八、实验三：实时指标计算与存储（主程序实现 + 模块化架构设计）

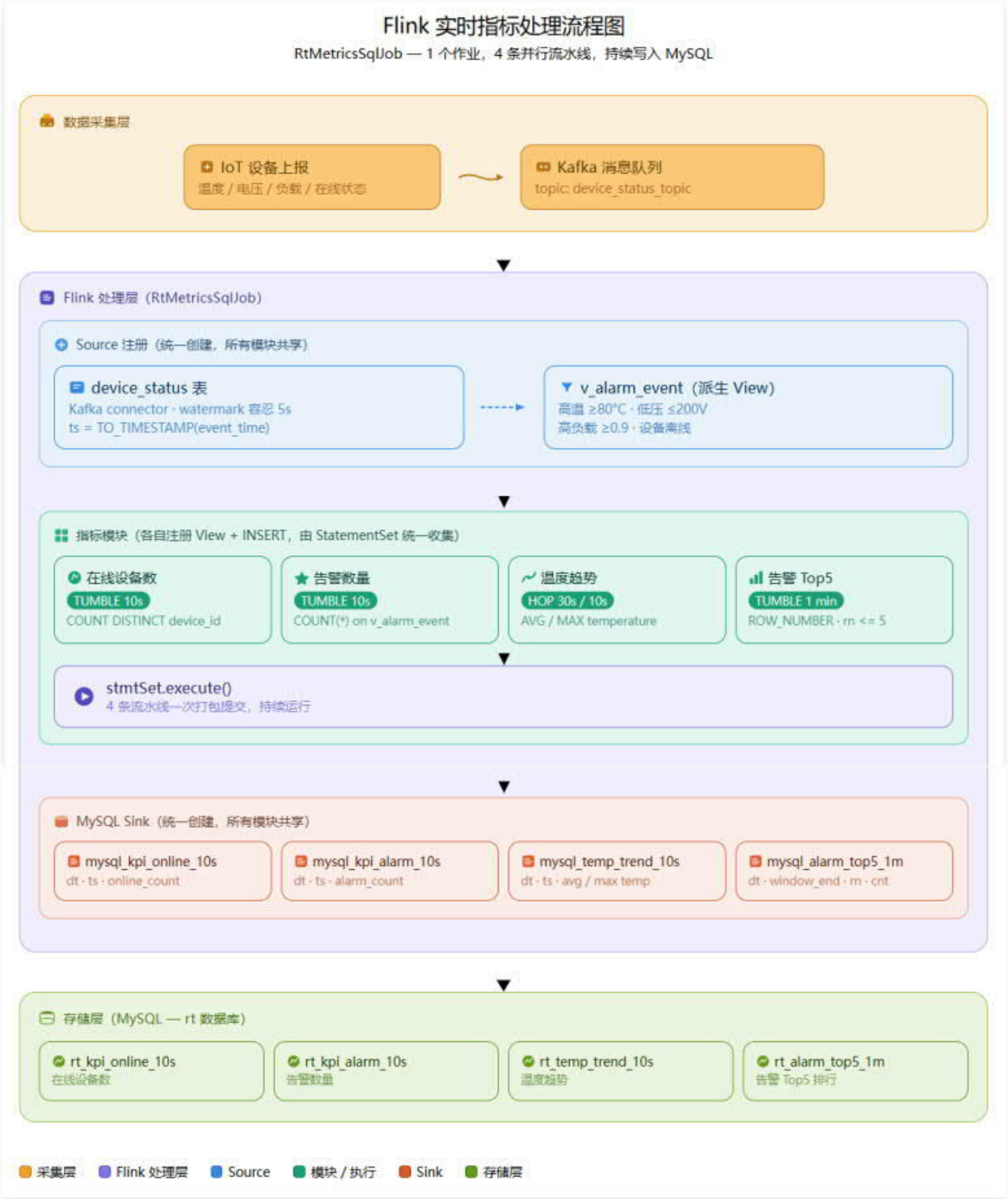
8.1 实验背景（应用场景）

在物联网和生产系统中，需要对数据进行**实时计算与监控**（如设备状态、温度、告警等）。

本实验基于 **Flink** 构建实时指标计算系统，实现数据的**实时处理与存储**。

8.2 实验目的

1. 掌握 Flink 实时流处理流程
2. 理解模块化架构设计思想
3. 实现多指标计算与统一提交
4. 学会在集群环境运行实时任务
5. 提升代码编写与调试能力



