行业分析报告智能体操作指南

一、系统概述

"行业分析报告智能体"是一个自动化工具，能够自主采集、处理数据并生成高质量的行业分析报告。该智能体能够从多种数据源获取信息，进行智能化处理，并以专业、美观的方式呈现分析结果，支持多种格式的输出。

二、核心功能模块

使用工具

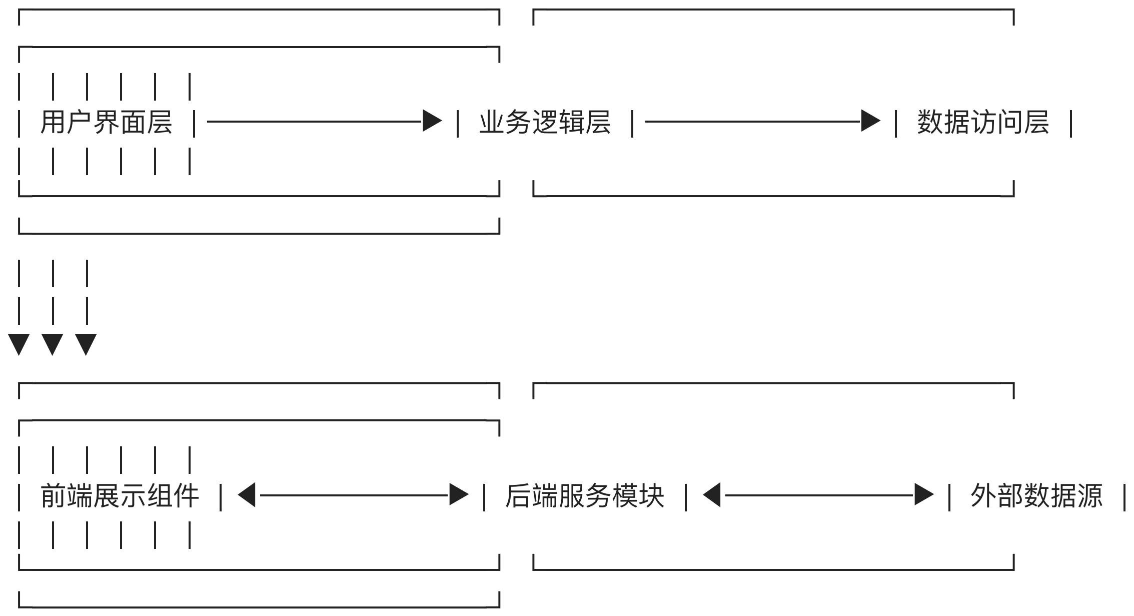
I

思考

让我思考一下完整的系统架构设计。首先需要分析用户的需求： 1. 需要一个能自动采集多个数据源的数据采集模块 2. 需要数据清洗和结构化分析模块 3. 需要智能图表生成模块 4. 需要报告排版和生成模块 5. 需要支持HTML和PDF导出 我会从这些方面进行详细设计： 1. 系统架构：梳理各模块间的关系和数据流 2. 技术栈选择：针对每个模块选择最合适的技术 3. 工作流程设计：从用户输入到最终报告生成的完整流程 4. 各模块详细配置：包括mcpServers配置、功能描述等 5. 文件结构：合理的项目组织结构

查看

1. 数据采集模块 (DataFetcher)
   该模块负责从多个数据源采集原始数据，支持多种采集方式：
   • 网页爬虫：自动抓取并解析网页内容
   • API 集成：通过公开/授权的 API 获取结构化数据
   • 数据库连接：直接从数据库获取历史数据
   • 文件导入：支持 CSV、Excel、JSON 等格式数据导入
   主要功能：
   • 多源数据采集调度
   • 自动处理反爬虫策略
   • 请求频率控制与代理轮换
   • 数据源健康监控
   • 增量采集与全量更新
2. 数据处理模块 (DataProcessor)
   负责对原始数据进行清洗、转换、标准化和结构化：
   • 数据清洗：去除噪声、处理缺失值、异常检测
   • 数据转换：单位统一、格式标准化
   • 数据融合：跨源数据整合与关联
   • 数据标记：自动添加时间、来源等元数据
   • 数据验证：一致性与有效性检查
3. 数据分析模块 (DataAnalyzer)
   对处理后的数据进行深入分析，提取有价值的洞见：
   • 统计分析：描述性统计、趋势分析
   • 预测模型：时间序列预测、回归分析
   • 关联分析：变量相关性、因果关系推断
   • 聚类分析：市场细分、客群画像
   • 文本分析：舆情分析、主题提取
4. 图表生成模块 (ChartRenderer)
   根据数据特征智能选择并生成适合的可视化图表：
   • 智能图表推荐：根据数据类型自动推荐合适的图表类型
   • 多样化图表库：支持柱状图、折线图、饼图、散点图等常用图表
   • 高级可视化：热力图、桑基图、地理信息可视化等
   • 交互式图表：支持缩放、筛选、钻取等交互功能
   • 自定义主题：可自定义颜色、字体、样式
5. 报告生成模块 (ReportGenerator)
   整合文字、图表、数据，生成完整的分析报告：

• 模板系统：提供多种专业报告模板
• 智能排版：自动优化图文布局
• 动态内容：根据数据自动生成分析文字
• 多格式导出：HTML、PDF、PPT 支持
• 响应式设计：适配不同设备显示
三、技术架构与系统设计
系统架构图


