1.关于 JVM 运行时候的内存区域

大多数 JVM 将内存区域划分为 Method Area(方法区)，Heap(堆)，Program Counter Register(程序计数器)，VM Stack(虚拟机栈,也有翻译成 JAVA 方法栈的)，Native Method Stack(本地方法栈)，其中 Method Area 和 Heap 是线程共享的，其余三个是非线程共享的。

一般的 Java 程序的工作过程：一个 Java 源程序文件，会被编译为字节码文件（以 class 为扩展名），每个 java 程序都需要运行在自己的 JVM 上，然后告知 JVM 程序的运行入口，再被 JVM 通过字节码解释器加载运行。

JVM 初试运行的时候都会分配好 Method Area(方法区)和 Heap(堆)，而 JVM 每遇到一个线程，就为其分配一个 Program Counter Register(程序计数器),VM Stack(虚拟机栈,也有翻译成 JAVA 方法栈的)，Native Method Stack(本地方法栈),当线程终止时，三者所占用的内存空间会被释放掉。这也是为什么我把内存区域分为线程共享和非线程共享的原因，非线程共享的那三个区域的生命周期与所属线程相同，而线程共享的区域与 JAVA 程序运行的生命周期相同，所以这也是系统垃圾回收的场所只发生在线程共享的区域（实际上对大部分虚拟机来说知发生在 Heap 上）的原因。

程序计数器

程序计数器是一块较小的内存区域，作用是看作是当前线程执行的字节码的位置指示器。分支、循环、跳转、异常处理和线程回复等基础功能都要依赖这个计算器来完成

2）VM Stack

先来了解下 JAVA 指令的构成：

JAVA 指令由 操作码 （方法本身）和 操作数 （方法内部变量） 组成。

1）方法本身是指令的 操作码 部分，保存在 Stack 中；

2）方法内部变量（局部变量）作为指令的 操作数 部分，跟在指令的操作码之后，保存在 Stack 中（实际上是简单类型（int,byte,short 等）保存在 Stack 中，对象类型在 Stack 中保存地址，在 Heap 中保存值）；

虚拟机栈也叫栈内存，是在线程创建时创建，它的生命期是跟随线程的生命 期，线程结束栈内存也就释放， 对于栈来说不存在垃圾回收问题，只要线程一结束，该栈就 Over，所以不存在垃圾回收 。 也有一些资料翻译成 JAVA 方法栈，大概是因为它所描述的是 java 方法执行的内存模型，每个方法执行的同时创建帧栈（Strack Frame）用于存储局部变量表（包含了对应的方法参数和局部变量），操作栈（Operand Stack，记录出栈、入栈的操作），动态链接、方法出口等信息，每个方法被调用直到执行完毕的过程，对应这帧栈在虚拟机栈的入栈和出栈的过程。

局部变量表存放了编译期可知的各种基本数据类型（boolean、byte、char、short、int、float、long、double）、对象的引用（reference 类型，不等同于对象本身，根据不同的虚拟机实现，可能是一个指向对象起始地址的引用指针，也可能是一个代表对象的句柄或者其他与对象相关的位置）和 returnAdress 类型（指向下一条字节码指令的地址）。局部变量表所需的内存空间在编译期间完成分配，在方法在运行之前，该局部变量表所需要的内存空间是固定的，运行期间也不会改变。栈帧是一个内存区块，是一个数据集， 是 一个有关方法(Method)和运行期数据的数据集 ， 当一个方法 A 被调用时就产生了一个栈帧 F1，并 被压入到栈中，

A 方法又调用了 B 方法，于是产生栈帧 F2 也被压入栈，执行完毕后，先弹出 F2 栈帧，再弹出 F1 栈帧，遵循“ 先进后出 ”原则。 光说比较枯燥，我们看一个图来理解一下 Java 栈，如下图所示：

3.Heap

Heap（堆）是 JVM 的内存数据区。Heap 的管理很复杂，是被所有线程共享的内存区域，在 JVM 启动时候创建，专门用来保存对象的实例。在 Heap 中分配一定的内存来保存对象实例，实际上也只是保存对象实例的属性值，属性的类型和对象本身的类型标记等，并不保存对象的方法（以帧栈的形式保存在 Stack 中），在 Heap 中分配一定的内存保存对象实例。而对象实例在 Heap 中分配好以后，需要在 Stack 中保存一个 4 字节的 Heap 内存地址，用来定位该对象实例在 Heap 中的位置，便于找到该对象实例，是垃圾回收的主要场所。java 堆处于物理不连续的内存空间中，只要逻辑上连续即可。

4.Method Area

Object Class Data(加载类的类定义数据) 是存储在方法区的。除此之外， 常量 、静态变量 、JIT(即时编译器)编译后的代码也都在方法区。正因为方法区所存储的数据与堆有一种类比关系，所以它还被称为 Non-Heap。方法区也可以是内存不连续的区域组成的，并且可设置为固定大小，也可以设置为可扩展的，这点与堆一样。

垃圾回收在这个区域会比较少出现，这个区域内存回收的目的主要针对常量池的回收和类的卸载

5.运行时常量池（Runtime Constant Pool）

方法区内部有一个非常重要的区域，叫做 运行时常量池（Runtime Constant Pool，简称 RCP） 。在字节码文件（Class 文件）中，除了有类的版本、字段、方法、接口等先关信息描述外，还有常量池（Constant Pool Table）信息，用于存储编译器产生的字面量和符号引用。这部分内容在类被加载后，都会存储到方法区中的 RCP。值得注意的是，运行时产生的新常量也可以被放入常量池中，比如 String 类中的 intern() 方法产生的常量。

常量池就是这个类型用到的常量的一个有序集合。包括 直接常量(基本类型，String) 和 对其他类型、方法、字段的符号引用.例如：

◆类和接口的全限定名；

◆字段的名称和描述符；

◆方法和名称和描述符。

池中的数据和数组一样通过索引访问。由于常量池包含了一个类型所有的对其他类型、方法、字段的符号引用，所以常量池在 Java 的动态链接中起了核心作用.

6.Native Method Stack

与 VM Strack 相似，VM Strack 为 JVM 提供执行 JAVA 方法的服务，Native Method Stack 则为 JVM 提供使用 native 方法的服务。

7.直接内存区

直接内存区并不是 JVM 管理的内存区域的一部分，而是其之外的。该区域也会在 Java 开发中使用到，并且存在导致内存溢出的隐患。如果你对 NIO 有所了解，可能会知道 NIO 是可以使用 Native Methods 来使用直接内存区的。

看 JVM 是如何运行的，也就是输入 java JVMShow 后，我们来看 JVM 是如何处理的：

第 1 步 、向操作系统申请空闲内存。JVM 对操作系统说“给我 64M（随便模

拟数据，并不是真实数据） 空闲内存”，于是，JVM 向操作系统申请空闲内存作系统就查找自己的内存分配表，找了段 64M 的内存写上“Java 占用”标签，然后把内存段的起始地址和终止地址给 JVM，JVM 准备加载类文件。
第 2 步，JVM 分配内存。JVM 获得到 64M 内存，就开始得瑟了，首先给 heap 分个内存，然后给栈内存也分配好。
第 3 步， 文件检查和分析 class 文件。 若发现有错误即返回错误。
第 4 步， 加载类。 由于没有指定加载器，JVM 默认使用 bootstrap 加载器，就把 rt.jar 下的所有类都加载到了堆类存的 Method Area，JVMShow 也被加载到内存中。我们来看看 Method Area 区域，如下图：（ 这时候包含了 main 方法和 runStaticMethod 方法的符号引用，因为它们都是静态方法，在类加载的时候就会加载 ）
Heap 是空，Stack 是空，因为还没有对象的新建和线程被执行。
第 5 步、执行方法。执行 main 方法。执行启动一个线程，开始执行 main 方

法，在 main 执行完毕前，方法区如下图所示：
为什么会有 Object 对象呢？是因为它是 JVMShowcase 的父类， JVM 是先初始化父类 ，然后再初始化子类，甭管有多少个父类都初始化。
于此同时，还创建了一个程序计数器指向下一条要执行的语句。
第 6 步， 释放内存。释放内存。运行结束，JVM 向操作系统发送消息，说“内存用完了，我还给你”，运行结束。
预备知识：
1. 一个 Java 文件，只要有 main 入口方法，我们就认为这是一个 Java 程序，可以单独编译运行。
2. 无论是普通类型的变量还是引用类型的变量(俗称实例)，都可以作为局部变量，他们都可以出现在栈中。只不过普通类型的变量在栈中直接保存它所对应的值，而引用类型的变量保存的是一个指向堆区的指针，通过这个指针，就可以找到这个实例在堆区对应的对象。因此，普通类型变量只在栈区占用一块内存，而引用类型变量要在栈区和堆区各占一块内存。
小结：
1. 分清什么是实例什么是对象。Class a= new Class();此时 a 叫实例，而不能说 a 是对象。实例在栈中，对象在堆中，操作实例实际上是通过实例的指针间接操作对象。多个实例可以指向同一个对象。
2. 栈中的数据和堆中的数据销毁并不是同步的。方法一旦结束，栈中的局部变量立即销毁，但是堆中对象不一定销毁。因为可能有其他变量也指向了这个对象，直到栈中没有变量指向堆中的对象时，它才销毁，而且还不是马上销毁，要等垃圾回收扫描时才可以被销毁。
3. 以上的栈、堆、代码段、数据段等等都是相对于应用程序而言的。每一个应用程序都对应唯一的一个 JVM 实例，每一个 JVM 实例都有自己的内存区域，互不影响。并且这些内存区域是所有线程共享的。这里提到的栈和堆都是整体上的概念，这些堆栈还可以细分。
4. 类的成员变量在不同对象中各不相同，都有自己的存储空间(成员变量在堆中的对象中)。而类的方法却是该类的所有对象共享的，只有一套，对象使用方法的时候方法才被压入栈，方法不使用则不占用内存。

2. 有关 jdbc statement 的

问题一：Statement 和 PreparedStatement 的区别
答：Statement 是 java 执行数据库操作的一个重要方法，用于在已经建立数据库连接的基础上，向数据库发送要执行的 SQL 语句。

Java 提供了 Statement、PreparedStatement 和 CallableStatement 三种方法来进行查询语句，其中 Statement 用于通用查询，PreparedStatement 用于执行参数化查询，而 CallableStatement 则是用于存储过程。

Statement 每次执行 sql 语句，数据库都要执行 sql 语句的编译，最好用于仅执行一次查询并返回结果的情形，效率要高于 PreparedStatement，但存在 sql 注入风险。PreparedStatement 是预编译执行的。在执行可变参数的一条 SQL 时，PreparedStatement 要比 Statement 的效率高，因为 DBMS 预编译一条 SQL 当然会比多次编译一条 SQL 的效率高，安全性更好，可以有效的防止 SQL 注入的问题，对于多次执行的语句，使用 PreparedStatement 效率会更高一点，执行 SQL 语句是可以带参数的，并且支持批量执行 SQL。由于采用了 Cache 机制，则预编译的语句就会放在 Cache 中，下次执行相同的 SQL 语句时，则可以直接从 Cache 中取出来。

PreparedStatement:数据库会对 SQL 语句进行预编译，下次执行相同的 sql 语句时，数据库段不会再进行预编译了，而直接用数据库的缓冲区，提高数据访问的效率，从安全的性能上看，PreparedStatement 是通过通配符“？”来传递参数的，避免了拼接 sql 而出现的 sql 注入的问题，开发中，推荐使用 preparedStatement

问题二：什么是 SQL 注入，怎么防止 SQL 注入？
答：所谓的 SQL 注入，就是通过把 SQL 命令插入到 WEB 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。具体来说，它是利用现有应用程序，将（恶意）的 SQL 命令注入到后台数据库引擎执行的能力，它可以通过在 WEB 表单中输入（恶意）SQL 语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

热么防止 SQL 注入呢？使用存储过程执行所有的查询；检查用户输入的合法性；将用户的登录名、密码等数据加密保存

关于 PreparedStatement 接口，需要重点记住的是：
1. PreparedStatement 可以写参数化查询，比 Statement 能获得更好的性能。
2. 对于 PreparedStatement 来说，数据库可以使用已经编译过及定义好的执行计划，这种预处理语句查询比普通的查询运行速度更快。
3. PreparedStatement 可以阻止常见的 SQL 注入式攻击。
4. PreparedStatement 可以写动态查询语句
5. PreparedStatement 与 java.sql.Connection 对象是关联的，一旦你关闭了 connection，PreparedStatement 也没法使用了。
6. “?” 叫做占位符。为了防止 SQL 注入，PreparedStatement 不允许占位符？有多个值
7. PreparedStatement 查询默认返回 FORWARD_ONLY 的 ResultSet，你只能往一个方向移动结果集的游标。当然你还可以设定为其他类型的值如：”CONCUR_READ_ONLY”。
8. 不支持预编译 SQL 查询的 JDBC 驱动，在调用 connection.prepareStatement(sql)

