画一张图

1. **html是什么**

HTML（HyperText Markup Language）是一种用于创建和展示万维网（World Wide Web）上的网页的标准标记语言。它由一系列的元素组成，这些元素通过标签（如 <html>、<title>、<body>、<a> 等）来定义，并且可以嵌套使用。HTML 文档通常以 .html 或 .htm 扩展名保存。

HTML 的主要作用是：

* 定义网页的结构。
* 包含文本内容，如标题、段落、列表等。
* 嵌入图片、音频、视频等多媒体内容。
* 创建超链接，允许用户导航到其他网页或网站。
* 表单元素，用于收集用户输入。

HTML 的版本随着时间的推移而发展。**目前最新的稳定版本是 HTML5**，它提供了更多的功能，如对多媒体的原生支持、新的语义元素等。

HTML 文档由浏览器解析，浏览器读取这些标签并据此渲染页面，展示给用户。**HTML、CSS**（Cascading Style Sheets）和 JavaScript 是构建和设计网页的三种核心技术，它们共同作用于网页的内容、表现形式和交互功能。

1. **html官方文档**

HTML的官方文档由万维网联盟（World Wide Web Consortium，W3C）和Web Hypertext Application Technology Working Group（WHATWG）共同维护。以下是一些官方文档的链接：

1. W3C的HTML5.3推荐标准（Recommendation）：
   * 英文版：https://www.w3.org/TR/html53/
   * 中文版（可能不是最新的）：https://www.w3.org/Translations/HTML-5.3/zh/
2. WHATWG的HTML Living Standard：
   * 英文版：https://html.spec.whatwg.org/

WHATWG的HTML Living Standard是一个持续更新的标准，它不同于W3C的版本，后者更倾向于发布固定的版本。WHATWG的文档会随着Web技术的发展而不断更新，而W3C则会在一段时间后发布一个新的推荐标准版本。

这些文档详细描述了HTML的语法、元素、属性和API，是开发人员在创建网页时的重要参考资源。由于HTML是一个不断发展的标准，因此建议定期查看这些文档以获取最新的信息和更新。

1. **哪些浏览器支持html5**

几乎所有的现代浏览器都支持 HTML5。以下是一些主流浏览器及其对 HTML5 的支持情况：

1. Google Chrome：Google Chrome 是基于 WebKit 的浏览器，它对 HTML5 的支持非常全面。
2. Mozilla Firefox：Firefox 是一个开源浏览器，它对 HTML5 的支持也非常好。
3. Apple Safari：Safari 是苹果公司的浏览器，它同样基于 WebKit，对 HTML5 有很好的支持。
4. Microsoft Edge：Microsoft Edge 是微软推出的新浏览器，它取代了老版本的 Internet Explorer，并且对 HTML5 的支持有了显著提升。
5. Opera：Opera 也对 HTML5 提供了很好的支持。
6. Internet Explorer 9 及以上版本：虽然 Internet Explorer 在某些 HTML5 特性上的支持不如现代浏览器，但是 IE9 及以上版本仍然提供了一定程度的 HTML5 支持。

需要注意的是，不同版本的浏览器对 HTML5 的支持程度可能会有所不同。一般来说，浏览器的最新版本会提供最好的 HTML5 支持。此外，随着 HTML5 标准的不断完善和更新，浏览器厂商也会不断更新他们的产品以提供更好的兼容性。

如果你想要测试特定浏览器对 HTML5 的支持情况，可以使用一些在线工具，如 HTML5 Compatibility Test 或 HTML5 Test，这些工具可以检测浏览器对 HTML5 特性的支持程度。

1. **CSS预处理器有哪些**

CSS 预处理器是一些工具，它们允许开发者使用一种更接近于编程语言的语法来编写 CSS，然后将这些代码转换为原生 CSS。预处理器提供了变量、嵌套、混合（mixins）、继承、函数和作用域等特性，这些特性在原生 CSS 中是不存在的。以下是一些流行的 CSS 预处理器：

