

编译原理

1. 导引

谭添

南京大学计算机系

2024 春季



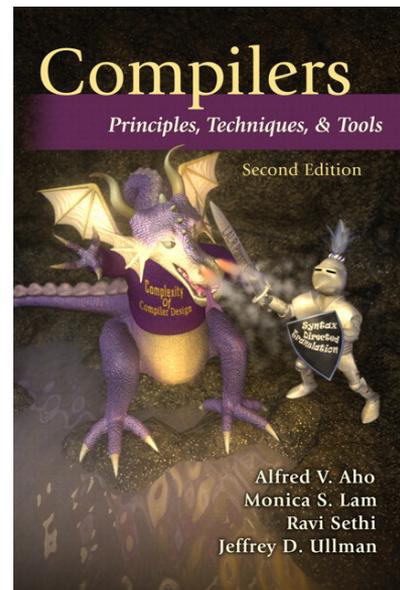
课程概要

- 教材

- 理论：《编译原理》(龙书)，第二版
- 实验：《编译原理实践与指导教程》

- 安排

- 主讲教师：谭添
- 网站：<https://cs.nju.edu.cn/tiantan/courses/compiler-2024/index.html>
- 学时：16周 (学时64节)
- 时段：周一5-6节、周四3-4节
- 教室：仙II-401
- 助教：张腾 (实验)、吴家焱 (作业)



