**一、谈谈对Java的认识**  
1、基本介绍  
 Java是一种跨平台的面向对象的开发语言。具有 简单性，面向对象，分布式，可靠，安全，解释型、平台无关，高性能，多线程，动态性等特点。

2、基本认识

（1）、简单。Java丢弃了C++ 中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别是，Java语言不使用指针，并提供了自动的垃圾回收机制，使得程序员不必为内存管理而担忧。

（2）、面向对象。封装、继承和多态三大特点； 只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为implements）。Java语言全面支持动态绑定。总之，Java语言是一个纯的面向对象程序设计语言。

（3）、分布式。Java语言支持网络应用的开发，在基本的Java应用编程接口中有一个网络应用编程接口（java.net），它提供了用于网络应用编程的类库，包括URL、URLConnection、Socket、 ServerSocket等。Java的RMI(远程方法激活)机制也是开发分布式应用的重要手段。

（4）、可靠（健壮）。Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。

（5）、安全。Java通常被用在网络环境中，Java提供了一个安全机制以防恶意代码的攻击。除了Java语言具有的许多安全特性以外，Java对通过网络下载的类具有一个安全防范机制（类ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类SecurityManager）让Java应用设置安全哨兵。

（6）、解释型。Java程序在Java平台上被编译为字节码，然后可以在实现Java平台的任何系统中运行。在运行时，Java平台中的Java解释器对这些字节码进行解释执行，执行过程中需要的类在联接阶段被载入到运行环境中。

（7）、平台无关。是指JAVA写的应用程序不用修改就可以在不同的软硬件平台上运行。依靠JAVA虚拟机JVM来实现，他依附在具体的操作系统上，有一套自己的虚机器指令，在JVM上有一个JAVA解释器用来解释JAVA编译器编译后的程序。

（8）、高性能。与解释型的高级脚本语言相比，Java的确是高性能的。事实上，Java的运行速度随着编译器技术的发展越来越接近于C++。

（9）、多线程。**JAVA**多线程实现方式主要有三种：

1：Java中访问控制

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 修饰词 | 本类内部 | 同一个包内 | 子类 | 其他类 |
| Private | 是 | 否 | 否 | 否 |
| Protected | 是 | 是 | 是 | 否 |
| Public | 是 | 是 | 是 | 是 |
| default | 是 | 是 | 否 | 否 |

2：PHP中访问控制权限

|  |  |  |  |
| --- | --- | --- | --- |
| 修饰词 | 本类内部 | 子类 | 其他类 |
| Private | 是 | 否 | 否 |
| Protected | 是 | 是 | 否 |
| public | 是 | 是 | 是 |

3：PHP中数字是有符号类型的，最大支持2的64次方。

**4：接口和抽象类的区别**

首先接口和抽象类的设计目的就是不一样的。**接口是对动作的抽象，而抽象类是对根源的抽象**。

**抽象类**：比如男人，女人这两个类，那我们可以为这两个类设计一个更高级别的抽象类--人。

**接口**：我们可以坐着吃饭，可以站着吃饭，可以用筷子吃饭，可以用叉子吃饭，甚至可以学三哥一样用手抓着吃饭，那么可以把这些吃饭的动作抽象成一个接口--吃饭。

所以在高级语言中（如Java,C#），一个类只能继承一个抽象类（因为你不可能同时是生物又是非生物）。但是一个类可以同时实现多个接口，比如开车接口，滑冰接口，踢足球接口，游泳接口。

