类和对象(3)

构造函数(下)

函数体内赋值

在创建对象时，编译器通过调用构造函数，给对象中各个成员变量一个合适的初始值。

```
Date(int year=0,int month=1.int day=1)
{
    //函数体内赋值
    _year=year;
    _month=month;
    _day=day;
}
```

初始化列表

初始化列表：以一个**冒号开始**，接着是一个以**逗号分隔的数据成员列表**，每个"**成员变量**"后面跟一个**放在括号中的初始值或表达式。**

```
Date(int year=0;int month=1;int day=1)
    :_year(year)
    ,_month(month)
    ,_day(day)
{
    //初始化列表
}
```

1. 每个成员变量只能在初始化列表中出现一次(初始化只能初始化一次)
2. 引用成员变量、const成员变量、自定义类型成员，必须放在初始化位置进行初始化
3. 尽量使用初始化列表初始化，因为不管是否使用初始化列表，对于自定义类型成员变量，一定会先使用初始化列表初始化
4. 成员变量在类中声明次序就是其在初始化列表中的初始化顺序，与其在初始化列表中的先后顺序无关

explicit关键字

```
class Date
{
public:
    explicit Date(int year)
        :_year(year)
    {
        //构造函数
    }
    Date(const Date& d)
    {
        //拷贝构造函数
    }
```

```
private:
    int _year;
};
int main()
{
    Date d1(1);
    //调用构造函数
    Date d2=2;
    //隐式类型的转化，构造出tmp(2)、再用tmp拷贝构造d2(tmp)
    const Date& d2=2;
    //引用就是中间产生的临时对象
    Date d3=d1;
    //拷贝构造
}
```

用explicit修饰构造函数，将会禁止单参构造函数的隐式转换

```
class Date
{
public:
    explicit Date(int year,int month,int day)
        :_year(year),_month(month),_day(day)
    {
        //构造函数
    }
    Date(const Date& d)
    {
        //拷贝构造函数
    }
private:
    int _year;
    int _month;
    int _day;
};
int main()
{
    Date d1(2020,1,1);
    //c++11
    Date d2={2020.10.1};

    return 0;
}
```

如果是多参数，在C++11中则可用上面的方法禁止隐式转换

static成员

用static修饰的**成员变量**，称之为**静态成员变量**；用static修饰的**成员函数**，称之为**静态成员函数。静态的成员变量一定要在类外进行初始化。**

特性

1. 静态成员保存在静态区中，为所有类对象所共享，不属于某个具体的实例
2. 静态成员变量必须在类外定义
3. 静态成员函数，没有隐藏的this指针，不能访问任何非静态成员
4. 类的静态成员可以用**类名::静态成员**或者**对象.静态成员**来访问

5. 静态成员和普通成员一样，也有public、private、protected访问级别

```
class A
{
public:
    //构造函数
    A()
    {
    }
    //静态成员函数，没有隐藏的this指针，
    static void print()
    {
        //Test();
        //cout << m << endl;
    }
    void Test()
    {
        print();
        cout << n << endl;
    }
private:
    int m;
    //静态成员函数在类内声明
    static int n;
};
//静态成员变量在类外定义
int A::n=1;
int main()
{
    A a;
    cout<<
}
```

C++11成员初始化

C++11支持非静态成员变量在声明时进行初始化赋值，但这里不是初始化，而是给声明的成员变量缺省值。

```
class Test
{
public:
    //构造函数
    Test()
    {

    }
private:
    //C++11
    //非静态成员变量，声明时给缺省值
    int n=1;
};
```

友元

友元分为：友元函数和友元类

友元提供了一种突破封装的方式，有时提供便利，但友元会增加耦合度

友元函数

友元函数可以直接访问类的私有成员，他是定义在类外部的普通函数，不属于任何类，但是需要在类的内部声明，声明时需要加friend关键字。

```
class A
{
public:
    //类内用关键字friend声明
    friend void Print();
    //构造函数
    Date(int m)
        :_m(m)
    {}
private:
    int _m;
};
//类外定义友元函数
void Print()
{
    cout<<_m<<emdl;
}
```

- 友元函数可以访问类的私有和包含成员，但不是类的成员函数
- 友元函数不能用const修饰
- 友元函数可以在类定义的任何地方声明，不受访问限定符限制
- 一个函数可以是多个类的友元函数
- 友元函数的调用与普通函数的调用和原理相同

友元类

友元类的所有成员函数都可以是另一个类的友元函数，都可以访问另一个类中的非公有成员。

- 友元关系是单向的，不具有交换性
- 友元关系不能传递

内部类

如果一个类定义在另一个类的内部，这个内部类就叫做内部类。此时这个内部类是一个独立的类，不属于外部类，也不能通过外部类的对象去调用内部类。

- 内部类就是外部类的友元类，内部类可以通过外部类的对象参数来访问外部类中的额所有成员
- neibui可以定义在外部类的public、protected、private都是可以的
- 内部类可以直接访问外部类中的static、枚举成员、不需要外部类的对象/类名
- sizeof(外部类)=外部类，和内部类没有关系