# 详细流程

# AI代码质量分析平台 - 详细开发规划

## 📋 项目概述

基于AI的代码质量分析平台，专注于静态代码分析、性能优化建议和智能重构推荐。

**总体时间规划：10-12周**
**建议投入时间：每天2-3小时，周末可投入6-8小时**

---

## 🚀 第一阶段：项目基础搭建 (第1-2周)

### Week 1: 环境搭建与架构设计

**预估时间：12-15小时**

### Day 1-2: 项目初始化

```bash
# 项目搭建
npm create vite@latest ai-code-analyzer -- --template react-ts
cd ai-code-analyzer
npm install @reduxjs/toolkit react-redux antd
npm install @monaco-editor/react
npm install @types/node typescript @babel/core @babel/parser -D
npm install recharts lucide-react
```

**任务清单：**

- [x] ~~Vite + React + TS 脚手架搭建~~
- [x] ~~ESLint + Prettier 配置~~
- [x] ~~目录结构设计~~
- [x] ~~Git 工作流设置~~
- [x] ~~基础依赖安装~~

### Day 3-5: 核心架构设计

```
// 项目结构
src/
├── components/          # 通用组件
│   ├── CodeEditor/     # 代码编辑器组件
│   ├── AnalysisReport/ # 分析报告组件
│   └── Charts/         # 图表组件
├── features/           # 功能模块
│   ├── editor/         # 编辑器相关
│   ├── analysis/       # 代码分析
│   └── ai-suggestions/ # AI建议
├── store/             # Redux store
├── services/          # API服务
├── utils/            # 工具函数
├── workers/          # Web Workers
└── types/            # TypeScript类型
```

**任务清单:**

- [x] ~~设计组件架构~~
- [x] ~~定义数据流模型~~
- [ ] 创建基础路由
- [ ] 配置Redux Toolkit
- [ ] 定义核心TypeScript类型

### Day 6-7: UI框架集成

```
// theme/index.ts
export const theme = {
  token: {
    colorPrimary: '#1890ff',
    borderRadius: 8,
    wireframe: false,
  },
  components: {
    Layout: {
      bodyBg: '#f5f5f5',
      headerBg: '#001529',
    },
  },
};
```

**任务清单：**

- [ ] Ant Design主题配置
- [ ] 基础布局组件开发
- [ ] 响应式设计适配
- [ ] 暗黑/明亮主题切换

## Week 2: 核心编辑器实现

**预估时间：15-18小时**

### Day 1-3: Monaco Editor集成

```
// components/CodeEditor/index.tsx
import Editor from '@monaco-editor/react';
import { useSelector, useDispatch } from 'react-redux';

interface CodeEditorProps {
  value: string;
  language: string;
  onChange: (value: string) => void;
  onAnalyze?: () => void;
}

const CodeEditor: React.FC<CodeEditorProps> = ({
  value,
  language,
  onChange,
  onAnalyze
}) => {
  const theme = useSelector(state => state.ui.theme);

  const handleEditorChange = (value: string | undefined) => {
    if (value !== undefined) {
      onChange(value);
    }
  };

  return (
    <div className="code-editor-container">
      <Editor
        height="600px"
        language={language}
        value={value}
        theme={theme === 'dark' ? 'vs-dark' : 'light'}
        onChange={handleEditorChange}
        options={{
          minimap: { enabled: true },
          fontSize: 14,
          lineNumbers: 'on',
          automaticLayout: true,
        }}
      />
    </div>
  );
};
```

**任务清单：**

- [ ] Monaco Editor基础集成
- [ ] 多语言支持 (TypeScript, JavaScript, Python, Java)
- [ ] 主题切换功能
- [ ] 代码格式化功能
- [ ] 文件上传与导入