8.3 代码设计

> 本方案基于前面"单指标 Job 独立运行"的实现方式进行升级，
> 通过统一入口程序 + 指标模块注册机制，实现多指标统一计算与统一提交，
> 更贴近企业真实生产环境的实时计算架构。

一、模块命名规范

八、实验三：实时指标计算与存储（主程序实现 + 模块化架构设计）_02.jpg

1）推荐命名

- **主程序入口：**
  - `RtMetricsSqlJob`：统一入口，一键启动所有指标
- **指标模块：** `Module`（不可直接运行，只负责 SQL 注册）
  - `OnlineKpi10sModule`
  - `TempTrendHopModule`
  - `AlarmKpi10sModule`
  - `AlarmTop5_1mModule`

📌 解释：

> **Job = 能跑的主程序入口**
> **Module = 插件模块，只负责把 SQL 注册进去**

二、项目结构

```
src/main/java/com/demo/flink/
  ├─ RtMetricsSqlJob.java                # ✅ 主程序入口
  └─ module/
     ├─ OnlineKpi10sModule.java          # ✅ 在线KPI模块
     ├─ TempTrendHopModule.java          # ✅ 温度趋势模块
     ├─ AlarmKpi10sModule.java           # ✅ 告警KPI模块
     └─ AlarmTop5_1mModule.java          # ✅ Top5模块
```

> 说明：放到 `module/` 子包里，项目结构更清爽。

三、主程序职责

> 主程序 RtMetricsSqlJob 作为统一实时指标作业入口，
> 在同一个 Flink 作业中完成全部四个指标计算与写入。

主要完成以下 5 件事：

1）创建 Kafka Source 表（device_status / v_alarm_event，仅创建一次）
2）创建 MySQL Sink 表（4 张指标表，仅创建一次）
3）创建 StatementSet，用于统一提交多个 INSERT
4）调用各指标模块的 register() 方法
5）统一 execute()，实现多指标并行计算

八、实验三：实时指标计算与存储（主程序实现 + 模块化架构设计）_03.jpg

8.3.1 在线设备数模块（OnlineKpi10sModule）

> 路径：`src/main/java/com/demo/flink/module/OnlineKpi10sModule.java`

```
package com.demo.flink.module;

import org.apache.flink.table.api.StatementSet;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

/**
 * 模块：在线设备数 KPI（10秒）
 * 作用：
 * ① 计算每10秒在线设备数量
 * ② 将结果写入 MySQL 表
 *
 * 注意：
 * - 这里只负责“计算逻辑 + 写入逻辑”
 * - Kafka源表、MySQL表由主程序统一创建
 */
public class OnlineKpi10sModule {

    public static void register(StreamTableEnvironment tEnv, StatementSet stmtSet) {

        // ====================================================
        // 1.创建临时视图：计算在线设备数（10秒窗口）
        // ====================================================
        // 说明：
        // - 使用 TUMBLE 滚动窗口（每10秒统计一次）
        // - 只统计 status = 1（在线设备）
        // - COUNT(DISTINCT device_id)：去重统计设备数
        String viewSql =
                "CREATE TEMPORARY VIEW v_kpi_online_10s AS \n" +
                        "SELECT\n" +
                        "  CAST(window_end AS DATE) AS dt,            -- 日期（用于分区）\n" +
                        "  window_end AS ts,                          -- 窗口结束时间\n" +
                        "  COUNT(DISTINCT device_id) AS online_count  -- 在线设备数\n" +
                        "FROM TABLE(\n" +
                        "  TUMBLE(TABLE device_status, DESCRIPTOR(ts), INTERVAL '10'
SECOND)\n" +
                        ")\n" +
                        "WHERE status = 1\n" +
                        "GROUP BY window_start, window_end";

        // 执行SQL，生成临时视图（只在当前会话有效）
        tEnv.executeSql(viewSql);

        // ====================================================
        // 2.写入 MySQL 表
        // ====================================================
        // 说明：
        // - 将计算结果插入 mysql_kpi_online_10s 表
        // - StatementSet 可以一次提交多个任务（推荐做法）
        stmtSet.addInsertSql(
```

```
                "INSERT INTO mysql_kpi_online_10s " +
                        "SELECT dt, ts, online_count FROM v_kpi_online_10s"
        );
    }
}
```

