博学谷
www.boxuegu.com

第十一章 Spring Cloud Alibaba Seata 分布式事务

一样的在线教育，不一样的教学品质

目录 Contents

Seata AT 模式工作原理

小节导学

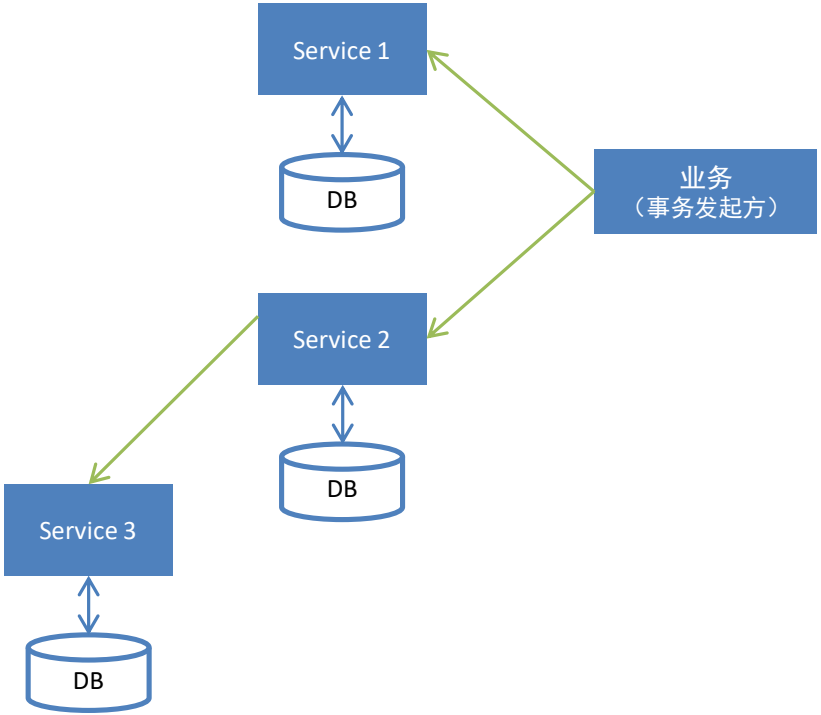
分布式事务是一个很麻烦的问题

如果有一个方法，可以近乎 0成本 的解决分布式事务问题，是不是感觉很神奇。

Seata AT 模式就是这个魔法棒，本节就学习一下 AT 模式。

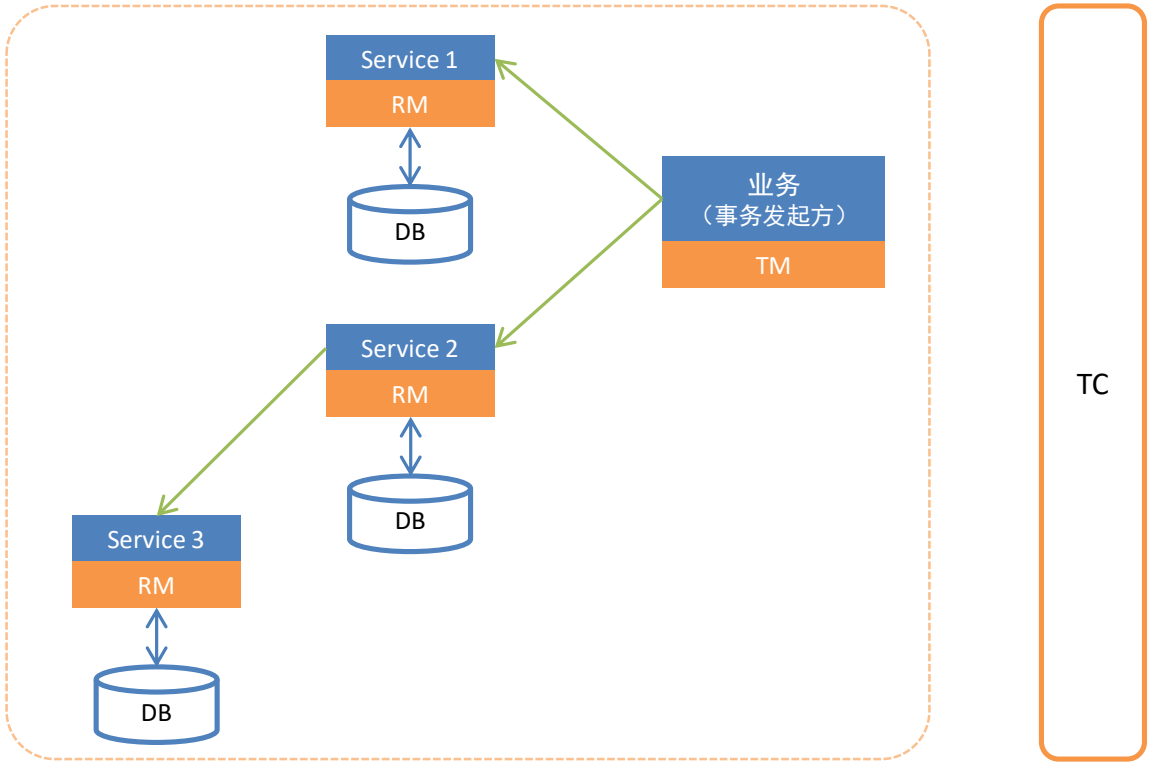
Seata AT 模式工作原理

工作机制



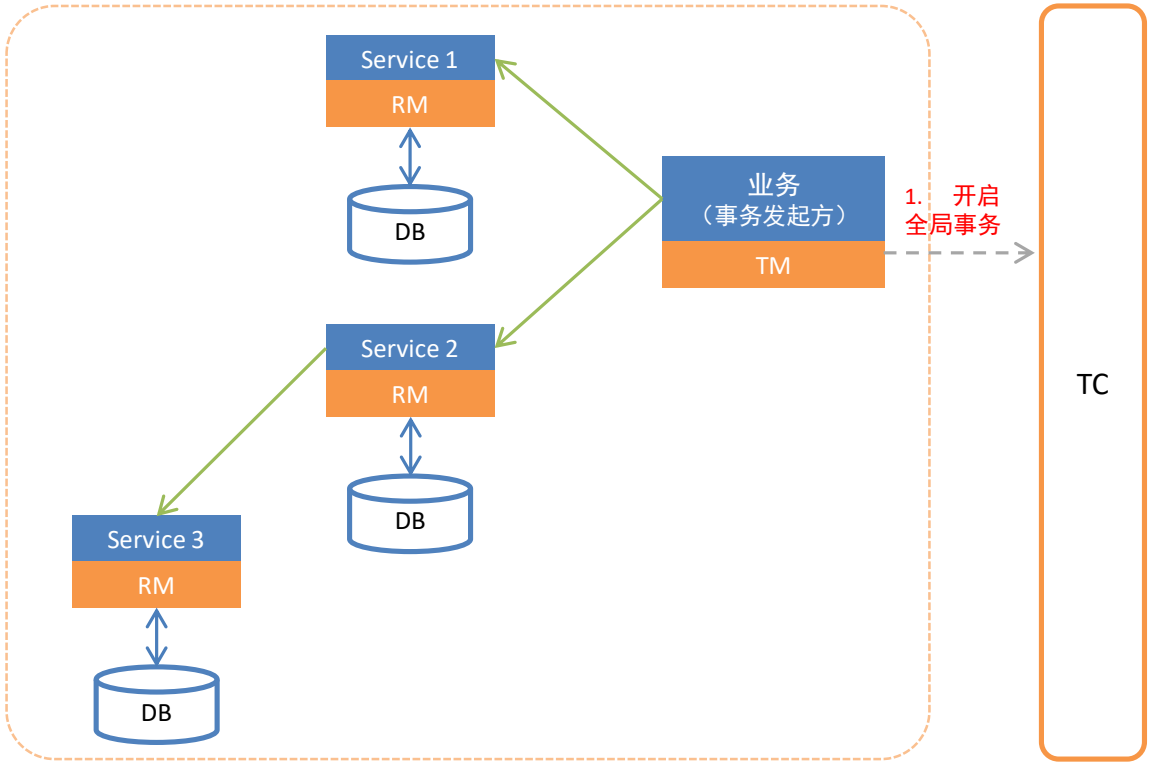
Seata AT 模式工作原理

工作机制



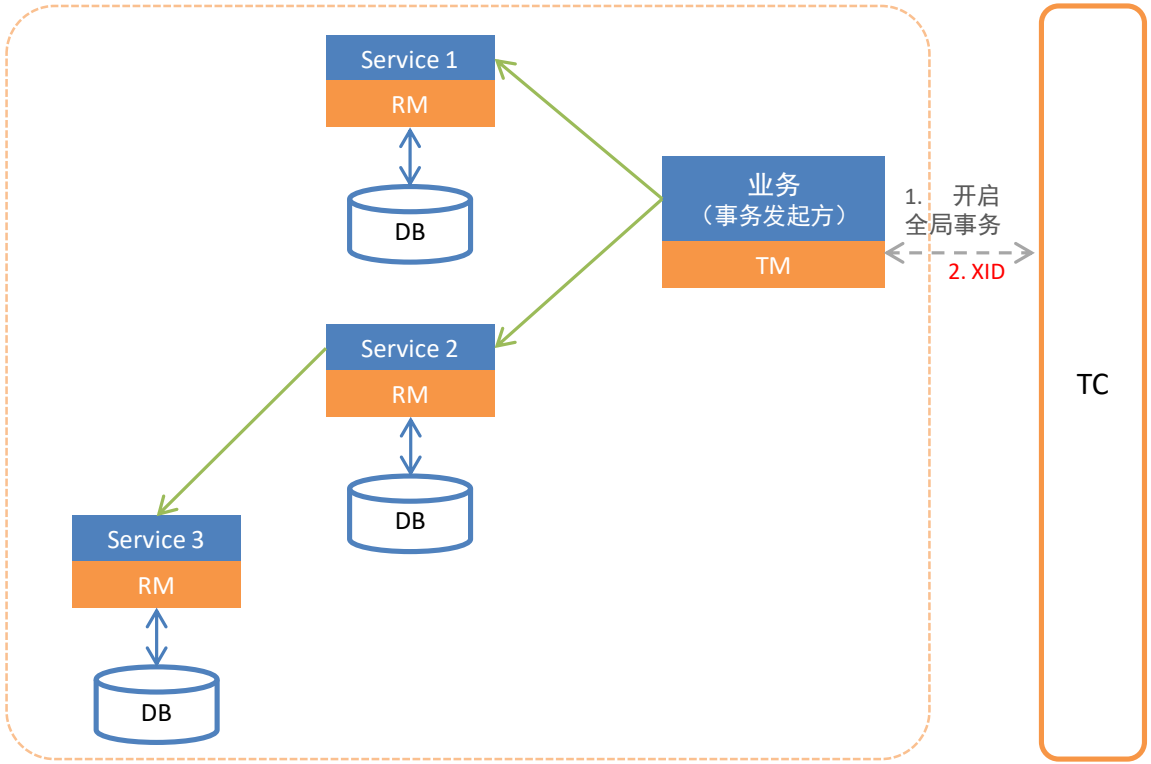
Seata AT 模式工作原理

工作机制



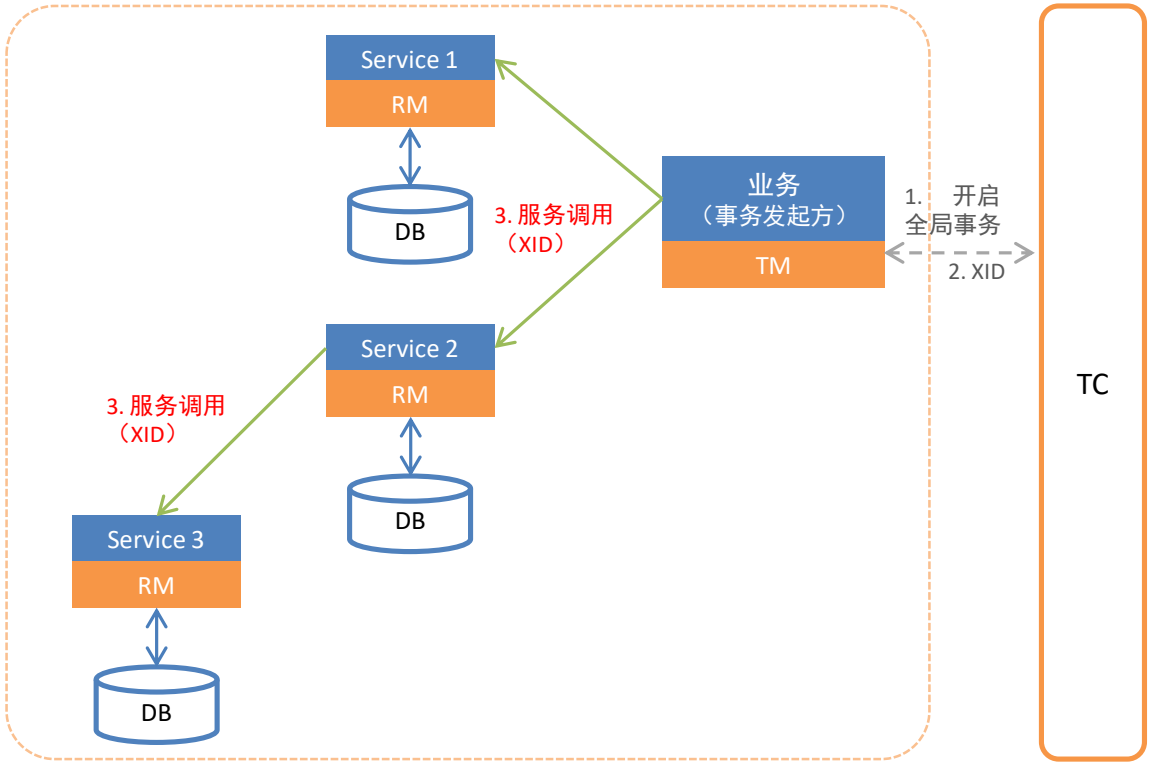
Seata AT 模式工作原理

工作机制



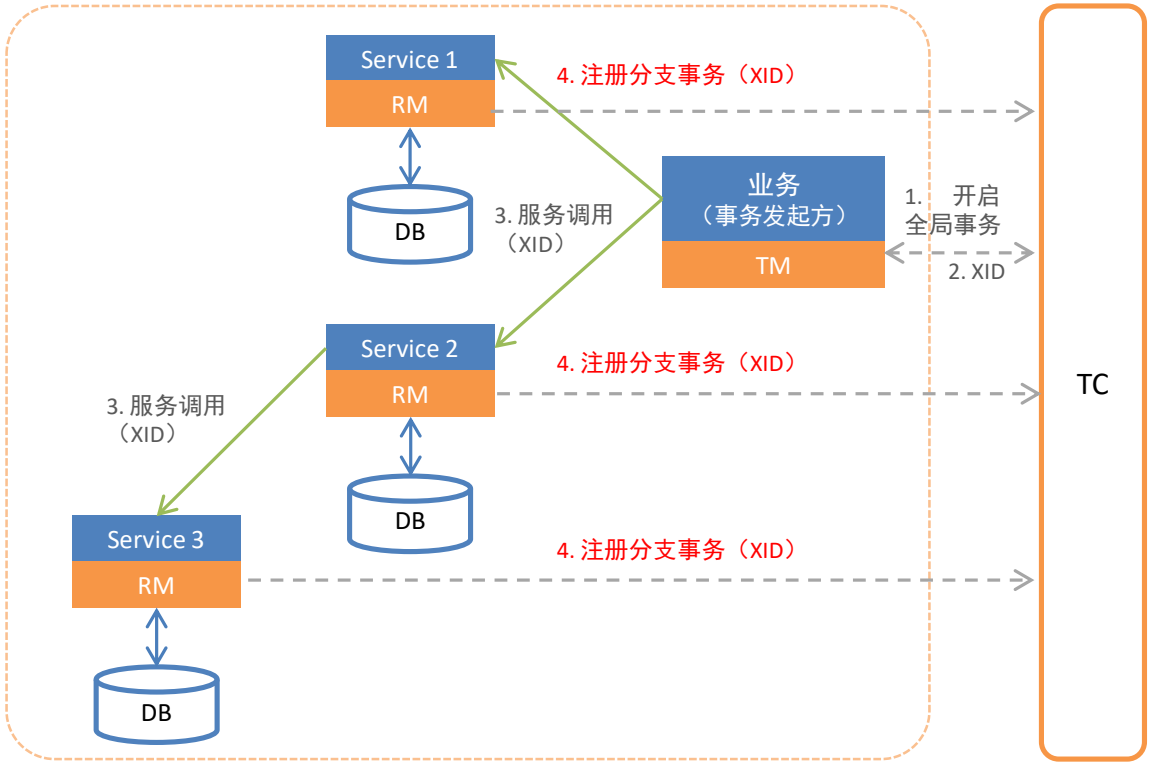
Seata AT 模式工作原理

工作机制



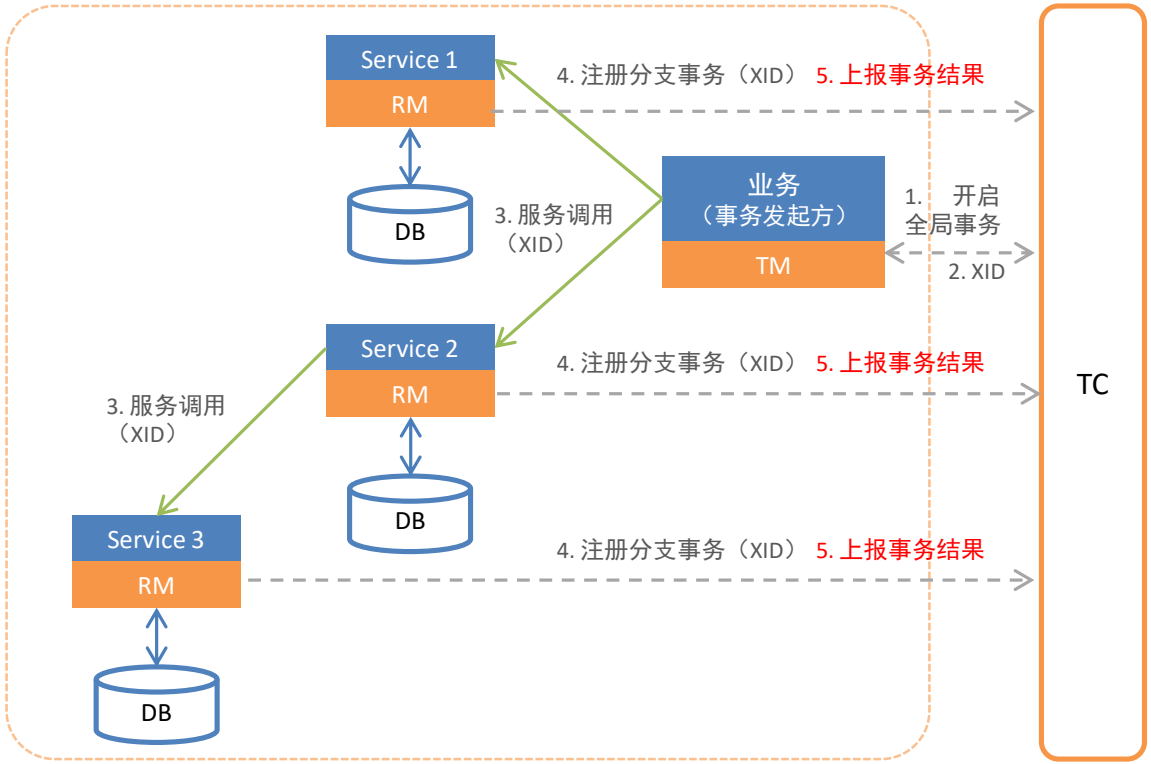
Seata AT 模式工作原理

工作机制



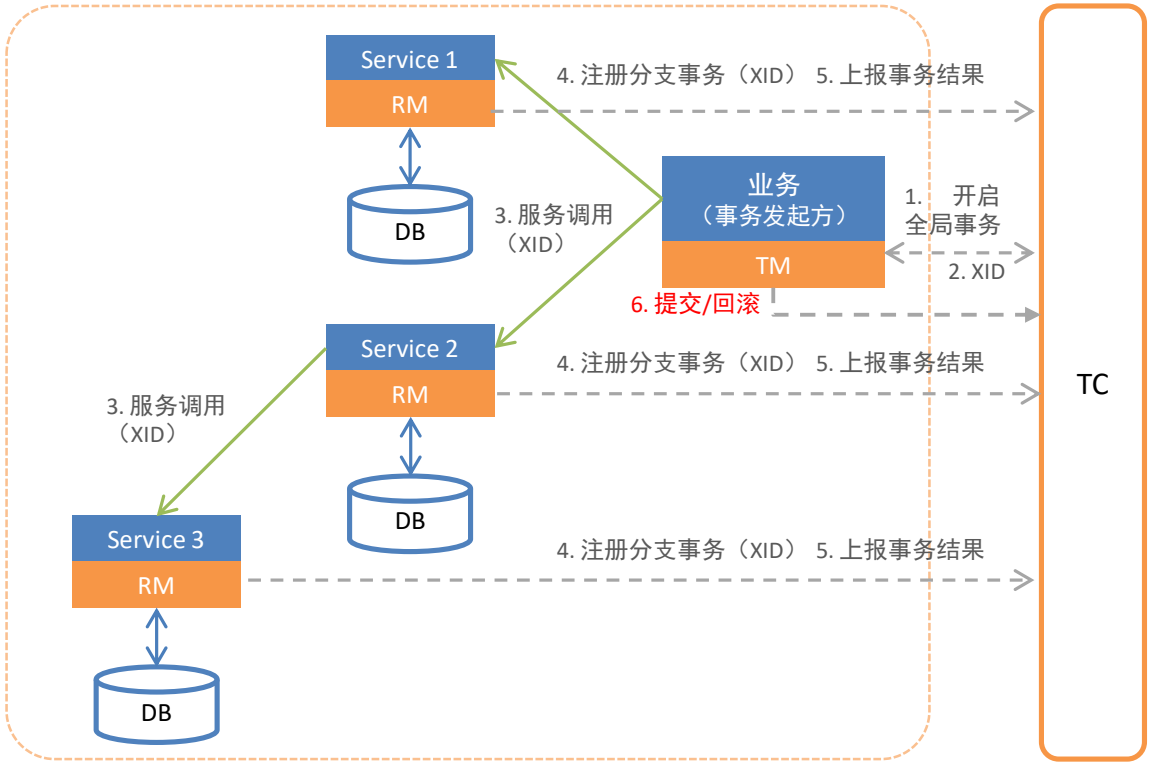
Seata AT 模式工作原理

工作机制



