Atitit 结算性能解决方法总结

目录

[1. 优先使用配置法解决 3](#_Toc19558)

[1.1. 优先配置解决模式（更加简单稳定可靠） 3](#_Toc17617)

[1.2. 代码微调（sql优化类） 3](#_Toc21086)

[1.3. 代码大调整（风险较大） 3](#_Toc28733)

[2. 数据库db mysql对参数调整 4](#_Toc573)

[2.1. 主要是对死锁优化 4](#_Toc3027)

[2.2. 执行响应速度优化 4](#_Toc26878)

[2.3. 多线程运行对优化 4](#_Toc31142)

[2.4. Cache缓存优化（开启查询 写入缓存） 4](#_Toc19811)

[2.5. 版本升级对性能的优化很大 4](#_Toc1668)

[3. Sql优化 4](#_Toc11720)

[3.1. 将sql存储在数据库端距离更近，响应时间更短更快 4](#_Toc12533)

[3.2. c) 尽可能减少基于范围的数据检索过滤条件，避免因为间隙锁 4](#_Toc10672)

[3.3. Sql强制索引，强制走某条性能更高的索引 4](#_Toc12582)

[3.4. 强制cache和no cache的使用优化 5](#_Toc7297)

[3.5. 多种索引hash btree 全文索引等适当分场景使用 5](#_Toc4390)

[3.6. Uniq索引的使用 5](#_Toc6983)

[3.7. Insert delay优化 5](#_Toc24613)

[3.8. Update 和delete语句的cache优化 5](#_Toc21884)

[4. 编程语言与Vm优化配置 5](#_Toc8340)

[4.1. 开启更高级别的vm代码cache 5](#_Toc6438)

[4.2. 开启性能日志profile仔细分析 5](#_Toc20564)

[4.3. 多条sql业务场景下慎用orm类型框架，会减慢sql响应时间 5](#_Toc31944)

[4.4. 外挂框架适当使用与限制范围 5](#_Toc9054)

[4.5. 调优vm 编译级别与gc模式 5](#_Toc1531)

[5. Os操作系统优化 5](#_Toc3194)

[5.1. 提升mysql进程优先级 降低非必要进程优先级 6](#_Toc6568)

[5.2. 全面调整cache设置 6](#_Toc19726)

[5.3. other 6](#_Toc11065)

[6. 硬件方面的优化（略） 6](#_Toc19469)

[7. 较大的配置解决法 6](#_Toc2627)

[7.1. Other 读写优先级设置 6](#_Toc13693)

[7.2. 集群 读写分离 6](#_Toc27244)

[7.3. 多主集群等 6](#_Toc12093)

[7.4. 针对不同的业务场景使用不同的数据隔离级别 7](#_Toc24477)

[8. 框架性能调整优化 7](#_Toc5361)

[8.1. Spring cache？ 7](#_Toc6346)

[8.2. -Tk.Mybatis框架优化 Mybatis框架性能调整 7](#_Toc31797)

[8.3. 密集操作慎用orm框架，拼接sql性能更优 7](#_Toc13766)

[8.4. 连接池框架性能调整 7](#_Toc2953)

[8.5. Mysql驱动性能 7](#_Toc14602)

[9. 框架优化》使用更快的组件 7](#_Toc16797)

[9.1. 框架优化 更高性能的rest web容器（略） 7](#_Toc23634)

[9.2. 更快的数据库连接池等 8](#_Toc30683)

[10. 框架优化 ----spring 8](#_Toc1913)

[10.1. 限制事务范围防止死锁 8](#_Toc611)

[10.2. 调整spring 事务管理器 缩小事务范围 减少表锁 8](#_Toc25352)

[10.3. 适当对cache如果有的化 8](#_Toc8411)

[10.4. Lazy 加载datasource 8](#_Toc3146)

[11. 框架优化 Tk.Mybatis框架优化 8](#_Toc20155)

[11.1. 框架性能优化---Mybatis框架性能调整 开启cache 到redis 8](#_Toc24287)

[11.2. 延迟加载机制 9](#_Toc14079)

[11.3. 二级缓存机制 9](#_Toc10285)

