智能车间生产监控管理系统源程序

```
/**
 * 生产监控系统首页
 *
 * @author 骆大利
 * @date  2024-10-10
 */
using KeLongLed.Model;
using KeLongLed.Service;
using KeLongLed.UI.Factories;
using KeLongLed.UI.Plan;
using KeLongLed.UI.Utils;
using Sunny.UI;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace KeLongLed
{
    public partial class Form1 : Form
    {

        private FlowLayoutPanel flowPanel;
        private Timer refreshTimer;
        private DataService dataService;

        public Form1()
        {
            InitializeComponent();
            InitializeUI();
            InitializeServices();
            InitializeTimer();
            this.BackColor = Color.Black;
            this.FormClosing += Form1_FormClosing;
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // 停止并释放定时器资源
            if (refreshTimer != null)
            {
                refreshTimer.Stop();
                refreshTimer.Dispose();
```

```
                refreshTimer = null; //  确保定时器被回收
            }
        }

        private void InitializeServices()
        {
            dataService = new DataService();
        }

        private void InitializeUI()
        {
            //  设置窗体属性
            this.Text = "智能车间生产进度监控管理软件";
            this.Size = new Size(800, 630);

            //  创建  FlowLayoutPanel
            flowPanel = new FlowLayoutPanel
            {
                Dock = DockStyle.Fill,
                AutoScroll = true,
                WrapContents = true,
                FlowDirection = FlowDirection.LeftToRight,
                Padding = new Padding(10,30,10,30)

            };
            this.Controls.Add(flowPanel);

        }

        private async Task RefreshDataAsync() {
            try
            {
                //  获取最新数据
                List<LedData> ledDataList = await Task.Run(() =>
dataService.GetLatestLedData());

                flowPanel.Invoke(new Action(() =>
                {
                    if (flowPanel.Controls.Count == 0)
                    {
                        foreach (var data in ledDataList)
                        {
                            Panel panel = PanelFactory.CreateDataPanel(data);
                            flowPanel.Controls.Add(panel);
                        }
```

```
                    }
                    else
                    {
                        for (int i = 0; i < ledDataList.Count; i++)
                        {
                            if (i < flowPanel.Controls.Count)
                            {

PanelFactory.UpdatePanelData((Panel)flowPanel.Controls[i], ledDataList[i]);
                            }
                            else
                            {
                                Panel panel =
PanelFactory.CreateDataPanel(ledDataList[i]);
                                flowPanel.Controls.Add(panel);
                            }
                        }

                        // 移除多余的面板
                        while (flowPanel.Controls.Count > ledDataList.Count)
                        {
                            flowPanel.Controls.RemoveAt(flowPanel.Controls.Count - 1);
                        }
                    }
                }));
            }
            catch (Exception ex)
            {
                MessageBox.Show($"数据刷新失败：{ex.Message}");
            }

        }


        private void InitializeTimer()
        {
            refreshTimer = new Timer
            {
                Interval = 5000 // 每 5 秒刷新一次
            };
            refreshTimer.Tick += async (s, e) => await RefreshDataAsync();
            refreshTimer.Start();
            // 手动触发一次数据刷新，确保在定时器第一次触发前加载数据
            _ = RefreshDataAsync();
        }
```

```
private void 关于我们 ToolStripMenuItem1_Click(object sender, EventArgs e)
{
    UIStyle.Gray, UIMessageBoxButtons.OK, false);
     AboutForm aboutForm = new AboutForm();
     aboutForm.Owner = this;
     aboutForm.Show();
}

//导入 Excel 事件
private void ImportPlan_Click(object sender, EventArgs e)
{
     using (OpenFileDialog openFileDialog = new OpenFileDialog())
     {
          openFileDialog.Filter = "Excel Files|*.xls;*.xlsx";
          if (openFileDialog.ShowDialog() == DialogResult.OK)
          {
               string filePath = openFileDialog.FileName;
               ImportDataService service = new ImportDataService();
               service.ImportExcelData(filePath);
          }
     }
}

private void 退出 ToolStripMenuItem_Click(object sender, EventArgs e)
{
     // 停止并释放定时器资源
     if (refreshTimer != null)
     {
          refreshTimer.Stop();
          refreshTimer.Dispose();
          refreshTimer = null; // 确保定时器被回收
     }
     this.Close();
}

private void PlanList_Click(object sender, EventArgs e)
{
     PlanListGroupForm planListForm = new PlanListGroupForm();
     planListForm.Owner = this;
     planListForm.Show(); // 弹出新窗口显示计划列表
}

private void DownLoadTemplate_Click(object sender, EventArgs e)
{
```

```
                SaveFileDialog saveFileDialog = new SaveFileDialog
                {
                    Filter = "Excel Files|*.xls",
                    FileName = "plan.xlsx"
                };

                if (saveFileDialog.ShowDialog() == DialogResult.OK)
                {
                    string sourceFilePath = AppDomain.CurrentDomain.BaseDirectory +
"plan.xlsx";
                    string destinationFilePath = saveFileDialog.FileName;

                    try
                    {
                        File.Copy(sourceFilePath, destinationFilePath, overwrite: true);
                        MessageBox.Show("模板已成功下载！");
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("下载模板失败: " + ex.Message);
                    }
                }
            }
        }
}
/**
 * 生产进度监控系统 Oracle 数据库操作类 *
 * @author 郑迎俊
 * @date 2024-10-10
 */
using Oracle.ManagedDataAccess.Client;
using System;
using System.Collections;
using System.Collections.Generic;
using System.Configuration;
using System.Data;


namespace Helper
{
    /// <summary>
    /// Oracle 数据库操作类
    /// </summary>
    public static class OracleHelper
    {
```

```
        //从配置文件中读取配置好的连接字符串
        public static readonly string ConnectionString_Default =
ConfigurationManager.ConnectionStrings["KlOracleConnString"].ConnectionString;

        public static string ExecuteSQL(string SQLString)
        {
            try
            {
                ExecuteNonQuery(ConnectionString_Default, SQLString);
                return "success";
            }
            catch (Exception ex)
            {
                return "fail." + ex.Message;
            }
        }

        public static string ExecuteSQL(ArrayList SQLStringArrayList)
        {
            try
            {
                ExecuteNonQuery(ConnectionString_Default, SQLStringArrayList);
                return "success";
            }
            catch (Exception ex)
            {
                return "fail." + ex.Message;
            }
        }

        public static string ExecuteSQL(List<string> SQLStringList)
        {
            try
            {
                ExecuteNonQuery(ConnectionString_Default, SQLStringList);
                return "success";
            }
            catch (Exception ex)
            {
                return "fail." + ex.Message;
            }
        }

        public static int ExecuteNonQuery(string SQLString)
```

```
        {
            return ExecuteNonQuery(ConnectionString_Default, SQLString);
        }

        public static void ExecuteNonQuery(ArrayList SQLStringList)
        {
            ExecuteNonQuery(ConnectionString_Default, SQLStringList);
        }

        public static DataTable ExecuteDataTable(string SQLString)
        {
            return ExecuteDataTable(ConnectionString_Default, CommandType.Text,
SQLString, null);
        }

        public static DataTable ExecuteDataTable(string SQLString, params OracleParameter[]
cmdParms)
        {
            return ExecuteDataTable(ConnectionString_Default, CommandType.Text,
SQLString, cmdParms);
        }


        public static object ExecuteScalar(string SQLString)
        {
            return ExecuteScalar(ConnectionString_Default, CommandType.Text, SQLString,
null);
        }

        public static OracleDataReader ExecuteDataReader(string SQLString) {

            return ExecuteReader(ConnectionString_Default, CommandType.Text, SQLString,
null);
        }

        /// <summary>
        /// 执行 SQL 语句，返回影响的记录数
        /// </summary>
        /// <param name="SQLString">SQL 语句</param>
        /// <returns>影响的记录数</returns>
        private static int ExecuteNonQuery(string connectionString, string SQLString)
        {
            using (OracleConnection connection = new OracleConnection(connectionString))
            {
```

```
                using (OracleCommand cmd = new OracleCommand(SQLString,
connection))
                {
                    try
                    {
                        connection.Open();
                        int rows = cmd.ExecuteNonQuery();
                        connection.Close();
                        return rows;
                    }
                    catch (OracleException E)
                    {
                        connection.Close();
                        throw new Exception(E.Message);
                    }
                }
            }
        }

        private static void ExecuteNonQuery(string connectionString, ArrayList
SQLStringArrayList)
        {
            using (OracleConnection conn = new OracleConnection(connectionString))
            {
                conn.Open();
                OracleCommand cmd = new OracleCommand();
                cmd.Connection = conn;
                OracleTransaction tx = conn.BeginTransaction();
                cmd.Transaction = tx;
                try
                {
                    for (int n = 0; n < SQLStringArrayList.Count; n++)
                    {
                        string strsql = SQLStringArrayList[n].ToString();
                        if (strsql.Trim().Length > 1)
                        {
                            cmd.CommandText = strsql;
                            cmd.ExecuteNonQuery();
                        }
                    }
                    tx.Commit();
                }
                catch (OracleException E)
                {
                    tx.Rollback();
```

```
                throw new Exception(E.Message);
