KNEX事务说明

资料

事务基础

https://www.runoob.com/mysql/mysql-transaction.html

KNEX官方地址

https://knexjs.org/guide/transactions.html

使用

基础

Plain Text　复制代码

```
await knex.transaction(async trx => {
    await common.knex('TEST_TRANSACTIONS').insert({NAME: "a"}).transacting(trx);
    await common.knex('TEST_TRANSACTIONS').insert({NAME: "b"}).transacting(trx);
});
```

knex.transaction后, **knex会生成一个新的会话, 并启动事务. 该会话为整个事务的生命周期. 但周期内锁住的数据, 外部访问也会受影响, 需要等待事务成功或失败, 才能继续执行. 但由于是一个新的会话, 所以不会有事务污染问题. 即回调过程中, 不会外部混入代码, 导致事务异常时, 被一起回滚. 同时, 由于是一个独立会话, 如果不关联trx会话句柄, 则会出现事务不一致问题. 需要特别注意.**

异常回滚

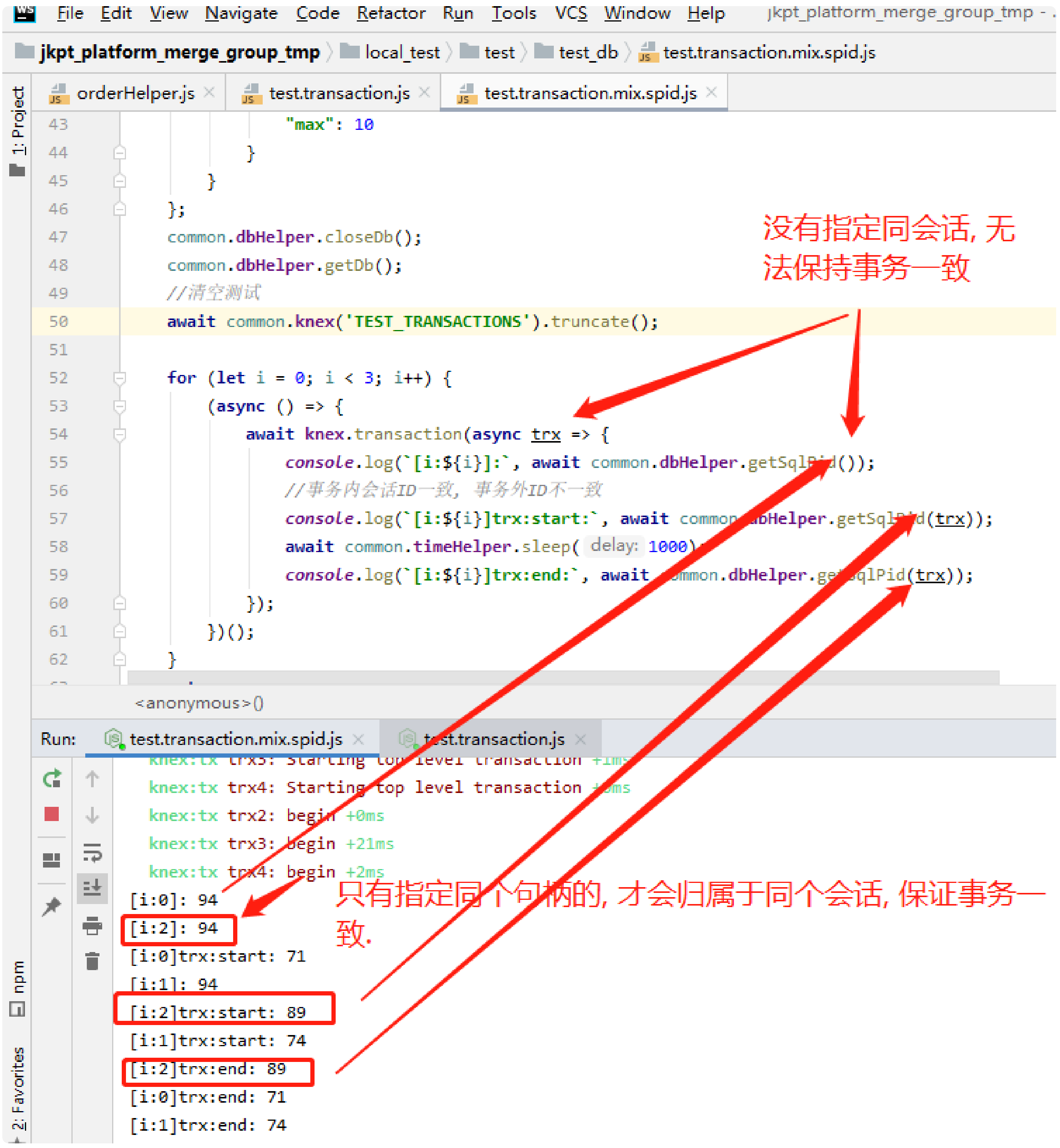
Plain Text　复制代码

```
await knex.transaction(async trx => {
    await common.knex('TEST_TRANSACTIONS').insert({NAME: "c"}).transacting(trx);
    throw new Error(`取消`);
});
```

函数出现异常, 就触发回滚, 正常就会提交

验证

验证会话句柄


File Edit View Navigate Code Refactor Run Tools VCS Window Help
jkpt_platform_merge_group_tmp
jkpt_platform_merge_group_tmp 〉 local_test 〉 test 〉 test_db 〉 test.transaction.mix.spid.js
orderHelper.js
test.transaction.js
test.transaction.mix.spid.js
1: Project
<anonymous>()
Run:
test.transaction.mix.spid.js
test.transaction.js
npm
2: Favorites
没有指定同会话, 无法保持事务一致
只有指定同个句柄的, 才会归属于同个会话, 保证事务一致.