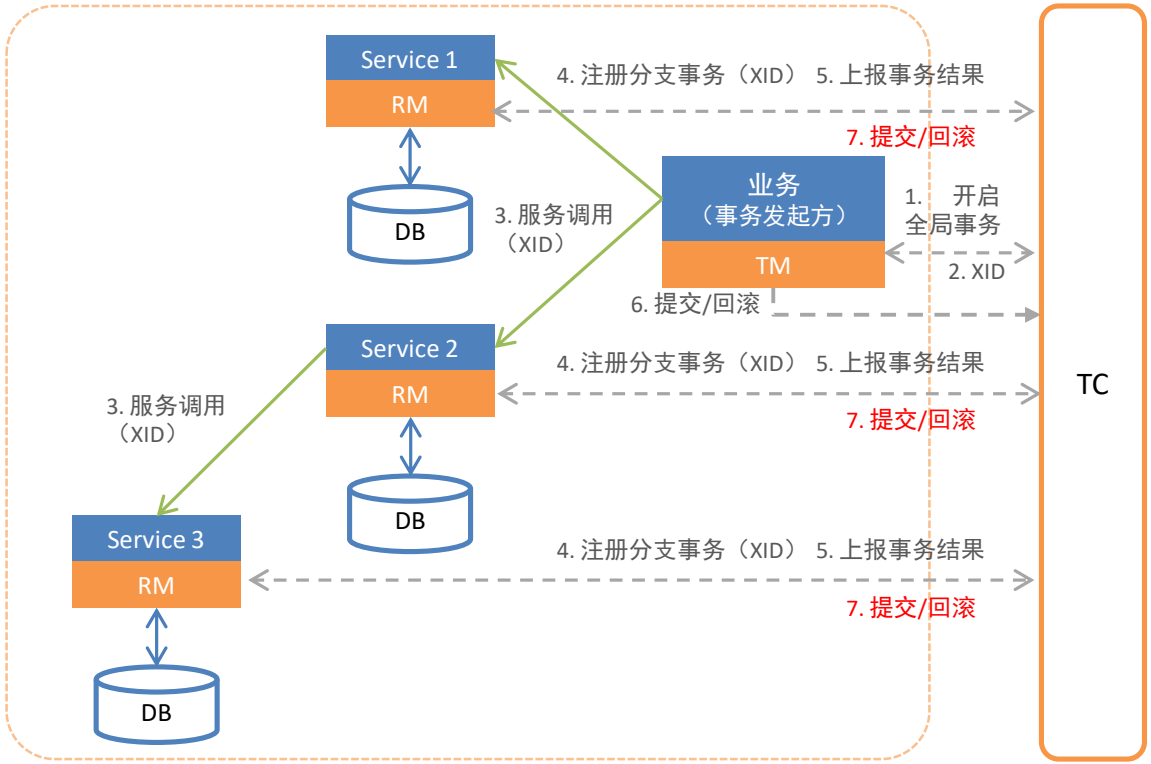
Seata AT 模式工作原理

工作机制



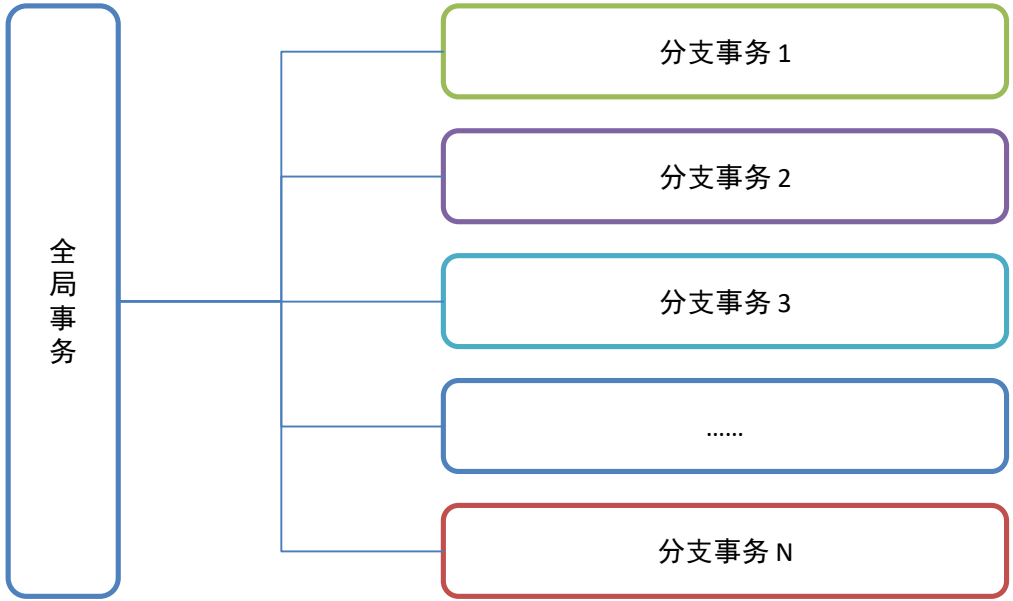
Seata AT 模式工作原理

工作机制



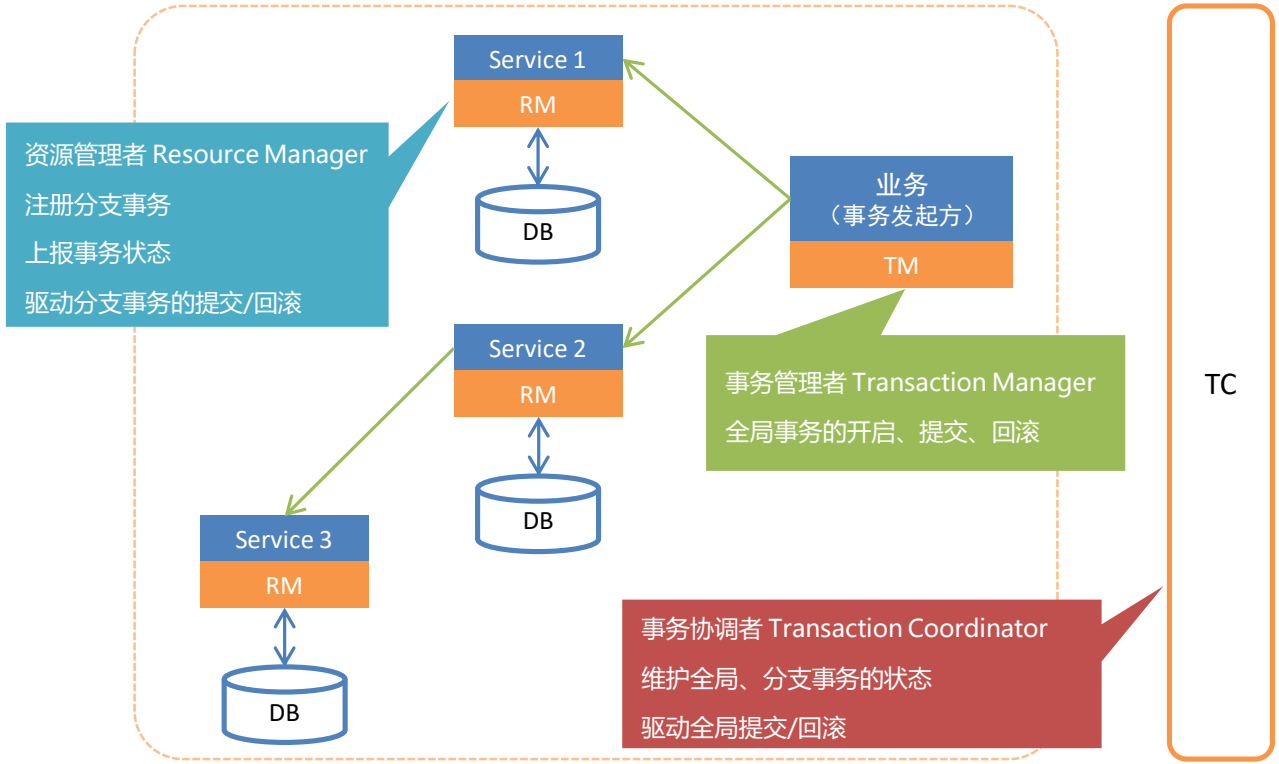
Seata AT 模式工作原理

工作机制



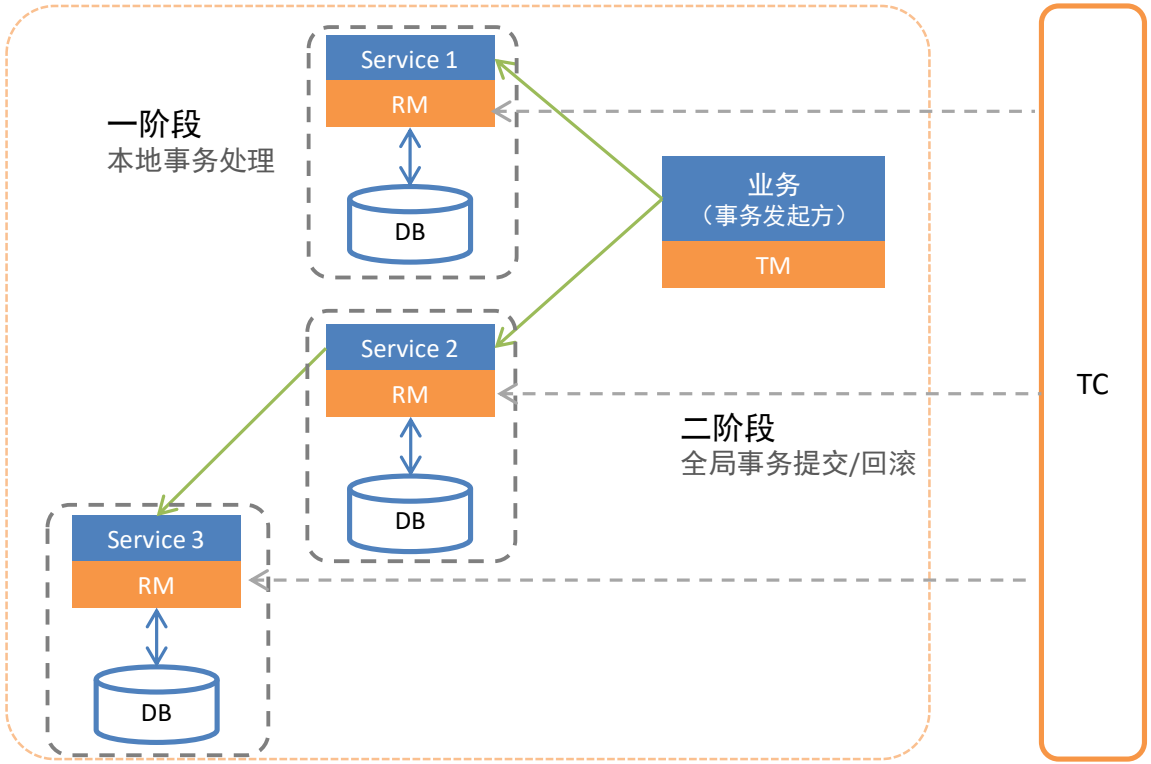
Seata AT 模式工作原理

核心组件



Seata AT 模式工作原理

两阶段提交

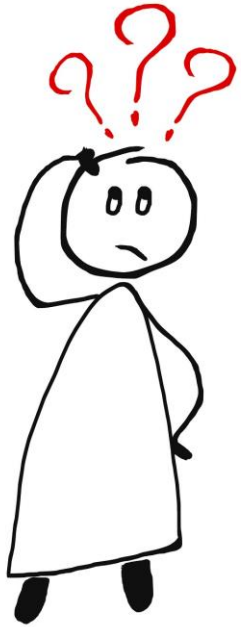


Seata AT 模式工作原理

疑问

最后的回滚是怎么做的？之前本地事务不都提交过了吗？

数据库里的回滚日志表是干啥用的？



Seata AT 模式工作原理

工作流程示例

业务表

Field	Type	Key
id	bigint(20)	PRI
name	varchar(100)	
since	varchar(100)	

分支事务的业务逻辑:

```
update product set name = 'GTS' where name = 'TXC';
```

Seata AT 模式工作原理

工作流程示例（一阶段）

```
update product set name = 'GTS' where name = 'TXC';
```

SQL 类型 => UPDATE

表 => product

条件 => where name = 'TXC’