[12. 框架优化--- Mysql驱动性能参数配置 9](#_Toc28260)

[12.1. allowMultiQueries 多语句支持开启 9](#_Toc31526)

[12.2. 不要使用ssl模式。。性能下降俩三倍之多 10](#_Toc28832)

[12.3. 驱动级连接池的优化 10](#_Toc20260)

[12.4. callableStmtCacheSize 缓存多少个可调用语句 10](#_Toc45)

[12.5. prepStmtCacheSize 应缓存多少个预处理语句 10](#_Toc11010)

[12.6. 开启 性能日志 12](#_Toc17828)

[12.6.1. 6.3.14调试/性能分析 12](#_Toc26002)

[13. 性能代码更改 12](#_Toc12985)

[13.1. 开启mlutisql 一次发送多条sql 12](#_Toc3869)

[13.2. 三级多级cache机制 13](#_Toc3674)

[13.3. 开启spring和mybatis query cache 13](#_Toc29589)

[13.4. 增加部分gc代码 强制超时释放连接 13](#_Toc31359)

[13.5. 可以对某些表不使用innodo存储引擎，使用其他更高性能存储引擎如pgsql msslq oralce等存储引擎 13](#_Toc25501)

[13.6. Nosql引擎 mongodb等 13](#_Toc2469)

[13.7. 实现页锁 对数据分页锁定缩小锁定范围。。会大概率减少锁定 13](#_Toc8854)

[13.8. 优化mysql行锁机制 实现真正数据行行锁 13](#_Toc26900)

[13.9. 适当分库，隔离锁定范围 14](#_Toc2643)

[13.10. 尽可能缩小事务范围 14](#_Toc3677)

[13.11. 开启线程池模式 14](#_Toc14623)

[13.12. 大力使用nosql 类型数据，json等。最新数据库基本都支持 14](#_Toc2477)

[14. 改业务代码（风险最大，建议最后解决方案） 14](#_Toc1333)

[14.1. 大力使用nosql 字段 14](#_Toc27634)

[14.2. 增加mq事件机制 14](#_Toc2284)

[14.3. 适当视图join 减少代码级别的join 14](#_Toc26336)

[14.4. 全面针对业务使用sp存储过程+触发器 优化 ，减少大量网络往返，缩短事务处理响应时间 14](#_Toc29967)

[14.5. 异步线程底层api太繁琐，嵌套太多。。 14](#_Toc15954)

[15. Ref 15](#_Toc836)

