

不是看几本书 课堂能学好设计模式

通过这些 学会设计设计模式的思维

如果不在产品 工作中学习设计模式 就是空话

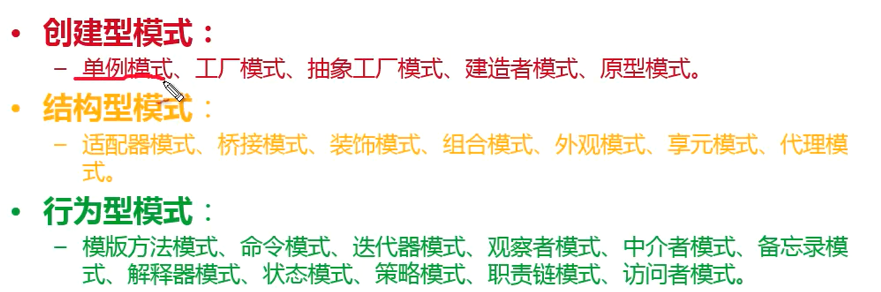


模式 ----- 就是套路 比较固定

更多是思维 思考方式 背设计模式的代码没有意义

思维 思考的方式最重要 编程语言就是工具

Gof23 ---- Group of Four ----四人帮 总结了23中套路

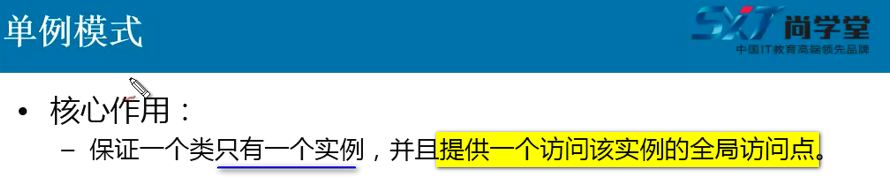


分成了创建型模式 结构型模式 和行为型模式

今天从创建者开始 就是用来建立对象的

建立的对象 有简单的 有复杂的

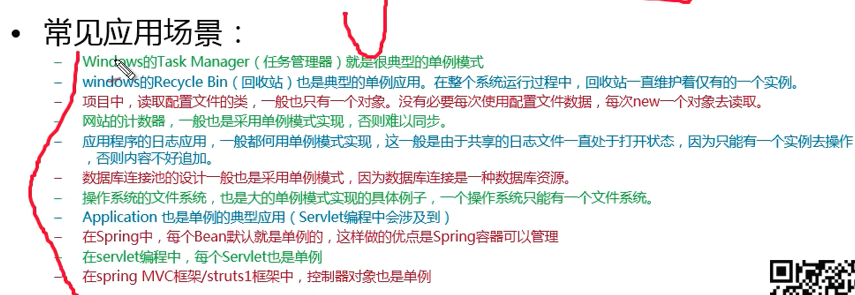
创建型模式 单例 工厂 抽象工厂 建造者 原型



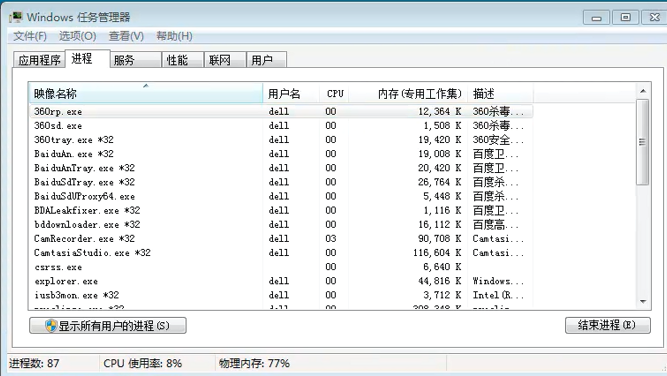
单例模式的细节要详细过一遍 ---- 为了面试

单例 核心作用----保证一个类有一个对象

**一定要提供一个访问该实例的的全局访问点**



比如 我启动windows任务管理器



这是一个 如果再单击右键 启动任务管理器



依旧还是这一个任务管理器

就无论启动多少个 任务管理器只有一个

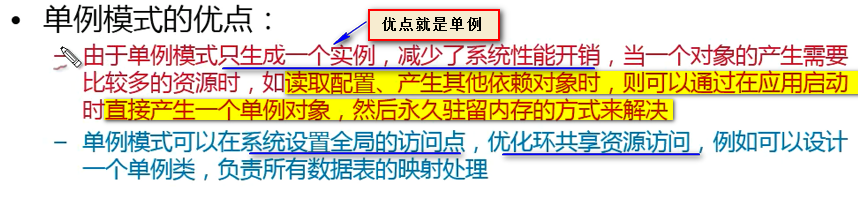