……

解析 SQL

查询前镜像

执行业务 SQL

查询后镜像

插入回滚日志

向 TC 注册分支

本地事务提交

本地事务结果上报TC

Seata AT 模式工作原理

工作流程示例（一阶段）

id	name	since
1	TXC	2014

解析 SQL

查询前镜像

执行业务 SQL

查询后镜像

插入回滚日志

向 TC 注册分支

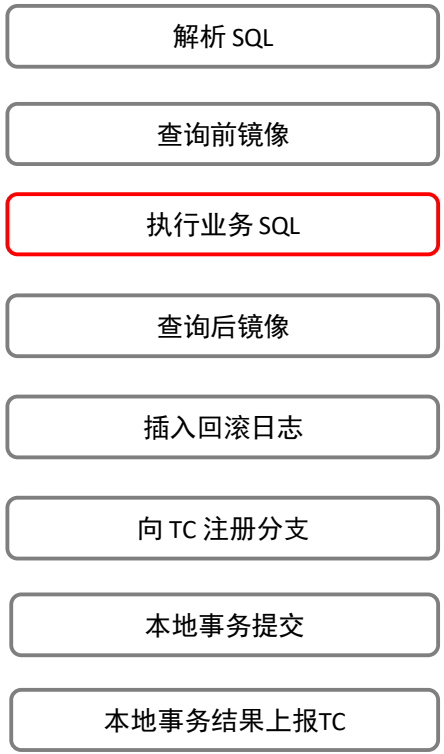
本地事务提交

本地事务结果上报TC

Seata AT 模式工作原理

工作流程示例（一阶段）

```
update product set name = 'GTS' where name = 'TXC';
```



Seata AT 模式工作原理

工作流程示例（一阶段）

id	name	since
1	GTS	2014

解析 SQL

查询前镜像

执行业务 SQL

查询后镜像

插入回滚日志

向 TC 注册分支

本地事务提交

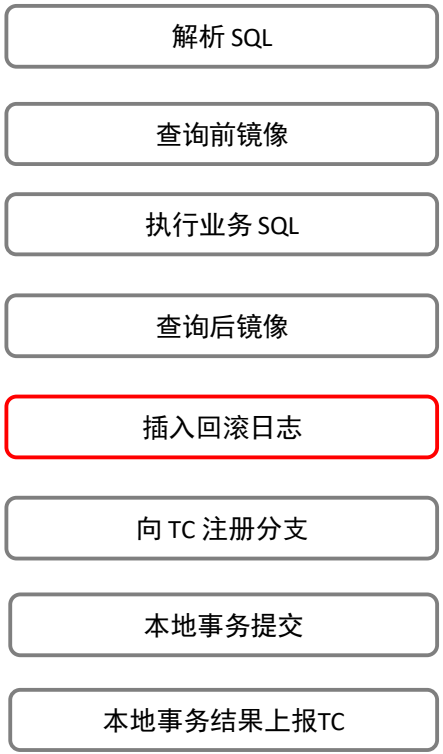
本地事务结果上报TC

Seata AT 模式工作原理

工作流程示例（一阶段）

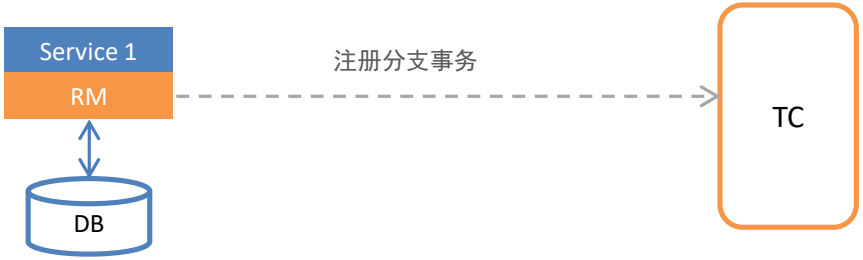
构造回滚日志，插入 UNDO_LOG 表

```
{
        “xid”: “xid:xxx”,
        "branchId": 641789253,
        "undoItems": [{
                "afterImage": {
                        "rows": [{
                                ……
                        }],
                        "tableName": "product"
                },
                "beforeImage": {
                        "rows": [{
                                ……
                        }],
                        "tableName": "product"
                },
                "sqlType": "UPDATE"
        }]
}
```



Seata AT 模式工作原理

工作流程示例（一阶段）



解析 SQL

查询前镜像

执行业务 SQL

查询后镜像

插入回滚日志

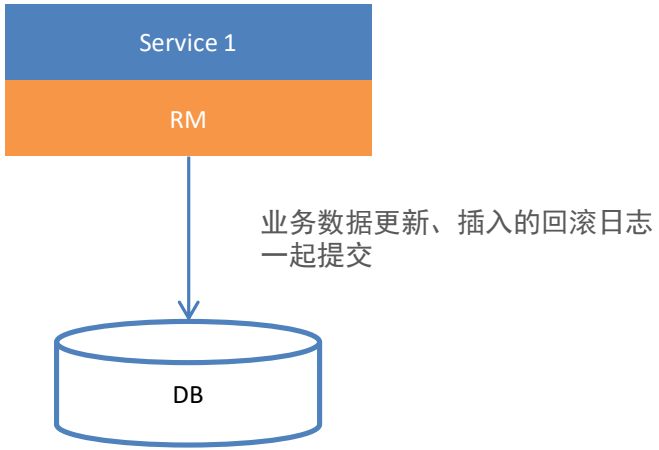
向 TC 注册分支

本地事务提交

本地事务结果上报TC

Seata AT 模式工作原理

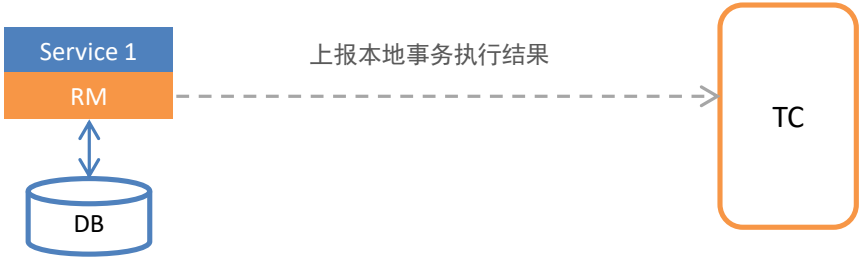
工作流程示例（一阶段）



- 解析 SQL
- 查询前镜像
- 执行业务 SQL
- 查询后镜像
- 插入回滚日志
- 向 TC 注册分支
- 本地事务提交
- 本地事务结果上报TC

Seata AT 模式工作原理

工作流程示例（一阶段）



解析 SQL

查询前镜像

执行业务 SQL

查询后镜像

插入回滚日志

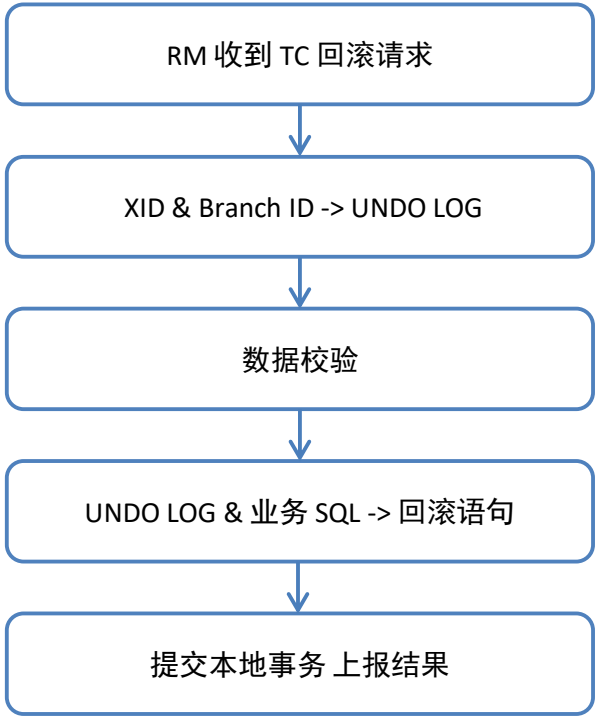
向 TC 注册分支

本地事务提交

本地事务结果上报TC

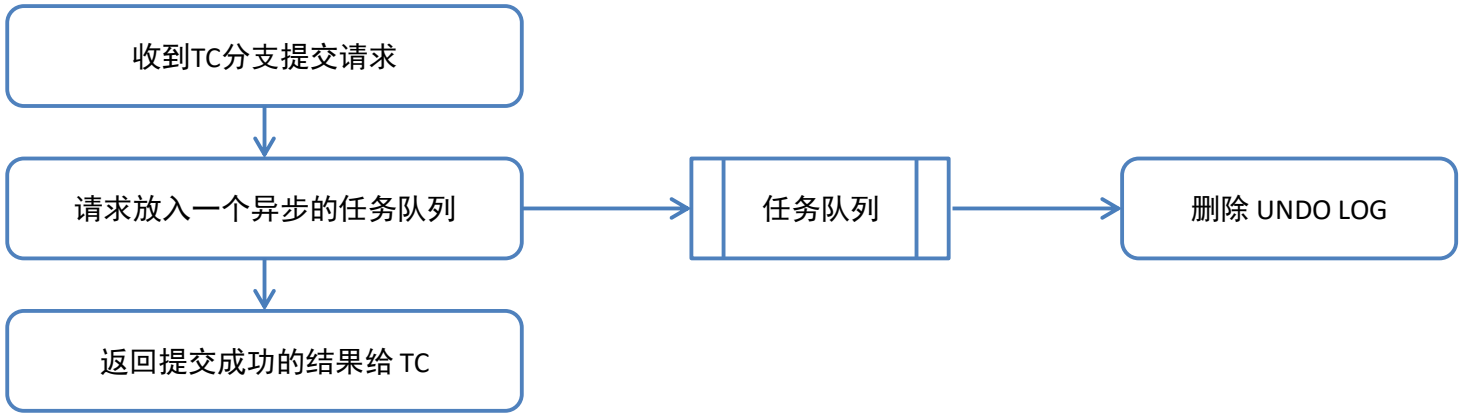
Seata AT 模式工作原理

工作流程示例（二阶段 – 回滚）



Seata AT 模式工作原理

工作流程示例（二阶段 – 提交）



Seata AT 模式工作原理

模式特点

基于 JDBC 驱动**自动化**处理，**方便**

也是**束缚**，项目中如果有服务不使用 JDBC 就无法应用

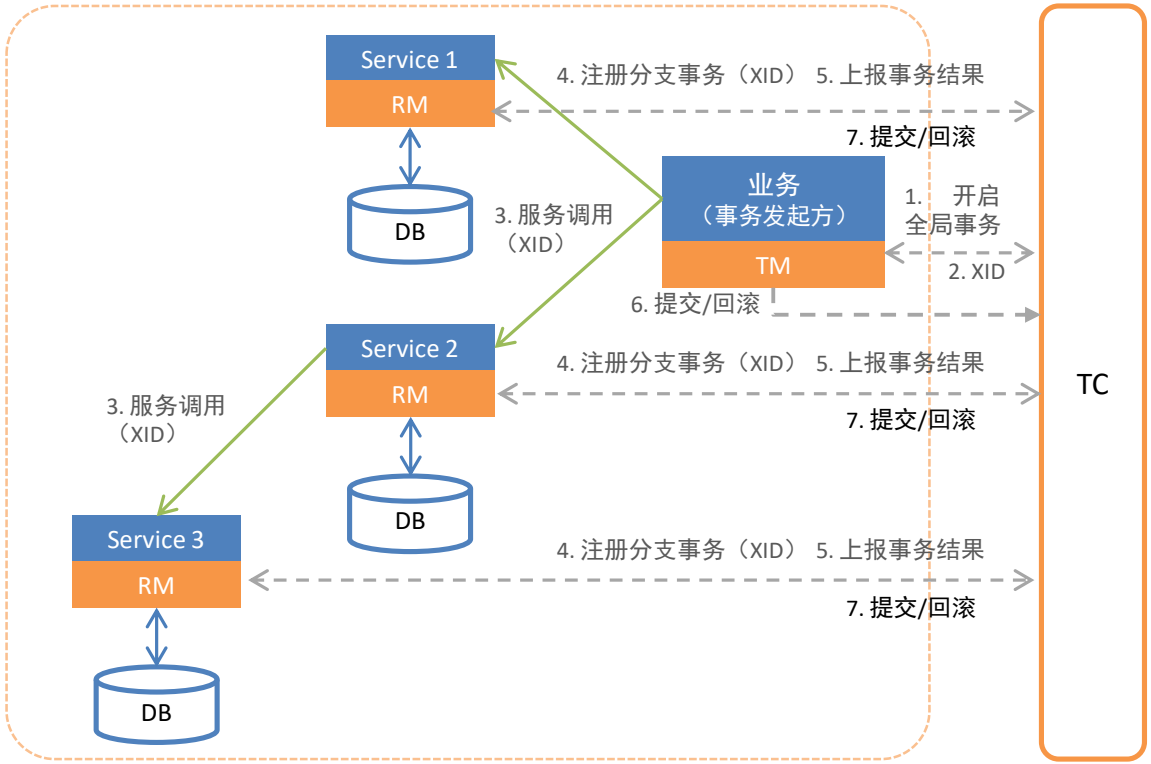
Seata AT 模式工作原理-总结

重难点

1. Seata AT 模式的整体工作流程
2. Seata TA 模式中 RM 的工作流程

Seata AT 模式工作原理-总结

整体流程



Seata AT 模式工作原理-总结

重难点

1. Seata AT 模式的整体工作流程
2. Seata TA 模式中 RM 的工作流程

Seata AT 模式工作原理-总结

一阶段

- 解析 SQL
- 查询前镜像
- 执行业务 SQL
- 查询后镜像
- 插入回滚日志
- 向 TC 注册分支
- 本地事务提交
- 本地事务结果上报TC

二阶段

回滚

- 收到TC分支回滚请求
- XID & Branch ID -> UNDO LOG
- 数据校验
- UNDO LOG & 业务 SQL -> 回滚语句
- 提交本地事务 上报结果

提交

- 收到TC分支提交请求
- 请求放入一个异步任务的队列
- 返回提交成功的结果给 TC

Seata AT 模式工作原理-总结

重难点

1. Seata AT 模式的整体工作流程
2. Seata TA 模式中 RM 的工作流程

下节

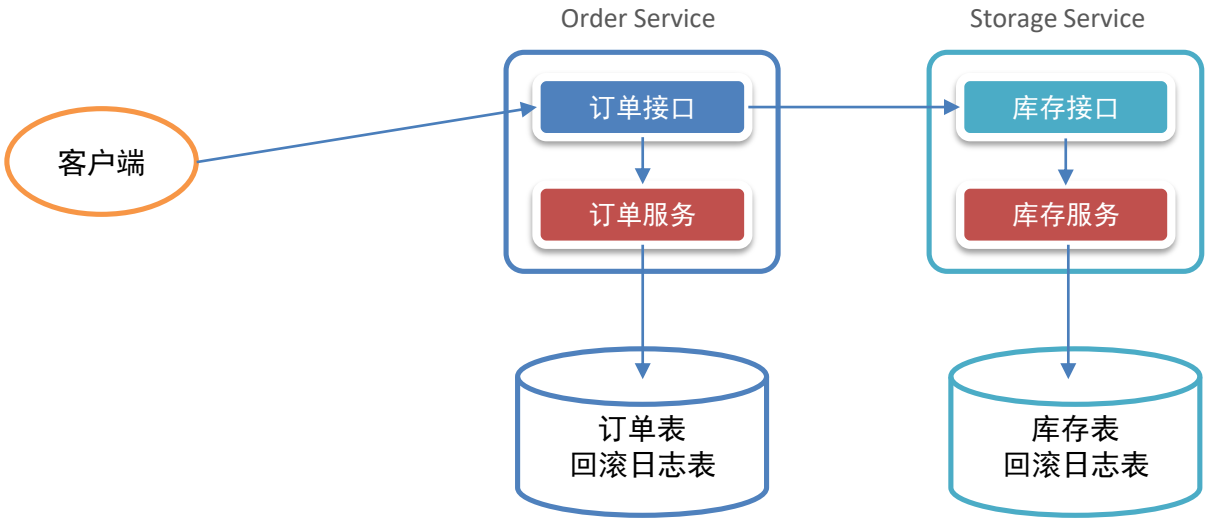
Seata AT 模式开发实践

目录 Contents

Seata AT 模式实践

小节导学

上节已经了解了 Seata AT 模式的工作原理，本节目标就是开发实践 AT 模式。



Seata AT 模式实践

实践流程

Seata AT 模式实践

实践步骤

1. 下载启动 Seata Server

```
https://github.com/seata/seata/releases
```

下载 seata-server-1.2.0.zip

解压后执行：

```
sh bin/seata-server.sh -p 8091 -h 127.0.0.1 -m file
```

Seata AT 模式实践

实践步骤

2. 创建回滚日志表 UNDO_LOG

```
CREATE TABLE `undo_log`
(
  `id`            BIGINT(20)   NOT NULL AUTO_INCREMENT,
  `branch_id`     BIGINT(20)   NOT NULL,
  `xid`           VARCHAR(100) NOT NULL,
  `context`       VARCHAR(128) NOT NULL,
  `rollback_info` LONGBLOB     NOT NULL,
  `log_status`    INT(11)      NOT NULL,
  `log_created`   DATETIME     NOT NULL,
  `log_modified`  DATETIME     NOT NULL,
  `ext`           VARCHAR(100) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB
  AUTO_INCREMENT = 1
  DEFAULT CHARSET = utf8;
```

Seata AT 模式实践

实践步骤

3. 添加 Seata 依赖

```
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-alibaba-seata</artifactId>
    <version>2.1.0.RELEASE</version>
</dependency>
<dependency>
    <groupId>io.seata</groupId>
    <artifactId>seata-all</artifactId>
    <version>1.1.0</version>
</dependency>
```

Seata AT 模式实践

实践步骤

4. 添加 Seata 配置

```
# application.yaml
spring:
  cloud:
    alibaba:
      seata:
        tx-service-group: my_test_tx_group
# file.conf
service {
  vgroupMapping.my_test_tx_group = "default”
 ……
}
```

Seata AT 模式实践

实践步骤

5. 改造代码

```
# 改用 Seata DataSource
@Primary
@Bean("dataSource")
public DataSourceProxy dataSourceProxy(DataSource druidDataSource) {
    return new DataSourceProxy(druidDataSource);
}
# 添加全局事务注解
@GlobalTransactional
@GetMapping("/order")
public String hi(String code, Integer count) {
    ……
}
```

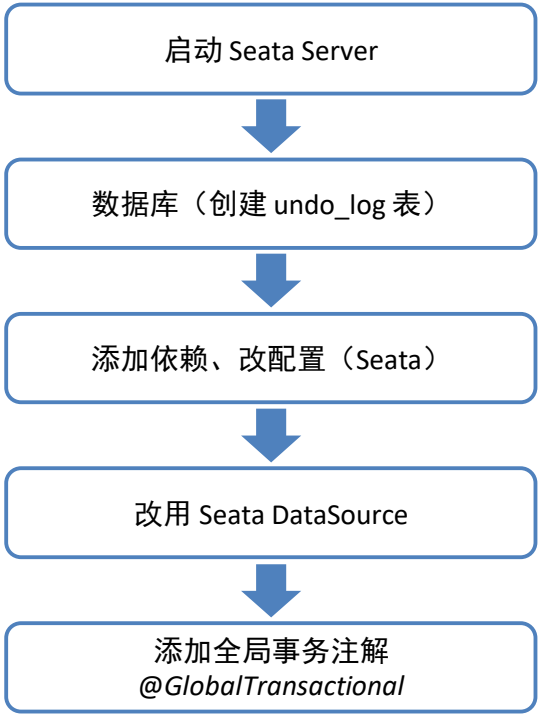
Seata AT 模式实践 – 总结

重难点

1. Seata TA 模式的开发流程

Seata AT 模式实践 – 总结

流程



Seata AT 模式实践 – 总结

重难点

1. Seata TA 模式的开发流程

下节

Seata TCC 模式工作原理

TCC 与 AT 的主要区别是什么?

