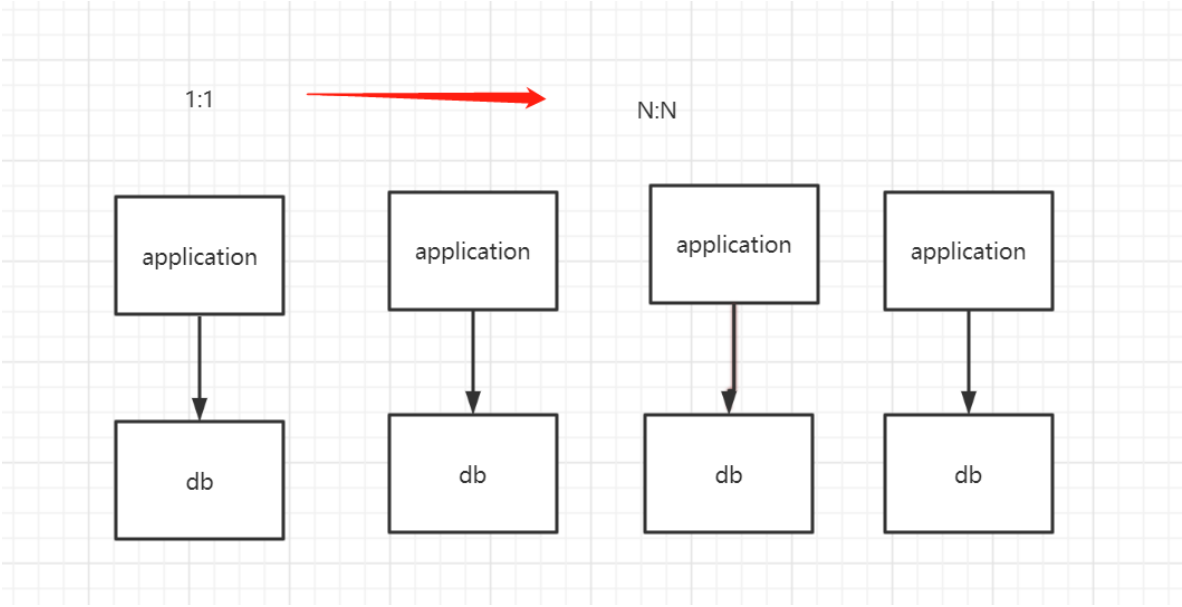
一、分布式事务

1、分布式事务的由来

由单机系统到微服务系统的扩展，

我们系统的某一个业务可能需要多个微服务的参与，而多个微服务有可能依赖于多个数据源，

那么在这个业务调用的过程中，我们怎么去保证它的所有的操作要么全部成功，要么全部失败。



模拟用户下单的业务场景，分为三个步骤

1. 创建订单
2. 扣减库存
3. 扣减账户余额

在1:1的系统架构下，我们所有的操作都是在一个应用和一个数据库 关于事务的操作我们可以用过@Transactional维护即可。

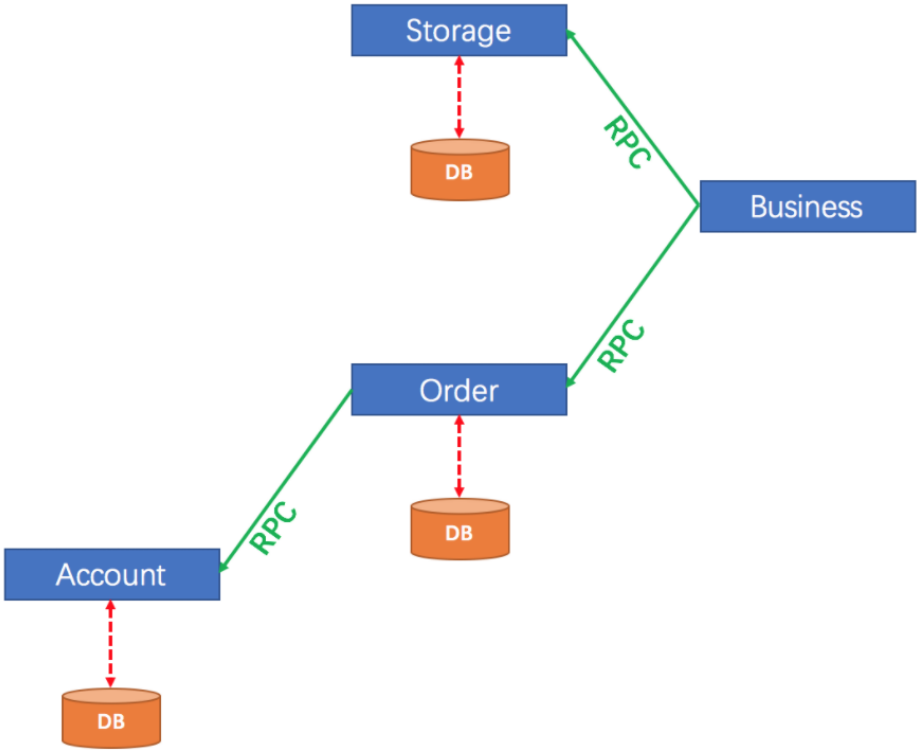
但是随着系统演变，我们单机应用可能会被拆分出相应的模块（N:N），订单模块连得是订单库，库存模块连得是库存库，余额模块连得是余额系统，那么以上业务场景就会牵扯到多数据源跨库的调用，物理上是三个不同的数据库，但是对于整个下单逻辑来说是一个整体。

