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，无法以引用类型传至主函数(已淘汰)

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

① 用同类对象初始化时

② 将函数的参数设定为类对象时 ← 不以引用形式传入

函数内局部对象进行同类初始化

① 当函数的返回类型为对象时，由于该对象为局部对象，生命周期很短无法传到主函数内，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cpp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 cpp11新特性

① 左值？右值？

左值：表达数据的表达式，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;
```

a 左值　10 右值

f(x,y) 这两个类型也称为右值

※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;` 错误，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;`

① 移动构造函数

类似于拷贝构造函数，但

拷贝构造 `sample(const sample& S)` ⇒ 深拷贝

常左值引用

移动构造 `sample(sample&& S)` ⇒ 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任经生命周期还未终止，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

```
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;
}
```

拷贝构造

将亡值 → 匿名对象

编译器进行优化

1、直接构造

2、str识别成右值

移动构造

```
int main()
{
    bit::string ret = bit::to_string(-1234567);
```

```
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;
}
```

移动构造

```
int main()
{
    bit::string ret = bit::to_string(-1234567);
```

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！！