1. **Sass**：
   * 语法：Sass 语法（SCSS）类似于 CSS，但包含了 Sass 的所有特性。
   * 编译器：sass 或 node-sass 命令。
   * 官方网站：https://sass-lang.com/
2. **Less**：
   * 语法：类似于 CSS，但增加了变量、混合（mixins）、嵌套和作用域等特性。
   * 编译器：lessc 或 less 命令。
   * 官方网站：http://lesscss.org/
3. **Stylus**：
   * 语法：与 CSS 相似，但提供了变量、嵌套、混合（mixins）和作用域等特性。
   * 编译器：stylus 命令。
   * 官方网站：http://learnboost.github.io/stylus/
4. **Sass 扩展**：
   * 语法：与 Sass 类似，但提供了更多功能和灵活性。
   * 编译器：sass 或 node-sass 命令。
   * 官方网站：https://sass-lang.com/extensions
5. **Pug**：
   * 语法：Pug 是一个模板引擎，它使用 JavaScript 来生成 HTML，并可以与 Sass 结合使用。
   * 编译器：pug 命令。
   * 官方网站：https://pugjs.org/
6. **Stylish**：
   * 语法：与 CSS 类似，但提供了变量、嵌套、混合（mixins）和作用域等特性。
   * 编译器：styl 命令。
   * 官方网站：https://github.com/modular-css/styl

这些预处理器各有特点，开发者可以根据项目需求和个人偏好选择合适的预处理器。在实际开发中，预处理器可以提高 CSS 代码的编写效率和质量。

1. **开发手册(三方)**

<https://www.w3cschool.cn/html/dict/>

1. **webKit是什么**

WebKit 是一个开源的浏览器引擎，它被用于许多现代浏览器的核心，包括苹果的 Safari 和腾讯的 QQ 浏览器。WebKit 的前身是 KDE 的 KHTML 和 KJS，它基于这些技术进行了改进和优化。

WebKit 主要由两部分组成：

1. WebCore：负责 HTML 和 CSS 的渲染，以及 DOM 的处理。
2. JavaScriptCore（也称为 SquirrelFish 或 Nitro）：负责 JavaScript 的解释和执行。

WebKit 支持多种网页标准，包括 HTML5、CSS3、SVG 和 WebGL 等。它的架构设计使得它能够快速渲染网页，并且具有良好的跨平台性能。

WebKit 项目由多个分支组成，其中一些是由不同的公司或组织维护的。例如：

* Apple 维护的WebKit：用于 Safari 浏览器。
* Chromium 项目中的 Blink：Google 在 WebKit 的分支上创建了 Blink，用于 Chrome 浏览器和 Opera 浏览器。
* Samsung 维护的 WebKit：用于 Samsung Internet 浏览器。

随着时间的推移，WebKit 引擎也在不断发展，增加了对新标准的支持，并进行了性能优化。此外，WebKit 还被用于一些非浏览器的应用程序中，如电子书阅读器和网页视图组件，因为它能够提供丰富的网页内容渲染能力。

1. **Svg是什么**

SVG（Scalable Vector Graphics，可缩放矢量图形）是一种基于XML的图形格式，用于在网页上创建和显示图形。与位图图像（如JPEG和PNG）不同，SVG图形是由数学公式定义的矢量路径，这意味着它们可以无限缩放而不会失真。

SVG的主要特点包括：

1. **矢量图形**：SVG使用数学公式来定义形状和路径，因此它们可以无限缩放而不会失真。
2. **基于XML**：SVG文件是XML格式的，这意味着它们可以被任何文本编辑器打开和编辑。
3. **可编程性**：SVG图形可以通过JavaScript编程来创建和修改，这使得网页设计师和开发者可以创建动态和交互式的图形。
4. **与CSS集成**：SVG元素可以与CSS样式表结合，以定义颜色、字体、大小等样式属性。
5. **动画和交互**：SVG支持动画和交互功能，允许图形元素在网页上以动画形式出现或响应用户的交互。
6. **支持多种图形对象**：SVG支持多种图形对象，包括直线、曲线、矩形、圆形、椭圆、多边形等。
7. **可扩展性**：SVG具有良好的可扩展性，可以轻松地添加新的图形对象和功能。

