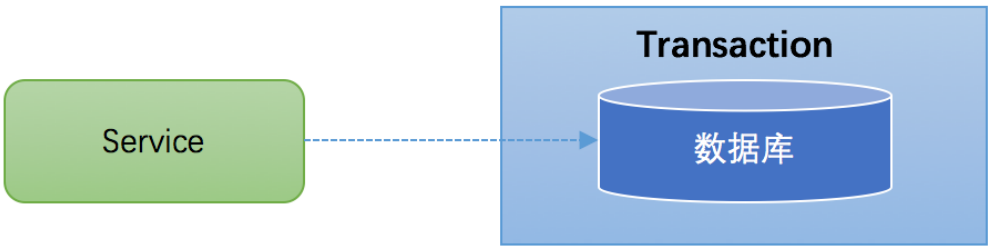
主讲老师： Fox

> 有道笔记链接：https://note.youdao.com/s/nMo1iIN

1. 分布式事务简介

1.1 本地事务

大多数场景下，我们的应用都只需要操作单一的数据库，这种情况下的事务称之为本地事务(Local Transaction)。本地事务的ACID特性是数据库直接提供支持。本地事务应用架构如下所示：



在JDBC编程中，我们通过java.sql.Connection对象来开启、关闭或者提交事务。代码如下所示：

```
Connection conn = ... //获取数据库连接
conn.setAutoCommit(false); //开启事务
try{
    //...执行增删改查sql
    conn.commit(); //提交事务
}catch (Exception e) {
   conn.rollback();//事务回滚
}finally{
    conn.close();//关闭链接
}
```

1.2 分布式事务

在微服务架构中，完成某一个业务功能可能需要横跨多个服务，操作多个数据库。这就涉及到到了分布式事务，需要操作的资源位于多个资源服务器上，而应用需要保证对于多个资源服务器的数据操作，要么全部成功，要么全部失败。本质上来说，分布式事务就是为了保证不同资源服务器的数据一致性。

典型的分布式事务应用场景

跨库事务

跨库事务指的是，一个应用某个功能需要操作多个库，不同的库中存储不同的业务数据。下图演示了一个服务同时操作2个库的情况：

分库分表

通常一个库数据量比较大或者预期未来的数据量比较大，都会进行分库分表。如下图，将数据库B拆分成了2个库：

对于分库分表的情况，一般开发人员都会使用一些数据库中间件来降低sql操作的复杂性。如，对于sql：insert into user(id,name) values (1,"张三"),(2,"李四")。这条sql是操作单库的语法，单库情况下，可以保证事务的一致性。 但是由于现在进行了分库分表，开发人员希望将1号记录插入分库1，2号记录插入分库2。所以数据库中间件要将其改写为2条sql，分别插入两个不同的分库，此时要保证两个库要不都成功，要不都失败，因此基本上所有的数据库中间件都面临着分布式事务的问题。

微服务架构

下图演示了一个3个服务之间彼此调用的微服务架构：

Service A完成某个功能需要直接操作数据库，同时需要调用Service B和Service C，而Service B又同时操作了2个数据库，Service C也操作了一个库。需要保证这些跨服务调用对多个数据库的操作要么都成功，要么都失败，实际上这可能是最典型的分布式事务场景。

小结：上述讨论的分布式事务场景中，无一例外的都直接或者间接的操作了多个数据库。如何保证事务的ACID特性，对于分布式事务实现方案而言，是非常大的挑战。同时，分布式事务实现方案还必须要考虑性能的问题，如果为了严格保证ACID特性，导致性能严重下降，那么对于一些要求快速响应的业务，是无法接受的。

1.3 实现思路：两阶段提交协议(2PC)

两阶段提交（Two Phase Commit），就是将提交(commit)过程划分为2个阶段(Phase)：

阶段1：

TM（事务管理器）通知各个RM（资源管理器）准备提交它们的事务分支。如果RM判断自己进行的工作可以被提交，那就对工作内容进行持久化，再给TM肯定答复；要是发生了其他情况，那给TM的都是否定答复。

以mysql数据库为例，在第一阶段，事务管理器向所有涉及到的数据库服务器发出prepare"准备提交"请求，数据库收到请求后执行数据修改和日志记录等处理，处理完成后只是把事务的状态改成"可以提交",然后把结果返回给事务管理器。

