



# C++ 基础

## 第 5 章：语句

主讲人 李伟

微软高级工程师

《C++ 模板元编程实战》作者





# 目录



1. 语句基础



2. 分支语句



3. 循环语句



4. 语句的综合应用——达夫设备



## 语句基础

- 语句的常见类别 语句刻画了程序执行的过程  
表达式的侧重点在于求值
  - 表达式语句：表达式后加分号，**对表达式求值后丢弃**，可能产生副作用
  - 空语句：仅包含一个分号的语句，可能与循环一起工作
  - 复合语句（语句体）：由大括号组成，无需在结尾加分号，形成独立的域（语句域）  
无需在大括号外面加分号
- 顺序语句与非顺序语句
  - 顺序语句
    - 从语义上按照先后顺序执行
    - 实际的执行顺序可能产生变化（**编译器优化、硬件乱序执行**）感兴趣可以查一下乱序执行
    - 与硬件流水线紧密结合，执行效率较高

- 非顺序语句 执行效率相对较低

- 在执行过程中引入跳转，从而产生复杂的变化
- 分支预测错误可能导致执行性能降低

```
// branch
// true: statement2
// false: statement3
```

在执行的时候可能会猜未来会执行哪个语句。  
比如说当branch语句还在执行的时候，就猜可能会运行statement2。如果branch是true，那么就会加速程序运行。如果猜错了，就会使执行性能降低



## 语句基础（续）

- 最基本的非顺序语句：goto

- 通过标签指定跳转到的位置

- 具有若干限制

- 不能跨函数跳转

- 向前跳转时不能越过对象初始化语句

- 向后跳转可能会导致对象销毁与重新初始化

- goto 本质上对应了汇编语言中的跳转指令

- 缺乏结构性的含义

- 容易造成逻辑混乱

- 除特殊情况外，应避免使用

虽然goto有问题，但是c++并没有取消这个关键字。对于一些特殊情况，还是可以用。

```
int main()
{
    int x = 3;
    goto label;
    int y = 4;
label:
    y = y + 1;
}
```

向前跳转不能越过对象初始化语句

```
#include <cstdio>

int main()
{
    int x = 3;
    if (x) goto label;
    x = x + 1;
label:
    return 0;
}
```

```
1 main:
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], 3
5     cmp     DWORD PTR [rbp-4], 0
6     jne     .L5
7     add     DWORD PTR [rbp-4], 1
8     jmp     .L3
9 .L5:
10    nop
11 .L3:
12    mov     eax, 0
13    pop     rbp
14    ret
```

在汇编语句中出现jmp和jne，则证明出现了跳转

```
int main()
{
    bool flag = true;
label:
    int x = 3;
    if (flag)
    {
        flag = false;
        goto label;
    }
}
```

backward 会导致对象销毁与重新初始化



## 分支语句——if

- 语法: <https://zh.cppreference.com/w/cpp/language/if>
- 使用语句块表示复杂的分支逻辑
- 从 if 到 if-else
  - 实现多重分支
  - else 会与最近的 if 匹配
  - 使用大括号改变匹配规则
- if V.S. constexpr if—— 运行期与编译期分支
- 带初始化语句的 if

```
// grade > 80 -- > Excellent
// grade <= 60 -- > Bad
|
int grade = 65;
if (grade > 60)
    if (grade > 80)
        std::cout << "Excellent\n";
else
    std::cout << "Bad\n";
```



# 分支语句— switch

在程序中加入  
[[fallthrough]]可以使程序  
取消warning的报错

```

switch (std::cin >> x; x)
{
    default:
        std::cout << "China\n";
        [[fallthrough]];
    case 3:
        std::cout << "Hello\n";
        break;
    case 4:
    case 5:
        std::cout << "World\n";
        break;
}

```

- 语法: <https://zh.cppreference.com/w/cpp/language/switch>
- 条件部分应当能够隐式转换为整形或枚举类型, 可以包含初始化的语句
- case/default 标签

常量表达式: 在编译期可以求值的表达式

匹配执行后续所有的代码这个概念叫做 fall through

- case 后面跟常量表达式, 用于匹配 switch 中的条件, 匹配时执行后续的代码

- 可以使用 break 跳出当前的 switch 执行

default 放到最后一个分支完全是大家的书写习惯。你完全可以把这个分支放到前面去。不过如果没有写break的话, 会执行后续的代码

- default 用于定义缺省情况下的逻辑

- 在 case/default 中定义对象要加太括号

- [[fallthrough]] 属性

fallthrough是c++17的特性

- 与 if 相比的优劣

缺点: 分支描述能力较弱

- 在一些情况下能引入更好的优化  
这里有链接可以点

switch在编译的时候可以做更好的优化。  
例如调表, 二元优化 会更快的选择出想去的case

```

switch (std::cin >> x; x)
{
    default:
        std::cout << "China\n";
    case 3:
        int y = 4;
        std::cout << "Hello\n";
    case 4:
        std::cout << "world\n";
        break;
}

