分享一次 ShardingJDBC 亿级数据分表真实经验！干货满满

前言

从入职以来写了一年的业务代码，突然接到来自领导的技术需求，说要给我们的借款、还款申请单分表。查看了一下借款表，只有几千万数据，再看还款表已经两亿多了，为了提高数据查询效率，降低数据库的压力。确实可以考虑分表了。另外.....这是入职一年以来第一个非业务需求！

设计方案

开始编码实现之前我们需要先做系统设计，主要是以下几块内容要跟领导开会对齐颗粒度！

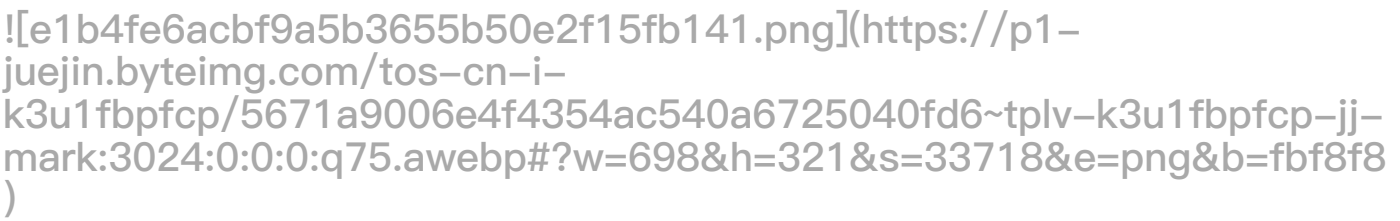
分表数据库基本信息

* 分表数据库使用新的实例，独立数据源
* 是否需要分库：不需要，只分表， `50` 张表足矣
* 分表数据库名 `tcbiz_ins_cash_mas_split` （领导定的名字，无特殊含义）
* 分表总数 `50` 张表
* 分表表名 `CashRepayApplySplit0${0..9}` ,`CashRepayApplySplit${10..49}`（对，你没看错，我司数据库表名是驼峰，字段也是驼峰）
* 分表列 `memberId` ，分表策略 `memberId` 截取后两位 `% 50` ，余数小于 `10` 左边补 `0` ，因为我们表名是两位数字后缀。
* 表结构，和原表结构完全一致，包括字段、索引等。哎等等，发现原表 `uptTime` 更新时间字段竟然没有索引，这里我们分表需要加上，注意主表不要动，几个亿数据的表不能随便加索引。

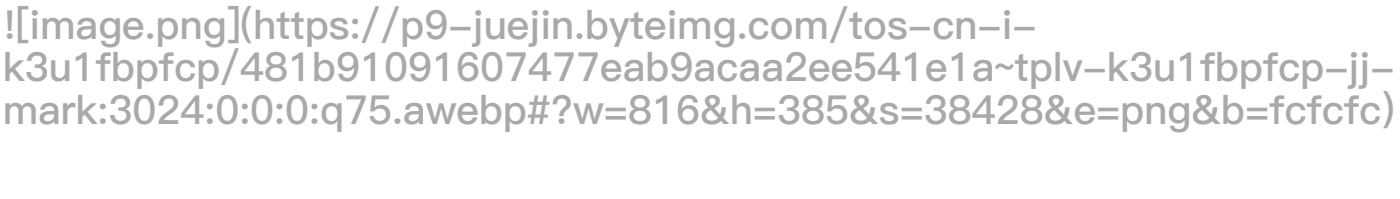
历史数据同步

这一点是非常重要的，分了五十张表之后，不仅新的业务数据要根据分表策略落入分表，也要使用手段将存量数据以分表策略优先写入 `CashRepayApplySplit0${0..9}`,`CashRepayApplySplit${10..49}`。