SVG文件通常以.svg扩展名保存，它们可以直接嵌入HTML页面中，也可以作为独立的文件链接到页面中。随着Web技术的不断发展，SVG已经成为网页设计和开发中不可或缺的一部分，特别是在创建图标、图表、按钮和其他图形元素时。

1. **WebGL是什么**

WebGL（Web Graphics Library）是一种JavaScript API，它允许开发者使用OpenGL ES（Open Graphics Library for Embedded Systems）在网页中创建高性能的2D和3D图形。WebGL 1.0 是 OpenGL ES 2.0 的实现，而 WebGL 2.0 则是 OpenGL ES 3.0 的实现。

WebGL 的一些关键特点包括：

**直接访问图形硬件**：WebGL 允许开发者直接访问用户的图形硬件，包括显卡和图形处理器。

**高性能图形渲染**：通过使用图形硬件，WebGL 能够提供高性能的图形渲染，使得在网页上创建复杂的图形和动画成为可能。

**2D和3D图形支持**：WebGL 支持2D和3D图形，可以用于创建游戏、虚拟现实（VR）应用、3D模型和动画等。

**与HTML5集成**：WebGL 可以与HTML5的其他技术，如Canvas、SVG和CSS3，结合使用，以创建更加丰富和动态的网页内容。

**跨平台兼容性**：WebGL 可以在不同的操作系统和设备上运行，包括桌面、移动设备和嵌入式设备。

**社区和库支持**：WebGL 有一个活跃的社区，提供大量的教程、示例和库，帮助开发者学习和使用 WebGL。

使用 WebGL 需要有一定的图形编程知识，特别是对OpenGL ES API的熟悉。然而，由于WebGL的强大功能和广泛的应用场景，它已经成为网页开发中一个非常重要的技术。

1. **OpenGl是什么**

OpenGL（Open Graphics Library）是一个跨语言、跨平台的API，用于渲染2D和3D矢量图形。它最初由SGI公司开发，旨在为C和C++提供图形编程的统一接口。OpenGL是行业标准之一，广泛应用于游戏开发、计算机辅助设计（CAD）、虚拟现实（VR）和科学可视化等领域。

OpenGL的主要特点包括：

1. **跨平台兼容性**：OpenGL可以在多个操作系统上运行，包括Windows、macOS、Linux、Android和iOS等。
2. **面向性能**：OpenGL的设计注重性能，允许开发者直接访问图形硬件，以实现高效的图形渲染。
3. **图形管道**：OpenGL定义了一个图形渲染管道，包括顶点处理、图形着色、光栅化、纹理映射和像素处理等阶段。
4. **可编程性**：OpenGL支持可编程着色器，开发者可以使用GLSL（OpenGL Shading Language）来编写顶点着色器和片元着色器，以实现复杂的图形效果。
5. **硬件抽象**：OpenGL提供了对图形硬件的抽象，允许开发者编写与硬件无关的代码，从而提高代码的通用性和可移植性。
6. **社区和库支持**：OpenGL有一个活跃的社区，提供大量的教程、示例和库，帮助开发者学习和使用 OpenGL。

OpenGL API分为两个版本：

* **OpenGL 1.x - 固定功能管线**：早期版本，使用固定的图形处理管道，开发者无法直接编写着色器。
* **OpenGL 2.x - 可编程管线**：引入了可编程着色器，开发者可以编写GLSL来控制图形渲染的细节。

随着技术的发展，OpenGL API也在不断更新和扩展，以支持新的图形技术和需求。OpenGL ES是OpenGL的一个子集，专为嵌入式设备设计，如移动电话和游戏机。

