atitit 虚拟机之道vm之道 runtime设计 运行时 .attilax著.docx

[1. Atitit 虚拟机的层次 架构与常见的虚拟机 3](#_Toc9013)

[1.1. Os隔离了硬件的区别 4](#_Toc23007)

[1.2. 4](#_Toc32052)

[1.3. Jvm clr又隔离了os的区别 4](#_Toc20076)

[1.4. Web 隔离了 vm的区别，webkit vm环境。。 4](#_Toc13706)

[1.5. 框架又是一个小的vm环境，ioc容器。。 4](#_Toc2699)

[1.6. 4](#_Toc23654)

[1.7. Dockert容器 4](#_Toc25707)

[1.8. 4](#_Toc17165)

[1.9. 安卓虚拟机。。等，可以集成app 4](#_Toc13508)

[1.10. Game虚拟机。。 4](#_Toc8630)

[2. 4](#_Toc5069)

[3. 4](#_Toc21807)

[4. Vm的未来 4](#_Toc17095)

[4.1. 1.5.1　模块化 1.5.2　混合语言 1.5.3　多核并行 1.5.4　进一步丰富语法 1.5.5　64位虚拟机 5](#_Toc15032)

[5. 自动内存管理机制 5](#_Toc20201)

[6. 执行引擎工作原理：方法调用 25 6](#_Toc9150)

[7. 常量池解析 206 6](#_Toc8374)

[8. 类变量解析 280 8](#_Toc26711)

[9. Java栈帧 344 8](#_Toc775)

[10. 类方法解析 440 9](#_Toc5806)

[11. 执行引擎 513 10](#_Toc18248)

[12. 类的生命周期 602 11](#_Toc28351)

[13. 垃圾收集器与内存分配策略 13](#_Toc6330)

[14. 第4章　虚拟机性能监控与故障处理工具 14](#_Toc5400)

[15. 第6章　类文件结构 15](#_Toc24254)

[16. 虚拟机类加载机制 16](#_Toc31866)

[17. 第8章　虚拟机字节码执行引擎 16](#_Toc20080)

[18. 第四部分　程序编译与代码优化 第10章　早期（编译期）优化 17](#_Toc17795)

[19. 晚期（运行期）优化 17](#_Toc3609)

[20. 第五部分　高效并发 第12章　Java内存模型与线程 18](#_Toc3288)

[21. 第13章　线程安全与锁优化 18](#_Toc8519)

[22. 两种计算模型 ，堆栈机和状态机（基于寄存器的虚拟机 19](#_Toc9059)

[22.1.1. 堆栈机 19](#_Toc19082)

[22.1.2. 状态机 20](#_Toc18837)

[23. 为什么状态机比堆栈机快呢？ 21](#_Toc14370)

[23.1. Stack based vm的指令 范例 23](#_Toc3551)

[24. 参考 23](#_Toc15146)

[25. 附录B　虚拟机字节码指令表 24](#_Toc12214)

[25.1. Mov系列指令 24](#_Toc1277)

[25.2. Push pop 24](#_Toc13271)

[25.3. 24](#_Toc1259)

[25.4. 算数逻辑移位 24](#_Toc31668)

[25.5. 24](#_Toc4221)

[25.6. 跳转指令 24](#_Toc294)

[25.7. Oo指令 24](#_Toc4814)

[26. 附录C　HotSpot虚拟机主要参数表 24](#_Toc8412)

[27. 附录D　对象查询语言（OQL）简介 24](#_Toc9163)

[28. 参考资料 25](#_Toc28964)

[28.1. Atitit.php opcode虚拟机指令集 分类以及详细解释 25](#_Toc11863)

[28.2. 25](#_Toc20470)

[28.3. C:\Users\Administrator\Desktop\vm虚拟机设计之道资料包>dir /b 25](#_Toc17571)

[28.4. Atitit .jvm 虚拟机指令详细解释.docx 25](#_Toc27247)

[28.5. Atitit .jvm 虚拟机指令详细解释.docx.txt 25](#_Toc31807)

