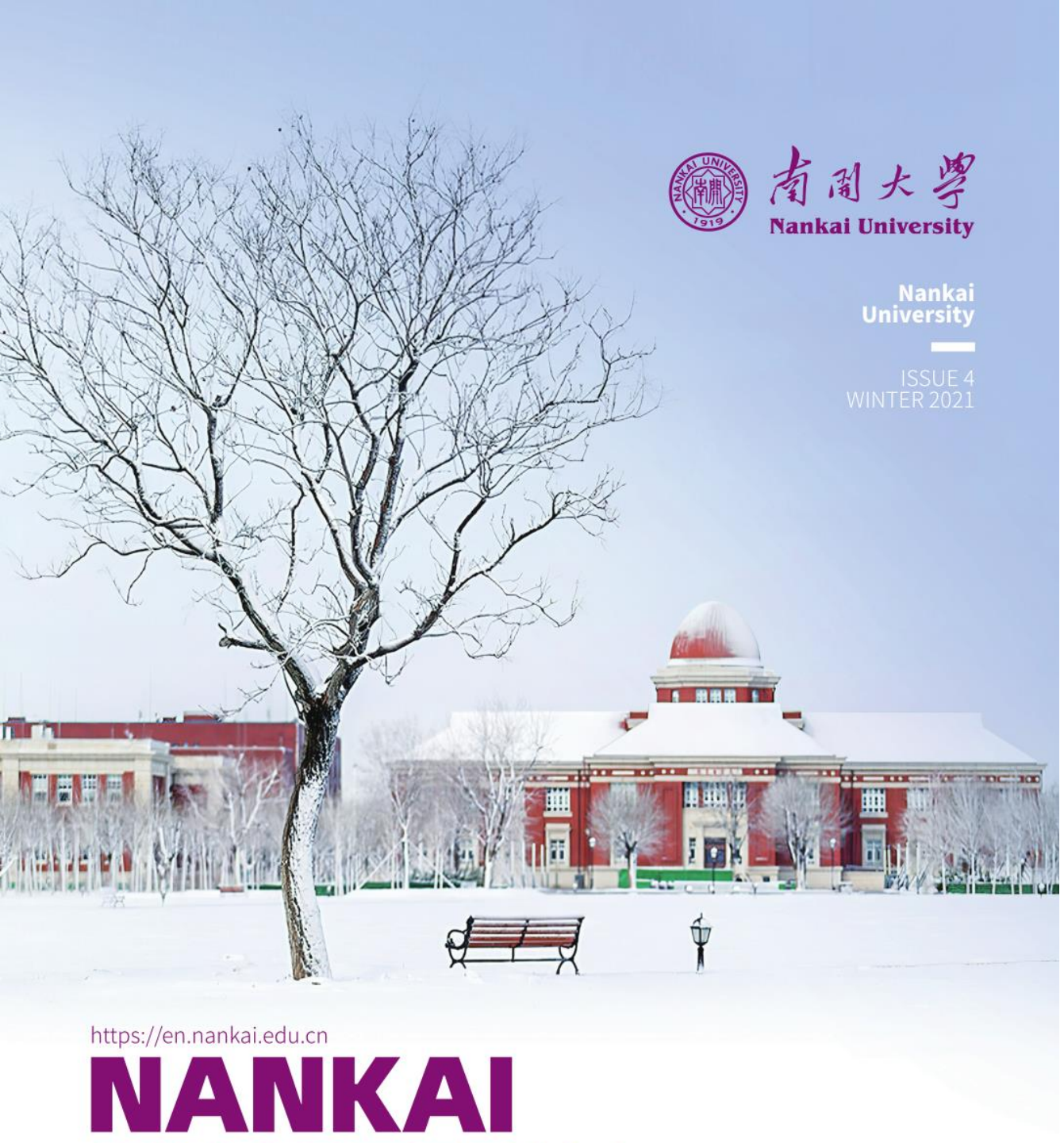

存取控制

龚成

cheng-gong@nankai.edu.cn

成员属性

私有属性

保护属性

公有属性

成员属性

- 成员属性的规定用于控制类成员的不同访问属性。通过将不同的成员划分为不同的访问控制属性，可以将同一类的成员封装为不同功能的。
- 例如，对于person类别，其name或者nickname属性可能往往是公开的，便于人名引用，而身份证ID或者血型之类的属性，往往应该是私密而不公开的，但是又应该person的必要成员。

```
class Person
{
private:
	int bloodType;
	int age;
	int personID;
public:
	string name;
	person(/* args */);
	~person();
};
```

