任务处理器

题目描述

请实现一个异步任务处理器，能够按照要求处理一系列异步任务。这个任务处理器需要遵循特定的执行步骤和错误处理机制。

要求

1. 点击开始处理按钮后，会调用 `onStartBtnClick` 函数，开始异步任务处理，具体执行步骤如下
   - 首先会调用 `loadConfig` 加载配置文件，获取文件列表
   - 然后会根据文件列表调用 `loadFile` 加载文件
   - 文件列表中的所有文件加载完毕后，会调用 `initSystem` 进行系统初始化
2. 针对文件加载，需要满足以下附加要求：
   - 加载文件时，需要做并发控制，最多同时加载 3 个文件
   - 加载文件时，需要添加超时控制，超时时间为 5 秒
   - 加载文件失败时，需要对单文件做 backoff retry 处理，重试次数为 3 次
   - 对错误进行捕获并打印输出

界面要求

1. 页面上应有一个按钮，点击后开始异步任务处理
2. 页面上应有一个控制台区域，用于显示任务执行过程中的日志输出
3. 页面应当显示任务进度以及总体执行状态（如进行中、已完成、失败等）

异步任务处理器

本程序演示一个具有并发控制、超时控制和重试机制的异步任务处理器。

开始处理 状态: 空闲

控制台输出:

提示

1. backoff retry 是指在失败后等待一段时间再重试，且每次失败后等待时间增加的策略

完成要求

1. 完成异步任务处理逻辑，确保满足所有要求
2. 实现一个能在主流浏览器中正常运行的网页应用
3. 保证代码的可读性和健壮性
4. 如无不便，请使用 github.com 或者 gitlab.com 提交答案， 回复公开代码库地址即可

关键代码脚手架

代码块

```
  /**
   * 开始按钮点击事件处理函数
   * 在这里实现异步任务处理逻辑
   */
  async onStartBtnClick() {
      // TODO: 请在此处开始作答，实现异步任务处理逻辑
      // loadConfig & loadFile & initSystem
  }
```

```
 9
10  /**
11   * 加载配置文件
12   * @returns {Promise<string[]>} 文件列表
13   */
14  async loadConfig() {
15      // Simulates an asynchronous file retrieval operation using setTimeout
16      //
17      // This function mocks a network or file system request by returning a
    Promise
18      // that resolves after a specified delay. The Promise resolves with an
    array
19      // containing 100 file objects, each with unique properties.
20      // The implementation includes comprehensive logging for both successful
    operations
21      // and error scenarios, providing visibility into the execution flow.
22  }
23
24  /**
25   * 加载文件
26   * @param {string} file 文件名
27   * @returns {Promise<void>}
28   */
29  async loadFile(file) {
30      // Asynchronously loads a specified file with simulated network delay
    (Mock using setTimeout)
31      //
32      // This method simulates the loading of a file by creating a Promise that
    resolves
33      // or rejects after a random timeout between 1-3 seconds. It has a 90%
    success rate
34      // and 10% failure rate to mimic real-world network conditions.
35      //
36      // The method logs the entire lifecycle of the file loading process
    including:
37      // - When loading begins
38      // - When loading completes successfully
39      // - When loading fails
40      //
41      // On success, it increments the internal counter of loaded files and
    updates
42      // the progress indicator.
43  }
44
45  /**
46   * 初始化系统
47   * @returns {Promise<void>}
```

```
48   */
49  async initSystem() {
50      // Initializes the system with required configurations and resources (Mock using setTimeout)
51      //
52      // This method performs the necessary setup procedures for the system to function
53      // properly. It returns a Promise that resolves after a simulated initialization
54      // process that takes 1 second to complete.
55      //
56      // The initialization process is always successful in this implementation and
57      // logs both the start and successful completion of the initialization process.
58      // This provides visibility into the system's startup sequence.
59  }
```