分布式事务

一、本地事务

> 事务,

1.1 ACID特性

- 原子性(A)
- 一致性(C)
- 隔离性(I)
- 持久性(D)

1.2 事务的隔离级别

> 两个或多个事务并发操作相同的数据的时候事务之间的相互访问关系

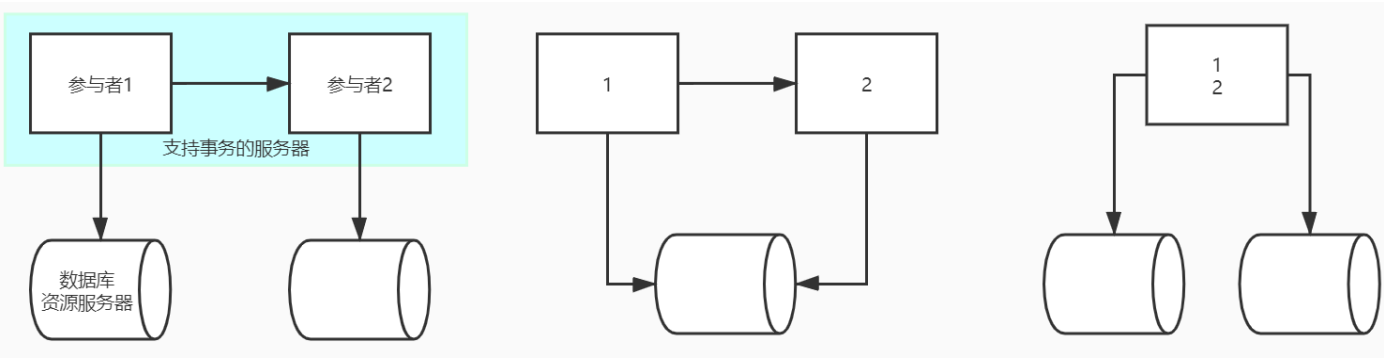
隔离级别	脏读	不可重复读	幻读
read uncommitted	√	√	√
read committed	×	√	√
repeatable read	×	×	√
serializable	×	×	×

- 查询当前隔离级别：select @@tx_isolation
- 设置隔离级别：set session transaction isolation level `隔离级别`
- 开启事务：start transaction
- 提交事务：commit
- 事务回滚：rollaback

