第七章 类

7.0 引子

类的基本思想是数据抽象和封装。

数据抽象依赖接口和实现分离。其中接口包括用户能使用的操作；类的实现则包括数据成员、负责接口实现的函数体以及定义类所需的私有函数。

封装则实现了接口与实现的分离，类在封装后隐藏了实现细节。

我们为了实现抽象和封装，就需要定义一个抽象数据类型。

7.1 定义抽象数据类型

7.1.1 设计sales_data类

设计类的接口时，要考虑如何使类易于使用。当使用类时，不需要顾及类的实现机理与细节。

7.1.2 定义改进的sales_data类

定义在类内部的函数是隐式的inline函数
成员函数的声明必须在类内部，定义则既可以在类内部，也可以在外部。
成员函数通过名为this的隐式参数访问对象，在调用成员函数时，用请求该函数的对象地址初始化this。
const 成员函数不会修改类的数据成员。
const 成员函数的声明和定义处都要加 const，声明方式：

```
int age() const;//在类内的声明
int Student::age() const {};//在类外的定义
```

使用 const 成员函数要注意，**不能用它调用本类的非 const 成员函数**，调用的也必须是 const 成员函数。

7.1.3 定义类相关的非成员函数

如果非成员函数是类接口的组成部分，那么这些函数的声明应该与类在同一个头文件内。

7.1.4 构造函数

构造函数用来初始化类对象的数据成员。当类的对象被创建时，就会执行构造函数
构造函数没有返回类型、可以重载，且不能被声明为const的，因为构造函数要修改类的成员变量。
在C++11新标准中，可以在参数列表后加上=default来要求编译器生成默认构造函数。

7.2 访问控制与封装

class 和 struct 的**唯一一点区别**就是默认访问权限不同。

用访问控制符加强类的封装性：public后的 成员在程序内可以被访问，private后的成员只能被类的成员函数访问。

当希望类的所有成员是 public 时，用struct。

7.2.1 友元

类可以允许其他类或函数访问它的非公有成员，方法就是令其他函数或类成为它的**友元**。要将一个函数作为类的友元，只需在类内部加一条关键字 friend 开头的函数声明语句即可。

友元声明只能出现在类的内部，但是具体位置不限，不是类的成员，**不受public、private 限制**。

友元的声明

类内对友元的声明只是指定了访问权限，并**不是通常意义上的函数声明**。如果要调用友元函数，还需要在类的外部再次声明。**并且要在调用位置之前声明。**

7.3 类的其他特性

7.3.1 类成员再探

定义类型成员

类可以自定义某种类型在类内的别名。类型成员一样有访问限制。

```
 typedef string::size_type pos;
// using pos = string::size_type;//使用类型别名，两种方式都可以
```

类型成员必须先定义后使用，因此类型成员应该出现在类开始的地方。

类内初始值

成员变量可以在类内定义的时候直接初始化。

此时构造函数的初始化列表可以不包含该成员变量，隐式使用其类内初始值。

类内初始值必须使用等号或花括号初始化。

内联成员函数

内联函数最初的目的：代替部分 #define 宏定义。

使用内联函数替代普通函数的目的：提高程序的运行效率。

4种方式使成员成为内联函数：

1. 在类内定义函数，为隐式内联。
2. 在类内用关键字 inline 显式声明成员函数。
3. 在类外用关键字 inline 定义成员函数。
4. 同时在类内类外用 inline 修饰

inline 成员函数应该与类定义在同一个头文件中

7.3.2 返回*this的成员函数

this 指针指向类，即 this 是类的地址，*this 就是类本身。

可以定义**返回类型为类对象的引用的函数**。如果定义的返回类型不是引用，返回的就是 *this 的副本了。

const 函数如果以引用的形式返回 this，返回类型就是一个常量引用。

7.3.3 类类型

一个类的成员类型不能是它自己，但是类允许包含指向它自身类型的引用或指针。

7.3.4 友元再探

可以把其他的类定义成友元，也可以把其他类的成员函数定义成友元。

如果一个类指定了友元类。则友元类的成员函数可以访问此类的所有成员。

友元关系不具有传递性。

重载函数名字相同，但是是不同的函数。如果想把一组重载函数声明为类的友元，需要对每一个分别声明。

7.4 类的作用域

当类的成员函数的返回类型也是类的成员时，在定义它时要指明类

```
Student::age Student::Getage(){}
```

7.4.1 名字查找与类的作用域

普通程序名字查找的过程

1. 首先在名字所在的块中寻找声明语句
2. 如果没找到，继续查找外层作用域
3. 如果最终还是没找到，报错

类的定义过程

1. 首先，编译成员的声明。
2. 直到全部类可见后才编译函数体。

特殊：在类内定义的类型名要放在类的开始，放在后面其他成员是看不见的。

类型名如果在类外已经定义过，不能在类内重定义。

不建议使用其他成员的名字作为某个成员函数的参数。

7.5 构造函数再探

7.5.1 构造函数初始值列表

使用初始值列表对类的成员初始化才是真正的初始化，在构造函数的函数体内赋值并不是初始化。

如果定义构造函数，必须对类的所有数据成员初始化或赋值。

如果成员是 const **或者是引用的话，必须初始化。**

如果成员是类并且该类没有定义构造函数的话，必须初始化。（如果该类定义了构造函数的话，就不用了）

使用初始值列表初始成员时，成员初始化的顺序是按照类定义种出现的顺序初始化的。

默认实参和构造函数

如果一个构造函数为所有参数提供了默认实参，则**它实际上相当于定义了默认构造函数。**

7.5.2 委托构造函数

委托构造函数通过其他构造函数来执行自己的初始化过程。

```
class Student{
public:    Student(string nameIn,int ageIn):name(nameIn),age(ageIn){}
           Student():Student(" ",18){} //这就是委托构造函数
           Student(string s):Student(s,18){} //这也是委托构造函数
}
```