[28.6. Atitit runtime设计 运行时 vm设计 虚拟机设计.docx 25](#_Toc20773)

[28.7. Atitit runtime设计 运行时 vm设计 虚拟机设计.docx.E5A681566CA1E2DE28F6B1BB7072448A.20180208225522212.wps 25](#_Toc14248)

[28.8. Atitit vm虚拟机原理与概论book.docx 25](#_Toc2028)

[28.9. atitit 虚拟机之道vm之道.attilax著.docx 25](#_Toc12648)

[28.10. Atitit 虚拟机的层次 架构与常见的虚拟机.docx 25](#_Toc13582)

[28.11. Atitit.java 虚拟机 指令集合.docx 25](#_Toc31995)

[28.12. Atitit.java 虚拟机 指令集合.docx.txt 25](#_Toc29448)

[28.13. Atitit.php opcode虚拟机指令集 分类以及详细解释.docx 26](#_Toc5142)

[28.14. atitit. atitit.基于虚拟机的启动器设计 v2 q37 --java 启动器 java生成exe.docx 26](#_Toc8508)

[28.15. Atitit.基于寄存器的虚拟机设计.docx 26](#_Toc31499)

[28.16. Atitit.简化的基于堆栈的虚拟机指令 参考java jvm虚拟机指令.docx 26](#_Toc13124)

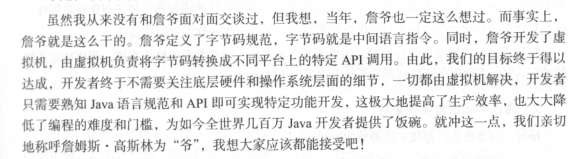
[28.17. Atitit.虚拟机与指令系统的设计.docx 26](#_Toc7515)

[28.18. Atitit。虚拟机指令集 的文本形式语法 二进制形式的规范.docx 26](#_Toc15299)

[28.19. Atitti.java android反编译解决方案-----虚拟机方案.docx 26](#_Toc19375)

[28.20. 《揭秘Java虚拟机：JVM设计原理与实现》(封亚飞)【简介\_书评\_在线阅读】 - 当当图书.mhtml 26](#_Toc5872)

