值类型与右值引用

```
set(cmake_CXX_STANDARD 14)
set(cmake_CXX_FLAGS -fno-elide-constructors   关闭返回值优化
```

① 引入，下面的函数执行几次拷贝

```
int gota(){
    int a=10; return a;}
int x = gota();
```

实际操作 ⟹ ① int temp = a;
② int x = temp;

② C++98 表达值类型 {左值：持久生命周期，可取地址
右值：临时存在，不可取地址

核心区分：是否能取地址

左值对象：所有变量、数组元素、左值引用的函数结果等

右值对象：除字符串外的所有字面量、表达式结果、临时对象

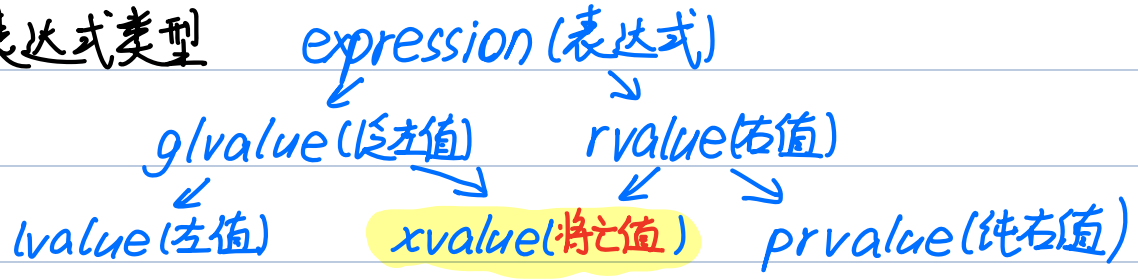
```
int fun(int &x){
int b=a;
a = a+1;
return b}
```

int *p = &x++;

int *p = &++x;

```
int &funint &a)
{
  a = a+1;
  return a;
}
```

③ C++11表达式类型



④ 赋值操作 类比公司单位用人(代码使用要求)

要求：公司要实习的我和公司中10年码龄的cpp老师父一样为害

a. 拷贝操作：我也撸十年代码，和师父一样吊，但公司已倒闭，师父已逝世

时间代价太大

b. 引用：我与师父共享大脑，我拥有和师父一样的cpp知识

但两人无法分开，我在想美女时师父也会想，两人想法同样共用

影响工作推进

C. 移动(郭修)：师父已到35岁高龄面临裁员，公司直接将师父的大脑移动至我的身体，并弃用原身体，如此我拥有知识，并呆在公司内

⑤右值引用与移动语义

如何得到右值引用 Type &&X

移动语义：std::move 将左值转为右值，此时变为将亡值

右值引用仅在接受右值时仍为右值，其它情况均为左值

⑥如何得到将亡值
- a. 将泛左值转为将亡值
  - stastic_cast<type&&>(x);
  - std::move(x);
- b. 临时变量实质化（C++17引用）

注：①纯右值也可使用std::move

②移动语义并未移动，完美转发并不完美

类中未实现移动操作，std::move仍为拷贝

③右值引用仍为左值

④右值绑定到右值引用上不会发生构造或移动(与左值移动类似)

⑤const && → 底层

```
cmake_minimum_required(VERSION 3.15)


# 项目名称

project(MyCppProject)


# 设置 C++ 标准为 C++17

set(CMAKE_CXX_STANDARD 17)

set(CMAKE_CXX_STANDARD_REQUIRED ON)

set(CMAKE_CXX_EXTENSIONS OFF)


# 关闭返回值优化（不同编译器的选项不同，这里同时支持 GCC/Clang 和 MSVC）

if (CMAKE_CXX_COMPILER_ID MATCHES "GNU|Clang")

    add_compile_options(-fno-elide-constructors)

elseif (MSVC)

    add_compile_options(/Od) # MSVC 没有专门关闭 RVO 的选项，用关闭优化来间接实现

endif()


# 添加可执行文件（替换为你的源文件名）

add_executable(MyExecutable
```

```
    Demo_04.cpp

    # 如果你有其他源文件，继续添加在这里

)
```