的时候，它不会把 SQL 查询语句发送给数据库做预处理，而是等到执行查询动作的时候（调用 executeQuery()方法时）才把查询语句发送个数据库，这种情况和使用 Statement 是一样的。
9. 占位符的索引位置从 1 开始而不是 0，如果填入 0 会导致*java.sql.SQLException invalid column index*异常。所以如果 PreparedStatement 有两个占位符，那么第一个参数的索引时 1，第二个参数的索引是 2.
且：statement 是不穿如参数的，PreparedStatement 是要传入 sql 语句的，通过 PreparedStatement 方式可以防止 sql 注入

3. Spring 事务的传播特性

Spring 中通过 Propagation 来设置事务的传播特性的
1）PROPAGATION_REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务，这是最常见的选择
2）PROPAGATION_SUPPORTS:：支持当前事务，如果当前没有事务，就以非事务方式执行
3）PROPAGATION_MANDATORY：支持当前事务，如果当前没有事务，就抛出异常
4）PROPAGATION_RQUEIRES_NEW：新建事务，如果当前存在事务，把当前事务挂起
5）PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起
6）PROPAGATION_NEVER：以非实物方式执行，如果当前存在事务，就抛出异常
7）PROPAGATION_NESTED：支持当前事务，新增 savepoint 点，与当前事务同步提交或回滚。

4. Servlet 和 CGI 的区别

1）Servlet 被服务器实例化后，容器运行其 init 方法，请求到达时运行其 service 方法，service 方法自动派遣运行与请求对应的 doXXX 方法（doGet 和 doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。
2）Servlet 处于服务器进程中，它通过多线程方式运行其 service 方法，一个实例可以服务于多个请求，并且其实例不会销毁，而 CGI 对每个请求都产生心的进程，服务完成后就销毁，效率上低于 servlet。
3）CGI 不可移植，为某以特定平台编写的 CGI 应用只能运行于这一环境中，每一个 CGI 应用存在于一个由客户端请求激活的进程中，并且在请求被服务后被卸载，这种模式引起很高的内存、CPU 开销；Servlet 提供了 Java 应用程序的所有优势，可移植性、稳健、易开发。使用 Servlet Tag 技术，Servlet 能够生成嵌入于静态 HTML 页面中的动态内容。
4）Servlet 对 CGI 的最主要优势在于一个 Servlet 被客户端发送的第一个请求激活，然后它将继续运行于后台，等待以后的请求。每个请求将生成一个新的线程，而不是一个完整的进程。多个客户能够在一个进程中同时得到服务。一般来说，Servlet 进程知识在 WEB Server 卸载事被卸载

5. 进程和线程的关系及区别

1）定义：

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源分配和调度的一个独立单位

线程是进程的一个实体，是 CPU 调度和分派的基本单位，它比进程更小的能独立运行的基本单位，线程自己基本上不拥有系统资源，只拥有一点在运行中必不可少的资源（如程序计数器），但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源

2）关系：

一个线程可以创建和撤销零一个线程，同一个进程中的多个线程之间可以并发执行

相对进程而言，线程是一个更加接近于执行体的概念，它可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列

3）区别：

进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。

1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

4.优缺点

线程和进程在使用上各有优缺点：线程执行开销小，但不利于资源的管理和保护；而进程正相反。同时，线程适合于在 SMP 机器上运行，而进程则可以跨机器迁移

6. 关于 Servlet 中的方法

1）Servlet 使用一个 HTML 表格来发送和接受数据。要创建一个 Servlet 就要扩展 HttpServlet 类，这个类是用专门方法来处理 HTML 表格的 GenericSerlet 的一个子类，当表单信息被提交，会指定服务器应执行哪一个 Servlet。其中包含 init、destroy 和 service 方法，其中 Init 和 destroy 是继承的

2）Service 方法

每当一个客户请求一个 HttpServlet 对象，该对象的 service 方法就要被调用，而

且传递给这个方法一个请求（ServletRequest）和一个响应（ServletResponse）对象作为参数
3）service()是在 java.servlet.Servlet 接口中定义的，在 javax.servlet.GenericServlet 中实现的；doGet 和 doPost 则是在 java.servlet.http.HttpServlet 中实现的。
4）当一个客户通过 HTML 表单发出一个 HTTP POST 请求时，doPost()方法被调用。与 POST 请求相关的参数作为一个单独的 HTTP 请求从浏览器发送到服务器。当需要修改服务器端的数据时，应该使用 doPost()方法。
当一个客户通过 HTML 表单发出一个 HTTP GET 请求或直接请求一个 URL 时，doGet()方法被调用。与 GET 请求相关的参数添加到 URL 的后面，并与这个请求一起发送。当不会修改服务器端的数据时，应该使用 doGet()方法。

7. 关于 Servlet 的生命周期

Servlet 的生命周期分为三个阶段
1）初始化阶段　调用 Init()方法，在 Servlet 生命周期中，仅执行一次 init()方法。它是在服务器装入 servlet 时执行的，负责初始化 Servlet 对象。可以配置服务器，可以在启动服务器或客户端机首次访问 Servlet 时装入 Servlet，无论多少个客户端访问 Servlet，都不会重复执行 init()
2）响应客户请求阶段　调用 service()方法，负责响应客户的请求。每当一个客户请求一个 HttpServlet 对象，该对象的 Service()方法就要调用，而且传递给这个方法一个"请求"对象和一个响应对象作为参数。
3）终止阶段　调用 destroy()方法，仅执行一次，在服务器端停止且卸载 Servlet 时执行该方法，当 Servlet 对象推出生命周期时，负责释放占用的资源。一个 Servlet 在运行 Service()方法时可能会产生其他的线程，因此需要确认在调用 destroy()方法时，这些线程已经终止或完成。
4）创建 Servlet 实例是在初始化方法 init()之前，servlet 里的实例变量，是被所有线程共享的，所以不是线程安全的

8. 关于 Strus1 和 Struts2 的区别

从 action 类上分析:
1.Struts1 要求 Action 类继承一个抽象基类。Struts1 的一个普遍问题是使用抽象类编程而不是接口。
2. Struts 2 Action 类可以实现一个 Action 接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2 提供一个 ActionSupport 基类去实现常用的接口。Action 接口不是必须的，任何有 execute 标识的 POJO 对象都可以用作 Struts2 的 Action 对象。
从 Servlet 依赖分析:
3. Struts1 Action 依赖于 Servlet API ,因为当一个 Action 被调用时 HttpServletRequest 和 HttpServletResponse 被传递给 execute 方法。
4. Struts 2 Action 不依赖于容器，允许 Action 脱离容器单独被测试。如果需要，Struts2 Action 仍然可以访问初始的 request 和 response。但是，其他的元素减少或者消除了直接访问 HttpServetRequest 和 HttpServletResponse 的必要性。
从 action 线程模式分析:
5. Struts1 Action 是单例模式并且必须是线程安全的，因为仅有 Action 的一个实例来处理所有的请求。单例策略限制了 Struts1 Action 能作的事，并且要在开发时特别小心。Action 资

源必须是线程安全的或同步的。
6. Struts2 Action 对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet 容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）

9. AWT 和 Swing 的区别

AWT 是基于本地方法的 C/C++程序，其运行速度比较快，；Swing 是基于 AWT 的 Java 程序，其运行速度比较慢

10.关于 forword 和 redirect 的区别

forward，服务器获取跳转页面内容传给用户，用户地址栏不变

redirect，是服务器向用户发送转向的地址，redirect 后地址栏变成新的地址

11.关于 java 中 sleep 和 wait 的区别

1）sleep 来自 Thread 类，wait 来自 object 类。sleep 是 Thread 的静态类方法，谁调用睡去睡觉
2）关于锁：最主要的是 sleep 方法没有释放掉锁，wait 方法释放掉了锁，使得其他线程可以使用同步控制块或方法；sleep 不让出系统资源，导致线程暂停执行指定事件，把执行机会给其他线程，但是监控状态依然保持；wait 是进入线程等待池等待，让出系统资源，其他线程可以占用 CPU，进入等待锁定池，只有针对此对象的 notify 方法或者 notifyAll 后本线程才进入对象锁定池准备获得对象锁进入运行状态。

12. String，StringBuffer 和 StringBuilder 的区别

1）可变与不可变
String 类中使用字符数组保存字符串，因为有 final 修饰符，所以可以知道 string 对象是不可变的。
StringBuilder 与 StringBuffer 都继承自 AbstractStringBuilder 类，在 AbstractStringBuilder 中也是使用字符数组保存字符串，可知这两种对象都是可变的。
2）是否多线程安全
String 中的对象是不可变的，也就可以理解为常量，显然线程安全；StringBuffer 对方法加了同步锁或者对调用的方法加了同步锁，所以是线程安全的；StringBuilder 并没有对方法进行加同步锁，所以是非线程安全的。

3）StringBuilder 和 StringBuffer 的共同点

StringBuilder 与 StringBuffer 有公共父类 AbstractStringBuilder(抽象类)。如果程序不是多线程的，那么使用 StringBuilder 效率高于 StringBuffer。

13.内存溢出和内存泄露的区别

内存溢出 out of memory，是指程序在申请内存时，没有足够的内存空间供其使用，出现 out of memory；比如申请了一个 Integer，但给他存了 long 才能存下的数，那就是内存溢出。

内存泄露 memory leak，是指程序在申请内存后，无法释放已申请的内存空间，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。

14.JSP 页面的动态 include 和静态 include

1）动态 include 用 jsp:include 动作实现，如<jsp:include page="included.jsp" flush="true">它总是会检查所含文件中的变化，适合用于包含动态页面，并且可以带参数。各个文件分别先编译，然后组合成一个文件。
2）静态 Include 用 include 伪码实现，它不会检查所含文件的变化，适用于包含静态页面<%@include file="included.html"%>。先将文件的代码被原封不动的加入到了主页面从而合成一个文件，然后再进行翻译，此时不允许有相同的变量
3）不同点：
一、执行时间
<%@include file = "relativeURL"%>是在翻译阶段执行
<jsp:include page="relativeURL" flush="true"/>在请求处理阶段执行
二、引入内容不同
<%@include file="relativbeURL"%>引入静态文本<html,jsp>在 JSP 页面被转化成 Servlet 之前和它融合到一起，<jsp:include page="re" flush="true">引入执行页面或 servlet 所声称的应答文本。且静态 include 不允许有变量名相同

15.Java 虚拟机的参数

常规配置
1.堆设置
-Xms:初试堆大小
-Xmx:最大堆大小
-XX:NewSize=n：设置年轻代大小
-XX:NewRatio=n：设置年轻代和年老代的比值。如为 3，表示年轻代与年老代比值为 1:3，年轻代占整个年轻代年老代和的 1/4。
-XX:SurvivorRadio=n：年清代中 Eden 区与两个 Survivor 区的比值，注意 Survivor 区有两个。如：3，表示 Eden：Survivor=3：2，一个 Survivor 区占整个年轻代的 1/5
-XX:MaxPermSize=n,设置持久代大小
2.收集器设置：
-XX：+UseSeriGC：设置串行收集器
-XX：+UserParallelGC:设置并行收集器
-XX：+UseParalledOldGC:设置并行年老收集器
-XX：+UserConcMarkSweepGC:设置并发收集器
3.并行收集器设置
-XX：ParaelGCThreads=n:设置并行收集器收集时使用的 CPU 数。并行收集线程数
-XX：MaxGCPauseMiles=n:设置并行手机最大暂停时间