技术栈选择
前端技术：
• 框架：React.js（组件化开发、状态管理）
• UI 库：Ant Design（企业级 UI 组件）
• 样式：Tailwind CSS（响应式设计、快速开发）
• 图表库：ECharts、D3.js（丰富的可视化选项）
• 文档渲染：React-PDF、html2canvas（报告导出）
后端技术：
• 运行时：Node.js（JavaScript 运行环境）
• Web 框架：Express.js/Koa.js（轻量级、灵活）
• 爬虫：Puppeteer/Cheerio（网页数据爬取）
• 数据处理：Pandas.js/Lodash（数据转换与分析）
• 任务队列：Bull（分布式任务处理）
数据存储：
• 文档数据库：MongoDB（灵活的数据模型）
• 缓存：Redis（高性能缓存和任务队列）
• 文件存储：MinIO/S3（报告和图表存储）
四、模块配置详细说明
6. mcpServer 配置模板
Copy
```
{
"mcpServers": {
"dataFetcher": {
"name": "DataFetcher",
"description": "多源数据采集服务",
"command": "node",
"args": ["src/services/dataFetcher"],
"workDir": "/users/industryreportagent/services",
"env": {
"NODE_ENV": "production",
"LOG_LEVEL": "info",
"PROXY_SERVER": "http://proxy.example.com:8080"
},
"dependencies": [
"axios@1.3.4",
"cheerio@1.0.0-rc.12",
"puppeteer@19.7.2",
"node-schedule@2.1.1",
"got@12.5.3",
"crawlee@3.3.1"
],
"resources": {
"cpuLimit": "1",
"memoryLimit": "2Gi"
}
},
"dataProcessor": {
"name": "DataProcessor",
"description": "数据清洗与转换服务",
"command": "node",
"args": ["src/services/dataProcessor"],
"workDir": "/users/industryreportagent/services",
"dependencies": [
"lodash@4.17.21",
"papaparse@5.4.1",
"xlsx@0.18.5",
"fast-csv@4.3.6",
"json-2-csv@4.0.0"
],
"env": {
"NODE_ENV": "production",
"DATA_DIR": "/users/industryreportagent/data"
},
"resources": {
"cpuLimit": "1",
"memoryLimit": "1Gi"
}
},
"dataAnalyzer": {
"name": "DataAnalyzer",
"description": "数据分析与洞察提取",
"command": "node",
"args": ["src/services/dataAnalyzer"],
"workDir": "/users/industryreportagent/services",
"dependencies": [
"ml.js@6.0.0",
"simple-statistics@7.8.2",
"regression@2.0.1",
"node-nlp@4.26.1",
"brain.js@2.0.0-beta.18"
],
"env": {
"NODE_ENV": "production",
"MODEL_DIR": "/users/industryreportagent/models"
},
"resources": {
"cpuLimit": "2",
"memoryLimit": "4Gi"
}
},
"chartRenderer": {
"name": "ChartRenderer",
"description": "智能图表生成服务",
"command": "node",
"args": ["src/services/chartRenderer"],
"workDir": "/users/industryreportagent/services",
"dependencies": [
"echarts@5.4.2",
"d3@7.8.4",
"chart.js@4.2.1",
"vega@5.24.0",
"plotly.js@2.20.0",
"canvas@2.11.2"
],
"env": {
"NODE_ENV": "production",
"CHART_CACHE": "/users/industryreportagent/charts"
},
"resources": {
"cpuLimit": "1",
"memoryLimit": "2Gi"
}
},
"reportGenerator": {
"name": "ReportGenerator",
"description": "报告生成与格式转换",
"command": "node",
"args": ["src/services/reportGenerator"],
"workDir": "/users/industryreportagent/services",
"dependencies": [
"handlebars@4.7.7",
"puppeteer@19.7.2",
"html-pdf-node@1.0.8",
"markdown-it@13.0.1",
"tailwindcss@3.2.7",
"jspdf@2.5.1"
],
"env": {
"NODE_ENV": "production",
"TEMPLATE_DIR": "/users/industryreportagent/templates",
"OUTPUT_DIR": "/users/industryreportagent/output"
},
"resources": {
"cpuLimit": "2",
"memoryLimit": "2Gi"
}
},
"apiGateway": {
"name": "ApiGateway",
"description": "API网关与服务协调",
"command": "node",
"args": ["src/api/server"],
"workDir": "/users/industryreportagent/api",
"dependencies": [
"express@4.18.2",
"cors@2.8.5",
"helmet@6.0.1",
"compression@1.7.4",
"morgan@1.10.0",
"jsonwebtoken@9.0.0"
],
"env": {
"NODE_ENV": "production",
"PORT": "3000",
"API_VERSION": "v1"
},
"resources": {
"cpuLimit": "1",
"memoryLimit": "1Gi"
}
}
}
}
```

7. 完整项目结构

```
industryreportagent/
├── api/ # API网关与接口
│   ├── controllers/ # 控制器
│   ├── middlewares/ # 中间件
│   ├── routes/ # 路由定义
│   └── server.js # API服务器入口
├── config/ # 配置文件
│   ├── data-sources.json # 数据源配置
│   ├── app-config.js # 应用配置
│   └── report-templates.js # 报告模板配置
├── data/ # 数据存储
│   ├── raw/ # 原始数据
│   ├── processed/ # 处理后数据
│   └── cache/ # 缓存数据
├── models/ # 分析模型
│   ├── prediction/ # 预测模型
│   └── classification/ # 分类模型
├── services/ # 核心服务
│   ├── dataFetcher/ # 数据采集服务
│   │   ├── adapters/ # 数据源适配器
│   │   ├── schedulers/ # 采集调度器
│   │   └── index.js # 服务入口
│   ├── dataProcessor/ # 数据处理服务
│   ├── dataAnalyzer/ # 数据分析服务
│   ├── chartRenderer/ # 图表渲染服务
│   └── reportGenerator/ # 报告生成服务
├── templates/ # 报告模板
│   ├── standard/ # 标准模板
│   ├── executive/ # 高管模板
│   └── components/ # 共享组件
├── charts/ # 图表输出
│   ├── svg/ # SVG格式图表
│   └── png/ # PNG格式图表
├── output/ # 报告输出
│   ├── html/ # HTML格式
│   └── pdf/ # PDF格式
├── utils/ # 工具函数
│   ├── formatters/ # 数据格式化
│   ├── validators/ # 数据验证
│   └── helpers/ # 通用助手函数
├── web/ # Web前端
│   ├── src/ # 源代码
│   │   ├── components/ # React组件
│   │   ├── pages/ # 页面组件
│   │   ├── hooks/ # 自定义Hooks
│   │   └── services/ # 前端服务
│   ├── public/ # 静态资源
│   └── package.json # 前端依赖
├── tests/ # 测试文件
│   ├── unit/ # 单元测试
│   └── integration/ # 集成测试
├── mcpServers.json # 智能体服务配置
├── package.json # 项目依赖
└── README.md # 项目说明
```

五、工作流程详解

8. 完整工作流程
   使用工具
   I

创建网页

查看

industry_report_agent_workflow
点击打开

以下是完整的工作流程详解：
步骤 1: 用户需求输入与配置
1 用户输入关键参数：
• 行业领域（如：新能源汽车、医疗健康、半导体）
• 分析维度（市场规模、竞争格局、趋势预测等）
• 报告时间范围（最近一年、三年或五年等）
• 报告侧重点（技术路线、商业模式、政策环境等）

