普通高等教育“十一五”国家级规划教材　计算机系列教材

# C++泛型STL原理和应用

任哲　房红征　张永忠　编著

清华大学出版社

普通高等教育“十一五”国家级规划教材

计算机系列教材

任哲 房红征 张永忠 编著

# C++泛型
# STL原理和应用

清华大学出版社
北京

## 内 容 简 介

本书先在阐述泛型基本概念的基础上，比较详细和全面地介绍 C++ 泛型实现的基本技术和基本机制，然后介绍 STL 的泛型实现技术及其应用方法。本书在内容选材及编写上注意泛型以及 STL 初学者的特点，语言通俗易懂，精练而不枯燥；以仿真的方式介绍 STL 的核心内容，从而达到理论和应用并重的学习效果。

本书是一本理论和应用兼顾，适合泛型设计及 STL 初学者阅读的读物，鉴于它的特点，也适合作为在校计算机专业、软件工程专业或与之相关专业的教材。



**图书在版编目(CIP)数据**

C++ 泛型：STL 原理和应用/任哲，房红征，张永忠编著. --北京：清华大学出版社，2016
计算机系列教材
ISBN 978-7-302-42175-7

Ⅰ. ①C…　Ⅱ. ①任… ②房… ③张…　Ⅲ. ①C 语言－程序设计－教材　Ⅳ. TP312

中国版本图书馆 CIP 数据核字(2015)第 272720 号

**责任编辑**：汪汉友　徐跃进
**封面设计**：常雪影
**责任校对**：焦丽丽
**责任印制**：王静怡

**出版发行**：清华大学出版社
**网　　址**：http://www.tup.com.cn，http://www.wqbook.com
**地　　址**：北京清华大学学研大厦 A 座　　**邮　　编**：100084
**社 总 机**：010-62770175　　**邮　　购**：010-62786544
**投稿与读者服务**：010-62776969，c-service@tup.tsinghua.edu.cn
**质量反馈**：010-62772015，zhiliang@tup.tsinghua.edu.cn
**课件下载**：http://www.tup.com.cn,010-62795954
**印 刷 者**：北京富博印刷有限公司
**装 订 者**：北京市密云县京文制本装订厂
**经　　销**：全国新华书店
**开　　本**：185mm×260mm　　**印　张**：22　　**字　　数**：537 千字
**版　　次**：2016 年 3 月第 1 版　　**印　　次**：2016 年 3 月第1 次印刷
**印　　数**：1～2000
**定　　价**：44.50 元

产品编号：065543-01

不止一次被问：什么是泛型？什么是 STL？泛型与 STL 有关系吗？

泛型和 STL 不仅有关系，而且有很重要的关系：泛型是一种设计思想，而 STL 则是泛型思想在 C++ 上所取得的成果。

所谓泛型就是一种更抽象、更宽泛的数据类型，用术语来说就是参数化了的数据类型，通俗点说就是一种“通用”数据类型。泛型就是企图使用这种“通用”类型来摆脱数据类型对代码复用的限制，从而提高软件开发效率。而 C++ 则依靠它的模板技术实现了泛型思想，C++ 把常用的模板都制作成便于用户应用的例程提供在模板类库中，这个模板类库就叫做 STL(Standard Template Library)。所以可以这样说：STL 是泛型思想的 C++ 实现。

泛型是不是很难？本科生有必要学习它吗？

泛型并不难，它实质上是人们把计算机语言向人类语言发展的一个重要步骤。也就是说，泛型更接近人类语言，所以从应用的角度来说它一点都不难。但是，要想把泛型(在 C++ 中就是模板及其应用技术)的工作机理弄明白也不是一件简单的事情，因为一般人很少能直接接触到数据类型的推导和处理这些以前全部由编译器做的工作，除非他搞过系统软件的设计。

