Atitit 系统性能优化流程

目录

[1. 分析性能 性能瓶颈在代码还是db 2](#_Toc14591)

[1.1. 要求立竿见影速度快，，配置式样优先 2](#_Toc10782)

[1.2. 优化三大件 简化 cache 多路 转移 2](#_Toc31729)

[2. 配置 优化vs 代码模式优化 2](#_Toc22233)

[3. 配置模式调优 根据 性能优化 top5 大方法 2](#_Toc21801)

[3.1. 负载均衡架构 数据库集群读写分离 分库 3](#_Toc27791)

[3.2. 前端cache cdn，后端db中间件cache（一般集成redis） ，db query cache 3](#_Toc28941)

[3.3. 调整各类中间件参数 看有没有性能与cache一类提升机制 3](#_Toc15981)

[3.4. 配置数据库连接池 3](#_Toc21743)

[3.5. 调整的内容包括更换服务器，从websphere改为tomcat， 3](#_Toc10229)

[3.6. 更改启动参数 3](#_Toc29683)

[4. 编程语言配置优化 3](#_Toc26295)

[4.1. 更高对版本 一般性能更好 3](#_Toc22627)

[4.2. 代码cache opcache一类对 3](#_Toc31466)

[4.3. Vm调优 gc ，， jit c1 c2等 3](#_Toc27743)

[4.4. 代码缓存 opcache一类的 3](#_Toc3534)

[5. 代码微调类 4](#_Toc1354)

[5.1. 热部署配置调整，加快启动速度，不要开始就加载所有 4](#_Toc5102)

[5.2. 简化 缩短各种调用 4](#_Toc12133)

[5.3. 针对每个模块可使用性能更高的组件 4](#_Toc21192)

[5.4. 去大型框架重型框架 orm之类对 4](#_Toc25459)

[6. 数据库优化 4](#_Toc27750)

[6.1. 多利用数据库对性能机制 4](#_Toc24163)

[6.2. 配置优化 query cache 4](#_Toc26846)

[6.3. 索引优化 sql优化 4](#_Toc21320)

[6.4. 并发线程数 4](#_Toc13540)

[6.5. 配置优化 4](#_Toc12851)

[7. 代码模式改动调优，较为麻烦 5](#_Toc1484)

[7.1. 简化 优化性能 5](#_Toc21155)

[7.2. 部分模块换用高性能语言重写 5](#_Toc22185)

[7.3. 前端转移部分业务逻辑 5](#_Toc1701)

[7.4. 使用View sp模式优化 5](#_Toc3959)

[7.5. 6. 减少不必要的类# 6](#_Toc995)

[7.6. 尽可能使用内置库函数 6](#_Toc11881)

[7.7. 业务逻辑前端化尽可能 6](#_Toc14892)

[8. 性能优化几大方法 6](#_Toc19666)

[8.1. 配置化解决和vm优化最简单 6](#_Toc11070)

[8.2. 架构配置优化 vm优化 代码语言优化 6](#_Toc25678)

[9. Ref 6](#_Toc13783)

# 分析性能 性能瓶颈在代码还是db

## 要求立竿见影速度快，，配置式样优先

性能瓶颈不在语言层面，比如瓶颈在数据库层面，外部依赖的接口等等。

## 优化三大件 简化 cache 多路 转移

# 配置 优化vs 代码模式优化

# 配置模式调优 根据 性能优化 top5 大方法

Atitit perf 性能优化 top5 大方法

## 负载均衡架构 数据库集群读写分离 分库

## 前端cache cdn，后端db中间件cache（一般集成redis） ，db query cache

## 调整各类中间件参数 看有没有性能与cache一类提升机制

## 配置数据库连接池

## 调整的内容包括更换服务器，从websphere改为tomcat，

## 更改启动参数

# 编程语言配置优化

## 更高对版本 一般性能更好

## 代码cache opcache一类对

## Vm调优 gc ，， jit c1 c2等

## 代码缓存 opcache一类的

# 代码微调类

## 热部署配置调整，加快启动速度，不要开始就加载所有

## 简化 缩短各种调用

## 针对每个模块可使用性能更高的组件

## 去大型框架重型框架 orm之类对

不使用外挂框架的 PHP 的运算性能表现都很优异，但是受到外加框架的影响，原本可以在数秒内处理上千个网页要求的 PHP，性能大幅下降，变为只能处理数十 个要求。

😂orm框架很影响性能啊，在这类复杂接口里面直接拼sql吧。。框架加载很耗时

# 数据库优化

## 多利用数据库对性能机制

## 配置优化 query cache

## 索引优化 sql优化

## 并发线程数

## 配置优化

****内存配置相关参数****

1.sort\_buffer\_size：排序缓冲区大小，这个参数定义了每个线程所使用的排序缓冲区的大小，当查询请求需要排序时，MySQL服务器会立即分配给链接这个参数所定义大小的缓冲区的全部内存。而不管这个排序是否需要这么大的内存，如果把这个参数设置成100M，那么10个需要排序查询的链接就会占用1G的内存，因此这个参数的配置要格外小心，不要将这个参数的值设置的过大。

2.join\_buffer\_size：这个参数定义的是MySQL的每个线程所使用的连接缓冲去的大小，如果查询中关联了多张表，那么MySQL就会为每一个连接分配一个缓冲区，因此这个参数也不要设置的过大。在这里需要提醒一下，在互联网环境中，为了提升性能，尽量避免使用连表查询，尤其是大表的连表查询。

****IO相关配置参数****

innodb是基于事务的存储引擎，每次数据修改，innodb首先要将事务预写到事务日志中，然后再写入磁盘，而不是每次提交都写入磁盘，因为磁盘IO的性能比较差。

1.innodb\_log\_file\_size：单个事务日志大小

2.innodb\_log\_files\_in\_group：innodb事务日志文件的数量

# 代码模式改动调优，较为麻烦

## 简化 优化性能

## 部分模块换用高性能语言重写

Atitit perf 性能优化 top5 大方法

## 前端转移部分业务逻辑

## 使用View sp模式优化

## **6.** 减少不必要的类**[#](https://learnku.com/articles/4685/php-application-performance-tuning-guide" \l "4a1bdf)**

## 尽可能使用内置库函数

## 业务逻辑前端化尽可能

业务逻辑尽可能前端化。。一般来说那个格式化逻辑貌似是前端要干的的事情啊 ，后端只提供最基础数据，前端格式化拼接展示，这样大大减轻后端压力提升并发哟

大力简化后端，提升性能并发

看能不能把部分 属于展示类的逻辑 转移到前端去处理咯

# 性能优化几大方法

## 配置化解决和vm优化最简单

负载均衡 cache

## 架构配置优化 vm优化 代码语言优化

# Ref

PHP 应用性能优化指南 | Laravel China 社区

MySQL数据库参数优化\_nch\_ren的博客-CSDN博客\_mysql数据库参数优化