公有属性

- 类的public成员不仅在类中可以访问，而且在建立类对象的其他模块中，也可以通过对象来访问。例如直接对公有属性访问或者赋值：

  a.public_attr=0;

- 类的public成员使用与struct的成员访问类似，可以直接赋值，也可以使用{}进行所有属性赋值：

```
class Car{
public:
        int type;
        char color;
        int price;
};
```

```
int main(){
        Car c1;
        c1.type = 1;
        c1.color = 'r';
        c1.price = 100;
        Car c2={2,'g',125};
        std::cout<<"c1 = {"<<c1.type<<","<<c1.color
                        <<","<<c1.price<<"}"<<std::endl;
        std::cout<<"c2 = {"<<c2.type<<","<<c2.color
                        <<","<<c2.price<<"}"<<std::endl;
}
```

私有属性

- 只可在本类中对类的private成员进行访问，在别处是“不可见的”，不可直接使用a.private_attr=0;对属性赋值。

```
class Car{
public:
        int type;
        char color;
        int price;
private:
   int speed;
};
```

```
int main(){
        Car c1;
        c1.type = 1;
        c1.color = 'r';
        c1.price = 100;
        c1.speed = 100; // error: 'speed' is a private member of 'Car'
        return 0;
}
```

私有属性

- 要访问私有属性的成员，需要通过友元或者**公有属性的函数成员**进行间接访问：

```
class Car{
public:
        int type;
        char color;
        int price;
        void setSpeed(int s){
          speed = s;
        }
private:
  int speed;
};
```

```
int main(){
        Car c1;
        c1.type = 1;
        c1.color = 'r';
        c1.price = 100;
        c1.speed = 100; // error
        c1.setSpeed(100); // ok
        return 0;
}
```

私有属性

- 注意：同一个类实例化的不同对象，其私有属性可以被其他同类对象的函数成员访问：

```
class Car{
public:
        int type;
        char color;
        int price;
        void setSpeed(int s){
          speed = s;
        }
        int getSpeed(Car& car){
          return car.speed;
        }
private:
  int speed;
};
```

```
int main(){
        Car c1,c2;
        // c1.speed = 100; // error
        c2.setSpeed(100); // ok
        std::cout<<"c2.speed="<<c1.getSpeed(c2)<<std::endl;
        return 0;
}
```

保护属性

- 类的protected成员在成员属性方面和private属性一致：
  - 只能在类内访问
  - 别处“不可见”
- 不同于private属性的地方在于：protected成员在其派生类中访问和继承关系依然为protected属性，同样不可被类外访问。

友元函数与友元类

友元说明

友元函数

友元函数与成员函数

友元类

友元说明

- 面向对象程序设计主张程序的封装、数据的隐藏，但任何事物都不是绝对的，友元的概念是 C++ 语言为用户提供的在局部打破这种封装和隐藏的手段。
- 用关键字 friend 说明友元的概念为 C++所特有，其作用是，在类的说明语句中出现：
  - 位于一个函数说明语句之前，指出该函数为**这个类的友元函数**。
  - 位于一个类名之前，指出该类是**这个类的友元类**。
- 在类 A中说明的友元函数f:
  - 它不是 A的函数成员。
  - f 的定义可以在类 A的说明内，也可以在类外。
  - 函数 f 虽不是 A的成员，但有权访问和调用 A的所有私有及保护成员。
- 在类 A中说明的友元类B:
  - 它可能是与 A无关的另外一个类。
  - 要在类外说明。
  - B的任一函数都有权访问和调用类 A的所有成员，包括私有及保护成员。

```
class A{
    ...
    friend int f (int a);
    ...
    friend class B;
    ...
};
```

