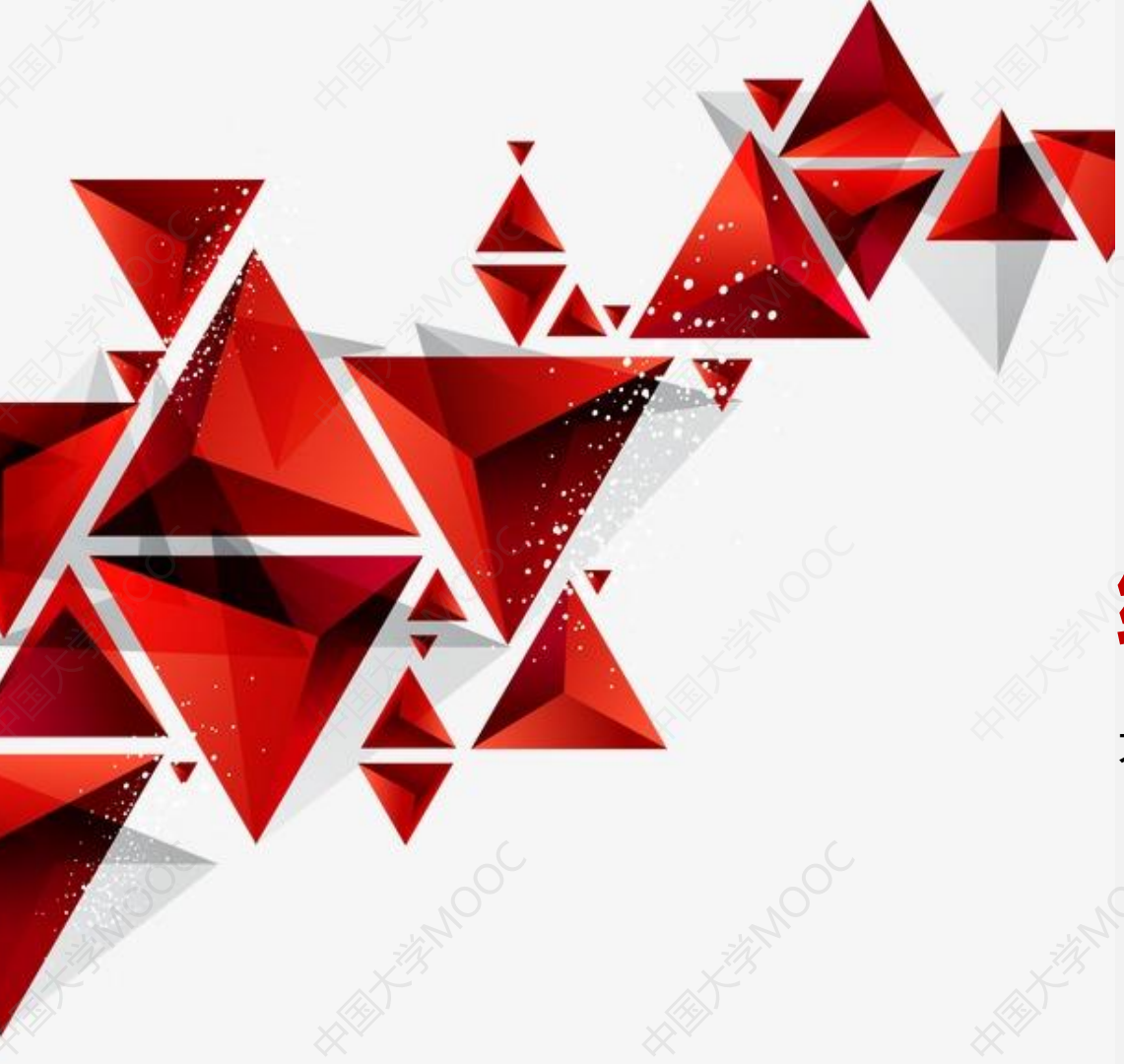
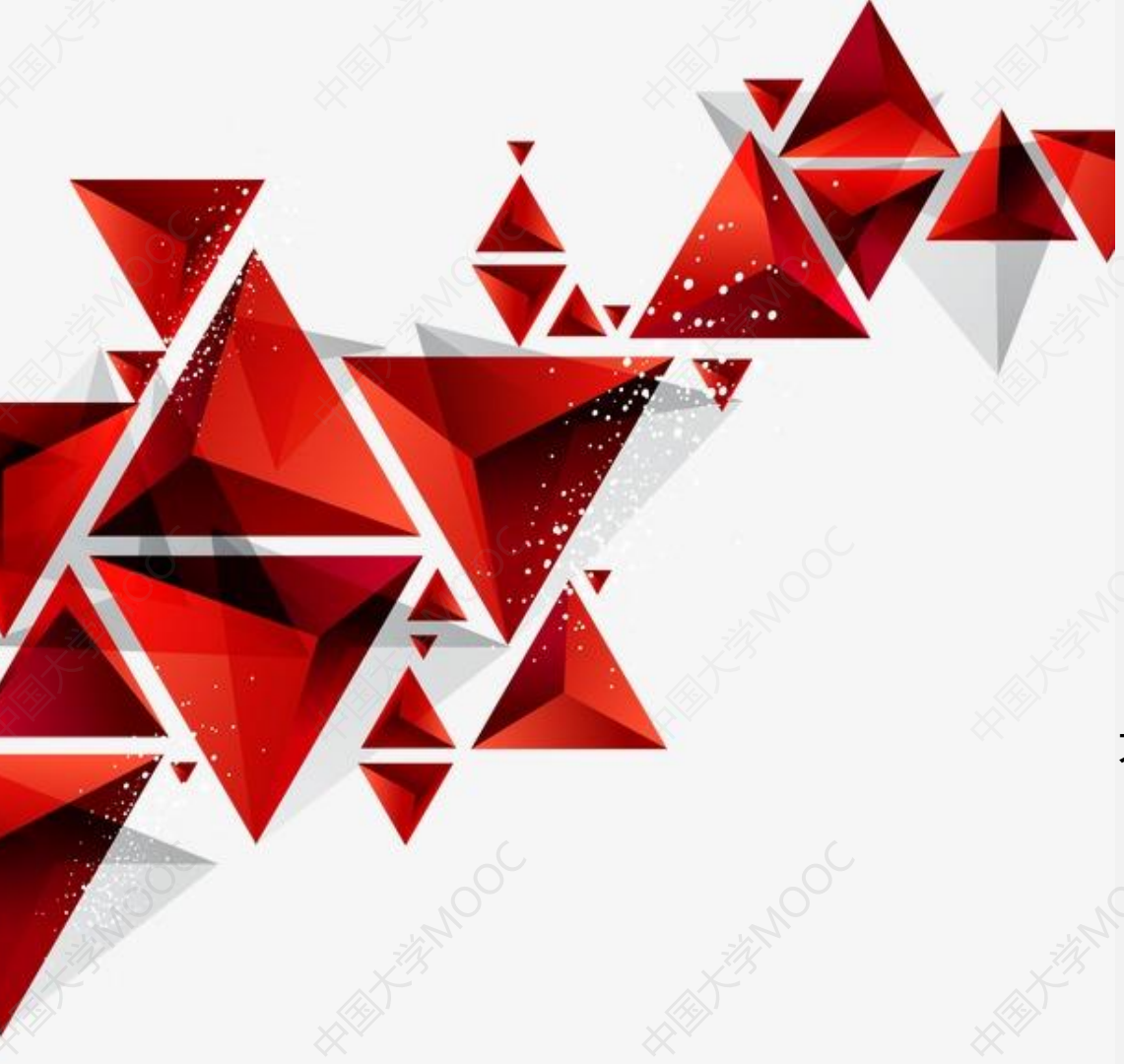


