# 实验四 智能制造系统生产计划管理模块设计与实现

# 1 EV-MES生产计划模块功能分析

## 1.1功能模块划分

**1.1.1 生产计划基础管理模块**

（1）计划信息管理：支持计划编号、关联订单、生产线、开始时间等基本信息管理。

（2）计划状态管理：实现计划状态流转（已计划→进行中→已完成→已取消）。

（3）计划查询功能：提供计划列表展示，支持分页显示和计划编号/客户名称模糊搜索。

（4）计划编辑功能：支持编辑现有计划的所有字段信息。

（5）计划删除功能：提供计划删除操作，支持数据完整性保护。

**1.1.2生产计划关联管理模块**

（1）订单关联：与订单管理系统建立关联关系，确保计划与需求匹配。

（2）库存关联：考虑库存物料情况制定生产计划。

**1.1.3库存可视化模块**

（1）统计图表展示：提供计划状态分布、生产线分布等图表。

（2）数据汇总展示：以卡片形式展示关键生产指标。

## 1.2系统功能图

如图1-1所示，生产计划管理功能依托“基础-关联-可视化”三模块闭环运作：基础模块完成计划创建、状态流转及增删改查，确保排产信息完整可控；关联模块同步订单需求与库存现状，实现按需排产、缺料预警；可视化模块以卡片和图表实时呈现计划状态分布与产能负荷，保障生产计划正确执行。



图1-1 系统功能图

# 2 EV-MES生产计划管理功能流程图

如图2-1所示，以生产计划全生命周期为主线：接收订单需求后，即时联动库存校验物料齐套、评估产能，自动检测时间冲突，确认后的计划直接进入执行监控环节，最终汇总产出效率统计，实现计划“制定—校验—优化—执行—评价”一体化闭环，确保生产计划的科学有效。

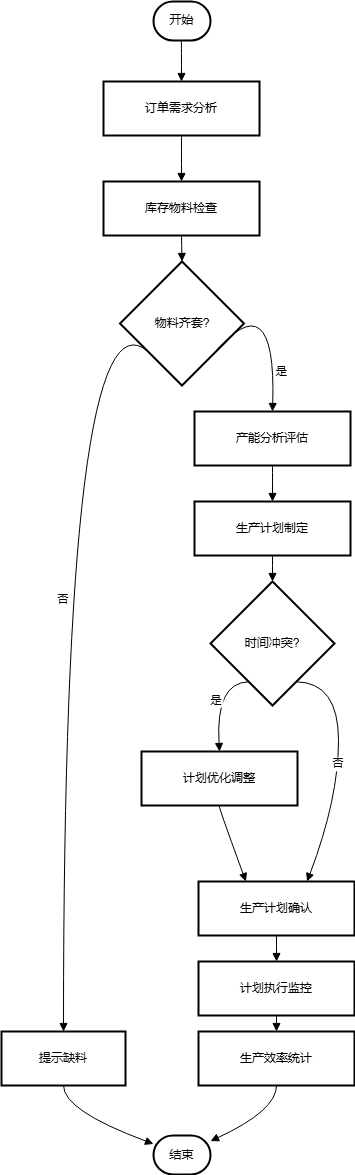


图2-1 功能流程图

# 3库存管理功能设计与实现

## 3.1 数据表设计

表3-1 production\_plans表

|  |  |  |
| --- | --- | --- |
| 字段 | 类型 | 约束及含义 |
| id  plan\_code | Integer  String(50) | 主键，自增  计划编号 |
| order\_id  line  start\_time  end\_time  status  created\_at  updated\_at | Integer  String(20)  DateTime  DateTime  String(20)  DateTime  DateTime | 关联订单ID  生产线  开始时间  结束时间  计划状态 创建时间  更新时间 |