友元函数

- 在类A的**友元函数**和类A的**成员函数**中（定义处）都可处理与使用类 A 的私有成员，但两种函数的最大使用区别是：
  - 对友元函数来说，参加运算的所有运算分量（如类对象）必须显式地列在友元函数的参数表中（由于友元函数中没有 this 指针，从而没有“当前调用者对象”的概念）。但对成员函数来说，它总以当前调用者对象（* this）作为该成员函数的隐式第一运算分量。若所定义的运算多于一个运算对象时，才将其余运算对象显式地列在该成员函数的参数表中。
  - 调用友元函数时根本不通过类对象（因为它并非类的成员），调用类成员函数时必须通过类对象。
  - 友元函数和友元类不是类成员，因此不受类的属性限制，放在private下也行。
  - 友元函数最常用于运算符重载，因为其**不存在隐藏运算分量**。

友元函数与成员函数

- 友元函数与成员函数的定义和使用区别如下所示：

```
class A{
private:
  int a;
public:
      int func(/*函数成员默认第一参数为对象本身，*/ int b){
            cout<<this->a<<", "<<b<<endl;
            return a;
      }
      friend int func(A& a/*参数必须显式地列在友元函数的参数表中*/, int b){
            cout<<a.a<<", "<<b<<endl;
            return a.a;
      }
};
```

```
int main(){
      A a;
      a.func(1);
      func(a,2);
      return 0;
}
```

友元函数与成员函数

```
class complex{
        double  real;                   //复数实部
        double  imag;                   //复数虚部
public:
        complex  ();                    //无参构造函数
        complex (double r, double  i);  //两参构造函数
        complex addCom (complex c2);    //调用者对象与对象 c2 相加
        void  outCom  ();               //输出调用者对象的有关数据(各分量)
};
```

成员函数

```
complex ();
complex (double r, double i);
friend complex addCom(complex c1, complex c2);
        //友元函数，实现复数 c1+c2（两参数对象相加）
friend void outCom (complex c);
        //友元函数，输出 complex 类对象 c 的有关数据（各分量）
```

友元函数

友元类

- 它可能是与 A无关的另外一个类。
- **友元声明只能出现在类定义中。因为友元不是授权类的成员，所以它不受其所在类的声明区域public private 和protected 的影响。友元类要在类外说明。**
- **友元类的任一函数都有权访问和调用申明该类的所有成员，包括私有及保护成员。**
- 友元关系**不能被继承**。
- 友元关系**是单向的，不具有交换性**。若类B是类A的友元，类A不一定是类B的友元，要看在类中是否有相应的声明。
- 友元关系**不具有传递性**。若类B是类A的友元，类C是B的友元，类C不一定是类A的友元，同样要看类中是否有相应的申明

```
class A {
private:
        friend class B;
public: A() {}
};
```

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

友元类

- 当说明类为别的类的友元类，则类的所有函数都可以访问对应类的所有成员。
- 友元类的说明与使用还应注意的是它具有“单方向”及“不传递”等特点：
  - 单方向：若 ClaA具有友元类 ClaB，并不意味着ClaB也具有友元类 ClaA（“非相互”）。
  - 不传递：若 ClaA具有友元类 ClaB（即 ClaB是 ClaA的友元类），又 ClaB具有友元类ClaC（即 ClaC 是 ClaB的友元类），并不意味从 ClaC 可以直接存取 ClaA的私有成员。
- 简单来说就是：**我把我家钥匙给你，不意味着你就必须把你家的钥匙给我，同时也不意味着你能随意给别人。**

```
class A {
private:
        friend class B;
public: A() {}
};
```

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

随机数生成

代码示例

随机种子设置与随机数生成

```cpp
#include <iostream>
#include <random>

int main() {
    std::mt19937 generator(0);
    int n;
    char s;
    std::cin>>n>>s;
    if (s == 'i') {
        std::uniform_int_distribution<int> distribution(0, 100);
        for (int i = 0; i < n; ++i) {
            std::cout << distribution(generator) << std::endl;
        }
    } else if (s == 'f') {
        std::uniform_real_distribution<float> distribution(0.0, 1.0);
        for (int i = 0; i < n; ++i) {
            std::cout << distribution(generator) << std::endl;
        }
    }else
        {
            std::cout<<"error"<<std::endl;
            return 0;
        }

    return 0;
}
```