课程QQ群

课程概要

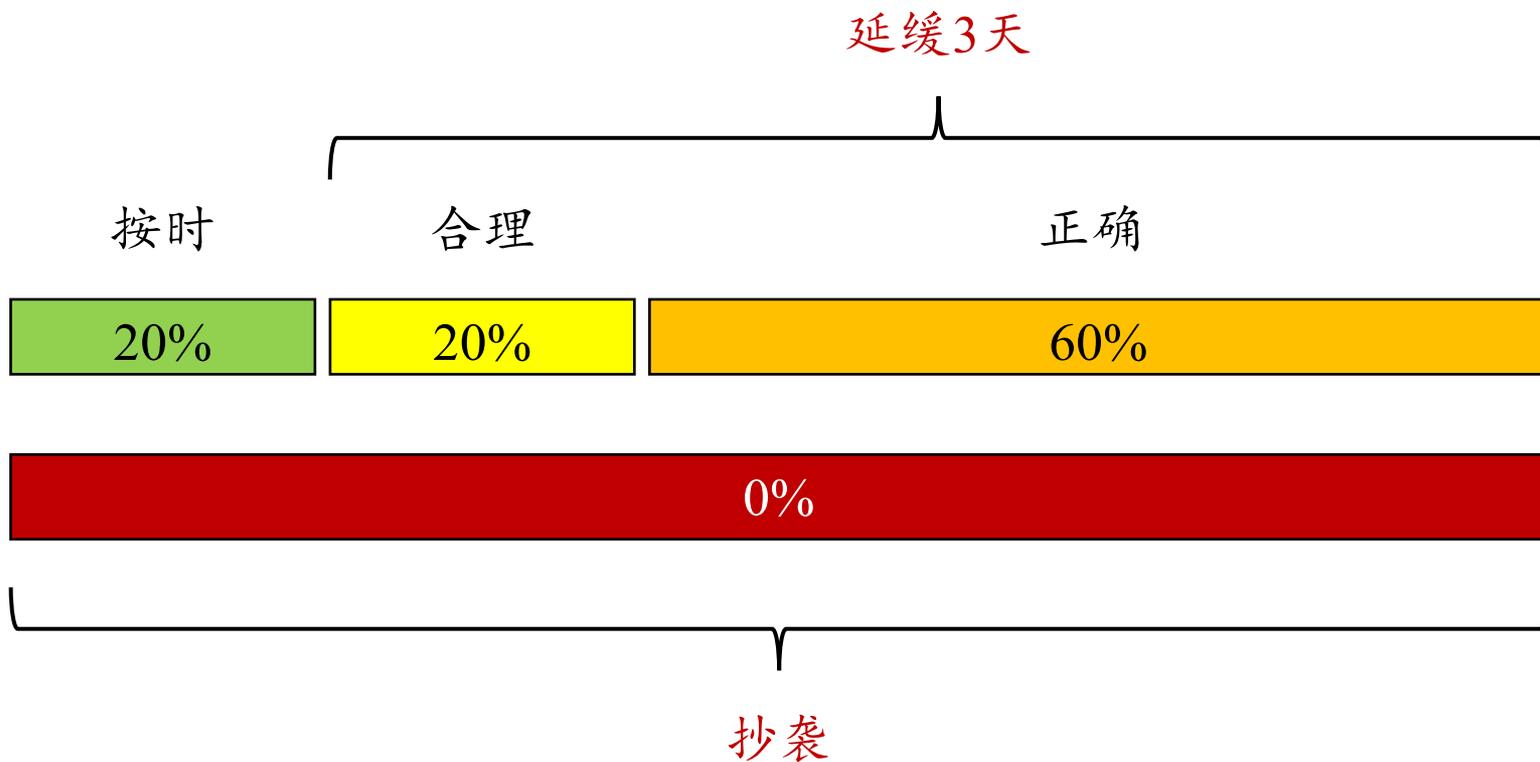
- 课程结构

- 理论部分：上课听讲，下课作业，交书面作业
- 实践部分：实现编译器的几个阶段，交实验作业

- 评分标准

- 书面作业：10% (20% + 20% + 60%)
- 上机实验：30% (20% + 20% + 60%)
 - 组队调整：110% (1人), 100-105% (2人), 90-95% (3人)
 - 实验内容：五次实验
- 期末考试：60%
- 不可控因素：随机签到 (加分)、抄袭检查 (归零) 等

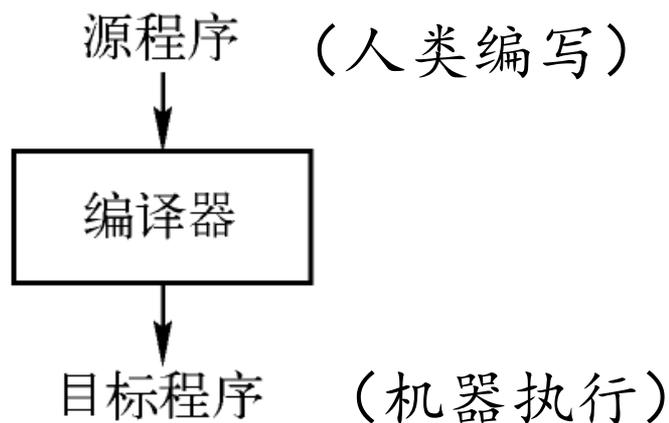
实验说明



什么是编译器？

- 本质上是一种翻译程序
- 读入以某种语言 (源语言) 编写的程序
 - C/C++、Java、Rust、Haskell.....
- 输出等价的用另一种语言 (目标语言) 编写的程序
 - 汇编、WebAssembly.....

通常目标程序是可执行的



为什么学习编译原理？

编译原理 = 编译器工作的原理

编译器：**最重要、最复杂**的基础系统软件之一

- 现代计算机系统的基石软件
- 为程序设计语言提供支撑
- 提高对程序设计语言的理解

- 现代编译器架构形如长流水线
- 涉及程序设计语言理论、操作系统、体系结构、数据结构与算法等技术
- 提高对复杂系统软件的理解

获得能力：创造属于你的编程语言

为什么是我？

- 20年前，命运的齿轮开始转动.....

- “根正苗红”

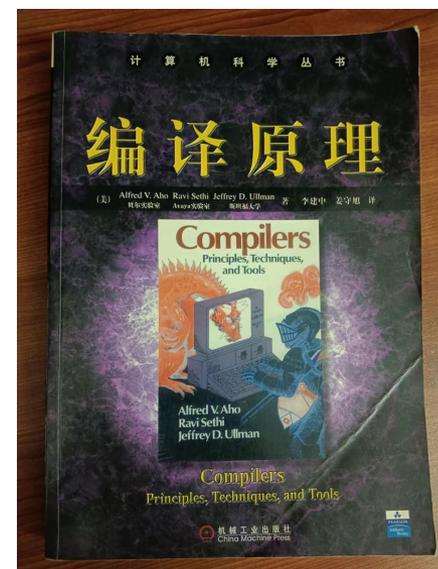
- 研究方向：程序设计语言/程序分析

- PLDI: ACM SIGPLAN Conference on **P**rogramming **L**anguage **D**esign and **I**mplementation
- TOPLAS: ACM **T**ransactions **o**n **P**rogramming **L**anguages and **S**ystems
- 程序分析平台 Tai-e 主要作者

 Tai-e Public

An easy-to-learn/use static analysis framework for Java

 Java  1.2k  150



CORG
C**O**mplier R**E**search G**R**oup@UNSW



太阿 (Tai-e)

Java程序分析平台“太阿” (Tai-e)

- 2020年1月启动
- 2022年8月开（出）源（道）
- 开发时长**两年半**

The screenshot shows the GitHub repository page for Tai-e. The repository is public and has 1.2k stars, 150 forks, and 28 unwatchers. The repository is managed by silverbullettt, who has 2,037 commits. The repository contains several folders: .github, buildSrc, config, docs, and gradle/wrapper. The repository is described as an easy-to-learn/use static analysis framework for Java. The repository is licensed under LGPL-3.0 and GPL-3.0. The repository is available at tai-e.pascal-lab.net/docs/index...

