内存图与 JS 世界

从入门到工作：JS 全解

版权声明

联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

操作系统常识

先来补充一些计算机常识

英语小课堂

英语	翻译	英语	翻译
Operating System	操作系统，简称OS	kernel	内核
runtime	运行时（需要的东西）	compile	编译
environment	环境，简称 env	memory	记忆、存储
person	一个人	people	一群人

一切都运行在内存里

- 开机 通电» 读固件» 加载开机程序» 开机程序被加载到内存里
  - ✓ 操作系统在 C 盘里（macOS 的在根目录下多个目录里）
  - ✓ 当你按下开机键，主板通电，开始读取固件
  - ✓ 固件就是固定在主板上的存储设备，里面有开机程序
  - ✓ 开机程序会将文件里的操作系统加载到内存中运行
- 操作系统（以 Linux 为例）
  - ✓ 首先加载操作系统内核 可以管理文件，可以操作显示器等
  - ✓ 然后启动初始化进程，编号为1，每个进程都有编号
  - ✓ 启动系统服务：文件、安全、联网
  - ✓ 等待用户登录：输入密码登录 / ssh 登录
  - ✓ 登录后，运行 shell，用户就可以和操作系统对话了
  - ✓ bash 是一种 shell，图形化界面可认为是一种 shell

打开浏览器

- chrome.exe
  - ✓ 你双击 Chrome 图标，就会运行 chrome.exe 文件
  - ✓ 开启 Chrome 进程，作为主进程
  - ✓ 主进程会开启一些辅助进程，如网络服务、GPU 加速
  - ✓ 你每新建一个网页，就有可能会开启一个子进程
- 浏览器的功能
  - ✓ 发起请求，下载 HTML，解析 HTML，下载 CSS，解析 CSS，渲染界面，下载 JS，解析 JS，执行 JS 等
  - ✓ 功能模块：用户界面、渲染引擎、JS 引擎、存储等
  - ✓ 上面功能模块一般各处于不同的线程（比进程更小）
  - ✓ 如果进程是车间，那么线程就是车间里的流水线

渲染：把HTML解析出来，把CSS解析出来，然后把它们合起来，显示在屏幕上。

JS 引擎

- JS 引擎举例
  - Chrome 用的是 V8 引擎，C++ 编写
  - 网景用的是 SpiderMonkey，后被 Firefox 使用，C++
  - Safari 用的是 JavaScriptCore
  - IE 用的是 Chakra (JScript9)
  - Edge 用的是 Chakra (JavaScript)
  - Node.js 用的是 V8 引擎
- 主要功能
  - 编译：把 JS 代码翻译为机器能执行的字节码或机器码
  - 优化：改写代码，使其更高效
  - 执行：执行上面的字节码或者机器码
  - 垃圾回收：把 JS 用完的内存回收，方便之后再次使用

执行 JS 代码

- 准备工作
  - 提供 API：window / document / setTimeout
  - 没错，上面这些东西都不是 JS 自身具备的功能
  - 我们将这些功能称为运行环境 runtime env
  - 一旦把 JS 放进页面，就开始执行 JS
- 等一下
  - JS 代码在哪里运行?
  - 答：内存
  - 哪里的内存?
  - 答：看下一张 PPT

内存图

为了快速补充你的科班知识，我自己发明的图示

瓜分内存

一颗一个G

OS

Chrome

辅助进程

Tab 1

Tab 2

Tab 3

Tab 4

渲染线程

用户界面

JS引擎

代码区

```
var a = 1;
var b = 2;
...
console.log(..)
```

放的什么区

a

b

c

Stack

Heap

连续存储

连接存储

红色区域

- 作用
  - 红色专门用来存放数据，我们目前只研究该区域
  - 红色区域并不存变量名，变量名在「不知什么区」
  - 每种浏览器的分配规则并不一样
  - 上图的区域并不完整
  - 我还没有画「调用栈」、「任务队列」等区域
- Stack 和 Heap
  - 红色区域分为 Stack 栈和 Heap 堆
  - 要解释为什么叫栈和堆需要数据结构知识，后面再讲
  - Stack 区特点：每个数据顺序存放
  - Heap 区特点：每个数据随机存放

Stach 和 Heap 举例

- 代码

```
var a = 1
var b = a
var person = {name:'frank', child: {name: 'jack'}}
var person2 = person
```