大连理工大学
DALIAN UNIVERSITY OF TECHNOLOGY

# 编译技术

大连理工大学软件学院

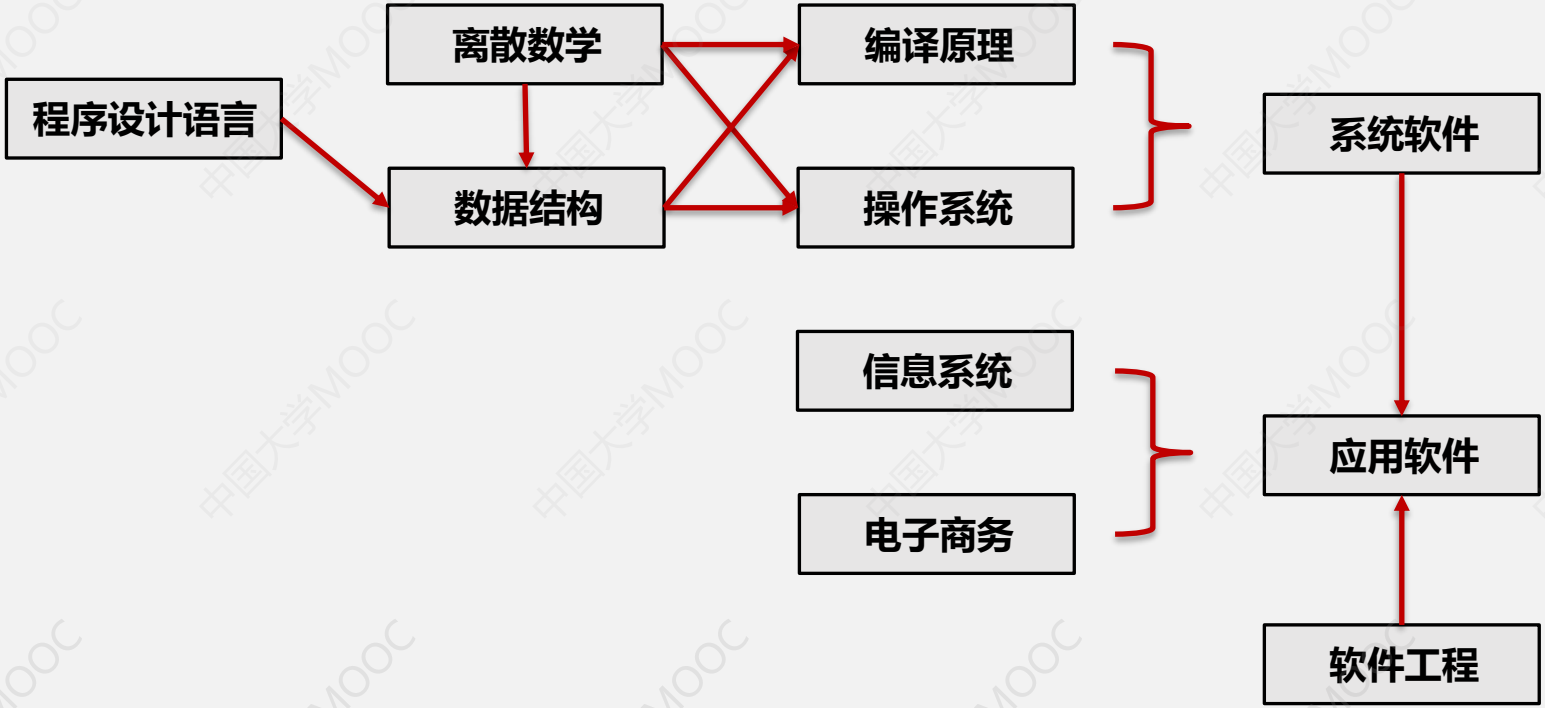


# 编译技术

# 绪 论

大连理工大学软件学院

# 编译原理课程在计算机科学技术中的地位：



课程简介

## 课 程 内 容

- 介绍编译器构造的一般原理和基本实现方法
- 理论知识：形式语言和自动机理论、语法制导的定义和属性文法、类型论与类型系统、程序分析原理等
- 强调形式化描述技术
- 强调对编译原理和技术的宏观理解，不把注意力分散到枝节算法，不偏向于某种源语言或目标机器