回收站也是一个单例应用

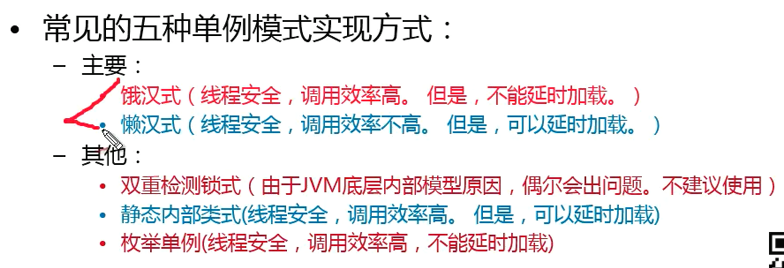


配置对象只有一个就够了



数据库连接池的设计 也是单例的模式





后面三种 会在面试中加分

先看饿汉式

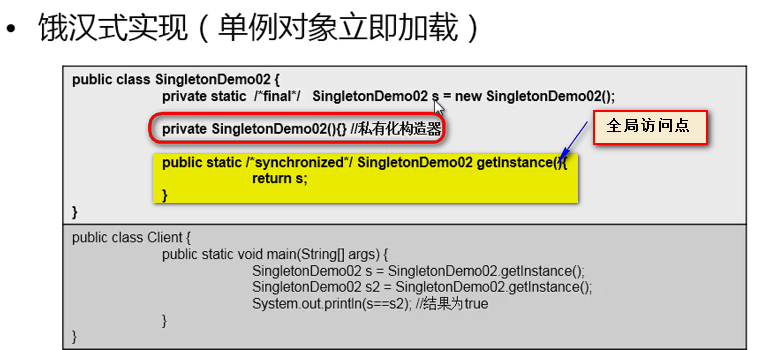
饿汉式线程安全 调用效率高 ---- 但是不能延时加载

对于一个单例模式 实现有三点

需要构造器私有化 --- 别人可以访问不了

提供一个static的变量 --- 私有的 static是属于类变量 这样类下面就有一个属性 --- 指定了我们的一个对象

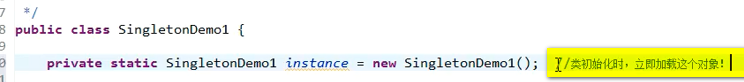
提供了一个全局访问点，别的都是私有的



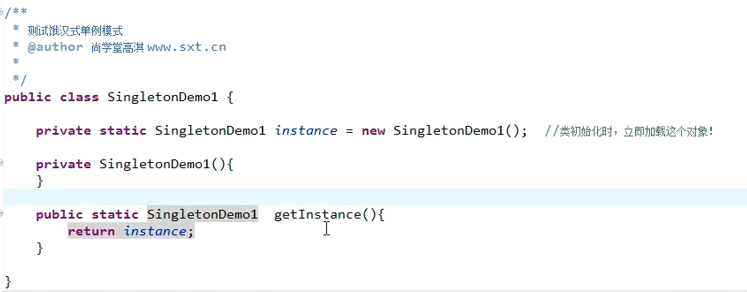
为什么称之饿汉？

类加载器加载这个类的时候 就加载了private static SingletonDemon02这个 就直接把这个对象new出来 无论后面要不要这个类

上来就建立好 实现了这个对象的立即加载



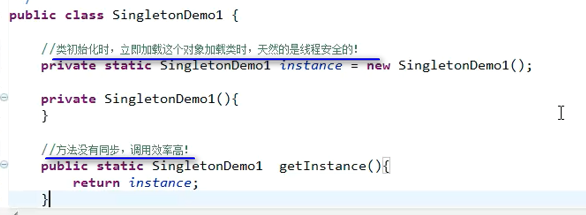
类加载了 如果new这个对象耗费时间 但是后面不用的时候 这个就白白加载了 所以一般希望延时加载 ----- 懒汉式