阶段2

TM根据阶段1各个RM prepare的结果，决定是提交还是回滚事务。如果所有的RM都prepare成功，那么TM通知所有的RM进行提交；如果有RM prepare失败的话，则TM通知所有RM回滚自己的事务分支。

以mysql数据库为例，如果第一阶段中所有数据库都prepare成功，那么事务管理器向数据库服务器发出"确认提交"请求，数据库服务器把事务的"可以提交"状态改为"提交完成"状态，然后返回应答。如果在第一阶段内有任何一个数据库的操作发生了错误，或者事务管理器收不到某个数据库的回应，则认为事务失败，回撤所有数据库的事务。数据库服务器收不到第二阶段的确认提交请求，也会把"可以提交"的事务回撤。

两阶段提交方案下全局事务的ACID特性，是依赖于RM的。一个全局事务内部包含了多个独立的事务分支，这一组事务分支要么都成功，要么都失败。各个事务分支的ACID特性共同构成了全局事务的ACID特性。也就是将单个事务分支支持的ACID特性提升一个层次到分布式事务的范畴。

2PC存在的问题

- **同步阻塞问题**

2PC 中的参与者是阻塞的。在第一阶段收到请求后就会预先锁定资源，一直到 commit 后才会释放。

- **单点故障**

由于协调者的重要性，一旦协调者TM发生故障，参与者RM会一直阻塞下去。尤其在第二阶段，协调者发生故障，那么所有的参与者还都处于锁定事务资源的状态中，而无法继续完成事务操作。

- **数据不一致**

若协调者第二阶段发送提交请求时崩溃，可能部分参与者收到commit请求提交了事务，而另一部分参与者未收到commit请求而放弃事务，从而造成数据不一致的问题。

2.Seata是什么

Seata 是一款开源的分布式事务解决方案，致力于提供高性能和简单易用的分布式事务服务。Seata 将为用户提供了 AT、TCC、SAGA 和 XA 事务模式，为用户打造一站式的分布式解决方案。AT模式是阿里首推的模式，阿里云上有商用版本的GTS（Global Transaction Service 全局事务服务）

> 官网：https://seata.io/zh-cn/index.html

2.1 Seata的三大角色

在 Seata 的架构中，一共有三个角色：

- **TC (Transaction Coordinator) - 事务协调者**

维护全局和分支事务的状态，驱动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**

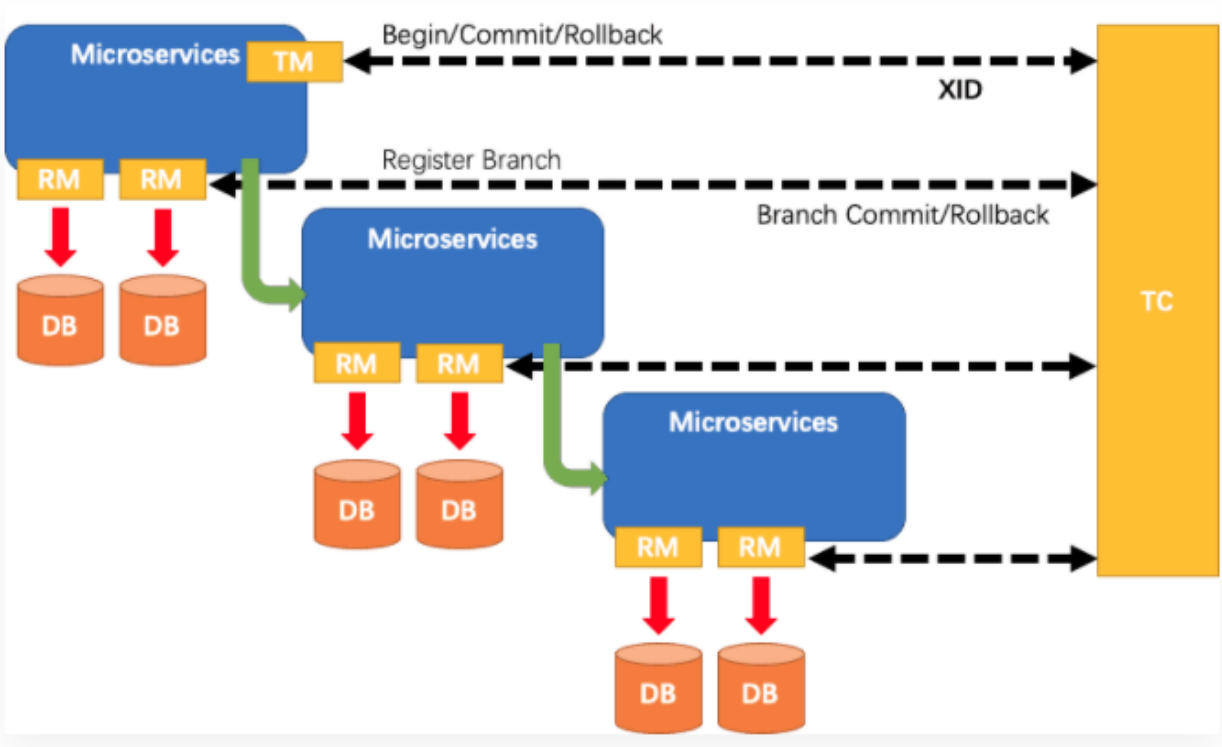
管理分支事务处理的资源，与TC交谈以注册分支事务和报告分支事务的状态，并驱动分支事务提交或回滚。