Folder	Commit Message	Time Ago
.github	Update issue templa...	3 months ago
buildSrc	Fix incorrect toolchai...	5 months ago
config	Add Gradle configur...	5 months ago
docs	Fix the greedy backti...	last month
gradle/wrapper	Bump Gradle to 8.5	3 months ago

“太阿”的用户

- 20+高校学者
- 10+企业部门主管
- 150+高校/科研院所
 - 华盛顿大学
 - 宾夕法尼亚大学
 - 普渡大学
 - 东京大学
 - 阿姆斯特丹大学
 - 清华大学
 - 北京大学
 - 香港大学
 -

Pulse

Contributors

silverbullettt

Overview Repositories 13



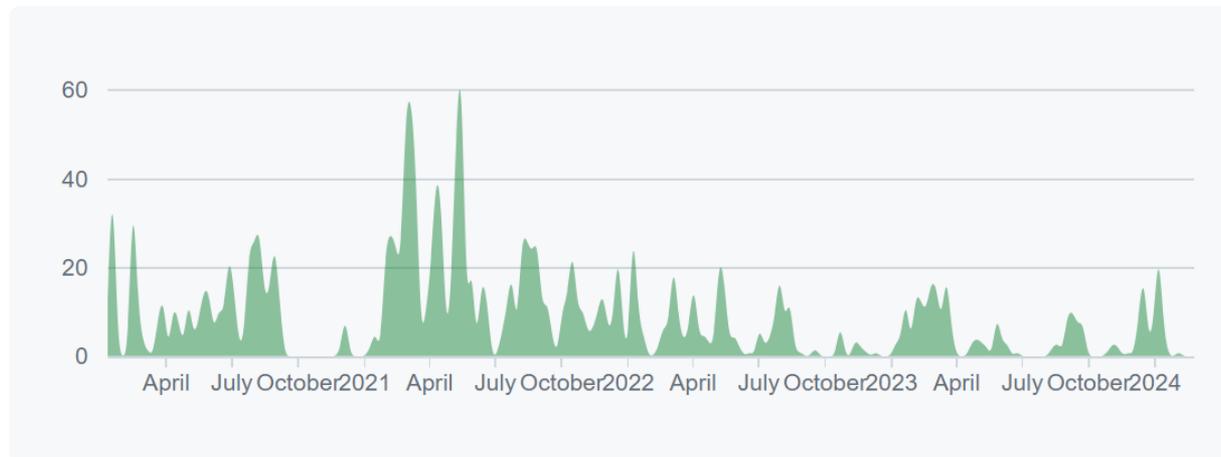
Tian Tan
silverbullettt

编写7万+行代码

Jan 12, 2020 – Feb 25, 2024

Contributions: Commits

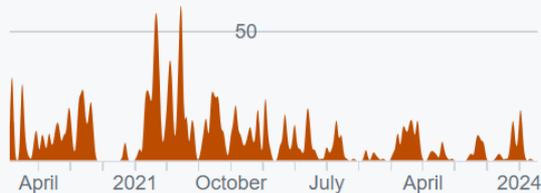
Contributions to master, excluding merge commits



