EventLogger - Vanilla JS 实现题（4 Part 递进）

系统已提供:

- **HTML**：6 个彩色方块（Part 1 使用）
- **CSS**：负责方块样式
- **utils.js**：提供 uploadEvents(events, { signal }) 异步函数
    - 返回 Promise
    - 支持 AbortController 中的 signal 参数
- **index.js**：入口文件，用于调用 EventLogger
- **EventLogger 类**：提供一个空壳结构，需要逐步实现功能

Part 1 — DOM 事件监听与基础日志记录

- 使用 **原生 JS**（如 document.querySelectorAll()）找到 .square 元素
- 给每个方块添加 **click** 监听器
- 每次点击调用 EventLogger.logEvent(event) 记录一个事件对象:
    - 方块颜色（style.backgroundColor）
    - 点击时间（new Date().toISOString()）
    - 其它自定义元数据

Part 2 — 按时间间隔批量上传

- 不再处理方块 UI，改为测试 **EventLogger** 类
- 功能要求:
    - 维护一个事件队列（queue）
    - **每 2 秒** 批量上传一次队列里的事件
    - 上传调用 uploadEvents(batch)

- 上传后清空已发送的事件

Part 3 — 串行化上传请求

- 模拟的 uploadEvents() 是异步的
- 要求：
  - **禁止并发上传**，必须按顺序完成
  - 如果上一次上传未完成，新批次要等待
- 关键点：
  - 使用 **Promise 链 / async-await** 或上传队列
  - 上传完成后才开始下一个批次

Part 4 — 超时与合并重试

- 如果一次上传超过 **3 秒**：
  - 调用 AbortController.abort() 取消上传
  - 取消的 batch 要与下一批事件合并后重新上传
- 上传仍需保持串行顺序
- 技术要点：
  - 使用 AbortController + setTimeout 实现超时
  - 超时批次不丢弃，而是延迟到下一轮一起发送

核心考点

1. 原生 DOM 操作与事件绑定
2. 队列管理与批量处理
3. 异步控制（串行执行、防并发）

4. 请求超时处理与重试机制
5. 熟悉 AbortController 用法

```
import { eventLogger } from "./eventLogger";

// ------------------ Part 1 ------------------
// 给页面所有 .square 方块绑定点击事件，点击时调用 eventLogger.logEvent
function setupClickListeners() {
  const squares = document.querySelectorAll(".square");

  squares.forEach((square) => {
    square.addEventListener("click", () => {
      const eventPayload = {
        hostname: window.location.hostname,
        data: {
          color: square.style.backgroundColor,
        },
      };
      eventLogger.logEvent("click", eventPayload);
    });
  });
}

setupClickListeners();
```

```
import { sendRequest } from "./utils";

export class EventLogger {
  constructor() {
    // Part 2 - 维护事件队列和定时器
    this.eventQueue = [];
    this.uploadInterval = 2000; // 2 秒上传一次
    this.timerId = null;

    // Part 3 - 串行上传控制
    this.isUploading = false;
    this.uploadQueue = [];

    // Part 4 - 超时控制和 abort controller
    this.currentAbortController = null;

    this.startBatchUpload();
  }

  // ---------------- Part 1 ----------------
  // Part 1: 每次点击立即上传
  // 但 Part 2 开启后，logEvent 改为入队，不立即上传
  logEvent(eventName, data) {
    const event = {
      eventName,
```

```
    hostname: data.hostname,
    timestamp: new Date().toISOString(),
    data: data.data,
  };

  // Part 2+ 后，改为入队等待批量上传
  this.eventQueue.push(event);
}

// Part 2 - 定时批量上传
startBatchUpload() {
  if (this.timerId) return; // 防止多次调用

  this.timerId = setInterval(() => {
    if (this.eventQueue.length === 0) return;

    // 取出当前批次的事件
    const batch = this.eventQueue;
    this.eventQueue = [];

    // 加入上传队列，保证串行上传（Part 3）
    this.enqueueUpload(batch);
  }, this.uploadInterval);
}
```

```
// Part 3 - 上传队列串行执行
enqueueUpload(batch) {
  return new Promise((resolve, reject) => {
    this.uploadQueue.push({ batch, resolve, reject });
    this.processQueue();
  });
}

async processQueue() {
  if (this.isUploading) return; // 已经有上传任务，等待完成

  if (this.uploadQueue.length === 0) return;

  const { batch, resolve, reject } = this.uploadQueue.shift();

  this.isUploading = true;

  try {
    await this.uploadBatchWithTimeout(batch);
    resolve();
  } catch (error) {
    reject(error);
  }

  this.isUploading = false;
```

```
  // 上传完成后，继续处理队列
  this.processQueue();
}

// Part 4 - 超时取消和合并重试
async uploadBatchWithTimeout(batch) {
  // 创建 AbortController 用于取消请求
  this.currentAbortController = new AbortController();
  const { signal } = this.currentAbortController;

  // 超时设置 3 秒
  const timeoutPromise = new Promise((_, reject) => {
    setTimeout(() => {
      // 超时，abort 请求
      this.currentAbortController.abort();
      reject(new Error("Upload timeout"));
    }, 3000);
  });

  // 发起上传请求
  const uploadPromise = sendRequest({ events: batch }, { signal });

  try {
    // Promise.race：哪个先完成，返回哪个
    await Promise.race([uploadPromise, timeoutPromise]);
```

```
      this.currentAbortController = null;
    } catch (err) {
      if (err.message === "Upload timeout") {
        // Part 4 - 超时合并处理
        // 把本次失败 batch 放回事件队列，并合并下一批事件再上传
        this.eventQueue = batch.concat(this.eventQueue);
        // 这里拒绝当前上传，下一轮 setInterval 会重新上传
        throw err;
      } else if (err.name === "AbortError") {
        // 请求被中止，也抛错让外层处理
        throw err;
      } else {
        throw err;
      }
    }
  }
}

const eventLogger = new EventLogger();
export { eventLogger };
```