天筹算法集成设计方案 (简化版)

一、项目概述

1.1 算法系统简介

天筹(Tianchou)智能制造决策系统是一个**双轨多目标优化**平台，包含两个核心模块：

Part 1: 技术优化 (NSGA-II遗传算法)

模式	适用行业	优化目标	输入参数	输出指标
轻工业模式	纺织、服装、家具	车间设备布局优化	车间尺寸、设备数量、搬运频率矩阵、产品线信息	搬运成本(f1)、设备移动成本(f2)、空间利用率(f3)
重工业模式	汽车制造、机械加工	AGV调度路径优化	工位坐标、AGV数量、任务分配	完工时间(f1)、瓶颈利用率(f2)

Part 2: 商业决策 (AHP-TOPSIS)

- **代表性方案筛选**: 低成本、短工期、高收益、性价比、中心点
- **AHP权重计算**: 用户交互式两两比较，一致性检验
- **TOPSIS综合评分**: 多准则决策排序，推荐最优方案

1.2 简化说明

与原方案的区别:

- ❌ 移除 Celery 异步任务队列
- ❌ 移除 Redis 缓存
- ✅ 使用 FastAPI BackgroundTasks 处理优化任务
- ✅ 保留所有数据库模型和表
- ✅ 保留所有 API 端点
- ✅ 算法模块完全基于 test6 代码结构

二、后端设计

2.1 依赖安装

```
# 在 backend 环境中安装算法依赖
cd backend
uv pip install deap numpy matplotlib pandas tqdm pymoo
```

2.2 目录结构

```
backend/app/
├── algorithms/                    # 算法模块 (基于test6)
│   ├── __init__.py
│   ├── part1_optimization.py      # Part 1: 技术优化 (NSGA-II)
│   ├── part2_decision.py          # Part 2: 商业决策 (AHP-TOPSIS)
│   ├── scheme_translator.py       # 技术->商业价值转换
│   └── visualizer.py              # 结果可视化工具
├── api/
│   └── routes/
│       └── tianchou.py            # 天筹API路由
└── models.py                      # 数据库模型 (添加天筹相关模型)
```

2.3 数据库模型设计

在 `backend/app/models.py` 中添加以下模型:

```
from enum import Enum
from sqlalchemy import Column
from sqlalchemy.dialects.postgresql import JSONB

class IndustryType(str, Enum):
    LIGHT = "light"    # 轻工业
    HEAVY = "heavy"    # 重工业

class TaskStatus(str, Enum):
    PENDING = "pending"
    RUNNING = "running"
    COMPLETED = "completed"
    FAILED = "failed"

class OptimizationTask(SQLModel, table=True):
    """优化任务主表"""
    __tablename__ = "optimization_tasks"

    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
    name: str = Field(index=True, max_length=255)
```

```
    industry_type: IndustryType

    # 输入参数 (JSON存储)
    input_params: dict = Field(default={}, sa_column=Column(JSONB))

    # 任务状态
    status: TaskStatus = Field(default=TaskStatus.PENDING)
    progress: int = Field(default=0)  # 0-100

    # 结果摘要
    pareto_solution_count: int = Field(default=0)
    recommended_solution_id: uuid.UUID | None = Field(default=None)

    # 商业决策权重
    weights_cost: float | None = Field(default=None)
    weights_time: float | None = Field(default=None)
    weights_benefit: float | None = Field(default=None)

    # 元数据
    created_at: datetime = Field(default_factory=datetime.utcnow)
    started_at: datetime | None = None
    completed_at: datetime | None = None
    created_by: uuid.UUID | None = Field(default=None,
foreign_key="users.id")

    # 关联关系
    solutions: list["ParetoSolution"] = Relationship(back_populates="task",
cascade_delete=True)
    decisions: list["DecisionRecord"] = Relationship(back_populates="task",
cascade_delete=True)

class ParetoSolution(SQLModel, table=True):
    """帕累托最优解"""
    __tablename__ = "pareto_solutions"

    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
    task_id: uuid.UUID = Field(foreign_key="optimization_tasks.id",
index=True)

    # 技术指标 (Part 1)
    f1: float  # 目标1
    f2: float  # 目标2
    f3: float | None = None  # 目标3 (轻工业)
```

```
    # 商业指标 (Part 2)
    total_cost: float = Field(default=0)
    implementation_days: float = Field(default=0)
    expected_benefit: float = Field(default=0)

    # 方案详情 (JSON)
    solution_data: dict = Field(default={}, sa_column=Column(JSONB))

    # 设备/路径方案 (JSON)
    technical_details: dict = Field(default={}, sa_column=Column(JSONB))

    # 排名和评分
    rank: int = Field(default=0)
    topsis_score: float | None = None

    # 关联
    task: OptimizationTask = Relationship(back_populates="solutions")

class DecisionRecord(SQLModel, table=True):
    """决策记录 (AHP-TOPSIS)"""
    __tablename__ = "decision_records"

    id: uuid.UUID = Field(default_factory=uuid.uuid4, primary_key=True)
    task_id: uuid.UUID = Field(foreign_key="optimization_tasks.id",
index=True)

    # AHP判断矩阵
    ahp_matrix: dict = Field(default={}, sa_column=Column(JSONB))

    # 计算权重
    weights: dict = Field(default={}, sa_column=Column(JSONB))
    consistency_ratio: float = Field(default=0)

    # TOPSIS结果
    best_solution_id: uuid.UUID | None = None
    decision_scores: dict = Field(default={}, sa_column=Column(JSONB))

    created_at: datetime = Field(default_factory=datetime.utcnow)

    task: OptimizationTask = Relationship(back_populates="decisions")
```

2.4 算法模块设计 (基于test6)

2.4.1 Part 1: 技术优化模块

`backend/app/algorithms/part1_optimization.py` - 直接从 test6 迁移核心代码

主要类和函数：

- `DualTrackAlgorithm`：双轨算法主类
- `run_light_industry_optimization()`：轻工业优化
- `run_heavy_industry_optimization()`：重工业优化

2.4.2 Part 2: 商业决策模块

`backend/app/algorithms/part2_decision.py` - 直接从 test6 迁移核心代码

主要函数：

- `select_representative_solutions()`：筛选代表性方案
- `ahp_weight_calculation()`：AHP权重计算
- `topsis_ranking()`：TOPSIS排序

2.4.3 转换层模块

`backend/app/algorithms/scheme_translator.py` - 从 test6/main_tianchou.py 提取

```python
class SchemeTranslator:
    """技术指标 -> 商业指标转换器"""

    def __init__(self, industry_type: str, params: dict):
        self.industry_type = industry_type
        self.P_unit = params.get('P_unit', 20000)
        self.base_cost = params.get('base_cost', 50000)
        # ... 其他参数

    def translate(self, technical_solutions: list) -> tuple:
        """
        将技术优化结果转换为商业指标

        Returns:
            (business_data, original_indices)
        """
        # 实现逻辑参考 test6/main_tianchou.py
        pass
```