5：抽象类和接口的对比

**1、抽象类和接口都不能被直接实例化，如果二者要实例化，就涉及到多态**。

如果抽象类要实例化，那么抽象类定义的变量必须指向一个子类对象，这个子类继承了这个抽象类并实现了这个抽象类的所有抽象方法。

如果接口要实例化，那么这个接口定义的变量要指向一个子类对象，这个子类必须实现了这个接口所有的方法。

**2、抽象类要被子类继承，接口要被子类实现**。

**3、接口里面只能对方法进行声明，不能有实现；抽象类既可以对方法进行声明也可以对方法进行实现。**

4、抽象类里面的抽象方法必须全部被子类实现，如果子类不能全部实现，那么子类必须也是抽象类。

接口里面的方法也必须全部被子类实现，如果子类不能实现那么子类必须是抽象类。

5、接口里面的方法只能声明，不能有具体的实现。这说明接口是设计的结果，抽象类是重构的结果。）

5、抽象类里面可以没有抽象方法。

6、如果一个类里面有抽象方法，那么这个类一定是抽象类。

7、抽象类中的方法都要被实现，所以抽象方法不能是静态的static，也不能是私有的private。

8、**接口（类）可以继承接口，甚至可以继承多个接口。但是类只能继承一个类。**

9、抽象级别（从高到低）：接口>抽象类>实现类。

10、抽象类主要是用来抽象类别，接口主要是用来抽象方法功能。当你关注事物的本质的时候，请用抽象类；当你关注一种操作的时候，用接口。

11、抽象类的功能应该要远多于接口，但是定义抽象类的代价较高。因为高级语言一个类只能继承一个父类，即你在设计这个类的时候必须要抽象出所有这个类的子类所具有的共同属性和方法；但是类（接口）却可以继承多个接口，因此每个接口你只需要将特定的动作方法抽象到这个接口即可。也就是说，接口的设计具有更大的可扩展性，而抽象类的设计必须十分谨慎。

**6：Java中重载与重写的区别**

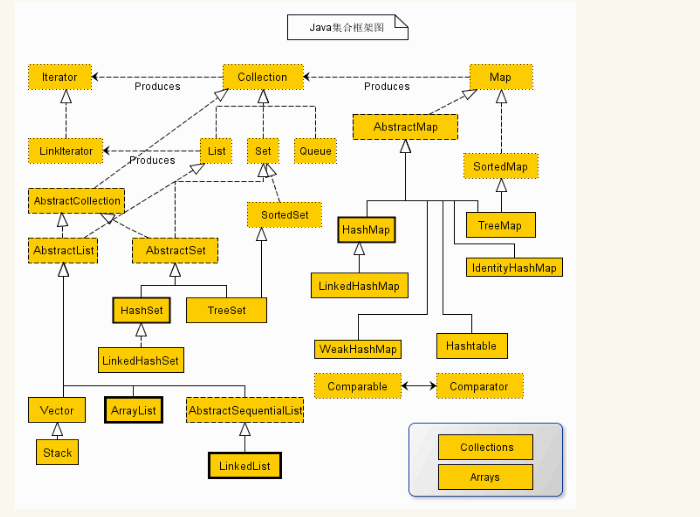
1. 重载(Overloading)  
    （1）重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。  
    （2）[Java](http://java.chinaitlab.com/)的方法重载，就是在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法, 这就是多态性。

2、重写（Overriding）  
  （1）如果在子类中定义某方法与其父类有相同的名称、参数和返回类型，我们说该方法被重写 (Overriding)，则新方法将覆盖父类中原有的方法（）如需父类中原有的方法，可使用super关键字，该关键字引用了当前类的父类。）。在[Java](http://java.chinaitlab.com/)中，子类可继承父类中的方法，而不需要重新编写相同的方法，但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。

（2）、这体现了父类与子类之间的多态性，对父类的函数进行重新定义。方法重写又称方法覆盖。  
（3）子类函数的访问修饰权限不能少于父类的；动态绑定。

**二、Java集合类解析**

详情链接：<http://www.cnblogs.com/leeplogs/p/5891861.html>



1、在使用Java的时候，我们都会遇到使用集合（[Collection](https://baike.baidu.com/item/Collection/80124)）的时候，但是[Java](https://baike.baidu.com/item/Java/85979) [API](https://baike.baidu.com/item/API/10154)提供了多种集合的实现。总的说来，[Java API](https://baike.baidu.com/item/Java%20API)中所用的集合类，都是实现了[Collection接口](https://baike.baidu.com/item/Collection%E6%8E%A5%E5%8F%A3)，他的一个类继承结构如下：

Collection<--List<--AbstractList<--[Vector](https://baike.baidu.com/item/Vector/3330482)

Collection<--List<--AbstractList <--[ArrayList](https://baike.baidu.com/item/ArrayList)

