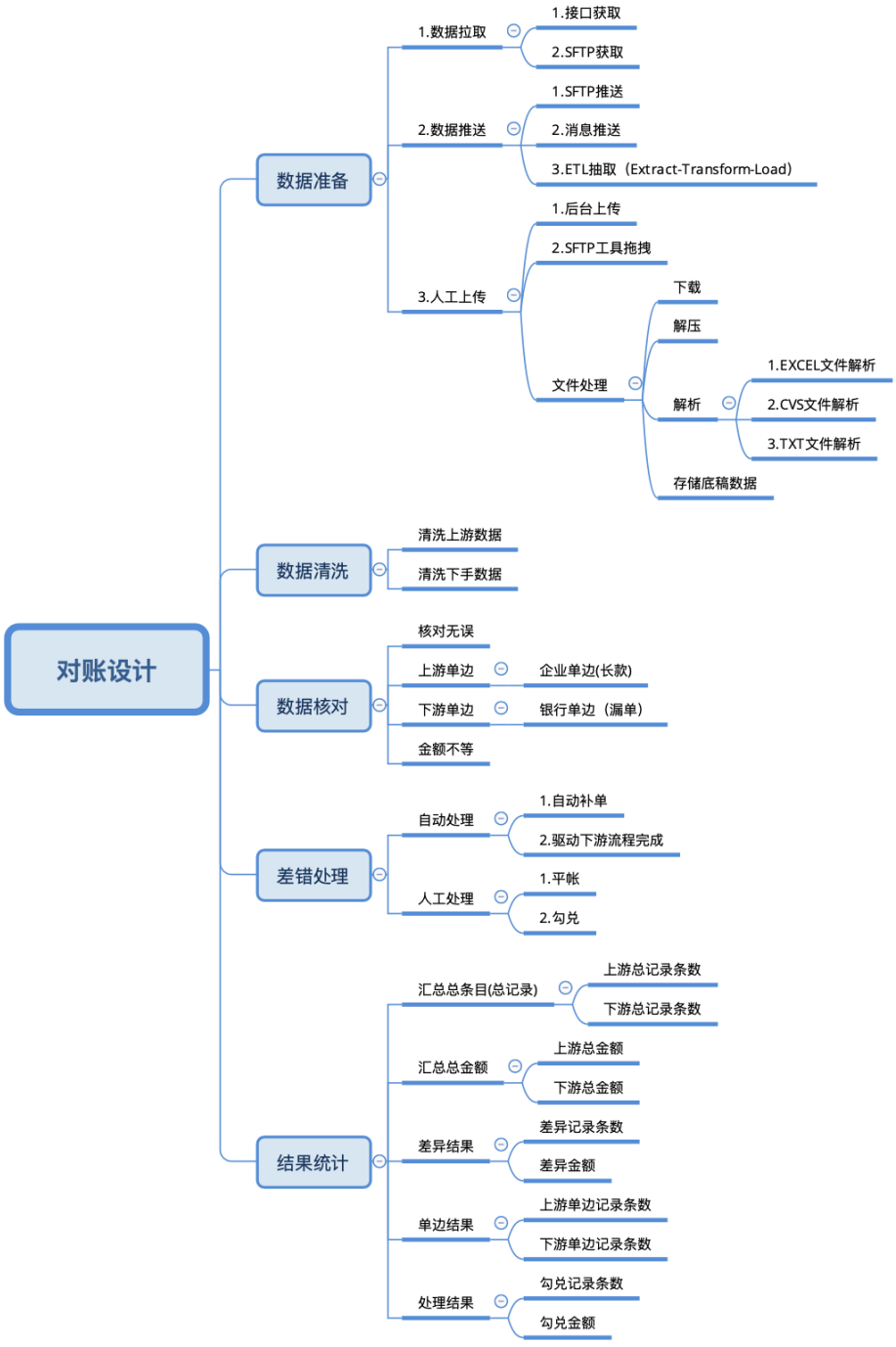
一、对账整体设计

从整体来看，按照时序维度的先后，系统对账主要分为三阶段的工作。分别是**数据准备**、**数据核对**和**差错处理**。数据准备细分一下，又分为文件获取、文件解析、数据清洗；在对账专业概念中，数据核对和差错处理又叫轧账和平账。