那么本科生有必要学习它吗？能把 STL 学明白吗？

作者的回答是：绝对有必要，因为它很有用，在各方面都很有用。

首先，STL 极具应用价值，因为大量的常规数据结构的维护管理工作它都自动完成了，用户可以把自己的精力完全集中于自己的业务逻辑上来，从而极大地提高应用程序的开发效率。

其次，由于 STL 是泛型设计的一个典型实现，而泛型设计又是软件发展的一个重要方向，因此作为打基础的本科学习阶段，学生必须对泛型设计的相关概念及技术具有一定程度的了解。特别是学习 C/C++ 语言程序设计之后，完全有必要让学生在对面向对象和面向过程程序设计思想混合应用的 STL 进行一番设计思想的学习和体验，从而从更深层次的角度看待上述两个思想。

另外，从程序设计技术层面上看，STL 不仅极具实用性，而其极具观赏性，从而对学生也是一个工程美学的熏陶和训练。

也有很多人问：作为一个泛型库(这里是 STL)的使用者，用得着去学习和理解那些复杂的泛型机制和泛型技术吗？这个问题不太好回答，因为泛型机制的形成以及泛型例程库的建设都是围绕着数据类型的处理展开的，并且数据类型的处理工作都是由编译器完成的，故除非是进行泛型库例程设计，否则作为泛型库例程的使用者确实很少会直接接触本书前两章的内容。这跟开汽车一样：我只想开车上班，从来没想过修汽车和造汽车，那么我有必要去学习汽车原理和制造技术吗？当然，如果你与汽车这个行业无关，你可以这样做，但如果你和汽车行业有关，而且很紧密，那么这个问题就见仁见智了。如果这个问题非得让我回

答的话，那就是学一些为好，特别是在做学生的学习阶段，因为在泛型技术中体现出了很多程序设计的思想。

总之，积作者数十年的经验，作者认为本科生有必要学习泛型设计和 STL。

本书是作者在泛型程序设计和 STL 教学多年经验总结基础上编撰而成一本教材。本书分为三部分：第 1～3 章为第一部分；第 4 章为第二部分；第 4～8 章则为第三部分。

第一部分又分为两个部分：第 1 章和第 2 章是一部分；第 3 章单独为一个部分。在第 1 章和第 2 章介绍泛型的概念，C++ 泛型技术基础以及机制基础，这部分内容大致属于 STL 库代码设计范畴，是理解及将来进一步研究 C++ 泛型的重要基础。由于这部分比较偏向理论，所以为了便于初学者掌握，采取了用实际代码引领理论的讲解方式，并兼顾与前期学习的 C/C++ 的平滑过渡，特别适合本科生学习。第 3 章则在前两章的基础上介绍 STL 及其与 C/C++ 之间的关系，然后重点复习 STL 所使用的除了模板之外的 C++ 技术，并以此为基础介绍这些技术在 STL 的应用，从而与第 1 章与第 2 章共同形成学习 STL 的基础。

第 4 章虽然只是一章，但其地位却非常重要，是本书的“眼”，它实现了从泛型理论到 C/C++ 泛型实现的过渡。第 4 章从 STL 的六大基础部件之中，把最重要的容器、迭代器和通用算法提取出来，并将它们作为 STL“三大件”进行单独的仿真设计。其目的是使学生通过仿真从代码实现的层次上理解泛型及泛型的意义，同时在学习 C/C++ 程序设计之后在教材和教师的带领之下，进行这种规模比较大的程序设计对学生也是一个极好的训练，特别是迭代器的设计中所遵循的思想及其采用的设计技术，都会使学生受益匪浅。

从第 5 章开始便进入本书的第三部分，这部分主要就是对 STL 部件应用的介绍和讲解，操作内容明显变多。但需要注意的是，通用算法、适配器和容器内存空间配置器的介绍还是接触了一些理论性内容的。

