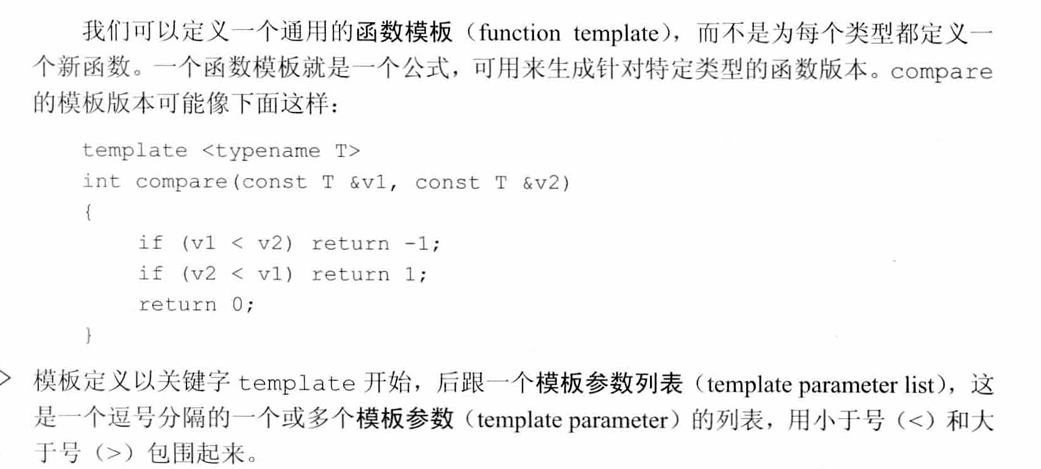
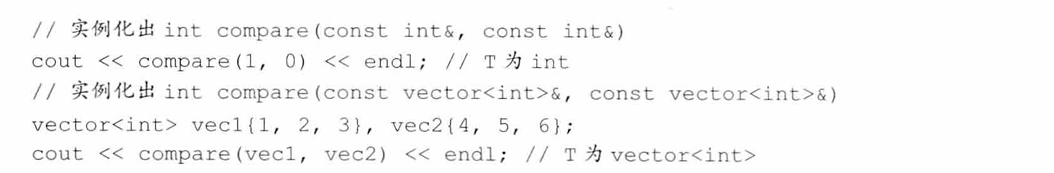
定义一个函数模板，下面例子中的T是一个类型参数，在编译时根据使用情况来确定T代表的类型

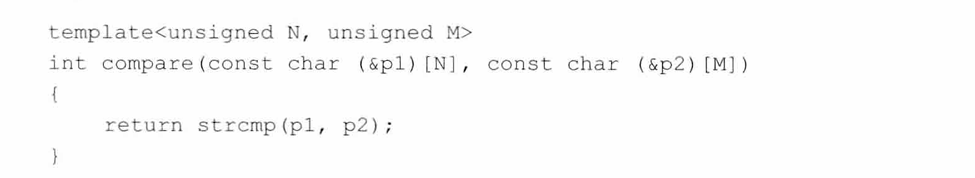


当调用一个函数模板时，编译器通常用函数实参推断模板实参。下面是书上的例子：



模板类型参数就是模板参数，模板参数可以用来指定返回类型或函数的参数类型。模板参数前必须使用关键字 class 或 typename，两个含义相同，但使用typename关键字显得更直观。编译器根据实参类型推断生成的版本，被称为模板的实例。

模板中还可以定义非类型参数。一个非类型参数表示一个值。非类型参数使用特定类型名（如 int，double 等）而非 typename 来指定。下面是书上的例子：

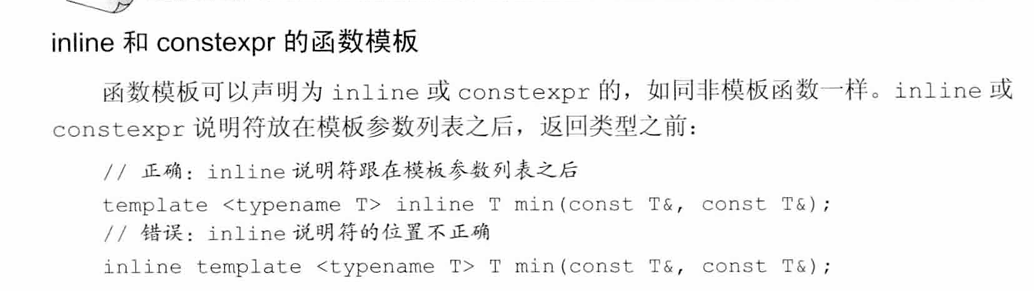


当一个模板实例化时，非类型参数被用户提供的值所代替，这些值必须是常量表达式，以允许编译器在编译时实例化模板。

非类型参数可以是一个整型或指针或引用。如果是指针或引用，绑定的实参必须有静态的生存期。

数组的大小是数组类型的一部分，所以数组做函数参数时其大小必须是固定的，可以通过非类型模板参数数组其大小可变。

inline和constexpr关键字放在函数模板中的位置，放在模板参数列表之后，下面是书上的例子：



编写泛型代码有两个重要原则：

1.模板中的函数参数应该是 const 的引用。引用保证了泛型函数可以用于不能拷贝的类型，如unique\_ptr, IO 类型。

2.函数体中的条件判断仅使用 < 比较运算。

ps：关于上面第二点，这么做的原因如下：

1.可以简化和统一代码的实现。许多泛型算法（如排序算法）仅需要一个基本的比较功能，通过约定只使用 < 运算符，可以避免复杂的逻辑判断和代码冗余。

2.兼容性好。几乎所有可以比较的类型都会定义 < 运算符。

编译器遇到模板定义时不生成代码，当实例化出模板的一个特定版本时才生成代码。这会影响错误何时被检测到（大多数编译错误在实例化期间报告）。

定义类时，普通的成员函数一般放在源文件中。但是模板不同，模板的头文件通常既包括声明也包括定义。因为编译器需要知道函数模板或类模板成员函数的完整定义才能进行实例化。

