PLC 监控项目规划

一、项目概述

开发一个基于Web的PLC监控系统，支持多设备访问和实时监控。采用 Vue3 + Node.js + SQLite 架构，提供跨平台的访问能力。

二、核心功能

1. **基础监控**
    - PLC通信
        - 默认支持 Modbus TCP
        - 预留协议扩展接口
        - 支持协议动态加载
        - 动态增减监控PLC
        - 全局报警监控
    - 数字量输入输出监控
    - 模拟量输入输出监控
    - 实时数据显示
2. **独立部署**
    - 绿色版免安装
    - 内置Web服务器
    - 双击运行启动
    - 系统托盘运行
    - 自动检测端口
    - 支持配置导入导出
3. **配置管理**
    - Excel 模板导入配置
    - SQLite 数据库存储
    - 点位配置管理
    - 导出配置备份
    - PLC配置管理

4. **报警监控与处理**
    - 监控指定 PLC 地址
    - 实时显示报警信息
    - 自动匹配处理方案
    - 显示解决步骤指导
    - 用户反馈记录系统
    - 报警历史查询

三、技术架构

```
[应用架构]
Web服务器 (Node.js)
  ├── Vue3 + Vant UI (前端界面)
  ├── WebSocket (实时通信)
  ├── SQLite (数据存储)
  └── Modbus TCP (PLC通信)

[文件结构]
project/
├── server/           # Node.js后端服务
├── src/              # Vue前端代码
├── database/         # 数据库操作
└── templates/        # 配置模板
```

四、开发阶段

1. **第一阶段：基础功能**
    - Web服务器搭建
    - PLC 通信服务
    - 基础数据监控
    - 移动端界面适配
2. **第二阶段：配置管理**
    - SQLite 数据库集成
    - Excel 导入导出
    - 配置管理界面
3. **第三阶段：报警功能**

