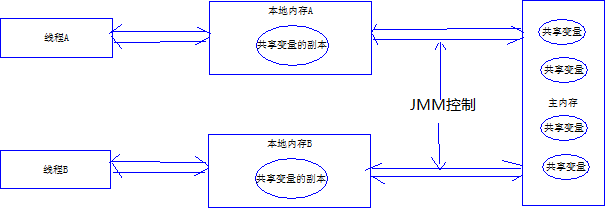
JMM（java memory model）-java内存模型

# 前言

什么是JMM？首先了解一下jvm中多线程工作原理。每个线程创建时JVM都会为其创建一个工作内存，用来存储该线程的私有数据(变量)。线程之间的共享变量存储在主内存中。如图所示(摘抄自网上)



JMM是通过各种操作来定义的，包括对变量的读/写操作，监视器额加锁和释放操作，以及线程的启动和合并操作。也就是说JMM就是一组规则。

# 重排序

# Happens-Before

Happens-Before 是JMM为程序中的所有操作定义了一个偏序关系。

规则包括

1. 程序顺序规则。如果程序中操作A在操作B之前，那么在线程中A操作将在B操作之前执行；
2. 监视器锁规则。在监视器锁上的解锁操作必须在同一个监视器锁上的加锁操作之前执行；
3. Volatile变量规则。对volatile变量的写入操作必须在对该变量的读操作之前执行。
4. 线程启动规则。在线程上对Thread.start的调用必须在其他线程检测到该线程已经结束之前执行。或者从Thread.join中成功返回，或者在调用Thread.isAlive时返回false。
5. 中断规则。当一个线程在另一个线程上调用interrupt时，必须在被中断线程检测到interrupt调用之前执行(通过抛出InterruptedException，或者调用isInterrupted和interrupted)。
6. 终结器规则。对象的构造函数必输在启动该对象的终结器之前执行。
7. 传递性。如果操作A在操作B之前执行。并且操作B在操作C之前执行，那么操作A必须在操作C之前执行。

# 发布

*/\*\*  
 \** ***@Description*** *不安全的延迟初始化(发布一个共享对象)  
 \** ***@ClassName*** *UnsafeLazyInitialization  
 \** ***@Author*** *Administrator  
 \** ***@Data*** *2019/6/29 11:44  
 \** ***@Version*** *1.0  
 \*/***public class** UnsafeLazyInitialization {  
 **private static** Object *object*;  
 **public static** Object getInstance(){  
 **if**(*object* == **null**){

*//不同线程之前的对象写入及读取不遵循Happens-Before关系*  
 *object* = **new** Object();  
 }  
 **return** *object*;  
 }  
}

*/\*\*  
 \** ***@Description*** *双重检查加锁是错误的(发布一个共享对象)  
 \** ***@ClassName*** *DoubleCheckedLocking  
 \** ***@Author*** *Administrator  
 \** ***@Data*** *2019/6/29 11:57  
 \** ***@Version*** *1.0  
 \*/***public class** DoubleCheckedLocking {  
 **private static** Object *object*;  
 **public static** Object getInstance(){  
 **if**(*object* == **null**){  
 **synchronized** (DoubleCheckedLocking.**class**){  
 **if** (*object* == **null**){  
 *//不同线程之前的对象写入及读取不遵循Happens-Before关系  
 object* = **new** Object();  
 }  
 }  
 }  
 **return** *object*;  
 }  
}

*/\*\*  
 \** ***@Description*** *线程安全的延迟初始化(发布一个共享对象)  
 \** ***@ClassName*** *SafeLazyInitialization  
 \** ***@Author*** *Administrator  
 \** ***@Data*** *2019/6/29 11:52  
 \** ***@Version*** *1.0  
 \*/***public class** SafeLazyInitialization {  
 **private static** Object *object*;  
 **public synchronized static** Object getInstance(){  
 **if**(*object* == **null**){

*object* = **new** Object();  
 }  
 **return** *object* }  
   
}

**import** java.util.HashMap;  
**import** java.util.Map;  
  
*/\*\*  
 \** ***@Description*** *不可变对象的初始化安全性  
 \** ***@ClassName*** *SafeStates  
 \** ***@Author*** *Administrator  
 \** ***@Data*** *2019/6/29 12:02  
 \** ***@Version*** *1.0  
 \*/***public class** SafeStates {  
 */\*\*  
 \* 初始化安全性只能保证通过final域可达的值从构造过程完成时开始的可见性。  
 \* 对于非final域可达的值，或者在构造过程完成后可能改变的值，必须采用同步来确保可见性。  
 \*/* **private final** Map<String, String> **states**;  
 **public** SafeStates(){  
 **states** = **new** HashMap<String, String>();  
 **states**.put(**"a"**, **"aa"**);  
 }  
 **public** String getValueByKey(String key){  
 **return states**.get(key);  
 }  
}

文字摘抄自<Java并发编程实践>