二、分布式事务

> 分布式事务：就是指事务的参与者、支持事务的服务器（数据库服务器）、资源服务器以及事务的管理器分布在分布式系统的不同节点中

2.1 分布式事务场景



2.2 分布式事务 & 分布式锁

- 分布式事务：完成事务的多个步骤位于不同的节点上
- 分布式锁：用于解决分布式系统中事务之间的并发访问问题

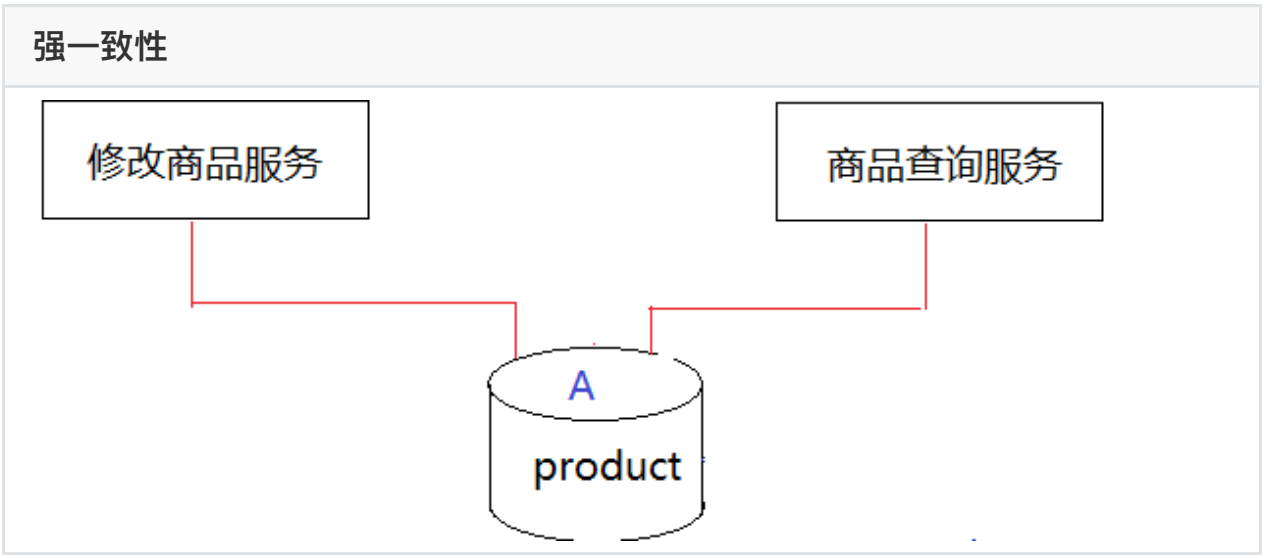
三、CAP定律和BASE理论

> 分布式系统设计中的CAP定律和base理论

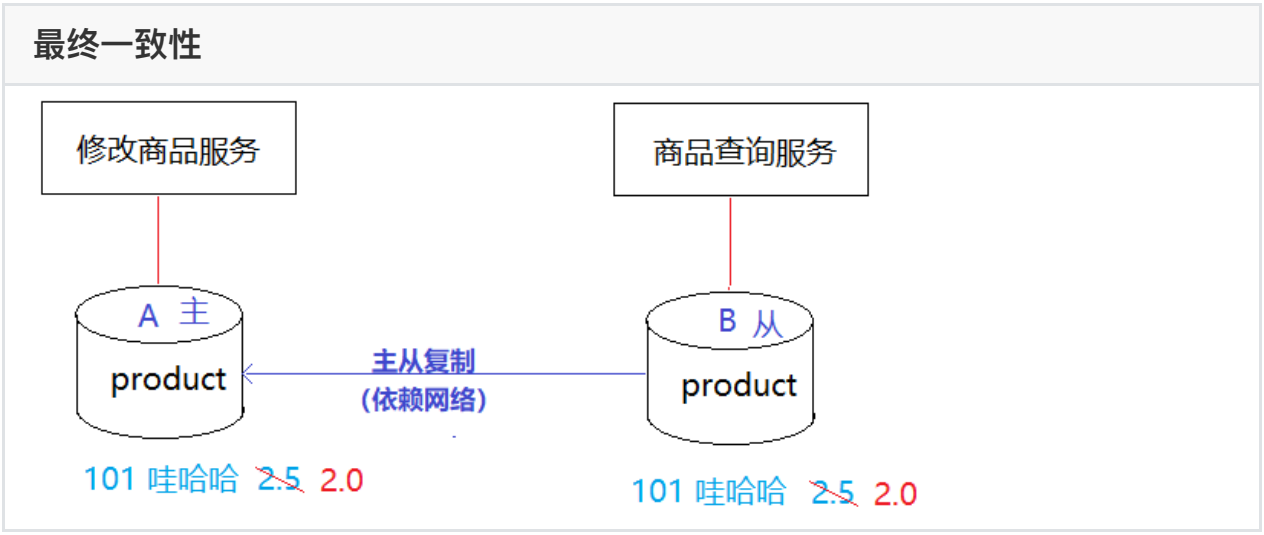
3.1 CAP定律

> - CAP原则又称CAP定律，指的是在一个分布式系统中的一致性（Consistency）、可用性（Availability）、分区容错性三者之间的权衡
> - CAP 原则指的是，这三个要素最多只能同时实现两点，不可能三者兼顾
>   - 在分布式系统无法同时满足CA，如果需要满足CA，则项目结构必须为单体架构
>   - 在分布式系统中可以满足CP 或者 AP，常规情况下微服务架构更多的是满足AP

- `（强）一致性(Consistency)`：如果系统对一个写操作返回成功，那么之后的读请求都必须读到这个新数据；如果写操作返回失败，则所有的读操作都不能读到这个数据，对调用者而言数据具有强一致性。
  - 强一致性：一旦写操作成功了，则所有的读操作都必须读取新数据；（如果想要保证数据的强一致性，就必须使用同一个数据存储/数据库）



- 弱一致性/最终一致性：当写操作成功之后，允许在一定的时间内读取到旧数据，但经过一段时间之后最终可以读取到新数据，保证数据最终是一致的



- `可用性(Availability)`：当用户请求服务时，服务一定要给与响应，可以是降级响应。
- `分区容错性(Partition tolerance)`：在分布式系统中服务节点都是网络分布，一个或部分节点出现故障，其他节点仍能对外提供服务。

3.2 BASE理论

> - BASE是Basically Available(基本可用)，Soft State（软状态）和Eventually Consistent（最终一致性）三个短语的缩写。
> - BASE理论，是对CAP中一致性和可用性权衡的结果，其来源于对大规模互联网分布式系统实践的总结，是基于CAP定律逐步演化而来。其核心思想是即使无法做到强一致性，但每个应用都可以根据自身业务特点，采用适当的方式来使系统达到最终一致性。