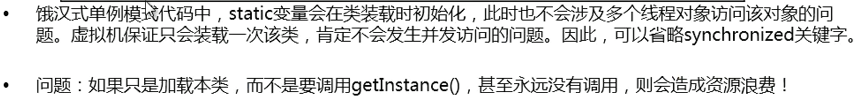
这个全局访问点 不需要添加synchronized 关键字 因为

【老师说 这儿 是前面讲的线程天然安全的问题】

天然线程安全 所以用的时候 直接返回对象 不需要同步块 ---- 没有同步块 调用效率就高



但是立即加载就没有延时加载的优势 可能会浪费资源



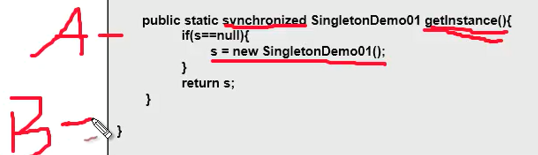
懒汉式的实现

有一个要点 就是static这个没有初始化 也就是类加载的时候 这个属性就是设置为null



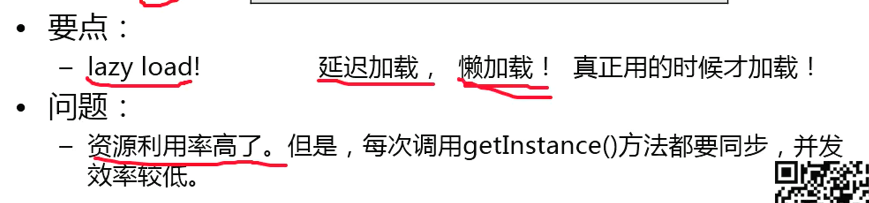
由于我这个方法可能多个线程调用

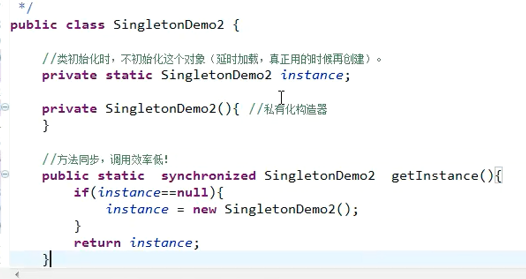
所以 添加一个synchronized

这样可以避免高并发的时候 出现了多个单例  


延迟加载 就是 上来没有new好 只有在中间需要的时候 才new 这是好处

懒汉式实现延迟加载 也叫懒加载 ----- 懒 就是到时候再说

但是 多个线程调用的时候 效率就低了



最后做了配置 测试应该差了两个数量级

下面这三种模式 来融合懒汉式 饿汉式来融合

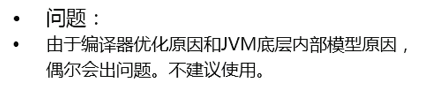
双重检测锁实现

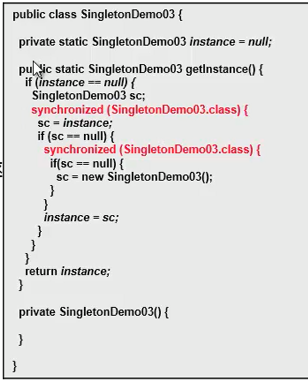
 ---- 这个有点问题

实际工作中用不着

双重检测锁实现的单例：思想 --- 之前懒汉式 是方法前面加了一个同步 ---- 思想就是 同步块更加细粒度 放到if下面

这样 只有第一次的时候 才加synchronized 如果不为空就直接返回 只有第一次延时加载

 编译器优化 和JVM底层内存模型 --- 导致数据调整 导致这里面出问题

这种不用试验了

====

静态内部类实现（也是一种懒加载的方式） ---- 实际生产中 用的非常多 效果很好、

【**为什么不使用两个类？要用内部类呢？原因就是 我的外部类可以直接访问内部类私有成员**】

线程安全 --- 调用效率高 懒加载

没有直接有static类

在静态内部类中 里面定义一个对象

全局访问点是访问静态内部类的时候才有的 这个式线程安全 是懒加载模式

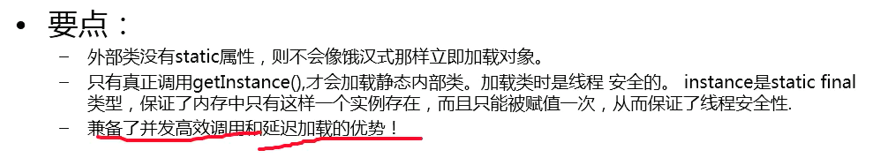
第一次初始化这个类的时候 外部类被加载 但是内部类没有被加载 所以 在内部类中是饿汉式---但是在外部类看来 不是饿汉式 是懒汉式 ---- 不一定初始化这个静态内部类 ------ 构建全局访问点的时候 也是访问到的时候

