极客学院 面试宝典

Java 面试宝典

有人说 Java 是互联网编程领域的语言之王，不管这种说法是否言过其辞，但 Java 在互联网中的应用范围之广确是不争的事实。程序员在面试 Java 开发岗位时经常会遇上各类笔试面试题，针对这些笔试面试题做准备是必要的环节，在此过程中也能加深对 Java 知识的理解。

本面试宝典的试题均来自于网上。我们按照 Java 知识点的分类整理出 17 章共 204 道笔试题，每道题都有解答，希望读者可以通过这些题目快速准备笔试面试并获得最终的求职成功。

需要说明的是，在选取笔试题时，我们尽力选取清晰、简练的题目以节省读者的时间，如果想深入理解某方面的知识，还需要阅读 《Thinking in Java》等经典教材。

适用人群

本课程主要适用于为 Java 开发职位准备的软件开发人员。

学习前提

在学习本课程之前，我们假定你已经是一名初级 Java 程序员，对 Java 有最基本的了解。

Java 基本概念

1. Java 语言的优点?

简单、高效

Java 语言与 C++ 类似，如果用户了解 C++ 和面向对象的概念，就可以很快编写出 Java 程序；此外，Java 又不同于诸如 C++ 语言提供的各种各样的方法，它只提供了基本的方法，这样就减少了编程的复杂性，例如去掉了头文件、指针变量、结构、运算符重载、多重继承等复杂特性。Java 语言虽然简单，却很高效，它可以用面向对象的方法来描述用户的每一个动作。

面向对象

Java 语言是一种新的面向对象的程序设计语言，它除了几种基本的数据类型外，大都是类似 C++ 中的对象和方法，程序代码大多体现了类机制，以类的形式组织，由类来定义对象的各种行为。Java 同样支持类继承，这样也减少了程序设计的复杂性。

平台无关性

所谓一处编译处处运行。Java 语言经编译后生成与计算机硬件结构无关的字节代码（ Bytecode ），这些字节代码被定义为不依赖任何硬件平台和操作系统。当 Java 程序在运行时，需要由一个解释程序对生成的字节代码解释执行。这体现了 Java 语言的与平台无关性，使得 Java 程序可以在任何平台上运行，如 MS-DOS，Windows，Unix 等，因此具有很强的移植性。

交互式特性

Java 是面向对象的网络编程语言，由于它支持 TCP / IP 协议，使得用户可以通过浏览器访问到 Internet 上的各种动态对象。并且在网络上用户可以交互式地进行各种动作，而多线程技术的引入使得这种交互式操作更为容易。

多线程机制

Java 语言支持多线程机制，多线程机制使得 Java 程序能够并行处理多项任务。Java 程序可以设计成具有多个线程，例如让一个线程负责数据的检索、查寻，另一个线程与用户进行交互，这样，两个线程得以并行执行。多线程机制可以很容易地实现网络上的交互式操作。

动态的内存管理机制

Java 语言采用了自动垃圾回收机制进行内存的管理。在 C++ 语言中，程序员在编写程序时要仔细地处理内存的使用，例如当某个内存快使用完毕时，要及时释放，以供其它程序使用，一旦内存管理不当，就有可能造成内存空间浪费或程序运行故障。 在 Java 系统中包括了一个自动垃圾回收程序，它可以自动、安全地回收不再使用的内存块，这样，程序员在编程时就无需担心内存的管理问题，从而使 Java 程序的编写变得简单，同时也减少了内存管理方面出错的可能性。

安全性

在类似 Internet 的这种分布式网络环境中，安全性是个不容忽视的问题。Java 语言在安全性方面引入了实时内存分配及布局来防止程序员直接修改物理内存布局；通过字节代码验证器对字节代码的检验，以防止网络病毒及其它非法代码侵入。此外，Java 语言还采用了许多面向对象的异常处理机制，负责对一些异常事件进行处理，如内存空间不够，程序异常中止等的处理