- 基本可用：指的是分布式系统中出现不可预知故障，允许其损失一部分的功能，但要保证整个系统的可用。

- 软状态：允许系统中数据存在中间状态，这个中间状态不会影响系统的可用性；也就是允许不同节点的数据副本之间在数据同步过程中存在延时。
- 最终一致性：要求所有的数据副本在经过一段时间的延时之后，最终能够达到一致的状态。

四、分布式事务解决方案

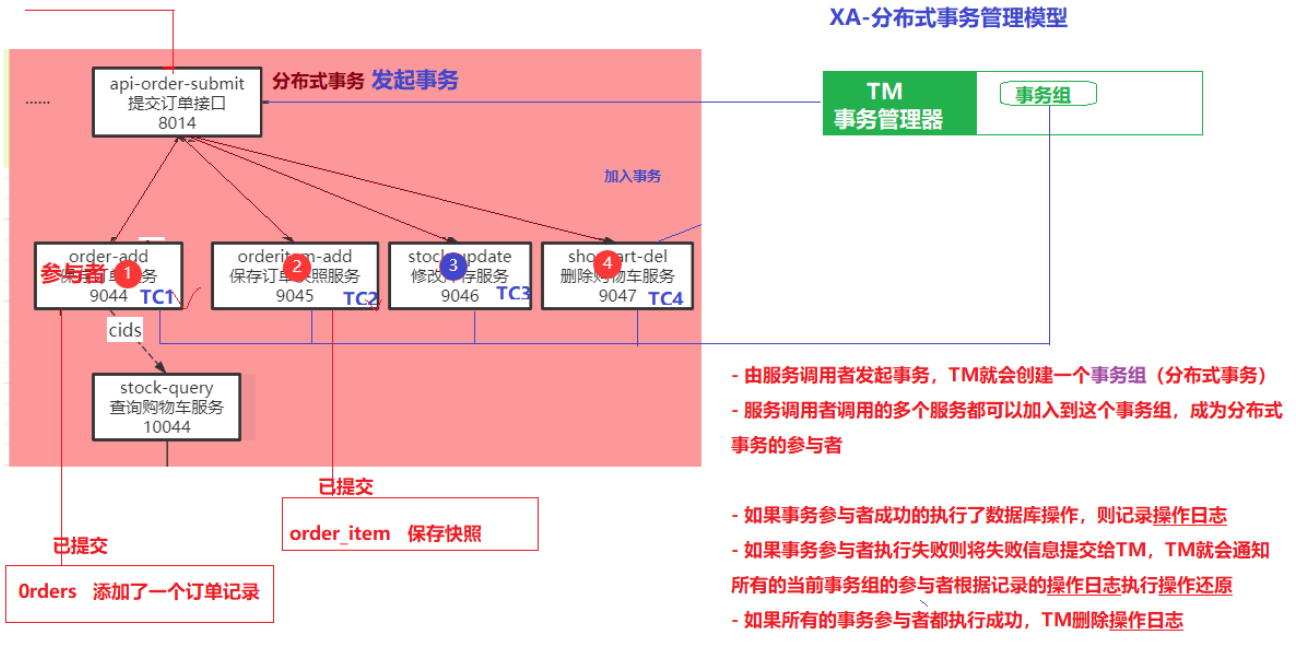
4.1 刚性事务与柔性事务

- 刚性事务：满足ACID特性的事务（强一致性）————本地事务
- 柔性事务：满足BASE理论的事务（最终一致性）————分布式事务
- 如何保证分布式事务的最终一致性?
  - XA-2PC
  - 补偿
  - 异步确保
  - 最大努力通知

