# 1线程不安全的懒汉单例

/\*\*

\* 对象设计（设计一个单例对象）

\* 单例对象设计？类的实例对象在一个JVM中只有一份

\*1.对类自身进行设计？

\*2.基于外部环境进行设计（对象创建以后存储到池中，需要时从池中获取）

\* 注意线程安全问题

\*/

/\*\*线程不安全的懒汉单例\*/

**class** Singleton01{//应用场景：单线程

**private** Singleton01() {

//执行一次说明只创建一次

System.***out***.println("Singleton01");

}

**private** **static** Singleton01 *singleton01*;

//线程不安全

**public** **static** Singleton01 getInstance(){

**if**(*singleton01* == **null**) {

*singleton01* = **new** Singleton01();

}

**return** *singleton01*;

}

}

# 2. 线程安全的懒汉单例

/\*\*线程安全的懒汉单例\*/

**class** Singleton02{//应用场景：大对象，稀少用，并发量不能太大

**private** Singleton02() {

System.***out***.println("Singleton02");

}

**private** **static** Singleton02 *singleton02*;

//线程安全但性能降低

**public** **static** **synchronized** Singleton02 getInstance(){

**if**(*singleton02* == **null**) {

*singleton02* = **new** Singleton02();

}

**return** *singleton02*;

}

}

# 3. 双重验证的懒汉单例

/\*\*双重验证的懒汉单例 \*/

**class** Singleton03{//应用场景：大对象，稀少用，并发量不能太大

**private** Singleton03() {

System.***out***.println("Singleton03");

}

/\*volatile 用于修饰属性

\* 1.可以保证对象修改时的可见性

\* 2.可以禁止指令重排序

\* 3.不能保证原子性 \*/

**private** **static** **volatile** Singleton03 *singleton03*;

**public** **static** Singleton03 getInstance(){

//第三个线程进来时就不会进入if语句而被锁住，直接执行return语句

**if**(*singleton03* == **null**) {

//静态对象使用的锁对象为静态方法所在类的字节码对象

**synchronized**(Singleton03.**class**) {

**if**(*singleton03* == **null**) {

*singleton03* = **new** Singleton03();

}

}

}

**return** *singleton03*;

}

}

# 4. 饿汉单例

/\*\*饿汉单例 \*/

**class** Singleton04{//应用场景：小对象，频繁用,高并发

**private** Singleton04() {}

//对象在类加载时初始化

**private** **static** **final** Singleton04 ***instance*** = **new** Singleton04();

**public** **static** Singleton04 getInstance() {

**return** ***instance***;

}

}

# 5. 基于内部类实现饿汉单例

/\*\*基于内部类实现饿汉单例\*/

**class** Singleton05{//大对象，频繁用，高并发

**private** Singleton05() {}

**static** **class** Inner{

**private** **final** **static** Singleton05 ***instance*** = **new** Singleton05();

}

**public** Singleton05 getInstance() {

**return** Inner.***instance***;

}

}

# 6. 枚举实现的饿汉单例

/\*\*枚举实现的饿汉单例\*/

**enum** Singleton06{//一个特殊的类

***instance***;//一个实例（类加载时创建）

}

**public** **class** TestObjectInstance4 {

**public** **static** **void** main(String[] args) {

Thread t1 = **new** Thread() {**public** **void** run() {Singleton01.*getInstance*();};};

Thread t2 = **new** Thread() {**public** **void** run() {Singleton01.*getInstance*();}};

t1.start();

t2.start();

}

}

# 7. 用池实现单例

/\*\*用池实现单例\*/

**class** A{**public** **void** show(){System.***out***.println("A");}}

**class** B{**public** **void** show(){System.***out***.println("B");}}

**class** ObjectContainer{//思考spring的单例设计

**public** **static** Map<String, Object> *map* = **new** HashMap<String, Object>();

**static** {

*map*.put("A", **new** A());

*map*.put("B", **new** B());

}

**public** **static** Object get(String key) {

**return** *map*.get(key);

}

}

**public** **class** TestObjectInstance5 {

**public** **static** **void** main(String[] args) {

Object o1 = ObjectContainer.*get*("A");

Object o2 = ObjectContainer.*get*("A");

System.***out***.println(01==02);

}

}