1. **OpenGL与Vulkan**

OpenGL 和 Vulkan 都是用于图形渲染的API，但它们有不同的设计理念和目标市场。

1. **OpenGL**：
   * **历史**：OpenGL 最初由SGI公司开发，并于1992年发布。它是一个跨平台的API，广泛用于桌面、移动设备和游戏开发。
   * **设计理念**：OpenGL 采用“图形管道”的概念，其中开发者可以通过一系列的函数来控制图形渲染的各个阶段。它提供了固定功能管线和可编程管线两种模式。
   * **用途**：OpenGL 主要用于游戏开发、科学可视化、虚拟现实和3D建模等领域。
   * **性能**：OpenGL 的性能依赖于图形硬件和驱动程序，而驱动程序的质量在不同平台上可能会有所不同。
2. **Vulkan**：
   * **历史**：Vulkan 是由 Khronos Group 在2015年发布的，它是基于Mantle API的，后者是由苹果公司开发的一个图形API。
   * **设计理念**：Vulkan 采用了一种称为“直接操作图形硬件”的设计理念，它允许开发者更直接地控制图形硬件，以提高性能和效率。
   * **用途**：Vulkan 主要用于高性能的图形应用，如游戏、虚拟现实和3D渲染。
   * **性能**：Vulkan 旨在提供比OpenGL更高的性能和效率，尤其是在多线程和多GPU场景下。

总结来说，OpenGL 和 Vulkan 都是用于图形渲染的API，但它们有不同的设计理念和目标市场。OpenGL 更注重于提供广泛的图形功能和跨平台兼容性，而 Vulkan 则更侧重于高性能和效率，特别是在多线程和多GPU场景下。开发者可能会根据项目的具体需求和目标平台来选择使用哪一个API。

1. **每个操作系统独有图形API**
   1. **苹果图形API(Metal)**

Linux 操作系统提供了一系列图形API，用于开发图形用户界面（GUI）应用程序。以下是一些主要的图形API：

1. **X Window System (X11)**：
   * **用途**：X11 是 Linux 系统中最常用的图形API，它提供了一个抽象层，允许应用程序与各种图形硬件和窗口管理器交互。
   * **集成**：X11 广泛集成在大多数Linux发行版中，是许多桌面环境的底层图形系统。
2. **Wayland**：
   * **用途**：Wayland 是一个更现代的图形服务器，旨在取代 X11。它提供了更简单的架构和更好的性能。
   * **集成**：Wayland 正在逐渐被一些Linux发行版所采用，如 Fedora 和 Ubuntu。
3. **Mesa**：
   * **用途**：Mesa 是一个开源的3D图形库，它提供了一个OpenGL和Vulkan的实现，用于在Linux上渲染图形。
   * **性能**：Mesa 旨在提供高性能的图形渲染，尤其是在使用OpenGL和Vulkan时。
4. **OpenGL**：
   * **用途**：OpenGL 是一个跨平台的图形API，用于开发2D和3D图形应用程序。它在Linux上通过Mesa库提供支持。
   * **性能**：OpenGL 在Linux上的性能依赖于图形硬件和驱动程序。
5. **Vulkan**：
   * **用途**：Vulkan 是一个高性能的图形API，它提供了一个低级接口，允许开发者直接控制图形硬件。它在Linux上通过Mesa库提供支持。
   * **性能**：Vulkan 旨在提供比OpenGL更高的性能和效率，尤其是在多线程和多GPU场景下。
6. **Qt**：
   * **用途**：Qt 是一个跨平台的C++库，它提供了一个全面的图形用户界面框架，用于开发GUI应用程序。
   * **集成**：Qt 支持多种图形API，包括OpenGL、Vulkan和X11。
7. **GTK**：
   * **用途**：GTK 是一个跨平台的C语言库，用于开发图形用户界面。它是GNOME桌面环境的一部分。
   * **集成**：GTK 支持多种图形API，包括X11、Wayland和OpenGL。
8. **EGL**：
   * **用途**：EGL（Embedded Graphics Library）是一个跨平台的API，用于在嵌入式系统中提供高性能的图形渲染。
   * **集成**：EGL 支持多种图形API，包括OpenGL和Vulkan。