有一件比较尴尬的事情，我司早几年已经做过一次还款申请单表的数据迁移，最原始的表名是 `CashRepayInfo`，四年前迁移后的表名是 `CashRepayApply`，也就是当前主业务表，目前我们的业务数据都是进行双写，先写到 `CashRepayApply`，然后同步到 `CashRepayInfo` ，保证 `CashRepayInfo` 里面是最全的数据，因为虽然 `CashRepayApply` 是主业务表，但是很多历史业务代码、报表等查询包括外部部分还是使用 `CashRepayInfo` 的。



这是我们当前的双写方式。但是有一部分最古老的历史数据存在于 `CashRepayInfo`，不在 `CashRepayApply`。当时迁移的时候没有完全同步，三者的关系如下图



上图可以知道，`CashRepayInfo` 是全量数据的表，包含 `CashRepayApply`，我们的目标是将全量数据同步到分表。所以现在要实现的是

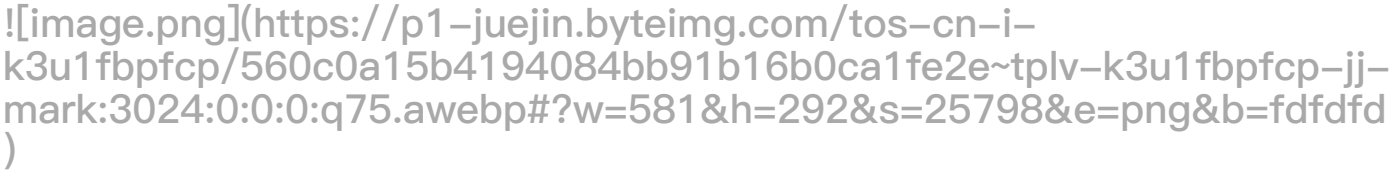
1. 先把 `CashRepayApply` 缺少的那部分最早的 `CashRepayInfo` 数据同步到 `CashRepayApplySplit`
2. 将 `CashRepayApply` 剩余的数据全部同步到 `CashRepayApplySplit`

疑问解答

这里可能会有一个疑问，既然 `CashRepayInfo` 是全量的数据，为什么不直接从 `CashRepayInfo` 同步到分表 `CashRepayApplySplit` 呢？这是因为数据量有两亿多，我们不可能全部让研发用代码同步，那就只能交给 `DBA` 。但是 `DBA` 同步的话存在一个问题就是他需要两边表的字段结构一致，但是 `CashRepayInfo` 和 `CashRepayApply` 是存在字段差异的，字段名称不同、字段个数也有略微差异。综合考虑之后使用上述方案。

具体细节

我们把下面这张图的两个箭头看做两张表的自增数据，如果要实现上述第一点，就需要找到垂直的黑色虚线与 `CashRepayInfo` 交点的 `id` 是多少。`CashRepayInfo` 表这个 `id`之前的数据就是我们需要使用代码完全同步到 `CashRepayApplySplit` 的数据。



用这个 `id` 来找到对应的 `CashRepayApply` 表的主键 `id`。`CashRepayApply.id` 以后的全部数据就是 `DBA` 需要帮我们同步的。

现有后台查询改造

目前现在公司的客服/运营后台管理系统全都是用单表去直接 `join` 的，如果分表之后，肯定没办法再以原来的展示维度去 `join` 查询了，那么需要定一个方案来解决这个问题。通过与领导沟通，暂定的方案是老表 `CashRepayApply` 只保留两~三年的数据，这部分数据可以像原来一样不指定 `memberId` 去 `join` 查询。

再历史（三年以前）的数据必须通过 `memberId` 查询，管理系统提供新的查询页面，必填条件 `memberId`。

外部部门通知

全量数据同步到分表之后，最老的 `CashRepayInfo` 逐步等待下线、废弃。所以要和其他部门比如风控、大数据部门沟通，告知他们后续报表等逻辑要用新的表 `CashRepayApplySplit` 查询，现在可以开始逐步切换了。

