webGPU docs

参考地址

介绍

更多背景信息

Web 浏览器中的沙盒 GPU 进程

1. webGPU设计限制： 必须在使用 GPU 进程架构的浏览器中可实现且高效
2. - GPU 驱动程序需要访问额外的内核系统调用
   - 许多 GPU 驱动程序容易挂起或崩溃
     为了提高稳定性和沙箱 =》浏览器使用一个包含 GPU 驱动程序的特殊进程 + 通过异步 IPC 与浏览器的其余部分进行对话
3. WebGPU 的大部分验证规则都是确保使用安全 -》 所有验证都需要在 GPU 过程中进行
4. 所有 GPU 驱动程序对象仅存在于 GPU 进程中，WebGPU对象使用的 CPU 和 GPU 内存在内容处理中不一定是已知的

GPU 和 GPU 进程的内存可见性

1. 原生 GPU API
   - 离散GPU和集成GPU
   - 大多数 GPU 内存分配对 CPU 不可见
   - 为了让 CPU 看到 GPU 缓冲区的内容，它必须被“映射” =》 GPU 进程和内容进程之间的共享内存中进行额外的“暂存”分配

JavaScript API

适配器和设备

1. WebGPU“适配器”（GPUAdapter）是一个对象，用于标识系统上的特定 WebGPU 实现
2. WebGPU“设备”（GPUDevice）表示与WebGPU适配器的逻辑连接, 单个网页上的多个组件可以各自拥有自己的 WebGPU 设备, 只要设备丢失或损坏，就可以（内部）释放这些对象(比如纹理等)

3. 与画布对象相关联，并且大多数命令是通过“子”对象发出的

适配器选择和设备初始化

1. navigator.gpu.requestAdapter(options)获取设备，传递options参数，options有问题，可能会返回null
2. adapter.requestDevice(),
    - 请求无效，即它超出了适配器的能力=>拒绝请求
    - 设备创建过程中出现任何其他问题，它将解析为已经丢失的 GPUDevice
      =》 避免额外的可能返回值，简化了应用程序必须处理的不同情况的数量

可选功能

1. 每个适配器可能具有不同的可选功能，adapter.features, 实现了哪些扩展功能, adapter.limits, 创建设备时可以请求的最大可能功能,根据设备的功能进行严格的验证。不同设备上的默认值可能是 不一样的，每个适配器上公开的一组可选功能由用户代理决定。

```
const adapter = await navigator.gpu.requestAdapter()
console.log(adapter)
for(i in adapter.limits)
   console.log(adapter.limits[i])
console.log(adapter.features)
adapter.features.forEach(item => console.log(item))


//device 中也包含limits、features， 并且可以在获取device时添加、修改对应属性
const device = await adapter.requestDevice()

const device = await adapter.requestDevice({
    requiredFeatures: ['texture-compression-astc'],
    requiredLimits: {
        maxStorageBufferBindingSize: adapter.limits.maxStorageBufferBindingSize
    }
})
```

对象有效性和破坏性

WebGPU’s Error Monad

1. 安全属性-》验证命令 => WebGPU 的设计使得 Javascript 调用可以直接转发到 GPU 进程并在那里进行验证
2. 在单帧期间可以创建相互依赖的 WebGPU 对象
3. 调用成功，返回的对象是有效的；调用发生错误，调用返回的对象也是无效的， 该对象在 GPU 进程端被标记。传染性

Mental Models

1. 每个 WebGPU 对象实际上在内部都是一个 Promise
2. GPU 进程上的每个命令的验证过程中，都会检查 isValid，如果验证失败，则返回一个新的无效对象。 在内容处理方面，GPUFoo 实现不知道对象是否有效。

WebGPU 对象的早期销毁

1. JavaScript 垃圾收集器 (GC) 在渲染器进程中，不知道 GPU 进程中的内存使用情况。
2. WebGPU 对那些可以保留任意内存量的对象类型有一个 .destroy() 方法， 它表示应用程序不再需要对象的内容并且可以尽快释放它

错误

Error Scopes

1. 在 WebGPU 中，每个设备1 维护一个持久的“错误范围”堆栈状态，最初，设备的错误范围堆栈是空的。
   - GPUDevice.pushErrorScope('validation') 或 GPUDevice.pushErrorScope('out-of-memory')
   - 被监测代码
   - GPUDevice.popErrorScope() 结束一个错误范围，将它从堆栈中弹出并返回一个 Promise<GPUError?>， 如果没有捕获到错误，则返回null
2. 来自操作的任何设备时间线错误都会在其发出时传递到堆栈的最顶层错误范围。
   - 如果错误范围捕获错误，则错误不会向下传递到堆栈。 每个错误范围只存储它捕获的第一个错误； 它捕获的任何其他错误都被默默地忽略。
   - 如果不是，则错误将沿堆栈向下传递到封闭的错误范围。
   - 如果错误到达堆栈底部，它可能2 在 GPUDevice3 上触发 uncapturederror 事件（并且可能发出控制台警告）
3. 错误范围状态实际上是 per-device, per-realm
4. 可能不会总是针对给定的错误触发事件做出处理，比如同类型的错误多次、频繁触发，防止影响性能

- 内核： 内核是操作系统的核心部分，它管理着系统的各种资源，比如CPU、内存、网络通讯等。内核可以看成连接应用程序和硬件的一座桥梁，是直接运行在硬件上的最基础的软件实体，在一些简单的硬件设备上可以没有内核或操作系统而直接运行程序，比方单片机等。这些设备通常只是用于特定的场合，也通常功能比较单一。如果我们需要强大的功能，就必须要使用内核。锚
- IPC（Inter-Process Communication，进程间通信）。进程间通信是指两个进程的数据之间产生交互
- 随机存取存储器（英语：Random Access Memory，缩写：RAM），也叫主存，是与CPU直接交换数据的内部存储器。它可以随时读写（刷新时除外），而且速度很快，通常作为操作系统或其他

正在运行中的程序的临时数据存储介质

- 只读存储器（Read-Only Memory，ROM）以非破坏性读出方式工作，只能读出无法写入信息。信息一旦写入后就固定下来，即使切断电源，信息也不会丢失，所以又称为固定存储器。
- VRAM： 显卡上的随机存取存储器，是一种双端口内存，允许在同一时间被所有硬件（包括中央处理器、显示芯片等）访问。主要功能是将显卡的视频数据输出到数模转换器中，有效降低绘图显示芯片的工作负担。
- GPUBuffer 表示可用于 GPU 操作的内存块
- GPUCommandBuffer-命令缓冲区是预先记录的GPU command列表，可以提交给GPUQueue执行。每个GPU command代表一个要在GPU上执行的任务，例如设置状态、绘图、复制资源等。
- User Agent中文名为用户代理，简称 UA，它是一个特殊字符串头，使得服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。