MoodDiary dApp： index.html 全部解析

（本讲义为 Overleaf / LaTeX 版本，可直接编译成 PDF）

1 目标与整体概念

本 dApp 的核心目标是：用前端网页通过 **MetaMask**（用户身份 + 签名）与 **Sepolia 测试网**上的 **MoodDiary 智能合约**交互，实现：

- **Get Mood**：读取链上状态（`getMood()`，**read / view**，不弹钱包、不花 gas）
- **Set Mood**：写入链上状态（`setMood(string)`，**write / transaction**，弹钱包、需要 gas）

2 1) HTML 骨架：网页外壳

```
<!doctype html>
<html lang="en">
```

- `<!doctype html>`：声明 HTML5，避免浏览器用旧渲染模式。
- `lang="en"`：页面语言标记（SEO / 辅助工具 / 翻译等会用到）。

3 2) <head>：页面设定与样式

3.1 Meta 标签（显示与兼容性）

```
<meta charset="UTF-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>LearnWeb3 First dApp</title>
```

- `charset="UTF-8"`：避免中文/特殊字符乱码。
- `X-UA-Compatible`：旧式兼容性声明（现代浏览器影响很小）。
- `viewport`：手机端显示比例正确（响应式基础）。
- `title`：浏览器标签页标题。

3.2 CSS：纯前端排版（与区块链逻辑无关）

```
<style>
  body { text-align: center; font-family: Arial, Helvetica, sans-serif;
     }
  div { width: 20%; margin: 0 auto; display: flex; flex-direction:
     column; }
  button { width: 100%; margin: 10px 0px 5px 0px; }
  input { padding: 8px; }
  #status { margin-top: 12px; font-size: 14px; opacity: 0.9; }
</style>
```

- div 使用 flex + column，让输入框/按钮纵向排列。
- width:20% + margin:0 auto：让内容窄窄居中。
- #status：专门显示“连接中 / 发交易 / 等确认 / 完成”等状态文字。

4 3) <body>：UI 元素（用户会操作的部分）

4.1 输入框：用户输入 mood

```
<input type="text" id="mood" placeholder="e.g. happy" />
```

- id="mood"：后续 JS 用 document.getElementById("mood").value 读用户输入。

4.2 按钮：调用 get / set

```
<button onclick="getMood()">Get Mood</button>
<button onclick="setMood()">Set Mood</button>
```

- 点击按钮会调用全局函数 getMood() / setMood()。
- **重点：**由于后面使用 <script type="module">，模块作用域下的函数默认不是全局，因此需要显式挂到 window.getMood / window.setMood，让 onclick 能找到。

4.3 输出区：显示 mood 与状态

```
<p id="showMood"></p>
<p id="status"></p>
```

- showMood：显示链上读到的 mood（例如 Your Mood: happy）。
- status：显示当前流程状态（连接钱包、发送交易、等待确认等）。

5 4) <script type="module">: dApp 核心逻辑

5.1 4.1 为什么要用 module？

```
<script type="module">
```

- 允许使用 ES Module 的 `import ... from ...` 直接引入 viem。
- 允许顶层使用 `await`（例如请求钱包地址）。
- **副作用：**模块内变量/函数默认不挂在全局，若要让 HTML 的 `onclick="..."` 调到，需要 `window.getMood = ...`。

6 5) 引入 viem：区块链接口工具

```
import {
  createWalletClient,
  createPublicClient,
  custom,
  getContract,
} from "https://esm.sh/viem";
import { sepolia } from "https://esm.sh/viem/chains";
```

- `createWalletClient`：用于 **写入**（需要钱包签名的交易）。
- `createPublicClient`：用于 **读取**与等待交易确认（不需要签名）。
- `custom(window.ethereum)`：使用 MetaMask 注入的 provider 作为 transport（桥接到链）。
- `getContract`：用 ABI+ 地址创建合约实例，支持 `.read` / `.write` 调用。
- `sepolia`：链配置（chainId、网络参数等）。

7 6) 合约地址与 ABI：告诉前端“找谁、怎么叫它”

7.1 合约地址

```
const MoodContractAddress = "0x8089Ad39704F3Bc5f9b85C4BE4427631A137e58e
    ";
```

- 这是你在 Remix 部署到 Sepolia 后得到的 **合约地址**。
- **合约地址**是链上“程序”的位置（不是你的钱包地址）。

7.2 ABI（接口说明书）

```
const MoodContractABI = [
  {
    inputs: [{ internalType: "string", name: "_mood", type: "string" }],
    name: "setMood",
    outputs: [],
    stateMutability: "nonpayable",
    type: "function",
  },
  {
    inputs: [],
    name: "getMood",
    outputs: [{ internalType: "string", name: "", type: "string" }],
    stateMutability: "view",
    type: "function",
  },
];
```

- ABI = **前端与合约沟通的契约/说明书**，描述函数签名与参数类型。
- ABI 不匹配会导致：读函数回 `0x`（无数据）或调用失败。

8 7) 取得 DOM 元素：更新画面文字

```
const statusEl = document.getElementById("status");
const showMoodEl = document.getElementById("showMood");

function setStatus(msg) {
  statusEl.textContent = msg || "";
}
```

- `statusEl` / `showMoodEl`：用于把信息写回页面。
- `setStatus`：让更新状态文字变得更简洁。

9 8) 检查 MetaMask 是否存在

```
if (!window.ethereum) {
  setStatus("MetaMask not detected. Please install MetaMask.");
  throw new Error("MetaMask not detected");
}
```