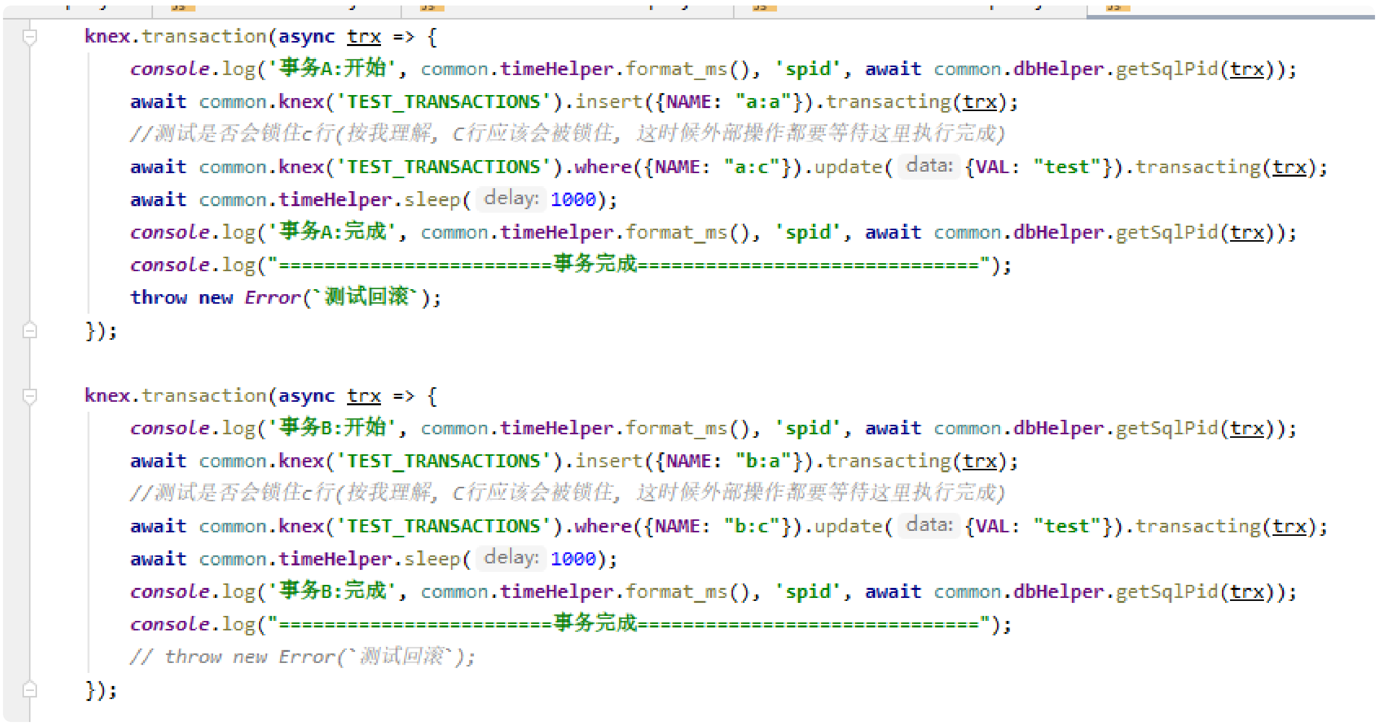
```
                "max": 10
            }
        }
    };
    common.dbHelper.closeDb();
    common.dbHelper.getDb();
    //清空测试
    await common.knex('TEST_TRANSACTIONS').truncate();

    for (let i = 0; i < 3; i++) {
        (async () => {
            await knex.transaction(async trx => {
                console.log(`[i:${i}]:`, await common.dbHelper.getSqlPid());
                //事务内会话ID一致, 事务外ID不一致
                console.log(`[i:${i}]trx:start:`, await common.dbHelper.getSqlPid(trx));
                await common.timeHelper.sleep( delay: 1000);
                console.log(`[i:${i}]trx:end:`, await common.dbHelper.getSqlPid(trx));
            });
        })();
    }
```

```
  knex:tx trx3: Starting top level transaction +1ms
  knex:tx trx4: Starting top level transaction +0ms
  knex:tx trx2: begin +0ms
  knex:tx trx3: begin +21ms
  knex:tx trx4: begin +2ms
[i:0]: 94
[i:2]: 94
[i:0]trx:start: 71
[i:1]: 94
[i:2]trx:start: 89
[i:1]trx:start: 74
[i:2]trx:end: 89
[i:0]trx:end: 71
[i:1]trx:end: 74
```

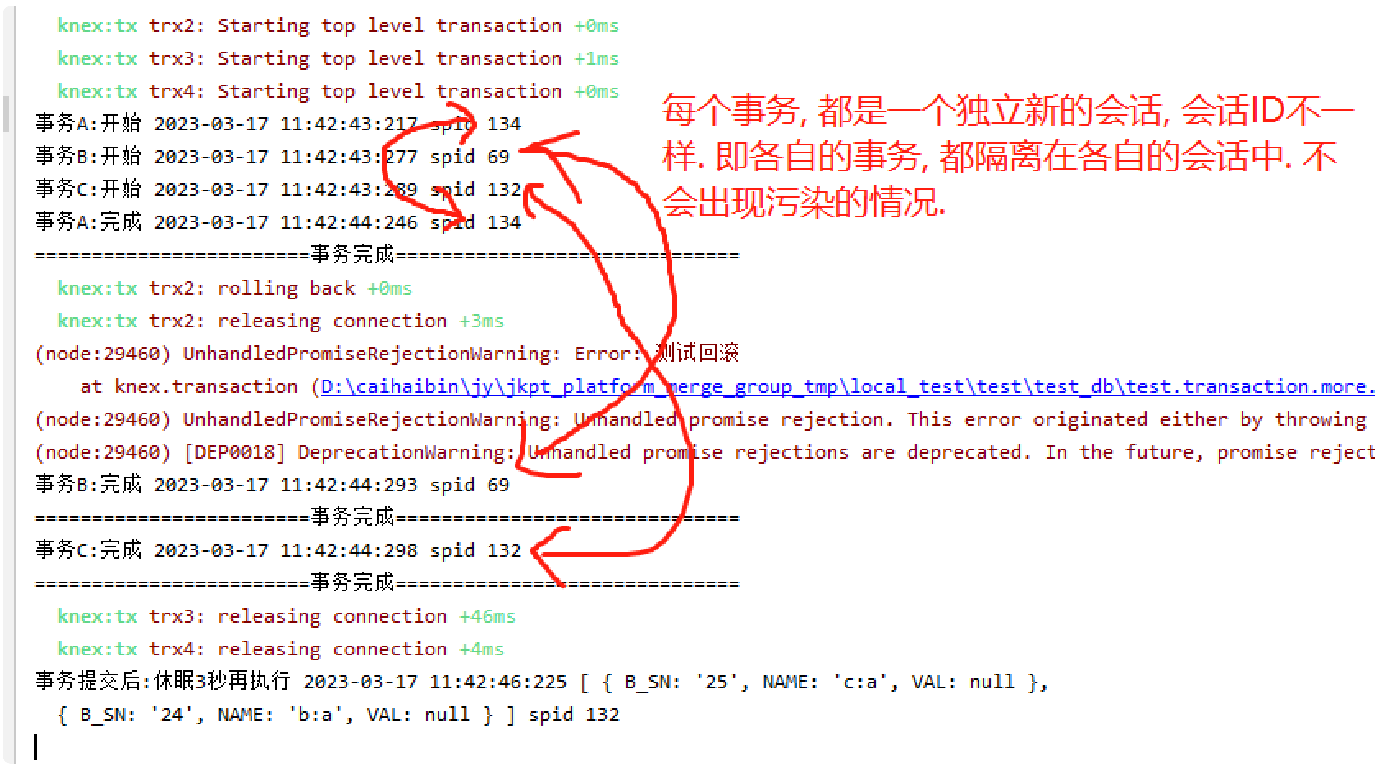
不是同个trx(会话), 会话ID不一致, 即, 即使包裹在同函数内, 没有使用trx, 也不是同个事务.