目录 Contents

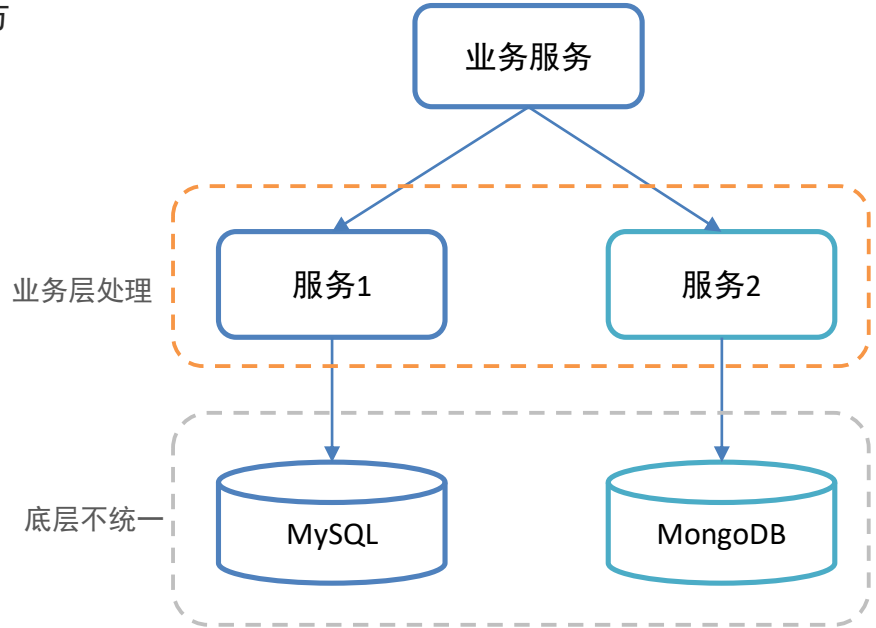
Seata TCC 模式工作原理

小节导学

AT 模式是 Seata 在技术底层帮我们处理数据库，使用方便的同时，也带来了局限性，因为只能使用 JDBC。

如果不同服务使用了不同的数据库，例如 服务1 使用了 MySQL、服务2 使用了 MongoDB，如何处理?

