在 LeanCloud 上解决数据一致性

(07-28 20:00)

事务?

场景

- 账户转账
- 批量更新
- 秒杀优惠券
- 支付订单处理
- 每天利息结算

问题

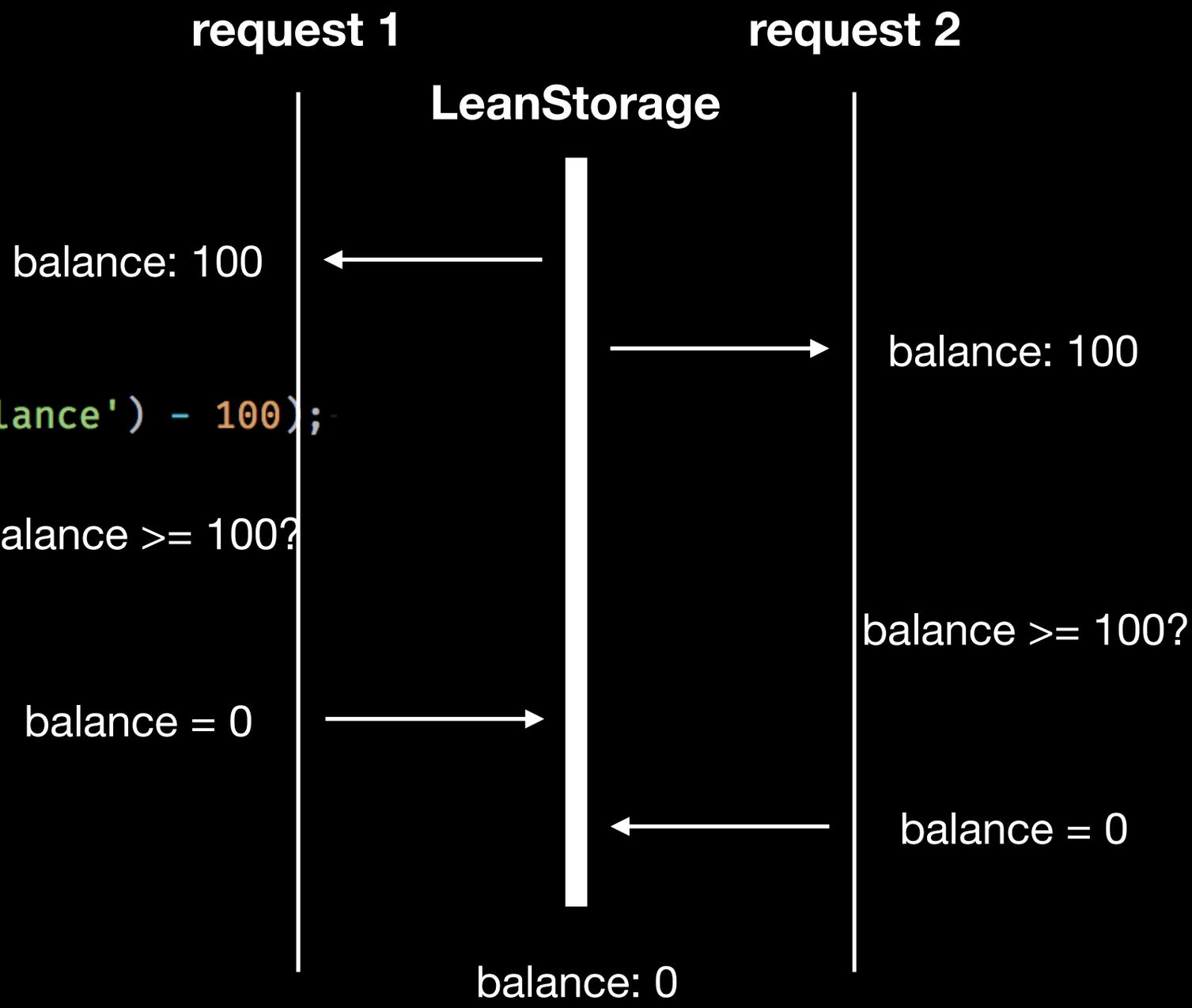
- **并发运行的任务读写同一份数据导致不一致**
- 任务在运行中可能会因为错误或进程重启而中断

基本工具

- increment（原子计数器）
- saveWithQuery（更新附带查询）
- Unique index（唯一索引）
- 对象级别原子性
- Redlock (LeanCache)