多事务验证



```
knex.transaction(async trx => {
    console.log('事务A:开始', common.timeHelper.format_ms(), 'spid', await common.dbHelper.getSqlPid(trx));
    await common.knex('TEST_TRANSACTIONS').insert({NAME: "a:a"}).transacting(trx);
    //测试是否会锁住c行(按我理解, c行应该会被锁住, 这时候外部操作都要等待这里执行完成)
    await common.knex('TEST_TRANSACTIONS').where({NAME: "a:c"}).update( data: {VAL: "test"}).transacting(trx);
    await common.timeHelper.sleep( delay: 1000);
    console.log('事务A:完成', common.timeHelper.format_ms(), 'spid', await common.dbHelper.getSqlPid(trx));
    console.log("=======================事务完成============================");
    throw new Error(`测试回滚`);
});

knex.transaction(async trx => {
    console.log('事务B:开始', common.timeHelper.format_ms(), 'spid', await common.dbHelper.getSqlPid(trx));
    await common.knex('TEST_TRANSACTIONS').insert({NAME: "b:a"}).transacting(trx);
    //测试是否会锁住c行(按我理解, c行应该会被锁住, 这时候外部操作都要等待这里执行完成)
    await common.knex('TEST_TRANSACTIONS').where({NAME: "b:c"}).update( data: {VAL: "test"}).transacting(trx);
    await common.timeHelper.sleep( delay: 1000);
    console.log('事务B:完成', common.timeHelper.format_ms(), 'spid', await common.dbHelper.getSqlPid(trx));
    console.log("=======================事务完成============================");
    // throw new Error(`测试回滚`);
});
```



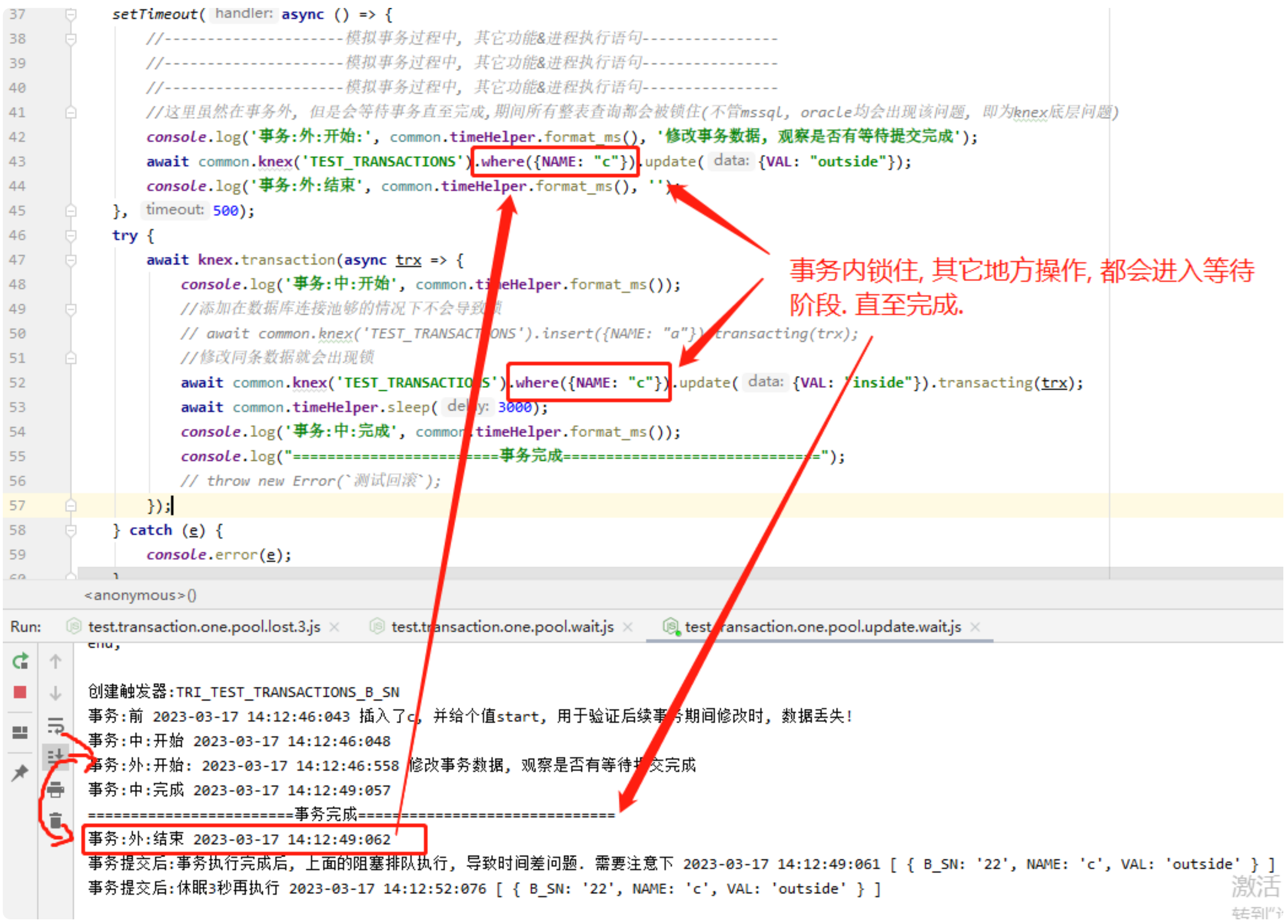
```
  knex:tx trx2: Starting top level transaction +0ms
  knex:tx trx3: Starting top level transaction +1ms
  knex:tx trx4: Starting top level transaction +0ms
事务A:开始 2023-03-17 11:42:43:217 spid 134
事务B:开始 2023-03-17 11:42:43:277 spid 69
事务C:开始 2023-03-17 11:42:43:289 spid 132
事务A:完成 2023-03-17 11:42:44:246 spid 134
========================事务完成==============================
  knex:tx trx2: rolling back +0ms
  knex:tx trx2: releasing connection +3ms
(node:29460) UnhandledPromiseRejectionWarning: Error: 测试回滚
    at knex.transaction (D:\caihaibin\jy\jkpt_platform_merge_group_tmp\local_test\test\test_db\test.transaction.more.
(node:29460) UnhandledPromiseRejectionWarning: Unhandled promise rejection. This error originated either by throwing
(node:29460) [DEP0018] DeprecationWarning: Unhandled promise rejections are deprecated. In the future, promise reject
事务B:完成 2023-03-17 11:42:44:293 spid 69
========================事务完成==============================
事务C:完成 2023-03-17 11:42:44:298 spid 132
========================事务完成==============================
  knex:tx trx3: releasing connection +46ms
  knex:tx trx4: releasing connection +4ms
事务提交后:休眠3秒再执行 2023-03-17 11:42:46:225 [ { B_SN: '25', NAME: 'c:a', VAL: null },
  { B_SN: '24', NAME: 'b:a', VAL: null } ] spid 132
```

