# 泛型

程序中的一种特性，允许程序员在强类型程序设计语言中编写代码时定义一些可变部分，这些部分在使用前指出即可（一般指数据类型）

##### 泛型的好处

泛型提高了程序对类型验证假设，并且提高了程序的安全性，增加了面向对象编程效力和灵活性

### C++中的模版

C++模版是泛型的基础，帮助我们在对多个逻辑操作模块但是类型有差异的地方，定义一种替代类型使用，我们可以不用关心类型的差异，完成整个逻辑的编写，并且达到逻辑的高可复用。**模版提供了参数化类型，即能够将类型名作为参数传递给接收方来建立函数或是类。**

模版操作时在编译阶段验证类型操作的正确性

### 定义模版

格式

template<typename T>

T Max(T a, T b)；

新C++中已经将class替换成typename用来接受更多数据类型。

T是模版参数（此参数名称可以随意拟定）

##### 模版函数

可以在不关心处理数据类型的前提下（操作不会涉及对象成员信息），进行更加广泛的逻辑处理

定义函数模版，可以作用域函数有效域内，借助函数模版可以引入多类型操作的逻辑，一个**函数模版就是一个公式，用来生成针对特定类型的函数版本**

模版参数的作用域：当前函数内有效

例子：（全局的模版函数）

template<typename N>

N Max(N a, N b)//编写函数时不决定参数的数据类型

{

return a > b ? a : b;

}

##### 类模版

类模版是用来生成类的蓝图的，与函数模版的不同之处是，编译器不能为类模版推断模版参数类型

由于模版编译与普通编译差异化，类模版的定义和函数声明通常都放在头文件中，函数模版也放在头文件中

模版参数作用域：当前类作用域内

# UE中的容器

##### TArray UE中的数组

最常用的数据容器，特点速度快，内存消耗小，安全性高。

TArray被称为同构容器：其所有元素均完全为相同类型。不能进行不同元素类型的混合。

TArray被设计成值类型，无法被继承，不要使用new和delete在堆上进行创建销毁。元素也是数值类型，为容器拥有。TArray被销毁时元素也被销毁。从一个TArray创建新的TArray变量，将把元素复制到新的变量中，不存在共享状态。

##### 构建数组并操作填充

第一种

TArray<FString> data; //普通构建方式，在栈中申请一块空间，Array的存储内容为FString

第二种

TArray<FString> data;

data.Init(TEXT("OK"), 10);//向数组中填充10个元素

##### **Add** 和 **Emplace** 函数可用于在数组末端创建新对象

Add和Emplace函数均可向数组中添加元素，元素被添加时，内存从分配器中被分配。Add和Emplace函数可达到同样的效果，但是存在细微不同。

·Add函数将把一个元素类型实例复制（或移动）到数组中

·Emplace函数将使用您赋予的参数构建一个元素类型的新实例（参数类型是一个右值引用，右值引用是短暂的，可以被随意修改的）

**Add是用于常规数据添加，Emplace当我们添加的数据是一个右值时使用**

E性能优于A

##### Append复制多个元素进入数组

T ary[] = {};

Array.Append(ary, ARRAY\_COUNT(ary));