silverbullettt

#1

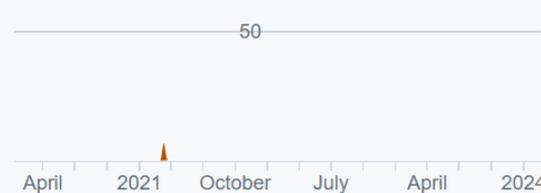
1,862 commits 340,690 ++ 264,221 --



yuelee

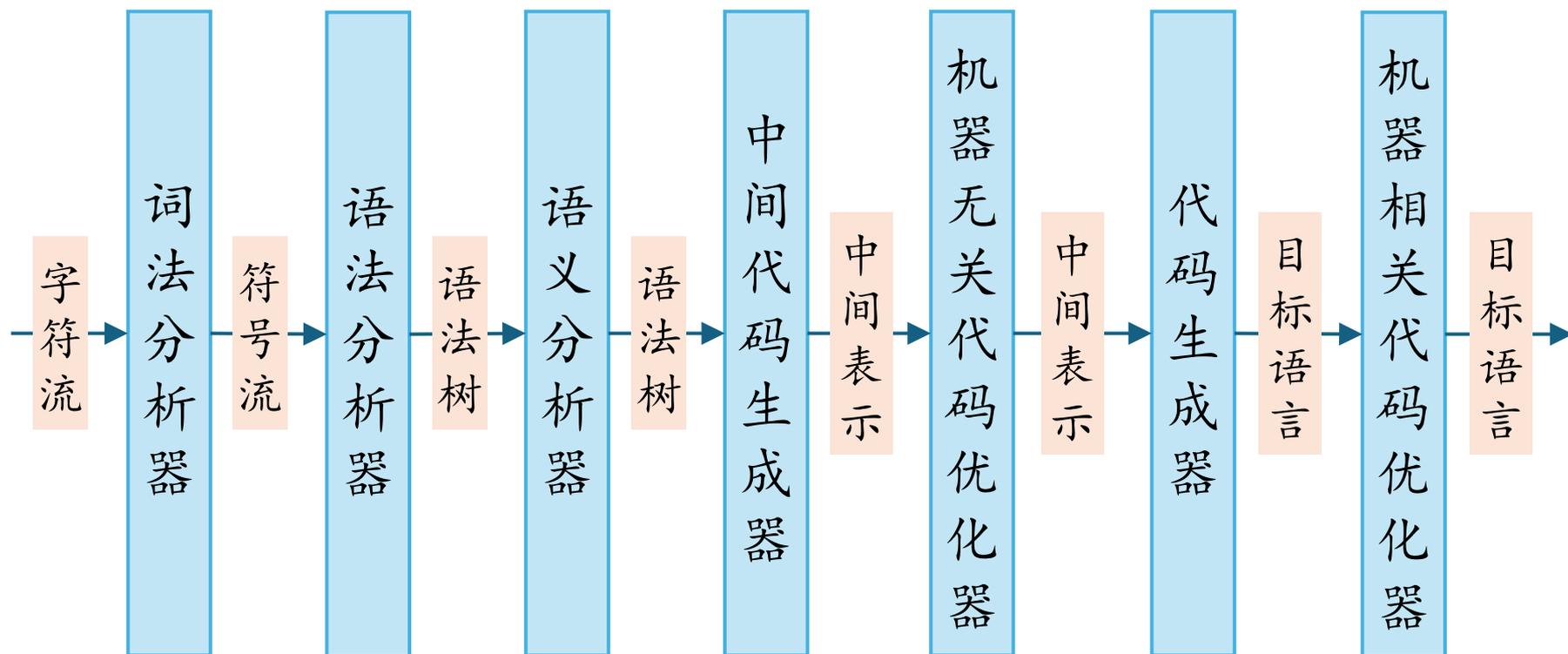
#2

10 commits 25,977 ++ 26,337 --



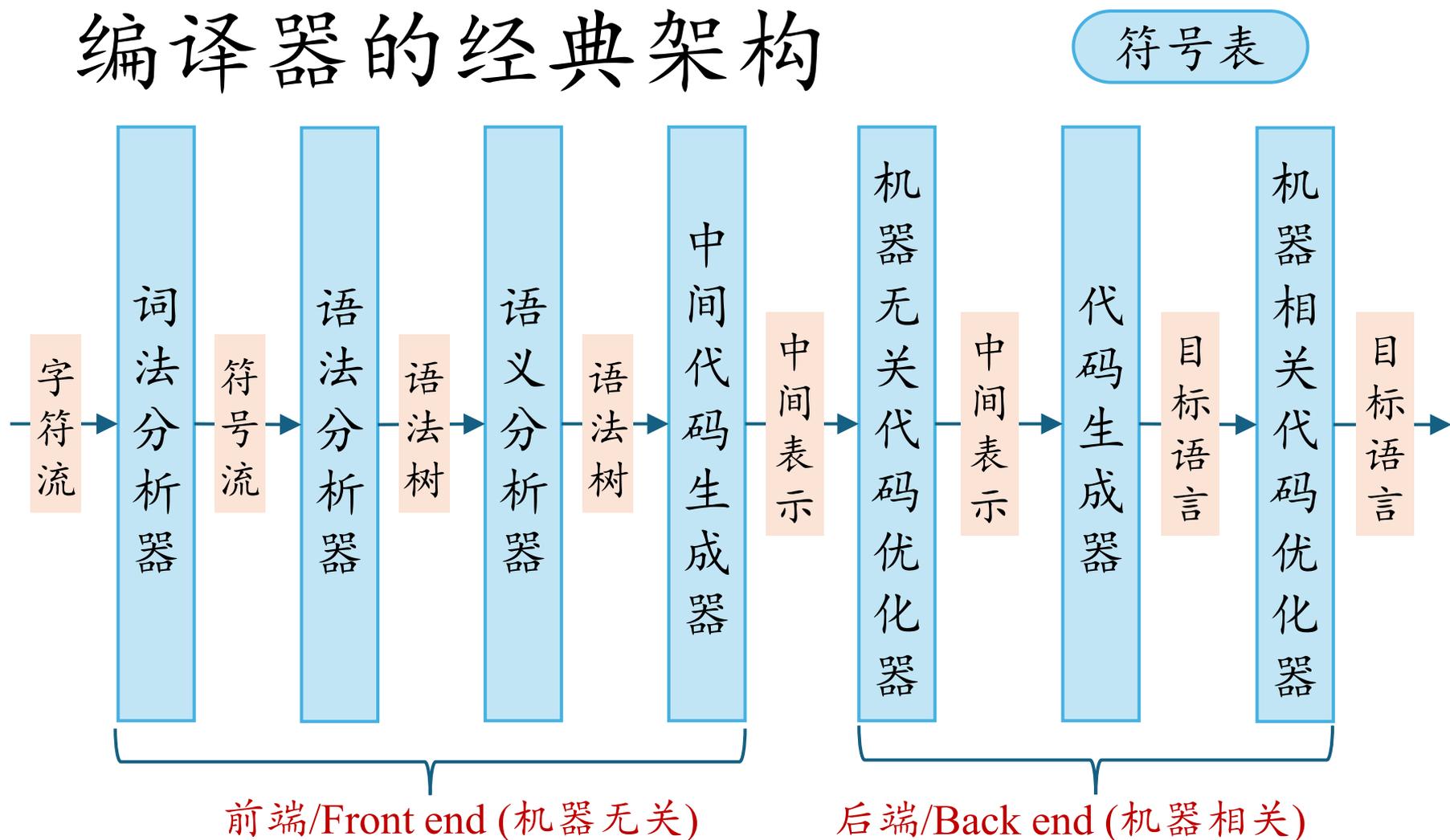
编译器的经典架构

符号表



“编译器是一条巨大的流水线”

编译器的经典架构



- 从源程序解析出以及相应的语法结构
- 使用这个结构创建源程序的中间表示
- 同时收集源程序信息，存入符号表

- 根据中间表示和符号表信息构造目标程序
- 同时对目标程序进行分析、优化

课程内容

1. 导引 (易)
 2. 词法分析 (难-)
 3. 语法分析 (难)
 4. 语法制导的翻译技术 (中)
 5. 中间代码生成 (难)
 6. 运行时刻环境 (易)
 7. 代码生成 (中)
 8. 机器无关优化 (中)
- 安排较紧
- 安排较松

符号表 (Symbol Table)

- 用于记录源程序中出现的标识符信息的数据结构
