动态翻译系统（JIT）架构概述

邱吉

qiuji@iscas.ac.cn

2023年8月19日

提纲

- 动态翻译系统的概念和分类
- 动态二进制翻译系统概述
- 动态二进制优化系统概述
- 高级语言虚拟机系统概述
- 总结

提纲

- 动态翻译系统的概念和分类
- 动态二进制翻译系统概述
- 动态二进制优化系统概述
- 高级语言虚拟机系统概述
- 总结

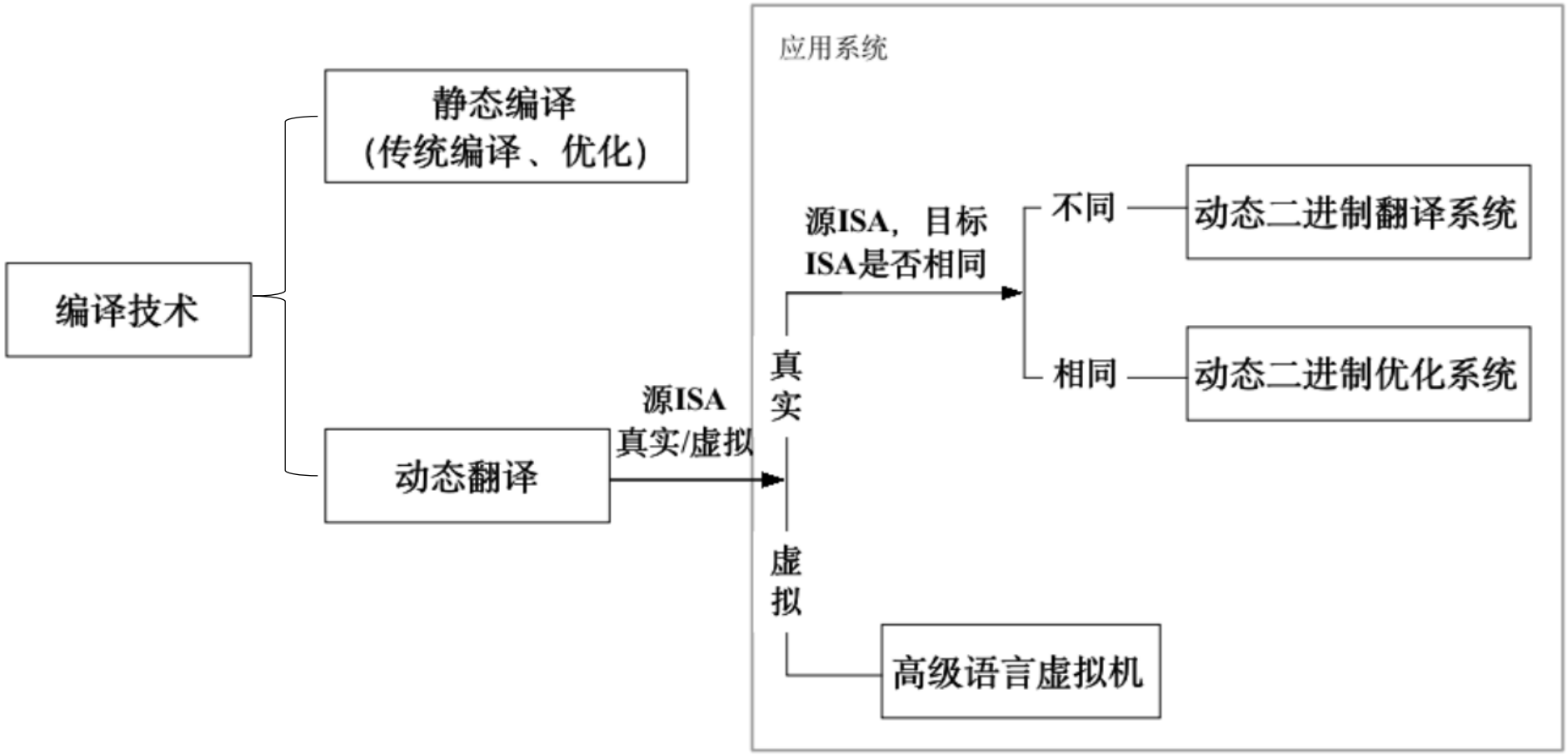
动态翻译系统的概念和分类

- 动态翻译技术是一种把**源指令集体系结构代码**翻译成**目标指令集体系结构代码**的即时编译技术
- 使用动态翻译技术，源指令代码被翻译后可以在**不兼容(或者不相同)**的目标体系结构上直接运行
- 为了提升翻译性能，动态翻译技术实施时，动态翻译系统除了对指令集进行动态翻译，还会**获取程序的动态信息**并进行代码的**优化**
- 一般来说，动态翻译技术的应用可以分为三类，包括
  - 动态二进制翻译系统
  - 动态二进制优化系统
  - 高级语言虚拟机