8.3.2 温度趋势模块（TempTrendHopModule）

> 路径：`src/main/java/com/demo/flink/module/TempTrendHopModule.java`

```
package com.demo.flink.module;

import org.apache.flink.table.api.StatementSet;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

/**
 * 模块：温度趋势（30秒窗口，每10秒滑动）
 * 作用：
 * ① 计算温度平均值和最大值
 * ② 形成连续的温度变化趋势
 *
 * 注意：
 * - 使用滑动窗口（HOP）
 * - Source / Sink 由主程序统一创建
 */
public class TempTrendHopModule {

    public static void register(StreamTableEnvironment tEnv, StatementSet stmtSet) {

        // ====================================================
        // 1.创建临时视图：温度趋势计算
        // ====================================================
        // 说明：
        // - 窗口大小：30秒
        // - 滑动步长：10秒（每10秒输出一次结果）
        // - AVG：平均温度
        // - MAX：最高温度
        String viewSql =
                "CREATE TEMPORARY VIEW v_temp_trend_hop AS \n" +
                        "SELECT\n" +
                        "  CAST(window_end AS DATE) AS dt,      -- 日期\n" +
                        "  window_end AS ts,                    -- 窗口结束时间\n" +
                        "  AVG(temperature) AS avg_temp,        -- 平均温度\n" +
                        "  MAX(temperature) AS max_temp         -- 最高温度\n" +
                        "FROM TABLE(\n" +
                        "  HOP(TABLE device_status, DESCRIPTOR(ts),\n" +
                        "      INTERVAL '10' SECOND, INTERVAL '30' SECOND)\n" +
                        ")\n" +
                        "GROUP BY window_start, window_end";

        // 执行SQL，生成临时视图
        tEnv.executeSql(viewSql);
```

```
        // ====================================================
        // 2.写入 MySQL 表
        // ====================================================
        // 说明：
        // - 每10秒写入一次温度趋势数据
        // - 用于前端折线图展示
        stmtSet.addInsertSql(
                "INSERT INTO mysql_temp_trend_10s " +
                        "SELECT dt, ts, avg_temp, max_temp FROM v_temp_trend_hop"
        );
    }
}
```

8.3.3 告警数模块（AlarmKpi10sModule）

> 路径：`src/main/java/com/demo/flink/module/AlarmKpi10sModule.java`

```
package com.demo.flink.module;

import org.apache.flink.table.api.StatementSet;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

/**
 * 模块：告警数量 KPI（10秒）
 * 作用：
 * ① 统计每10秒内的告警数量
 * ② 反映系统异常情况
 *
 * 注意：
 * - 数据来源是 alarm_event（已过滤出的异常数据）
 * - Source / Sink 由主程序统一创建
 */
public class AlarmKpi10sModule {

    public static void register(StreamTableEnvironment tEnv, StatementSet stmtSet) {

        // ====================================================
        // 1.创建临时视图：告警数量统计
        // ====================================================
        // 说明：
        // - 使用 TUMBLE 滚动窗口（每10秒统计一次）
        // - COUNT(*)：统计告警总数
        String viewSql =
                "CREATE TEMPORARY VIEW v_kpi_alarm_10s AS \n" +
                        "SELECT\n" +
                        "  CAST(window_end AS DATE) AS dt,    -- 日期\n" +
                        "  window_end AS ts,                  -- 窗口结束时间\n" +
                        "  COUNT(*) AS alarm_count            -- 告警数量\n" +
                        "FROM TABLE(\n" +
```

```
                        "  TUMBLE(TABLE v_alarm_event, DESCRIPTOR(ts), INTERVAL '10'
SECOND)\n" +
                        ")\n" +
                        "GROUP BY window_start, window_end";

        // 执行SQL，生成临时视图
        tEnv.executeSql(viewSql);


        // ====================================================
        // 2.写入 MySQL 表
        // ====================================================
        // 说明：
        // - 每10秒输出一次告警统计结果
        // - 可用于大屏告警指标展示
        stmtSet.addInsertSql(
                "INSERT INTO mysql_kpi_alarm_10s " +
                        "SELECT dt, ts, alarm_count FROM v_kpi_alarm_10s"
        );
    }
}
```