# 为什么需要虚拟机



# Atitit 虚拟机的层次 架构与常见的虚拟机

## Os隔离了硬件的区别

## 

## Jvm clr又隔离了os的区别

## Web 隔离了 vm的区别，webkit vm环境。。

## 框架又是一个小的vm环境，ioc容器。。

## 

## Dockert容器

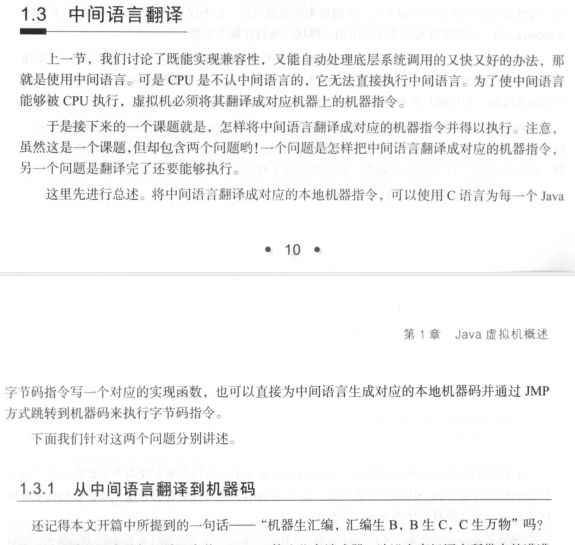
## 

## 安卓虚拟机。。等，可以集成app

## Game虚拟机。。

# 

# 中间语言il



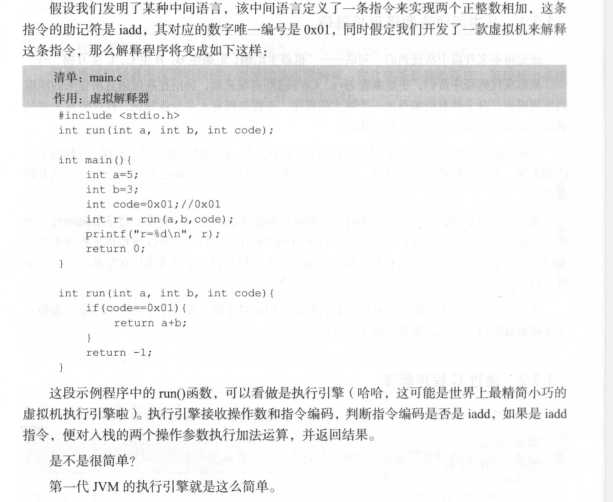
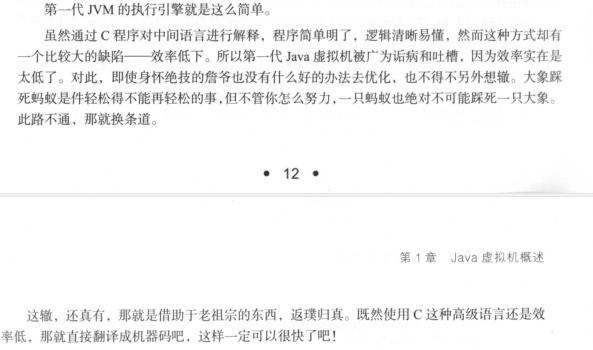
# Vm的未来

## 1.5.1　模块化 1.5.2　混合语言 1.5.3　多核并行 1.5.4　进一步丰富语法 1.5.5　64位虚拟机

# 自动内存管理机制

第2章　Java内存区域与内存溢出异常  
2.1　概述  
2.2　运行时数据区域  
2.2.1　程序计数器  
2.2.2　Java虚拟机栈  
2.2.3　本地方法栈  
2.2.4　Java堆  
2.2.5　方法区  
2.2.6　运行时常量池  
2.2.7　直接内存

# 执行引擎工作原理：方法调用 25

1. 
2. 
3. 2.1 方法调用 26
4. 2.1.1 真实的机器调用 26
5. 2.1.2 C语言函数调用 41
6. 2.2 JVM的函数调用机制 47
7. 2.3 函数指针 53
8. 2.4 CallStub函数指针定义 60
9. 2.5 \_call\_stub\_entry例程 72
10. 2.6 本章总结 115