- 规律

✓ 数据分两种：非对象和对象
✓ 非对象都存在 Stack 只有数字，字符串，布尔
✓ 对象都存在 Heap 数组是对象，函数是对象
✓ = 号总是会把右边的东西复制到左边（不存在什么传值和传址）

区分值和地址

不会画内存图的人才需要做这件事

对象被篡改了

- 代码

```
var person = {name:'frank'}
var person2 = person
person2.name = 'ryan'
console.log(person.name) // 'ryan'
```

盘古开天辟地之前

（JS 的）世界是怎样的

神说要有光，就有了光

神挺厉害呀

JS 开发者说要有 window

就有了 window（浏览器提供）

还要什么

- 要有 console
  - 于是就有了 console，并且挂到 window 上
- 要有 document
  - 于是就有了 document，并且挂到 window 上
- 要有对象
  - 于是就有了 Object，并且挂到 window 上
  - var person = { } 等价于 var person = new Object()
- 要有数组（一种特殊的对象）
  - 于是就有了 Array，并且挂到 window 上
  - var a = [1,2,3] 等价于 var a = new Array(1,2,3)
- 要有函数（一种特殊的对象）
  - 于是就有了 Function，并且挂到 window 上
  - function f(){} 等价于 var f = new Function()

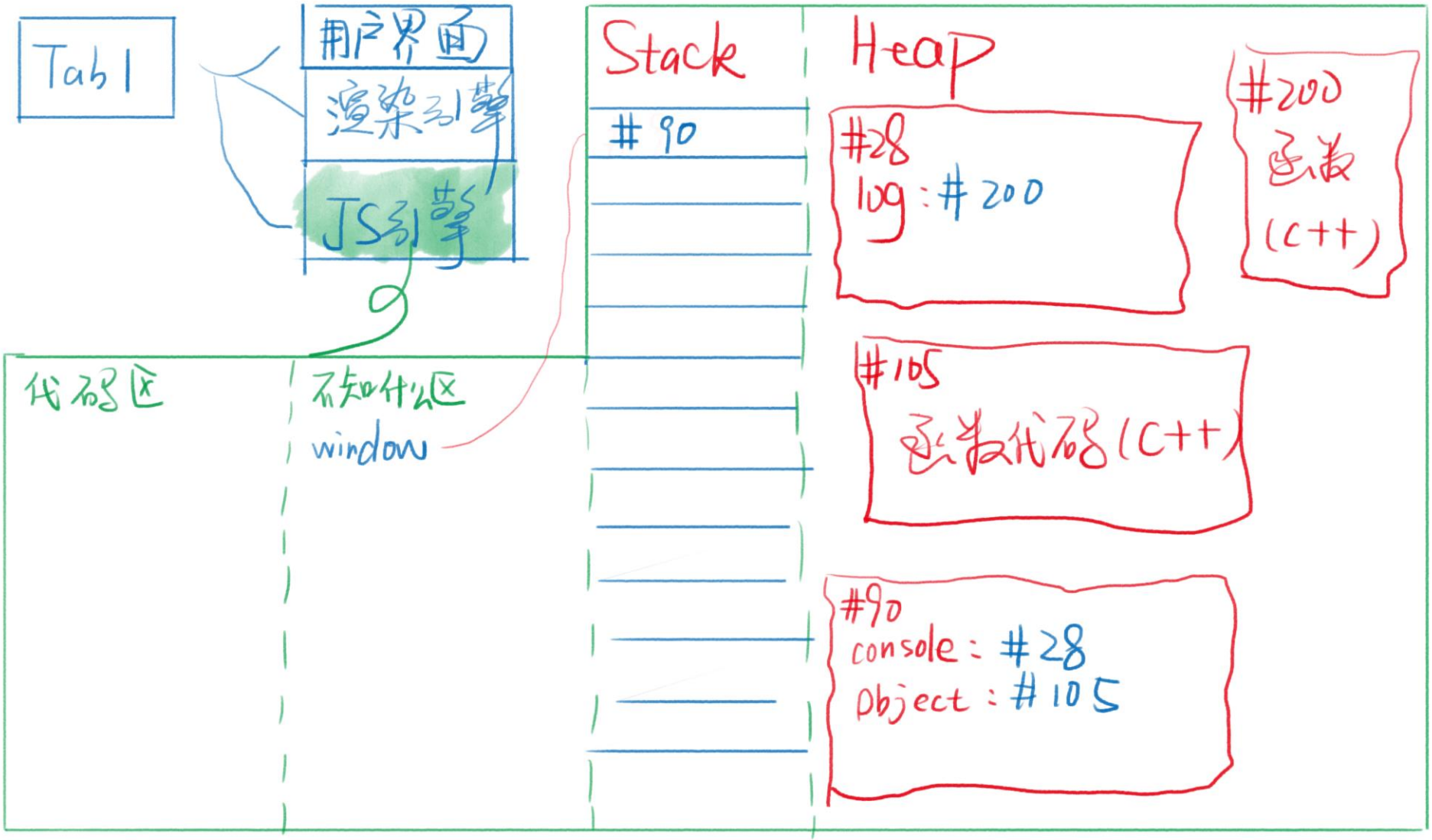
题外话

- 同学提问
  - 为什么有 var a = [ ]，还要提供 var a = new Array() 呢
  - 答：因为后者是正规写法，但是没人用。前者不正规，但是好用啊。
- 同学提问
  - 为什么有 function f(){}，还要提供 var f = new Function 写法呢?
  - 答：原因同上