2 智能体初始化配置：
• 系统自动根据行业选择合适的数据源
• 构建采集策略和分析框架
• 准备报告模板与排版方案
步骤 2: 多源数据采集与预处理
1 数据源调度：
• 对各数据源进行优先级排序
• 确定采集策略（全量/增量/差异化采集）
• 分配计算资源并发起采集任务
2 数据抓取执行：
• 网页爬虫采集：对东方财富网、艾瑞咨询等网站内容抓取
• API 数据获取：从 Statista、SimilarWeb 获取结构化数据
• 数据库直连：对接已有数据仓库获取历史数据
• 公开数据集：从 IMF、World Bank 下载公开数据集
3 原始数据缓存：
• 将采集的数据按来源分类存储
• 添加采集时间戳和元信息
• 进行数据完整性初步校验
步骤 3: 数据清洗与结构化处理
1 数据清洗：
• 去除无效数据和重复记录
• 标准化数据格式（日期、数值、单位一致性）
• 处理空值和异常值
• 去除广告和无关内容
2 数据转换与标准化：
• 时间序列对齐（按月/季/年标准化）
• 跨数据源指标统一（确保相同指标含义一致）
• 货币单位转换（统一使用同一货币单位）
• 数值缩放与归一化
3 数据标记与结构化：
• 自动添加数据分类标签
• 建立数据间的关联关系
• 生成结构化的中间数据集
步骤 4: 智能数据分析与洞察提取
1 统计分析：
• 描述性统计（均值、中位数、分布特征等）
• 时间序列分析（趋势、周期、季节性等）
• 相关性分析（变量间关联程度）
2 高级分析：
• 市场结构分析（市场集中度、竞争格局）
• 趋势预测（时间序列预测、回归分析）
• 文本挖掘（舆情分析、主题提取）
• 聚类分析（市场细分、客群特征）
3 洞察生成：
• 自动识别关键变化和异常
• 提取重要趋势和模式
• 发现潜在机会和风险点
• 生成数据支撑的观点和结论
步骤 5: 智能图表生成
1 图表类型选择：
• 根据数据特征自动推荐最佳图表类型
• 考虑数据维度、分布特征和展示目的

• 选择合适的颜色方案和样式
2 图表渲染与优化：
• 自动确定坐标轴范围和刻度
• 添加图例、标签和注释
• 优化视觉布局和色彩方案
• 生成高清图表文件
3 交互功能嵌入（HTML版本）：
• 添加数据筛选和排序功能
• 实现缩放和局部查看功能
• 支持数据点悬停显示详情
• 增加图表间联动效果
步骤 6: 报告内容生成与排版
1 内容组织：
• 根据行业特点和分析维度构建报告框架
• 自动生成各章节标题和导言
• 按逻辑顺序组织图表和文字说明
2 文本生成：
• 自动撰写数据解读和分析文字
• 生成图表说明和关键点注释
• 归纳总结与前景展望
3 排版设计：
• 应用选定的报告模板
• 优化图文排版（上下、左右或并排）
• 调整字体、颜色和间距
• 添加页眉、页脚和页码
步骤 7: 报告导出与交付
1 格式转换：
• 生成 HTML 版本（支持交互功能）
• 转换为 PDF 格式（便于分享和打印）
• 可选生成 PPT 演示版本
2 质量检查：
• 自动检查排版一致性
• 验证图表与文字的一致性
• 检查数据引用和来源标注
• 拼写和格式检查
3 最终交付：
• 生成带目录和索引的完整报告
• 添加参考文献和数据来源说明
• 提供报告下载和在线访问链接
六、关键模块功能详细说明

1. 数据采集模块 (DataFetcher)
   数据源适配器设计
   每个数据源都有专门的适配器处理其特殊性：
   Copy
   ```
   // 东方财富网适配器示例
   class EastMoneyAdapter {
   constructor(config) {
   this.baseUrl = "https://www.eastmoney.com";
   this.browser = null;
   this.config = config;
   }
   ```

```
async initialize() {
this.browser = await puppeteer.launch({
headless: true,
args: ['--no-sandbox', '--disable-setuid-sandbox']
});
}

async fetchIndustryData(industryCode, timeRange) {
const page = await this.browser.newPage();
await page.goto(${this.baseUrl}/industry/${industryCode});
```

```
// 执行页面交互与数据提取
const data = await page.evaluate((timeRange) => {
  // 提取DOM中的数据并返回结构化结果
  return {
    // 具体数据提取逻辑
  };
}, timeRange);

await page.close();
return data;
```

```
}

async fetchCompanyFinancials(stockCode) {
// 获取公司财务数据逻辑
}

async close() {
if (this.browser) await this.browser.close();
}
}

// 导出适配器
module.exports = EastMoneyAdapter;
```

智能调度系统

Copy

```
// 调度器示例
class DataFetchScheduler {
constructor(config) {
this.adapters = {};
this.queue = new PriorityQueue();
this.config = config;
}

registerAdapter(sourceName, adapterClass, priority) {
this.adapters[sourceName] = {
adapterClass,
priority,
instance: null
};
}
```

```
async initializeAdapters() {
for (const [name, adapter] of Object.entries(this.adapters)) {
const instance = new adapter.adapterClass(this.config);
await instance.initialize();
this.adapters[name].instance = instance;
}
}

scheduleTask(sourceName, method, params, priority) {
this.queue.push({
sourceName,
method,
params,
priority: priority || this.adapters[sourceName].priority
});
}

async processQueue() {
while (!this.queue.isEmpty()) {
const task = this.queue.pop();
const adapter = this.adapters[task.sourceName].instance;
```

```
 try {
   const result = await adapter[task.method](...task.params);
   await this.saveResult(task.sourceName, task.method, result);
 } catch (error) {
   console.error(`Error processing task: ${error.message}`);
   // 重试或降级处理逻辑
 }
}
```

```
}

async saveResult(sourceName, method, data) {
// 保存数据到适当位置
const fileName = ${sourceName}_${method}_${Date.now()}.json;
const filePath = path.join(this.config.dataDir, 'raw', fileName);
```

```
await fs.promises.mkdir(path.dirname(filePath), { recursive: true });
await fs.promises.writeFile(filePath, JSON.stringify(data), 'utf8');

return filePath;
```

```
}
}
```

2. 数据处理模块 (DataProcessor)
   数据清洗器
   Copy
   ```
   class DataCleaner {
   constructor(options = {}) {
   this.options = {
   removeNulls: true,
   removeDuplicates: true,
   ```

```
        standardizeDates: true,
        ...options
        };
        }
```

// 处理空值
handleNulls(data, strategy = 'remove') {
if (!this.options.removeNulls) return data;

```
switch (strategy) {
  case 'remove':
    return data.filter(row => Object.values(row).every(v => v !== null && v !== undefined));
  case 'fill':
    return data.map(row => {
      const newRow = {...row};
      Object.keys(newRow).forEach(key => {
        if (newRow[key] === null || newRow[key] === undefined) {
          newRow[key] = this._getDefaultValue(data, key);
        }
      });
      return newRow;
    });
  default:
    return data;
}
```

