河北师范大学软件学院
Software College of Hebei Normal University

C++

第七讲 类和对象（三）

C++备课组 丁盟

自我介绍

丁盟

qq：2622885094

上一讲教学目标

- 掌握C++中类的构造函数
- 掌握C++中类的析构函数
- 掌握C++中类的拷贝构造函数

本讲教学目标

- 掌握C++中静态成员的使用
- 掌握C++中对象成员的使用
- 掌握C++中const与类的结合使用

1 静态成员

2 对象成员

3 const成员

静态成员

静态成员

- 一、静态数据成员
- 二、静态成员函数

静态成员 – 静态数据成员

❖以关键字static声明的数据成员

```
class Box
{
public:
    int volume()const;
private:
    int m_iWidth;
    int m_iLength;
    int m_iHeight;
    static int s_iCount;
};
```

静态成员 – 静态数据成员

类型 类名::静态数据成员名 [= 初值];

- 静态数据成员必须初始化且只能在类外初始化
- 初始化时不能加static

```
class Box
{
public:
    int volume()const;
private:
    int m_iWidth;
    int m_iLength;
    int m_iHeight;
    static int s_iCount;
};
int  Box::s_iCount;
```

必须在类外初始化

静态成员 – 静态数据成员

```
class Box
{
public:
    int volume()const;
    static int s_iCount;
private:
    int m_iWidth;
    int m_iLength;
    int m_iHeight;
};
```

```
Box::Box()
{
    Box::s_iCount = 5;
    … …
}
```

不能通过构造函数初始化

静态成员 – 静态数据成员

```
class Box
{
public:
    int volume()const;
    static int s_iCount;
private:
    int m_iWidth;
    int m_iLength;
    int m_iHeight;
};
```

```
Box::Box()
    : s_iCount(8)
{
    … …
}
```

不能通过初始化列表初始化

静态成员 – 静态数据成员

- 静态数据成员属于类而不属于具体的对象，为不同对象共有因此，公有静态数据成员在类外的访问方式有两种：

类名::公有静态数据成员

对象名.公有静态数据成员

- 静态数据成员可被其所在类的任何成员函数直接引用。

```
void Box::display()const
{
      cout << Box::s_iCount << " ";
      cout << s_iCount << " ";
}
```

静态成员 – 静态数据成员

- 静态数据成员在程序启动时开辟内存单元，占据全局区，类的对象创建前就能使用。
- 静态数据成员的内存单元独立开辟，不属于具体的某个对象，为不同的对象共有。

静态成员 – 静态数据成员

- 为什么要引入静态数据成员？

各对象之间的数据有了沟通的渠道，实现了数据的共享，C++中涉及到对象之间的数据共享时应使用静态数据成员，而不要使用全局变量，因为全局变量不能体现封装特性。

静态成员 – 静态数据成员

- 静态数据成员应用举例：统计创建对象的个数

```
class Student
{
public:
    Student(string aName = "某某某")
    {
        m_strName  =  aName;
        s_iCount++;
    }
    ~Student()
    {
        s_iCount--;
    }
    void printCount()
    {
        cout << "Student Count = " << s_iCount << endl;
    }
private:
    static int s_iCount;
    string m_strName;
};

int Student::s_iCount = 0;

int main(void)
{
    Student stu1("关羽");
    Student stu2("张飞");
    Student * pStu = new Student("刘备");
    stu1.printCount();
    delete pStu;

    return 0;
}
```

静态成员 – 静态成员函数

❖**概念**：

用**static**声明的**成员函数**

```
static 类型 成员函数名(参数表);
```

类体外实现时不能加static

❖**访问方式**:

属于类类型的而不属于具体对象

```
类名::函数名([参数表]);
对象名.函数名([参数表]);
```

静态成员 – 静态成员函数

- **功能**：专门用于访问静态成员。
- **思考**：静态数据成员可直接被普通成员访问。那么静态成员函数有什么意义呢？
- **实验**：静态数据成员在对象创建之前静态成员就可以访问了，如果把static int s_iCount;设为private，在创建对象前设法访问静态成员s_iCount。

