阿里云 intel.

Beyond Stream Processing

2022第四届 实时计算FLINK挑战赛 天池大赛

基于Flink流计算实现的股票交易实时资产应用

The Applications of Realtime Stock Trading Based on Implementation of Flink Streaming

guanxing | 2022.12

阿里云 intel.

成员&背景介绍

成员

guanxing 港美股互联网券商-数据开发

数据开发工程师，目前在大数据部门负责业务数据开发、数据平台建设、数据资产建设等相关工作，对流计算应用开发有一定经验

背景

在见证Flink的发展的同时，不断跟进并将其用于平台优化

- 2020年，数据平台的架构为Canal+Kafka+Flink+Mysql，平台仅提供数据接入和查询能力
- 2021年，随着业务的发展和数据规模的膨胀，数据平台开始对数据仓库进行完善，引入了Hive、Hbase、Clickhouse等组件。此外，平台内部也开始推广FlinkSQL，在提高代码的可读性的同时也降低了开发门槛，由此数据平台的定位也不再仅仅局限于简单的数据接入和提供查询，而是开始逐渐涉及一些复杂业务的数据应用开发，由此，“数据中台“的独特优势开始显露
- 2022年，数据平台开始使用FlinkCDC取代Canal进行数据接入，使用Flink技术栈对ETL实现了统一。至此平台的架构演进成了FlinkCDC+Kafka+Flink+数仓，不仅提供数据的接入查询能力，还对外提供各层次的数据应用

CONTENT

目录>>

01 券商股票交易系统

02 传统ELT架构下遇到的难题

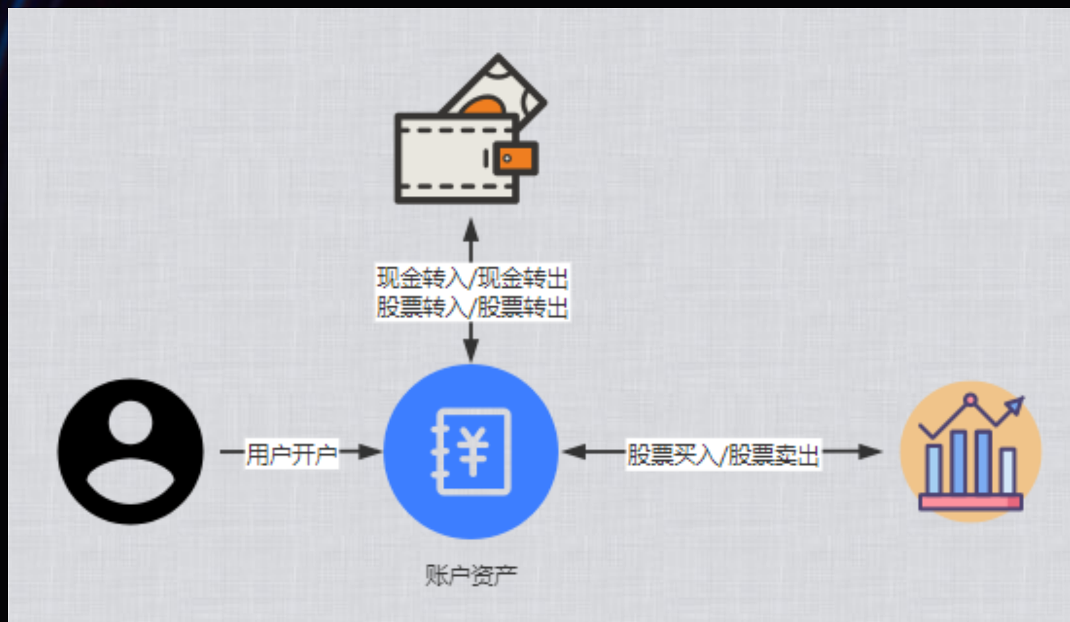
03 解决方案：流计算ETL架构的引入

04 流计算扩展应用：走势&收益

05 其他挑战&总结

01 券商股票交易系统

股票交易系统



基本流程&概念

- **开户**：在券商系统下建立一个专属账户
- **流水**：出入金 = 往账户中存入/取出现金
  出入货 = 往账户中存入/取出股票
- **交易**：买入股票 = 现金减少，股票增加
  卖出股票 = 现金增加，股票减少

