



C++ 模板类详细介绍

模板类是C++中实现泛型编程的重要特性，它允许你编写与数据类型无关的代码。下面我将详细介绍模板类的各个方面。

基本概念

模板类是一种可以操作多种数据类型的类，而不需要为每种类型都重写代码。编译器会根据使用的具体类型自动生成相应的类。

基本语法

cpp

复制 下载

```
template <typename T> // 或者 template <class T>
class ClassName {
    // 类成员使用T作为类型
};
```

typename 和 **class** 在这里可以互换使用。

模板类示例

cpp

复制 下载

```
template <typename T>
class Box {
private:
    T content;
public:
    void setContent(const T& newContent) {
        content = newContent;
    }
    T getContent() const {
        return content;
    }
};
```

使用这个模板类：

cpp

 复制  下载

```
Box<int> intBox;  
intBox.setContent(123);  
  
Box<std::string> stringBox;  
stringBox.setContent("Hello Templates!");
```

模板参数

多类型参数

cpp

 复制  下载

```
template <typename T, typename U>  
class Pair {  
    T first;  
    U second;  
    // ...  
};
```

非类型模板参数

cpp

 复制  下载

```
template <typename T, int size>  
class Array {  
    T arr[size];  
    // ...  
};
```

使用：

cpp

 复制  下载

```
Array<double, 10> doubleArray;
```

默认模板参数

cpp

 复制  下载



```
template <typename T = int, int size = 10>  
class Buffer {  
    // ...  
};
```

成员函数实现

模板类的成员函数可以在类内或类外定义：

类内定义

cpp

 复制  下载

```
template <typename T>
class Box {
    T content;
public:
    void setContent(T value) { content = value; }
    // ...
};
```

类外定义

cpp

 复制  下载

```
template <typename T>
class Box {
    T content;
public:
    void setContent(T value);
    // ...
};



template <typename T>
void Box<T>::setContent(T value) {
    content = value;
}
```

特化与偏特化

完全特化

为特定类型提供特殊实现：

cpp



 复制  下载

```
template <>
class Box<char> {
    char content;
public:
    void setContent(char value) {
        content = std::toupper(value); // 特殊处理: 存储为大写
    }
    // ...
};
```

偏特化

对部分模板参数进行特化:

cpp



 复制  下载

```
template <typename T>
class Box<T*> {
    T* content;
public:
    void setContent(T* value) {
        content = new T(*value); // 深拷贝
    }
    // ...
};
```

模板类与继承

模板类可以参与继承关系:

cpp



 复制  下载

```
template <typename T>
class Base {
    // ...
};

template <typename T>
class Derived : public Base<T> {
    // ...
};
```

模板类与友元

cpp

 复制  下载

```
template <typename T>
class Box {
    T content;
    friend std::ostream& operator<<(std::ostream& os, const Box<T>& box) {
        os << box.content;
        return os;
    }
};
```

可变参数模板(C++11)

cpp

 复制  下载

```
template <typename... Args>
class Tuple {
    // ...
};
```

模板别名(C++11)

cpp

 复制  下载

```
template <typename T>
using Vec = std::vector<T>;

Vec<int> intVector; // 等同于 std::vector<int>
```

注意事项

模板类的定义通常放在头文件中，因为编译器需要看到完整的定义来实例化模板



模板类不是真正的类，而是生成类的"配方"

每个不同的模板参数组合都会生成一个独立的类

模板类可以嵌套在其他模板类中

实际应用示例

cpp

 复制  下载

```
template <typename T>
class Stack {
private:
    std::vector<T> elements;
public:
    void push(const T& element) {
        elements.push_back(element);
    }
    T pop() {
        if (elements.empty()) {
            throw std::out_of_range("Stack<>::pop(): empty stack");
        }
        T element = elements.back();
        elements.pop_back();
        return element;
    }
    bool empty() const {
        return elements.empty();
    }
};
```

使用示例:

cpp

 复制  下载

```
Stack<int> intStack;
intStack.push(7);
intStack.push(42);

Stack<std::string> stringStack;
stringStack.push("hello");
stringStack.push("world");
```

模板类是C++强大功能的体现，它极大地提高了代码的复用性和灵活性。理解并熟练使用模板类对于编写高质量的C++代码至关重要。