开发者可以根据应用程序的需求和平台选择合适的图形API。例如，如果需要高性能的3D图形渲染，可能会选择OpenGL或Vulkan；如果需要跨平台兼容性，可能会选择Qt或GTK。

* 1. **Windows图形API(DirectX)**

Windows操作系统提供了一系列图形API，用于开发图形用户界面（GUI）应用程序。以下是一些主要的图形API：

1. **DirectX**：
   * **用途**：DirectX 是一套由微软开发的API集合，主要用于游戏开发和实时图形应用程序。它包括多个组件，如Direct3D（用于3D图形渲染）、Direct2D（用于2D图形和文本渲染）、DirectSound（用于音频处理）等。
   * **集成**：DirectX 紧密集成在Windows操作系统中，提供了高性能的图形渲染。
2. **Windows Presentation Foundation (WPF)**：
   * **用途**：WPF 是一个用于创建Windows桌面应用程序的图形框架，它提供了一个基于.NET的图形引擎，用于创建丰富的用户界面。
   * **集成**：WPF 紧密集成在.NET框架中，支持Windows桌面应用程序的开发。
3. **Windows Forms**：
   * **用途**：Windows Forms 是另一个用于创建Windows桌面应用程序的图形框架，它提供了一个基于.NET的图形引擎，用于创建图形用户界面。
   * **集成**：Windows Forms 紧密集成在.NET框架中，支持Windows桌面应用程序的开发。
4. **Windows API**：
   * **用途**：Windows API 是一套用于开发Windows应用程序的函数和数据类型集合，它提供了广泛的系统级功能，包括图形渲染。
   * **集成**：Windows API 是Windows操作系统的一部分，支持所有类型的Windows应用程序。
5. **Direct3D 12**：
   * **用途**：Direct3D 12 是 DirectX 的一部分，提供了一个高性能的3D图形渲染API。它被设计为提供更好的性能和更低的功耗，尤其是在现代硬件上。
   * **集成**：Direct3D 12 紧密集成在Windows操作系统中，支持高性能的图形渲染。

开发者可以根据应用程序的需求和平台选择合适的图形API。例如，如果需要高性能的3D图形渲染，可能会选择DirectX或Direct3D 12；如果需要跨平台兼容性，可能会选择WPF或Windows Forms。

* 1. **Linux图形API（x11等）**

Linux操作系统提供了一系列图形API，用于开发图形用户界面（GUI）应用程序。以下是一些主要的图形API：

1. **X Window System (X11)**：
   * **用途**：X11 是 Linux 系统中最常用的图形API，它提供了一个抽象层，允许应用程序与各种图形硬件和窗口管理器交互。
   * **集成**：X11 广泛集成在大多数Linux发行版中，是许多桌面环境的底层图形系统。
2. **Wayland**：
   * **用途**：Wayland 是一个更现代的图形服务器，旨在取代 X11。它提供了更简单的架构和更好的性能。
   * **集成**：Wayland 正在逐渐被一些Linux发行版所采用，如 Fedora 和 Ubuntu。
3. **Mesa**：
   * **用途**：Mesa 是一个开源的3D图形库，它提供了一个OpenGL和Vulkan的实现，用于在Linux上渲染图形。
   * **性能**：Mesa 旨在提供高性能的图形渲染，尤其是在使用OpenGL和Vulkan时。
4. **OpenGL**：
   * **用途**：OpenGL 是一个跨平台的图形API，用于开发2D和3D图形应用程序。它在Linux上通过Mesa库提供支持。
   * **性能**：OpenGL 在Linux上的性能依赖于图形硬件和驱动程序。
5. **Vulkan**：
   * **用途**：Vulkan 是一个高性能的图形API，它提供了一个低级接口，允许开发者直接控制图形硬件。它在Linux上通过Mesa库提供支持。
   * **性能**：Vulkan 旨在提供比OpenGL更高的性能和效率，尤其是在多线程和多GPU场景下。
6. **Qt**：
   * **用途**：Qt 是一个跨平台的C++库，它提供了一个全面的图形用户界面框架，用于开发GUI应用程序。
   * **集成**：Qt 支持多种图形API，包括OpenGL、Vulkan和X11。
7. **GTK**：
   * **用途**：GTK 是一个跨平台的C语言库，用于开发图形用户界面。它是GNOME桌面环境的一部分。
   * **集成**：GTK 支持多种图形API，包括X11、Wayland和OpenGL。
8. **EGL**：
   * **用途**：EGL（Embedded Graphics Library）是一个跨平台的API，用于在嵌入式系统中提供高性能的图形渲染。
   * **集成**：EGL 支持多种图形API，包括OpenGL和Vulkan。

