快狗打车，数据库架构一致性最佳实践

快狗打车-沈剑

关于-我

- **“架构师之路”作者**，深夜写写技术文章
- 百度 - 高级工程师
- 58同城 - 高级架构师，技术委员会主席，技术学院优秀讲师
- 58到家 - 高级技术总监，技术委员会主席
- 快狗打车（原58速运） - CTO
- 本质：技术人一枚

目录

不一致的优化历程，也是数据库架构演进的过程

都有哪些坑在等着我们呢？

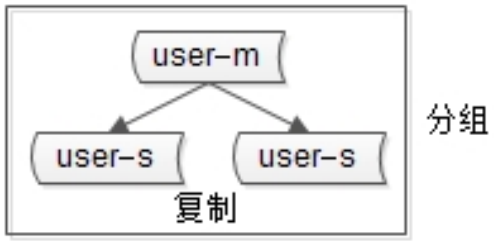
最初的数据库架构

单库

1. 系统架构如何？
2. 单库的特点如何？
3. 数据库最先碰到什么问题？

数据库读瓶颈，最先想到什么优化方案？

主从同步，读写分离，扩充读性能

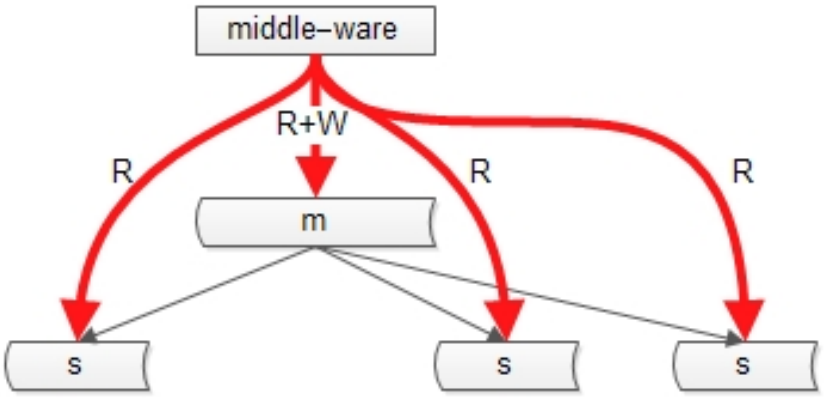


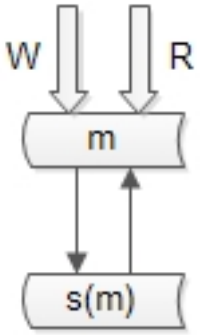
1. 主从的特点如何？
2. 主从会碰到什么问题？

主从不一致，如何解决？

解决主从不一致

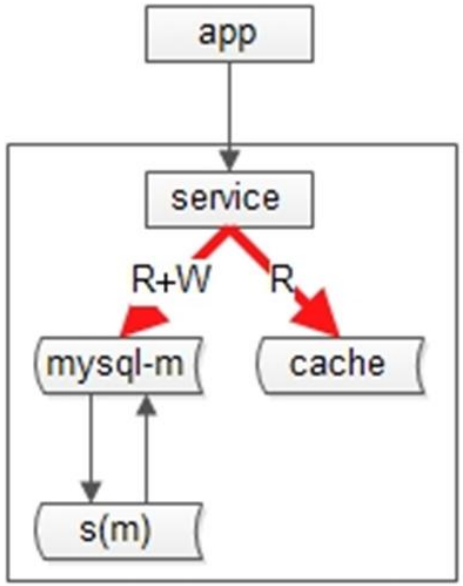
1. 中间件实践
