# C++ Template of template

2014/10/16 HERESY 3 則迴響

template 在 C++ 裡面，算是一個很重要、也很實用的概念。它可以用同樣的程式、來處理不同類型的資料，大幅簡化程式碼的重複性。

而這一篇，算是來簡單紀錄一下，所謂的「template of template」的玩法吧～主要的參考資料，是《Templates of templates》這篇文章。也請注意，這篇只是 Heresy 自己簡單的紀錄，寫得不算很嚴謹。

首先，一般的 template 寫法，可能會像是下面這樣：

```
template<typename DType>
class CData
{
public:
    std::vector<DType>    mContent;
};
```

在上面的類別 CData 裡，是用一個 std::vector 來儲存資料，而裡面的型別則是 DType；使用的時候，基本上會是：

```
CData<int> x;
```

---

但是，如果希望可以讓使用者決定要用哪種 STL 容器（container）時該怎麼辦呢？比較直覺的方法，就是寫成：

```
template<typename DType, typename TContainer >
class CData
{
public:
    TContainer<DType>    mContent;
};
```

然後在要使用的時候，寫成：

```
CData<int,std::vector> x;
```

也就是把 std::vector 的部分，也用 template 的方法來寫。但是很遺憾，這樣的語法是有問題的。

---

而比較簡單的修改，應該是改成：

```
template<typename TContainer >
class CData
{
public:
    TContainer    mContent;
};
```

這樣寫的話，使用時則是會像下面這樣：

```
CData< std::vector<int> > x;
```

但是這樣的缺點，就是把資料型別和容器的型別綁在一起了，其實有的時候可能不是那麼地實用。

---

而如果要把資料型別和容器的類別切割開的話，則可以寫成：

```
template<    typename DType,
            template<typename T> class TContainer >
class CData
{
public:
    TContainer<DType>    mContent;
};
```

在這邊，就是很明確地告訴編譯器，TContainer 是一個 template 的類別，而他有一個 template 的參數。這樣一來，就可以任意地套用符合形式的容器了～

不過，如果是要使用 STL 的 vector 的話，由於他實際上有兩個 template 參數（參考），所以直接拿來用會因為 template 參數不相同、而有編譯階段的錯誤；如果要用的話，則是需要再封包一次才行，封包方法大致上可以寫成：

```
template<typename T>
class CVector : public std::vector<T>
{};
```

實際使用時是：

```
CData<int, CVector> x;
```

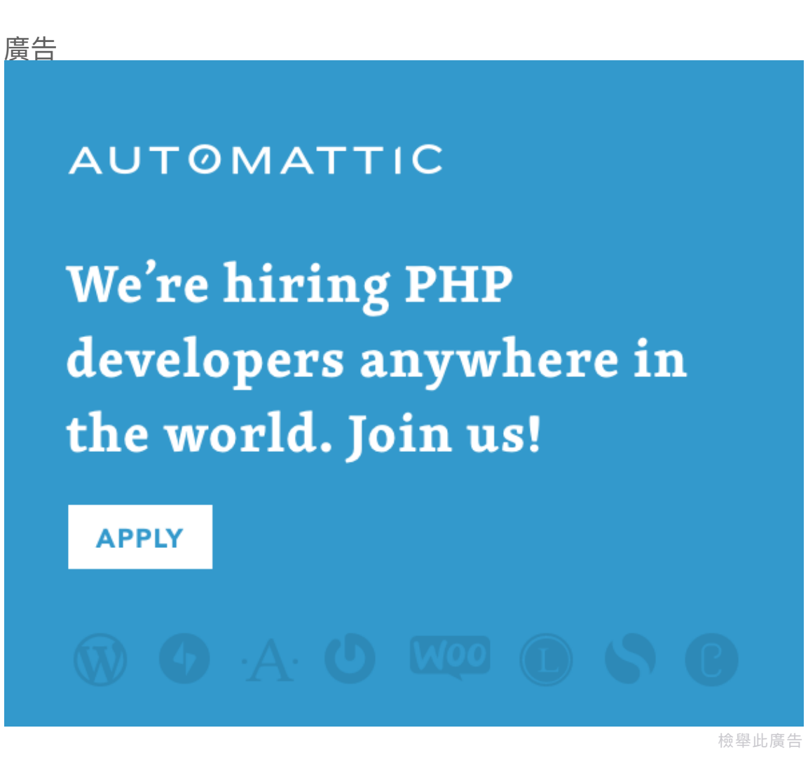
而對於其他 STL 的容器、或是自己定義的資料類別，基本上也都是可以用類似的方法來操作的。

