**Singleton模式**

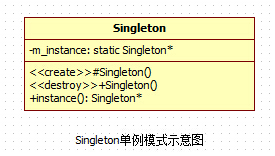
**问题:**

个人认为Singleton 模式是设计模式中最为简单、最为常见、最容易实现，也是最应该熟悉和掌握的模式。且不说公司企业在招聘的时候为了考察员工对设计的了解和把握，考的最多的就是Singleton模式。

Singleton 模式解决问题十分常见，我们怎样去创建一个唯一的变量（对象）？在基于对象的设计中我们可以通过创建一个全局变量（对象）来实现，在面向对象和面向过程结合的设计范式（如C++中）中，我们也还是可以通过一个全局变量实现这一点。但是当我们遇到了纯粹的面向对象范式中，这一点可能就只能是通过Singleton 模式来实现了，可能这也正是很多公司在招聘Java开发人员时候经常考察Singleton模式的缘故吧。Singleton模式在开发中非常有用，具体使用在讨论给出。

**模式选择:**

Singleton模式典型的结构图为：



在Singleton模式的结构图中可以看到，我们通过维护一个static的成员变量来记录这个唯一的对象实例。通过提供一个staitc的接口instance来获得这个唯一的实例。

**实现:**

完整代码示例（code）

// @ File Name : Singleton.h

#if !defined(\_SINGLETON\_H)

#define \_SINGLETON\_H

class Singleton

{

public:

~Singleton();

static Singleton\* instance();

protected:

Singleton();

private:

static Singleton\* m\_instance;

};

#endif //\_SINGLETON\_H

// @ Project : Singleton

// @ File Name : Singleton.cpp

// @ Date : 2014/7/15

// @ Author : www.newdebug.com

//

#include "Singleton.h"

#include <iostream>

Singleton\* Singleton::m\_instance = 0;

Singleton::Singleton()

{

std::cout << "Singleton..." << std::endl;

}

Singleton::~Singleton()

{

}

Singleton\* Singleton::instance()

{

if( 0 == m\_instance )

{

/\* 如果使用多线程: \*/

/\* 详情参考C++设计新思维泛型编程与设计模式之应用 \*/

/\* 第6章Singleton实现技术的第6.9节生活在多线程世界 \*/

// Guard myGuard(m\_lock);

// if( 0 == m\_instance )

// {

m\_instance = new Singleton();

// }

}

return m\_instance;

}

#include "Singleton.h"

#include <iostream>

using namespace std;

int main()

{

Singleton\* sgn = Singleton::instance();

return 0;

}

D:\NewDebugSVN\GithubCode\NewDebug\CPP\UML\Singleton\images\SingletonScreen.jpg

**代码说明**

Singleton模式的实现无须补充解释，需要说明的是，Singleton不可以被实例化，因此我们将其构造函数声明为protected或者直接声明为private。

Singleton 模式在开发中经常用到，且不说我们开发过程中一些变量必须是唯一的，比如说打印机的实例等等。Singleton模式经常和Factory（AbstractFactory）模式在一起使用，因为系统中工厂对象一般来说只要一个，笔者在开发VisualCMCS的时候，语义分析过程（以及其他过程）中都用到工厂模式来创建对象（对象实在是太多了），这里的工厂对象实现就是同时是一个Singleton模式的实例，因为系统我们就只要一个工厂来创建对象就可以了。