## 3.2生产计划列表功能实现

（1）实现图

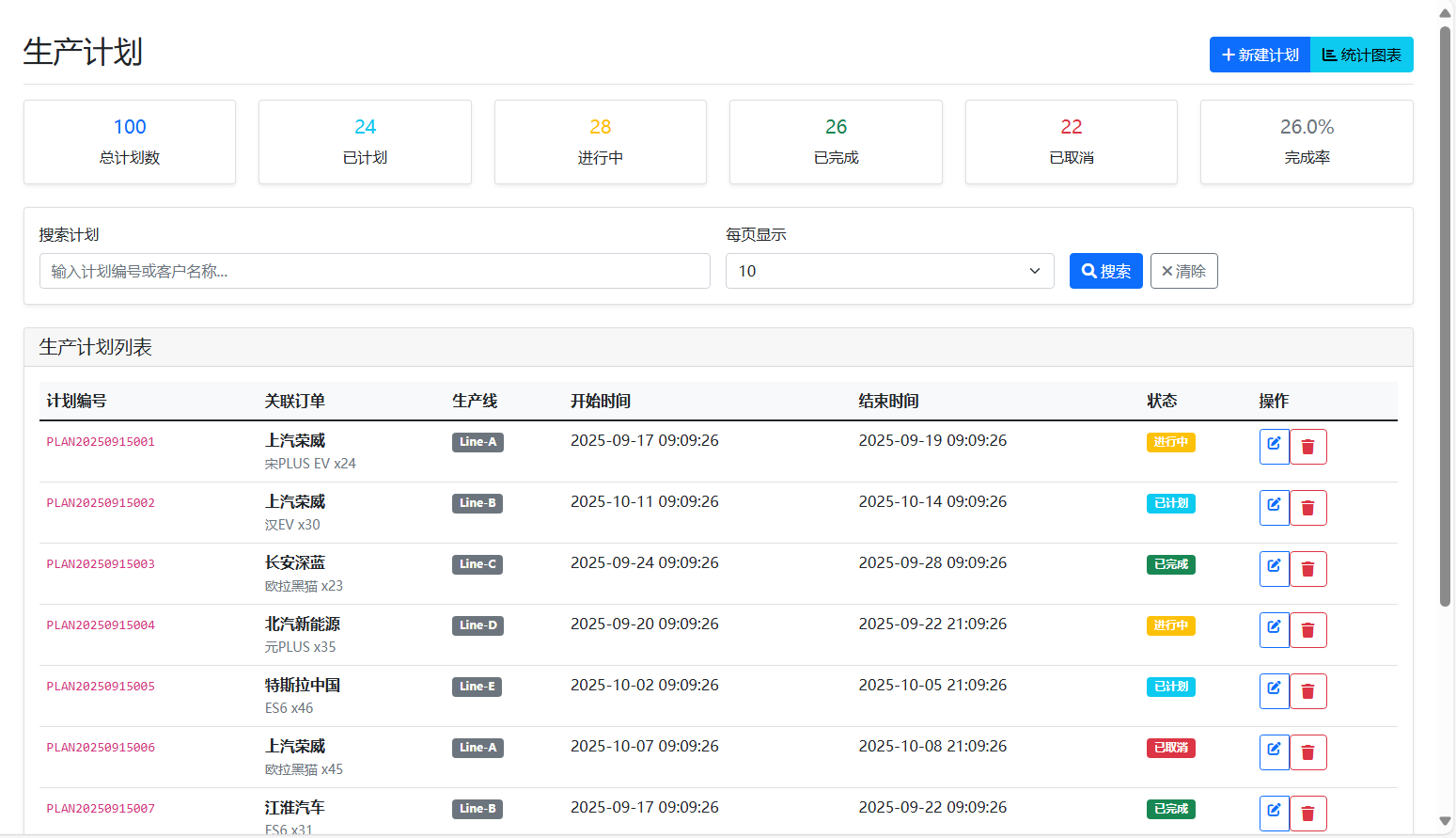


图3-1 生产计划列表功能

（2）核心代码

|  |
| --- |
| @production\_bp.route('/')  def page\_production\_list():  try:  db = session\_factory()  production\_service = ProductionService(db)    page = int(request.args.get('page', 1))  per\_page = int(request.args.get('per\_page', 10))  search = request.args.get('search', '')    result = production\_service.get\_plans(page=page, per\_page=per\_page, search=search)    stats = production\_service.get\_production\_statistics()    return render\_template('production/list.html',  plans=result['plans'],  pagination=result,  search=search,  stats=stats,  status\_options=PRODUCTION\_STATUS)  except Exception as e:  flash(f'获取生产计划列表失败: {str(e)}', 'error')  return render\_template('production/list.html', plans=[], pagination={}, search='', stats={}, status\_options=PRODUCTION\_STATUS)  finally:  db.close() |