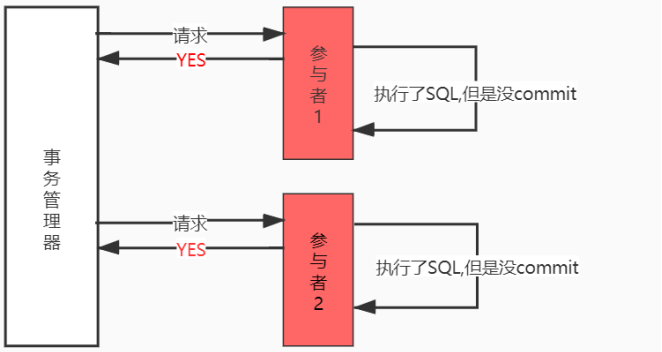
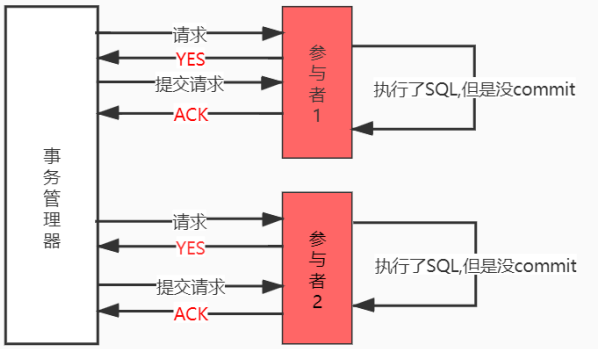
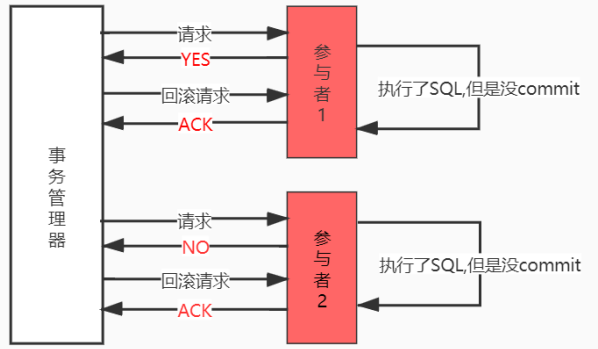
4.2 XA-分布式事务管理模型

> XA模型—为分布式事务的多个参与者添加一个事务管理器(事务协调者)

XA模型



4.3 2PC—两段式提交

第一阶段	第二阶段
第一阶段： a. 协调者向所有的参与者发送请求，询问是否可以提交事务，等待答复 b. 各个事务参与者执行事务（预执行），将执行信息记录到事务日志中（不提交事务） c. 每个参与者执行成功就返回yes，否则返回no  	第2阶段： 第一种情况：所有参与者都返回了YES,则提交事务 a.协调者向所有的事务参与者发出提交请求（commit） b.事务参与者执行提交 c.参与者完成提交之后，返回应答信息（ACK） d.事务协调者接受到所有的参与者的ACK则事务结束  第2阶段： 第二种情况：有参与者返回NO,则回滚事务 a.协调者向所有的事务参与者发出回滚请求（rollback） b.事务参与者执行回滚 c.参与者完成回滚之后，返回应答信息（ACK） d.事务协调者接受到所有的参与者的ACK则事务结束 

