第3章-类成员进阶

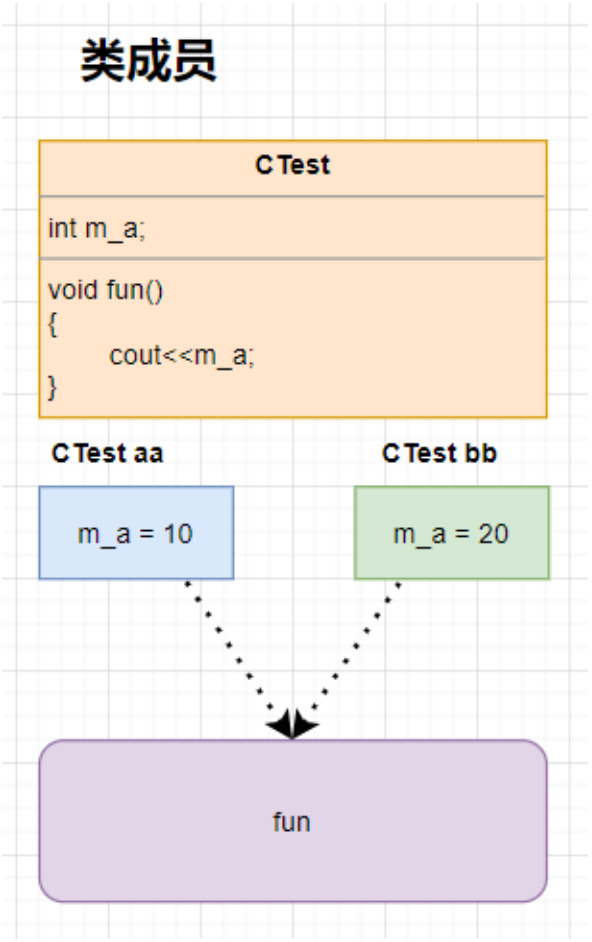
author：岳石磊 copyright：科林明伦 内部资料禁止外泄

1. 类成员

C++标准规定，凡是一个独立的(非附属)对象都必须具有非零大小，所以一个空类即使没有任何数据存储其大小不能为0，空类实例占用内存的大小为1，是用来在内存中占位的，不同的对象在内存中的地址不同。
`sizeof(类型)`表示，分配当前类型的变量所占用的空间大小。当一个类中存在非静态成员变量的时候，`sizeof（类）`占位用的一个字节就不会单独存在，因为变量的地址已经起到了占位、标识的作用；

类成员属性，属于对象的，只有在定义对象的时候，才会真正的存在（在内存中分配空间），定义多个对象，成员属性存在多份（表现为在内存中的地址不同），彼此独立，互不干扰。

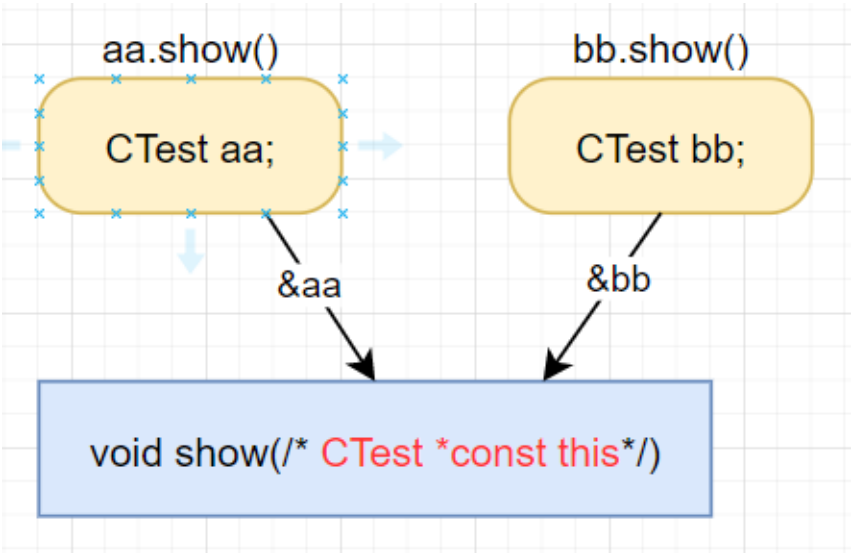
类成员函数，属于类的，在编译期就存在了，与是否定义对象无关。类成员函数只有一份，多个对象共享同一个函数。



2. this 指针

类中的非静态成员函数包括构造析构，会有一个默认的隐藏的参数 this **指针**，它是编译器默认加上的在所有参数之前，类型为当前类的指针 即： `类 * const this`，当我们在用对象调用函数的时候，this 指针就指向了调用的对象，在函数中使用类成员属性或其他类成员函数都是默认通过this指针调用的，在平时写代码的时不用显示的指明this，因为编译器会默认加上。
this指针作用：连接对象和成员函数的桥梁，可以在函数中无感知的使用成员。

```
class CTest{
    int m_a;
    void show(/* CTest * const this*/){
        m_a;  //等效于 this->m_a;
    }
}
```



