??-??

??????????

1. ????

[illegible]

2. ????

2.1 ????

[illegible] [illegible]

```
wimoor-ui/src/views/finance/report/
├── items/                # [illegible]
│   ├── components/
│   │   └── report_sheet.vue  # [illegible]
│   └── index.vue              # [illegible]
└── templates/            # [illegible]
    ├── components/
    │   ├── template_item_edit.vue  # [illegible]
    │   └── template_items.vue      # [illegible]
    └── index.vue                   # [illegible]
```

[illegible] [illegible]

- `items/index.vue`: [illegible]
- `report_sheet.vue`: [illegible]
- `templates/index.vue`: [illegible]
- `template_items.vue`: [illegible]
- `template_item_edit.vue`: [illegible]

2.2 ????

核心控制器 :

- `ReportController.java`: 报表生成控制器，处理报表生成请求
- `FinReportItemsController.java`: 报表项目控制器，处理报表项目的 CRUD操作

核心服务 :

- `IFinReportTemplatesService.java`: 报表模板服务接口
- `FinReportTemplatesServiceImpl.java`: 报表模板服务实现，包含报表生成核心逻辑

数据传输对象 :

- `ReportGenerateRequest.java`: 报表生成请求 DTO
- `ReportGenerateResponse.java`: 报表生成响应 DTO
- `ReportItemValueDTO.java`: 报表项目值 DTO

3. ????

3.1 ??????

功能说明 :

- 创建、编辑、删除报表模板
- 配置模板的基本信息（名称、编码、类型、描述等）
- 管理模板的状态

核心代码 :

```
<!-- 模板树结构 -->
<el-tree
  class="template-tree"
  :data="templateTree"
  :highlight-current="true"
  :props="treeProps"
  node-key="templateId"
  @node-click="handleTemplateSelect"
>
  <!-- 自定义节点内容 -->
</el-tree>
```

```
<!-- 模板新增/修改对话框 -->
<el-dialog :title="formTitle" v-model="openForm" width="600px" append-to-body>
  <el-form ref="templateFormRef" :model="form" :rules="rules" label-width="120px">
    <el-form-item label="模板名称" prop="templateName">
      <el-input v-model="form.templateName" placeholder="请输入模板名称"></el-input>
    </el-form-item>
    <!-- 其他表单字段 -->
  </el-form>
</el-dialog>
```

3.2 ??????

功能说明　　：

- 为报表模板配置项目
- 设置项目的层级关系（父级项目）
- 配置项目的计算公式和数据来源
- 管理项目的显示属性（是否显示、是否显示零值等）

核心代码　　：

```
<!-- 项目列表 -->
<el-table
    v-loading="loading"
    :data="itemsList"
    border
    size="small"
    :row-style="setRowStyle"
>
  <el-table-column prop="lineNumber" label="行次" width="60" align="center"></el-table-column>
  <el-table-column prop="itemCode" label="项目编码" width="300"></el-table-column>
  <el-table-column label="项目名称">
    <template #default="{ row }">
      <span :style="{ 'padding-left': (row.itemLevel - 1) * 20 + 'px' }">{{ row.itemName
}}</span>
    </template>
  </el-table-column>
  <!-- 其他列 -->
</el-table>
```

3.3 ????

功能说明　　：

- 根据选择的模板和期间生成报表数据
- 支持多期间比较（当前期间与对比期间）
- 自动计算项目金额和财务指标
- 生成图表数据用于可视化展示

核心代码　　：

```
public ReportGenerateResponse generateReport(ReportGenerateRequest request) {
    ReportGenerateResponse response = new ReportGenerateResponse();

    try {
        String groupid = request.getGroupid();
        String templateCode = request.getTemplateCode();
        String period = request.getPeriod();

        // 1. 获取报表模板
        List<FinReportTemplates> templates =
this.finReportTemplatesMapper.selectFinReportTemplatesList(finReportTemplates);

        // 2. 获取报表项目结构
        List<FinReportItems> reportItems =
finReportItemsService.selectFinReportItemsList(itemsquery);

        // 3. 计算当前期间数据
        Map<String, BigDecimal> currentAmounts = calculateReportAmounts(groupid, period,
reportItems, template);

        // 4. 计算比较期间数据（如果需要）
        Map<String, BigDecimal> compareAmounts = new HashMap<>();
        if (request.getIncludeComparison() && comparePeriod != null &&
!comparePeriod.isEmpty()) {
            compareAmounts = calculateReportAmounts(groupid, comparePeriod, reportItems,
template);
        }

        // 5. 构建响应数据
```

```
        List<ReportItemValueDTO> itemDTOs = buildReportItems(reportItems, currentAmounts,
compareAmounts, amountUnit);

        // 6. 计算财务指标、图表数据等

        // 7. 组装响应
        response.setItems(itemDTOs);
        response.setFinancialRatios(financialRatios);
        response.setChartData(chartData);
        response.setSummary(summary);

    } catch (Exception e) {
        response.setSuccess(false);
        response.setMessage("报表生成失败: " + e.getMessage());
    }

    return response;
}
```