其中，TC 为单独部署的 Server 服务端，TM 和 RM 为嵌入到应用中的 Client 客户端。

2.2 Seata的生命周期

在 Seata 中，一个分布式事务的生命周期如下：

1. TM 请求 TC 开启一个全局事务。TC 会生成一个 XID 作为该全局事务的编号。XID会在微服务的调用链路中传播，保证将多个微服务的子事务关联在一起。
2. RM 请求 TC 将本地事务注册为全局事务的分支事务，通过全局事务的 XID 进行关联。
3. TM 请求 TC 告诉 XID 对应的全局事务是进行提交还是回滚。
4. TC 驱动 RM 们将 XID 对应的自己的本地事务进行提交还是回滚。



3. Seata快速开始

Seata分TC、TM和RM三个角色，TC（Server端）为单独服务端部署，TM和RM（Client端）由业务系统集成。

3.1 Seata Server（TC）环境搭建

Server端存储模式（store.mode）支持三种：

Server端存储模式（store.mode）现有file、db、redis三种（后续将引入raft,mongodb）

- file：单机模式，全局事务会话信息内存中读写并持久化本地文件root.data，性能较高
- db：高可用模式，全局事务会话信息通过db共享，相应性能差些
- redis：1.3及以上版本支持,性能较高,存在事务信息丢失风险,请提前配置适合当前场景的redis持久化配置

资源目录：

- https://github.com/seata/seata/tree/v1.7.0/script
- client
  - 存放client端sql脚本，参数配置
- config-center
  - 各个配置中心参数导入脚本，config.txt(包含server和client)为通用参数文件
- server
  - server端数据库脚本及各个容器配置

db存储模式+Nacos(注册&配置中心)方式部署

步骤一：下载安装包

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

步骤二：建表(db模式)

创建数据库seata，执行sql脚本，https://github.com/seata/seata/blob/v1.7.0/script/server/db/mysql.sql

步骤三：配置Nacos注册中心

注册中心可以说是微服务架构中的”通讯录“，它记录了服务和服务地址的映射关系。在分布式架构中，服务会注册到注册中心，当服务需要调用其它服务时，就到注册中心找到服务的地址，进行调用。比如Seata Client端(TM,RM)，发现Seata Server(TC)集群的地址,彼此通信。

注意：Seata的注册中心是作用于Seata自身的，和微服务自身配置的注册中心无关，但可以共用注册中心。

Seata支持哪些注册中心?

- eureka
- consul
- nacos

- etcd
- zookeeper
- sofa
- redis
- file (直连)

配置将Seata Server注册到Nacos，修改conf/application.yml文件

```
registry:
    # support: nacos, eureka, redis, zk, consul, etcd3, sofa
    type: nacos
    nacos:
      application: seata-server
      server-addr: 127.0.0.1:8848
      group: SEATA_GROUP
      namespace:
      cluster: default
      username:
      password:
```

注意：请确保client与server的注册处于同一个namespace和group，不然会找不到服务。

启动 Seata-Server 后，会发现Server端的服务出现在 Nacos 控制台中的注册中心列表中。

步骤四：配置Nacos配置中心

配置中心可以说是一个"大货仓",内部放置着各种配置文件,你可以通过自己所需进行获取配置加载到对应的客户端。比如Seata Client端(TM,RM),Seata Server(TC),会去读取全局事务开关,事务会话存储模式等信息。

注意：Seata的配置中心是作用于Seata自身的，和微服务自身配置的配置中心无关，但可以共用配置中心。

Seata支持哪些配置中心?