## 课 程 内 容

**学习意义**

对编程语言的设计和实现有深刻的理解，
对和编程语言有关的理论有所了解，
对宏观上把握编程语言来说，起一个奠基的作用

从软件工程看，编译器是一个很好的实例，所介绍的概念和技术能应用到一般的软件设计之中

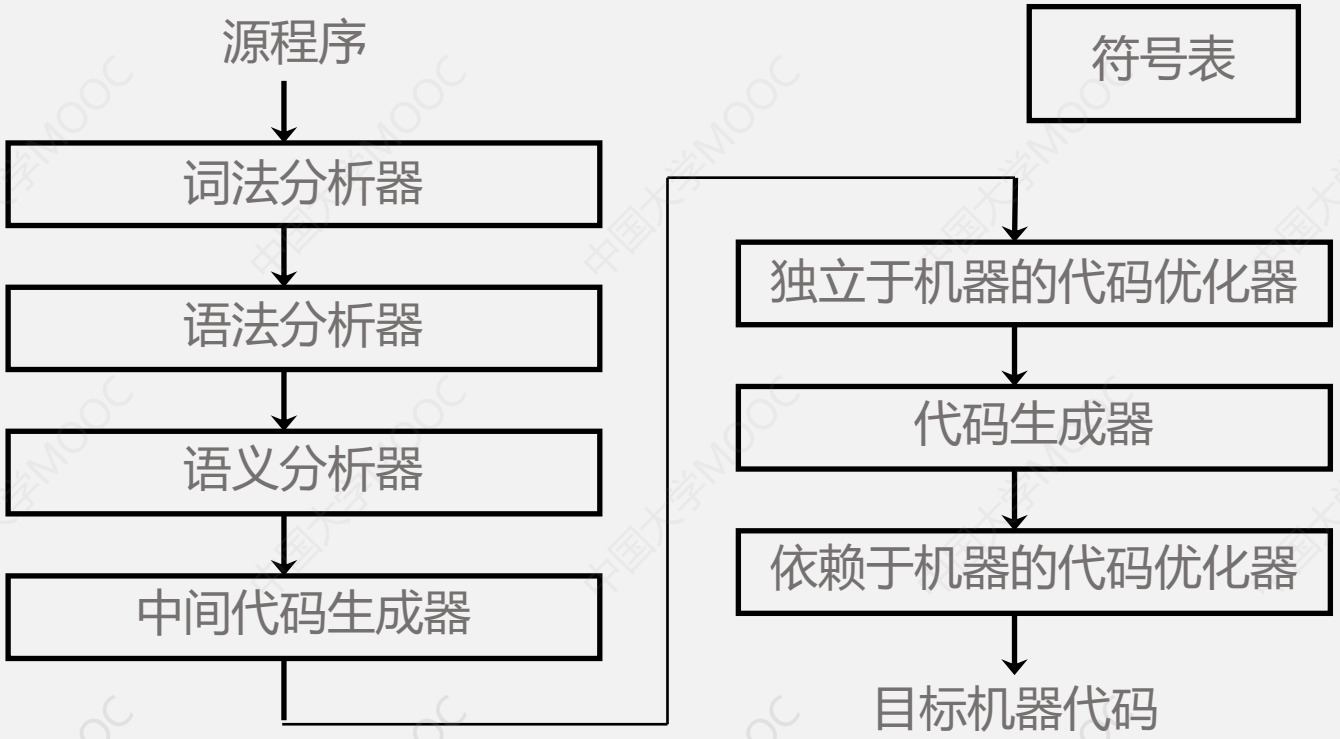
编译技术的应用和编译技术的发展

高级语言设计、计算机体系结构的优化（并行、内存分层）、
新型计算机体系结构设计、程序翻译、
提高软件开发效率的工具 、高可信软件

# 引论

- 编译器从逻辑上可以分成若干个阶段
- 每个阶段把源程序从一种表示变换成另一种表示
- 本章通过描述编译器的各个阶段来介绍编译这个课题

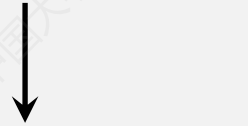
编译器概述



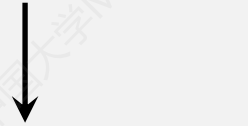
编译器概述

词法分析：源程序 -〉词法记号（token）流

position = initial + rate * 60 ← 字符流



词法分析器



⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩ ← 记号流

符号表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