            }
        }
    }

    private static void ExecuteNonQuery(string connectionString, List<string>
SQLStringList)
    {
        using (OracleConnection conn = new OracleConnection(connectionString))
        {
            conn.Open();
            OracleCommand cmd = new OracleCommand();
            cmd.Connection = conn;
            OracleTransaction tx = conn.BeginTransaction();
            cmd.Transaction = tx;
            try
            {
                for (int n = 0; n < SQLStringList.Count; n++)
                {
                    string strsql = SQLStringList[n].ToString();
                    if (strsql.Trim().Length > 1)
                    {
                        cmd.CommandText = strsql;
                        cmd.ExecuteNonQuery();
                    }
                }
                tx.Commit();
            }
            catch (OracleException E)
            {
                tx.Rollback();
                throw new Exception(E.Message);
            }
        }
    }

    /// <summary>
    /// 执行数据库非查询操作,返回受影响的行数
    /// </summary>
    /// <param name="connectionString">数据库连接字符串</param>
    /// <param name="cmdType">命令的类型</param>
    /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
    /// <param name="cmdParms">命令参数集合</param>
    /// <returns>当前操作影响的数据行数</returns>
```

```
        private static int ExecuteNonQuery(string connectionString, CommandType cmdType,
string cmdText, params OracleParameter[] cmdParms)
        {
            using (OracleCommand cmd = new OracleCommand())
            {
                using (OracleConnection conn = new OracleConnection(connectionString))
                {
                    PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                    int val = cmd.ExecuteNonQuery();
                    cmd.Parameters.Clear();
                    return val;
                }
            }
        }

        /// <summary>
        /// 执行数据库事务非查询操作,返回受影响的行数
        /// </summary>
        /// <param name="transaction">数据库事务对象</param>
        /// <param name="cmdType">Command 类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前事务操作影响的数据行数</returns>
        private static int ExecuteNonQuery(OracleTransaction trans, CommandType cmdType,
string cmdText, params OracleParameter[] cmdParms)
        {
            OracleCommand cmd = new OracleCommand();
            PrepareCommand(cmd, trans.Connection, trans, cmdType, cmdText, cmdParms);
            int val = cmd.ExecuteNonQuery();
            cmd.Parameters.Clear();
            return val;
        }

        /// <summary>
        /// 执行数据库非查询操作,返回受影响的行数
        /// </summary>
        /// <param name="connection">Oracle 数据库连接对象</param>
        /// <param name="cmdType">Command 类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前操作影响的数据行数</returns>
        private static int ExecuteNonQuery(OracleConnection connection, CommandType
cmdType, string cmdText, params OracleParameter[] cmdParms)
        {
            if (connection == null)
```

```
                throw new ArgumentNullException("当前数据库连接不存在");
            OracleCommand cmd = new OracleCommand();
            PrepareCommand(cmd, connection, null, cmdType, cmdText, cmdParms);
            int val = cmd.ExecuteNonQuery();
            cmd.Parameters.Clear();
            return val;
        }

        /// <summary>
        /// 执行数据库查询操作,返回 OracleDataReader 类型的内存结果集
        /// </summary>
        /// <param name="connectionString">数据库连接字符串</param>
        /// <param name="cmdType">命令的类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前查询操作返回的 OracleDataReader 类型的内存结果集</returns>
        private static OracleDataReader ExecuteReader(string connectionString, CommandType
cmdType, string cmdText, params OracleParameter[] cmdParms)
        {
            OracleCommand cmd = new OracleCommand();
            OracleConnection conn = new OracleConnection(connectionString);
            try
            {
                PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                OracleDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
                cmd.Parameters.Clear();
                return reader;
            }
            catch
            {
                cmd.Dispose();
                conn.Close();
                throw;
            }
        }

        /// <summary>
        /// 执行数据库查询操作,返回 DataSet 类型的结果集
        /// </summary>
        /// <param name="connectionString">数据库连接字符串</param>
        /// <param name="cmdType">命令的类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前查询操作返回的 DataSet 类型的结果集</returns>
```

```
        private static DataSet ExecuteDataSet(string connectionString, CommandType cmdType,
string cmdText, params OracleParameter[] cmdParms)
        {
            OracleCommand cmd = new OracleCommand();
            OracleConnection conn = new OracleConnection(connectionString);
            DataSet ds = null;
            try
            {
                PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                OracleDataAdapter adapter = new OracleDataAdapter();
                adapter.SelectCommand = cmd;
                ds = new DataSet();
                adapter.Fill(ds);
                cmd.Parameters.Clear();
            }
            catch
            {
                throw;
            }
            finally
            {
                cmd.Dispose();
                conn.Close();
                conn.Dispose();
            }

            return ds;
        }

        /// <summary>
        ///  执行数据库查询操作,返回 DataTable 类型的结果集
        /// </summary>
        /// <param name="connectionString">数据库连接字符串</param>
        /// <param name="cmdType">命令的类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前查询操作返回的 DataTable 类型的结果集</returns>
        private static DataTable ExecuteDataTable(string connectionString, CommandType
cmdType, string cmdText, params OracleParameter[] cmdParms)
        {
            OracleCommand cmd = new OracleCommand();
            OracleConnection conn = new OracleConnection(connectionString);
            DataTable dt = null;
            try
            {
```

```
                PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                OracleDataAdapter adapter = new OracleDataAdapter();
                adapter.SelectCommand = cmd;
                dt = new DataTable();
                adapter.Fill(dt);
                cmd.Parameters.Clear();
            }
            catch
            {
                throw;
            }
            finally
            {
                cmd.Dispose();
                conn.Close();
                conn.Dispose();
            }
            return dt;
        }

        /// <summary>
        ///  执行数据库查询操作,返回结果集中位于第一行第一列的 Object 类型的值
        /// </summary>
        /// <param name="connectionString">数据库连接字符串</param>
        /// <param name="cmdType">命令的类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前查询操作返回的结果集中位于第一行第一列的 Object 类型的值
</returns>
        private static object ExecuteScalar(string connectionString, CommandType cmdType,
string cmdText, params OracleParameter[] cmdParms)
        {
            OracleCommand cmd = new OracleCommand();
            OracleConnection conn = new OracleConnection(connectionString);
            object result = null;
            try
            {
                PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                result = cmd.ExecuteScalar();
                cmd.Parameters.Clear();
            }
            catch
            {
                throw;
            }
```

```
            finally
            {
                cmd.Dispose();
                conn.Close();
                conn.Dispose();
            }

            return result;
        }

        /// <summary>
        /// 执行数据库事务查询操作,返回结果集中位于第一行第一列的 Object 类型的值
        /// </summary>
        /// <param name="trans">一个已存在的数据库事务对象</param>
        /// <param name="commandType">命令类型</param>
        /// <param name="commandText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前事务查询操作返回的结果集中位于第一行第一列的 Object 类型的值</returns>
        private static object ExecuteScalar(OracleTransaction trans, CommandType cmdType, string cmdText, params OracleParameter[] cmdParms)
        {
            if (trans == null)
                throw new ArgumentNullException("当前数据库事务不存在");
            OracleConnection conn = trans.Connection;
            if (conn == null)
                throw new ArgumentException("当前事务所在的数据库连接不存在");

            OracleCommand cmd = new OracleCommand();
            object result = null;

            try
            {
                PrepareCommand(cmd, conn, trans, cmdType, cmdText, cmdParms);
                result = cmd.ExecuteScalar();
                cmd.Parameters.Clear();
            }
            catch
            {
                throw;
            }
            finally
            {
                trans.Dispose();
                cmd.Dispose();
```

```
                conn.Close();
                conn.Dispose();
            }

            return result;
        }

        /// <summary>
        /// 执行数据库查询操作,返回结果集中位于第一行第一列的 Object 类型的值
        /// </summary>
        /// <param name="conn">数据库连接对象</param>
        /// <param name="cmdType">Command 类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        /// <returns>当前查询操作返回的结果集中位于第一行第一列的 Object 类型的值
</returns>
        private static object ExecuteScalar(OracleConnection conn, CommandType cmdType,
string cmdText, params OracleParameter[] cmdParms)
        {
            if (conn == null) throw new ArgumentException("当前数据库连接不存在");
            OracleCommand cmd = new OracleCommand();
            object result = null;

            try
            {
                PrepareCommand(cmd, conn, null, cmdType, cmdText, cmdParms);
                result = cmd.ExecuteScalar();
                cmd.Parameters.Clear();
            }
            catch
            {
                throw;
            }
            finally
            {
                cmd.Dispose();
                conn.Close();
                conn.Dispose();
            }

            return result;
        }

        /// <summary>
        /// 执行数据库命令前的准备工作
```

```
        /// </summary>
        /// <param name="cmd">Command 对象</param>
        /// <param name="conn">数据库连接对象</param>
        /// <param name="trans">事务对象</param>
        /// <param name="cmdType">Command 类型</param>