4. ??????

4.1 ????????

功能说明：

- 支持多种公式类型: DIRECT（直接科目）、FORMULA（自定义公式）、CUSTOM（自定义规则）、CALCULATED（自动计算）
- 按层级从高到低排序计算，确保父级项目在子级之后计算

核心代码：

```
private Map<String, BigDecimal> calculateReportAmounts(String groupid, String period,
                                                       List<FinReportItems> reportItems,
                                                       FinReportTemplates template) {
    // 获取科目余额数据
    Map<String, BigDecimal> amounts = subjectBalanceService.getAllSubjectBalance(groupid,
period);

    // 按层级从高到低排序计算，确保父级项目在子级之后计算
```

```
    List<FinReportItems> sortedItems = reportItems.stream()
            .sorted(Comparator.comparing(FinReportItems::getItemLevel).reversed())
            .collect(Collectors.toList());

    for (FinReportItems item : sortedItems) {
        if (!item.getIsShow()) {
            continue; // □□□□□□□□□
        }

        BigDecimal amount = calculateItemAmount(groupid, period, item, amounts, reportItems);
        amounts.put(item.getItemCode(), amount);
    }

    return amounts;
}

private BigDecimal calculateItemAmount(String groupid, String period, FinReportItems item,
                                       Map<String, BigDecimal> amounts, List<FinReportItems>
allItems) {
    String formulaType = item.getFormulaType();
    String calculationRule = item.getCalculationRule();
    String dataSource = item.getDataSource();

    try {
        switch (formulaType) {
            case "DIRECT":
                return calculateDirectAmount(groupid, period, item, dataSource);
            case "FORMULA":
                return calculateFormulaAmount(calculationRule, amounts,
item.getFormulaContent());
            case "CUSTOM":
                return calculateCustomAmount(groupid, period, calculationRule, amounts,
allItems);
            case "CALCULATED":
                return calculateAutoAmount(item, amounts, allItems);
            default:
                return BigDecimal.ZERO;
        }
    } catch (Exception e) {
```

```
            System.err.println("计算项目失败: " + item.getItemCode() + " - " + e.getMessage());
            return BigDecimal.ZERO;
        }
    }
```

4.2 ???????

功能说明　　：

- 使用　AviatorEvaluator解析和执行复杂公式
- 支持科目代码引用、数学运算、函数调用等
- 自动处理科目代码到变量名的转换

核心代码　　：

```
private BigDecimal evaluateFormula(String formula, Map<String, BigDecimal> amounts) {
    try {
        // 预处理公式
        String processedFormula = processAccountFormula(formula);

        // 准备环境变量
        Map<String, Object> env = prepareEnvironment(amounts);

        // 执行计算
        AviatorEvaluator.setOption(Options.ALWAYS_USE_DOUBLE_AS_DECIMAL, false);
        Object result = AviatorEvaluator.execute(processedFormula, env);

        return convertToBigDecimal(result);
    } catch (Exception e) {
        throw new RuntimeException("公式计算错误: " + formula, e);
    }
}

private String processAccountFormula(String formula) {
    // 清理空格
    String cleaned = formula.replaceAll("\\s+", "");

    // 使用正则匹配科目代码（4位数字），并添加前缀
    Pattern pattern = Pattern.compile("\\d{4,}"); // 匹配4位及以上数字
    Matcher matcher = pattern.matcher(cleaned);
```

```
    StringBuffer sb = new StringBuffer();

    while (matcher.find()) {
        String accountCode = matcher.group();
        matcher.appendReplacement(sb, "ACC_" + accountCode);
    }
    matcher.appendTail(sb);

    return sb.toString();
}
```

5. ????

