A.3 默认函数

删除函数的函数可以不进行实现，默认函数就则不同：编译器会创建函数实现，通常都是“默认”实现。当然，这些函数可以直接使用(它们都会自动生成)：默认构造函数，析构函数，拷贝构造函数，移动构造函数，拷贝赋值操作符和移动赋值操作符。

为什么要这样做呢？这里列出一些原因：

- 改变函数的可访问性——编译器生成的默认函数通常都是声明为public(如果想让其为protected或private成员，必须自己实现)。将其声明为默认，可以让编译器来帮助你实现函数和改变访问级别。
- 作为文档——编译器生成版本已经足够使用，那么显式声明就利于其他人阅读这段代码，会让代码结构看起来很清晰。
- 没有单独实现的时候，编译器自动生成函数——通常默认构造函数来做这件事，如果用户没有定义构造函数，编译器将会生成一个。当需要自定一个拷贝构造函数时(假设)，如果将其声明为默认，也可以获得编译器为你实现的拷贝构造函数。
- 编译器生成虚析构函数。
- 声明一个特殊版本的拷贝构造函数，比如：参数类型是非const引用，而不是const引用。
- 利用编译生成函数的特殊性质(如果提供了对应的函数，将不会自动生成对应函数——会在后面具体讲解)。

就像删除函数是在函数后面添加 `= delete` 一样，默认函数需要在函数后面添加 `= default` ，例如：

```
class Y
{
private:
  Y() = default;  // 改变访问级别
public:
  Y(Y&) = default;  // 以非const引用作为参数
  T& operator=(const Y&) = default; // 作为文档的形式，声明为默认函数
protected:
  virtual ~Y() = default;  // 改变访问级别，以及添加虚函数标签
};
```

编译器生成函数都有独特的特性，这是用户定义版本所不具备的。最大的区别就是编译器生成的函数都很简单。

列出了几点重要的特性：

- 对象具有简单的拷贝构造函数，拷贝赋值操作符和析构函数，都能通过memcpy或memmove进行拷贝。
- 字面类型用于constexpr函数(可见A.4节)，必须有简单的构造，拷贝构造和析构函数。
- 类的默认构造，拷贝，拷贝赋值操作符合析构函数，也可以用在一个已有构造和析构函数(用户定义)的联合体内。
- 类的简单拷贝赋值操作符可以使用 `std::atomic<>` 类型模板(见5.2.6节)，为某种类型的值提供原子操作。

仅添加 `= default` 不会让函数变得简单——如果类还支持其他相关标准的函数，那这个函数就是简单的——不过，用户显式的实现就不会让这些函数变简单。

第二个区别，编译器生成函数和用户提供的函数等价，也就是类中无用户提供的构造函数可以看作为一个aggregate，并且可以通过聚合初始化函数进行初始化：

```
struct aggregate
{
  aggregate() = default;
  aggregate(aggregate const&) = default;
  int a;
  double b;
};
aggregate x={42,3.141};
```

例子中，x.a被42初始化，x.b被3.141初始化。

第三个区别，编译器生成的函数只适用于构造函数；换句话说，只适用于符合某些标准的默认构造函数。

```
struct X
{
  int a;
};
```

如果创建了一个X的实例(未初始化)，其中int(a)将会被默认初始化。

如果对象有静态存储过程，那么a将会被初始化为0；另外，当a没赋值的时候，其不定值可能会触发未定义行为：

```
X x1;  // x1.a的值不明确
```

另外，当使用显示调用构造函数的方式对X进行初始化，a就会被初始化为0：

```
X x2 = X();  // x2.a == 0
```

这种奇怪的属性会扩展到基础类和成员函数中。当类的默认构造函数是由编译器提供，并且一些数据成员和基类都是有编译器提供默认构造函数时，还有基类的数据成员和该类中的数据成员都是内置类型的时候，其值要不就是不确定的，要不就是被初始化为0(与默认构造函数是否能被显式调用有关)。

虽然这条规则令人困惑，并且容易造成错误，不过也很有用；当你编写构造函数的时候，就不会用到这个特性；数据成员，通常都可以被初始化(指定了一个值或调用了显式构造函数)，或不会被初始化(因为不需要)：

```
1 X::X():a(){}  // a == 0
2 X::X():a(42){}  // a == 42
3 X::X(){}  // 1
```

第三个例子中①，省略了对a的初始化，X中a就是一个未被初始化的非静态实例，初始化的X实例都会有静态存储过程。

通常的情况下，如果写了其他构造函数，编译器就不会生成默认构造函数。所以，想要自己写一个的时候，就意味着你放弃了这种奇怪的初始化特性。不过，将构造函数显示声明成默认，就能强制编译器为你生成一个默认构造函数，并且刚才说的那种特性会保留：

```
X::X() = default;  // 应用默认初始化规则
```

这种特性用于原子变量(见5.2节)，默认构造函数显式为默认。初始值通常都没有定义，除非具有(a)一个静态存储的过程(静态初始化为0)，(b)显式调用默认构造函数，将成员初始化为0，(c)指定一个特殊的值。注意，这种情况下的原子变量，为允许静态初始化过程，构造函数会通过一个声明为constexpr(见A.4节)的值为原子变量进行初始化。