1. final域
2. 重排序规则

对于final域，编译器和处理器要遵守两个重排序规则：

1. 在构造函数内对一个final域的写入，与随后把这个被构造对象的引用赋值给一个引用变量，这两个操作之间不能重排序。
2. 初次读一个包含final域的对象的引用，与随后初次读这个final域，这两个操作之间不能重排序。
3. 写final域的重排序规则

写final域的重排序规则禁止把final域的写重排序到构造函数之外，同时编译器在final域的写之后，构造函数return之前，插入一个StoreStore屏障，这个屏障禁止处理器把final域写重排序到构造函数之外。

1. 读final域的重排序规则

在一个线程中，初次读对象引用与初次读该对象包含的final域，JMM禁止处理器重排序这两个操作，编译器会在读final域操作前面插入一个LoadLoad屏障，确保读final域之前该对象已经初始化。

1. final引用不能逸出构造函数

写final域的重排序规则可以保证，在引用变量为任意线程可见之前，该引用变量指向的对象的final域已经在构造函数中被正确初始化了。要得到这种效果，还需要保证在构造函数内部，不能让这个被构造对象的引用被其他线程所见。

1. JSR-133增强final语义的意义

通过为final域增加写和读重排序规则，可以为Java程序员提供初始化安全保证，只要对象是正确构造的，那么不需要同步就可以保证任意线程都能看到这个final域在构造函数中被初始化之后的值 。

1. happens-before

JSR-133使用happens-before来指定两个操作之间的执行顺序，由于这两个操作可以在一个线程之内，也可以在不同线程之间，因此，JMM通过happens-before关系向程序员提供跨线程的内存可见性保障，即确保正确同步的多线程程序的执行结果不被改变。

1. JMM的内存可见性保证

Java的内存可见性保证分为三类：

1. 单线程程序

不会出现内存可见性问题，因为编译器和处理器会确保单线程程序执行结果和顺序一致性模型中的执行结果一致，通过重排序规则来禁止有依赖关系的重排序优化。

1. 正确同步的多线程程序

将具有顺序一致性，JMM通过限制编译器和处理器的重排序来提供内存可见性保证。

1. 未正确同步的多线程程序

JMM提供了最小安全性保障，线程执行时读取到的值，要么是之前某个线程写入的值，要么是默认值（0，null，false）。