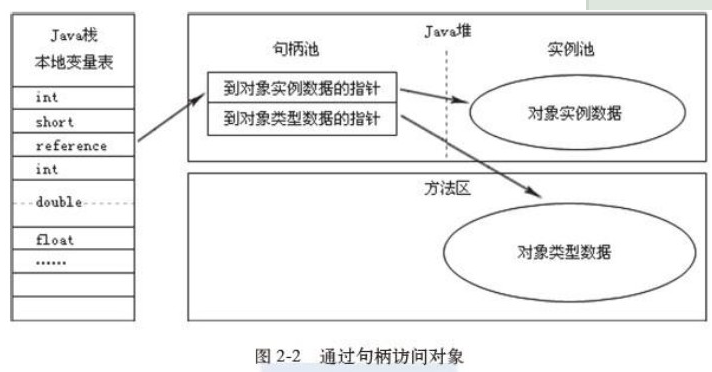
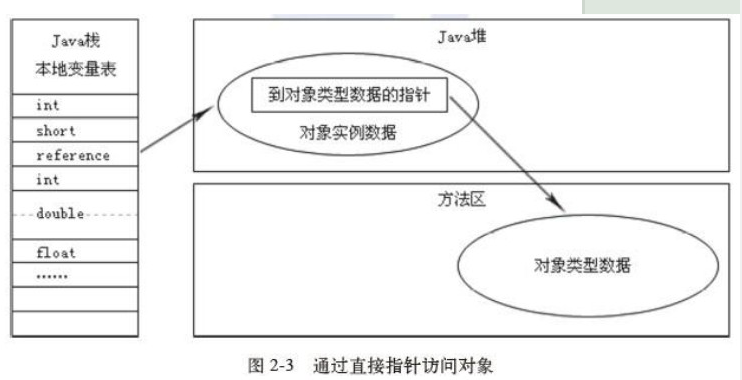
1. 面向对象
   1. 封装：通过合并特征和行为来创建新的数据类型，通过将细节私有化来隐藏实现过程。
   2. 继承：当创建一个派生类的对象时，该对象包含了一个基类的子对象。这个对象与你用基类直接创建的对象是一样的。二者的区别在于，后者来自于外部，而基类的子对象被包含在派生类的内部。JAVA会自动在派生类的构造器中插入对基类构造器的调用。Super可以用来显示地调用基类构造器。
   3. 多态：消除类型之间的耦合关系，允许将多种类型视为同一基类来进行处理。
      1. 向上转型upcasting：父类引用指向子类对象，Father f1 = new Son();
      2. 静态绑定：static、final方法
      3. 动态绑定
      4. 向下转型downcasting：指向子类对象的父类引用赋给子类引用叫向下转型，要强制转换。Son s1 = (Son)f1;
   4. 方法重载：每个方法都有独一无二的参数类型列表
   5. 方法重写（覆盖）：某方法是基类的一部分，子类继承后修改具体操作过程。
   6. this: 只能在方法内部使用，表示对“调用方法的那个对象”的引用
   7. static
      1. 变量：全局变量，当声明一个对象时，并不产生static变量的拷贝，而是该类所有的实例变量共用同一个static变量。静态变量与静态方法类似。所有此类实例共享此静态变量，也就是说在类装载时，只分配一块存储空间，所有此类的对象都可以操控此块存储空间。
      2. 方法：调用一个静态方法就是“类名.方法名”，仅能调用其他的static方法，只能访问static数据，不能以任何方式引用this或super。
      3. 类：通常一个普通类不允许声明为静态的，只有一个内部类才可以。这时这个声明为静态的内部类可以直接作为一个普通类来使用，而不需实例一个外部类。
   8. final
      1. 数据：对于基本类型，final使数值恒定不变；对于引用类型，final使引用恒定不变。一旦引用被初始化指向一个对象，就无法再把它改为指向另外一个对象。然而对象自身确是可以被修改的。Java未提供使任何对象恒定不变的途径。运行时才知道具体数值。
      2. 方法：把方法锁定，以防止任何继承类修改它的含义，确保在继承中是方法行为保持不变，并且不会被覆盖。类中所有的private方法都隐式的被指定为final的。
      3. 类：不希望被继承，类中的所有方法都被指定为final
   9. 接口：完全抽象的类，没有提供任何具体实现，同时实现多个接口，不管这些接口之间有没有关系，所以接口弥补了抽象类不能多重继承的缺陷
      1. Interface的方所有法访问权限自动被声明为public。确切的说只能为public，当然你可以显示的声明为protected、private，但是编译会出错！
      2. 接口中可以定义“成员变量”，或者说是不可变的常量，因为接口中的“成员变量”会自动变为为public static final。可以通过类命名直接访问：ImplementClass.name。
      3. 接口中不存在实现的方法。
      4. 实现接口的非抽象类必须要实现该接口的所有方法。抽象类可以不用实现。
      5. 不能使用new操作符实例化一个接口，但可以声明一个接口变量，该变量必须引用（refer to)一个实现该接口的类的对象。可以使用 instanceof 检查一个对象是否实现了某个特定的接口。例如：if(anObject instanceof Comparable){}。
      6. 在实现多接口的时候一定要避免方法名的重复。
   10. 抽象类：包含抽象方法的类，用来被继承
       1. 抽象类不能被实例化，实例化的工作应该交由它的子类来完成，它只需要有一个引用即可。
       2. 抽象方法必须由子类来进行重写。
       3. 只要包含一个抽象方法的抽象类，该方法必须要定义成抽象类，不管是否还包含有其他方法。