开发者可以根据应用程序的需求和平台选择合适的图形API。例如，如果需要高性能的3D图形渲染，可能会选择OpenGL或Vulkan；如果需要跨平台兼容性，可能会选择Qt或GTK。

1. **OpenCV与OpenGl区别**

OpenCV 和 OpenGL 是两个完全不同的库，它们各自用于不同的目的和领域。

1. **OpenCV**：
   * **用途**：OpenCV（Open Source Computer Vision Library）是一个开源的计算机视觉库，用于图像处理、计算机视觉和机器学习。它提供了大量用于图像和视频分析的算法和函数。
   * **主要功能**：OpenCV 包括图像和视频的读取和写入、图像处理、特征提取、物体检测、跟踪、图像识别和机器学习等。
   * **集成**：OpenCV 可以与多种编程语言集成，包括C++、Python、Java等。
   * **性能**：OpenCV 侧重于提供易于使用的接口和丰富的功能，而不一定是最快的性能。
2. **OpenGL**：
   * **用途**：OpenGL（Open Graphics Library）是一个跨平台的图形库，用于创建和操作2D和3D图形。它主要用于游戏开发、科学可视化、虚拟现实和3D建模等领域。
   * **主要功能**：OpenGL 提供了图形渲染的基本功能，包括着色器、纹理、几何形状、光照、阴影和动画等。
   * **集成**：OpenGL 支持跨平台开发，包括Windows、macOS、Linux等。
   * **性能**：OpenGL 专注于提供高性能的图形渲染，特别是对于图形硬件和驱动程序的支持。

总结来说，OpenCV 和 OpenGL 的主要区别在于它们的目标领域和提供的功能。OpenCV 专注于计算机视觉和图像处理，而 OpenGL 专注于图形渲染和2D/3D图形处理。开发者可以根据项目的具体需求和目标平台来选择使用哪一个库。

1. **Opencl是什么**

OpenCL（Open Computing Language）是一种用于编写能够在各种计算设备上运行的并行程序的编程语言和API。它由一个行业联盟维护，包括苹果、IBM、英特尔、AMD、NVIDIA、Sony和德州仪器等公司。OpenCL 的设计目标是提供一个统一的标准，以便在不同的计算设备上编写高性能的并行代码，包括中央处理器（CPU）、图形处理器（GPU）、数字信号处理器（DSP）、通用处理器（FPGA）和专用集成电路（ASIC）。

OpenCL 的一些关键特点包括：

1. **可移植性**：OpenCL 代码可以在不同的计算设备上运行，只要这些设备支持 OpenCL。
2. **并行计算**：OpenCL 允许开发者编写并行代码，以利用多核CPU、GPU或其他并行处理硬件的计算能力。
3. **面向性能**：OpenCL 提供了对底层硬件的直接访问，这使得开发者可以编写优化性能的代码。
4. **跨平台兼容性**：OpenCL 支持跨平台开发，包括Windows、macOS、Linux等操作系统。
5. **支持多种编程模型**：OpenCL 支持多种编程模型，包括命令式、函数式和数据并行编程。
6. **社区和库支持**：OpenCL 有一个活跃的社区，提供大量的教程、示例和库，帮助开发者学习和使用 OpenCL。