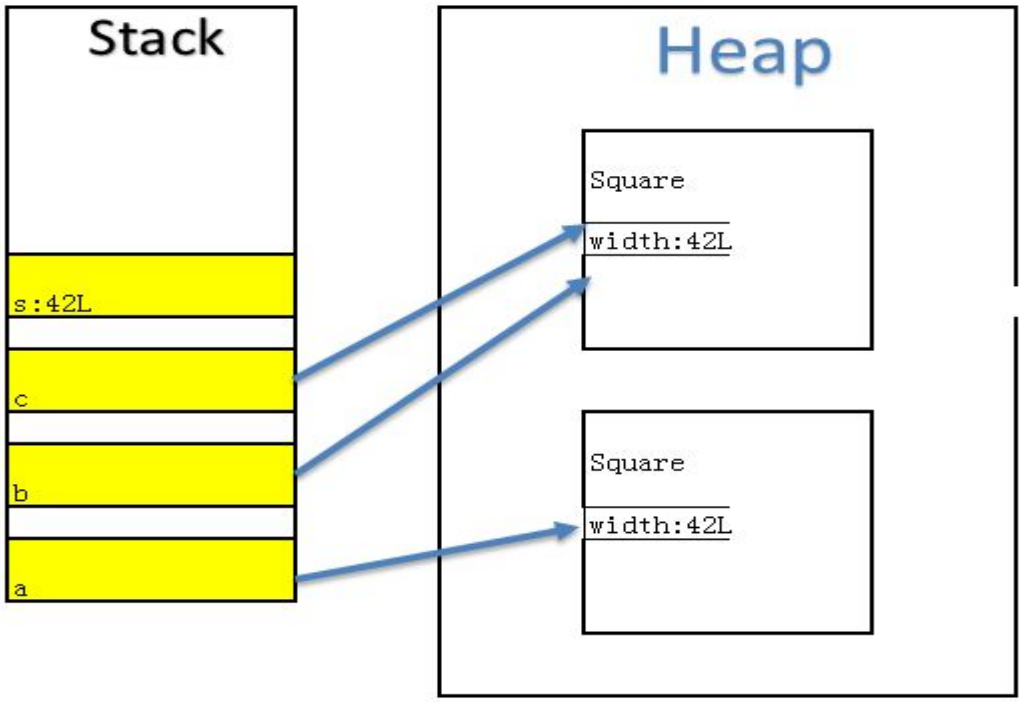
16.Integer.valueof 和 new Integer

Integer.valueof 的效率高，因为它用到了缓存，当传入整数 i(-128<i<127)时，从缓存里取出整数对应的 Integer 对象，否则创建一个 Integer 对象。new Integer 返回的永远是不同的对

象，但是当整数范围在-128<i<=127 时，Integer.valueOf 返回的是同一个对象

17.关于栈和堆

```
这题考的是引用和内存。
//声明了3个Square类型的变量a, b, c
//在stack中分配3个内存，名字为a, b, c
Square a, b, c;
//在heap中分配了一块新内存，里边包含自己的成员变量width值为48L，然后stack中的a指向这块内存
a = new Square(42L);
//在heap中分配了一块新内存，其中包含自己的成员变量width值为48L，然后stack中的b指向这块内存
b = new Square(42L);
//stack中的c也指向b所指向的内存
c = b;
//在stack中分配了一块内存，值为42
long s = 42L;
```



来看 4 个选项：

A: a == b

由图可以看出 a 和 b 指向的不是同一个引用，故 A 错

B：s == a

一个 Square 类型不能与一个 long 型比较，编译就错误，故 B 错

c：b == c

由图可以看出 b 和 c 指向的是同一个引用，故 C 正确

d：a equal s

程序会把 s 封装成一个 Long 类型，由于 Square 没有重写 Object 的 equals 方法， 所以调用的是 Object 类的 equals 方法，源码如下

```
public boolean equals(Object obj) {
        return (this == obj);
}
```

18.关于异常

1）Java 中，所有的异常都有一个共同的祖先 Throwable，他有两个中药的子类：Exception(异常)和 Error（错误）。

2）Error（错误）:是程序无法处理的错误，表示运行应用程序中较严重问题。大多数错误于代码编写者执行的操作无关，而表示代码运行时 JVM 出现了问题。如 Java 虚拟机运行错误，这些错误是不可查的，因为他们在应用程序的控制和处理能力之外，而且绝大多是是程序运行时不允许出现的状况。

3）Exception（异常）:是程序本身可以处理的异常。 Exception 类有一个重要的子类 RuntimeException。RuntimeException 类及其子类表示“JVM 常用操作”引发的错误。例如，若试图使用空值对象引用、除数为零或数组越界，则分别引发运行时异常（NullPointerException、ArithmeticException）和 ArrayIndexOutOfBoundException。

注意：异常能被程序本身处理，错误是无法处理的

4）通常，Java 的异常可分为可查的异常和不可查的异常。

可查异常：正确的程序运行中，很容易出现的、情理可容的异常状况。可查异常虽然是异常状况，但在一定程度上他的发生是可以预计的，一旦发生必须采取方式处理

5）Exception 这种异常分两大类运行时异常和非运行时异常(编译异常)。程序中应当尽可能去处理这些异常。

运行时异常：都是 RuntimeException 类及其子类异常，如 NullPointerException(空指针异常)、IndexOutOfBoundsException(下标越界异常)等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

运行时异常的特点是 Java 编译器不会检查它，也就是说，当程序中可能出现这类异常，即使没有用 try-catch 语句捕获它，也没有用 throws 子句声明抛出它，也会编译通过。

非运行时异常 （编译异常）：是 RuntimeException 以外的异常，类型上都属于 Exception 类及其子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过。如 IOException、SQLException 等以及用户自定义的 Exception 异常，一般情况下不自定义

检查异常

处理异常机制

在 Java 应用程序中，异常处理机制为：抛出异常，捕捉异常。

抛出异常：当一个方法出现错误引发异常时，方法创建异常对象并交付运行时系统，异常对象中包含了异常类型和异常出现时的程序状态等异常信息。运行时系统负责寻找处置异常的代码并执行。

捕获异常：在方法抛出异常之后，运行时系统将转为寻找合适的异常处理器（exception handler）。潜在的异常处理器是异常发生时依次存留在调用栈中的方法的集合。当异常处理器所能处理的异常类型与方法抛出的异常类型相符时，即为合适 的异常处理器。运行时系统从发生异常的方法开始，依次回查调用栈中的方法，直至找到含有合适异常处理器的方法并执行。当运行时系统遍历调用栈而未找到合适 的异常处理器，则运行时系统终止。同时，意味着 Java 程序的终止。

对于运行时异常、错误或可查异常，Java 技术所要求的异常处理方式有所不同。

由于运行时异常的不可查性，为了更合理、更容易地实现应用程序，Java 规定，运行时

异常将由 Java 运行时系统自动抛出，允许应用程序忽略运行时异常。

对于方法运行中可能出现的 Error，当运行方法不欲捕捉时，Java 允许该方法不做任何抛出声明。因为，大多数 Error 异常属于永远不能被允许发生的状况，也属于合理的应用程序不该捕捉的异常。

对于所有的可查异常，Java 规定：一个方法必须捕捉，或者声明抛出方法之外。也就是说，当一个方法选择不捕捉可查异常时，它必须声明将抛出异常。

能够捕捉异常的方法，需要提供相符类型的异常处理器。所捕捉的异常，可能是由于自身语句所引发并抛出的异常，也可能是由某个调用的方法或者 Java 运行时 系统等抛出的异常。也就是说，一个方法所能捕捉的异常，一定是 Java 代码在某处所抛出的异常。简单地说，异常总是先被抛出，后被捕捉的。

19.4 号牛客网

1）第一题：定义在接口中的方法只能是 public 的；容器保存的是对象的引用；在子类中，是可以覆盖父类的同步方法，而且不管子类是否同步；构造方法不需要同步化。
2）第二题：方法重载的返回值的类型可以不同，因为判断方法重载的方法主要是根据方法的参数不同来判定；方法重写的返回值类型需要相同，重写就是子类继承父类的方法，并在此方法上重写属于自己的特征，既然是继承过来的，那么它的返回值类型就必须要相同。
3）第三题：StringBuffer 类的对象调用 toString()方法将转换成为 String 类型；StringBuffer 类中有 append()方法，Stringl 类没有 append 方法；引用类型只有 String 可以直接复制，其他的都需要 new 出来，比如可以直接将字符串“test”复制给声明的 String 类，而想要给 StringBuffer 则需要 new 出来；StringBuffer 类的实例可以被改变，而 String 类不可被改变。
4）List 和 Set 都继承了 Collection 接口，Set 里面不允许有重复的元素且取元素的时候，没法说取第几个，只能以 iterator 接口取得所有的元素，再逐一遍历各个元素；List 元素允许有重复的值，但是他们有先后进入的顺序，且 List 除了可以使用 iterator 来取得所有的元素，还可以使用 get(index i)来明确说明取第几个；Map 没有继承 Collection 接口，每次存储都会存储一对 key/value，其中 key 不允许重复，这个重复的规定是按 equals 比较的，且既可以通过 get(Object key)获得值为 key 所对应的 value，也可以获得所有的 key 结合，也可以获得所有的 value 结合，还可以获得 key 和 value 组合成的 Map.Entry 对象的集合
5）关于 Spring 的以来注入：有两种：设值注入和构造注入，设值注入表示 IOC 容器使用属性 setter 方法来注入被依赖的实例，这种注入方式简单、直观；构造注入表示 IoC 容器使用构造器来注入被依赖的实例。在配置构造注入<constructor-arg>元素时，可以指定一个 index 属性，用于指定该构造参数值将作为第几个构造参数值。在创建实例中属性的时机不同，设置注入是先通过无参数的构造器创建一个 Bean 实例，然后调用对应的 setter 方法注入依赖关系；而构造注入则直接调用有参数的构造器，当 Bean 实例创建完成后，已经完成了依赖关系的注入。当设值注入和构造注入同时存在时，先执行设值注入，在执行构造注入。
6）HttpServletRequest 类主要处理是：1）读取和写入 HTTP 头标，2）取得和设置 cookies，3）取得路径信息，4）表示 HTTP 会话。。。设定响应的 content 类型的是 response 接口
7）ServletContext 对象：servlet 容器在启动时会加载 web 应用，并为每个 web 应用创建唯一的 servlet context 对象，可以把 ServletContext 看成是一个 web 应用的服务器端组件的共享内存，在 ServletContext 中可以存放共享数据。ServletContext 对象是真正的一个全局对象，凡是 web 容器中的 Servlet 都可以访问；而 ServletConfig 对象：用于封装 Servlet 的配置信息，从一个 servlet 被实例化后，对任何客户端在任何时候访问有效，但仅对 servlet 自身有效，

一个 servlet 的 servletConfig 对象不能另一个 servlet 访问
8） 机制：spring mvc 的入口是 servlet，而 struts2 是 filter。
补充几点知识：
Filter 实现 javax.servlet.Filter 接口，在 web.xml 中配置与标签指定使用哪个 Filter 实现类过滤哪些 URL 链接。只在 web 启动时进行初始化操作。 filter 流程是线性的， url 传来之后，检查之后，可保持原来的流程继续向下执行，被下一个 filter, servlet 接收等，而 servlet 处理之后，不会继续向下传递。filter 功能可用来保持流程继续按照原来的方式进行下去，或者主导流程，而 servlet 的功能主要用来主导流程。
特点：可以在响应之前修改 Request 和 Response 的头部，只能转发请求，不能直接发出响应。filter 可用来进行字符编码的过滤，检测用户
spring 会稍微比 struts 快。 spring mvc 是基于方法的设计 ， 而 sturts 是基于类 ，每次发一次请求都会实例一个 action，每个 action 都会被注入属性，而 spring 基于方法，粒度更细(粒度级别的东西比较 sychronized 和 lock)，但要小心把握像在 servlet 控制数据一样。spring3 mvc 是方法级别的拦截，拦截到方法后根据参数上的注解，把 request 数据注入进去，在 spring3 mvc 中，一个方法对应一个 request 上下文。而 struts2 框架是类级别的拦截，每次来了请求就创建一个 Action，然后调用 setter getter 方法把 request 中的数据注入；struts2 实际上是通过 setter getter 方法与 request 打交道的；struts2 中，一个 Action 对象对应一个 request 上下文。
3. 参数传递：struts 是在接受参数的时候，可以用属性来接受参数，这就说明参数是让多个方法共享的。所以 D 是对的。
4. 设计思想上： struts 更加符合 oop 的编程思想 ， spring 就比较谨慎，在 servlet 上扩展。
5. intercepter(拦截器)的实现机制：struts 有以自己的 interceptor 机制， spring mvc 用的是独立的 AOP 方式 。这样导致 struts 的配置文件量还是比 spring mvc 大，虽然 struts 的配置能继承，所以我觉得，就拿使用上来讲，spring mvc 使用更加简洁， 开发效率 Spring MVC 确实比 struts2 高 。 spring mvc 是方法级别的拦截，一个方法对应一个 request 上下文，而方法同时又跟一个 url 对应，所以说从架构本身上 spring3 mvc 就容易实现 restful url 。 struts2 是类级别的拦截，一个类对应一个 request 上下文；实现 restful url 要费劲，因为 struts2 action 的一个方法可以对应一个 url；而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了。 spring3 mvc 的方法之间基本上独立的，独享 request response 数据，请求数据通过参数获取，处理结果通过 ModelMap 交回给框架方法之间不共享变量， 而 struts2 搞的就比较乱，虽然方法之间也是独立的，但其所有 Action 变量是共享的，这不会影响程序运行，却给我们编码，读程序时带来麻烦。
6. 另外，spring3 mvc 的验证也是一个亮点，支持 JSR303， 处理 ajax 的请求更是方便 ，只需一个注解@ResponseBody ，然后直接返回响应文本即可。
9）用 System.in 创建 inStream 对象，表示从标准输入中获取数据，用 Sytem.out 创建 OutputStream 对象，表示输出到标准输出设备中。
10)Hibernate2 延迟加载实现：a)实体对象 b)集合。Hibernate3 提供了属性的延迟加载功能，当 Hibernate 在查询数据的时候，数据并没有存在内存中，当程序真正对数据的操作时，对象才存在内存中，就实现了延迟加载，他节省了服务器的内存开销，从而提高了服务器的性能；Hibernate 使用 Java 反射机制，而不是字节码增量程序来实现透明性；Hibernate 的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多等各种复杂关系；load 支持延迟加载，而 get 不支持延迟加载。
11）优化 Hibernate 所鼓励的七大措施：1）尽量使用 many-to-one，避免使用单项 one-to-many；2）灵活使用单向 one-to-many；3)不用一对一，使用多对一代替一对一，4）配置对象缓存，