- `window.ethereum` 是 MetaMask 注入到网页的对象。
- 没有它就无法连接钱包、无法签名交易，因此直接 `throw` 停止执行更安全。

10 9) 确保网络是 Sepolia（关键防踩坑）

```
const chainId = await window.ethereum.request({ method: "eth_chainId" })
    ;
if (chainId !== "0xaa36a7") {
  setStatus("Please switch MetaMask network to Sepolia, then refresh.");
  try {
    await window.ethereum.request({
      method: "wallet_switchEthereumChain",
      params: [{ chainId: "0xaa36a7" }],
    });
    setStatus("Switched to Sepolia. Refreshing...");
    location.reload();
  } catch (e) {}
}
```

- `eth_chainId`：询问当前连接的链。
- Sepolia chainId（十六进制）为 `0xaa36a7`。
- 若用户不在 Sepolia：
  - 提示用户切网
  - 尝试自动触发 MetaMask 切网
  - 成功后刷新页面，确保后续 client 走对网络
- **为什么重要：**合约地址只在对应网络存在；如果连错链，读合约常见表现是返回 `0x` 或调用失败。

11 10) 创建两个 client：read / write 分工

```
const publicClient = createPublicClient({
  chain: sepolia,
  transport: custom(window.ethereum),
});

const walletClient = createWalletClient({
  chain: sepolia,
  transport: custom(window.ethereum),
});
```

- `publicClient`：负责 **读取**（`eth_call`）与等待交易确认。
- `walletClient`：负责 **写入**（发交易、弹 MetaMask 签名）。
- 分工清晰、稳定，也更符合 viem 的常见用法。

12 11) 请求用户钱包地址（连接网站）

```
setStatus("Connecting wallet...");
const [address] = await walletClient.requestAddresses();
setStatus(`Connected: ${address.slice(0, 6)}...${address.slice(-4)}`);
```

- `requestAddresses()` 会触发 MetaMask 弹窗，用户点 Connect 后返回地址列表。
- `address` 将用于写入交易的 `account` 参数。
- 使用 `slice` 只显示地址头尾，属于常见 UX。

13 12) 创建合约实例：把 ABI + Address 变成可调用对象

```
const MoodContract = getContract({
  address: MoodContractAddress,
  abi: MoodContractABI,
  client: { public: publicClient, wallet: walletClient },
});
```

- 这一步把“合约地址 + ABI + client”组合起来，得到可直接调用的对象。
- 后续使用：
  - `MoodContract.read.getMood()`
  - `MoodContract.write.setMood(...)`

14 13) `window.getMood`：读链（不弹 MetaMask）

```
window.getMood = async function () {
  try {
    setStatus("Reading mood from contract...");
    const mood = await MoodContract.read.getMood();
    showMoodEl.innerText = `Your Mood: ${mood}`;
    setStatus("Read successful.");
  } catch (err) {
    console.error(err);
```

```
    setStatus("Read failed. Check console for details.");
  }
};
```

- **read** 调用本质是 `eth_call`：不改变链上状态、不消耗 gas、不需要签名，所以不会弹钱包。
- 成功后把 mood 显示在 `#showMood`，状态显示在 `#status`。

15 14) `window.setMood`：写链（弹 MetaMask + 等确认）

```
window.setMood = async function () {
  try {
    const mood = document.getElementById("mood").value.trim();
    if (!mood) {
      setStatus("Please enter a mood first.");
      return;
    }

    setStatus("Sending transaction (MetaMask will pop up)...");
    const hash = await MoodContract.write.setMood([mood], { account:
        address });

    setStatus(`Tx sent: ${hash.slice(0, 10)}... Waiting for confirmation
        ...`);

    await publicClient.waitForTransactionReceipt({ hash });

    setStatus("Transaction confirmed! Reading updated mood...");
    const updated = await MoodContract.read.getMood();
    showMoodEl.innerText = `Your Mood: ${updated}`;
    setStatus("Done  ");
  } catch (err) {
    console.error(err);
    setStatus("Write failed or rejected. Check console for details.");
  }
};
```

流程拆解（为什么这样写）

1. 读取输入框 mood，并进行非空验证（避免无意义交易、节省 gas）。
2. 调用 `write.setMood`：
   - 会弹 MetaMask

- 用户确认后广播交易
- 返回 **交易 hash**（此时还未确认）

3. 用 `waitForTransactionReceipt` 等待交易被打包确认（避免读到旧状态）。
4. 交易确认后再次调用 `getMood()`，并更新 UI 显示最新状态。

16 15) 可选：页面加载自动读取

```
// window.getMood();
```

- 取消注释后，页面一打开就会自动读取链上 mood 并显示。

17 总结：这份 dApp 的本质

一句话概括：

用 MetaMask 提供身份与签名，用 viem 把网页变成合约的遥控器：
`Get Mood` → 免费读取链上状态； `Set Mood` → 发送交易写入链上状态（需要 gas）。

附：建议的下一步（可选提升）

- 把 ABI 独立成 `abi.json`（更像真实项目结构）
- 把 JS 独立成 `app.js`（可维护性更好）
- 增加 loading/disable 按钮（等待确认时禁用重复点击）
- 合约加入 `event MoodChanged(address user, string mood)` 并在前端监听事件，实现自动更新