# 单例的实现

## 单例设计模式的问题

1. 线程安全问题 用synchronized修饰实例化部分代码
2. 性能问题 –采用懒汉式实例化
3. 指令重排序问题 –用volatile修饰实例
4. 反序列化攻击问题 –构造函数判断存在实例时抛异常
5. 反射攻击问题 –增加readResolve方法
6. 不符合开闭原则，需要改代码

## 五种单例实现方式

### 饿汉式

package me.muphy.singleton;  
  
import java.io.Serializable;  
  
*/\*\*  
 \* 2019/4/1  
 \* 莫非  
 \*/*public class HungrySingleton implements Serializable {  
  
 private static final HungrySingleton *hungrySingleton* = new HungrySingleton();  
  
 private HungrySingleton() {  
  
 }  
  
 public static HungrySingleton getInstance() {  
 return *hungrySingleton*;  
 }  
  
 private Object readResolve() {  
 return *hungrySingleton*;  
 }  
  
}

### 懒汉式 延时加载方式

package me.muphy.singleton;  
  
*/\*\*  
 \* 2019/4/1  
 \* 莫非  
 \*/*public class LazySingleton {  
  
 private volatile static LazySingleton *lazySingleton* = null;  
  
 private LazySingleton() {  
 if (*lazySingleton* != null) {  
 throw new RuntimeException("此类以单例存在！");  
 }  
 }  
  
 public static LazySingleton getInstance() throws Exception {  
 if (*lazySingleton* == null) {  
 synchronized (LazySingleton.class) {  
 if (*lazySingleton* == null) {  
 *lazySingleton* = new LazySingleton();  
 }  
 }  
 }  
 return *lazySingleton*;  
 }  
  
 private Object readResolve() {  
 return *lazySingleton*;  
 }  
  
}

### 懒汉式 内部内实现单例

package me.muphy.singleton;  
  
*/\*\*  
 \* 2019/4/1  
 \* 莫非  
 \*/*public class LazyJvmSingleton {  
  
 private LazyJvmSingleton(){  
  
 }  
  
 public static LazyJvmSingleton getInstance(){  
 return LazySingleton.*lazyJvmSingleton*;  
 }  
  
 private static class LazySingleton{  
 public static final LazyJvmSingleton *lazyJvmSingleton* = new LazyJvmSingleton();  
 }  
  
 private Object readResolve(){  
 return LazySingleton.*lazyJvmSingleton*;  
 }  
}

### 注册式 枚举式单例

package me.muphy.singleton;  
  
*/\*\*  
 \* 2019/4/1  
 \* 莫非  
 \*/*public enum EnumSingleton {  
 *INSTANCE*;  
  
 private Object data;  
  
 public Object getData() {  
 return data;  
 }  
  
 public void setData(Object data) {  
 this.data = data;  
 }  
  
 public static EnumSingleton getInstance() {  
 return *INSTANCE*;  
 }  
}

### 注册式 容器式单例

package me.muphy.singleton;  
  
import java.util.Map;  
import java.util.concurrent.ConcurrentHashMap;  
  
*/\*\*  
 \* 2019/4/1  
 \* 莫非  
 \*/*public class ContainerSingleton {  
 private ContainerSingleton() {  
 }  
  
 private static Map<String, Object> *ioc* = new ConcurrentHashMap<>();  
  
 public static Object getBean(String className) {  
 synchronized (*ioc*) {  
 if (!*ioc*.containsKey(className)) {  
 try {  
 Object obj = Class.*forName*(className).newInstance();  
 *ioc*.put(className, obj);  
 return obj;  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 } catch (IllegalAccessException e) {  
 e.printStackTrace();  
 } catch (InstantiationException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 return *ioc*.get(className);  
 }  
}

## 线程内部的单例

线程内部的单例采用注册式单例，是伪线程安全的，可用来实现多数据源切换

package me.muphy.singleton;  
  
*/\*\*  
 \* 2019/4/2  
 \* 莫非  
 \*/*public class ThreadLocalSingleton {  
  
 private ThreadLocalSingleton() {  
 }  
  
 private static final ThreadLocal<ThreadLocalSingleton> *threadLocalInstance* =  
 new ThreadLocal<ThreadLocalSingleton>() {  
  
 @Override  
 protected ThreadLocalSingleton initialValue() {  
 return new ThreadLocalSingleton();  
 }  
 };  
  
 public static ThreadLocalSingleton getInstance() {  
 return *threadLocalInstance*.get();  
 }  
}