        /// <param name="cmdText">Oracle 存储过程名称或 PL/SQL 命令</param>
        /// <param name="cmdParms">命令参数集合</param>
        private static void PrepareCommand(OracleCommand cmd, OracleConnection conn,
OracleTransaction trans, CommandType cmdType, string cmdText, OracleParameter[] cmdParms)
        {
            if (conn.State != ConnectionState.Open)
                conn.Open();

            cmd.Connection = conn;
            cmd.CommandText = cmdText;

            if (trans != null)
                cmd.Transaction = trans;

            cmd.CommandType = cmdType;

            if (cmdParms != null)
            {
                foreach (OracleParameter parm in cmdParms)
                    cmd.Parameters.Add(parm);
            }
        }

        /// <summary>
        /// 将.NET 日期时间类型转化为 Oracle 兼容的日期时间格式字符串
        /// </summary>
        /// <param name="date">.NET 日期时间类型对象</param>
        /// <returns>Oracle 兼容的日期时间格式字符串（如该字符串：
TO_DATE('2007-12-1','YYYY-MM-DD')）</returns>
        private static string GetOracleDateFormat(DateTime date)
        {
            return "TO_DATE('" + date.ToString("yyyy-M-dd") + "','YYYY-MM-DD')";
        }

        /// <summary>
        /// 将.NET 日期时间类型转化为 Oracle 兼容的日期格式字符串
        /// </summary>
        /// <param name="date">.NET 日期时间类型对象</param>
        /// <param name="format">Oracle 日期时间类型格式化限定符</param>
```

```
        /// <returns>Oracle 兼容的日期时间格式字符串（如该字符串：
TO_DATE('2007-12-1','YYYY-MM-DD')）</returns>
        private static string GetOracleDateFormat(DateTime date, string format)
        {
            if (format == null || format.Trim() == "") format = "YYYY-MM-DD";
            return "TO_DATE('" + date.ToString("yyyy-M-dd") + "','" + format + "')";
        }

        /// <summary>
        /// 将指定的关键字处理为模糊查询时的合法参数值
        /// </summary>
        /// <param name="source">待处理的查询关键字</param>
        /// <returns>过滤后的查询关键字</returns>
        private static string HandleLikeKey(string source)
        {
            if (source == null || source.Trim() == "") return null;

            source = source.Replace("[", "[]]");
            source = source.Replace("_", "[_]");
            source = source.Replace("%", "[%]");

            return ("%" + source + "%");
        }

        /// <summary>
        /// 执行存储过程
        /// </summary>
        /// <param name="connection">SqlServer 数据库连接对象</param>
        /// <param name="storedProcName">存储过程名</param>
        /// <param name="parameters">存储过程参数</param>
        /// <returns>SqlDataReader 对象</returns>
        private static OracleDataReader RunStoredProcedure(OracleConnection connection,
string storedProcName, IDataParameter[] parameters)
        {
            OracleDataReader returnReader = null;
            connection.Open();
            OracleCommand command = BuildSqlCommand(connection, storedProcName,
parameters);
            returnReader = command.ExecuteReader(CommandBehavior.CloseConnection);
            return returnReader;
        }

        /// <summary>
        /// 构建 SqlCommand 对象
        /// </summary>
```

```
        /// <param name="connection">数据库连接</param>
        /// <param name="storedProcName">存储过程名</param>
        /// <param name="parameters">存储过程参数</param>
        /// <returns>SqlCommand</returns>
        private static OracleCommand BuildSqlCommand(OracleConnection connection, string
storedProcName, IDataParameter[] parameters)
        {
            OracleCommand command = new OracleCommand(storedProcName, connection);
            command.CommandType = CommandType.StoredProcedure;
            foreach (OracleParameter parameter in parameters)
            {
                command.Parameters.Add(parameter);
            }
            return command;
        }
    }

}

/**
 * 生产进度监控系统面板工厂类
 *
 * @author 郑迎俊
 * @date  2024-09-10
 */
using System;
using System.Drawing;
using System.Windows.Forms;
using KeLongLed.Model;
using KeLongLed.Service;
using Sunny.UI;

namespace KeLongLed.UI.Factories
{
    public static class PanelFactory
    {
        public static Panel CreateDataPanel(LedData data)
        {
            Panel panel = new Panel
            {
                Width = 200,
                Height = 120,
                Margin = new Padding(10),
                BackColor = Color.FromArgb(50, 50, 50),
                BorderStyle = BorderStyle.FixedSingle
```

```
};

Label lblDeviceName = new Label
{
    Text = data.Name,
    Dock = DockStyle.Top,
    TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Arial Black", 15, FontStyle.Bold),
    ForeColor = Color.DimGray,
    Height = 40
};
/**
Label lblPlanQuantity = new Label
{
    Text = $"计划数量：{data.PlannedQuantity}",
    Dock = DockStyle.Top,
    TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
    ForeColor = Color.Lime,
    Height = 40
};


Label lblActualQuantity = new Label
{
    Text = $"实际数量：{data.ActualQuantity}",
    Dock = DockStyle.Top,
    TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
    ForeColor = Color.Red,
    Height = 40
};
**/
UILedDisplay lblPlanQuantity = new UILedDisplay
{
    Text = $"{data.PlannedQuantity}",
    Dock = DockStyle.Top,
    //TextAlign = ContentAlignment.MiddleCenter,
    Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
    ForeColor = Color.Lime,
    Height = 40
};


UILedDisplay lblActualQuantity = new UILedDisplay
```

```
        {
            Text = $"{data.ActualQuantity}",
            Dock = DockStyle.Top,
            //TextAlign = ContentAlignment.MiddleCenter,
            Font = new Font("Microsoft Sans Serif", 10, FontStyle.Bold),
            ForeColor = Color.AliceBlue,
            Height = 40
        };
        panel.Controls.Add(lblActualQuantity);
        panel.Controls.Add(lblPlanQuantity);
        panel.Controls.Add(lblDeviceName);


        // 创建右键菜单
        ContextMenuStrip contextMenu = new ContextMenuStrip();
        ToolStripMenuItem sendPlanMenuItem = new ToolStripMenuItem("下发计划");
        sendPlanMenuItem.Click += (sender, e) =>
        {
            // 点击菜单项时弹出输入计划数量的窗体
            ShowPlanInputForm(data, panel);
        };
        contextMenu.Items.Add(sendPlanMenuItem);
        panel.ContextMenuStrip = contextMenu;

        return panel;
    }

    // 弹出输入计划数量的窗体
    private static void ShowPlanInputForm(LedData data, Panel panel)
    {
        Form inputForm = new Form
        {
            Text = $"向编号{data.LedId}屏幕下发计划",
            Width = 300,
            Height = 200
        };

        Label lblInputPrompt = new Label
        {
            Text = "请输入计划数量:",
            Location = new Point(20, 20),
            Width = 260
        };

        TextBox txtPlanQuantity = new TextBox
```

```
            {
                Location = new Point(20, 60),
                Width = 260,
                Text = GetLatestPlanQuantityFromPanel(panel).ToString()
            };

            Button btnSubmit = new Button
            {
                Text = "下发计划",
                Location = new Point(20, 100),
                Width = 260
            };

            btnSubmit.Click += (sender, e) =>
            {
                // 提交计划数量并更新面板
                //将实际数量也加入 if 判断

                if (int.TryParse(txtPlanQuantity.Text, out int plannedQuantity))
                {
                    data.PlannedQuantity = plannedQuantity;
                    // 创建 ImportDataService 实例
                    ImportDataService importDataService = new ImportDataService();
                    // 调用插入和下发单个数据的方法
                    importDataService.InsertSingleDataIntoDatabase(data.LedId,
data.PlannedQuantity);


                    //插入一条实时数据
                    LedDataInfoService ledDataInfoService = new LedDataInfoService();
                    ledDataInfoService.InsertSingleData(new LedDataInfo
                    {
                        LedId = data.LedId,
                        LedPlanData = plannedQuantity,
                        LedRealData = GetLatestRealQuantityFromPanel(panel)
                    });
                    UpdatePanelData(panel, data);  // 更新面板上的数据
                    inputForm.Close();  // 关闭输入窗体
                }
                else
                {
                    MessageBox.Show("请输入有效的计划数量！");
                }
            };
```

```
            inputForm.Controls.Add(lblInputPrompt);
            inputForm.Controls.Add(txtPlanQuantity);
            inputForm.Controls.Add(btnSubmit);
            // 获取鼠标位置并设置窗体位置
            Point mousePos = Control.MousePosition;
            inputForm.StartPosition = FormStartPosition.Manual;
            inputForm.Location = new Point(mousePos.X - inputForm.Width / 2, mousePos.Y
- inputForm.Height - 10);
            inputForm.ShowDialog();  // 显示窗体
        }


        public static void UpdatePanelData(Panel panel, LedData data)
        {
            Label lblDeviceName = panel.Controls[2] as Label;
            //Label lblPlanQuantity = panel.Controls[1] as Label;
            //Label lblActualQuantity = panel.Controls[0] as Label;
            UILedDisplay lblPlanQuantity = panel.Controls[1] as UILedDisplay;
            UILedDisplay lblActualQuantity = panel.Controls[0] as UILedDisplay;


            lblDeviceName.Text = data.Name;
            lblPlanQuantity.Text = $"{data.PlannedQuantity}";
            lblActualQuantity.Text = $"{data.ActualQuantity}";
        }


        // 新增辅助方法，用于从面板中动态获取最新计划数量
        private static int GetLatestPlanQuantityFromPanel(Panel panel)
        {
            if (panel.Controls[1] is UILedDisplay lblPlanQuantity &&
int.TryParse(lblPlanQuantity.Text, out int latestQuantity))
            {
                return latestQuantity;
            }
            return 0; // 如果解析失败，返回默认值
        }

        private static int GetLatestRealQuantityFromPanel(Panel panel)
        {
            if (panel.Controls[0] is UILedDisplay lblRealQuantity &&
int.TryParse(lblRealQuantity.Text, out int latestRealQuantity))
            {
                return latestRealQuantity;
            }
```

```
                return 0; // 如果解析失败，返回默认值
            }
        }

}
/**
 * 生产进度监控系统计划管理页面 *
 * @author 骆大利
 * @date 2024-09-15
 */
using KeLongLed.Model;
using KeLongLed.Service;
using Sunny.UI;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
namespace KeLongLed.UI.Plan
{
    public partial class PlanListGroupForm : Form
    {
        private DataGridView dataGridView;