Collection<--List<--AbstractList <--AbstractSequentialList<--LinkedList

Collection<--Set<--AbstractSet<--[HashSet](https://baike.baidu.com/item/HashSet)

Collection<--Set<--AbstractSet<--HashSet<--LinkedHashSet

Collection<--Set<--SortedSet<--TreeSet

Collection<--Map<--AbstractMap<--HashMap<--LinkedHashMap

Collection<--Map<--AbstractMap<--WeakHashMap

Collection<--Map<--AbstractMap<--HashTable

Collection<--Map<--AbstractMap<--IdentityHashMap

Collection<--Map<--SortedMap<--TreeMap

注：

Vector : 基于Array的List，其实就是封装了Array所不具备的一些功能方便我们使用，它不可能走出Array的限制。性能也就不可能超越Array。所以，在可能的情况下，我们要多运用Array。

另外很重要的一点就是Vector“synchronized”的，这个也是Vector和ArrayList的唯一的区别。

[ArrayList](https://baike.baidu.com/item/ArrayList)：同Vector一样是一个基于数组实现的，但是不同的是ArrayList不是同步的，所以在性能上要比Vector优越一些。但是当运行到多线程环境中时，可需要自己在管理线程的同步问题。与Vector相比，如果不考虑线程安全因素，一般使用arrayList效率较高。

如果集合中的元素的数目大于目前集合数组的长度时，vector增长率为目前数组长度的100%，而arraylist增长率为目前数组长度的50%。如果在集合中使用数据量比较大的数据，用vector有一定的优势。

LinkedList：ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。LinkedList也是非同步的。它每一个节点（Node）都包含两方面的内容：1.节点本身的数据（data）；2.下一个节点的信息（nextNode）。所以当对LinkedList做添加，删除动作的时候就不用像基于Array的List一样，必须进行大量的数据移动。只要更改nextNode的相关信息就可以实现了。这就是LinkedList的优势。

**2、Map集合**

实现类：HashMap、Hashtable（同步的）、LinkedHashMap和TreeMap

HashMap

HashMap是最常用的Map，它根据键的HashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。因为键对象不可以重复，所以HashMap最多允许一条记录的键为Null，允许多条记录的值为Null，是非同步的

Hashtable

Hashtable与HashMap类似，是HashMap的线程安全版，它支持线程的同步，即任一时刻只有一个线程能写Hashtable，因此也导致了Hashtale在写入时会比较慢，效率较低。它不允许记录的key或者value为null。

LinkedHashMap

LinkedHashMap保存了记录的插入顺序，在用Iteraor遍历LinkedHashMap时，先得到的记录肯定是先插入的，在遍历的时候会比HashMap慢，有HashMap的全部特性。

TreeMap

TreeMap实现SortMap接口，能够把它保存的记录根据键排序，默认是按键值的升序排序（自然顺序），也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排过序的。不允许key值为空，非同步的；

ConcurrentHashMap

线程安全，并且锁分离。ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

**3、实现原理**

（1）、HashMap的实现原理

HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

<http://zhangshixi.iteye.com/blog/672697>

（2）、TreeMap实现原理

TreeMap的实现是红黑树[算法](http://lib.csdn.net/base/datastructure)的实现。 <http://blog.csdn.net/a616413086/article/details/52586006>