- 报警点位配置
- 报警状态监控
- 解决方案数据库建立
- 报警处理流程实现
- 用户反馈系统开发

4. **第四阶段：系统优化**
    - 性能优化
    - 界面美化
    - 稳定性测试

五、技术细节

PLC管理设计

```typescript
// PLC配置接口
interface PLCConfig {
    id: number;
    name: string;
    protocol?: string;    // 可选，默认为 'modbus-tcp'
    host: string;
    port?: number;        // 可选，使用协议默认端口
    enabled: boolean;     // 是否启用监控
    params?: {            // 协议特定参数
        [key: string]: any;
    };
}

// PLC管理器
class PLCManager {
    private static instance: PLCManager;
    private protocols: Map<string, ProtocolPlugin> = new Map();
    private plcInstances: Map<number, PLCProtocol> = new Map();
    private monitoringTasks: Map<number, NodeJS.Timer> = new Map();

    // 单例模式获取实例
    static getInstance(): PLCManager {
        if (!PLCManager.instance) {
            PLCManager.instance = new PLCManager();
        }
        return PLCManager.instance;
    }

    // 添加新的PLC
    async addPLC(config: PLCConfig): Promise<boolean> {
        try {
            const plc = this.createProtocol(config);
            this.plcInstances.set(config.id, plc);

            if (config.enabled) {
                await this.startMonitoring(config.id);
            }

            // 保存配置到数据库
            await this.savePLCConfig(config);

            return true;
        } catch (error) {
            console.error(`Failed to add PLC ${config.name}:`, error);
            return false;
        }
    }

    // 移除PLC
    async removePLC(plcId: number): Promise<boolean> {
```

```
        try {
            // 停止监控
            await this.stopMonitoring(plcId);

            // 断开连接
            const plc = this.plcInstances.get(plcId);
            if (plc) {
                await plc.disconnect();
            }

            // 移除实例
            this.plcInstances.delete(plcId);

            // 从数据库删除配置
            await this.deletePLCConfig(plcId);

            return true;
        } catch (error) {
            console.error(`Failed to remove PLC ${plcId}:`, error);
            return false;
        }
    }

    // 启用PLC监控
    async enablePLC(plcId: number): Promise<boolean> {
        const plc = this.plcInstances.get(plcId);
        if (!plc) return false;

        try {
            await this.startMonitoring(plcId);
            await this.updatePLCStatus(plcId, true);
            return true;
        } catch (error) {
            console.error(`Failed to enable PLC ${plcId}:`, error);
            return false;
        }
    }

    // 禁用PLC监控
    async disablePLC(plcId: number): Promise<boolean> {
        try {
            await this.stopMonitoring(plcId);
            await this.updatePLCStatus(plcId, false);
            return true;
        } catch (error) {
            console.error(`Failed to disable PLC ${plcId}:`, error);
            return false;
        }
    }

    // 开始监控特定PLC
```

```
    private async startMonitoring(plcId: number) {
        const plc = this.plcInstances.get(plcId);
        if (!plc || this.monitoringTasks.has(plcId)) return;

        try {
            await plc.connect();

            const task = setInterval(async () => {
                try {
                    // 读取配置的地址
                    const addresses = await this.getPLCAddresses(plcId);
                    const values = new Map();

                    for (const addr of addresses) {
                        const value = await plc.readPoint(addr);
                        values.set(addr, value);
                    }

                    // 更新UI
                    this.updateUI(plcId, values);

                    // 检查报警
                    await this.checkAlarms(plcId, values);

                } catch (error) {
                    console.error(`Error monitoring PLC ${plcId}:`, error);
                    // 尝试重新连接
                    await this.reconnect(plcId);
                }
            }, 1000); // 1秒更新一次

            this.monitoringTasks.set(plcId, task);

        } catch (error) {
            console.error(`Failed to start monitoring PLC ${plcId}:`, error);
            throw error;
        }
    }

    // 停止监控特定PLC
    private async stopMonitoring(plcId: number) {
        const task = this.monitoringTasks.get(plcId);
        if (task) {
            clearInterval(task);
            this.monitoringTasks.delete(plcId);
        }
    }

    // 重连机制
    private async reconnect(plcId: number) {
        const plc = this.plcInstances.get(plcId);
```

```
        if (!plc) return;

        try {
            await plc.disconnect();
            await plc.connect();
        } catch (error) {
            console.error(`Failed to reconnect PLC ${plcId}:`, error);
        }
    }
}

// UI接口
interface PLCManagerUI {
    // PLC列表管理
    showPLCList(): void;
    addPLCDialog(): void;
    editPLCDialog(plcId: number): void;
    confirmRemovePLC(plcId: number): void;

    // 状态显示
    updatePLCStatus(plcId: number, status: PLCStatus): void;
    showPLCData(plcId: number, values: Map<string, any>): void;
    showAlarms(plcId: number, alarms: Alarm[]): void;
}

// 使用示例
async function main() {
    const manager = PLCManager.getInstance();

    // 添加新PLC
    await manager.addPLC({
        id: 1,
        name: "注塑机1号",
        host: "192.168.1.101",
        enabled: true
    });

    // 添加另一个PLC但不立即启用
    await manager.addPLC({
        id: 2,
        name: "注塑机2号",
        host: "192.168.1.102",
        enabled: false
    });

    // 稍后启用PLC
    await manager.enablePLC(2);

    // 临时禁用PLC
    await manager.disablePLC(1);
```

```
    // 完全移除PLC
    await manager.removePLC(2);
}
```

界面设计
```typescript
// PLC选择器组件
interface PLCSelector {
    selectedPLCId: number | null;
    plcList: PLCConfig[];

    // 切换选中的PLC
    selectPLC(plcId: number): void;

    // 获取当前选中的PLC
    getSelectedPLC(): PLCConfig | null;
}

// 主界面布局
interface MainLayout {
    // 顶部工具栏
    header: {
        plcSelector: PLCSelector;     // PLC选择下拉框
        connectionStatus: boolean;     // 连接状态指示
        lastUpdateTime: Date;         // 最后更新时间
    };

    // 主显示区域
    content: {
        digitalInputs: PointGroup;    // 数字量输入
        digitalOutputs: PointGroup;   // 数字量输出
        analogInputs: PointGroup;     // 模拟量输入
        analogOutputs: PointGroup;    // 模拟量输出
        dataRegisters: PointGroup;    // 数据寄存器
    };

    // 报警区域
    alarmPanel: {
        activeAlarms: AlarmList;      // 活动报警
        alarmHistory: AlarmList;      // 历史报警
    };
}

// 监控点组件
interface PointGroup {
    points: PLCPoint[];
    layout: 'grid' | 'list';
    updateValue(address: string, value: any): void;
    updateStatus(address: string, status: PointStatus): void;
}
```

```
// PLC管理器扩展
class PLCManager {
    private currentPLCId: number | null = null;

    // 切换当前监控的PLC
    async switchPLC(plcId: number) {
        // 停止当前PLC的UI更新
        if (this.currentPLCId) {
            await this.stopUIUpdates(this.currentPLCId);
        }

        // 切换到新的PLC
        this.currentPLCId = plcId;

        // 开始新PLC的UI更新
        if (plcId) {
            await this.startUIUpdates(plcId);
        }

        // 触发UI更新事件
        this.emit('plcChanged', plcId);
    }

    // 更新UI显示
    private async startUIUpdates(plcId: number) {
        const plc = this.plcInstances.get(plcId);
        if (!plc) return;

        // 获取PLC点位配置
        const points = await this.getPLCPoints(plcId);

        // 更新界面布局
        this.updateUILayout(points);

        // 开始数据更新
        this.startDataUpdates(plcId, points);
    }

    // 停止UI更新
    private async stopUIUpdates(plcId: number) {
        // 停止数据更新
        this.stopDataUpdates(plcId);

        // 清除界面显示
        this.clearUIDisplay();
    }

    // 数据更新循环
    private startDataUpdates(plcId: number, points: PLCPoint[]) {
        const updateTask = setInterval(async () => {
            try {
```

```
                const plc = this.plcInstances.get(plcId);
                if (!plc) return;

                // 按类型分组读取数据
                for (const point of points) {
                    const value = await plc.readPoint(point.address);
                    this.updatePointValue(point.address, value);
                }

                // 更新最后更新时间
                this.updateLastUpdateTime();

            } catch (error) {
                console.error(`Error updating PLC ${plcId} data:`, error);
                this.updateConnectionStatus(false);
            }
        }, 1000); // 1秒更新一次

        this.monitoringTasks.set(plcId, updateTask);
    }
}

// Vue组件示例
```html
<!-- PLC选择器组件 -->
<template>
  <div class="plc-selector">
    <select v-model="selectedPLCId" @change="onPLCChange">
      <option v-for="plc in plcList" :key="plc.id" :value="plc.id">
        {{ plc.name }} ({{ plc.host }})
      </option>
    </select>