8.3.4 告警Top5模块（AlarmTop5_1mModule）

> 路径：`src/main/java/com/demo/flink/module/AlarmTop5_1mModule.java`

```
package com.demo.flink.module;

import org.apache.flink.table.api.StatementSet;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

/**
 * 模块：告警 Top5（1分钟窗口）
 * 作用：
 * ① 统计每台设备1分钟内的告警次数
 * ② 找出每个时间窗口内告警次数最多的前5台设备
 *
 * 注意：
 * - 本模块分两步完成：先统计，再排名
 * - Source / Sink 由主程序统一创建
 */
public class AlarmTop5_1mModule {

    public static void register(StreamTableEnvironment tEnv, StatementSet stmtSet) {

        // ====================================================
        // 1.创建临时视图：统计每台设备每1分钟内的告警次数
        // ====================================================
        // 说明：
        // - 使用 TUMBLE 滚动窗口，窗口大小为1分钟
        // - 按 device_id 分组
        // - COUNT(*) 表示该设备在这个窗口内触发了多少次告警
```

```
        String cntView =
                "CREATE TEMPORARY VIEW v_alarm_cnt_1m AS \n" +
                        "SELECT\n" +
                        "  window_end,                       -- 窗口结束时间\n" +
                        "  device_id,                        -- 设备编号\n" +
                        "  COUNT(*) AS cnt                   -- 告警次数\n" +
                        "FROM TABLE(\n" +
                        "  TUMBLE(TABLE v_alarm_event, DESCRIPTOR(ts), INTERVAL '1'
MINUTE)\n" +
                        ")\n" +
                        "GROUP BY window_start, window_end, device_id";

        // 执行SQL，生成“每台设备每分钟告警次数”视图
        tEnv.executeSql(cntView);


        // ====================================================
        // 2.创建临时视图：对每个窗口内的设备进行排名，取前5名
        // ====================================================
        // 说明：
        // - ROW_NUMBER()：给每个窗口内的设备按告警次数排序编号
        // - PARTITION BY window_end：每个窗口单独排名
        // - ORDER BY cnt DESC：按告警次数从大到小排序
        // - WHERE rn <= 5：只保留前5名
        String top5View =
                "CREATE TEMPORARY VIEW v_alarm_top5_1m AS \n" +
                        "SELECT\n" +
                        "  CAST(window_end AS DATE) AS dt,    -- 日期\n" +
                        "  window_end,                       -- 窗口结束时间\n" +
                        "  device_id,                        -- 设备编号\n" +
                        "  cnt,                              -- 告警次数\n" +
                        "  rn                                -- 排名\n" +
                        "FROM (\n" +
                        "  SELECT\n" +
                        "    window_end,\n" +
                        "    device_id,\n" +
                        "    cnt,\n" +
                        "    ROW_NUMBER() OVER (PARTITION BY window_end ORDER BY cnt DESC)
AS rn\n" +
                        "  FROM v_alarm_cnt_1m\n" +
                        ")\n" +
                        "WHERE rn <= 5";

        // 执行SQL，生成 Top5 结果视图
        tEnv.executeSql(top5View);

        // ====================================================
        // 3.写入 MySQL 表
        // ====================================================
        // 说明：
        // - 将每个窗口内告警次数前5的设备写入 MySQL
        // - 可用于大屏 Top5 排行榜展示
        stmtSet.addInsertSql(
```

```
                "INSERT INTO mysql_alarm_top5_1m " +
                        "SELECT dt, window_end, device_id, cnt, rn FROM v_alarm_top5_1m"
        );
    }
}
```

