## 第八章 单例模式

1. **在（B）时可使用单例模式。**
   1. 隔离菜单项对象的创建和使用
   2. 防止一个资源管理器窗口被实例化多次
   3. 使用一个已有的查找算法而不想修改既有代码
   4. 不能创建子类，需要扩展一个数据过滤类
2. **以下关于单例模式的描述正确的是（B）。**
   1. 它描述了只有一个方法的类的集合。
   2. 它能够保证一个类只产生一个唯一的实例。
   3. 它描述了只有一个属性的类的集合。
   4. 它能够保证一个类的方法只能被一个唯一的类调用。
3. **以下（B）不是单例模式的要点。**
   1. 某个类只有一个实例
   2. 单例类不能被继承
   3. 必须自行创建单个实例
   4. 必须自行向整个系统提供单个实例
4. **分析并理解饿汉式单例和懒汉式单例的异同。**

1）延迟加载：饿汉式单例模式在类被加载时就将自己实例化；懒汉式单例模式在第一次使用时被创建，无需一直占用系统资源，实现了延迟加载。

2）线程安全：饿汉式单例模式线程安全吗，懒汉式单例模式线程不安全

3）响应时间：由于饿汉式单例模式的单例对象在类加载是对象就得以创建所以比懒汉式单例模式响应时间短。

1. **什么是双重检查锁定？为什么进行双重检查锁定？Java如何实现双重检查锁定？**
2. **分别采用双重检查锁定和IoDh实现8.3节的负载均衡器。**

|  |  |
| --- | --- |
| **/\*\***  **\***双重检查锁定  \*  **\*/**  **public class** Singleton {  **private volatile static** Singleton *instance* = **null**;  **private** Singleton(){}  **public static** Singleton getInstance(){  *//第一重判断* **if** (*instance*==**null**){  *//锁定代码块* **synchronized** (Singleton.**class**){  *//第二重判断* **if** (*instance*==**null**){  *instance*=**new** Singleton();*//创建单例实例* }  }  }  **return** *instance*;  } } | **饿汉式单例类不能实现延迟加载，不管将来用不用始终占用内存；**  **懒汉式单例类线程安全控制繁琐，而且性能受影响；**  **通过使用IoDH既可以实现延迟加载，又可以保证线程安全，不影响系统性能，不失为最好的java语言单例模式的实现方式；其缺点你是与编程语言本身的特性相关，很多面向对象语言并不支持IoDh。** |
| ***/\*\*  \* IODH：Initialization on Demand Holder  \*/* public class Singleton{  private Singleton(){}    *//静态内部类* private static class HoldClass{  private final static Singleton *instance* = new Singleton();  }    public static Singleton getInstance(){  return HoldClass.*instance*;  } }** |