这类场景就适合使用 TCC 模式。



Seata TCC 模式工作原理

TCC 两阶段设计

TCC 全称：

Try-Confirm-Cancel（尝试 - 确认 - 取消）

■ 第一阶段：Try

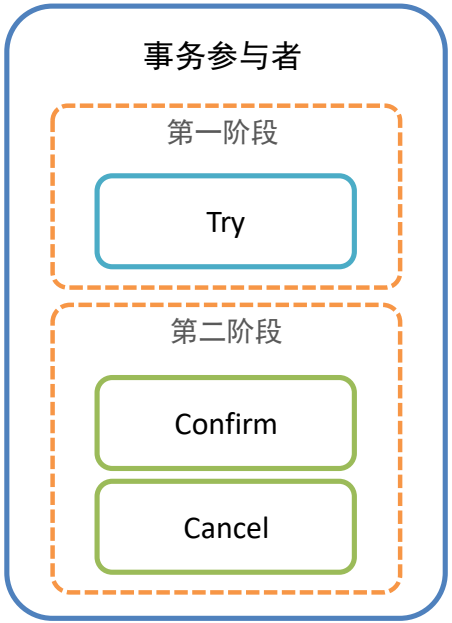
尝试，检查是否可以满足操作所需要的条件，如果满足就预留资源

■ 第二阶段：Confirm/Cancel

Confirm 确认，真正的操作资源

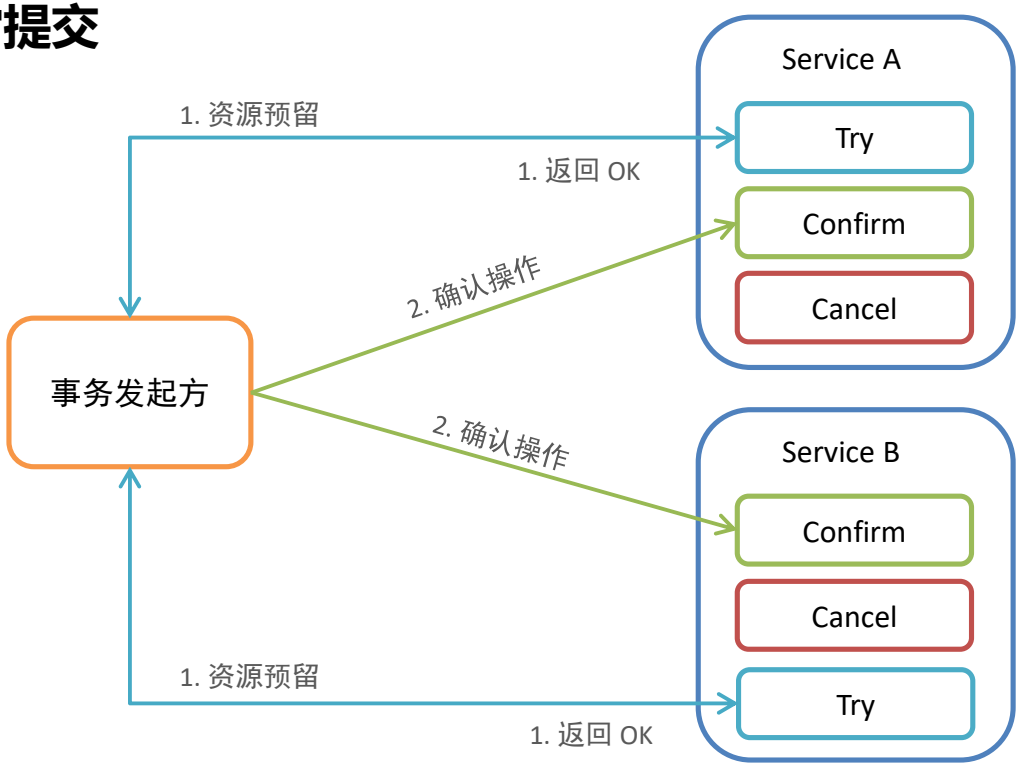
Cancel 取消，释放预留资源

每个事务参与者都需要实现这3个操作。



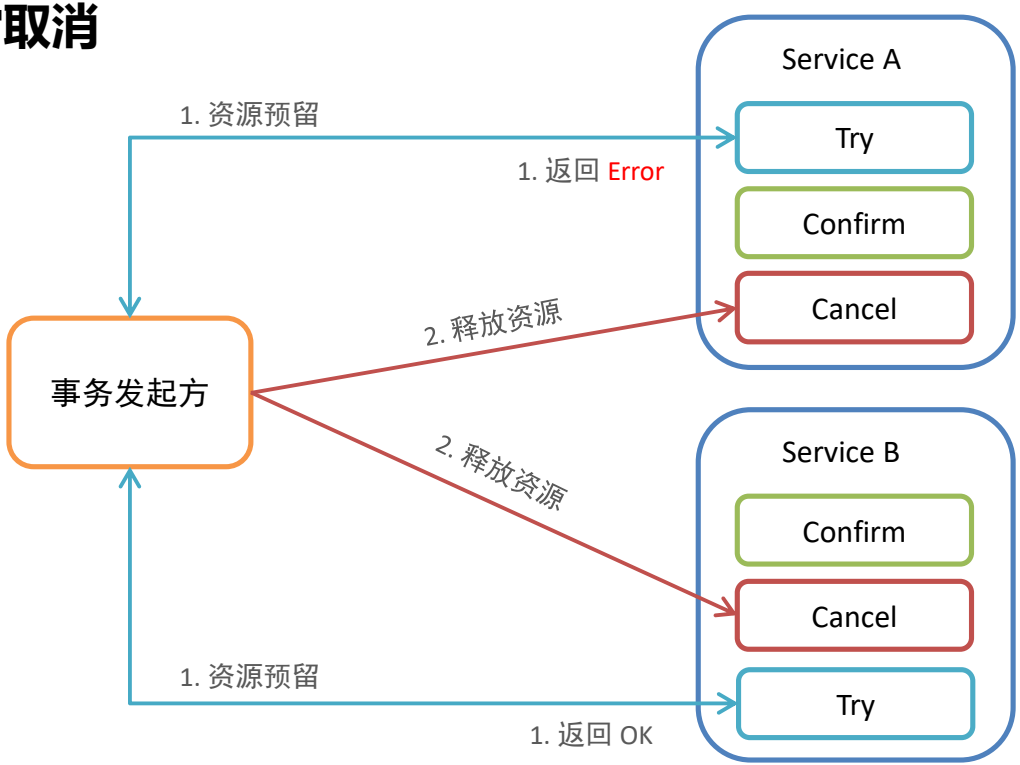
Seata TCC 模式工作原理

TCC - 正常提交



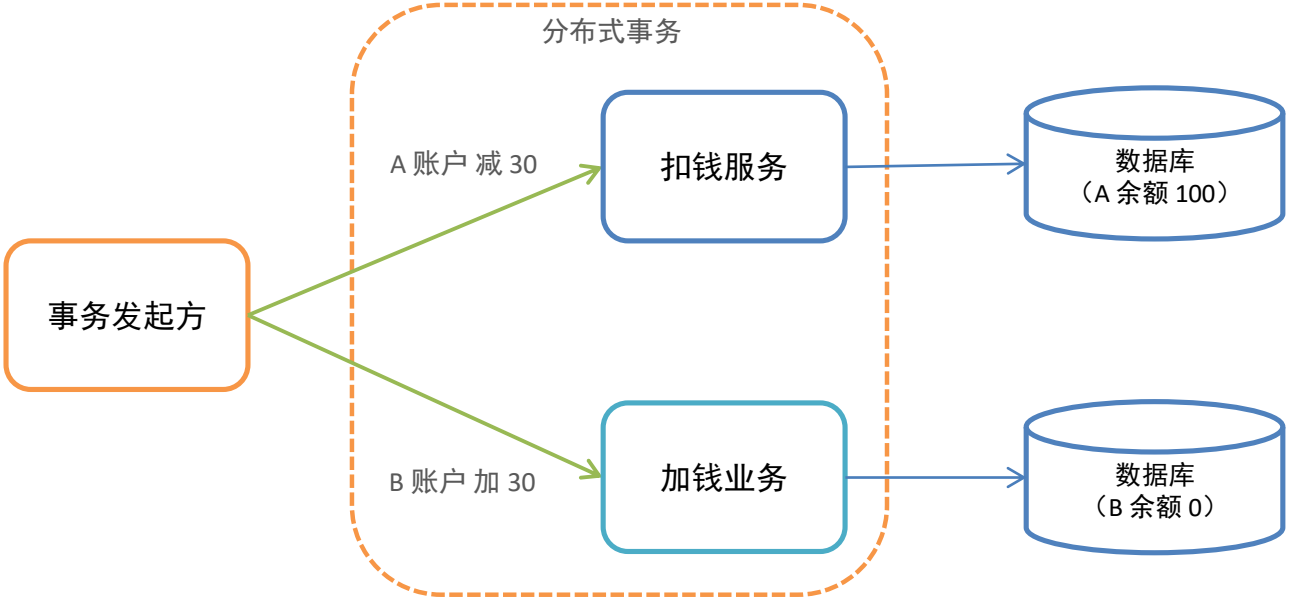
Seata TCC 模式工作原理

TCC - 异常取消



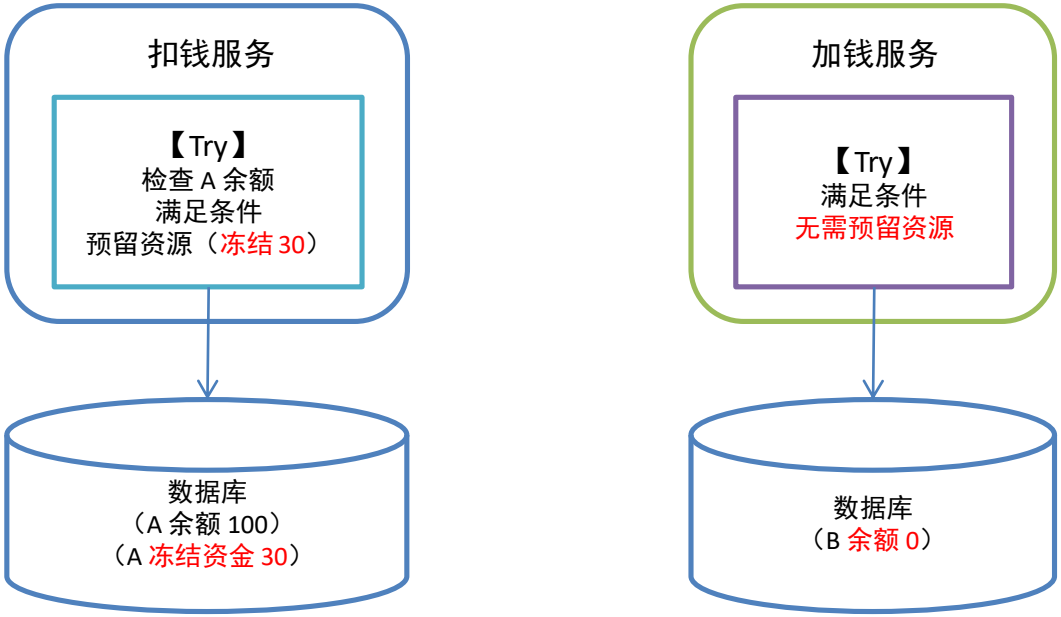
Seata TCC 模式工作原理

TCC 工作流程示例 - 需求



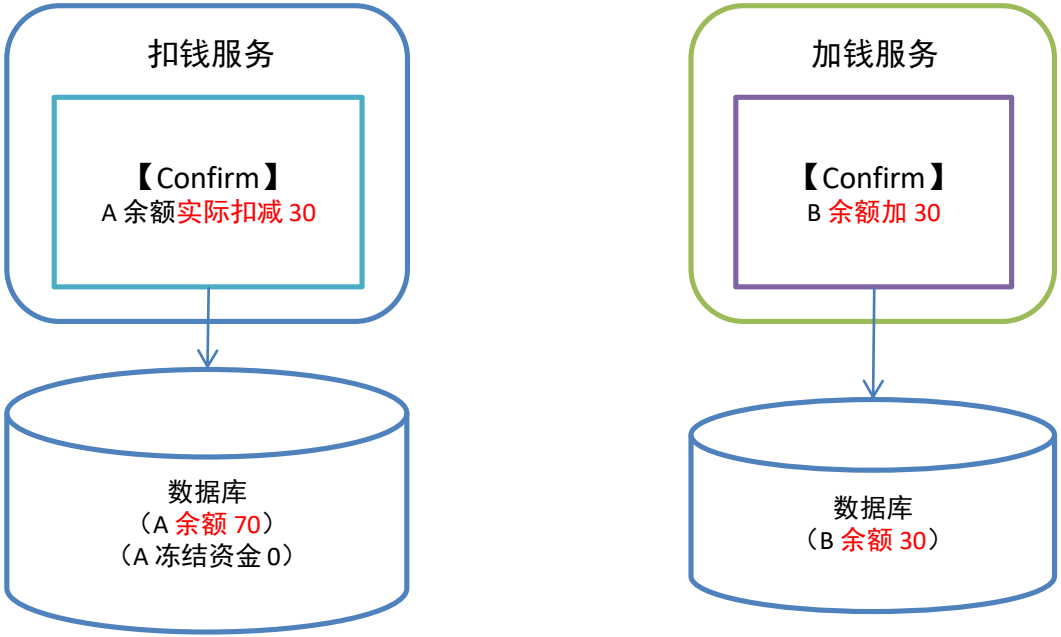
Seata TCC 模式工作原理

TCC 工作流程示例 - 第一阶段 Try



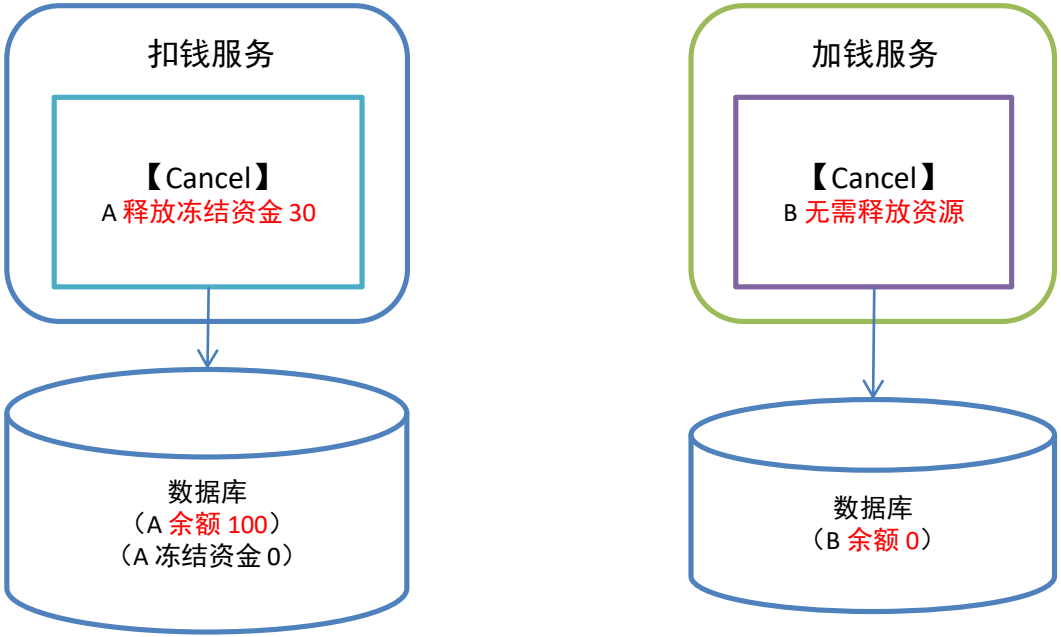
Seata TCC 模式工作原理

TCC 工作流程示例 - 第二阶段 Confirm



Seata TCC 模式工作原理

TCC 工作流程示例 - 第二阶段 Cancel



Seata TCC 模式工作原理

TCC 异常情况

空回滚

Try 没执行
Cancel 执行
资源误释放

幂等

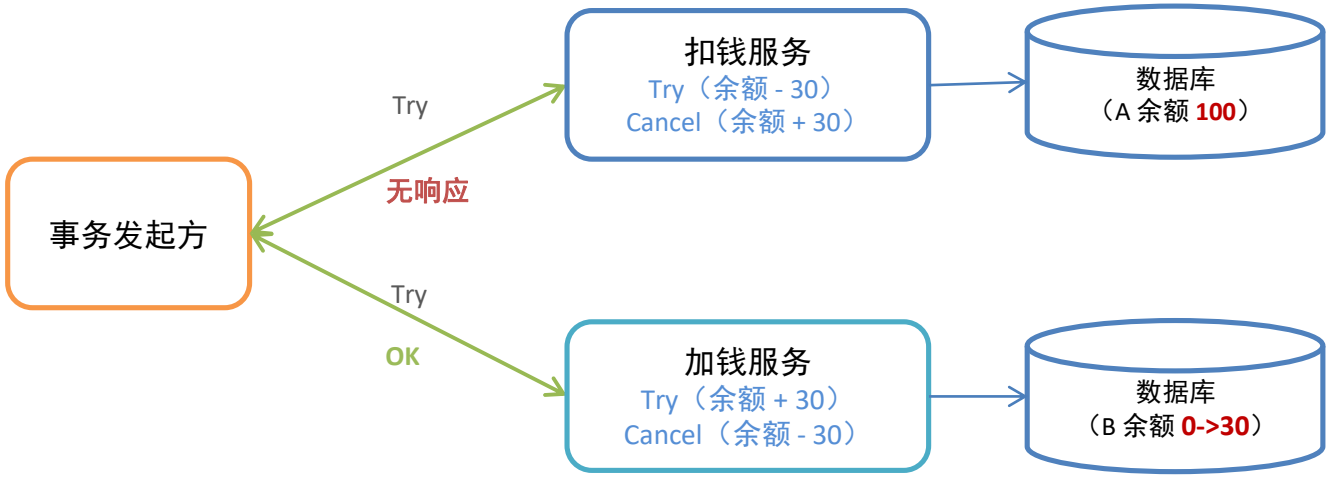
Confirm / Cancel
重复调用
资源操作重复

空悬挂

先调用 Cancel
后执行的 Try
资源无法释放

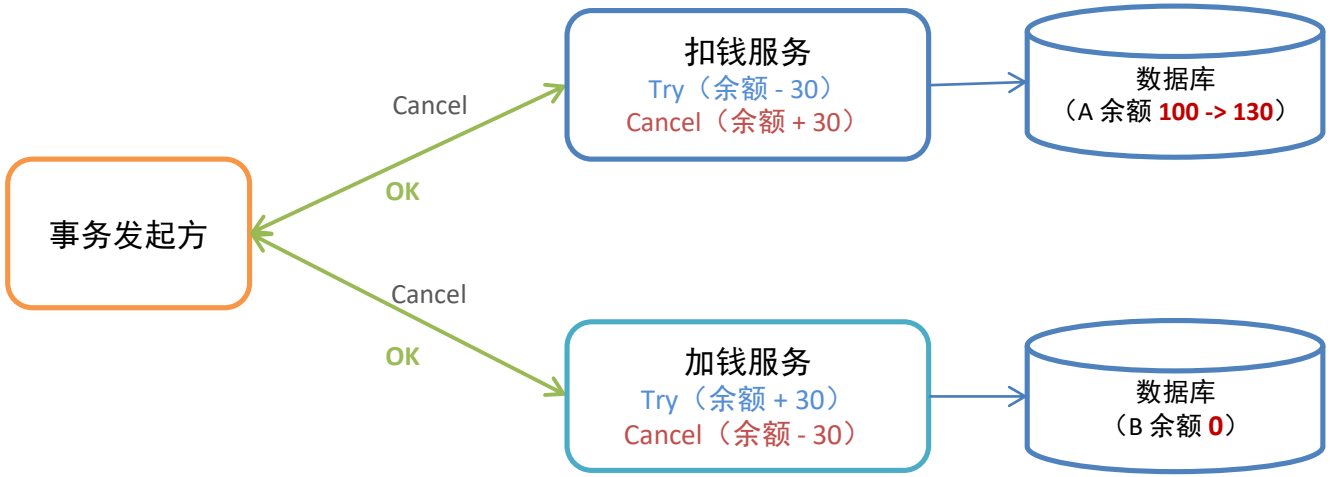
Seata TCC 模式工作原理

TCC 异常情况 – 空回滚



Seata TCC 模式工作原理

TCC 异常情况 – 空回滚



Seata TCC 模式工作原理

TCC 异常情况 – 空回滚

原因

Try 超时，分布式事务回滚，触发 Cancel

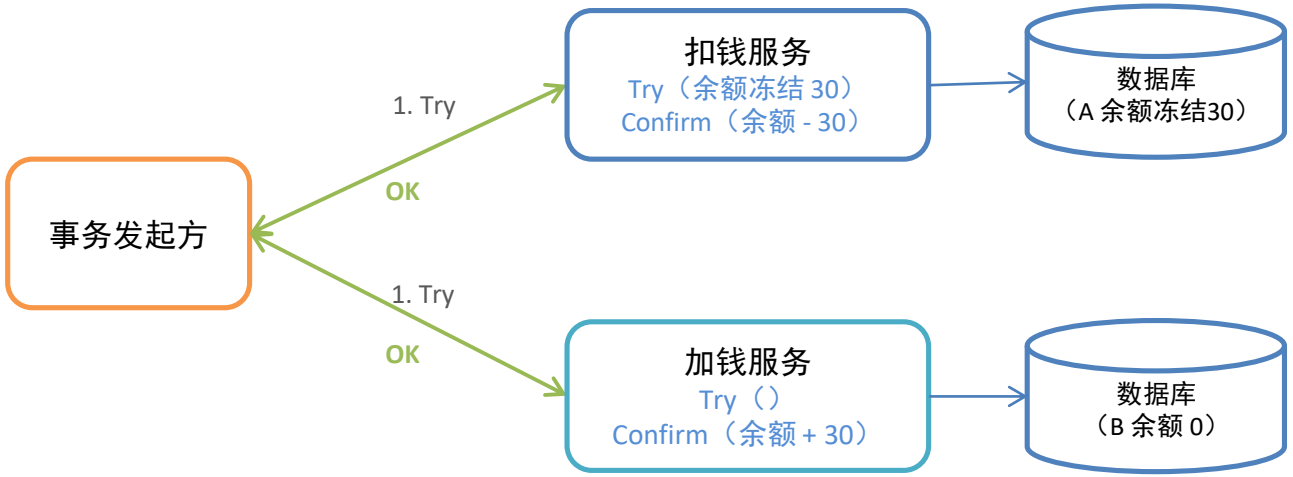
未收到 Try，收到了 Cancel

解决方法

Cancel 执行之前，判断**事务执行状态**，如果没有执行 Try 就不能执行自己的业务逻辑

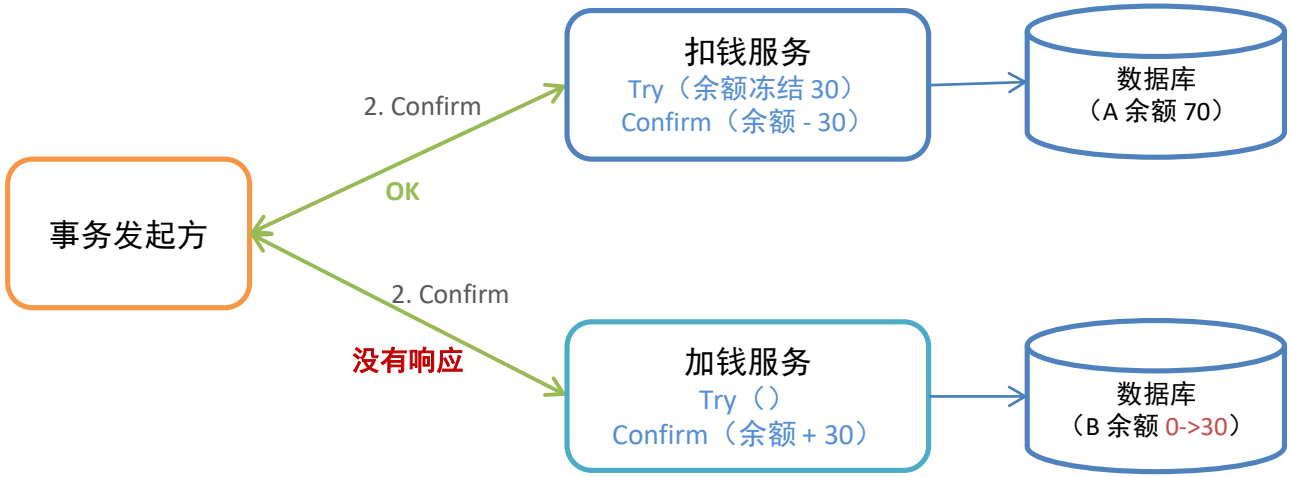
Seata TCC 模式工作原理

TCC 异常情况 – 幂等



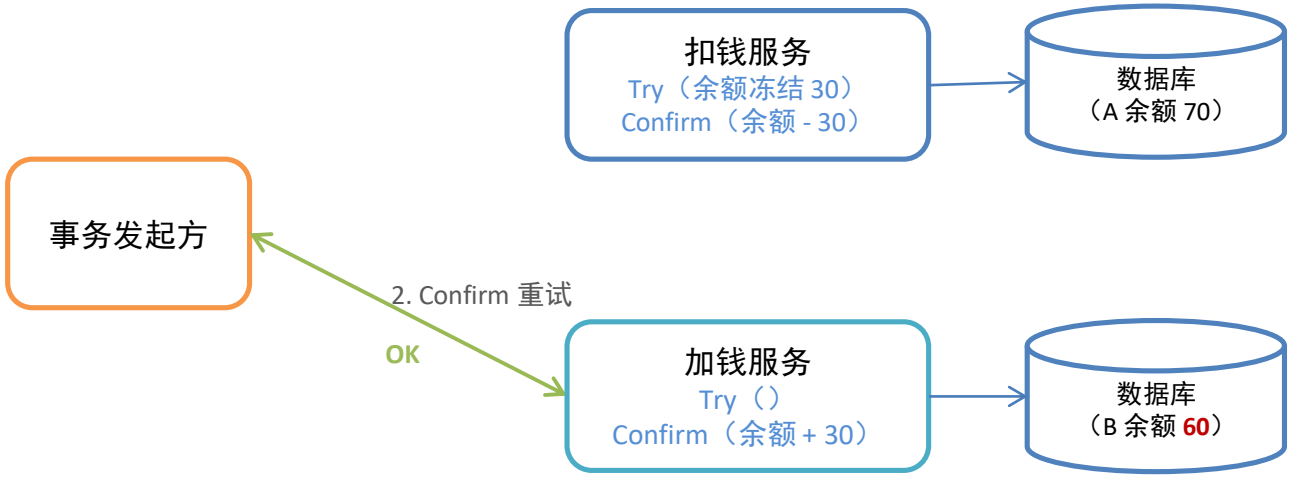
Seata TCC 模式工作原理

TCC 异常情况 – 幂等



Seata TCC 模式工作原理

TCC 异常情况 – 幂等



Seata TCC 模式工作原理

TCC 异常情况 – 幂等

原因

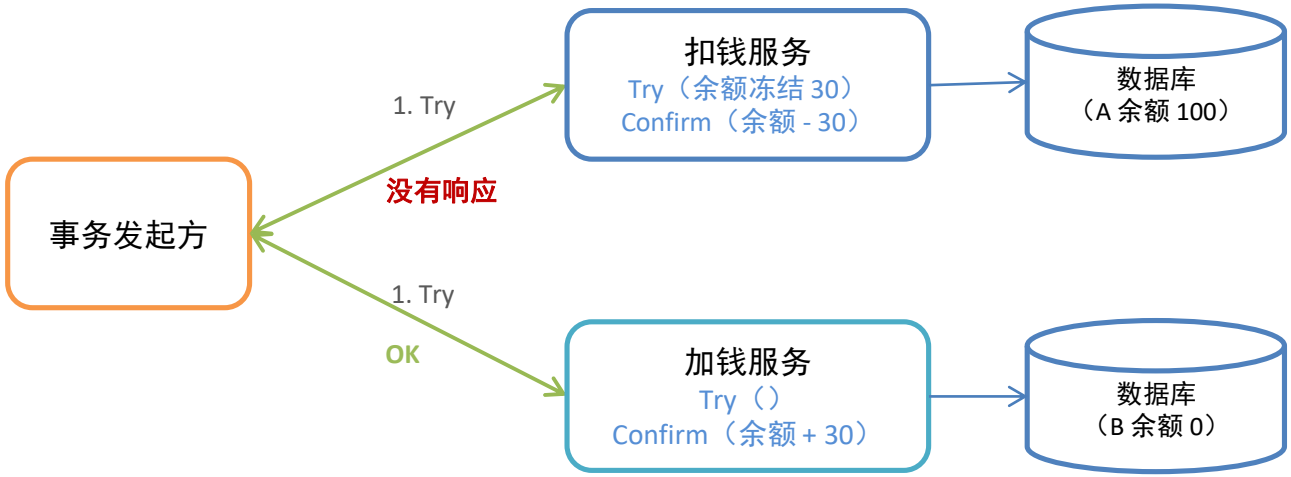
网络异常导致重复调用二阶段方法

解决方法

执行前先判断**事务执行状态**，如果是已经提交或已经取消的状态，不再执行

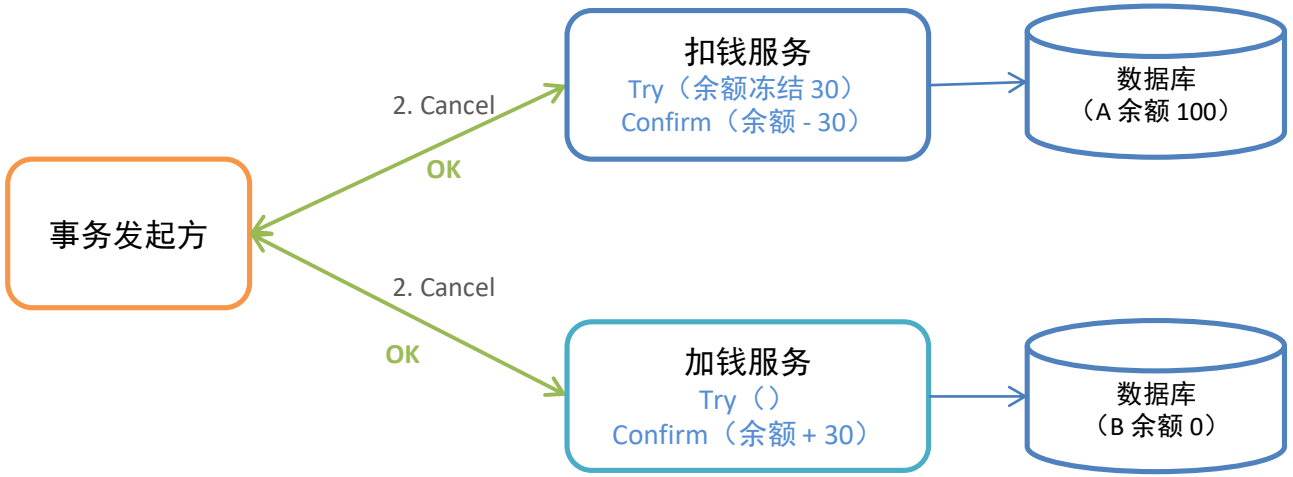
Seata TCC 模式工作原理