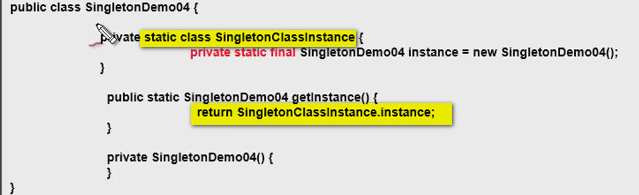
这样就变相实现了延时加载

因为 

这个是final了 ----- 就不能被修改了 相当于构造方法私有化

这就兼备了





还有一种是枚举方式 兼备这些特点的话 枚举方式是最简单的

枚举类天然都是单例的



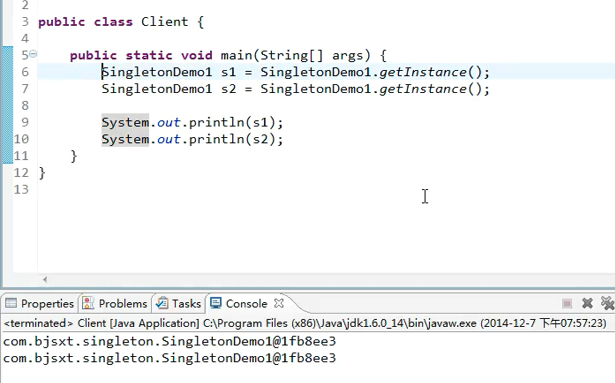
通过JVM内部

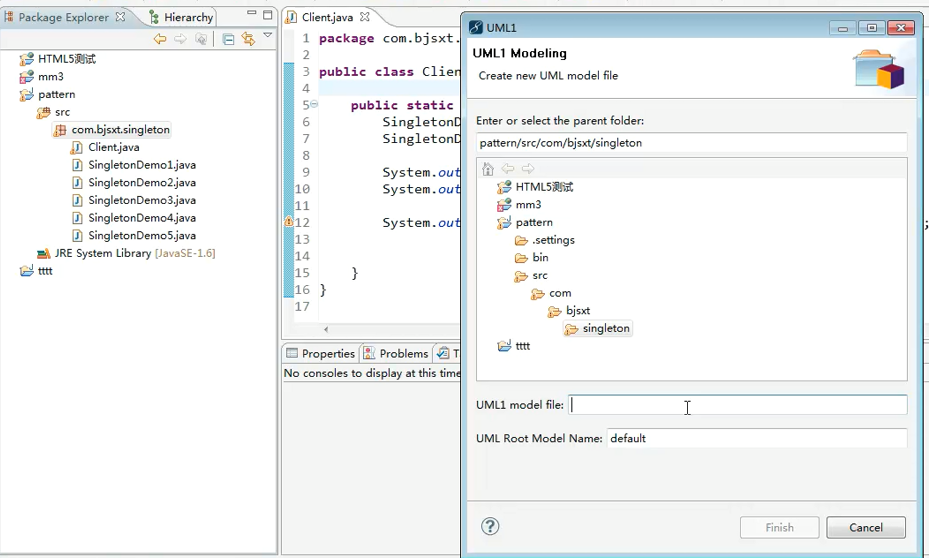


反射可以去调用私有化的构造方法

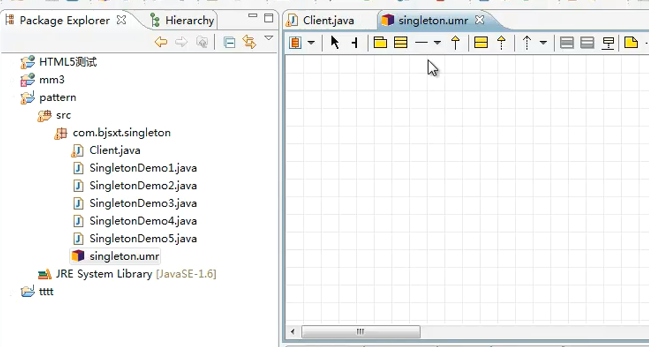


进行简单的测试



