# 个人财务追踪器

## JavaScript 面向对象与函数式编程综合实践

计算机大类专业导论课程组

2025-11-14

## 题目描述

请实现一个简单的**个人财务追踪系统**，帮助用户记录和分析日常收支情况。这个系统需要支持添加交易记录、查询统计信息、以及按条件筛选交易。

本题目旨在综合考查：

- ✅ **面向对象编程**：类的设计与实现
- ✅ **函数式编程**：高阶函数的应用
- ✅ **数据处理**：数组和对象的操作
- ✅ **数据验证**：输入参数的检查
- ✅ **ES6 特性**：现代 JavaScript 语法

## 功能要求

### 1. 交易记录类（Transaction）

创建一个 `Transaction` 类，包含以下属性：

| 属性 | 类型 | 说明 |
|---|---|---|
| `id` | Number | 交易 ID（自动生成，从 1 开始递增） |
| `type` | String | 交易类型（`"income"` 收入 或 `"expense"` 支出） |
| `amount` | Number | 金额（必须为正数） |
| `category` | String | 分类（如：工资、餐饮、交通、娱乐等） |
| `description` | String | 交易描述 |
| `date` | Date | 交易日期（Date 对象，默认为当前日期） |

**方法要求：**

```
class Transaction {
    constructor(id, type, amount, category, description, date = new Date()) {
        // 实现构造函数
        // 需要验证：type 必须是 "income" 或 "expense"
        //           amount 必须是正数
    }

    // 返回交易的字符串表示
    toString() {
        // 返回格式化的交易信息
    }
}
```

## 2. 财务追踪器类（FinanceTracker）

创建一个 `FinanceTracker` 类，实现以下方法：

### 2.1 基础功能（必做）

```
class FinanceTracker {
    constructor() {
        // 初始化交易列表和ID计数器
    }

    /**
     * 添加交易记录
     * @param {string} type - 交易类型 ("income" 或 "expense")
     * @param {number} amount - 金额（正数）
     * @param {string} category - 分类
     * @param {string} description - 描述
     * @returns {Transaction} 新创建的交易对象
     */
    addTransaction(type, amount, category, description) {
        // 1. 验证参数
        // 2. 创建Transaction对象
        // 3. 添加到交易列表
        // 4. 返回新创建的交易
    }

    /**
     * 获取所有交易记录
     * @returns {Array<Transaction>} 所有交易记录
     */
    getAllTransactions() {
        // 返回所有交易记录数组
    }

    /**
```

```
     * 计算总收入
     * @returns {number} 总收入金额
     * 要求：使用 filter + reduce 实现
     */
    getTotalIncome() {
        // 筛选收入类型，累加金额
    }

    /**
     * 计算总支出
     * @returns {number} 总支出金额
     * 要求：使用 filter + reduce 实现
     */
    getTotalExpense() {
        // 筛选支出类型，累加金额
    }

    /**
     * 计算余额（总收入 - 总支出）
     * @returns {number} 当前余额
     */
    getBalance() {
        // 计算并返回余额
    }

    /**
     * 按分类筛选交易
     * @param {string} category - 分类名称
     * @returns {Array<Transaction>} 指定分类的所有交易
     * 要求：使用 filter 方法
     */
    getTransactionsByCategory(category) {
        // 筛选指定分类的交易
    }

    /**
     * 按类型筛选交易
     * @param {string} type - 交易类型 ("income" 或 "expense")
     * @returns {Array<Transaction>} 指定类型的所有交易
     * 要求：使用 filter 方法
     */
    getTransactionsByType(type) {
        // 筛选指定类型的交易
    }

    /**
     * 获取分类支出统计
     * @returns {Object} 键为分类名，值为该分类的总支出
```

```
     * 示例：{ "餐饮": 400, "交通": 150, "娱乐": 200 }
     * 要求：使用 reduce 方法实现
     */
    getExpenseByCategory() {
        // 统计各分类的支出
    }

    /**
     * 删除交易记录
     * @param {number} id - 交易ID
     * @returns {boolean} true表示删除成功，false表示未找到
     */
    deleteTransaction(id) {
        // 根据ID删除交易记录
    }
}
```

### 2.2 进阶功能（选做）

实现以下一个或多个进阶功能可获得加分：

#### 选做 1：日期范围查询（+3 分）

```
/**
 * 按日期范围查询交易
 * @param {Date} startDate - 开始日期
 * @param {Date} endDate - 结束日期
 * @returns {Array<Transaction>} 日期范围内的所有交易
 */
getTransactionsByDateRange(startDate, endDate) {
    // 筛选日期范围内的交易
}
```