每个事务, 都是一个独立新的会话, 会话ID不一样. 即各自的事务, 都隔离在各自的会话中. 不会出现污染的情况.

相对来说, 资源成本变高. 如果是太多, 且高频率的. 统一走语句块形式. 非必要不走事务.

示范等待

锁



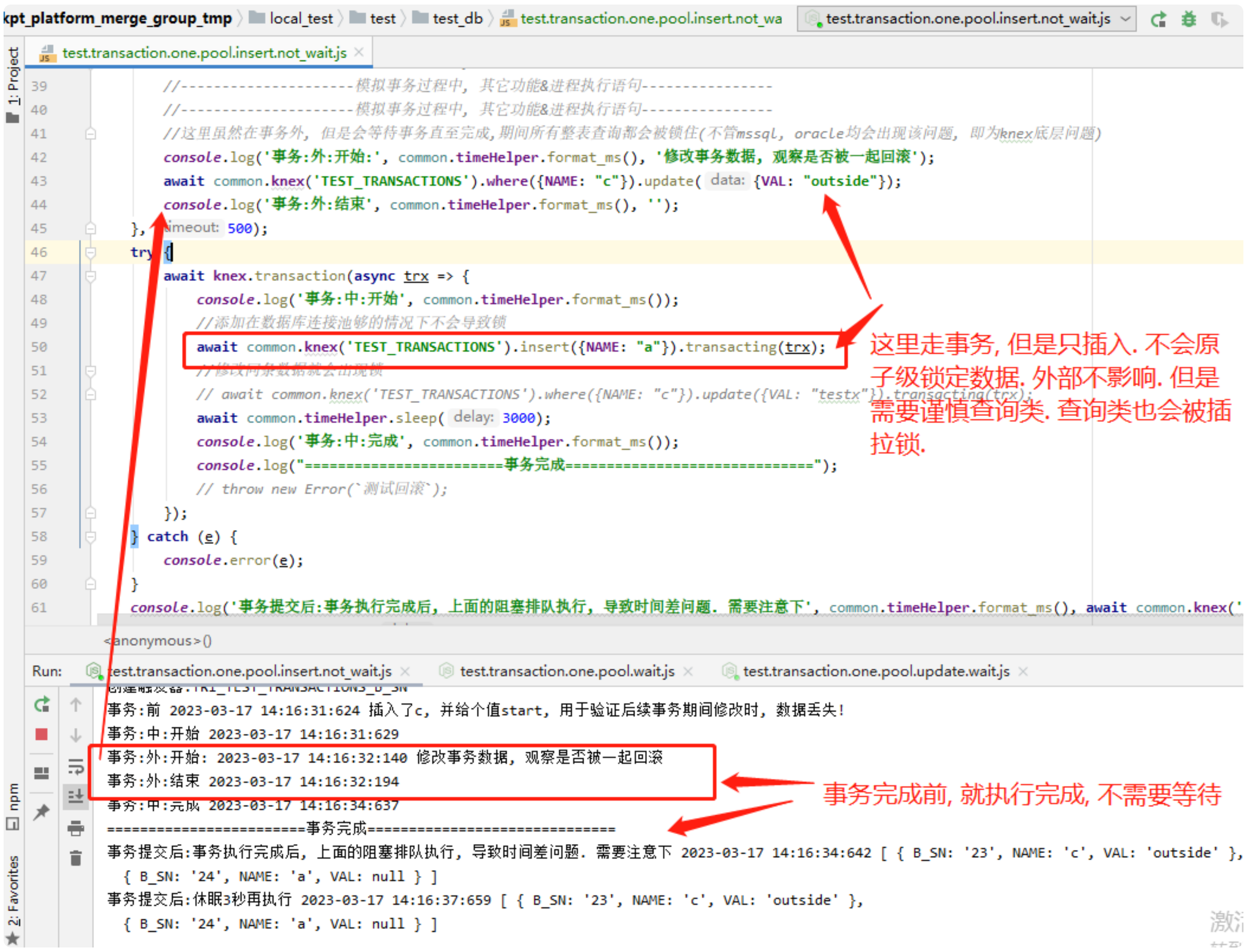
```
setTimeout( handler: async () => {
    //--------------------模拟事务过程中, 其它功能&进程执行语句---------------
    //--------------------模拟事务过程中, 其它功能&进程执行语句---------------
    //--------------------模拟事务过程中, 其它功能&进程执行语句---------------
    //这里虽然在事务外, 但是会等待事务直至完成,期间所有整表查询都会被锁住(不管mssql, oracle均会出现该问题, 即为knex底层问题)
    console.log('事务:外:开始:', common.timeHelper.format_ms(), '修改事务数据, 观察是否有等待提交完成');
    await common.knex('TEST_TRANSACTIONS').where({NAME: "c"}).update( data: {VAL: "outside"});
    console.log('事务:外:结束', common.timeHelper.format_ms(), '');
}, timeout: 500);
try {
    await knex.transaction(async trx => {
        console.log('事务:中:开始', common.timeHelper.format_ms());
        //添加在数据库连接池够的情况下不会导致锁
        // await common.knex('TEST_TRANSACTIONS').insert({NAME: "a"}).transacting(trx);
        //修改同条数据就会出现锁
        await common.knex('TEST_TRANSACTIONS').where({NAME: "c"}).update( data: {VAL: "inside"}).transacting(trx);
        await common.timeHelper.sleep( delay: 3000);
        console.log('事务:中:完成', common.timeHelper.format_ms());
        console.log("======================事务完成============================");
        // throw new Error(`测试回滚`);
    });
} catch (e) {
    console.error(e);
```