        //private Button btnPrevious, btnNext;
        private UISymbolButton btnPrevious, btnNext, btnSend;
        private int currentPage = 1;
        private int pageSize = int.Parse(ConfigurationManager.AppSettings["PageSize"]);
        private PlanDataService planDataService;
        public PlanListGroupForm()
        {
            InitializeComponent();
            InitializeUI();
            planDataService = new PlanDataService();
            LoadPlanDataGrouped();
        }
        private void DataGridView_CellDoubleClick(object sender, DataGridViewCellEvent
Args e)
        {
            if (e.RowIndex >= 0) // 确保不是表头
            {
                // 获取选中行的数据
                var selectedGroup = (PlanDataGroup)dataGridView.Rows[e.RowIndex].Dat
aBoundItem;
                // 打开详细数据窗体
```

```
            PlanListForm detailsForm = new PlanListForm(selectedGroup.PlanDate);
            detailsForm.Owner = this;
            detailsForm.Show();
        }
    }
    private void InitializeUI()
    {
        this.Text = "计划列表";
        this.Size = new Size(800, 600);
        // DataGridView 控件
        dataGridView = new DataGridView
        {
            Dock = DockStyle.Top,
            AutoGenerateColumns = true,
            Height = 400,
            AllowUserToAddRows = false,  // 禁止用户添加行
            AllowUserToDeleteRows = false,  // 禁止用户删除行
            ReadOnly = true,  // 设置为只读
            AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill
        };
        this.Controls.Add(dataGridView);
        // 添加双击事件
        dataGridView.CellDoubleClick += DataGridView_CellDoubleClick;
        // 添加下发按钮
        Panel buttonPanel = new Panel
        {
            Dock = DockStyle.Bottom,
            Height = 90, // 你可以根据需要调整按钮区域的高度
            Padding = new Padding(180,0,180,20)
        };
        this.Controls.Add(buttonPanel);
        btnSend = new UISymbolButton
        {
            Symbol = 61528, // 可以选择合适的符号
            Text = "下发计划到设备",
            Dock = DockStyle.Fill
        };
        btnSend.Click += BtnSend_Click;
        buttonPanel.Controls.Add(btnSend);
        // 分页控制
        Panel pagingPanel = new Panel
        {
            Dock = DockStyle.Bottom,
            Height = 40,
            Padding = new Padding(10,0,10,10)
```

```
            };

            btnPrevious = new UISymbolButton
            {
                Symbol=61514,
                //Text = "上一页",
                Dock = DockStyle.Left,
            };
            btnPrevious.Click += (s, e) => ChangePage(-1);

            btnNext = new UISymbolButton
            {
                Symbol = 61518,
                //Text = "下一页",
                Dock = DockStyle.Right,
            };
            btnNext.Click += (s, e) => ChangePage(1);

            pagingPanel.Controls.Add(btnPrevious);
            pagingPanel.Controls.Add(btnNext);
            this.Controls.Add(pagingPanel);
        }
        private void BtnSend_Click(object sender, EventArgs e)
        {
            // 弹出确认框，询问用户是否确认下发
            DialogResult result = MessageBox.Show("确定要下发当天未下发的计划到设
备吗？", "确认下发", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            // 如果用户点击了“是”
            if (result == DialogResult.Yes)
            {
                // 获取当天所有未下发的计划
                PlanDataService planDataService = new PlanDataService();
                List<LedData> unsentPlans = planDataService.GetUnsentPlansForToday();

                ModbusService modbusService = new ModbusService();
                modbusService.SendPlansToDevicePro(unsentPlans);
                LedDataInfoService ledDataInfoService = new LedDataInfoService();
                // 从设备读取数据
                List<LedDataInfo> ledDataInfos = modbusService.ReadPlansFromDevice
();
                // 将读取的数据批量插入数据库
                ledDataInfoService.InsertDataList(ledDataInfos);

                // 将已下发的计划更新为已下发
                planDataService.UpdatePlanDataForToday();
```

```
                // 重新加载计划数据
                LoadPlanDataGrouped();

                // 显示下发成功的提示框
                MessageBox.Show("下发成功！");
            }
            else
            {
                // 如果用户点击了“否”，则不进行任何操作
                return;
            }
        }
        private void LoadPlanDataGrouped()
        {
            List<PlanDataGroup> planDataGroupList = planDataService.GetPlanDataGroup
ed();
            // 计算总页数
            int totalPages = (int)Math.Ceiling(planDataGroupList.Count / (double)pageSiz
e);
            // 修正当前页码防止越界
            if (currentPage > totalPages) currentPage = totalPages;
            if (currentPage < 1) currentPage = 1;
            // 分页加载数据
            var pagedData = planDataGroupList.Skip((currentPage - 1) * pageSize).Take(p
ageSize).ToList();
            // 更新数据源
            dataGridView.DataSource = pagedData;
            // 检查最新日期的未下发数量，并根据条件启用或禁用下发按钮
            //根据当前时间判断是白班还是夜班，白班的话取白班的未下发数量，夜班的
话取夜班的未下发数量
            //每天的上午 7 点 30 分到晚上 7 点 30 分为白班，晚上 7 点 30 分到上午 7 点 3
0 分为夜班
            // 获取当前时间，并判断是白班还是夜班
            DateTime currentTime = DateTime.Now;
            bool isDayShift = currentTime.TimeOfDay >= new TimeSpan(7, 30, 0) &&
currentTime.TimeOfDay <= new TimeSpan(19, 30, 0);
            // 获取最新日期的计划数据
            var latestPlanData = planDataGroupList.FirstOrDefault();
            // 根据当前班次判断未下发数量，并更新按钮状态
            if (latestPlanData != null)
            {
                int unsentCount = isDayShift ? latestPlanData.DayUnsentCount : latestPla
nData.NightUnsentCount;
```

```
                // 如果未下发数量为 0，禁用下发按钮
                if (unsentCount == 0)
                {
                    btnSend.Enabled = false;
                }
                else
                {
                    btnSend.Enabled = true;
                }
            }
            // 修改表头
            CustomizeColumnHeaders();
            // 更新分页按钮状态
            UpdatePaginationButtons(totalPages);
        }

        private void CustomizeColumnHeaders()
        {
            // 确保列顺序和自定义标题对应
            dataGridView.Columns["PlanDate"].HeaderText = "计划日期";
            dataGridView.Columns["PlanCount"].HeaderText = "下计划个数";
            dataGridView.Columns["DaySentCount"].HeaderText = "白班已下发";
            dataGridView.Columns["DayUnsentCount"].HeaderText = "白班未下发";
            dataGridView.Columns["NightSentCount"].HeaderText = "夜班已下发";
            dataGridView.Columns["NightUnsentCount"].HeaderText = "夜班未下发";
        }
        private void UpdatePaginationButtons(int totalPages)
        {
            // 上一页按钮仅在当前页不是第一页时可用
            btnPrevious.Enabled = currentPage > 1;

            // 下一页按钮仅在当前页不是最后一页时可用
            btnNext.Enabled = currentPage < totalPages;
        }

        private void ChangePage(int direction)
        {
            // 改变页码
            currentPage += direction;

            // 重新加载数据
            LoadPlanDataGrouped();
        }
    }
}
```

```
/**
 * 生产进度监控系统计划列表页面 *
 * @author 骆大利
 * @date 2024-09-15
 */
using KeLongLed.Model;
using KeLongLed.Service;
using Sunny.UI;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace KeLongLed.UI.Plan
{
    public partial class PlanListForm : Form
    {
        private DataGridView dataGridView;
        //private Button btnPrevious, btnNext;
        private UISymbolButton btnPrevious, btnNext;
        private int currentPage = 1;
        private int pageSize = int.Parse(ConfigurationManager.AppSettings["PageSize"]);
        private PlanDataService planDataService;
        private DateTime planDate;

        public PlanListForm(DateTime planDate)
        {
            this.planDate = planDate;
            planDataService = new PlanDataService();

            InitializeUI();
            LoadPlanDetails();
        }

        private void InitializeUI()
        {
            this.Text = "计划详细数据";
            this.Size = new Size(800, 600);

            // DataGridView 控件
            dataGridView = new DataGridView
            {
                Dock = DockStyle.Top,
```

```
            AutoGenerateColumns = true,
            Height = 500,
            AllowUserToAddRows = false,
            AllowUserToDeleteRows = false,
            ReadOnly = true,
            AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill
        };

        this.Controls.Add(dataGridView);

        // 分页控制
        Panel pagingPanel = new Panel
        {
            Dock = DockStyle.Bottom,
            Height = 40,
            Padding = new Padding(10, 0, 10, 10)
        };

        btnPrevious = new UISymbolButton
        {
            Symbol = 61514,
            //Text = "上一页",
            Dock = DockStyle.Left
        };
        btnPrevious.Click += (s, e) => ChangePage(-1);

        btnNext = new UISymbolButton
        {
            Symbol = 61518,
            //Text = "下一页",
            Dock = DockStyle.Right
        };
        btnNext.Click += (s, e) => ChangePage(1);

        pagingPanel.Controls.Add(btnPrevious);
        pagingPanel.Controls.Add(btnNext);

        this.Controls.Add(pagingPanel);
    }

    private void LoadPlanDetails()
    {
        // 获取计划详细数据
        List<PlanData> planDetails = planDataService.GetPlanDetails(planDate);
```

```
            // 分页加载数据
            int totalPages = (int)Math.Ceiling(planDetails.Count / (double)pageSize);

            if (currentPage > totalPages) currentPage = totalPages;
            if (currentPage < 1) currentPage = 1;

            var pagedData = planDetails.Skip((currentPage - 1) * pageSize).Take(pageSiz
e).ToList();

            dataGridView.DataSource = pagedData;

            // 修改表头
            CustomizeColumnHeaders();
            //隐藏不需要显示的列
            dataGridView.Columns["WorkType"].Visible = false;
            dataGridView.Columns["PlanType"].Visible = false;
            dataGridView.Columns["PushStatus"].Visible = false;

            // 更新分页按钮状态
            btnPrevious.Enabled = currentPage > 1;
            btnNext.Enabled = currentPage < totalPages;
        }

        private void CustomizeColumnHeaders()
        {
            dataGridView.Columns["LedId"].HeaderText = "LED 编号";
            dataGridView.Columns["Name"].HeaderText = "名称";
            dataGridView.Columns["LedPlanData"].HeaderText = "计划数量";
            dataGridView.Columns["PlanDate"].HeaderText = "计划日期";
            dataGridView.Columns["CreateTime"].HeaderText = "创建时间";