# 常量池解析 206

5.1 常量池内存分配 208

5.1.1 常量池内存分配总体链路 209

5.1.2 内存分配 215

5.1.3 初始化内存 223

5.2 oop-klass模型 224

5.2.1 两模型三维度 225

5.2.2 体系总览 227

5.2.3 oop体系 229

5.2.4 klass体系 231

5.2.5 handle体系 234

5.2.6 oop、klass、handle的相互转换 239

5.3 常量池klass模型（1） 244

5.3.1 klassKlass实例构建总链路 246

5.3.2 为klassOop申请内存 249

5.3.3 klassOop内存清零 253

5.3.4 初始化mark 253

5.3.5 初始化klassOop.\_metadata 258

5.3.6 初始化klass 259

5.3.7 自指 260

5.4 常量池klass模型（2） 261

5.4.1 constantPoolKlass模型构建 261

5.4.2 constantPoolOop与klass 264

5.4.3 klassKlass终结符 267

5.5 常量池解析 267

5.5.1 constantPoolOop域初始化 268

5.5.2 初始化tag 269

5.5.3 解析常量池元素 271

5.6 本章总结 279

# 类变量解析 280

6.1 类变量解析 281

6.2 偏移量 285

6.2.1 静态变量偏移量 285

6.2.2 非静态变量偏移量 287

6.2.3 Java字段内存分配总结 312

6.3 从源码看字段继承 319

6.3.1 字段重排与补白 319

6.3.2 private字段可被继承吗 325

6.3.3 使用HSDB验证字段分配与继承 329

6.3.4 引用类型变量内存分配 338

6.4 本章总结 342

# Java栈帧 344

7.1 entry\_point例程生成 345

7.2 局部变量表创建 352

7.2.1 constMethod的内存布局 352

7.2.2 局部变量表空间计算 356

7.2.3 初始化局部变量区 359

7.3 堆栈与栈帧 368

7.3.1 栈帧是什么 368

7.3.2 硬件对堆栈的支持 387

7.3.3 栈帧开辟与回收 390

7.3.4 堆栈大小与多线程 391

7.4 JVM的栈帧 396

7.4.1 JVM栈帧与大小确定 396

7.4.2 栈帧创建 399

7.4.3 局部变量表 421

7.5 栈帧深度与slot复用 433

7.6 最大操作数栈与操作数栈复用 436

7.7 本章总结 439

# 类方法解析 440

8.1 方法签名解析与校验 445

8.2 方法属性解析 447

8.2.1 code属性解析 447

8.2.2 LVT＆LVTT 449

8.3 创建methodOop 455

8.4 Java方法属性复制 459

8.5 ＜clinit＞与＜init＞ 461

8.6 查看运行时字节码指令 482

8.7 vtable 489

8.7.1 多态 489

8.7.2 C 中的多态与vtable 491

8.7.3 Java中的多态实现机制 493

8.7.4 vtable与invokevirtual指令 500

8.7.5 HSDB查看运行时vtable 502

8.7.6 miranda方法 505

8.7.7 vtable特点总结 508

8.7.8 vtable机制逻辑验证 509

8.8 本章总结 511

# 执行引擎 513

9.1 执行引擎概述 514

9.2 取指 516

9.2.1 指令长度 519

9.2.2 JVM的两级取指机制 527

9.2.3 取指指令放在哪 532

9.2.4 程序计数器在哪里 534

9.3 译码 535

9.3.1 模板表 535

9.3.2 汇编器 540

9.3.3 汇编 549

9.4 栈顶缓存 558

9.5 栈式指令集 565

9.6 操作数栈在哪里 576

9.7 栈帧重叠 581

9.8 entry\_point例程机器指令 586

9.9 执行引擎实战 588

9.9.1 一个简单的例子 588

9.9.2 字节码运行过程分析 590

9.10 字节码指令实现 597

9.10.1 iconst\_3 598

9.10.2 istore\_0 599

9.10.3 iadd 600

9.11 本章总结 601

# 类的生命周期 602

10.1 类的生命周期概述 602

10.2 类加载 605

10.2.1 类加载——镜像类与静态字段 611

10.2.2 Java主类加载机制 617

10.2.3 类加载器的加载机制 622

10.2.4 反射加载机制 623

10.2.5 import与new指令 624

10.3 类的初始化 625

10.4 类加载器 628

10.4.1 类加载器的定义 628

10.4.2 系统类加载器与扩展类加载器创建 634

10.4.3 双亲委派机制与破坏 636

10.4.4 预加载 638

10.4.5 引导类加载 640

10.4.6 加载、链接与延迟加载 641