问题

1.性能问题：所有事务的参与者在提交阶段处于阻塞状态，占用系统资源（数据库连接）

2.可靠性问题：如果事务协调者出现单点故障，将导致所有的参与者都处于锁定状态

3.数据一致性问题：事务协调者和部分参与者在事务提交阶段挂了，有可能导致数据一致性问题

优点

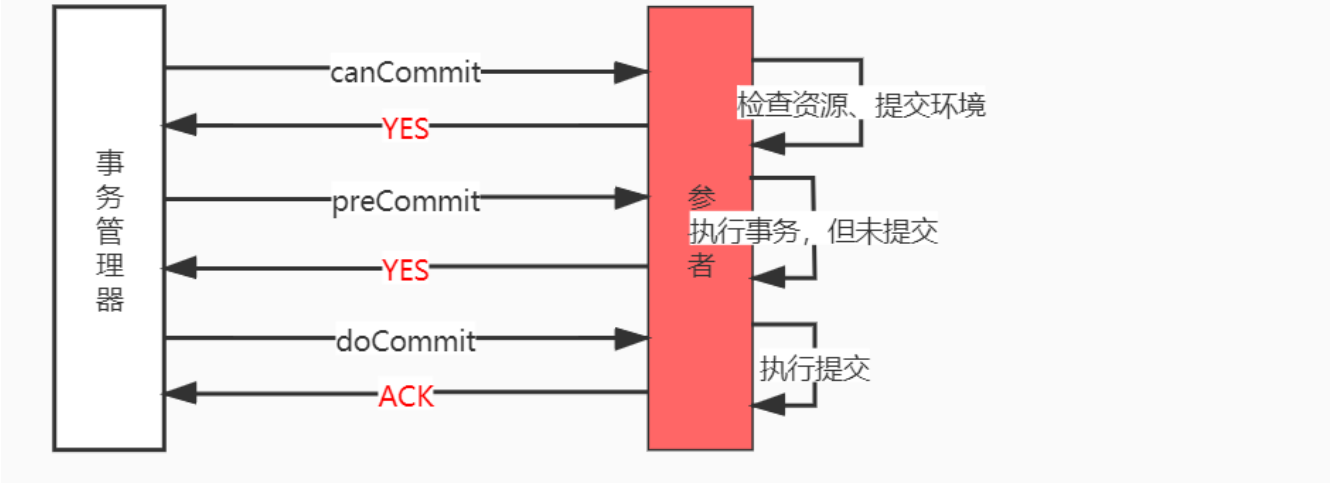
近乎100%的保证了数据的一致性

缺点

实现复杂，牺牲了可用性，对性能影响比较大；适用于并发不高但是对数据一致性要求比较高的场景。

4.4 3PC—三段式提交

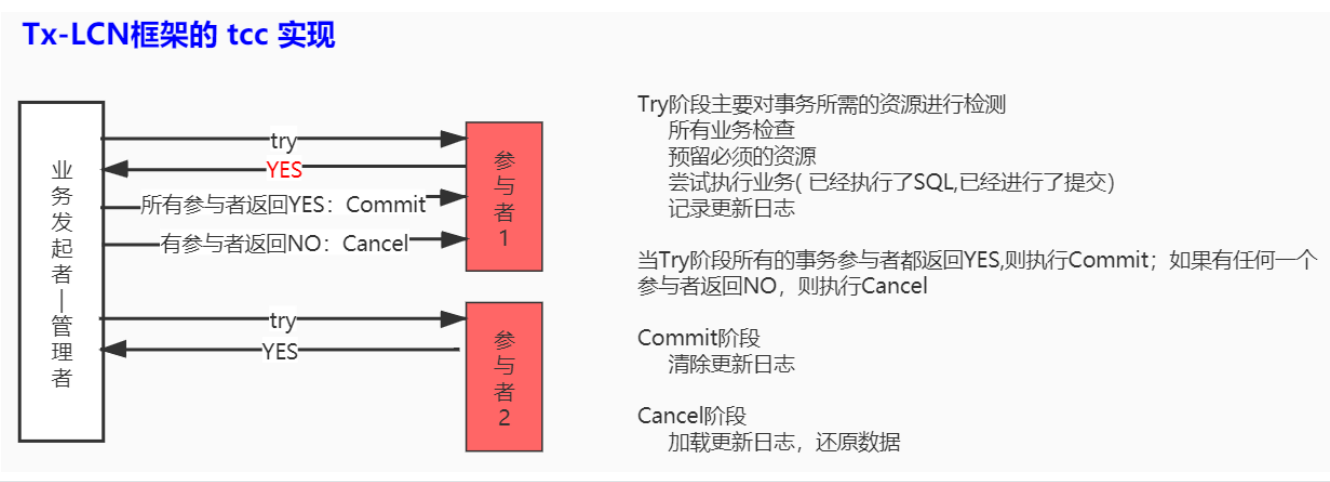
> 三段式提交就是在两段式提交进行改进的版本：
>
> - 增了一个资源检查阶段（询问是否可以提交）
> - 增加了超时设置——避免因TM故障导致TC长时间等待占用系统资源