3. 静态成员

静态成员需要使用关键字 `static` 修饰。

```
static int m_a;
static void show(){}
```

静态成员变量

- 属于类的，不参与类对象的空间占用，在编译期就存在，所有的对象共享这个静态变量。
- 需要在类外进行初始化，格式：类型 类名::变量名 = 初始化值。
- 可直接类名作用域去调用，也可以通过对象去调用。

静态成员函数

- 属于类的，不参与类对象的空间占用，在编译期就存在，所有的对象共享同一份静态成员函数。
- 可直接类名作用域去调用，也可以通过对象去调用。

与普通的成员函数的区别：

1. 没有隐藏的this指针参数，也就不能使用普通的成员变量和函数，只能使用静态成员变量和函数。但在普通成员函数中可以使用静态成员。
2. 静态成员函数是否通过对象都可以调用，而普通的成员函数只能通过对象去调用。

4. 常量

常量的特性：定义就必须初始化，一旦初始化后就不能再去修改其值（*不能通过正常手段修改*）。

当类中有const类型的变量时，在定义的时候必须要初始化，而这个初始化操作是在**初始化参数列表**中完成的，而构造函数的函数体代码中进行的操作严格来说是赋值，而并非初始化。先执行初始化参数列表，在执行构造函数体中的代码。对于普通的变量来说也可在初始化参数列表中初始化。

写法：在构造函数的参数列表后加上一个冒号 : 后面是初始化的成员，用圆括号 ()的形式指定初始化值（而不是用=等号），多个成员用逗号,分割。

```
class CTest{
    int m_a;
    const int m_b;  //常量
    char m_c;
    //初始化参数列表
    CTest(int b):m_a(10),m_b(b),m_c('c'){
        m_a = 20;   //赋值，允许
        //m_b = 30; //赋值，不允许
    }
};
```

初始化成员顺序为**成员在类中定义的顺序**，而不是写在初始化参数列表中的顺序。

5. 常函数

常量指针 和 指针常量：

```
int a=100;
int b =200;
const int c =300;

const int *p1=&a;  //等同于 int const *p1=&a; const 修饰*p1，指明了 *p1 是不允许修改的，
//*p1=200; //是非法的
p1 = &b;   //是合法的

int * const p2=&a; //const 修饰 p2，  即p2的指向不能修改，所以其必须初始化
```

```
//p2 =&b;  //是非法的
*p2  =400; //是合法的


const int *const p3=&c; //指明了当前的指针指向不能修改，且指向的具体的空间的内容也不能修改
```

常量指针升降级问题：

```
const int a = 10;
const int* p1 = &a;

int b = 20;
int* p2 = &b;

//指针的安全级别降级操作，不允许
//p2 = p1;  //error C2440: “=”: 无法从“const int *”转换为“int *”

//指针的安全级别升级操作，允许
p1 = p2;   //int*   ->  const int*
```

常函数：类中的成员函数参数列表后面有const修饰时，称之为常函数，其主要作用是为了能够保护类中的成员变量、限制修改。

特性是：**不能修改类中的非静态成员**，因为const修饰this指针变为`const 类* const this`，也就是执行 `this->变量=val` 操作是非法的，但可以查看成员变量。对于静态成员属性不但能查看，也能对其修改，因为静态成员不是属于对象的，并不在const 约束范围内。

在常函数中可以查看普通的变量、常量、静态变量等，也可以调用其他常函数，但是却不能调用普通的成员函数，因为其this指针的类型并不相同，
`CTest* const this = const CTest* const this`
这是指针安全级别降级的非法操作。

```
class CTest{
    int        m_a;
    const int  m_b;
    static int m_c;
    void f(){}
    void c2()const{}
    void c1()const{
        m_a;//查看合法
        m_b;//查看合法
        m_c;//查看合法
```

```
        m_c = 20; //修改合法
        //f(); //error C2662: “CTest::f”: 不能将“this”指针从“const CTes
    t”转换为“CTest &”
        c2();
    }
};
```

但在普通的成员函数中却是可以调用常函数的，
`const CTest* const this = &CTest`这是一个指针安全级别升级的合法操作。

如果在常函数中想修改部分指定的普通成员，我们可以使用关键字`mutable`来修饰该成员。

```
class CTest{
    int         m_a;
    mutable int m_b;
    void c1()const{
        //m_a = 10; //非法
        m_b = 20;   //合法
    }
};
```

6. 常量对象

常量对象：使用const 修饰的对象（如 `const CTest tst;`） 不能调用普通的成员函数，只能调用常函数。这里面涉及到了 this指针的安全级别升级还是降级的操作。
`CTest *const this = &const CTest`，这是一个安全级别降级的非法的操作，而普通的对象调用常函数 `const CTest* const this = &CTest` 这是一个安全级别升级的合法操作。