    <div class="status-indicator" :class="{ connected: isConnected }">
      {{ connectionStatus }}
    </div>

    <div class="last-update">
      最后更新: {{ formatTime(lastUpdateTime) }}
    </div>
  </div>
</template>

<!-- 监控点显示组件 -->
<template>
  <div class="point-group">
    <div class="group-header">
      <h3>{{ title }}</h3>
      <div class="layout-switch">
        <button @click="switchLayout('grid')" :class="{ active: layout === 'grid' }">
          网格
```
```

```
          </button>
          <button @click="switchLayout('list')" :class="{ active: layout === 'list' }">
            列表
          </button>
        </div>
      </div>

      <div class="points" :class="layout">
        <div v-for="point in points" :key="point.address" class="point-item">
          <div class="point-name">{{ point.description }}</div>
          <div class="point-address">{{ point.address }}</div>
          <div class="point-value" :class="point.status">
            {{ formatValue(point.value, point.type) }}
          </div>
        </div>
      </div>
    </div>
  </template>
```

样式设计

```css
/* PLC选择器样式 */
.plc-selector {
    display: flex;
    align-items: center;
    padding: 10px;
    background: #f5f5f5;
    border-bottom: 1px solid #ddd;
}

.status-indicator {
    width: 10px;
    height: 10px;
    border-radius: 50%;
    margin: 0 10px;
    background: #ff4444;
}

.status-indicator.connected {
    background: #00C851;
}

/* 监控点样式 */
.point-group {
    margin: 15px;
    border: 1px solid #ddd;
    border-radius: 4px;
}

.points.grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 10px;
    padding: 10px;
}

.points.list {
    display: flex;
    flex-direction: column;
}

.point-item {
    padding: 10px;
    background: white;
    border: 1px solid #eee;
    border-radius: 4px;
}

.point-value {
    font-size: 1.2em;
    font-weight: bold;
```

```
}

.point-value.normal { color: #00C851; }
.point-value.warning { color: #ffbb33; }
.point-value.alarm { color: #ff4444; }
```