因为模板直到实例化时才生成代码，所以错误可能出现在：

1.编译模板本身时。此时只能检测到语法错误。

2.编译器遇到模板使用时。只能检测模板实参的类型是否与形参相匹配。

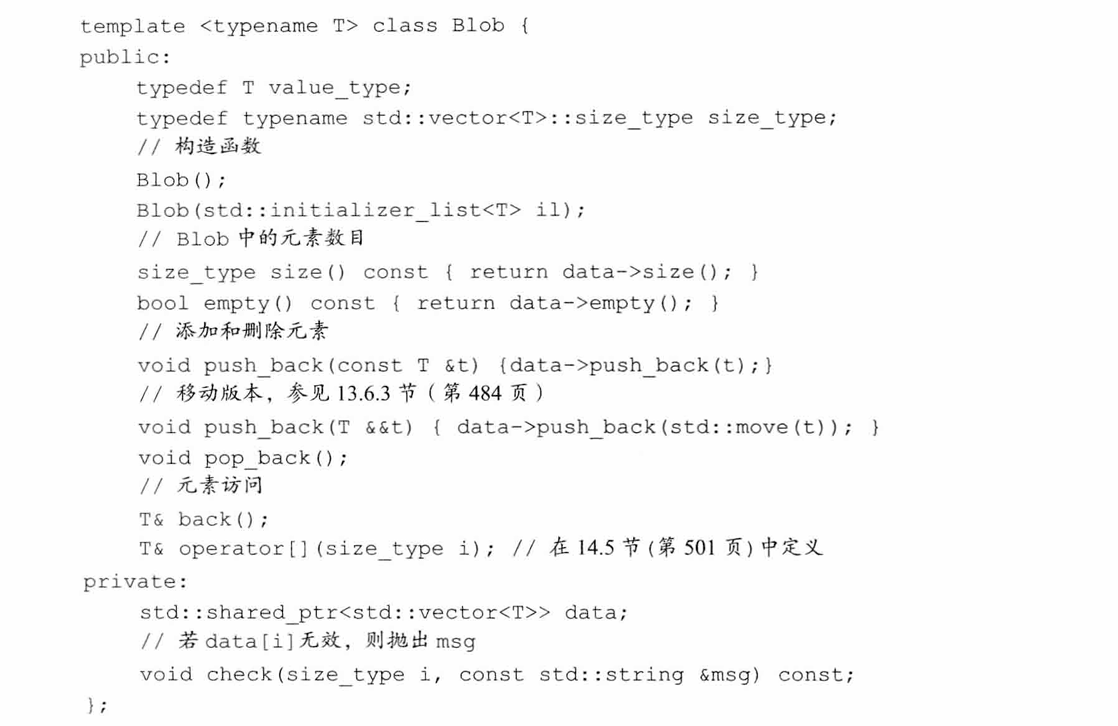
3.模板实例化时。这时才能发现类型相关的错误。

类模板

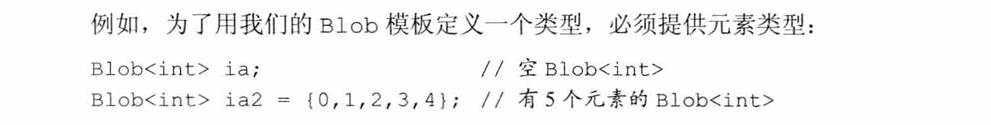
类模板不同于函数模板的地方在于编译器不能为类模板推断参数类型，因此必须在模板名后的尖括号内提供额外信息。