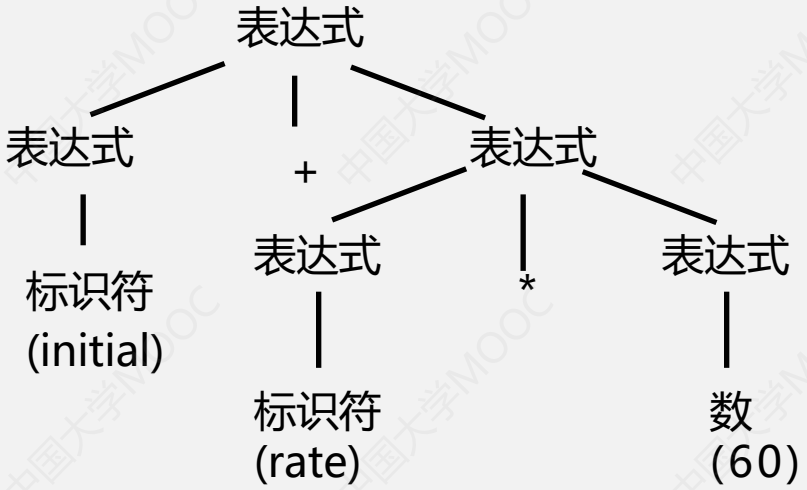
**语法分析：词法记号（token）流-〉语法短语**

编译器概述

表达式的语法特征

- 任何一个标识符都是表达式
- 任何一个数都是表达式
- 如果$e_1$和$e_2$都是表达式，那么
  - $e_1 + e_2$
  - $e_1 * e_2$
  - $(e_1)$

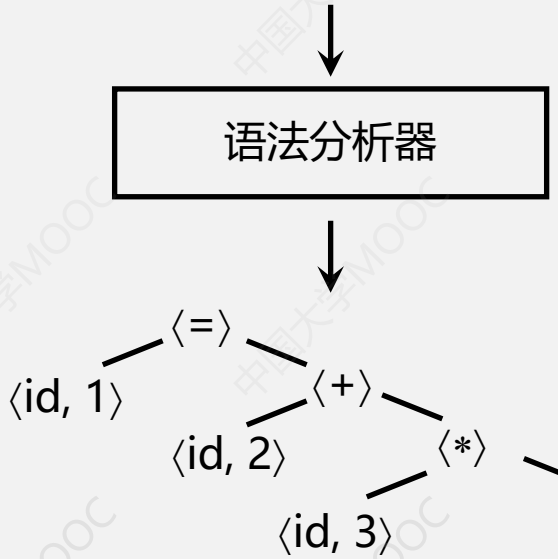
也都是表达式




initial + rate * 60的分析树

编译器概述

⟨id, 1⟩ ⟨=⟩ ⟨id, 2⟩ ⟨+⟩ ⟨id, 3⟩ ⟨*⟩ ⟨60⟩ ← 记号流



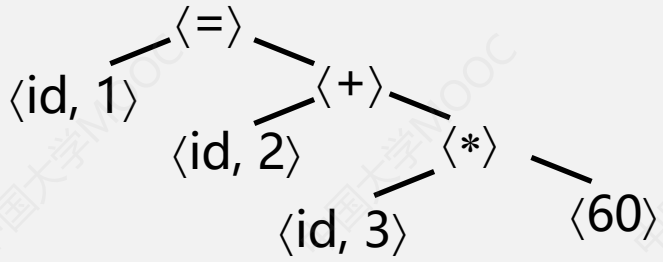

← 语法树

符号表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

**语义分析：检查程序的语义正确性，如类型检查等**

编译器概述

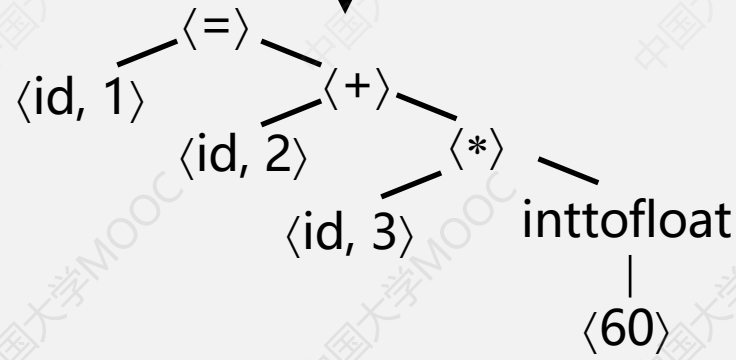



← 语法树

符 号 表

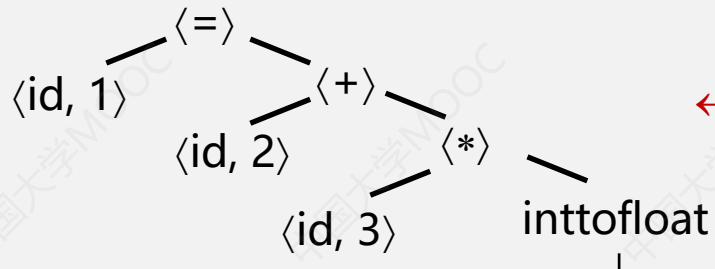
| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