       4. 抽象类中可以包含具体的方法，当然也可以不包含抽象方法。
       5. 子类中的抽象方法不能与父类的抽象方法同名。
       6. abstract不能与final、private、static并列修饰同一个类。
   11. 初始化
       1. 初始化顺序
       2. 静态数据初始化：无论创建多少个对象，静态数据都只占用一份存储区域。新静态对象，再“非静态对象”。
       3. 静态块：首次生成对象时执行
       4. 实例初始化块：在构造器之前执行
   12. 访问权限控制：public、protected（子类、同一个包中的类）、default（同一个包的类）、private（只有类本身可以访问，不对外公开）。
2. 容器类
   1. Collection
      1. ArrayList，长于随机访问元素，插入和移除元素较慢
      2. LinkedList，长于插入和删除操作，随机访问效率较低
      3. Set，不包括重复元素
      4. Queue，用LinkedList实现
      5. 遍历方法
   2. Map
      1. HashMap：HashMap是非线程安全的，HashTable是线程安全的。HashMap的键和值都允许有null值存在，而HashTable则不行。因为线程安全的问题，HashMap效率比HashTable的要高。
      2. HashTable
      3. ConcurrentHashMap
      4. 遍历方法
3. 异常处理
   1. error表示系统级的错误，是java运行环境内部错误或者硬件问题，不能指望程序来处理这样的问题，除了退出运行外别无选择，它是Java虚拟机抛出的。exception 表示程序需要捕捉、需要处理的异常，是由与程序设计的不完善而出现的问题，程序必须处理的问题
   2. Java提供了两类主要的异常：runtimeException和checkedException
      1. 一般异常（checkedException）主要是指IO异常、SQL异常等。对于这种异常，JVM要求我们必须对其进行cathc处理，所以，面对这种异常，不管我们是否愿意，都是要写一大堆的catch块去处理可能出现的异常。
      2. 运行时异常（runtimeException）我们一般不处理，当出现这类异常的时候程序会由虚拟机接管。比如，我们从来没有去处理过NullPointerException，而且这个异常还是最常见的异常之一。出现运行时异常的时候，程序会将异常一直向上抛，一直抛到遇到处理代码，如果没有catch块进行处理，到了最上层，如果是多线程就有Thread.run()抛出，如果不是多线程那么就由main.run()抛出。抛出之后，如果是线程，那么该线程也就终止了，如果是主程序，那么该程序也就终止了。
   3. 在Java中，每个异常都是一个对象，它都是Throwable或其子类的实例。当一个方法出现异常后就会抛出一个异常对象，该对象中包含有异常信息，调用这个对象的方法可以捕获到这个异常并对异常进行处理。Java的异常处理是通过5个关键词来实现的：try catch throw throws finally。
   4. 尽量避免出现runtimeException 。例如对于可能出现空指针的代码，带使用对象之前一定要判断一下该对象是否为空，必要的时候对runtimeException也进行try catch处理。进行try catch处理的时候要在catch代码块中对异常信息进行记录，通过调用异常类的相关方法获取到异常的相关信息，返回到web端，不仅要给用户良好的用户体验，也要能帮助程序员良好的定位异常出现的位置及原因。例如，以前做的一个项目，程序遇到异常页面会显示一个图片告诉用户哪些操作导致程序出现了什么异常，同时图片上有一个按钮用来点击展示异常的详细信息给程序员看的。
4. 字符串
   1. String对象是不可变的，每一个看起来修改String值的方法，实际上都是创建一个全新的string对象，以包含修改后的字符串内容
   2. StringBuilder：修改字符串的内容，而不创建新的对象。