阿里云 intel

账户总资产的计算

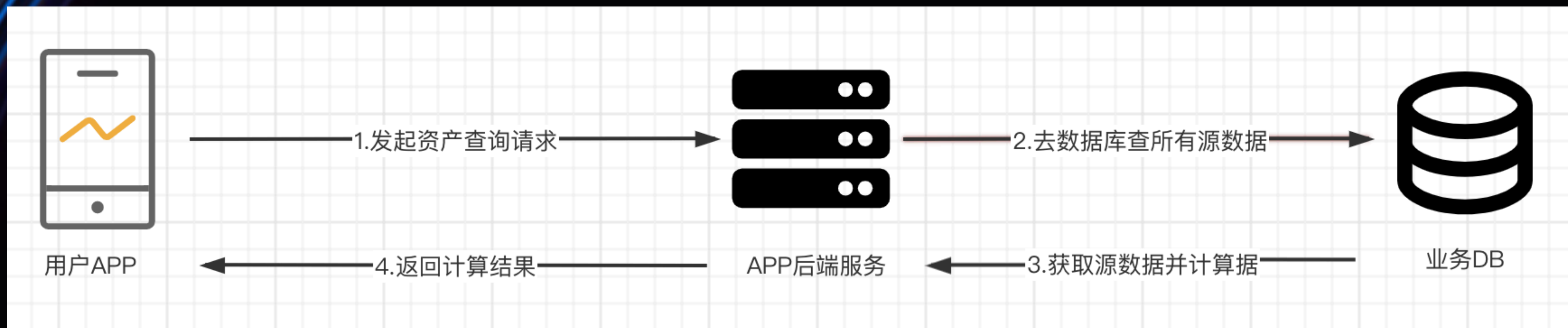
总资产计算公式

总资产=现金+股票市值

- 现金 = 账户现金
- 股票市值 = 所持仓股票 * 对应的最新报价（变化）

02 传统ELT架构下遇到的难题

传统ELT架构

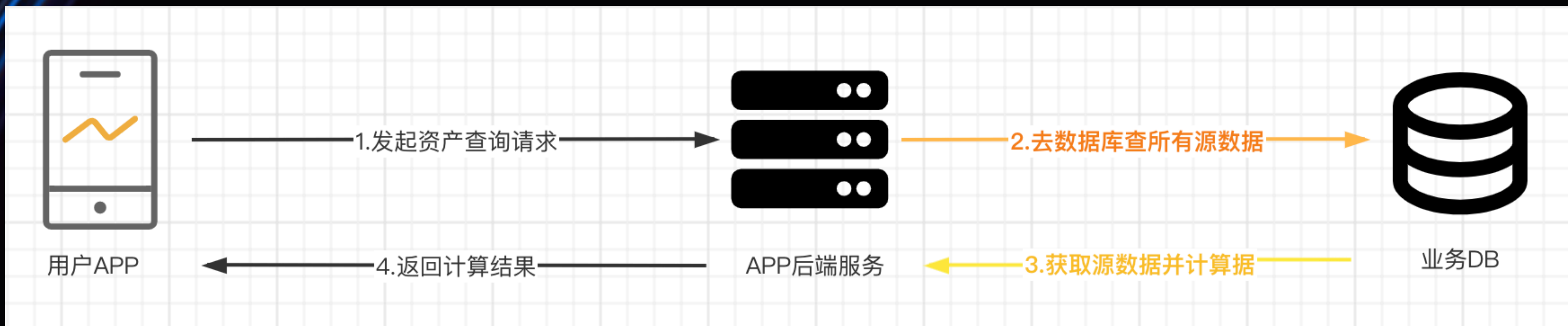


总资产查询ELT流程

1. 用户从客户端发起资产请求到后端
2. 后端进程去业务DB里查询用户现金表、用户持仓股票表以及最新股票报价表数据
3. 后端进程根据查询到的数据计算出用户持仓的市值，加上用户现金得到用户最新总资产
4. 将算出的总资产结果返回客户端展示

阿里云 intel

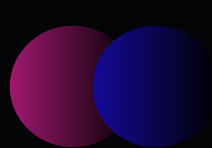
遇到的困难



资产查询ELT流程

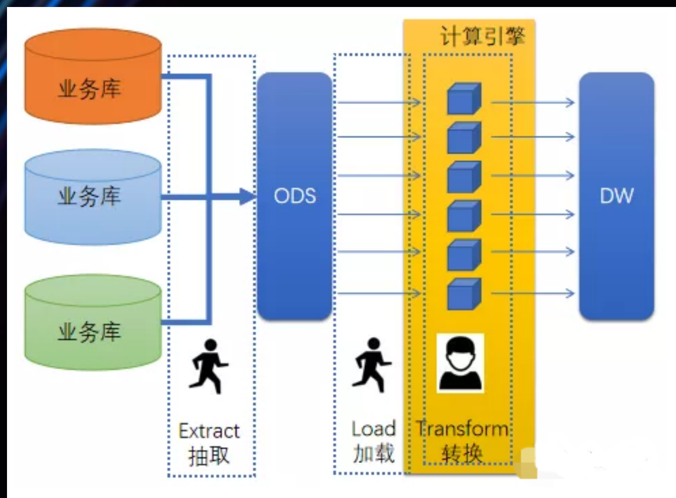
当请求数量变多时，服务端程序计算和数据库读取的压力会明显增大，容易达到瓶颈！

1. 用户从客户端发起资产请求到后端
2. 后端进程去业务DB里查询所有用户现金表、用户持仓股票表以及最新股票报价表数据
3. 后端进程根据查询到的数据计算出用户持仓的市值，加上用户现金得到出用户最新总资产
4. 将算出的总资产结果返回客户端展示