1. nacos
2. consul
3. apollo
4. etcd

5. zookeeper
6. file (读本地文件, 包含conf、properties、yml配置文件的支持)

1）配置Nacos配置中心地址，修改conf/application.yml文件

```yaml
seata:
  config:
    # support: nacos, consul, apollo, zk, etcd3
    type: nacos
    nacos:
      server-addr: 127.0.0.1:8848
      namespace: 7e838c12-8554-4231-82d5-6d93573ddf32
      group: SEATA_GROUP
      data-id: seataServer.properties
      username:
      password:
```

2）上传配置至Nacos配置中心

https://github.com/seata/seata/blob/v1.7.0/script/config-center/config.txt

a) 获取/seata/script/config-center/config.txt，修改为db存储模式，并修改mysql连接配置

```
store.mode=db
store.lock.mode=db
store.session.mode=db
store.db.driverClassName=com.mysql.jdbc.Driver
store.db.url=jdbc:mysql://127.0.0.1:3306/seata?
useUnicode=true&rewriteBatchedStatements=true
store.db.user=root
store.db.password=root
```

在store.mode=db时，由于seata是通过jdbc的executeBatch来批量插入全局锁的，根据MySQL官网的说明，连接参数中的rewriteBatchedStatements为true时，在执行executeBatch，并且操作类型为insert时，jdbc驱动会把对应的SQL优化成`insert into () values (), ()`的形式来提升批量插入的性能。

根据实际的测试，该参数设置为true后，对应的批量插入性能为原来的10倍多，因此在数据源为MySQL时，建议把该参数设置为true。

b) 配置事务分组， 要与client配置的事务分组一致

- 事务分组：seata的资源逻辑，可以按微服务的需要，在应用程序（客户端）对自行定义事务分组，每组取一个名字。
- 集群：seata-server服务端一个或多个节点组成的集群cluster。 应用程序（客户端）使用时需要指定事务逻辑分组与Seata服务端集群的映射关系。

事务分组如何找到后端Seata集群（TC）？

1. 首先应用程序（客户端）中配置了事务分组（GlobalTransactionScanner 构造方法的txServiceGroup参数）。若应用程序是SpringBoot则通过seata.tx-service-group 配置。
2. 应用程序（客户端）会通过用户配置的配置中心去寻找service.vgroupMapping .[事务分组配置项]，取得配置项的值就是TC集群的名称。若应用程序是SpringBoot则通过seata.service.vgroup-mapping.事务分组名=集群名称 配置
3. 拿到集群名称程序通过一定的前后缀+集群名称去构造服务名，各配置中心的服务名实现不同（前提是Seata-Server已经完成服务注册，且Seata-Server向注册中心报告cluster名与应用程序（客户端）配置的集群名称一致）
4. 拿到服务名去相应的注册中心去拉取相应服务名的服务列表，获得后端真实的TC服务列表（即Seata-Server集群节点列表）

c) 在nacos配置中心中新建配置，dataId为seataServer.properties，配置内容为上面修改后的config.txt中的配置信息

从v1.4.2版本开始，seata已支持从一个Nacos dataId中获取所有配置信息,你只需要额外添加一个dataId配置项。

添加后查看：

步骤五：启动Seata Server

启动命令:

```
bin/seata-server.sh
```

启动成功，查看控制台，账号密码都是seata。http://localhost:7091/#/login

在Nacos注册中心中可以查看到seata-server注册成功

3.2 Seata Client快速开始

微服务整合Seata AT模式实战

业务场景

用户下单，整个业务逻辑由三个微服务构成：

- 库存服务：对给定的商品扣除库存数量。
- 订单服务：根据采购需求创建订单。
- 帐户服务：从用户帐户中扣除余额。

1) 环境准备

- 父pom指定微服务版本

Spring Cloud Alibaba Version	Spring Cloud Version	Spring Boot Version	Seata Version
2022.0.0.0	2022.0.0	3.0.2	1.7.0

- 启动Seata Server(TC)端，Seata Server使用nacos作为配置中心和注册中心
- 启动nacos服务

2) 微服务导入seata依赖

spring-cloud-starter-alibaba-seata内部集成了seata，并实现了xid传递

```
<!-- seata-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-seata</artifactId>
</dependency>
```