语义分析器




← 语法树

编译器概述




← 语法树

符 号 表

| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

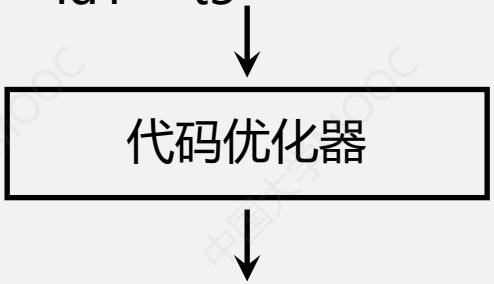
中间代码生成器

```
t1 = inttofloat(60)
t2 = id3 * t1
t3 = id2 + t2
id1 = t3
```

← 三地址中间代码

编译器概述

t1 = inttofloat(60) ← 三地址中间代码
t2 = id3 * t1
t3 = id2 + t2
id1 = t3




t1 = id3 * 60.0
id1 = id2 + t1 ← 三地址中间代码

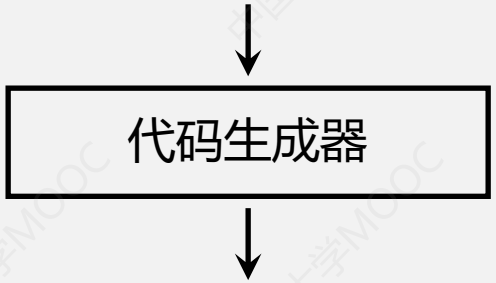
符号表

| | | |
|---|---|---|
| 1 | position | ··· |
| 2 | initial | ··· |
| 3 | rate | ··· |
| | | |

编译器概述

t1 = id3 * 60.0
id1 = id2 + t1

← 三地址中间代码




MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

← 汇编代码

符号表

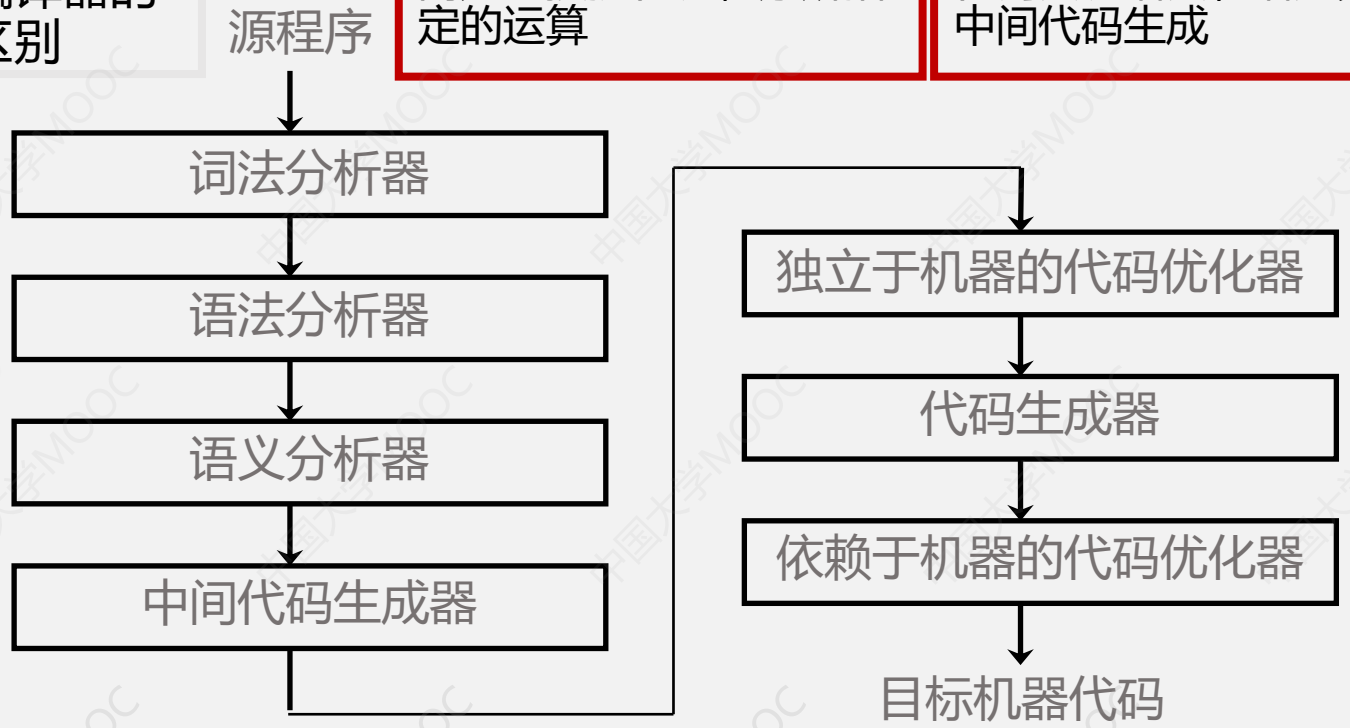
| | | |
|---|---|---|
| 1 | position | · · · |
| 2 | initial | · · · |
| 3 | rate | · · · |
| | | |