}

// 获取列的默认值（如平均值、中位数）
_getDefaultValue(data, column) {
const values = data.map(row => row[column]).filter(v => v !== null && v !== undefined);

```
// 根据数据类型选择默认值
if (values.length === 0) return null;

const firstValue = values[0];
if (typeof firstValue === 'number') {
  // 数值类型使用平均值
  return values.reduce((sum, v) => sum + v, 0) / values.length;
} else if (typeof firstValue === 'string') {
  // 字符串类型使用最常见值
  const counts = values.reduce((acc, v) => {
    acc[v] = (acc[v] || 0) + 1;
    return acc;
  }, {});

  return Object.entries(counts).sort((a, b) => b[1] - a[1])[0][0];
}

return null;
```

}

```
// 移除重复数据
removeDuplicates(data, keyFields = []) {
if (!this.options.removeDuplicates) return data;
```

```
const seen = new Set();
return data.filter(row => {
  const key = keyFields.length > 0
    ? keyFields.map(field => row[field]).join('|')
    : JSON.stringify(row);

  if (seen.has(key)) return false;
  seen.add(key);
  return true;
});
```

```
}

// 标准化日期格式
standardizeDates(data, dateFields = []) {
if (!this.options.standardizeDates) return data;
```

```
return data.map(row => {
  const newRow = {...row};
  dateFields.forEach(field => {
    if (newRow[field]) {
      newRow[field] = this._standardizeDateFormat(newRow[field]);
    }
  });
  return newRow;
});
```

```
}

// 将各种日期格式转换为标准格式
_standardizeDateFormat(dateValue) {
try {
const date = new Date(dateValue);
if (isNaN(date.getTime())) return dateValue;
return date.toISOString().split('T')[0]; // YYYY-MM-DD
} catch (e) {
return dateValue;
}
}

// 异常值检测与处理
handleOutliers(data, numericFields = [], method = 'iqr') {
const result = [...data];
```

```
numericFields.forEach(field => {
  const values = data.map(row => row[field]).filter(v => typeof v === 'number');

  if (method === 'iqr') {
    // IQR方法检测异常值
    const sorted = [...values].sort((a, b) => a - b);
    const q1 = sorted[Math.floor(sorted.length * 0.25)];
    const q3 = sorted[Math.floor(sorted.length * 0.75)];
    const iqr = q3 - q1;
    const lowerBound = q1 - 1.5 * iqr;
    const upperBound = q3 + 1.5 * iqr;

    // 替换异常值
    result.forEach(row => {
      if (row[field] < lowerBound || row[field] > upperBound) {
        row[field] = null;
      }
    });
  }
});

return result;
```

}

// 执行所有清洗步骤
clean(data, options = {}) {
let result = [...data];

```
// 按顺序执行清洗操作
result = this.handleNulls(result, options.nullStrategy);
result = this.removeDuplicates(result, options.keyFields);
result = this.standardizeDates(result, options.dateFields);
result = this.handleOutliers(result, options.numericFields);

return result;
```

}
}
数据转换器
Copy
class DataTransformer {
constructor() {
this.transformers = {};
}

// 注册转换器
registerTransformer(name, transformFn) {
this.transformers[name] = transformFn;
}

```
// 应用转换逻辑
transform(data, transformations = []) {
let result = [...data];
```

```
for (const transformation of transformations) {
  const { name, params, fields } = transformation;

  if (this.transformers[name]) {
    result = this._applyTransformation(result, this.transformers[name], fields, params);
  }
}

return result;
```

```
}

// 将转换应用到指定字段
_applyTransformation(data, transformFn, fields, params) {
return data.map(row => {
const newRow = {...row};
```

```
  fields.forEach(field => {
    if (field.includes('->')) {
      // 源字段到目标字段的映射转换
      const [sourceField, targetField] = field.split('->').map(f => f.trim());
      newRow[targetField] = transformFn(row[sourceField], params);
    } else {
      // 原地转换
      newRow[field] = transformFn(row[field], params);
    }
  });

  return newRow;
});
```

```
}

// 内置转换器初始化
initializeBuiltInTransformers() {
// 数值单位转换
this.registerTransformer('unitConversion', (value, params) => {
if (typeof value !== 'number') return value;
```

```
  const { fromUnit, toUnit, conversionFactor } = params;
  return value * conversionFactor;
});

// 货币转换
this.registerTransformer('currencyConversion', (value, params) => {
  if (typeof value !== 'number') return value;

  const { fromCurrency, toCurrency, exchangeRate } = params;
  return value * exchangeRate;
});

// 文本标准化
this.registerTransformer('textNormalization', (value, params) => {
  if (typeof value !== 'string') return value;

  const { lowercase = false, trim = true, removeSpecialChars = false } = params;

  let result = value;
  if (lowercase) result = result.toLowerCase();
  if (trim) result = result.trim();
  if (removeSpecialChars) result = result.replace(/[^\w\s]/g, '');

  return result;
});

// 数值归一化
this.registerTransformer('normalize', (value, params) => {
  if (typeof value !== 'number') return value;

  const { min, max } = params;
  return (value - min) / (max - min);
});
```

}
}

3. 图表生成模块 (ChartRenderer)
   智能图表推荐系统
   Copy
   class ChartRecommender {
   constructor() {
   this.rules = [];
   }

// 添加推荐规则
addRule(rule) {
this.rules.push(rule);
}

// 根据数据特征推荐图表类型
recommendChartType(data, columns) {
for (const rule of this.rules) {

```
const recommendation = rule(data, columns);
if (recommendation) return recommendation;
}
```

```
// 默认返回柱状图
return { type: 'bar' };
```

```
}

// 初始化内置规则
initializeRules() {
// 时间序列数据 -> 折线图
this.addRule((data, columns) => {
const timeColumn = columns.find(col =>
col.type === 'date' ||
(col.type === 'string' && this._looksLikeDate(data.map(row => row[col.name])))
);
```

```
  const numericColumns = columns.filter(col => col.type === 'number');

  if (timeColumn && numericColumns.length > 0) {
    return {
      type: 'line',
      xAxis: timeColumn.name,
      yAxis: numericColumns.map(col => col.name)
    };
  }

  return null;
});

// 分类占比数据 -> 饼图
this.addRule((data, columns) => {
  const categoryColumn = columns.find(col => col.type === 'string');
  const numericColumn = columns.find(col => col.type === 'number');

  if (categoryColumn && numericColumn && data.length <= 10) {
    const total = data.reduce((sum, row) => sum + row[numericColumn.name], 0);

    // 检查是否符合"部分与整体"的关系
    if (Math.abs(total - 100) < 5 || this._looksLikeProportions(data, numericColumn.name)) {
      return {
        type: 'pie',
        category: categoryColumn.name,
        value: numericColumn.name
      };
    }
  }

  return null;
});
```

```
// 多分类比较 -> 柱状图
this.addRule((data, columns) => {
  const categoryColumn = columns.find(col => col.type === 'string');
  const numericColumns = columns.filter(col => col.type === 'number');

  if (categoryColumn && numericColumns.length > 0) {
    return {
      type: 'bar',
      xAxis: categoryColumn.name,
      yAxis: numericColumns.map(col => col.name)
    };
  }

  return null;
});

// 相关性分析 -> 散点图
this.addRule((data, columns) => {
  const numericColumns = columns.filter(col => col.type === 'number');

  if (numericColumns.length >= 2) {
    // 检查两列之间是否存在相关性
    const col1 = numericColumns[0].name;
    const col2 = numericColumns[1].name;

    const correlation = this._calculateCorrelation(
      data.map(row => row[col1]),
      data.map(row => row[col2])
    );

    if (Math.abs(correlation) > 0.3) {
      return {
        type: 'scatter',
        xAxis: col1,
        yAxis: col2
      };
    }
  }

  return null;
});

// 地理数据 -> 地图
this.addRule((data, columns) => {
  const geoColumn = columns.find(col =>
    col.type === 'string' &&
    this._looksLikeGeoData(data.map(row => row[col.name]))
  );

  const numericColumn = columns.find(col => col.type === 'number');

  if (geoColumn && numericColumn) {
    return {
      type: 'map',
      geo: geoColumn.name,
      value: numericColumn.name
    };
  }

  return null;
});
```