<anonymous>()
Run: test.transaction.one.pool.lost.3.js × test.transaction.one.pool.wait.js × test.transaction.one.pool.update.wait.js ×

```
end;

创建触发器:TRI_TEST_TRANSACTIONS_B_SN
事务:前 2023-03-17 14:12:46:043 插入了c, 并给个值start, 用于验证后续事务期间修改时, 数据丢失!
事务:中:开始 2023-03-17 14:12:46:048
事务:外:开始: 2023-03-17 14:12:46:558 修改事务数据, 观察是否有等待提交完成
事务:中:完成 2023-03-17 14:12:49:057
=======================事务完成============================
事务:外:结束 2023-03-17 14:12:49:062
事务提交后:事务执行完成后, 上面的阻塞排队执行, 导致时间差问题. 需要注意下 2023-03-17 14:12:49:061 [ { B_SN: '22', NAME: 'c', VAL: 'outside' } ]
事务提交后:休眠3秒再执行 2023-03-17 14:12:52:076 [ { B_SN: '22', NAME: 'c', VAL: 'outside' } ]
```

无锁



kpt_platform_merge_group_tmp 〉 local_test 〉 test 〉 test_db 〉 test.transaction.one.pool.insert.not_wa

test.transaction.one.pool.insert.not_wait.js

```
        //--------------------模拟事务过程中, 其它功能&进程执行语句---------------
        //--------------------模拟事务过程中, 其它功能&进程执行语句---------------
        //这里虽然在事务外, 但是会等待事务直至完成,期间所有整表查询都会被锁住(不管mssql, oracle均会出现该问题, 即为knex底层问题)
        console.log('事务:外:开始:', common.timeHelper.format_ms(), '修改事务数据, 观察是否被一起回滚');
        await common.knex('TEST_TRANSACTIONS').where({NAME: "c"}).update( data: {VAL: "outside"});
        console.log('事务:外:结束', common.timeHelper.format_ms(), '');
    }, timeout: 500);
    try {
        await knex.transaction(async trx => {
            console.log('事务:中:开始', common.timeHelper.format_ms());
            //添加在数据库连接池够的情况下不会导致锁
            await common.knex('TEST_TRANSACTIONS').insert({NAME: "a"}).transacting(trx);
            //修改同条数据就会出现锁
            // await common.knex('TEST_TRANSACTIONS').where({NAME: "c"}).update({VAL: "testx"}).transacting(trx);
            await common.timeHelper.sleep( delay: 3000);
            console.log('事务:中:完成', common.timeHelper.format_ms());
            console.log("=======================事务完成=============================");
            // throw new Error(`测试回滚`);
        });
    } catch (e) {
        console.error(e);
    }
    console.log('事务提交后:事务执行完成后, 上面的阻塞排队执行, 导致时间差问题. 需要注意下', common.timeHelper.format_ms(), await common.knex('
```

这里走事务, 但是只插入. 不会原子级锁定数据. 外部不影响. 但是需要谨慎查询类. 查询类也会被插拉锁.

Run: test.transaction.one.pool.insert.not_wait.js × test.transaction.one.pool.wait.js × test.transaction.one.pool.update.wait.js ×

```
事务:前 2023-03-17 14:16:31:624 插入了c, 并给个值start, 用于验证后续事务期间修改时, 数据丢失!
事务:中:开始 2023-03-17 14:16:31:629
事务:外:开始: 2023-03-17 14:16:32:140 修改事务数据, 观察是否被一起回滚
事务:外:结束 2023-03-17 14:16:32:194
事务:中:完成 2023-03-17 14:16:34:637
=======================事务完成=============================
事务提交后:事务执行完成后, 上面的阻塞排队执行, 导致时间差问题. 需要注意下 2023-03-17 14:16:34:642 [ { B_SN: '23', NAME: 'c', VAL: 'outside' },
  { B_SN: '24', NAME: 'a', VAL: null } ]
事务提交后:休眠3秒再执行 2023-03-17 14:16:37:659 [ { B_SN: '23', NAME: 'c', VAL: 'outside' },
  { B_SN: '24', NAME: 'a', VAL: null } ]
```