#### 选做 2：数据导出功能（+2 分）

```
/**
 * 导出为JSON格式
 * @returns {string} JSON字符串
 */
exportToJSON() {
    // 将所有交易记录转换为JSON字符串
}

/**
 * 从JSON导入
 * @param {string} jsonString - JSON字符串
 */
importFromJSON(jsonString) {
```

```
    // 从JSON字符串恢复交易记录
}
```

### 选做 3：预算管理功能（+5 分）

```
/**
 * 设置分类预算
 * @param {string} category - 分类名称
 * @param {number} amount - 预算金额
 */
setBudget(category, amount) {
    // 为某个分类设置月预算
}

/**
 * 检查预算状态
 * @param {string} category - 分类名称
 * @returns {Object} { budget, spent, remaining, percentage }
 */
checkBudget(category) {
    // 返回预算使用情况
    // budget: 预算总额
    // spent: 已花费
    // remaining: 剩余额度（可能为负数）
    // percentage: 使用百分比
}

/**
 * 获取所有预算报告
 * @returns {Array} 所有分类的预算状态
 */
getAllBudgetReports() {
    // 返回所有设置了预算的分类的使用情况
}
```

### 选做 4：数据可视化（+5 分）

```
/**
 * 生成支出分布的简单文本图表
 * @returns {string} ASCII艺术图表
 * 示例输出：
 * 支出分布：
 * 餐饮    ████████ 40%
 * 交通    ████     20%
 * 娱乐    ██████   30%
 */
generateExpenseChart() {
```

```
    // 生成文本图表
}

/**
 * 生成月度收支趋势
 * @param {number} months - 最近N个月
 * @returns {Object} 月度统计数据
 */
getMonthlyTrend(months = 6) {
    // 按月统计收入和支出
}
```

## 测试要求

你的代码应该能够通过以下测试用例：

```
// ========== 基础功能测试 ==========
console.log("========== 个人财务追踪器测试 ==========\n");

// 1. 创建追踪器
const tracker = new FinanceTracker();
console.log("✓ 创建财务追踪器");

// 2. 添加收入记录
console.log("\n--- 添加收入记录 ---");
tracker.addTransaction("income", 5000, "工资", "2024年1月工资");
tracker.addTransaction("income", 500, "奖金", "绩效奖金");
console.log("✓ 添加2条收入记录");

// 3. 添加支出记录
console.log("\n--- 添加支出记录 ---");
tracker.addTransaction("expense", 300, "餐饮", "周末聚餐");
tracker.addTransaction("expense", 150, "交通", "地铁卡充值");
tracker.addTransaction("expense", 200, "娱乐", "电影票");
tracker.addTransaction("expense", 100, "餐饮", "工作日外卖");
console.log("✓ 添加4条支出记录");

// 4. 统计功能测试
console.log("\n--- 统计信息 ---");
console.log(`总收入: ¥${tracker.getTotalIncome()}`);        // 应该是 5500
console.log(`总支出: ¥${tracker.getTotalExpense()}`);       // 应该是 750
console.log(`当前余额: ¥${tracker.getBalance()}`);          // 应该是 4750

// 5. 筛选功能测试
console.log("\n--- 筛选功能 ---");
const foodExpenses = tracker.getTransactionsByCategory("餐饮");
console.log(`餐饮支出记录数: ${foodExpenses.length}`);      // 应该是 2
```

```
console.log(`餐饮总支出: ¥${foodExpenses.reduce((sum, t) => sum + t.amount, 0)}
`); // 400

const allExpenses = tracker.getTransactionsByType("expense");
console.log(`支出记录总数: ${allExpenses.length}`);          // 应该是 4

// 6. 分类统计测试
console.log("\n--- 分类支出统计 ---");
const expenseByCategory = tracker.getExpenseByCategory();
console.log(expenseByCategory);
// 应该输出: { "餐饮": 400, "交通": 150, "娱乐": 200 }

// 7. 删除功能测试
console.log("\n--- 删除操作 ---");
const deleted = tracker.deleteTransaction(1);
console.log(`删除ID为1的记录: ${deleted ? '成功' : '失败'}`);  // true
console.log(`删除后余额: ¥${tracker.getBalance()}`);           // -250

// 8. 所有交易记录
console.log("\n--- 所有交易记录 ---");
tracker.getAllTransactions().forEach(t => {
    console.log(t.toString());
});

console.log("\n========== 测试完成 ==========");
```