8.3.5 主程序 RtMetricsSqlJob 实现（统一入口，一键启动）

> 路径：`src/main/java/com/demo/flink/RtMetricsSqlJob.java`

```
package com.demo.flink;

import com.demo.flink.module.AlarmKpi10sModule;
import com.demo.flink.module.AlarmTop5_1mModule;
import com.demo.flink.module.OnlineKpi10sModule;
import com.demo.flink.module.TempTrendHopModule;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.table.api.EnvironmentSettings;
import org.apache.flink.table.api.StatementSet;
import org.apache.flink.table.api.bridge.java.StreamTableEnvironment;

/**
 * Job：实时指标统一入口（模块化组合）
 *
 * 设计目标：只启动 1 个 Flink 作业，同时计算 4 个实时指标，并持续写入 MySQL
 *
 * 核心思路：
 * 1）主程序统一创建 Source（Kafka 表）与 Sink（MySQL 表）
 * 2）各指标模块只负责注册：View + Insert SQL（不再重复建 Source/Sink）
 * 3）最后由 StatementSet.execute() 一次性提交所有 Insert（只 execute 一次）
 */
public class RtMetricsSqlJob {

    public static void main(String[] args) throws Exception {

        // 设置日志级别为 "ERROR"，减少不必要的日志输出
        System.setProperty("org.slf4j.simpleLogger.defaultLogLevel", "error");

        // ============================================================
        // 1.Flink 执行环境（相当于“运行引擎”）
        //    - 并行度设为 1：教学演示更容易观察（日志更少、数据更直观）
        //    - 开启 checkpoint：流作业容错能力（断点恢复、保证一致性）
        // ============================================================
        StreamExecutionEnvironment env =
StreamExecutionEnvironment.getExecutionEnvironment();
        env.setParallelism(1);
        env.enableCheckpointing(30000); // 30s 一次（别太频繁，避免压力过大）

        System.out.println("====== 1. Flink 执行环境初始化完成 ======");

        // ============================================================
        // 2.Table 环境（Flink SQL 的入口）
```

```
//    后续所有 executeSql(...) 都是向这个 SQL 环境注册表 / 视图 / 写入任务
// ============================================================
EnvironmentSettings settings = EnvironmentSettings.newInstance()
        .inStreamingMode()
        .build();
StreamTableEnvironment tEnv = StreamTableEnvironment.create(env, settings);

System.out.println("====== 2. Flink  Table 环境（Flink SQL 的入口）初始化完成 ======");

// ============================================================
// 3.统一创建 Kafka Source（只建一次，所有模块共享）
//    表名：device_status
//    - topic：device_status_topic
//    - ts：从 event_time 解析得到事件时间
//    - watermark：允许最大 5 秒乱序（教学常用配置）
// ============================================================
String createDeviceStatus =
        "CREATE TABLE device_status (\n" +
                "  event_type STRING,\n" +
                "  device_id STRING,\n" +
                "  status INT,\n" +
                "  temperature DOUBLE,\n" +
                "  `load` DOUBLE,\n" +
                "  voltage DOUBLE,\n" +
                "  event_time STRING,\n" +
                "  ts AS TO_TIMESTAMP(event_time),\n" +
                "  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND\n" +
                ") WITH (\n" +
                "  'connector' = 'kafka',\n" +
                "  'topic' = 'device_status_topic',\n" +
                "  'properties.bootstrap.servers' = 'master:9092',\n" +
                "  'properties.group.id' = 'rt_metrics_all_g1',\n" +
                "  'scan.startup.mode' = 'latest-offset',\n" +
                "  'format' = 'json',\n" +
                "  'json.ignore-parse-errors' = 'true'\n" +