OpenCL 广泛应用于高性能计算、科学计算、图形处理和移动计算等领域。开发者可以根据项目的具体需求和目标平台来选择使用 OpenCL。

1. **opencl与cuda区别**

OpenCL（Open Computing Language）是一种用于编写能够在各种计算设备上运行的并行程序的编程语言和API。它由一个行业联盟维护，包括苹果、IBM、英特尔、AMD、NVIDIA、Sony和德州仪器等公司。OpenCL 的设计目标是提供一个统一的标准，以便在不同的计算设备上编写高性能的并行代码，包括中央处理器（CPU）、图形处理器（GPU）、数字信号处理器（DSP）、通用处理器（FPGA）和专用集成电路（ASIC）。

OpenCL 的一些关键特点包括：

1. **可移植性**：OpenCL 代码可以在不同的计算设备上运行，只要这些设备支持 OpenCL。
2. **并行计算**：OpenCL 允许开发者编写并行代码，以利用多核CPU、GPU或其他并行处理硬件的计算能力。
3. **面向性能**：OpenCL 提供了对底层硬件的直接访问，这使得开发者可以编写优化性能的代码。
4. **跨平台兼容性**：OpenCL 支持跨平台开发，包括Windows、macOS、Linux等操作系统。
5. **支持多种编程模型**：OpenCL 支持多种编程模型，包括命令式、函数式和数据并行编程。
6. **社区和库支持**：OpenCL 有一个活跃的社区，提供大量的教程、示例和库，帮助开发者学习和使用 OpenCL。

OpenCL 广泛应用于高性能计算、科学计算、图形处理和移动计算等领域。开发者可以根据项目的具体需求和目标平台来选择使用 OpenCL。

1. **Emscripten是什么**

Emscripten是一个开源的LLVM编译器基础设施项目，它能够将C和C++代码编译成WebAssembly或ASM.js，这样就可以在Web浏览器中运行。Emscripten允许开发者将现有的C/C++代码库和应用程序带到Web上，而不需要重写它们。

WebAssembly（通常缩写为Wasm）是一种可以在现代Web浏览器中运行的低级语言，它提供了一种方法，以接近原生性能的方式在Web上运行代码。ASM.js是WebAssembly之前的一种技术，它是一种JavaScript的严格子集，可以被浏览器优化以获得更好的性能。

Emscripten的主要特点包括：

1. **性能**：编译后的代码可以运行得非常快，接近原生性能。
2. **兼容性**：支持大多数C/C++代码，包括复杂的代码库和应用程序。
3. **工具链**：提供了一套完整的工具链，包括编译器、链接器和调试工具。
4. **API支持**：允许访问多种Web API，如WebGL、WebAudio、WebSocket等。
5. **集成**：可以与现有的Web开发工作流程和工具集成，如Node.js、npm、Webpack等。

Emscripten使得开发者可以在Web上部署复杂的C/C++应用程序，例如游戏、科学模拟和桌面应用程序的Web版本。它为Web平台带来了新的可能性，并促进了Web与其他平台之间的互操作性。