### Day 4-7: 编辑器增强功能

```
// hooks/useEditorEnhancements.ts
export const useEditorEnhancements = (editor: any) => {
  const addErrorMarkers = (errors: AnalysisError[]) => {
    const markers = errors.map(error => ({
      startLineNumber: error.line,
      startColumn: error.column,
      endLineNumber: error.line,
      endColumn: error.column + error.length,
      message: error.message,
      severity: error.severity,
    }));

    monaco.editor.setModelMarkers(editor.getModel(), 'analysis', markers);
  };

  return { addErrorMarkers };
};
```

**任务清单：**

- [ ] 语法高亮优化

- [ ] 实时错误提示显示
- [ ] 代码折叠功能
- [ ] 搜索替换功能
- [ ] 快捷键支持
- [ ] 代码片段模板

---

## 🔍 第二阶段：代码分析引擎 (第3-5周)

### Week 3: AST解析与基础分析

**预估时间：18-22小时**

### Day 1-4: AST解析器开发

```
// services/analyzers/TypeScriptAnalyzer.ts
import * as ts from 'typescript';
import { AnalysisResult, ComplexityReport } from '../../types/analysis';

export class TypeScriptAnalyzer {
  parseCode(code: string): ts.SourceFile {
    return ts.createSourceFile(
      'temp.ts',
      code,
      ts.ScriptTarget.Latest,
      true
    );
  }

  analyzeComplexity(sourceFile: ts.SourceFile): ComplexityReport {
    let cyclomaticComplexity = 1;
    let cognitiveComplexity = 0;
    let functions: FunctionInfo[] = [];

    const visit = (node: ts.Node) => {
      // 圈复杂度计算
      if (this.isComplexityNode(node)) {
        cyclomaticComplexity++;
      }

      // 认知复杂度计算
      cognitiveComplexity += this.calculateCognitiveComplexity(node);
```

```
        // 函数信息提取
        if (ts.isFunctionDeclaration(node) || ts.isMethodDeclaration(node)) {
          functions.push(this.extractFunctionInfo(node));
        }

        ts.forEachChild(node, visit);
      };

      visit(sourceFile);

      return {
        cyclomaticComplexity,
        cognitiveComplexity,
        functions,
        maintainabilityIndex: this.calculateMaintainabilityIndex(sourceFile)
      };
    }

    private isComplexityNode(node: ts.Node): boolean {
      return ts.isIfStatement(node) ||
             ts.isWhileStatement(node) ||
             ts.isForStatement(node) ||
             ts.isSwitchStatement(node) ||
             ts.isConditionalExpression(node);
    }
  }
```

**任务清单:**

- [ ] TypeScript AST解析实现
- [ ] JavaScript解析支持
- [ ] 基础语法错误检测
- [ ] AST遍历工具类
- [ ] 多文件依赖分析

## Day 5-7: 代码质量分析

```
// services/analyzers/QualityAnalyzer.ts
export class QualityAnalyzer {
  analyzeCodeSmells(code: string): CodeSmell[] {
    const smells: CodeSmell[] = [];

    // 长函数检测
    const longFunctions = this.detectLongFunctions(code);
    smells.push(...longFunctions);

    // 重复代码检测
    const duplicates = this.detectDuplicateCode(code);
    smells.push(...duplicates);

    // 死代码检测
    const deadCode = this.detectDeadCode(code);
    smells.push(...deadCode);

    return smells;
  }

  private detectLongFunctions(code: string): CodeSmell[] {
    const functions = this.extractFunctions(code);
    return functions
      .filter(fn => fn.lineCount > 50)
      .map(fn => ({
        type: 'long-function',
        severity: 'warning',
        message: `Function ${fn.name} is too long (${fn.lineCount} lines)`,
        line: fn.startLine,
        suggestion: 'Consider breaking this function into smaller functions'
      }));
  }
}
```