Java 是解释型的 运行 Java 程序需要一个解释器。Java 程序编译成 Java 虚拟机编码，这种编码称为字节码。字节码是独立于计算机的，他能在所有具有 Java 解释器的机器上运行，Java 解释器也是 JVM 的一部分。

Java 的效率

早期 Sun 发现 JVM 的用户注意到了 Java 运行很慢，但是新的 Java 虚拟机显著加快。新的虚拟机使用了一种称为实时编译的技术，他把字节码编译并储存成本机的机器码。当执行字节码时，重调本地码。

2. 什么是 Java 虚拟机？为什么 Java 被称作是"平台无关的编程语言"？

Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机执行的字节码文件。

Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java 虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

3. Java 和 C++ 有何区别?

Java 和 C++ 都是面向对象语言。也就是说，它们都能够实现面向对象思想（封装，继乘，多态）。而由于 C++ 为了照顾大量的 C 语言使用者，而兼容了 C ，使得自身仅仅成为了带类的 C 语言，多多少少影响了其面向对象的彻底性！Java 则是完全的面向对象语言，它句法

更清晰，规模更小，更易学。它是在对多种程序设计语言进行了深入细致研究的基础上，据弃了其他语言的不足之处，从根本上解决了 C++ 的固有缺陷。

Java 和 C++ 的相似之处多于不同之处，但两种语言问几处主要的不同使得 Java 更容易学习，并且编程环境更为简单。我在这里不能完全列出不同之处，仅列出比较显著的区别：

指针

Java 语言让编程者无法找到指针来直接访问内存无指针，并且增添了自动的内存管理功能，从而有效地防止了 C/C++ 语言中指针操作失误，如野指针所造成的系统崩溃。但也不是说 Java 没有指针，虚拟机内部还是使用了指针，只是外人不得使用而已。这有利于 Java 程序的安全。

多重继承

C++ 支持多重继承，这是 C++ 的一个特征，它允许多父类派生一个类。尽管多重继承功能很强，但使用复杂，而且会引起许多麻烦，编译程序实现它也很不容易。Java 不支持多重继承，但允许一个类继承多个接口 (extends+implement) ，实现了 C++ 多重继承的功能，又避免了 C++ 中的多重继承实现方式带来的诸多不便。

数据类型及类

Java 是完全面向对象的语言，所有函数和变量部必须是类的一部分。除了基本数据类型之外，其余的都作为类对象，包括数组。对象将数据和方法结合起来，把它们封装在类中，这样每个对象都可实现自己的特点和行为。而 C++ 允许将函数和变量定义为全局的。此外，Java 中取消了 C/C++ 中的结构和联合，消除了不必要的麻烦。

自动内存管理

Java 程序中所有的对象都是用 new 操作符建立在内存堆栈上，这个操作符类似于 C++ 的 new 操作符。下面的语句由一个建立了一个类 Read 的对象，然后调用该对象的 work 方法：

```
Read r=new Read();
r.work();
```

语句 `Read r=new Read()` ；在堆栈结构上建立了一个 Read 的实例。Java 自动进行无用内存回收操作，不需要程序员进行删除。而 C++ 中必须由程序员释放内存资源，增加了程序设计者的负担。Java 中当一个对象不被再用到时，无用内存回收器将给它加上标签以示删除。Java 里无用内存回收程序是以线程方式在后台运行的，利用空闲时间工作。

操作符重载

Java 不支持操作符重载。操作符重载被认为是 C++ 的突出特征，在 Java 中虽然类大体上可以实现这样的功能，但操作符重载的方便性仍然丢失了不少。Java 语言不支持操作符重载是为了保持 Java 语言尽可能简单。

预处理功能

Java 不支持预处理功能。C/C++ 在编译过程中都有一个预编泽阶段，即众所周知的预处理器。预处理器为开发人员提供了方便，但增加了编译的复杂性。Java 虚拟机没有预处理器，但它提供的引入语句 (import) 与 C++ 预处理器的功能类似。