03 解决方案：流计算ETL架构的引入

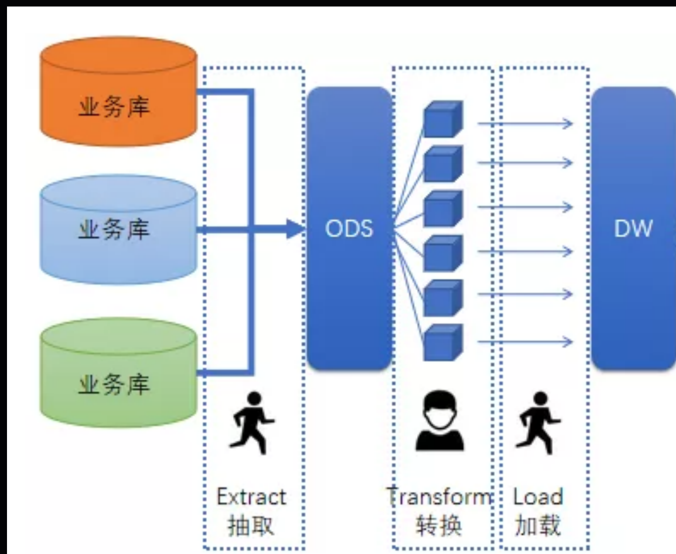
ETL架构



ELT与ETL

ELT 是Extract（抽取）、Load（加载）、**Transform（转换）**

ETL 是Extract（抽取）、**Transform（转换）**、Load（加载）



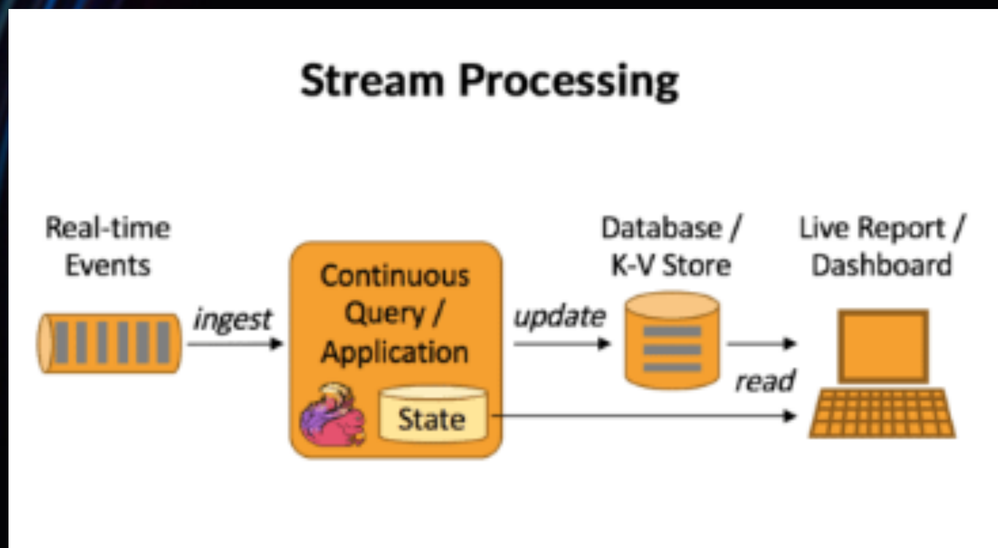
ETL模式的优缺点

对于既定的逻辑计算和大量结果查询的场景，适合把计算过程T前置！

优点：在固定的处理环节前提下，建好之后只需要维护，方便下游直接使用

缺点：流程长、不灵活，逻辑修改成本高

流计算的方案

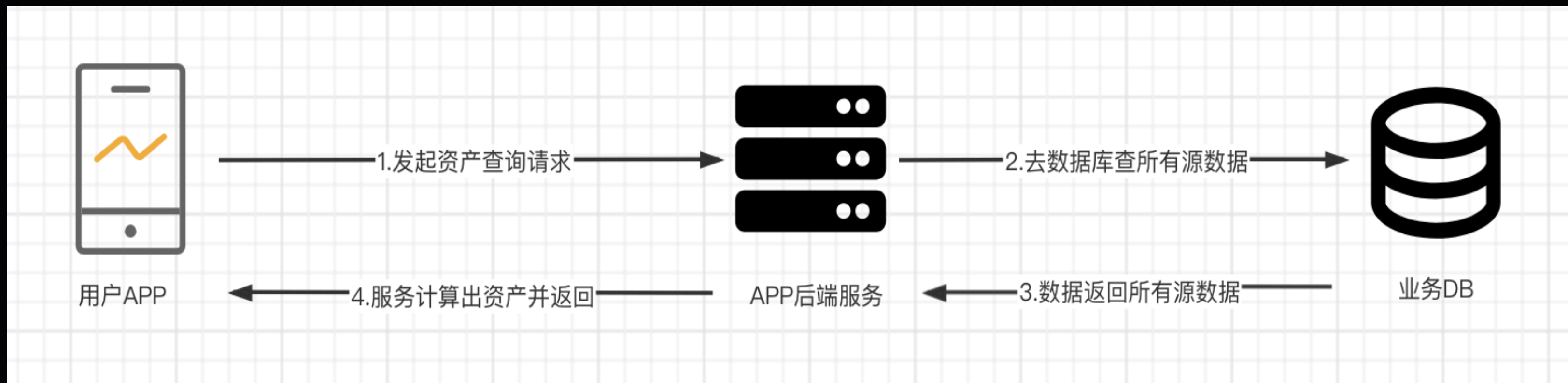


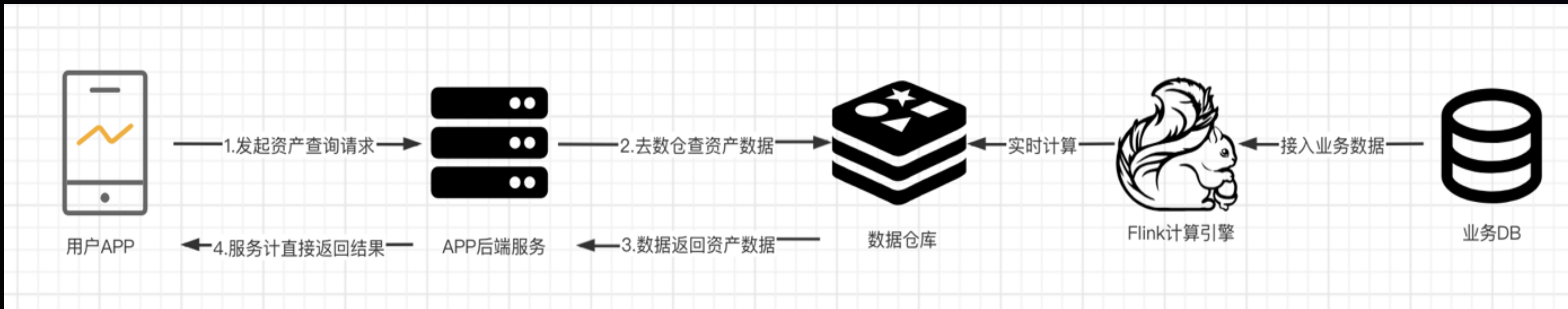
事件驱动的流计算

- 使用Flink引擎，通过订阅相关的上游数据源变动，将数据和状态存储在State中，在对应上游变更时触发算出对应的用户资产计算
- 只需获取对应数据源的changelog，开销小
- 事件驱动触发计算，实时性高