10.4.7 父加载器 645

10.4.8 加载器与类型转换 648

10.5 类实例分配 649

10.5.1 栈上分配与逃逸分析 652

10.5.2 TLAB 655

10.5.3 指针碰撞与eden区分配 657

10.5.4 清零 658

10.5.5 偏向锁 658

10.5.6 压栈与取指 659

10.6 本章总结 661

# 垃圾收集器与内存分配策略

1. 3.1　概述  
   3.2　对象已死吗  
   3.2.1　引用计数算法  
   3.2.2　可达性分析算法  
   3.2.3　再谈引用  
   3.2.4　生存还是死亡  
   3.2.5　回收方法区  
   3.3　垃圾收集算法  
   3.3.1　标记-清除算法  
   3.3.2　复制算法  
   3.3.3　标记-整理算法  
   3.3.4　分代收集算法  
   3.4　HotSpot的算法实现  
   3.4.1　枚举根节点  
   3.4.2　安全点  
   3.4.3　安全区域  
   3.5　垃圾收集器  
   3.5.1　Serial收集器  
   3.5.2　ParNew收集器  
   3.5.3　Parallel Scavenge收集器  
   3.5.4　Serial Old收集器  
   3.5.5　Parallel Old收集器  
   3.5.6　CMS收集器  
   3.5.7　G1收集器  
   3.5.8　理解GC日志  
   3.5.9　垃圾收集器参数总结  
   3.6　内存分配与回收策略  
   3.6.1　对象优先在Eden分配  
   3.6.2　大对象直接进入老年代  
   3.6.3　长期存活的对象将进入老年代  
   3.6.4　动态对象年龄判定  
   3.6.5　空间分配担保  
   3.7　本章小结

# 第4章　虚拟机性能监控与故障处理工具

4.1　概述  
4.2　JDK的命令行工具  
4.2.1　jps：虚拟机进程状况工具  
4.2.2　jstat：虚拟机统计信息监视工具  
4.2.3　jinfo：Java配置信息工具  
4.2.4　jmap：Java内存映像工具  
4.2.5　jhat：虚拟机堆转储快照分析工具  
4.2.6　jstack：Java堆栈跟踪工具  
4.2.7　HSDIS：JIT生成代码反汇编  
4.3　JDK的可视化工具  
4.3.1　JConsole：Java监视与管理控制台  
4.3.2　VisualVM：多合一故障处理工具  
4.4　本章小结

# 第6章　类文件结构

6.1　概述  
6.2　无关性的基石  
6.3　Class类文件的结构  
6.3.1　魔数与Class文件的版本  
6.3.2　常量池  
6.3.3　访问标志  
6.3.4　类索引、父类索引与接口索引集合  
6.3.5　字段表集合  
6.3.6　方法表集合  
6.3.7　属性表集合  
6.4　字节码指令简介  
6.4.1　字节码与数据类型  
6.4.2　加载和存储指令  
6.4.3　运算指令  
6.4.4　类型转换指令  
6.4.5　对象创建与访问指令  
6.4.6　操作数栈管理指令  
6.4.7　控制转移指令  
6.4.8　方法调用和返回指令  
6.4.9　异常处理指令  
6.4.10　同步指令  
6.5　公有设计和私有实现  
6.6　Class文件结构的发展  
6.7　本章小结

# 虚拟机类加载机制

1. 7.1　概述  
   7.2　类加载的时机  
   7.3　类加载的过程  
   7.3.1　加载  
   7.3.2　验证  
   7.3.3　准备  
   7.3.4　解析  
   7.3.5　初始化  
   7.4　类加载器  
   7.4.1　类与类加载器  
   7.4.2　双亲委派模型  
   7.4.3　破坏双亲委派模型  
   7.5　本章小结

# 第8章　虚拟机字节码执行引擎

1. 8.1　概述  
   8.2　运行时栈帧结构  
   8.2.1　局部变量表  
   8.2.2　操作数栈  
   8.2.3　动态连接  
   8.2.4　方法返回地址  
   8.2.5　附加信息  
   8.3　方法调用  
   8.3.1　解析  
   8.3.2　分派  
   8.3.3　动态类型语言支持  
   8.4　基于栈的字节码解释执行引擎  
   8.4.1　解释执行  
   8.4.2　基于栈的指令集与基于寄存器的指令集  
   8.4.3　基于栈的解释器执行过程  
   8.5　本章小结

# 第四部分　程序编译与代码优化 第10章　早期（编译期）优化

10.1　概述  
10.2　Javac编译器  
10.2.1　Javac的源码与调试  
10.2.2　解析与填充符号表  
10.2.3　注解处理器  
10.2.4　语义分析与字节码生成  
10.3　Java语法糖的味道  
10.3.1　泛型与类型擦除  
10.3.2　自动装箱、拆箱与遍历循环  
10.3.3　条件编译  
10.4　实战：插入式注解处理器  
10.4.1　实战目标  
10.4.2　代码实现  
10.4.3　运行与测试  
10.4.4　其他应用案例  
10.5　本章小结