Java 不支持缺省函数参数，而 C++支持

在 C 中，代码组织在函数中，函数可以访问程序的全局变量。C++ 增加了类，提供了类算法，该算法是与类相连的函数，C++ 类方法与 Java 类方法十分相似，然而，由于 C++ 仍然支持 C ，所以不能阻止 C++ 开发人员使用函数，结果函数和方法混合使用使得程序比较混乱。

Java 没有函数，作为一个比 C++ 更纯的面向对象的语言，Java 强迫开发人员把所有例行程序包括在类中，事实上，用方法实现例行程序可激励开发人员更好地组织编码。

字符串

C 和 C++ 不支持字符串变量，在 C 和 C++ 程序中使用 Null 终止符代表字符串的结束，在 Java 中字符串是用类对象（ string 和 stringBuffer)来实现的，这些类对象是 Java 语言的核心，用类对象实现字符串有以下几个优点：

(1)在整个系统中建立字符串和访问字符串元素的方法是一致的； (2)Java 字符串类是作为 Java 语言的一部分定义的，而不是作为外加的延伸部分； (3)Java 字符串执行运行时检查，可帮助排除一些运行时发生的错误； (4)可对字符串用“+”进行连接操作。

goto 语句

“可怕”的 goto 语句是 C 和 C++ 的“遗物”，它是该语言技术上的合法部分，引用 goto 语句引起了程序结构的混乱，不易理解，goto 语句子要用于无条件转移子程序和多结构分支技术。鉴于以广理由，Java 不提供 goto 语句，它虽然指定 goto 作为关键字，但不支持它的使用，使程序简洁易读。

类型转换

在 C 和 C++ 中有时出现数据类型的隐含转换，这就涉及了自动强制类型转换问题。例如，在 C++ 中可将一浮点值赋予整型变量，并去掉其尾数。Java 不支持 C++ 中的自动强制类型转换，如果需要，必须由程序显式进行强制类型转换。

异常

Java 中的异常机制用于捕获例外事件，增强系统容错能力

```
try{

   //可能产生意外的代码
}catch(exceptionType name){
    //处理
}
```

其中 exceptionType 表示异常类型。而 C++ 则没有如此方便的机制。

4. JDK 和 JRE 的区别是什么？

Java 运行时环境(JRE)是将要执行 Java 程序的 Java 虚拟机。它同时也包含了执行 Applet 需要的浏览器插件。Java 开发工具包(JDK)是完整的 Java 软件开发包，包含了 JRE ，编译器和其他的工具(比如： JavaDoc，Java 调试器)，可以让开发者开发、编译、执行 Java 应用程序。

5. Java 支持的数据类型有哪些？什么是自动拆装箱？

Java 语言支持的 8 中基本数据类型是：

- byte
- short
- int
- long
- float
- double
- boolean
- char

自动装箱是 Java 编译器在基本数据类型和对应的对象包装类型之间做的一个转化。比如：把 int 转化成 Integer ，double 转化成 Double，等等。反之就是自动拆箱。

6. 什么是值传递和引用传递？

对象被值传递，意味着传递了对象的一个副本。因此，就算是改变了对象副本，也不会影响源对象的值。

对象被引用传递，意味着传递的并不是实际的对象，而是对象的引用。因此，外部对引用对象所做的改变会反映到所有的对象上。

7. 一个 ".java" 源文件中是否可以包括多个类（不是内部类）？有什么限制？

可以有多个类，但只能有一个 public 的类，并且 public 的类名必须与文件名相一致。

8. 静态变量和实例变量的区别？

在语法定义上的区别：静态变量前要加 static 关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

例如，对于下面的程序，无论创建多少个实例对象，永远都只分配了一个 staticVar 变量，并且每创建一个实例对象，这个 staticVar 就会加 1 ；但是，每创建一个实例对象，就会分配一个 instanceVar ，即可能分配多个 instanceVar ，并且每个 instanceVar 的值都只自加了 1 次。

```
 public class VariantTest{
    public static int staticVar = 0;
    public int instanceVar = 0;
    public VariantTest(){
        staticVar++;
        instanceVar++;
        System.out.println(“staticVar=” + staticVar + ”,instanceVar=” + in
stanceVar);
        }
 }
```

9. 不通过构造函数也能创建对象吗?

