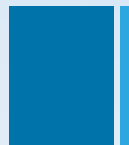




C++

第十八讲 模板（二）

C++备课组 丁盟



自我介绍

丁盟

QQ : 2622885094





上一讲教学目标

- 理解C++中模板的含义
- 掌握C++中函数模板的使用方法



本讲教学目标

- 掌握C++中类模板的使用方法
- 理解C++中模板的实现原理

1

类模板

2

类模板和模板类

3

模板的实现原理

类模板

- ❖ 如果我们要用一个stack类来处理数据，那么可能要定义多个类，以适应不同的数据类型。

```
const int KSize = 10;
class Stack {
Public:
    Stack();
    ~Stack();
    void push(const double&);
    double pop();
private:
    double data[KSize];
};
```

```
const int KSize = 10;
class Stack {
Public:
    Stack();
    ~Stack();
    void push(const int&);
    int pop();
private:
    int data[KSize];
};
```

类模板

- ❖ 这些栈的基本操作完全一样，唯一的差别是数据成员的**数据类型不同**，如果我们可以把类型作为参数，就可以很容易的产生很多个类型不同的栈，这种机制就是**类模板**。

忽略具体的数据类型

```
class string_stack  
{ ... };
```

```
class int_stack  
{ ... };
```

```
class double_stck  
{ ... };
```

```
template<class T, int size>  
class stack  
{  
    ...  
private:  
    T data[size];  
};
```

类模板 - 类模板的声明

- ❖ 类模板的定义和声明都以关键字`template`开头，关键字后面是一个用逗号分隔的模板参数表。

```
template <class Type>  
class Classname  
{  
    definition  
};
```

模板参数表,可以有多个类型,类型之间用逗号隔开。其中每个类型必须使用关键字`class`或`typename`


- 类型参数可以用来声明数据成员、成员函数的形参，以及成员函数的返回值。
- 类型参数在类模板中的使用方式与内置的或用户定义的类型在非模板类定义中的用法一样。

类模板 – 类模板的声明

```
const int KStackSize = 10;
template<class Type, int KStackSize>
class Stack
{
public:
    void push( const Type& aVal );
    Type pop();
private:
    Type data[KStackSize];
};
```

类模板 – 类模板的声明

- ❖ 如果模板参数是类型参数，则实参必须是类型；
- ❖ 如果模板参数是非类型参数，那么这类非类型参数是右值的，对应的实参必须是：
 - 一个常量表达式。
 - 具有外部链接的对象或函数的地址
 - 非重载的指向成员的指针



```
template<class Type, int size>
class Stack {
    ...
};
```

类模板 – 类模板的声明

- ❖ 类模板的参数可以有缺省实参。这对类型参数和非类型参数都一样。模板参数的缺省实参是一个类型或值。当模板被**实例化**时，如果没有指定实参，则使用该类型或者值。

```
template <class Type = char, int size = 10>
class Stack
{    ...    };
```

- 正如函数参数的缺省实参的情形一样，在向左边的参数提供缺省实参之前，必须**首先给最右边未初始化的参数提供缺省实参**。

类模板 – 类模板的声明

```
template <class T, int KStackSize >
class Stack {
public:
    Stack( int aSize = KStackSize );
    ~Stack();
    bool push( const T& aVal );
    bool pop( T& aVal );
    bool isEmpty() const;
    bool isFull() const;
private:
    int size;
    int top;
    T* stackPtr;
};
```

一个完整的
类模板声明

类模板 – 类模板的定义

```
template <class Type>
return_type Classname< Type >::member_name()
{
    definition
}
```

- ❖ 类模板成员函数的定义具有如上形式
 - 必须以关键字template开头，后接类的模板形参表
 - 必须指出是哪个类的成员
 - **类名必须包含其模板形参**

类模板 – 类模板的定义

// 构造函数定义

```
template<class Type>
Stack<Type>::Stack(int aSize) {
    size = aSize;
    top = -1;
    stackPtr = new Type[size];
}
```

// 析构函数析构

```
template<class Type>
Stack<Type>::~~Stack() {
    delete []stackPtr;
}
```

类模板 – 类模板的定义

```
// 普通成员函数的定义
template<class Type>
bool Stack<Type>::push(const Type& aVal)
{
    if ( !isFull() )
    {
        stackPtr[++top] = aVal;
        return true;
    }
    return false;
}
```

类模板 – 类模板的实例化

Class_name< **类型实参** > **对象名** ;

```
Stack<int> ist;  
Stack<int> stack(100);  
Stack<string> des;
```

类模板对象调用成员函数的方式和普通类调用方法类似

```
Stack<int> stk(100);  
stk.push( 1 );  
stk.push( 2 );  
stk.push( 3 );  
bool res = stk.isFull();
```


类模板 - 类模板的静态成员

❖ 类模板可以像任意其他类一样声明static成员

```
template<class T>
class Counter {
public:
    static int count();
    // other members
private:
    static int m_iCount;
    // other members
};
template<class T>
int Counter<T>::m_iCount = 0;
```