 - 标识符的**名字**
 - 变量名、函数名
 - 标识符在源程序中的**位置**
 - 文件名 + 起止行列号
 - 用于精准报错
 - 标识符的**类型**
 - 用于语义分析、代码生成
 -

在编译器运行过程中逐步构建

词法分析 (Lexical Analysis)

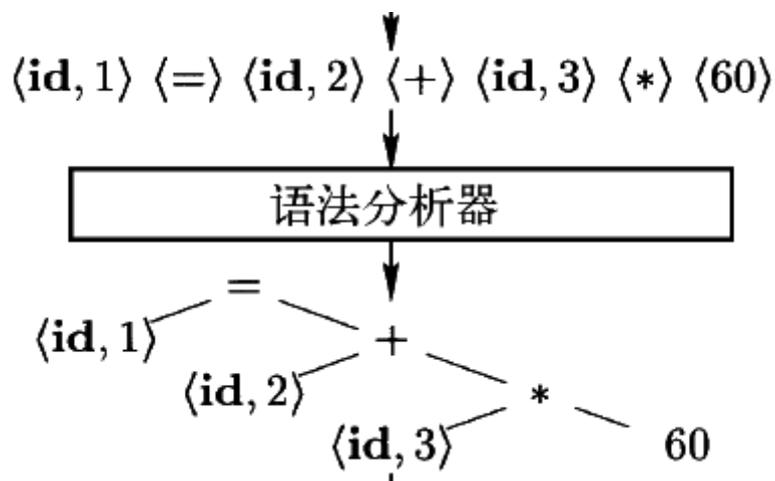
- 读入源程序的字符流，输出为词法单元 (Token)
 - Token kind: 表示词素种类
 - Attribute value: 表示词素附带属性信息
 - Source location: 表示词素在源代码中的位置

符号表

- 例子
 - $\text{position} = \text{initial} + \text{rate} * 60$
 - $\langle \text{id}, 1 \rangle \langle =, \rangle \langle \text{id}, 2 \rangle \langle +, \rangle \langle \text{id}, 3 \rangle \langle *, \rangle \langle \text{number}, 60 \rangle$