2.5 API路由设计

`backend/app/api/routes/tianchou.py`：

```python
from fastapi import APIRouter, Depends, HTTPException, BackgroundTasks
from sqlmodel import Session, select
from app.core.db import get_session
from app.models import OptimizationTask, ParetoSolution, DecisionRecord,
TaskStatus
from app.algorithms import part1_optimization, part2_decision,
scheme_translator
import numpy as np

router = APIRouter(prefix="/api/v1/tianchou", tags=["天筹优化"])

# ============ 请求/响应模型 ============

class OptimizationRequest(BaseModel):
    """优化任务请求"""
    name: str
    industry_type: str  # "light" or "heavy"

    # 轻工业参数
    workshop_length: float | None = None
    workshop_width: float | None = None
    device_count: int | None = None
    # ... 其他参数

    # 重工业参数
    station_count: int | None = None
    agv_count: int | None = None
    # ... 其他参数

    # 商业参数
    daily_output_value: float = 20000
    base_cost: float = 20000
    construction_rate: float = 3000

# ============ 后台任务函数 ============

def run_optimization_task(task_id: str, session: Session):
    """后台执行优化任务"""
    try:
        # 1. 更新任务状态
        task = session.get(OptimizationTask, task_id)
```

```
        task.status = TaskStatus.RUNNING
        task.started_at = datetime.utcnow()
        session.commit()

        # 2. 执行 Part 1: 技术优化
        if task.industry_type == "light":
            optimizer = part1_optimization.DualTrackAlgorithm()
            results =
optimizer.run_light_industry_optimization(task.input_params)
        else:
            optimizer = part1_optimization.DualTrackAlgorithm()
            results =
optimizer.run_heavy_industry_optimization(task.input_params)

        pareto_solutions = results['pareto_solutions']

        # 3. 执行商业价值映射
        translator = scheme_translator.SchemeTranslator(
            task.industry_type,
            task.input_params
        )
        business_data, original_indices =
translator.translate(pareto_solutions)

        # 4. 保存帕累托解
        for idx, sol in enumerate(pareto_solutions):
            biz_idx = original_indices.index(idx) if idx in original_indices
else None
            if biz_idx is None:
                continue

            biz = business_data[biz_idx]

            solution = ParetoSolution(
                task_id=task_id,
                f1=sol['f1'],
                f2=sol['f2'],
                f3=sol.get('f3'),
                total_cost=biz[0],
                implementation_days=biz[1],
                expected_benefit=biz[2],
                solution_data=sol,
                technical_details=sol.get('individual', {})
```

```
            )
            session.add(solution)

        # 5. 更新任务状态
        task.status = TaskStatus.COMPLETED
        task.pareto_solution_count = len(pareto_solutions)
        task.completed_at = datetime.utcnow()
        task.progress = 100
        session.commit()

    except Exception as e:
        task.status = TaskStatus.FAILED
        session.commit()
        raise e

# ============ API端点 ============

@router.post("/tasks")
async def create_optimization_task(
    request: OptimizationRequest,
    background_tasks: BackgroundTasks,
    session: Session = Depends(get_session)
):
    """创建新的优化任务 (后台执行)"""

    # 创建任务记录
    task = OptimizationTask(
        name=request.name,
        industry_type=request.industry_type,
        input_params=request.dict()
    )
    session.add(task)
    session.commit()
    session.refresh(task)

    # 添加后台任务
    background_tasks.add_task(run_optimization_task, str(task.id), session)

    return {
        "task_id": str(task.id),
        "name": task.name,
        "status": task.status,
        "progress": task.progress,
```

```
        "created_at": task.created_at
    }

@router.get("/tasks/{task_id}")
async def get_task_status(
    task_id: str,
    session: Session = Depends(get_session)
):
    """获取任务状态和进度"""
    task = session.get(OptimizationTask, task_id)
    if not task:
        raise HTTPException(status_code=404, detail="任务不存在")

    return {
        "task_id": str(task.id),
        "name": task.name,
        "status": task.status,
        "progress": task.progress,
        "solution_count": task.pareto_solution_count,
        "recommended_solution_id": str(task.recommended_solution_id) if
task.recommended_solution_id else None,
        "created_at": task.created_at,
        "started_at": task.started_at,
        "completed_at": task.completed_at
    }

@router.get("/tasks/{task_id}/solutions")
async def get_pareto_solutions(
    task_id: str,
    limit: int = 20,
    session: Session = Depends(get_session)
):
    """获取帕累托最优解列表"""
    statement = select(ParetoSolution).where(
        ParetoSolution.task_id == task_id
    ).limit(limit)
    solutions = session.exec(statement).all()

    return [
        {
            "id": str(s.id),
            "rank": s.rank,
            "f1": s.f1,
```

```
            "f2": s.f2,
            "f3": s.f3,
            "total_cost": s.total_cost,
            "implementation_days": s.implementation_days,
            "expected_benefit": s.expected_benefit,
            "topsis_score": s.topsis_score
        }
        for s in solutions
    ]

@router.post("/tasks/{task_id}/decide/ahp")
async def calculate_ahp_weights(
    task_id: str,
    matrix_01: float,
    matrix_02: float,
    matrix_12: float,
    session: Session = Depends(get_session)
):
    """计算AHP权重"""

    # 构建判断矩阵
    matrix = np.array([
        [1, matrix_01, matrix_02],
        [1/matrix_01, 1, matrix_12],
        [1/matrix_02, 1/matrix_12, 1]
    ])

    # 计算权重
    weights = part2_decision.ahp_weight_calculation(matrix)

    if weights is None:
        raise HTTPException(status_code=400, detail="一致性检验失败")

    # 保存决策记录
    record = DecisionRecord(
        task_id=task_id,
        ahp_matrix={"matrix": matrix.tolist()},
        weights={
            "cost": float(weights[0]),
            "time": float(weights[1]),
            "benefit": float(weights[2])
        }
    )
```

```
    session.add(record)
    session.commit()

    return {
        "weights": {
            "cost": round(float(weights[0]), 4),
            "time": round(float(weights[1]), 4),
            "benefit": round(float(weights[2]), 4)
        }
    }

@router.post("/tasks/{task_id}/decide/topsis")
async def run_topsis_decision(
    task_id: str,
    weights: dict | None = None,
    session: Session = Depends(get_session)
):
    """运行TOPSIS综合评分"""

    # 获取所有方案
    statement = select(ParetoSolution).where(ParetoSolution.task_id ==
task_id)
    solutions = session.exec(statement).all()

    # 构建决策矩阵
    decision_matrix = np.array([
        [s.total_cost, s.implementation_days, s.expected_benefit]
        for s in solutions
    ])

    # 使用默认权重或用户提供的权重
    if weights is None:
        weights_array = np.array([0.33, 0.33, 0.34])
    else:
        weights_array = np.array([
            weights.get('cost', 0.33),
            weights.get('time', 0.33),
            weights.get('benefit', 0.34)
        ])

    # 计算TOPSIS得分
    scores = part2_decision.topsis_ranking(decision_matrix, weights_array)
```

```
    # 更新方案排名
    for i, (solution, score) in enumerate(zip(solutions, scores)):
        solution.topsis_score = float(score)
        solution.rank = i + 1

    session.commit()

    best_idx = np.argmax(scores)

    return {
        "best_solution_id": str(solutions[best_idx].id),
        "scores": [
            {"solution_id": str(s.id), "score": round(float(sc), 4)}
            for s, sc in zip(solutions, scores)
        ]
    }
```