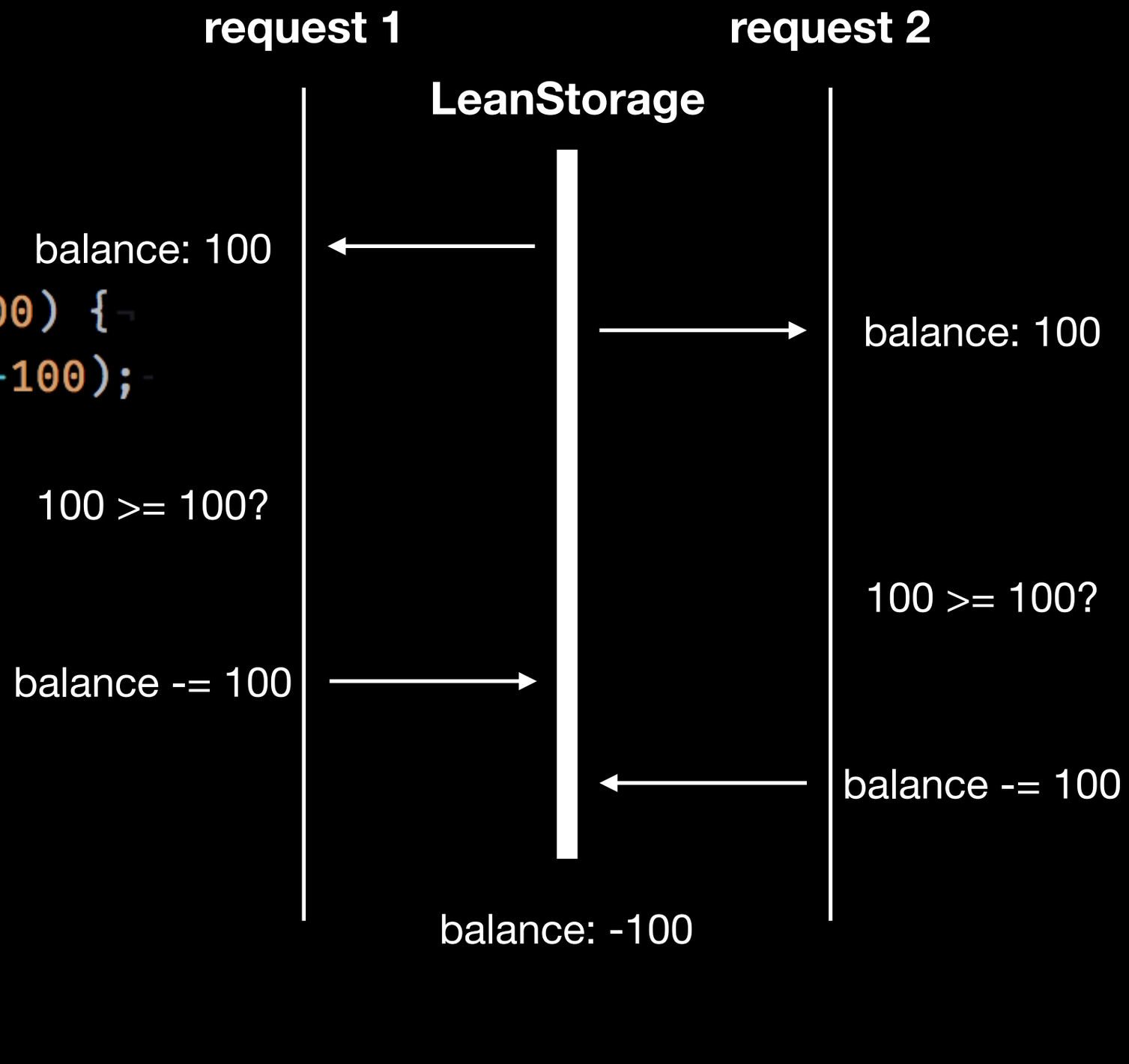
场景：扣减余额

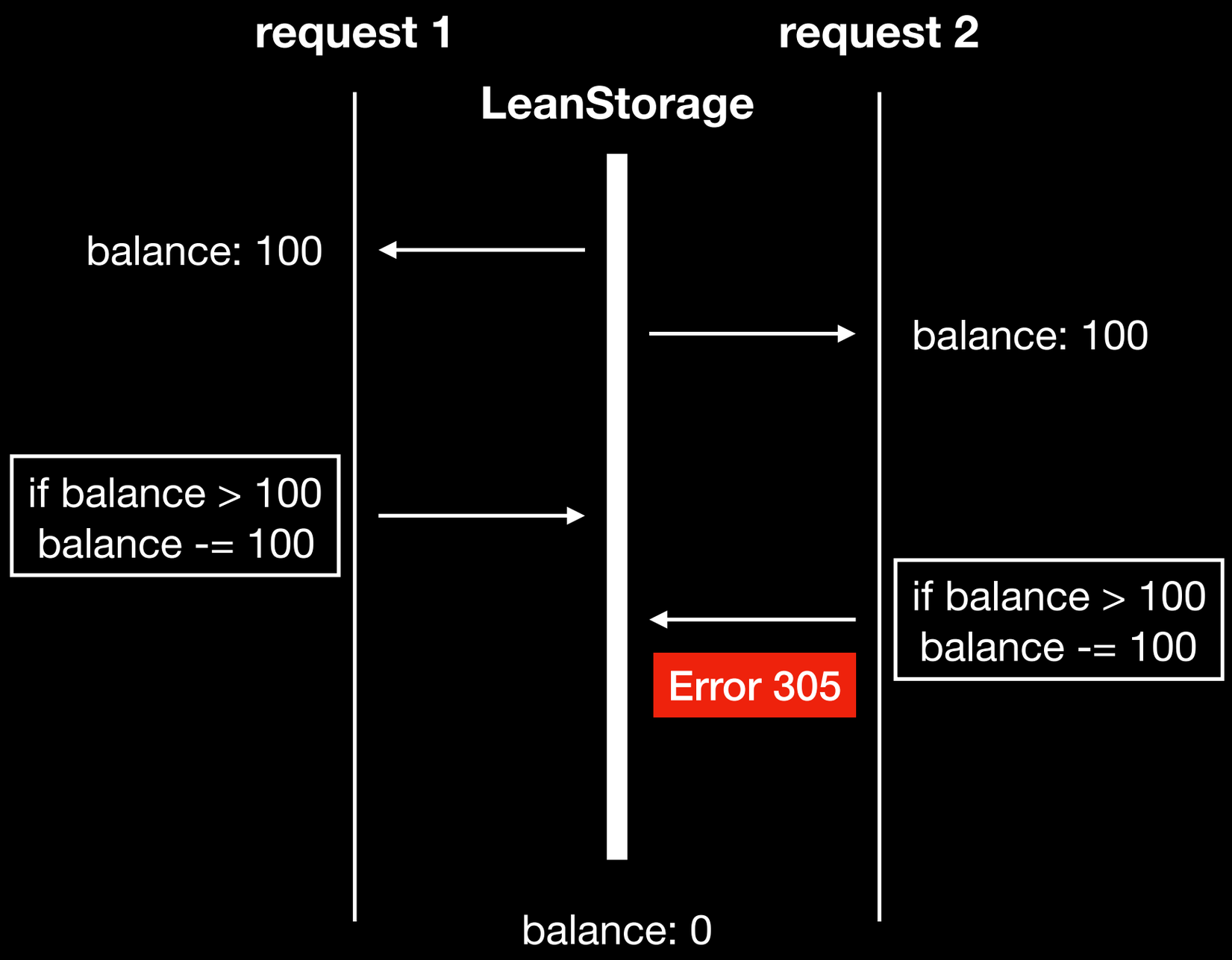
```
user.fetch().then( user ⇒ {
  if (user.get('balance') ⩾ 100) {
    user.set('balance', user.get('balance') - 100);
    return user.save();
  } else {
    console.error('余额不足');
  }
});
```



increment

```
user.fetch().then( user ⇒ {
  if (user.get('balance') ⩾ 100) {
    user.increment('balance', -100);
    return user.save();
  } else {
    console.error('余额不足');
  }
});
```




request 1
request 2
LeanStorage
balance: 100
balance: 100
if balance > 100
balance -= 100
if balance > 100
balance -= 100
Error 305
balance: 0

saveWithQuery

```
user.increment('balance', -100);
user.save(null, {
  query: new AV.Query(AV.User).greaterThanOrEqualTo('balance', 100),
  fetchWhenSave: true
}).then( user => {
  console.log('当前余额', user.get('balance'));
}).catch( err => {
  if (err.code === 305) {
    console.error('余额不足');
  }
});
```

场景：订单处理

```
app.post('/weapp-pay-callback', (req, res) ⇒ {
  new AV.Query(Order).equalTo('tradeId', req.body.out_trade_no)
    .first().then( order ⇒ {
      return order.save({
        status: 'success'
      }, {
        query: new AV.Query(Order).equalTo('status', 'pending')
      }).then( () ⇒ {
        user.increment('balance', order.get('amount'));
        return user.save();
      }).catch( err ⇒ {
        if (err.code === 305) {
          console.error('订单已被确认');
        }
      });
  });
});
```

场景：订单处理

```
app.post('/weapp-pay-callback', (req, res) ⇒ {
  var order = new Order({
    tradeId: req.body.out_trade_no
  });

  order.save().then( () ⇒ {
    user.increment('balance', req.body.amount);
    return user.save();
  }).catch( err ⇒ {
    if (err.code === 137) {
      console.error('订单已被确认');
    }
  });
});
```