}

// 辅助方法：检查是否可能是日期数据
_looksLikeDate(values) {
const datePatterns = [
/^\d{4}[-/]\d{1,2}[-/]\d{1,2}$/, // YYYY-MM-DD
/^\d{1,2}[-/]\d{1,2}[-/]\d{4}$/, // DD-MM-YYYY
/^\d{4}年\d{1,2}月$/, // YYYY年MM月
/^Q[1-4] \d{4}$/, // Q1 2023
/^\d{4}$/ // YYYY
];

```
const sampleSize = Math.min(values.length, 10);
const samples = values.slice(0, sampleSize);

return samples.some(value =>
  typeof value === 'string' &&
  datePatterns.some(pattern => pattern.test(value))
);
```

}

// 辅助方法：检查是否是比例数据
_looksLikeProportions(data, column) {
const sum = data.reduce((total, row) => total + row[column], 0);
const count = data.length;

```
// 检查平均值是否接近于 100/n 或 1/n
return (Math.abs(sum - 100) < 5) || (Math.abs(sum - 1) < 0.05);
```

}

// 辅助方法：计算两个数组的相关系数
_calculateCorrelation(x, y) {
const n = x.length;
let sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0, sumY2 = 0;

```
for (let i = 0; i < n; i++) {
  sumX += x[i];
  sumY += y[i];
  sumXY += x[i] * y[i];
  sumX2 += x[i] * x[i];
  sumY2 += y[i] * y[i];
}

const numerator = n * sumXY - sumX * sumY;
const denominator = Math.sqrt((n * sumX2 - sumX * sumX) * (n * sumY2 - sumY * sumY));

return denominator === 0 ? 0 : numerator / denominator;
```

}

// 辅助方法：检查是否可能是地理数据
_looksLikeGeoData(values) {
// 简单检查是否包含省份、城市或国家名称
const geoKeywords = [
'省', '市', '县', '区',
'Province', 'City', 'County',
'America', 'Europe', 'Asia'
];

```
const sampleSize = Math.min(values.length, 10);
const samples = values.slice(0, sampleSize);

return samples.some(value =>
  typeof value === 'string' &&
  geoKeywords.some(keyword => value.includes(keyword))
);
```

}
}
ECharts 图表渲染器
Copy
class EChartsRenderer {
constructor(options = {}) {
this.options = {
width: 800,
height: 600,
theme: 'light',
...options
};

```
// 初始化ECharts主题
this.themes = {
  light: {
    backgroundColor: '#ffffff',
    textStyle: { color: '#333333' },
    title: { textStyle: { color: '#333333' } },
    visualMap: { color: ['#2a5caa', '#b8c5e2'] },
    color: ['#5470c6', '#91cc75', '#fac858', '#ee6666', '#73c0de', '#3ba272', '#fc8452', '#9a60b4']
  },
  dark: {
    backgroundColor: '#333333',
    textStyle: { color: '#ffffff' },
    title: { textStyle: { color: '#ffffff' } },
    visualMap: { color: ['#8378EA', '#02FEFF'] },
    color: ['#5470c6', '#91cc75', '#fac858', '#ee6666', '#73c0de', '#3ba272', '#fc8452', '#9a60b4']
  },
  business: {
    backgroundColor: '#ffffff',
    textStyle: { color: '#333333' },
    title: { textStyle: { color: '#333333' } },
    visualMap: { color: ['#1a4882', '#c3e0e8'] },
    color: ['#4e79a7', '#f28e2b', '#e15759', '#76b7b2', '#59a14f', '#edc948', '#b07aa1', '#ff9da7']
  }
```