与传统编译技术的区别

- 编译对象不同
- 工作时机不同

	源对象	目标对象	工作时机
传统编译	高级语言源程序	面向某种体系结构的目标二进制代码	在程序运行前就完成了所有的编译和优化工作
动态翻译	高级语言源程序、某种体系结构的目标代码（实际硬件或虚拟机ISA）	另外一种体系结构的目标二进制代码	在程序运行的同时展开编译和优化工作


应用系统
静态编译
(传统编译、优化)
编译技术
动态翻译
源ISA
真实/虚拟
真实
虚拟
源ISA，目标
ISA是否相同
不同
相同
动态二进制翻译系统
动态二进制优化系统
高级语言虚拟机

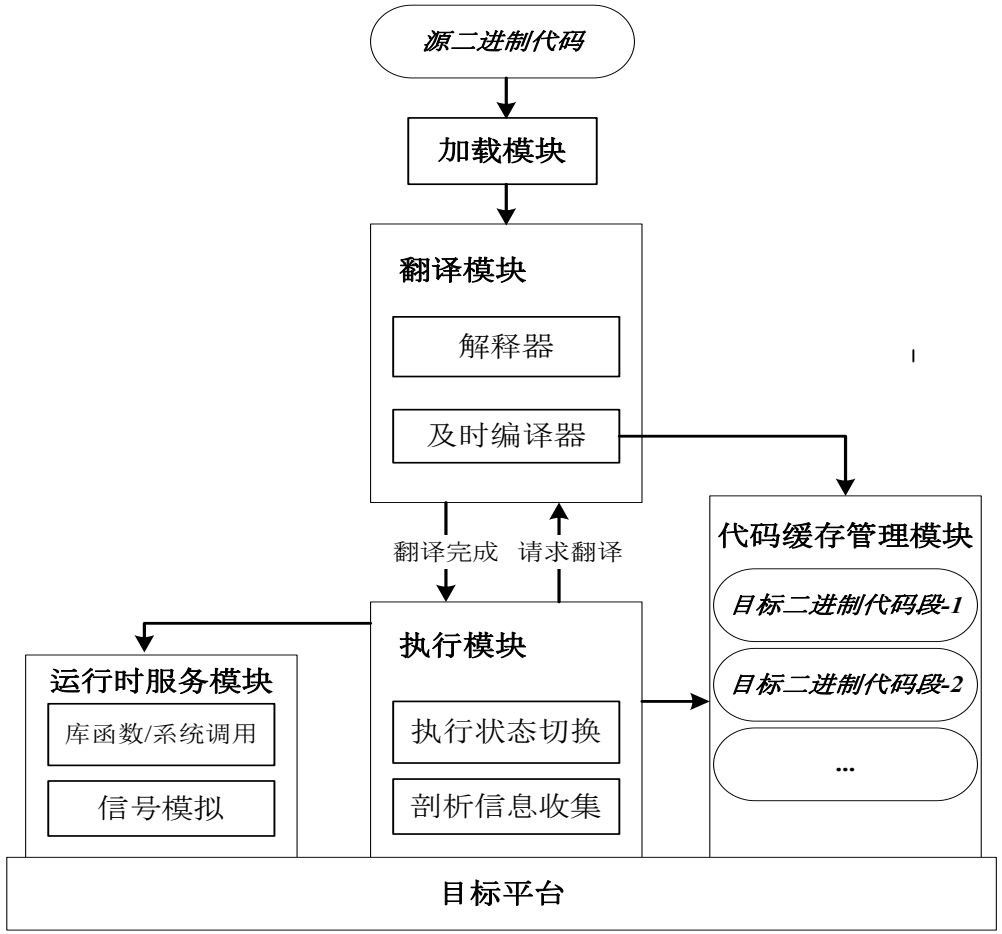
提纲

- 动态翻译系统的概念和分类
- 动态二进制翻译系统概述
- 动态二进制优化系统概述
- 高级语言虚拟机系统概述
- 总结

动态二进制翻译系统的特点和应用领域

- 特点：
  - 源指令集体系结构和目标指令集体系结构二者属于不同的真实的处理器平台
  - 在二进制程序运行时对执行到的片断进行翻译，从而能够保证兼容性
- 应用领域：
  - 解决老旧二进制代码在新处理器上的运行问题，促进CPU ISA创新
  - 改善处理器性能、降低功耗和设计复杂度
  - 辅助和加速体系结构的研究和设计、提早软件开发