全局报警监控设计

```typescript
// 全局报警管理器
class GlobalAlarmManager {
    private static instance: GlobalAlarmManager;
    private alarms: Map<number, Set<Alarm>> = new Map();  // plcId -> alarms
    private listeners: Set<(plcId: number, alarm: Alarm) => void> = new Set();

    // 添加新报警
    async addAlarm(plcId: number, alarm: Alarm) {
        if (!this.alarms.has(plcId)) {
            this.alarms.set(plcId, new Set());
        }
        this.alarms.get(plcId)!.add(alarm);

        // 通知所有监听器
        this.notifyListeners(plcId, alarm);

        // 保存到数据库
        await this.saveAlarm(alarm);
    }

    // 获取指定PLC的报警数量
    getAlarmCount(plcId: number): number {
        return this.alarms.get(plcId)?.size || 0;
    }

    // 获取所有活动报警
    getAllActiveAlarms(): Map<number, Alarm[]> {
        const result = new Map();
        for (const [plcId, alarms] of this.alarms) {
            result.set(plcId, Array.from(alarms));
        }
        return result;
    }

    // 清除报警
    async clearAlarm(plcId: number, alarmId: number) {
        const alarms = this.alarms.get(plcId);
        if (!alarms) return;

        const alarm = Array.from(alarms).find(a => a.id === alarmId);
        if (alarm) {
            alarms.delete(alarm);
            await this.updateAlarmStatus(alarmId, 'cleared');
        }
    }
}

// PLC管理器扩展
class PLCManager {
    private alarmManager: GlobalAlarmManager;
```

```
    // 监控所有PLC的报警
    private monitorAllPLCAlarms() {
        // 遍历所有PLC实例
        for (const [plcId, plc] of this.plcInstances) {
            this.startAlarmMonitoring(plcId);
        }
    }

    // 监控单个PLC的报警
    private startAlarmMonitoring(plcId: number) {
        const plc = this.plcInstances.get(plcId);
        if (!plc) return;

        const task = setInterval(async () => {
            try {
                // 获取报警点位
                const alarmPoints = await this.getAlarmPoints(plcId);

                // 检查每个报警点
                for (const point of alarmPoints) {
                    const value = await plc.readPoint(point.address);

                    // 检查是否触发报警
                    if (this.checkAlarmCondition(point, value)) {
                        await this.alarmManager.addAlarm(plcId, {
                            id: Date.now(),
                            pointId: point.id,
                            plcId: plcId,
                            value: value,
                            timestamp: new Date(),
                            description: point.description,
                            status: 'active'
                        });
                    }
                }
            } catch (error) {
                console.error(`Error monitoring alarms for PLC ${plcId}:`, error);
            }
        }, 1000); // 每秒检查一次

        this.monitoringTasks.set(`alarm-${plcId}`, task);
    }
}

// Vue组件示例
```html
<!-- 报警状态栏组件 -->
<template>
  <div class="alarm-status-bar">
    <div v-for="[plcId, alarmInfo] in plcAlarms"
```
```

```
          :key="plcId"
          class="plc-alarm-status"
          :class="{ 'has-alarm': alarmInfo.count > 0 }"
          @click="onPLCClick(plcId)">
        <div class="plc-info">
          <span class="plc-name">{{ alarmInfo.name }}</span>
          <span v-if="alarmInfo.count > 0" class="alarm-count">{{ alarmInfo.count }}
</span>
        </div>
        <div v-if="alarmInfo.count > 0" class="latest-alarm">
          {{ alarmInfo.latestAlarm }}
        </div>
      </div>
    </div>
</template>
```

报警样式

```css
/* 报警状态栏 */
.alarm-status-bar {
    position: fixed;
    bottom: 0;
    left: 0;
    right: 0;
    display: flex;
    flex-wrap: wrap;
    background: #f8f9fa;
    border-top: 1px solid #ddd;
    padding: 5px;
    gap: 5px;
}

.plc-alarm-status {
    flex: 1;
    min-width: 150px;
    max-width: 300px;
    padding: 8px;
    border-radius: 4px;
    background: #fff;
    border: 1px solid #ddd;
    cursor: pointer;
    transition: all 0.3s ease;
}

.plc-info {
    display: flex;
    justify-content: space-between;
    align-items: center;
}

.plc-name {
    font-weight: 500;
}

.latest-alarm {
    font-size: 0.9em;
    color: #666;
    margin-top: 4px;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
}

.plc-alarm-status.has-alarm {
    background: #fff0f0;
    border-color: #ff4444;
    animation: pulse 2s infinite;
}
```

```css
.alarm-count {
    background: #ff4444;
    color: white;
    padding: 2px 6px;
    border-radius: 10px;
    font-size: 0.9em;
    font-weight: bold;
}

/* 报警动画 */
@keyframes pulse {
    0% { box-shadow: 0 0 0 0 rgba(255, 68, 68, 0.4); }
    70% { box-shadow: 0 0 0 6px rgba(255, 68, 68, 0); }
    100% { box-shadow: 0 0 0 0 rgba(255, 68, 68, 0); }
}

/* 移动端适配 */
@media (max-width: 768px) {
    .alarm-status-bar {
        padding: 3px;
    }

    .plc-alarm-status {
        min-width: 120px;
        padding: 6px;
    }

    .latest-alarm {
        display: none;
    }
}
```

{{ ... }}