5.1 ??????

接口地址 ： `/api/report/generate`

请求方法 ： POST

请求参数 ：

```
{
  "templateCode": "BALANCE_SHEET_STANDARD",
  "period": "202312",
  "comparePeriod": "202311",
  "groupid": "123456",
  "amountUnit": 1,
  "includeComparison": true,
  "includeChartData": true
}
```

响应参数 ：

```
{
  "success": true,
  "message": "报表生成成功",
  "templateCode": "BALANCE_SHEET_STANDARD",
  "templateName": "标准资产负债表",
  "period": "202312",
```

```
  "reportDate": "2023-12-31",
  "items": [
    {
      "itemCode": "ASSET_TOTAL",
      "itemName": "资产总计",
      "itemLevel": 1,
      "lineNumber": 1,
      "amount": 1000000.00,
      "comparisonAmount": 950000.00,
      "changeAmount": 50000.00,
      "changeRate": 5.26,
      "isLeaf": false
    }
  ],
  "financialRatios": {},
  "chartData": {},
  "summary": {}
}
```


5.2 ????????

接口地址 ： `/report/items/list`

请求方法 ： GET

请求参数 ：

- `templateId`: 模板 ID

响应参数 ：


```
{
  "data": [
    {
      "itemId": 1,
      "templateId": 1,
      "itemCode": "ASSET_TOTAL",
      "itemName": "资产总计",
      "parentCode": "",
      "itemLevel": 1,
```

```
      "lineNumber": 1,
      "formulaType": "CALCULATED",
      "calculationRule": "SUM(CHILDREN)",
      "isShow": true,
      "isLeaf": false
    }
  ]
}
```

6. ???

6.1 ????

- Vue 3
- Element Plus
- ECharts (图表展示)
- Axios（ HTTP请求)

6.2 ????

- Spring Boot
- MyBatis-Plus
- AviatorEvaluator (公式计算)
- Redis (可选, 用于缓存)

7. ????

7.1 ??????

1. 用户在前端选择报表模板和期间
2. 前端调用报表生成接口（ `/api/report/generate` ）
3. 后端获取报表模板和项目结构
4. 后端计算当前期间和比较期间的项目金额
5. 后端生成财务指标和图表数据
6. 后端将结果返回给前端
7. 前端展示报表数据和图表

7.2 ????????

1. 用户在模板管理页面选择一个模板
2. 进入模板项目管理页面
3. 点击 "新增项目 "按钮，填写项目信息
4. 配置项目的计算公式和数据来源
5. 保存项目配置
6. 可以对项目进行验证，确保计算公式正确

8. ??

财务报表系统是一个功能完整、灵活可扩展的财务报表解决方案。该系统通过前端提供友好的用户界面，后端提供强大的计算能力，能够满足企业对财务报表的各种需求。系统支持多种报表类型、灵活的项目配置和复杂的计算公式，是企业财务管理的重要工具。

??????????

1. ????????

1.1 ????????（`calculateReportAmounts`）

```
private Map<String, BigDecimal> calculateReportAmounts(String groupid, String period,
                                                       List<FinReportItems> reportItems,
                                                       FinReportTemplates template) {
    // 缓存机制（目前注释掉）
    String cacheKey = groupid + "_" + template.getTemplateCode() + "_" + period;

    // 1. 获取所有科目余额作为基础数据
    Map<String, BigDecimal> amounts = subjectBalanceService.getAllSubjectBalance(groupid,
period);

    // 2. 关键排序：按层级从高到低排序，确保父级项目在子级之后计算
    List<FinReportItems> sortedItems = reportItems.stream()
            .sorted(Comparator.comparing(FinReportItems::getItemLevel).reversed())
            .collect(Collectors.toList());

    // 3. 逐个计算项目金额
    for (FinReportItems item : sortedItems) {
        if (!item.getIsShow()) continue;
```

```
        BigDecimal amount = calculateItemAmount(groupid, period, item, amounts, reportItems);
        amounts.put(item.getItemCode(), amount);
    }

    // 4. 结果缓存
    calculationCache.put(cacheKey, amounts);
    return amounts;
}
```

