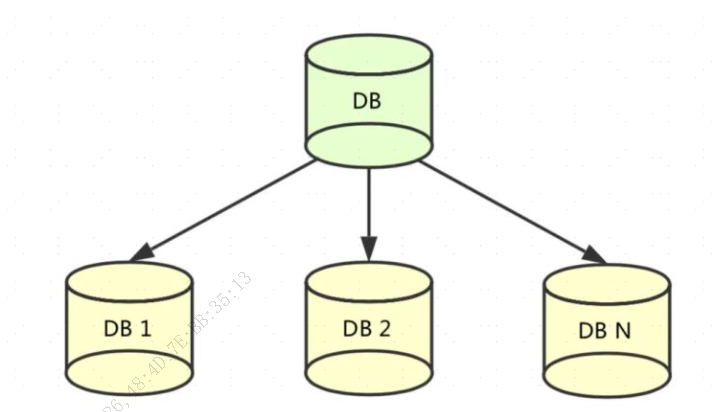
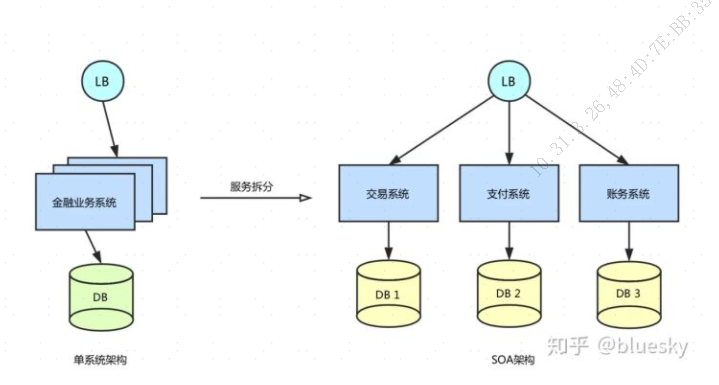
# Seata背景-基本介绍-模式原理-基本使用

### 背景

分布式架构演进之 - 数据库的水平拆分  
蚂蚁金服的业务数据库起初是单库单表，但随着业务数据规模的快速发展，数据量越来越大，单库单表逐渐成为瓶颈。所以我们对数据库进行了水平拆分，将原单库单表拆分成数据库分片。  
如下图所示，分库分表之后，原来在一个数据库上就能完成的写操作，可能就会跨多个数据库，这就产生了跨数据库事务问题。



1.2 分布式架构演进之 - 业务服务化拆分  
在业务发展初期，“一块大饼”的单业务系统架构，能满足基本的业务需求。但是随着业务的快速发展，系统的访问量和业务复杂程度都在快速增长，单系统架构逐渐成为业务发展瓶颈，解决业务系统的高耦合、可伸缩问题的需求越来越强烈。  
如下图所示，蚂蚁金服按照面向服务架构（SOA）的设计原则，将单业务系统拆分成多个业务系统，降低了各系统之间的耦合度，使不同的业务系统专注于自身业务，更有利于业务的发展和系统容量的伸缩。

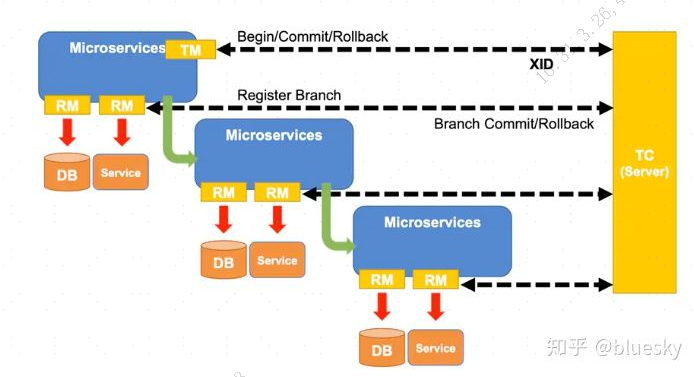


### 基本介绍

Seata事务管理中有三个重要的角色：

* **TC (Transaction Coordinator) - 事务协调者：**维护全局和分支事务的状态，协调全局事务提交或回滚。
* **TM (Transaction Manager) - 事务管理器：**定义全局事务的范围、开始全局事务、提交或回滚全局事务。
* **RM (Resource Manager) - 资源管理器：**管理分支事务处理的资源，与TC交谈以注册分支事务和报告分支事务的状态，并驱动分支事务提交或回滚。