动态二进制翻译系统框架



动态二进制翻译系统主要模块解析

- 加载模块
  - 解析源二进制代码，定位并加载其代码和数据等内容
- 翻译模块
  - 把源二进制代码翻译成目标二进制代码
- 执行模块
  - 在动态二进制翻译系统自身的执行和生成的目标二进制代码执行之间的切换工作以及剖析信息的收集工作
- 代码缓存管理模块
  - 对有限大小的代码缓存进行管理，包括代码块的查找、替换和重新布局等
- 运行时服务模块
  - 负责对运行环境进行仿真，例如，对源二进制的库函数调用、系统调用和信号等进行处理和转换

动态二进制翻译系统主要模块解析

- 加载模块
  - 解析源二进制代码，定位并加载其代码和数据等内容
- 翻译模块
  - 把源二进制代码翻译成目标二进制代码
- 执行模块
  - 在动态二进制翻译系统自身的执行和生成的目标二进制代码执行之间的切换工作以及剖析信息的收集工作
- 代码缓存管理模块
  - 对有限大小的代码缓存进行管理，包括代码块的查找、替换和重新布局等
- 运行时服务模块
  - 负责对运行环境进行仿真，例如，对源二进制的库函数调用、系统调用和信号等进行处理和转换

动态二进制翻译系统的翻译模块

- 翻译模块
  - 解释器： 翻译的初始阶段采用解释器可以减小启动开销，并剖析和计数程序热点区域（function or trace)
  - 即时编译器：
    - 当热点执行次数达到某个阈值时，翻译引擎就开始对源二进制代码翻译，生成并保存目标二进制代码
    - 在分阶段的动态二进制翻译系统中，有的即时编译器会在初始生成的目标二进制中插入剖析代码，并在运行在收集优化信息。当出现热点代码区域时，即时编译器将利用收集到的优化信息，再次被翻译成更高质量的目标二进制代码

动态二进制翻译系统的代表系统

- QEMU
  - A generic and open source machine emulator and virtualize
  - https://www.qemu.org/
  - full system emulation/user mode/virtualization
  - multi-source and multi-target
- CMS (Code Morphing Software)
  - Transmeta crusoe: 4 width VLIW, in-order
  - 软硬件协同设计的整体方案，模拟IA-32指令集下的整个系统和应用

动态二进制翻译系统的技术难点-1

- 降低运行开销：剖析和翻译

开销分类	说明	开销来源
采样式剖析	未经修改的程序在固定或随机的时间间隔内被中断执行，同时捕获与程序相关的时间实例，从而在获得足够过的采样后，形成对程序行为的统计描述，其	开销来源于中断、处理和恢复的开销。
插桩式剖析	通常针对特定的与程序相关的事件，并且对剖析事件的所有实例进行计数 常见的软件插桩会产生较大开销	在程序内置入插桩的开销和进行实际收集时的开销
翻译开销		分析和翻译过程

动态二进制翻译系统的技术难点-2

- 提升代码质量：优化

按照优化范围分类	主要的优化方法/算法
代码块（翻译单元）内部优化	常量传播、复写传播、循环不变式外提、冗余分值消除、常量折叠、代码下沉、强度削弱、死代码消除等传统编译器优化，这部分优化的共同特点是无需程序的高层语义信息，能在代码序列的一次遍历中完成，代价很小
代码块之间优化	代码重排序、超块的形成、函数内联等，这类优化能够显著减少分支和跳转指令，同时也提高了代码的局部性，被证明能给动态二进制翻译系统带来极大的性能提升

动态二进制翻译系统的技术难点-3

- 提高代码缓存管理效率
  - 动态二进制翻译系统中保存翻译后代码块的一块软件实现的缓冲区，与传统的硬件指令cache相比，它被缓存的代码块没有固定大小；由于代码块间被跳转链接到一起，因此代码块间存在位置的相互依赖
  - I/D 一致性问题维护：
  - 尽量降低代码缓存在硬件I-cache上的失效（提升局部性）
  - 提升源PC到目标PC的双向映射速度
  - 替换策略：命中率/替换算法复杂度/替换后代码的重链接开销