ps：类模板的名字不是类型名。类模板用来生成一个具体的类型，具体的模板类型总是包含模板参数。

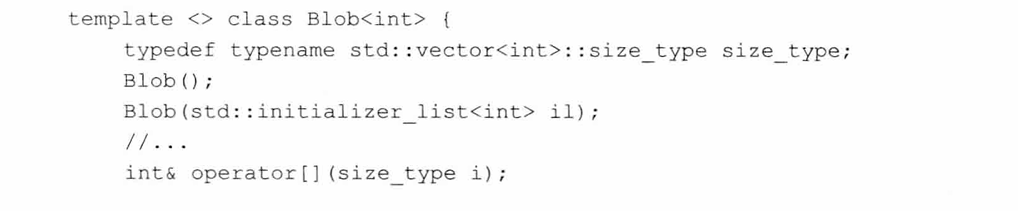
书上定义类模板的例子：

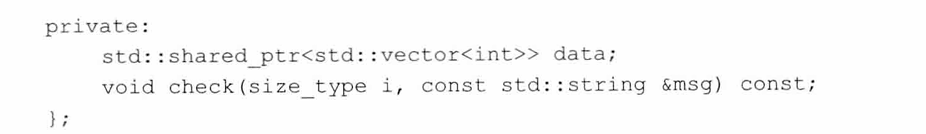


对上面那个模板类的实例化：



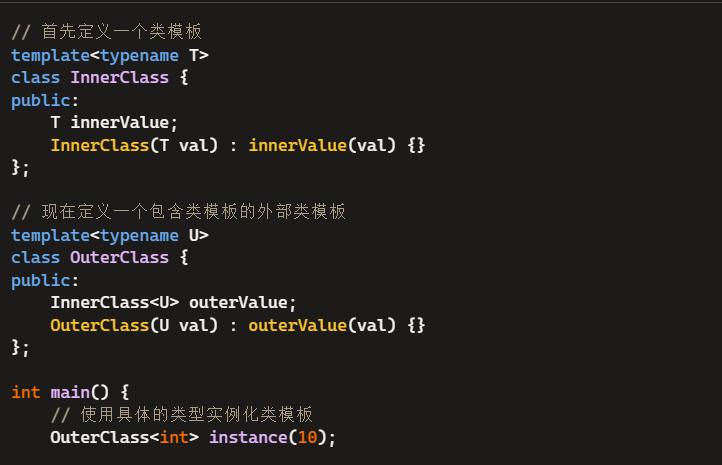
基于模板实参，编译器创建出下面的模板实例：





模板嵌套

在一个类模板中，当我们使用另一个模板作为成员或函数时，通常不会使用具体的类型或值作为模板参数，而是使用当前类模板的模板参数。即当定义一个类模板并在其内部使用另一个模板时，内部模板的参数通常会与外部模板的参数相关联。下面有个例子：



可以得知T和U都是int类型

image.png

类模板的成员函数

可以在类模板内部，也可以在外部定义成员函数。定义在类模板内部的函数隐式声明为内联函数。

定义在类模板之外的成员函数必须以关键字 template 开始，后接类模板参数列表。

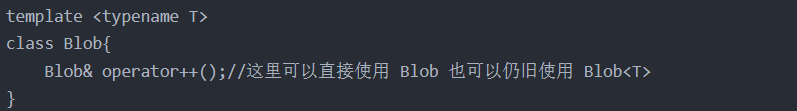
image.png

默认情况下，一个类模板的成员函数只有当用到它时才进行实例化，在未调用之前不会被实例化。

image.png

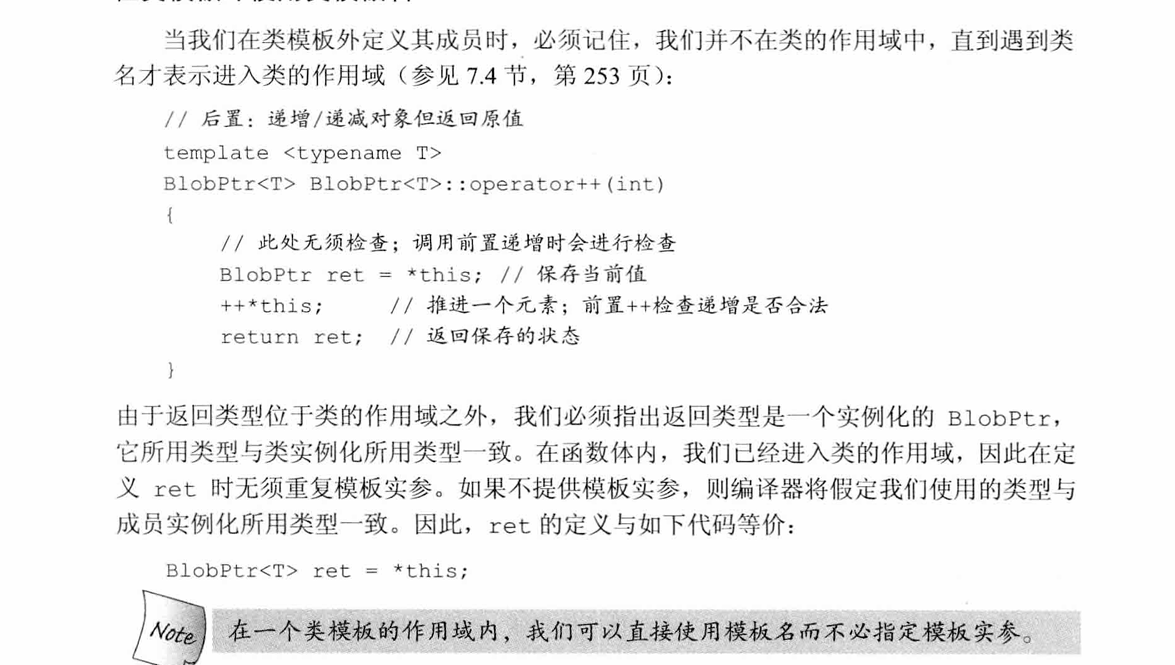
模板类内使用类名

在类代码内能够简化模板类名的使用——使用一个类模板类型时必须提供模板实参，但有一个例外：在类模板自己的作用域内部，可以直接使用模板名而不提供实参。下面是一个例子：