5. I/O 文件处理
   1. 字符流：在程序中一个字符等于两个字节，那么java提供了Reader、Writer两个专门操作字符流的类。
   2. 字节流：字节流主要是操作byte类型数据，以byte数组为准，主要操作类就是OutputStream、InputStream。
   3. 序列化：Java对象的序列化指将一个java对象写入OI流中，与此对应的是，对象的反序列化则从IO流中恢复该java对象。
6. 内存管理及垃圾回收
   1. 运行时数据区域
      1. 程序计数器：是一块较小的内存空间，它的作用可以看做是当前线程所执行的字节码的行号指示器。“线程私有”的内存。
      2. Java虚拟机栈：方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。
      3. 本地方法栈：本地方法栈（Native Method Stacks）与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行Java方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的Native方法服务。
      4. Java堆：Java堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。
      5. 方法区：方法区（Method Area）与Java堆一样，是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
      6. 运行时常量池：运行时常量池（Runtime Constant Pool）是方法区的一部分。用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后存放到方法区的运行时常量池中。
   2. 对象访问的过程　Object obj = new Object();
      1. 句柄访问方式：java堆中将划分出一块内存来作为句柄池，reference中存储的就是对象的句柄地址，而句柄中包含了对象实例数据和类型数据各自的具体地址信息。最大好处就是reference中存储的是稳定的句柄地址，在对象被移动（垃圾收集时移动对象是非常普遍的行为）时只会改变句柄中的实例数据指针，而reference本身不需要被修改。

* + 1. 指针访问方式：reference变量中直接存储的就是对象的地址，而java堆对象一部分存储了对象实例数据，另外一部分存储了对象类型数据。最大好处就是速度更快，它节省了一次指针定位的时间开销，由于对象的访问在Java中非常频繁，因此这类开销积少成多后也是一项非常可观的执行成本。
  1. 垃圾回收
     1. 引用计数法：在这种方法中，堆中每个对象实例都有一个引用计数。当一个对象被创建时，且将该对象实例分配给一个变量，该变量计数设置为1。当任何其它变量被赋值为这个对象的引用时，计数加1（a = b,则b引用的对象实例的计数器+1），但当一个对象实例的某个引用超过了生命周期或者被设置为一个新值时，对象实例的引用计数器减1。任何引用计数器为0的对象实例可以被当作垃圾收集。当一个对象实例被垃圾收集时，它引用的任何对象实例的引用计数器减1。
     2. 标记-清除算法：根搜索算法是从离散数学中的图论引入的，程序把所有的引用关系看作一张图，从一个节点GC ROOT开始，寻找对应的引用节点，找到这个节点以后，继续寻找这个节点的引用节点，当所有的引用节点寻找完毕之后，剩余的节点则被认为是没有被引用到的节点，即无用的节点。
     3. 当一个实例被创建了，首先会被存储在堆内存年轻代的Eden区中。作为年轻代 GC（Minor GC）周期的一部分，存活的对象（仍然被引用的）从 Eden 区被移动到Survivor区的S0中。类似的，垃圾回收器会扫描S0然后将存活的实例移动到S1中。当垃圾回收器执行 Minor GC 周期时，在 S1 Survivor 区中的存活实例将会被晋升到老年代，而未被引用的对象被标记为回收。相对于 Java 垃圾回收过程，老年代是实例生命周期的最后阶段。Major GC 扫描老年代的垃圾回收过程。如果实例不再被引用，那么它们会被标记为回收，否则它们会继续留在老年代中。