动态二进制翻译系统的技术难点-4

- 解决体系结构之间存在的差异

寄存器类型和数量的差异	1. Stack , Accumulator, and Register-set Architecture 2. GPR/FPR/VectorR/FLAGS/Control Regs 3. 数量
数据表示的差异	大小端的差异 舍入模式差异 NaN generation and propagation
I/O地址映射的差异	I/O、内存统一编址/独立编址
特权指令的翻译	

动态二进制翻译系统的技术难点-5

- 操作系统差异
  - 应用级二进制翻译：需要精确转换系统调用
  - 当源OS和目标OS相同：需要处理ABI不同的情况
  - 当源OS和目标OS不同：更为复杂

动态二进制翻译系统的技术难点-6

- 操作系统差异保证精确例外的优化调度
  - 在任意时刻，如果应用程序产生了例外，该系统必须能够产生一个与在源体系结构机器上完全一直的程序状态。精确例外要求在一条指令发生例外时，该指令之前的所有指令对机器状态的修改均已结束，而该指令之后的任何指令都还未执行。
  - 在动态二进制翻译系统中，由于每一条源指令都被翻译成了若干条目标指令。因此，动态二进制系统必须在某条目标指令发生例外时，把这个例外状态对应到源指令“本来应该”具备的精确例外状态上去。如果被翻译后的目标指令集代码已经被优化过，上述精确异常的支持就更为困难。因为优化可能带来指令的重排序或者消除，从而使发生例外的目标指令无法再简单的对应到源指令上去。

动态二进制翻译系统的技术难点-7

- 处理自修改代码
  - 一些早期的应用程序往往利用运行时对代码的修改来节省内存，或者对程序对自身加密。在这种情况下，代码先被作为数据写入内存，然后再从内存读回继续执行。在动态二进制翻译系统中，被修改的代码必须被重新编译，以保证以最新的代码同步
  - 对采用显式的方式处理数据、代码的一致性的源体系结构类型，动态二进制翻译系统只需对这类指令进行跟踪，并对被更新的源指令重新翻译即可。对于那些由硬件自动保证数据、指令缓存一致性的源体系结构而言，翻译器必须在执行时，对源指令代码的修改进行探测，并即时的对被修改的代码重新翻译