效率更高，实时性更好的选择！

流计算的架构

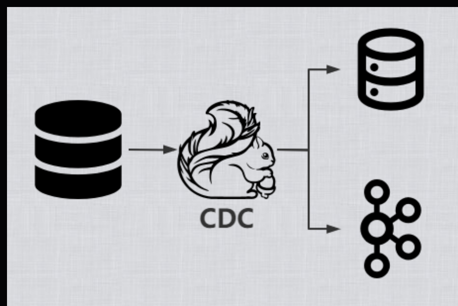




架构实现-总览

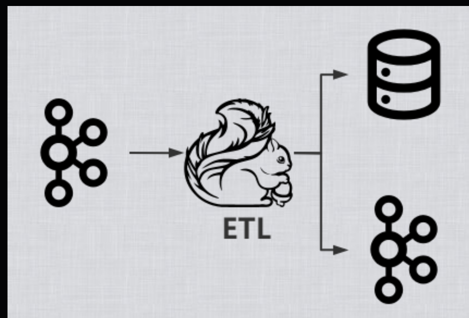
Step1. 数据接入

准备好业务DB相关的库表，开启binlog，使用FlinkCDC接入业务DB的数据，保存到数仓DB和消息队列中



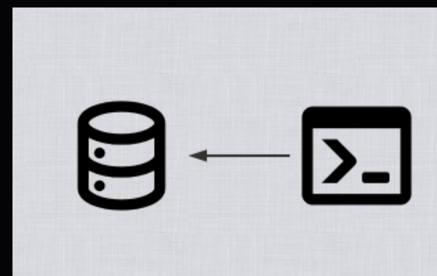
Step2. 数据ETL

使用Flink SQL/Streaming对已接入数仓的数据进行业务逻辑计算的ETL，并将计算结果也保存到数仓DB和消息队列中



Step3. 提供数据

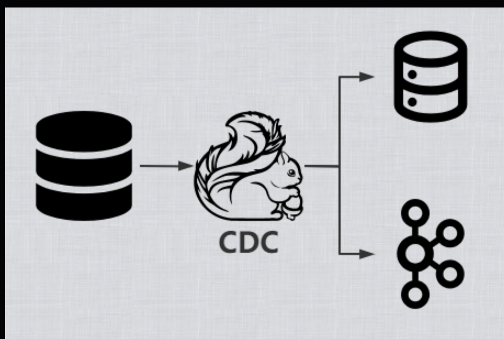
将数仓DB的数据开发或以接口的形式对外提供查询



架构实现-数据接入

Step1. 数据接入

准备好业务DB相关的库表，开启binlog，使用FlinkCDC接入业务DB的数据，保存到数仓DB和消息队列中

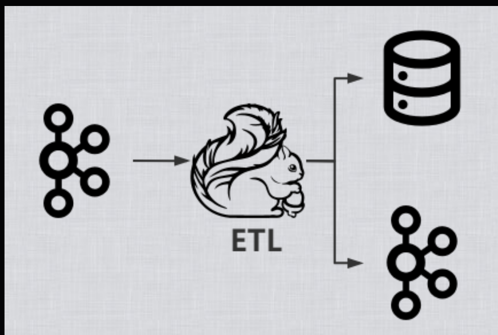



```
// 构建业务数据表和目标表
createSourceSinkTable(tEnv);

// 同步到实时数仓消息队列中
statementSet.addInsertSql("insert into kafka_user_cash select * from mysql_user_cash");
statementSet.addInsertSql("insert into kafka_user_position select * from mysql_user_position");
statementSet.addInsertSql("insert into kafka_stock_quotation select * from mysql_stock_quotation");
statementSet.addInsertSql("insert into kafka_user_inout select * from mysql_user_inout");

statementSet.execute();
```

架构实现-数据ETL

Step2. 数据ETL

使用Flink SQL/Streaming对已接入数仓的数据进行业务逻辑计算的ETL，并将计算结果也保存到数仓DB和消息队列中



Java Scala Python SQL CLI

```java
// instantiate table environment
Configuration configuration = new Configuration();
// set low-level key-value options
configuration.setString("table.exec.mini-batch.enabled", "true");
configuration.setString("table.exec.mini-batch.allow-latency", "5 s");
configuration.setString("table.exec.mini-batch.size", "5000");
EnvironmentSettings settings = EnvironmentSettings.newInstance()
        .inStreamingMode().withConfiguration(configuration).build();
TableEnvironment tEnv = TableEnvironment.create(settings);

// access flink configuration after table environment instantiation
TableConfig tableConfig = tEnv.getConfig();
// set low-level key-value options
tableConfig.set("table.exec.mini-batch.enabled", "true");
tableConfig.set("table.exec.mini-batch.allow-latency", "5 s");
tableConfig.set("table.exec.mini-batch.size", "5000");
```

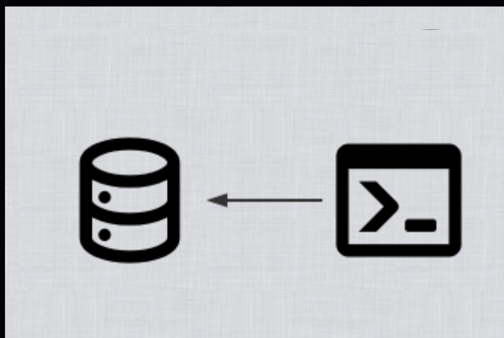
```
ON user_cash.uid = user_position_asset.uid;
```

total_value

架构实现-提供数据

Step3. 提供数据

数仓DB的数据以接口的形式对外提供查询



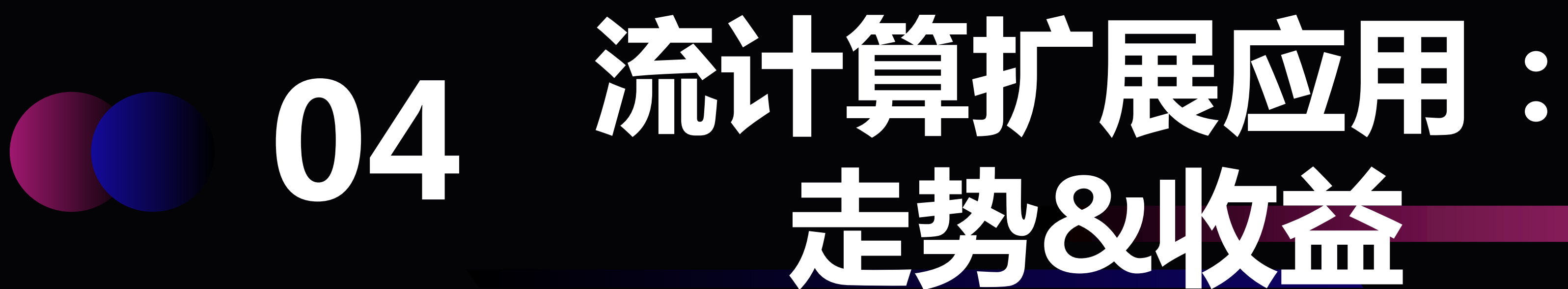
```
stEnv.executeSql("" +
        "CREATE TABLE kafka_user_asset (\n" +
        "  uid INT NOT NULL,\n" +
        "  cash_value double,\n" +
```

```
// 数据结果\
stEnv.creat
createSinkT
statementSe                    total_value from user_asset");
statementSe                    total_value from user_asset");
statementSe
```

uid	cash_value	position_value	total_value
1	100	500	600
2	200	200	400
3	0	900	900

