

# 第十一部分 引用及拷贝构造函数

## 1. 引用 → 给变量取别名

引用类型 多出现于函数的参数列表和 返回类型，使得被引用的变量在函数内如同指针传递般被更改，但更简洁和更安全

本质上，引用类型在参数的传递时传递的是参数所在的内存地址，并将该内存地址借给函数内形参进行改动。

※

定义在函数内的对象(局部对象)由于其访问生命周期的限制，无法从引用类型传至主函数(已消亡)

## 2. 拷贝构造函数

对于同一类型的对象，可以相互之间借用进行初始化。

属性  
直接  
复制

可缺省，  
直接拷贝

核心是调用了  
拷贝构造函数

```
Class Sample {  
    Sample S1;  
    Sample S2(S1), S3(S2);  
    Sample S4=S3;  
};
```

语法: `Sample(const Sample& p);`

即要求参数为同类型的常态引用，其他不变

拷贝构造函数的被调用场景

不以引用形式传入

① 用同类对象初始化时

② 将函数的参数设定为类对象时

和上

函数内局部对象将进行同类初始化

③ 当函数的返回类型为对象时, 由于该对象为局部对象, 无法传回主函数内, 因此会使用拷贝构造函数在主函数内创建一个匿名对象作为返回值

在 C++ 中使用类名(参数) 会创建匿名对象

### 3. 右值引用 & 移动构造函数 C++11 新特性

① 左值? 右值?

左值: 表达数据的表达式, 可取地址, 可赋值, 出现在赋值号左侧

右值: 表达数据的表达式, 不可取地址, 不可赋值, 出现在赋值号右侧

```
int a = 10;
int b = f(x, y);
int c = x + y;
```

左值      右值

这两个类型也称将亡值

可以使用函数 `std::move` (左值) 将左值转化为右值

② 左值引用? 右值引用?

左值引用: 给左值设定别名, 要求只能引用左值

```
int a = 10;    int& b = a;
```

右值引用 = 给右值设定别名, 要求只能引用右值

`int a=10; int&& b=a;` 错, 引用左值

`int&& b=10;` 对

※ 特别的 `const` 类的左值引用既可以引用左值, 也可以引用右值  
[因为不可赋值, 与右值类似]

`const int& a=10;`

### ⑤ 移动构造函数

类似于拷贝构造函数, 但

拷贝  
构造

`sample (const sample& S)`  $\Rightarrow$  深拷贝  
常左值引用

移动  
构造

`sample (sample&& S)`  $\Rightarrow$  资源交换  
右值引用

在拷贝构造中, 传入的为左值的常引用, 由于该参数对象 `S` 往往生命周期还未终止, 因而要对其成员变量进行拷贝

而在移动构造中, 传入的为右值引用, 该参数 `S` 通常为将亡值, 因而总可以将其成员变量直接移动给待构造对象

## 将this与将亡值交换

The diagram illustrates the difference between copy construction and move construction in C++ using a `string` function and a `main` function.

**Left Side (Copy Construction):**

```
string to_string(int value)
{
    bool flag = true;
    if (value < 0)
    {
        flag = false;
        value = 0 - value;
    }
    //...
    mj:string str;
    std::reverse(str.begin(), str.end());
    return str;
}
```

`int main()`

```
{
    bit::string ret = bit::to_string(-1234567)
}
```

Annotations for the left side:

- 编译器进行优化** (Compiler optimization) points to the `std::reverse` call.
- 1、直接构造** (1. Direct construction) points to the `std::reverse` call.
- 2、str识别成右值** (2. str identified as rvalue) points to the `str` variable.
- 移动构造** (Move construction) points to the `ret` variable in `main`.
- 将亡值** (Rvalue) points to the `str` variable in `to_string`.
- 匿名对象** (Anonymous object) points to the `str` variable in `to_string`.

**Right Side (Move Construction):**

```
string to_string(int value)
{
    bool flag = true;
    if (value < 0)
    {
        flag = false;
        value = 0 - value;
    }
    //
    mj:string str;
    std::reverse(str.begin(), str.end());
    return str;
}
```

`int main()`

```
{
    bit::string ret = bit::to_string(-1234567)
}
```

Annotation for the right side:

- 移动构造** (Move construction) points to the `ret` variable in `main`.

在有了移动构造以后，再经过编译器的优化，就可以做到直接移动构造（资源的交换），实现0拷贝，效率极高！！