A. 是

B. 否

答案：A

解析：Java 创建对象的几种方式（重要）：

(1) 用 new 语句创建对象，这是最常见的创建对象的方法。 (2) 运用反射手段,调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance() 实例方法。 (3) 调用对象的 clone() 方法。 (4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject() 方法。

(1)和(2)都会明确的显式的调用构造函数 ；(3)是在内存上对已有对象的影印，所以不会调用构造函数 ；(4)是从文件中还原类的对象，也不会调用构造函数。

10. 静态变量和实例变量的区别？

答：静态变量是被 static 修饰符修饰的变量，也称为类变量，它属于类，不属于类的任何一个对象，一个类不管创建多少个对象，静态变量在内存中有且仅有一个拷贝；实例变量必须依存于某一实例，需要先创建对象然后通过对象才能访问到它。静态变量可以实现让多个对象共享内存。在 Java 开发中，上下文类和工具类中通常会有大量的静态成员。

11. 是否可以从一个静态（static）方法内部发出对非静态（non-static）方法的调用？

答：不可以，静态方法只能访问静态成员，因为非静态方法的调用要先创建对象，因此在调用静态方法时可能对象并没有被初始化。

12. 如何实现对象克隆？

答：有两种方式：

1.实现 Cloneable 接口并重写 Object 类中的 clone() 方法；

2.实现 Serializable 接口，通过对象的序列化和反序列化实现克隆，可以实现真正的深度克隆。

13. 一个“.java”源文件中是否可以包含多个类（不是内部类）？有什么限制？

答：可以，但一个源文件中最多只能有一个公开类（public class）而且文件名必须和公开类的类名完全保持一致。

14. Anonymous Inner Class(匿名内部类)是否可以继承其它类？是否可以实现接口？

答：可以继承其他类或实现其他接口，在 Swing 编程中常用此方式来实现事件监听和回调。

15. 内部类可以引用它的包含类（外部类）的成员吗？有没有什么限制？

答：一个内部类对象可以访问创建它的外部类对象的成员，包括私有成员

16. 列出自己常用的 jdk 包.

答：JDK 常用的 package

java.lang：这个是系统的基础类，比如 String 等都是这里面的，这个 package 是唯一一个可以不用 import 就可以使用的 Package

java.io: 这里面是所有输入输出有关的类，比如文件操作等

java.net: 这里面是与网络有关的类，比如 URL,URLConnection 等。

java.util: 这个是系统辅助类，特别是集合类 Collection,List,Map 等。

java.sql: 这个是数据库操作的类，Connection, Statememt，ResultSet 等

17. JDK, JRE 和 JVM 的区别？

JDK, JRE 和 JVM 是 Java 编程语言的核心概念。尽管它们看起来差不多，作为程序员我们也不怎么关心这些概念，但是它们是不同的针对特定目的的产品。这是一道常见的 Java 面试题，而本文则会一一解释这些概念并给出它们之间的区别。

Java 开发工具包 (JDK)

Java 开发工具包是 Java 环境的核心组件，并提供编译、调试和运行一个 Java 程序所需的所有工具，可执行文件和二进制文件。JDK 是一个平台特定的软件，有针对 Windows，Mac 和 Unix 系统的不同的安装包。可以说 JDK 是 JRE 的超集，它包含了 JRE 的 Java 编译器，调试器和核心类。目前 JDK 的版本号是 1.7，也被称为 Java 7。

Java 虚拟机(JVM)

JVM 是 Java 编程语言的核心。当我们运行一个程序时，JVM 负责将字节码转换为特定机器代码。JVM 也是平台特定的，并提供核心的 Java 方法，例如内存管理、垃圾回收和安全机制等。JVM 是可定制化的，我们可以通过 Java 选项(java options)定制它，比如配置 JVM 内存的上下界。JVM 之所以被称为虚拟的是因为它提供了一个不依赖于底层操作系统和机器硬件的接口。这种独立于硬件和操作系统的特性正是 Java 程序可以一次编写多处执行的原因。