三、数据库迁移

3.1 创建迁移文件

```
cd backend
uv run alembic revision --autogenerate -m "Add tianchou optimization tables"
uv run alembic upgrade head
```

3.2 表结构

- `optimization_tasks`：优化任务表
- `pareto_solutions`：帕累托解表
- `decision_records`：决策记录表

四、前端设计

前端设计保持与原方案一致，主要变化：

1. **轮询机制**: 由于没有 WebSocket，使用定时轮询获取任务状态
2. **进度显示**: 通过 `GET /tasks/{task_id}` 获取 progress 字段

可视化渲染:

- **方案**: 前端使用Recharts + D3.js
- **原因**:
  - 帕累托前沿图使用Recharts Scatter Chart
  - 布局/路径可视化使用D3.js SVG渲染
  - 支持交互式缩放、拖拽

3.1 页面结构

```
frontend/src/pages/
├── Tianchou.tsx                    # 天筹主页面
├── Tianchou/
│   ├── index.tsx                   # 页面入口
│   ├── components/
│   │   ├── TaskConfigForm.tsx      # 任务配置表单
│   │   ├── TaskProgress.tsx        # 任务进度条
│   │   ├── ParetoFrontChart.tsx    # 帕累托前沿图
│   │   ├── SolutionCard.tsx        # 方案卡片
│   │   ├── SolutionDetailModal.tsx # 方案详情弹窗
│   │   ├── LayoutVisualizer.tsx    # 布局可视化（轻工业）
│   │   ├── AGVVisualizer.tsx       # AGV路径可视化（重工业）
│   │   ├── AHPWizard.tsx           # AHP权重向导
│   │   ├── ComparisonTable.tsx     # 方案对比表
│   │   └── RecommendationPanel.tsx # 推荐方案面板
│   ├── hooks/
│   │   ├── useTianchou.ts          # 天筹状态管理
│   │   └── useOptimization.ts      # 优化任务Hook
│   ├── services/
│   │   └── tianchouService.ts      # API调用服务
│   └── types/
│       └── tianchou.ts             # 类型定义
```

3.2 类型定义

```typescript
// frontend/src/pages/Tianchou/types/tianchou.ts

export enum IndustryType {
  LIGHT = 'light',
  HEAVY = 'heavy'
}

export enum TaskStatus {
  PENDING = 'pending',
```

```
  RUNNING = 'running',
  COMPLETED = 'completed',
  FAILED = 'failed'
}

export interface OptimizationTask {
  id: string;
  name: string;
  industry_type: IndustryType;
  status: TaskStatus;
  progress: number;
  pareto_solution_count: number;
  recommended_solution_id?: string;
  created_at: string;
  started_at?: string;
  completed_at?: string;
}

export interface ParetoSolution {
  id: string;
  rank: number;
  // 技术指标
  f1: number;
  f2: number;
  f3?: number;
  // 商业指标
  total_cost: number;
  implementation_days: number;
  expected_benefit: number;
  topsis_score?: number;
  // 详情
  details: {
    moved_devices?: Array<{
      device_id: number;
      device_name: string;
      original_position: [number, number];
      new_position: [number, number];
      distance: number;
      cost: number;
    }>;
    agv_routes?: Array<{
      agv_id: number;
      route: Array<[number, number]>;
```

```
      completion_time: number;
    }>;
  };
}

export interface AHPWeights {
  cost: number;
  time: number;
  benefit: number;
}

export interface RepresentativeSolutions {
  min_cost: ParetoSolution;
  min_time: ParetoSolution;
  max_benefit: ParetoSolution;
  best_overall: ParetoSolution;
}
```

3.3 API服务

```
// frontend/src/pages/Tianchou/services/tianchouService.ts

import axios from 'axios';
import type {
  OptimizationTask,
  ParetoSolution,
  AHPWeights,
  RepresentativeSolutions
} from '../types';

const api = axios.create({
  baseURL: '/api/v1/tianchou',
});

export const tianchouService = {
  // 创建优化任务
  async createTask(params: {
    name: string;
    industry_type: string;
    // 轻工业参数
    workshop_length?: number;
    workshop_width?: number;
    device_count?: number;
```

```
    // 重工业参数
    station_count?: number;
    agv_count?: number;
    // 商业参数
    daily_output_value?: number;
    base_cost?: number;
  }): Promise<OptimizationTask> {
    const { data } = await api.post('/tasks', params);
    return data;
  },

  // 获取任务状态
  async getTaskStatus(taskId: string): Promise<OptimizationTask> {
    const { data } = await api.get(`/tasks/${taskId}`);
    return data;
  },

  // 获取帕累托解列表
  async getSolutions(
    taskId: string,
    limit = 20
  ): Promise<ParetoSolution[]> {
    const { data } = await api.get(`/tasks/${taskId}/solutions`, {
      params: { limit }
    });
    return data;
  },

  // 获取方案详情
  async getSolutionDetail(taskId: string, solutionId: string):
Promise<ParetoSolution> {
    const { data } = await
api.get(`/tasks/${taskId}/solutions/${solutionId}`);
    return data;
  },

  // 计算AHP权重
  async calculateAHP(
    taskId: string,
    matrix: { m01: number; m02: number; m12: number }
  ): Promise<{ weights: AHPWeights; consistency_ratio: number; is_valid:
boolean }> {
    const { data } = await api.post(`/tasks/${taskId}/decide/ahp`, matrix);
```

```
    return data;
  },

  // 运行TOPSIS评分
  async runTOPSIS(
    taskId: string,
    weights?: AHPWeights
  ): Promise<{ best_solution_id: string; scores: Array<{ solution_id:
string; score: number }> }> {
    const { data } = await api.post(`/tasks/${taskId}/decide/topsis`,
weights);
    return data;
  },

  // 获取任务总结
  async getTaskSummary(taskId: string): Promise<{
    task: OptimizationTask;
    representative_solutions: RepresentativeSolutions;
  }> {
    const { data } = await api.get(`/tasks/${taskId}/summary`);
    return data;
  }
};
```