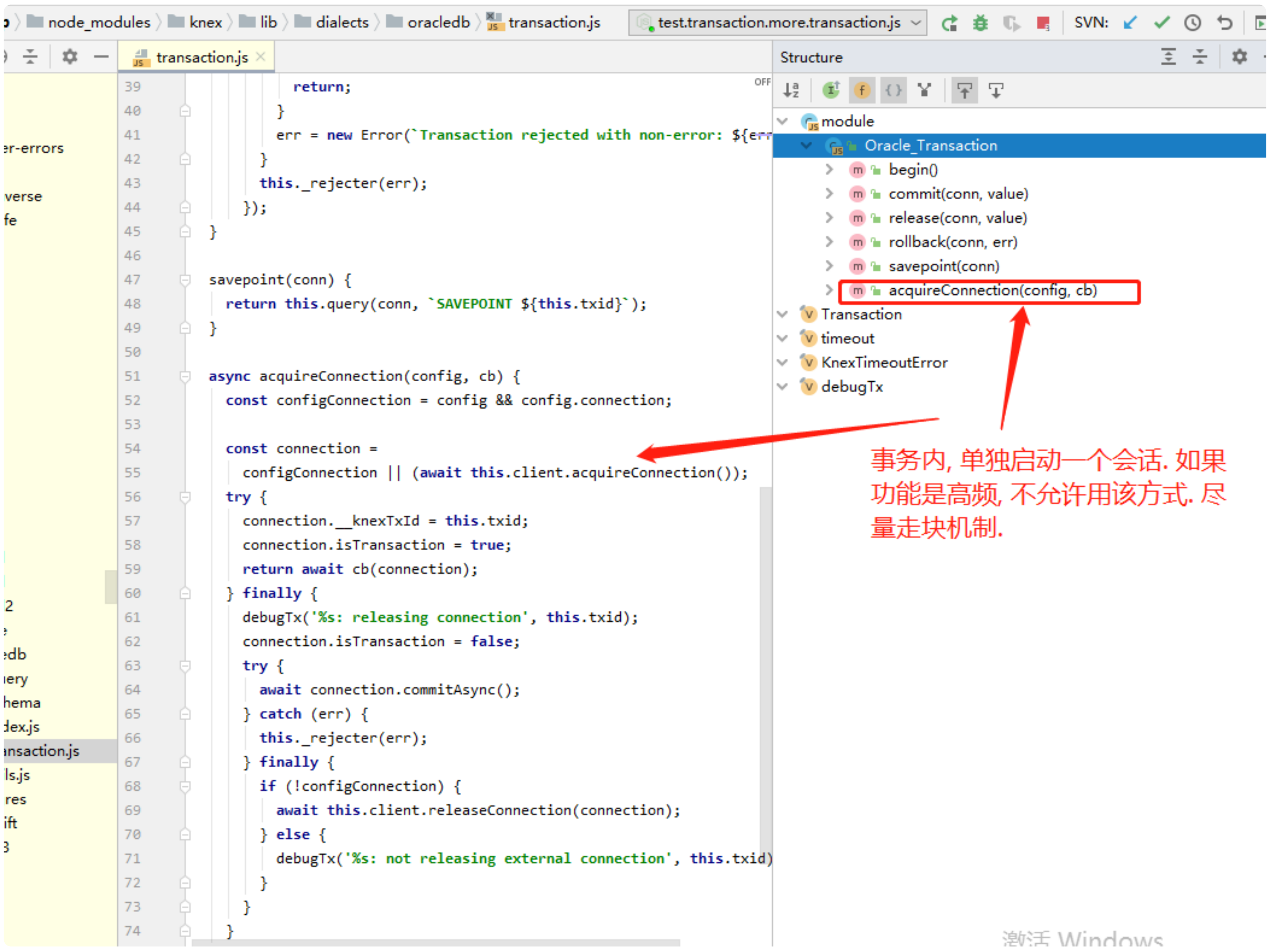
事务完成前, 就执行完成, 不需要等待

规范

禁止耗时处理

排查各个系统的代码，检查在事务中是否存在RPC调用、HTTP调用、消息队列操作、缓存、循环查询等耗时的操作，这个操作应该移到事务之外，理想的情况是事务内只处理数据库操作。**耗时操作不允许放在事务内部, 避免形成死锁. 所有查询类操作, 且非事务必要性, 均外部查询完成再进入事务进行逻辑处理.**

禁止高频操作



```
        return;
      }
      err = new Error(`Transaction rejected with non-error: ${err
    }
    this._rejecter(err);
  });
}

savepoint(conn) {
  return this.query(conn, `SAVEPOINT ${this.txid}`);
}

async acquireConnection(config, cb) {
  const configConnection = config && config.connection;

  const connection =
    configConnection || (await this.client.acquireConnection());
  try {
    connection.__knexTxId = this.txid;
    connection.isTransaction = true;
    return await cb(connection);
  } finally {
    debugTx('%s: releasing connection', this.txid);
    connection.isTransaction = false;
    try {
      await connection.commitAsync();
    } catch (err) {
      this._rejecter(err);
    } finally {
      if (!configConnection) {
        await this.client.releaseConnection(connection);
      } else {
        debugTx('%s: not releasing external connection', this.txid)
      }
    }
  }
```