编译器概述

解释器和编译器的区别

解释器不生成目标代码，而是直接执行源程序所指定的运算

解释器也需要对源程序进行词法、语法和语义分析，中间代码生成



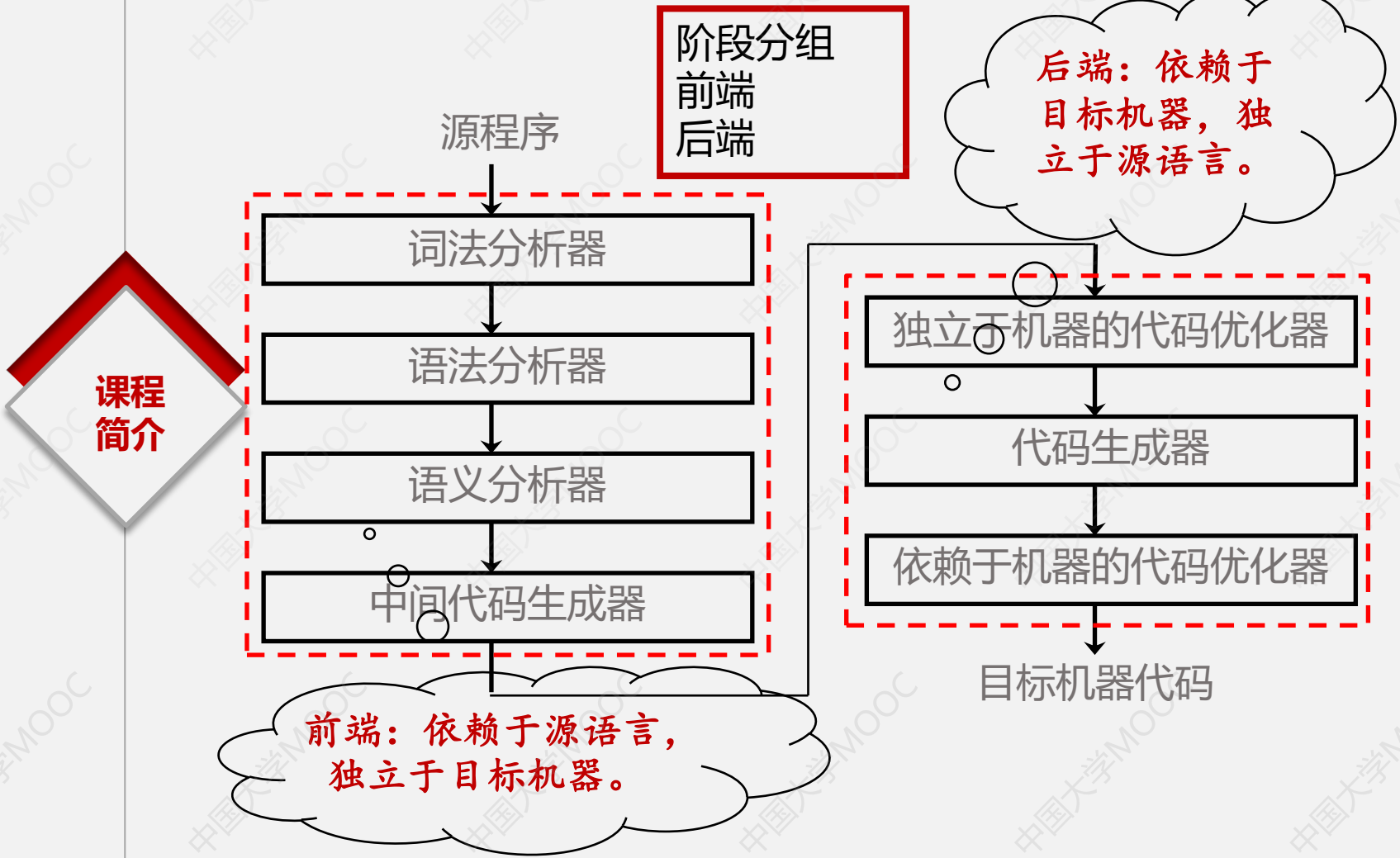
编译器概述

## BASIC年代的解释器

- 功能：它将高级语言的源程序翻译成一种中间语言程序，然后对中间语言程序进行解释执行
- 在那个年代，编译和解释两个功能是合在一个程序中，该程序被称为解释器

## Java年代的解释器

- 解释器的上述两个功能分在两个程序中
- 前一个叫做编译器，它把源程序翻译成一种叫做字节码的中间语言程序
- 后一个叫做解释器，它对字节码程序进行解释执行


阶段分组
前端
后端
源程序
词法分析器
语法分析器
语义分析器
中间代码生成器
前端：依赖于源语言，
独立于目标机器。
后端：依赖于
目标机器，独
立于源语言。
独立于机器的代码优化器
代码生成器
依赖于机器的代码优化器
目标机器代码
课程
简介