Java 运行时环境(JRE)

JRE 是 JVM 的实施实现，它提供了运行 Java 程序的平台。JRE 包含了 JVM、Java 二进制文件和其它成功执行程序的类文件。JRE 不包含任何像 Java 编译器、调试器之类的开发工具。如果你只是想要执行 Java 程序，你只需安装 JRE 即可，没有安装 JDK 的必要。

JDK, JRE 和 JVM 的区别

JDK 是用于开发的而 JRE 是用于运行 Java 程序的。
JDK 和 JRE 都包含了 JVM，从而使得我们可以运行 Java 程序。
JVM 是 Java 编程语言的核心并且具有平台独立性。

即时编译器(JIT)

有时我们会听到 JIT 这个概念，并说它是 JVM 的一部分，这让我们很困惑。JIT 是 JVM 的一部分，它可以在同一时间编译类似的字节码来优化将字节码转换为机器特定语言的过程相似的字节码，从而将优化字节码转换为机器特定语言的过程，这样减少转换过程所需要花费的时间。

面向对象编程

1. Java 中的方法覆盖 (Overriding) 和方法重载 (Overloading) 是什么意思？

Java 中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型。覆盖者可能不会限制它所覆盖的方法的访问。

2. Overload 和 Override 的区别？ Overloaded 的方法是否可以改变返回值的类型?

Overload 是重载的意思，Override 是覆盖的意思，也就是重写。

重载 Overload 表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写 Override 表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

至于 Overloaded 的方法是否可以改变返回值的类型这个问题，要看你倒底想问什么呢？这个题目很模糊。如果几个 Overloaded 的方法的参数列表不一样，它们的返回者类型当然也可以不一样。但我估计你想问的问题是：如果两个方法的参数列表完全一样，是否可以让它们

的返回值不同来实现重载 Overload。这是不行的，我们可以用反证法来说明这个问题，因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用 `map.remove(key)` 方法时，虽然 remove 方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同，Java 就无法确定编程者倒底是想调用哪个方法了，因为它无法通过返回结果类型来判断。

Override 可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外，我们在继承中也可能会在子类覆盖父类中的方法。在覆盖要注意以下的几点：

1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；

2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；

3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；

4、被覆盖的方法不能为 private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

Overload 对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM 就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是 `fun(int,float)`，但是不能 `fun(int,int)`）；2、不能通过访问权限、返回类型、抛出的异常进行重载；
3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是 priavte，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

3. Java 中，什么是构造函数？什么是构造函数重载？什么是复制构造函数？

当新对象被创建的时候，构造函数会被调用。每一个类都有构造函数。在程序员没有给类提供构造函数的情况下，Java 编译器会为这个类创建一个默认的构造函数。

Java 中构造函数重载和方法重载很相似。可以为一个类创建多个构造函数。每一个构造函数必须有它自己唯一的参数列表。

Java 不支持像 C++ 中那样的复制构造函数，这个不同点是因为如果你不自己写构造函数的情况下，Java 不会创建默认的复制构造函数。

4. 构造器 Constructor 是否可被 Override?

