## 单例模式

1. 懒汉式

/\*\*

\* 多线程的情况下线程安全的，因为某些情况JVM隐含地执行了同步

\* 1.由静态初始化器（在静态字段上或static{}块中的初始化器）初始化数据时

\* 2.访问final字段时

\* 3.在创建线程之前创建对象时

\* 4.线程可以看见它将要处理的对象时

\*

\* 内部类通过这个思想，进一步提升了这个饿汉模式，成为懒汉模式

\*

\* **@author** Administrator

\*

\*/

**public** **class** EagerSingleton {

**private** **static** Instance *instance* = **new** Instance();//在类加载的时候，instance赋的是null值，当都已经分配内存了，只有当第一次调用getInstance来初始化EagerSingleton类时，才被赋予了真正对象

**private** **static** **final** Instance ***instance1*** = **new** Instance();//在类加载的时候，就赋了真正对象了，不加final的话更节省资源，相当于单线程中的懒汉模式了吧！

**private** EagerSingleton(){};//私有构造器，是为了在不让程序员显示的创建这一类对象，只有在调用此类的静态成员（除static final，反射）时，虚拟机才去主动的创建一个此对象。所以保证了整个程序生命过程中只有一个这个对象。

//因为这个类，相当于一个工具类（单例工厂类），用来产生单例，所以整个过程中只需要有一个工厂就行了，通过new 来创建工厂毫无意义。

//最好是用private修饰，这样以后看着类时，很直观的知道，这个类不应该去new它

**public** **static** Instance getInstance(){

**return** *instance*;

}

}

1. 饿汉式

/\*\*

\* 这个感觉还不如，饿汉模式呢，加载的时候都分配内存啦，唯一的的区别是，如果单例工具类中有许多单例的话，饿汉式，无论调用哪个单例，其他单例都会初始化！

\* **@author** Administrator

\*

\*/

**public** **class** LazySingletonDoubleCheck {

**private** **volatile** **static** Instance *instance* = **null**;

**private** LazySingletonDoubleCheck(){};

**public** **static** Instance getInstance(){

**if**(*instance* == **null**){//去主存中去看有没有实例化

**synchronized** (LazySingletonDoubleCheck.**class**) {

**if**(*instance* == **null**){//主要是为了，当好几个线程都已经判断第一个instance==null时，当第一个线程释放锁时，还是有线程进入这里的，所以还要判断一下

*instance* = **new** Instance();

}

}

}

**return** *instance*;

}

}

**public** **class** LazySingletonInnerClass {

**private** LazySingletonInnerClass(){};

**private** **static** **class** SingletonHolder{

**private** **static** Instance *instance* = **new** Instance();

}

// public static void getString(){

// System.out.println("-----------------------------");

// }

**public** **static** Instance getInstance(){

**return** SingletonHolder.*instance*;

}

}

/\*\*

\* 枚举模式，只能一个枚举定义一个单例

\* **@author** Administrator

\*

\*/

**public** **enum** LazySingletion {

***INSTANCE***

// ,INSD{

// public Insd getIns() {

//

// return new Insd();

// }

// }

;

**private** Instance instance;

**private** LazySingletion() {

instance = **new** Instance();

}

**public** Instance getInstance(){

**return** instance;

}

}