3.4 页面布局设计

```
// frontend/src/pages/Tianchou/index.tsx

import React, { useState, useEffect, useCallback } from 'react';
import { useSearchParams } from 'react-router-dom';
import { TaskConfigForm } from './components/TaskConfigForm';
import { TaskProgress } from './components/TaskProgress';
import { ParetoFrontChart } from './components/ParetoFrontChart';
import { SolutionCard } from './components/SolutionCard';
import { AHPWizard } from './components/AHPWizard';
import { RecommendationPanel } from './components/RecommendationPanel';
import { LayoutVisualizer } from './components/LayoutVisualizer';
import { useTianchou } from './hooks/useTianchou';
import { tianchouService } from './services/tianchouService';
import { TaskStatus, type ParetoSolution } from './types';

const TianchouPage: React.FC = () => {
  const [searchParams] = useSearchParams();
```

```
  const {
    task,
    setTask,
    solutions,
    setSolutions,
    selectedSolution,
    setSelectedSolution,
    ahpWeights,
    setAhpWeights
  } = useTianchou();

  const [view, setView] = useState<'config' | 'optimizing' | 'results'>
('config');
  const [showAHPWizard, setShowAHPWizard] = useState(false);

  // 创建优化任务
  const handleCreateTask = useCallback(async (params: any) => {
    try {
      const newTask = await tianchouService.createTask(params);
      setTask(newTask);
      setView('optimizing');

      // 开始轮询任务状态
      pollTaskStatus(newTask.id);
    } catch (error) {
      console.error('创建任务失败:', error);
    }
  }, [setTask]);

  // 轮询任务状态
  const pollTaskStatus = async (taskId: string) => {
    const poll = async () => {
      const status = await tianchouService.getTaskStatus(taskId);
      setTask(status);

      if (status.status === TaskStatus.RUNNING) {
        setTimeout(poll, 2000);
      } else if (status.status === TaskStatus.COMPLETED) {
        // 加载方案列表
        const sols = await tianchouService.getSolutions(taskId);
        setSolutions(sols);
        setView('results');
      } else if (status.status === TaskStatus.FAILED) {
```

```
        // 处理失败
        console.error('任务执行失败');
      }
    };
    poll();
  };

  // 选择方案查看详情
  const handleSelectSolution = async (solution: ParetoSolution) => {
    const detail = await tianchouService.getSolutionDetail(task!.id,
solution.id);
    setSelectedSolution(detail);
  };

  // 运行AHP-TOPSIS决策
  const handleRunDecision = async (weights: AHPWeights) => {
    setAhpWeights(weights);
    const result = await tianchouService.runTOPSIS(task!.id, weights);
    // 更新方案排名
    // ...
  };

  return (
    <div className="min-h-screen bg-gray-50 p-6">
      <header className="mb-6">
        <h1 className="text-3xl font-bold text-gray-900">天筹优化决策系统</h1>
        <p className="text-gray-600 mt-2">基于多目标遗传算法的智能制造优化方案
</p>
      </header>

      <main className="max-w-7xl mx-auto">
        {/* 配置阶段 */}
        {view === 'config' && (
          <TaskConfigForm onSubmit={handleCreateTask} />
        )}

        {/* 优化执行阶段 */}
        {view === 'optimizing' && task && (
          <TaskProgress
            task={task}
            onCancel={() => setView('config')}
          />
        )}
```

```
        {/* 结果展示阶段 */}
        {view === 'results' && task && (
          <div className="grid grid-cols-12 gap-6">
            {/* 左侧：帕累托前沿图 */}
            <div className="col-span-8">
              <div className="bg-white rounded-lg shadow p-6">
                <h2 className="text-xl font-semibold mb-4">帕累托最优解集</h2>
                <ParetoFrontChart
                  solutions={solutions}
                  onSelect={handleSelectSolution}
                  selectedId={selectedSolution?.id}
                />
              </div>

              {/* 布局可视化（轻工业） */}
              {task.industry_type === 'light' && selectedSolution && (
                <div className="mt-6 bg-white rounded-lg shadow p-6">
                  <h2 className="text-xl font-semibold mb-4">车间布局方案
</h2>
                  <LayoutVisualizer
                    solution={selectedSolution}
                    originalLayout={/* 原始布局数据 */}
                  />
                </div>
              )}

              {/* AGV路径可视化（重工业） */}
              {task.industry_type === 'heavy' && selectedSolution && (
                <div className="mt-6 bg-white rounded-lg shadow p-6">
                  <h2 className="text-xl font-semibold mb-4">AGV调度路径</h2>
                  <AGVVisualizer solution={selectedSolution} />
                </div>
              )}
            </div>

            {/* 右侧：方案列表和决策面板 */}
            <div className="col-span-4 space-y-6">
              {/* 推荐面板 */}
              <RecommendationPanel
                task={task}
                solutions={solutions}
                onRunAHP={() => setShowAHPWizard(true)}
```

```
                />

                {/* 方案列表 */}
                <div className="bg-white rounded-lg shadow p-6">
                  <h2 className="text-xl font-semibold mb-4">候选方案列表</h2>
                  <div className="space-y-3 max-h-96 overflow-y-auto">
                    {solutions.slice(0, 10).map(solution => (
                      <SolutionCard
                        key={solution.id}
                        solution={solution}
                        isSelected={selectedSolution?.id === solution.id}
                        onClick={() => handleSelectSolution(solution)}
                      />
                    ))}
                  </div>
                </div>
              </div>
            </div>
          )}
        </main>

        {/* AHP向导弹窗 */}
        {showAHPWizard && (
          <AHPWizard
            onComplete={handleRunDecision}
            onClose={() => setShowAHPWizard(false)}
          />
        )}
      </div>
    );
};

export default TianchouPage;
```