算法特点　　：

- 层级依赖处理　　：按项目层级倒序计算，确保父级项目计算时子级项目已完成计算
- 缓存机制　　：避免重复计算相同条件的报表
- 增量计算　　：基于基础科目余额数据进行增量计算

1.2 ???????? (`calculateItemAmount`)

```
private BigDecimal calculateItemAmount(String groupid, String period, FinReportItems item,
                                       Map<String, BigDecimal> amounts, List<FinReportItems>
allItems) {
    String formulaType = item.getFormulaType();
    String calculationRule = item.getCalculationRule();
    String dataSource = item.getDataSource();

    try {
        switch (formulaType) {
            case "DIRECT":    // 直接取值
                return calculateDirectAmount(groupid, period, item, dataSource);
            case "FORMULA":   // 公式计算
                return calculateFormulaAmount(calculationRule, amounts,
item.getFormulaContent());
            case "CUSTOM":    // 自定义规则
                return calculateCustomAmount(groupid, period, calculationRule, amounts,
allItems);
            case "CALCULATED":// 自动计算
                return calculateAutoAmount(item, amounts, allItems);
            default:
                return BigDecimal.ZERO;
        }
```

```
    } catch (Exception e) {
        System.err.println("计算项目失败: " + item.getItemCode() + " - " + e.getMessage());
        return BigDecimal.ZERO;
    }
}
```

算法特点：

- 策略模式：根据公式类型采用不同计算策略
- 容错机制：单个项目计算失败不影响整体报表生成
- 灵活扩展：支持新增公式类型和计算策略

2. ?????????

2.1 ???? (`processAccountFormula`)

```
private String processAccountFormula(String formula) {
    // 1. 清理空格
    String cleaned = formula.replaceAll("\\s+", "");

    // 2. 正则匹配科目代码（4位及以上数字）并添加前缀
    Pattern pattern = Pattern.compile("\\d{4,}");
    Matcher matcher = pattern.matcher(cleaned);
    StringBuffer sb = new StringBuffer();

    while (matcher.find()) {
        String accountCode = matcher.group();
        matcher.appendReplacement(sb, "ACC_" + accountCode);
    }
    matcher.appendTail(sb);

    return sb.toString();
}
```

算法特点：

- 标准化处理：统一公式格式，移除空格干扰
- 科目代码识别：使用正则表达式精准匹配会计科目代码
- 变量转换：将科目代码转换为Aviator引擎可识别的变量名（如：1001 → ACC_1001）

2.2 ???? (`evaluateFormula`)

```
private BigDecimal evaluateFormula(String formula, Map<String, BigDecimal> amounts) {
    try {
        // 1. 公式预处理
        String processedFormula = processAccountFormula(formula);

        // 2. 准备计算环境
        Map<String, Object> env = prepareEnvironment(amounts);

        // 3. 执行计算（禁用double作为小数类型，确保BigDecimal精度）
        AviatorEvaluator.setOption(Options.ALWAYS_USE_DOUBLE_AS_DECIMAL, false);
        Object result = AviatorEvaluator.execute(processedFormula, env);

        // 4. 结果转换
        return convertToBigDecimal(result);
    } catch (Exception e) {
        throw new RuntimeException("公式计算错误: " + formula, e);
    }
}
```

算法特点　　：

- 精度控制　　：使用　BigDecimal确保财务计算精度
- 表达式引擎　　：集成　Aviator引擎支持复杂数学运算
- 异常封装　　：提供清晰的错误信息便于调试

2.3 ???? (`convertToBigDecimal`)

```
private BigDecimal convertToBigDecimal(Object value) {
    if (value instanceof BigDecimal) {
        return (BigDecimal) value;
    } else if (value instanceof Number) {
        return new BigDecimal(value.toString());
    } else {
        throw new RuntimeException("无法转换的类型: " + value.getClass());
    }
}
```

算法特点　　：

- 类型安全 ：支持多种数值类型转换为 BigDecimal
- 异常明确 ：对不支持的类型给出明确错误信息

3. ?????????

3.1 ?????? (`calculateDirectAmount`)

```
private BigDecimal calculateDirectAmount(String groupid, String period, FinReportItems item,
String dataSource) {
    switch (dataSource) {
        case "SUBJECT":    // 科目余额
            if (item.getSubjectCodes() != null && !item.getSubjectCodes().isEmpty()) {
                return getSubjectBalance(groupid, period, item.getSubjectCodes(),
item.getAmountType());
            }
            break;

        case "CONSTANT":   // 常量值
            if (item.getCalculationRule() != null) {
                try {
                    return new BigDecimal(item.getCalculationRule());
                } catch (NumberFormatException e) {
                    return BigDecimal.ZERO;
                }
            }
            break;

        case "CUSTOM":     // 自定义数据源
            return getCustomAmount(groupid, period, item.getCalculationRule());
    }
    return BigDecimal.ZERO;
}
```

算法特点 ：

- 多数据源支持 ：支持科目余额、常量、自定义三种数据源
- 容错处理 ：常量值解析失败返回 0而非抛出异常

3.2 ????? (`sumChildrenAmounts`)

```
private BigDecimal sumChildrenAmounts(String parentCode, List<FinReportItems> allItems,
Map<String, BigDecimal> amounts) {
    return allItems.stream()
            .filter(item -> parentCode.equals(item.getParentCode()) && item.getIsShow())
            .map(item -> amounts.getOrDefault(item.getItemCode(), BigDecimal.ZERO))
            .reduce(BigDecimal.ZERO, BigDecimal::add);
}
```

算法特点：

- 流式计算：使用 Java 8 Stream API实现高效汇总
- 条件过滤：仅汇总显示的子项目
- 默认值处理：未计算的项目使用 0作为默认值

4. ?????????

4.1 ??????

问题：父级项目通常依赖子级项目的计算结果

解决方案：按项目层级从高到低排序计算，确保父级项目在所有子级项目计算完成后执行

4.2 ??????

问题：财务计算对精度要求极高，浮点数计算易产生误差

解决方案：

- 统一使用 BigDecimal进行所有计算
- 禁用 Aviator引擎的 double类型转换
- 所有中间结果和最终结果保持 BigDecimal类型

4.3 ????

问题：相同条件下重复生成报表会造成性能浪费

解决方案：实现基于 groupid、templateCode和 period 的缓存机制，避免重复计算（目前代码中已实现但注释掉）

4.4 ????

[illegible] [illegible]
[illegible] [illegible]

- [illegible]
- [illegible]
- [illegible] 0[illegible]

5. ???????

5.1 ?????????

```
1. [illegible]1001[illegible]1002[illegible]
2. [illegible]
   - [illegible]"[illegible]"[illegible]1001+1002[illegible]
   - [illegible]"[illegible]"[illegible]+[illegible]+[illegible]+...[illegible]
   - [illegible]"[illegible]"[illegible]+[illegible]
3. [illegible]
4. [illegible]"[illegible]=[illegible]+[illegible]"
```

5.2 ???????

```
1. [illegible]
2. [illegible]-[illegible]-[illegible]-...[illegible]
3. [illegible]+[illegible]-[illegible]
4. [illegible]-[illegible]
```

6. ??????

6.1 ??????

[illegible]

```
// [illegible]
if (calculationCache.containsKey(cacheKey)) {
    return calculationCache.get(cacheKey);
}
```

6.2 ?????

对于频繁使用的公式，可以实现预编译机制提高性能：

```
// 在模板加载时预编译公式
Map<String, Expression> compiledFormulas = new HashMap<>();

// 计算时直接使用预编译表达式
Expression expr = compiledFormulas.computeIfAbsent(formula, AviatorEvaluator::compile);
Object result = expr.execute(env);
```

6.3 ??????

当前实现中单个项目异常会打印日志，建议实现批量异常收集功能，生成报表时统一展示错误信息：

```
// 收集计算错误
List<String> calculationErrors = new ArrayList<>();

// 计算失败时记录错误
calculationErrors.add("项目 " + item.getItemCode() + " 计算失败: " + e.getMessage());

// 在报表响应中返回错误信息
response.setCalculationErrors(calculationErrors);
```

通过以上解析，可以看到报表模板算法设计注重了财务计算的精度、性能和可扩展性，采用了分层计算、策略模式、流式处理等现代 Java编程技术，确保了报表生成的准确性和高效性。

版本号 #2
由 Admin 创建于 2026-01-21 18:19:07 CST
由 Admin 更新于 2026-01-21 18:30:41 CST