                ")";
tEnv.executeSql(createDeviceStatus);

System.out.println("====== 3. Kafka Source 初始化完成 ======");

// ============================================================
// 4.统一生成“告警事件流”（从状态流推导）
//     目的：不依赖 alarm_event_topic，只用 device_status 就能产生告警事件
//     输出字段：
//       - device_id：设备
//       - alarm_type：告警类型（根据规则判定）
//       - ts：事件时间（沿用 device_status 的 ts）
// ============================================================
String createAlarmEventView =
        "CREATE TEMPORARY VIEW v_alarm_event AS \n" +
                "SELECT\n" +
                "  device_id,\n" +
                "  CASE\n" +
```

```
                        "    WHEN temperature >= 80 THEN 'HIGH_TEMP'\n" +
                        "    WHEN voltage <= 200 THEN 'LOW_VOLT'\n" +
                        "    WHEN `load` >= 0.90 THEN 'HIGH_LOAD'\n" +
                        "    WHEN status = 0 THEN 'DEVICE_OFFLINE'\n" +
                        "  END AS alarm_type,\n" +
                        "  ts\n" +
                        "FROM device_status\n" +
                        "WHERE temperature >= 80 OR voltage <= 200 OR `load` >= 0.90 OR
status = 0";
        tEnv.executeSql(createAlarmEventView);

        System.out.println("====== 4.“告警事件流”视图生成完成 ======");

        // ============================================================
        // 5.统一创建 MySQL Sink（只建一次，所有模块共享）
        //    说明：
        //    - Flink JDBC Sink 若产生更新流（update/delete），必须声明 PRIMARY KEY
        // ============================================================

        // 5.1 在线设备数指标（10秒滚动窗口结果写入 MySQL）
        String mysqlOnline =
                "CREATE TABLE mysql_kpi_online_10s (\n" +
                        "  dt DATE,\n" +
                        "  ts TIMESTAMP(3),\n" +
                        "  online_count BIGINT\n" +
                        ") WITH (\n" +
                        "  'connector'='jdbc',\n" +
                        "  'url'='jdbc:mysql://master:3306/rt?
useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai',\n" +
                        "  'table-name'='rt_kpi_online_10s',\n" +
                        "  'username'='root',\n" +
                        "  'password'='123456',\n" +
                        "  'driver'='com.mysql.cj.jdbc.Driver',\n" +
                        "  'sink.buffer-flush.max-rows'='100',\n" +
                        "  'sink.buffer-flush.interval'='2s'\n" +
                        ")";
        tEnv.executeSql(mysqlOnline);

        // 5.2 告警数量指标表（10秒滚动窗口结果写入 MySQL）
        String mysqlAlarm =
                "CREATE TABLE mysql_kpi_alarm_10s (\n" +
                        "  dt DATE,\n" +
                        "  ts TIMESTAMP(3),\n" +
                        "  alarm_count BIGINT\n" +
                        ") WITH (\n" +
                        "  'connector'='jdbc',\n" +
                        "  'url'='jdbc:mysql://master:3306/rt?
useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai',\n" +
                        "  'table-name'='rt_kpi_alarm_10s',\n" +
                        "  'username'='root',\n" +
                        "  'password'='123456',\n" +
                        "  'driver'='com.mysql.cj.jdbc.Driver',\n" +
                        "  'sink.buffer-flush.max-rows'='100',\n" +
```