7.5.3 默认构造函数的作用

在实际中，当对象被默认初始化或值初始化时自动执行默认构造函数。或是类的某些数据成员缺少默认构造函数。那么除了定义了其他构造函数，**最好也提供一个默认构造函数。**

7.5.4 隐式的类类型转换

如果构造函数只接受一个实参，则称作**转换构造函数**，它实际上定义了转换为此类类型的隐式转换机制。

一个实参的构造函数定义了一条从构造函数的参数类型向类类型隐式转换的规则

只允许一步类型转换

在进行隐式转换时，编译器只会自动地执行一步类型转换。

```
string null_book = "9-999";
item.combine(null_book); //conbine 函数接受 Sales_data 类类型，但该类定义了
//一个接受 string 参数的转换构造函数，
//所以这里会执行从 string 到该类类型的隐式转换，是正确的。
item.combine("9-999"); //隐式地使用了两种转换规则，所以是错误的。
item.combine(string("9-999"));
//先显示地转换为 string，再隐式地转换为 Sales_data 类类型。是正确的。
```

explicit-抑制构造函数定义的隐式转换

将转换构造函数声明为 explicit 会阻止隐式转换。

关键字 explicit 只对一个实参的构造函数有效。因为需要多个实参的构造函数本来就不执行隐式转换。

explicit 只在类内声明构造函数时使用，在类外定义时不加。类似 static 成员函数

```
 class Sales_data {
 public:
                explicit Sales_data(const string& s) : bookNo(s) { }
//不能再执行从 string 到 Sales_data 的隐式转换。
 private
                string bookNo;
 }
 item.combine(null_book);   //错误，不能执行从 string 到 Sales_data 的隐式初始化
```

explicit 构造函数只能用于直接初始化

explicit 构造函数只能用于直接初始化，不能用于使用 "=" 的拷贝初始化。因为“=”实际上是采用了拷贝赋值运算符，在传参时会进行隐式转换。

不加 explicit 的转换构造函数，可以在赋值、传参、从函数返回等场合执行隐式转换，加了 explicit 后，就不能隐式转换了，也就是加了 explicit 的转换构造函数的意义就只是定义了一个新的构造函数，不具有提供隐式转换机制的额外功能了。

```
Sales_data item1(null_book);  //正确
Sales_data item2 = null_book; //错误
```

7.5.5 聚合类

满足以下四个条件的类是聚合类：

1. 所有成员都是public的
2. 没有定义任何构造函数
3. 没有类内初始值
4. 没有基类和 virtual 函数

聚合类可以像结构体一样用花括号初始值列表初始化。如果花括号内元素数量少于类成员数量，靠后的成员将被值初始化。

```
Student stu = {"Li Ming",18};
```

---### 7.5.6 **字面值常量类**

constexpr 函数的参数和返回值都必须是字面值类型。

算术类型、引用和指针都是字面值类型，此外**字面值常量类**也是字面值类型。

字面值类型属于常量表达式，constexpr 就是用来声明常量表达式的。

聚合类属于字面值常量类。

如果不是聚合类，满足以下四个条件的类也是字面值常量类：

1. 数据成员都是字面值类型。
2. **类至少含有一个 constexpr 构造函数**
3. 如果一个数据成员有类内初始值，则初始值必须是常量表达式（如果成员是类，则初始值必须使用成员自己的 constexpr 构造函数）
4. 类必须使用析构函数的默认定义。

constexpr 构造函数

类的构造函数不能是 const 的，但字面值常量类的构造函数可以是 constexpr 函数。

constexpr 构造函数可以声明成 =default 或 =delete。

constexpr 构造函数的函数体应该是空的（原因：constexpr 函数的函数体只能包含一条返回语句，而构造函数不能包含返回语句）
constexpr 构造函数必须初始化所有数据成员。初始值必须是常量表达式或使用其自己的 constexpr 构造函数。
使用前置关键字 constexpr 来声明 constexpr 构造函数

7.6 类的静态成员

类的静态成员与类本身直接关联，而不是与类的对象保持关联。

静态成员可以是 public 或 private 的。

静态成员不与任何对象绑定在一起。可以是常量、引用、指针、类等。

静态成员函数不包含 this 指针，不能声明为 const 的，不能在 static 函数体内使用 this 指针。

因为 static 函数不能使用 this 指针，所以它是无法使用类的非 static 数据成员的。
使用作用域运算符可以直接访问静态成员。类的对象也可以直接访问静态成员

定义静态成员

可以在类内或类外定义静态成员。当在类外定义时，**不能重复 static 关键字**，static 只出现在类内的声明中。

只有 constexpr 类型的静态数据成员可以在类内初始化，但是也需要在类外定义。

其他的静态数据成员**都在类内声明，类外定义并初始化**。

静态成员可以用的特殊场景

静态数据成员可以是**不完全类型**，比如静态数据成员的类型可以是它所属的类类型本身。

静态成员可以作为默认实参。