本书是一本理论和应用兼顾，为泛型设计及 STL 初学者阅读的读物，鉴于它的特点，也适合作为在校计算机专业、软件工程专业或与之相关专业的教材。

本书的作者为任哲、房红征和张永忠，任哲负责全书统稿工作。

在本书的编写过程中，作者参阅了大量的参考文献及网上资料，并引用了一些相关文字和例程，在此谨向这些文献与资料的作者表示衷心的谢意。

由于作者水平有限，尽管很努力，但书中一定还会有很多错误和不尽如人意之处，在此敬请各位读者指出并不吝赐教，作者将不尽感谢。作者邮箱为 renzhe71@sina.com。

最后，祝各位读者学习进步。

作　者

2015 年 12 月

# 《C++ 泛型：STL 原理和应用》 目录

# 第1章　C++泛型技术基础——模板

在计算机程序设计领域，为避免因数据类型的不同而被迫重复编写大量具有相同业务逻辑的代码，人们发展了泛型(generic type)及泛型编程(generic programming)技术。其实说到底，泛型也是一种类型，只不过它是一种用来代替所有类型的“通用类型”。在C++中，用以实现泛型的重要技术基础就是模板。

**本章主要内容：**

- 泛型基本概念。
- 函数模板、类模板。
- 模板参数及特化模板。
- 右值引用及参数完美转发模板。

## 1.1　泛型与模板

C++中，用以支持泛型应用的是标准模板类库STL(Standard Template Library)，作为C++标准库的一个重要组成部分，它为用户提供了C++泛型设计常用的类模板和函数模板，并用它们支持C++的泛型设计。可以说，支持C++泛型的核心技术就是模板。

### 1.1.1　泛型的基本概念

所谓泛型程序设计，实质上就是不使用具体数据类型而是使用一种通用类型进行程序设计的方法，这种方法不仅能使人们把精力集中于业务逻辑的实现，而且还能大规模地减少程序代码的编写量。也就是说，泛型设计能有效避免或减少数据类型给程序设计所带来的麻烦。

**1. 数据类型给程序设计带来的困扰及解决方案**

为说明数据类型的麻烦所在，首先看一个程序设计场景：有用户提出需要一个函数，功能是返回两个int型数据之间的值较大的那个。很简单，可以编写代码如下：

```
int maxt(int x, int y)
{
    return(x>y)?x:y ;
}
```

没过几天，用户又提出需要编写一个能返回两个double型数据之间值较大数的函数。于是又编写代码如下：

```
double maxt(double x, double y)
{
    return(x>y)?x:y ;
}
```

之后不久的某一天，用户又提出要求再编写一个能返回两个双精度类型数据之间值较大数的函数，当然，你会很快地满足用户的要求，因为这个代码你已经很熟悉了，只需把数据类型变为双精度型即可。后来又有一天，用户又提出……××类型……，于是你就又……

很烦，是吧？烦什么？烦的是“数据类型”！因为就是因数据类型不同才迫使你不得不把具有同样功能的代码写了若干遍。于是聪明或有能力的人开始找帮手了，他把这种代码写成如下形式：

```
T maxt(T x, T y)
{
    return(x>y)?x:y ;
}
```

然后告诉他的帮手，当用户需要某个数据类型数据的 maxt 函数时，你就把这里的 T 替换成用户所需要的实际数据类型就行了。例如，用户需要的数据类型类型为 int，那么就用 int 替换 T 形成真正的函数：

```
int maxt(int x, int y)
{
    return(x>y)?x:y ;
}
```

如果用户需要的数据类型是 double，那么就用 double 替换 T 形成如下函数：

```
double maxt(double x, double y)
{
    return(x>y)?x:y ;
}
```

那么这里的这个 T 是什么？显然这是一个占位符，再仔细点说就是类型占位符，意思是说将来在 T 这个位置的是一个真实、具体的数据类型，至于是什么类型，那就看用户的需要。

如果硬要把上面所说的类型占位符（即上面的 T）也叫做一种数据类型，那么似乎可以有这样几个名称可供选择：“待定类型”、“通用类型”、“抽象类型”或者直接叫“假类型”。但就目前来说，这种做法的发明者把它叫做 generic type，翻译成中文就是“泛型”。于是，这种使用类型占位符的编程方法也就称为“泛型编程”。