     4. 一旦实例从堆内存中被删除，其位置就会变空并且可用于未来实例的分配。这些空出的空间将会使整个内存区域碎片化。为了实例的快速分配，需要进行碎片整理。基于垃圾回收器的不同选择，回收的内存区域要么被不停地被整理，要么在一个单独的GC进程中完成。
     5. 串行垃圾回收器：冻结其他线程，使用一个单独的线程进行垃圾回收；
     6. 并行垃圾回收器：JVM默认的垃圾回收器，同时开启多个垃圾回收线程；
     7. 并发标记扫描垃圾回收器：可以与一般的线程并行执行；
     8. G1垃圾回收器：将堆内存分割成不同的区域，并且并发的对其进行垃圾回收。G1也可以在回收内存之后对剩余的堆内存空间进行压缩。

1. 并发编程
   1. Java中有几种方法可以实现一个线程？
      1. ThreadTest extends Thread，重写run方法，之后调用start方法；一个线程对象只能启动一个线程，无论你调用多少遍start()方法，结果只有 一个线程。
      2. ThreadTest implements Runnable，new Thread(t).start()。创建一个资源对象，但要创建多个线程去处理同一个资源对象，并且每个线程上所运行的是相同的程序代码。
      3. Runnable接口相对于继承Thread类来说，有如下显著的好处：适合多个相同程序代码的线程去处理同一资源的情况，把虚拟CPU（线程）同程序的代码，数据有效的分离，较好地体现了面向对象的设计思想。可以避免由于Java的单继承特性带来的局限。
   2. notify()和notifyAll()都是Object对象用于通知处在等待该对象的线程的方法。
      1. notifyAll使所有原来在该对象上等待被notify的线程统统退出wait的状态，变成等待该对象上的锁，一旦该对象被解锁，他们就会去竞争。
      2. notify则文明得多他只是选择一个wait状态线程进行通知，并使它获得该对象上的锁，但不惊动其他同样在等待被该对象notify的线程们
      3. 当第一个线程运行完毕以后释放对象上的锁此时如果该对象没有再次使用notify语句，则即便该对象已经空闲，其他wait状态等待的线程由于没有得到该对象的通知，继续处在wait状态，直到这个对象发出一个notify或notifyAll，它们等待的是被notify或notifyAll，而不是锁。
   3. sleep()和 wait()有什么区别?
      1. sleep()使当前线程进入停滞状态（阻塞当前线程），让出CUP的使用、目的是不让当前线程独自霸占该进程所获的CPU资源，以留一定时间给其他线程执行的机会；sleep()是Thread类的Static(静态)的方法；因此他不能改变对象的机锁，所以当在一个Synchronized块中调用Sleep()方法是，线程虽然休眠了，但是对象的机锁并木有被释放，其他线程无法访问这个对象（即使睡着也持有对象锁）。在sleep()休眠时间期满后，该线程不一定会立即执行，这是因为其它线程可能正在运行而且没有被调度为放弃执行，除非此线程具有更高的优先级。
      2. wait()方法是Object类里的方法；当一个线程执行到wait()方法时，它就进入到一个和该对象相关的等待池中，同时失去（释放）了对象的机锁（暂时失去机锁，wait(long timeout)超时时间到后还需要返还对象锁）；其他线程可以访问；wait()使用notify或者notifyAlll或者指定睡眠时间来唤醒当前等待池中的线程。wiat()必须放在synchronized block中，否则会在program runtime时扔出”java.lang.IllegalMonitorStateException“异常。
   4. synchronized：Java语言的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。
      1. 修饰一个代码块
      2. 修饰一个方法
      3. 修饰一个静态方法
      4. 修饰一个类