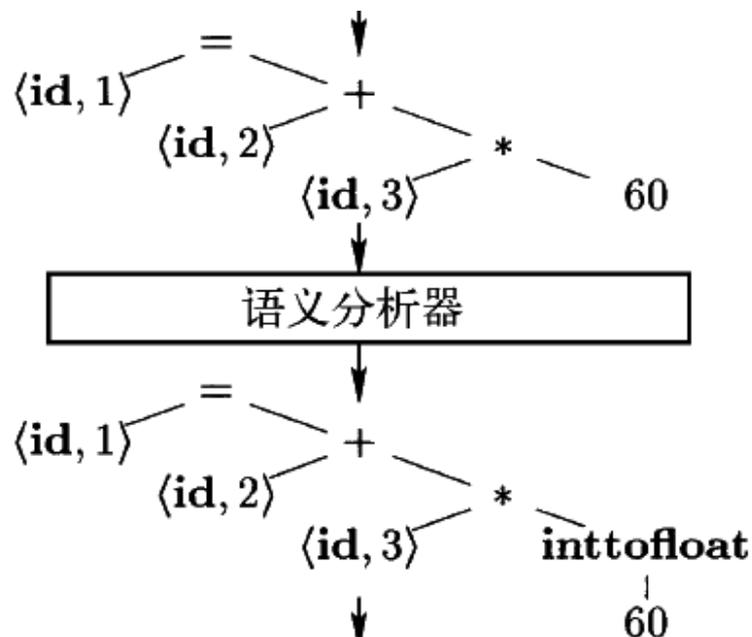
语法分析 (Syntax Analysis/Parsing)

- 根据各个词法单元的第一个分量来创建树型的中间表示形式，通常是**语法树 (Syntax tree)**
- 中间表示形式指出了词法单元流的语法结构



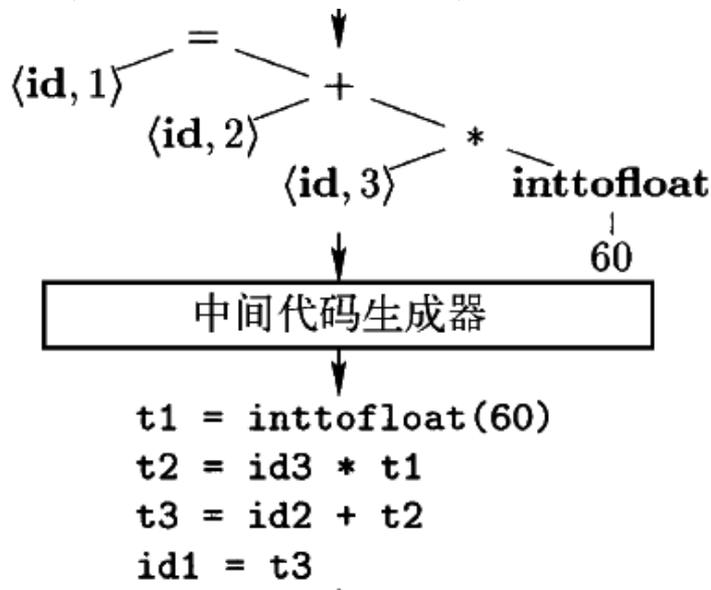
语义分析 (Semantic analysis)

- 使用语法树和符号表中的信息，检查源程序是否满足语言定义的语义约束
- 同时收集类型信息，用于代码生成、类型检查、类型转换



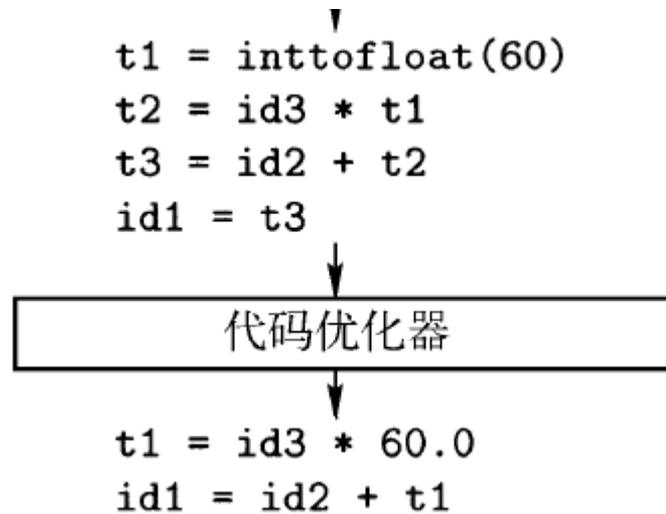
中间表示生成 (IR Generation)