在类模板外使用类模板名

在类模板外使用类模板名必须提供模板实参。



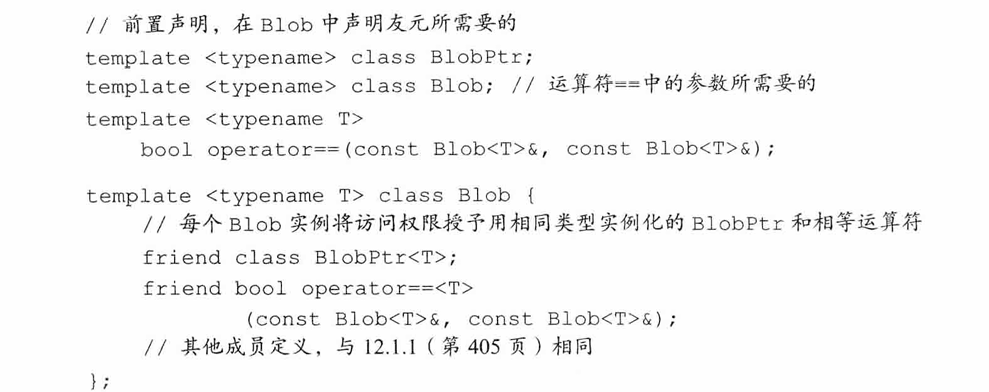
类模板和友元

如果一个类模板包含一个非模板友元，则该友元可以访问该模板的所有实例。

如果友元也是模板，类可以授权给所有友元模板实例，也可以只授权给特定实例。

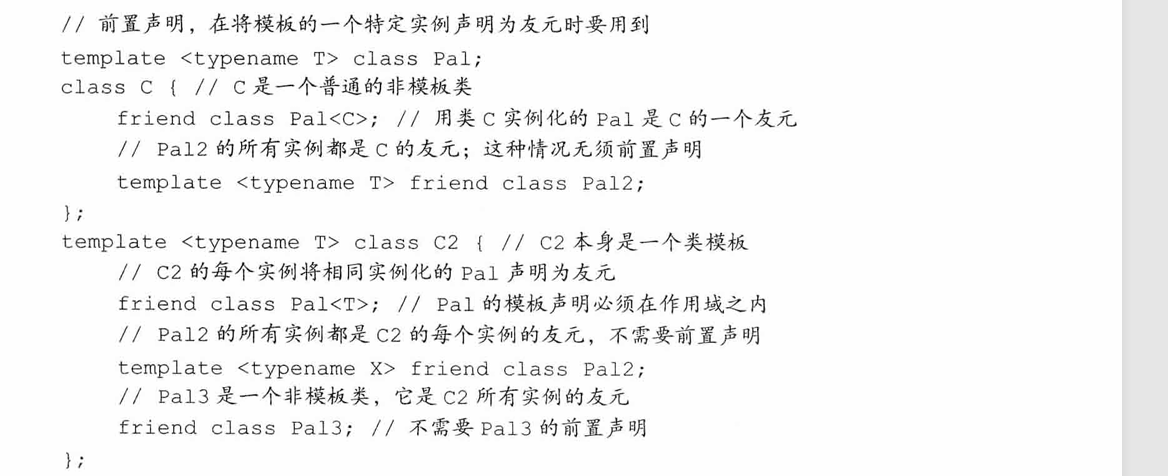
一对一友好关系

类模板与另一个模板间友好关系的最常见形式是建立对应实例和及其友元间的友好关系。下面是书上的例子：

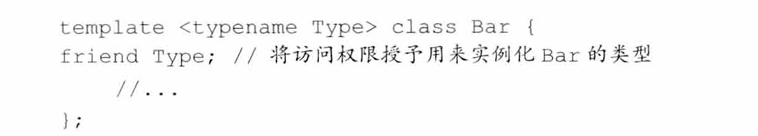


通用和特定模板的友好关系

一个非模板类可以将模板类的所有实例声明为友元，或者限定特定的模板类实例为友元。下面是树上的例子：



令模板自己的类型参数成为友元：



类模板的 static 成员

如果类模板定义了 static 成员，那么模板生成的每个具体的类都有自己独有的 static 成员实例。

static 数据成员定义时也要定义为模板。类似其他成员函数，static成员函数只有在被使用时才会实例化。

模板参数与作用域

模板参数的可用范围是其声明之后，至模板声明或定义结束之前。

模板参数会隐藏外层作用域中的相同名字，但是注意在模板内不能重用模板参数名。

image.png

使用类的类型成员

默认情况下，C++ 假定通过作用域运算符访问的名字不是类型（比如可能是静态成员）。

如果希望使用一个模板类型参数的类型成员，必须使用关键字 typename 显式地告诉编译器该名字是一个类型。这样可以减少开销。

默认模板实参