1. **asm.js与WebAssembly的不同**

ChatGPT

asm.js和WebAssembly（Wasm）都是用于在Web浏览器中运行高性能代码的技术，但它们有一些重要的区别：

1. **语言特性**：
   * **asm.js**：asm.js是一种JavaScript的严格子集，通过严格的静态类型注解和限制，可以让浏览器引擎将JavaScript代码转换为高效的本机代码。asm.js可以在所有支持JavaScript的浏览器上运行，但是它仍然是JavaScript代码。
   * **WebAssembly**：WebAssembly是一种全新的二进制指令格式，设计用于在Web浏览器中运行，并且可以与JavaScript共存。WebAssembly是一种独立的语言，具有自己的二进制格式和编译器工具链。
2. **性能**：
   * **asm.js**：asm.js可以提供比传统JavaScript更高的性能，因为它可以通过静态类型注解来优化代码。但是，性能仍然受限于JavaScript的解释器和运行时环境。
   * **WebAssembly**：WebAssembly旨在提供比asm.js更好的性能，因为它是一种更接近本机代码的格式。WebAssembly的目标是让Web应用程序能够运行类似于本机应用程序的性能。
3. **二进制格式**：
   * **asm.js**：asm.js代码仍然是文本格式的JavaScript代码，虽然有一些特定的语法和注解，但它仍然是可以阅读和修改的文本。
   * **WebAssembly**：WebAssembly代码是一种二进制格式，无法直接由人类阅读或编辑，它需要通过编译器工具链将高级语言编译成WebAssembly模块。
4. **浏览器支持**：
   * **asm.js**：asm.js可以在所有现代的JavaScript引擎中运行，因此几乎所有支持JavaScript的浏览器都可以运行asm.js代码。
   * **WebAssembly**：WebAssembly需要浏览器提供对其二进制格式的原生支持。目前，大多数现代浏览器都支持WebAssembly，包括Chrome、Firefox、Safari和Edge。

综上所述，asm.js是JavaScript的严格子集，旨在通过静态类型注解提供更高的性能，而WebAssembly是一种全新的二进制指令格式，旨在为Web应用程序提供与本机应用程序类似的性能。

1. **asm.js与WebAssembly能调用gpu资源么**

是的，asm.js 和 WebAssembly 都能够调用 GPU 资源，但它们各自的方式和场景有所不同。

1. **asm.js 调用 GPU**:
   * asm.js 本身并不直接与 GPU 交互。然而，asm.js 代码可以通过 JavaScript 的 API 来访问 GPU 资源。例如，asm.js 可以通过 WebGL API 来实现高性能的 2D 和 3D 图形渲染。由于 asm.js 是 JavaScript 的一个子集，它可以直接调用任何 JavaScript API，包括 WebGL。
2. **WebAssembly 调用 GPU**:
   * WebAssembly 也可以通过 JavaScript 的 API 来访问 GPU。与 asm.js 类似，WebAssembly 可以使用 WebGL 来进行图形渲染。此外，WebAssembly 还可以通过 WebGPU API（目前仍处于实验阶段）来访问 GPU。WebGPU 是一个新一代的图形 API，旨在提供更高效的 GPU 访问，支持 Vulkan、DirectX 12 和 Metal 等现代图形接口。

在两种情况下，都需要使用特定的 API 来编写与 GPU 交互的代码。通常，这涉及到编写或使用已经存在的工具和库，这些工具和库封装了底层的 GPU API 调用，使得开发者可以更容易地在 asm.js 或 WebAssembly 中进行 GPU 加速的应用开发。

例如，Emscripten 编译器提供了一个名为 emscripten 的 JavaScript 库，它包含了对 WebGL 的支持，允许开发者从 C/C++ 代码中直接调用 WebGL API。同样，对于 WebAssembly，开发者可以使用类似的库来访问 GPU 资源。

总的来说，虽然 asm.js 和 WebAssembly 本身不直接提供 GPU 访问，但它们可以通过 JavaScript 的图形和计算 API 来间接地利用 GPU 资源。随着 Web 平台的发展，未来可能会有更多专门为 WebAssembly 设计的 API，以提供更直接的 GPU 访问和更高效的性能。

1. **程序实例**

18.1）html,css,js实例

18.2）css预处理（sass为例）

18.3）svg实例

18.4）webGL实例

18.5）openGL与vulkan实例

18.6）操作系统独有API(linux图形API)

18.7）openCV实例

18.8）opengl与cuda实例

18.9）Emscripten与asm.js,webAssembly实例

1. **Nodejs模块化前端开发**
2. **Xxx**