TCC 异常情况 – 空悬挂



Seata TCC 模式工作原理

TCC 异常情况 – 空悬挂



Seata TCC 模式工作原理

TCC 异常情况 – 空悬挂

Seata TCC 模式工作原理

TCC 异常情况 – 空悬挂

原因

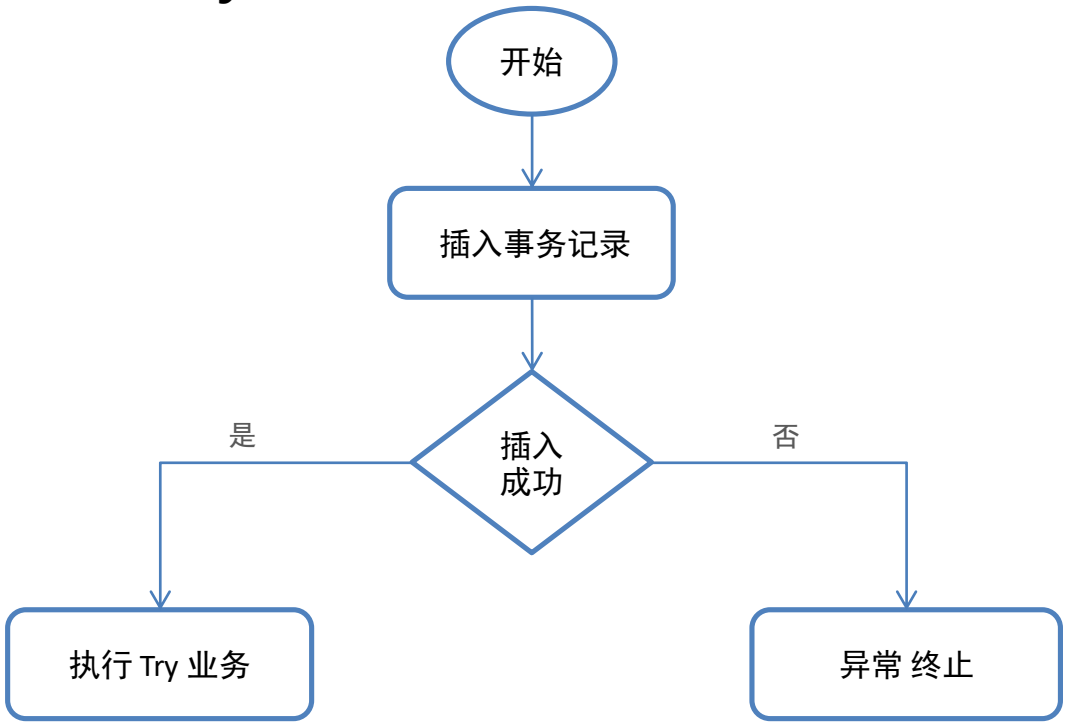
网络原因导致 Try 迟到，被认为失败，执行了二阶段，之后 Try 的请求才到

解决方法

Try 执行前判断**事务执行状态**，二阶段发现无记录时插入记录，一阶段执行时检查记录是否存在

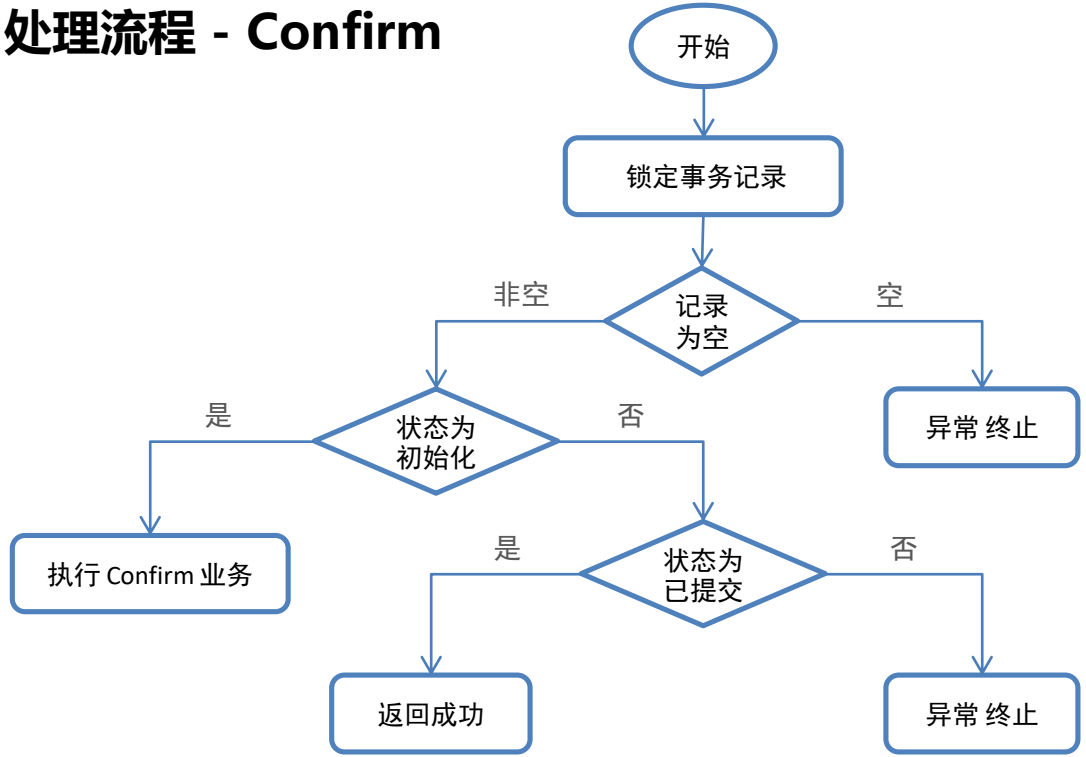
Seata TCC 模式工作原理

TCC 处理流程 - Try



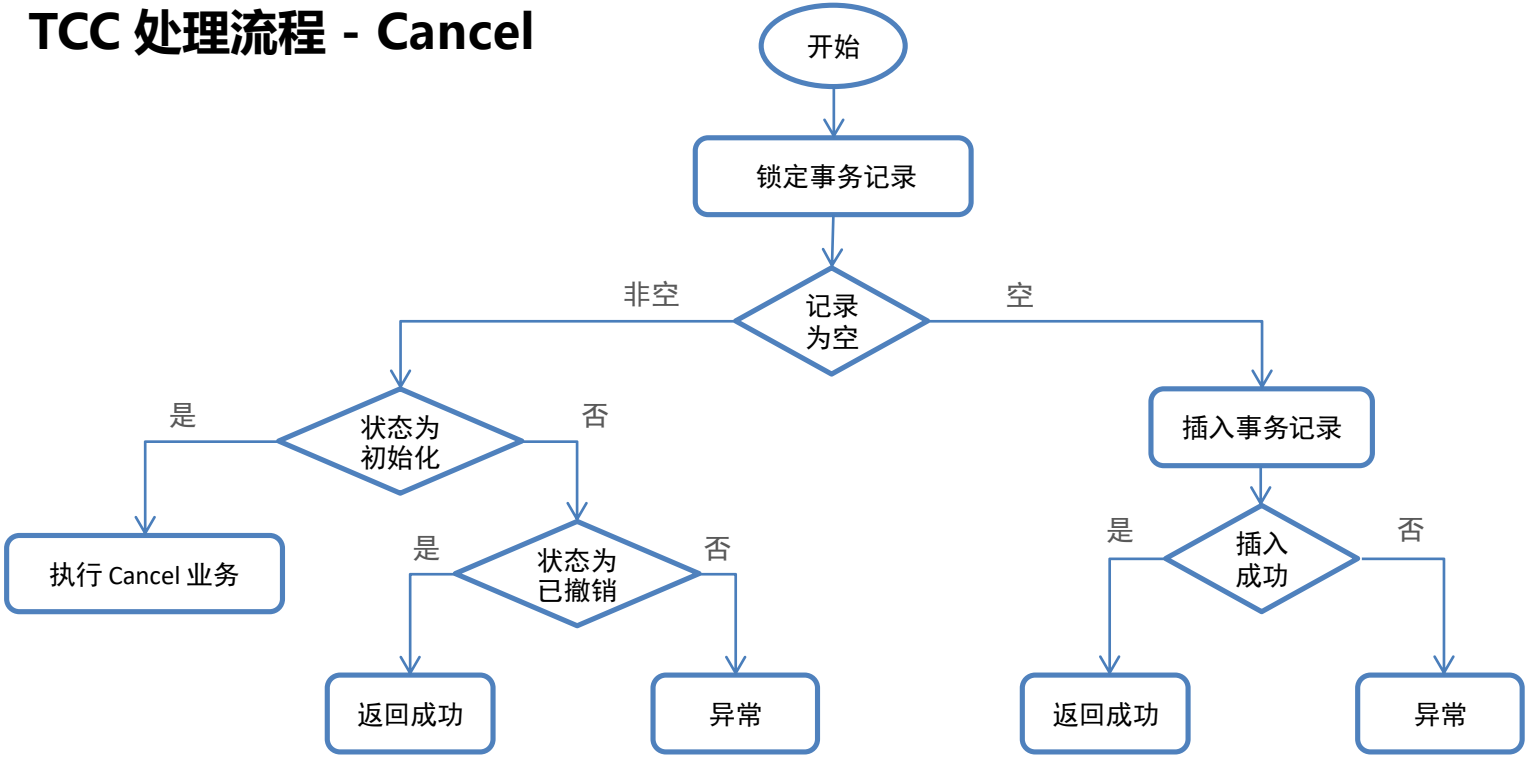
Seata TCC 模式工作原理

TCC 处理流程 - Confirm



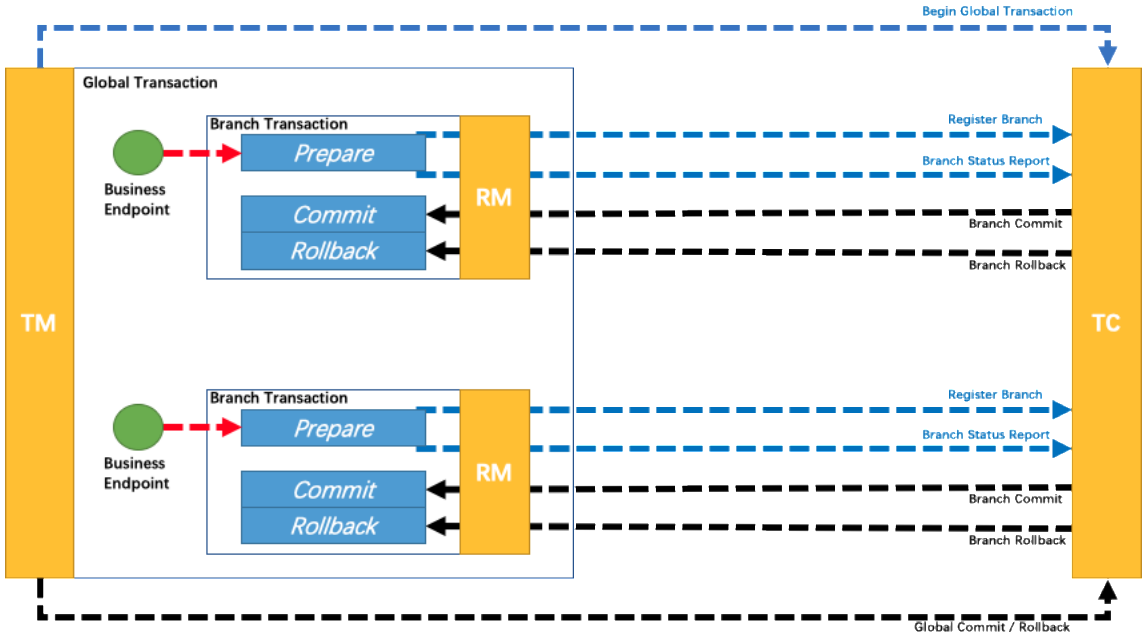
Seata TCC 模式工作原理

TCC 处理流程 - Cancel



Seata TCC 模式工作原理

Seata TCC 工作流程



Seata TCC 模式工作原理

TCC 模式特点

- 工作在业务层，与底层技术无关，技术自由度较高
- 每个事务相关服务都需要实现 Try、Confirm、Cancel 这3个逻辑，开发量较大

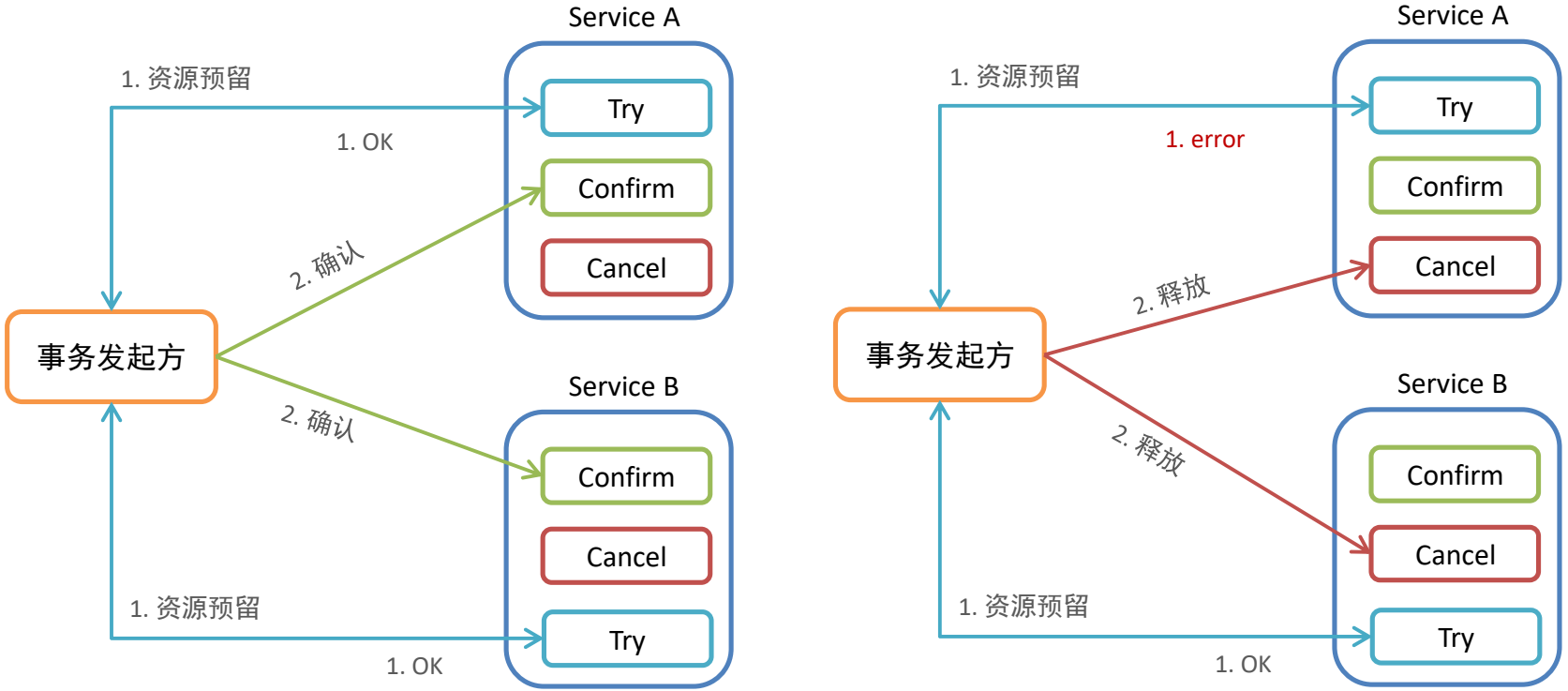
Seata TCC 模式工作原理 – 总结

重难点

1. TCC 模式的二阶段提交
2. TCC 模式的典型问题
3. Seata TCC 模式的工作流程

Seata TCC 模式工作原理 – 总结

TCC 两阶段设计



Seata TCC 模式工作原理 – 总结

重难点

1. TCC 模式的二阶段提交
2. TCC 模式的典型问题
3. Seata TCC 模式的工作流程

Seata TCC 模式工作原理 – 总结

TCC 典型问题

空回滚

Try 没执行
Cancel 执行
资源误释放

幂等

Confirm / Cancel
重复调用
资源操作重复

空悬挂

先调用 Cancel
后执行的 Try
资源无法释放

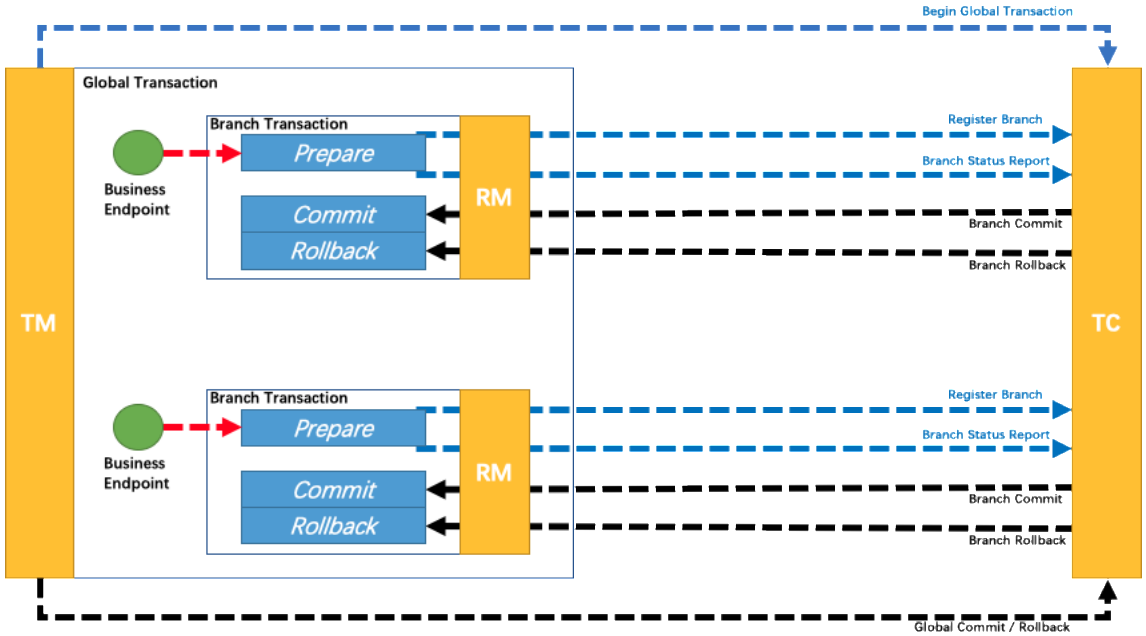
Seata TCC 模式工作原理 – 总结

重难点

1. TCC 模式的二阶段提交
2. TCC 模式的典型问题
3. Seata TCC 模式的工作流程

Seata TCC 模式工作原理 – 总结

Seata TCC 工作流程



Seata TCC 模式工作原理 – 总结

重难点

1. TCC 模式的二阶段提交
2. TCC 模式的典型问题
3. Seata TCC 模式的工作流程

下节

Seata TCC 模式**开发实践**

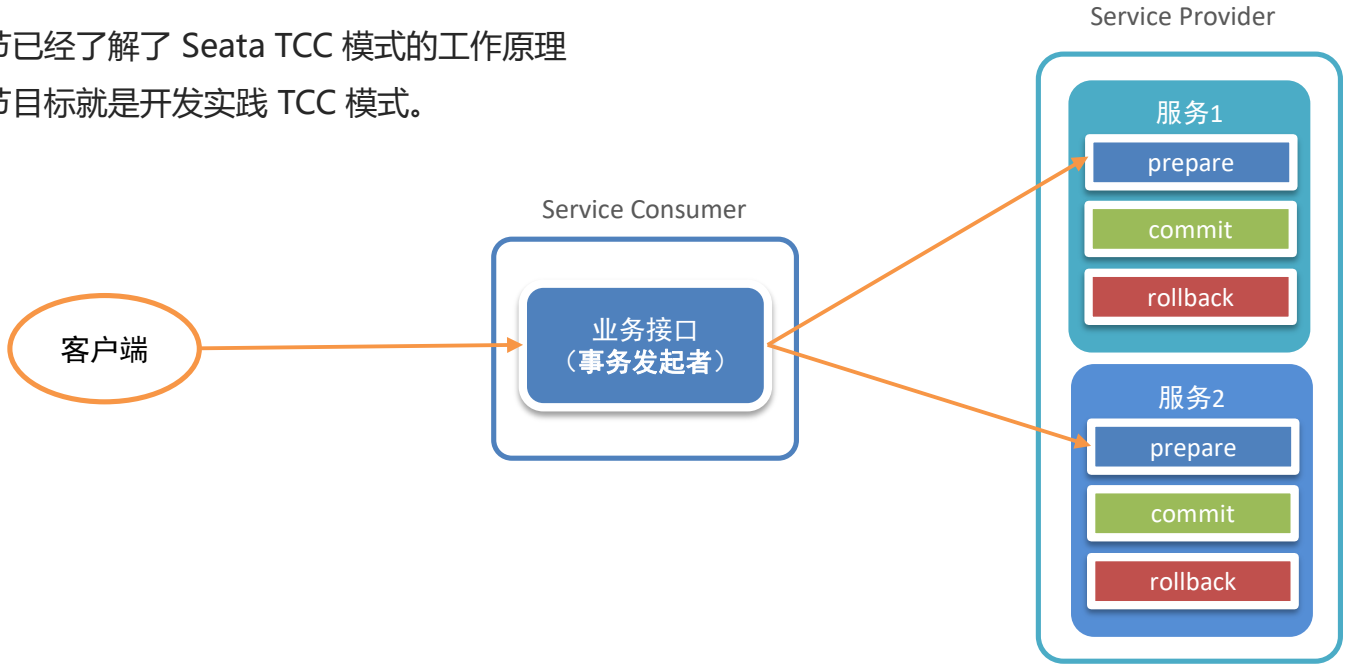
目录 Contents

Seata TCC 模式实践

小节导学

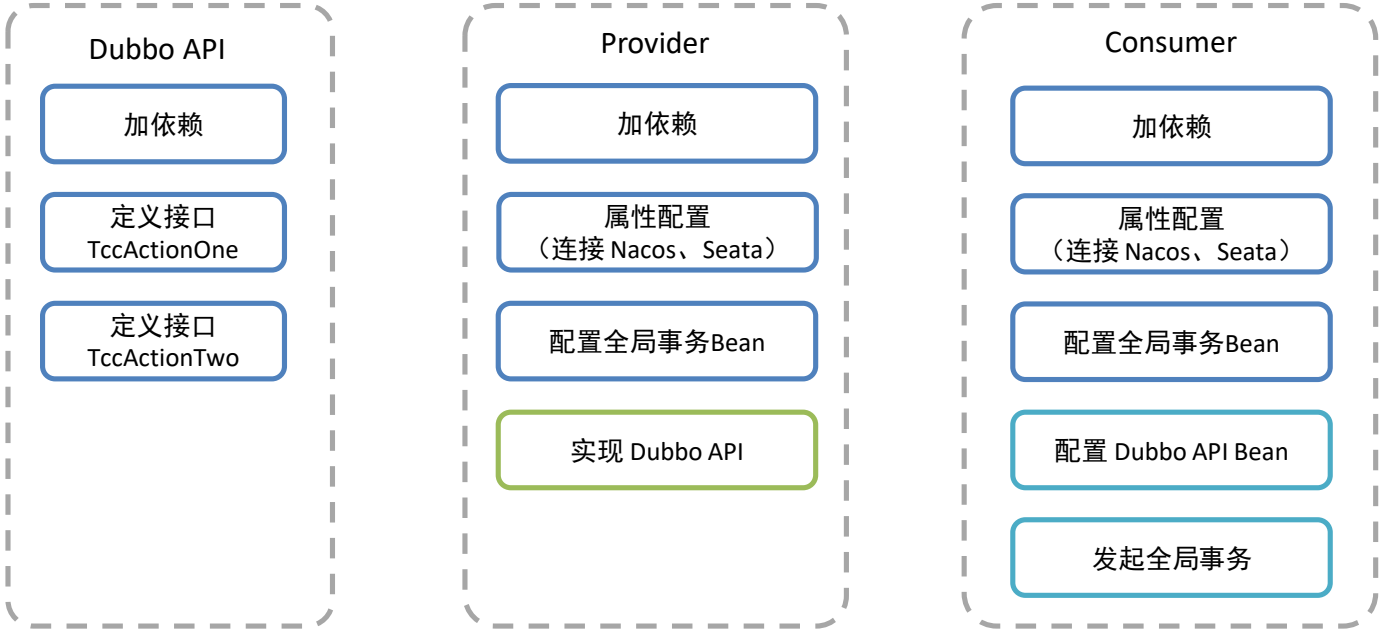
上节已经了解了 Seata TCC 模式的工作原理

本节目标就是开发实践 TCC 模式。



Seata TCC 模式实践

实践流程



Seata TCC 模式实践

关键步骤 - Dubbo API

- 需要使用 Seata 提供的 `@TwoPhaseBusinessAction` 注解，指定 commit、rollback 方法