**任务清单：**

- [ ] 圈复杂度计算
- [ ] 认知复杂度分析
- [ ] 代码重复检测
- [ ] 函数长度分析
- [ ] 变量命名规范检查
- [ ] 代码异味识别

## Week 4: 性能与安全分析

**预估时间：16-20小时**

### Day 1-4: 性能分析器

```
// services/analyzers/PerformanceAnalyzer.ts
export class PerformanceAnalyzer {
  analyzePerformanceIssues(code: string): PerformanceIssue[] {
    const issues: PerformanceIssue[] = [];

    // 检测潜在内存泄漏
    issues.push(...this.detectMemoryLeaks(code));

    // 检测低效循环
    issues.push(...this.detectInefficiëntLoops(code));

    // 检测不必要的重新渲染
    issues.push(...this.detectUnnecessaryReRenders(code));

    return issues;
  }

  private detectMemoryLeaks(code: string): PerformanceIssue[] {
    const issues: PerformanceIssue[] = [];

    // 检测未清理的事件监听器
    const eventListenerPattern = /addEventListener\s*\(/g;
    const removeListenerPattern = /removeEventListener\s*\(/g;

    const addCount = (code.match(eventListenerPattern) || []).length;
    const removeCount = (code.match(removeListenerPattern) || []).length;

    if (addCount > removeCount) {
      issues.push({
        type: 'memory-leak',
        severity: 'error',
        message: 'Potential memory leak: Event listeners not properly removed',
        suggestion: 'Ensure all event listeners are removed in cleanup
functions'
      });
    }

    return issues;
  }
}
```

## Day 5-7: 安全分析器

```
// services/analyzers/SecurityAnalyzer.ts
export class SecurityAnalyzer {
  analyzeSecurity(code: string): SecurityIssue[] {
    const issues: SecurityIssue[] = [];

    issues.push(...this.checkSQLInjection(code));
    issues.push(...this.checkXSSVulnerabilities(code));
    issues.push(...this.checkInsecureRandomness(code));

    return issues;
  }

  private checkSQLInjection(code: string): SecurityIssue[] {
    const sqlInjectionPatterns = [
      /query\s*\(\s*[`"'].*\$\{.*\}.*[`"']\s*\)/g,
      /execute\s*\(\s*[`"'].*\+.*[`"']\s*\)/g,
    ];

    const issues: SecurityIssue[] = [];

    sqlInjectionPatterns.forEach(pattern => {
      const matches = code.match(pattern);
      if (matches) {
        issues.push({
          type: 'sql-injection',
          severity: 'critical',
          message: 'Potential SQL injection vulnerability detected',
          suggestion: 'Use parameterized queries or prepared statements'
        });
      }
    });

    return issues;
  }
}
```

**任务清单：**

- [ ] 性能瓶颈检测
- [ ] 内存泄漏识别
- [ ] SQL注入漏洞扫描
- [ ] XSS漏洞检测
- [ ] 密码安全检查
- [ ] 最佳实践检查

## Week 5: Web Workers优化

### 预估时间：14-18小时

```
// workers/analysisWorker.ts
import { TypeScriptAnalyzer } from '../services/analyzers/TypeScriptAnalyzer';
import { QualityAnalyzer } from '../services/analyzers/QualityAnalyzer';
import { PerformanceAnalyzer } from '../services/analyzers/PerformanceAnalyzer';

interface WorkerMessage {
  type: 'ANALYZE_CODE';
  payload: {
    code: string;
    language: string;
    options: AnalysisOptions;
  };
}

self.onmessage = function(e: MessageEvent<WorkerMessage>) {
  const { type, payload } = e.data;

  if (type === 'ANALYZE_CODE') {
    try {
      // 发送进度更新
      self.postMessage({ type: 'PROGRESS', progress: 0 });

      const tsAnalyzer = new TypeScriptAnalyzer();
      const qualityAnalyzer = new QualityAnalyzer();
      const perfAnalyzer = new PerformanceAnalyzer();

      // 执行分析
      const complexityResult = tsAnalyzer.analyzeComplexity(payload.code);
      self.postMessage({ type: 'PROGRESS', progress: 33 });

      const qualityResult = qualityAnalyzer.analyzeCodeSmells(payload.code);
      self.postMessage({ type: 'PROGRESS', progress: 66 });

      const performanceResult =
perfAnalyzer.analyzePerformanceIssues(payload.code);
      self.postMessage({ type: 'PROGRESS', progress: 100 });

      // 发送结果
      self.postMessage({
        type: 'ANALYSIS_COMPLETE',
        result: {
          complexity: complexityResult,
          quality: qualityResult,
          performance: performanceResult,
          timestamp: Date.now()
        }
      });
    } catch (error) {
      self.postMessage({
        type: 'ANALYSIS_ERROR',
        error: error.message
      });
    }
  }
};
```