- 根据语义分析输出，生成类机器语言的中间表示
- 三地址代码
 - 每个指令最多包含三个运算分量
 - $t1 = \text{inttofloat}(60); t2 = \text{id3} * t1; t3 = \text{id2} + t2; \dots$
 - 很容易生成机器语言指令



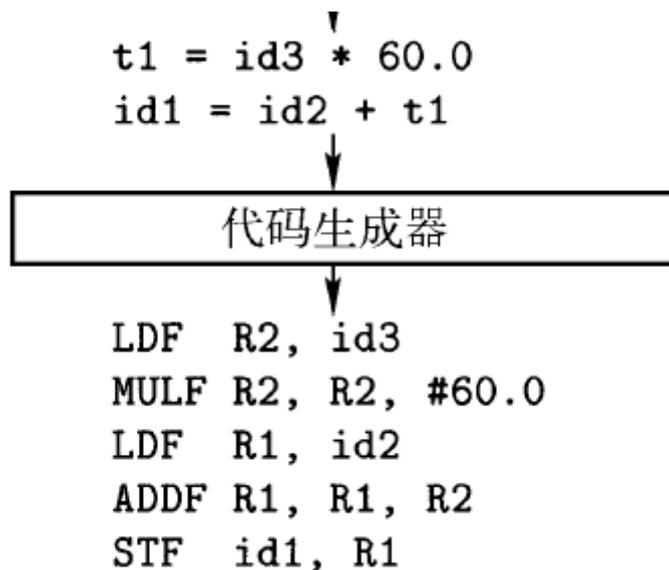
中间代码优化 (Optimization)

- 通过对中间代码的分析，改进中间代码的质量
 - 更快、更短、能耗更低



目标代码生成 (Code Generation)

- 把中间表示形式映射到目标语言
 - 指令选择
 - 寄存器的分配



编译器与解释器

- 编译器

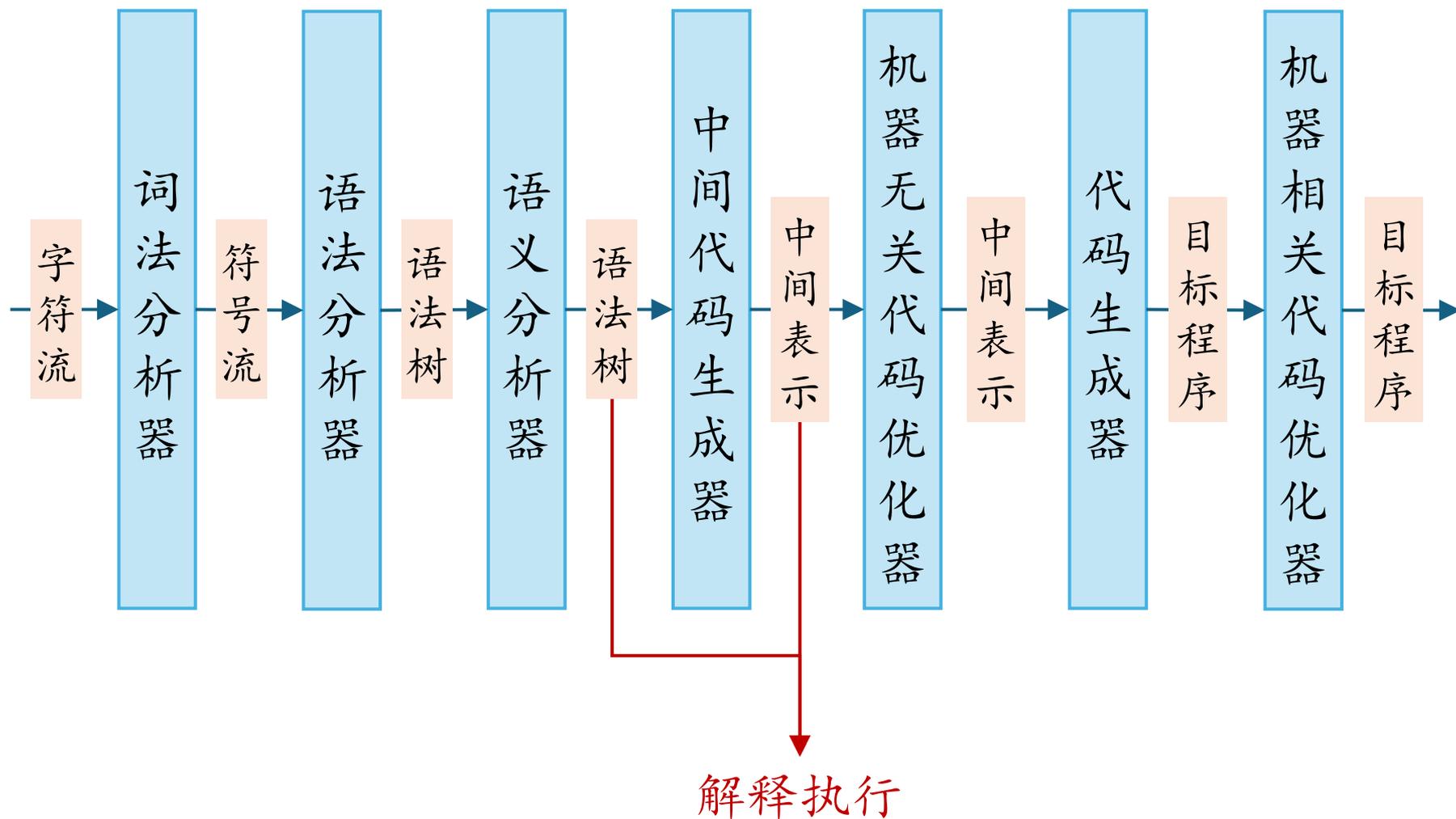
- 读入以某种语言 (源语言) 编写的程序
- **输出等价的用另一种语言** (目标语言) 编写的程序, 通常目标程序是可执行的
- C/C++、Rust、Go.....