**预期输出：**

```
========== 个人财务追踪器测试 ==========

✓ 创建财务追踪器

--- 添加收入记录 ---
✓ 添加2条收入记录

--- 添加支出记录 ---
✓ 添加4条支出记录

--- 统计信息 ---
总收入: ¥5500
总支出: ¥750
当前余额: ¥4750

--- 筛选功能 ---
餐饮支出记录数: 2
餐饮总支出: ¥400
支出记录总数: 4
```

```
--- 分类支出统计 ---
{ 餐饮: 400, 交通: 150, 娱乐: 200 }

--- 删除操作 ---
删除ID为1的记录: 成功
删除后余额: ￥-250

--- 所有交易记录 ---
[#2] 收入 ￥500.00 - 奖金 (绩效奖金)
[#3] 支出 ￥300.00 - 餐饮 (周末聚餐)
[#4] 支出 ￥150.00 - 交通 (地铁卡充值)
[#5] 支出 ￥200.00 - 娱乐 (电影票)
[#6] 支出 ￥100.00 - 餐饮 (工作日外卖)

========== 测试完成 ==========
```

# 技术要求

## 必需使用的技术

1. **ES6 Class 语法**
   - 使用 `class` 关键字定义类
   - 实现构造函数和方法
2. **数据验证**
   - 对 `type` 参数进行验证（只能是 “income” 或 “expense”）
   - 对 `amount` 参数进行验证（必须是正数）
   - 对无效输入抛出错误或返回错误信息
3. **函数式编程**
   - 必须使用 `filter` 方法进行数据筛选
   - 必须使用 `reduce` 方法进行数据累加和统计
   - 可以使用 `map`、`find`、`some`、`every` 等其他方法
4. **ES6 现代语法**
   - 使用箭头函数
   - 使用模板字符串
   - 使用解构赋值（可选）
   - 使用默认参数

## 代码规范

1. **命名规范**
   - 类名使用大驼峰（PascalCase）：`Transaction`, `FinanceTracker`
   - 变量和方法名使用小驼峰（camelCase）：`getTotalIncome`, `addTransaction`
   - 常量使用大写下划线（UPPER_SNAKE_CASE）：`MAX_AMOUNT`

2. **注释要求**
   - 每个类添加文档注释说明其用途
   - 关键方法添加注释说明参数和返回值
   - 复杂逻辑添加行内注释
3. **错误处理**
   - 对用户输入进行验证
   - 对异常情况给出友好的错误提示

# 提交要求

## 1. 文件组织

创建项目目录结构：

```
finance-tracker/
├── finance-tracker.js    # 主要代码文件
├── test.js               # 测试代码
├── package.json          # Node.js配置（如果使用模块）
└── README.md             # 项目说明
```

## 2. 必需文件

**finance-tracker.js**

包含 `Transaction` 和 `FinanceTracker` 两个类的完整实现。

**test.js**

包含完整的测试代码，演示所有功能。

**README.md**

包含以下内容：

```
# 个人财务追踪器

## 功能说明
(列出实现的功能)

## 运行方法
\`\`\`bash
node test.js
\`\`\`

## 实现的功能
- [x] 基础交易管理
- [x] 统计查询功能
- [x] 数据验证
- [ ] 日期范围查询（选做）
```

```
- [ ] 数据导出（选做）
...

## 技术栈
- JavaScript ES6
- Node.js

## 开发者
姓名：xxx
学号：xxx
```

### 3. 实验报告

使用提供的 `report-template.qmd` 模板完成实验报告，需要包含：

- 需求分析和设计思路
- 核心代码说明
- 运行截图
- 测试结果
- 问题与解决
- 概念理解（开放式问答）
- 心得体会

## 评分标准

### 基础功能（60 分）

| 项目 | 分值 | 评分要点 |
|---|---|---|
| **Transaction 类** | 10 分 | |
| • 属性定义完整 | 3 分 | 包含所有必需属性 |
| • 数据验证 | 4 分 | type 和 amount 验证正确 |
| • toString 方法 | 3 分 | 格式化输出 |
| **FinanceTracker 类** | 50 分 | |
| • addTransaction | 8 分 | 正确创建和存储交易 |
| • getTotalIncome | 6 分 | 使用 filter+reduce |
| • getTotalExpense | 6 分 | 使用 filter+reduce |
| • getBalance | 4 分 | 正确计算余额 |
| • getTransactionsByCategory | 6 分 | 使用 filter 筛选 |
| • getTransactionsByType | 6 分 | 使用 filter 筛选 |

| 项目 | 分值 | 评分要点 |
|---|---|---|
| • getExpenseByCategory | 8 分 | 使用 reduce 统计 |
| • deleteTransaction | 6 分 | 正确删除记录 |

### 代码质量（20 分）

| 项目 | 分值 | 评分要点 |
|---|---|---|
| **代码规范** | 6 分 | 命名规范、格式统一 |
| **注释文档** | 6 分 | 关键代码有注释 |
| **错误处理** | 4 分 | 输入验证、边界处理 |
| **ES6 特性** | 4 分 | 箭头函数、模板字符串等 |

