Volatile保证可见性和有序性**：**

是两条实现原则：

1.Lock前缀指令会引起处理器缓存会写到内存

当对volatile变量进行写操作的时候，JVM会向处理器发送一条lock前缀的指令，将这个缓存中的变量回写到系统主存中

1. 一个处理器的缓存回写到内存会导致其他处理器的缓存失效
2. 只有当线程T对变量V执行的前一个动作是load，线程T对变量V才能执行use动作；同时只有当线程T对变量V执行的后一个动作是use的时候线程T对变量V才能执行load操作 只有当线程T对变量V执行的前一个动作是assign的时候，线程T对变量V才能执行store动作；同时只有当线程T对变量V执行的后一个动作是store的时候，线程T对变量V才能执行assign动作。

处理器使用嗅探技术保证内部缓存 系统内存和其他处理器的缓存的数据在总线上保持一致。（什么是缓存一致性协议呢？  
每个处理器通过嗅探在总线上传播的数据来检查自己缓存的值是不是过期了，当处理器发现自己缓存行对应的内存地址被修改，就会将当前处理器的缓存行设置成无效状态，当处理器要对这个数据进行修改操作的时候，会强制重新从系统内存里把数据读到处理器缓存里。）

（禁止指令重排）

1.在每个volatile写操作前插入StoreStore屏障；对于这样的语句Store1; StoreStore; Store2，在Store2及后续写入操作执行前，保证Store1的写入操作对其它处理器可见。

2.在每个volatile写操作后插入StoreLoad屏障；对于这样的语句Store1; StoreLoad; Load2，在Load2及后续所有读取操作执行前，保证Store1的写入对所有处理器可见。

3.在每个volatile读操作前插入LoadLoad屏障；对于这样的语句Load1;LoadLoad; Load2，在Load2及后续读取操作要读取的数据被访问前，保证Load1要读取的数据被读取完毕。

4.在每个volatile读操作后插入LoadStore屏障；对于这样的语句Load1; LoadStore; Store2，在Store2及后续写入操作被刷出前，保证Load1要读取的数据被读取完毕。

## ReentrantLock:

ReentrantLock lock = new ReentrantLock();//创建一个锁对象。

lock.lockInterruptibly();//加的是可中断锁，等待锁的过程中会立即响应中断。即，在获取锁之前就会中断线程。

Lock.unlock();//释放锁最好在finally语句中释放，否则可能导致锁无法释放。

lock.lock();和可中断锁一样,在调用lock.interrupt()都会响应中断，不同的是可中断锁会立即响应中断，lock()方法会在中断点响应中断，比如sleep()方法 。

Lock.trylock()尝试获取锁，Lock.trylock(long time, TimeUnit unit)在一定时间范围内重复尝试获取锁。这两个都是成功返回true,失败返回false.

lock.wait()方法最终通过ObjectMonitor的 wait(jlong millis, bool interruptable, TRAPS)实现

## wait的底层原理

    1、将当前线程封装成ObjectWaiter对象node

    2、通过ObjectMonitor::AddWaiter方法将node添加到\_WaitSet列表中

1. 通过ObjectMonitor::exit方法释放当前的ObjectMonitor对象，这样其它竞争线程就可以获取该ObjectMonitor对象（p21）
2. 泛型限定符，上界不存，下界不取。Plate<? extends fruit> = new Plate<Apple>(); 然而Plate<? extends fruit>（fruit是泛型的上界）所以可能是Plate< fruit>也可能是Plate<Apple> ,也可能是Plate<Banana>.所以存的时候不知道要存哪些类型。下界不取是同样的道理。

## 线程回调

为了实现异步执行。异步与同步的区别就是，有一个线程A调用了B，直到B一直运行完结果返回给A，A才继续往下执行，但是这样有一个弊端，这样子往下执行，实在浪费资源，所以才有了异步，A调用B，然后A继续往下执行，B拿到了结果，再返回给A，A再继续处理结果，这样很明显，可以让线程不用闲着。（https://www.jianshu.com/p/4ee633f75783）。