既然类型占位符不是真正的数据类型，那么使用泛型编写的代码也就不是真正的程序实体，而只是一个程序实体的样板，将来使用真正数据类型替换类型占位符后才形成真正的程序实体，故此人们形象地将这种使用了类型占位符的代码称为“模板”，对应的，那些根据模板和实际数据类型生成的代码就叫做程序实体，产生程序实体的过程叫做模板的具现。

当然，在计算机技术如此发达的今天，人们不会再使用人来实施占位符的替换工作，而会把这种简单的替换工作交由计算机完成，具体来说就是由编译器在编译阶段实现模板的具现。

**2. 有关数据类型的几句话**

相信大多数人在初次学习类似 C/C++ 这类计算机语言时，都多多少少会对数据类型有

些不习惯，因为在日常生活中人们在表达一个数据时从来没提到过数据类型这类东西。之所以如此，是因为人比计算机聪明，能够根据数据的用途和具体语境判断出相应的数据类型，而计算机则不然，没有相应的说明，它就不知道北京市电话号长度为十进制 8 位，也不知道中国人姓名的长度一般不会超过 4 个中文字符，更不知道存放一个人年龄所需的存储空间只需十进制 3 位就足够。总之，如果想让计算机理解一个数据，那么就必须在数据之外为其提供相应的附加信息，这个附加信息最基本内容便是这个数据的字长及数据格式。在计算机中，用以表达数据基本附加信息的就是数据类型，于是计算机才能根据数据类型为数据分配相应的存储空间以及其他必要的处理。

事情就是这么有意思，一旦当人们熟悉并习惯了数据类型后，大多数人都会忘了这样一个事实，那就是编程必须使用数据类型这件事本来就是被计算机这个傻瓜逼的，从而对不使用数据类型就能编程序感到疑惑了。幸亏世界上还有一些不疑惑的人（例如 Alexander Stepanov 等，读者可到网上去了解他们的事迹），他们经过多年不懈的努力，终于通过类型占位符这个技术手段使计算机可以接受“通用”数据类型了，尽管人们还没有完全摆脱数据类型的束缚，但使用“泛”型（generic type）来编程，比使用 int、double……这些精准数据类型轻松多了。

### 1.1.2 C++ 模板及其定义

为了能够规范地进行泛型程序设计，无论对模板的定义，还是对模板的调用，C++ 都制定了清晰的语法规则和格式。

规则制定者们认为，既然模板是一种样板，那么它除了需要声明类型占位符之外，其他内容应与目标实体代码完全相同。所以在声明一个模板时，仅需在实体代码前面，按如下格式写一条说明语句就可以了：

```
template <typename 占位符 1,typename 占位符 2,...,占位符 n>
```

template 为模板声明关键字，关键字后面那一对尖括号括起来的是模板参数列表，里面使用关键字 typename 声明了类型占位符。

目前，C++ 有函数模板和类模板两种模板。

**1. 函数模板**

例如，如果把前面的 maxt 函数定义成一个模板，那么按照 C++ 模板的定义规则，其代码如下：

```
template<typename T,typename R,typename S >
T maxt(R x, S y)
{
    return(x>y)?x:y ;
}
```

因为这种模板的目标实体为与模板同名的函数，所以这种模板叫做函数模板。

template 参数列表中的 T、R 和 S 便是类型占位符，将来在模板需要生成实体函数时，这些占位符会由用户程序事先提供的实际数据类型所替换。即在源程序中调用模板生成函数实体并实现这个实体调用的是一条语句，以 maxt 模板及其函数实体为例，其调用格式

如下：

```
int x;
double y, z;
⋮
maxt<int, double, double>(x, y, z);        //模板及实体函数调用语句
```