Unique index

索引

新建唯一索引

objectId | interest | date

ACL | updatedAt | createdAt

允许 缺失值

创建

已创建唯一索引 不包括内置索引

索引名称	索引键值	索引选项	操作
date_1	{"date":1}	{"background":true,"unique":true,"v":1}	删除

对象级别原子性

```
object.increment('count', 1);
object.set('field1', value);
object.add('books', book);
object.save(); // atomically
```

RedLock

```
var Redlock = require('redlock');
var redlock = new Redlock([redisClient]);

redlock.lock(`user:${user.id}`, 10000).then( lock ⇒ {
  return Promise.try( () ⇒ {
    // ...
  }).finally( () ⇒ {
    return lock.unlock();
  });
});
```

小结（并发修改）

- saveWithQuery 保证并发的更新操作只有一个可以成功
- 将需要唯一性的操作转化为插入操作（唯一索引）
- 通过 Redlock 手动加锁可以保证只有一个进程进行操作

服务器 Or 客户端?

问题

- 并发运行的任务读写同一份数据导致不一致

- 任务在运行中可能会因为错误或进程重启而中断

State

A.balance -= 100

B.balance += 100

log: transfer 100 from A to B

A.balance -= 100

log: decreased balance of A

B.balance += 100

log: increased balance of B

log: transfer completed

log: transfer 100 from A to B

A.balance -= 100

log: decreased balance of A

B.balance += 100

log: increased balance of B

log: transfer completed

两阶段提交

initial　初始化

```
new Transaction({from: 'A', to: 'B', amount: 100, state: 'pending'}).save();
```

initial => pending　开始操作（阶段一）

A.balance -= 100
log: decreased balance of A

B.balance += 100
log: increased balance of B

```
fromUser.increment('balance', -transaction.amount);
fromUser.add('pendingTransactions', transaction);

return fromUser.save(null, {
  query: new AV.Query(AV.User)
    .notEqualTo('pendingTransactions', transaction)
});
```

pending => applied　操作完成 进行清理（阶段二）

```
fromUser.remove('pendingTransactions', transaction);
toUser.remove('pendingTransactions', transaction);
```

applied => done　彻底完成

改进订单处理

```
app.post('/weapp-pay-callback', (req, res) ⇒ {
  Order.equalTo('tradeId', req.body.out_trade_no).first().then( order ⇒ {
    return order.save({status: 'confirming'}, {
      query: new AV.Query(Order).equalTo('status', 'pending')
    }).then( () ⇒ {
      user.increment('balance', order.get('amount'));
      user.add('confirmingOrders', order);
      return user.save(null, {
        query: new AV.Query(AV.User).notEqualTo('confirmingOrders', order)
      });
    }).then( () ⇒ {
      return order.save({status: 'confirmed'}).then( () ⇒ {
        user.remove('confirmingOrders', order);
        return user.save();
      }).then( () ⇒ {
        return order.save({status: 'success'});
      });
    }).catch( err ⇒ {
      if (err.code === 137) {
        console.error('订单已被确认');
      }
```

如果发生中断

- order.status == confirming
  - order not in user.confirmingOrders：未增加余额
  - order in user.confirmingOrders：已增加余额
- order.status == confirmed
  - 已增加余额

如果发生中断

```
new AV.Query(Order).containedIn('status', ['confirming', 'confirmed'])
.lessThan('updatedAt', new Date(Date.now() - 600000))
.includes('user').find().then( orders => {
  return Promise.map(orders, order => {
    if (order.get('status') === 'confirming') {
      if ((order.get('user').get('confirmingOrders').includes(order))) {
        // 已加余额，跳到 `order.save({status: 'confirmed'})` 继续
      } else {
        // 未加余额，跳到 `user.increment('balance', order.get('amount'))` 继续
      }
    } else if (order.get('status') === 'confirmed') {
      // 已加余额，跳到 `user.remove('confirmingOrders', order)` 继续
    }
  });
});
```

小结（中断）

- 将需要原子性保证的数据集中到一个对象上
- 使用 saveWithQuery 进行原子地比较和更新
- 使用两阶段提交来将完整的状态保留下来
- 使用定时任务来恢复被中断的事务

总结

- 我们面临的主要问题是「并发修改」和「执行过程中断」
- saveWithQuery 可以进行有条件的更新，使只有一个修改能够成功
- 将需要唯一性的操作转化为插入操作（唯一索引）
- 通过 Redlock 手动加锁可以保证只有一个进程进行操作
- 将需要原子性保证的数据集中到一个对象上
- 两阶段提交提供了从中断中恢复或回滚的能力