## 3.3 编辑库存功能实现

（1）实现图

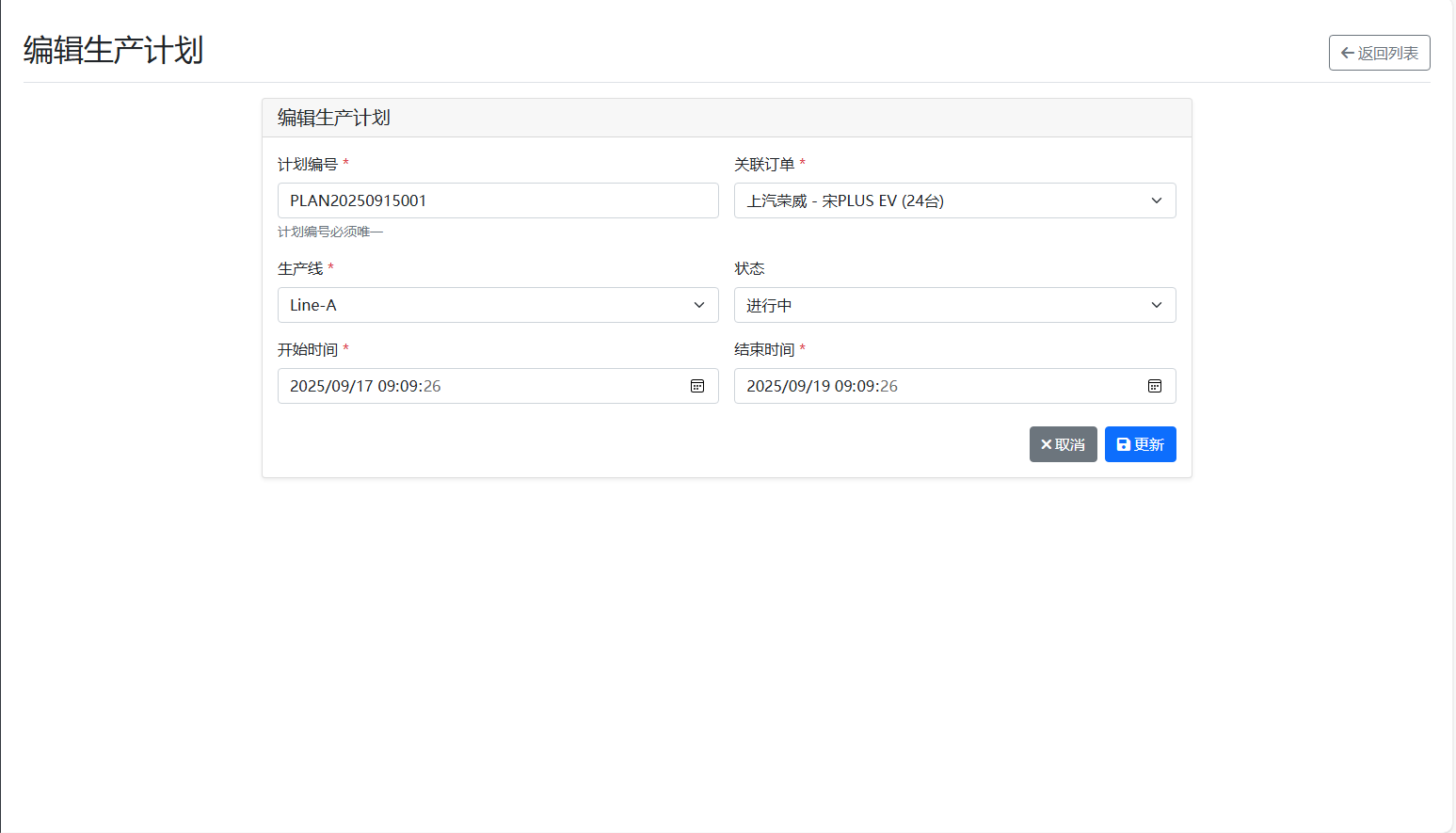


图3-2 编辑生产计划功能

（2）核心代码

|  |
| --- |
| @production\_bp.route('/<int:plan\_id>/edit', methods=['GET', 'POST'])  def page\_production\_edit(plan\_id):  try:  db = session\_factory()  production\_service = ProductionService(db)  order\_service = OrderService(db)    if request.method == 'GET':  plan = production\_service.get\_plan\_by\_id(plan\_id)  if not plan:  flash('生产计划不存在', 'error')  return redirect(url\_for('production.page\_production\_list'))    orders = order\_service.get\_orders(page=1, per\_page=1000)['orders']    return render\_template('production/form.html',  plan=plan.to\_dict(),  orders=orders,  status\_options=PRODUCTION\_STATUS)    plan\_data = {  'plan\_code': request.form.get('plan\_code'),  'order\_id': int(request.form.get('order\_id', 0)),  'line': request.form.get('line'),  'start\_time': request.form.get('start\_time'),  'end\_time': request.form.get('end\_time'),  'status': request.form.get('status')  }    plan = production\_service.update\_plan(plan\_id, plan\_data)  if not plan:  flash('生产计划不存在', 'error')  return redirect(url\_for('production.page\_production\_list'))    flash('生产计划更新成功', 'success')  return redirect(url\_for('production.page\_production\_list'))    except Exception as e:  flash(f'更新生产计划失败: {str(e)}', 'error')  return redirect(url\_for('production.page\_production\_edit', plan\_id=plan\_id))  finally:  db.close() |