不使用集合缓存，5）一对多使用 Bag,多对一使用 Set，6）继承使用显示多台 HQL，避免查处所有对象，7）消除大表，使用二级缓存

12）struts2 是一个基于 MVC 设计模式的 Web 框架，将视图，模型，控制层分开进行管理，结构变得清晰，方便了代码的维护。缺点是由于进行了分层管理，响应的 Action 就会变多，文件数目增多。

13）在 struts 框架中如果使用 validation 作验证的话，需要使用 DynaValidationActionForm

14）Spring mvc 中的 DispatcherServlet 是 Servlet，负责接收 HTTP 请求，可以在 web.xml 中配置，用于加载配置信息。当没有上下文时，他就会创建一个上下文。DispatcherServlet 用于分发 http 到具体的业务方法，所以实现业务其实是具体的 bean 方法。

15）JSP 的内置对象：

1.request 对象

客户端的请求信息被封装在 request 对象中，通过它才能了解到客户的需求，然后做出响应。它是 HttpServletRequest 类的实例。

2.response 对象

response 对象包含了响应客户请求的有关信息，但在 JSP 中很少直接用到它。它是 HttpServletResponse 类的实例。

3.session 对象

session 对象指的是客户端与服务器的一次会话，从客户连到服务器的一个 WebApplication 开始，直到客户端与服务器断开连接为止。它是 HttpSession 类的实例.

4.out 对象

out 对象是 JspWriter 类的实例,是向客户端输出内容常用的对象

5.page 对象

page 对象就是指向当前 JSP 页面本身，有点象类中的 this 指针，它是 java.lang.Object 类的实例

6.application 对象

application 对象实现了用户间数据的共享，可存放全局变量。它开始于服务器的启动，直到服务器的关闭，在此期间，此对象将一直存在；这样在用户的前后连接或不同用户之间的连接中，可以对此对象的同一属性进行操作；在任何地方对此对象属性的操作，都将影响到其他用户对此的访问。服务器的启动和关闭决定了 application 对象的生命。它是 ServletContext 类的实例。

7.exception 对象

exception 对象是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个 JSP 页面要应用此对象，就必须把 isErrorPage 设为 true，否则无法编译。他实际上是 java.lang.Throwable 的对象

8.pageContext 对象

pageContext 对象提供了对 JSP 页面内所有的对象及名字空间的访问，也就是说他可以访问到本页所在的 SESSION，也可以取本页面所在的 application 的某一属性值，他相当于页面中所有功能的集大成者，它的本 类名也叫 pageContext。

9.config 对象

config 对象是在一个 Servlet 初始化时，JSP 引擎向它传递信息用的，此信息包括 Servlet 初始化时所要用到的参数（通过属性名和属性值构成）以及服务器的有关信息（通过传递一个 ServletContext 对象）

16）exception 是 JSP 九大内置对象之一，其实例代表其他页面的异常和错误。只有当页面是错误处理页面时，即 isErroePage 为 true 时，该对象才可以使用。对于 C 项，errorPage

的实质就是 JSP 的异常处理机制,发生异常时才会跳转到 errorPage 指定的页面，没必要给 errorPage 再设置一个 errorPage。所以当 errorPage 属性存在时， isErrorPage 属性值为 false

9 月 11 号

1.HashMap 的 key 和 value 允许为 null，HashTable 的 key 和 value 不允许为 null

Java 的快速失败机制，即 fail-fast，它是 java 集合的一种错误检测机制。当多个线程对集合进行结构上的改变操作时，有可能会产生 fail-fast 机制。记住是可能的不是一定的。

HashMap 和 HashTable 两个类都实现了 Map 接口，二者保存 K-V；HashTable 的方法是 Synchronize 的，而 HashMap 不是，在多个线程访问 HashTable 时，不需要自己为它的方法实现同步，而 HashMap 就必须为之提供外同步；所有 HashMap 类的 collection 视图方法所返回的迭代器都是快速失败的：在迭代器创建之后，如果从结构上对映射进行修改，除非通过迭代器本身的 remove 方法，其他任何时间任何方式的修改，迭代器都将抛出 ConcurrentModicationException.。Hashtable 和 HashMap 的区别主要是前者是同步的，厚着是快速失败机制保证

2.Java 中的包

java.awt：包含构成抽象窗口工具集的多个类，用来构建和管理应用程序的图形用户界面

java.lang：提供 java 编程语言的程序设计的基础类

java.io：包含提供各种输出输入功能的类

java.net：包含执行与网络有关的类，如 URL,SOCKET,SETVERSOCKET

java.applet：包含 java 小应用程序的类

java.util：包含一些应用性的类

3. 依赖注入（Dependency Injection，简称 DI）是一个重要的面向对象编程的法则来削减计算机程序的耦合问题。依赖注入应用比较广泛。可以使应用程序的配置和依赖性规范与实际的应用程序代码分开。其中一个特点就是通过文本的配置文件进行应用程序组件间相互关系的配置，而不用重新修改并编译具体的代码。因此依赖注入降低了组件之间的耦合性，而不是使组件之间相互依赖。

4.常用的 servlet 包的名称是 javax.servlet 和 javax.servlet.http

5.关于赋值符号：+=是对的，<<=左移赋值，>>>=右移赋值，但是<<<=不是

6.HttpServletResponse：设置 HTTP 头标；设置 cookie；设定响应的 content 类型；输出返回的数据

7. Java 编译后生成字节码文件即.class 文件，然后 JVM 将字节码文件翻译成机器码文件

8.x++是运算完加 1，++x 是先加 1 后参与运算

9.Arraylist 构造参数有三个

1）ArrayList()构造一个初试容量为 10 的空列表

2）ArrayList(Collection<? extends E> c)构造一个包含指定 collection 的元素的列表，这些元素是按照该 collection 的迭代器返回他们的顺序排列的

3）ArrayList(int initialCapacity)构成一个具有指定初试容量的空列表

10.字符串常量池

一种为字面量形式，如 String str = "droid"，JVM 会先对这个字面量进行检查，如果字符串常量池中存在相同内容的字符串对象的引用，则将这个引用返回，否则新的字符串对象被创建，然后将这个引用放入字符串常量池，并返回该引用

另一种是使用 new 这种标准的构造对象的方法，如 String str = new String("droid")，此时，

不管字符串常量池中有没有相同内容的对象的引用，新的字符串对象都会创建。

例如：String str = new String("abc")，当你 new String("abc")时，其实会先在字符串常量区生成一个 abc 的对象，然后 new String()时会在堆中分配空间，然后此时会把字符串常量区中 abc 复制一个给堆中的 String，故 abc 应该在堆中和字符串常量区

9 月 12 号

1. ServletContext 对象：servlet 容器在启动时会加载 web 应用，并为每个 web 应用创建唯一的 servlet context 对象，可以把 ServletContext 看成是一个 Web 应用的服务器端组件的共享内存，在 ServletContext 中可以存放共享数据。ServletContext 对象是真正的一个全局对象，凡是 web 容器中的 Servlet 都可以访问。整个 web 应用只有唯一的一个 ServletContext 对象 servletConfig 对象：用于封装 servlet 的配置信息。从一个 servlet 被实例化后，对任何客户端在任何时候访问有效，但仅对 servlet 自身有效，一个 servlet 的 ServletConfig 对象不能被另一个 servlet 访问。

2. AOP 和 OOP 都是一套方法论，也可以说成设计模式、思维方式、理论规则等等。
AOP 不能替代 OOP，OOP 是 obejct abstraction，而 AOP 是 concern abstraction，前者主要是对对象的抽象，诸如抽象出某类业务对象的公用接口、报表业务对象的逻辑封装，更注重于某些共同对象共有行为的抽象，如报表模块中专门需要报表业务逻辑的封装，其他模块中需要其他的逻辑抽象，而 AOP 则是对分散在各个模块中的共同行为的抽象，即关注点抽象。一些系统级的问题或者思考起来总与业务无关又多处存在的功能，可使用 AOP，如异常信息处理机制统一将自定义的异常信息写入响应流进而到前台展示、行为日志记录用户操作过的方法等，这些东西用 OOP 来做，就是一个良好的接口、各处调用，但有时候会发现太多模块调用的逻辑大都一致、并且与核心业务无大关系，可以独立开来，让处理核心业务的人专注于核心业务的处理，关注分离了，自然代码更独立、更易调试分析、更具好维护。
核心业务还是要 OOP 来发挥作用，与 AOP 的侧重点不一样，前者有种纵向抽象的感觉，后者则是横向抽象的感觉， AOP 只是 OOP 的补充，无替代关系

3. 在接口中可以声明方法，属性信息和属性，但不能实例化方法，不能有字段。

4. Swing 是在 AWT 的基础上构建的一套新的图形界面系统，它提供了 AWT 所能够提供的所有功能，并且用纯粹的 Java 代码对 AWT 的功能进行了大幅度的扩充。AWT 是基于本地方法的 C/C++程序，其运行速度比较快；Swing 是基于 AWT 的 Java 程序，其运行速度比较慢。

5. 含有 abstract 修饰符的 class 即为抽象类，abstract 类不能创建的实例对象。含有 abstract 方法的类必须定义为 abstract class，abstract class 类中的方法不必是抽象的。abstract class 类中定义抽象方法必须在具体(Concrete)子类中实现，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为 abstract 类型。 接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为 public static final。
下面比较一下两者的语法区别：

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以有普通成员变量，接口中没有普通成员变量

3.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非

抽象的普通方法。

4. 抽象类中的抽象方法的访问类型可以是 public，protected 和（默认类型,虽然 eclipse 下不报错，但应该也不行），但接口中的抽象方法只能是 public 类型的，并且默认即为 public abstract 类型。

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

6. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类型。

7. 一个类可以实现多个接口，但只能继承一个抽象类。

6.JVM 的功能

（1）通过 ClassLoader 寻找和装载 class 文件

（2）解释字节码成为指令并执行，提供 class 文件的运行环境

（3）进行运行期间垃圾回收

（4）提供与硬件交互的平台

7._floor 返回不大于的最大整数

round 则是 4 舍 5 入的计算，入的时候是到大于它的整数

round 方法，它表示“四舍五入”，算法为 Math.floor(x+0.5)，即将原来的数字加上 0.5 后再向下取整，所以，Math.round(11.5)的结果为 12，Math.round(-11.5)的结果为-11。

ceil 则是不小于他的最小整数

例如 Math.ceil(-0.5)=-0.0

8. 定义在类中的变量是类的成员变量，可以不进行初始化，Java 会自动进行初始化，如果是引用类型默认初始化为 null,如果是基本类型例如 int 则会默认初始化为 0

局部变量是定义在方法中的变量，必须要进行初始化，否则不同通过编译

被 static 关键字修饰的变量是静态的，静态变量随着类的加载而加载，所以也被称为类变量

被 final 修饰发变量是常量

9. 其实都是引用传递，只是因为 String 是个特殊的 final 类，所以每次对 String 的更改都会重新创建内存地址并存储（也可能是在字符串常量池中创建内存地址并存入对应的字符串内容），但是因为这里 String 是作为参数传递的，在方法体内会产生新的字符串而不会对方法体外的字符串产生影响。

```
public class Example {
    String str = new String("good");
    char[] ch = { 'a', 'b', 'c' };

    public static void main(String args[]) {
        Example ex = new Example();
        ex.change(ex.str, ex.ch);
        System.out.print(ex.str + " and ");
        System.out.print(ex.ch);
    }

   public void change(String str, char ch[])
   {
        str = "test ok";
        ch[0] = 'g';
    }
}
```

结束后 str 的值没变，而 ch 的第一个变了

10.当一个优先级高的线程进入就绪状态时，他只是有较高的概率能够抢到 CPU 的执行权，不是一定就能得到执行权；当线程抛出一个例外时，该线程就终止了；当前线程调用 sleep（）方法或者 wait()方法时，知识暂时停止了该线程的运行，不是终止线程；当创建一个新线程的时候，该线程也加入到了抢占 cpu 执行权的队伍中，但是是否能抢到，并不清楚

11.对于线程局部存储 TLS（thread local storage）：解决多线程中的对同一变量的访问冲突的一种技术；TLS 会为每一个线程维护一个和该线程都绑定的变量的副本；Java 平台的 java.lang.ThreadLocal 是 TLS 技术的一种实现；

ThreadLocal 可以给一个初始值，而每个线程都会获得这个初始化值的一个副本，这样才能保证不同的线程都有一份拷贝。ThreadLocal 不是用于解决共享变量的问题的，不是为了协调线程同步而存在，而是为了方便每个线程处理自己的状态而引入的一个机制.

