1. 熟练掌握单例模式的常见写法。

代码在我的github上;

1. 总结每种单例写法的优、缺点。

2.1、饿汉式单例

饿汉式单例是在类加载的时候就立即初始化，并创建单例对象。线程绝对安全，而且被final修饰后，对象是唯一的，不可能存在访问安全问题。

优点: 没有加任何的锁、执行效率比较高，在用户体验上来说，比懒汉式更好。

缺点：类加载的时候就初始化了，不管用与不用都是占着空间，浪费内存

2.2、懒汉式单例

懒汉式单例是被外部类调用的时候内部类才会加载

优点：与饿汉式单例对比，它只是被调用时才会占用内存空间

缺点：多个线同时调用getInstance方法时，线程是不安全的，需要人为去加sychronized关键字和双重锁机制去解决线程问题，底层JVM给对象分配内存地址和初始化对象等等操作时会出现JVM执行顺序不一样问题，加volatile解决此问题，但被volatile修饰的变量的值，将不会被本地线程缓存，所有对该变量的读写都是直接操作共享内存,从而确保多个线程能正确的处理该变量。

2.3、内部内单例

内部内单例是结合了懒汉式单例的优点和饿汉式优点（不用加sychronized关键字）

优点： 调用方法时加载内部类去创建实例，创建的实例是静态的用了final修饰，不存在线程安全问题;

缺点：通过反射构造方法去创建一个新的实例，破坏了单例原则，可以在私有的构方法加决断，阻止通过反射来创建一个新的单例;

2.4、注册式单例

注册式单例就是将每一实例都登记到某个地方，使用唯一的标识获取实例。

它分为2种写法：一种为容器缓存，一种为枚举登记

优点：枚举登记在java中是安全单例模式，它的层底很好的规避了通过反射、序列化接口来创建新的实例;

缺点：底层是通过饿汉式创建的实例，可以说不算是缺点;

2.5、反射式单例

反射式单例是将这个类实现Serializable接口;

优点：

缺点：通过反序化时，创建出来了一个新的单例，破坏单例原则，因为底层通过反射建构出来的一个新的单例对象，可以用readResolve方法很好的解决底层反射的新的单例对象不需要返回，直接这回类里的实例，这样会损耗内存

2.6、本地线程单例

优点：在同一个线程内创建的实例是唯一的单例对象

缺点：多个线程创建的实例的对象不是唯一的，破坏单例原规

1. 思考破坏单例模式的方式有哪些？并且归纳总结。

破坏单例的方法有：1.通过线程，2.通过反射，3，通过构造方法

饿汉式单例和注册式单例不能被以上3点破坏

懒汉式单例、序例化单例能被以上3点破坏

1. 梳理内部类的执行逻辑，并画出时序图。

