设计模式

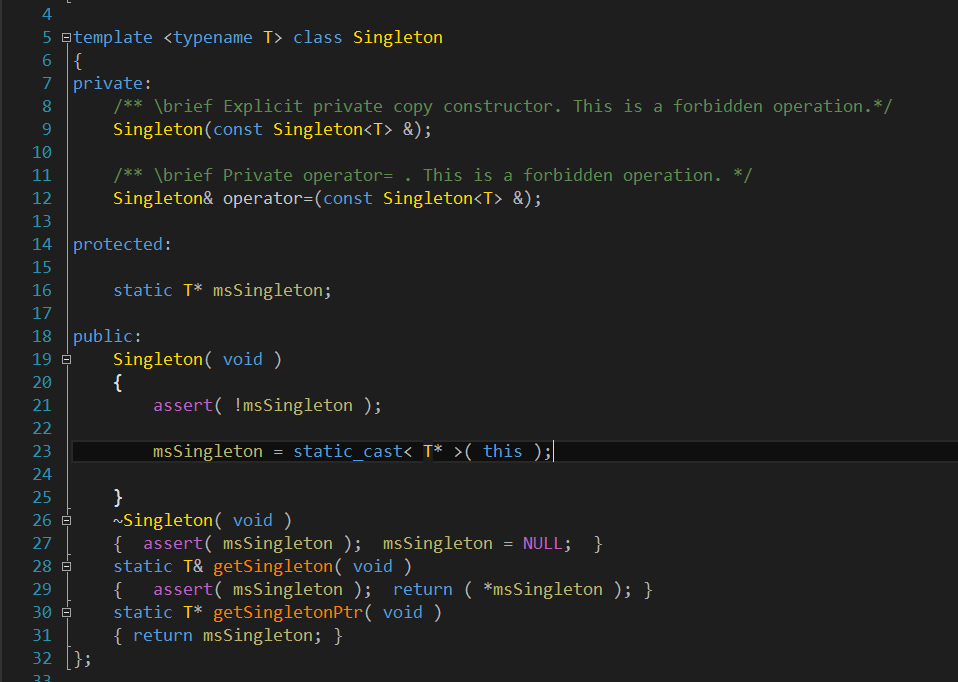
1. 单例模式

单例模式，可以说设计模式中最常应用的一种模式了。但是如果没有学过设计模式的人，可能不会想到要去应用单例模式，面对单例模式适用的情况，可能会优先考虑使用全局或者静态变量的方式，这样比较简单，也是没学过设计模式的人所能想到的最简单的方式了。

一般情况下，我们建立的一些类是属于工具性质的，基本不用存储太多的跟自身有关的数据，在这种情况下，每次都去new一个对象，即增加了开销，也使得代码更加臃肿。其实，我们只需要一个实例对象就可以。如果采用全局或者静态变量的方式，会影响封装性，难以保证别的代码不会对全局变量造成影响。使用全局对象能够保证方便地访问实例，但是不能保证只声明一个对象——也就是说除了一个全局实例外，仍然能创建相同类的本地实例。

单例模式通过类本身来管理其唯一实例，这种特性提供了解决问题的方法。唯一的实例是类的一个普通对象，但设计这个类时，让它只能创建一个实例并提供对此实例的全局访问。唯一实例类Singleton在静态成员函数中隐藏创建实例的操作。

单例模式就是在系统中只允许产生这个类的一个实例。



用户访问唯一实例的方法只有getSingleton()或者getSingletonPtr()成员函数。如果不通过这个函数，任何创建实例的尝试都将失败，因为类的构造函数是私有的。getSingleton ()或者getSingletonPtr()使用懒惰初始化，也就是说它的返回值是当这个函数首次被访问时被创建的。

上述只能在单线程中，保证只取得一个实例。但是在多线程的环境下确实不行的，因为很可能两个线程同时运行到新建实例的程序，导致可能产生两个实例。为了解决多线程问题，需要在程序中加锁。

Singleton\* getInstance()

{

lock();

if(instance == NULL)

{

instance = new Singleton();

}

unlock();

return instance;

}

但这样写的话，会稍稍映像性能，因为每次判断是否为空都需要被锁定，如果有很多线程的话，就爱会造成大量线程的阻塞。于是大神们又想出了双重锁定。

Singleton\* getInstance()

{

if (instance == NULL)

{

lock();

if (instance == NULL)

{

instance = new Singleton();

}

unlock();

}

return instance;

}

这样只够极低的几率下，通过越过了if (instance == NULL)的线程才会有进入锁定临界区的可能性，这种几率还是比较低的，不会阻塞太多的线程，但为了防止一个线程进入临界区创建实例，另外的线程也进去临界区创建实例，又加上了一道防御if (instance == NULL)，这样就确保不会重复创建了。

1. 工厂模式

简单工厂、工厂方法、抽象工厂