**单例类模式**

参考：<https://www.cnblogs.com/kuoAT/p/6725808.html>

1. 懒汉式

// 首次调用getInstance()时初始化单例，线程不安全

**public** **class** Singleton {

**private** **static** Singleton instance;

**private** Singleton (){}

**public** **static** Singleton getInstance() {

**if** (instance == **null**) {

        instance = **new** Singleton();

    }

**return** instance;

    }

}

// 首次调用getInstance()时初始化单例，线程安全

**public** **class** Singleton {

**private** **static** Singleton instance;

**private** Singleton (){}

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

**if** (instance == **null**) {

        instance = **new** Singleton();

    }

**return** instance;

    }

}

1. 饿汉式

这种方式基于classloder机制避免了多线程的同步问题， instance在类装载时就实例化。但是当使用多个类加载器加载时，各个加载器各自实例化，会出现多个实例对象。

// 类加载时初始化单例，线程安全

**public** **class** Singleton {

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

**private** Singleton (){}

**public** **static** Singleton getInstance() {

**return** instance;

    }

}

// 静态代码块，类加载时初始化单例，线程安全

**public** **class** Singleton {

**private** **static** Singleton instance = **null**;

**static** {

     instance = **new** Singleton();

    }

**private** Singleton (){}

**public** **static** Singleton getInstance() {

**return** **this**.instance;

    }

}

1. 静态内部类

参考：<https://blog.csdn.net/qq_43279637/article/details/84982874>

Singleton类被装载了，instance不一定被初始化。因为SingletonHolder类没有被主动使用，只有显示通过调用getInstance方法时，才会显示装载SingletonHolder类，从而实例化instance。

**public** **class** Singleton {

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

**private** **static** **final** Singleton INSTANCE = **new** Singleton();

    }

**private** Singleton (){}

**public** **static** **final** Singleton getInstance() {

**return** SingletonHolder.INSTANCE;

    }

}

1. 双重校验锁

**public** **class** Singleton {

// volatile防止指令重排序，只能对变量使用，从内存而不从寄存器中取值，不会阻塞线程

**private** **volatile** **static** Singleton singleton;

**private** Singleton (){}

**public** **static** Singleton getSingleton() {

**if** (singleton == **null**) {

// synchronized可对变量和方法使用，可能造成线程阻塞

**synchronized** (Singleton.**class**) {

**if** (singleton == **null**) {

             singleton = **new** Singleton();

         }

        }

    }

**return** singleton;

    }

}

1. 枚举类

线程安全，不能懒加载，天然防止反射和反序列化

**public** **enum** Singleton {

INSTANCE;

**public** **void** doSomething() {

   }

}