DBA 操作过程中新产生业务数据同步方案

前面我们已经定好了同步步骤，第一步是研发自己同步一部分，第二步给到一个起始 `id` 给 `DBA`，`DBA` 从这个 `id` 开始同步 `CashRepayApply` 表剩余的数据到分表。这里有个问题就是 `DBA` 的结束 `id` 是不确定的，因为 `CashRepayApply` 这张表在 `DBA` 操作同步的过程中一直都有新的业务数据写入。`DBA` 同学在开始操作之前必须要给定一个结束 `id` 给到同步工具，但

是新业务的一直写入导致 `DBA` 同步必定会漏一部分数据。

我们总不能为了这个数据同步，停止用户的还款对不对，所以我给 `DBA` 的方案是，让 `DBA` 同步的最晚数据是 `operatorDate – 5 Day`。筛选数据库 `uptTime < 操作时间减去五天` 的数据，这样得到一个确定性的结束 `id`。当 `DBA` 操作结束后还会剩一部分刚刚操作过程中产生的最新的业务数据（下图最右边的虚线数据，我色弱不太认识颜色），那这部分数据依然是研发自己用代码同步，等晚上 `23:00` 关闭还款之后研发用功能代码同步。

这样一来我们所有存量数据就按照创建时间排序，全量的同步到分表了。然后就可以开启三写的开关，完美完成这次数据库分表迁移！

数据三写同步

表的下线需要时间，其他部门改造业务切换分表也需要时间，所以在未来的一段时间内，我们仍然要保证 `CashRepayInfo` 数据的完整性，我们三张表要同步三写，先写 `CashRepayApply`、再写 `CashRepayInfo`、再写 `CashRepayApplySplit`

同步的时候要注意，由于分表的库是不同的数据源，需要声明指定的事务管理器。

```
@Transactional(propagation = Propagation.REQUIRES_NEW, transactionManager = "transactionManagerSplit")
```

另外，不要问我为什么在代码中实时同步数据，而不用一些中间件？比如监听

`MySQL` 的 `binlog` 去同步？因为 `DBA` 告诉我不好实现（我严重怀疑是这个 `DBA` 小姐姐不想帮我弄.....对，我司 `DBA` 是个小姐姐）那就只能研发自己来了。

同步数据的动态开关

注意我们需要提供一个动态开关去控制开启和关闭新的业务数据从 `CashRepayApply` 同步到 `CashRepayApplySplit` 分表，也就是双写的开关，因为需求上线之后肯定是先同步一阶段古老的数据，再同步二阶段 `DBA` 可以同步的数据，然后三阶段研发同步新产生的部分业务数据，全部完毕之后开启同这个开关完成无缝对接。

最终的目的除了完全完成数据同步之外，还有一点就是让越早的数据越在表的前面。

定时任务扫描

由于我们是在代码中去双写数据到分表，分表数据库是新的实例，和原业务表的的操作不能控制在一个事务中，所以这就有潜在的隐患导致数据写到 `CashRepayApply` 表，未成功/正确写入到 `CashRepayApplySplit` 分表。尽管概率很小，我们也要预防。

所以前面在我们双写的时候一定要捕捉写入到分表的异常，确保即使写入分表失败，也不能影响主业务流程。然后每天用定时任务扫描今日产生的还款申请单数据，`CashRepayApply` 和 `CashRepayApplySplit` 做比对，是否存在差异字段，如果有，推送告警出来研发排查。

艰难的 Demo 之路

因为公司的项目比较老，`shardingsphere` 的版本也比较低，为了紧贴社会潮流，这篇文章的 `Demo` 我是自己选了一个相对比较新的版本 `SpringBoot3` 去整合，然后呢发现官方有 `sharding-jdbc-spring-boot-starter` 我就拿过来用了，二话不说直接上了最新版本，我想着最新版本肯定能兼容 `SpringBoot3` 呀。

