**C++ 和 Singleton模式**

Singleton，是广泛使用的设计模式，也是设计模式中最简单的。 程序中必然有需要全局都使用的功能和变量，有人看到全局就反感，但是如果强改成局部的话，就需要传递很多变量，代码带来的效果是一样的，但是代码更加复杂，可以说这些是‘必要的恶’。

说完原理，再说如何在C++中实现

常见方法

* 使用静态变量，静态指针，全局函数

这是相当原始的方法，跟用C语言实现没有差别。如果这些函数有公共变量需要保存，需要在.cc文件中定义静态变量，并且每个方法都依赖于这些变量，导致通用性不够

* 使用静态类变量

这是《设计模式》中提到的一种方法，这种方法比前一种可以省掉很多的全局函数。只需要用正常的对象方法就可以了，要是哪天不需要使用Singleton，这个类要复用还是比较好更改的

* 使用静态指针类型

这是《设计模式》中提到的另外的一种方法，这种方法使用延迟加载，在需要的时候才new出一个对象。但是这种方法会导致内存泄露，其他效果类似于上一种

存在的问题

* 生命周期的问题？

虽然Singleton天然很难控制生命周期，但是使用静态变量使得情况更加糟糕，静态变量在程序开始就构造好，影响启动性能，而且启动状态下程序的运行环境和正常的环境还不相同，有的系统调用是不能使用，比如Windows上可能会遇到loader加锁的问题。

* 泄露的问题？

为了避免生命周期的问题， 使用静态指针类型的方法，就会遇到内存泄露，虽然Singleton对象就那么几个，不会造成内存一直增长，但是在当你的程序是DLL，而且被重复加载的时候，问题就严重了，并且这样Singleton对象的析构方法也不会被调用，利用析构做一些清理也就不会被执行。一种处理方法是显示加上delete方法，在程序要退出的时候进行释放。

* 多线程问题？
* if (!m\_pInstance)
* m\_pInstance = new Logger;

这段代码在多线程环境下，可能会生成多于一个Logger，而且多出来的Logger没法获得，相当于内存泄露了。

* 重复代码

在解决了这么多问题后，代码也不简单了，突然又需要一个Singleton 对象，这些代码还要再写一遍，OMG。还不说重复写可能造成的意外疏忽。

**chromium base中的singleton方法**

chromium 一如既往的提供了对好的设计模式的支持。

相关文件是 base/memory/singleton.h 注释超级详细，已经可以算得上是使用用文档。它有这些特性：

* 线程安全

使用原子变量、双重测试、spinlock的方法，这段代码博主也只能看看，讲清楚估计还有待时日。

* 基于模板，不用继承

如果使用继承的方法，实现复用的Singelton模板是不可能的，因为static变量和static函数，天生就不能用继承来重载。

* 可以控制生存周期（泄露，析构（默认），自定义Add Release之类的不同对象申请释放方法）

Singleton模式有个问题，就是Singleton间不能依赖，至少在析构函数里面不能有相互依赖。这样就需要控制释放顺序，但是这个就很麻烦，最具代表性的就是logging功能，这是全局都要使用的，有可能析构的时候都要使用。对于这种对象，泄露比释放造成的危害少很多。所以需要根据需要定制生存周期。chromium中使用模板traits的方法，使得使用不同的模板参数就能实现需要的小姑。

* 可以使用静态内存

这可是实现Singleton的高级技巧，不仅不用new或者malloc申请内存，而且没有内存泄露的风险。原理是使用一块static数组，然后将数组当成申请的空间执行构造函数。

* 控制访问线程

线程有Joinable，no-Joinable的区分，一般的程序，但需要释放Singleton的时候相关的Joinable的线程已经退出，当然就可以安全释放了。但是no-Joinable的生存周期有可能超出main()。

* 强制命名为GetInstance()

保证代码质量

**使用示例**

* In your header:
* template <typename T> struct DefaultSingletonTraits;
* class FooClass {
* public:
* static FooClass\* GetInstance(); <-- See comment below on this.
* void Bar() { ... }
* private:
* FooClass() { ... }
* friend struct DefaultSingletonTraits<FooClass>;
* DISALLOW\_COPY\_AND\_ASSIGN(FooClass);
* };
* In your source file:
* #include "base/memory/singleton.h"
* FooClass\* FooClass::GetInstance() {
* return Singleton<FooClass>::get();
* }
* And to call methods on FooClass:
* FooClass::GetInstance()->Bar();

**注意事项**

1. 如果需要控制Singleton的构造函数的访问，需要将相关的Traits模板设置为friend类型。
2. Singleton模式中需要的获得方法要命名为GetInstance()
3. GetInstance()需要放在源文件中。如果遇到在多DLL中使用Singleton对象的情况，放在头文件总会导致创建多个对象。
4. 不要间singleton.h放在头文件中。

**参考**

* 《设计模式--可复用面向对象软件的基础》