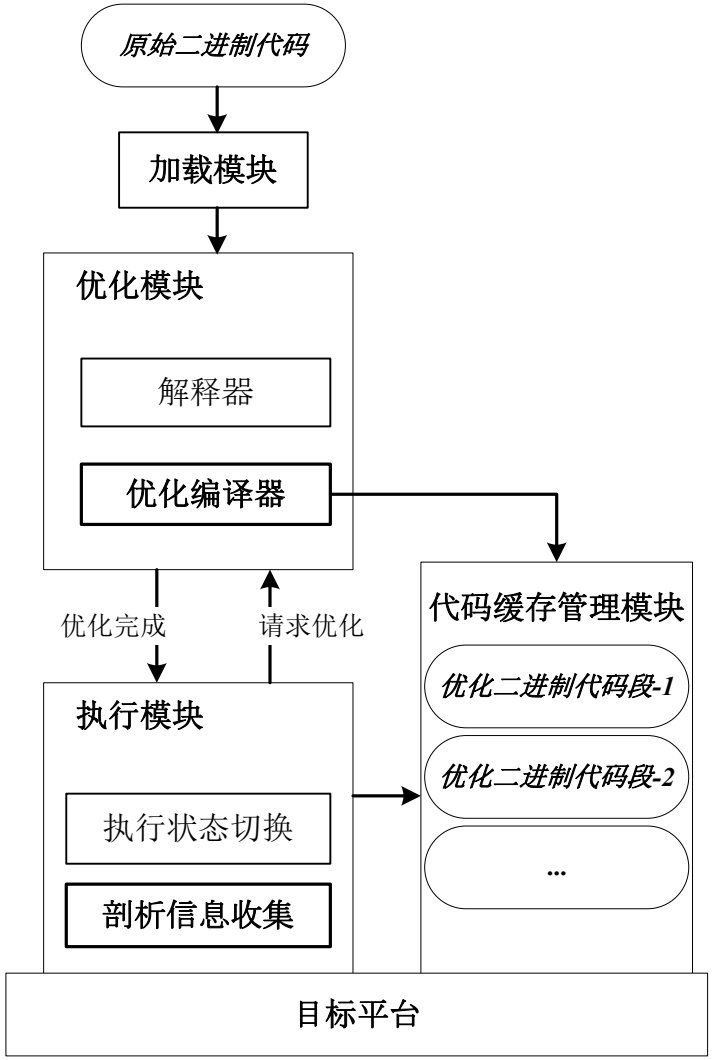
提纲

- 动态翻译系统的概念和分类
- 动态二进制翻译系统概述
- 动态二进制优化系统概述
- 高级语言虚拟机系统概述
- 总结

动态二进制优化系统的特点

- 特点：
  - 源指令集体系结构和目标指令集体系结构相同！
  - 聚焦优化：在运行时通过搜集程序信息来实施优化
- 代表系统：
  - Dynamo 和 DynamoRIO
    - 因其技术创新和优秀的文档是动态二进制优化系统的重要代表和参考系统
  - Mojo
  - Adore

动态二进制优化系统框架



动态二进制优化系统的技术难点

- 没有多余的寄存器可以使用
  - 分析热代码块，找出不活跃的寄存器来使用
  - 溢出到内存
  - 超级代码块的寄存器重新分配
- 不能做负优化
  - 要提供衡量优化收益和取消优化的机制
  - 在性能收益和剖析开销、和优化开销之间精确取舍

提纲

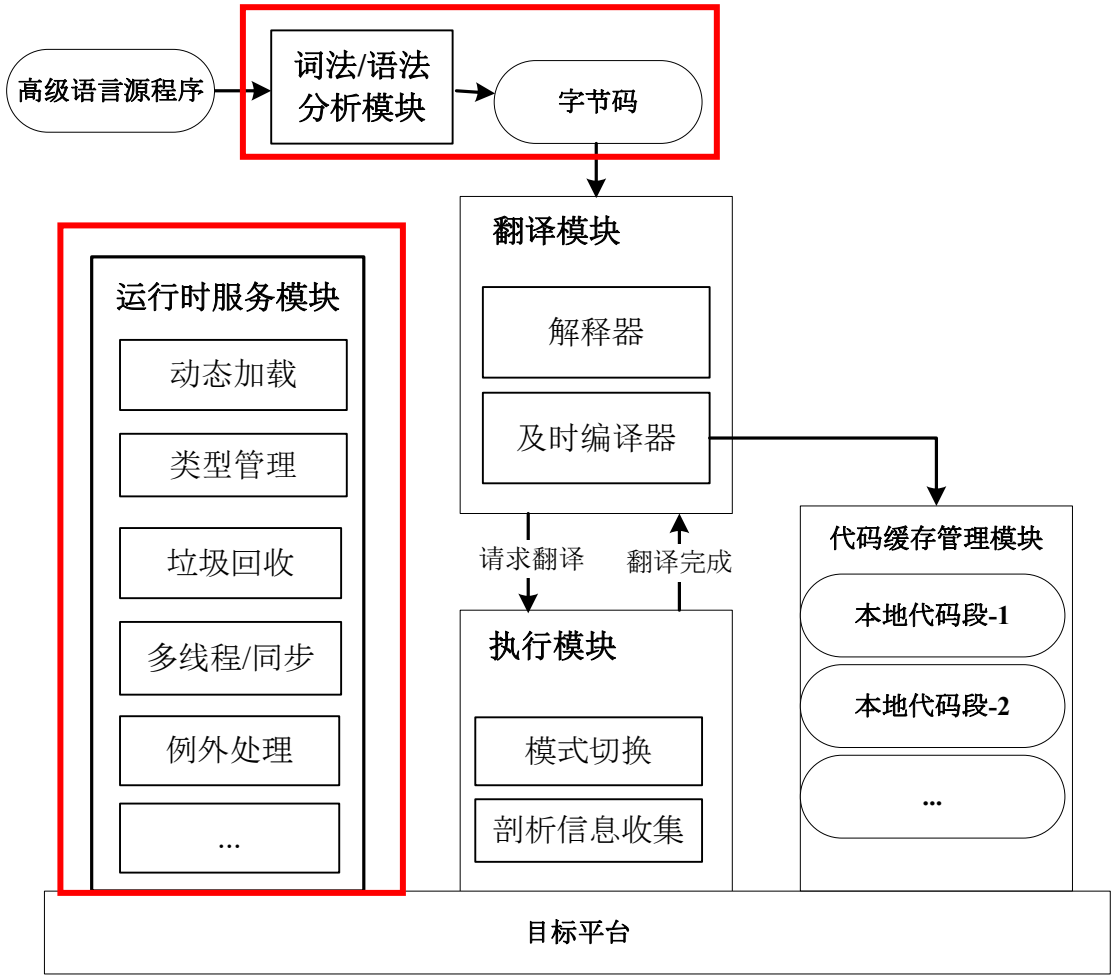
- 动态翻译系统的概念和分类
- 动态二进制翻译系统概述
- 动态二进制优化系统概述
- 高级语言虚拟机系统概述
- 总结

