



链滴

从 Icalingua 项目谈 SQL 数据库结构的最佳实践

作者: [lixiang810](#)

原文链接: <https://ld246.com/article/1633586851807>

来源网站: 链滴

许可协议: [署名-相同方式共享 4.0 国际 \(CC BY-SA 4.0\)](#)

这最早是 [Icalingua](#) (下简称 II) 项目下的一个 issue, 因比较详细地叙述了一系列 bug 的诞生, 具有一定的参考价值, 故独立出来发布。

一、万恶之源——不当的数据组织方式

存储消息时, 每个房间都建一个表 (名为 `msg${roomId}`)。这是 II 在用 json 存记录时的某种优化却在随后的 MongoDB 存储方式中也被继承, 并影响到了 `SQLStorageProvider` 和 `RedisStorageProvider` 的编写。在 MongoDB 和 Redis 这样的 NoSQL 数据库中, 这么做并不会增加太多复杂度。然在 `SQLStorageProvider` 中, 这种“表名与表数量都不固定”的结构却成倍增加了代码量和复杂度。

二、水土不服——SQL 相关代码的编写

NoSQL 数据库中, 尝试向表 (只有 Mongo 有对应“表”的概念——Collections, 但我在 Redis 里实现了这一层, 姑且这么叫吧) 中写入数据时, 若表不存在, 它会被自动创建。然而, SQL 数据库并有类似 `insertOrReplace` 和 `createTableIfNotExist` 这样的功能 (除了 PostgreSQL, 但我用 `knex` 实现的 `SQLStorageProvider` 在功能上取三个 SQL 的交集以保证兼容性)。

同时, 因为 SQL 的 column 有较为严格的限制, 我们需要对数据库版本进行管理, 当 columns 有改时, 运行升级函数去 alter 各表, 改动它们的 columns。然而, 由于表名与表数量都不固定, 我只好建一表 (表名为 `msgTableNameTable`, 是不是有屎山的感受了?), 表中存储各 `msg` 表的表名。

于是我实现了一个 `createMsgTable` 方法, 在每次对表进行操作以前调用该方法检查目标表是否存在若不存在则先创建, 创建时也向 `msgTableNameTable` 写入 `msg` 表的表名, 以便此后进行升级操作能修改它们的 columns。

由于上述额外操作, 以及对各 column 类型的定义, `SQLStorageProvider.ts` 的行数是 `MongoStorageProvider.ts` 的两倍以上。众所周知, bug 量与代码行数正相关, 因此 `SQLStorageProvider` 自诞生日起就 bug 不断。

- 使用 SQLite 时, 无法正确创建数据库目录, 导致崩溃
- 使用三类 SQL 作为存储方式时, 房间不会按消息先后顺序进行排序
- 使用三类 SQL 作为存储方式时, 艾特所有人的消息无法入库
- 使用三类 SQL 或 IndexedDB 作为存储方式时, 无法撤回消息
- 使用 MongoDB 以外的数据库作为存储方式时, 托盘图标红点通知工作不正常

单是被我记录的 bug 就有 5 条。在修复它们后, `SQLStorageProvider` 进入了一段稳定时期。`SQLStorageProvider` 具有庞大的码量, 主作者并没有时间理解它, 从而无法进行修改。我所做的就是主作者提出新增/修改 columns 的 issues 时去编写对应的升级函数。此后, 我为 `SQLStorageProvider` 加了相当详细的 jsDoc 注释, 于是我也不再需要自己编写升级函数, 只需偶尔 review 一下主作者的改即可。

对一个从一开始就以 NoSQL 的思路设计的程序, 后续增加的 SQL 数据库将水土不服。无论如何, 一系列问题都姑且得到了解决, 尽管它们让作者们和用户们都体会到了痛苦 (请手动播放《Unravel》), 直到这一天的到来。