**4、HashMap和HashSet的区别**

<http://www.importnew.com/6931.html>

**三、Java中多线程的内容--同步机制**

参考网址：<http://blog.csdn.net/u012891504/article/details/51332298>

1、多线程的线程同步机制实际上是靠锁的概念来控制的。

2、在Java程序运行时环境中，JVM需要对两类线程共享的数据进行协调：  
1）保存在堆中的实例变量  
2）保存在方法区中的类变量

这两类数据是被所有线程共享的。程序不需要协调保存在Java 栈当中的数据。因为这些数据是属于拥有该栈的线程所私有的。

3、在java虚拟机中，每个对象和类在逻辑上都是和一个监视器相关联的。

1）对于对象来说，相关联的监视器保护对象的实例变量；

2）对于类来说，监视器保护类的类变量；

为了实现监视器的排他性监视能力，java虚拟机为每一个对象和类都关联一个锁。代表任何时候只允许一个线程拥有的特权。线程访问实例变量

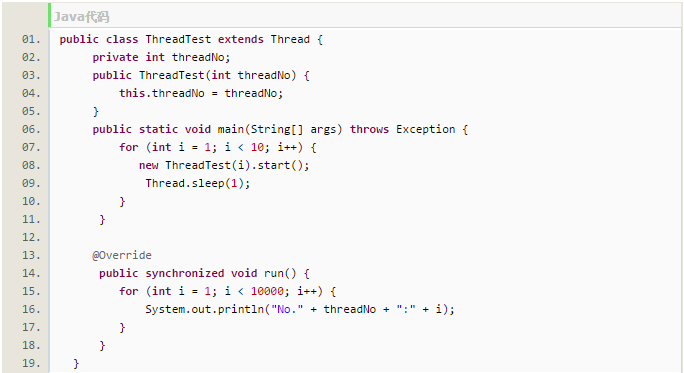
或者类变量不需锁。但是如果线程获取了锁，那么在它释放这个锁之前，就没有其他线程可以获取同样数据的锁了。（锁住一个对象就是获取对象相关联的监视器）

4、类锁实际上用对象锁来实现。

当虚拟机装载一个class文件的时候，它就会创建一个java.lang.Class类的实例。当锁住一个对象的时候，实际上锁住的是那个类的Class对象。  
一个线程可以多次对同一个对象上锁。对于每一个对象，java虚拟机维护一个加锁计数器，线程每获得一次该对象，计数器就加1，每释放一次，计数器就减 1，当计数器值为0时，锁就被完全释放了。java编程人员不需要自己动手加锁，对象锁是java虚拟机内部使用的。  
5、在java程序中，只需要使用synchronized块或者synchronized方法就可以标志一个监视区域。当每次进入一个监视区域时，java 虚拟机都会自动锁上对象或者类。  
重点：当一个有限的资源被多个线程共享的时候，为了保证对共享资源的互斥访问，我们一定要给他们排出一个先来后到：对象锁在这里起着非常重要的作用。多线程是如何处理共享资源的，以及保证他们对资源进行互斥访问所依赖的重要机制：对象锁。

**6、传统的同步实现方式以及这背后的原理**

（1）、Java多线程编程中，重要的关键字-->synchronized。“但是这么一个关键字，我也看不出来Java程序锁住了哪个对象阿？“ 示例如下：

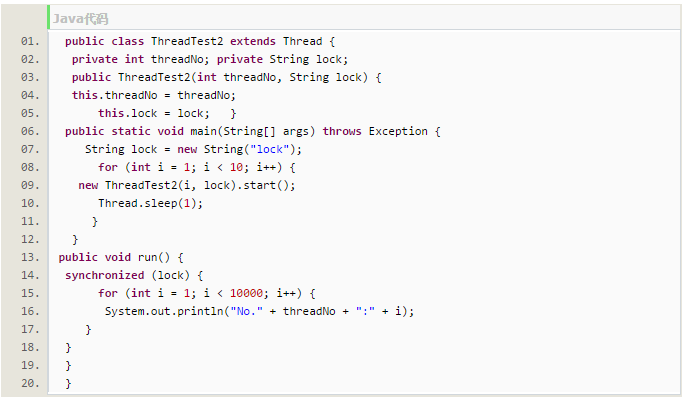


分析：

（1）、这个程序其实就是让10个线程在控制台上数数，从1数到9999。理想情况下，我们希望看到一个线程数完，然后才是另一个线程开始数数。

（2）、但是这个程序的执行过程告诉我们，这些线程还是乱糟糟的在那里抢着报数，丝毫没有任何规矩可言。  
     （3）、但是run方法还是加了一个synchronized关键字的，按道理说，这些线程应该可以一个接一个的执行这个run方法才对阿。  
     （4）、上面我们提到：对于一个成员方法加synchronized关键字，这实际上是以这个成员方法所在的对象本身作为对象锁。在本例中，就是以ThreadTest类的一个具体对象：该线程自身作为对象锁的。一共十个线程，每个线程持有自己线程对象的那个对象锁。这必然不能产生同步的效果。换句话说，**如果要对这些线程进行同步，那么这些线程所持有的对象锁应当是共享且唯一的!**

