DTCC

数，造，未，来

第十二届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2021

2021 年 8 月 18 日 - 20 日 | 北京国际会议中心

Spark
Hadoop
OLAP
HANA
MPP
Oracle

IT168.com ChinaUnix ITPUB

移动时代，从业务出发，性能优化与实践

沈剑

简介

- **“架构师之路”作者，深夜写写技术文章**
- [ex] 百度 - 高级工程师
- [ex] 58同城 - 高级架构师，技术委员会主席，技术学院优秀讲师
- 到家集团(原58到家集团) - 技术委员会主席
- **快狗打车(原58速运) - CTO**

这里联系我

目录

- 移动时代，用户体验的挑战
- 移动时代，提升用户体验，**常见架构方法（数据库）**
- 移动时代，提升用户体验，**性能优化实践（数据库）**
- 总结

移动时代，我们对APP的体验期待是什么？

- APP秒登陆
- APP页面秒开
- 省电
- 省流量

移动时代，潜在的困难是什么？

- 网络带宽受限（传输慢）
- 连接稳定性差（电梯，隧道，网随时断）
- 稍微量大就卡（和上两点有关）

导致的结果是什么呢?

- 登录耗时长
- 页面卡顿
- 丢消息

问题一：为什么登录耗时长？

登录过程在干嘛?

(内心OS：不就是验证个用户名密码，至于这么久吗)

登陆耗时长，是在做**数据同步**，为登录后的页面做数据准备

登录过程在准备/同步什么数据?

（内心OS：这是一个业务问题）

登陆过程数据同步

- 手Q为例

- 好友（用于好友列表展现）

- 分组（用于分组列表展现）

- 群组，群友（用于群组，群友展现）

- 消息（用于离线消息展现）

...

- 快狗打车为例

- 车型（用于车型展现）

- 收藏司机（用于收藏司机列表展现）

- 订单（用于订单列表展现）

- 优惠券（用于优惠券列表展现）

...

需要在登陆过程中，准备好全部的数据吗？

优化一：延迟拉取，按需拉取

- 原则：需要的时候再拉取
- 拉取哪些：首屏展示相关的

- 快狗为例

- 车型，**提前**拉取
- 收藏司机，**延迟**拉取
- 订单，**延迟**拉取
- 优惠券，**延迟**拉取

...

延迟拉取存在什么问题？

延迟拉取的不足

- 并不减少需要同步的数据量
- 只是分散了拉取时间
- 仍会出现什么问题呢?

问题二：页面卡顿

（延时并没有减少数据量，数据量大时，该卡的时候还是会卡）

能不能减少需要同步的数据量呢?

优化二：时间戳（版本号）

- 原则：只拉取变化的数据，而不是全部的数据

拉取什么数据，怎么设计时间戳（版本号）？

通常要拉取两种数据

- 关系数据，一般是key -> id-list结构
- 详情数据，一般是key -> object结构

- 关系数据举例：好友列表，群列表

t_friends(uid, friend_uid)
100, 101
100, 102
100, 103

t_groups(uid, group_id)
100, 1001
100, 1002
100, 1003
101, 1001
102, 1001

- 详情数据举例：好友详情，群详情

user_info(uid, name, age, sex, intro)
100, shenjian, 18, no, NULL
101, zhangsan, 19, no, NULL
102, lisi, 18, no, NULL
103, wangwu, 35, yes, NULL

group_info(gid, name, intro)
1001, dota爱好者协会, NULL
1002, 单身RD, NULL
1003, 读书会, NULL

两种数据的时间戳设计

- 关系数据，时间戳表
- 详情数据，时间戳属性
- 什么时候修改时间戳表?
- 什么时候修改时间戳属性?

