Atitit 编程语言性能优化方法总结

目录

[1. 编程方法优化 1](#_Toc11297)

[1.1. 少用虚函数 减少多态动态 尽可能静态方法函数 1](#_Toc20441)

[1.2. 方法不要代码太少，加大内联inline代码 2](#_Toc778)

[1.3. 尽可能使用stack变量，不要传递引用 2](#_Toc14237)

[1.4. 对象序列化传递可能比加载更快 2](#_Toc29666)

[1.5. 指定类型 优化obj模式 2](#_Toc9164)

[1.6. 代数简化 2](#_Toc19922)

[1.7.  final、static 域等），称为静态解析 2](#_Toc10915)

[2. 编译器优化 2](#_Toc10997)

[2.1. Jit .JIT编译 (JIT Compile) 2](#_Toc12125)

[2.2. 使用更快对存储寄存器，使用stack变量优先 2](#_Toc29953)

[2.3. 垃圾回收管理 2](#_Toc17073)

[2.4. 3.内嵌缓存(inline cache) 2](#_Toc30781)

[2.5. 快照 2](#_Toc19572)

[2.6. 强类型机制 或相同类型防止触发优化 回滚 2](#_Toc16173)

[2.7. other 2](#_Toc28437)

[3. 10.编译器的优化 3](#_Toc1179)

[3.1. 1）方法内联优化 3](#_Toc23550)

[3.2. 2）公共子表达式消除 与 代数简化 5](#_Toc14052)

[3.3. 3） 数组边界检查消除 5](#_Toc18482)

[3.4. 4）逃逸分析 5](#_Toc7860)

[4. Java对优化方法 6](#_Toc27290)

[4.1. C1 和 C2 编译器 6](#_Toc3220)

[4.2. 分层编译技术 6](#_Toc31190)

[5. 6](#_Toc19445)

[6. Node v8引擎优化 6](#_Toc3477)

[6.1. 隐藏类 6](#_Toc20392)

[7. Php7对性能优化 6](#_Toc10183)

[8. ref 7](#_Toc27349)

# 编程方法优化

## 少用虚函数 减少多态动态 尽可能静态方法函数

## 方法不要代码太少，加大内联inline代码

通过注释分割 加强可读性

## 尽可能使用stack变量，不要传递引用

## 对象序列化传递可能比加载更快

## 指定类型 优化obj模式

## 代数简化

## final、static 域等），称为静态解析

# 编译器优化

## Jit .JIT编译 (JIT Compile)

## 使用更快对存储寄存器，使用stack变量优先

## 垃圾回收管理

## 3.内嵌缓存(inline cache)

## **快照**

V8引入了快照机制，将内置的对象和函数加载之后的内存保存并序列化。序列化以后的结果很容易反序列化，经过快照机制的启动时间可以缩减几毫秒。快照机制也可以将一些开发者认为需要的JS文件序列化来减少处理事件。

## 强类型机制 或相同类型防止触发优化 回滚

* 。一个函数应该使用比较少的数据类型；对于数组，应尽量存放相同类型的数据，这样就可以通过偏移位置来访问。

## other

多使用栈内存

方法内联

# 10.编译器的优化

## 1）方法内联优化

内联优化是：一是去除方法调用的成本（如建立栈帧等），二是为了其他优化建立良好的基础

****方法的调用过程：****

　　(1) 首先会有个执行栈，存储目前所有活跃的方法，以及它们的本地变量和参数；

　　(2) 当一个新的方法被调用了，一个新的栈帧会被加到当前线程的栈顶，分配的本地变量和参数会存储在这个栈帧中；

　　(3) 跳到目标方法代码执行；

　　(4) 方法返回的时候，本地方法和参数会被销毁，栈顶被移除；

　　(5) 返回原来地址执行；

****方法内联的原理：****

方法内联就是把被调用方函数代码”复制”到调用方函数中，来减少因函数调用开销的技术。

一个简单的两数相加程序，被内联前的代码：

private int add1(int a, int b, int c, int d){

return add2(a + b) + add2(c + d);

}

private int add2(int x, int y){

return x + y;

}

运行一段时间后JVM会把add2方法去掉，并把你的代码翻译成：

private int add1(int a, int b, int c, int d){

return a + b + c + d;

}

****注意点：****

1）方法内联的其他隐含条件：

虽然JIT号称可以针对代码全局的运行情况而优化，但是JIT对一个方法内联之后，还是可能因为方法被继承，导致需要类型检查而没有达到性能的效果

想要对热点的方法使用上内联的优化方法，最好尽量使用final、private、static这些修饰符修饰方法，避免方法因为继承，导致需要额外的类型检查，而出现效果不好情况。

## 2）公共子表达式消除 与 代数简化

）代码进入即时编译器JIT中：

　　int d = E \* 12 + a + (a + E);

3) 这时即时编译器有可能进行另外一种优化：代数简化

　　int d = E \* 13 + a \* 2;

## 3） 数组边界检查消除

Java语言是一门动态安全的语言。如果有一个数组foo[]，在Java语言中访问数组元素foo[i]的时候系统将会自动进行上下界的范围检查，即检查i必须满足i >=0 && i < foo.length这个条件，否则将抛出一个运行时异常：java.lang.ArrayIndexOutOfBoundsException。

无论如何，为了安全，数组便捷检查肯定是必须做的，但数组边界检查是不是必须在运行期间一次不漏的检查则是可以“商量”的事情。例如：数组下标是一个常量，如 foo[3]，只要在编译期根据数据流分析来确定foo.length的值，并判断下标“3”没有越界，执行的时候就无需判断了

## 4）逃逸分析

逃逸分析的基本行为就是分析对象动态作用域：当一个对象在方法中被定以后，它可能被外部方法所引用，例如作为调用参数传递到其他方法中，称为方法逃逸。甚至还有其可能被外部线程访问到，譬如赋值给类变量或可以在其他线程中访问的实例变量，称为线程逃逸！

# Java对优化方法

## C1 和 C2 编译器

## 分层编译技术

# 

# Node v8引擎优化

## 隐藏类

# Php7对性能优化

为什么PHP7的性能可以提高这么多？

1. JIT

2. Zval的改变

3. 内部类型zend\_string

4. PHP数组的变化（HashTable和Zend Array）

5. 函数调用机制（Function Calling Convention）

6. 通过宏定义和内联函数（inline），让编译器提前完成部分工作

PHP7性能提升原因总结：

1、存储变量的结构体变小，尽量使结构体里成员共用内存空间，减少引用，这样内存占用降低，变量的操作速度得到提升。

2、字符串结构体的改变，字符串信息和数据本身原来是分成两个独立内存块存放，php7尽量将它们存入同一块内存，提升了cpu缓存命中率。

3、数组结构的改变，数组元素和hash映射表在php5中会存入多个内存块，php7尽量将它们分配在同一块内存里，降低了内存占用、提升了cpu缓存命中率。

4、改进了函数的调用机制，通过对参数传递环节的优化，减少一些指令操作，提高了执行效率。

# ref

Java性能优化系列之-JIT即时编译器与Java内存管理机制\_mike\_jun的博客-CSDN博客

为什么V8引擎这么快？-阿里云开发者社

V8带来的JS性能优化 - 云+社区 - 腾讯云区

PHP小白要知道：PHP7 性能为何能翻倍的关键因素是什么 - 云+社区 - 腾讯云