12.关键字 super 的作用：用来访问父类被隐藏的成员变量；用来调用父类中被重载的方法；使用 super 调用父类的构造方法

13.webservice 是基于 web 服务，是一种跨平台，跨语言的服务，它是采用 XML 传输格式化的数据，它的通信协议是 SOAP(简单对象访问协议)

14.HashMap 不能保证元素的顺序，HashMap 能够将键设为 Null，也可以将值设为 null，与只对应的是 HashTable，HashTable 不能将键和值设为 null，否则运行时会报空指针异常错误 HashMap 是线程不安全的，HashTable 是线程安全

15.关于 final：final 所修饰的成员变量只能赋值一次，可以在类方法中赋值，也可以在声明的时候直接赋值，而 final 修饰的局部变量可以在声明的时候初始化，也可以在第一次使用的时候通过方法或者表达式给它赋值。

final 修饰的成员变量为基本数据类型，在赋值之后无法改变。当 final 修饰的成员变量为引用数据类型的时候，在赋值后，其指向的地址无法改变，但是对象内容还是可以改变的。

final 修饰的成员变量在赋值时可有三种方式：1）在声明的时候直接赋值 2）在构造器中赋值。3）在初始化代码块中赋值

16.线程安全的同步类：Vector，stack 堆栈类, hashtable, enumeration 每句

17.JSP 只会在客户端第一次请求的时候被编译，之后的请求不会在编译，同时 tomcat 能自动检测 jsp 变更与否，变更了再进行编译。第一次编译在初始化时调用 init()方法，销毁时调用 destroy()方法。在整个 jsp 生命周期中均只调用一次。service()方法是接受请求，返回响应

的方法，每次请求都执行一次，该方法被 HttpServlet 封装为 doGet 和 doPost 方法
18. 顺序表可以采用链式存储或顺序存储。顺序存储占用连续的内存空间，但插入和删除操作需要移动其他元素，复杂度和插入或删除的位置有关，均摊复杂度达到 O(n),不便于插入和删除；链式存储不必占用连续的内存空间，插入删除操作只用操作相关节点的指针，是方便的

9 月 14 号

1.从 Servlet 中的 API 可以看出，抽象类 GenericServlet 实现了 Servlet，ServletConfig 接口
抽象类 HttpServlet 继承自 GenericServlet
2.标识符是以字母开头的字母数字序列
数字是指 0~9,字母指大小写英文字母、下划线（_）和美元符号（$）,也可以是 unicode 字符中的字符如汉字、数字等字符的任意组合，不能包含+，-等字符，不能使用关键字
3.将 GBK 编码字节流到 UTF 编码字节流的转换：先解码后编码
用 new String(src,"GBK")解码得到字符串，用 getBytes("UTF-8")得到 UTF8 编码字符数组
4.一般关系数据模型和对象数据模型之间有以下对应关系：表对应类，记录对应对象，表的字段对应类的属性
5.Java 的类加载器：
1）Booststrap ClassLoader/扩展类加载器
主要负责 jdk_home/lib 目录下的核心 api 或-xbootclasspath 选项指定的 jar 包装入工作
2）Extension ClassLoader/扩展类加载器
主要负责 jdk_home/lib/ext 目录下的 jar 包或-Djava.ext.dirs 指定目录下的 jar 包装入工作
3）System ClassLoader/系统类加载器
主要负责 java-classpath/-Djava.class.path 所指的目录下的类与 jar 包入工作
4）User Custom ClassLoder/用户自定义类加载器（java.lang.ClassLoader 的子类）
在程序运行期间，通过 java.lang.ClassLoader 的子类动态加载 class 文件，体现 java 动态实时类装入特性。

9 月 17 号

1.ASP 中 session 对象有效期为 20 分钟，JSP 中为 30 分钟
2.final 类型的变量一定要初始化，因为 final 类型的变量不可变
3.值传递：就是将该值的副本传过去（基本数据类型+String 类型的传递）
引用传递：就是将值的内存地址传过去（除了基本数据类型+String 以外类型的传递）
4. EJB 容器：Enterprise java bean 容器。更具有行业领域特色。他提供给运行在其中的组件 EJB 各种管理功能。只要满足 J2EE 规范的 EJB 放入该容器，马上就会被容器进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如邮件服务、事务管理。
JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系，让其它各地的应用程序在其上面留下自己的索引，从而

满足快速查找和定位分布式应用程序的功能。
JMS：（Java Message Service）JAVA 消息服务。主要实现各个应用程序之间的通讯。包括点对点和广播。
JTA：（Java Transaction API）JAVA 事务服务。提供各种分布式事务服务。应用程序只需调用其提供的接口即可。
JAF：（Java Action FrameWork）JAVA 安全认证框架。提供一些安全控制方面的框架。让开发者通过各种部署和自定义实现自己的个性安全控制策略。
RMI/IIOP:（Remote Method Invocation /internet 对象请求中介协议）他们主要用于通过远程调用服务。例如，远程有一台计算机上运行一个程序，它提供股票分析服务，我们可以在本地计算机上实现对其直接调用。当然这是要通过一定的规范才能在异构的系统之间进行通信。RMI 是 JAVA 特有的。
5. run 方法是线程内重写的一个方法，start 一个线程后使得线程处于就绪状态，当 jvm 调用的时候，线程启动会运行 run。run 函数是线程里面的一个函数不是多线程的。
6. 方法重载是指在一个类中定义多个同名的方法，但要求每个方法具有不同的参数的类型或参数的个数。
原则如下：
一.方法名一定要相同。
二.方法的参数表必须不同，包括参数的类型或个数，以此区分不同的方法体。
1.如果参数个数不同，就不管它的参数类型了！
2.如果参数个数相同，那么参数的类型或者参数的顺序必须不同。
三.方法的返回类型、修饰符可以相同，也可不同。
7. request 的 forward 和 response 的 redirect
1.redirect 地址栏变化，forward 发生在服务器端内部从而导致浏览器不知道响应资源来自哪里,即内部重定向
2.redirect 可以重定向到同一个站点上的其他应用程序中的资源，forward 只能将请求 转发给同一个 WEB 应用中的组件
3.redirect 默认是 302 码，包含两次请求和两次响应
4.redirect 效率较低
8. 会话跟踪是一种灵活、轻便的机制，它使 Web 上的状态编程变为可能。
HTTP 是一种无状态协议，每当用户发出请求时，服务器就会做出响应，客户端与服务器之间的联系是离散的、非连续的。当用户在同一网站的多个页面之间转换时，根本无法确定是否是同一个客户，会话跟踪技术就可以解决这个问题。当一个客户在多个页面间切换时，服务器会保存该用户的信息。
有四种方法可以实现会话跟踪技术：URL 重写、隐藏表单域、Cookie、Session。
1）.隐藏表单域：<input type="hidden">，非常适合步需要大量数据存储的会话应用。
2）.URL 重写:URL 可以在后面附加参数，和服务器的请求一起发送，这些参数为名字/值对。
3）.Cookie:一个 Cookie 是一个小的，已命名数据元素。服务器使用 SET-Cookie 头标将它作为 HTTP
响应的一部分传送到客户端，客户端被请求保存 Cookie 值，在对同一服务器的后续请求使用一个
Cookie 头标将之返回到服务器。与其它技术比较，Cookie 的一个优点是在浏览

器会话结束后，甚至在客户端计算机重启后它仍可以保留其值
4）.Session：使用 setAttribute(String str,Object obj)方法将对象捆绑到一个会话
9.

```
public class Test{
    public int add(int a,int b){
        try {
            return a+b;
        }
        catch (Exception e) {
            System.out.println("catch语句块");
        }
        finally{
            System.out.println("finally语句块");
        }
        return 0;
    }
     public static void main(String argv[]){
        Test test =new Test();
        System.out.println("和是："+test.add(9, 34));
     }
}
```

先执行 finally，在执行 return

9 月 18 号

1.JSP 分页代码中的执行次序是：先去本页的数据，得到总页数，再取总记录数，最后显示所有的记录
2. 派生类可以访问父类的 public 和 protected 属性成员
3.

```
int i=0;
Integer j = new Integer(0);
System.out.println(i==j);
System.out.println(j.equals(i));
```

答案：true、true

==如果是 primitive 主类型，那么比较值；如果是对象，那么比较引用地址

equals 需要根据具体对象的实现来判断，在 Integer 里面是判断值是否相等

一般说来，如果是两个 Integer 类型进行==比较，就是比较两个 Integer 对象的地址。但是有一点需要注意的是在-128 至 127 这个区间，如果创建 Integer 对象的时候（1）Integer i = 1; （2） Integer i = Integer.valueOf(1); 如果是这两种情况创建出来的对象，那么其实只会创建一个对象，这些对象已经缓存在一个叫做 IntegerCache 里面了，所以==比较是相等的。

如果不在-128 至 127 这个区间，不管是通过什么方式创建出来的对象，==永远是 false，也就是说他们的地址永远不会相等。

4. public interface CallableStatement extends PreparedStatement
public interface PreparedStatement extends Statement
5. 首先==与 equals 是有明显区别的。==强调栈中的比较，可以理解为地址比较 equals 强调对象的内容比较 String s="hello"；会在栈中生成 hello 字符串，并存入字符串常量池中。String t="hello" ；创建时，会在字符串常量池中寻找，当找到需要的 hello 时，不进行字符串的创建，引用已有的。所以，s==t 返回 trues.equals(t)也是 true。char c[]={'h','e','l','l','o'}; c==s 这个是不存在的，==两边类型不同，t.equals(c)这个语句在 anObject instanceof String 这步判断不会通过，也就是 cha[]压根不能与 String 相比较，类型不是相同的。返回 false
6. 初始化块在构造器执行之前执行，类初始化阶段先执行最顶层父类的静态初始化块，依次向下执行，最后执行当前类的静态初始化块；创建对象时，先调用顶层父类的构造方法，依次向下执行，最后调用本类的构造方法。
7. 释放掉占据的内存空间是由 gc 完成，但是程序员无法明确强制其运行，该空间在不被引用的时候不一定会立即被释放，这取决于 GC 本身，无法由程序员通过代码控制。
8. 源码程序中用到了一个重要的内部接口：Map.Entry，每个 Map.Entry 其实就是一个 key-value 对。当系统决定存储 HashMap 中的 key-value 对时，完全没有考虑 Entry 中的 value，仅仅只是根据 key 来计算并决定每个 Entry 的存储位置。Entry 是数组，数组中的每个元素上挂这个一条链表。链表法就是将相同 hash 值的对象组织成一个链表放在 hash 值对应的槽位；开放地址法是通过一个探测算法，当某个槽位已经被占据的情况下继续查找下一个可以使用的槽位。很显然我们使用的不是开放地址法。
9.Java 只允许为类的扩展做单一继承，但是允许使用接口做多重扩展；接口中可以定义成员变量，且必须是 static final 的；抽象类和接口都不能使用 new 操作
10. 1).HashTable 的方法是同步的，HashMap 未经同步，所以在多线程场合要手动同步 HashMap 这个区别就像 Vector 和 ArrayList 一样。
2).HashTable 不允许 null 值(key 和 value 都不可以),HashMap 允许 null 值(key 和 value 都可以)。
3)HashTable 有一个 contains(Object value)，功能和 containsValue(Object value)功能一样。
4).HashTable 使用 Enumeration，HashMap 使用 Iterator。
5).HashTable 中 hash 数组默认大小是 11，增加的方式是 old*2+1。HashMap 中 hash 数组的默认大小是 16，而且一定是 2 的指数。
6).哈希值的使用不同，HashTable 直接使用对象的 hashCode，而 HashMap 重新计算 hash 值，而且用与代替求模：
11. 先计算 return 里的值，然后，去执行 finally 里的代码，再回到 return 来返回
12. Java 中，如果对整数不指定类型，默认时 int 类型，对小数不指定类型，默认

是 double 类型

13. Thread.sleep()会抛出 InterruptedException，这个属于 checked exception，也就是编译时异常，我们必须显式的捕获异常而不能继续上外层抛出，因为这个异常需要该线程自己来解决。

14. 执行父类的带参构造前要先对父类中的对象进行初始化

15. Iterator 支持从源集合中安全地删除对象，只需在 Iterator 上调用 remove() 即可。这样做的好处是可以避免 ConcurrentModifiedException ，当打开 Iterator 迭代集合时，同时又在对集合进行修改。有些集合不允许在迭代时删除或添加元素，但是调用 Iterator 的 remove() 方法是个安全的做法。

16. Java 把内存划分成两种：一种是栈内存，另一种是堆内存。

在函数中定义的一些基本类型的变量和对象的引用变量都是在函数的栈内存中分配，当在一段代码块定义一个变量时，Java 就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java 会自动释放掉为该变量分配的内存空间，该内存空间可以立即被另作它用。数组和对象在没有引用变量指向它的时候，才变为垃圾，不能再被使用，但仍然占据内存空间不放，在随后的一个不确定的时间被垃圾回收器收走（释放掉）。这也是 Java 比较占内存的原因。

17. 基本类型作为形式参数传递不会改变实际参数，引用类型作为形式参数传递会改变实际参数，但是 JDK1.5 以后，对基本类型的包装类型（int-Integer,double-Double）提供了自动拆装箱的功能，把 Integer 类型作为参数传递，会自动拆箱为基本类型，不会改变实际参数的值

18.