**任务清单：**

- [ ] Web Workers实现
- [ ] 大文件处理优化
- [ ] 进度反馈机制
- [ ] 错误处理优化
- [ ] 虚拟滚动实现

---

## 🤖 第三阶段：AI集成与智能建议 (第6-8周)

### Week 6: OpenAI API集成

**预估时间：16-20小时**

```
// services/AIService.ts
import OpenAI from 'openai';

export class AICodeAnalyzer {
  private openai: OpenAI;

  constructor() {
    this.openai = new OpenAI({
      apiKey: process.env.REACT_APP_OPENAI_API_KEY,
      dangerouslyAllowBrowser: true
    });
  }

  async getRefactoringSuggestions(
    code: string,
```

```typescript
    analysisResult: AnalysisResult
  ): Promise<RefactoringSuggestion[]> {
    const prompt = this.buildRefactoringPrompt(code, analysisResult);

    try {
      const response = await this.openai.chat.completions.create({
        model: "gpt-4",
        messages: [
          {
            role: "system",
            content: `You are an expert code reviewer specializing in code
quality and refactoring.
                     Analyze the provided code and suggest specific
improvements.`
          },
          {
            role: "user",
            content: prompt
          }
        ],
        temperature: 0.3,
        max_tokens: 1500
      });

      return this.parseAISuggestions(response.choices[0].message.content);
    } catch (error) {
      console.error('AI analysis failed:', error);
      throw new Error('Failed to get AI suggestions');
    }
  }

  private buildRefactoringPrompt(code: string, analysis: AnalysisResult): string
{
    return `
      Analyze this code and provide specific refactoring suggestions:

      Code:
      \`\`\`typescript
      ${code}
      \`\`\`

      Current Analysis:
      - Cyclomatic Complexity: ${analysis.complexity.cyclomaticComplexity}
      - Code Smells: ${analysis.quality.length} issues found
      - Performance Issues: ${analysis.performance.length} issues found

      Please provide:
      1. Specific refactoring suggestions with code examples
```

```
        2. Explanation of why each change improves the code
        3. Priority level for each suggestion (high/medium/low)
        4. Estimated effort required for each change

        Format your response as JSON with this structure:
        {
          "suggestions": [
            {
              "type": "refactoring",
              "priority": "high",
              "title": "Extract Method",
              "description": "...",
              "originalCode": "...",
              "improvedCode": "...",
              "benefits": ["..."],
              "effort": "low"
            }
          ]
        }
      `;
    }
  }
```

**任务清单：**

- [ ] OpenAI API接口封装
- [ ] Prompt工程优化
- [ ] 响应解析与格式化
- [ ] 错误处理与重试机制
- [ ] API调用限制处理

