# 概述

计算机科学领域使用符号表（symbol table）来表示“字典”。

## 功能

符号表是将值（value）与键（key）关联起来的数据结构，它支持下列操作：

1、搜索特定的键名，以判断符号表中是否存在这个键；

2、获取与某个键名相关联的属性值；

3、修改与某个键名相关联的属性值；

4、将新的键名及与该键相关联的属性值插入符号表；

5、删除某个键名及其属性值。

## 符号表 vs 哈希表

符号表是一种抽象数据类型，而哈希表是一种实现符号表的具体数据结构。

**区别：**

图形用户界面

中度可信度描述已自动生成

**联系（从实现角度）：**

符号表是一个抽象模型，它描述的是一种“名字到值”的映射，类似于数学中的映射或编程语言中的变量表。

哈希表是符号表最常用的底层实现方式，尤其在需要快速查找、插入和删除时。

在 Java 或 C++ 中，HashMap / unordered\_map 就可以用来作为符号表的实现。

# 实现

可以实现符号表的方式：

## 无序数组

1、无序的数组实现

这种实现方式只需要维护一个普通的数组就可以。

其查找、插入及删除在最差的情况下时间复杂度均为O(n)。

## 有序数组

2、有序的数组实现

这种实现方式需要将键与值存放在有序的数组中。

由于元素是按照顺序存放在数组中的，因此可以采用简单的二分搜索法查找，这需要花费O(logn)级别的时间。而插入与删除操作在最差情况下的复杂度为O(n)。

## 无序链表

3、无序的链表实现

这种链表的每个结点都有两个字段，分别用来存放键与值。

查找、插入与删除操作在最差的情况下复杂度为O(n)。

## 有序链表

4、有序的链表实现

如果用这种方法实现，那么向链表中插入新键的时候，就必须按照顺序，把该键调整到它应该在的位置上。

其查找、插入及删除操作在最差的情况下复杂度为O(n)。

## 二叉搜索树

5、二叉搜索树实现

这种办法的好处在于，开发者所要编写的代码比较少，而且查找操作的速度比较快（一般来说，只需要耗费O(logn)级别的时间）。

## 平衡的二叉搜索树

6、平衡的二叉搜索树实现

这是在二叉搜索树基础上的扩充。其查找、插入及删除操作在最差的情况下都需要花费O(logn)级别的时间。

## 三元搜索树

7、三元搜索树实现

## 哈希

8、哈希技术实现

## 区别

各种实现方式的比较：

表格

描述已自动生成

# 应用

符号表的主要应用领域

1、编译器/解释器

最经典、最重要的应用场景。

作用：

存储变量、常量、函数名、类名等标识符。

记录每个标识符的类型、作用域、内存位置等元信息。

使用阶段：

词法分析：识别关键字、标识符等；

语义分析：检查变量是否定义、类型是否匹配；

中间代码生成：需要知道变量或函数的地址、类型等。

示例：

int x = 5;

float y = 3.14;

x = x + y; // 编译器会检查 x 和 y 是否定义，是否类型匹配

2、IDE 的代码补全、语法高亮

符号表可以帮助开发工具提供智能提示：

自动补全变量、函数名；

提示参数类型；

实现跳转定义、查找引用。

3、链接器与装载器（Linker & Loader）

维护符号表记录不同模块的全局符号（如函数和变量）。

判断符号是否重复定义或未定义。

将符号地址解析为真实的内存地址。

4、调试器（Debugger）

使用符号表将内存地址映射为变量名/函数名，供开发者查看调试信息。

支持断点、变量值查看、栈帧分析等功能。

5、脚本语言的运行时环境

Python、JavaScript、Lua 等解释型语言在运行时维护符号表（或环境表）：

每个函数、作用域、模块有自己的局部/全局符号表；

动态添加/删除变量。

a = 10 # 全局符号表

def func():

b = 5 # 局部符号表

6、数据库管理系统（DBMS）

类似符号表的机制用于维护表名、列名、索引、约束等元数据。

查询优化时也要查表结构信息。

7、操作系统（OS）符号解析

内核符号表（如 Linux 的 /proc/kallsyms）用于调试内核模块。

动态库加载时查找函数符号。

总结：