1. HashMap,TreeMap 未进行同步考虑，是线程不安全的。
2. HashTable 和 ConcurrentHashMap 都是线程安全的。区别在于他们对加锁的范围不同，HashTable 对整张 Hash 表进行加锁，而 ConcurrentHashMap 将 Hash 表分为 16 桶(segment)，每次只对需要的桶进行加锁。
3. Collections 类提供了 synchronizedXxx()方法，可以将指定的集合包装成线程同步的集合。比如，

```
List   list = Collections.synchronizedList(new ArrayList());
Set   set = Collections.synchronizedSet(new HashSet())
```

19. Java 语言提供了一种稍弱的同步机制,即 volatile 变量.用来确保将变量的更新操作通知到其他线程,保证了新值能立即同步到主内存,以及每次使用前立即从主内存刷新. 当把变量声明为 volatile 类型后,编译器与运行时都会注意到这个变量是共享的

9 月 19 号

1. ResultSet 跟普通的数组不同，索引从 1 开始而不是从 0 开始

2. (1)抽象类中的抽象方法（其前有 abstract 修饰）不能用 private、static、synchronized、native 访问修饰符修饰。原因如下：抽象方法没有方法体，是用来被继承的，所以不能用 private 修饰；static 修饰的方法可以通过类名来访问该方法（即该方法的方法体），抽象方法用 static 修饰没有意义；使用 synchronized 关键字是为该方法加一个锁。。而如果该关键字修饰的方法是 static 方法。则使用的锁就是 class 变量的锁。如果是修饰 类方法。则用 this 变量锁。但是抽象类不能实例化对象，因为该方法不是在该抽象类中实现的。是在其子类实现的。所以。锁应该归其子类所有。所以。抽象方 法也就不能用 synchronized 关键字修饰了；native，这个东西本身就和 abstract 冲突，他们都是方法的声明，只是一个吧方法实现移交给子类，另一个是移交给本地操作系统。如果同时出现，就相当于即把实现移交给子类，又把实现移交给本地操作系统，那到底谁来实现具体方法呢？

(2)接口是一种特殊的抽象类，接口中的方法全部是抽象方法（但其前的 abstract 可以省略），所以抽象类中的抽象方法不能用的访问修饰符这里也不能用。而且 protected 访问修饰符也不能使用，因为接口可以让所有的类去 实现（非继承），不只是其子类，但是要用 public 去修饰。接口可以去继承一个已有的接口。

3. 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。这里的所谓动态数组并不是那个“ 有多少元素就申请多少空间 ”的意思，通过查看源码，可以发现，这个动态数组是这样实现的，如果没指定数组大小，则申请默认大小为 10 的数组，当元素个数增加，数组无法存储时，系统会另个申请一个长度为当前长度 1.5 倍的数组，然后，把之前的数据拷贝到新建的数组。

B. 对于随机访问 get 和 set，ArrayList 觉得优于 LinkedList，因为 LinkedList 要移动指针。//正确，ArrayList 是数组，所以，直接定位到相应位置取元素，LinkedLIst 是链表，所以需要从前往后遍历。

C. 对于新增和删除操作 add 和 remove，LinedList 比较占优势，因为 ArrayList 要移动数据。//正确，ArrayList 的新增和删除就是数组的新增和删除，LinkedList 与链表一致。

D. ArrayList 的空间浪费主要体现在在 list 列表的结尾预留一定的容量空间，而 LinkedList 的空间花费则体现在它的每一个元素都需要消耗相当的空间。//正确，因为 ArrayList 空间的增长率为 1.5 倍，所以，最后很可能留下一部分空间是没有用到的，因此，会造成浪费的情况。对于 LInkedList 的话，由于每个节点都需要额外的指针

4. 在 java 中，声明一个数组时，不能直接限定数组长度，只有在创建实例化对象时，才能对给定数组长度.

5.重载是在同一个类中，有多个方法名相同，参数列表不同（参数个数不同，参数类型不同），与方法的返回值无关，与权限修饰符无关。

6.一个完整的 URL 地址由协议，主机名，端口和文件四部分组成

超文本传输协议（HTTP）的统一资源定位符将从因特网获取信息的五个基本元素包括在一个简单的地址中：1)传送协议。2)服务器。3)端口号。（以数字方式

表示，若为 HTTP 的默认值“:80”可省略）。4)路径。（以“/”字符区别路径中的每一个目录名称）。5)查询。（GET 模式的窗体参数，以“?”字符为起点，每个参数以“&”隔开，再以“=”分开参数名称与数据，通常以 UTF8 的 URL 编码，避开字符冲突的问题）

7.关于垃圾回收器：java 中垃圾回收线程基本是所有线程中优先级最低的；垃圾回收器处理哪一个对象是由 JVM 决定的；垃圾回收与内存溢出是不同的概念，垃圾回收释放不用的内存，不能保证内存不会溢出；进入 dead 的线程最后会调用 finilized 方法，有可能是 dead 线程重新复活

8.CopyOnWriteArrayList 适用于写少读多的并发场景；ReadWriteLock 即为读写锁，它要求写与写之间互斥，读与写之间互斥，读与读之间可以并发执行。在读多写少的情况下可以提高效率；ConcurrentHashMap 是同步的 HashMap，读写都加锁；volatile 只保证多线程操作的不可见性，不保证原子性

9 月 20 号

1.关于面向对象的基本原则是：

1）单一职责原则（Single-Resposibility Principle）：一个类，最好只做一件事，只有一个引起它的变化。单一职责原则可以看做是低耦合、高内聚在面向对象原则上的引申，将职责定义为引起变化的原因，以提高内聚性来减少引起变化的原因。

2）开放封闭原则（Open-Closed principle）：软件实体应该是可扩展的，而不可修改的。也就是，对扩展开放，对修改封闭的。

3）替换原则（Liskov-Substituion Principle）：子类必须能够替换其基类。这一思想体现为对继承机制的约束规范，只有子类能够替换基类时，才能保证系统在运行期内识别子类，这是保证继承复用的基础。

4）依赖倒置原则（Dependecy-Inversion Principle）：依赖于抽象。具体而言就是高层模块不依赖于底层模块，二者都同依赖于抽象；抽象不依赖于具体，具体依赖于抽象。

5）接口隔离原则（Interface-Segregation Principle）：使用多个小的专门的接口，而不要使用一个大的总接口

2.最终类就是被 final 修饰的类

3.Java 并发编程的同步器有：

1）Java 并发库 的 Semaphore 可以很轻松完成信号量控制，Semaphore 可以控制某个资源可被同时访问的个数，通过 acquire() 获取一个许可，如果没有就等待，而 release() 释放一个许可。

2）CyclicBarrier 主要的方法就是一个：await()。await() 方法没被调用一次，计数便会减少 1，并阻塞住当前线程。当计数减至 0 时，阻塞解除，所有在此 CyclicBarrier 上面阻塞的线程开始运行。

3）直译过来就是倒计数(CountDown)门闩(Latch)。倒计数不用说，门闩的意思顾名思义就是阻止前进。在这里就是指 CountDownLatch.await() 方法在倒计数为 0 之前会阻塞当前线程。

4. 有关 Java threadlocal 的说法：ThreadLocal 存放的值是线程封闭，线程间互斥的，主要用于线程内共享一些数据，避免通过参数来传递；线程的角度看，每个线程都保持一个对其线程局部变量副本的隐式引用，只要线程是活动的并且 ThreadLocal 实例是可访问的；在线程消失之后，其线程局部实例的所有副本都会

被垃圾回收；在 Thread 类中有一个 Map，用于存储每一个线程的变量的副本；对于多线程资源共享的问题，同步机制采用了“以时间换空间”的方式，而 ThreadLocal 采用了“以空间换时间”的方式

5. 接口（interface）可以说成是抽象类的一种特例，接口中的所有方法都必须是抽象的。接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为 public static final。另外，接口和抽象类在方法上有区别：

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

3.抽象类中可以有普通成员变量，接口中没有普通成员变量

4. 抽象类中的抽象方法的访问类型可以是 public，protected 和默认类型

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

6. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类型

7. 一个类可以实现多个接口，但只能继承一个抽象类。二者在应用方面也有一定的区别：接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用，例如，模板方法设计模式是抽象类的一个典型应用，假设某个项目的所有 Servlet 类都要用相同的方式进行权限判断、记录访问日志和处理异常，那么就可以定义一个抽象的基类，让所有的 Servlet 都继承这个抽象基类，在抽象基类的 service 方法中完成权限判断、记录访问日志和处理异常的代码，在各个子类中只是完成各自的业务逻辑代码。

6.有关 java 并发过程：

CopyOnWriteArrayList 适用于写少读多的并发场景；ReadWriteLock 即为读写锁，他要求写与写之间互斥，读与写之间互斥，读与读之间可以并发执行。在读多写少的情况下可以提高效率；ConcurrentHashMap 是同步的 HashMap，读写都加锁；volatile 只保证多线程操作的可见性，不保证原子性

7.关于线程：suspend() 和 resume() 方法：两个方法配套使用，suspend()使得线程进入阻塞状态，并且不会自动恢复，必须其对应的 resume() 被调用，才能使得线程重新进入可执行状态

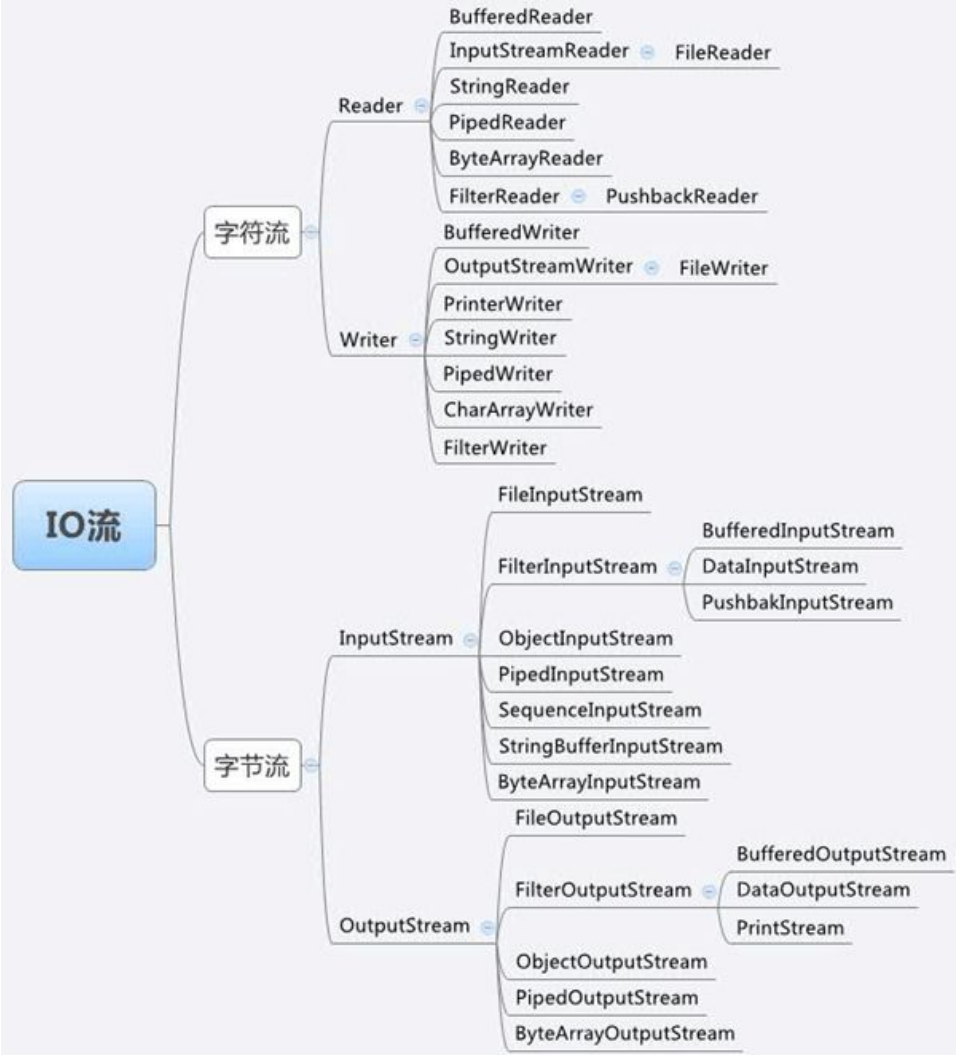
8.关于 Java 序列化：能够对对想进行传输的只有 ObjectOutputStream 和 ObjectInputStream 这些以 Object 开头的流对象；序列化的类必须继承 Seriallizable 接口，；transient 修饰的变量在对象序列化的时候并不会将所赋值的值保存到串中，串化的对象从磁盘读取出来仍然是 Null

9.类中有缺省构造器，不一定要定义构造器；方法可以与类同名，和构造器的区别在于必须要有返回值类型；一个类可以定义多个参数列表不同的构造器，实现构造器重载

10.Java 程序的种类有：1）内嵌于 Web 文件中，由浏览器来观看的 Applet.2)可独立运行的 Application 3）服务器端的 Servlets

9 月 21 号

1.