用户资产表

```
        ")" +
        "");
```



04 流计算扩展应用：走势&收益

数据扩展应用

数据中台的定位

是数据中心，但不仅仅是数据中心

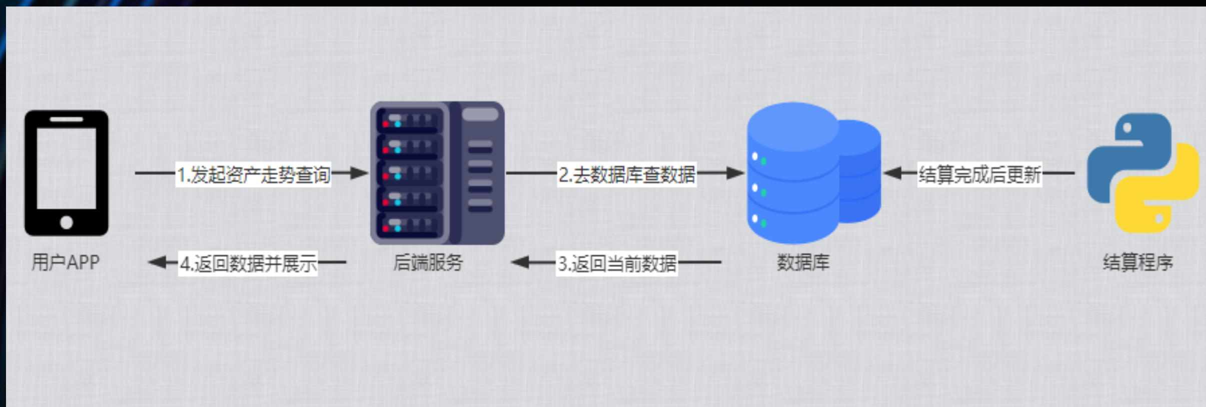
数据治理、统一口径、数据建设

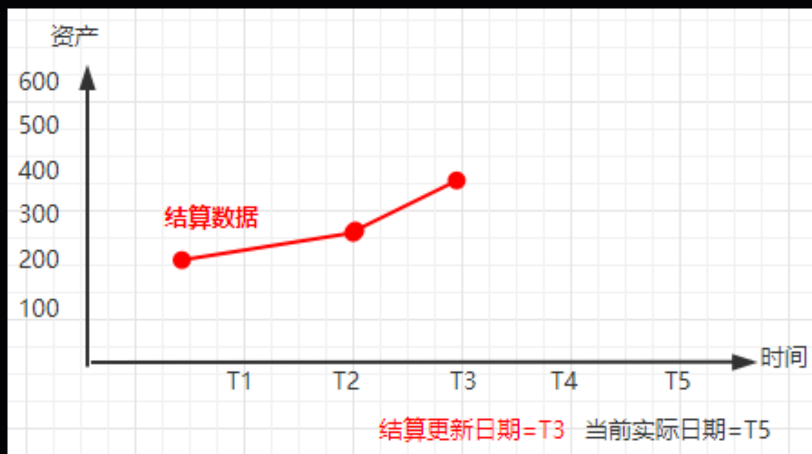
数据的价值与应用

发挥自身优势，充分挖掘数据资产，探索数据可能

数据共享、数据价值变现

应用1.资产走势





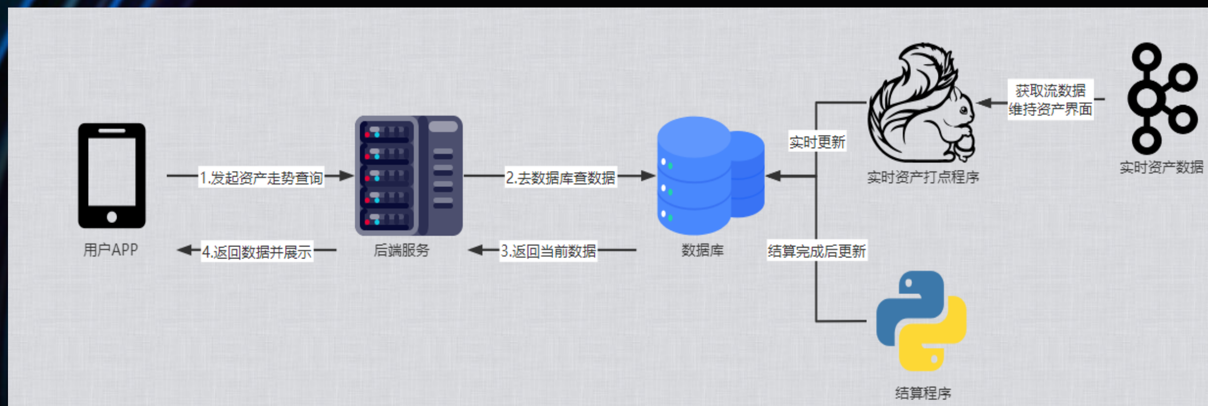
背景

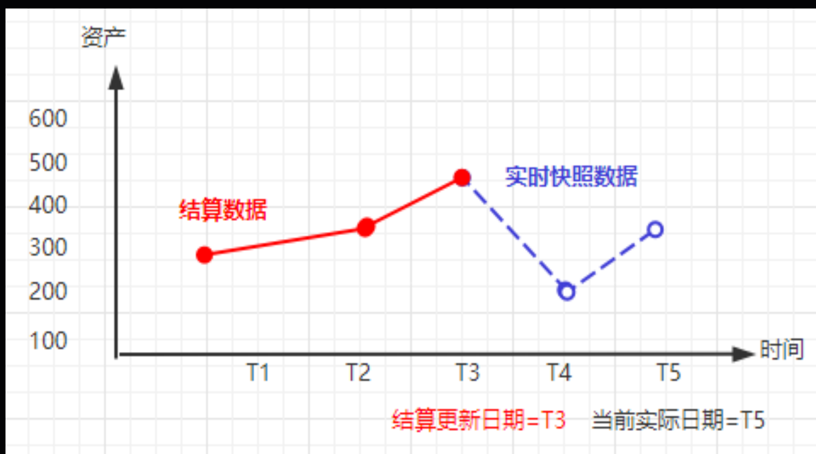
前端需要展示客户一段时间内的资产变化，而每日的资产数据由后台在结算后导入数据库

由于结算的过程涉及交易所处理数据，以及受一些外部的流程和节假日的影响，所以得到结算数据的时间相对不可控，短则1-2天，长则3-5天，期间的用户资产因为没有更新无法被查到，客户端只能读取到数据库中已有的、后结算程序不定期更新的数据，而近几日的数据因未结算无法看到，用户体验较差

(-)阿里云 intel.

