Atitit.提升软件Web应用程序 app性能的方法原理 h5 js java c# php python android .net

[1. 提升单例有能力的 1](#_Toc7391)

[2. 减少工作数量 2](#_Toc26963)

[2.1. 减少距离 2](#_Toc28627)

[2.2. 添加索引 2](#_Toc4268)

[2.3. Dma api 2](#_Toc26952)

[2.4. Cdn 2](#_Toc25966)

[2.5. Cache 2](#_Toc30907)

[2.6. Pool技术 2](#_Toc6790)

[2.7. 减少HTTP请求数 2](#_Toc2372)

[2.8.  ·更大的数据格式 2](#_Toc19227)

[2.9. 循环展开 2](#_Toc31763)

[2.10. 循环转置 3](#_Toc25504)

[2.11. 提高Cache命中率 3](#_Toc25549)

[2.12. 小组件替换大组件 4](#_Toc3127)

[3. 并行处理 4](#_Toc26468)

[3.1. 多线程 4](#_Toc11957)

[3.2. 数据库分区 4](#_Toc20260)

[4. 减少等候 4](#_Toc18791)

[4.1.  2、流水线（Pipeline） 4](#_Toc19419)

[4.2. 乱序执行---相关 5](#_Toc9458)

[4.3. 分支预测 5](#_Toc7112)

[4.4. 静态指令调度and 动态指令调整 5](#_Toc1704)

[4.5. 运算顺序调整 6](#_Toc5838)

[5. 预准备好 6](#_Toc30521)

[5.1. 查表法 6](#_Toc23000)

[5.2. 预加载 6](#_Toc17164)

[6. Lazy load 6](#_Toc15915)

[6.1. 延时脚本 6](#_Toc10958)

[7. 参考 6](#_Toc1110)

# 提升单例有能力的

# 减少工作数量

## 减少距离

## 添加索引

## Dma api

## Cdn

## Cache

## Pool技术

## ****减少HTTP请求数****

## ·****更大的数据格式****

如今的 64 位处理器是当初的 4 位处理器长期演变的结果。每次转变都使得处理更大的数据元素速度更快，并支持处理器直接寻址更大的内存。这些先进特性对于性能的提升和能力上的扩充是十分关键的，而没有这些先进特性，今天的数据密集型应用将难以想像。

对比一下可发现，32 位处理器可直接寻址高达 4 GB 的内存；而 64 位处理器理论上则可寻址高达约 1,800 万 TB的内存（实际寻址容量取决于具体部署）。这是一次巨大的飞跃，很可能需要经过一段时间以后，硬件及软件开发人员才能充分利用那些他们现在已经能够利用的较大寻址空间。

作者::  ★(attilax)>>>  绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ） 汉字名：艾龙，  EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax

## 循环展开

循环展开将两个或更多个循环迭代合并到一起，以减少循环计数。循环展开虽然常常会导致代码长度增加，但它可以减少必须执行的指令数。下面是一个非常简单的循环展开示例，它从循环中删除了一个分支：

这是经典的速度优化,但许多编译程序(如gcc -funroll-loops)能自动完成这个事,所以现在你自己来优化这个显得效果不明显。

不过注意:对于中间变量或结果被更改的循环,编译程序往往拒绝展开,(怕担责任呗),这时候就需要你自己来做展开工作了.

还有一点请注意,在有内部指令cache的CPU上(如MMX芯片),因为循环展开的代码很大,往往cache溢出,这时展开的代码会频繁地在CPU 的cache和内存之间调来调去,又因为cache速度很高,所以此时循环展开反而会变慢.还有就是循环展开会影响矢量运算优化。

在最内层循环避免使用全局变量和静态变量,除非你能确定它在循环周期中不会动态变化,大多数编译器们优化变量仅有置成寄存器变量一招,而对于动态变量,它们干脆放弃对整个表达式的优化.