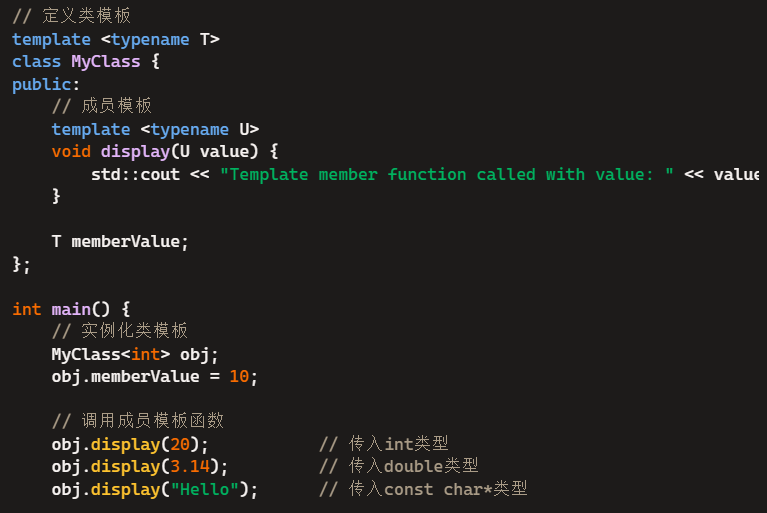
可以为函数模板和类模板的模板参数提供默认模板实参，就像可以为函数参数提供默认实参一样。

image.png

成员模板

一个类（无论是普通类还是类模板）可以包含本身是模板的成员函数。这种成员称为成员模板。

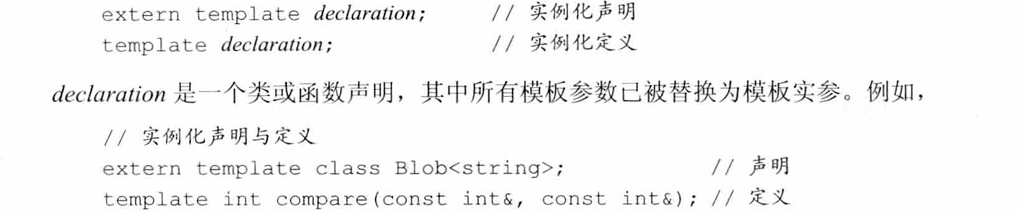
这样能获得更多的灵活性，在模板类中声明成员模板的例子：



使用模板类时传入模板实参创建模板类型和实例，调用方法时通过传入参数运行时动态推断模板参数

控制实例化

当两个及以上的源文件使用了相同的模板并提供相同的模板实参时，实际上会通过模板类得到的具体的类创建两个实及以上的实例，此时使用显式实例化可以避免这种开销（只生成一个实例）。



实例化实例的同时会实例化该模板的所有成员

模板实参推断

从函数实参来确定模板实参的过程被称为模板实参推断。

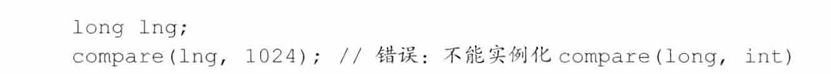
能应用于函数模板的类型转换：

1.非const引用可以传递给const引用

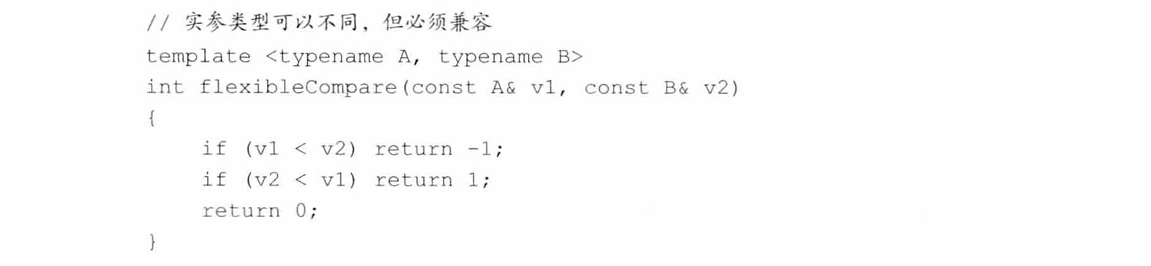
2.如果形参不是引用类型，可以把数组实参转换为指向首元素的指针，把函数实参转换为指向函数的指针。

使用相同的模板参数时，推断错误导致报错的解决方式：

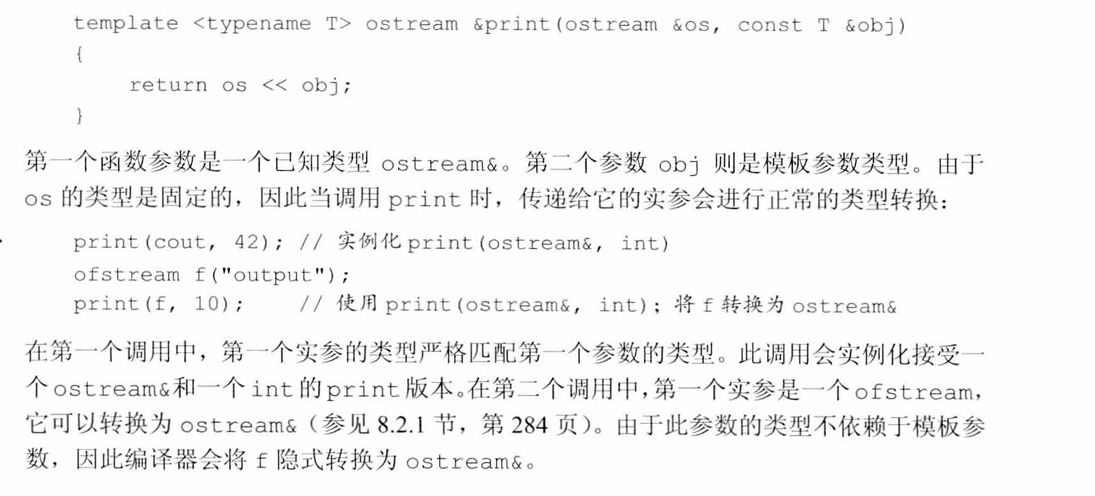
下面的例子中compare的两个形参都是const T&类型，此时lng推断为long类型，1024推断为int会报错，就像下面这样