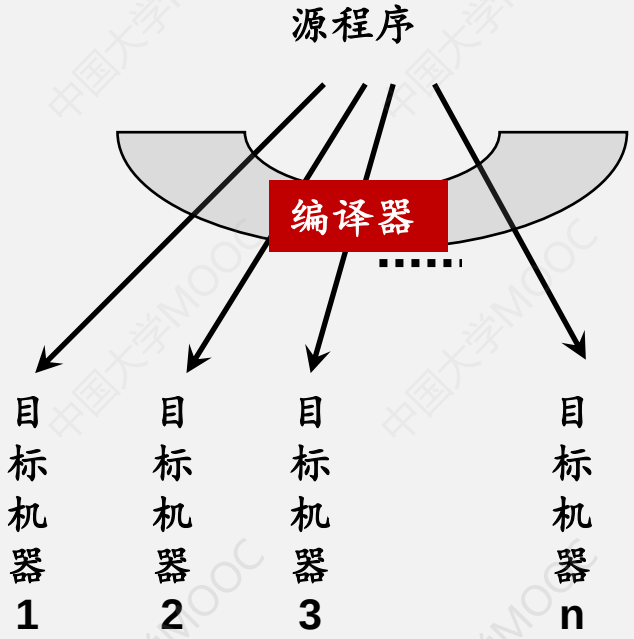
# 引　论

## 前端和后端

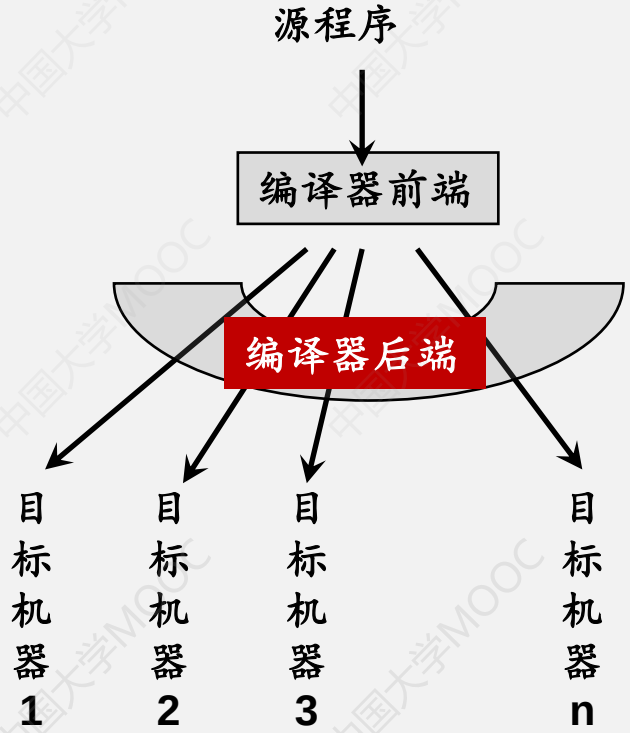
- 把编译过程分成前端和后端两部分
- 前端：只依赖于源程序，独立于目标机器（生成中间代码）
- 后端：依赖于目标机器，与源程序无关，只与中间语言有关（从中间代码生成目标代码）
- 好处：提高开发编译器的效率
  - 取一个编译器的前端，重写它的后端以产生同一源语言在另一机器上的编译器
  - 不同的前端使用同一个后端，从而得到一个机器上的几个编译器（采用同一中间语言）