- 时间戳表举例

t_friends(uid, friend_uid)
100, 101
100, 102
100, 103

t_groups(uid, group_id)
100, 1001
100, 1002
100, 1003
101, 1001
102, 1001

t_list_timestamp(uid, friends_v, groups_v)
100, v1, v10
101, v1, v1
102, v1, v100
103, v1, v1

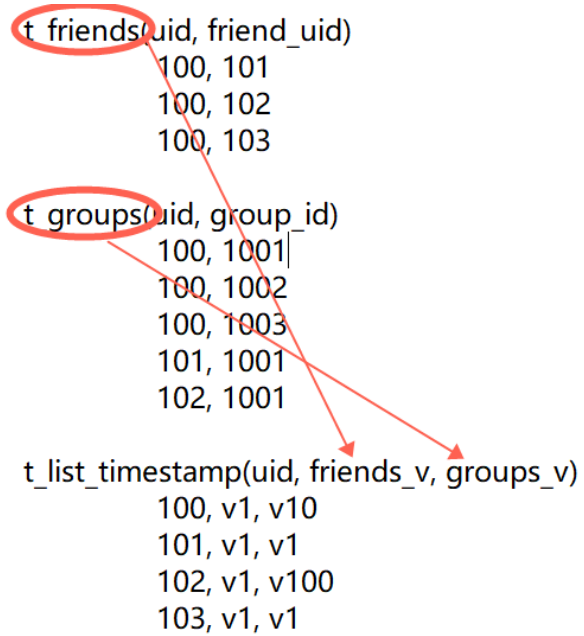
- 时间戳属性举例

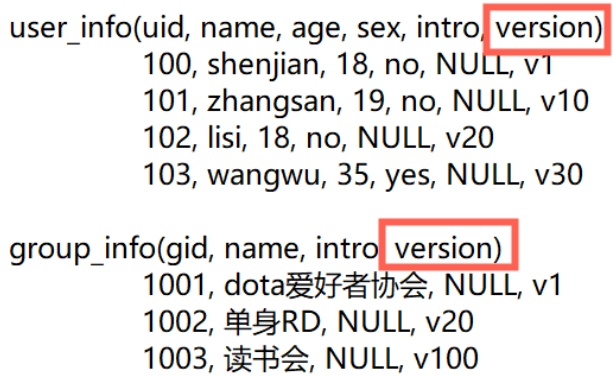
user_info(uid, name, age, sex, intro, version)
100, shenjian, 18, no, NULL, v1
101, zhangsan, 19, no, NULL, v10
102, lisi, 18, no, NULL, v20
103, wangwu, 35, yes, NULL, v30

group_info(gid, name, intro, version)
1001, dota爱好者协会, NULL, v1
1002, 单身RD, NULL, v20
1003, 读书会, NULL, v100

如何通过时间戳，来减少数据同步的数据量？

数据同步步骤升级





- 原步骤

（1）拉取好友列表

（2）拉取群列表

（3）拉取好友详情

（4）拉取群详情

- 加入时间戳之后步骤

（1）**拉取**时间戳表

（2）本地时间戳比对，时间戳变化，才**拉取**相应的列表（**90%概率不变**）

（3）**拉取**好友详情时间戳属性

（4）本地时间戳比对，时间戳变化，才**拉取**相应的好友详情（**90%的数据不变**）

（5）**拉取**群详情时间戳属性

（6）本地时间戳比对，时间戳变化，才**拉取**相应的群详情（**90%的数据不变**）

能够减少90%的同步数据量

还有没有优化空间?

优化三：上传时间戳（版本号）

- 方法：

由拉取服务端时间戳，客户端本地比对版本号，再拉取差异数据

升级为

上传客户端时间戳，服务端比对版本号，直接返回差异

t_friends(uid, friend_uid)
100, 101
100, 102
100, 103

t_groups(uid, group_id)
100, 1001
100, 1002
100, 1003
101, 1001
102, 1001

t_list_timestamp(uid, friends_v, groups_v)
100, v1, v10
101, v1, v1
102, v1, v100
103, v1, v1

user_info(uid, name, age, sex, intro, version)
100, shenjian, 18, no, NULL, v1
101, zhangsan, 19, no, NULL, v10
102, lisi, 18, no, NULL, v20
103, wangwu, 35, yes, NULL, v30

group_info(gid, name, intro, version)
1001, dota爱好者协会, NULL, v1
1002, 单身RD, NULL, v20
1003, 读书会, NULL, v100