解决方式是增加一个模板参数，现在的compare方法变成下面的样子，此时推断得到long和int就不会报错了

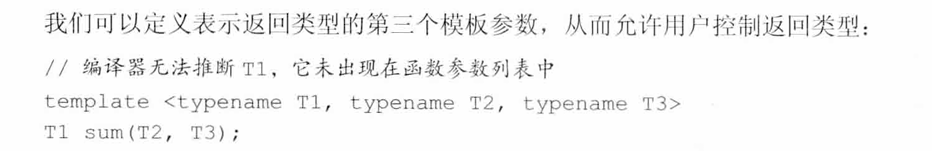


通过推断得到的模板参数类型，正常的已确定的参数也可以进行类型转换

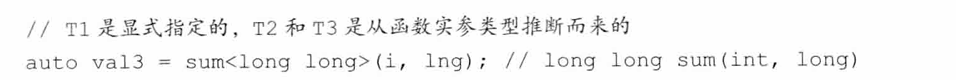


为函数模板指定显式模板实参

下面这个例子中无法推断出T1的类型，因此每次调用sum时必须使用尖括号为T1传入模板实参



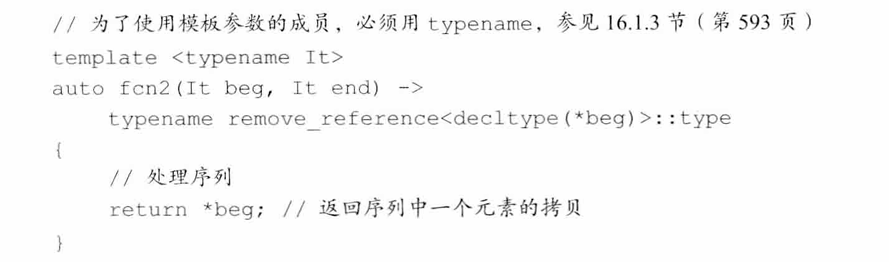
指定T1为long long类型，T2和T3通过传入的实参推断得到



尾指返回类型和实参推断

还是上面那个sum函数，有时无法确定返回类型，因此无法在调用时为T1传入模板实参，此时需要在参数列表之后声明返回类型。

下面是书上的例子：



这里必须使用尾置类型的原因是，在编译器遇到函数的参数列表之前，beg 都是不存在的。



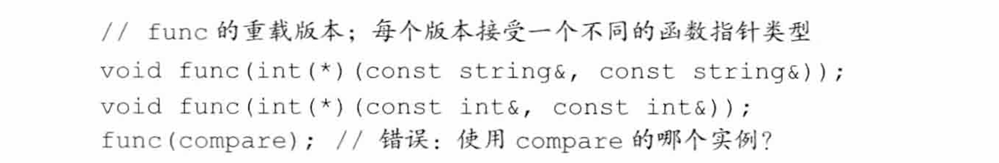
函数指针的实参推断

当我们用一个函数模板初始化一个函数指针或为一个函数指针赋值时，编译器使用指针的类型来推断模板实参。pf1 中参数的类型决定了

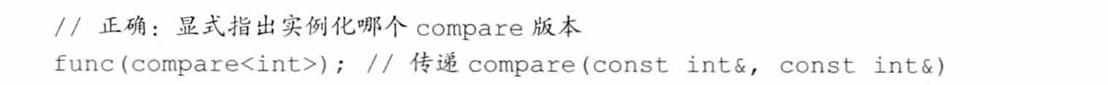
T 的模板实参的类型。



此时会存在歧义的问题，就比如下面这样



通过显式模板实参消除歧义



模板实参推断和引用

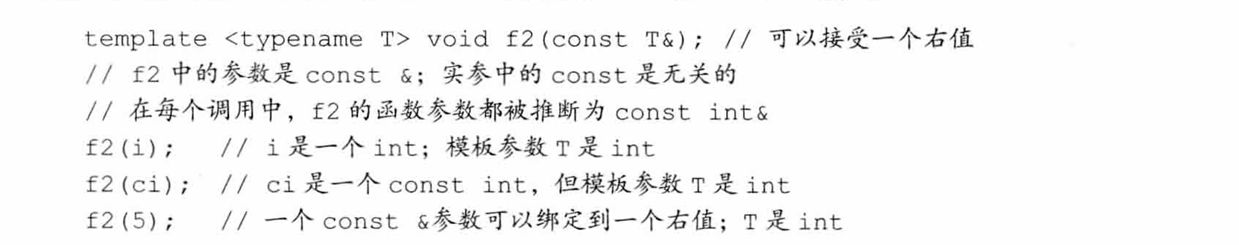
从左值引用函数参数推断类型

当一个函数参数是模板类型参数的一个普通（左值）引用时（即，形如 T&），绑定规则告诉我们，只能传递给它一个左值（如一个变量或一个返回引用类型的表达式）。实参可以是 const 类型，也可以不是，因此出于安全需求（例如临时对象生命周期不可预测，会引发未定义行为）只能被绑定到左值。如果实参是 const 的，则 T 将被推断为 const 类型。下面是例子：