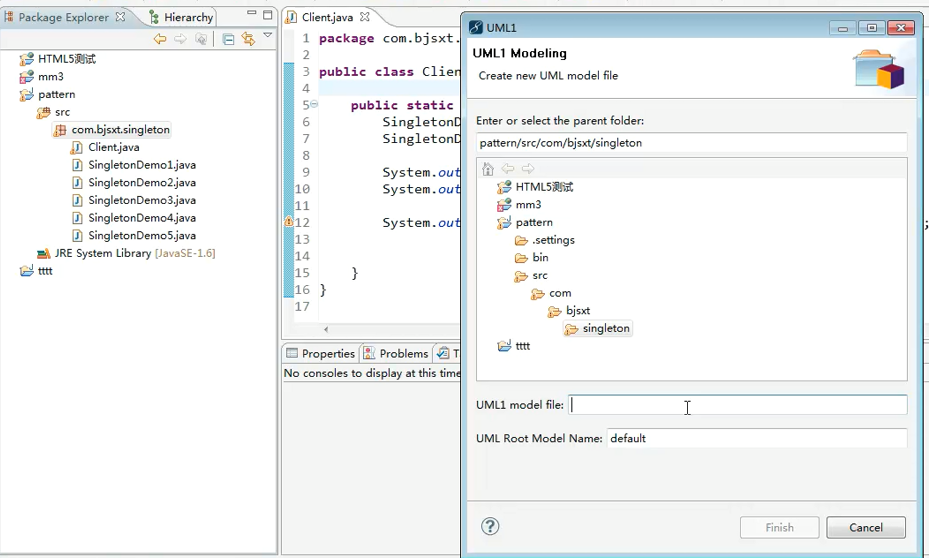


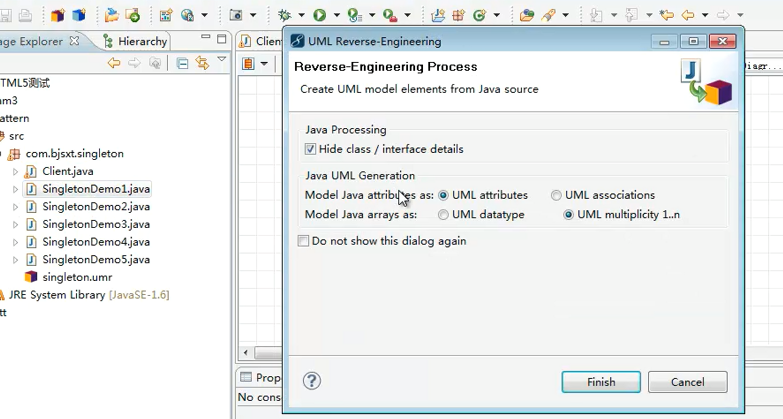
现在在这里创建UML图 每一个包下面



这个是MyEclipse的插件去画的

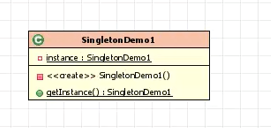
拖进来一个类 之后



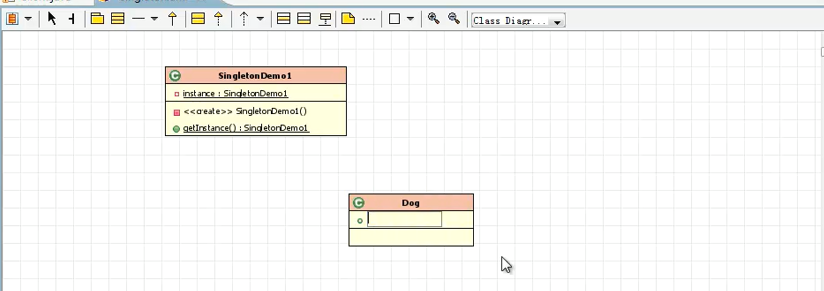




这个选中 类中的内容都看不到



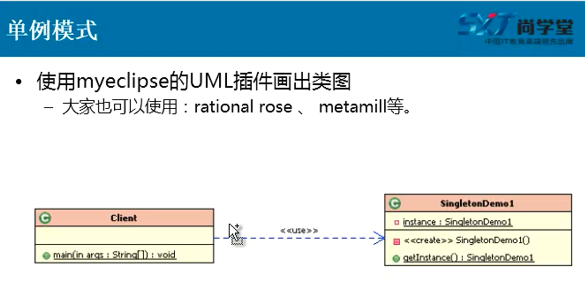
可以自己画



