第1章-C++基础

author：岳石磊 copyright：科林明伦 内部资料禁止外泄

1. 面向过程、面向对象：

C语言采用了一种有序的编程方法-结构化编程：将一个大型程序分解为一个个小型的、易于编写模块，所有的模块有序的调动起来形成了一个程序的完整的运行链。这种结构化编程反映出来过程性编程的思想，即C语言是一门面向过程的语言，更注重**程序实现逻辑**、怎么更好、更快、更直接的完成某一功能。

C语言是种面向过程编程的语言，在编写大型项目时，并不利于程序的**复用性**、**扩展性**，导致了在后期维护时带来了很多繁琐的工作，面临巨大挑战。针对于此，OOP（Object-Oriented Programming）的概念诞生了，与结构化编程不同的是，OOP更注重数据，让语言来满足问题的需求，设计出与问题本质特性相对应的数据格式。

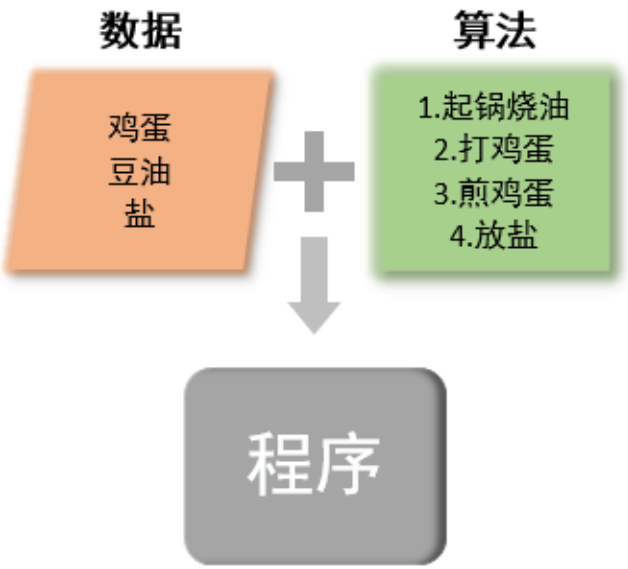
OOD（Object-Oriented Design）:面向对象的设计，OOA（Object-Oriented Analysis）：面向对象的分析

C++ 是一门面向对象编程的语言，把问题分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描述某个事物在整个解决问题的步骤中的行为,更注重的是**程序的整体设计**，方便程序后期维护、优化和管理，让一个功能尽可能的通用。面向对象编程只有一个价值：**应对需求的变化，本意是要处理大型复杂系统的设计和实现。**
常说的面向过程和面向对象，其本质还是在其**设计思想**上。

面向过程优点：性能比面向对象高，比如单片机、嵌入式开发、 Linux/Unix等一般采用面向过程开发，性能是最重要的因素。
缺点：没有面向对象易维护、易复用、易扩展。

面向对象语言优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护。
缺点：因为类调用时需要实例化，开销比较大，比较消耗资源，性能比面向过程低。

C++是由C衍生出来的一门语言，不但兼容包含了C语言还增加了一些新特性函数重载，类、继承、多态，支持泛型编程（模板函数、模板类），强大的STL库等。
面向对象的三大特性：封装、继承、多态。



概念介绍：

1. 封装 ： 将零散的数据和算法放到一个集合里，方便管理和使用。
2. 复用性：公共功能、过程的抽象，体现为能被重复使用的类、方法，就要求我们针对某一类功能而不是针对某一个功能去设计。
3. 扩展性：增加新的功能不影响原来已经封装好的功能。

2. 输入、输出

C++ 中的输出使用cout对象，输入使用cin对象，他是定义在文件 iostream 命名空间std中的ostream和istream类对象，此文件已经默认包含了stdio.h文件，所以仍然可以使用C中的printf，scanf等。

```
_STD_BEGIN
  ...
__PURE_APPDOMAIN_GLOBAL extern _CRTDATA2 istream cin, *_Ptr_cin;
__PURE_APPDOMAIN_GLOBAL extern _CRTDATA2 ostream cout, *_Ptr_cout;
...
```

使用输出、输入两步走：1. 包含头文件。2. 打开标准命名空间std。常用写法如下：

```
#include<iostream>     //引入文件，已经包含了#include<stdio.h>
using namespace std;   //打开标准命名空间 std
```

cout输出需要结合<< 输出操作符一起使用，对于内置的基本类型来说，可直接输出。
输出时换行使用endl，其本质上是函数，作用是插入换行符并刷新输出流。
cin 输入需要结合>>输入操作符一起使用，对于内置的基本类型来说，可直接输入，但与scanf 不同的是，不需要对变量取地址(&)。

```cpp
#include<iostream>    //1. 包含对应的头文件
using namespace std;  //2. 打开std标准命名空间

int  a = 0;
char b = 'b';
cin>>a>>b;   //不需要使用&取地址符号
cout<<a<<"   "<<b<<endl;
```