(2)、使用lock + synchronized实现多线程同步



分析：

  （1）、该程序通过在main方法启动10个线程之前，创建了一个String类型的对象。并通过ThreadTest2的构造函数，将这个对象赋值给每一个ThreadTest2线程对象中的私有变量lock。

（2）、根据Java方法的传值特点，这些线程的lock变量实际上指向的是堆内存中的同一个区域，即存放main函数中的lock变量的区域。  
       （3）、程序将原来run方法前的synchronized关键字去掉，换用了run方法中的一个synchronized块来实现。这个同步块的对象锁，就是 main方 法中创建的那个String对象。换句话说，他们指向的是同一个String类型的对象，对象锁是共享且唯一的！

（3）、使用静态方法实现多线程同步



分析：

（1）、这段代码没有使用main方法中创建的String对象作为这10个线程的线程锁。而是通过在run方法中调用本线程中一个静态的同步方法abc而实现了线程的同步。  
 （2）、synchronized静态方法是用什么来做对象锁的呢？ 对于同步静态方法，对象锁就是该静态放发所在的类的Class实例。由于在JVM中，所有被加载的类都有唯一的类对象，具体到本例就是唯一的ThreadTest3.class对象。不管我们创建了该类的多少实例，但是它的类实例仍然是一个！

（4）、总结

（1）、对于同步的方法或者代码块来说，必须获得对象锁才能够进入同步方法或者代码块进行操作；

（2）、如果采用method级别的同步，则对象锁即为method所在的对象；如果是静态方法，对象锁即指method所在的Class对象(唯一)；

（3）、对于代码块，对象锁即指synchronized(lock)中的lock；

（4）、第一种情况：对象锁即为每一个线程对象，因此有多个，所以同步失效；

第二种情况：共用同一个对象锁lock，因此同步生效；

第三种情况：因为是static，所以对象锁为ThreadTest3的class对象（唯一），因此同步生效。

（5）、同步有两种方式，同步块和同步方法（为什么没有wait和notify？这个我会在补充章节中做出阐述）  
 （5.1）、如果是同步代码块，则对象锁需要编程人员自己指定，一般有些代码为synchronized(this)只有在单态模式才生效；  
 （5.2）、如果是同步方法，**则分静态和非静态两种** 。  
 静态方法则一定会同步，非静态方法需在单例模式才生效，推荐用静态方法(不用担心是否单例)。  
 所以说，在Java多线程编程中，最常见的synchronized关键字实际上是依靠对象锁的机制来实现线程同步的。我们似乎可以听到synchronized在向我们说：“给我**一把** 锁，我能创造一个规矩”。

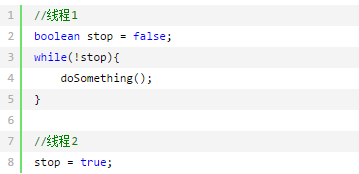
Volatile关键字： <http://www.cnblogs.com/dolphin0520/p/3920373.html>

1、一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，那么就具备了两层语义：

1）、保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。

2）、禁止进行指令重排序。

举例：



这段代码为何有可能导致无法中断线程：

前面提到每个线程在运行过程中都有自己的工作内存，那么线程1在运行的时候，会将stop变量的值拷贝一份放在自己的工作内存当中。那么当线程2更改了stop变量的值之后，但是还没来得及写入主存当中，线程2转去做其他事情了，那么线程1由于不知道线程2对stop变量的更改，因此还会一直循环下去。

但是用volatile修饰之后就变得不一样了：

1、保证了操作的可见性：

第一：使用volatile关键字会强制将修改的值立即写入主存；

第二：使用volatile关键字的话，当线程2进行修改时，会导致线程1的工作内存中缓存变量stop的缓存行无效

（反映到硬件层的话，就是CPU的L1或者L2缓存中对应的缓存行无效）；

第三：由于线程1的工作内存中缓存变量stop的缓存行无效，所以线程1再次读取变量stop的值时会去主存读取。

总结：

那么在线程2修改stop值时（当然这里包括2个操作，修改线程2工作内存中的值，然后将修改后的值写入内存），会使得线程1的工作内存中缓存变量stop的缓存行无效，然后线程1读取时，发现自己的缓存行无效，它会等待缓存行对应的主存地址被更新之后，然后去对应的主存读取最新的值。那么线程1读取到的就是最新的正确的值。

