## 记一次慢查询程序的优化过程

* 业务场景：

查询数据需要连接A，B，C，D四张表，目前的数据A表70000000+，BCD表都将近3000000。用户首先根据时间段及其他一些条件筛选出一部分数据在界面上分组展示，然后选择其中的一个组进行抽样。

* 初次的做法及效率：

分组和抽样都由数据库处理并返回结果。发现一个月的数据平均需要1-2s秒。几个月的数据需要好几十秒。（这是在toad里观察到的时间，用户界面等待需要更久）

* 优化一：

因为SQL本身没有什么特别的，在SQL上的优化只是进行了where条件的调整，主键和索引能用的都用上。最后经过对比，不进行分组的情况下一个月的数据查询降到了100-200ms，几个月的数据也在1s以内。但是加上分组，效率还是和原先一样，执行计划也看不出什么大问题。（为什么？）

* 优化二：

将分组的处理放到程序里。具体做法是通过reader逐条读出数据，利用字典统计分组。经过测试，一个月的数据平均需要10s（用户界面等待时间）左右，几个月的数据大概需要几十秒，就用户的等待时间来说虽有提升但总体来说还是不乐观。另外抽样怎么办？在事先不知道数据总量的情况下抽样好像比较麻烦。从而有了优化三。

* 优化三：

在此业务场景中，用户要进行多次检索，检索的数据大部分情况下都落在一个固定的范围内。那么是否可以对这一部分数据进行会话级的缓存？

可行性：30家医院，每家医院顶多5个人，最多会在一个月的某几天使用。所以保守估计同时在线会话数撑死也不会超过总人数的一半。另外如果缓存一个月的数据大约需要5M左右的内存。所以缓存内存需求300M应该足够。

具体做法：根据用户必选（也就是范围尽可能大）的几个条件作为缓存标识，后续的检索条件如果在已缓存的条件范围内那么直接在缓存的数据上进行过滤并返回分组和抽样的结果，否则就重新从数据库检索并更新缓存。

缺点：会话存储由之前的进程外（需要频繁的序列化和反序列化大量的缓存数据）改为进程内。

效果：第一次或者缓存没有命中的情况下，检索需要花点时间（一个月10s）。在缓存命中的情况下，几个月的数据基本不需要等待时间。

程序正在试用中，等待后续反馈。。。

* 总结：

1. 当sql查询无能为力的时候（或者给SQL太多职责的时候），可以考虑通过程序来分担一部分工作。因为sql复杂度有可能急剧降低其效率。
2. 优化需要根据实际情况进行，不是什么优化方法都可以拿来套用。
3. 不能做到最好，但往往可以做得更好。