可见，调用模板及其函数的方法与调用普通函数基本相同，只不过要多传递一套以真实数据类型表示的模板实参(即尖括号里的内容)。

当 C++ 编译器对源程序进行预编译并看到关键字 template 时，就会记住这个模板名，当编译器看到用户程序中出现了调用该模板的语句时，编译器会去寻找模板代码，并用调用语句中尖括号里提供的实际数据类型在模板尖括号中找到与之对应的占位符并替换它们，一旦替换完毕，一个实体函数也就产生了，接下来就如同调用普通函数那样调用这个实体函数。应用程序调用模板及其函数实体的过程如下：

$$\underbrace{\underbrace{\text{模板名}<\text{实际类型 1},\text{实际类型 2},\cdots,\text{实际类型 } n>}_{\text{这一段为模板调用，生成实体函数}}(\text{函数实参}\cdots);}_{\text{这一段为函数调用}}$$

**例 1-1** 修改并使用上面的函数模板 maxt 分别将 4 与 6 以及 6.98 与 5.33 这两对数据的较大值显示于显示器上。

**解：**

(1) 程序代码。

因本题要求的两个函数中的参数都是 int 和 double 同一种数据类型，故函数模板 maxt 只声明一个类型占位符 T 就可以了，于是可编写程序如下：

```
#include <iostream>
using namespace std;
//函数模板
template<typename T >
T maxt(T x,T y)
{
    return(x>y)?x:y ;
}

//主函数
int main(int argc, char *argv[])
{
    cout<<maxt<int>(4,6)<<endl;
    cout<<maxt<double>(6.98,5.33)<<endl;
    return 0;
}
```