Seata 中有三大模块，分别是 TM、RM 和 TC



### 模式原理

Seata有四种模式 AT 、TCC、saga 、XA

**1、XA模式**

RM一阶段的工作：

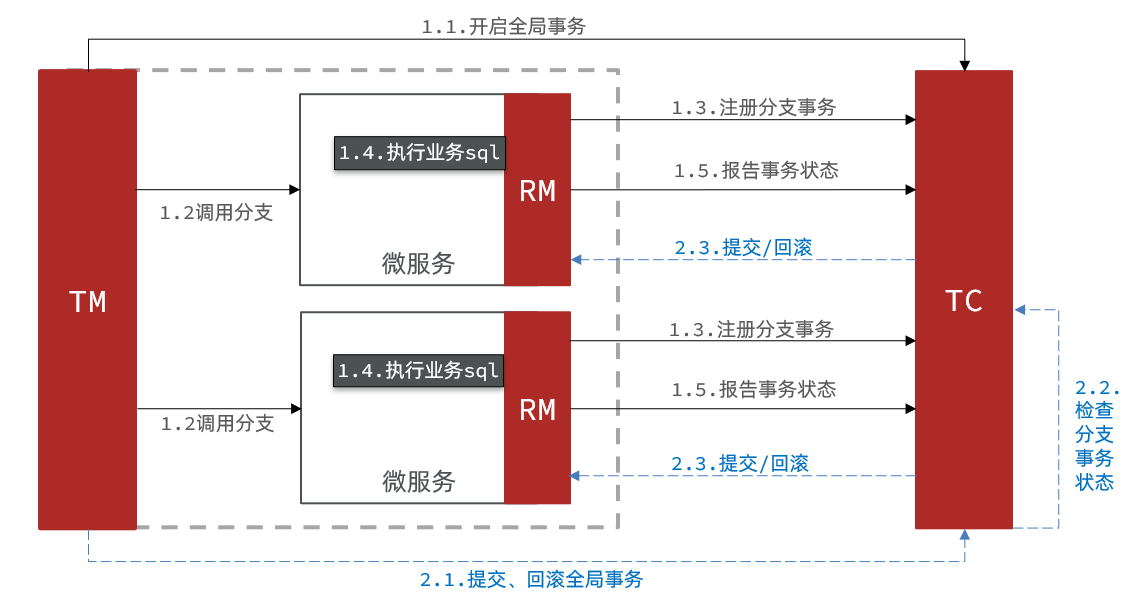
* 注册分支事务到TC
* 执行分支业务sql但不提交
* 报告执行状态到TC

TC二阶段的工作：

* TC检测各分支事务执行状态
* 如果都成功，通知所有RM提交事务
* 如果有失败，通知所有RM回滚事务

RM二阶段的工作：

* 接收TC指令，提交或回滚事务



优缺点

优点:（1）事务的强一致性，满足ACID原则。

（2）常用数据库都支持，实现简单，并且没有代码侵入

缺点:（1）因为一阶段需要锁定数据库资源，等待二阶段结束才释放，性能较差

（2）依赖关系型数据库实现事务

**2、AT模式**

阶段一RM的工作：

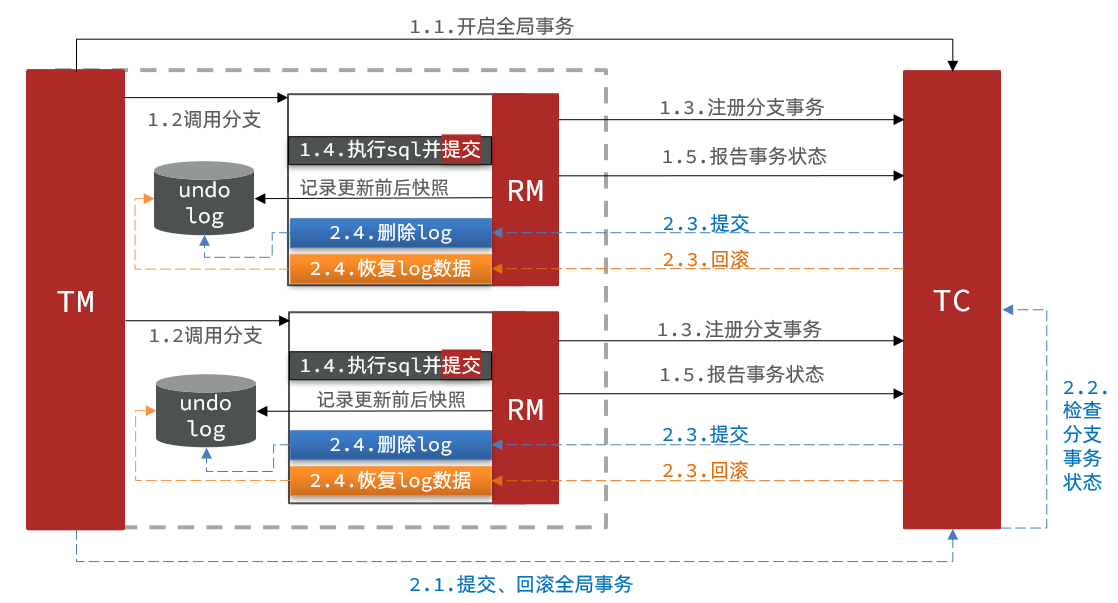
* + 注册分支事务
  + 记录undo-log（数据快照）
  + 执行业务sql并提交
  + 报告事务状态