牵扯到这样全局的，跨数据库的，多数据源的统一调度，这就是分布式事务的前身。

2、跨应用，跨数据库的痛点

单应用被拆分成微服务应用，原来三个模块被拆分成三个独立的应用，分别使用三个独立的数据源，

业务操作需要调用三个服务来完成。此时**每个服务内部的数据一致性由本地事务来保证，但是全局的数据一致性没法保证。**



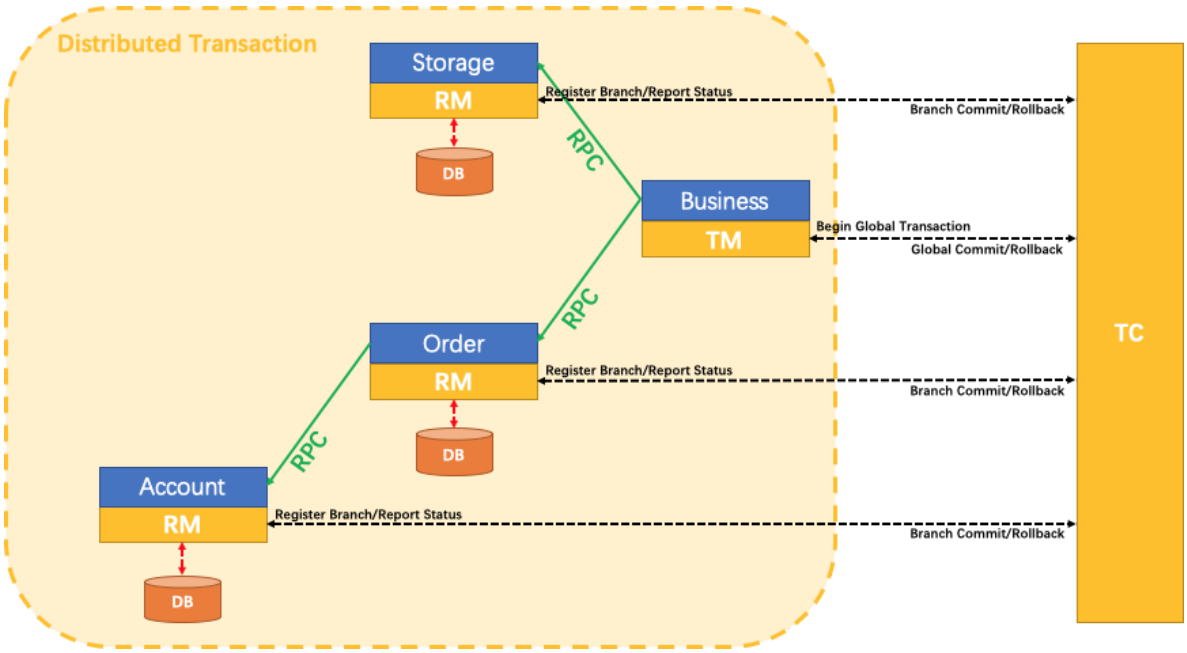
3、SpringCloud Alibaba Seata

> 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。

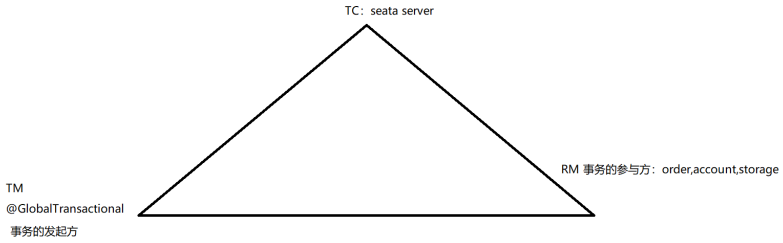
1、分布式处理过程

1. 全局唯一事务ID：Transaction ID XID,只要是同一个XID下，多个数据源的一致性都由这个它维护