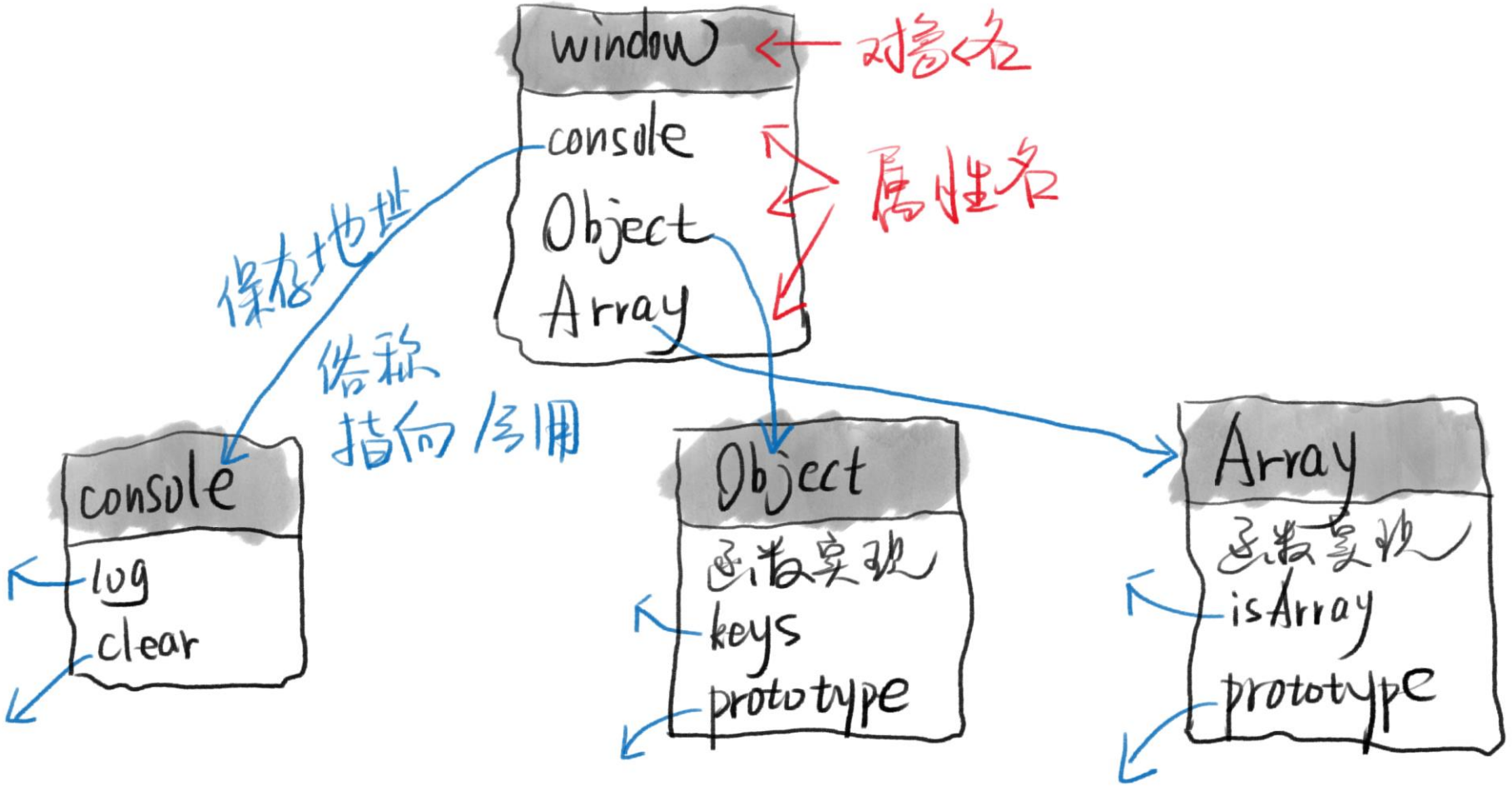
怎么什么都挂在 window 上

因为方便，挂在 window 上的东西可以在任何地方直接用

把 window 用内存图画出来



更简单的画法


window
对象名
console
Object
Array
属性名
保存地址
给称
指向引用
console
log
clear
Object
函数实现
keys
prototype
Array
函数实现
isArray
prototype

细节

- 关于 window
  - window 变量和 window 对象是两个东西
  - window 变量是一个容器，存放 window 对象的地址
  - window 对象是 Heap 里的一坨数据
  - 不信的话，可以让 var x = window，那么这个 x 就指向了 window 对象，window 变量可以去死了
  - 但是这样的代码会弄晕新手，所以不要这样写
- 同理
  - console 和 console 对象不是同一个东西
  - Object 和 Object 函数对象不是同一个东西
  - 前者是内存地址，后者是一坨内存

原型链

图里的 prototype 是干什么用的

打印出来看看

console.dir(window.Object.prototype)

当然，window. 可以省略

看起来就是一些无用的函数

毫无用处呀

问大家一个问题

- 代码

```
var obj = {}
obj.toString()
```

- ✓ 为什么不报错？为什么可以运行？

- 画图给你解释

- ✓ obj 有一个隐藏属性
- ✓ 隐藏属性存储了 Object.prototype 对象的地址
- ✓ obj.toString() 发现 obj 上没有 toString
- ✓ 就去隐藏属性对应的对象里面找
- ✓ 于是就找到了 Object.prototype.toString

再问大家一个问题

- 代码

```
var obj2 = {}
obj2.toString()
```

- ✓ obj 和 obj2 有什么联系

- 相同点
- ✓ 都可以调用 .toString()

- 不同点