具体设计脑图如下：



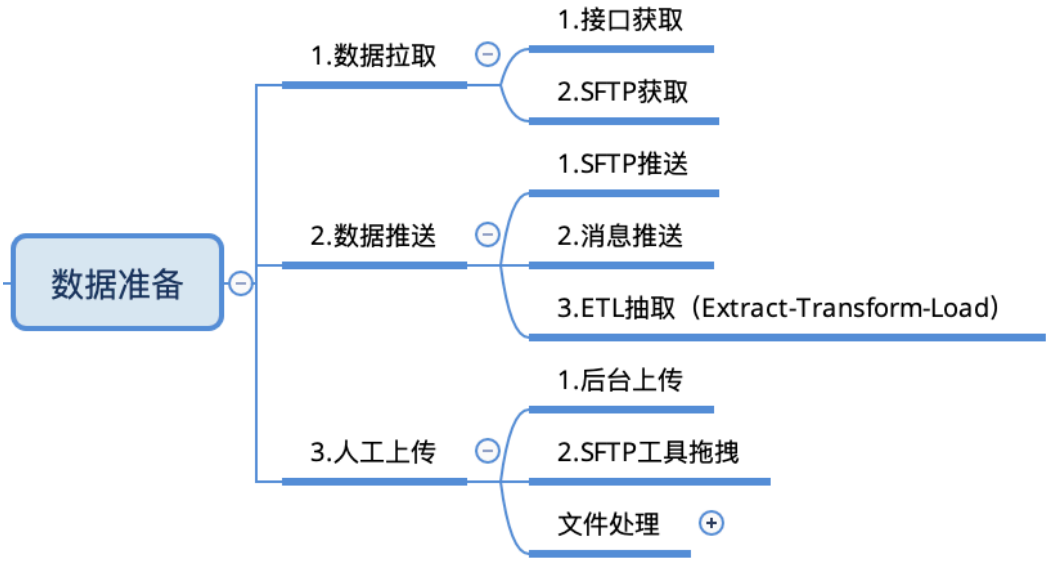
二、对账各个模块设计

1.数据准备

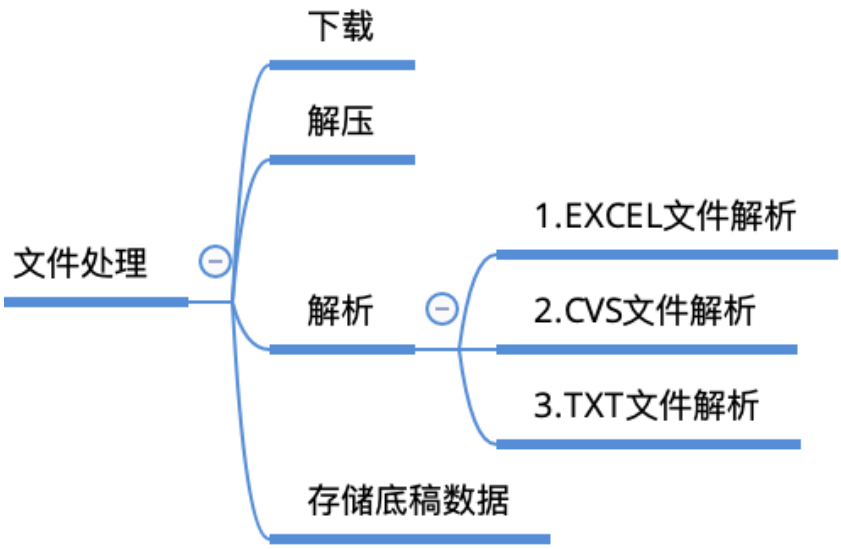
数据准备，顾名思义，我们需要把对账所需的全部数据，接入到我们的对账系统。该模块主要实现两个目标：

- 为不同的外部系统提供多元化的接入机制。
- 通过数据适配的手段把外部数据以统一的格式进行转换和存储。

在数据接入层，我们会针对不同的数据接入方提供三种不同的数据接入模式。如下图：



- 数据拉取：主动拉取数据，并通过数据适配的方式，将数据存储到对账数据池中。
- 数据推送：指定标准规范和格式，供各个接入方使用，统一格式推送到对账服务。
- 人工上传：提供标准的文件模板，由业务接入方填充数据，通过后台文件上传或SFTP上传工具的方式将数据上传到对账服务。



