Ch.6 参数、常量、易变变量

2017 / 11

传值参数

```
void add_3(int a)
{
        a += 3;
}

add_3(51);                  //有实际的变量值，无变量名
int x= 23; add_3(x) //实际上同上，传进来的是值23，
                            //而不是变量本身
                            //相当于调用add_3(23);
```

类与对象

```
void peel_potato(Potato p)
{
        p.peel();
}
```

p是值，并非对象本身

传引用参数

```
void add_3(int & a)
{
        a += 3;
}

add_3(51);                  //有实际的值，无变量名（临时）
int x= 23; add_3(x);//不是对值进行操作
                            //是对 x本身进行操作，add_3(x);
```

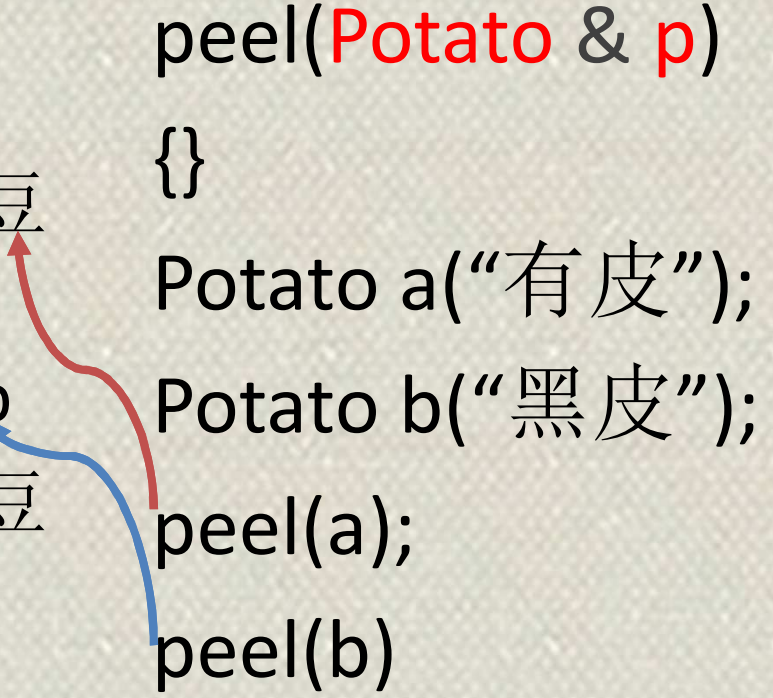
举例说明

- 值方式
  - 削个有皮的土豆
  - 削个黑皮的土豆
  - a/b 只是描述信息，实际上削了哪个，不确定

```
peel(Potato p)
{};
Potato a(“有皮”);
Potato b(“黑皮”);
peel(a);
peel(b)
```

举例说明

- 传引用方式
  - 把有皮的“这个”土豆削了
  - 削了这个黑皮的土豆b
  - a和b，是指具体的土豆



値 – 引用

const 修饰

```
const Potato & Potato::changeName(const string & name) const
```

(1) (a) (2) (b) (3)

三个const:

(1) 返回的值不可以被修改

(2) 传入的引用参数，不允许被修改

(3) 不允许修改Potato类中的成员

二个&

(a) 返回的是个引用，一个具体的对象

(b) 传入的是个引用

连续赋值问题

- Potato a("Obama");
- Potato b("Trump");
- Potato x = (b = a);　// 不会有问题
- Potato y = b = a;　// 不会有问题
- x = b = a;　// 有问题，根源是什么？

问题的根源

- 左值要能赋值
- 赋值=，初始化是拷贝构造函数，非初始化调用的是重载的 =

mutable是做什么的？

- mutable修饰的成员，可以在const修饰的函数中加以修改