### 1、懒汉，线程不安全

public class Singleton {

private static Singleton instance;

private Singleton (){}

public static synchronized Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

}

return instance;

}

}

这种写法能够在多线程中很好的工作，而且看起来它也具备很好的lazy loading，但是，遗憾的是，效率很低，99%情况下不需要同步。

### 2、饿汉

public class Singleton {

private static Singleton instance = new Singleton();

private Singleton (){}

public static Singleton getInstance() {

return instance;

}

}

这种方式基于classloder机制避免了多线程的同步问题，instance在类装载时就实例化。目前java单例是指一个虚拟机的范围，因为装载类的功能是虚拟机的，所以一个虚拟机在通过自己的ClassLoader装载饿汉式实现单例类的时候就会创建一个类的实例。

这就意味着一个虚拟机里面有很多ClassLoader，而这些classloader都能装载某个类的话，就算这个类是单例，也能产生很多实例。当然如果一台机器上有很多虚拟机，那么每个虚拟机中都有至少一个这个类的实例的话，那这样 就更不会是单例了。(这里讨论的单例不适合集群！)

### 3、静态内部类

public class Singleton {

private static class SingletonHolder {

private static final Singleton INSTANCE = new Singleton();

}

private Singleton (){}

public static final Singleton getInstance() {

return SingletonHolder.INSTANCE;

}

}

这种方式同样利用了classloder的机制来保证初始化instance时只有一个线程，这种方式是Singleton类被装载了，instance不一定被初始化。因为SingletonHolder类没有被主动使用，只有显示通过调用getInstance方法时，才会显示装载SingletonHolder类，从而实例化instance。更多面试题，欢迎关注公众号 Java面试题精选

想象一下，如果实例化instance很消耗资源，我想让他延迟加载！这个时候，这种方式相比第2种方式就显得很合理。

### 4、枚举

public enum Singleton {

INSTANCE;

public void whateverMethod() {

}

}

这种方式是Effective Java作者Josh Bloch 提倡的方式，它不仅能避免多线程同步问题，而且还能防止反序列化重新创建新的对象，可谓是很坚强的壁垒啊，不过，个人认为由于1.5中才加入enum特性，用这种方式写不免让人感觉生疏，在实际工作中，我也很少看见有人这么写过。

### 5、双重校验锁(jdk1.5)

public class Singleton {

private volatile static Singleton singleton;

private Singleton (){}

public static Singleton getSingleton() {

if (singleton == null) {

synchronized (Singleton.class) {

if (singleton == null) {

singleton = new Singleton();

}

}

}

return singleton;

}

}

这样方式实现线程安全地创建实例，而又不会对性能造成太大影响。它只是第一次创建实例的时候同步，以后就不需要同步了。

由于volatile关键字屏蔽了虚拟机中一些必要的代码优化，所以运行效率并不是很高，因此建议没有特别的需要不要使用。双重检验锁方式的单例不建议大量使用，根据情况决定。

### 6、总结

有两个问题需要注意：

* 如果单例由不同的类装载器装入，那便有可能存在多个单例类的实例。假定不是远端存取，例如一些servlet容器对每个servlet使用完全不同的类装载器，这样的话如果有两个servlet访问一个单例类，它们就都会有各自的实例。
* 如果Singleton实现了java.io.Serializable接口，那么这个类的实例就可能被序列化和复原。不管怎样，如果你序列化一个单例类的对象，接下来复原多个那个对象，那你就会有多个单例类的实例。

对第一个问题修复的办法是：

private static Class getClass(String classname)

throws ClassNotFoundException {

ClassLoader classLoader = Thread.currentThread().getContextClassLoader();

if(classLoader == null)

classLoader = Singleton.class.getClassLoader();

return (classLoader.loadClass(classname));

}

}

对第二个问题修复的办法是：

public class Singleton implements java.io.Serializable {

public static Singleton INSTANCE = new Singleton();

protected Singleton() {

}

private Object readResolve() {

return INSTANCE;

}

}