2.计算机中以补码存储：正数的原码/反码/补码相同，所以

10 存储为 00000000 00000000 00000000 00001010

~10 的原码为：11111111 11111111 11111111 11110101(10 取反)

~10 的反码为：10000000 00000000 00000000 00001010（最高位符号位不变，其余为取反）

~10 的补码：10000000 00000000 00000000 00001001（负数的补码=反码+1）所以~10=-11

3.写 Java 程序的时候只是设定事务的隔离级别，而不是去实现它；Hibernate 是一个 java 的数据持久化框架，方便数据库的访问；事务隔离级别由数据库系统实现，是数据库系统本身的一个功能；JDBC 是 java database connector,也就是 java

访问数据库的驱动；在数据库操作中，为了有效保证并发读取数据的正确性，提出的事务隔离级别；为了解决更新丢失，脏读，不可重复读（包括虚读和幻读）等问题在标准 SQL 规范中，定义了 4 个事务隔离级别，分别为未授权读取，也成为读未提交（read uncommitted）；授权读取，也成为读提交（read committed);可重复读取（repeatable read）;序列化（serializable）

4.Java 中使用包是为了方便管理和维护.java 文件，我们可以编写相同名字的 java 文件放到不同的包下。package 语句是 java 文件的第一句话，并且只有一句，而 import 语句可以标明来自其他包中的类，且 import 语句可以有多句

5.Java 线程之间的通信由 Java 内存模型（简称 JMM）控制，JMM 决定一个线程对共享变量的写入何时对另一个线程可见，从抽象的角度来看，JMM 定义了线程和主内存之间的抽象关系；线程之间的共享变量存储在主内存中，每个线程都有一个私有的本地内存，本地内存中存储了该线程以读/写共享变量的副本。本地内存是 JMM 的一个抽象概念，并不真实存在。它涵盖了缓存，写缓冲区，寄存器以及其他的硬件和编译器优化；volatile 变量的写-读可以实现线程之间的通信。从内存语义的角度来说，volatile 与监视器锁有相同的效果；volatile 写和监视器的释放有相同的内存语义；volatile 读与监视器的获取有相同的内存语义。

6.static 变量只会创建一份，所以不管创建几个对象，其中的 static 变量的值都一份

7.文件有文件名和数据，这在 linux 上被分成两个部分：用户数据和元数据。用户数据，即文件数据块，数据块是记录文件真实内容的地方；而元数据则是文件的附加属性，如文件大小、创建时间、所有者等信息。在 linux 中，元数据中的 inode 号（inode 是文件元数据的一部分但其并不包含文件名，inode 号即索引节点号）才是文件的唯一标识而非文件名。文件名仅是为了方便人们的记忆和使用，系统或程序通过 inode 号寻找正确的文件数据块；硬连接和软连接，连接为 linux 系统解决了文件的共享使用，还带来了隐藏文件路径、增加权限安全及节省存储等好处。若一个 inode 号对应多个文件名，则称这些文件为硬连接。换言之，硬连接就是同一个文件使用多个别名。若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软链接。可以使用 stat 命令来查看文件更多的元数据信息

8. java 不允许单独的方法，过程或函数存在,需要隶属于某一类中；java 语言中的方法属于对象的成员,而不是类的成员。不过，其中静态方法属于类的成员。

9.Java 语言中的异常处理包括声明异常、抛出异常、捕获异常和处理异常四个环节

throw 用于抛出异常；throws 关键字可以在方法上声明该方法要抛出的异常，然后在方法内部通过 throw 抛出异常对象；try 是用于检测被包住的语句块是否出现异常，如果有异常，则抛出异常，并执行 catch 语句；cacth 用于捕获从 try 中抛出的异常并作出处理；finally 语句块是不管有没有出现异常都要执行的内容。

10.

修饰符	类内部	同一个包	子类	任何地方
private	Yes			
default	Yes	Yes		
protected	Yes	Yes	Yes	
public	Yes	Yes	Yes	Yes

11.构造方法：
1）构造方法可以被重载，一个构造方法可以通过 this 关键字调用另一个构造方法，this 语句必须位于构造方法的第一行；重载：重载构成的条件：方法的名称相同，但是参数类型或参数个数不同，才能构成方法的重载。
2）当一个类中没有定义任何构造方法，Java 将自动提供一个缺省的构造方法
3）子类通过 super 关键字调用父类的一个构造方法
4）当子类的某个构造方法没有通过 super 关键字调用父类的构造方法时，通过这个构造方法创建子类对象时，会自动先调用父类的缺省构造方法
5）构造方法不能被 static、final、synchronized、abstract、native 修饰，但可以被 public、private、protected 修饰
6）构造方法不是类的成员方法
7）构造方法不能被继承
12.泛型通配符： ？表示任意类型，如果没有明确，那么就是 Object 以及任意的 Java 类；？ extends E：向下限定，E 及其子类；？ super E：向上限定，E 及其父类
13.List，Set，Map 在 java.util 包下都是接口，其中 List 有两个实现类：ArrayList 和 LinkedList；Set 有两个实现类：HashSet 和 LinkedHashSet；AbstractSet 实现了 Set
14.静态初始化代码快、构造代码快、构造方法，当涉及到继承时，需按照如下顺序执行：1）执行父类的静态代码快 2）执行子类的静态代码快 3）执行父类的构造代码块 4）执行父类的构造函数 5）执行子类的构造代码快 6）执行子类的构造函数
15.关于 struts1 和 struts2
从 action 类上分析:
1)Struts1 要求 Action 类继承一个抽象基类。Struts1 的一个普遍问题是使用抽象类编程而不是接口。
2) Struts 2 Action 类可以实现一个 Action 接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2 提供一个 ActionSupport 基类去实现常用的接口。Action 接口不是必须的，任何有 execute 标识的 POJO 对象都可以用作 Struts2 的 Action 对象。
从 Servlet 依赖分析:
3) Struts1 Action 依赖于 Servlet API ,因为当一个 Action 被调用时 HttpServletRequest 和 HttpServletResponse 被传递给 execute 方法。
4) Struts 2 Action 不依赖于容器，允许 Action 脱离容器单独被测试。如果需要，Struts2 Action 仍然可以访问初始的 request 和 response。但是，其他的元素减少或者消除了直接访问 HttpServetRequest 和 HttpServletResponse 的必要性。
从 action 线程模式分析:

5) Struts1 Action 是单例模式并且必须是线程安全的，因为仅有 Action 的一个实例来处理所有的请求。单例策略限制了 Struts1 Action 能作的事，并且要在开发时特别小心。Action 资源必须是线程安全的或同步的。
6) Struts2 Action 对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet 容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）
16.Object 方法的介绍

1)getClass()返回一个对象的运行时类。该 Class 对象是由所表示类的 static synchronized 方法锁定的对象

返回：表示该对象的运行时类的 java.lang.Class 对象。此结果属于类型 Class<? extends X>，其中 X 表示清除表达式中的静态类别，该表达式调用 getClass

2)hashCode()方法返回该对象的哈希码值

3)equals()方法指示某个其他对象是否与此对象相等；当此方法被重写时，有必要重写 hashCode 方法，以维护 hashCode 方法的常协规定。

4)clone()方法：创建并返回此对象的一个副本，如果对象的类不支持 Cloneable 接口，则抛出 CloneNotSupportedException

5)toString()方法，返回该对象的字符串表示。

6)notify()方法唤醒在此对象监视器上等待的单个线程。如果所有线程都在此对象上等待，则会选择唤醒其中一个对象，选择是任意性的，并在对实现做出决定时发生。线程通过调用其中一个 wait 方法，在此对象的监视器上等待。

此方法只应由作为此对象监视器的所有者的线程来调用，通过一下三种方法之一，线程可以成为此对象监视器的所有者：

1)通过执行此对象的同步(synchronized)实例方法

2)通过执行在此对象上进行同步 synchronized 语句的正文

3)对于 Class 类型的对象，可以通过执行该类的同步静态方法。

一次只能有一个线程拥有对象的监视器

7)notifyAll()唤醒再次对象监视器上等待的所有线程

8)finalize()方法，当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收期调用此方法。finalize 的常规协定是：当 JavaTM 虚拟机已确定尚未终止的任何线程无法再通过任何方法访问此对象时，将调用此方法，除非由于准备终止的其他某个对象或类的终结操作执行了某个操作。finalize 方法可以采取任何操作，其中包括再次使此对象对其他线程可用；不过，finalize 的主要目的是在不可撤消地丢弃对象之前执行清除操作。例如，表示输入/输出连接的对象的 finalize 方法可执行显式 I/O 事务，以便在永久丢弃对象之前中断连接。Object 类的 finalize 方法执行非特殊性操作；它仅执行一些常规返回。Object 的子类可以重写此定义。Java 编程语言不保证哪个线程将调用某个给定对象的 finalize 方法。但可以保证在调用 finalize 时，调用 finalize 的线程将不会持有任何用户可见的同步锁定。如果 finalize 方法抛出未捕获的异常，那么该异常将被忽略，并且该对象的终结操作将终止。在启用某个对象的 finalize 方法后，将不会执行进一步操作，直到 Java 虚拟机再次确定尚未终止的任何线程无法再通过任何方法访问此对象，其中包括由准备终止的其他对象或类执行的可能操作，在执行该操作时，对象可能被丢弃。对于任何给定对象，Java 虚拟机最多只调用一次 finalize 方法。finalize 方法抛出的任何异常都会导致此对象的终结操作停止，但可以通过其他方法忽略它。

9)wait()导致当前的线程等待，知道其他线程调用此对象的 notify()方法或者 notifyAll()方法，或者超过指定的时间量

17.java.exe 是 java 虚拟机；javadoc.exe 用来制作 java 文档；jdb.exe 是 java 的调试器；javaprof.exe 是剖析工具

18.Java 中的变量和基本类型的内存分配见：

http://my.oschina.net/u/551903/blog/134000

19.局部变量（在方法体内声明的变量）假如没有显示初始化，根据所在 java 环境不同，编译器会给出一个警告或者一个错误。

9 月 22 号

1.计算余弦值使用 Math 的 cos()方法；toRadians()是将角度转换为弧度；toDegrees()是将弧度转化为角度。

2.Java 鲁棒性：Java 能检查程序在编译和运行时的错误；Java 自己操纵内存减少了内存出错的可能性；Java 还实现了真数组，避免了覆盖数据的可能

3.形式参数即函数定义时设定的参数，实际参数是调用函数时所使用的实际的参数，形式参数可被视为 local variable；在函数的中真正传递的是实参；形参可以是对象，是对象的时候传递引用。

4.若子类继承了父类，当实例化一个子类时，需要去调用父类的无参数的构造函数，若父类没有无参的构造函数，所以子类需要在自己的构造函数中显示调用父类的构造函数，需要添加 super()

5.Ant 和 Maven 是基于 Java 的构建工具：

1）Ant 没有一个约定的目录结构，必须明确让 ant 做什么，什么时候做，然后编译，打包，没有生命周期，必须定义目标及其实现的任务序列，没有集成依赖管理

2）Maven 拥有约定，知道代码在哪里，放到哪里去，拥有一个声明周期，例如执行 mvn install 就可以自动执行编译，测试，打包等，构建过程只需要定义一个 pom.xml，然后把源码放到默认的目录，Maven 帮你处理其他事情，拥有依赖管理，仓库管理

6.ThreadLocal 使用场合主要解决多线程中数据因并发产生不一致的问题。ThreadLocal 为每个线程中的并发访问的数据提供一个副本，通过访问副本来运行业务，这样的结果是耗费了内存，但大大减少了线程同步所带来性能的消耗，也减少了线程并发控制的复杂度。ThreadLocal 不能使用原子类型，只能使用 Object 类型。ThreadLocal 的使用比 synchronized 要简单的多，但是也有本质的区别。