### 测试与文档（10 分）

| 项目 | 分值 | 评分要点 |
|---|---|---|
| **测试覆盖** | 5 分 | 测试用例完整 |
| **README 文档** | 3 分 | 说明清晰 |
| **运行演示** | 2 分 | 能正常运行 |

### 进阶功能（10 分）

| 功能 | 分值 |
|---|---|
| 日期范围查询 | 3 分 |
| 数据导入导出 | 2 分 |
| 预算管理 | 5 分 |
| 数据可视化 | 5 分 |
| 其他创新功能 | 2-5 分 |

**总计：100 分（基础 90 分 + 进阶 10 分）**

## 参考资料

### 课程资源

- **Slides**：`slides/javascript/index.qmd`
  - 第③部分：函数式编程
  - 第④部分：面向对象编程
- **示例代码**：`slides/javascript/school/`

- 学校管理系统示例

### 在线资源

- MDN - Array 方法
- JavaScript.info - 类
- ES6 入门 - 阮一峰

### 函数式编程方法速查

```
// filter - 筛选符合条件的元素
arr.filter(item => item.type === "expense")

// reduce - 累加/归约
arr.reduce((sum, item) => sum + item.amount, 0)

// map - 转换每个元素
arr.map(item => item.category)

// find - 查找第一个符合条件的元素
arr.find(item => item.id === 1)

// some - 是否有元素满足条件
arr.some(item => item.amount > 1000)

// every - 是否所有元素满足条件
arr.every(item => item.amount > 0)
```

## 学习目标

完成本题目后，你应该能够：

1. ✅ 熟练使用 ES6 的 Class 语法定义类
2. ✅ 理解面向对象编程的封装概念
3. ✅ 掌握 JavaScript 数组的函数式编程方法
4. ✅ 理解函数式编程的声明式风格
5. ✅ 学会数据验证和错误处理
6. ✅ 培养系统设计和问题分解能力

## 提示与建议

### 开发步骤建议

1. **第一步**：实现 `Transaction` 类
   - 先完成基本的构造函数
   - 添加数据验证
   - 实现 `toString` 方法

2. **第二步**：实现 `FinanceTracker` 的基础功能
   - 实现 `addTransaction`
   - 实现 `getAllTransactions`
   - 测试这两个方法
3. **第三步**：实现统计功能
   - 实现 `getTotalIncome` 和 `getTotalExpense`
   - 实现 `getBalance`
   - 学习使用 filter 和 reduce
4. **第四步**：实现筛选和分类统计
   - 实现按分类和类型筛选
   - 实现分类支出统计
   - 重点练习 reduce 的使用
5. **第五步**：完善其他功能
   - 实现删除功能
   - 添加错误处理
   - 编写测试代码
6. **第六步**（可选）：实现进阶功能

## 常见问题

### Q1: 如何自动生成递增的 ID？

A: 在 `FinanceTracker` 中维护一个计数器：

```
constructor() {
    this.transactions = [];
    this.nextId = 1;  // ID计数器
}

addTransaction(...) {
    const transaction = new Transaction(this.nextId++, ...);
    this.transactions.push(transaction);
    return transaction;
}
```

### Q2: reduce 方法如何使用？

A: reduce 基本语法：

```
array.reduce((累加器, 当前元素) => {
    // 处理逻辑
    return 新的累加器值;
}, 初始值);

// 示例：求和
```

```
[1, 2, 3, 4].reduce((sum, num) => sum + num, 0);  // 10

// 示例：分组统计
transactions.reduce((result, t) => {
    result[t.category] = (result[t.category] || 0) + t.amount;
    return result;
}, {});
```

**Q3: 如何验证日期对象？**

A: 使用 `instanceof` 和 `isNaN`：

```
if (!(date instanceof Date) || isNaN(date)) {
    throw new Error("无效的日期");
}
```

**Q4: 如何格式化金额输出？**

A: 使用 `toFixed()` 方法：

```
amount.toFixed(2)  // "100.00"
`¥${amount.toFixed(2)}`  // "¥100.00"
```

# 扩展思考

完成基础功能后，思考以下问题：

1. **设计模式**：如果要支持多个用户，如何设计数据结构？
2. **数据持久化**：如何将数据保存到文件，下次运行时恢复？
3. **性能优化**：如果有 10 万条交易记录，如何优化查询性能？
4. **功能扩展**：
   - 如何实现定期自动记账（如每月工资）？
   - 如何实现标签功能（一笔交易可以有多个标签）？
   - 如何实现搜索功能（按描述关键字搜索）？
5. **用户界面**：如果要开发 Web 界面，需要哪些 API？

---

**如有问题，请及时与老师联系。** 📊💰