高级语言虚拟机的特点和应用领域

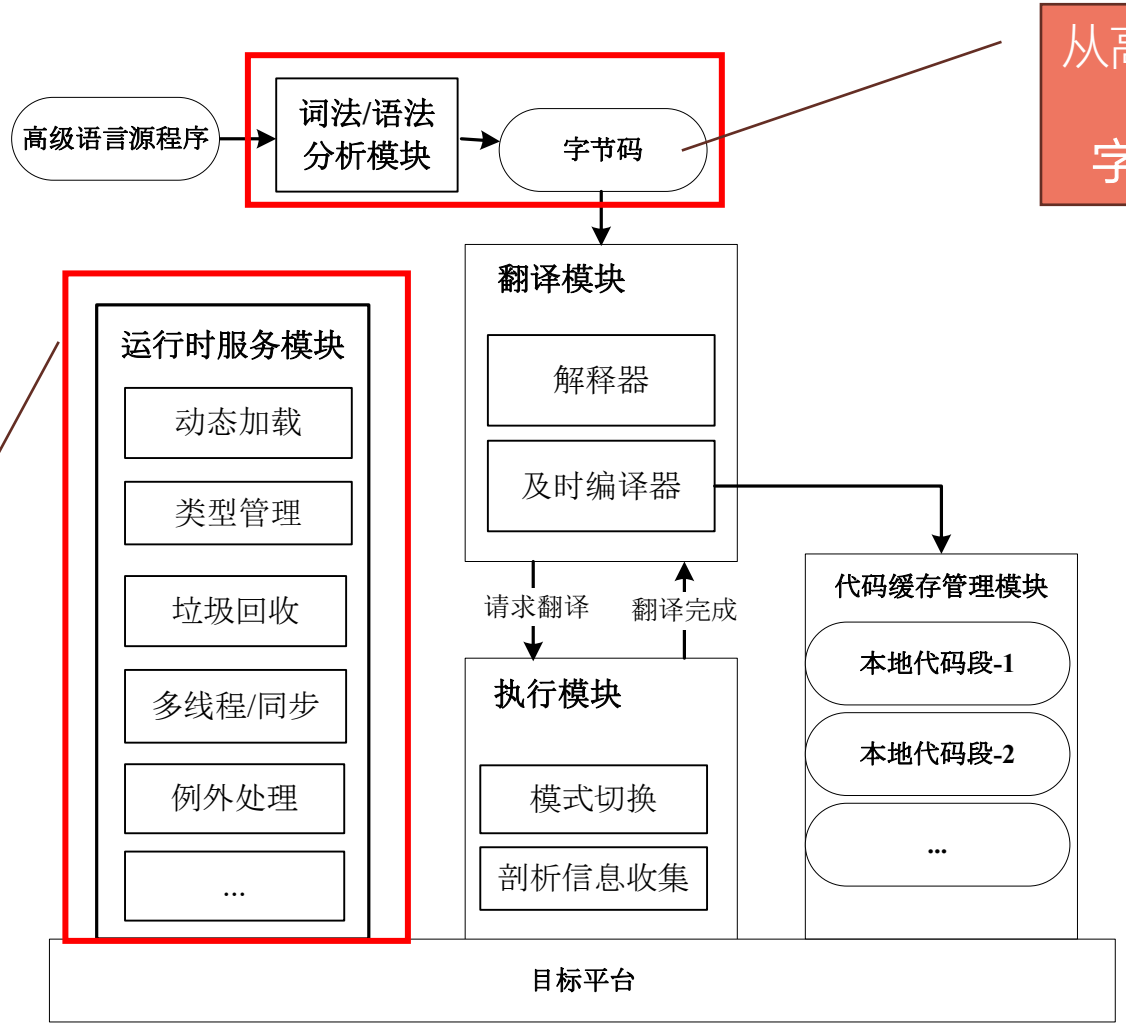
体系结构研究的热点方向

- 特点：
  - 伴随着高级语言和应用程序开发环境一起设计
  - 源指令集体系结构是一种虚拟体系结构
  - 目标指令集体系结构是硬件
  - 关注可移植性和高级语言特性支持（安全特性、自动内存管理、线程管理）
- 代表系统：
  - Java VM
  - JavaScript Engine
  - Microsoft Common Language Infrastructure(CLI)
  - PHP、Python、Ruby、Lua…

高级语言虚拟机系统框架

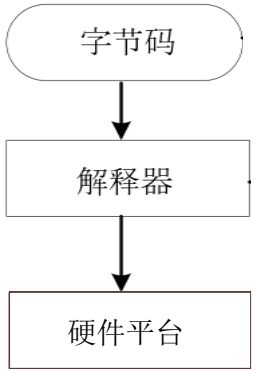


高级语言虚拟机系统框架

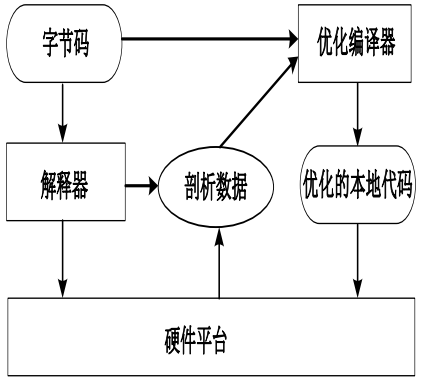


高级语言虚拟机系统主要模块解析-1

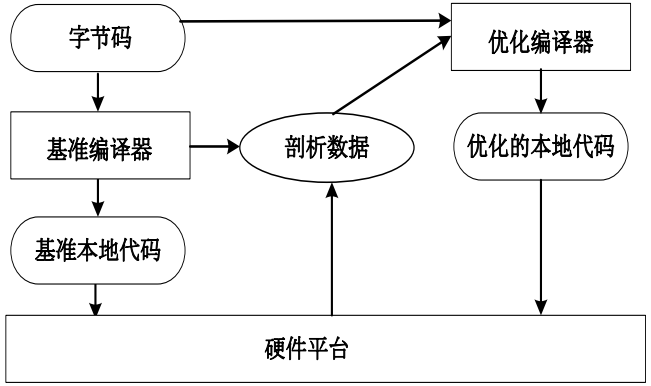
- 词法/语法分析模块
  - 产生字节码, 为了便于生成不同寄存器数量的目标二进制代码，通常设计成基于栈的指令集
- 翻译和执行模块的三种结构



a. 资源紧缺型



b. 资源相对充裕，有一定的性能需求