3)微服务对应数据库中添加undo_log表(仅AT模式)

https://github.com/seata/seata/blob/v1.7.0/script/client/at/db/mysql.sql

```
-- for AT mode you must to init this sql for you business database. the seata server
not need it.
CREATE TABLE IF NOT EXISTS `undo_log`
(
    `branch_id`     BIGINT       NOT NULL COMMENT 'branch transaction id',
    `xid`           VARCHAR(128) NOT NULL COMMENT 'global transaction id',
```

```
    `context`       VARCHAR(128) NOT NULL COMMENT 'undo_log context,such as serialization',
    `rollback_info` LONGBLOB     NOT NULL COMMENT 'rollback info',
    `log_status`    INT(11)      NOT NULL COMMENT '0:normal status,1:defense status',
    `log_created`   DATETIME(6)  NOT NULL COMMENT 'create datetime',
    `log_modified`  DATETIME(6)  NOT NULL COMMENT 'modify datetime',
    UNIQUE KEY `ux_undo_log` (`xid`, `branch_id`)
) ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8mb4 COMMENT ='AT transaction mode undo table';
ALTER TABLE `undo_log` ADD INDEX `ix_log_created` (`log_created`);
```

4) 微服务application.yml中添加seata配置

```
seata:
  application-id: ${spring.application.name}
  # seata 服务分组，要与服务端配置service.vgroup_mapping的后缀对应
  tx-service-group: default_tx_group
  registry:
    # 指定nacos作为注册中心
    type: nacos
    nacos:
      application: seata-server
      server-addr: 127.0.0.1:8848
      namespace:
      group: SEATA_GROUP

  config:
    # 指定nacos作为配置中心
    type: nacos
    nacos:
      server-addr: 127.0.0.1:8848
      namespace: 7e838c12-8554-4231-82d5-6d93573ddf32
      group: SEATA_GROUP
      data-id: seataServer.properties
```

注意：请确保client与server的注册中心和配置中心namespace和group一致

5）在全局事务发起者中添加@GlobalTransactional注解

核心代码

```
@Override
@GlobalTransactional(name="createOrder",rollbackFor=Exception.class)
public Order saveOrder(OrderVo orderVo){
    log.info("=============用户下单=================");
    log.info("当前 XID: {}", RootContext.getXID());

    // 保存订单
    Order order = new Order();
    order.setUserId(orderVo.getUserId());
    order.setCommodityCode(orderVo.getCommodityCode());
    order.setCount(orderVo.getCount());
    order.setMoney(orderVo.getMoney());
    order.setStatus(OrderStatus.INIT.getValue());

    Integer saveOrderRecord = orderMapper.insert(order);
    log.info("保存订单{}", saveOrderRecord > 0 ? "成功" : "失败");

    //扣减库存
    storageFeignService.deduct(orderVo.getCommodityCode(),orderVo.getCount());

    //扣减余额
    accountFeignService.debit(orderVo.getUserId(),orderVo.getMoney());

    //更新订单
    Integer updateOrderRecord =
orderMapper.updateOrderStatus(order.getId(),OrderStatus.SUCCESS.getValue());
    log.info("更新订单id:{} {}", order.getId(), updateOrderRecord > 0 ? "成功" : "失
败");

    return order;

}
```

6）测试分布式事务是否生效

- 分布式事务成功，模拟正常下单、扣库存，扣余额
- 分布式事务失败，模拟下单扣库存成功、扣余额失败，事务是否回滚

4. Seata AT模式的设计思路

Seata AT模式的核心是对业务无侵入，是一种改进后的两阶段提交，其设计思路如下:

- 一阶段：业务数据和回滚日志记录在同一个本地事务中提交，释放本地锁和连接资源。
- 二阶段：
  - 提交异步化，非常快速地完成。
  - 回滚通过一阶段的回滚日志进行反向补偿。

一阶段

业务数据和回滚日志记录在同一个本地事务中提交，释放本地锁和连接资源。核心在于对业务sql进行解析，转换成undolog，并同时入库，这是怎么做的呢?

二阶段

- 分布式事务操作成功，则TC通知RM异步删除undolog
- 分布式事务操作失败，TM向TC发送回滚请求，RM 收到协调器TC发来的回滚请求，通过 XID 和 Branch ID 找到相应的回滚日志记录，通过回滚记录生成反向的更新 SQL 并执行，以完成分支的回滚。