应用1.资产走势



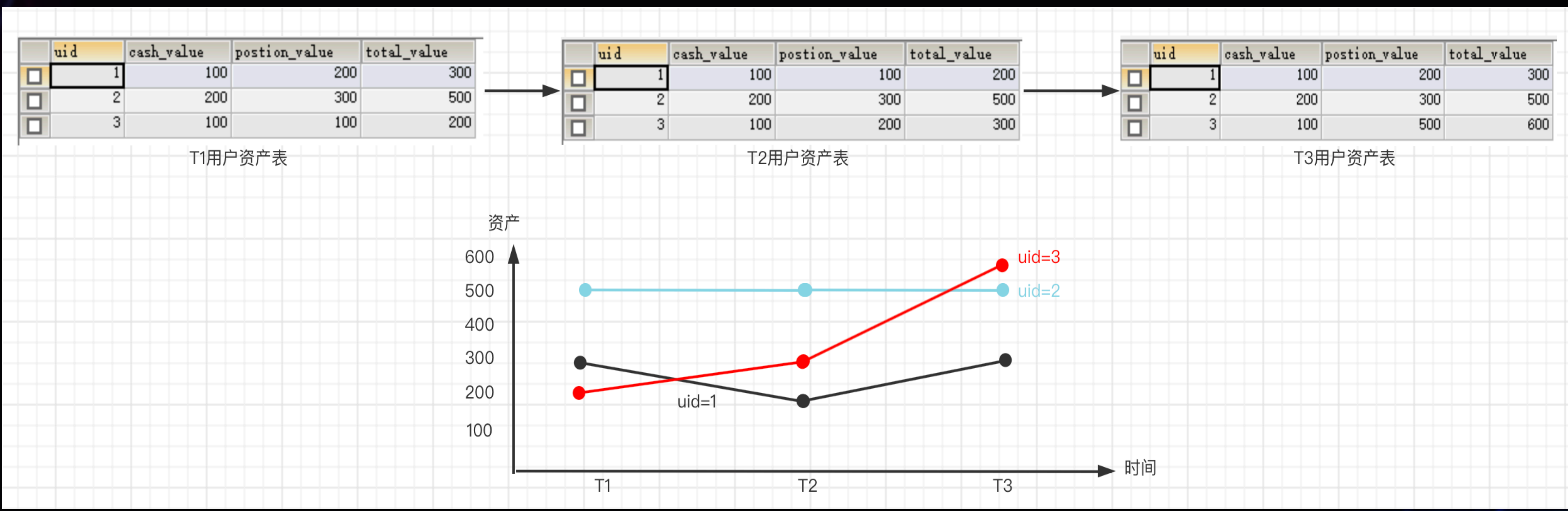


方案

这里如果引入Flink引擎，将业务数据库或上游里的总资产为数据源输入，通过Flink实时地给全量用户总资产打点，便可得到到用户实时资产的走势数据，再与原有的结算的资产走势数据做合并，便可用于弥补资产走势无最新数据展示的问题，提高了用户体验

此外，打点粒度支持到分钟级，所以资产走势除了原本支持的按日查询外，还能展示今日、近5日等分钟级别的资产走势

应用1.资产走势



uid	cash_value	postion_value	total_value
1	100	200	300
2	200	300	500
3	100	100	200

T1用户资产表

uid	cash_value	postion_value	total_value
1	100	100	200
2	200	300	500
3	100	200	300

T2用户资产表

uid	cash_value	postion_value	total_value
1	100	200	300
2	200	300	500
3	100	500	600

T3用户资产表

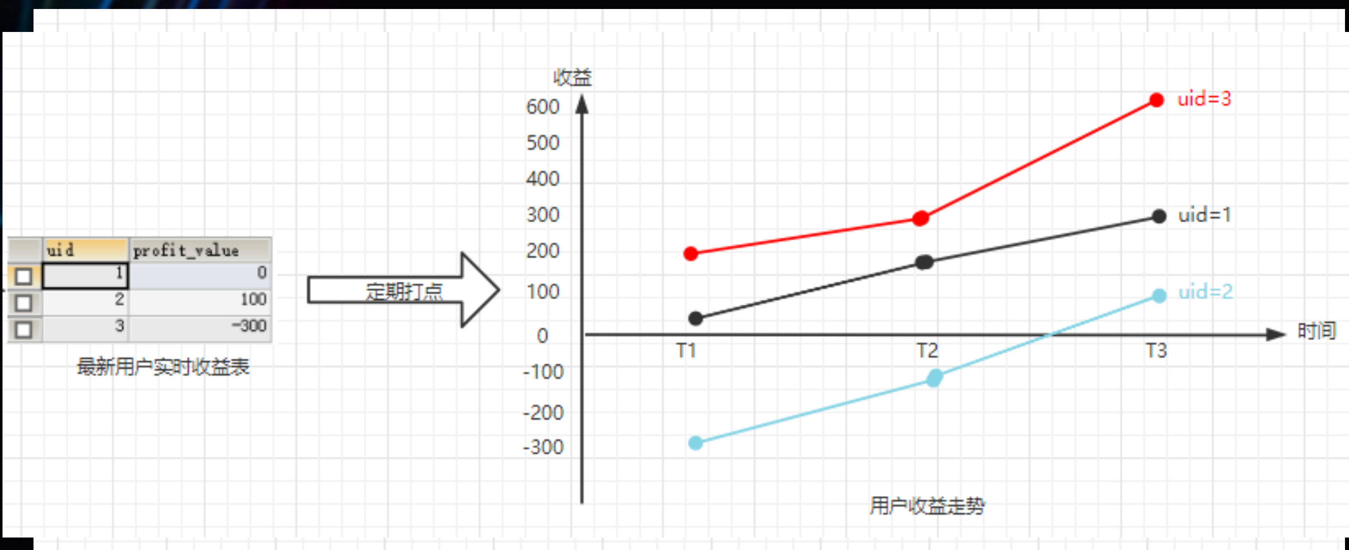
应用1.资产走势

```java
@Override
public void onTimer(long timestamp, ProcessFunction<UserAsset, UserAssetSnapshot>.OnTimerContext ctx,
    // 每分钟进行触发输出
    ctx.timerService().registerProcessingTimeTimer(timestamp + 60 * 1000);
    // Timer触发时间小于等于有记录的asset的最新update_time，否则说明数据有被更新过或定时器触发延迟
    var snapshot = snapshotState.value();
    if (timestamp <= snapshot.getUpdate_time()) {
        return;
    }
    snapshot.setUpdate_time(roundMinuteTs(timestamp));
    collector.collect(snapshot);
}

private Long roundMinuteTs(long ts) {
    LocalDateTime localDateTime = new Date(ts).toInstant().atZone(ZoneOffset.UTC).toLocalDateTime();
    localDateTime = localDateTime.truncatedTo(ChronoUnit.MINUTES);
    return localDateTime.toInstant(ZoneOffset.UTC).toEpochMilli();
}
```

