深蓝学院
shenlanxueyuan.com

C++ 基础

第 2 章：对象与基本类型

主讲人 李伟

微软高级工程师

《C++ 模板元编程实战》作者

目录

从初始化 / 赋值语句谈起

- 初始化 / 赋值语句是程序中最基本的操作，其功能是将某个值与一个对象关联起来
  - 值：字面值、对象（变量或常量）所表示的值……
  - 标识符：变量、常量、引用……
  - 初始化基本操作：
    - 在内存中开辟空间，保存相应的数值
    - 在编译器中构造符号表，将标识符与相关内存空间关联起来
  - 值与对象都有类型
  - 初始化 / 赋值可能涉及到类型转换

类型详述

- 类型是一个编译期概念，可执行文件中不存在类型的概念
- C++ 是强类型语言
- 引入类型是为了更好地描述程序，防止误用
- 类型描述了：
  - 存储所需要的尺寸 (sizeof ，标准并没有严格限制)
  - 取值空间 (std::numeric_limits ，超过范围可能产生溢出)
  - 对齐信息（ alignof ）
  - 可以执行的操作

类型详述

- 类型可以划分为基本类型与复杂类型
  - 基本（内建）类型： C++ 语言中所支持的类型
    - 数值类型
      - 字符类型（ char, wchar_t, char16_t, char32_t ）
      - 整数类型
        - 带符号整数类型： short, int, long, long long
        - 无符号整数类型： unsigned + 带符号整数类型
      - 浮点类型
        - float, double, long double
    - void
  - 复杂类型：由基本类型组合、变种所产生的类型，可能是标准库引入，或自定义类型

类型详述

- 与类型相关的标准未定义部分
  - char 是否有符号
  - 整数中内存中的保存方式：大端 小端

  - 每种类型的大小（间接影响取值范围）
    - C++11 中引入了固定尺寸的整数类型，如 int32_t

类型详述—字面值及其类型

- 字面值：在程序中直接表示为一个具体数值或字符串的值
- 每个字面值都有其类型
  - 整数字面值： 20 （十进制）， 024 （八进制）， 0x14 （十六进制） -- int 型
  - 浮点数： 1.3, 1e8 – double 型
  - 字符字面值： 'c', '\n', '\x4d' – char 型
  - 字符串字面值： "Hello" – char[6] 型
  - 布尔字面值： true, false – bool 型
  - 指针字面值： nullptr – nullptr_t 型

类型详述—字面值及其类型

- 可以为字面值引入前缀或后缀以改变其类型
  - 1.3 （double） -- 1.3f （float）
  - 2 （int） -- 2ULL (unsigned long long)
- 可以引入自定义后缀来修改字面值类型

类型详述—变量及其类型

- 变量：对应了一段存储空间，可以改变其中内容
- 变量的类型在其首次声明（定义）时指定：
  - int x：定义一个变量 x ，其类型为 int
  - 变量声明与定义的区别： extern 前缀
- 变量的初始化与赋值
  - 初始化：在构造变量之初为其赋予的初始值
    - 缺省初始化
    - 直接 / 拷贝初始化
    - 其它初始化
  - 赋值：修改变量所保存的数值

类型详述——（隐式）类型转换

- 为变量赋值时可能涉及到类型转换
  - bool 与 整数之间的转换
  - 浮点数与整数之间的类型转换
- 隐式类型转换不只发生在赋值时
  - if 判断
  - 数值比较
    - 无符号数据与带符号数据之间的比较
    - std::cmp_XXX （C++ 20）

复合类型：从指针到引用

- 指针：一种间接类型

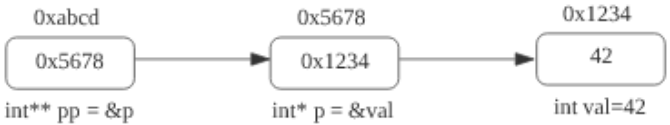
- 特点
  - 可以“指向”不同的对象
  - 具有相同的尺寸
- 相关操作
  - & – 取地址操作符
  - * – 解引用操作符