3.5 关键组件设计

帕累托前沿图

```
// frontend/src/pages/Tianchou/components/ParetoFrontChart.tsx

import React, { useMemo } from 'react';
import {
  ScatterChart,
```

```
  Scatter,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  ResponsiveContainer,
  Legend
} from 'recharts';
import type { ParetoSolution } from '../types';

interface Props {
  solutions: ParetoSolution[];
  onSelect: (solution: ParetoSolution) => void;
  selectedId?: string;
}

const ParetoFrontChart: React.FC<Props> = ({ solutions, onSelect, selectedId
}) => {
  // 准备图表数据
  const chartData = useMemo(() => {
    return solutions.map((sol, index) => ({
      ...sol,
      x: sol.total_cost,
      y: sol.implementation_days,
      z: sol.expected_benefit,
      index: index + 1
    }));
  }, [solutions]);

  // 推荐方案高亮
  const recommendedData = chartData.filter(s => s.topsis_score ===
Math.max(...solutions.map(s => s.topsis_score || 0)));
  const otherData = chartData.filter(s => s.topsis_score !==
Math.max(...solutions.map(s => s.topsis_score || 0)));

  return (
    <ResponsiveContainer width="100%" height={400}>
      <ScatterChart>
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis
          type="number"
          dataKey="x"
          name="总成本"
```

```
            unit="元"
            tickFormatter={(v) => `${(v/10000).toFixed(1)}万`}
          />
          <YAxis
            type="number"
            dataKey="y"
            name="工期"
            unit="天"
          />
          <Tooltip
            formatter={(value: any, name: string) => {
              if (name === '总成本') return [`${value.toLocaleString()}元`,
name];
              if (name === '工期') return [`${value.toFixed(1)}天`, name];
              return [value, name];
            }}
            labelFormatter={(label) => `方案 #${label}`}
            content={({ active, payload }) => {
              if (active && payload && payload.length) {
                const data = payload[0].payload;
                return (
                  <div className="bg-white p-3 shadow-lg rounded-lg border">
                    <p className="font-semibold">方案 #{data.index}</p>
                    <p className="text-sm">总成本:
{data.total_cost.toLocaleString()}元</p>
                    <p className="text-sm">工期:
{data.implementation_days.toFixed(1)}天</p>
                    <p className="text-sm">预期收益:
{data.expected_benefit.toLocaleString()}元</p>
                    {data.topsis_score && (
                      <p className="text-sm font-medium text-blue-600">
                        TOPSIS评分: {data.topsis_score.toFixed(4)}
                      </p>
                    )}
                  </div>
                );
              }
              return null;
            }}
          />
          <Legend />
          <Scatter
            name="候选方案"
```

```
            data={otherData}
            fill="#3498db"
            onClick={(data) => onSelect(data as unknown as ParetoSolution)}
            cursor="pointer"
          />
          <Scatter
            name="推荐方案"
            data={recommendedData}
            fill="#e74c3c"
            shape="star"
            onClick={(data) => onSelect(data as unknown as ParetoSolution)}
          />
        </ScatterChart>
      </ResponsiveContainer>
    );
};

export default ParetoFrontChart;
```

AHP权重向导

```
// frontend/src/pages/Tianchou/components/AHPWizard.tsx

import React, { useState } from 'react';
import { Modal } from '@/components/Modal';
import { tianchouService } from '../services/tianchouService';
import type { AHPWeights } from '../types';

interface Props {
  onComplete: (weights: AHPWeights) => void;
  onClose: () => void;
}

const AHPWizard: React.FC<Props> = ({ onComplete, onClose }) => {
  const [step, setStep] = useState(1);
  const [matrix, setMatrix] = useState({ m01: '1', m02: '1', m12: '1' });
  const [result, setResult] = useState<{ weights: AHPWeights;
consistency_ratio: number; is_valid: boolean } | null>(null);
  const [loading, setLoading] = useState(false);

  // 解析输入值
  const parseValue = (v: string): number => {
    if (v.includes('/')) {
```

```
      const [a, b] = v.split('/');
      return parseFloat(a) / parseFloat(b);
    }
    return parseFloat(v);
  };

  // 计算权重
  const handleCalculate = async () => {
    setLoading(true);
    try {
      const res = await tianchouService.calculateAHP('default-task', {
        m01: parseValue(matrix.m01),
        m02: parseValue(matrix.m02),
        m12: parseValue(matrix.m12)
      });
      setResult(res);
      setStep(3);
    } catch (error) {
      console.error('计算失败:', error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal title="AHP权重设定向导" onClose={onClose} size="lg">
      <div className="p-6">
        {/* 步骤指示器 */}
        <div className="flex items-center justify-center mb-8">
          {[1, 2, 3].map(s => (
            <React.Fragment key={s}>
              <div className={`w-10 h-10 rounded-full flex items-center
justify-center ${
                step >= s ? 'bg-blue-600 text-white' : 'bg-gray-200 text-
gray-500'
              }`}>
                {s}
              </div>
              {s < 3 && <div className={`w-16 h-1 ${step > s ? 'bg-blue-600'
: 'bg-gray-200'}`} />}
            </React.Fragment>
          ))}
        </div>
```

```
        {/* 步骤1: 说明 */}
        {step === 1 && (
          <div className="text-center">
            <h3 className="text-xl font-semibold mb-4">欢迎使用AHP权重设定
</h3>
            <p className="text-gray-600 mb-6">
              层次分析法(AHP)帮助您量化决策偏好。请比较以下三要素的重要性:
            </p>
            <div className="grid grid-cols-3 gap-4 text-left bg-gray-50 p-4
rounded-lg">
              <div>
                <span className="font-medium">💰 成本</span>
                <p className="text-sm text-gray-500">方案的实施总成本</p>
              </div>
              <div>
                <span className="font-medium">⏱️ 工期</span>
                <p className="text-sm text-gray-500">方案的实施周期</p>
              </div>
              <div>
                <span className="font-medium">📈 收益</span>
                <p className="text-sm text-gray-500">方案的预期年收益</p>
              </div>
            </div>
            <button
              className="mt-6 px-6 py-2 bg-blue-600 text-white rounded-lg"
              onClick={() => setStep(2)}
            >
              开始设定
            </button>
          </div>
        )}

        {/* 步骤2: 两两比较 */}
        {step === 2 && (
          <div>
            <h3 className="text-xl font-semibold mb-6">请进行两两比较</h3>

            <div className="space-y-6">
              {/* 问题1: 成本 vs 工期 */}
              <div className="bg-gray-50 p-4 rounded-lg">
                <p className="mb-3">
                  相比于<span className="font-medium">工期</span>,
```

```
                  <span className="font-medium">成本</span>有多重要?
                </p>
                <div className="flex items-center gap-4">
                  <input
                    type="text"
                    value={matrix.m01}
                    onChange={(e) => setMatrix({ ...matrix, m01:
e.target.value })}
                    className="flex-1 px-3 py-2 border rounded-lg"
                    placeholder="1-9 或 分数如 1/3"
                  />
                  <select
                    className="px-3 py-2 border rounded-lg"
                    onChange={(e) => setMatrix({ ...matrix, m01:
e.target.value })}
                  >
                    <option value="1">同等重要 (1)</option>
                    <option value="3">稍微重要 (3)</option>
                    <option value="5">明显重要 (5)</option>
                    <option value="7">非常重要 (7)</option>
                    <option value="9">极端重要 (9)</option>
                    <option value="1/3">稍微不重要 (1/3)</option>
                    <option value="1/5">明显不重要 (1/5)</option>
                  </select>
                </div>
              </div>

              {/* 问题2: 成本 vs 收益 */}
              <div className="bg-gray-50 p-4 rounded-lg">
                <p className="mb-3">
                  相比于<span className="font-medium">收益</span>,
                  <span className="font-medium">成本</span>有多重要?
                </p>
                <div className="flex items-center gap-4">
                  <input
                    type="text"
                    value={matrix.m02}
                    onChange={(e) => setMatrix({ ...matrix, m02:
e.target.value })}
                    className="flex-1 px-3 py-2 border rounded-lg"
                  />
                  <select
                    className="px-3 py-2 border rounded-lg"
```

```
                    onChange={(e) => setMatrix({ ...matrix, m02:
e.target.value })}
                  >
                    <option value="1">同等重要 (1)</option>
                    <option value="3">稍微重要 (3)</option>
                    <option value="5">明显重要 (5)</option>
                  </select>
                </div>
              </div>

              {/* 问题3: 工期 vs 收益 */}
              <div className="bg-gray-50 p-4 rounded-lg">
                <p className="mb-3">
                  相比于<span className="font-medium">收益</span>,
                  <span className="font-medium">工期</span>有多重要?
                </p>
                <div className="flex items-center gap-4">
                  <input
                    type="text"
                    value={matrix.m12}
                    onChange={(e) => setMatrix({ ...matrix, m12:
e.target.value })}
                    className="flex-1 px-3 py-2 border rounded-lg"
                  />
                  <select
                    className="px-3 py-2 border rounded-lg"
                    onChange={(e) => setMatrix({ ...matrix, m12:
e.target.value })}
                  >
                    <option value="1">同等重要 (1)</option>
                    <option value="3">稍微重要 (3)</option>
                    <option value="5">明显重要 (5)</option>
                  </select>
                </div>
              </div>

              <div className="flex justify-between mt-6">
                <button
                  className="px-4 py-2 border rounded-lg"
                  onClick={() => setStep(1)}
                >
                  上一步
```

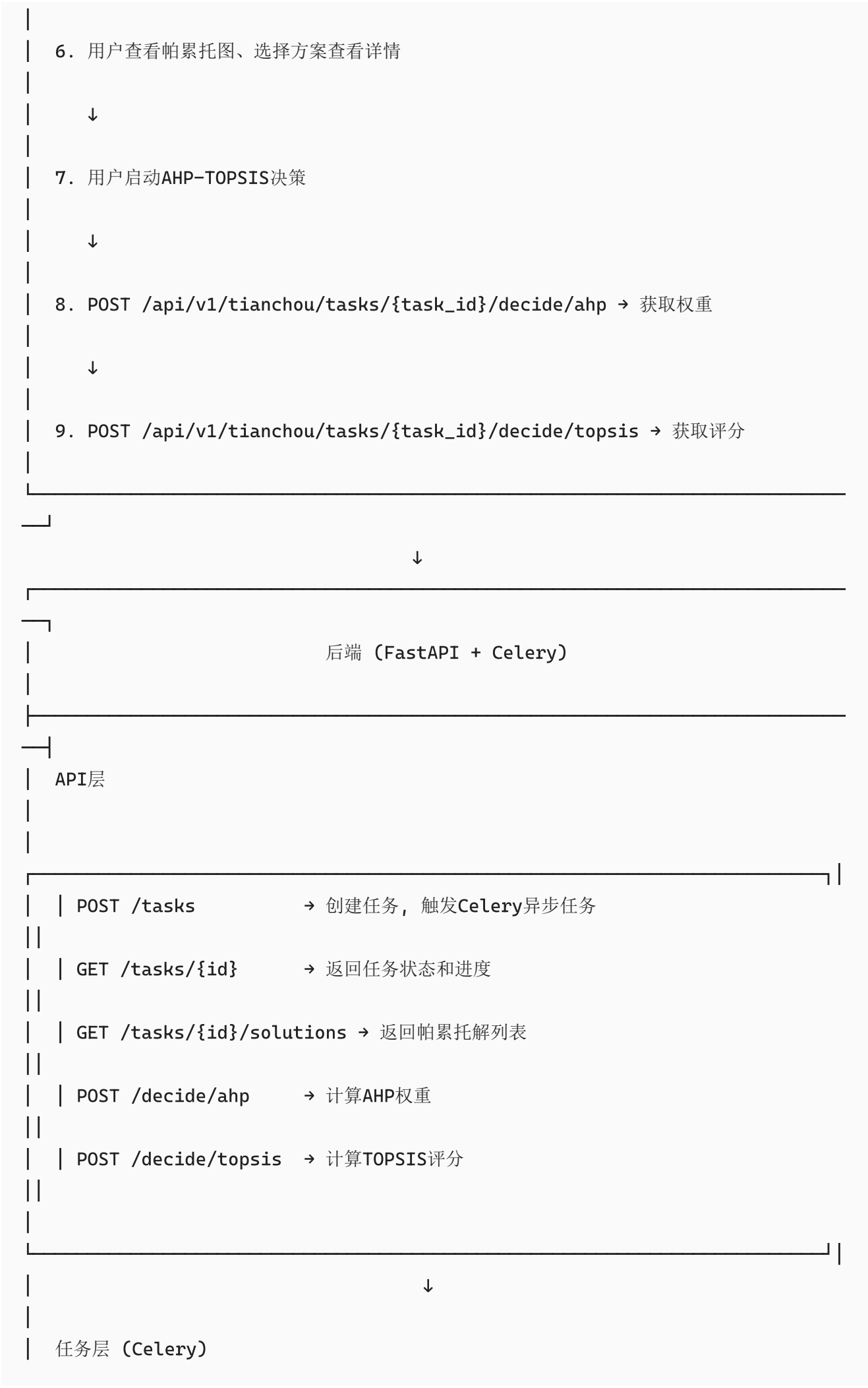
```
              </button>
              <button
                className="px-6 py-2 bg-blue-600 text-white rounded-lg"
                onClick={handleCalculate}
                disabled={loading}
              >
                {loading ? '计算中...' : '计算权重'}
              </button>
            </div>
          </div>
        )}

        {/* 步骤3: 结果 */}
        {step === 3 && result && (
          <div>
            <h3 className="text-xl font-semibold mb-6">计算结果</h3>

            <div className="text-center mb-6">
              <p className="text-gray-600 mb-2">一致性比率 (CR)</p>
              <p className={`text-2xl font-bold ${result.is_valid ? 'text-
green-600' : 'text-red-600'}`}>
                {result.consistency_ratio.toFixed(4)}
              </p>
              <p className={`text-sm ${result.is_valid ? 'text-green-600' :
'text-red-600'}`}>
                {result.is_valid ? '✅ 一致性检验通过' : '❌ 一致性检验失败，请
重新设定'}
              </p>
            </div>

            {result.is_valid && (
              <>
                <div className="bg-gray-50 p-4 rounded-lg mb-6">
                  <p className="font-medium mb-3">最终权重分配：</p>
                  <div className="space-y-2">
                    <div className="flex items-center">
                      <span className="w-20">💰 成本</span>
                      <div className="flex-1 h-4 bg-gray-200 rounded
overflow-hidden">
                        <div
                          className="h-full bg-blue-600"
                          style={{ width: `${result.weights.cost * 100}%` }}
                        />
```

