# 单例

## 题目

设计一个类，我们只能生成该类的一个实例

## 解题思路

* 线程安全
* 延迟加载
* 序列化与反序列化安全

/\*\*
 \* 需要额外的工作(Serializable、transient、readResolve())来实现序列化，否则每次反序列化一个序列化的对象实例时都会创建一个新的实例。
 \* <p>
 \* 可能会有人使用反射强行调用我们的私有构造器（如果要避免这种情况，可以修改构造器，让它在创建第二个实例的时候抛异常）。
 \*
 \* @author haoyang.shi
 \*/
public class Singleton {

 private Singleton() {
 }

 public static Singleton getInstance() {
 return Holder.instance;
 }

 private static final class Holder {
 private static Singleton instance = new Singleton();
 }
}

/\*\*
 \* 使用枚举除了线程安全和防止反射强行调用构造器之外，还提供了自动序列化机制，防止反序列化的时候创建新的对象。
 \* <p>
 \* 因此，Effective Java推荐尽可能地使用枚举来实现单例。
 \*/
enum SingletonEnum {
 INSTANCE;

 private String name;

 public String getName() {
 return name;
 }

 public void setName(String name) {
 this.name = name;
 }
}