## 3.4 可视化统计功能实现

（1）实现图

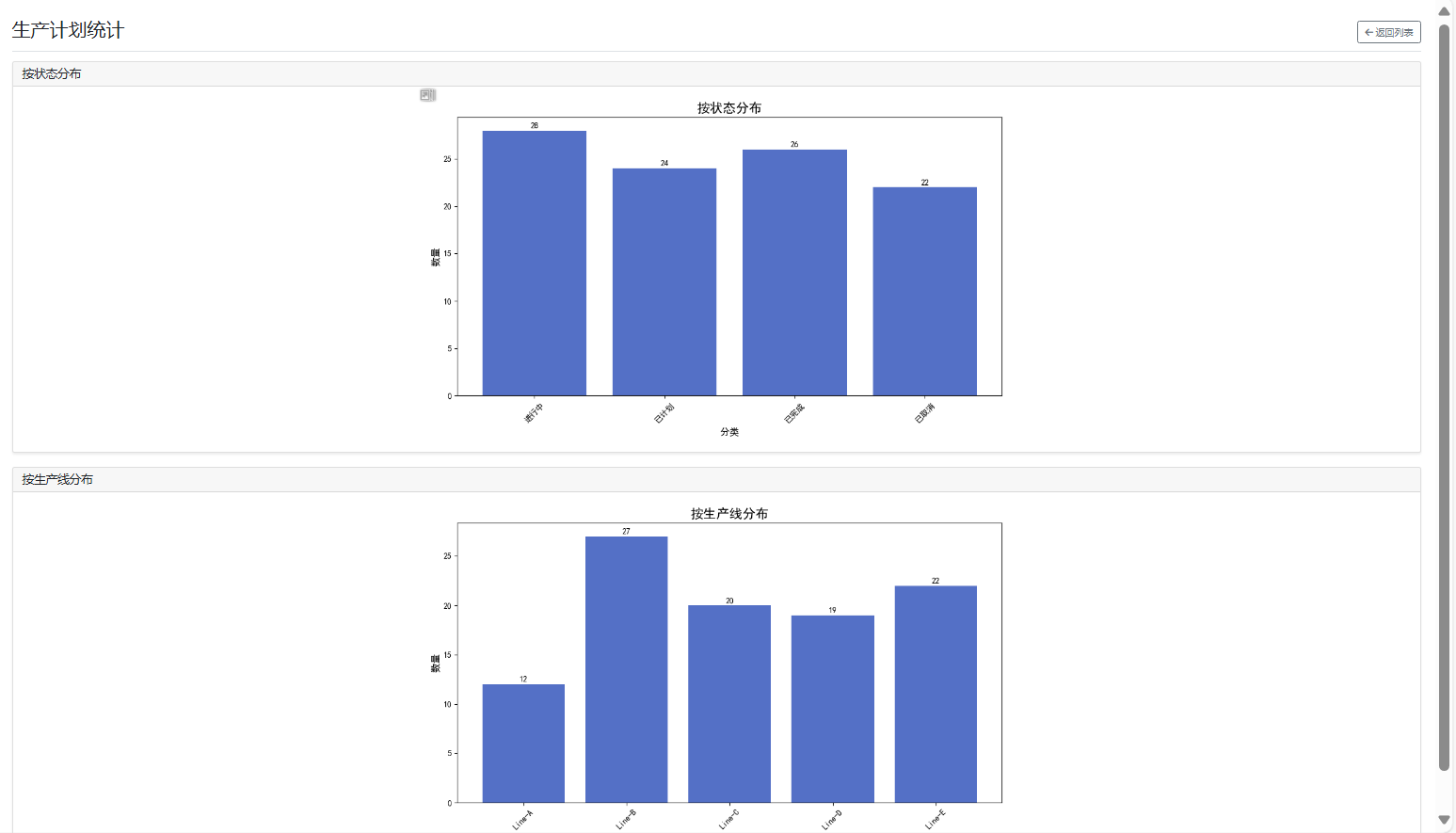


图3-3 可视化统计功能

（2）核心代码

|  |
| --- |
| @production\_bp.route('/gantt')  def page\_production\_gantt():  try:  db = session\_factory()  production\_service = ProductionService(db)    result = production\_service.get\_plans(page=1, per\_page=1000)  plans = result['plans']    charts = create\_gantt\_chart(plans)    return render\_template('production/gantt.html',  status\_chart=charts.get('status\_chart', ''),  line\_chart=charts.get('line\_chart', ''),  plans=plans)    except Exception as e:  flash(f'获取统计图表失败: {str(e)}', 'error')  return render\_template('production/gantt.html',  status\_chart='',  line\_chart='',  plans=[])  finally:  db.close()  def create\_gantt\_chart(plans):  try:  from src.utils.matplotlib\_charts import MatplotlibCharts  from src.utils.status\_mapping import StatusMapping    if not plans:  return {'status\_chart': '', 'line\_chart': ''}    status\_stats = {}  line\_stats = {}    for plan in plans:  status = plan['status']  line = plan['line']    if status not in status\_stats:  status\_stats[status] = 0  status\_stats[status] += 1    if line not in line\_stats:  line\_stats[line] = 0  line\_stats[line] += 1    translated\_status\_stats = StatusMapping.translate\_status\_dict(status\_stats)  translated\_line\_stats = StatusMapping.translate\_line\_dict(line\_stats)    status\_chart = MatplotlibCharts.create\_bar\_chart(  title='按状态分布',  data=translated\_status\_stats  )    line\_chart = MatplotlibCharts.create\_bar\_chart(  title='按生产线分布',  data=translated\_line\_stats  )    return {  'status\_chart': status\_chart,  'line\_chart': line\_chart  }    except Exception as e:  print(f"创建生产计划图表失败: {e}")  return {'status\_chart': '', 'line\_chart': ''} |