# 第一周综合练习：文件日志分析器 - 详细需求说明

## 🎯 练习目标与价值

### 核心目标

通过这个综合练习，将第1周学习的Java核心知识融会贯通，同时为后续Monkey测试工具的日志分析模块打下技术基础。

### 学习价值

- \*\*技术整合\*\*: 将分散的Java知识点串联成完整项目

- \*\*实战导向\*\*: 解决测试工程师日常工作中的实际问题

- \*\*能力检验\*\*: 验证第1周学习成果，发现知识盲区

- \*\*项目预热\*\*: 为后续大型项目积累开发经验

## 📋 详细功能需求

### 1. 基础统计功能

#### 1.1 文件读取与解析

```java

// 需求描述：读取日志文件并解析为结构化数据

输入: 日志文件路径

输出: 解析后的日志对象集合

处理逻辑:

- 支持常见日志格式（如Log4j、Spring Boot默认格式）

- 自动识别日志级别和时间戳

- 处理文件读取异常和格式错误

```

#### 1.2 基础指标统计

```java

// 需求描述：计算基础统计指标

指标清单:

□ 日志总行数

□ 有效日志行数（成功解析的行）

□ 不同日志级别的数量分布

- INFO级别数量及占比

- WARN级别数量及占比

- ERROR级别数量及占比

- DEBUG级别数量及占比

□ 日志时间范围

- 最早日志时间

- 最晚日志时间

- 日志覆盖时长

```

### 2. 错误分析功能

#### 2.1 错误日志提取

```java

// 需求描述：专门分析ERROR级别的日志

功能点:

□ 提取所有ERROR级别日志

□ 按错误类型自动分类

□ 统计每种错误出现的频率

□ 错误发生的时间趋势分析

```

#### 2.2 错误模式识别

```java

// 需求描述：识别常见的错误模式

识别模式:

□ 空指针异常（NullPointerException）

□ 数据库连接异常

□ 文件读写异常

□ 网络超时异常

□ 业务逻辑异常

```

### 3. 性能分析功能

#### 3.1 执行时间分析

```java

// 需求描述：从日志中提取性能数据

分析内容:

□ 识别包含执行时间的日志条目

□ 统计平均执行时间

□ 找出执行时间超过阈值的操作

□ 性能瓶颈定位

```

#### 3.2 资源使用分析

```java

// 需求描述：分析系统资源使用情况

分析维度:

□ 内存使用情况趋势

□ 线程数量变化

□ 数据库连接池状态

□ GC日志分析（如果存在）

```

## 🛠️ 技术实现要求

### 1. 必须使用的Java特性

#### 1.1 文件操作

```java

// 要求：使用Java NIO API

必用API:

□ Files.readAllLines() - 读取文件

□ Paths.get() - 路径处理

□ try-with-resources - 资源管理

禁用:

❌ 传统的FileInputStream

❌ 手动关闭资源

```

#### 1.2 集合框架

```java

// 要求：合理选择集合类型

集合使用规范:

□ List<String> - 存储日志行

□ List<LogEntry> - 存储解析后的日志对象

□ Map<String, Integer> - 级别统计

□ Map<String, List<LogEntry>> - 错误分类

```

#### 1.3 Stream API

```java

// 要求：大量使用Stream操作

Stream应用场景:

□ 数据过滤: filter()

□ 数据转换: map()

□ 数据分组: groupingBy()

□ 数据统计: summarizingDouble()

□ 数据排序: sorted()

```

#### 1.4 异常处理

```java

// 要求：完善的异常处理机制

异常处理策略:

□ 检查异常: 必须捕获或声明

□ 运行时异常: 合理处理

□ 自定义异常: 创建日志解析异常

□ 资源泄漏: 确保资源正确释放

```

### 2. 代码结构规范

#### 2.1 包结构

```

src/main/java/com/yourname/loganalyzer/

├── Main.java # 程序入口

├── model/

│ ├── LogEntry.java # 日志条目实体

│ └── AnalysisResult.java # 分析结果封装

├── service/

│ ├── LogFileReader.java # 文件读取服务

│ ├── LogParser.java # 日志解析服务

│ ├── LogAnalyzer.java # 分析引擎

│ └── ReportGenerator.java # 报告生成服务

└── util/

├── DateUtils.java # 日期工具类

└── StringUtils.java # 字符串工具类

```

#### 2.2 类职责划分

\*\*LogEntry.java\*\* - 数据模型

```java

public class LogEntry {

private LocalDateTime timestamp;

private String level;

private String threadName;

private String className;

private String message;

private Map<String, Object> extraFields;

// 构造方法、getter、setter

// 解析方法：从字符串解析为LogEntry

public static LogEntry parse(String logLine) { ... }

}

```

\*\*LogFileReader.java\*\* - 文件操作

```java

public class LogFileReader {

public List<String> readLogFile(String filePath) { ... }

public List<LogEntry> readAndParseLogFile(String filePath) { ... }

}

```

\*\*LogAnalyzer.java\*\* - 核心业务逻辑

```java

public class LogAnalyzer {

public AnalysisResult analyze(List<LogEntry> entries) { ... }

public Map<String, Long> countByLevel(List<LogEntry> entries) { ... }

public List<LogEntry> findErrors(List<LogEntry> entries) { ... }

public PerformanceStats analyzePerformance(List<LogEntry> entries) { ... }

}