            dataGridView.Columns["PlanTypeDescription"].HeaderText = "计划类型";  //
新增 PlanTypeDescription 列头
            dataGridView.Columns["WorkTypeDescription"].HeaderText = "班次";
            dataGridView.Columns["PushStatusDescription"].HeaderText = "下发状态";

        }

        private void ChangePage(int direction)
        {
            currentPage += direction;
            LoadPlanDetails();
        }
    }
}
```

```
/**
 * 生产进度监控系统计划数据服务类*
 * @author 骆大利
 * @date 2024-08-05
 */
using KeLongLed.Model;
using Oracle.ManagedDataAccess.Client;
using System;
using System.Collections.Generic;
using System.Data;

namespace KeLongLed.Service
{
    public class PlanDataService
    {

        // 获取计划数据
        public List<PlanDataGroup> GetPlanDataGrouped()
        {
            string sql = @"
                    SELECT PLAN_DATE,
                      COUNT(CASE WHEN PUSH_STATUS = 1 AND WORK_TYPE
=1 THEN 1 END) AS DAY_SENT_COUNT,
                      COUNT(CASE WHEN PUSH_STATUS = 1 AND WORK_TYPE
=2 THEN 1 END) AS NIGHT_SENT_COUNT,
                       COUNT(CASE WHEN PUSH_STATUS = 0 AND WORK_TYP
E=1 THEN 1 END) AS DAY_UNSENT_COUNT,
                      COUNT(CASE WHEN PUSH_STATUS = 0 AND WORK_TYPE
=2 THEN 1 END) AS NIGHT_UNSENT_COUNT,
                      COUNT(*) AS PlanCount
                      FROM LED_PLAN_INFO
                      GROUP BY PLAN_DATE
                      ORDER BY PLAN_DATE DESC
                  ";

            List<PlanDataGroup> planDataGroupList = new List<PlanDataGroup>();

            using (DataTable dataTable = Helper.OracleHelper.ExecuteDataTable(sql))
            {
                foreach (DataRow row in dataTable.Rows)
                {
                    PlanDataGroup group = new PlanDataGroup
                    {
                        PlanDate = Convert.ToDateTime(row["PLAN_DATE"]),
                        PlanCount = Convert.ToInt32(row["PlanCount"]),
```

```
                    DaySentCount = Convert.ToInt32(row["DAY_SENT_COUNT"]),
                    NightSentCount = Convert.ToInt32(row["NIGHT_SENT_COUNT
"]),
                    DayUnsentCount = Convert.ToInt32(row["DAY_UNSENT_COU
NT"]),
                    NightUnsentCount = Convert.ToInt32(row["NIGHT_UNSENT_C
OUNT"])
                };

                planDataGroupList.Add(group);
            }
        }

        return planDataGroupList;
    }

    // 获取计划详情
    public List<PlanData> GetPlanDetails(DateTime planDate)
    {
        string sql = @"
                    SELECT A.LED_ID, B.LED_NAME, A.LED_PLAN_DATA,
 A.PLAN_DATE, A.CREATE_TIME,A.PLAN_TYPE,A.WORK_TYPE,A.PUSH_STATUS
                    FROM LED_PLAN_INFO A LEFT JOIN LED_DEVICE_I
NFO B ON A.LED_ID = B.LED_ID
                    WHERE TRUNC(A.PLAN_DATE) = TRUNC(:planDate)
                    ORDER BY A.WORK_TYPE,A.CREATE_TIME DESC";
        List<PlanData> planDetails = new List<PlanData>();
        // 构造参数
        OracleParameter[] parameters = new OracleParameter[]
        {
            new OracleParameter("planDate", planDate)
        };

        using (DataTable dataTable = Helper.OracleHelper.ExecuteDataTable(sql,parame
ters))
        {

            foreach (DataRow row in dataTable.Rows)
            {
                PlanData planData = new PlanData
                {
                    LedId = Convert.ToInt32(row["LED_ID"]),
                    Name = row["LED_NAME"].ToString(),
                    LedPlanData = Convert.ToInt32(row["LED_PLAN_DATA"]),
                    PlanDate = Convert.ToDateTime(row["PLAN_DATE"]),
```

```
                    CreateTime = Convert.ToDateTime(row["CREATE_TIME"]),
                    // 设置 PlanType
                    PlanType = Convert.ToInt32(row["PLAN_TYPE"]),
                    WorkType = Convert.ToInt32(row["WORK_TYPE"]),
                    PushStatus = Convert.ToInt32(row["PUSH_STATUS"])
                };

                planDetails.Add(planData);
            }
        }

        return planDetails;
    }

    //获取今日未下发的计划
    public List<LedData> GetUnsentPlansForToday()
    {
        string sql = @"
                SELECT LED_ID, LED_PLAN_DATA,ROWID  FROM LED_PLAN
_INFO WHERE TO_CHAR(PLAN_DATE, 'YYYY-MM-DD') = TO_CHAR(SYSDATE, 'YY
YY-MM-DD')
                AND WORK_TYPE = CASE WHEN TO_CHAR(SYSDATE, 'HH24
MI') BETWEEN '0730' AND '1930' THEN 1 ELSE 2 END
                AND PUSH_STATUS = 0
              ";
        List<LedData> unsentPlans = new List<LedData>();
        using (DataTable dataTable = Helper.OracleHelper.ExecuteDataTable(sql))
        {
            foreach (DataRow row in dataTable.Rows)
            {
                LedData ledData = new LedData
                {
                    LedId = Convert.ToInt32(row["LED_ID"]),
                    PlannedQuantity = Convert.ToInt32(row["LED_PLAN_DATA"])
                };
                unsentPlans.Add(ledData);
            }
        }
        return unsentPlans;
    }

    // 更新计划数据，将今天的数据 PUSH_STATUS 设置为已下发
    public void UpdatePlanDataForToday()
    {
        string sql = @"
```

```
                    UPDATE LED_PLAN_INFO
                    SET PUSH_STATUS = 1
                    WHERE TO_CHAR(PLAN_DATE, 'YYYY-MM-DD') = TO_CHAR
(SYSDATE, 'YYYY-MM-DD')
                    AND WORK_TYPE = CASE WHEN TO_CHAR(SYSDATE, 'HH24
MI') BETWEEN '0730' AND '1930' THEN 1 ELSE 2 END
                    AND PUSH_STATUS = 0
                ";
            Helper.OracleHelper.ExecuteNonQuery(sql);
        }
    }
}

/**
 * 生产进度监控系统 Modbus 服务类*
 * @author 骆大利
 * @date 2024-08-05
 */
using KeLongLed.Model;
using Modbus.Device;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Net.Sockets;
using System.Threading;
namespace KeLongLed.Service
{
    public class ModbusService
    {
        private readonly string ipAddress;
        private readonly int port;
        private int ledCount;

        public ModbusService()
        {
            // 从配置文件读取 IP 和端口
            ipAddress = ConfigurationManager.AppSettings["ModbusIpAddress"];
            string portString = ConfigurationManager.AppSettings["ModbusPort"];
            ledCount = int.Parse(ConfigurationManager.AppSettings["LedCount"]);
            if (string.IsNullOrEmpty(ipAddress) || string.IsNullOrEmpty(portString) || !int.T
ryParse(portString, out port))
            {
                throw new Exception("Modbus 配置文件中的 IP 或端口无效，请检查 A
pp.config。");
            }
```

```
        }

        // 向设备写入多个计划数据，按顺序从 0 地址依次往后写入
        public void SendPlansToDevice(List<int> planDataList)
        {
            using (TcpClient tcpClient = new TcpClient(ipAddress, port))
            {
                //创建 Modbus 主站
                IModbusMaster master = ModbusIpMaster.CreateIp(tcpClient);

                //起始地址
                ushort startAddress = 0; //对应 Modbus 地址 40001
                foreach (int planData in planDataList)
                {
                    //将计划数据拆分为高 16 位和低 16 位
                    ushort planHigh = (ushort)((planData >> 16) & 0xFFFF);
                    ushort planLow = (ushort)(planData & 0xFFFF);

                    //写入计划数据到当前地址
                    ushort[] writeData = { planHigh, planLow };
                    master.WriteMultipleRegisters(1, startAddress, writeData);//假设从站 I
D 为 1

                    //跳过实际数据的两个寄存器，准备写入下一组
                    startAddress += 4; //每组占用 4 个寄存器
                }
            }
        }

        //向设备写入多个计划数据，参数是 LedData 实体集合，LedData 里面有计划数量 P
lanData，还有屏幕 ID，ledID,每个屏幕占 4 个寄存器
        public void SendPlansToDevicePro(List<LedData> ledDatas)
        {
            using (TcpClient tcpClient = new TcpClient(ipAddress, port))
            {
                // 创建 Modbus 主站
                IModbusMaster master = ModbusIpMaster.CreateIp(tcpClient);

                // 遍历每个 LedData
                foreach (var ledData in ledDatas)
                {
                    // 获取屏幕 ID，计算起始地址（每个屏幕占用 4 个寄存器）
                    ushort startAddress = (ushort)((ledData.LedId - 1) * 4);  // LedId 从
1 开始，地址从 0 开始
```

```
                    // 拆分计划数据为高 16 位和低 16 位
                    ushort planHigh = (ushort)((ledData.PlannedQuantity >> 16) & 0xFF
FF);
                    ushort planLow = (ushort)(ledData.PlannedQuantity & 0xFFFF);

                    // 将计划数据写入 Modbus 寄存器（每个屏幕占用 2 个寄存器，高 1
6 位和低 16 位）
                    ushort[] writeData = { planHigh, planLow };
                    master.WriteMultipleRegisters(1, startAddress, writeData); // 假设从
站 ID 为 1
                    // 添加 500 毫秒的等待时间
                    Thread.Sleep(500);
                }
            }
        }
        // 向设备写入单个计划数据
        public void SendSinglePlanToDevice(int screenId, int planData)
        {
            using (TcpClient tcpClient = new TcpClient(ipAddress, port))
            {
                // 创建 Modbus 主站
                IModbusMaster master = ModbusIpMaster.CreateIp(tcpClient);

                // 计算起始地址，屏幕 ID 决定了起始地址
                ushort startAddress = (ushort)((screenId-1) * 4); // 每个屏幕占用 4 个寄
存器
                // 将计划数据拆分为高 16 位和低 16 位
                ushort planHigh = (ushort)((planData >> 16) & 0xFFFF);
                ushort planLow = (ushort)(planData & 0xFFFF);

                // 写入计划数据到当前地址
                ushort[] writeData = { planHigh, planLow };
                master.WriteMultipleRegisters(1, startAddress, writeData); // 假设从站 ID
为 1
            }
        }



        public List<LedDataInfo> ReadPlansFromDevice()
        {
            List<LedDataInfo> ledDataInfos = new List<LedDataInfo>();

            // 假设屏幕数量是从配置文件或自定义参数中读取
            int screenCount = ledCount; // 获取屏幕数量
```

```
            using (TcpClient tcpClient = new TcpClient(ipAddress, port))
            {
                // 创建 Modbus 主站
                IModbusMaster master = ModbusIpMaster.CreateIp(tcpClient);

                // 遍历每个屏幕，读取计划数量和实际数量
                for (int ledId = 1; ledId <= screenCount; ledId++)
                {
                    // 计算起始地址（每个屏幕占用 4 个寄存器）
                    ushort startAddress = (ushort)((ledId - 1) * 4);  // 假设 LedId 从 1
开始，地址从 0 开始

                    // 读取 4 个寄存器：2 个寄存器用于计划数量，2 个寄存器用于实际
数量
                    ushort[] readData = master.ReadHoldingRegisters(1, startAddress, 4);
 // 假设从站 ID 为 1

                    // 将读取的数据拆分为计划数量和实际数量
                    int plannedQuantity = (readData[0] << 16) | readData[1]; // 高 16 位
和低 16 位拼接为计划数量