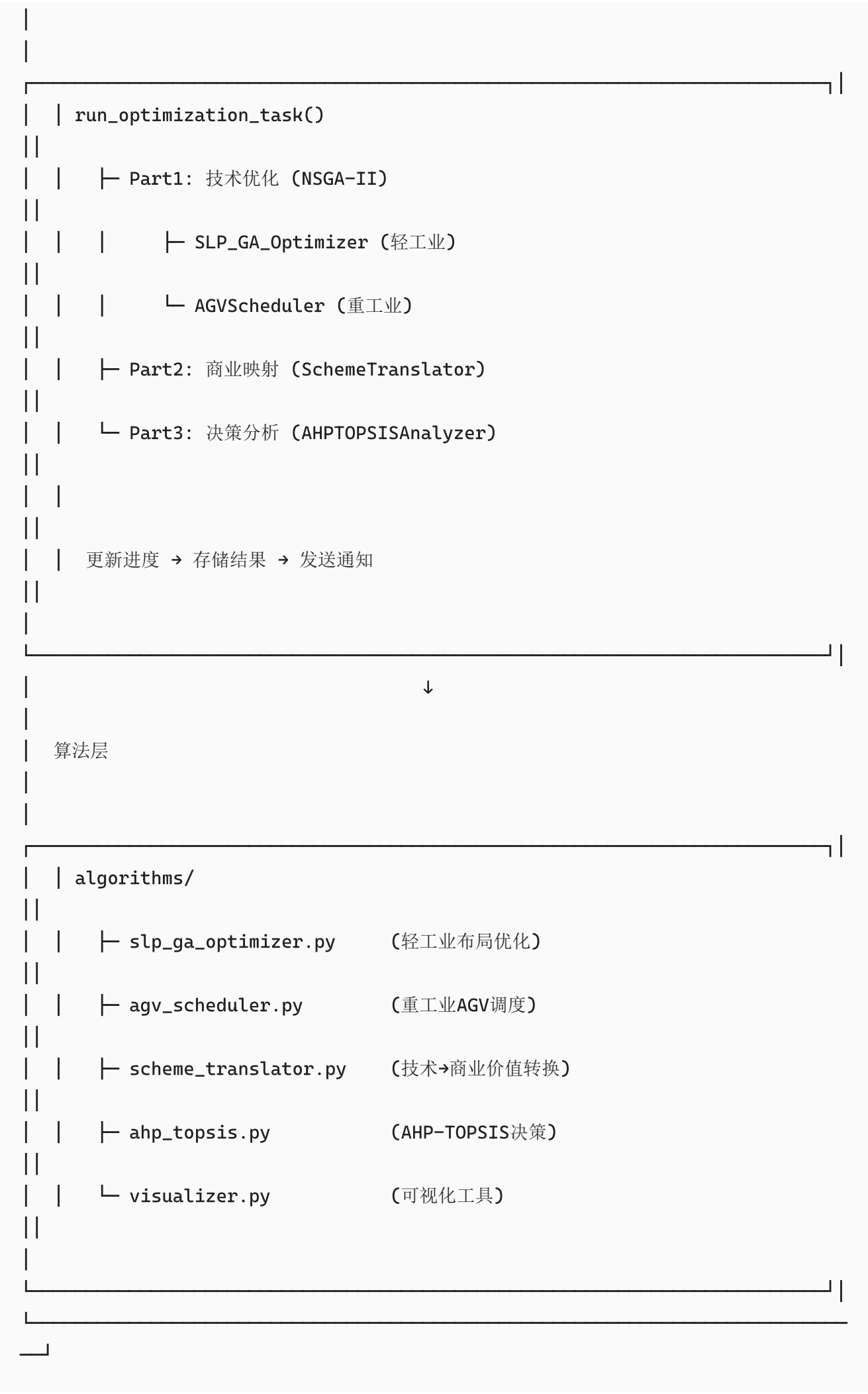
```
                    </div>
                    <span className="w-16 text-right">
{(result.weights.cost * 100).toFixed(1)}%</span>
                  </div>
                  <div className="flex items-center">
                    <span className="w-20">⏱️ 工期</span>
                    <div className="flex-1 h-4 bg-gray-200 rounded
overflow-hidden">
                      <div
                        className="h-full bg-green-600"
                        style={{ width: `${result.weights.time * 100}%` }}
                      />
                    </div>
                    <span className="w-16 text-right">
{(result.weights.time * 100).toFixed(1)}%</span>
                  </div>
                  <div className="flex items-center">
                    <span className="w-20">📈 收益</span>
                    <div className="flex-1 h-4 bg-gray-200 rounded
overflow-hidden">
                      <div
                        className="h-full bg-purple-600"
                        style={{ width: `${result.weights.benefit * 100}%`
}}
                      />
                    </div>
                    <span className="w-16 text-right">
{(result.weights.benefit * 100).toFixed(1)}%</span>
                  </div>
                </div>
              </div>

              <div className="flex justify-end gap-4">
                <button
                  className="px-4 py-2 border rounded-lg"
                  onClick={() => setStep(2)}
                >
                  重新设定
                </button>
                <button
                  className="px-6 py-2 bg-blue-600 text-white rounded-lg"
                  onClick={() => onComplete(result.weights)}
                >
```