- 解释器

- 直接利用用户提供的输入, 执行源程序中指定的操作
- **不生成目标程序**, 而是根据源程序的语义直接运行
- Python、Ruby、JavaScript👎.....

编译器与解释器

符号表



编译器与解释器

- 二者界限并非明确，许多语言结合了编译和解释
 - 同时包含编译与解释
 - 例如Java, Scala, Kotlin等JVM-based语言，先编译成字节码(bytecode)，然后解释执行
 - 解释执行的过程中加入编译 (Just-in-time, JIT)
 - 例如Java (Hotspot, GraalVM), JavaScript (V8)
 - 在你解释之前，我已经编译了 (Ahead-of-time, AOT)
 - 例如.....没错还是Java

编译器架构演化

- 经典架构：直线型

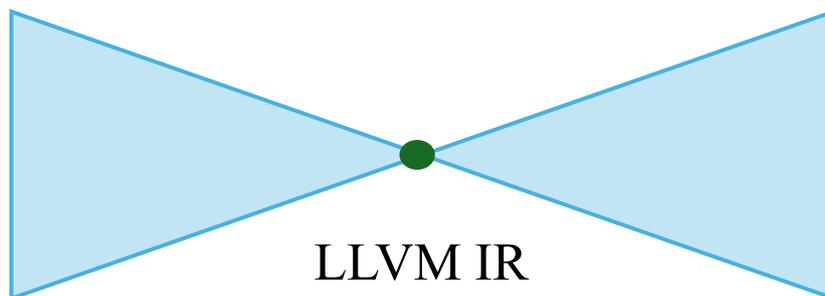


- 现代架构：沙漏型

- 为多前端、多后端的需求提供一套公共接口 (中间表示)

- 例如LLVM

- C/C++
- Rust
- Go
-



- X64
- ARM
- RISC-V
-

编译器架构演化

- 经典架构：批量处理型
 - 一个阶段全部运行完毕后再进入下一阶段
- 现代架构：请求驱动型
 - 既能编译完整程序，也能响应分散的IDE请求
 - 例如Roslyn

程序设计语言的发展历程

- 历程

- 机器语言

- 汇编语言 (宏命令)

- 高级语言

- 通用语言: Fortran、Cobol、Lisp、C、C++、...

- 特定应用语言: SQL、Postscript、NOMAD、...

- 基于逻辑和约束的语言: Prolog、OPS5、...

- 演化出不同的语言范式

- 命令式语言

- 函数式语言

- 逻辑式语言

许多语言**借鉴**其它语言特性
各种范式呈现**融合**趋势

编程语言和编译器之间的关系

- 程序设计语言的发展向编译器设计者提出新要求
 - 设计相应的算法和表示方法来翻译和支持新的语言特征，如多态、动态绑定、类、类属(模板)、...
- 通过降低高级语言的执行开销，推动这些高级语言的使用
- 编译器设计者还需要更好地利用新硬件的能力
 - RISC技术、多核技术、大规模并行技术

编译技术的应用

- 文本处理与程序翻译
 - 二进制翻译/硬件合成/数据查询解释器/编译后模拟
- 软件开发工具
 - 智能提示/代码补全/错误提示
- 软件质量工具
 - 程序分析/软件测试/内存管理