阶段二提交时RM的工作：

* + 删除undo-log即可

阶段二回滚时RM的工作：

* + 根据undo-log恢复数据到更新前



优缺点

优点:（1）一阶段完成直接提交事务，释放数据库资源，性能比较好

（2）没有代码侵入，框架自动完成回滚和提交

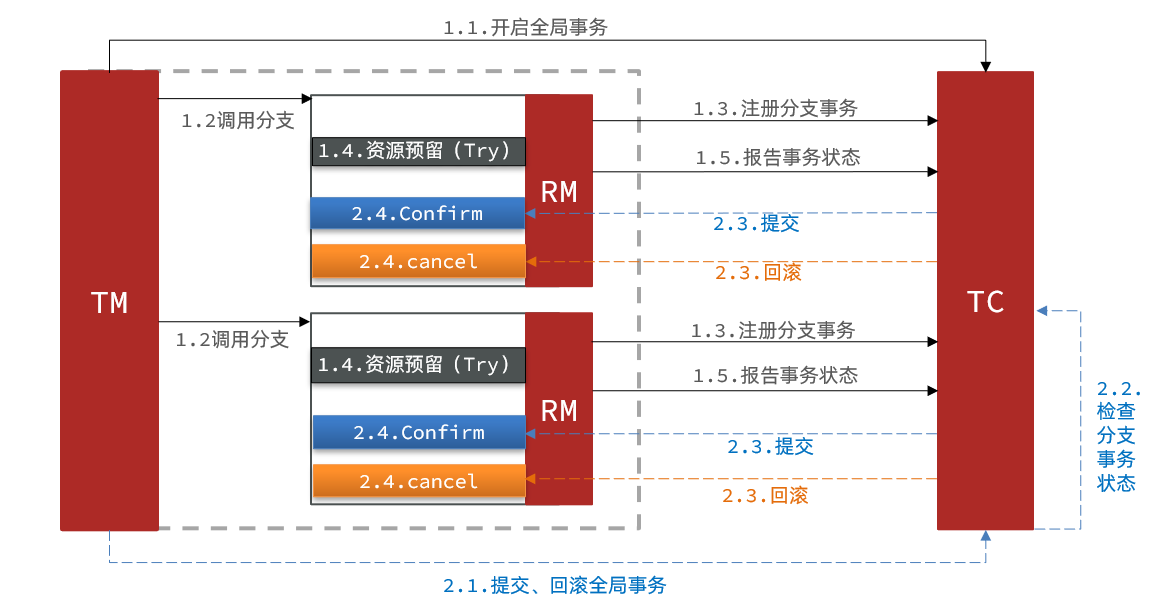
缺点:（1）两阶段之间属于软状态，属于最终一致

（2）框架的快照功能会影响性能，但比XA模式要好很多

**3、TCC模式**

TCC模式与AT模式非常相似，每阶段都是独立事务，不同的是TCC通过人工编码来实现数据恢复。需要实现三个方法：

* Try：资源的检测和预留；
* Confirm：完成资源操作业务；要求 Try 成功 Confirm 一定要能成功。
* Cancel：预留资源释放，可以理解为try的反向操作。