- `@BusinessActionContextParameter` 定义的参数可以自动传递到 commit、rollback 方法

```
public interface TccActionOne {
    @TwoPhaseBusinessAction(name = "DubboTccActionOne", commitMethod = "commit", rollbackMethod =
"rollback")
    public boolean prepare(BusinessActionContext actionContext,
                           @BusinessActionContextParameter(paramName = "a") int a);
    public boolean commit(BusinessActionContext actionContext);
    public boolean rollback(BusinessActionContext actionContext);
}
```

Seata TCC 模式实践

关键步骤 – 配置 Seata 全局事务扫描器

Provider 和 Consumer 都需要配置全局事务扫描器，否则不能调用。

```
@Bean
public GlobalTransactionScanner globalTransactionScanner() {
    return new GlobalTransactionScanner("seata-tcc-consumer", "my_test_tx_group");
}
```

Seata TCC 模式实践

关键步骤 – Consumer 配置服务接口 Bean

Consumer 需要配置好引用的服务接口Bean，不能在 Controller 中使用时再引用，因为 Seata 也要用。

```
@Bean
ReferenceBean<TccActionOne> tccActionOneReferenceBean(){
    ReferenceBean<TccActionOne> tccActionOneReferenceBean = new ReferenceBean<>();
    tccActionOneReferenceBean.setInterface(TccActionOne.class);
    tccActionOneReferenceBean.setCheck(false);
    tccActionOneReferenceBean.setTimeout(3000);
    tccActionOneReferenceBean.setLazy(true);
    return tccActionOneReferenceBean;
}
```

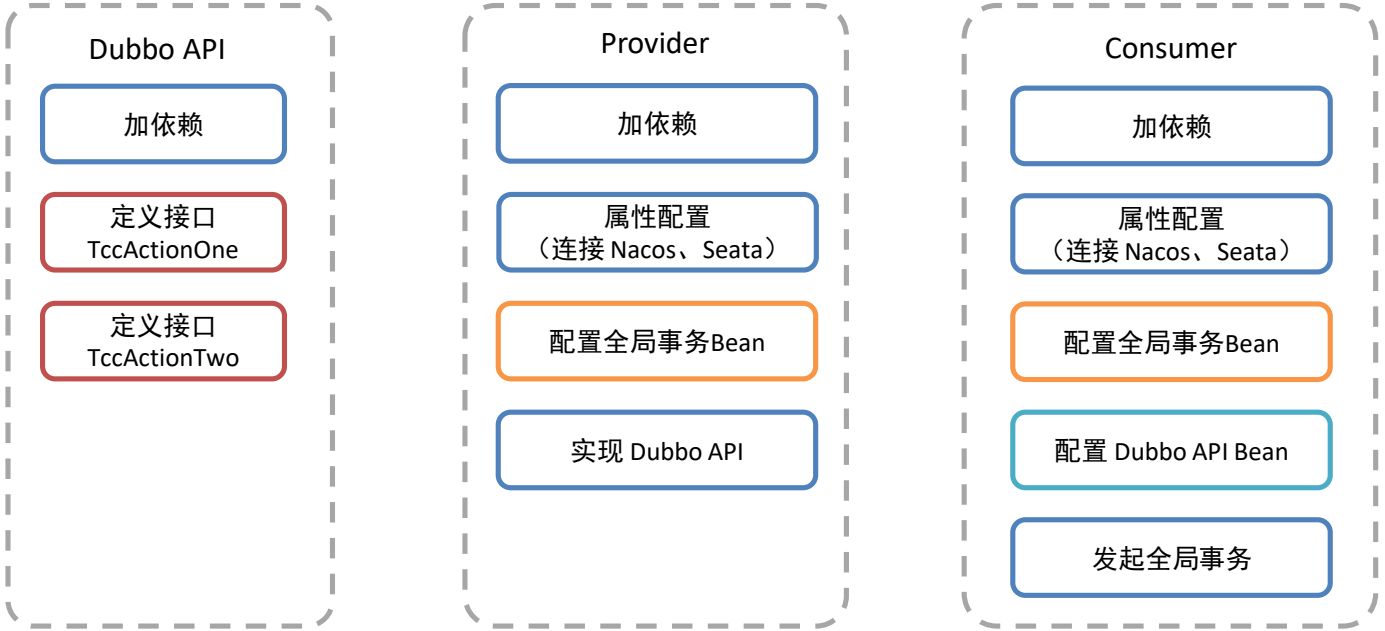
Seata TCC 模式实践 – 总结

重难点

1. Seata TCC 模式的开发流程

Seata TCC 模式实践 – 总结

实践流程



Seata TCC 模式实践 – 总结

重难点

1. Seata TCC 模式的开发流程

下节

Seata **Saga** 模式的**工作原理**

与 AT 模式、TCC 模式 完全不同的处理思路

目录 Contents

Seata Saga 模式工作原理

小节导学

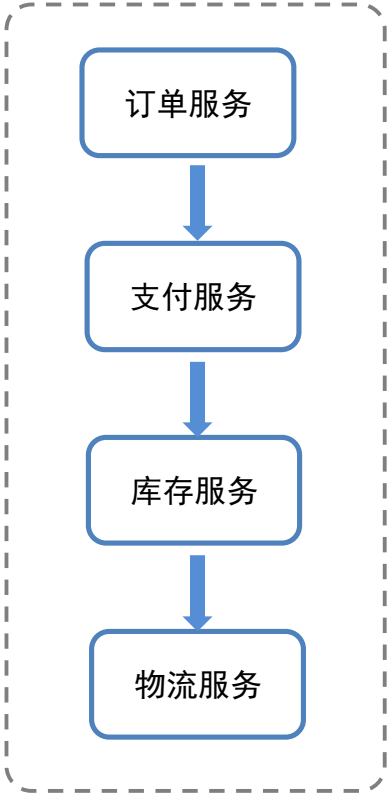
Saga 适用于**长事务**场景。

什么是**长事务**?