实现

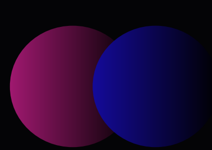
1. 接收上游不断更新的全量用户资产数据，并在Flink内部不断维护最新的用户资产截面
2. 配置定时器，定期地扫描最新的用户资产截面，配上系统设定的时间戳，得到当前截面的资产快照数据
3. 将当前截面的资产快照数据输出到下游的数据库或消息队列中

应用2.收益&走势



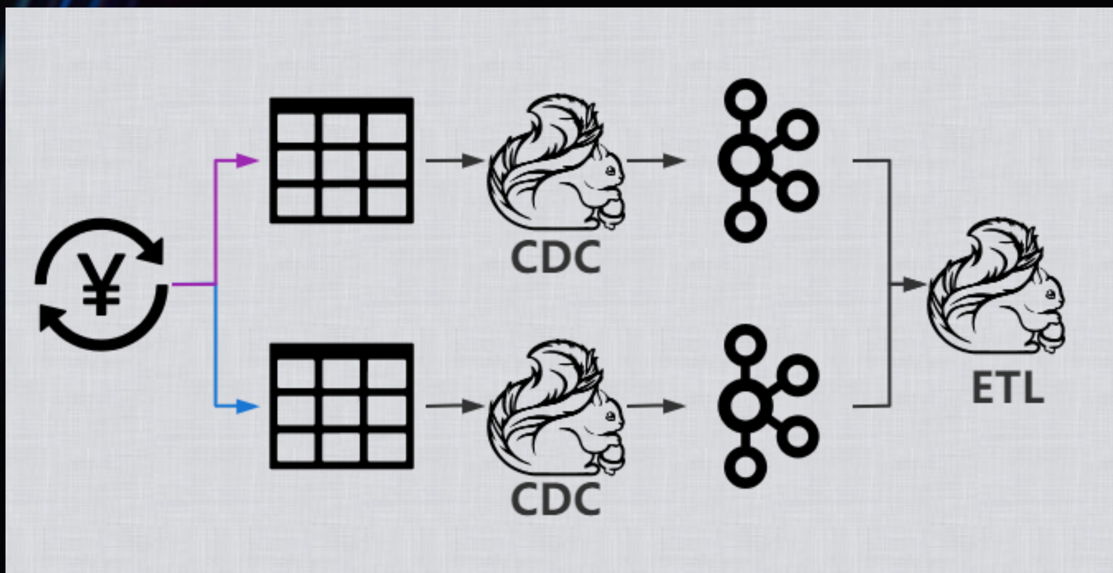
概览

- 根据实时总资产和流水数据，算出实时总收益
- 根据收益=期末资产-期初资产-期间净流入资产；这里我们若取期初资产为0，期末资产为最新时间点用户资产，那么收益=期末资产-所有净流入资产
- 可以按同样方式计算出收益的走势，可用于评估当前的交易策略



05 其他挑战与总结

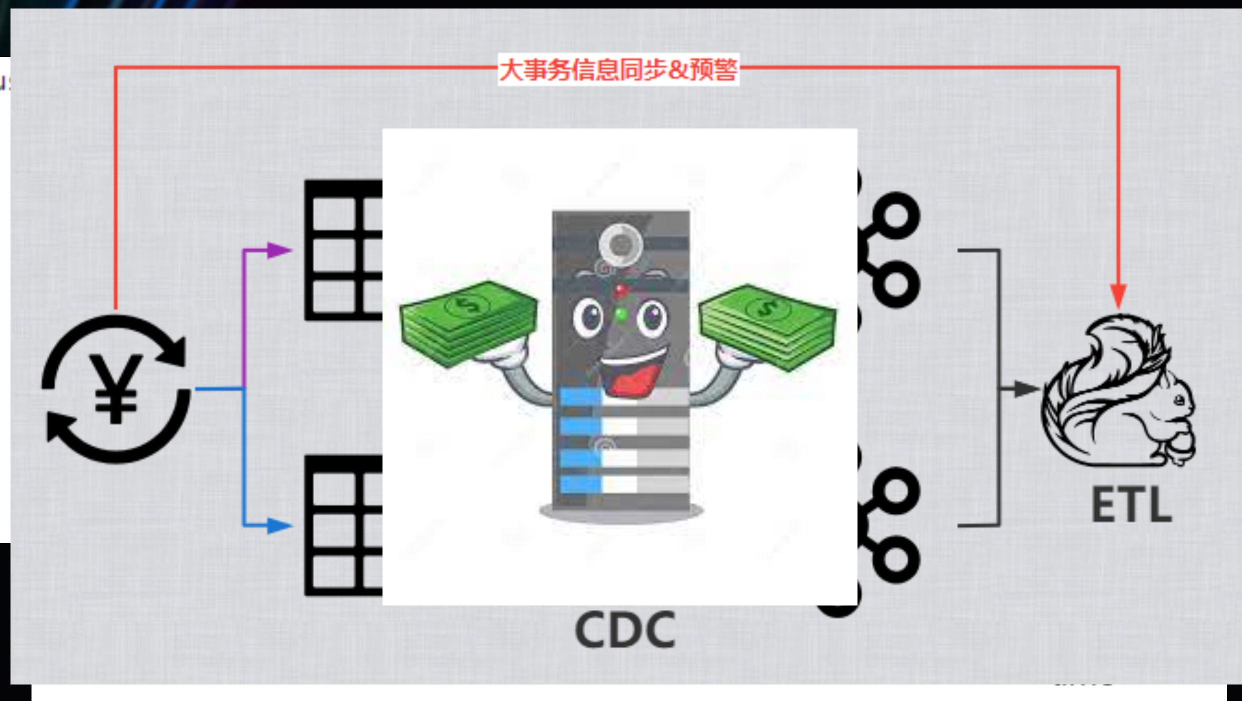
挑战1-DB事务



事务与CDC

因现金表和持仓表在CDC和流计算的逻辑中已经彻底解耦，所以若DB中出现事务性的关联变更操作时，过程对Flink而言是不可见的，虽然能保证最终一致性，但计算中途可能出现错误的结果（例如一个交易大事务，会同时修改现金表和持仓表的数据，因下游处理过程可能不一致，有可能导致计算总资产中途可能会出现钱货变动不同步的情况，在此期间算出的总资产不准确）

