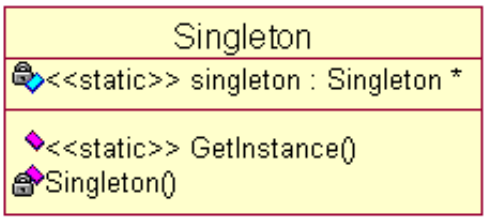
設計模式C++實現（4）——單例模式

星期六, 2013 12月 14, 1:02 上午

軟件領域中的設計模式為開發人員提供了一種使用專家設計經驗的有效途徑。設計模式中運用了面向對象編程語言的重要特性：封裝、繼承、多態，真正領悟設計模式的精髓是可能一個漫長的過程，需要大量實踐經驗的積累。最近看設計模式的書，對於每個模式，用C++寫了個小例子，加深一下理解。主要參考《大話設計模式》和《設計模式:可復用面向對象軟件的基礎》（DP）兩本書。本文介紹單例模式的實現。

單例的一般實現比較簡單，下面是代碼和UML圖。由於構造函數是私有的，因此無法通過構造函數實例化，唯一的方法就是通過調用靜態函數GetInstance。

UML圖：




```
1. //Singleton.h
2. class Singleton
3. {
4. public:
5.     static Singleton* GetInstance();
6. private:
7.     Singleton() {}
8.     static Singleton *singleton;
9. };
10. //Singleton.cpp
11. Singleton* Singleton::singleton = NULL;
12. Singleton* Singleton::GetInstance()
13. {
14.     if(singleton == NULL)
15.         singleton = new Singleton();
16.     return singleton;
17. }
```

這裡只有一個類，如何實現Singleton類的子類呢？也就說Singleton有很多子類，在一種應用中，只選擇其中的一個。最容易就是在GetInstance函數中做判斷，比如可以傳遞一個字符串，根據字符串的內容創建相應的子類實例。這也是DP書上的一種解法，書上給的代碼不全。這裡重新實現了一下，發現不是想像中的那麼簡單，最後實現的版本看上去很怪異。在VS2008下測試通過。

```
1. //Singleton.h
2. #pragma once
3. #include <iostream>
4. using namespace std;
5.
```

```
6.  class Singleton
7.  {
8.  public:
9.      static Singleton* GetInstance(const char* name);
10.     virtual void Show() {}
11. protected: //必須為保護，如果是私有屬性，子類無法訪問父類的構造函數
12.     Singleton() {}
13. private:
14.     static Singleton *singleton; //唯一實例的指針
15. };
16.
17. //Singleton.cpp
18. #include "Singleton.h"
19. #include "SingletonA.h"
20. #include "SingletonB.h"
21. Singleton* Singleton::singleton = NULL;
22. Singleton* Singleton::GetInstance(const char* name)
23. {
24.     if(singleton == NULL)
25.     {
26.         if(strcmp(name, "SingletonA") == 0)
27.             singleton = new SingletonA();
28.         else if(strcmp(name,"SingletonB") == 0)
29.             singleton = new SingletonB();
30.         else
31.             singleton = new Singleton();
32.     }
33.     return singleton;
34. }
```

```
1.  //SingletonA.h
2.  #pragma once
3.  #include "Singleton.h"
4.  class SingletonA: public Singleton
5.  {
6.      friend class Singleton; //必須為友元類，否則父類無法訪問子類的構造函數
7.  public:
8.      void Show() { cout<<"SingletonA"<<endl; }
9.  private:   //為保護屬性，這樣外界無法通過構造函數進行實例化
10.     SingletonA() {}
11. };
12. //SingletonB.h
13. #pragma once
14. #include "Singleton.h"
15. class SingletonB: public Singleton
16. {
17.     friend class Singleton; //必須為友元類，否則父類無法訪問子類的構造函數
18. public:
19.     void Show(){ cout<<"SingletonB"<<endl; }
20. private:  //為保護屬性，這樣外界無法通過構造函數進行實例化
21.     SingletonB() {}
22. };
```

```
1. #include "Singleton.h"
2. int main()
3. {
4.     Singleton *st = Singleton::GetInstance("SingletonA");
5.     st->Show();
6.     return 0;
7. }
```

上面代碼有一個地方很詭異，父類為子類的友元，如果不是友元，函數GetInstance會報錯，意思就是無法調用SingletonA和SIngletonB的構造函數。父類中調用子類的構造函數，我還是第一次碰到。當然了把SingletonA和SIngletonB的屬性設為public，GetInstance函數就不會報錯了，但是這樣外界就可以定義這些類的對象，違反了單例模式。

看似奇怪，其實也容易解釋。在父類中構建子類的對象，相當於是外界調用子類的構造函數，因此當子類構造函數的屬性為私有或保護時，父類無法訪問。為共有時，外界就可以訪問子類的構造函數了，此時父類當然也能訪問了。只不過為了保證單例模式，所以子類的構造函數不能為共有，但是又希望在父類中構造子類的對象，即需要調用子類的構造函數，這裡沒有辦法才出此下策：將父類聲明為子類的友元類。