- ✓ 地址不同 obj !== obj2
- ✓ 可以拥有不同的属性，反过来说是什么意思呢?

XXX.prototype 存储了 XXX 对象的共同属性

这就是原型

这有什么好处

如果没有原型会怎样

如果没有原型

- 声明一个对象

```
var obj = {
  toString: window.Object.prototype.toString,
  hasOwnPropertyOf: window.Object.......
}
obj.toString()
var obj2 = {
  toString: window.Object.prototype.toString,
  hasOwnPropertyOf: window.Object.......
}
obj2.toString()
```

- 你是不是想累死自己
  - ✓ 好让我帮你还你的蚂蚁花呗?

原型让你无需重复声明共有属性

省代码，省内存

每个对象都有一个隐藏属性

指向原型（对象）

如果没有这个隐藏属性

那 obj 特么怎么知道自己的共有属性在哪

这个隐藏属性叫什么

__proto__

prototype 和 __proto__ 区别是什么

都存着原型的地址

只不过 prototype 挂在函数上

__proto__ 挂在每个新生成的对象上

问大家一个问题

- 代码

```
var arr = [1,2,3]
arr.join('-')
```

  - ✓ 为什么不报错？为什么可以运行？

- 画图给你解释

  - ✓ arr 有一个隐藏属性
  - ✓ 隐藏属性存储了 Array.prototype 对象的地址
  - ✓ arr.join() 发现 arr 上没有 join
  - ✓ 就去隐藏属性对应的对象里面找
  - ✓ 于是就找到了 Array.prototype.join

简洁的 JS 世界

太美了，我们重新画一遍吧

参考文章

- 进程与线程的一个简单解释
- 浏览器进程？线程？傻傻分不清楚！
- 浏览器的工作原理
- How JavaScript works: inside the V8 engine
- JS 中 __proto__ 和 prototype 存在的意义是什么？