# 晚期（运行期）优化

1. 11.1　概述  
   11.2　HotSpot虚拟机内的即时编译器  
   11.2.1　解释器与编译器  
   11.2.2　编译对象与触发条件  
   11.2.3　编译过程  
   11.2.4　查看及分析即时编译结果  
   11.3　编译优化技术  
   11.3.1　优化技术概览  
   11.3.2　公共子表达式消除  
   11.3.3　数组边界检查消除  
   11.3.4　方法内联  
   11.3.5　逃逸分析  
   11.4　Java与CC++的编译器对比  
   11.5　本章小结

# 第五部分　高效并发 第12章　Java内存模型与线程

1. 12.1　概述  
   12.2　硬件的效率与一致性  
   12.3　Java内存模型  
   12.3.1　主内存与工作内存  
   12.3.2　内存间交互操作  
   12.3.3　对于volatile型变量的特殊规则  
   12.3.4　对于long和double型变量的特殊规则  
   12.3.5　原子性、可见性与有序性  
   12.3.6　先行发生原则  
   12.4　Java与线程  
   12.4.1　线程的实现  
   12.4.2　Java线程调度  
   12.4.3　状态转换  
   12.5　本章小结

# 第13章　线程安全与锁优化

13.1　概述  
13.2　线程安全  
13.2.1　Java语言中的线程安全  
13.2.2　线程安全的实现方法  
13.3　锁优化  
13.3.1　自旋锁与自适应自旋  
13.3.2　锁消除  
13.3.3　锁粗化  
13.3.4　轻量级锁  
13.3.5　偏向锁  
13.4　本章小结

# **两种计算模型** ，堆栈机和状态机（基于寄存器的虚拟机

有了上面的基础只是，我们就知道，堆栈机和状态机不过是两种不同的图灵完整的计算模型而已。

### 堆栈机

所谓堆栈机，就是计算机的状态是存在于堆栈之中，通过对堆栈中的元素进行运算和调整，来实现计算功能的计算机。   
例如，要进行一个1+2的加法运算，那么就：

| 操作 | 堆栈状态 |
| --- | --- |
| 初始状态 |  |
| 将1压入栈中 | 1 |
| 将2压入栈中 | 1, 2 |
| 调用加法运算 | 3 |

其典型代表就是Python的虚拟机，代码如下：

**push** 1**push** 2

Add

作者:: 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

### 状态机

状态机的基本原理就在于，它可以有有限种状态，指令能够让它在不同的状态之间进行转换。   
听起来很抽象？   
但其实，我们大部分写代码的时候都是对状态机编程，比如c代码：

**int** a = 1;**int** b = 2;**int** c = a + b;

其实这个状态机有2^96种状态（假设int是32位的），因为变量a有2^32种状态（-2147483648~2147483647），b、c亦然。

|  |  |  |  |
| --- | --- | --- | --- |
| 操作 | a的状态 | b的状态 | c的状态 |
| 初始状态 | 0 | 0 | 0 |
| a=1 | 1 | 0 | 0 |
| b=2 | 1 | 2 | 0 |
| c=a+b | 1 | 2 | 3 |

典型代表就是Lua的虚拟机，应的代码就是：

loadk 0 1

loadk 1 2

add 2 0 1

意思就是：

register[0] = 1register[1] = 2register[2] = register[0] + register[1]

# 为什么状态机比堆栈机快呢？

既然他们是图灵等价的，那大家一定会很疑惑，为何状态机比堆栈机快呢？   
那么我们要深入到虚拟机内部，看看这些指令都是怎么实现的。   
为了便于大家理解，我所有的代码都不是vm中的实际代码，而是伪代码。   
首先来看看堆栈机：

**switch**(op) {**case** PUSH:

STACK\_ADJ(1);

STACK\_TOP = oprand;