---

為什麼會跑來看這個？其實 Heresy 主要是把這個用在 callable object 上，也就是想用 template 來展開函式。簡單講，就是想下類似下面架構的程式：

```
#include <iostream>

template<template<typename T> class TFunc>
void FuncTemp(int iMode)
{
    if (iMode == 0 )
    {
        TFunc<int>()();
    }
    else if ( iMode == 1 )
    {
        TFunc<float>()();
    }
}

template<typename T>
class CFunc1
{
public:
    void operator()(){
    }
};

template<typename T>
class CFunc2
{
public:
    void operator()(){
    }
};

int main()
{
    FuncTemp<CFunc1>(1);
    FuncTemp<CFunc2>(0);
}
```

本來 CFunc1 和 CFunc2 都不是想寫成 class、而是想寫成 template function，但是那樣寫是不合法的，所以只能寫成類別。而後來也因為這樣寫太繁瑣了，還是放棄了…



幫忙推廣一下吧！

Tweet 更多

正在載入...

C++、程式設計

舊文章
修正 Windows 影像中心上傳照片可能不會調整大小的問題

新文章
2014 獅頭山水濂洞

## 對「C++ Template of template」的想法

**使用 Parameter Pack 取代 switch 來做型別的展開 | Heresy's Space** 說道： 回覆
2017/05/25 at 18:04

[…] template class 傳進來取代 ExecFunc() 當作函式用、增加通用性（參考《C++ Template of template》）；但是，如果又要對應不同的組合（例如有的函式不支援 […]

★ 讚

**Nicky C** 說道： 回覆
2014/10/24 at 10:19

我懷疑 TContainer 不太適合作 template template parameter 。

如果將 TContainer 改作 variadic template ，又無法自動取得 vector 的預設 allocator 。再者，人手把 vector 包裝成 CVector 太繁瑣，還不如直接傳入 vector 。更何況，STL 的 container adaptor（例如 queue 和 stack）都不用 template template parameter。

《Templates of templates》文中刻意避開個別容器的模板參數的數量，也許該文作者其實也注意到這弱點。

–

FuncTemp 的例子，這樣寫更簡單、更通用：

```
template <class NullaryOp0, class NullaryOp1>
void functionSelect(int select, NullaryOp0 op0, NullaryOp1 op1)
{
    if (select == 0)
        op0();
    else if (select == 1)
        op1();
}

template <class T>
void cFunc1() {}

functionSelect(1, cFunc1<int>, cFunc1<float>);
```

甚至：

```
#include <iostream>
using namespace std;

template <class... NoOp>
void functionSelect(int select, NoOp...) {cout << "nothing" << endl;
}

template <class NullaryOp, class... OtherNullaryOps>
void functionSelect(int select, NullaryOp op, OtherNullaryOps... oth
ers)
{
    if (select == 0)
        op();
    else
        functionSelect(select-1, others...);
}

template <class T>
void cFunc1() {cout << static_cast<T>(1) << endl;}

template <class T>
void cFunc2() {cout << static_cast<T>(2) << endl;}

int main()
{
    for (int i = 0 ; i < 10 ; ++i)
        functionSelect(i, cFunc1<int>, cFunc1<float>, cFunc2<int>, cFun
    return 0;
}
```

★ 讚

**Heresy** 說道： 回覆
2014/10/24 at 18:01

是的，實際上 Heresy 完全同意 TContainer 不適合用來接 STL 的 container，而作者感覺上也刻意沒提這件事…Heresy 一開始也沒想到，也是實際測試才發現的。

另外，您所提供的 functionSelect 其實不見得比較方便，因為這樣必須在每次使用時，窮舉出所有需要的型別；如果型別多、且可能會被呼叫數次多的時候，這樣寫反而麻煩。

當然啦～這也取決於實際上的應用的需求。

★ 讚