复合类型：从指针到引用

- 指针的定义
  - int* p = &val;
  - int* p = nullptr;
- 关于 nullptr
  - 一个特殊的对象（类型为 nullptr_t ），表示空指针
  - 类似于 C 中的 NULL ，但更加安全
- 指针与 bool 的隐式转换：非空指针可以转换为 true ；空指针可以转换为 false
- 指针的主要操作：解引用；增加、减少；判等
- void* 指针
  - 没有记录对象的尺寸信息，可以保存任意地址
  - 支持判等操作

复合类型：从指针到引用

- 指针的指针



- 指针 V.S. 对象
  - 指针复制成本低，读写成本高
- 指针的问题
  - 可以为空
  - 地址信息可能非法
  - 解决方案：引用

复合类型：从指针到引用

- 引用
  - int& ref = val;
  - 是对象的别名，不能绑定字面值
  - 构造时绑定对象，在其生命周期内不能绑定其它对象（赋值操作会改变对象内容）
  - 不存在空引用，但可能存在非法引用——总的来说比指针安全
  - 属于编译期概念，在底层还是通过指针实现
- 指针的引用
  - 指针是对象，因此可以定义引用
  - int* p = &val; int* & ref = p;
  - 类型信息从右向左解析

常量类型与常量表达式

- 常量与变量相对，表示不可修改的对象
  - 使用 const 声明常量对象
  - 是编译期概念，编译器利用其
    - 防止非法操作
    - 优化程序逻辑
- 常量指针与顶层常量（ top-level const ）
  - const int* p;
  - int* const p;
  - const int* const p;
  - 常量指针可指向变量

常量类型与常量表达式

- 常量引用（也可绑定变量）
  - const int&
  - 可读但不可写
  - 主要用于函数形参
  - 可以绑定字面值
- 常量表达式 （从 C++11 开始）
  - 使用 constexpr 声明
  - 声明的是编译期常量
  - 编译器可以利用其进行优化
  - 常量表达式指针： constexpr 位于 * 左侧，但表示指针是常量表达式

类型别名与类型的自动推导

- 可以为类型引入别名，从而引入特殊的含义或便于使用（如： size_t ）
- 两种引入类型别名的方式
  - typedef int MyInt;
  - using MyInt = int; （从 C++11 开始）
- 使用 using 引入类型别名更好
  - typedef char MyCharArr[4];
  - using MyCharArr = char[4];
- 类型别名与指针、引用的关系
  - 应将指针类型别名视为一个整体，在此基础上引入常量表示指针为常量的类型
  - 不能通过类型别名构造引用的引用

类型别名与类型的自动推导

- 类型的自动推导
  - 从 C++11 开始，可以通过初始化表达式自动推导对象类型
  - 自动推导类型并不意味着弱化类型，对象还是强类型
  - 自动推导的几种常见形式
    - auto：最常用的形式，但会产生类型退化
    - const auto / constexpr auto：推导出的是常量 / 常量表达式类型
    - auto&：推导出引用类型，避免类型退化
    - decltype(exp)：返回 exp 表达式的类型（左值加引用）
    - decltype(val)：返回 val 的类型
    - decltype(auto)：从 c++14 开始支持，简化 decltype 使用
    - concept auto：从 C++20 开始支持，表示一系列类型（ std::integral auto x = 3; ）

域与对象的生命周期

- 域 (scope) 表示了程序中的一部分，其中的名称有唯一的含义
- 全局域（ global scope ）：程序最外围的域，其中定义的是全局对象
- 块域（ block scope ），使用大括号所限定的域，其中定义的是局部对象
- 还存在其它的域：类域，名字空间域……
- 域可以嵌套，嵌套域中定义的名称可以隐藏外部域中定义的名称
- 对象的生命周期起始于被初始化的时刻，终止于被销毁的时刻
- 通常来说
  - 全局对象的生命周期是整个程序的运行期间
  - 局部对象生命周期起源于对象的初始化位置，终止于所在域被执行完成

深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening