2.3静态成员

静态成员

- 静态数据成员（静态成员变量）
  - 不破坏封装性
  - 解决**对象之间的通信**（数据共享）
- 静态成员函数（静态方法）
  - 访问静态数据成员

静态数据成员

- 关于类的属性（成员变量）
  - 不同对象具有**不同**的属性值（实例属性）。
  - 所有对象具有**相同**的属性值（类属性）。
  - 类属性声明：

```
static<数据类型><静态成员名>
```

- 初始化时机
  - 实例属性：对象创建时，构造函数。
  - 类属性：所有类共享，类外部初始化。

```
<数据类型><类名>::<静态数据成员名>=<初始值>
```

静态数据成员

- 静态成员的访问
  - 通过对象访问：person.m_nCount
  - 通过类名访问：CPerson::m_nCount
- 静态成员的权限属性：公有、私有、保护。

例子：静态数据成员

```
//CPerson.hpp
#include <iomanip>
using namespace std;

class CPerson
{
private:
    char m_strName[20];
    long m_ID;
    double m_wage;
    static int m_nCount; //私有静态成员变量，表示已创建对象的数量
```

```
    static double m_nTotalWage; //私有静态成员变量，表示所有雇员的工资总额
public:
    CPerson(const char* strName = nullptr, long id = 0, double wage = 0);

    friend ostream& operator << (ostream& os, const CPerson& person)
    {
        os << "(" << person.m_strName << "," << person.m_ID << "," << person.m_wage << ")";
        os << " Count: " << CPerson::m_nCount << ", Total Wage: " << CPerson::m_nTotalWage;
        return os;
    }

};
```

```
#include<bits/stdc++.h>
#include "CPerson.hpp"

int CPerson::m_nCount = 0;
double CPerson::m_nTotalWage = 0;


CPerson::CPerson(const char* strName, long ID, double wage)
{
    strcpy(this->m_strName, strName);
    this->m_ID = ID;
    this->m_wage = wage;


    CPerson::m_nCount++;
    CPerson::m_nTotalWage += this->m_wage;

}


int main()
{
    CPerson person1("Tom", 1001, 1000);
    cout << person1 << endl;

    CPerson person2("Jack", 1002, 2000);
    cout << person2 << endl;

    CPerson person3("Mary", 1003, 3000);
    cout << person3 << endl;

    return 0;
}
```

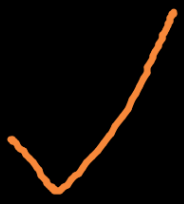
```
(LiMing, 1101051, 3000) Count: 1, Total Wage: 3000
(HanMeimei, 1101052, 5000) Count: 2, Total Wage: 8000
```

例子：静态数据成员 *

- 修改员工资setWage()，怎么写?

```cpp
void CPerson::setWage(double wage) {
    this->m_wage = wage;
}
```

```cpp
void CPerson::setWage(double wage) {
    CPerson::m_totalWage -= this->m_wage;
    this->m_wage = wage;
    CPerson::m_totalWage += this->m_wage;
}
```



```cpp
void CPerson::setWage(double wage) {
    CPerson::m_totalWage += wage - this->m_wage;
    this->m_wage = wage;
}
```

原则：

① 尽量少用全局性的数据。

② 全局性数据的操作尽量单一。

静态成员函数

- 静态成员函数也与一个类相关联，而不只与一个特定的对象相关联。
- 静态成员函数只能访问类的静态成员（成员变量和成员函数），而不能访问类的非静态成员。
- 静态成员访问非静态成员的方法：将对象作为静态成员函数的参数。 **（不建议）**
- 建议：既要访问静态成员，又要访问非静态成员，则用普通成员函数。
- 区别非静态成员函数，静态成员函数没有 **this指针**，因为类的静态成员函数只有一个运行实例。

例子：静态成员函数

```cpp
#ifndef CPoint_hpp
#define CPoint_hpp

#include <iomanip>
using namespace std;

class CPoint
{
    privare:
        double x, y;
        static int m_nCount;
    public:
        CPoint(double x = 0, double y = 0);
        static int GetCount(const CPoint& lastPoint);
        friend ostream& operator<<(ostream& os, const CPoint& point)
```

```
        {
            os << "(" << point.x << ", " << point.y << ")";
            return os;
        }
};

#endif /* CPoint_hpp */
```

```
int CPoint::m_nCount = 0;

CPoint::CPoint(double x, double y)
{
    this->x = x;
    this->y = y;
    CPoint::m_nCount++;
}

int CPoint::getCount()
{
    return CPoint::m_nCount;
}

void eg3_9()
{
    CPoint point1, point2(1, 1);
    cout << "point1 = " << point1 << endl;
    cout << "point2 = " << point2 << endl;
    cout << "Count = " << CPoint::getCount() << endl;
}
```