然而不出意外的话意外就出现了，启动一直报错，于是我去 `github` 上找 `issue`。 第一次没找到类似的报错。于是我专门提了一个 `issue`，第二天我又去看 `issue` 发现了有人提过的 [# Is ShardingSphere 4.1.1 version

compatible with Spring Boot 3.0.5 version?](http://cxyroad.com/”https://github.com/apache/shardingsphere/issues/27597”) 在这里可以看到官方回复说因为 `SpringBoot` 版本的迭代导致他们为了维护 `starter` 会消耗很多人力财力。所以关于 `shardingsphere` 的 `spring-boot-starter` 从 `5.3.0` 版本就不更新了。官方推荐使用 `5.3.0` 以上的版本去适配 `SpringBoot3` ，并且使用 `ShardingSphereDriver` 的方式去集成 `shardingsphere`。官网也有配置示例。

于是我按照官方的文档引入 `Maven` 坐标，照着文档配置，但中途还是遇到了很多问题.....各种版本兼容问题，我想吐槽一下官网的配置文档，下面会一一列出，这个 `Demo` 做的还是蛮艰难的！

分表策略 & 代码实现

上一节已经和领导开会评审了我们的设计，领导给出赞赏的目光，设计考虑的很全面很周到！那么接下来我们开始示例代码整合 `shardingsphere` 完成数据库分表。

各技术组件版本

`SpringBoot 3.2.4 + JDK19 + MySQL8 + shardingsphere 5.4.1 + MyBatisPlus 3.5.5`

Maven 依赖

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.19</version>
</dependency>

<dependency>
    <groupId>org.apache.shardingsphere</groupId>
    <artifactId>shardingsphere-jdbc-core</artifactId>
    <version>5.4.1</version>
</dependency>

<!--mybatisplus3-->
```

```
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-spring-boot3-starter</artifactId>
    <version>3.5.5</version>
</dependency>

<!--缺少会报错-->
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>2.3.3</version>
</dependency>
```

application.yml 配置文件

```
spring:
  application:
    name: sharding-jdbc-demo
  datasource:
    driver-class-name:
org.apache.shardingsphere.driver.ShardingSphereDriver
    url: jdbc:shardingsphere:classpath:sharding-config.yaml #分表配置文件
  profiles:
    active: default

mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

sharding-config.yaml 文件

```
dataSources:
  ds0:
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driverClassName: com.mysql.cj.jdbc.Driver
    jdbcUrl:
```

```
jdbc:mysql://localhost:3306/tcbiz_ins?characterEncoding=utf8&useAffectedRows=true&useSSL=false&serverTimezone=Asia/Shanghai&allowPublicKeyRetrieval=true
    username: root
    password: 你的数据库密码

rules:
  - !SHARDING
    tables:
      #表名
      cash_repay_apply:
        #数据节点，所有表名
        actualDataNodes: ds0.cash_repay_apply_0$->{0..9},ds0.cash_repay_apply_$->{10..49}
        #分表策略
        tableStrategy:
          #分表类型，单列作为分表键
          standard:
            shardingColumn: member_id #分表列
            shardingAlgorithmName: memberId-suffix-mode-50 #分表算法名称，与下面对应
    shardingAlgorithms:
      memberId-suffix-mode-50:
        type: INLINE
        props:
          #分片表达式，groovy 脚本
          algorithm-expression: cash_repay_apply_$->{String.valueOf(Long.parseLong(String.valueOf(member_id).substring(String.valueOf(member_id).size() - 2)) % 50 ).padLeft(2,'0')}

props:
  sql-show: true


```

到这其实配置就结束了，接下来我们写个方法访问数据库

```
@Mapper
public interface CashRepayApplyMapper extends BaseMapper<CashRepayApply> {

@Select("select * from cash_repay_apply where member_id = #{memberId}")
List<CashRepayApply> testSharding(@Param("memberId") String
```

memberId);
}

```

调用这个方法即可，`shardingsphere` 会自动帮我们创建分表数据源，路由对应的分表。

实测发现这里 `SpringBoot3.2.4` 会报以下错