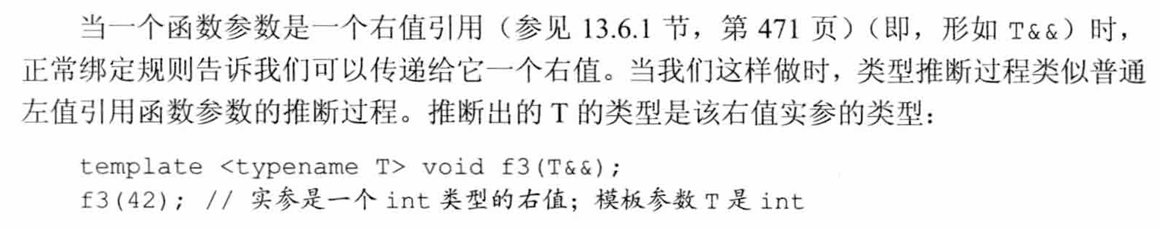
ps：字面值是右值，因为被看作临时变量。

而参数类型为const T&时，对象的生命周期被延长到引用的声明周期，且const确保引用的对象不会因为被修改而引发未定义行为。此时const T&可以绑定右值和左值，就像书上的例子：



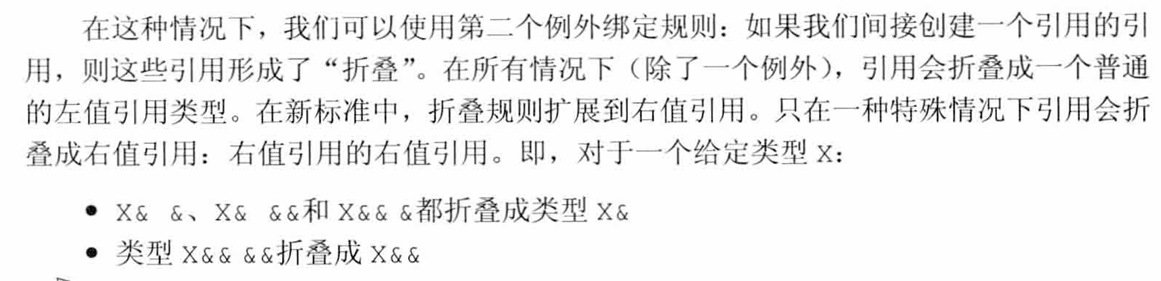
从右值引用函数参数推断类型

下面例子推断出是一个右值引用类型

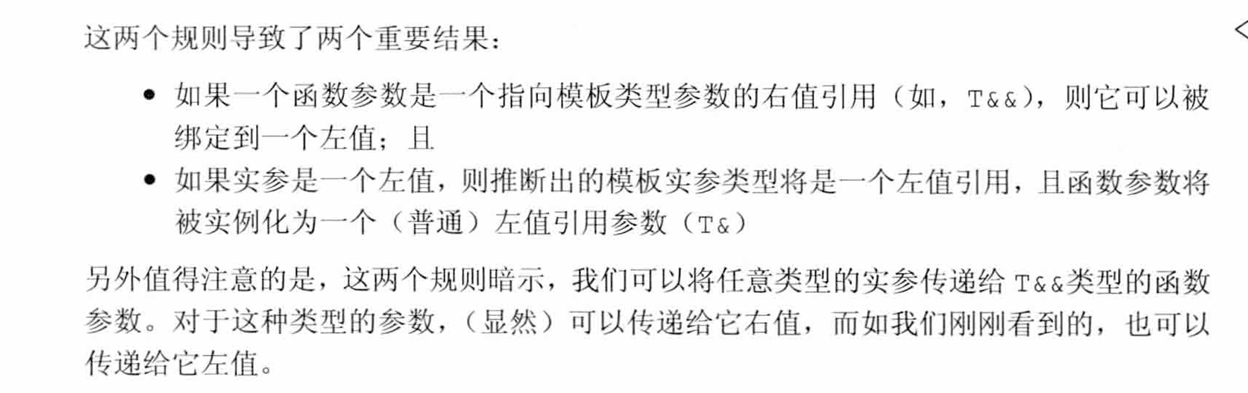


通过C++提供的移动方法，将右值引用绑定到左值引用

引用折叠



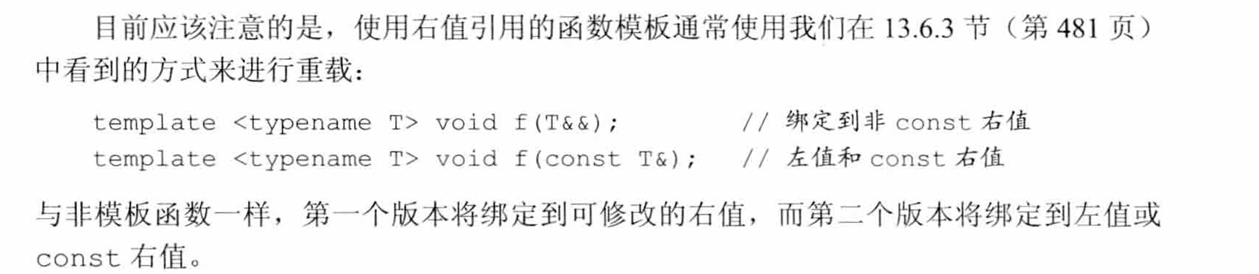
因此此时将右值引用绑定到左值引用，推断出的类型是T&