(2) 编译器生成实体函数的过程。

图 1-1 给出了编译器根据调用语句 maxt<int, int, int>(4,6)生成实体函数的过程。

在程序第一次调用(即 maxt<int >(4,6))时，编译器根据函数模板为程序生成了实体函

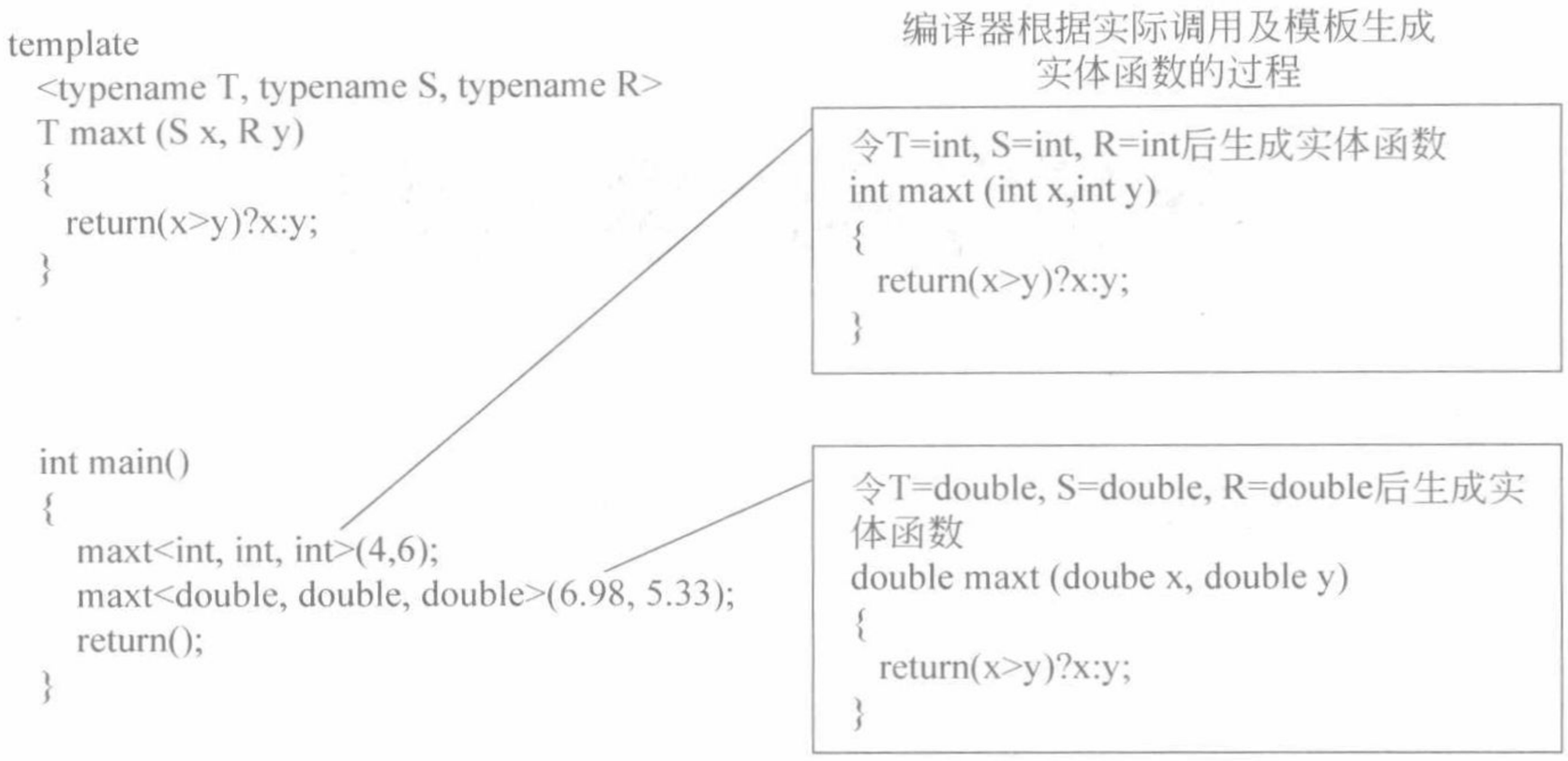


图 1-1 编译器根据实际调用由模板生成实体函数的过程

数 int maxt(int,int),程序输出结果为 6;在第二次调用(即 maxt<double>(6.98,5.33))时,则生成了实体函数 double maxt(double, double),程序输出结果为 6.98。程序使用同一个函数模板生成两个实体函数。也就是说,本例程序实质上相当于如下没有使用模板技术的程序:

```
#include <iostream>
using namespace std;
//数据类型为 int 的函数
int maxt(int x,int y)
{
    return(x>y)?x:y ;
}
//数据类型为 double 的函数
double maxt(double x,double y)
{
    return(x>y)?x:y ;
}
//主函数
int main(int argc, char *argv[])
{
    cout<<maxt(4,6)<<endl;
    cout<<maxt(6.98,5.33)<<endl;
    return 0;
}
```

其实在以函数作为模板代码时,在调用时由于函数通常都会有参数传递,而函数参数数据类型往往就是模板参数将来的实际数据类型,故当编译器可以从函数参数类型中推测出模板参数所需的实际类型时,模板调用语句中的尖括号可以省略。例如,例 1-1 中的模板调用可以写成如下的样子:

```
cout<<maxt(4,6)<<endl;
cout<<maxt(6.98,5.33)<<endl;
```

(3) 程序运行结果。

程序运行结果如图 1-2 所示。

```
6
6.98
```

图 1-2　例 1-1 程序运行结果

### 2. 类模板

除了可以声明函数模板，在 C++ 中还可以声明类模板。在一个类声明之前写出模板声明语句得到的便是类模板。例如如下代码：

```
template<typename T>
class Circle
{
private:
    T Radius;
public:
    Circle(T r){Radius=r;}
    T  Area(){
        return PI * Radius * Radius;
    }
};
```

上述代码声明了一个叫做 Circle 的用来描述一个圆的类模板，其私有数据成员为 Radius(圆半径)，函数成员为构造函数和一个用来求圆面积的 Area()。

类模板的调用方法与函数模板完全相同，如果用户希望在应用程序中使用类模板生成实体类并定义类对象，则需要在类模板名称后面的尖括号中填入模板实参，其语法格式如下：

```
类名<模板实参>对象名;
```