- 业务链长
- 执行时间长

AT 模式不能用，全都执行下来的话时间太长，数据锁不能长时间使用，而且可能使用了不同的数据库。

TCC 模式需要每个服务实现3个操作，如果是旧系统，可能无法改造。



Seata Saga 模式工作原理

Saga 设计

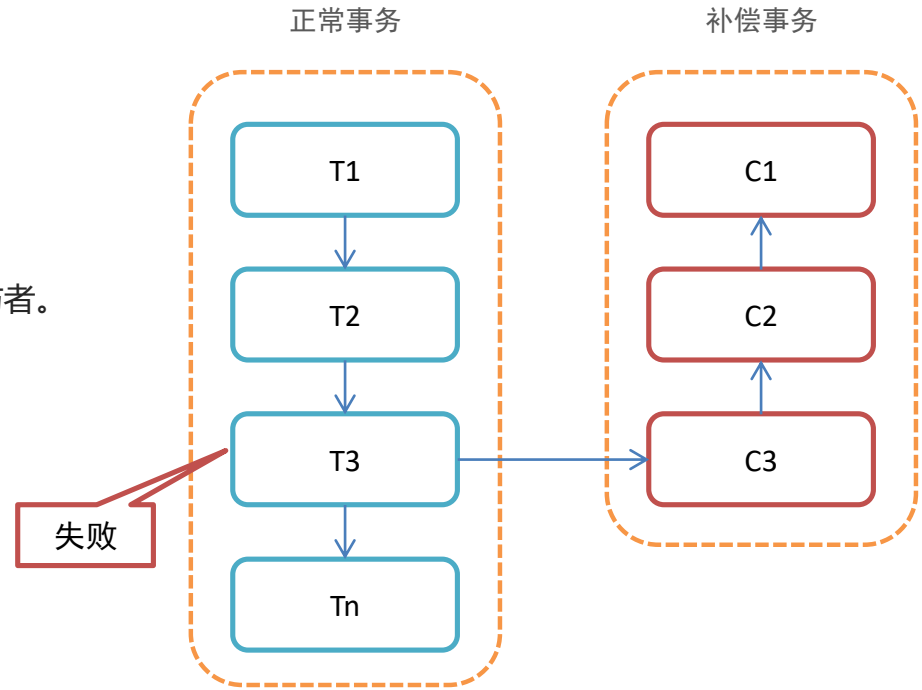
Saga 模式是长事务解决方案。

业务流程中每个参与者都提交本地事务

- 都成功，则全局成功。
- 某一个参与者失败，则补偿前面已经成功的参与者。

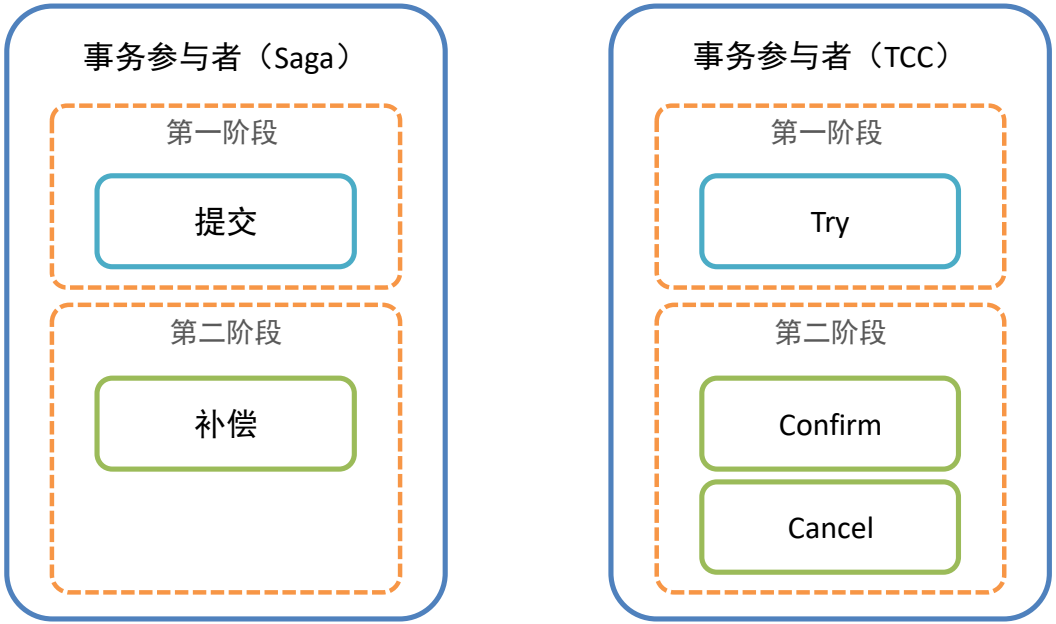
每个事务参与者需要开发2个方法：

- 一阶段正向服务（正常操作）
- 二阶段补偿服务（回滚操作）



Seata Saga 模式工作原理

Saga vs TCC



Seata Saga 模式工作原理

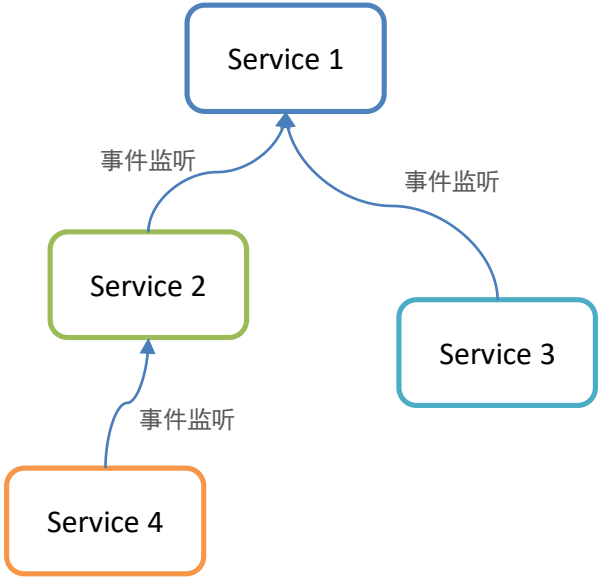
Saga 实现方式 – 事件

服务1 执行一个事务，然后发布一个事件。

该事件被一个或多个服务进行监听，这些服务再执行本地事务并发布（或不发布）新的事件。

实现简单，容易理解

如果参与者多了以后，整体关系就混乱了，可能循环监听



Seata Saga 模式工作原理

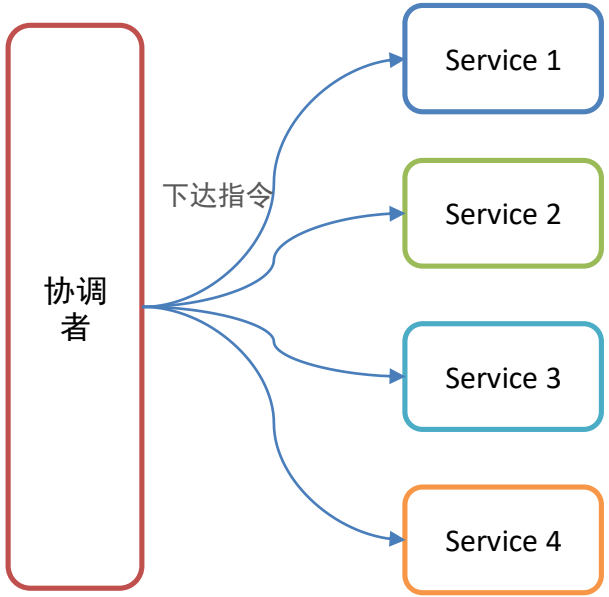
Saga 实现方式 – 协调者

由一个中央协调者来命令各个服务做什么

每个服务执行正常业务方法，还是执行回滚方法，都由协调者决定

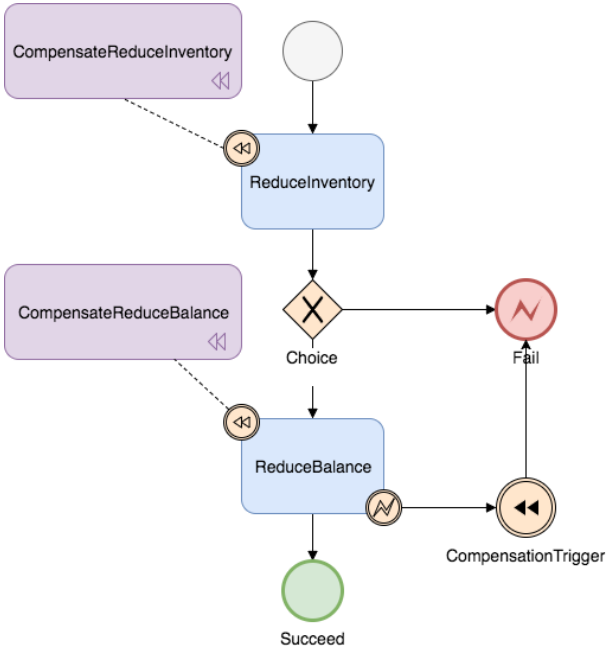
一个服务执行完成后，下一个应该是哪个服务执行，也由协调者决定

协调者就是**总指挥**，有**操作手册**，按照说明来下达指令



Seata Saga 模式工作原理

Seata Saga 实现方式 -基于状态机引擎来实现

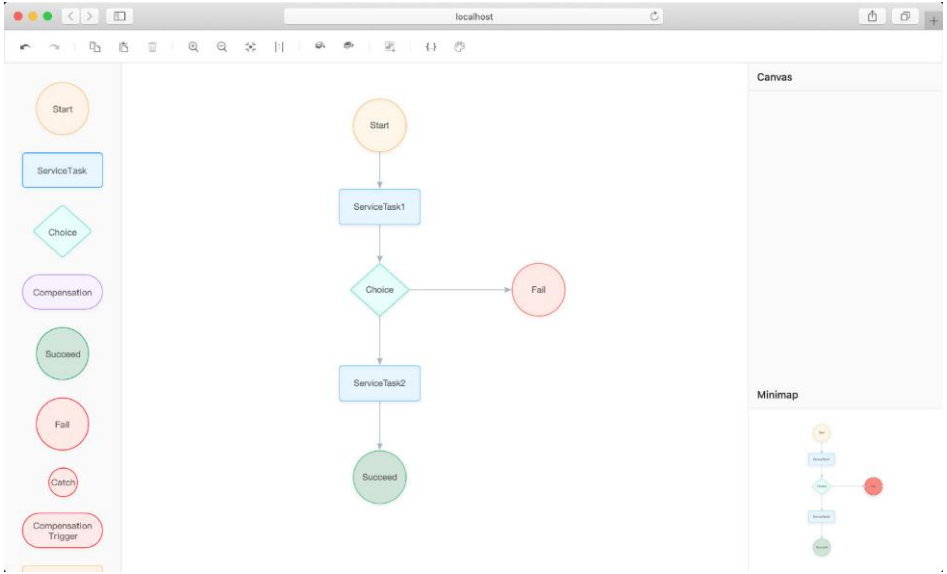



```
...
    "ReduceInventory": {
        "Type": "ServiceTask",
        "ServiceName": "inventoryAction",
        "ServiceMethod": "reduce",
        "CompensateState": "CompensateReduceInventory",
        "Next": "ChoiceState",
        "Input": [...],
        "Output": {...},
        "Status": {
            "#root == true": "SU", ...
        }
    },
...
```

Seata Saga 模式工作原理

Saga 状态机设计器

```
$ git clone
    https://github.com/seata/seata.git
$ cd saga/saga-statemachine-designer
$ npm install
$ npm start
```



Seata Saga 模式工作原理 – 总结

重难点

1. Saga 模式的思路
2. Seata Saga 模式的实现方式

Seata Saga 模式工作原理 – 总结

Saga 实现思路

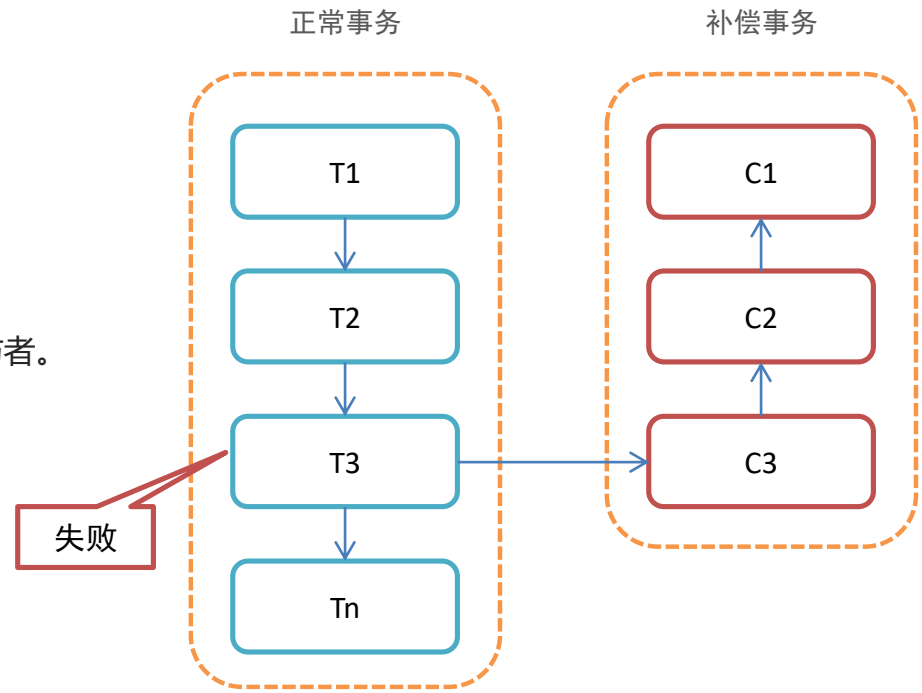
Saga 适用于长事务场景。

每个事务参与者都提交本地事务

- 都成功，则全局成功。
- 某一个参与者失败，则补偿前面已经成功的参与者。

每个事务参与者需要开发2个方法：

- 正常操作
- 回滚操作

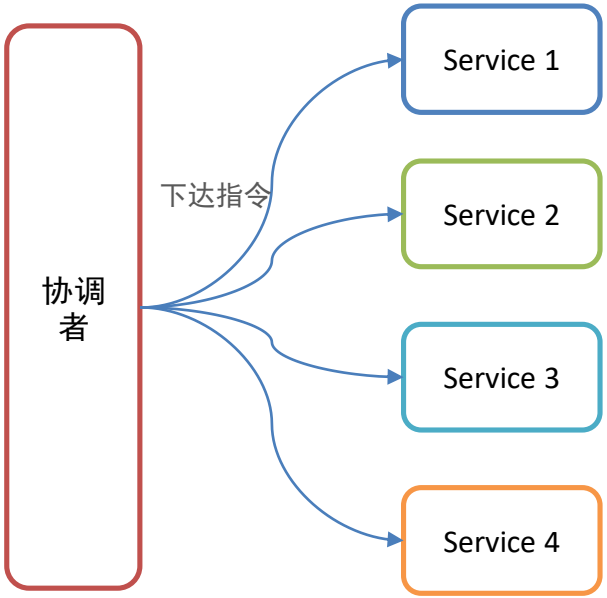


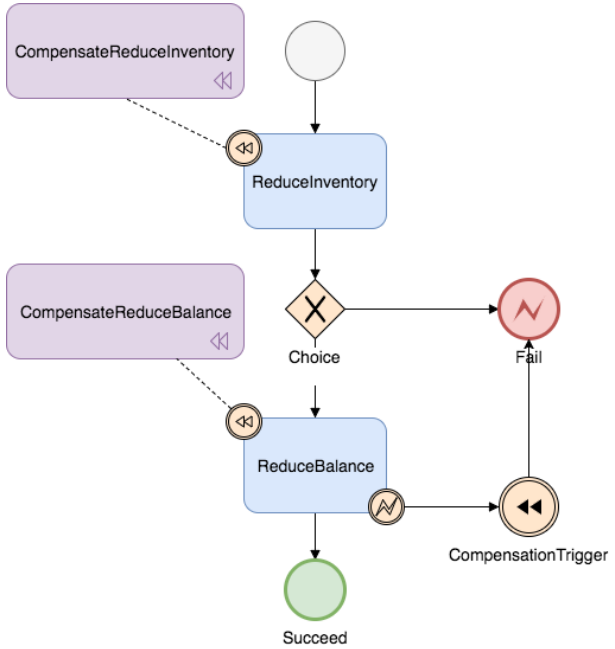
Seata Saga 模式工作原理 – 总结

重难点

1. Saga 模式的思路
2. Seata Saga 模式的实现方式

Seata Saga 模式工作原理 – 总结





Seata Saga 模式工作原理 – 总结

重难点

1. Saga 模式的思路
2. Seata Saga 模式的实现方式

下节

Seata Saga 模式**开发实践**

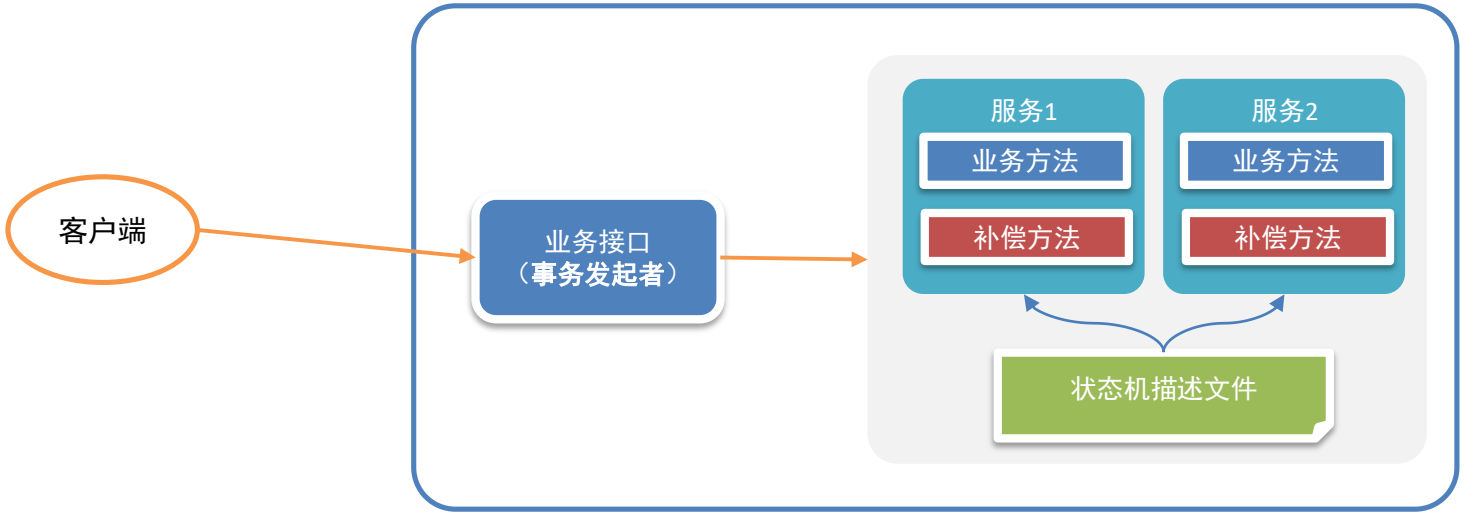
目录 Contents

Seata Saga 模式实践

小节导学

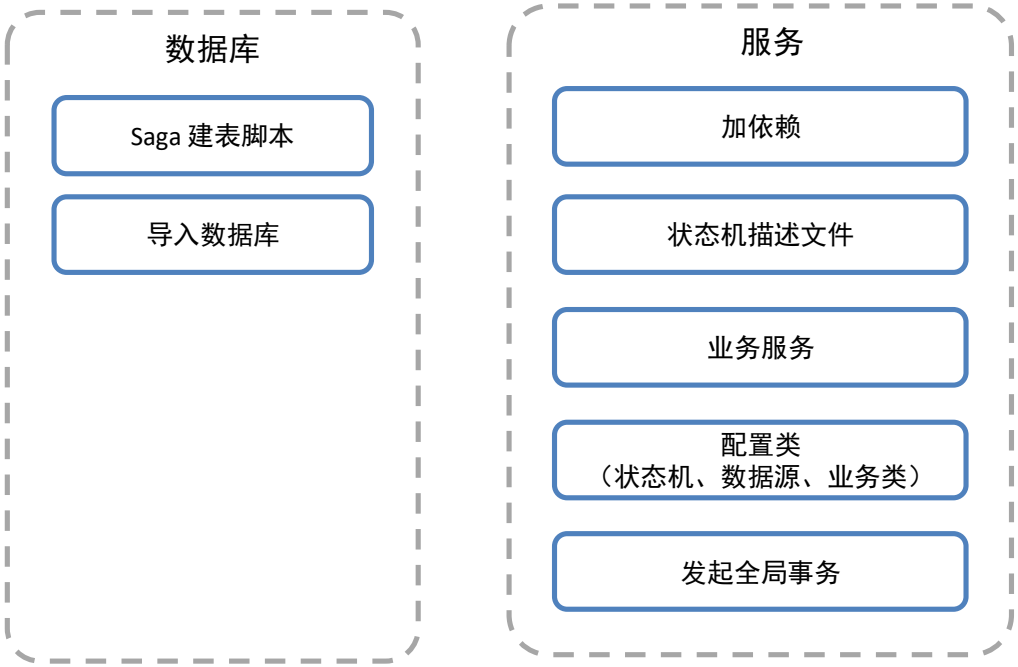
上节已经了解了 Seata Saga 模式的工作原理

本节目标就是开发实践 Saga 模式。



Seata Saga 模式实践

实践流程



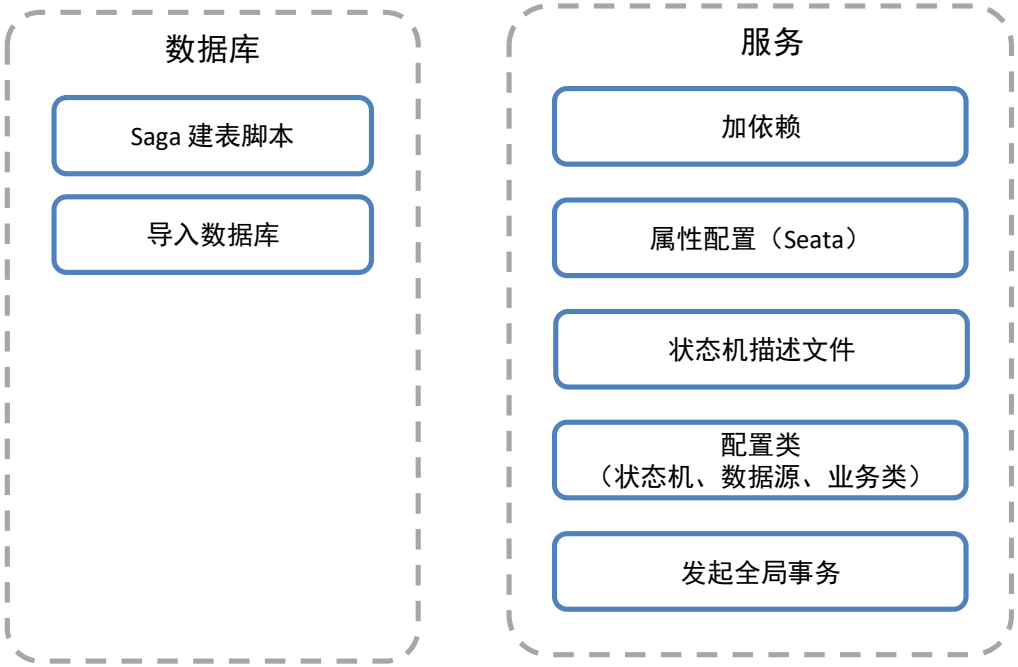
Seata Saga 模式实践 – 总结

重难点

Seata Saga 模式的开发流程

Seata Saga 模式实践 – 总结

开发流程



Seata Saga 模式实践 – 总结

重难点

Seata Saga 模式的开发流程

下节

本章总结

一样的在线教育，不一样的教学品质