```
java.lang.NoSuchMethodError:
org.yaml.snakeyaml.representer.Representer: method 'void <init>()' not found
```

这是因为 jar 包版本问题导致的，从网上看到一个很简单的解决办法，直接把报错的这个类拷贝，然后粘贴到本项目中，包名要和它完全一致，然后添加一个无参构造方法覆盖原 jar 包中的类

```
public Representer( ) {
super(new DumperOptions());
this.representers.put(null, new RepresentJavaBean());
}
```

再次启动发现已经可以自动根据 `member_id` 路由到分表了。

踩坑之路

第一个坑是当前 `SpringBoot 3.2.4` 版本 `Maven` 需要引入

```
<dependency>
    <groupId>com.sun.xml.bind</groupId>
```
```

```
    <artifactId>jaxb-impl</artifactId>
    <version>2.3.3</version>
</dependency>
```

第二个坑是 `org.yaml.snakeyaml.representer.Representer` 的无参构造不存在

第三个坑是 `algorithm-expression` 分片表达式的脚本，在使用 `sharing-column % 50` 的时候可能会报一个错

```
java.sql.SQLException: Inline sharding algorithms expression `cash_repay_apply_${member_id % 50}` and sharding column `member_id` do not match.
```

这是因为 `groovy` 脚本工具解析这个表达式报错了，断点打过去发现异常里面给我们提示一句话

```
groovy.lang.MissingMethodException: No signature of method: java.lang.String.mod() is applicable for argument types: (Integer) values: [50]
Possible solutions: md5(), drop(int), drop(int), any(), find(), any(groovy.lang.Closure)
```

猜测可能是 `groovy` 的解析工具版本有什么升级，去官方 `github` 下找到了别人提的 `issue` 官方让用 `Long` 解析一下 `member_id` 的类型 。改成下面的写法

```
algorithm-expression: cash_repay_apply_$->{Long.parseLong(member_id) % 50}
```

因为我这里数据库里面 `member_id` 是 `varchar` ，我们是后两位对 `50` 取模，可能会存在小于 `10` 的数据，所以为了映射表名要在前面补 `0` 。使用 `groovy` 脚本就是我上面的配置代码。

Demo 源码下载

`Demo` 已分享到 `github` [点击去 GitHub 下载](http://cxyroad.com/”https://github.com/yanzhisishui/sharding-jdbc-demo”)

研发同步数据的代码

上面咱们已经说过了有一部分数据是需要研发自己去同步的，这部分同步数据的代码应该如何写呢。最早我是想用 `ForkJoinPool` 工具类实现的，因为这种大数据量的分治太适合了。可以参考这篇文章 记录一次发送千万级别数量消息的定时任务优化。

但是考虑到以下三点：

* 我们核销业务交易项目比较古老了，使用的还是架构组提供的自研的定时任务，没法做分片参数
* 核心交易服务不能出现差错，尽量不要在这个项目中过多使用线程池，消耗 `CPU` 资源
* 尽量让产生的数据落入到分表时，创建时间早的分布在表的前面

