Templates in C++

什么是C++模板?

C++模板是一种机制，让你编写可以处理不同数据类型的通用代码。例如，你可以创建一个排序函数，适用于整数、浮点数或自定义类型，而无需为每种类型重复编写代码。这提高了代码的可重用性和灵活性。

模板的定义与目的

模板本质上是C++中定义类、函数或变量的蓝图，这些蓝图可以根据提供的类型参数生成具体实例。例如，标准库中的`std::vector`就是一个类模板，可以实例化为`std::vector<int>`或`std::vector<double>`。模板的核心目标是提升代码重用性，减少冗余代码的编写。

模板可以参数化类型、常量和模板模板参数。当提供模板参数时，编译器会生成特定类型的特化实例。

模板的语法

- **函数模板**：用`template <typename T>`定义，例如：

  ```
  1  template <typename T>
  2  T add(T a, T b) { return a + b; }
  ```

  调用时，编译器根据参数类型自动生成具体函数。
- **类模板**：类似地定义类，例如：

  ```
  1  template <typename T>
  2  class Vector { /* 实现 */ };
  ```

  使用时需指定类型，如`Vector<int> vec;`。

语法详解

函数模板

函数模板允许定义通用的函数，适用于多种数据类型。其基本语法为：

```
template <typename T>
T functionName(T param) { /* 实现 */ }
```

例如，一个求最大值的函数模板：

```
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```

调用时，如`max(3, 4)`或`max(3.5, 4.5)`，编译器会根据参数类型生成具体函数。

类模板

类模板定义一族类，其语法类似函数模板。例如：

```
template <typename T>
class Box {
public:
    T value;
    Box(T v) : value(v) {}
};
```

使用时需实例化，如`Box<int> intBox(5);`。类模板的成员函数可以在类定义内或外部定义，前提是保持模板参数的一致性。

模板特化

模板特化允许为特定类型提供定制实现。例如，全特化一个类模板：

```
template <>
class Box<bool> {
    // 针对bool的特殊实现
};
```

函数模板也可以特化，但更常见的是通过函数重载实现类似效果。

模板参数

模板参数包括类型参数（如`typename T`）、非类型参数（如`int N`）和模板模板参数（如`template <typename> class Container`）。例如：

```
template <int N, typename T>
class Array {
    T data[N];
};
```

这种灵活性允许更复杂的泛型设计。

使用场景

模板常用于：

- 通用容器，如向量、列表。
- 通用算法，如排序、搜索。
- 实用函数，如最小值、最大值计算。

注意事项

使用模板时需注意：

- 多次实例化可能导致代码膨胀，增加程序大小。
- 编译时间可能变长，尤其当模板使用复杂。
- 错误信息可能复杂，难以调试。
- 模板定义通常放在头文件中，以确保编译时可见。