人工上传文件处理方式步骤如下：

1）下载文件

从指定SFTP服务器下载文件；

2）解压文件

一般为zip压缩包，节省存储空间，提高上传和下载速度；

3）解析文件

一般文件格式为EXCEL（财务人工上传文件，一般从银行或第三方支付后台下载）、CSV（数据接入方一般从数据库导出格式，或第三方支付提供的文件格式）、TXT

4）存储数据

将第三步得到的数据存储、持久化到数据库，一般底稿数据都存储最全数方便问题追溯

2.数据清洗

顾名思义，即对准备的上下游数据进行清洗。

清洗的作用或原因：

1）从底稿提取对账核心字段

一般参与对账的字段就几个，具体字段：银行卡号、银行单号、业务单号、支付金额、支付方式、支付完成时间、核对状态；上文提到底稿一般建议存储所有数据，数据太多影响对账性能和效率

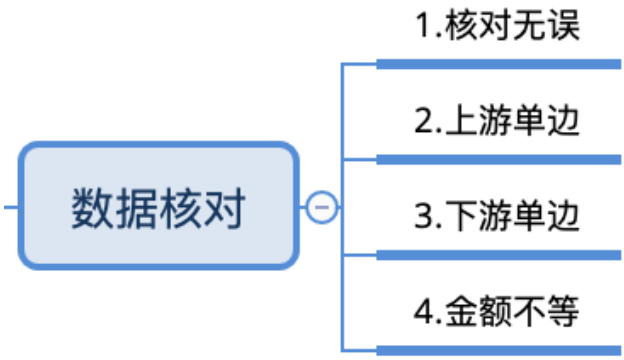
2）合并、排除无用数据

上文提到底稿一般建议存储所有数据，难免有无用数据需要剔除，或者排除业务或财务指定无需对账的数据；合并特殊业务或流程产生的N对1数据。

3数据核对

数据核对是对账的核心，对账的主逻辑；一般对账方式有两种，即对明细账和对总账，对总账一般包括总金额和总条目。一般做法为对明细账，即上下游每条记录逐一比对。

核对一般就是两个结果：对上帐和对不上账，对不上账有分三种结果，上游单边（支付中一般称为企业单边，即长款），下游单边（支付中一般称为银行单边，即漏单），金额不等（即两边都有数据，金额对不上）。设计normal和different两张表，分别存储不同的核对结果数据，方便后期统计以及为业务提供能力。



具体对账的方法有多种，比如：

1）**sql核对**

exist insert select minus；最简单的方式，也是问题比较多方式，对数据库压力比较大，数据特别多的时候，对账效率比较低。

2）**redis核对**

set集合分别diff （inter）上游、下游数据；比较好的方式，可以降低数据库压力，redis方便根据数据量做水平扩展，

3）**sprak核**

采用流式运算进行比对；（具体做法待扩展）

4.差错处理

在一般系统中，差错处理分为两种，一种人工来处理，一种系统自动来处理。人工处理一般两个操作：平账和勾兑，勾兑一般处理的是单边情况，比如由于系统bug出现的单边问题，经由人工溯源修复bug之后，相关业务人员即可在对账后台将该条数据进行勾兑；系统自动处理一般为：自动补单和驱动下游流程完成两种方式。