## Week 7: 智能建议系统

**预估时间：18-22小时**

```
// components/AIAssistant/SuggestionCard.tsx
interface SuggestionCardProps {
  suggestion: RefactoringSuggestion;
  onApply: (suggestion: RefactoringSuggestion) => void;
  onDismiss: (suggestionId: string) => void;
}

const SuggestionCard: React.FC<SuggestionCardProps> = ({
  suggestion,
  onApply,
  onDismiss
}) => {
  const [isExpanded, setIsExpanded] = useState(false);

  return (
    <Card className="suggestion-card">
      <div className="suggestion-header">
        <Tag color={getPriorityColor(suggestion.priority)}>
          {suggestion.priority.toUpperCase()}
        </Tag>
        <h4>{suggestion.title}</h4>
        <Button
          type="text"
          icon={<CloseOutlined />}
          onClick={() => onDismiss(suggestion.id)}
        />
      </div>

      <p className="suggestion-description">
        {suggestion.description}
      </p>

      <div className="suggestion-benefits">
        <h5>Benefits:</h5>
        <ul>
          {suggestion.benefits.map((benefit, index) => (
            <li key={index}>{benefit}</li>
          ))}
        </ul>
      </div>

      <Collapse>
        <Panel header="View Code Changes" key="1">
          <div className="code-comparison">
            <div className="code-before">
              <h6>Before:</h6>
              <pre><code>{suggestion.originalCode}</code></pre>
            </div>
            <div className="code-after">
              <h6>After:</h6>
              <pre><code>{suggestion.improvedCode}</code></pre>
            </div>
          </div>
        </Panel>
      </Collapse>

      <div className="suggestion-actions">
```

```
          <Button
            type="primary"
            onClick={() => onApply(suggestion)}
          >
            Apply Suggestion
          </Button>
          <Button
            onClick={() => setIsExpanded(!isExpanded)}
          >
            {isExpanded ? 'Show Less' : 'Show More'}
          </Button>
        </div>
      </Card>
    );
  };
```

**任务清单：**

- [ ] 重构建议生成
- [ ] 代码优化推荐
- [ ] 最佳实践建议
- [ ] 上下文相关分析
- [ ] 建议应用功能

## Week 8: AI功能增强

**预估时间：16-20小时**

```
// features/ai-suggestions/hooks/useAIAnalysis.ts
export const useAIAnalysis = () => {
  const [suggestions, setSuggestions] = useState<RefactoringSuggestion[]>([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState<string | null>(null);

  const analyzeCode = async (code: string, analysisResult: AnalysisResult) => {
    setLoading(true);
    setError(null);

    try {
      const aiService = new AICodeAnalyzer();
      const aiSuggestions = await aiService.getRefactoringSuggestions(code,
analysisResult);
      setSuggestions(aiSuggestions);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  const applySuggestion = async (suggestion: RefactoringSuggestion) => {
    // 应用AI建议到代码
    try {
      const updatedCode = applyCodeChange(suggestion);
      return updatedCode;
    } catch (err) {
      throw new Error('Failed to apply suggestion');
    }
  };

  return {
    suggestions,
    loading,
    error,
    analyzeCode,
    applySuggestion
  };
};
```

**任务清单：**

- [ ] 自动代码修复
- [ ] 代码生成助手
- [ ] 智能重命名
- [ ] 依赖分析建议
- [ ] 批量建议应用

---

## 📊 第四阶段：数据展示与优化 (第9-10周)

### Week 9: 可视化报告系统

**预估时间：18-22小时**

```
// components/ReportDashboard/index.tsx
import { LineChart, BarChart, PieChart, RadarChart } from 'recharts';
```

```
const ReportDashboard: React.FC<{ analysisHistory: AnalysisResult[] }> = ({
  analysisHistory
}) => {
  const complexityTrend = useMemo(() =>
    analysisHistory.map((result, index) => ({
      version: index + 1,
      complexity: result.complexity.cyclomaticComplexity,
      maintainability: result.complexity.maintainabilityIndex,
      timestamp: result.timestamp
    })), [analysisHistory]);

  const qualityDistribution = useMemo(() => {
    const distribution = analysisHistory.reduce((acc, result) => {
      result.quality.forEach(issue => {
        acc[issue.type] = (acc[issue.type] || 0) + 1;
      });
      return acc;
    }, {} as Record<string, number>);

    return Object.entries(distribution).map(([type, count]) => ({
      type,
      count,
      percentage: (count / analysisHistory.length) * 100
    }));
  }, [analysisHistory]);

  return (
    <div className="report-dashboard">
      <Row gutter={[16, 16]}>
        <Col span={12}>
          <Card title="Complexity Trend">
            <LineChart width={400} height={300} data={complexityTrend}>
              <XAxis dataKey="version" />
              <YAxis />
              <Tooltip />
              <Legend />
              <Line type="monotone" dataKey="complexity" stroke="#8884d8" />
              <Line type="monotone" dataKey="maintainability" stroke="#82ca9d"
/>
            </LineChart>
          </Card>
        </Col>

        <Col span={12}>
          <Card title="Quality Issues Distribution">
            <PieChart width={400} height={300}>
              <Pie
                data={qualityDistribution}
                dataKey="count"
                nameKey="type"
                cx="50%"
                cy="50%"
                outerRadius={80}
                fill="#8884d8"
                label
              />
              <Tooltip />
            </PieChart>
          </Card>
        </Col>
      </Row>

      <Row gutter={[16, 16]} style={{ marginTop: 16 }}>
        <Col span={24}>
          <Card title="Quality Score Over Time">
            <QualityScoreChart data={analysisHistory} />
          </Card>
        </Col>
      </Row>
    </div>
  );
};
```