构造器 Constructor 不能被继承，因此不能重写 Override，但可以被重载 Overload。

5. Java 支持多继承么？

不支持，Java 不支持多继承。每个类都只能继承一个类，但是可以实现多个接口。

6. 接口和抽象类的区别是什么？

Java 提供和支持创建抽象类和接口。它们的实现有共同点，不同点在于：

接口中所有的方法隐含的都是抽象的。而抽象类则可以同时包含抽象和非抽象的方法。

类可以实现很多个接口，但是只能继承一个抽象类

类如果要实现一个接口，它必须要实现接口声明的所有方法。但是，类可以不实现抽象类声明的所有方法，当然，在这种情况下，类也必须得声明成是抽象的。

抽象类可以在不提供接口方法实现的情况下实现接口。

Java 接口中声明的变量默认都是 final 的。抽象类可以包含非 final 的变量。

Java 接口中的成员函数默认是 public 的。抽象类的成员函数可以是 private，protected 或者是 public 。

接口是绝对抽象的，不可以被实例化。抽象类也不可以被实例化，但是，如果它包含 main 方法的话是可以被调用的。

也可以参考 JDK8 中抽象类和接口的区别

7. 下列说法正确的有（）

A．class 中的 constructor 不可省略

B．constructor 必须与 class 同名，但方法不能与 class 同名

C．constructor 在一个对象被 new 时执行

D．一个 class 只能定义一个 constructor

答案：C

解析：这里可能会有误区，其实普通的类方法是可以和类名同名的，和构造方法唯一的区分就是，构造方法没有返回值。

8. Java 接口的修饰符可以为?

A. private

B. protected

C. final

D. abstract

答案：CD

解析：接口很重要，为了说明情况，这里稍微啰嗦点：

（1）接口用于描述系统对外提供的所有服务,因此接口中的成员常量和方法都必须是公开(public)类型的,确保外部使用者能访问它们；

（2）接口仅仅描述系统能做什么,但不指明如何去做,所以接口中的方法都是抽象(abstract)方法；

（3）接口不涉及和任何具体实例相关的细节,因此接口没有构造方法,不能被实例化,没有实例变量，只有静态（static）变量；

（4）接口的中的变量是所有实现类共有的，既然共有，肯定是不变的东西，因为变化的东西也不能够算共有。所以变量是不可变(final)类型，也就是常量了。

（5） 接口中不可以定义变量？如果接口可以定义变量，但是接口中的方法又都是抽象的，在接口中无法通过行为来修改属性。有的人会说了，没有关系，可以通过 实现接口的对象的行为来修改接口中的属性。这当然没有问题，但是考虑这样的情况。如果接口 A 中有一个 public 访问权限的静态变量 a。按照 Java 的语义，我们可以不通过实现接口的对象来访问变量 a，通过 A.a = xxx; 就可以改变接口中的变量 a 的值了。正如抽象类中是可以这样做的，那么实现接口 A 的所有对象也都会自动拥有这一改变后的 a 的值了，也就是说一个地方改变了 a，所有这些对象中 a 的值也都跟着变了。这和抽象类有什么区别呢，怎么体现接口更高的抽象级别呢，怎么体现接口提供的统一的协议呢，那还要接口这种抽象来做什么呢？所以接口中 不能出现变量，如果有变量，就和接口提供的统一的抽象这种思想是抵触的。所以接口中的属性必然是常量，只能读不能改，这样才能为实现接口的对象提供一个统 一的属性。

通俗的讲，你认为是要变化的东西，就放在你自己的实现中，不能放在接口中去，接口只是对一类事物的属性和行为更高层次的抽象。对修改关闭，对扩展（不同的实现 implements）开放，接口是对开闭原则的一种体现。

所以：

接口的方法默认是 public abstract；

接口中不可以定义变量即只能定义常量(加上 final 修饰就会变成常量)。所以接口的属性默认是 public static final 常量，且必须赋初值。

注意：final 和 abstract 不能同时出现。

9. 下面是 People 和 Child 类的定义和构造方法，每个构造方法都输出编号。在执行 new Child("mike") 的时候都有哪些构造方法被顺序调用？请选择输出结果

```
class People {
    String name;

    public People() {
        System.out.print(1);
    }

    public People(String name) {
        System.out.print(2);
        this.name = name;
    }
}

class Child extends People {
    People father;

    public Child(String name) {
        System.out.print(3);
        this.name = name;
        father = new People(name + ":F");
    }

    public Child() {
        System.out.print(4);
    }

}
```