2. 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。
3. 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。
4. 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。

2、处理过程



1. Tm向TC申请开启一个全局事务，全局事务创建成功并生成一个全局唯一的XID
2. XID在微服务调用链路的上下文中传播
3. RM向TC注册分支事务，将其纳入XID对应全局事务的管辖
4. TM向TC发起针对XID的全局提交或回滚
5. TC调度XID下管辖的全部分支事务完成提交或回滚



3、默认模式AT

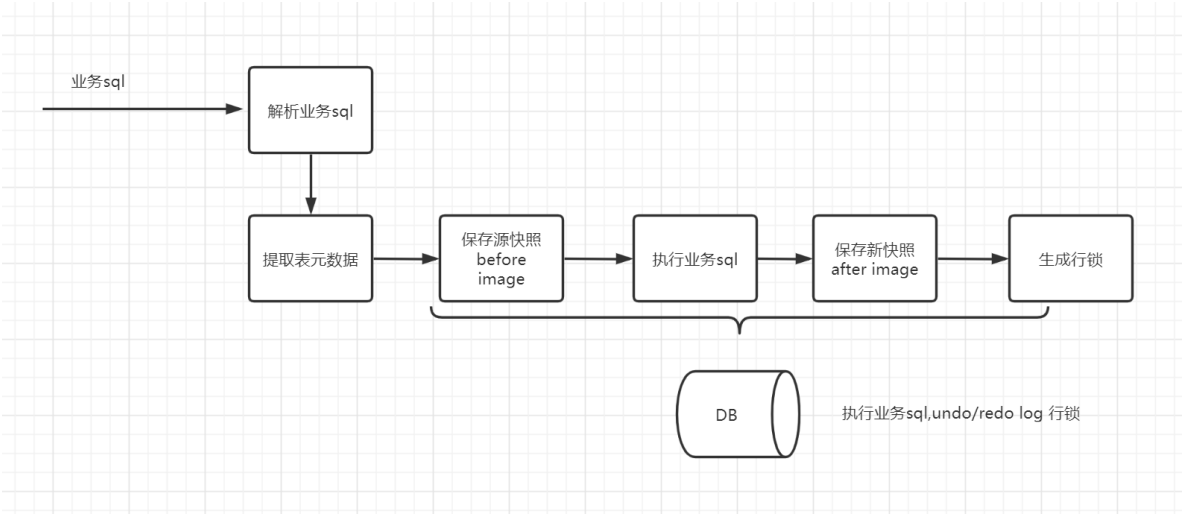
前提：基于支持本地 ACID 事务的关系型数据库。Java 应用，通过 JDBC 访问数据库。