```

// 程序会报错。因为case3中定义了一个变量。使得系统不能跳转到case 4。那么会导致Y这个变量的生命周期不确定。因此c++不允许在case中定义变量

```

switch(x)
{
    case 3:
        {
            int y = 4;
            break;
        }
    case 4:
        std::cout <<"";
}
// 可以, 用大括号确定y的生命周期

```

```

switch (std::cin >> x; x)
{
    default:
        std::cout << "China\n";
        break;
    case 3:
        {
            std::cout << "Hello\n";
            break;
        }
    case 4:
    case 5:
        std::cout << "World\n";
        break;
}

```

如果我们想case 4和case5输出相同的逻辑



## 循环语句—— while

- 语法: <https://zh.cppreference.com/w/cpp/language/while>
- 处理逻辑:
  - 1. 判断条件是否满足, 如果不满足则跳出循环
  - 2. 如果条件满足则执行循环体
  - 3. 执行完循环体后转向步骤 1
- 注意: 在 while 的条件部分不包含额外的初始化内容 包含额外初始化内容就可转化为for 循环



## 循环语句—— do-while

- 语法: <https://zh.cppreference.com/w/cpp/language/do>
  - 注意结尾处要有分号, 表示一条语句的结束
- 处理逻辑:
  - 1. 执行循环体
  - 2. 断条件是否满足, 如果不满足则跳出循环
  - 3. 如果条件满足则转向步骤 1





## 循环语句—— for

- 语法: <https://zh.cppreference.com/w/cpp/language/for>
- 处理逻辑
  - 1. 初始化语句会被首先执行
  - 2. 条件部分会被执行, 执行结果如果为 false , 则终止循环
  - 3. 否则执行循环体
  - 4. 迭代表达式会被求值, 之后转向 2
- 在初始化语句中声明多个名字
- 初始化语句、条件、迭代表达式可以为空

- for 的更多示例 for 循环的官方示例很不错

```
std::vector<int> v = {3, 1, 4, 1, 5, 9}
for (auto iter = v.begin(); iter != v.end(); ++iter) {
    std::cout << *iter << " ";
}
std::cout << "\n";
```



## 循环语句——基于范围的 for 循环

- 语法: <https://zh.cppreference.com/w/cpp/language/range-for>
- 本质: 语法糖, 编译器会转换为 for 循环的调用方式
- 转换形式的衍化: C++11 / C++17 / C++20
- 使用常量左值引用读元素; 使用“万能引用 (universal reference)”修改元素

auto&& v: 万能引用

```
std::vector<std::string> arr{"h", "e", "l"};
for (std::string v : arr)
    std::cout << v << '\n';
```

这个代码并不好, 我们只对v进行读操作。可是实际上我们构造了一个对象 (如右图), 进行了对象的拷贝, 在for循环结束后, 还有对象的析构。这样是十分耗时的。

```
std::vector<std::string> arr{"h", "e", "l"};
for (const std::string& v : arr)
    std::cout << v << '\n';
```

可简写为右边的形势

```
std::vector<std::string> arr{"h", "e", "l"};
for (const auto& v : arr)
    std::cout << v << '\n';
```

```
std::vector<int> arr{1, 2, 3};
for (auto& v : arr)
    v = v + 1;
```

不会引入新的中间变量, 直接对v操作。  
下面是万能引用

```
std::vector<bool> arr{true, false, true};
for (auto&& v : arr)
    v = false;
```

```
std::vector<std::string> arr = std::vector<std::basic_string<char>, std::allocator<
{
    std::vector<std::basic_string<char>, std::allocator<std::basic_string<char> > > &
    __gnu_cxx::__normal_iterator<std::basic_string<char> *, std::vector<std::basic_st
    __gnu_cxx::__normal_iterator<std::basic_string<char> *, std::vector<std::basic_st
    for (; __gnu_cxx::operator!=(__begin1, __end1); __begin1.operator++()) {
        std::string v = std::basic_string<char>(__begin1.operator*());
        std::operator<<(std::operator<<(std::cout, v), '\n');
```



## 循环语句—— break / continue

- 含义（转自 cpp reference ）
  - break: 导致外围的 for 、 范围 for 、 while 或 do-while 循环或 switch 语句终止
  - continue: 用于跳过整个 for 、 while 或 do-while 循环体的剩余部分。
- 注意这二者均不能用于多重嵌套循环，多重嵌套循环的跳转可考虑 goto 语句

```
int main(void)
{
    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 5; k++) {
            if (k == 2) goto label;           //只有此循环会受到 break 的影响
            std::cout << j << k << " ";
        }
    }
label:
    ;
}
```



## 语句的综合应用——达夫设备

- 使用循环展开提升系统性能
- 处理无法整除的情形
  - 额外增加一个循环语句
  - 将 switch 与循环结合——达夫设备

感谢聆听 !  
Thanks for Listening

