第5次课-设计模式-第1讲-创建型-单例模式

内容一：设计模式-分类？

创建型模式（创建和管理对象）->5种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式

结构型模式->7种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式

行为型模式->11种：策略模式、模板方法模式、观察者模式、迭代器模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式

解决问题：你对单例模式理解？

内容二：创建型模式-单例模式-基本概念？

第一点：单利模式->为什么使用单例模式？

节约内存开销，提高效率（性能），提高资源使用率。

第二点：单例模式-特点？

判定一个类结构（它使用什么设计模式？）

特点一：构造方法私有化

特点二：定义静态方法返回当前对象

特点三：确保这个对象是唯一的

特点四：确保在序列化和反序列化操作的过程中同样保证同一个对象。

特点五：不允许有子类（太监类->final）

第三点：单例模式-角色划分？

两个角色

角色一：客户端（调用）

角色二：单例类

例如：在Activity里面调用Application创建图片

角色一：客户端（调用者）->Activity

角色二：单例类->Application

第四点：单例模式-UML类图结构？

补充一个知识？

Java序列化和反序列知识？

Intent其实就实现了序列化和反序列化

在纯Java语言里面->序列化和反序列代码

内容三：创建型模式-单例模式-模式变种？

第一点：单例模式-Java语言

1、单例模式->恶汉式

优点：安全（为你提前准备好，不管你用不用）

缺陷：耗费内存

2、单例模式->懒汉式

优点：性能高->在一定程度上节约了内存（用到了我才给你，创建）

缺点：多线程问题很难解决（并发）

Sptring框架、Hebernate框架等等…

问题一：不加锁？

多线程并发情况下，创建多个对象

解决方案：加锁解决多创建多个对象问题

问题二：阻塞？

100个线程访问这个对象，等待上一个线程用完了才允许使用（耗费性能）

3、单例模式-双重检查

优势：既能够保证创建对象单例对象，同时也保证了多线程安全

第一步：分析编译器编译过程

以下代码做了什么事情？

Singleton instance = new Singleton()

1-首先：分配内存（new关键字）

2-其次：调用构造方法初始化参数

3-最后：将instance对象指向这快内存区域（内存空间）

问题：双重检查失败（失效）？

在Java虚拟机（JVM 1.5之前）中Cache、寄存器到主内存回写数据顺序很有可能乱序，有可能是1-2-3，也有可能是1-3-2。

1-2-3，是先初始化，在指向内存

1-3-2，是先执行空内存，在初始化

解决方案

在Java虚拟机（JVM）1.5版本以及之后版本，做了优化关键字volatile （volatile含义：去掉虚拟机优化代码）

第二步：存在缺陷

为了提高代码稳定性，程序正确性，消耗性能。（权衡）

第三步：这样的场景下我们可以使用

使用没有进行优化过编译器或者共享内存处理器，正常运行

4、单例模式-静态内部类？->官方推荐使用

优势：既能够保证内存优化，同时也能够保证安全（单例）

5、单例模式-枚举？

高级用法（很多作用）

6、单例模式->集合方法-对象管理->不是标准的了

总结：第五种方式和第六种只是为了保证对象为一个

第二点：单例模式-Swift语言->结构->设计思想

和Java语法差别（本质是一模一样）->框架设计思想

第三点：单例模式->C++语言->结构->设计思想

C++模版编程->Java泛型设计

框架设计角度看待C++（学习C++不一样，效率会高）

而不是一个初级程序员

5个月学习->框架设计，深层次理解

总结

内容四：创建型模式-单例模式-具体应用?

ImageLoader、Application、LayoutInflater、系统服务（集合方式）等等… 地图管理器

开发中：工具类（用了单例、静态方法）