挑战1-DB事务

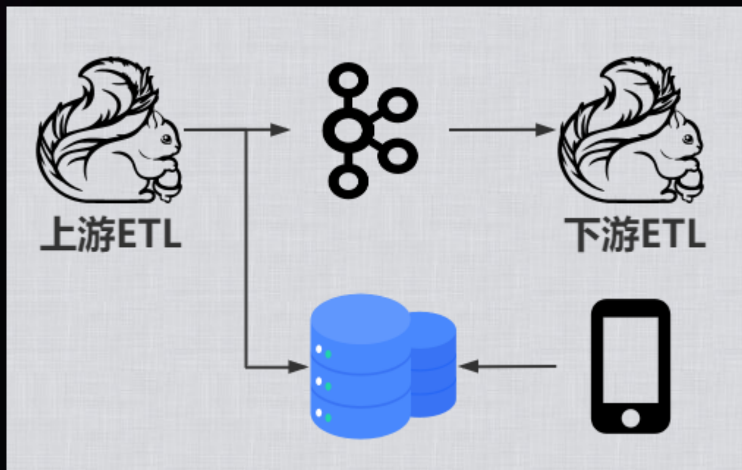


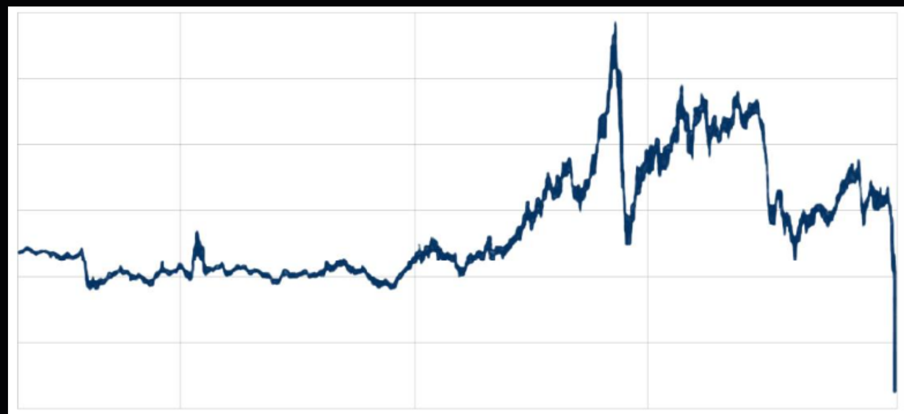
应对方案

在计算时要结合事务的特点做一些特定优化，可能的方案包括：

1. 使用窗口聚合数据（Session-window）
2. Sink端针对大事务感知并做延迟处理
3. 保证机器性能足够

挑战2-数据稳定性



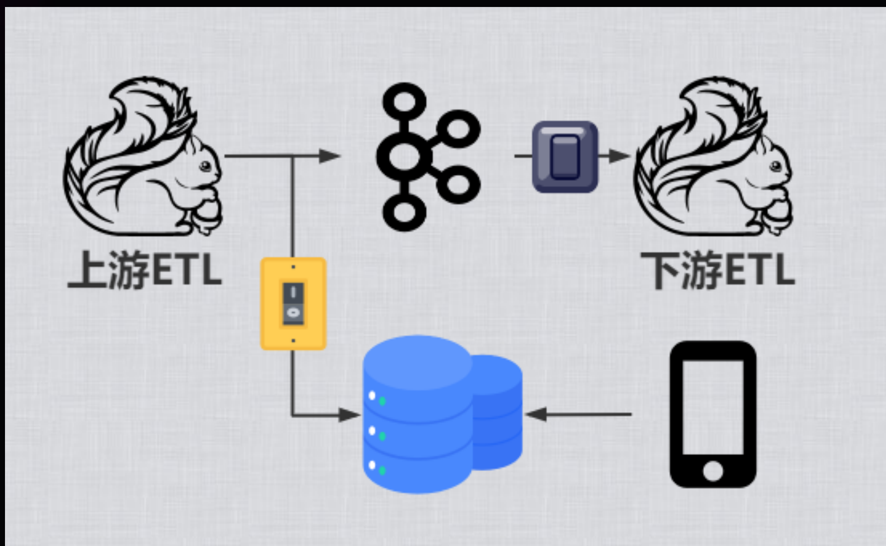


数据波动对下游影响

程序在发布、升级、上游修改等操作特殊期间，产生的中间数据不一定稳定准确，如果被使用或者传到到下游，可能会引发一系列的错误

（例如上游修数据导致资产计算结果中间出现了脏数据，正好被下游资产走势程序写进了数据库，会导致曲线出现明显的尖刺，且无法重跑修复）

挑战2-数据稳定性



应对方案

尽可能给下游稳定的数据，在可预见的变动和操作之前，通过数据“断路”的方案，将变动期间产生的数据进行屏蔽；当数据变动完成稳定之后，在把“断路”的数据恢复，提供给下游

总结

解决架构痛点

- 减轻业务DB压力
- 计算逻辑的复用
- 算力的动态扩缩容

构建数据资产

- 实现和业务相同的逻辑
- 实现数据治理、数据共享

开发数据应用

- 提供实时数据分析应用
- 提供数据二次开发的能力

总结

数据源：用户现金表、用户持仓表、股票价格表、用户流水表

uid	cash_value
1	100
2	200

用户现金表
user_cash

uid	stock_id	quantity
1	1	100
1	2	200
2	1	200
3	3	300
(NULL)	(NULL)	(NULL)

用户持仓表
user_position

stock_id	price
1	1
2	2
3	3
(NULL)	(NULL)

股票行情报价表
stock_quotation

uid	inout_type	inout_value
1	cash_in	300
2	cash_in	400
3	position_in	500

用户流水表

uid	cash_value	position_value	total_value
1	100	500	600
2	200	200	400
3	0	900	900

用户资产表

uid	ts	asset
1	10001	100
1	10002	200
1	10003	300
2	10002	300
2	10003	300

用户资产走势

uid	profit_value
1	0
2	100
3	-300

最新用户实时收益表

uid	ts	profit
1	10001	100
1	10002	150
1	10003	-50
2	10002	100
2	10003	50

用户收益走势

THANK YOU

谢 谢 观 看