其中，synchronized 是利用锁的机制，使变量或代码块在某一时刻只能被一个线程访问。而 ThreadLocal 为每一个线程都提供了变量的副本，使得每个线程在某一时间访问到的并不是同一个对象，这样就隔离了多个线程对数据的数据共享。而 synchronized 正好相反，他用于多个线程间通信时能够获得数据的共享。

synchronized 用于线程间的数据共享，而 threadlocal 则用于线程间的数据隔离

ThreadLocal 中定义了一个哈希表用于为每个线程都提供一个变量的副本

7.final 修饰类、方法、属性！但是不能修饰抽象类，因为抽象类一般是需要被继承的，final 修饰后就不能继承了，final 修饰的方法不能被重写而不是重载！final 修饰属性，此属性就是一个常量，不能被再次赋值！

8.类的声明只能是 public、abstract 和 final，不能是私有的 private 的

9.静态方法不能使用 this 关键字
10.构造方法可以有任何访问的修饰，如 public、protected、private 或者没有修饰符，但是构造器不能用：abstract、final、native、static 和 synchronized 修饰。在构造器中，如果要使用 this，必须放在构造器的第一行；构造器不能被继承，子类可以继承父类的任何方法，且构造器中构造自己是不对的；当一个对象被创建时，初始化的顺序如下：
1）设置成员的值为默认的初始值（0，false,null）
2）调用对象的构造方法（但是还没有执行构造方法体）
3）调用父类的构造方法
4）使用初始化程序和初始块初始化成员
5）执行构造方法体
构造方法总是默认调用 super()方法，除非第一行是 super(args),this(),this(args)，如果父类中没有默认的构造方法，必须明确使用 super(args)调用父类的某个构造方法。
初始化顺序：父类静态变量->子类静态变量->父类非静态变量->父类静态代码快->父类构造函数->子类非静态变量->子类静态代码快->子类构造函数
11.switch 中的表达式必须为 byte、short、int 或 char 类型。每个 case 语句后的值 value 必须是与表达式类型兼容的特定的一个常量。case 后面必须为 byte、short、int 或 char 类型，不能为 String 类型，default 语句后面必须加上冒号
12.if 条件必须是一个布尔类型的表达式，而不能是一个整数表达式
13.for(f1;f2;f3)的执行顺序，首先 f1，在 f2，f3，如果 f2 满足条件，继续 f2,f3，其中 f1 只会执行一次
14.如果在函数中创建了异常，并抛出，则该函数可以不返回值
15.关于 break 和 continue 的描述
1)简答的一个 continue 会退回最内层循环的开头（顶部）,并继续执行
2)带有标签的 continue 会到达标签的位置，并重新进入紧接在那个标签后面的循环
3)break 会中断当前循环，并移离当前标签的末尾
4)代表前的 break 会中断当前循环，并移离由那个标签指示的循环的末尾
16.在 try 或者 catch 中的 return 语句，而在 finally 中又出现 return 语句，那么这种情况下不要期待 try 或者 catch 中的 return 语句返回值给上级调用函数获取到，上级调用函数获取到的只是 finally 中的返回值，因为 try 和 catch 中的 return 语句知识转移控制权的作用
17.基本数据类型和 String 类型时值传递，数组和对象时引用传递，引用类型作为形参传递会改变实参的值。
18.普通的 java 对象是通过 new 关键字把对应类的字节码文件加载到内存，然后创建该对象的。反射是通过一个名为 Class 的特殊类，用 Class.forName("className")得到类的字节码对象，然后用 newInstance()方法在虚拟机内部构造这个对象（针对无参构造函数）。也就是说反射机制让我们可以先拿到 java 类对应的字节码对象，然后动态的进行任何可能的操作，包括：
-在运行时判断任意一个对象所属的类
-在运行时构造任意一个类的对象
-在运行时判断任意一个类所具有的成员变量和方法
-在运行时调用任意一个对象的方法

19.环境变量可在编译 source code 时指定；javac 一次可同时编译数个 Java 源文件；javac.exe 能指定编译结果要置于哪个目录(directory)
20.如果两个 Integer 类型进行==比较，就是比较他们的地址，但是 int 类型和 integer 进行比较，Integer 会自动拆箱成 int 类型，在比较
21.volatile 简述：Java 语言提供了一种稍弱的同步机制，即 volatile 变量，用来确保将变量的更新操作通知到其他线程，保证了新值能立即同步到主内存，以及每次使用前立即从主内存刷新，当把变量声明为 volatile 类型后，编译器与运行时都会注意到这个变量是共享的。且 volatile 是线程不安全的

9 月 22 号

1.HttpSession session = request.getSession()等同于 request.getSession(true)，如果不存在就创建一个新的 session。HttpSession session = request.getSession(false)不创建 session
2.关于原生数据类型 1)byte 默认值为 0，一个字节，范围是-2^7~2^7-1
2)short 默认值为 0，两个字节，范围-2^15—2^15-1
3)int 默认值为 0，四个字节，范围：-2^31—2^31-1
4)long 默认值为 0，八个字节，范围：-2^63—2^63-1
5)float 默认值为 0.0f 四个字节，范围：-2^31—2^31-1
6)double 默认为 0.0d 八个字节，范围：-2^63—2^63-1
7)char 默认为'\u0000' 两个字节，范围：0—2^16-1
8)boolean 默认值为 false 范围：true\false
其中，char 类型占 16 位，没有负值，所以最小值是 0，最大值是 1111111111111111 = 2^16 - 1
3.获取 ServletContext 设置的参数值：context.getInitParameter();
4.正则表达式：.*?是非贪婪匹配的意思，表示找到最小的就可以了
(?=Expression)顺序环视，(?=\\()就是匹配正括号
5.Java 不允许单独的方法、过程或函数存在，需要隶属于某一类；Java 语言中的方法属于对象的成员，而不是类的成员。不过静态方法属于类的成员
6.String 类保存字符串只是保存所有单单的字符串，而 char[]字符数组会在最后自动加上'\n'。
7.

```
enum AccountType
{
    SAVING, FIXED, CURRENT;
    private AccountType()
    {
        System.out.println(“It is a account type”);
    }
}
class EnumOne
{
    public static void main(String[]args)
    {
        System.out.println(AccountType.FIXED);
    }
}
```

枚举类有三个实例，故调用三次构造方法，打印三次 It is a account type，然后再打印出 FIXED

8.以下 JSP 代码中定义了一个变量，如何输出呢？

```
 <bean:define id="stringBean" value="helloWorld"/>
1)<bean:write name="stringBean">
2)<%String
myBean=(String)pageContext.getAttribute("stringBean",PageContext.PAGE_SCOPE)
;%>
```

9.静态内部类中才能定义静态方法。

10.

```
List list1 = new ArrayList();
list1.add(0);
List list2 = list1;
System.out.println(list1.get(0) instanceof Integer);
System.out.println(list2.get(0) instanceof Integer);
```

List 没有规定数据类型，任何类型数据加入到 List 中后都会默认为 Object 类型，当再要拿出时就要进行强制类型转换，否则编译不通过。所以 list1.get(0)应该是一个 Object 对象，这里理解为 Object 对象是任何对象类型的实例。

11.

```
public class Test2
{
    public void add(Byte b)
    {
        b = b++;
    }
    public void test()
    {
        Byte a = 127;
        Byte b = 127;
        add(++a);
        System.out.print(a + " ");
        add(b);
        System.out.print(b + "");
    }
}
```

这里涉及到 java 的自动装包/自动拆包问题，Byte 的首字母为大写，是类，看似是引用传递，但是在 add 函数里实现了++操作，会自动拆包成 byte 值传递类型，所以 add 函数还是不能实现自增功能。也就是说 add 函数只是个摆设，没有任何作用。Byte 类型值的大小为-128~127 之间。add(++a)这里++a 会越界，a 的值变为-128，add(b)前面说了，add 不起任何作用，b 还是 127

12.关于线程的问题：run()方法是用来执行线程体中具体的内容；start()方法用来启动线程对象，使其进入就绪状态；sleep()方法用来使线程进入睡眠状态，在此期间线程不消耗 CPU 资源；suspend()方法使线程挂起，暂停执行，想恢复线程，必须由其他线程调用 resume()方法。

13.表达式的数据类型自动提升，规则如下：

1)所有的 byte、short、char 类型的值将被提升为 int 型，即操作数中没有 long、float、double 类型的，操作数将被转换成 int 类型

2)如果有一个操作数是 long 型的，计算结果是 long 型
3)如果有一个操作数是 float 型的，计算结果是 float 型
4)如果有一个操作数是 double 型，计算结果是 double 型
byte->short->int->long，int->double，char->int 都是低级向高级的，自动转换，高级向低级的转换则需要强制类型转换
14.Java 用监视器机制实现了进程之间的异步执行
15.关于静态的说法：类的静态成员与类直接相关，与对象无关，在一个类的所有实例之间共享同一个静态成员；静态成员函数中不能调用非静态成员；非静态成员函数中可以调用静态成员；常量成员不能修改，静态成员变量必须初始化，但可以修改。
16.覆盖方法满足的约束：
1)子类方法的名称、参数名和返回类型必须与父类的一致
2)子类方法不能缩小父类方法的访问权限
3)子类方法不能抛出比父类方法更多的异常。子类抛出的异常必须和父类方法的相同，或者是父类方法抛出异常类的子类
4)父类的静态方法不能被子类覆盖为非静态方法
5)子类可以定义与父类的静态方法同名的静态方法，以便在子类中隐藏父类的静态方法。其中：子类的参数名和返回类型必须和父类一致，不能缩小父类方法的访问权限，不能抛出更多的异常。
6)父类的非静态方法不能被子类覆盖为静态方法
7)父类的私有方法不能被子类覆盖
8)父类的抽象方法可以被子类通过两种图像覆盖：1）一种是子类实现父类的抽象方法；2）一种是子类重新声明父类的抽象方法
9)父类的非抽象方法可以被覆盖为抽象方法
17.重载和覆盖
相同点：1）都要求方法同名　2）都可以用于抽象方法和非抽象方法
不同点：1）覆盖要求参数必须一直，重载要求参数必须不一致 2）覆盖要求返回类型必须一直，重载对此不做限制 3）覆盖只能用于子类覆盖父类的方法，重载用于同一个类的所有方法 4）覆盖对访问权限和抛出的异常有特殊要求，重载则没要求 5）父类的一个方法只能被子类覆盖一次，而一个方法在所在的类中可以被重载多次
18.子类继承父类并不集成构造函数
19.父类中没有默认的构造函数，这就意味着子类中的构造函数必须显示地调用父类中带有参数的构造函数；所以，在子类的构造函数中，应该使用 super(num)来完成；如果父类中没有声明构造函数，隐含的默认构造函数会调用 super()。只要父类中有一个默认的构造函数，super()和 this()语句就不是强制的。构造函数默认的第一条语句是 super()。一个构造函数中不能同时出现 this()和 super()。调用 super()不一定能正常工作，因为父类中可能没有默认的构造函数

9 月 24

1.当 JVM 的拦截收集器调用一个合适对象的 finalize()方法时，它会忽略任何由 finalize()方法抛出的异常。其他情况下，finalize()方法中的异常处理同普通方法处理异常是一样的。Object 对象有一个 finalize()方法，由于所有的类都是 Object 类继承来的，因此，所有的对象都有一个 finalize()方法。类可以覆盖 finalize()

方法，而且和普通的方法覆盖规则一样，不能降低 finalize()方法的可访问性。调用 finalize()方法本身不会破坏该对象。
2.Java 初始化顺序：1)继承体系的所有静态成员初始化(先父类，后子类)。2)父类初始化完成(普通成员的初始化-->构造函数的调用)。3)子类初始化(普通成员-->构造函数)
3.线程和进程的主要差别在于：他们是不同的操作系统资源管理方式。
进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其他进程产生影响，而线程只是一个进程中不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程
4.Thread 类实现了 Runnable 接口，不是抽象类；当最后一个非后台线程结束时，程序也就终止了；Runnable 接口有一个 run()方法，不过该接口没有规定必须定义一个 start()方法。在一个 Runnable 对象上调用 run()方法无需创建新线程；run()是由线程执行的方法。必须创建 Thread 类的实例，以生成大量的新线程。
5.为了确保可以在线程之间以受控制方式共享数据，Java 提供了两个关键字：synchronized 和 volatile
1)synchronized：一次只有一个线程可以执行代码的受保护部分；一个线程更改的数据对于其他线程是可见的
2)volatile 比同步更简单，只适合于控制对基本变量的单个实例的访问，当一个变量被声明为 volatile，任何对该变量的写操作都会绕过高速缓存，直接写入主内存，而任何对该变量的读取也将绕过高速缓存，直接取主内存。