上传时间戳，减少交互

- 拉取时间戳步骤

（1）**拉取**时间戳表

（2）本地时间戳比对，时间戳变化，才**拉取**相应的列表（**90%概率不变**）

（3）**拉取**好友详情时间戳属性

（4）本地时间戳比对，时间戳变化，才**拉取**相应的好友详情（**90%的数据不变**）

（5）**拉取**群详情时间戳属性

（6）本地时间戳比对，时间戳变化，才**拉取**相应的群详情（**90%的数据不变**）

- 上传时间戳步骤

（1）**上传**时间戳表，服务器时间戳比对，如果变化，返回列表（**90%概率不变**）

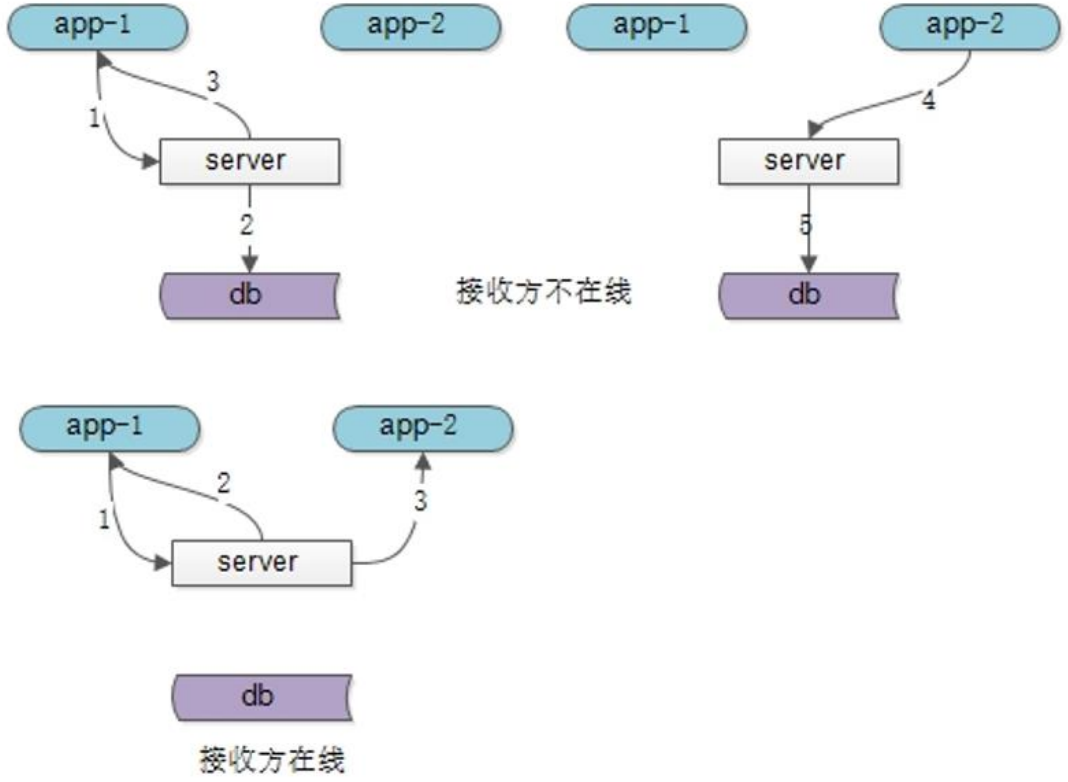
（2）**上传**好友详情时间戳属性， 服务器时间戳比对，如果变化，返回差异（**90%的数据不变**）

（3）**上传**群详情时间戳属性， 服务器时间戳比对，如果变化，返回差异（**90%的数据不变**）

问题三：移动环境，为什么容易丢消息?