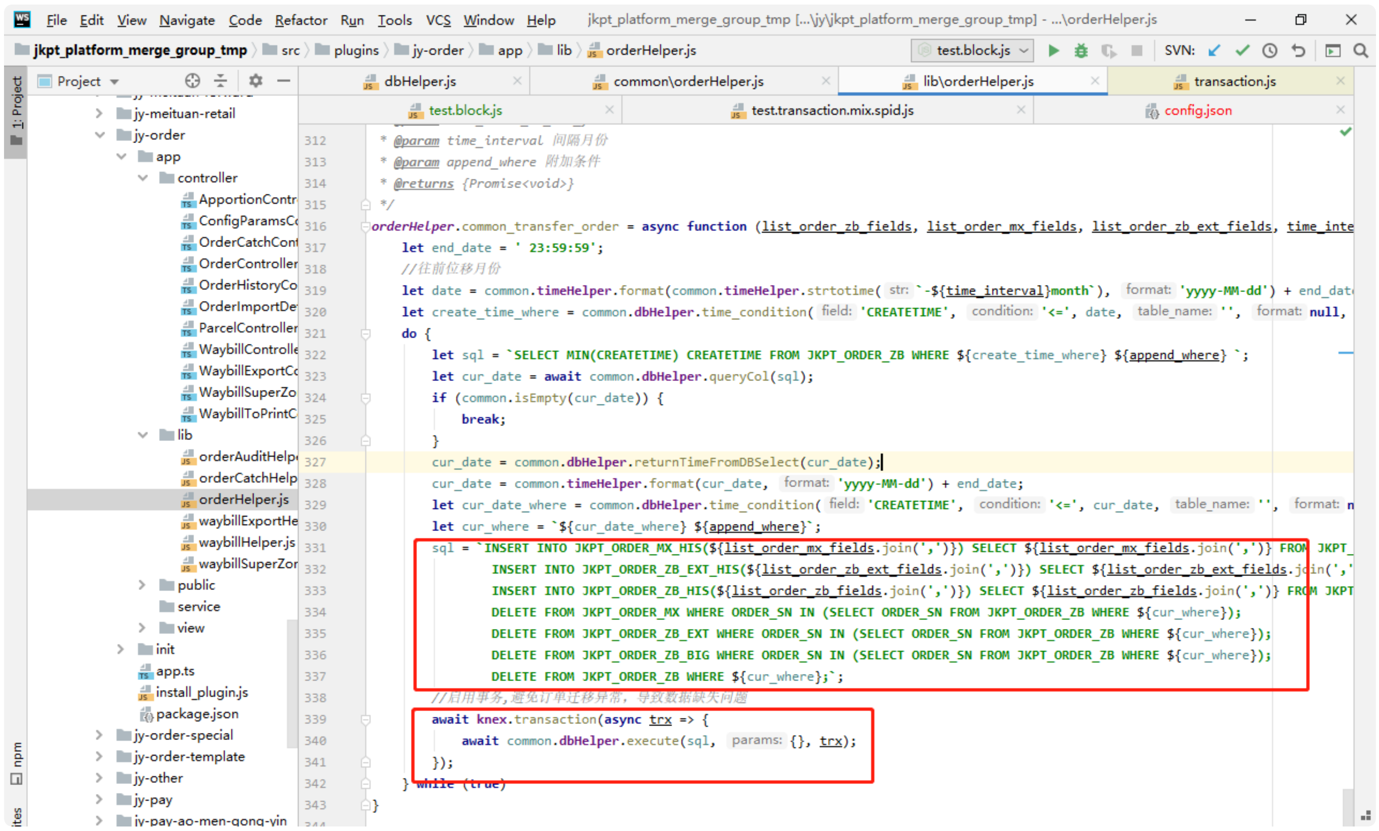
Structure
```
module
  Oracle_Transaction
    begin()
    commit(conn, value)
    release(conn, value)
    rollback(conn, err)
    savepoint(conn)
    acquireConnection(config, cb)
  Transaction
  timeout
  KnexTimeoutError
  debugTx
```

事务内, 单独启动一个会话. 如果功能是高频, 不允许用该方式. 尽量走块机制.