                    int actualQuantity = (readData[2] << 16) | readData[3]; // 高 16 位
和低 16 位拼接为实际数量

                    // 创建 LedDataInfo 对象
                    LedDataInfo ledDataInfo = new LedDataInfo
                    {
                        LedId = ledId,
                        LedPlanData = plannedQuantity,
                        LedRealData = actualQuantity
                    };

                    // 添加到列表
                    ledDataInfos.Add(ledDataInfo);
                }
            }

            return ledDataInfos;
        }

    }
}
```

```
/**
 * 生产进度监控系统屏幕数据服务类*
 * @author 骆大利
 * @date 2024-08-05
 */
using KeLongLed.Model;
using Oracle.ManagedDataAccess.Client;
using System;
using System.Collections.Generic;
using System.Data;
namespace KeLongLed.Service
{
    public class LedDataInfoService
    {
        // 插入单条数据
        public void InsertSingleData(LedDataInfo ledData)
        {
            string sql = @"
                INSERT INTO LED_DATA_INFO (LED_ID, LED_PLAN_DATA, LED_
REAL_DATA, CREATE_TIME)
                VALUES (:LedId, :LedPlanData, :LedRealData, :CreateTime)
            ";
            // 将 CREATE_TIME 设置为当前日期时间
            DateTime createTime = DateTime.Now;
            OracleParameter[] parameters = new OracleParameter[]
            {
                new OracleParameter(":LedId", ledData.LedId),
                new OracleParameter(":LedPlanData", ledData.LedPlanData),
                new OracleParameter(":LedRealData", ledData.LedRealData),
                new OracleParameter(":CreateTime", createTime)
            };

            Helper.OracleHelper.ExecuteDataTable(sql, parameters);
        }

        // 删除数据
        public void DeleteData(int ledId, DateTime planDate)
        {
            string sql = @"
                DELETE FROM LED_DATA_INFO
                WHERE LED_ID = :LedId AND LED_PLAN_DATE = :LedPlanDate
            ";

            OracleParameter[] parameters = new OracleParameter[]
            {
```

```
            new OracleParameter(":LedId", ledId),
            new OracleParameter(":LedPlanDate", planDate)
        };

        Helper.OracleHelper.ExecuteDataTable(sql, parameters);
    }
    // 批量插入数据
    public void InsertDataList(List<LedDataInfo> ledDataList)
    {
        foreach (var ledData in ledDataList)
        {
            InsertSingleData(ledData);  // 调用 InsertSingleData 方法插入每一条数据
        }
    }
    // 更新数据
    public void UpdateData(LedDataInfo ledData)
    {
        string sql = @"
            UPDATE LED_DATA_INFO
            SET LED_PLAN_DATE = :LedPlanDate,
                LED_REAL_DATA = :LedRealData,
                CREATE_TIME = :CreateTime
            WHERE LED_ID = :LedId
        ";
        OracleParameter[] parameters = new OracleParameter[]
        {
            new OracleParameter(":LedId", ledData.LedId),
            new OracleParameter(":LedPlanDate", ledData.LedPlanData),
            new OracleParameter(":LedRealData", ledData.LedRealData),
            new OracleParameter(":CreateTime", ledData.CreateTime)
        };

        Helper.OracleHelper.ExecuteDataTable(sql, parameters);
    }
    // 获取 LED 数据列表
    public List<LedDataInfo> GetLedDataList(DateTime startDate, DateTime endDate)
    {
        string sql = @"
            SELECT LED_ID, LED_PLAN_DATE, LED_REAL_DATA, CREATE_TI
ME
            FROM LED_DATA_INFO
            WHERE LED_PLAN_DATE BETWEEN :StartDate AND :EndDate
            ORDER BY LED_PLAN_DATE DESC
        ";
```

```
            OracleParameter[] parameters = new OracleParameter[]
            {
                new OracleParameter(":StartDate", startDate),
                new OracleParameter(":EndDate", endDate)
            };

            List<LedDataInfo> ledDataList = new List<LedDataInfo>();

            using (DataTable dataTable = Helper.OracleHelper.ExecuteDataTable(sql, para
meters))
            {
                foreach (DataRow row in dataTable.Rows)
                {
                    LedDataInfo ledData = new LedDataInfo
                    {
                        LedId = Convert.ToInt32(row["LED_ID"]),
                        LedPlanData = Convert.ToInt32(row["LED_PLAN_DATA"]),
                        LedRealData = Convert.ToInt32(row["LED_REAL_DATA"]),
                        CreateTime = Convert.ToDateTime(row["CREATE_TIME"])
                    };

                    ledDataList.Add(ledData);
                }
            }

            return ledDataList;
        }
    }
}

/**
 * 生产进度监控系统数据导入服务类*
 * @author 骆大利
 * @date 2024-08-15
 */

using ExcelDataReader;
using KeLongLed.UI.Utils;
using System;
using System.Collections.Generic;
using System.Data;
using System.IO;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace KeLongLed.Service
{
    public class ImportDataService
    {

        // 将 Excel 数据插入数据库
        private void InsertDataIntoDatabase(DataTable dt)
        {
            int suc_rows = 0;
            List<int> planDataList = new List<int>();//收集计划数据
            ModbusService modbusService = new ModbusService();
            DateTime today = DateTime.Now.Date; // 获取当天日期
            // 格式化当天日期为 Oracle 可识别的字符串
            string todayStr = today.ToString("yyyy-MM-dd");
            // 删除当天数据
            string deleteQuery = $"DELETE FROM LED_PLAN_INFO WHERE PLAN_
TYPE=1 AND TRUNC(PLAN_DATE) = TO_DATE('{todayStr}', 'YYYY-MM-DD')";
            Helper.OracleHelper.ExecuteNonQuery(deleteQuery);

            foreach (DataRow row in dt.Rows)
            {
                int ledId = Convert.ToInt32(row["设备"]);
                //DateTime planDate = Convert.ToDateTime(row["计划日期"]); // 从 Exc
el 获取的日期类型
                DateTime createTime = DateTime.Now; // 当前系统时间
                int ledPlanData = Convert.ToInt32(row["计划数量"]);
                int workType = Convert.ToInt32(row["白夜班"]);
                // 格式化日期为 Oracle 可识别的字符串
                //string planDateStr = planDate.ToString("yyyy-MM-dd");
                string createTimeStr = createTime.ToString("yyyy-MM-dd HH:mm:ss");
                // 拼接 SQL 字符串
                string query = $"INSERT INTO LED_PLAN_INFO (LED_ID, LED_PL
AN_DATA, WORK_TYPE,PLAN_DATE, CREATE_TIME) " +
                                $"VALUES ({ledId}, {ledPlanData},{workType}, TO_D
ATE('{todayStr}', 'YYYY-MM-DD'), " +
                                $"TO_DATE('{createTimeStr}', 'YYYY-MM-DD HH24:
MI:SS'))";
                int result = Helper.OracleHelper.ExecuteNonQuery(query);
                if (result > 0)
                {
                    suc_rows++;
                    //planDataList.Add(ledPlanData); // 添加到计划数据列表
                }

            }
```

```
            try
            {
                //下发所有计划数据
                //modbusService.SendPlansToDevice(planDataList);
                MessageBox.Show("成功导入" + suc_rows + "条数据！");
            }
            catch (Exception ex)
            {
                MessageBox.Show("数据导入失败：" + ex.Message);
            }
        }

        // 将单条数据插入数据库
        public void InsertSingleDataIntoDatabase(int ledId, int ledPlanData)
        {
            ModbusService modbusService = new ModbusService();
            DateTime planDate = DateTime.Now.Date; // 设置为当天日期
            DateTime createTime = DateTime.Now;
            string planDateStr = planDate.ToString("yyyy-MM-dd");
            string createTimeStr = createTime.ToString("yyyy-MM-dd HH:mm:ss");

            //加一个白夜班变量，如果当前时间是早上 7 点半到晚上 7 点半，就是白班，
否则是夜班
            int workType = 1;
            if (DateTime.Now.Hour >= 19 || DateTime.Now.Hour < 7)
            {
                workType = 2;
            }
            // 插入单条数据到数据库
            string query = $"INSERT INTO LED_PLAN_INFO (LED_ID, LED_PLAN_D
ATA, WORK_TYPE, PLAN_DATE, CREATE_TIME,PLAN_TYPE,PUSH_STATUS) " +
                           $"VALUES ({ledId}, {ledPlanData},{workType}, TO_DATE('
{planDateStr}', 'YYYY-MM-DD'), " +
                           $"TO_DATE('{createTimeStr}', 'YYYY-MM-DD HH24:MI:SS
'),2,1)";
            int result = Helper.OracleHelper.ExecuteNonQuery(query);

            if (result > 0)
            {
                // 数据成功插入数据库后，进行下发操作
                try
                {
                    // 下发单条数据
                    modbusService.SendSinglePlanToDevice(ledId, ledPlanData);
                    MessageBox.Show("成功插入并下发数据！");
```

```
                }
                catch (Exception ex)
                {
                    MessageBox.Show("数据下发失败：" + ex.Message);
                }
            }
            else
            {
                MessageBox.Show("单个数据插入失败！");
            }
        }
        // 从 Excel 导入数据
        public async void ImportExcelData(string filePath)
        {
            // 创建并显示加载窗口
            LoadingForm loadingForm = new LoadingForm("正在导入数据，请稍候...");
            loadingForm.Show();
            try
            {
                DataSet result = await Task.Run(() =>
                {
                    using (FileStream stream = File.Open(filePath, FileMode.Open, File
Access.Read))
                    using (IExcelDataReader excelReader = ExcelReaderFactory.CreateRe
ader(stream))
                    {
                        return excelReader.AsDataSet(new ExcelDataSetConfiguration()
                        {
                            ConfigureDataTable = (_) => new ExcelDataTableConfigur
ation() { UseHeaderRow = true }
                        });
                    }
                });

                DataTable dataTable = result.Tables[0];
                InsertDataIntoDatabase(dataTable);
            }
            catch (Exception ex)
            {
                // 关闭加载窗口
                loadingForm.Close();
                MessageBox.Show("导入失败: " + ex.Message);
            }
            finally
            {
```

```
                // 关闭加载窗口
                loadingForm.Close();
            }
        }
    }
}

/**
 * 生产进度监控系统数据服务类*
 * @author 骆大利
 * @date 2024-08-15
 */

using KeLongLed.Model;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;

namespace KeLongLed.Service
{
    public class DataService
    {

        // 获取最新的 LED 数据
        public List<LedData> GetLatestLedData()
        {
            string sql = "select a.led_id,a.led_plan_data,a.led_real_data,b.led_name from (s
elect * from led_data_info where id in (SELECT max(id) as id FROM LED_DATA_INFO
 GROUP BY LED_ID) ) a left join  led_device_info b on a.led_id = b.led_id order by a.