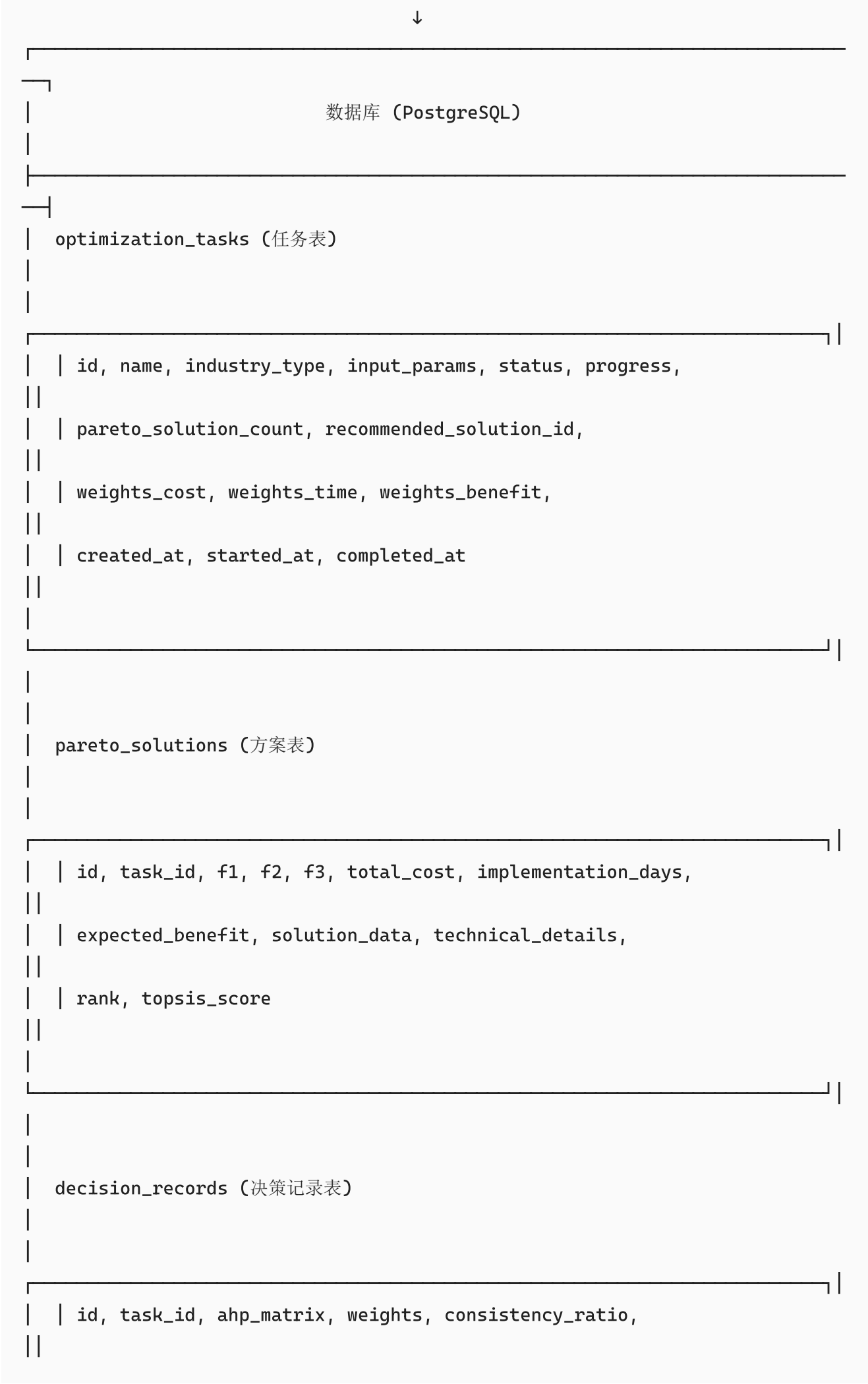
```
                    应用权重并决策
                  </button>
                </div>
              </>
            )}
          </div>
        )}
      </div>
    </Modal>
  );
};

export default AHPWizard;
```

数据流设计

```
┌─────────────────────────────────────────────────────────────────────────┐
│                              前端 (React)                               │
│                                                                         │
├─────────────────────────────────────────────────────────────────────────┤
│  1. 用户填写配置表单                                                    │
│     ↓                                                                   │
│  2. POST /api/v1/tianchou/tasks → 返回 task_id                          │
│     ↓                                                                   │
│  3. 轮询 GET /api/v1/tianchou/tasks/{task_id}                           │
│     ↓                                                                   │
│  4. 状态变为 COMPLETED → 获取方案列表                                   │
│     ↓                                                                   │
│  5. GET /api/v1/tianchou/tasks/{task_id}/solutions                      │
│     ↓                                                                   │
```


6. 用户查看帕累托图、选择方案查看详情
7. 用户启动AHP-TOPSIS决策
8. POST /api/v1/tianchou/tasks/{task_id}/decide/ahp → 获取权重
9. POST /api/v1/tianchou/tasks/{task_id}/decide/topsis → 获取评分
后端 (FastAPI + Celery)
API层
POST /tasks → 创建任务, 触发Celery异步任务
GET /tasks/{id} → 返回任务状态和进度
GET /tasks/{id}/solutions → 返回帕累托解列表
POST /decide/ahp → 计算AHP权重
POST /decide/topsis → 计算TOPSIS评分
任务层 (Celery)


run_optimization_task()
Part1: 技术优化 (NSGA-II)
SLP_GA_Optimizer (轻工业)
AGVScheduler (重工业)
Part2: 商业映射 (SchemeTranslator)
Part3: 决策分析 (AHPTOPSISAnalyzer)
更新进度 → 存储结果 → 发送通知
算法层
algorithms/
slp_ga_optimizer.py (轻工业布局优化)
agv_scheduler.py (重工业AGV调度)
scheme_translator.py (技术→商业价值转换)
ahp_topsis.py (AHP-TOPSIS决策)
visualizer.py (可视化工具)


数据库 (PostgreSQL)
optimization_tasks (任务表)
id, name, industry_type, input_params, status, progress,
pareto_solution_count, recommended_solution_id,
weights_cost, weights_time, weights_benefit,
created_at, started_at, completed_at
pareto_solutions (方案表)
id, task_id, f1, f2, f3, total_cost, implementation_days,
expected_benefit, solution_data, technical_details,
rank, topsis_score
decision_records (决策记录表)
id, task_id, ahp_matrix, weights, consistency_ratio,

```
|  | best_solution_id, decision_scores, created_at
||
|
└──────────────────────────────────────────────────────────────────────┘|
└──────────────────────────────────────────────────────────────────────┘
──┘
```

五、实施步骤

Phase 1: 算法模块迁移 (1天)

1. 从 test6 复制核心算法代码到 `backend/app/algorithms/`
2. 调整导入路径和依赖
3. 测试算法模块独立运行

Phase 2: 数据库和模型 (0.5天)

1. 在 `models.py` 中添加天筹相关模型
2. 创建并执行数据库迁移
3. 验证表结构

Phase 3: API实现 (1天)

1. 实现 `tianchou.py` 路由
2. 实现后台任务函数
3. 测试 API 端点

Phase 4: 前端集成 (2-3天)

1. 创建天筹页面组件
2. 实现任务配置表单
3. 实现结果展示和可视化
4. 实现 AHP-TOPSIS 交互

六、关键差异说明

功能	原方案 (Celery)	简化方案 (BackgroundTasks)
任务执行	Celery Worker	FastAPI BackgroundTasks
任务队列	Redis	内存队列

功能	原方案 (Celery)	简化方案 (BackgroundTasks)
进度通知	WebSocket/轮询	轮询
并发处理	多Worker	单进程后台线程
适用场景	高并发、大规模	小规模、低并发

优点:

- 部署简单，无需额外服务
- 代码结构清晰
- 适合用户量少的场景

缺点:

- 无法水平扩展
- 服务重启会丢失运行中的任务
- 不适合长时间运行的任务

文档版本: 2.0 (简化版)
创建日期: 2026-02-12
项目: 天工·弈控 - 天筹优化决策系统