三、异步灾难——过度优化打开的潘多拉魔盒

当打开空的群聊的时候报错, 起初它没有得到足够的重视, 因为我有约两个月的时期没有使用 Linux Icalingua。然而一周后的今天, 当我注意到该 issues 并尝试解决问题时, 这场灾难终于揭开了帷幕。

```
Uncaught (in promise) Error: Error invoking remote method 'fetchMessage': Error: create table `msg-*****` (`id` varchar(255), ..., primary key (`id`)) - SQLITE_ERROR: table `msg-104649684` already exists
```

这一行错误让我相当震惊——按道理，我的代码不应该出现这个错误，因为在建表前，`createMsgTable` 会先调用 `hasTable` 方法来确认这个表不存在。

于是我开始了 debug。对 electron 的 main 进程中的错误进行 debug 是一件比较痛苦的事情。因为我似乎没法正常使用 VS Code 中的调试器，我只好以插入 `console.log()` 这样的原始方法进行调试。

大约5分钟之后，我发现了罪魁祸首：`StorageProvider` 类（`SQLStorageProvider` 为对该类的一个实例）中的 `addRoom` 和 `fetchMessages` 方法被调用，但后者并非是等待前者异步返回后再被调用，是同时被调用。

```
const hasMsgTable = await this.db.schema.hasTable(`msg${roomId}`)
if (!hasMsgTable) {
  await this.db.schema.createTable(`msg${roomId}`, (table) => {
    .....
  })
}
```

这是 `createMsgTable` 方法的片段。`addRoom` 和 `fetchMessages` 均在开头调用了 `createMsgTable` 方法。该方法的作用为判断目标表是否存在，若不存在则建表。然而，当这个方法被以上述方式调用，灾难出现了。

由于主程序对 `addRoom` 和 `fetchMessages` 两个方法异步调用且不等待返回的设计，两次调用中，`hasMsgTable` 均为 `false`，因此 `createTable` 会被调用两次。第二次调用时，由于表已存在，`SQLStorageProvider` 会抛错，用户报告见到的正是这个抛错。

对“异步调用不等待返回”，主作者的理由是：

那我如果确实不需要等待呢？我希望多线程

实际上这并不是多线程，但却有和多线程相当相似的结果：

你有一个问题

你试图用多线程解决它

现在你在两个问题间有了

一个很自然的想法是通过加锁来解决。我在 `SQLStorageProvider` 中增加了一个私有数组，每次调用 `createMsgTable` 方法时，都会检查数组中是否有目标表名，若有，则跳过后续操作；若没有，则将目标表名 `push` 进这个数组，并进行后续建表操作。

然而我的锁并没有锁住异步的魔鬼。`SQLStorageProvider` 报错依旧。进一步的 `debug` 显示，我加的锁毫无作用，因为第二次被调用时，数组依然为空。

我并不知道 `SQLStorageProvider` 以何种方式被实例化，但如果它在主程序中有大于等于两个实例，我的锁也将无能为力。即使只有一个实例，以这种异步调用的方式，锁也未必能生效。而即使这个锁以生效，由于主程序异步调用不等待返回，`createMsgTable` 方法也将无法如它的设计目标那样，保证目标表在被访问前已被建立。

最终的解决方案是道选择题：

1. 放弃异步调用不等待返回的过度优化。

2. 放弃不靠谱的动态表名，将所有消息存入同一张表中，这个表像其它几个表一样，在 `StorageProvider` 的 `connect` 方法中被创建，该方法的异步调用会等待返回。

第二种选择的代码我已经完成，正在进行相关的性能测试。已有的结果（一晚，共收到 3632 条消息表明：在有索引的情况下，用一个表存储所有消息并不会带来显著的性能损失。但按正常的使用方式这个表最终会有十万甚至百万级别的行数，在被同等体量的消息测试确认没有问题以前，[这个 branch](#) 不会合并到主分支。

四、fill the rabbit hole

上面已经说到，整个项目的作者和用户们终于为动态表名付出了代价。为此，我编写了第二种方案的代码，以及对应的[升级函数](#)。

将所有同类型数据存到同一个表里，这应该是 SQL 数据库的最佳实践。无论如何，这一系列的问题总算告一段落了。