5_引用的本质分析

- 引用作为 `变量别名` 而存在，因此可以在一些场合 `代替指针`
- 引用 `相对于指针` 来说具有更好的可读性和实用性
- 引用被声明的同时必须被初始化，但在函数中的作为形参是不需要进行初始化，这种情况是在函数调用时被初始化
- 指针版交换两个变量的值
- 引用版交换两个变量的值

特殊的引用

- const 引用
  - 在 C++中可以声明 `const` 引用
  - 语法：`const Type& name = var;`
  - `const` 引用让变量拥有 `只读属性`
  - 当使用 **常量**对 `const` 引用进行初始化时，C++编译器会为常量值分配空间，并将引用名作为这段空间的别名

    > 使用常量对 `const` 引用初始化后，将生成一个只读变量。

  - example lesson_5-2.cpp
    输出

    ```
    Example:
    a = 5
    b = 5
    Demo
    c = 1
    c = 5
    ```

    > 想让一个已经存在的变量只拥有只读属性，可以定义这个变量的 `const引用` 即可

  - 引用是否拥有自己的存储空间?
    - example lesson_5-3.cpp
      输出

      ```
      sizeof(char&) = 1
      sizeof(rc) = 1
      sizeof(TRef) = 8
      ```

```
4 | sizeof(ref.r) = 1
```

- 引用在C++中的 `内部实现` 是一个 `指针常量`

> `Type &name` 等价于 `Type * const name`
>
> 注意：
>
> i. C++编译器在编译过程中用 `指针常量` 作为引用的 `内部实现`，因此 `引用所占用的空间大小与指针相同`
>
> ii. 从使用的角度，`引用是一个别名`，C++为了实用性而隐藏了引用的存储空间这一细节

  - example lesson_5-4.cpp
    输出

```
1 | sizeof(r) = 24
2 | sizeof(r.before) = 8
3 | sizeof(r.after) = 8
4 | &r.before = 0x7ffe80120a20
5 | &r.after = 0x7ffe80120a30
```

> 从以上代码的输出，证明引用的占用的空间是与指针一样。

  - C++引用在大多数情况下 代替指针
    - 功能性：可以满足多数需要使用指针的场合
    - 安全性：可以避开由于指针操作不当而带来的内存错误
    - 操作性：简单易用，又不失功能强大
  - 函数返回引用

    example lesson_5-5.cpp
    输出

```
demo: d = 3
func: a = 4

Segmentation fault (core dumped)
```

> 被调函数返回引用(局部变量的地址)，然后通过这个引用去访问刚释放的堆栈空间以致于崩溃

小结

- 引用作为 变量别名 而存在旨在 `代替指针`
- `const` 引用可以使得 `变量具有只读属性`
- `引用` 在编译器内部使用 **指针常量实现**
- `引用` 的最终 `本质为指针`
- 引用可以尽可能的避开内存错误