led_id asc";
            using (DataTable result = Helper.OracleHelper.ExecuteDataTable(sql))
            {
                return result.Rows.Cast<DataRow>().Select(row => new LedData
                {
                    LedId = Convert.ToInt32(row["led_id"]),
                    Name = row["led_name"].ToString(),
                    PlannedQuantity = Convert.ToInt32(row["led_plan_data"]),
                    ActualQuantity = Convert.ToInt32(row["led_real_data"])
                }).ToList();
            }
        }
    }

}
```

```
/**
 * 生产进度监控系统实体类*
 * @author 骆大利
 * @date 2024-08-15
 */

using System;
namespace KeLongLed.Model
{
    public class PlanDataGroup
    {
        public DateTime PlanDate { get; set; }
        public int PlanCount { get; set; }
        public int DaySentCount { get; set; }
        public int DayUnsentCount { get; set; }
        public int NightSentCount { get; set; }
        public int NightUnsentCount { get; set; }

    }

}

using System;
namespace KeLongLed.Model
{
    public class PlanData
    {
        public int LedId { get; set; }
        public string Name { get; set; }
        public int LedPlanData { get; set; }
        public DateTime PlanDate { get; set; }
        public DateTime CreateTime { get; set; }
        public int WorkType { get; set; }
        public int PushStatus { get; set; }

        // 新增 PlanType 字段
        public int PlanType { get; set; }  // 1 为批量导入，2 为手动导入

        // 新增 PlanTypeDescription 字段
        public string PlanTypeDescription
        {
            get
            {
                return PlanType == 1 ? "批量导入" : "手动导入";
            }
```

```
        }


        public string WorkTypeDescription
        {
            get
            {
                return WorkType == 1 ? "白班" : "夜班";
            }
        }

        public string PushStatusDescription
        {
            get
            {
                return PushStatus == 1 ? "已下发" : "未下发";
            }
        }
    }

}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace KeLongLed.Model
{
    public class LedDataInfo
    {
        public int LedId { get; set; }              // LED_ID
        public int LedPlanData { get; set; } // LED_PLAN_DATA
        public int LedRealData { get; set; }       // LED_REAL_DATA
        public DateTime CreateTime { get; set; } // CREATE_TIME
    }
}

using System;


namespace KeLongLed.Model
{
```

```
    public class LedData
    {
        public int Id { get; set; }
        public int LedId { get; set; }
        public string Name { get; set; }
        public int PlannedQuantity { get; set; }
        public int ActualQuantity { get; set; }
    }
}


package com.KeLongLed.led.service.impl;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.KeLongLed.led.mapper.LedDeviceInfoMapper;
import com.KeLongLed.led.domain.LedDeviceInfo;
import com.KeLongLed.led.service.ILedDeviceInfoService;

/**
* 电子屏设备信息 Service 业务层处理
*
* @author 骆大利
* @date 2024-11-14
*/
@Service
public class LedDeviceInfoServiceImpl implements ILedDeviceInfoService
{
    @Autowired
    private LedDeviceInfoMapper ledDeviceInfoMapper;

    /**
     * 查询电子屏设备信息
     *
     * @param id 电子屏设备信息主键
     * @return 电子屏设备信息
     */
    @Override
    public LedDeviceInfo selectLedDeviceInfoById(Long id)
    {
        return ledDeviceInfoMapper.selectLedDeviceInfoById(id);
    }

    /**
```

```
     * 查询电子屏设备信息列表
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 电子屏设备信息
     */
    @Override
    public List<LedDeviceInfo> selectLedDeviceInfoList(LedDeviceInfo ledDeviceInfo)
    {
        return ledDeviceInfoMapper.selectLedDeviceInfoList(ledDeviceInfo);
    }

    /**
     * 新增电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
     */
    @Override
    public int insertLedDeviceInfo(LedDeviceInfo ledDeviceInfo)
    {
        return ledDeviceInfoMapper.insertLedDeviceInfo(ledDeviceInfo);
    }

    /**
     * 修改电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
     */
    @Override
    public int updateLedDeviceInfo(LedDeviceInfo ledDeviceInfo)
    {
        return ledDeviceInfoMapper.updateLedDeviceInfo(ledDeviceInfo);
    }

    /**
     * 批量删除电子屏设备信息
     *
     * @param ids 需要删除的电子屏设备信息主键
     * @return 结果
     */
    @Override
    public int deleteLedDeviceInfoByIds(Long[] ids)
    {
        return ledDeviceInfoMapper.deleteLedDeviceInfoByIds(ids);
```

```
    }

    /**
     * 删除电子屏设备信息信息
     *
     * @param id 电子屏设备信息主键
     * @return 结果
     */
    @Override
    public int deleteLedDeviceInfoById(Long id)
    {
        return ledDeviceInfoMapper.deleteLedDeviceInfoById(id);
    }
}
/**
     * 电子屏设备信息 MAPPER 类
     *
     *
     *
     */
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.ruoyi.led.mapper.LedDeviceInfoMapper">

    <resultMap type="LedDeviceInfo" id="LedDeviceInfoResult">
        <result property="ledId"    column="led_id"    />
        <result property="ledName"    column="led_name"    />
        <result property="depName"    column="dep_name"    />
        <result property="depId"    column="dep_id"    />
        <result property="depIdOa"    column="dep_id_oa"    />
        <result property="gongxu"    column="gongxu"    />
        <result property="id"    column="id"    />
    </resultMap>

    <sql id="selectLedDeviceInfoVo">
        select led_id, led_name, dep_name, dep_id, dep_id_oa, gongxu, id from led_device
_info
    </sql>

    <select id="selectLedDeviceInfoList" parameterType="LedDeviceInfo" resultMap="LedDev
iceInfoResult">
        <include refid="selectLedDeviceInfoVo"/>
        <where>
```

```
            <if test="ledId != null "> and led_id = #{ledId}</if>
            <if test="ledName != null  and ledName != ""> and led_name like concat(co
ncat('%', #{ledName}), '%')</if>
            <if test="depName != null  and depName != ""> and dep_name like concat(c
oncat('%', #{depName}), '%')</if>
            <if test="depId != null "> and dep_id = #{depId}</if>
            <if test="depIdOa != null "> and dep_id_oa = #{depIdOa}</if>
            <if test="gongxu != null  and gongxu != ""> and gongxu like concat(concat
('%', #{gongxu}), '%')</if>
        </where>
    </select>

    <select id="selectLedDeviceInfoById" parameterType="Long" resultMap="LedDeviceInfoR
esult">
        <include refid="selectLedDeviceInfoVo"/>
        where id = #{id}
    </select>

    <insert id="insertLedDeviceInfo" parameterType="LedDeviceInfo">
        <selectKey keyProperty="id" resultType="long" order="BEFORE">
            SELECT seq_led_device_info.NEXTVAL as id FROM DUAL
        </selectKey>
        insert into led_device_info
        <trim prefix="(" suffix=")" suffixOverrides=",">
            <if test="ledId != null">led_id,</if>
            <if test="ledName != null">led_name,</if>
            <if test="depName != null">dep_name,</if>
            <if test="depId != null">dep_id,</if>
            <if test="depIdOa != null">dep_id_oa,</if>
            <if test="gongxu != null">gongxu,</if>
            <if test="id != null">id,</if>
         </trim>
        <trim prefix="values (" suffix=")" suffixOverrides=",">
            <if test="ledId != null">#{ledId},</if>
            <if test="ledName != null">#{ledName},</if>
            <if test="depName != null">#{depName},</if>
            <if test="depId != null">#{depId},</if>
            <if test="depIdOa != null">#{depIdOa},</if>
            <if test="gongxu != null">#{gongxu},</if>
            <if test="id != null">#{id},</if>
         </trim>
    </insert>

    <update id="updateLedDeviceInfo" parameterType="LedDeviceInfo">
        update led_device_info
```

```
        <trim prefix="SET" suffixOverrides=",">
            <if test="ledId != null">led_id = #{ledId},</if>
            <if test="ledName != null">led_name = #{ledName},</if>
            <if test="depName != null">dep_name = #{depName},</if>
            <if test="depId != null">dep_id = #{depId},</if>
            <if test="depIdOa != null">dep_id_oa = #{depIdOa},</if>