静态成员 – 静态成员函数

❖特点

- 静态成员函数专门用于访问静态成员
  （包括静态数据成员和静态成员函数）
- 是属于类的而不属于具体对象，因此既可以通过类名访问，也可以通过对象名引用
- 其实静态成员函数就是在类内的全局函数
- 静态成员函数没有this指针（先记住）

静态成员 – 静态成员函数

```
class Student {
public:
    Student(string aName = "某某") {
        m_strName  =  aName;
        s_iCount++;
    }
    ~Student() {
        s_iCount--;
    }
    static void printCount();

private:
    static int s_iCount;
    string m_strName;
};
```

```
void Student::printCount() {
    cout << "Student Count = " << s_iCount << endl;
}
int Student::s_iCount = 0;

int main(void) {
    Student stu1("关羽");
    Student stu2("张飞");
    Student * pStu = new Student("刘备");
    stu1.printCount();
    delete pStu;

    return 0;
}
```

静态成员 – 静态成员函数

- **[注意]**
  - 非静态成员函数可以访问本类中的任何成员
  - 静态成员函数专门用于访问静态成员，不能直接访问非静态成员。

1 静态成员

2 对象成员

3 const成员

对象成员

```
class Engine
{
public:
    Engine();
};

class Car
{
public:
    Car();
private:
    Engine  m_Engine; //组合
};
```

对象成员

- 对象作为类的数据成员

```
class Engine
{
public:
    Engine();
};

class Car
{
public:
    Car();
private:
    Engine  m_Engine;
};
```

学习要点：

- 构造函数如何定义？
- 子对象构造函数与析构函数的执行次序？

对象成员

```
class  B
{
    B(){}
};

class  A
{
    B obj;
    A():obj()
    {}
};
```

- 类A的构造函数应为：

```
A::A(参数表1):obj([参数表2])
{
    函数体
}
```

- 构造函数执行顺序：先子对象再本类
- 析构函数执行顺序：与构造函数的顺序相反

对象成员

```
class B {
public:
    B(): m_iB(1) {
        cout << "Cons B" << endl;
    }
    void printb() {
        cout << "m_iB ="
             << m_iB << endl;
    }
private:
    int m_iB;
};
class A {
public:
    A(int i): m_iA(i), m_tVal() {
        cout << "Cons A" << endl;
    }
    void printa() {
        cout << "m_iA =" << m_iA
<< endl;
        m_tVal.printb();
    }
private:
    int m_iA;
    B m_tVal;
};
int main(void) {
    A m(2);
    m.printa();

    return 0;
}
```

```
Cons B
Cons A
m_iA =2
m_iB =1
```

对象成员

说明：

- 当调用子对象的不带参数构造函数时，要在参数初始化表中省略“:子对象名( )” 。
- 有多个子对象时，则在初始化表中依次给出各子对象的初始化表达式，之间用逗号分隔。
- 子对象构造函数的执行顺序按照类中的声明顺序进行，与初始化表中的顺序无关。

对象成员

- 在含有子对象的类A中若有析构函数，则执行顺序为以子对象在类A中声明相反的顺序调用各类的析构函数。

1 静态成员

2 对象成员

3 const成员

const成员

- const数据成员（常成员）
- const成员函数（常成员函数）
- const对象

const成员 – 常数据成员

- **常数据成员**：如果类中的某些属性为常量，则将其设定为常数据成员

```
class Math {
public:
    Math(double arg=0);
    void set(double dval);
    void print();
private:
    const double m_kPi;
    const double &r;
    double m_iFirst;
};
Math::Math(double arg)
    :m_kPi(3.14), r(m_kPi),
     m_iFirst(arg)
{}
```

```
void Math::print() {
    cout << m_kPi << " " << r;
}

void Math::set(double dval) {
    m_iFirst = dval;
}

int main(void) {
    Math obj;
    obj.print();
    return 0;
}
```

const成员 – 常数据成员

[注意]