```

## 📊 输入输出规范

### 1. 输入格式支持

#### 1.1 标准日志格式

```

// Spring Boot默认格式

2024-01-15 10:00:00.123 INFO com.example.Service - 业务处理开始

2024-01-15 10:00:01.456 ERROR com.example.Service - 数据库连接失败

```

#### 1.2 自定义格式扩展

```java

// 支持通过正则表达式自定义格式

public class LogPattern {

private String pattern; // 正则表达式

private String[] groups; // 分组映射

}

```

### 2. 输出格式要求

#### 2.1 控制台输出

```

=== 日志分析报告 ===

文件: /path/to/app.log

分析时间: 2024-01-15 14:30:00

📊 基础统计

总行数: 1,024

有效行数: 1,000 (97.7%)

时间范围: 2024-01-15 10:00:00 - 2024-01-15 10:15:30

📈 级别分布

INFO: 650 (65.0%)

WARN: 200 (20.0%)

ERROR: 150 (15.0%)

DEBUG: 0 (0.0%)

🚨 错误分析

错误总数: 150

最常见错误:

- 数据库连接超时: 45次

- 空指针异常: 30次

- 文件未找到: 25次

⚡ 性能分析

平均响应时间: 125ms

超时操作(>500ms): 12个

最大响应时间: 2.3s

```

#### 2.2 文件输出（可选）

```java

// 支持导出分析结果到文件

格式选择:

□ 文本格式 (.txt)

□ JSON格式 (.json)

□ HTML格式 (.html) - 带图表

```

## 🧪 测试数据准备

### 1. 测试日志文件示例

创建 `test-data/sample.log`：

```

2024-01-15 10:00:00.123 INFO [main] com.example.Application - Starting Application v1.0

2024-01-15 10:00:01.234 INFO [main] com.example.Service - 初始化数据库连接

2024-01-15 10:00:02.345 DEBUG [pool-1] com.example.Dao - 执行SQL: SELECT \* FROM users

2024-01-15 10:00:03.456 INFO [pool-1] com.example.Service - 用户查询完成, 耗时: 120ms

2024-01-15 10:00:04.567 WARN [scheduler-1] com.example.Job - 任务执行较慢: 处理了1000条数据, 耗时: 1500ms

2024-01-15 10:00:05.678 ERROR [http-nio-8080] com.example.Controller - 用户登录失败: NullPointerException

2024-01-15 10:00:06.789 INFO [http-nio-8080] com.example.Controller - 请求处理完成: /api/login, 耗时: 45ms

2024-01-15 10:00:07.890 ERROR [pool-2] com.example.Service - 数据库连接池耗尽

2024-01-15 10:00:08.901 INFO [main] com.example.Application - Application shutdown

```

### 2. 边界情况测试

```java

// 需要处理的特殊情况

边界情况:

□ 空文件

□ 超大文件(>100MB)

□ 格式错误的日志行

□ 混合多种日志格式

□ 编码问题(UTF-8, GBK等)

```

## 📝 验收标准

### 1. 功能验收清单

#### 核心功能 ✅

- [ ] 能够正确读取和解析标准格式日志文件

- [ ] 准确统计各级别日志数量

- [ ] 正确识别和分类错误日志

- [ ] 性能数据分析基本准确

#### 代码质量 ✅

- [ ] 使用Stream API进行数据处理

- [ ] 合理的异常处理机制

- [ ] 清晰的代码结构和命名

- [ ] 适当的代码注释

#### 用户体验 ✅

- [ ] 清晰的命令行输出格式

- [ ] 有意义的错误提示信息

- [ ] 支持相对路径和绝对路径

### 2. 性能要求

```java

// 性能基准

处理能力:

□ 10MB日志文件: 处理时间 < 5秒

□ 内存使用: 峰值 < 512MB

□ CPU使用: 平均 < 50%

```

## 🚀 扩展挑战（可选）

### 1. 高级功能

```java

// 如果基础功能完成顺利，可以尝试：

扩展功能:

□ 实时日志监控（监听文件变化）

□ 多文件批量分析

□ 自定义分析规则

□ 生成可视化图表报告

```

### 2. 架构优化

```java

// 为后续项目做准备：

架构改进:

□ 使用设计模式重构（如策略模式支持不同日志格式）

□ 添加单元测试覆盖

□ 配置化支持（通过配置文件定义规则）

□ 插件化架构（可扩展的分析器）

```

## 💡 开发建议

### 1. 开发流程

1. \*\*先设计\*\*: 明确类结构和接口定义

2. \*\*分模块\*\*: 逐个实现文件读取、解析、分析模块

3. \*\*早测试\*\*: 每完成一个功能立即测试

4. \*\*重构\*\*: 功能完成后优化代码结构

### 2. 调试技巧

```java

// 推荐的调试方法

调试策略:

□ 使用小的测试文件快速验证

□ 打印中间结果检查解析正确性

□ 使用IDE调试器单步跟踪

□ 编写单元测试验证边界情况

```

### 3. 学习重点

```java

// 这个练习要掌握的核心技能

技术重点:

✅ Java文件操作和NIO

✅ 集合框架的合理使用

✅ Stream API的熟练应用

✅ 正则表达式解析文本

✅ 面向对象设计思想

```

这个详细的练习需求应该能让你清楚地知道第一周要完成什么，以及如何将学到的Java知识应用到实际项目中。记得先从简单的功能开始，逐步完善！