由于是独立会话形式, 如果高频执行, 会产生大量会话, 导致线程池满问题. 需要合理使用

错误使用

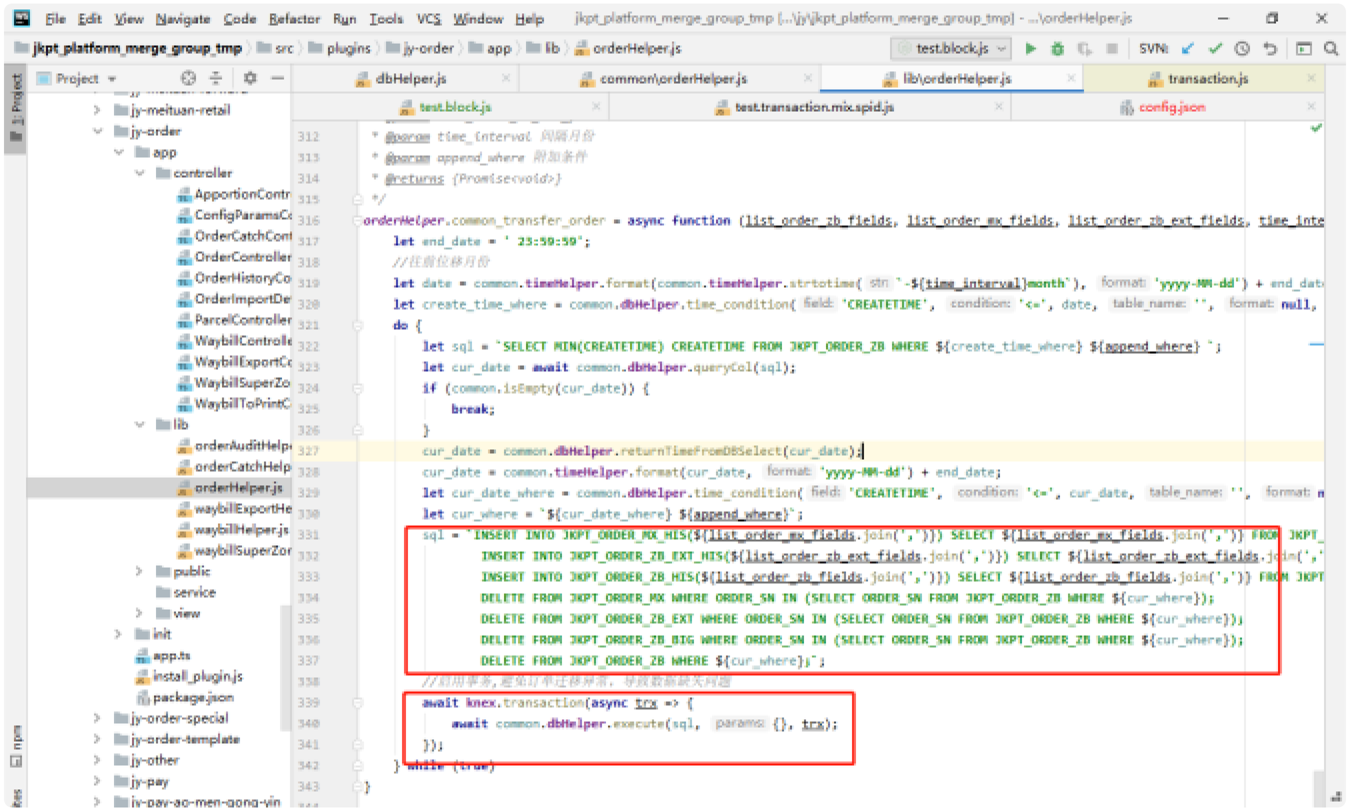
案例一:非必要不走事务



```
 * @param time_interval 间隔月份
 * @param append_where 附加条件
 * @returns {Promise<void>}
 */
orderHelper.common_transfer_order = async function (list_order_zb_fields, list_order_mx_fields, list_order_zb_ext_fields, time_inte
    let end_date = ' 23:59:59';
    //往前位移月份
    let date = common.timeHelper.format(common.timeHelper.strtotime( str: `-${time_interval}month`),  format: 'yyyy-MM-dd') + end_dat
    let create_time_where = common.dbHelper.time_condition( field: 'CREATETIME',  condition: '<=', date,  table_name: '',  format: null,
    do {
        let sql = `SELECT MIN(CREATETIME) CREATETIME FROM JKPT_ORDER_ZB WHERE ${create_time_where} ${append_where} `;
        let cur_date = await common.dbHelper.queryCol(sql);
        if (common.isEmpty(cur_date)) {
            break;
        }
        cur_date = common.dbHelper.returnTimeFromDBSelect(cur_date);
        cur_date = common.timeHelper.format(cur_date,  format: 'yyyy-MM-dd') + end_date;
        let cur_date_where = common.dbHelper.time_condition( field: 'CREATETIME',  condition: '<=', cur_date,  table_name: '',  format: n
        let cur_where = `${cur_date_where} ${append_where}`;
        sql = `INSERT INTO JKPT_ORDER_MX_HIS(${list_order_mx_fields.join(',')}) SELECT ${list_order_mx_fields.join(',')} FROM JKPT_
               INSERT INTO JKPT_ORDER_ZB_EXT_HIS(${list_order_zb_ext_fields.join(',')}) SELECT ${list_order_zb_ext_fields.join(','
               INSERT INTO JKPT_ORDER_ZB_HIS(${list_order_zb_fields.join(',')}) SELECT ${list_order_zb_fields.join(',')} FROM JKPT
               DELETE FROM JKPT_ORDER_MX WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
               DELETE FROM JKPT_ORDER_ZB_EXT WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
               DELETE FROM JKPT_ORDER_ZB_BIG WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
               DELETE FROM JKPT_ORDER_ZB WHERE ${cur_where};`;
        //启用事务,避免订单迁移异常, 导致数据缺失问题
        await knex.transaction(async trx => {
            await common.dbHelper.execute(sql,  params: {}, trx);
        });
    } while (true)
}
```

语句块本身就是事务, 再嵌套事务就是浪费操作. 同时, 历史转移, 实际上并不需要事务操作. 如果时间太久. 事务期间, 所有操作都会进入排队. **将任务拆分比较合适**. 优先转移历史主表. 再走not in 条件将其它数据进行转移. 避免执行时间过久. 导致所有相关功能等待整个事务完成而被堵死的潜在风险.

案例二:重复事务



```
 * @param time_interval 间隔月份
 * @param append_where 附加条件
 * @returns {Promise<void>}
 */
orderHelper.common_transfer_order = async function (list_order_zb_fields, list_order_mx_fields, list_order_zb_ext_fields, time_inte
    let end_date = ' 23:59:59';
    //往前位移月份
    let date = common.timeHelper.format(common.timeHelper.strtotime(`-${time_interval}month`), 'yyyy-MM-dd') + end_dat
    let create_time_where = common.dbHelper.time_condition('CREATETIME', '<=', date, '', null,
    do {
        let sql = `SELECT MIN(CREATETIME) CREATETIME FROM JKPT_ORDER_ZB WHERE ${create_time_where} ${append_where} `;
        let cur_date = await common.dbHelper.queryCol(sql);
        if (common.isEmpty(cur_date)) {
            break;
        }
        cur_date = common.dbHelper.returnTimeFromDBSelect(cur_date);
        cur_date = common.timeHelper.format(cur_date, 'yyyy-MM-dd') + end_date;
        let cur_date_where = common.dbHelper.time_condition('CREATETIME', '<=', cur_date, '', n
        let cur_where = `${cur_date_where} ${append_where}`;
        sql = `INSERT INTO JKPT_ORDER_MX_HIS(${list_order_mx_fields.join(',')}) SELECT ${list_order_mx_fields.join(',')} FROM JKPT_
            INSERT INTO JKPT_ORDER_ZB_EXT_HIS(${list_order_zb_ext_fields.join(',')}) SELECT ${list_order_zb_ext_fields.join(','
            INSERT INTO JKPT_ORDER_ZB_HIS(${list_order_zb_fields.join(',')}) SELECT ${list_order_zb_fields.join(',')} FROM JKPT
            DELETE FROM JKPT_ORDER_MX WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
            DELETE FROM JKPT_ORDER_ZB_EXT WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
            DELETE FROM JKPT_ORDER_ZB_BIG WHERE ORDER_SN IN (SELECT ORDER_SN FROM JKPT_ORDER_ZB WHERE ${cur_where});
            DELETE FROM JKPT_ORDER_ZB WHERE ${cur_where};`;
        //启用事务, 避免订单迁移异常, 导致数据缺失问题
        await knex.transaction(async trx => {
            await common.dbHelper.execute(sql, {}, trx);
        });
    } while (true)
}
```

语句块本身就是事务, 然后外面又重新发起事务, 导致重复.