整体机制：

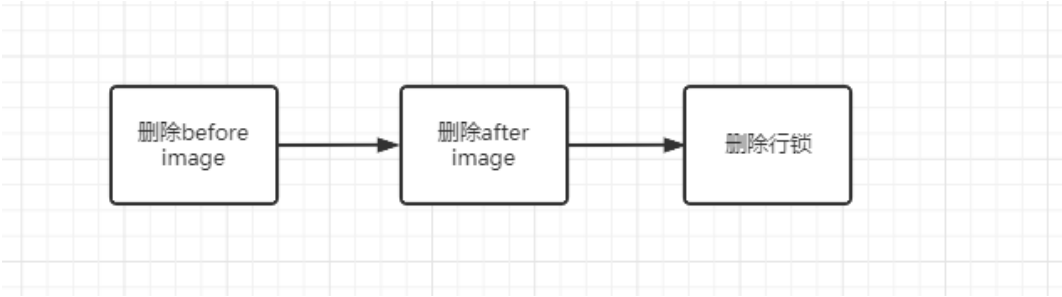
- 一阶段：业务数据和回滚记录在同一个本地事务中提交，释放本地锁和资源
- 二阶段：
  - 提交异步化，非常快速的完成
  - 回滚通过一阶段的回滚日志进行反向补偿

详细过程：

- 在一阶段，seata会拦截业务SQL
  - 解析sql语义，找到业务sql要更新的业务数据，在业务数据更新钱，将其保存成before Image
  - 执行业务sql，更新业务数据，在业务数据更新之后，将其保存成after image 最后生成行锁
  - 以上操作全部在一个数据库事务内完成，这样保证了一阶段操作的原子性

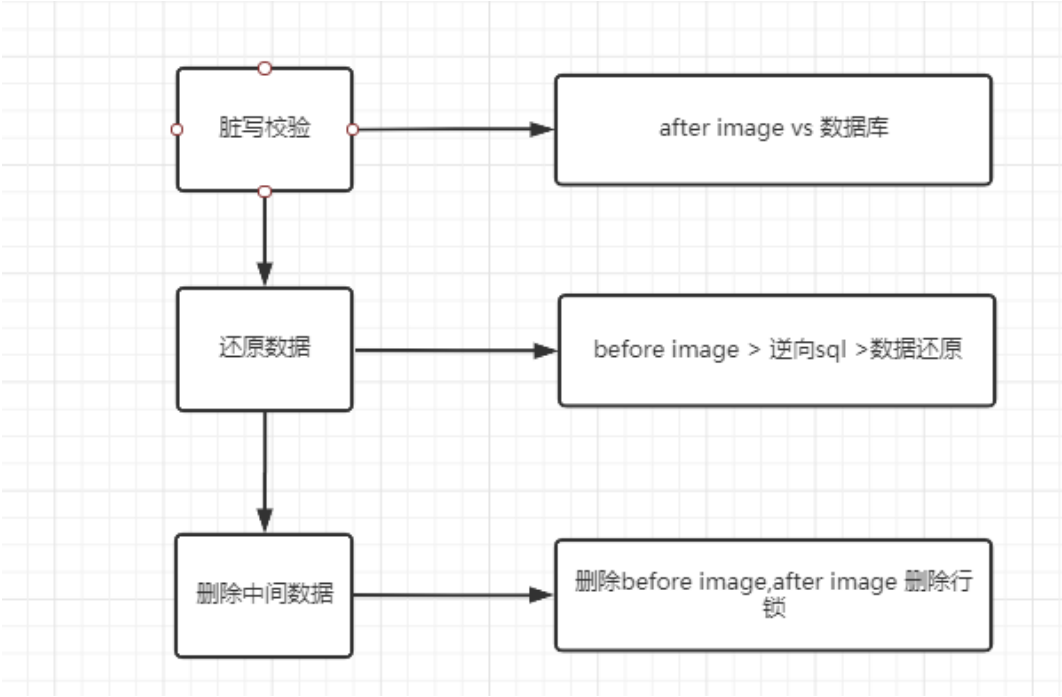


- 二阶段，业务流程成功执行，提交
  - 因为业务sql在一阶段已经提交到数据库，所以seata 只需要将一阶段保存的快照数据和行锁删掉，完成数据清理即可



- 二阶段 如果回滚
  - seata就需要回滚一阶段已经执行的业务sql，还原业务数据

    回滚方式便是用 before image 还原业务数据，但在还原前首先要进行脏写校验，对比当前数据局业务数据喝after image,如果两份数据完全一直说明没有脏写，可以还原数据，如果不一致就说明有脏写，这时候就要人工进行干预。



