Web Worker

Web Worker 是一种在 Web 应用程序中运行后台任务的技术，它允许在浏览器的单独线程中执行 JavaScript 代码，从而不会阻塞主线程或用户界面。因此，适合放在 Web Worker 子线程中处理的任务主要是那些需要大量计算或可能导致界面卡顿的操作。以下是一些实际工作中适合使用 Web Worker 的例子：

1. **大数据处理和计算**：假设你正在开发一个数据分析应用，用户需要上传大量数据并进行复杂的计算。如果将这些计算放在主线程中执行，可能会导致界面无响应或卡顿。使用 Web Worker，你可以将这些计算任务放在子线程中执行，同时保持主线程的流畅运行，从而提供更好的用户体验。

2. **图片处理**：在图像处理应用中，经常需要对图片进行复杂的操作，如滤镜、旋转、缩放等。这些操作通常很耗时，并且如果直接在主线程中执行，可能会导致界面冻结。通过将图片处理任务放在 Web Worker 中，可以确保主线程继续响应用户的操作，从而提供流畅的界面体验。

3. **文件上传和处理**：在用户上传大文件时，你可能需要对这些文件进行解析、验证或转换。这些操作可能会占用大量时间，如果放在主线程中执行，会影响用户界面的响应性。使用 Web Worker，你可以在后台线程中处理文件，同时允许用户继续与界面进行交互。

4. **实时通信和聊天应用**：在实时通信或聊天应用中，消息的处理和传递通常需要快速响应。通过在 Web Worker 中处理消息，可以确保消息的快速传递和处理，同时不会干扰用户界面的其他操作。

5. **游戏物理模拟和AI决策**：对于需要大量计算的游戏逻辑，如物理模拟或AI决策，使用 Web Worker 可以避免游戏在主线程上运行时出现卡顿。通过将这些计算密集型任务放在子线程中执行，可以确保游戏的流畅运行和更好的用户体验。

6. **音频和视频处理**：对于需要实时处理音频和视频的应用，如实时语音识别或视频流分析，Web Worker 可以用来执行这些操作。这样，主线程可以专注于处理用户界面和其他交互任务，而音频和视频处理则在后台线程中独立进行。

下面以后台数值计算为例演示：

```
// main.js
let worker = new Worker("./worker.js");

worker.postMessage({
    status: 0
});

worker.onmessage = function (event) {
    document.querySelector(".calc").innerHTML = event.data;
    worker.terminate();
};
```

```
// worker.js
onmessage = function (event) {
    console.log(event);
    let status = event.data.status;
```

```
    if (status === 0) {
        startToCalc();
    }
}

function startToCalc() {
    let arr = [];
    for (let i = 1; i <= 1000; i++) {
        arr.push(i);
    }
    let tStart = new Date().getTime();
    let total = arr.reduce((sum, item) => sum + item);
    let tEnd = new Date().getTime();
    console.log("耗时: ", tEnd - tStart, "ms");
    postMessage("处理完成, 耗时: ", tEnd - tStart, "ms"); // 使用中文可能会引起乱码
    self.close();
}
```