链滴

设计模式学习笔记之单例模式

作者：hjljy
原文链接：https://ld246.com/article/1560935380707
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

前言

这是一篇学习笔记，内容很多是来源于网上的资料，按照自己学习进行的总结。

我的个人博客：海加尔金鹰

什么是单例模式

定义：一个类在系统当中只存在一个实例，每次获取到这个类的实例都是同一个。主要用于处理系统中某个频繁创建和摧毁的类。

特点：

- 单例类只有一个实例对象
- 该单例对象必须由单例类自行创建
- 单例类对外提供一个访问该单例的全局访问点。

单例模式的实现

单例模式的核心代码：构造函数私有化，提供对外访问的方式。

单例模式的实现方式有很多种，我这里就只记录推荐使用的方式：

枚举

```
public enum Singleton {
    INSTANCE;
}
```

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，而且还能防反序列化重新创建新的对象。[1]

这个我感觉是用来面试装逼的，记得以前和朋友们聊面试的时候，开玩笑的说：面试不要慌，先写个例模式在说。至于反序列化重新创建对象的问题，目前没有遇见过。

讲真的如果不是专门学习单例模式的话，我都不知道有这种写法。

饿汉式

```
public class Singleton {

    private final static Singleton INSTANCE = new Singleton();

    private Singleton(){}

    public static Singleton getInstance(){
        return INSTANCE;
    }
}
```

在单例对象声明的时候就直接初始化对象，可以避免多线程问题，但是如果对象初始化比较复杂，会致程序初始化缓慢。并且初始化后如果没有使用到的话，也会造成资源的浪费。

静态内部类

```
public class Singleton {

    private Singleton() {}

    private static class SingletonInstance {
        private static final Singleton INSTANCE = new Singleton();
    }

    public static Singleton getInstance() {
        return SingletonInstance.INSTANCE;
    }
}
```

这种方式跟饿汉式方式采用的机制类似，但又有不同。两者都是采用了类装载的机制来保证初始化实时只有一个线程。

不同的地方在饿汉式方式是只要Singleton类被装载就会实例化，没有Lazy-Loading的作用，而静态部类方式在Singleton类被装载时并不会立即实例化，而是在需要实例化时，调用getInstance方法，会装载SingletonInstance类，从而完成Singleton的实例化。

类的静态属性只会在第一次加载类的时候初始化，所以在这里，JVM帮助我们保证了

线程的安全性，在类进行初始化时，别的线程是无法进入的。

优点：避免了线程不安全，延迟加载，效率高。[2]

注意事项

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

ervlet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，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。[1]

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;
    }
}
```

单例模式的优缺点

优点：

1.在单例模式中，活动的单例只有一个实例，对单例类的所有实例化得到的都是相同的一个实例。这就 防止其它对象对自己的实例化，确保所有的对象都访问一个实例

2.单例模式具有一定的伸缩性，类自己来控制实例化进程，类就在改变实例化进程上有相应的伸缩性。

3.提供了对唯一实例的受控访问。

4.由于在系统内存中只存在一个对象，因此可以节约系统资源，当 需要频繁创建和销毁的对象时单例式无疑可以提高系统的性能。

5.允许可变数目的实例。

6.避免对共享资源的多重占用。[3]

缺点：

1.不适用于变化的对象，如果同一类型的对象总是要在不同的用例场景发生变化，单例就会引起数据错误，不能保存彼此的状态。

2.由于单利模式中没有抽象层，因此单例类的扩展有很大的困难。

3.单例类的职责过重，在一定程度上违背了“单一职责原则”。

4.滥用单例将带来一些负面问题，如为了节省资源将数据库连接池对象设计为的单例类，可能会导致享连接池对象的程序过多而出现连接池溢出；如果实例化的对象长时间不被利用，系统会认为是垃圾被回收，这将导致对象状态的丢失。[3]

推荐资料

Singleton is an angel but an evil！写的比较有深度，可以仔细看看。

1. 单例模式的七种写法:https://cantellow.iteye.com/blog/838473
2. 单例模式的八种写法比较:https://www.cnblogs.com/zhaoyan001/p/6365064.html
3. 单例模式的优缺点和使用场景:https://www.cnblogs.com/restartyang/articles/7770856.html