TCC模式的每个阶段

* Try：资源检查和预留
* Confirm：业务执行和提交
* Cancel：预留资源的释放

优缺点

优点:（1）一阶段完成直接提交事务，释放数据库资源，性能好

（2）相比AT模型，无需生成快照，无需使用全局锁，性能最强

（3）不依赖数据库事务，而是依赖补偿操作，可以用于非事务型数据库

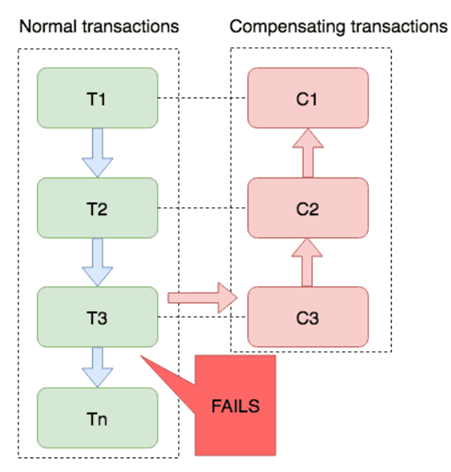
缺点:（1）有代码侵入，需要人为编写try、Confirm和Cancel接口，太麻烦

（2）需要考虑Confirm和Cancel的失败情况

**4、Saga模式**

Saga模式是SEATA提供的长事务解决方案。也分为两个阶段：

* 一阶段：直接提交本地事务
* 二阶段：成功则什么都不做；失败则通过编写补偿业务来回滚



优缺点

优点:（1）事务参与者可以基于事件驱动实现异步调用，吞吐高

（2）一阶段直接提交事务，无锁，性能好

缺点:（1）软状态持续时间不确定，时效性差

（2）没有锁，没有事务隔离，会有脏写

**四种模式对比**



Seata基于上述架构提供了四种不同的分布式事务解决方案：

- XA模式：强一致性分阶段事务模式，牺牲了一定的可用性，无业务侵入

- TCC模式：最终一致的分阶段事务模式，有业务侵入

- AT模式：最终一致的分阶段事务模式，无业务侵入，也是Seata的默认模式

- SAGA模式：长事务模式，有业务侵入

Nacos 集成seate

1、Seate 下载地址 http://seata.io/zh-cn/blog/download.html

2、解压后 修改conf目录下的registry.conf文件：

registry {

# tc服务的注册中心类，这里选择nacos，也可以是eureka、zookeeper等

type = "nacos"

nacos {

# seata tc 服务注册到 nacos的服务名称，可以自定义

application = "seata-tc-server"

serverAddr = "127.0.0.1:8848"

group = "DEFAULT\_GROUP"

namespace = ""

cluster = "SH"

username = "nacos"

password = "nacos"

}

}

config {

# 读取tc服务端的配置文件的方式，这里是从nacos配置中心读取，这样如果tc是集群，可以共享配置

type = "nacos"

# 配置nacos地址等信息

nacos {

serverAddr = "127.0.0.1:8848"

namespace = ""

group = "SEATA\_GROUP"

username = "nacos"

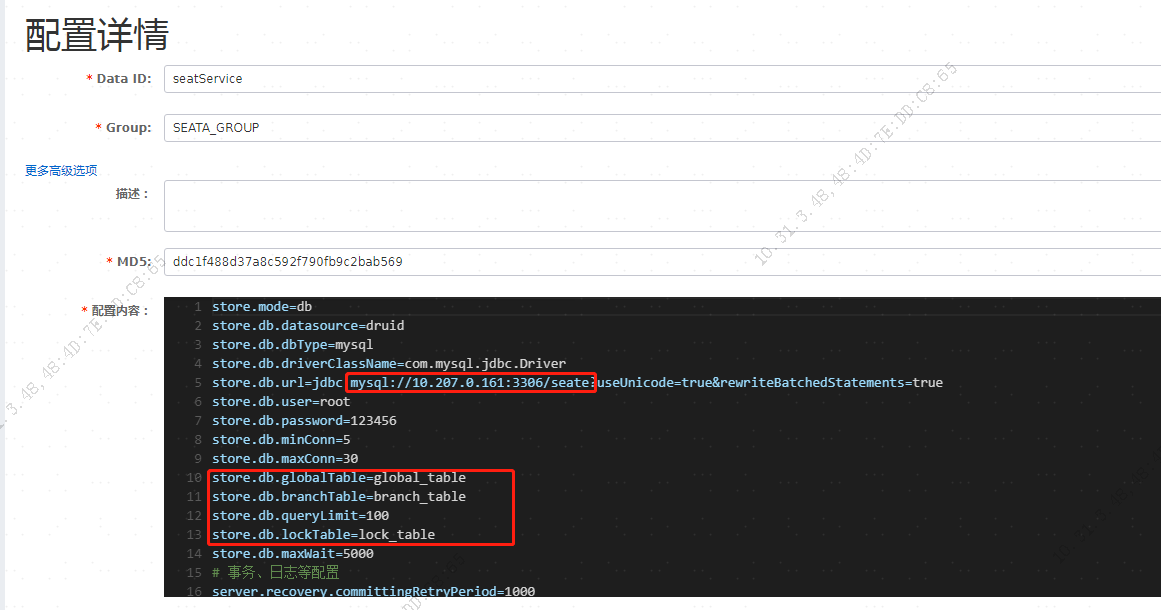
password = "nacos"