复制

```
};
this.renderers = {};
this._initializeRenderers();
}
// 初始化各类图表渲染器 _initializeRenderers() { // 柱状图渲染器 this.renderers.bar = (data, config) => { const { xAxis, yAxis, title = '', subtitle = '' } = config; const categories = [...new Set(data.map(item => item[xAxis]))];
// 处理多个y轴的情况
const series = Array.isArray(yAxis) ?
yAxis.map(y => ({
name: y,
type: 'bar',
data: categories.map(cat => {
const item = data.find(d => d[xAxis] === cat);
return item ? item[y] : null;
})
})) :
[{
name: yAxis,
type: 'bar',
data: categories.map(cat => {
const item = data.find(d => d[xAxis] === cat);
return item ? item[yAxis] : null;
})
}];
```

```
return {
title: {
text: title,
subtext: subtitle
},
tooltip: { trigger: 'axis' },
legend: {
data: Array.isArray(yAxis) ? yAxis : [yAxis]
},
xAxis: {
type: 'category',
data: categories
},
yAxis: { type: 'value' },
series
};
};

// 折线图渲染器
this.renderers.line = (data, config) => {
const { xAxis, yAxis, title = '', subtitle = '', smooth = true } = config;
const categories = [...new Set(data.map(item => item[xAxis]))].sort();

// 处理多个y轴的情况
const series = Array.isArray(yAxis) ?
yAxis.map(y => ({
name: y,
type: 'line',
smooth,
data: categories.map(cat => {
const item = data.find(d => d[xAxis] === cat);
return item ? item[y] : null;
})
})) :
[{
name: yAxis,
type: 'line',
smooth,
data: categories.map(cat => {
const item = data.find(d => d[xAxis] === cat);
return item ? item[yAxis] : null;
})
}];

return {
title: {
text: title,
subtext: subtitle
},
tooltip: { trigger: 'axis' },
legend: {
data: Array.isArray(yAxis) ? yAxis : [yAxis]
```

```
},
xAxis: {
type: 'category',
data: categories
},
yAxis: { type: 'value' },
series
};
};

// 饼图渲染器
this.renderers.pie = (data, config) => {
const { category, value, title = '', subtitle = '' } = config;

const seriesData = data.map(item => ({
name: item[category],
value: item[value]
}));

return {
title: {
text: title,
subtext: subtitle
},
tooltip: {
trigger: 'item',
formatter: '{a} <br/>{b} : {c} ({d}%)'
},
legend: {
type: 'scroll',
orient: 'vertical',
right: 10,
top: 20,
bottom: 20,
data: seriesData.map(item => item.name)
},
series: [
{
name: value,
type: 'pie',
radius: '55%',
center: ['40%', '50%'],
data: seriesData,
emphasis: {
itemStyle: {
shadowBlur: 10,
shadowOffsetX: 0,
shadowColor: 'rgba(0, 0, 0, 0.5)'
}
}
}
```

```
]
};
};

// 散点图渲染器
this.renderers.scatter = (data, config) => {
const { xAxis, yAxis, title = '', subtitle = '', symbolSize = 10 } = config;

const seriesData = data.map(item => [item[xAxis], item[yAxis]]);

return {
title: {
text: title,
subtext: subtitle
},
tooltip: {
trigger: 'item',
formatter: function (params) {
return ${xAxis}: ${params.value[0]}<br/>${yAxis}: ${params.value[1]};
}
},
xAxis: {
type: 'value',
name: xAxis,
scale: true
},
yAxis: {
type: 'value',
name: yAxis,
scale: true
},
series: [
{
type: 'scatter',
data: seriesData,
symbolSize
}
]
};
};

// 雷达图渲染器
this.renderers.radar = (data, config) => {
const { dimensions, categories, title = '', subtitle = '' } = config;

const indicator = dimensions.map(dim => ({
name: dim,
max: Math.max(...data.map(item => item[dim])) * 1.2
}));

const series = [{
type: 'radar',
data: categories.map(cat => ({
```

```
name: cat,
value: dimensions.map(dim => {
const item = data.find(d => d.category === cat);
return item ? item[dim] : 0;
})
}))
}];

return {
title: {
text: title,
subtext: subtitle
},
tooltip: {},
legend: {
data: categories
},
radar: {
indicator
},
series
};
};

// 热力图渲染器
this.renderers.heatmap = (data, config) => {
const { xAxis, yAxis, value, title = '', subtitle = '' } = config;

const xCategories = [...new Set(data.map(item => item[xAxis]))];
const yCategories = [...new Set(data.map(item => item[yAxis]))];

const seriesData = data.map(item => [
xCategories.indexOf(item[xAxis]),
yCategories.indexOf(item[yAxis]),
item[value]
]);

return {
title: {
text: title,
subtext: subtitle
},
tooltip: {
position: 'top',
formatter: function (params) {
return ${xCategories[params.value[0]]}, ${yCategories[params.value[1]]}: ${params.value[2]};
}
},
grid: {
height: '50%',
top: '10%'
},
xAxis: {
```

```
type: 'category',
data: xCategories,
splitArea: { show: true }
},
yAxis: {
type: 'category',
data: yCategories,
splitArea: { show: true }
},
visualMap: {
min: Math.min(...data.map(item => item[value])),
max: Math.max(...data.map(item => item[value])),
calculable: true,
orient: 'horizontal',
left: 'center',
bottom: '15%'
},
series: [
{
name: value,
type: 'heatmap',
data: seriesData,
label: {
show: true
},
emphasis: {
itemStyle: {
shadowBlur: 10,
shadowColor: 'rgba(0, 0, 0, 0.5)'
}
}
}
]
};
};
}
// 渲染图表 async renderChart(data, chartType, config) { if (!this.renderers[chartType]) { throw new
Error(Unsupported chart type: ${chartType}); }
// 获取图表配置
const chartConfig = this.renderers[chartType](data, config);

// 应用主题
const theme = this.themes[this.options.theme] || this.themes.light;
Object.assign(chartConfig, {
backgroundColor: theme.backgroundColor,
textStyle: theme.textStyle,
color: theme.color
});

// 使用Node环境下的canvas渲染
const echarts = require('echarts');
const { createCanvas } = require('canvas');
const canvas = createCanvas(this.options.width, this.options.height);

// 创建ECharts实例
const chart = echarts.init(canvas);
chart.setOption(chartConfig);

// 生成图片
const buffer = canvas.toBuffer('image/png');

// 保存图片
const fileName = chart_${Date.now()}.png;
const filePath = path.join(this.options.outputDir, fileName);

await fs.promises.mkdir(this.options.outputDir, { recursive: true });
await fs.promises.writeFile(filePath, buffer);

return {
path: filePath,
url: /charts/${fileName},
type: chartType,
config: chartConfig
};
} }
```

4. 报告生成模块 (ReportGenerator)

