## 一、单例模式

单例模式是最简单的设计模式，在程序运行期间，类只有唯一一个实例。能使用单例模式的类不能含有状态数据，只能提供一些服务方法。

单例模式的类如果存在状态数据，在多线程下，很容易造成数据的不一致，这一点在应用Spring bean的时候尤其注意(Spring Bean默认是单例模式).

应用单例模式，需按照下面三步进行：

1) 类的构造函数必须是私有化的，这样外部才不能随便实例化对象。

2) 类中存在一个静态私有变量，存放类实例的引用(内部可以实例化)

3) 向外提供一个公用的静态方法，将静态私有变量返回给调用者

### 1.1 最简单的单例模式(饿汉模式)

public class Singleton {  
 */\*\*  
 \* 存放单例对象的引用  
 \*/* private static Singleton *instance* = new Singleton();  
  
 */\*\*  
 \* 构造函数私有化  
 \*/* private Singleton(){  
  
 }  
  
 */\*\*  
 \* 将单例实例返回给调用者  
 \** ***@return*** *\*/* public static Singleton getInstance(){  
 return *instance*;  
 }  
}

上面的实现方式，在JVM载入Singleton类的时候，在初始化步骤，会对初始化静态变量，所以当外部调用者调用getInstance()方法时，返回的就是内存中唯一的对象引用。

这种方式，由于是在调用者获取实例的时候，就已经实例化好了，也把这种实现方式叫做“饿汉模式”。

### 1.2 懒汉模式与双重判断

懒汉模式在在调用者需要的时候，才对对象进行实例化，如下：

public class Singleton\_02 {  
 */\*\*  
 \* 私有化构造函数  
 \*/* private Singleton\_02(){  
  
 }  
   
 */\*\*  
 \* 存放单例对象的引用  
 \*/* private static Singleton\_02 *instance*;  
  
 */\*\*  
 \* 将单例实例返回给调用者  
 \** ***@return*** *\*/* public static Singleton\_02 getInstance(){  
 if(*instance* == null){  
 *instance* = new Singleton\_02();  
 }  
  
 return *instance*;  
 }  
}

这种方式，遇到多线程，则可能会产生多个实例：当两个线程同时到达instance == null时，此时还没实例化对象，此时会各自实例化一个对象，然后返回。

在多线程下，遇到数据不一致时，通常的做法是加锁：synchronized

*/\*\*  
 \* 将单例实例返回给调用者  
 \** ***@return*** *\*/*public static Singleton\_02 getInstance(){  
 if(*instance* == null){  
 synchronized (Singleton\_02.class){  
 *instance* = new Singleton\_02();  
 }  
 }  
  
 return *instance*;  
}

这种方式还是存在问题，原因在于：两个线程同时到达instance == null时，线程1跑得快一点，先获得了锁，线程1运行完后，线程2获得锁，还是会执行：instance = new Singleton\_02();这个时候还是会实例化多个对象。

终极方式就是采用锁+双重判断，如：

*/\*\*  
 \* 将单例实例返回给调用者  
 \** ***@return*** *\*/*public static Singleton\_02 getInstance(){  
 if(*instance* == null){  
 synchronized (Singleton\_02.class){  
 if(*instance* == null){  
 *instance* = new Singleton\_02();  
 }  
 }  
 }  
  
 return *instance*;  
}

这种情况下，即使两个线程同时达到instance == null，线程1获得锁，先实例化了对象，此时instance已经有值了；线程2获得锁，进行第二层的instance == null判断时，不成立，就不会进行实例化操作了。

### 1.3 单例模式与多线程

单例模式虽然是最简单的设计模式，但是当遇到多线程时，还是有很多事情需要考虑的(典型的1+1 > 2).

## 二、代理模式



## 三、工厂模式

### 3.1 简单工厂



### 3.2 工厂方法模式



### 3.3 抽象工厂



## 四、适配器模式