c. 资源充裕，有极致性能需求

高级语言虚拟机系统主要模块解析-2

- 运行时服务模块
  - 类型管理
    - 如JavaScript语言中，变量无需申明即可使用，在使用中可类型也可以任意变化
    - 属性的管理、查找和优化
  - 垃圾回收
    - 自动回收程序不再被使用的数据对象所占据的内存空间，使程序员无需考虑显式回收内存，从而在很大程度上避免了潜在内存错误的发生
    - 对象的引用计数
    - 内存分配和清扫、复制
  - 多线程和同步
  - 动态加载
  - 反射
  - 例外处理

高级语言虚拟机系统技术特点

- 输入信息更丰富
  - 包含虚拟指令和元数据
    - 元数据可以在程序加载和运行时用于类型安全验证，可以使不同高级语言产生的代码可以相互操作，最重要的作用是可以使得虚拟机看到程序和数据的高层语义信息，以便进行更为彻底的控制流和数据流分析来指导优化。
  - 运行时服务带来了额外的机遇和挑战
    - 额外的运行时服务，在实现高级语言特性的同时给高级语言虚拟机带来了额外的运行时开销。因此，在高级语言虚拟机技术领域内，对这些运行时服务的优化也是高级语言虚拟机性能优化必须考虑的问题（成为研究和工程的热点和难点）

总结

- 综述了动态翻译系统的概念、分类、系统框架和技术难点
- 预告：LLVM JIT的解析