# 优先使用配置法解决

改业务代码可能带来较大分险，导致业务正确性出错更麻烦。。

## 优先配置解决模式（更加简单稳定可靠）

包括db配置，框架配置，vm配置 os配置等

## 代码微调（sql优化类）

## 代码大调整（风险较大）

# 数据库db mysql对参数调整

## 主要是对死锁优化

## 执行响应速度优化

## 多线程运行对优化

## Cache缓存优化（开启查询 写入缓存）

## 版本升级对性能的优化很大

# Sql优化

## 将sql存储在数据库端距离更近，响应时间更短更快

特别是批量修改的情况下

## c) 尽可能减少基于范围的数据检索过滤条件，避免因为间隙锁

带来的负面影响而锁定了不该锁定

的记录；

强制索引的使用

## Sql强制索引，强制走某条性能更高的索引

## 强制cache和no cache的使用优化

## 多种索引hash btree 全文索引等适当分场景使用

## Uniq索引的使用

## Insert delay优化

## Update 和delete语句的cache优化

# 编程语言与Vm优化配置

## 开启更高级别的vm代码cache

## 开启性能日志profile仔细分析

## 多条sql业务场景下慎用orm类型框架，会减慢sql响应时间

## 外挂框架适当使用与限制范围

## 调优vm 编译级别与gc模式

# Os操作系统优化

适当的swap

## 提升mysql进程优先级 降低非必要进程优先级

如有可能限制其他进程使用的cpu核心数量

## 全面调整cache设置

## other

# 硬件方面的优化（略）

# 较大的配置解决法

## Other 读写优先级设置

low\_priority\_updates 等

## 集群 读写分离

使主库集中精力做修改类操作。。读取类操作从从库进行。。原则上，大部分80%是查询语句。。

## 多主集群等

主库也多台分散了压力

## 针对不同的业务场景使用不同的数据隔离级别

# 框架性能调整优化

## Spring cache？

## -Tk.Mybatis框架优化 Mybatis框架性能调整

## 密集操作慎用orm框架，拼接sql性能更优

## 连接池框架性能调整

## Mysql驱动性能

# 框架优化》使用更快的组件

## 框架优化 更高性能的rest web容器（略）

更高性能 Web 服务器 更高版本

## 更快的数据库连接池等

# 框架优化 ----spring

## 限制事务范围防止死锁

## 调整spring 事务管理器 缩小事务范围 减少表锁

尽量控制事务的大小，减少锁定的资源量和锁定时间长度；

## 适当对cache如果有的化

## Lazy 加载datasource

多ds下只加载需要的。。比如test dev环境等。。

介绍如何实现dataDataSource的lazy加载。好处在于：（1）对于不同任务，其实依赖dataSoruce不同，不会因为不相干的数据库挂掉，而阻塞任务；（2）而且还有另外一个好处在于减少不必要连接数，只初始化用到的dataSourche。

# 框架优化 Tk.Mybatis框架优化

tkmybatis开启二级缓存\_苏凯的博客-CSDN博客

## 框架性能优化---Mybatis框架性能调整 开启cache 到redis

## 延迟加载机制

延迟加载又叫懒加载，也叫按需加载。

<!-- 配置mybatis的缓存，延迟加载等等一系列属性 -->

<settings>

<setting name=*"lazyLoadingEnabled"* value=*"true"* />

## 二级缓存机制

因为二级缓存是mapper级别的，当一个商品的信息发送更新，所有的商品信息缓存数据都会清空。

解决此类问题，需要在业务层根据需要对数据有针对性的缓存。

比如可以对经常变化的 数据操作单独放到另一个namespace的mapper中。

# 框架优化--- Mysql驱动性能参数配置

## allowMultiQueries 多语句支持开启

## 不要使用ssl模式。。性能下降俩三倍之多

## 驱动级连接池的优化

## callableStmtCacheSize 缓存多少个可调用语句

如果启用了“ cacheCallableStmts”，应缓存多少个可调用语句？

|  |  |
| --- | --- |
| **默认值** | 100 |

## prepStmtCacheSize 应缓存多少个预处理语句

如果启用了预处理语句缓存，则应缓存多少个预处理语句？

|  |  |
| --- | --- |
| **默认值** | 100 |
|  |  |

prepStmtCacheSqlLimit

如果启用了预处理语句缓存，驱动程序将缓存解析的最大SQL是什么？

|  |  |
| --- | --- |
| **默认值** | 256 |
| **自版本** | 3.0.10 |

cacheCallableStmts

驱动程序是否应该缓存CallableStatements的解析阶段

|  |  |
| --- | --- |
| **默认值** | 假 |
| **自版本** | 3.1.2 |

cachePrepStmts

驱动程序是否应该缓存客户端准备好的语句的PreparedStatements的解析阶段，服务器端准备好的语句和服务器端准备好的语句本身的“检查”？

|  |  |
| --- | --- |
| **默认值** | 假 |
| **自版本** | 3.0.10 |

cacheResultSetMetadata

驱动程序是否应该为语句和PreparedStatements缓存ResultSetMetaData？（要求JDK-1.4 +，true / false，默认为“ false”）