对于本例，如果希望编译器生成一个半径数据类型为 int，值为 100 的对象，则定义语句如下：

```
Circle<int>(100);
```

与实体类相同，类模板也可将其成员函数实现在类模板之外，但必须将它们声明为函数模板。

**例 1-2**　定义一个描述圆的类模板，其中包含一个求圆面积的成员函数。

**解：**

(1) 程序代码。

圆类模板和测试程序代码如下：

```
#include<iostream>
using namespace std;
#define PI 3.14159
//Circle 类模板
template<typename T>
```

```
class Circle
{
private:
    T Radius;
public:
    Circle(T r);
    T Area();
};
//主函数-----------------------------------------------
int main()
{
    //定义一个整型的 Circle 对象
    Circle<int>circle_1(10);
    cout<<circle_1.Area()<<endl;
    //定义一个 double 型的 Circle 对象
    Circle<double>circle_2(12.786);
    cout<<circle_2.Area()<<endl;
    return 0;
}
//类模板外实现的类成员函数-------------
template<typename T>
Circle<T>::Circle(T r)
{
    Radius=r;
}
template<typename T>
T Circle<T>::Area()
{
    return PI * Radius * Radius;
}
```

(2) 程序运行结果。

程序运行结果如图 1-3 所示。

```
314
513.593
```

图 1-3　例 1-2 程序运行结果

### 1.1.3　几点说明和小结

**1. 说明**

首先要说明，模板参数列表中所定义类型占位符的作用域仅限于本模板。例如，例 1-2 中在类模板 Circle 外部定义的类函数模板的占位符不一定非得相同。

其次，STL 的模板编程对面向对象技术并不感兴趣，它认为类对数据的过度封装影响了程序的执行效率，之所以 STL 的模板编程中还大量地使用类模板，是因为类这种形式可

以对程序代码进行形式上的分割，从而使代码更便于阅读和管理，所以读者将会在 STL 中看到大量使用没有访问限制的 struct 制作的类模板，例如将上面的类模板 Circle 写成了如下形式：

```
//Circle类模板
template<typename T>
struct Circle
{
    T Radius;
    Circle(T r);
    T Area();
};
```

最后还必须指出，定义模板参数的关键字除了 typename 之外，还有一个早期遗留下来的 class，即上述模板还可以写为：

```
//Circle类模板
template<class T>
struct Circle
{
    T Radius;
    Circle(T r);
    T Area();
};
```

但是，语义上 typename 比 class 更为准确而不会引起误解，除了早期的代码，人们通常都使用 typename。

**2. C++ 新标准对泛型设计的努力——auto 和 decltype**

在 C++ 11 和 C++ 14 这两个新标准中，C++ 编译器的类型处理能力又向前跨了一大步，其中一个比较重要的具体体现就是废除了关键字 auto 的原语义，并为之赋予了新语义，这个新语义就是数据类型的自动推导。即使用这个关键字可以自动为变量确定类型。除此之外，为配合 auto，新标准还增加了 decltype 表达式，从而可以实现函数返回值类型的自动推导。

显然，从上述内容可知，关于 auto 和 decltype 的内容应该属于 C++ 语言基础内容，并不适合在本章关于 C++ 模板的内容中出现，但因其涉及了泛型设计，故本书不得不在这里对它们加以说明。

首先，auto 关键字是为自动推导数据类型所设，它能在定义一个变量时根据这个变量的初始化数据自动推导出变量或对象的数据类型。例如，如下语句：

```
auto a=100;
```

执行这条语句之后，编译器就会根据这条语句中的初始化数据 100 自动将变量 a 的数据类型定义为 int。

但目前，auto 的这个能力还有限，只对系统的内置数据类型有效。对那些用户自定义或较复杂的数据类型，只有当编译器取得了足够的经验后，才具备自动推导这种数据类型的