- 存在的问题：
  - 和2PC提交一样，执行SQL之后需要保持数据库连接，影响系统性能

4.5 TCC

> TCC, 即Try-Commit-Cancel

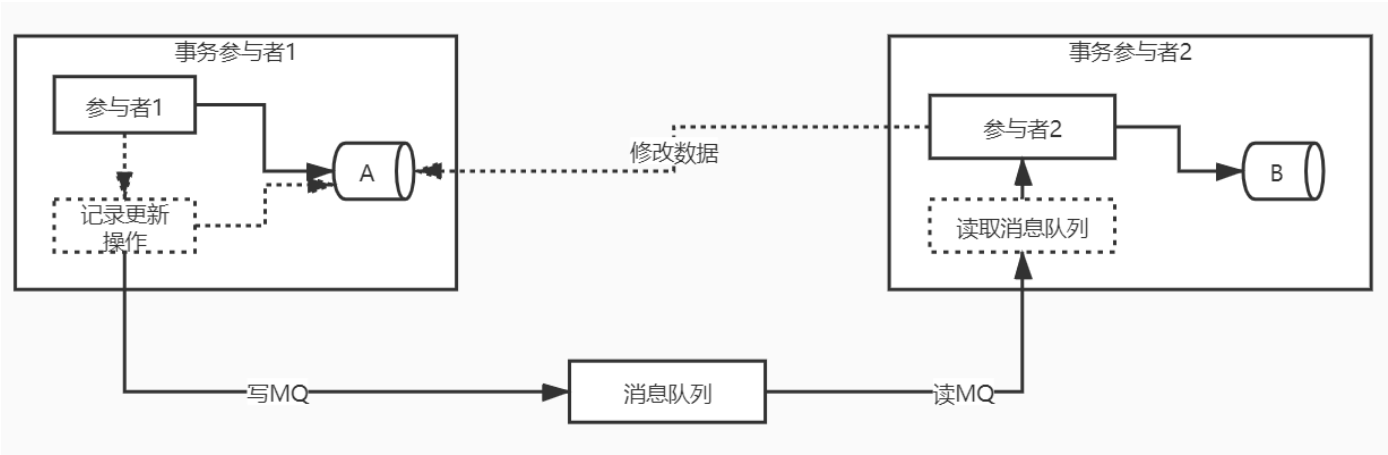


优点

1.性能提升：资源占用的粒度较小，不会长时间锁定所有资源

2.数据的最终一致性：基于commit和cancel的幂等性

4.6 消息队列



五、分布式事务框架 Tx-LCN

> 3PC——适用于对数据一致性要求较高的场景，对性能会有一定损耗
> TCC——性能优于3PC，但是不能保证数据的强一致性，可以保证最终一致性

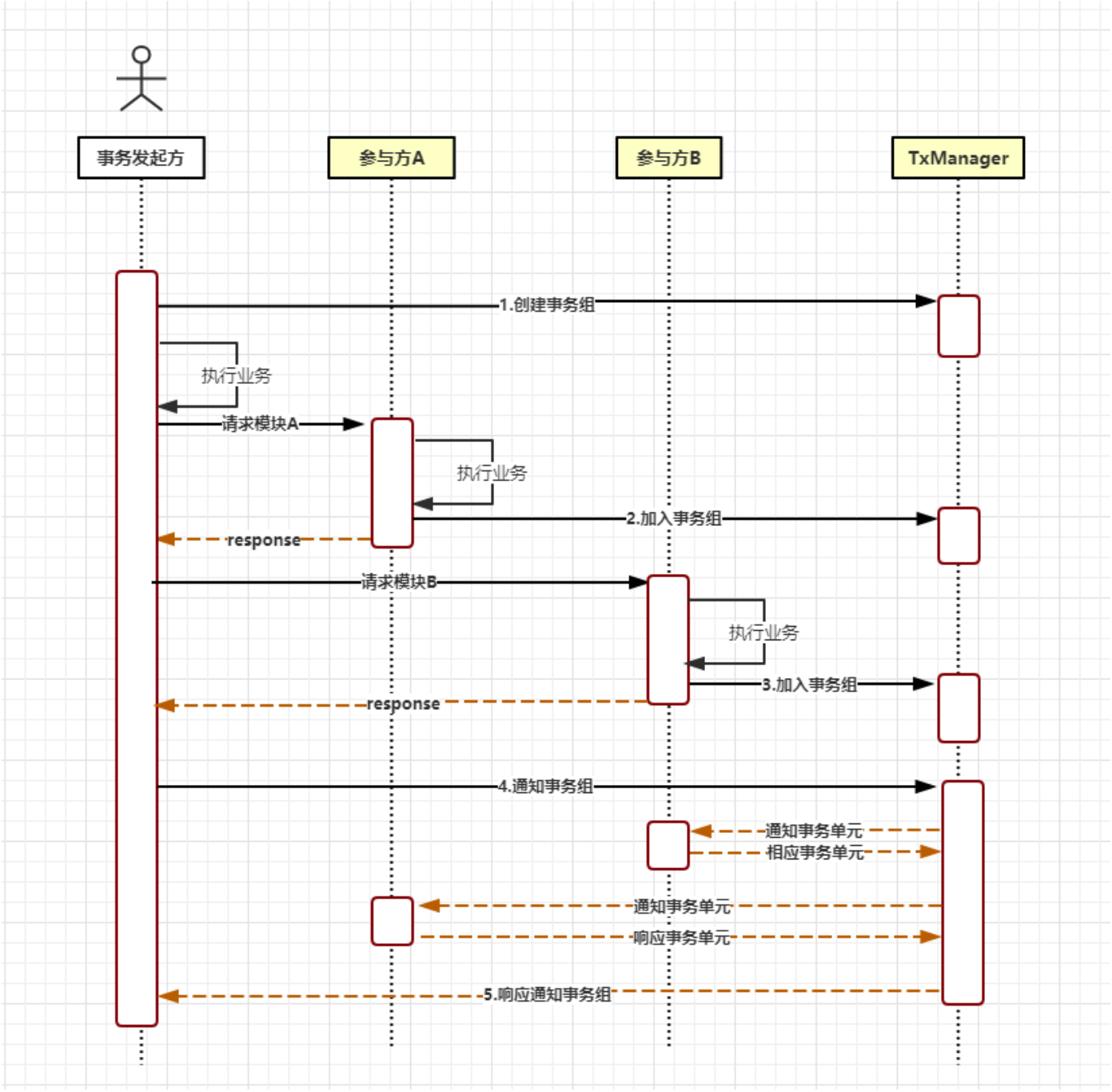
基于3PC、TCC等分布式事务解决方案已经有成熟的落地框架：

- zookeeper
- Tx-LCN
- Spring Cloud alibaba seata

> LCN模式是通过代理Connection方式实现对本地事务的操作，然后由TxManager统一协调管理

官方文档：https://www.codingapi.com/docs/txlcn-preface/

5.1 工作流程



5.2 Tx-LCN使用案例

- txlcn使用案例.md

5.3 TxLCN支持的分布式事务管理方式

- @LcnTransaction lcn模式
    - LCN模式是通过代理Connection的方式实现对本地事务的操作，然后在由TxManager统一协调控制事务。当本地 事务提交回滚或者关闭连接时将会执行假操作，该代理的连接将由LCN连接池管理。
- @TxcTransaction txc模式
    - TXC模式命名来源于淘宝，实现原理是在执行SQL之前，先查询SQL的影响数据，然

后保存执行的SQL快照信息和 创建锁。当需要回滚的时候就采用这些记录数据回滚数据库，目前锁实现依赖redis分布式锁控制。

- @TccTransaction tcc模式
  - TCC事务机制相对于传统事务机制（X/Open XA Two-Phase-Commit），其特征在于它不依赖资源管理器(RM)对XA 的支持，而是通过对（由业务系统提供的）业务逻辑的调度来实现分布式事务。主要由三步操作，Try: 尝试执行业务、 Confirm:确认执行业务、 Cancel: 取消执行业务。

六、搭建TM服务器

- 按照TM的要求建库建表

```
CREATE TABLE `t_tx_exception`  (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `group_id` varchar(64) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `unit_id` varchar(32) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `mod_id` varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NULL DEFAULT NULL,
  `transaction_state` tinyint(4) NULL DEFAULT NULL,
  `registrar` tinyint(4) NULL DEFAULT NULL,
  `ex_state` tinyint(4) NULL DEFAULT NULL COMMENT '0 待处理 1已处理',