|  |  |
| --- | --- |
| **默认值** | 假 |
| **自版本** | 3.1.1 |

largeRowSizeThreshold

JDBC驱动程序应将哪种大小的结果集行视为“大”，从而使用内存效率更高的方法在内部表示该行？

|  |  |
| --- | --- |
| **默认值** | 2048 |
| **自版本** | 5.1.1 |

rewriteBatchedStatements

在调用executeBatch（）时，驱动程序应该使用多查询（无论“ allowMultiQueries”的设置如何）以及将INSERT的准备好的语句重写为多值插入吗？请注意，如果使用纯java.sql.Statements并且您的代码未正确清理输入，则这有可能进行SQL注入。请注意，对于预处理语句，服务器端预处理语句当前无法利用此重写选项，并且如果在使用PreparedStatement.set \* Stream（）时未指定流长度，则驱动程序将不会 不能确定每批参数的最佳数量，您可能会从驱动程序收到错误消息，提示结果包太大。这些重写语句的Statement.getGeneratedKeys（）仅在整个批处理都包含INSERT语句时才起作用。请注意，在INSERT上使用rewriteBatchedStatements = true。在重复键更新中，对于重写的语句服务器，批处理中所有受影响（或找到的）行的总和仅返回一个值，并且无法将其正确映射到初始语句；在这种情况下，如果总计数为0，则驱动程序针对每个批处理语句返回0，然后返回该语句。

|  |  |
| --- | --- |
| **默认值** | 假 |
| **自版本** | 3.1.13 |

useReadAheadInput

从服务器读取数据时，使用更新的，优化的非阻塞缓冲输入流吗？

|  |  |
| --- | --- |
| **默认值** | 真正 |
| **自版本** | 3.1.5 |

* MySQL :: MySQL Connector / J 8.0开发人员指南:: 6.3.13性能扩展

## 开启 性能日志

### 6.3.14调试/性能分析

autoGenerateTestcaseScript 等参数 日志参数等

驱动程序是否应该转储正在执行的SQL，包括向STDERR发送服务器端准备好的语句？

|  |  |
| --- | --- |
| **默认值** | 假 |
| **自版本** | 3.1.9 |

# 性能代码更改

## 开启mlutisql 一次发送多条sql

特别是select 再修改类的代码，一次性在服务端执行更快，免去网络io延时带来对事务延时

## 三级多级cache机制

## 开启spring和mybatis query cache

## 增加部分gc代码 强制超时释放连接

数据库连接池 立即释放 过长事务和死锁连接

## 可以对某些表不使用innodo存储引擎，使用其他更高性能存储引擎如pgsql msslq oralce等存储引擎

不过对视图join和存储过程sp会带来些问题，fdw外部表挂载功能或许可以缓解此类问题，要不就要代码实现join

## Nosql引擎 mongodb等

## 实现页锁 对数据分页锁定缩小锁定范围。。会大概率减少锁定

比如分为100页数据，则分别加锁加在100个页上，每个页面内锁定对概率就降低到百分之一类。。

## 优化mysql行锁机制 实现真正数据行行锁

在MySQL机制下，实现行锁，需要拆分表格数据。。。表锁行锁一体化。有多少用户，就要建立多少表。。虽然mysql没有限制库表数量，但大量表格可能影响些问题。。

## 适当分库，隔离锁定范围

## 尽可能缩小事务范围

## 开启线程池模式

## 大力使用nosql 类型数据，json等。最新数据库基本都支持

# 改业务代码（风险最大，建议最后解决方案）

## 大力使用nosql 字段

## 增加mq事件机制

## 适当视图join 减少代码级别的join

## 全面针对业务使用sp存储过程+触发器 优化 ，减少大量网络往返，缩短事务处理响应时间

Sp也可编译为机器码，更快

## 异步线程底层api太繁琐，嵌套太多。。

# Ref

提高MySQL性能的7个关键 信息世界

mysql在高内存、IO利用率上的几个优化点 (sync+fsync) 猎豹移动技术博客 - zengkefu - 博客园

MySQL 8.0新特性全面认识\_数据库技术\_Linux公社-Linux系统门户网站

Atitit 提升稳定性 数据库死锁

目录

1.1. 配置数据库死锁检测超时时间从默认50s到10s 1

1.2. 调整隔离级别到read commit 1

1.3. mysql数据库连接使用完毕立即释放掉。不在mysql上使用连接池 1

1.4. 每个连接增加gc检测，超时强制销毁退出。 1

1.5. 分库，分为100个库，自然提升稳定性， 2

1.6. 使用小事务机制 只针对必要业务上大事务 2

2. 常见死锁问题原因 2

2.1. 行锁变表锁导致死锁问题 2

2.2. Ab ba死锁问题 2

2.2.1. .1 死锁案例一 3

2.2.2. 3.2 死锁案例二 gap锁 3

2.2.3. 3.3 死锁案例三 范围批量修改 5