# C++11 move()函数：将左值强制转换为右值

通过学习《C++11移动构造函数》一节我们知道，C++11 标准中借助右值引用可以为指定类添加移动构造函数，这样当使用该类的右值对象（可以理解为临时对象）初始化同类对象时，编译器会优先选择移动构造函数。

注意，移动构造函数的调用时机是：用同类的右值对象初始化新对象。那么，用当前类的左值对象（有名称，能获取其存储地址的实例对象）初始化同类对象时，是否就无法调用移动构造函数了呢？当然不是，C++11 标准中已经给出了解决方案，即调用 move() 函数。

move 本意为 "移动"，但该函数并不能移动任何数据，它的功能很简单，就是将某个左值强制转化为右值。

> 基于 move() 函数特殊的功能，其常用于实现移动语义。

move() 函数的用法也很简单，其语法格式如下：

```
move( arg )
```

其中，arg 表示指定的左值对象。该函数会返回 arg 对象的右值形式。

【例 1】move() 函数的基础应用。

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

class movedemo{
public:
    movedemo():num(new int(0)){
        cout<<"construct!"<<endl;
    }
    //拷贝构造函数
    movedemo(const movedemo &d):num(new int(*d.num)){
        cout<<"copy construct!"<<endl;
    }
    //移动构造函数
    movedemo(movedemo &&d):num(d.num){
        d.num = NULL;
        cout<<"move construct!"<<endl;
    }
public:     //这里应该是 private，使用 public 是为了更方便说明问题
    int *num;
};
```

```
21.
22.  int main(){
23.      movedemo demo;
24.      cout << "demo2:\n";
25.      movedemo demo2 = demo;
26.      //cout << *demo2.num << endl;    //可以执行
27.      cout << "demo3:\n";
28.      movedemo demo3 = std::move(demo);
29.      //此时 demo.num = NULL，因此下面代码会报运行时错误
30.      //cout << *demo.num << endl;
31.      return 0;
32.  }
```

程序执行结果为：

```
construct!
demo2:
copy construct!
demo3:
move construct!
```

通过观察程序的输出结果，以及对比 demo2 和 demo3 初始化操作不难得知，demo 对象作为左值，直接用于初始化 demo2 对象，其底层调用的是拷贝构造函数；而通过调用 move() 函数可以得到 demo 对象的右值形式，用其初始化 demo3 对象，编译器会优先调用移动构造函数。

> 注意，调用拷贝构造函数，并不影响 demo 对象，但如果调用移动构造函数，由于函数内部会重置 demo.num 指针的指向为 NULL，所以程序中第 30 行代码会导致程序运行时发生错误。

【例 2】灵活使用 move() 函数。

```
01.  #include <iostream>
02.  using namespace std;
03.
04.  class first {
05.  public:
06.      first() :num(new int(0)) {
07.          cout << "construct!" << endl;
08.      }
09.      //移动构造函数
10.      first(first &&d) :num(d.num) {
11.          d.num = NULL;
12.          cout << "first move construct!" << endl;
```

```
13.     }
14. public:    //这里应该是 private，使用 public 是为了更方便说明问题
15.     int *num;
16. };
17.
18. class second {
19. public:
20.     second() :fir() {}
21.     //用 first 类的移动构造函数初始化 fir
22.     second(second && sec) :fir(move(sec.fir)) {
23.         cout << "second move construct" << endl;
24.     }
25. public:    //这里也应该是 private，使用 public 是为了更方便说明问题
26.     first fir;
27. };
28.
29. int main() {
30.     second oth;
31.     second oth2 = move(oth);
32.     //cout << *oth.fir.num << endl;   //程序报运行时错误
33.     return 0;
34. }
```

程序执行结果为：

```
construct!
first move construct!
second move construct
```

程序中分别构建了 first 和 second 这 2 个类，其中 second 类中包含一个 first 类对象。如果读者仔细观察不难发现，程序中使用了 2 此 move() 函数：

- 程序第 31 行：由于 oth 为左值，如果想调用移动构造函数为 oth2 初始化，需先利用 move() 函数生成一个 oth 的右值版本；
- 程序第 22 行：oth 对象内部还包含一个 first 类对象，对于 oth.fir 来说，其也是一个左值，所以在初始化 oth.fir 时，还需要再调用一次 move() 函数。