  `remark` varchar(10240) NULL DEFAULT NULL COMMENT '备注',
  `create_time` datetime(0) NULL DEFAULT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 967 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Dynamic;

SET FOREIGN_KEY_CHECKS = 1;
```

- 配置并启动redis
- 创建SpringBoot项目
- 导入tm依赖

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.47</version>
 </dependency>

<dependency>
    <groupId>com.codingapi.txlcn</groupId>
    <artifactId>txlcn-tm</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

- 配置application.properties(官方说明文档提供)

```
spring.application.name=TransactionManager
server.port=8801
# JDBC 数据库配置
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://47.96.11.185:3306/fmmall2?characterEncoding=UTF-8&serverTimeZone=UTC
spring.datasource.username=root
spring.datasource.password=admin123

# 数据库方言
spring.jpa.database-platform=org.hibernate.dialect.MySQL57Dialect

# 为TM创建持久化数据库表
spring.jpa.hibernate.ddl-auto=update

# TM监听Socket端口. 默认为 ${server.port} - 100
tx-lcn.manager.port=8070
# TM后台登陆密码，默认值为codingapi
tx-lcn.manager.admin-key=admin123
# 雪花算法的sequence位长度，默认为12位.
tx-lcn.manager.seq-len=12
# 异常回调开关。开启时请制定ex-url
tx-lcn.manager.ex-url-enabled=false
# 开启日志,默认为false
tx-lcn.logger.enabled=true
tx-lcn.logger.driver-class-name=${spring.datasource.driver-class-name}
tx-lcn.logger.jdbc-url=${spring.datasource.url}
```

```
tx-lcn.logger.username=${spring.datasource.username}
tx-lcn.logger.password=${spring.datasource.password}

# redis 的设置信息. 线上请用Redis Cluster
spring.redis.host=47.96.11.185
spring.redis.port=6379
spring.redis.password=12345678
```

