一

输出流函数重载为成员方法以及全局函数

①以下输出流重载为成员方法虽然解决了this指针抢占第一个参数的问题，但方法的调用必须用对象去驱动如下：

```
t1 << cout ;
或
t1 << cout << endl;
```

这种输出流重载为成员方法不符合人的使用习惯

所以摒弃

```
//输出流函数重载------>输出流函数重载为成员方法
#include <iostream>
#include <time.h>
using namespace std;
class Test
{
public:
	Test(int d = 0) : m_data(d)
	{cout << "Create Test Object : " << this << endl;}
	//Test t1 = t;
	Test(const Test& t)
	{
		cout << "Copy Create Test Object : " << this << endl;
		m_data = t.m_data;
	}
	//=重载 运算符的重载
	Test& operator=(const Test& t)
	{
		cout << "Assign: " << this << " = " << &t << endl;
		if (this != &t)
		{
			m_data = t.m_data;
		}
		return *this;
	}
	~Test()
	{cout << "Free Test Object : " << this << endl;    }
public:
	int GetData()const
```

```
	{return m_data;}
	int SetData(int data)
	{m_data = data;	}

	ostream& operator<<(ostream& os)   //输出流函数重载为成员方法
	{
		os << m_data;
		return os;
	}

private:
	int m_data;
};

Test fun(Test t)
{
	cout<<"fun : " << endl;
	int value = t.GetData();
	Test tmp(value);
	return tmp;
}

int main()
{
	Test t1(100);
	Test t2, t3;
	t2 = t1; //t3.operator=( t2.operator=(t1) )
			//t2.operator=(t1); //
	Test tmp = fun(t2);
	//cout << "t1 = " << t1 << endl;
	t1 << cout;			//输出流函数重载为成员方法用对象去驱动
	t1 << cout << endl;	//输出流函数重载为成员方法用对象去驱动
	return 0;
}
```

当我们想要将输出流重载为成员方法时如下：

```
ostream& operator<<(ostream& os)   //输出流函数重载为成员方法
	{
		os << m_data;
		return os;
	}
```

使用时就必须用对象去驱动用法如下：

```
t1 << cout ;
```

或

```
t1 << cout << endl;      //输出流函数重载为成员方法
```

fun是类外全局函数，可以对对象访问

```
Test fun(Test t)
{
        cout<<"fun : " << endl;
        int value = t.GetData();
        Test tmp(value);
        return tmp;
}

Test tmp = fun(t2);
```

②输出流重载为全局函数

用全局函数也无法满足要求，全局函数无法访问类的私有数据成员如下

```
ostream & operator<<(ostream& os, const Test& t)  //输出流函数重载为全局方法
{
    os << t.m_data;      //全局函数无法访问类的私有数据成员
    return os;
}

int main()
{
    Test t1(100);
    Test t2, t3;
    t2 = t1; //t3.operator=( t2.operator=(t1) )
             //t2.operator=(t1); //
    cout << t1;
    return 0;
}
```

二
二

输出流重载友元函数

根据以上输出流重载为全局函数的经验，全局函数只是不能访问类的私有成员，友元这一概念打破了类的封装性。将全局函数定义成类的友缘函数，就使得全局函数能够去访问私有数据成员

```
#include <iostream>
using namespace std;
class Test
{
	//输出流重载友元函数
	friend ostream& operator<<(ostream& out, const Test& t);
public:
	Test(int d = 0) : m_data(d)
	{cout << "Create Test Object : " << this << endl;}
	//Test t1 = t;
	Test(const Test& t)
	{
		cout << "Copy Create Test Object : " << this << endl;
		m_data = t.m_data;
	}
	//=重载 运算符的重载
	Test& operator=(const Test& t)
	{
		cout << "Assign: " << this << " = " << &t << endl;
		if (this != &t)
		{
			m_data = t.m_data;
		}
		return *this;
	}
	~Test()
	{cout << "Free Test Object : " << this << endl;}
public:
	int GetData()const
	{return m_data;}
	int SetData(int data)
	{m_data = data;}
private:
```

```
    int m_data;
};

Test fun(Test t)
{
    int value = t.GetData();
    Test tmp(value);
    return tmp;
}

ostream& operator<<(ostream& out, const Test& t)          //输出流重载友元函数
{
    out << t.m_data;
    return out;
}

int main()
{
    Test t1(100);
    Test t2, t3;
    t2 = t1; //t3.operator=( t2.operator=(t1) )
             //t2.operator=(t1); //

    cout << "t1 = " << t1 << endl;
    cout << t1;
    cout << t1 << endl;
    return 0;
}
```

以上输出流重载友元函数可以按人的使用习惯正常去实现输出的功能（防止重载为①成员方法使用习惯不符的尴尬）

```
cout << t1;
cout << t1 << endl;
```