# 引　论

不区分前端和后端的编译器




区分前端和后端的编译器

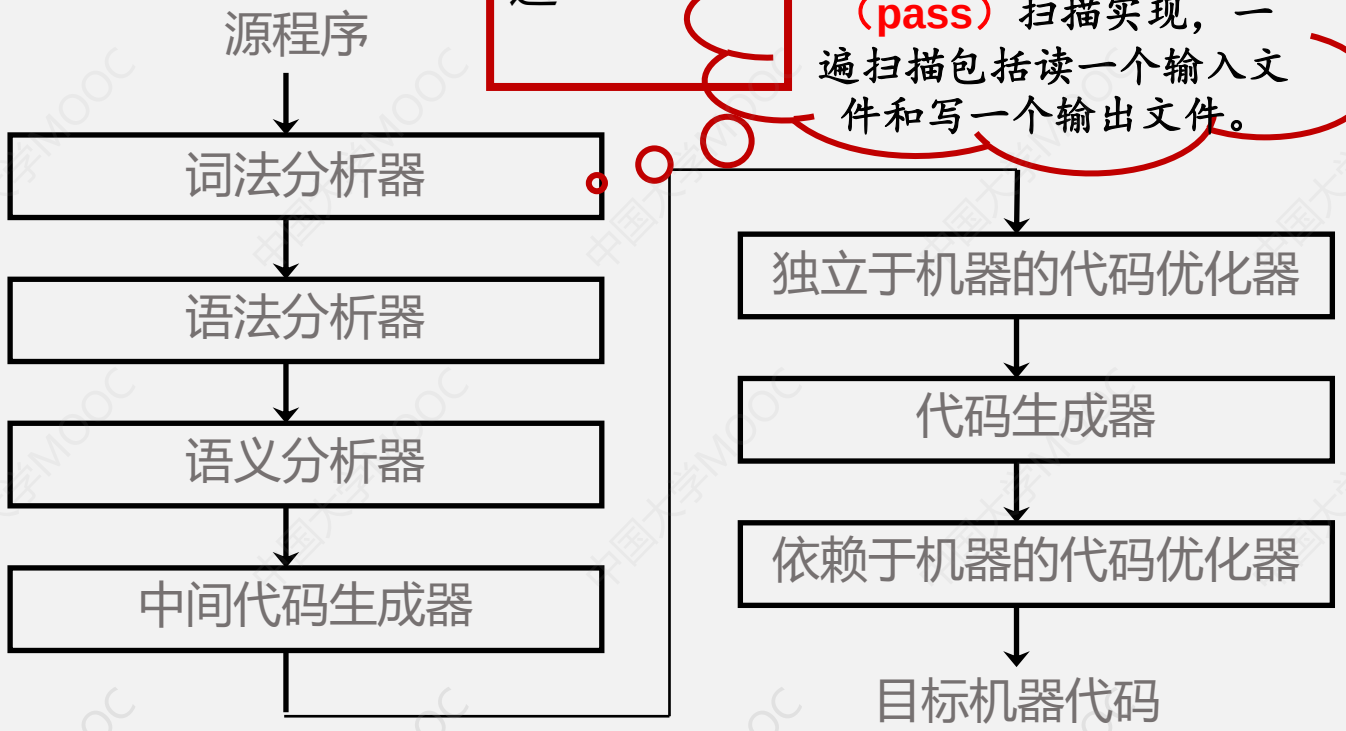


课程简介

阶段分组

一遍

编译的几个阶段常用一遍（pass）扫描实现，一遍扫描包括读一个输入文件和写一个输出文件。



## 遍（趟）：

- 一遍或一趟：是指编译程序在编译时刻把源程序或源程序的等价物（中间程序）从头到尾扫描一遍并转换成另一紧邻的等价物的全过程。
- 单遍扫描与多遍扫描：每一遍的扫视可完成上述一个阶段或多个阶段的工作。每一遍的输入都是上一遍的输出，第一遍的输入是源程序正文，最后一遍的输出是目标代码。
- 单遍与多遍的比较：
  - 遍数多：编译器结构清晰，但时间效率不高
  - 遍数少：编译速度快，但对机器的内存要求高
- 遍数的确定：主要因素是源程序和机器（目标机器）的特征。

# 编译器技术的应用

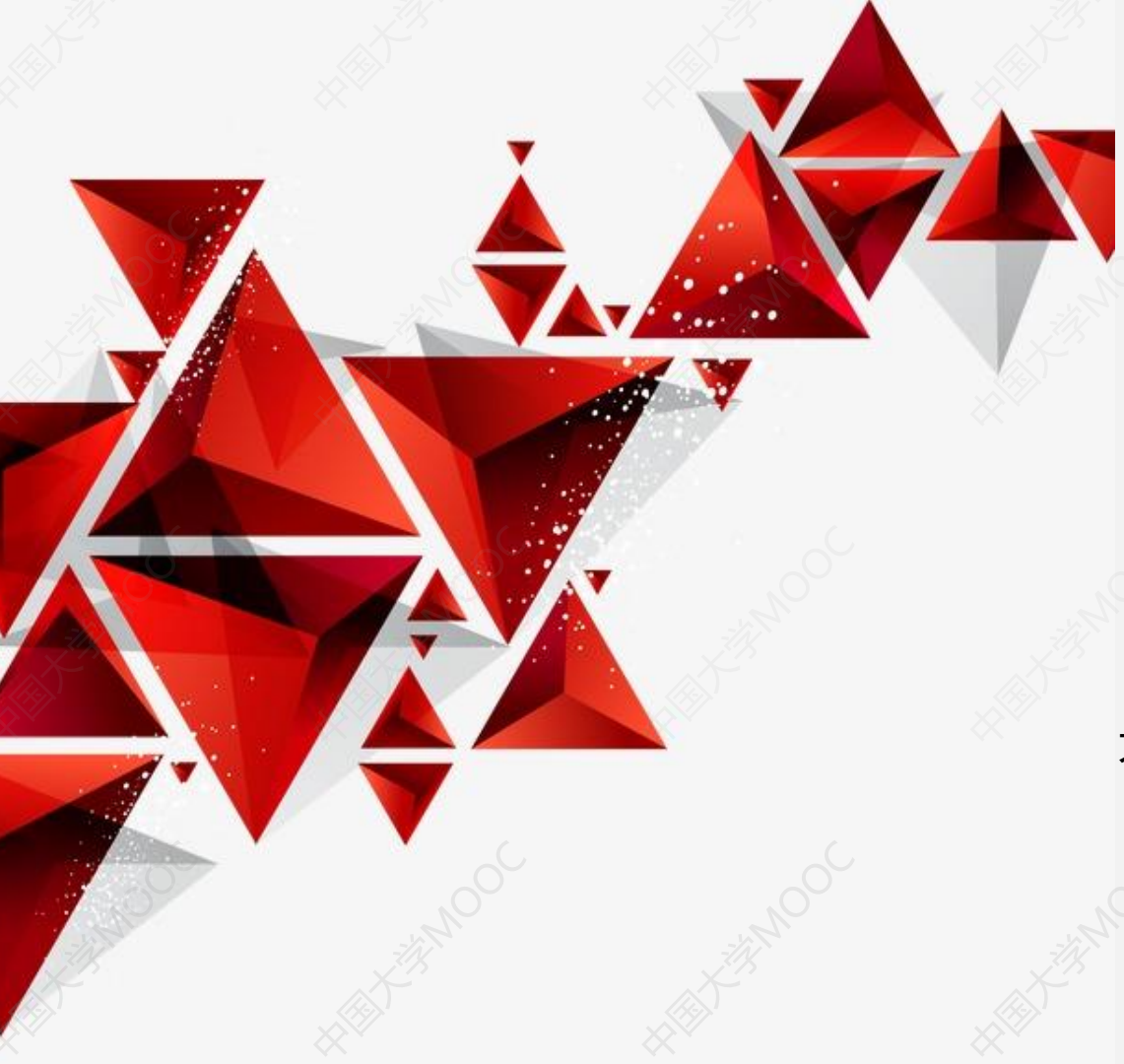
## 提高软件开发效率的工具

源于编译器中代码优化技术的程序分析一直在改进软件开发效率。

- 类型检查：类型检查是一种捕捉程序中前后不一致的成熟而有效的技术。
- 边界检查：数据流分析技术可用来定位缓冲区溢出。
- 内存管理：自动的内存管理删除内存泄漏等内存管理错误。

# 编译器技术的应用

- 语法制导的结构化编辑器
- 程序格式化工具
- 软件测试工具
- 程序理解工具
- 高级语言的翻译工具
- 等等



# 编译技术

# 绪 论

大连理工大学软件学院