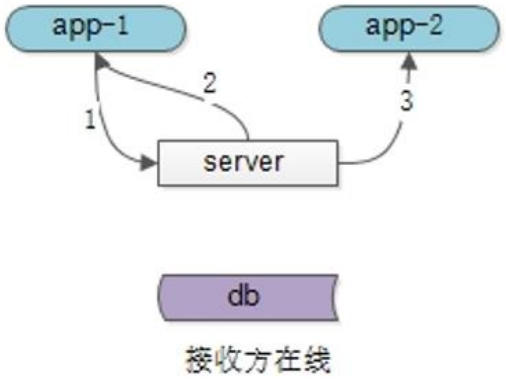
正常离线，在线消息投递流程

- 消息接收方离线时：先存，登陆时再拉
- 消息接收方在线时：直接投递



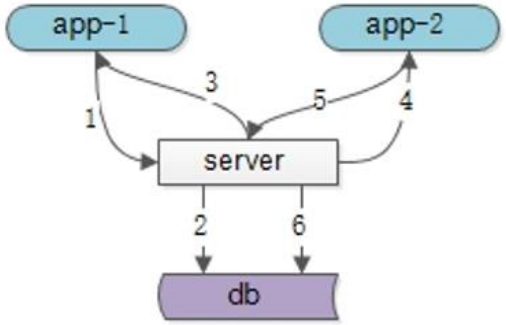
为啥容易丢消息?

- 消息接收方在线时：网容易抖，步骤3容易丢消息
- 怎么办?



优化四：先落地，再应用层ACK

- 原则：即使接收方在线，也将消息先落地，收到后再ACK删除



接收方在线

总结

移动时代，新问题，新挑战

- 登录耗时长
- 页面卡顿
- 丢消息

优化一：数据同步很耗时，延迟拉取，按需拉取

- 原则：需要的时候再拉取
- 拉取哪些：首屏展示相关的

- 快狗为例

- 车型，**提前**拉取
- 收藏司机，**延迟**拉取
- 订单，**延迟**拉取
- 优惠券，**延迟**拉取

...

优化二：减少数据同步量，时间戳（版本号）

- 原则：只拉取变化的数据，而不是全部的数据

两种数据的时间戳设计细节

- 关系数据，时间戳表
- 详情数据，时间戳属性
- 列表改变，修改时间戳表
- 详情改变，修改时间戳属性

- 时间戳表举例

t_friends(uid, friend_uid)
100, 101
100, 102
100, 103

t_groups(uid, group_id)
100, 1001
100, 1002
100, 1003
101, 1001
102, 1001

t_list_timestamp(uid, friends_v, groups_v)
100, v1, v10
101, v1, v1
102, v1, v100
103, v1, v1

- 时间戳属性举例

user_info(uid, name, age, sex, intro, version)
100, shenjian, 18, no, NULL, v1
101, zhangsan, 19, no, NULL, v10
102, lisi, 18, no, NULL, v20
103, wangwu, 35, yes, NULL, v30

group_info(gid, name, intro, version)
1001, dota爱好者协会, NULL, v1
1002, 单身RD, NULL, v20
1003, 读书会, NULL, v100

优化三：减少交互次数，上传时间戳（版本号）

- 方法：

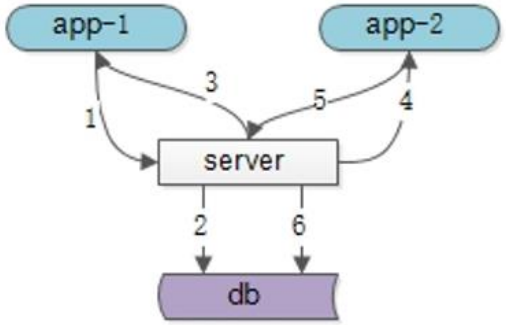
由拉取服务端时间戳，客户端本地比对版本号，再拉取差异数据

升级为

上传客户端时间戳，服务端比对版本号，直接返回差异

优化四：保证消息可达性，先落地，再应用层ACK

- 原则：即使接收方在线，也将消息先落地，收到后再ACK删除



接收方在线


THANKS
时序
开源
自研
文档
Spark
HANA
SQL
OLAP
Oracle
Hadoop
MPP
DB2