动态报告内容生成器

```
class ReportContentGenerator {
  constructor(options = {}) {
    this.options = options;
  }

  // 生成报告摘要
  generateSummary(data, insights) {
    const summary = [];

    // 市场规模摘要
    if (insights.marketSize) {
      const { currentSize, growthRate, forecastSize } = insights.marketSize;
      summary.push(`本报告分析的${insights.industry}行业市场规模现已达到${currentSize}，同比增长
${growthRate}%。预计未来5年将保持${forecastSize.cagr}%的复合增长率，到${forecastSize.year}年市场规
模将达到${forecastSize.size}。`);
    }

    // 竞争格局摘要
    if (insights.competition) {
      const { topPlayers, concentration } = insights.competition;
      const topPlayersList = topPlayers.slice(0, 3).map(p => p.name).join('、');
      summary.push(`行业竞争格局方面，${topPlayersList}等企业占据主导地位，CR${topPlayers.length}达
${concentration}%。`);
    }
    // 趋势摘要
    if (insights.trends && insights.trends.length > 0) {
      const topTrends = insights.trends.slice(0, 3).map(t => t.name).join('、');
      summary.push(`未来行业发展将受${topTrends}等关键趋势影响，企业需积极应对市场变化。`);
    }

    return summary.join(' ');
  }

  // 生成市场规模章节
  generateMarketSizeSection(data, insights) {
    const { marketSize } = insights;

    let content = `
      <section id="market-size">
        <h2>市场规模分析</h2>
        <p>
          根据本报告数据分析，${insights.industry}行业市场规模在过去5年保持了稳定增长。
          ${marketSize.currentYear}年，全球市场规模达到${marketSize.currentSize}，同比增长
${marketSize.growthRate}%。
        </p>

        <div class="chart-container">
          <img src="${marketSize.historicalChart.url}" alt="历年市场规模走势" class="chart" />
          <div class="chart-description">
            <h4>图表：${insights.industry}行业历年市场规模走势</h4>
            <p>${marketSize.historicalChart.description}</p>
          </div>
        </div>

        <p>
          从区域分布来看，${marketSize.regions[0].name}占据最大市场份额
(${marketSize.regions[0].share}%)，
          其次是${marketSize.regions[1].name}(${marketSize.regions[1].share}%)和
${marketSize.regions[2].name}(${marketSize.regions[2].share}%)。
        </p>

        <div class="chart-container chart-side-by-side">
          <img src="${marketSize.regionChart.url}" alt="各区域市场分布" class="chart" />
          <div class="chart-description">
            <h4>图表：区域市场分布</h4>
            <p>${marketSize.regionChart.description}</p>
          </div>
        </div>

        <p>
          未来预测方面，受${marketSize.growthDrivers.join('、')}等因素推动，预计市场将保持
${marketSize.forecastSize.cagr}%的年复合增长率，
          到${marketSize.forecastSize.year}年市场规模将达到${marketSize.forecastSize.size}。
        </p>

        <div class="chart-container">
```

```
        <img src="${marketSize.forecastChart.url}" alt="市场规模预测" class="chart" />
        <div class="chart-description">
          <h4>图表：${insights.industry}行业市场规模预测(${marketSize.currentYear}-
${marketSize.forecastSize.year})</h4>
          <p>${marketSize.forecastChart.description}</p>
        </div>
      </div>
    </section>
  `;

  return content;
 }

 // 生成竞争格局章节
 generateCompetitionSection(data, insights) {
  const { competition } = insights;

  let content = `
    <section id="competition-landscape">
      <h2>竞争格局分析</h2>
      <p>
        ${insights.industry}行业的竞争格局呈现${competition.pattern}特征，行业集中度
CR${competition.topPlayers.length}为${competition.concentration}%。
      </p>

      <div class="chart-container">
        <img src="${competition.marketShareChart.url}" alt="主要企业市场份额" class="chart" />
        <div class="chart-description">
          <h4>图表：主要企业市场份额分布</h4>
          <p>${competition.marketShareChart.description}</p>
        </div>
      </div>

      <h3>主要企业分析</h3>
  `;

  // 添加主要企业分析内容
  competition.topPlayers.forEach(player => {
    content += `
      <div class="company-analysis">
        <h4>${player.name}</h4>
        <div class="company-info">
          <div class="company-basic">
            <p>市场份额：${player.marketShare}%</p>
            <p>营收规模：${player.revenue}</p>
            <p>增长率：${player.growthRate}%</p>
          </div>
          <div class="company-description">
            <p>${player.description}</p>
          </div>
        </div>
      </div>
    `;
  });

  content += `
      <h3>竞争策略分析</h3>
      <p>
        基于对主要企业的分析，${insights.industry}行业的竞争策略主要围绕
${competition.strategies.join('、')}等方面展开。
        企业需要在${competition.keyFactors.join('、')}等关键成功因素上建立竞争优势。
      </p>

      <div class="chart-container chart-side-by-side">
        <img src="${competition.strategyChart.url}" alt="企业战略对比" class="chart" />
        <div class="chart-description">
          <h4>图表：主要企业战略对比</h4>
          <p>${competition.strategyChart.description}</p>
        </div>
      </div>
    </section>
  `;

  return content;
}

// 生成趋势分析章节
generateTrendsSection(data, insights) {
  const { trends } = insights;

  let content = `
    <section id="industry-trends">
      <h2>行业趋势分析</h2>
      <p>
        通过对${insights.industry}行业的历史数据分析和未来展望，我们识别出以下关键趋势:
      </p>

      <div class="trends-container">
  `;

  // 添加关键趋势内容
  trends.forEach((trend, index) => {
    content += `
      <div class="trend-card">
        <h3>${index + 1}. ${trend.name}</h3>
        <p>${trend.description}</p>
        ${trend.chart ? `
        <div class="trend-chart">
          <img src="${trend.chart.url}" alt="${trend.name}" class="chart" />
          <p class="chart-caption">${trend.chart.description}</p>
        </div>
        ` : ''}
        <div class="trend-impact">
          <h4>影响分析</h4>
          <p>${trend.impact}</p>
        </div>
      </div>
    `;
  });

  content += `
      </div>

      <h3>综合趋势分析</h3>
      <p>${insights.trendsSummary}</p>

      <div class="chart-container">
        <img src="${insights.trendsRadarChart.url}" alt="行业趋势雷达图" class="chart" />
        <div class="chart-description">
          <h4>图表：${insights.industry}行业趋势影响雷达图</h4>
          <p>${insights.trendsRadarChart.description}</p>
        </div>
      </div>
    </section>
  `;

  return content;
}

// 生成完整的报告内容
generateFullReport(data, insights) {
  const summary = this.generateSummary(data, insights);
  const marketSizeSection = this.generateMarketSizeSection(data, insights);
  const competitionSection = this.generateCompetitionSection(data, insights);
  const trendsSection = this.generateTrendsSection(data, insights);

  const currentDate = new Date().toISOString().split('T')[0];

  const reportContent = `
    <!DOCTYPE html>
    <html lang="zh-CN">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>${insights.industry}行业分析报告</title>
      <link href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css" rel="stylesheet">
      <style>
        /* 报告样式定义 */
        body {
          font-family: 'Microsoft YaHei', Arial, sans-serif;
          line-height: 1.6;
          color: #333;
          max-width: 1200px;
          margin: 0 auto;
          padding: 20px;
        }
```

```
.report-header {
  text-align: center;
  margin-bottom: 40px;
  padding-bottom: 20px;
  border-bottom: 1px solid #eaeaea;
}

.report-title {
  font-size: 28px;
  font-weight: bold;
  margin-bottom: 10px;
}

.report-subtitle {
  font-size: 18px;
  color: #666;
  margin-bottom: 20px;
}

.report-meta {
  font-size: 14px;
  color: #888;
}

.executive-summary {
  background-color: #f9f9f9;
  padding: 20px;
  border-radius: 5px;
  margin-bottom: 30px;
}

section {
  margin-bottom: 40px;
}

h2 {
  font-size: 24px;
  font-weight: bold;
  padding-bottom: 10px;
  border-bottom: 2px solid #3498db;
  margin-bottom: 20px;
}

h3 {
  font-size: 20px;
  font-weight: bold;
  margin: 25px 0 15px;
  color: #2980b9;
}

h4 {
  font-size: 18px;
  font-weight: bold;
  margin: 20px 0 10px;
}

.chart-container {
  margin: 30px 0;
  border: 1px solid #eee;
  padding: 15px;
  border-radius: 5px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.05);
}

.chart {
  width: 100%;
  max-height: 400px;
  object-fit: contain;
}

.chart-side-by-side {
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  align-items: center;
}

.chart-side-by-side .chart {
  width: 60%;
}

.chart-side-by-side .chart-description {
  width: 35%;
}

.chart-description {
  margin-top: 15px;
}

.company-analysis {
  margin-bottom: 25px;
  padding: 15px;
  background-color: #f9f9f9;
  border-radius: 5px;
}

.company-info {
  display: flex;
  flex-wrap: wrap;
  margin-top: 10px;
}

.company-basic {
  width: 25%;
  min-width: 200px;
}

.company-description {
  width: 70%;
}

.trends-container {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
  gap: 20px;
  margin: 30px 0;
}

.trend-card {
  border: 1px solid #eee;
  border-radius: 5px;
  padding: 15px;
  box-shadow: 0 2px 5px rgba(0,0,0,0.05);
}

.trend-chart {
  margin: 15px 0;
}

.chart-caption {
  font-size: 14px;
  color: #666;
  margin-top: 8px;
}

.trend-impact {
  background-color: #f5f5f5;
  padding: 10px;
  border-radius: 5px;
  margin-top: 15px;
}

.report-footer {
  margin-top: 40px;
  padding-top: 20px;
  border-top: 1px solid #eaeaea;
  font-size: 14px;
  color: #888;
  text-align: center;
}

@media print {
  body {
    padding: 0;
  }
```

```
          .chart-container {
            break-inside: avoid;
          }

          section {
            break-after: always;
          }
        }
      </style>
    </head>
    <body>
      <div class="report-header">
        <div class="report-title">${insights.industry}行业分析报告</div>
        <div class="report-subtitle">市场现状、竞争格局与发展趋势</div>
        <div class="report-meta">生成日期：${currentDate}</div>
      </div>

      <div class="executive-summary">
        <h2>报告摘要</h2>
        <p>${summary}</p>
      </div>

      <div class="report-content">
        ${marketSizeSection}
        ${competitionSection}
        ${trendsSection}
      </div>

      <div class="report-footer">
        <p>本报告由行业分析报告智能体自动生成，数据来源包括东方财富网、艾瑞咨询、Statista等。</p>
        <p>© ${new Date().getFullYear()} 行业分析报告智能体 - 保留所有权利</p>
      </div>
    </body>
    </html>
  `;

  return reportContent;
 }
}
```
HTML/PDF 导出器