最终我还是选择了使用朴实无华的方式，只用一个线程去跑批同步数据，每次跑 `500` 条，依次循环往下跑，直到结束。

```
/**
  * 同步数据示例代码
  */
public void dataHandle(Long startId, Long endId) throws
AppBizException {
log.info(”CashRepayApplyServiceImpl#dataHandle start startId-{},endId-
{}”, startId, endId);
if (endId <= 0L) {
endId = null;
} else if (startId > endId) {
```

```
throw new
AppBizException(ErrorCodeEnum.PARAM_CHECK_ERROR.getCode(), ”起
始id不能大于结束id”);
}
//查询条件
QueryCashRepayInfoCondition condition = new
QueryCashRepayInfoCondition();
condition.setGteId(startId);
condition.setLteId(endId);
condition.setOrders(”id+”);//id正序排序
List<CashRepayInfo> infoList = cashRepayInfoDao.query(condition, 0, –
1);//公司内部持久层框架，最多查询条数 500
long lastId; //结束 id
while (CollectionUtil.isNotEmpty(infoList)) {
lastId = infoList.get(infoList.size() – 1).getId() + 1;//下次循环的起始id
infoList.forEach(history –> {
try {
if(StringUtil.isBlank(history.getMemberId()) ||
StringUtil.isBlank(history.getRepayNo())){
log.error(”CashRepayApplyServiceImpl#dataHandle error memberId or
repayNo is null id–{}”,history.getId());
return;
}
//分表查询条件
QueryCashRepayApplySplitCond splitCond = new
QueryCashRepayApplySplitCond();
splitCond.setMemberId(history.getMemberId());
splitCond.setRepayApplyNo(history.getRepayNo());
CashRepayApplySplit exist =
cashRepayApplySplitDao.getUnique(splitCond);
CashRepayApplySplit splitData = buildCashRepayApplySplit(history);
if (exist == null) {
cashRepayApplySplitDao.add(splitData);
} else {
splitData.setId(exist.getId());
cashRepayApplySplitDao.update(splitData);

}
} catch (Exception e) {
log.error(”CashRepayApplyServiceImpl#dataHandle error id–{},repayNo–
{}”,history.getId(),history.getRepayNo());
throw new RuntimeException(e);
}
});

LOG.info(”dataHandle finish startId–{},endId–
{}”,condition.getGteId(),endId);
//每 500 条查询一次缓存是否要终止循环，因为这是一个几千万数据的接口
```

，为了可控，要提供一个能近乎实时结束的功能
String redisCache =
RedisCacheUtils.getRedisCache(CashApplyRedisKeyConsts.TERMINATE_
SPLIT_SYNC_DATA);
if(StringUtil.isNotEmpty(redisCache)){
//说明我们人为的要终止这次数据跑批处理
LOG.info("CashRepayApplyServiceImpl#dataHandle need terminate loop
startId-{}",condition.getGteId());
break;
}
//更新起始 id，继续循环跑批
condition.setGteId(lastId);
infoList = cashRepayInfoDao.query(condition, 0, -1);
}
}

```

组装分表实体代码

```
/**
 * 将 CashRepayInfo 转换成 CashRepayApplySplit 实体
 * */
private CashRepayApplySplit buildCashRepayApplySplit(CashRepayInfo
history) {
CashRepayApplySplit split = new CashRepayApplySplit();
CashRepayApply apply = cashRepayApplyDao.get(history.getRepayNo());
if(apply != null){
//CashRepayApply 表如果已经有的话直接用
BeanUtils.copyProperties(apply, split, "id");
return split;
}
//...省略把 CashRepayInfo 组装成 CashRepayApplySplit 的代码
return split;
}

```

结语
--

回想在进公司之前，面试的时候经常会有问到分库分表的面试官，问的完全不知道怎么回答。因为没有过实际的经验，那时候总感觉分库分表是个很难很难，很高大上的东西。不知道是不是我们公司的分库分表太简单了，实际经历之
```

后发现其实也就是看看官方文档配一些配置，调用 API 即可。

其实自己亲身经历之后才发现这种需求难得根本就不是代码，而是给到我们这样一个需求之后，我们怎样去设计方案。抽象到更大的一个团队业务架构层面、甚至公司级别的业务架构层面，协调外部多部门，保证方案不影响现有业务，又能较好的完成需求。

最后，不管是不是我司分表业务简单，但是至少咱也算有了亿级数据分表经验是不?

如果这篇文章对你有帮助，记得点赞加！你的支持就是我继续创作的动力！

原文链接: https://juejin.cn/post/7371423114381557760