            <if test="gongxu != null">gongxu = #{gongxu},</if>
        </trim>
        where id = #{id}
    </update>

    <delete id="deleteLedDeviceInfoById" parameterType="Long">
        delete from led_device_info where id = #{id}
    </delete>

    <delete id="deleteLedDeviceInfoByIds" parameterType="String">
        delete from led_device_info where id in
        <foreach item="id" collection="array" open="(" separator="," close=")">
            #{id}
        </foreach>
    </delete>
</mapper>

package com.KeLongLed.led.domain;

import com.fasterxml.jackson.annotation.JsonFormat;
import org.apache.commons.lang3.builder.ToStringBuilder;
import org.apache.commons.lang3.builder.ToStringStyle;
import com.KeLongLed.common.annotation.Excel;
import com.KeLongLed.common.core.domain.BaseEntity;

/**
* 电子屏设备信息对象类 led_device_info
*
* @author 骆大利
* @date 2024-11-14
*/
public class LedDeviceInfo extends BaseEntity
{
    private static final long serialVersionUID = 1L;

    /** 屏幕 ID 号 */
    @Excel(name = "屏幕 ID 号")
    private Long ledId;
```

```
/** 设备名称 */
@Excel(name = "设备名称")
private String ledName;

/** 事业部名称 */
@Excel(name = "事业部名称")
private String depName;

/** 事业部 ID */
@Excel(name = "事业部 ID")
private Long depId;

/** OA 事业部 ID */
@Excel(name = "OA 事业部 ID")
private Long depIdOa;

/** 工序 */
@Excel(name = "工序")
private String gongxu;

/** ID 主键 seq_led_device_info */
private Long id;

public void setLedId(Long ledId)
{
    this.ledId = ledId;
}

public Long getLedId()
{
    return ledId;
}
public void setLedName(String ledName)
{
    this.ledName = ledName;
}

public String getLedName()
{
    return ledName;
}
public void setDepName(String depName)
{
    this.depName = depName;
}
```

```
public String getDepName()
{
    return depName;
}
public void setDepId(Long depId)
{
    this.depId = depId;
}

public Long getDepId()
{
    return depId;
}
public void setDepIdOa(Long depIdOa)
{
    this.depIdOa = depIdOa;
}

public Long getDepIdOa()
{
    return depIdOa;
}
public void setGongxu(String gongxu)
{
    this.gongxu = gongxu;
}

public String getGongxu()
{
    return gongxu;
}
public void setId(Long id)
{
    this.id = id;
}

public Long getId()
{
    return id;
}

@Override
public String toString() {
    return new ToStringBuilder(this,ToStringStyle.MULTI_LINE_STYLE)
```

```
                .append("ledId", getLedId())
                .append("ledName", getLedName())
                .append("depName", getDepName())
                .append("depId", getDepId())
                .append("depIdOa", getDepIdOa())
                .append("gongxu", getGongxu())
                .append("id", getId())
                .toString();
    }
}

package com.KeLongLed.led.service;

import java.util.List;
import com.KeLongLed.led.domain.LedDeviceInfo;

/**
* 电子屏设备信息 Service 接口类
*
* @author 骆大利
* @date 2024-11-14
*/
public interface ILedDeviceInfoService
{
    /**
     * 查询电子屏设备信息
     *
     * @param id 电子屏设备信息主键
     * @return 电子屏设备信息
     */
    public LedDeviceInfo selectLedDeviceInfoById(Long id);

    /**
     * 查询电子屏设备信息列表
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 电子屏设备信息集合
     */
    public List<LedDeviceInfo> selectLedDeviceInfoList(LedDeviceInfo ledDeviceInfo);

    /**
     * 新增电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
```

```
     */
    public int insertLedDeviceInfo(LedDeviceInfo ledDeviceInfo);

    /**
     * 修改电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
     */
    public int updateLedDeviceInfo(LedDeviceInfo ledDeviceInfo);

    /**
     * 批量删除电子屏设备信息
     *
     * @param ids 需要删除的电子屏设备信息主键集合
     * @return 结果
     */
    public int deleteLedDeviceInfoByIds(Long[] ids);

    /**
     * 删除电子屏设备信息信息
     *
     * @param id 电子屏设备信息主键
     * @return 结果
     */
    public int deleteLedDeviceInfoById(Long id);
}

package com.KeLongLed.led.mapper;
import java.util.List;
import com.KeLongLed.common.annotation.DataSource;
import com.KeLongLed.common.enums.DataSourceType;
import com.KeLongLed.led.domain.LedDeviceInfo;

/**
* 电子屏设备信息 Mapper 接口类
*
* @author 骆大利
* @date 2024-11-14
*/
@DataSource(value = DataSourceType.SLAVE)
public interface LedDeviceInfoMapper
{
    /**
     * 查询电子屏设备信息
```

```
     *
     * @param id 电子屏设备信息主键
     * @return 电子屏设备信息
     */
    public LedDeviceInfo selectLedDeviceInfoById(Long id);

    /**
     * 查询电子屏设备信息列表
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 电子屏设备信息集合
     */
    public List<LedDeviceInfo> selectLedDeviceInfoList(LedDeviceInfo ledDeviceInfo);

    /**
     * 新增电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
     */
    public int insertLedDeviceInfo(LedDeviceInfo ledDeviceInfo);

    /**
     * 修改电子屏设备信息
     *
     * @param ledDeviceInfo 电子屏设备信息
     * @return 结果
     */
    public int updateLedDeviceInfo(LedDeviceInfo ledDeviceInfo);

    /**
     * 删除电子屏设备信息
     *
     * @param id 电子屏设备信息主键
     * @return 结果
     */
    public int deleteLedDeviceInfoById(Long id);

    /**
     * 批量删除电子屏设备信息
     *
     * @param ids 需要删除的数据主键集合
     * @return 结果
     */
    public int deleteLedDeviceInfoByIds(Long[] ids);
```

```
}
package com.KeLongLed.led.controller;

import java.util.List;
import javax.servlet.http.HttpServletResponse;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.KeLongLed.common.annotation.Log;
import com.KeLongLed.common.core.controller.BaseController;
import com.KeLongLed.common.core.domain.AjaxResult;
import com.KeLongLed.common.enums.BusinessType;
import com.KeLongLed.led.domain.LedDeviceInfo;
import com.KeLongLed.led.service.ILedDeviceInfoService;
import com.KeLongLed.common.utils.poi.ExcelUtil;
import com.KeLongLed.common.core.page.TableDataInfo;

/**
* 电子屏设备信息 Controller
*
* @author 骆大利
* @date 2024-11-14
*/
@RestController
@RequestMapping("/led/ledinfo")
public class LedDeviceInfoController extends BaseController
{
    @Autowired
    private ILedDeviceInfoService ledDeviceInfoService;

    /**
     * 查询电子屏设备信息列表
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:list')")
    @GetMapping("/list")
    public TableDataInfo list(LedDeviceInfo ledDeviceInfo)
    {
        startPage();
```

```
        List<LedDeviceInfo> list = ledDeviceInfoService.selectLedDeviceInfoList(ledDeviceI
nfo);
        return getDataTable(list);
    }

    /**
     * 导出电子屏设备信息列表
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:export')")
    @Log(title = "电子屏设备信息", businessType = BusinessType.EXPORT)
    @PostMapping("/export")
    public void export(HttpServletResponse response, LedDeviceInfo ledDeviceInfo)
    {
        List<LedDeviceInfo> list = ledDeviceInfoService.selectLedDeviceInfoList(ledDeviceI
nfo);
        ExcelUtil<LedDeviceInfo> util = new ExcelUtil<LedDeviceInfo>(LedDeviceInfo.clas
s);
        util.exportExcel(response, list, "电子屏设备信息数据");
    }

    /**
     * 获取电子屏设备信息详细信息
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:query')")
    @GetMapping(value = "/{id}")
    public AjaxResult getInfo(@PathVariable("id") Long id)
    {
        return success(ledDeviceInfoService.selectLedDeviceInfoById(id));
    }

    /**
     * 新增电子屏设备信息
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:add')")
    @Log(title = "电子屏设备信息", businessType = BusinessType.INSERT)
    @PostMapping
    public AjaxResult add(@RequestBody LedDeviceInfo ledDeviceInfo)
    {
        return toAjax(ledDeviceInfoService.insertLedDeviceInfo(ledDeviceInfo));
    }

    /**
     * 修改电子屏设备信息
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:edit')")
```

```
    @Log(title = "电子屏设备信息", businessType = BusinessType.UPDATE)
    @PutMapping
    public AjaxResult edit(@RequestBody LedDeviceInfo ledDeviceInfo)
    {
        return toAjax(ledDeviceInfoService.updateLedDeviceInfo(ledDeviceInfo));
    }

    /**
     * 删除电子屏设备信息
     */
    @PreAuthorize("@ss.hasPermi('led:ledinfo:remove')")
    @Log(title = "电子屏设备信息", businessType = BusinessType.DELETE)
    @DeleteMapping("/{ids}")
    public AjaxResult remove(@PathVariable Long[] ids)
    {
        return toAjax(ledDeviceInfoService.deleteLedDeviceInfoByIds(ids));
    }
}
```