Copy
```
class ReportExporter {
 constructor(options = {}) {
  this.options = {
    outputDir: './output',
    tempDir: './temp',
    ...options
  };

  // 确保目录存在
  this._ensureDirectoriesExist();
```

```
}

// 确保必要的目录存在
async _ensureDirectoriesExist() {
  const fs = require('fs').promises;
  await fs.mkdir(this.options.outputDir, { recursive: true });
  await fs.mkdir(this.options.tempDir, { recursive: true });
}

// 保存HTML文件
async saveHtml(content, filename) {
  const fs = require('fs').promises;
  const path = require('path');

  const filePath = path.join(this.options.outputDir, `${filename}.html`);
  await fs.writeFile(filePath, content, 'utf8');

  return {
    path: filePath,
    url: `/reports/${filename}.html`
  };
}

// 生成PDF
async generatePdf(htmlContent, filename) {
  const fs = require('fs').promises;
  const path = require('path');
  const puppeteer = require('puppeteer');

  // 首先保存为临时HTML
  const tempHtmlPath = path.join(this.options.tempDir, `${filename}_temp.html`);
  await fs.writeFile(tempHtmlPath, htmlContent, 'utf8');

  // 使用Puppeteer生成PDF
  const browser = await puppeteer.launch({
    headless: true,
    args: ['--no-sandbox', '--disable-setuid-sandbox']
  });

  const page = await browser.newPage();

  // 设置PDF选项
  const pdfOptions = {
    path: path.join(this.options.outputDir, `${filename}.pdf`),
    format: 'A4',
    printBackground: true,
    margin: {
      top: '2cm',
      right: '2cm',
      bottom: '2cm',
      left: '2cm'
    },
      displayHeaderFooter: true,
      headerTemplate: '<div style="font-size:8px; margin-left:2cm;">行业分析报告智能体</div>',
      footerTemplate: '<div style="font-size:8px; margin-left:2cm; width:100%; text-align:center;"><span
class="pageNumber"></span> / <span class="totalPages"></span></div>'
    };

    // 加载HTML内容
    await page.goto(`file://${tempHtmlPath}`, { waitUntil: 'networkidle0' });

    // 生成PDF
    await page.pdf(pdfOptions);

    // 关闭浏览器
    await browser.close();

    // 移除临时HTML文件
    await fs.unlink(tempHtmlPath);

    return {
      path: pdfOptions.path,
      url: `/reports/${filename}.pdf`
    };
  }

  // 导出报告（同时生成HTML和PDF）
  async exportReport(content, filename) {
    const htmlResult = await this.saveHtml(content, filename);
    const pdfResult = await this.generatePdf(content, filename);

    return {
      html: htmlResult,
      pdf: pdfResult
    };
  }
}
```
七、完整工具链与依赖清单

前端技术栈：

- React.js - 组件化界面开发
- Tailwind CSS - 响应式样式框架
- Ant Design - 企业级UI组件库
- ECharts - 数据可视化图表库
- D3.js - 高级数据可视化
- React-PDF - PDF预览和处理

后端技术栈：

- Node.js - JavaScript运行时环境
- Express.js - Web应用框架
- Puppeteer - 浏览器自动化
- Cheerio - HTML解析
- Axios - HTTP客户端
- Lodash - JavaScript工具库
- MongoDB - NoSQL数据库
- Redis - 缓存与消息队列

核心依赖包版本：
Copy

```
{
  "dependencies": {
    "axios": "^1.3.4",
    "cheerio": "^1.0.0-rc.12",
    "puppeteer": "^19.7.2",
    "crawlee": "^3.3.1",
    "lodash": "^4.17.21",
    "papaparse": "^5.4.1",
    "xlsx": "^0.18.5",
    "echarts": "^5.4.2",
    "d3": "^7.8.4",
    "chart.js": "^4.2.1",
    "ml.js": "^6.0.0",
    "simple-statistics": "^7.8.2",
    "compression": "^1.7.4",
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "helmet": "^6.0.1",
    "morgan": "^1.10.0",
    "mongodb": "^5.1.0",
    "redis": "^4.6.5",
    "handlebars": "^4.7.7",
    "html-pdf-node": "^1.0.8",
    "jspdf": "^2.5.1",
    "canvas": "^2.11
```