二、JSR303

> JSR-303 是JAVA EE 6 中的一项子规范，叫做Bean Validation，Hibernate Validator 是 Bean Validation 的参考实现 . Hibernate Validator 提供了 JSR 303 规范中所有内置 constraint 的实现，除此之外还有一些附加的 constraint。
>
> 简单来说就是JSR303是一个运行时的数据校验框架，数据验证失败后立即返回。

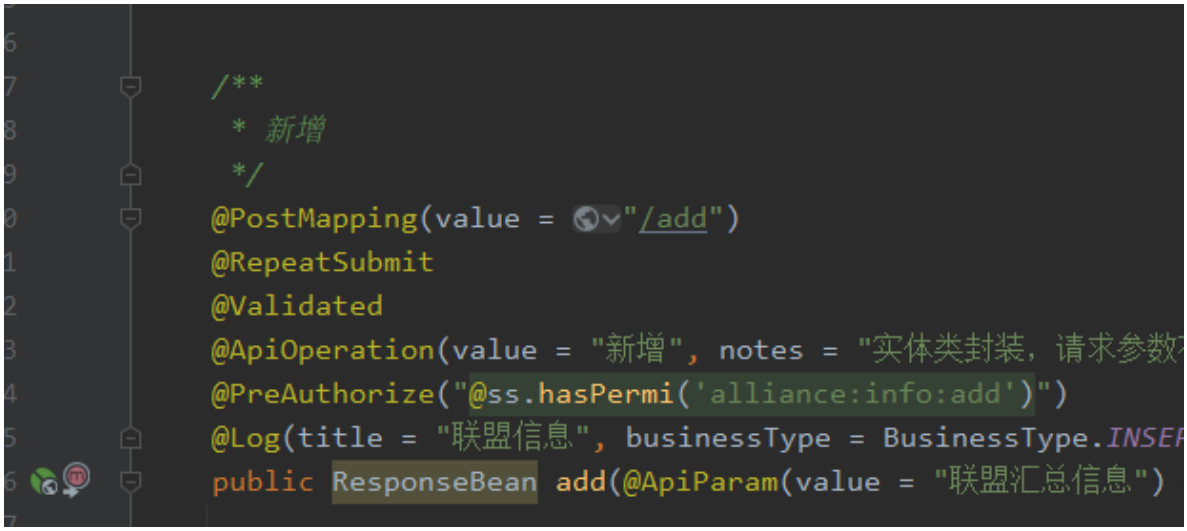
javax提供了@Valid（标准JSR-303规范），

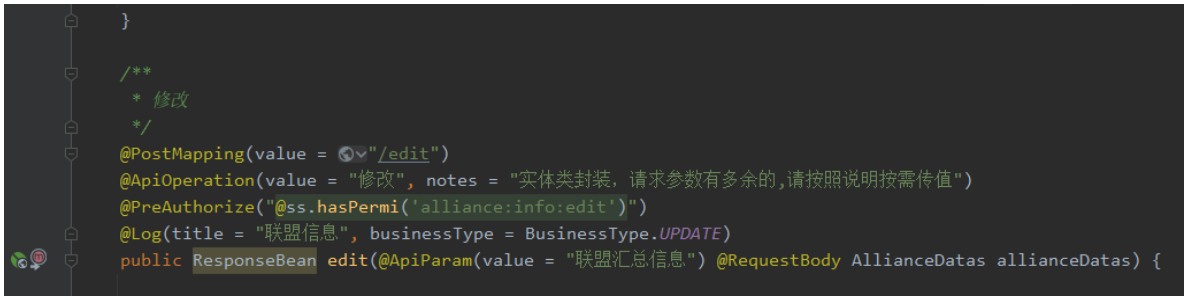
Spring Validation验证框架对参数的验证机制提供了@Validated（Spring's JSR-303规范，是标准JSR-303的一个实现)

优点：不用在业务层写相关的校验逻辑，校验写在实体bean

业务场景思考?