主要有如下情况：

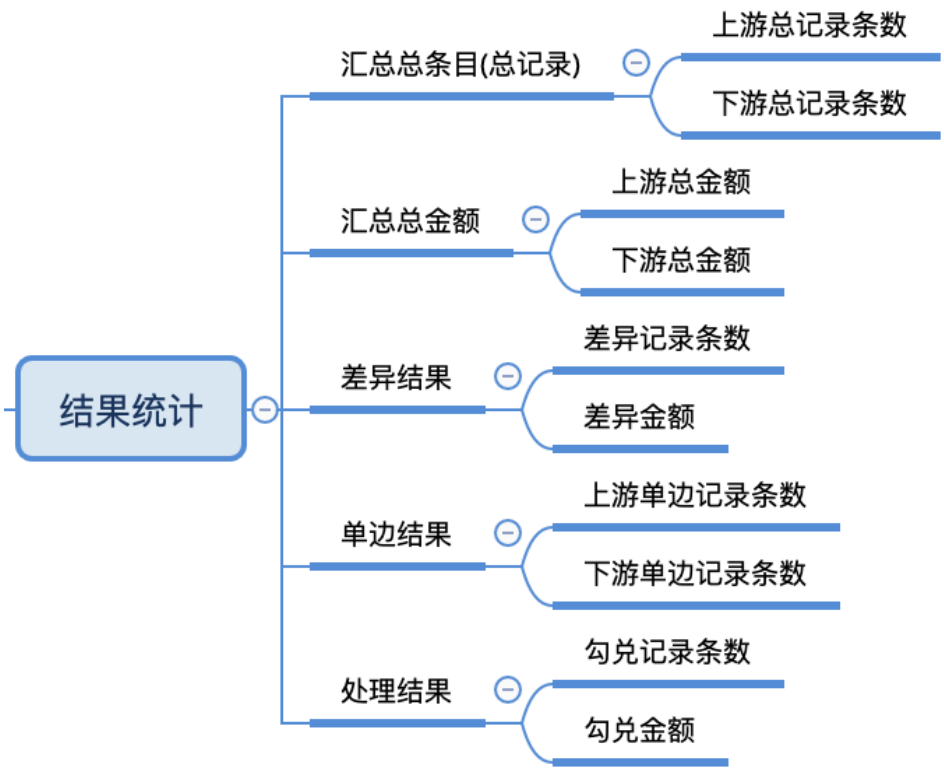
- **下游单边（银行单边）情况**：业务未支付，支付渠道已支付。这主要是本

地未正确接收到渠道下发的异步通知导致。 一般处理是将本地状态修改为已支付，并做响应的后续处理，比如通知业务方等。

- **上游单边（企业单边）情况**：业务已支付，但是支付渠道中无记录；或者本地无记录，支付渠道有记录。在排除跨日因素外，这种情况非常少见，需要了解具体原因后做处理。
- **金额不等情况**：业务已支付，支付渠道已支付，但是金额不同，这个需要人工核查。

5.对账统计

根据对账处理结果，统计的数据由：汇总总条目、汇总总金额、汇总差异结果、汇总单边结果、汇总处理结果；业务和财务关系的统计的相关信息有：对账完成时间、对账是否成功、平账的金额和订单数、差错的金额和订单数、缓存池金额和订单数等。



三、对账系统相关设计

1.分布式定时系统

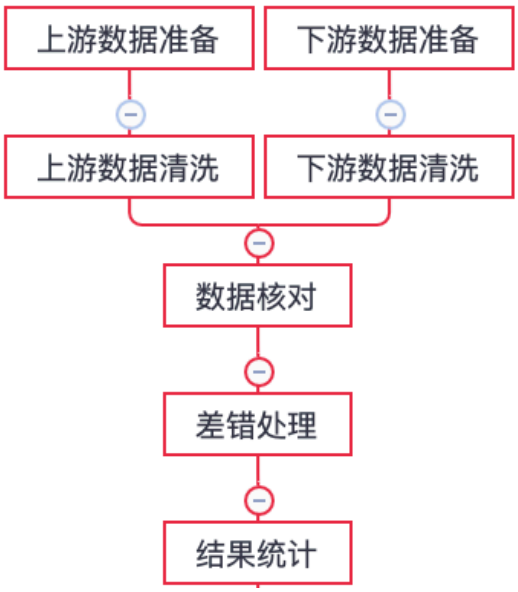
一般对账系统都是N+1离线对账，所有上述所有模块的设计一般使用定时任务去执行。不可能所有模块、所有银行卡都用一个任务去执行，也不可能只用一台机器去执行，这样一天可能都跑不完所有的数据。所以考虑到优化，一般设置

为集群分布式的去跑任务，所以涉及一个分布式定时系统对对账系统来说很重要，考虑成本和时间问题，可以简单实现分布式任务效果。分布式定时系统的设计后续再单独探讨。

即使所有任务都按模块化去进行划分，按模块化单独起任务去执行业务逻辑，也会存在时间效率的瓶颈（因为下文提到的依赖关系，导致并不能让所有的任务都并行起来）；再加上银行卡号比较多的情况下，最好情况就是各个银行卡号并行处理，即并发粒度设计到银行卡号维度，使用多线程把所有银行卡的对账任务并行起来。

2.依赖链设计

所有模块是存在依赖关系的，比如清洗之前，肯定要数据准备完成；但是上游数据和下游数据的准备、清洗可以并发的执行。整体依赖链如下：



3.核心对账优化

上文模块设计有提到核心对账的多种实现思路，这里推荐的两种：1）redis 实现　2）sprak 实现