类模板 – 类模板的静态成员

❖ 使用类模板的静态成员

```
Counter<int> cnt;  
Counter<int> crt;  
int size = Counter::count(); // error  
int size = Counter<int>::count();
```

```
Counter<double> cit;  
Counter<double> cdt;  
int count = Counter<double>::count();
```

1

类模板

2

类模板和模板类

3

模板的实现原理

类模板和模板类

- ❖ 类模板是一种模板，类模板通过在类定义上传入类型参数的形式，表示具有相似操作的类的集合。
- ❖ 模板类是指从类模板实例化的类。通常通过传递给类模板以类型实参而得到模板类。
- ❖ 类模板的实例是模板类；类的实例是对象。
- ❖ 对象可在运行时确定，而类必须在编译时确定。

```
template<class T> Stack; // 类模板  
Stack<int>;           // 模板类  
Stack<int> stk;        // 模板类定义的对象
```

类模板和模板类 - 模板的多态性

- ❖ 模板是一种静态多态，是在编译过程中进行类型识别的。

```
class Circle {  
    public:  
        int area() const;  
};
```

```
class Rectangle {  
    public:  
        int area() const;  
};
```

```
template<class T>  
void getArea( const T& a )  
{  
    cout << a.area() << endl;  
}
```

类模板和模板类 - 模板的多态性

```
int main(void)
{
    Circle cir;
    Rectangle rect;

    getArea( cir );
    getArea( rect );

    return 0;
}
```