2. 强制读主
3. 存在什么问题？





读性能，如何解决？

服务化+缓存

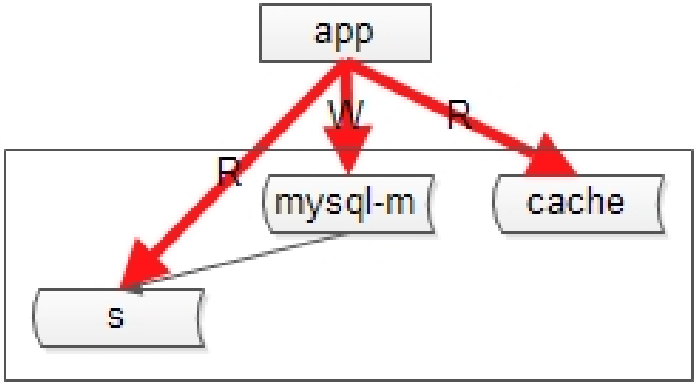


1. 通过缓存提升读性能？
2. 缓存会碰到什么问题？

缓存不一致，如何解决？

解决缓存与数据库不一致

1. Cache Aside Pattern
2. 为什么会不一致？
3. 如何解决？



常见玩法：缓存+数据

异步淘汰缓存，确保从库已经同步成功

设定超时时间，极限情况下有机会修正

数据量过大，新的瓶颈

2. 多实例，多库

多实例带来什么问题？

数据量大，怎么解决？

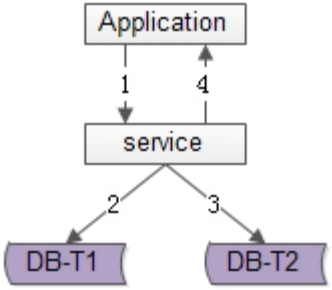
1. 分库

分库带来怎么问题？

怎么解决？

数据冗余

数据冗余带来什么问题？

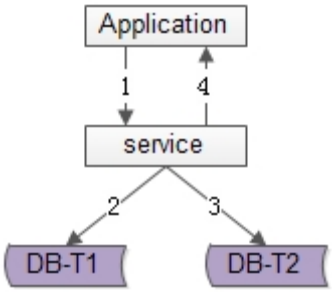


T1(oid, user_id, sj_id, xxoo)

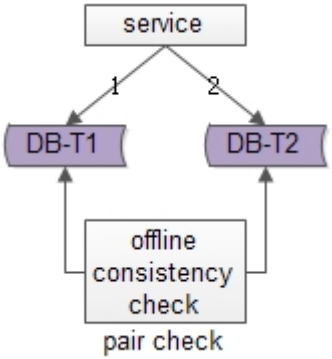
T2(sj_id, user_id, oid)

