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

这最早是 Icalingua（下简称 <code>Il/code>）项目下的一个 issue，因比较详细地叙述了一系列 bug 的诞生，具备一定的参考价值，故立出来发布。
<h2 id="一-万恶之源--不当的数据组织方式">一、万恶之源——不当的数据组织方式</h2>
存储消息时，每个房间都建一个表（名为 <code>msg${roomId}</code>）。这是 Il 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 SQLStorageProvidr 和 RedisStorageProvider 的编写。在 MongoDB 和 Redis 这样的 NoSQL 数据库中，这么做并不增加太多复杂度。然而在 SQLStorageProvider 中，这种“表名与表数量都不固定”的结构却成倍增了代码量和复杂度。
<h2 id="二-水土不服--SQL-相关代码的编写">二、水土不服——SQL 相关代码的编写</h2>
NoSQL 数据库中，尝试向表（只有 Mongo 有对应“表”的概念——Collections，但我在 Redi 里也实现了这一层，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 数库并没有类似 <code>insertOrReplace</code> 和 <code>createTableIfNotExist</code> 这样功能（除了 PostgreSQL，但我用 <code>knex</code> 实现的 <code>SQLStorageProvider</cde> 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 有改动时，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，只好另建一表（表名为 <code>msgTableNameTable</code>，是不是有屎山的感觉了？），表存储各 <code>msg</code> 表的表名。
于是我实现了一个 <code>createMsgTable</code> 方法，在每次对表进行操作以前调用该法检查目标表是否存在，若不存在则先创建，创建时也向 <code>msgTableNameTable</code> 入 <code>msg</code> 表的表名，以便此后进行升级操作时能修改它们的 <code>columns</coe>。
由于上述额外操作，以及对各 <code>column</code> 类型的定义，<code>SQLStorageProvder.ts</code> 的行数是 <code>MongoStorageProvider.ts</code> 的两倍以上。众所周知，bug 量与代码行数正相关，因此 <code>SQLStorageProvider</code> 自诞生之日起就 bug 不断。

使用 SQLite 时，无法正创建数据库目录，导致崩溃
使用三类 SQL 作为存储方时，房间不会按消息先后顺序进行排序
使用三类 SQL 作为存储式时，艾特所有人的消息无法入库
使用三类 SQL 或 IndexeDB 作为存储方式时，无法撤回消息
使用 MongoDB 以外的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，<code>SQLStorageProvider </code> 进入一段稳定时期。<code>SQLStorageProvider</code> 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。我所做的就是在主作者提出新增/修改 columns 的 issues 时去编写对应的升级函数。此后，我为 <code>SQLStorageProvider</code> 添了相当详细的 jsDoc 注释，于是我也不再需要自己编写升级函数，只需偶尔 review 一下主作者的修即可。
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

，这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《Unrael》），直到这一天的到来。
<h2 id="三-异步灾难--过度优化打开的潘多拉魔盒">三、异步灾难——过度优化打开的潘多拉魔盒</h2>
当打开空的群聊的时候报，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 和 Icalingua。然而一后的今天，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。
<pre><code class="language-bash highlight-chroma">Uncaught (in promise) Error: Error invoking remote method 'fetchMessage': Error: create table `</span>msg-********<span class="highlight-sb">` (`</span>_id<span class="highlight-sb">` varchar(255), ..., primary key (`</span>_id<span class="highlight-sb">`)) - SQLITE_ERROR: table `</span>msg-1046496784<span class="highlight-sb">` already xists
</code></pre>
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，<code>createMsgTable</code> 会先调用 <code>hasTable</code> 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 <code>console.log()</code> 这的原始方法进行调试。
大约 5 分钟之后，我发现了罪魁祸首：<code>StorageProvider</code> 类（<code>SQLStorageProvider</code> 为对该类的一个实现）中的 <code>addRoom</code> 和 <code>fetchMessages</code> 方法被调用，但后者并非是等待前者异步返回后再被调用，而是同时被调用。
<pre><code class="language-ts highlight-chroma">const hasMsgTable = await this.db.schema.hasTable(`msg</span><span class="highlight-si">${</span><span class="highlight-nx">roomId</span><span class="highlight-si">}</span><span class="highlight-sb">`)
if (!hasMsgTable) {
  await this.db.schema.createTable(`msg</span><span class="highlight-si">${</span><span class="highlight-nx">roomId</span><span class="highlight-si">}</span><span class="highlight-sb">`, (table) => {
    .....
  }
}
</code></pre>
这是 <code>createMsgTable</code> 方法的片段。<code>addRoom</code> 和 <code>etchMessages</code> 均在开头调用了 <code>createMsgTable</code> 方法。该方法的作用

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时，灾难出现了。
由于主程序对 <code>addRoom</code> 和 <code>fetchMessages</code> 两个方法异步用且不等待返回的设计，两次调用中，<code>hasMsgTable</code> 均为 false，因此 createTable 会被调用两次。第二次调用时，由于表已存在，<code>SQLStorageProvider</code> 会抛错，用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：
<blockquote>
那我如果确实不需要等待呢？我希望多线程
</blockquote>
实际上这并不是多线程，但却有和多线程相当相似的结果：
<blockquote>
你有一个问题
你试图用多线程解决它
现你在两有个题问了有
</blockquote>
一个很自然的想法是通过加锁来解决。我在 <code>SQLStorageProvider</code> 中增加了个私有数组，每次调用 <code>createMsgTable</code> 方法时，都会检查数组中有无目标表名，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。<code>SQLStorageProvider</code> 报错依旧。进一步 <code>debug</code> 显示，我加入的的锁毫无作用，因为第二次被调用时，数组依然为空。</p

我并不知道 <code>SQLStorageProvider</code> 以何种方式被实例化，但如果它在主程序有大于等于两个实例，那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，锁也未能生效。而即使这个锁得以生效，由于主程序异步调用不等待返回，<code>createMsgTable</cod> 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：

放弃异步调用不等待返回的过度优化。
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 <code>StoageProvider</code> 的 <code>connect</code> 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，<a href "https://link.ld246.com/forward?goto=https%3A%2F%2Fgithub.com%2FIcalingua%2FIcalingu%2Ftree%2Ftest%2Ffill-the-rabbit-hole" target="_blank" rel="nofollow ugc">这个 branch 不会合并到主分支。
<h2 id="四-fill-the-rabbit-hole">四、fill the rabbit hole</h2>
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，我编写了第二种方的代码，以及对应的升级数。
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，这一系列的题可算告一段落了。