## 线程可重入原理：

P58 && https://blog.csdn.net/mulinsen77/article/details/88635558

## JVM 是如何实现线程的。P70

## Bean的生命周期：

## 持久层需要考虑的问题p85

## 数据库两种引擎

## 事务的隔离级别p120

## 驱动的注册

<https://blog.csdn.net/u013679744/article/details/56298283>

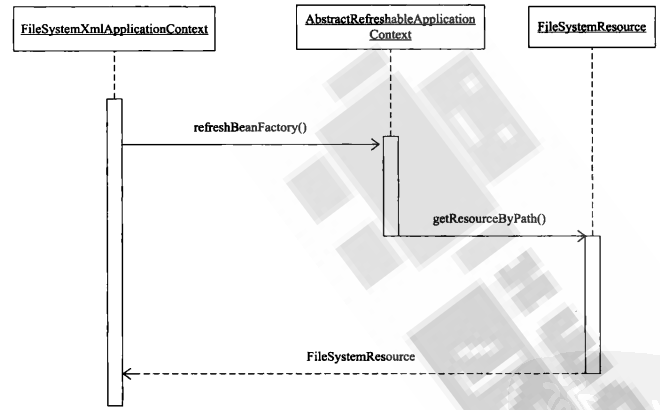
## 三次握手四次挥手

<https://www.cnblogs.com/amingxiansen/p/9072074.html>

## 银行家算法

安全序列是指一个进程序列{P1，…，Pn}是安全的，即对于每一个进程Pi(1≤i≤n），它以后尚需要的资源量不超过系统当前剩余资源量与所有进程Pj (j < i )当前占有资源量之和。

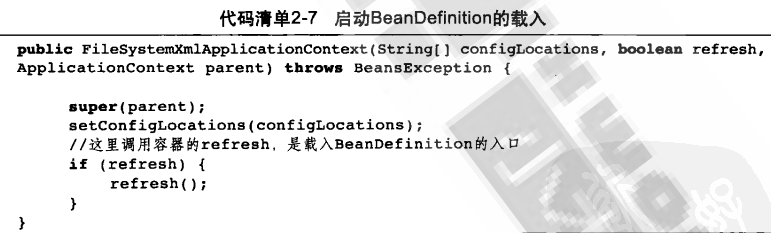
DefaultListableBeanfactory核心容器。



在初始化fileSystemXmlApplicationContext时，通过IOC容器的初始化的refresh来启动整个调用。Refresh方法会调用refreashBeanFactory使用的ioc容器是DefaultListableBeanfactory。

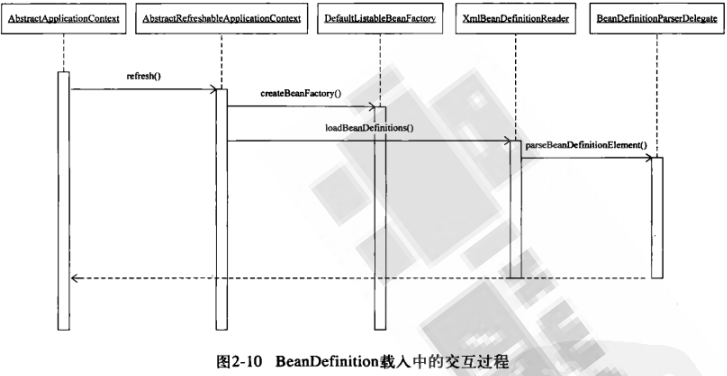
1587989518(1)





对ioc容器调用refresh方法的过程：





BeanPostProcessor的具体实现，他的before和after都发生在依赖注入之后。

InitializeBean()方法中完成了各种aware方法对beanname,beanfactory完成了设置。中间会调用init-method方法。