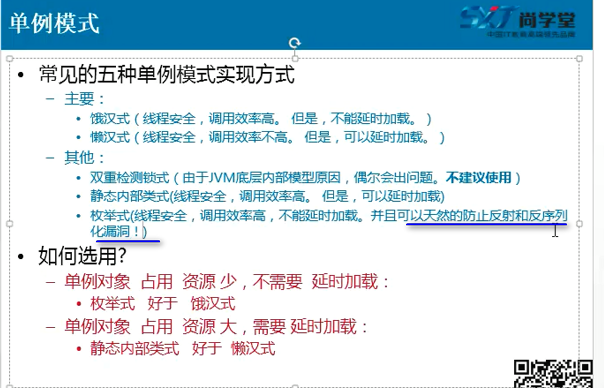
最后自己画出来关系



可以把这儿类的结构图 可以输出图片



总结

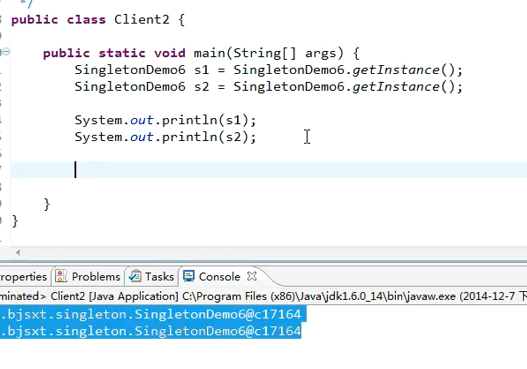


前面介绍了创建单例的模式策略

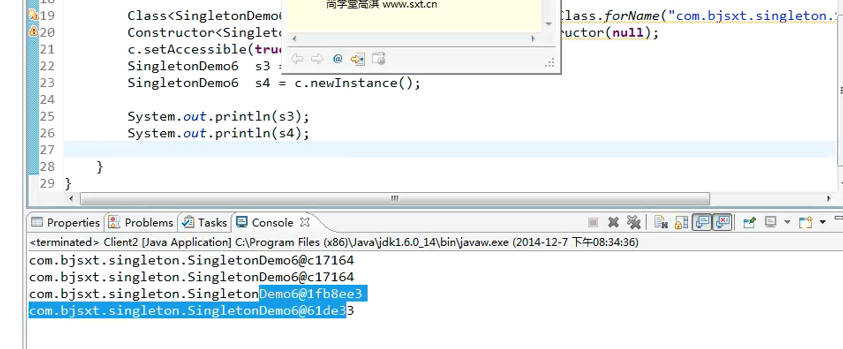
如何通过反射破解单例模式（枚举是不能破解的）

实际上枚举方式是最安全的

通过反射如何破解？【工作中不用考虑这样的问题】



现在添加反射



这样就完蛋了

现在 多次调用 创建runtimeException



还有一个问题 通过序列化和反序列化来破解这个问题

7：42