**任务清单：**

- [ ] Recharts图表集成
- [ ] 代码质量仪表盘
- [ ] 趋势分析图表
- [ ] 交互式数据展示
- [ ] 自定义报告配置

## Week 10: 性能优化与部署

**预估时间：14-18小时**

```
// Performance optimizations
// 1. 代码分割
const AIAssistant = lazy(() => import('../features/ai-
suggestions/AIAssistant'));
const ReportDashboard = lazy(() => import('../components/ReportDashboard'));

// 2. 虚拟化长列表
```

```
import { FixedSizeList as List } from 'react-window';

const SuggestionsList = ({ suggestions }: { suggestions: RefactoringSuggestion[]
}) => {
  const Row = ({ index, style }: { index: number; style: CSSProperties }) => (
    <div style={style}>
      <SuggestionCard suggestion={suggestions[index]} />
    </div>
  );

  return (
    <List
      height={600}
      itemCount={suggestions.length}
      itemSize={200}
      width="100%"
    >
      {Row}
    </List>
  );
};

// 3. 缓存策略
const analysisCache = new Map<string, AnalysisResult>();

const getCachedAnalysis = (codeHash: string): AnalysisResult | null => {
  return analysisCache.get(codeHash) || null;
};

const setCachedAnalysis = (codeHash: string, result: AnalysisResult) => {
  analysisCache.set(codeHash, result);
};
```

**任务清单：**

- [ ] Bundle大小优化
- [ ] 代码分割和懒加载
- [ ] 缓存策略实现
- [ ] 虚拟滚动优化
- [ ] 生产环境部署
- [ ] 性能监控集成

---

## 💡 开发建议与注意事项

### 1. 时间管理建议

- **工作日**：每天2小时，专注核心功能
- **周末**：6-8小时，处理复杂模块
- **缓冲时间**：每个阶段预留20%时间处理bug

### 2. 技术风险预估

| 功能模块 | 风险等级 | 时间缓冲 | 主要挑战 |
|---|---|---|---|
| AST解析 | 🔴 高 | +30% | 语法解析复杂度 |
| AI集成 | 🟡 中 | +20% | API调用限制 |
| 性能优化 | 🟡 中 | +15% | 大文件处理 |
| 可视化 | 🟢 低 | +10% | 图表配置 |

### 3. 学习资源准备

- **AST解析**：TypeScript Compiler API文档
- **AI集成**：OpenAI API文档，Prompt工程指南
- **性能优化**：React性能优化最佳实践
- **数据可视化**：Recharts官方文档

### 4. 面试准备要点

- **第5周后**：可以开始准备基础版本演示
- **第8周后**：具备完整的技术难点讲解
- **第10周后**：拥有商业级别的完整项目

### 5. 核心技术亮点

1. **AST深度解析**：展示编译原理功底
2. **AI集成经验**：符合当前技术趋势
3. **性能优化**：Web Workers + 虚拟化
4. **工程化实践**：完整的开发流程

这个规划既保证了项目的完整性，又考虑了学习曲线和技术难度。建议先完成基础版本，再逐步添加高级功能。