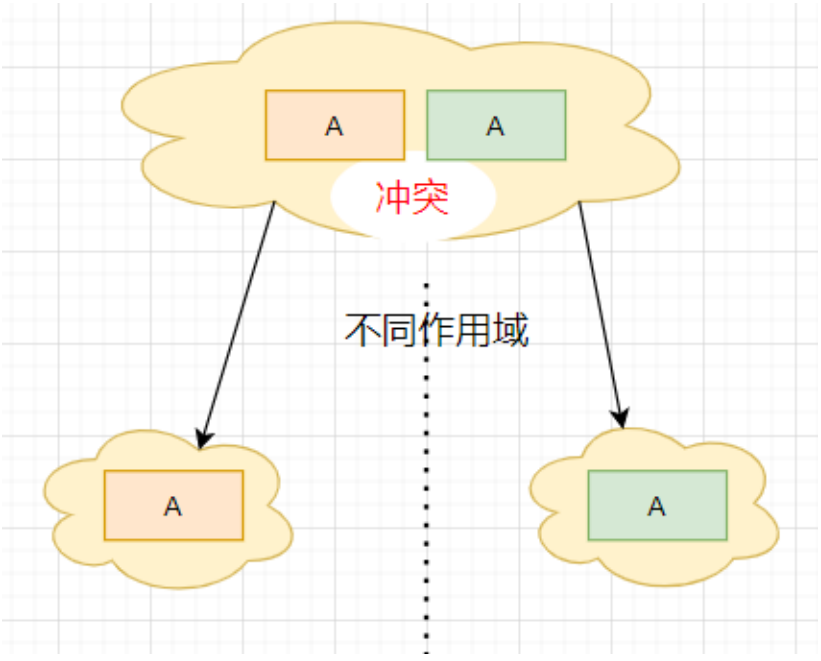
总结C和C++输入输出：C中的格式化输出更加灵活写，但是写起来相对复杂一些，C++中的cin、cout简单易用，根据不同的场景选择合适的输入输出方法。

3. 命名空间

作用域运算符::，作用域::成员指明了成员的归属范围，如果 :: 前不加任何作用域，代表取全局作用域。常用的作用域有命名空间、结构体、类等。

```cpp
int A = 10;
void fun() {
    int A = 1;
    cout << A << endl;    //1   局部
    cout << ::A << endl;  //10  全局
}
```

对于全局和局部作用域可以定义相同的成员，但如果是同一个作用域中存在相同的成员，则会报重定义的错误。可以使用命名空间去区分。



namespace： C++ 中的关键字，用于定义命名空间，命名空间主要用于区分同一个作用域下的相同成员。定义命名空间：

```
namespace 命名空间名
{
    变量...
    函数...
    结构体...
};
```

使用命名空间

1. using 编译指令打开命名空间，`using namespace 命名空间`，这是一种懒惰的做法，打开一次里面的成员将全部对外开放。打开了多个命名空间如果成员名相同会出现二义性。
2. 显示指定命名空间成员，**命名空间::成员**，这种方式简单直接，如：`std::cout<<10;`，但是每次使用的时候都需要指明命名空间。

4. 动态申请空间

在C中使用`malloc`，`free`来动态申请、释放堆区内存空间，使用时需要显式的指定申请空间的大小。而在C++中使用两个关键字：`new`，`delete` 来动态申请、释放空间，同样操作的是**堆区**，new 申请空间不需要指定空间大小，而是指定类型，根据类型自动计算所需要的空间大小，并返回申请内存空间的地址，地址指向的类型为`new` 后写的类型，格式：`type * pointer = new type`，例：

```
int *p1 =new int;
int *p2 =new int(10);  //申请空间并初始化
```

new 数组返回的是首元素的首地址。

```
int *p1 =new int[10];
int *p2 =new int[10]();  //新建int类型数组并初始化，每个元素为0
char *p3 =new char[10](); //新建char类型数组并初始化，每个元素为空字符'\0'
```

对于数组空间回收，在指针前一般加上`[]`，代表回收整个数组空间。

```
delete []p1;
delete []p2;
delete []p3;
```

delete 回收空间并不包含指针本身，而是指针指向的内存空间，同一块内存空间不要重复释放，除非指针已经被赋空，对空指针使用delete是安全的。对栈区的内存空间不能使用delete来释放。

两者的区别：

1. new、delete是关键字 需要C++的编译器支持，malloc()、free() 是函数，需要头文件支持。

2. new 申请空间不需要指定申请大小，根据类型自动计算，new 返回的是申请类型的地址，不需要强转，malloc() 需要显式的指定申请空间的大小（字节），返回void*，需要强转成我们需要的类型。
3. new 申请空间的同时可以设置初始化，而malloc 需要手动赋值。
4. 至关重要的一点：new **申请类、结构体对象内存空间会自动调用构造函数**，delete **会自动调用类、结构体的析构函数，单独的**malloc() **和**free() **则不会调用构造、析构函数。**