八、实验三：实时指标计算与存储（主程序实现 + 模块化架构设计）_11.jpg

```
                        "  'sink.buffer-flush.interval'='2s'\n" +
                        ")";
        tEnv.executeSql(mysqlAlarm);

        // 5.3 温度趋势指标表（30秒窗口、每10秒输出一次结果，写入 MySQL）
        String mysqlTrend =
                "CREATE TABLE mysql_temp_trend_10s (\n" +
                        "  dt DATE,\n" +
                        "  ts TIMESTAMP(3),\n" +
                        "  avg_temp DOUBLE,\n" +
                        "  max_temp DOUBLE\n" +
                        ") WITH (\n" +
                        "  'connector'='jdbc',\n" +
                        "  'url'='jdbc:mysql://master:3306/rt?
useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai',\n" +
                        "  'table-name'='rt_temp_trend_10s',\n" +
                        "  'username'='root',\n" +
                        "  'password'='123456',\n" +
                        "  'driver'='com.mysql.cj.jdbc.Driver',\n" +
                        "  'sink.buffer-flush.max-rows'='100',\n" +
                        "  'sink.buffer-flush.interval'='2s'\n" +
                        ")";
        tEnv.executeSql(mysqlTrend);

        // 5.4 告警 Top5 排行表（1分钟窗口结果写入 MySQL）：Rank 会产生更新流 → 必须声明主键
        String mysqlTop5 =
                "CREATE TABLE mysql_alarm_top5_1m (\n" +
                        "  dt DATE,\n" +
                        "  window_end TIMESTAMP(3),\n" +
                        "  device_id STRING,\n" +
                        "  cnt BIGINT,\n" +
                        "  rn BIGINT,\n" +
                        "  PRIMARY KEY (dt, window_end, rn) NOT ENFORCED\n" +
                        ") WITH (\n" +
                        "  'connector'='jdbc',\n" +
                        "  'url'='jdbc:mysql://master:3306/rt?
useSSL=false&characterEncoding=utf8&serverTimezone=Asia/Shanghai',\n" +
                        "  'table-name'='rt_alarm_top5_1m',\n" +
                        "  'username'='root',\n" +
                        "  'password'='123456',\n" +
                        "  'driver'='com.mysql.cj.jdbc.Driver',\n" +
                        "  'sink.buffer-flush.max-rows'='100',\n" +
                        "  'sink.buffer-flush.interval'='2s'\n" +
                        ")";
        tEnv.executeSql(mysqlTop5);

        System.out.println("====== 5. MySQL Sink（四个表） 创建完成 ======");

        // ============================================================
        // 6. 定义StatementSet，作用：收集多个 INSERT 语句（最后统一提交）
        // ============================================================
        StatementSet stmtSet = tEnv.createStatementSet();
```

```
        System.out.println("====== 6. 定义StatementSet完成 ======");

        // ============================================================
        // 7.注册模块：模块只负责（View + Insert）
        //     - 每个模块负责：创建视图（View） + 注册写入任务（Insert）
        //     - 执行后：
        //         tEnv 中新增一个临时视图（逻辑表）
        //         stmtSet 中新增一条写入 MySQL 的任务
        // ============================================================
        OnlineKpi10sModule.register(tEnv, stmtSet);
        TempTrendHopModule.register(tEnv, stmtSet);
        AlarmKpi10sModule.register(tEnv, stmtSet);
        AlarmTop5_1mModule.register(tEnv, stmtSet);

        System.out.println("====== 7.1 在线设备数统计模块：已加入执行任务（等待执行）");
        System.out.println("====== 7.2 温度趋势统计模块：已加入执行任务（等待执行）");
        System.out.println("====== 7.3 告警数量统计模块：已加入执行任务（等待执行）");
        System.out.println("====== 7.4 告警Top5统计模块：已加入执行任务（等待执行）");

        // ============================================================
        // 8.统一执行：真正提交作业（IDEA 运行时，这一行开始“常驻运行”）
        // ============================================================
        System.out.println("====== 8. 实时数据分析作业开始持续执行.....");
        stmtSet.execute();
    }
}
```

总结：

- **以前**：每个 Job 自己建环境、自己 execute → 资源占用大、难统一管理
- **现在**：主程序统一建环境 + 模块只负责注册 → **更省资源、更好维护、更适合分工演示**

八、实验三：实时指标计算与存储（主程序实现 + 模块化架构设计）_14.jpg