1

类模板

2

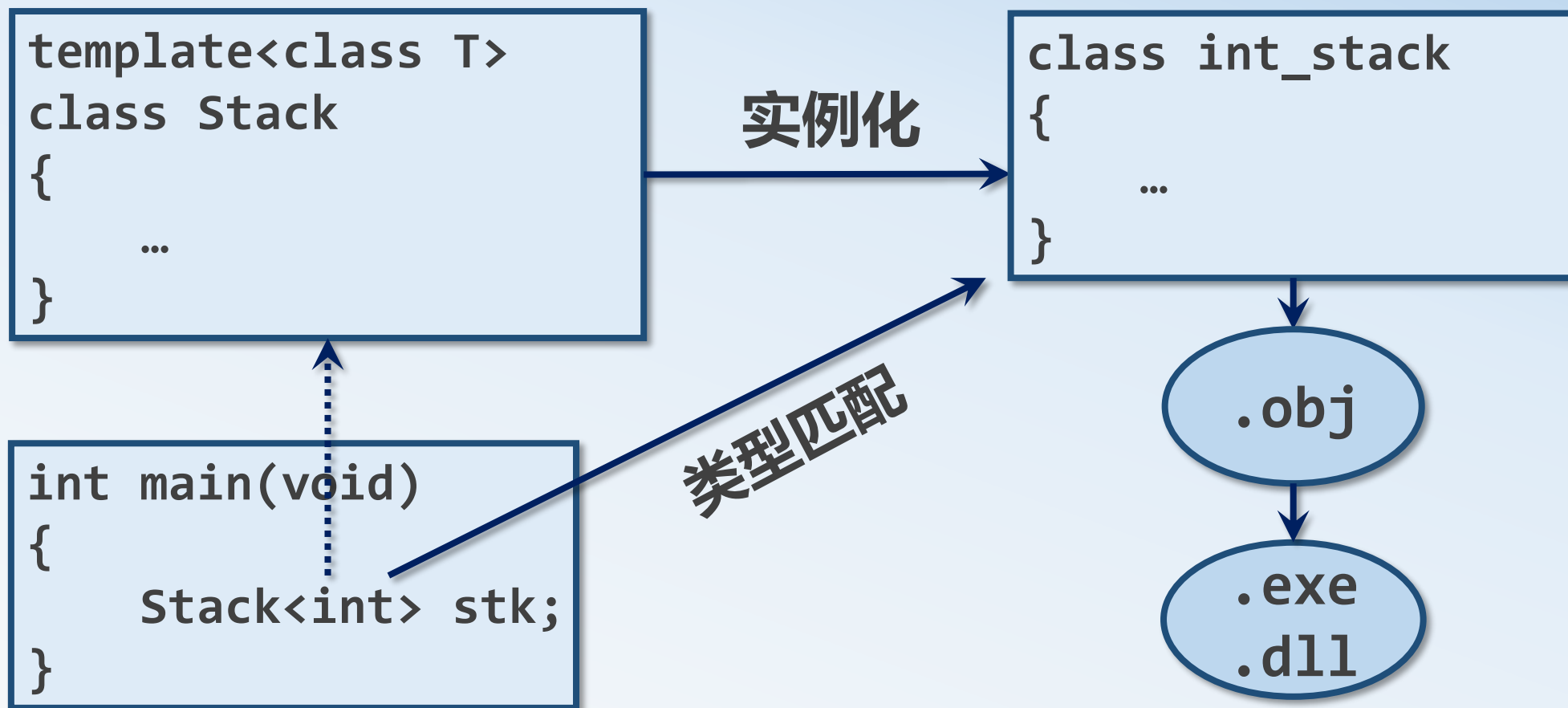
类模板和模板类

3

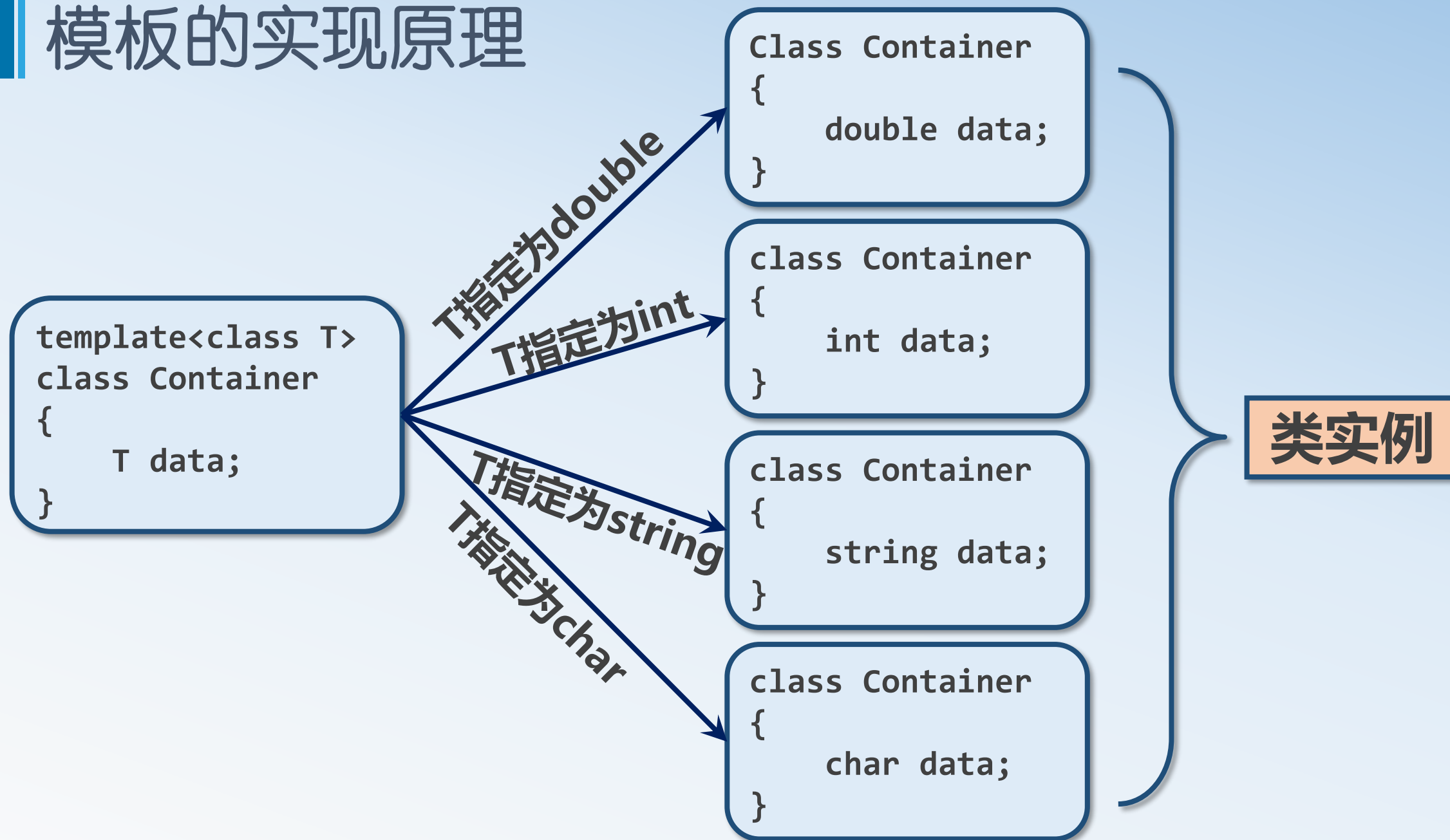
模板的实现原理

模板的实现原理

- ❖ 具体的机制是在**编译过程**中自动形成具体模板类的源代码。



模板的实现原理



模板的实现原理

❖ [总结]

- 模板就是实现代码重用机制的一种工具，它可以实现类型参数化，即把类型定义为参数，从而实现了真正的代码可重用性。模版可以分为两类，一个是函数模版，另外一个为是类模版。
- 模板的数据类型只能在编译时才能被确定，是静态多态的一种实现。



本讲教学目标

- 掌握C++中类模板的使用方法
- 理解C++中模板的实现原理



THANKS