5. bool

BOOL 类型在定义在minwindef.h中，我们使用的时候通常只需要包含#include<windows.h>即可，这是window系统给我们提供的，并不是C++自带的类型。它是对int类型起的一个别名。

```
typedef int                 BOOL;
#define FALSE               0
#define TRUE                1
```

所以 BOOL类型的变量占用4个字节的内存空间。
C++中也提供了一种bool类型，关键字，占用一个字节，对应的真假为 true、false。

6. string

string 是 C++提供的字符串存储、操作的类需要包含头文件并打开std命名空间。

```
#include <string>
using namespace std;
```

其天然支持字符串的初始化、赋值、比较、拼接、下标访问等操作。
字符串的截取需要用函数substr，并返回截取后的字符串。

```
_Myt substr(size_type _Off = 0,//字符串截取的下标，从0开始,如果下标超限则访问越界
size_type _Count = npos)  //截取的长度，如果长度超限制，则取有效长度
```

size()，length() 则返回字符串的长度（字符数量）。
将string 类型转换为字符指针，用：

```
const char *c_str()
```

7. 增强的范围for

通常我们遍历数组时，常用的写法for(;;)，新标准允许了下面简化的写法：

```
type arr[n];
for(type val:arr)  //== type val = arr[i]
```

增强的for循环这种遍历方式，适用于普通的数组、string、和支持 begin , end操作的容器等。

8. 函数重载

C++中允许函数的参数列表指定默认值，而且这个默认值必须 从右向左依次指定，不能间断，一般在函数的声明中去指定，在函数的定义中指定编译可能不会报错，但多数情况下是毫无意义的操作。

```
void fun(int a, int b= 30);  //函数声明

void fun(int a, int b ) {    //函数定义
    cout << a << "  "<<b << endl;
}
```

函数重载：在同一个作用域，函数名一样，参数列表不同（参数类型、数量、顺序等不同）。注意函数重载描述的是多个函数之间的关系。

```
void fun();
void fun(int ,char) ;
void fun(int ,int ) ;
void fun(int, int,int) ;
```

函数的重载使得 C++ 程序员对完成类似功能的不同函数可以统一命名，减少了命名所花的心思。在调用函数时，编译器会根据实参的类型、数量自动匹配对应类型函数。

9.nullptr

我们在使用指针的时候，要尽量避免野指针的存在，即定义了就要初始化，如果暂时不知道具体指向哪块空间，那么可以初始化为空，我们有两种方式 1. NULL 和 nullptr (C++11 新标准引入的)，

```
int *p1 = NULL;
```

在C++中 NULL 被如下定义：

```
//stio.h
/* Define NULL pointer value */
#ifndef NULL
```

```
#ifdef __cplusplus
#define NULL    0
```

我们看到NULL它是一个宏，在C++中被明确定义为整数0，这就导致了整型与指针的混用，nullptr的出现解决了这个问题，它明确了自己的含义，空指针。
nullptr C++11 新标准引入的一个关键字，代表是空指针，如果出现了多个类型的空指针，仍然会语义不明确，则需要使用强转来明确空指针的类型。
C++中 NULL 和 nullptr 的区别：

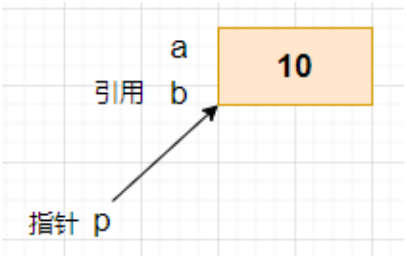
1. NULL 是宏 替换的是0，nullptr 是关键字。
2. 含义不同，nullptr 代表是空指针，NULL 代表整型数字。

10. 引用

引用：使用&符号定义引用，给某个变量起别名，定义引用就要初始化，

```
int a=10;
int &b = a;
```

引用与被引用的变量指的是同一块内存空间，对引用操作就是对原变量进行操作。
引用不是值不占用内存空间，且引用一旦被初始化，就不能再重新引用其他空间。



函数传参的三种方式：值传递、地址传递、引用传递。
对于一些基本的数据类型值传递问题不大，但是对于一些复合类型来说，参数中的局部变量会占用空间（虽然调用函数结束空间会自动被回收），对于结构体、类来说会调用构造、析构，可能出现浅拷贝的问题，所以复合类型强烈不建议使用值传递。如果想要在函数内修改外面的实参的值，则使用引用和指针。

引用和指针的区别：

1. 引用定义了就要初始化，指针可不用初始化（但不推荐）。
2. 引用初始化后就不能再重新引用其他的变量、空间，指针可以随意指向。
3. 有NULL的指针，没有NULL的引用。
4. 指针变量额外占用空间，引用不额外申请空间。
5. 指针可以有多级，但引用只能有一级。