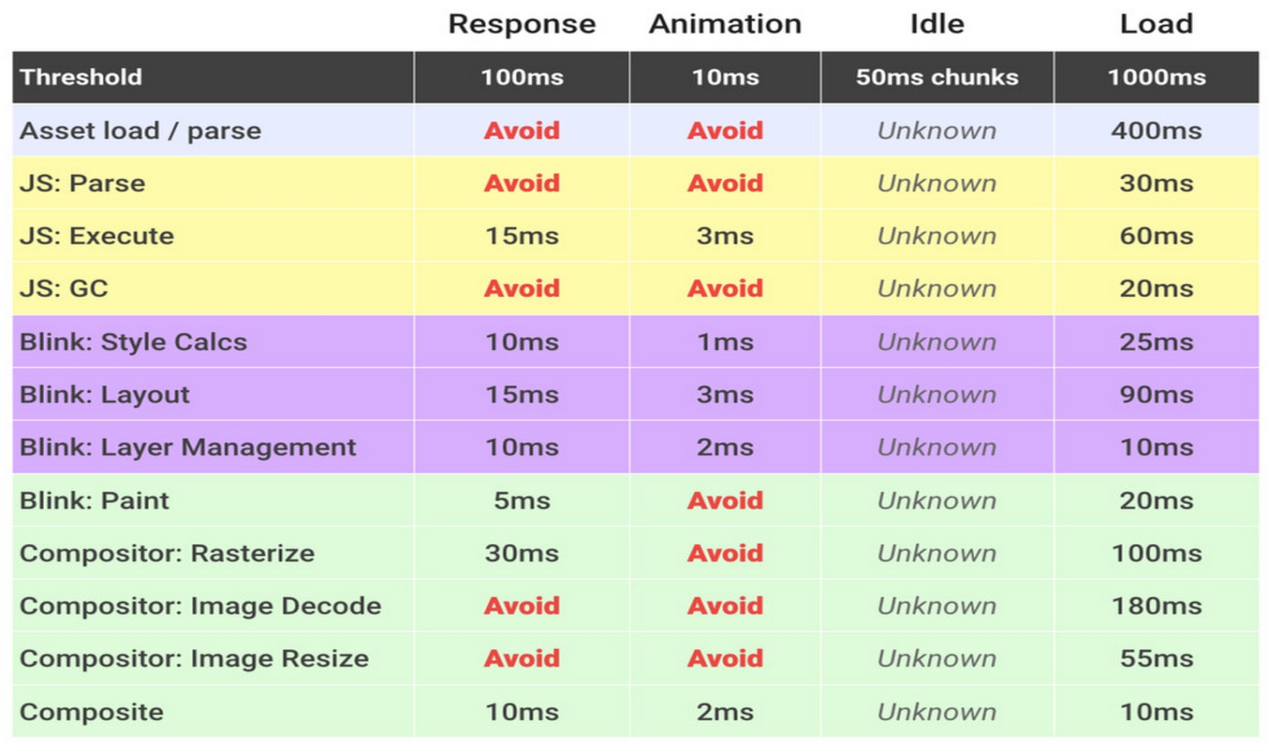
F first 初始位置 L last 结束位置 I倒转 invert P play 播放

找到卡片的起始点 然后找到卡片展开后的结束点 衡量了所以元素展开前后的位置这样就知道所有元素在网页上需要移动多远

那些互动或者动画需要达到60帧/每秒

任何涉及动作或屏幕手指操作的互动都需要达到

两个除外 切换表格控件以及应用主题的更改有100ms的响应，但之后必须继续达到60帧每秒

不同任务的可用时间表 任务时间预算

**JS优化**

任何对页面做出外观改变的操作都应该发生在requestAnimationFrame内

对动画使用requestAnimationFrame API 而不是setTimeout或setInterval

只有IE9 不支持 ，使用它的polyfill即可

运行时间长的JS使用web Workers

<https://developer.mozilla.org/zh-CN/docs/Web/API/Web_Workers_API/Using_web_workers>

<https://www.html5rocks.com/zh/tutorials/workers/basics/>

web Workers线程实现

var myWorker = new Worker(url);

url 需要遵守同源策略 即相对index.html路径的位置

需要开启本地服务用来访问worker.js文件

**Worker 无法使用**：

DOM（非线程安全）

window 对象

document 对象

parent 对象

**子 Worker**

Worker 可以生成子 Worker。这对于在运行时进一步拆分大任务来说非常重要。但是，子 Worker 还有几点注意事项：

子 Worker 必须托管在与父网页相同的来源中。

子 Worker 中的 URI 应相对于父 Worker 的位置进行解析（与主网页不同）。

**哪种应用可以利用 Web Worker：**

预先抓取和/或缓存数据以便稍后使用

突出显示代码语法或其他实时文本格式

拼写检查程序

分析视频或音频数据

背景 I/O 或网络服务轮询

处理较大数组或超大 JSON 响应

<canvas> 中的图片过滤

更新本地网络数据库中的多行内容

**JS内存管理**

在timeline中

勾选设置中的memory 查看内存分析器

取消勾选memory， ctr+f 搜索GC查看垃圾回收

如果因为垃圾回收导致丢失帧需要手动清除垃圾

编写快速且节省内存的js

<https://www.smashingmagazine.com/2012/11/writing-fast-memory-efficient-javascript/>

MDN的内存管理

<https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Memory_Management>

高性能垃圾收集器友好代码

<http://buildnewgames.com/garbage-collector-friendly-code/>

比较不同的写法运行速度

<https://jsperf.com/>

**样式和布局**

BEM

Block 块

Element 元素

Modifier 修饰

使用forEach 方法

function getDomNodeArray(selector) {

// get the elements as a DOM collection

var elemCollection = document.querySelectorAll(selector);

// coerce the DOM collection into an array

var elemArray = Array.prototype.slice.apply(elemCollection);

return elemArray;

};

FSL forced synchronous Layout 强制同步布局

要避免FSL

FSL的原因 <https://gist.github.com/paulirish/5d52fb081b3570c81e3a>

如何（不）在WebKit中触发布局

<http://gent.ilcore.com/2011/03/how-not-to-trigger-layout-in-webkit.html>

chrome和Firefox 使用will-change CSS属性告诉浏览器讲出现外观改变，浏览器然后会将元素创建在新的合成图层上

好处是 让浏览器不必匆忙地创建并绘制新的图层

旧版浏览器和Safari 使用transform: translateZ(0)

will-change: transform

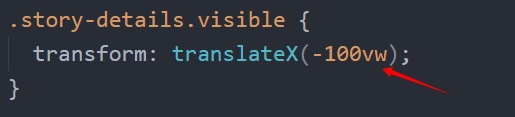
告诉浏览器打算在某个时间点更改元素transform属性 也可以使用left top width height或其他外观属性

will-change看上去挺诱人，但是会消耗大量内存并花费很多合成时间最终可能造成的问题比解决的还要多

关键在于尝试在图层管理中合成时间和管道的其他部分的时间之间找到平衡点

performance 勾选enable ------ paint选项

点击frames 出现layers查看图层



vw viewport width

AJAX

异步的js请求

ajax请求允许在不重新加载网页的情况下检索和显示内容 异步是指请求不会阻止其他事件的发生， 仅在服务器返回数据或对数据采取操作

ajax请求返回的数据不尽相同

以前的是XML <entry></entry> 主要的层次数据格式

JSON {properry:data} 更加流行

HTML <div></div>

很多ajax请求都是JEDGE请求 表示异步js和JSON请求 叫法上还是叫ajax请求

加快初次渲染的速度

**常规 HTML**：是指可用于任何用户的 HTML 和 CSS。例如，所有用户都会需要“最喜爱的谷物”、“最喜爱的意大利面食”的 HTML，以及描述如何设置用户图片样式的 CSS。

**唯一 HTML**：是指特定于某个用户的 HTML 内容。例如，“脆谷乐”、“通心粉”、“早餐”和我的图片就是唯一 HTML。

**jsonp**

data 可以放在url上

也可以放在ajax的data上

jsonp 没有错误处理流程

错误处理方法

自定义setTimeout 的方法 请求成功返回响应时就清除timeout

**JS设计模式**

MOV简介

M model 所有数据储存在模型中 代表数据

包括来自服务器和来自用户的数据 Collections models的集合

O octopus 章鱼 将model 和view 连接起来 称为controller 或 View Model

V view 用户看到以及与其互动的所有事物 包括routers 路由 路由跟踪URL的状态在某种程度上像视图一样

包括DOM元素、输入元素、按钮和图片

基本是用户与应用之间的接口

用于提供应用数据和读取数据

MV C / VM / P /\* 等模式

后面的都是O

对于MVO 将所有的明智功能，过滤功能都放O上

MVC 模型视图控制器 章鱼‘控制器’

MVVM 模型视图 章鱼‘视图模型’

MVP 模型视图演示器 章鱼‘演示器’

console.dir(object) 是可以在现代 Web 浏览器中使用的函数，用于显示指定 JavaScript 对象的属性列表

**探索新的代码库建议**

1 运行应用，看看如何运作

2 模块加载器其实就是以正确顺序加载其他库的一个库

3 看看加载的js文件有哪些

4 所有这些库的实际用途是什么

IIFE（立即调用的函数表达式）

修改代码

1 是否应该改变数据存储方式，还是只是外观的改变

2 这些变化可以在哪里实现

3 是否有类似的功能

git commit 格式 <https://gdgdocs.org/document/d/1HZ9Bo1mDKhe3JZzmFvekL5P2WHafpCaEXTymj__FUYw/pub?embedded=true>

类型：主题

消息正文

注释

**类型**

类型位于在标题内，有以下几种可能：

* **feat：** 新功能
* **fix：**错误修复
* **docs：**文档修改
* **style：**格式、分号缺失等，代码无变动
* **refactor：**生产代码重构
* **test：**测试添加、测试重构等，生产代码无变动
* **chore：**构建任务更新、程序包管理器配置等，生产代码无变动。