dataId = "seataServer "

}

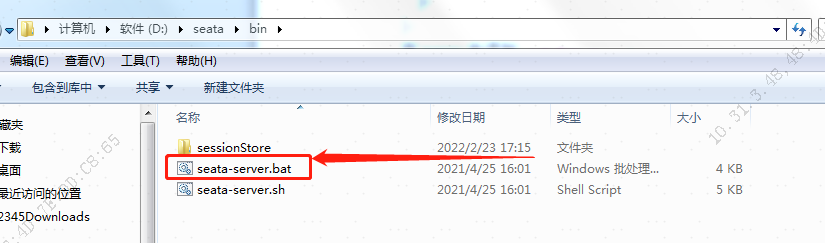
}

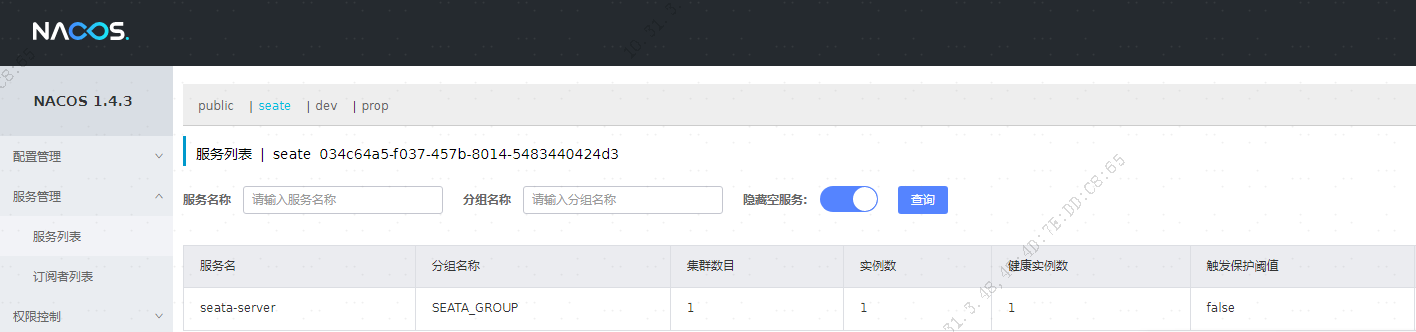
在nacos中添加seataServer



* global\_table：全局事务
* branch\_table：分支事务
* lock\_table：全局锁

然后双击这个文件，就会启动seate服务，在nacos服务列表中，就会出现seate的服务





1. 微服务集成Seata
2. 引入seata相关依赖：

<!--引用 seate jar包-->  
 <dependency>  
 <groupId>com.alibaba.cloud</groupId>  
 <artifactId>spring-cloud-starter-alibaba-seata</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.seata</groupId>  
 <artifactId>seata-spring-boot-starter</artifactId>  
 <version>${seata.version}</version>  
 </dependency>

1. 配置application.yml，让微服务通过注册中心找到seata-server

**seata**:  
 **registry**: *# TC服务注册中心的配置，微服务根据这些信息去注册中心获取tc服务地址*

**type**: *nacos* **nacos**: *# tc* **server-addr**: 127.0.0.1:8848  
 **namespace**: 034c64a5-f037-457b-8014-5483440424d3 **group**: DEFAULT\_GROUP  
 **application**: seata-server *# tc服务在nacos中的服务名称*  
 **tx-service-group**: seata-demo *# 事务组*

**service**:  
 **vgroup-mapping**: *# 事务组与TC服务cluster的映射关系*

**data-source-proxy-mode**: XA *# 设置seate的模式*