移动构造函数

类 T 的移动构造函数是非模板构造函数，其首个形参是 `T&&`、`const T&&`、`volatile T&&` 或 `const volatile T&&`，且无其他形参，或剩余形参均有默认值。

语法

类名 (*类名* &&)	(1)	(C++11 起)
class_name (*类名* &&) = `default;`	(2)	(C++11 起)
class_name (*类名* &&) = `delete;`	(3)	(C++11 起)

其中 *类名* 必须指名当前类（或类模板的当前实例化），或在命名空间作用域或友元声明中声明时，必须是有限定的类名。

解释

1. 移动构造函数的典型声明。
2. 强制编译器生成移动构造函数。
3. 避免隐式移动构造函数。

当（以直接初始化或复制初始化）从同类型的右值（亡值或纯右值） (C++17 前)亡值 (C++17 起)初始化对象时，调用移动构造函数，情况包括

- 初始化：`T a = std::move(b);` 或 `T a(std::move(b));`，其中 b 类型为 T；
- 函数实参传递：`f(std::move(a));`，其中 a 类型为 T 而 f 为 `Ret f(T t)`；
- 函数返回：在如 `T f()` 的函数中的 `return a;`，其中 a 类型为 T，它有移动构造函数。

当初始化器为纯右值时，通常会优化掉 (C++17 前)始终不会进行 (C++17 起)对移动构造函数的调用，见复制消除。

典型的移动构造函数“窃取”实参曾保有的资源（例如指向动态分配对象的指针，文件描述符，TCP socket，I/O 流，运行的线程，等等），而非复制它们，并使其实参遗留于某个合法但不确定的状态。例如，从 `std::string` 或从 `std::vector` 移动可以导致实参被置为空。但是不应依赖此行为。对于某些类型，例如 `std::unique_ptr`，移动后的状态是完全指定的。

隐式声明的移动构造函数

若不对类类型（`struct`、`class` 或 `union`）提供任何用户定义的移动构造函数，且下列各项均为真：

- 没有用户声明的复制构造函数；
- 没有用户声明的复制赋值运算符；
- 没有用户声明的移动赋值运算符；
- 没有用户声明的析构函数；
- 隐式声明的移动构造函数并未因为下一节所详述的条件而被定义为*弃置的*， (C++14 前)

则编译器将声明一个移动构造函数，作为其类的非 explicit 的 `inline public` 成员，签名为 `T::T(T&&)`。

类可以拥有多个移动构造函数，例如 `T::T(const T&&)` 和 `T::T(T&&)`。当存在用户定义的移动构造函数时，用户仍可用关键词 `default` 强制编译器生成隐式声明的移动构造函数。

隐式声明（或在其首个声明被预置）的移动构造函数，具有动态异常说明 (C++17 前)异常说明 (C++17 起)中所描述的异常说明。

弃置的隐式声明的移动构造函数

若下列任何一项为真，则类 T 的隐式声明或预置的移动构造函数被定义为*弃置的*：

- T 拥有无法移动（拥有被弃置、不可访问或有歧义的移动构造函数）的非静态数据成员；
- T 拥有无法移动（拥有被弃置、不可访问或有歧义的移动构造函数）的直接或虚基类；
- T 拥有带被弃置或不可访问的析构函数的直接或虚基类；
- T 是联合式的类，且拥有带非平凡移动构造函数的变体成员；

- T 拥有非静态数据成员或直接或虚基类，它无法平凡复制且没有移动构造函数。 (C++14 前)

重载决议忽略被弃置的隐式声明的移动构造函数（否则它会阻止从右值复制初始化）。 (C++14 起)

平凡移动构造函数

当下列各项全部为真时，类 T 移动构造函数为平凡的：

- 它不是用户提供的（即它是隐式定义或预置的）；
- T 没有虚成员函数；
- T 没有虚基类；
- 为 T 的每个直接基类选择的移动构造函数都是平凡的；
- 为 T 的每个类类型（或类类型数组）的非静态成员选择的移动构造函数都是平凡的；
- T 没有 volatile 限定类型的非静态数据成员。 (C++14 起)

