[转+注]单例模式的七种写法 (http://www.hollischuang.com/archives/205)

2015-04-18 分类：Java (http://www.hollischuang.com/archives/category/java) 阅读(6476) 评论(2)

本站采用[知识共享署名-非商业性使用-相同方式共享 许可协议 (http://creativecommons.org/licenses/by-nc-sa/3.0/deed.zh)]进行许可，转载请在 正文明显处 注明原文地址

> 写在前面：单例模式，是设计模式中最简单的一种，但是，他却有很多的东西需要注意，性能、线程安全等。这篇文章是我转载的，转载之后我仔细研究了一下并加了备注和相关知识链接（鼠标悬浮在带链接的文字上就可以看到我的注释,例如：鼠标悬浮在这）。

原文地址：http://cantellow.iteye.com/blog/838473 (http://cantellow.iteye.com/blog/838473)

第一种（懒汉，线程不安全）：

```
public class Singleton {
    private static Singleton instance;
    private Singleton (){}

    public static Singleton getInstance() {
    if (instance == null) {
        instance = new Singleton();
    }
    return instance;
    }
}
```

第二种（懒汉，线程安全）：

```
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。

第三种（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，在深度分析Java的ClassLoader机制（源码级别）(/archives/197)和Java类的加载、链接和初始化(/archives/201)两个文章中有关于CLassload而机制的线程安全问题的介绍，避免了多线程的同步问题，不过， instance 在类装载时就实例化，虽然导致类装载的原因有很多种，在单例模式中大多数都是调用 getInstance 方法， 但是也不能确定有其他的方式（或者其他的静态方法）导致类装载，这时候初始化 instance 显然没有达到 lazy loading 的效果。

第四种（饿汉，变种）：

```
public class Singleton {
    private Singleton instance = null;
    static {
    instance = new Singleton();
    }
    private Singleton (){}
    public static Singleton getInstance() {
    return this.instance;
    }
}
```

表面上看起来差别挺大，其实更第三种方式差不多，都是在类初始化即实例化instance。

第五种（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，它跟第三种和第四种方式不同的是（很细微的差别）：第三种和第四种方式是只要Singleton类被装载了，那么instance就会被实例化（没有达到lazy loading效果），而这种方式是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。想象一下，如果实例化instance很消耗资源，我想让他延迟加载，另外一方面，我不希望在Singleton类加载时就实例化，因为我不能确保Singleton类还可能在其他的地方被主动使用从而被加载，那么这个时候实例化instance显然是不合适的。这个时候，这种方式相比第三和第四种方式就显得很合理。

第六种（枚举 (http://www.hollischuang.com/index.php/archives/345/)）：

```
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，在深度分析Java的枚举类型—-枚举的线程安全性及序列化问题(http://www.hollischuang.com/index.php/archives/349/)中有详细介绍枚举的线程安全问题和序列化问题，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。

第七种（双重校验锁）：

```
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;
    }
}
```

总结

有两个问题需要注意：

1.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。

2.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。单例与序列化的那些事儿 (http://www.hollischuang.com/archives/1144)

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;
      }
}
```

对我来说，我比较喜欢第三种和第五种方式，简单易懂，而且在JVM层实现了线程安全（如果不是多个类加载器环境），一般的情况下，我会使用第三种方式，只有在要明确实现lazy loading效果时才会使用第五种方式，另外，如果涉及到反序列化创建对象时我会试着使用枚举的方式来实现单例，不过，我一直会保证我的程序是线程安全的，而且我永远不会使用第一种和第二种方式，如果有其他特殊的需求，我可能会使用第七种方式，毕竟，JDK1.5已经没有双重检查锁定的问题了。

不过一般来说，第一种不算单例，第四种和第三种就是一种，如果算的话，第五种也可以分开写了。所以说，一般单例都是五种写法。懒汉，恶汉，双重校验锁，枚举和静态内部类。

【公告】版权声明 (http://www.hollischuang.com/转载说明)

(全文完)

查看 1 条评论

欢迎关注HollisChuang微信公众账号

如未加特殊说明，此网站文章均为原创，转载必须注明出处。HollisChuang's Blog (http://www.hollischuang.com) » [转+注]单例模式的七种写法 (http://www.hollischuang.com/archives/205)

标签： 单例 (http://www.hollischuang.com/archives/tag/%e5%8d%95%e4%be%8b)

分享到： 更多 (4)

相关推荐

- 设计模式（三）——JDK中的那些单例 (http://www.hollischuang.com/archives/1383)
- 单例与序列化的那些事儿 (http://www.hollischuang.com/archives/1144)
- 为什么说Java中只有值传递。 (http://www.hollischuang.com/archives/2275)
- 你离BAT之间，只差这一套Java面试题。 (http://www.hollischuang.com/archives/2223)
- 我反编译了Java 10的本地变量类型推断 (http://www.hollischuang.com/archives/2187)
- 人人都能掌握的Java服务端性能优化方案 (http://www.hollischuang.com/archives/2181)
- 全网把Map中的hash()分析的最透彻的文章，别无二家。 (http://www.hollischuang.com/archives/2091)
- Java 10将于本月发布，它会改变你写代码的方式 (http://www.hollischuang.com/archives/2064)

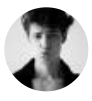
登录

来说两句吧...

评论

4 人参与, 1 条评论

最新评论



HollisChuang网友 [北京市网友] 2016年3月15日 10:00

java static内部不能使用非static的属性和方法，第四种写法会报错，不能通过编译，看楼主的意思，应该是个笔误，属性应该被static修饰。另外，第四种写法和饿汉很相似，确实如楼主所言，使用上和饿汉没有区别，感觉没有必要。

回复 1

HollisChuang正在使用畅言 (http://changyan.kuaizhan.com/)

联系我 (http://mail.qq.com/cgi-bin/qm_share?t=qm_mailme&email=-JSTkJCVj5_UiZ2Sm7yNjdKfk5E)

关于我 (/sample-page)

© 2018 HollisChuang's Blog (http://www.hollischuang.com)　网站地图 (http://www.hollischuang.com/sitemap.xml)|阿里云 (http://www.aliyun.com)|Alibaba.com (http://www.alibaba.com)| (http://www.cnzz.com/stat/website.php?web_id=1256611093)