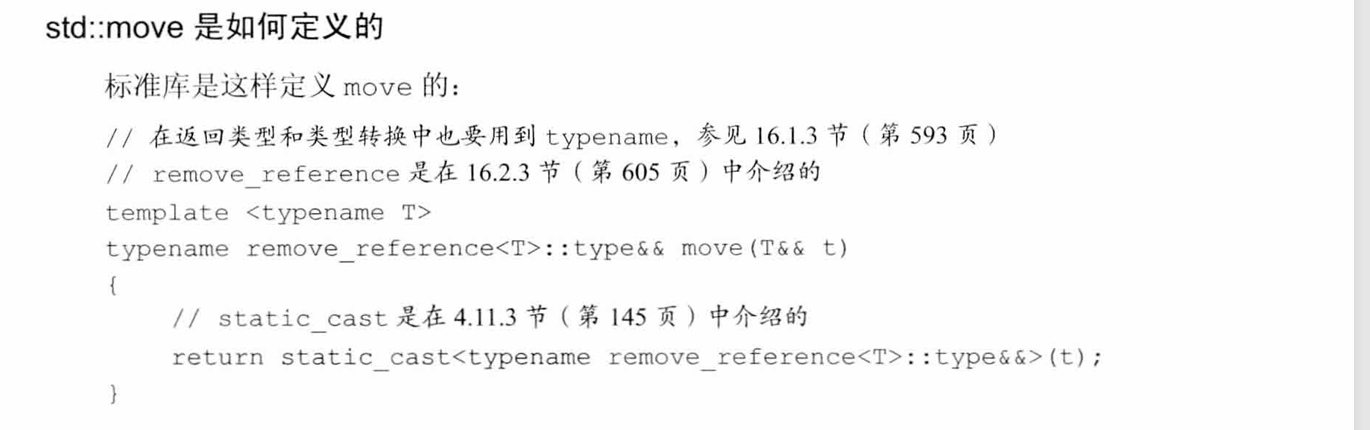
这是一个接受右值引用参数函数的例子，这个函数不能接受const右值：

image.png

使用右值引用函数模板通常需要重载一个接受左值引用和const右值的版本

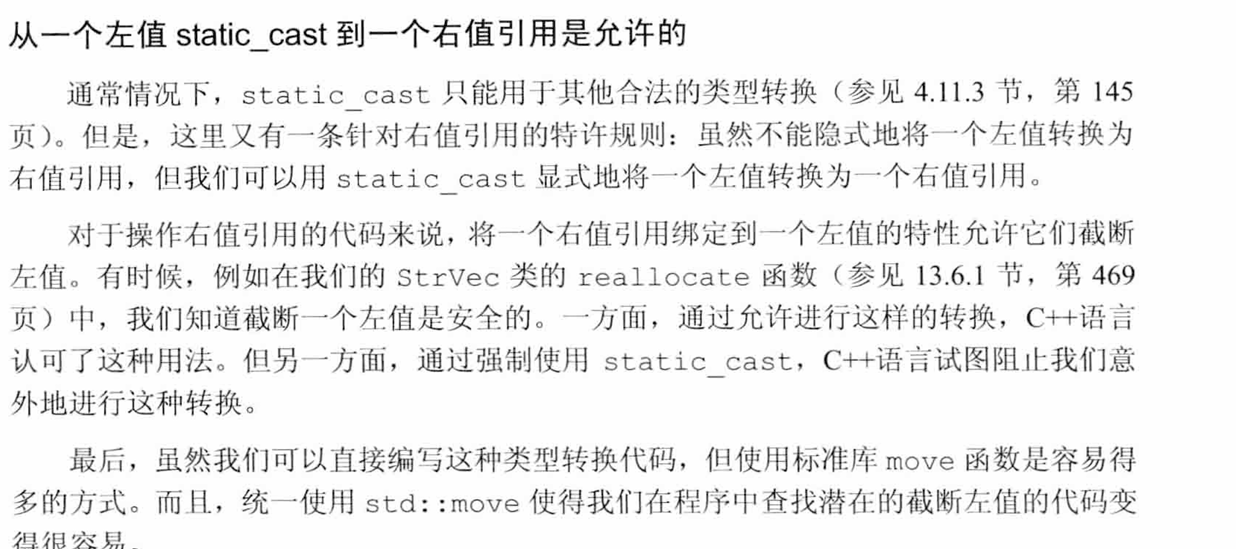


std::move方法的定义



关于static\_cast强制转换，能将一个左值引用显式转换为右值引用

ps：std::move方法的返回值实际为T& &&，被折叠为T&



重载与模板

函数模板可以被另一个模板或普通非模板函数重载。名字相同的函数必须具有不同数量或类型的参数。



可变参数模板

一个可变参数模板就是一个接受可变数目参数的模板函数或模板类。可变数目的参数被称为参数包。存在两种参数包：模板参数包，表示零个或多个模板参数；函数参数包，表示零个或多个函数参数。

书上的例子：