平凡移动构造函数是与平凡复制构造函数实施相同动作的构造函数，即它如同用 `std::memmove` 来进行对象表示的复制。所有与 C 兼容的数据类型（POD 类型）均为可平凡移动的。

隐式定义的移动构造函数

若隐式声明的移动构造函数既未弃置亦非平凡，则当其被 ODR 式使用时，它为编译器所定义（生成并编译函数体）。对于 `union` 类型，隐式定义的移动构造函数（如同以 `std::memmove`）复制其对象表示。对于非联合类类型（`class` 与 `struct`），移动构造函数用以亡值实参执行的直接初始化，按照初始化顺序，对对象的各基类和非静态成员进行完整的逐对象移动。若它满足对于 constexpr 构造函数的要求，则生成的移动构造函数为 `constexpr`。

注解

为使强异常保证可行，用户定义的移动构造函数不应抛出异常。例如，`std::vector` 在需要重新放置元素时，基于 `std::move_if_noexcept` 在移动和复制之间选择。

若一同提供了复制和移动构造函数而没有其他可行的构造函数，则当实参是相同类型的*右值*（如 `std::move` 的结果的*亡值*，或如无名临时量的*纯右值*）时，重载决议选择移动构造函数，而当实参是*左值*（具名对象或返回左值引用的函数/运算符）时，重载决议选择复制构造函数。若只提供复制构造函数，则所有实参类别都选择它（只要它接收到 const 的引用，因为右值能绑定到 const 引用），这使得在移动不可用时，以复制为移动的后备。

当接收右值引用为其形参时，构造函数被称作‘移动构造函数’。它没有义务移动任何内容，不要求类拥有要被移动的资源，而且在受允许（但可能没意义）的以 const 右值引用（const T&&）为形参的情况中，‘移动构造函数’可能无法移动资源。

示例

运行此代码

```
#include <string>
#include <iostream>
#include <iomanip>
#include <utility>

struct A
{
    std::string s;
    int k;
    A() : s("test"), k(-1) { }
    A(const A& o) : s(o.s), k(o.k) { std::cout << "move failed!\n"; }
    A(A&& o) noexcept :
           s(std::move(o.s)),       // 类类型成员的显式移动
           k(std::exchange(o.k, 0)) // 非类类型成员的显式移动
    { }
};

A f(A a)
{
    return a;
}
```

```
struct B : A
{
    std::string s2;
    int n;
    // 隐式移动构造函数 B::(B&&)
    // 调用 A 的移动构造函数
    // 调用 s2 的移动构造函数
    // 并进行 n 的逐位复制
};

struct C : B
{
    ~C() { } // 析构函数阻止隐式移动构造函数 C::(C&&)
};

struct D : B
{
    D() { }
    ~D() { }          // 析构函数阻止隐式移动构造函数 D::(D&&)
    D(D&&) = default; // 强制生成移动构造函数
};

int main()
{
    std::cout << "Trying to move A\n";
    A a1 = f(A()); // 按值返回时，从函数形参移动构造其目标
    std::cout << "Before move, a1.s = " << std::quoted(a1.s) << " a1.k = " << a1.k << '\n';
    A a2 = std::move(a1); // 从亡值移动构造
    std::cout << "After move, a1.s = " << std::quoted(a1.s) << " a1.k = " << a1.k << '\n';

    std::cout << "Trying to move B\n";
    B b1;
    std::cout << "Before move, b1.s = " << std::quoted(b1.s) << "\n";
    B b2 = std::move(b1); // 调用隐式移动构造函数
    std::cout << "After move, b1.s = " << std::quoted(b1.s) << "\n";

    std::cout << "Trying to move C\n";
    C c1;
    C c2 = std::move(c1); // 调用复制构造函数

    std::cout << "Trying to move D\n";
    D d1;
    D d2 = std::move(d1);
}
```

输出：

```
Trying to move A
Before move, a1.s = "test" a1.k = -1
After move, a1.s = "" a1.k = 0
Trying to move B
Before move, b1.s = "test"
After move, b1.s = ""
Trying to move C
move failed!
Trying to move D
```

来自“https://zh.cppreference.com/mwiki/index.php?title=cpp/language/move_constructor&oldid=63762”