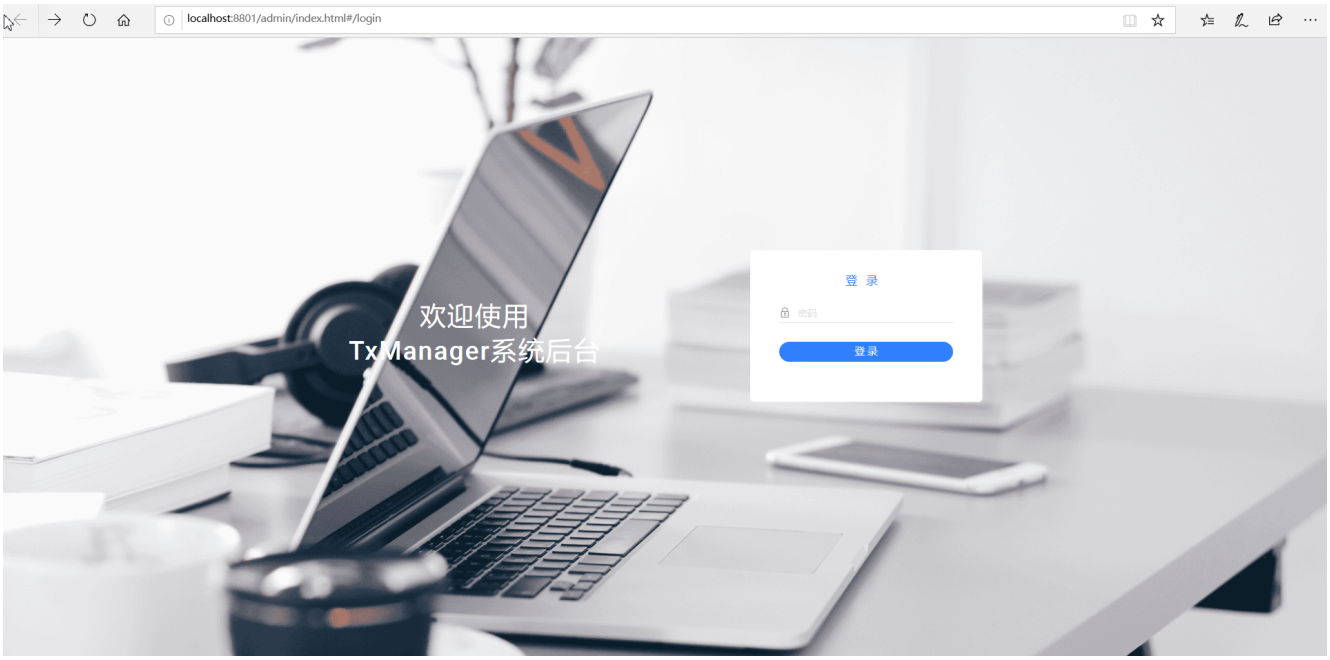
- 启动类添加`@EnableTransactionManagerServer`注解

```
@SpringBootApplication
@EnableTransactionManagerServer
public class TxmanagerApplication {

    public static void main(String[] args) {
        SpringApplication.run(TxmanagerApplication.class, args);
    }

}
```

- 启动项目，访问8801，出现如下界面（使用设置的密码登录）：



七、在服务中添加分布式事务支持

- 添加TC依赖

```
<dependency>
    <groupId>com.codingapi.txlcn</groupId>
    <artifactId>txlcn-tc</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
<dependency>
    <groupId>com.codingapi.txlcn</groupId>
    <artifactId>txlcn-txmsg-netty</artifactId>
    <version>5.0.2.RELEASE</version>
</dependency>
```

- 配置服务链接到TM

```
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/fmmall2?characterEncoding=utf-8
    username: root
    password: admin123

tx-lcn:
  client:
    manager-address: localhost:8070
```

- 在启动类添加`@EnableDistributedTransaction`注解
- 添加分布式事务管理注解

```
@TccTransaction
@Transactional
public void addOrder(Order order) {
    orderDAO.insertOrder(order);
    ResultVO vo = repoInvokeService.update(order.getGid(), 1);
    System.out.println(vo);
}
```

  - @LcnTransaction lcn模式

        - LCN模式是通过代理Connection的方式实现对本地事务的操作，然后在由TxManager统一协调控制事务。当本地 事务提交回滚或者关闭连接时将会执行假操作，该代理的连接将由LCN连接池管理。
    - @TxcTransaction txc模式
        - TXC模式命名来源于淘宝，实现原理是在执行SQL之前，先查询SQL的影响数据，然后保存执行的SQL快走信息和 创建锁。当需要回滚的时候就采用这些记录数据回滚数据库，目前锁实现依赖redis分布式锁控制。
    - @TccTransaction tcc模式
        - TCC事务机制相对于传统事务机制（X/Open XA Two-Phase-Commit），其特征在于它不依赖资源管理器(RM)对XA 的支持，而是通过对（由业务系统提供的）业务逻辑的调度来实现分布式事务。主要由三步操作，Try: 尝试执行业务、Confirm:确认执行业务、 Cancel: 取消执行业务。

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静!