**break**;**case** ADD:

STACK\_SECOND = STACK\_TOP + STACK\_SECOND;

STACK\_ADJ(-1);

**break**;

}

我们可以看到，大部分情况下，执行一条指令，除了原始的赋值操作外，还需要调整堆栈的栈顶指针（那些STACK\_ADJ宏定义），再看看状态机的实现：

**switch**(op) {**case** LOADK:

REGISTER[oprand0] = oprand1;

**break**;**case** ADD:

REGISTER[oprand0] = REGISTER[oprand1] + REGISTER[oprand2];

**break**;

}

大家可以看到，在执行大部分指令时，状态机虚拟机会比堆栈机要少一次调整堆栈的操作，这对性能会有很明显的影响。   
当然这也主要适用于Interpreting的情况，在Jit的情况下，会有很多深度优化，从而使得堆栈机的性能也能和状态机一样。

## Stack based vm的指令 范例

一般都是在当前stack中获取和保存操作数的。比如一个简单的加法赋值运算:a=b+c，对于stack based vm，一般会被转化成如下的指令:

**[plain]** [view plain](http://blog.csdn.net/yuanlin2008/article/details/8423951" \o "view plain) [copy](http://blog.csdn.net/yuanlin2008/article/details/8423951" \o "copy" \t "http://blog.csdn.net/yuanlin2008/article/details/_blank) [print](http://blog.csdn.net/yuanlin2008/article/details/8423951" \o "print" \t "http://blog.csdn.net/yuanlin2008/article/details/_blank)[?](http://blog.csdn.net/yuanlin2008/article/details/8423951" \o "?)

1. push b; // 将变量b的值压入stack
2. push c; // 将变量c的值压入stack
3. add;    // 将stack顶部的两个值弹出后相加，将结果压入stack
4. mov a;  // 将stack顶部结果放到a中

# 参考

高效动态语言虚拟机的设计（二） – 堆栈机vs状态机 - 推酷.htm

# 附录B　虚拟机字节码指令表

## Mov系列指令

## Push pop

## 

## 算数逻辑移位

## 

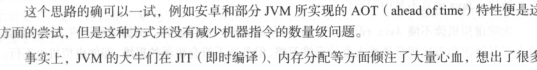
## 跳转指令

## Oo指令

# 附录C　HotSpot虚拟机主要参数表

# 附录D　对象查询语言（OQL）简介

# aot



# 参考资料

## Atitit.php opcode虚拟机指令集 分类以及详细解释

## 

## C:\Users\Administrator\Desktop\vm虚拟机设计之道资料包>dir /b

## Atitit .jvm 虚拟机指令详细解释.docx

## Atitit .jvm 虚拟机指令详细解释.docx.txt

## Atitit runtime设计 运行时 vm设计 虚拟机设计.docx

## Atitit runtime设计 运行时 vm设计 虚拟机设计.docx.E5A681566CA1E2DE28F6B1BB7072448A.20180208225522212.wps

## Atitit vm虚拟机原理与概论book.docx

## atitit 虚拟机之道vm之道.attilax著.docx

## Atitit 虚拟机的层次 架构与常见的虚拟机.docx

## Atitit.java 虚拟机 指令集合.docx

## Atitit.java 虚拟机 指令集合.docx.txt

## Atitit.php opcode虚拟机指令集 分类以及详细解释.docx

## atitit. atitit.基于虚拟机的启动器设计 v2 q37 --java 启动器 java生成exe.docx

## Atitit.基于寄存器的虚拟机设计.docx

## Atitit.简化的基于堆栈的虚拟机指令 参考java jvm虚拟机指令.docx

## Atitit.虚拟机与指令系统的设计.docx

## Atitit。虚拟机指令集 的文本形式语法 二进制形式的规范.docx

## Atitti.java android反编译解决方案-----虚拟机方案.docx

## 《揭秘Java虚拟机：JVM设计原理与实现》(封亚飞)【简介\_书评\_在线阅读】 - 当当图书.mhtml