- 常成员和引用成员必须通过初始化列表赋初值

```
Math::Math() : m_kPi(3.14), r(m_kPi)
{
}
```

- 常数据成员是常量不能被修改

```
void Math::set(double dval)
{
    m_iFirst = dval;
    // m_kPi = ival;error
}
```

const成员 – 常成员函数

❖常成员函数

格式：

类型 函数名（参数表）**const**;

注意：

- const在声明和定义时都要有
- 常成员函数只可引用本类的数据成员而不能修改他们
- 如果成员函数仅是访问数据成员，而不修改则应为常成员函数，定义为常成员函数。
- 静态成员函数不能声明为常成员函数

const成员 – 常成员函数

- 常成员函数不能调用非常成员函数,只能调用常成员函数

```
class Math {
public:
    Math(double arg=0);
    void set(double dval);
    void print()const;
private:
    const double m_kPi;
    const double &r;
    double m_iFirst;
};
Math::Math(double arg)
    :m_kPi(3.14), r(m_kPi),
     m_iFirst(arg)
{}
```

```
void Math::print() const {
    //set(3.4);
    cout << m_kPi << " " << r;
}

void Math::set(double dval) {
    m_iFirst = dval;
}

int main(void) {
    Math obj;
    obj.print();
    return 0;
}
```

const成员 – 常成员函数

- 常成员函数可以作为函数重载的依据

```
#include <iostream>
using std::cout;
using std::endl;

class A {
public:
    void print()const;
    void print();
};

void A::print() const {
    cout << "const print be called"
         << endl;
}
```

```
void A::print()
{
    cout << "print be called";
}

int main(void)
{
    A a1;
    a1.print();
    const A a2;
    a2.print();

    return 0;
}
```

const成员 – 常对象

❖常对象

格式：

```
类名    const  对象名[(实参表)];
const  类名    对象名[(实参表)]; // 推荐
```

含义：对象的数据成员（属性）不能被修改

const成员 – 常对象

[注意]

- 常对象必须初始化

```
class A
{
public:
    A(int a = 1):m_iVal(a)
    {
    }
private:
    int m_iVal;
};
```

```
int main(void)
{
    const A obj1;
    const A obj2(2);
    return 0;
}
```

const成员 – 常对象

常对象的数据成员是常量

常对象的数据成员不能被修改

const对象只能引用const成员函数

const成员 – 常对象

```
class CTime {
public:
    Time(int a=0, int b=0,
         int c=0);
    void print()const;
    static int getCount();
private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
    static int s_iCount;
};
int Time::s_iCount;
int Time::getCount() {
    return s_iCount;
}
```

```
Time::Time(int a, int b, int c)
    :m_iHour(a), m_iMinute(b),
     m_iSec(c)
{
    s_iCount++;
}
void Time::print() const {
    cout << m_iHour << " "
      << m_iMinute << " "
      << m_iSec << endl;
}
int main(void) {
    const Time obj;
    obj.print();
    return 0;
}
```

const成员 – 常对象

- 常对象只能引用常成员函数，普通对象可引用任何成员函数，因此const成员函数可以作为重载依据

```
class A
{
public:
    void print()const;
    void print();
};

void A::print()const
{
    cout << "const" << endl;
}
```

```
void A::print() {
    cout << "print";
}
int main(void)
{
    A a1;
    a1.print();
    const A a2;
    a2.print();
    return 0;
}
```

const成员 – 常对象

- 有时要求，一定要修改常对象中的某个数据成员的值，只需对该数据成员声明为mutable

```
类型 mutable 数据成员名;
```

```
mutable 类型 数据成员名;
```

const成员 – 常对象

总结：

数据成员	非const成员函数	const成员函数
非const数据成员	可以引用， 也可以改变值	可以引用， 不可以改变值
const数据成员	可以引用， 不可以改变值	可以引用 不可以改变值

本讲教学目标

- 掌握C++中静态成员的使用
- 掌握C++中对象成员的使用
- 掌握C++中const与类的结合使用


THANKS