2、volatile也无法保证对变量的任何操作都是原子性的： 使用synchronized 或则lock都可以修正。

3、volatile 能在一定程度上保证有序性。

volatile关键字禁止指令重排序有两层意思：

1）当程序执行到volatile变量的读操作或者写操作时，在其前面的操作的更改肯定全部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行；

2）在进行指令优化时，不能将在对volatile变量访问的语句放在其后面执行，也不能把volatile变量后面的语句放到其前面执行。

4、volatile的应用场景：

　synchronized关键字是防止多个线程同时执行一段代码，那么就会很影响程序执行效率；

volatile关键字在某些情况下性能要优于synchronized，但是要注意volatile关键字是无法替代synchronized关键字的，因为volatile关键字无法保证操作的原子性。

通常来说，使用volatile必须具备以下2个条件：

　　 1）对变量的写操作不依赖于当前值 i++ 就不行

　　 2）该变量没有包含在具有其他变量的不变式中 i= j + 1 就不行

　　实际上，这些条件表明，可以被写入 volatile 变量的这些有效值独立于任何程序的状态，包括变量的当前状态。

　　事实上，我的理解就是上面的2个条件需要保证操作是原子性操作，才能保证使用volatile关键字的程序在并发时能够正确执行。

下面列举几个Java中使用volatile的几个场景。

**1）、状态标记量；2）double check**

**四、Java中的原始数据类型都有哪些，它们的大小及对应的封装类是什么？**

参考网址：http://www.cnblogs.com/BillLei/p/5568917.html

**（1）boolean --** 封装类是Boolean

   boolean数据类型非true即false。这个数据类型表示1 bit的信息，但是它的大小并没有精确定义；这样我们可以得出boolean类型单独使用是4个字节，在数组中又是1个字节。

那虚拟机为什么要用int来代替boolean呢？为什么不用byte或short，这样不是更节省内存空间吗？实际上，使用int的原因是，对于当下32位的CPU来说，一次进行32位的数据交换更加高效。

综上，我们可以知道：官方文档对boolean类型没有给出精确的定义，《Java虚拟机规范》给出了“单独时使用4个字节，boolean数组时1个字节”的定义，具体还要看虚拟机实现是否按照规范来，所以1个字节、4个字节都是有可能的。这其实是一种时空权衡。

（2）byte——1 byte——Byte

（3）char——2 bytes——Character

（4）short——2 bytes——Short

（5）int——4 bytes——Integer

（6）float——4 bytes——Float

（7）long——8 bytes——Long

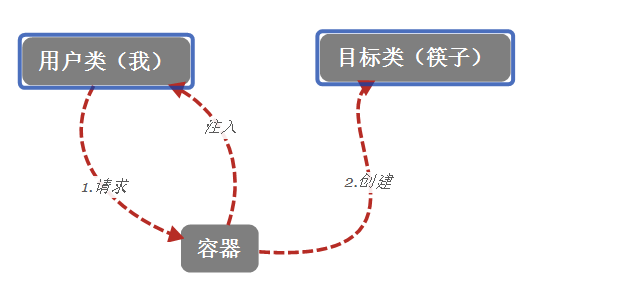
（8）double——8 bytes——Double

JAVA中SSM框架的准备

一、IOC：inverse of Control：控制反转。 <http://www.cnblogs.com/xxzhuang/p/5948902.html>

程序中的之间的关系，不用代码控制，而完全是由容器来控制。在运行阶段，容器会根据配置信息直接把他们的关系注入到组件中。

依赖注入和控制反转其实是一个概念。只不过强调的不同而已，依赖注入强调关系的注入是由容器在运行时完成，而控制反转强调关系是由容器控制。其实本质是一样的。



在控制反转中获取资源的过程叫做依赖注入，那么这里代码实现也是专注于依赖注入。

依赖注入有3种方式，分别为构造注入，设置注入，接口注入（spring不支持）。

事务(Transaction)是访问并可能更新数据库中各种数据项的一个程序执行单元(unit)。