## 循环转置

有些机器对JNZ(为0转移

有些机器对JNZ(为0转移)有特别的指令处理,速度非常快,如果你的循环对方向不敏感,可以由大向小循环。

旧代码:  
     for (i = 1; i <= MAX; i++)  
    {  
        ...  
    }

新代码:  
    i = MAX+1;  
    while (--i)  
    {  
        ...  
    }

不过千万注意,如果指针操作使用了i值,这种方法可能引起指针索引超界的严重错误(i = MAX+1;).当然你可以通过对i做加减运算来纠正,但是这样加速的作用就没有了除非类似于以下情况

## 提高Cache命中率

·数组合并·循环交换·循环合并

·分块

集中访问可取入cache的块状矩阵，避免全行或全列的读写，以增强时间局部性。

## 小组件替换大组件

Bootstrap 替换 amazeui

# 并行处理

## 多线程

## 数据库分区

# 减少等候

## ****2、流水线（Pipeline）****

　　流水线是现代RISC核心的一个重要设计，它极大地提高了性能。  
　　对于一条具体的指令执行过程，通常可以分为五个部分：取指令，指令译码，取操作数，运算（ALU），写结果。其中前三步一般由指令控制器完成，后两步则由运算器完成。按照传统的方式，所有指令顺序执行，那么先是指令控制器工作，完成第一条指令的前三步，然后运算器工作，完成后两步，在指令控制器工作，完成第二条指令的前三步，在是运算器，完成第二条指令的后两部……很明显，当指令控制器工作是运算器基本上在休息，而当运算器在工作时指令控制器却在休息，造成了相当大的资源浪费。解决方法很容易想到，当指令控制器完成了第一条指令的前三步后，直接开始第二条指令的操作，运算单元也是。这样就形成了流水线系统，这是一条2级流水线。

## 乱序执行---****相关****

　　在一个流水线系统中，如果第二条指令需要用到第一条指令的结果，这种情况叫做相关。以上面哪个5级流水线为例，当第二条指令需要取操作数时，第一条指令的运算还没有完成，如果这时第二条指令就去取操作数，就会得到错误的结果。所以，这时整条流水线不得不停顿下来，等待第一条指令的完成。这是很讨厌的问题，特别是对于比较长的流水线，比如20级，这种停顿通常要损失十几个时钟周期。目前解决这个问题的方法是乱序执行。乱序执行的原理是在两条相关指令中插入不相关的指令，使整条流水线顺畅。比如上面的例子中，开始执行第一条指令后直接开始执行第三条指令（假设第三条指令不相关），然后才开始执行第二条指令，这样当第二条指令需要取操作数时第一条指令刚好完成，而且第三条指令也快要完成了，整条流水线不会停顿。当然，流水线的阻塞现象还是不能完全避免的，尤其是当相关指令非常多的时候。

一般来说，当一条语句计算出来的值，后续语句不要理解使用，需要隔几条语句（通常为4条语句）后再使用前面计算得出的值，这是因为流水线原因，当前面的值计算比较费时，后面的指令就不能执行，流水线就会被阻塞，有些编译器可以优化，采用乱序执行方式，让计算语句后面的指令在不等计算语句完成前接着执行，这样就能够保证流水线的平滑。

## 分支预测

****转移****  
　　另一个大问题是条件转移。在上面的例子中，如果第一条指令是一个条件转移指令，那么系统就会不清楚下面应该执行那一条指令？这时就必须等第一条指令的判断结果出来才能执行第二条指令。条件转移所造成的流水线停顿甚至比相关还要严重的多。所以，现在采用分支预测技术来处理转移问题。虽然我们的程序中充满着分支，而且哪一条分支都是有可能的，但大多数情况下总是选择某一分支。比如一个循环的末尾是一个分支，除了最后一次我们需要跳出循环外，其他的时候我们总是选择继续循环这条分支。根据这些原理，分支预测技术可以在没有得到结果之前预测下一条指令是什么，并执行它。现在的分支预测技术能够达到90%以上的正确率，但是，一旦预测错误，CPU仍然不得不清理整条流水线并回到分支点。这将损失大量的时钟周期。所以，进一步提高分支预测的准确率也是正在研究的一个课题。  
　　越是长的流水线，相关和转移两大问题也越严重，所以，流水线并不是越长越好，超标量也不是越多越好，找到一个速度与效率的平衡点才是最重要的。

## ****静态指令调度and 动态指令调整****

## 运算顺序调整

通常的算法设计和程序实现中，人们习惯在需要某数据的地方才计算出该数据的值，紧接着使用该数据。这是很自然的思维习惯，但对于流水线则会造成麻烦。

两个运算相继进行，但后一个运算需要的操作数还没有被计算出来，只有原地等待，造成了流水线的停滞。

# 预准备好

## 查表法

## 预加载

# Lazy load

## 延时脚本

# 参考

提升Web应用程序性能的最佳实践-CSDN.NET.html

[分享]提升系统性能方法初步探讨 - 软件行业 - 畅享论坛.html