数据冗余，数据不一致，如何解决？

数据冗余，数据不一致，怎么解决（一）？

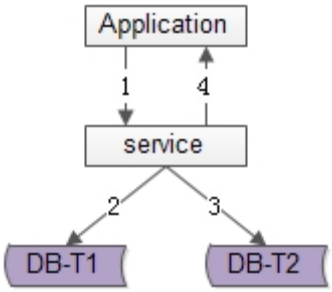


万一只有一半成功呢？

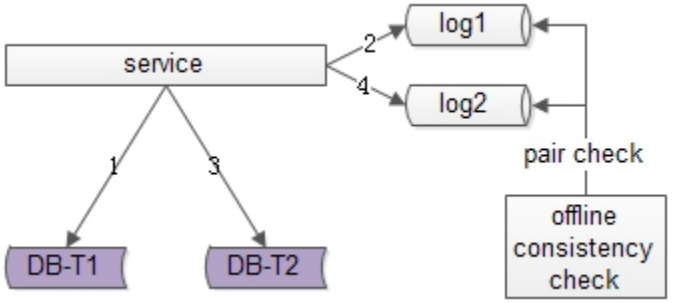


最终一致性：扫全量

数据冗余，数据不一致，怎么解决（二）？

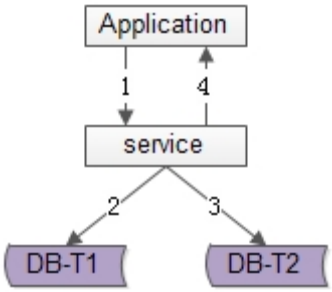


万一只有一半成功呢？

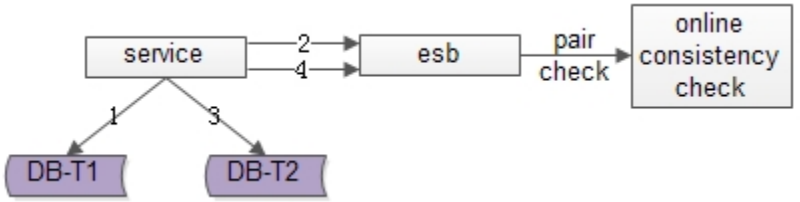


最终一致性：扫增量

数据冗余，数据不一致，怎么解决（三）？



万一只有一半成功呢？



最终一致性：实时消息对

多实例的问题，事务怎么办？

```
start transaction;
CURD table t_account; any Exception rollback;
CURD table t_order; any Exception rollback;
CURD table t_flow; any Exception rollback;
commit;
```

```
start transaction1;
// 第一个库事务执行
CURD table t_account;  any Exception rollback;
// 第一个库事务提交
commit1;

start transaction2;
// 第二个库事务执行
CURD table t_order;  any Exception rollback;
// 第二个库事务提交
commit2;

start transaction3;
// 第三个库事务执行
CURD table t_flow;  any Exception rollback;
// 第三个库事务提交
commit3;
```

多实例，多库事务，不一致，怎么办？

伪分布式事务，如何解决（一）？

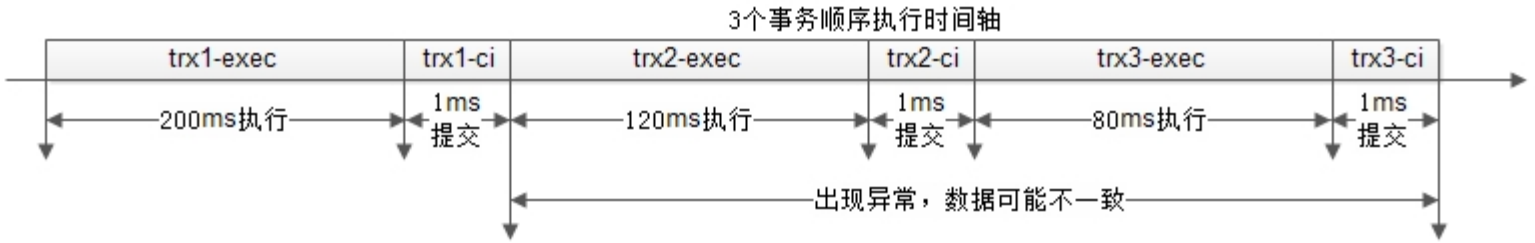
扣减余额
```
int Do_AccountT(uid, money){
start transaction;
// 余额改变money这么多
CURD table t_account with money;
any Exception rollback return NO;
commit;
return YES;
}
```

补偿事务，增加余额：
```
int Compensate_AccountT(uid, money){
// 做一个money的反向操作
return Do_AccountT(uid, -1*money){
}
```

Do_OrderT，新增一个订单
Compensate_OrderT，删除一个订单

Do_FlowT，新增一条流水
Compensate_FlowT，删除一条流水

伪分布式事务，如何解决（二）？



trx1.exec(); trx1.commit();
trx2.exec(); trx2.commit();
trx3.exec(); trx3.commit();

trx1.exec(); trx2.exec(); trx3.exec();
trx1.commit(); trx2.commit(); trx3.commit();

总结

读性能瓶颈

方案一：主从同步，读写分离->主从不一致

(1)中间件

(2)强制读主

方案二：服务化与缓存->缓存与数据库不一致

(1)异步淘汰

(2)设定超时时间

数据量太大

方案一：分库->数据冗余+数据不一致

(1)最终一致性：扫全量

(2)最终一致性：扫增量

(3)最终一致性：实时消息对

方案二：多实例->多库事务不一致

(1)补偿事务

(2)后置提交

Q&A

谢谢！

“架构师之路”公众号