1、新增修改如何加





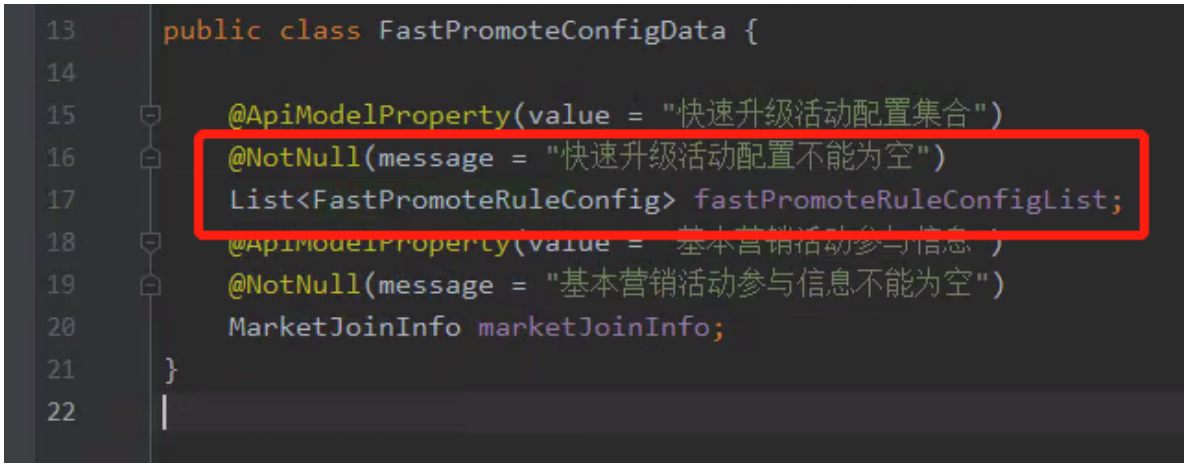
解决方法：分组校验

```
public @interface NotNull {
    String message() default "{javax.validation.constraints.NotNull.message}";

    Class<?>[] groups() default {};
```

```
 @NotNull(message = "修改必须指定id",groups = {UpdateGroup.class})
    @Null(message = "新增不能指定id",groups = {AddGroup.class})
    private Long id;
```

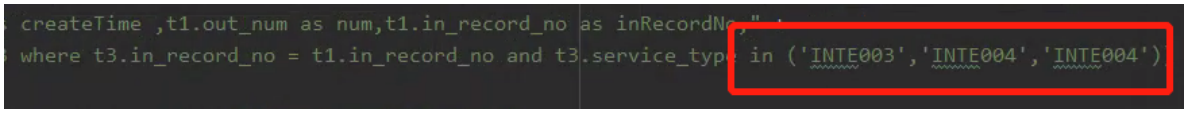
2、嵌套参数如何校验



@Valid

```
 /**
     * 校验嵌套级联属性必须在属性上加@Valid
     */
    @Valid
    @NotNull
    @Size(min=1)
    private List<AddressInfo> addressInfos;
```

3、自定义校验器




```
/**
 * @author : quanhz
 * @date : Created in 2021/2/19 11:47
 * @Description : 校验性别 0-女 1-男 2-未知
 */
@Documented
@Constraint(validatedBy = { GenderValueConstraintValidator.class })
@Target({ METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER, TYPE_USE })
@Retention(RUNTIME)
public @interface IntValue {

    String message() default "{com.hinz.jsr3.valid.GenderValue.message}";

    Class<?>[] groups() default { };

    Class<? extends Payload>[] payload() default { };

    int[] accessVals() default { };

}
```

```
@IntValue(accessVals = {1,2},groups = {UpdateGroup.class})
private Integer gender;
```

```
import javax.validation.ConstraintValidator;
import javax.validation.ConstraintValidatorContext;
import java.util.HashSet;
import java.util.Set;

public class GenderValueConstraintValidator implements
ConstraintValidator<IntValue,Integer> {

    private Set<Integer> set = new HashSet<>();
    //初始化方法
    @Override
    public void initialize(IntValue constraintAnnotation) {

        int[] vals = constraintAnnotation.accessVals();
```

```
            for (int val : vals) {
                set.add(val);
            }

        }

        //判断是否校验成功

        /**
         *
         * @param value 需要校验的值
         * @param context
         * @return
         */
        @Override
        public boolean isValid(Integer value, ConstraintValidatorContext context) {

            return set.contains(value);
        }
    }
```