扩展：

通常情况下，输出运算符的

1. 一个形参是一个非常量的ostream 对象的引用。（非常量是因为向流写入内容会改变其状态; 用引用是因为流对象不支持复制）
2. 第二个参数一般来说是一个常量的引用，该常量是我们想要输出的类类型

思考：

但是仔细思考来

重载的输出流函数

```
ostream& operator<<(ostream& out, const Test& t)                    //输出流重载友元
```

函数

```
{
        out << t.m_data;
        return out;
}
```

的返回值ostream&能不能省略呢？

接下来实验一下：

将输出流重载友元函数--省略返回值

```
operator<<(ostream& out, const Test& t)    //输出流重载友元函数--省略返回值
{
      out << t.m_data;
}

int main()
{
      Test t1(100);
      Test t2, t3;
      t2 = t1; //t3.operator=( t2.operator=(t1) )
                  //t2.operator=(t1); //

      cout << t1;
      cout << t1 << endl;              //不能编译通过也不能运行
      cout << "t1 = " << t1 << endl;     //不能编译通过也不能运行
      return 0;
}
```

输出流重载友元函数--省略返回值

编译期间

可以编译通过：

cout << t1;

```
cout << t1;
```

不可以编译通过：

cout << t1 << endl;
cout << "t1 = " << t1 << endl;

```
cout << t1 << endl;
cout << "t1 = " << t1 << endl;
```

错误提示信息：

```
template<class _Elem, class _Traits> std::basic_ostream<_Elem, _Traits> &__cdecl
std::endl(std::basic_ostream<_Elem, _Traits> &_Ostr)
MANIPULATORS

联机搜索

无法确定需要哪个 函数模板 "std::endl" 实例

联机搜索
```

```
operator<<(ostream& out, const Test& t)     //输出流重载友元函数
{
    out << t.m_data;
    return out;
}

int main()
{
    Test t1(100);
    Test t2, t3;
    t2 = t1; //t3.operator=( t2.operator=(t1) )
             //t2.operator=(t1); //

    cout << t1;
    cout << t1 << endl;
    cout << "t1 = " << t1 << endl;
    return 0;
}
```

```
template<class _Elem, class _Traits> std::basic_ostream<_Elem, _Traits> &__cdecl
std::endl(std::basic_ostream<_Elem, _Traits> &_Ostr)
MANIPULATORS

联机搜索

无法确定需要哪个 函数模板 "std::endl" 实例

联机搜索
```

```
6  0
错误 6   警告 0   消息 0
```

结论：

给输出流重载友元函数--省略返回值后

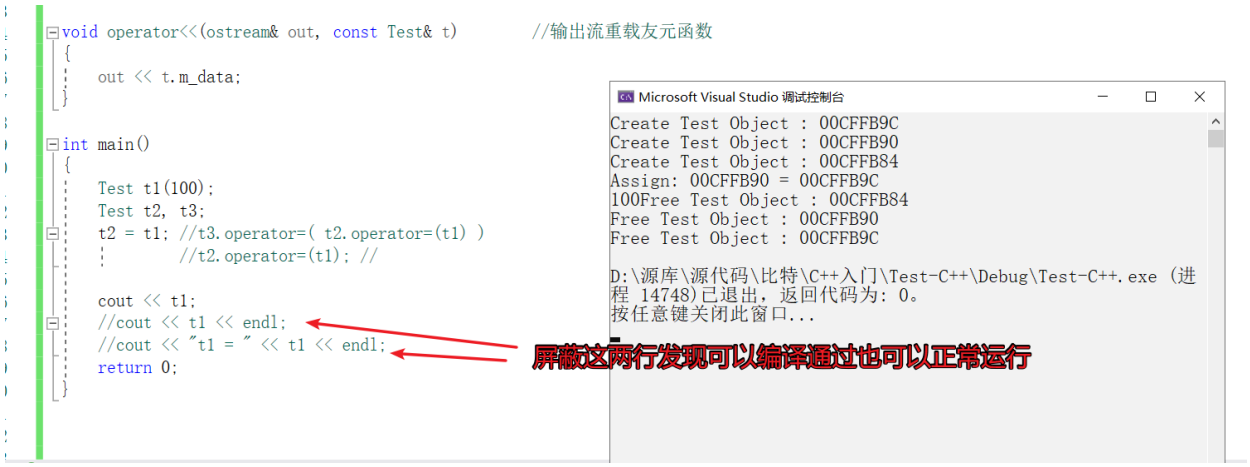
```
cout << t1 << endl;
```

```
cout << "t1 = " << t1 << endl;
```

这两行不能编译通过也不能运行

说明就不能连续打印：

所以输出流重载为友元函数的返回值不能省略



通过对输出流的重载我们对友元函数有了更深刻的认识；

打破类的封装性---->将类外的全局函数声明为类的友元可以使全局函数自由的访问私有数据成员

本笔记所做的四个小测试cpp源文件如下：可自行下载验证

输出流函数重载为成..法.cpp
1.2KB

运算符的重载请各位朋友持续关注后续更新：谢谢指教