A. 312

B. 32

C. 432

D. 132

答案：D

解析：考察的又是父类与子类的构造函数调用次序。在 Java 中，子类的构造过程中必须调用其父类的构造函数，是因为有继承关系存在时，子类要把父类的内容继承下来。但如果父类有多个构造函数时，该如何选择调用呢？

第一个规则：子类的构造过程中，必须调用其父类的构造方法。一个类，如果我们不写构造方法，那么编译器会帮我们加上一个默认的构造方法（就是没有参数的构造方法），但是如果你自己写了构造方法，那么编译器就不会给你添加了，所以有时候当你 new 一个子类对象的时候，肯定调用了子类的构造方法，但是如果在子类构造方法中我们并没有显示的调用基类的构造方法，如：super(); 这样就会调用父类没有参数的构造方法。

第二个规则：如果子类的构造方法中既没有显示的调用基类构造方法，而基类中又没有无参的构造方法，则编译出错，所以，通常我们需要显示的：super(参数列表)，来调用父类有参数的构造函数，此时无参的构造函数就不会被调用。

总之，一句话：子类没有显示调用父类构造函数，不管子类构造函数是否带参数都默认调用父类无参的构造函数，若父类没有则编译出错。

10. 构造器（constructor）是否可被重写（override）？

答：构造器不能被继承，因此不能被重写，但可以被重载。

11. 两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对？

答：不对，如果两个对象 x 和 y 满足 x.equals(y) == true，它们的哈希码（hash code）应当相同。Java 对于 eqauls 方法和 hashCode 方法是这样规定的：(1)如果两个对象相同（equals 方法返回 true ），那么它们的 hashCode 值一定要相同；(2)如果两个对象的 hashCode 相同，它们并不一定相同。当然，你未必要按照要求去做，但是如果你违背了上述原则就会发现在使用容器时，相同的对象可以出现在 Set 集合中，同时增加新元素的效率会大大下降（对于使用哈希存储的系统，如果哈希码频繁的冲突将会造成存取性能急剧下降）。

补充：关于 equals 和 hashCode 方法，很多 Java 程序都知道，但很多人也就是仅仅知道而已，在 Joshua Bloch 的大作《Effective Java》（很多软件公司，《Effective Java》、《Java 编程思想》以及《重构：改善既有代码质量》是 Java 程序员必看书籍，如果你还没

看过，那就赶紧去亚马逊买一本吧）中是这样介绍 equals 方法的：首先 equals 方法必须满足自反性（x.equals(x) 必须返回 true）、对称性（x.equals(y) 返回 true 时，y.equals(x) 也必须返回 true）、传递性（x.equals(y)和 y.equals(z)都返回 true 时，x.equals(z)也必须返回 true）和一致性（当 x 和 y 引用的对象信息没有被修改时，多次调用 x.equals(y)应该得到同样的返回值），而且对于任何非 null 值的引用 x，x.equals(null)必须返回 false。实现高质量的 equals 方法的诀窍包括：

1. 使用 == 操作符检查"参数是否为这个对象的引用"；
2. 使用 instanceof 操作符检查"参数是否为正确的类型"；
3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；
4. 编写完 equals 方法后，问自己它是否满足对称性、传递性、一致性；
5. 重写 equals 时总是要重写 hashCode；
6. 不要将 equals 方法参数中的 Object 对象替换为其他的类型，在重写时不要忘掉 @Override 注解。

12. 接口是否可继承（extends）接口? 抽象类是否可实现（implements）接口? 抽象类是否可继承具体类（concrete class）?

答：接口可以继承接口。抽象类可以实现(implements)接口，抽象类可继承具体类，但前提是具体类必须有明确的构造函数。

13. 指出下面程序的运行结果:

```
class A{

    static{
        System.out.print("1");
    }

    public A(){
        System.out.print("2");
    }
}

class B extends A{

    static{
        System.out.print("a");
    }

    public B(){
        System.out.print("b");
    }
}

public class Hello{

    public static void main(String[] args){
```

```
        A ab = new B();
        ab = new B();
    }

}
```

答：执行结果：1a2b2b。创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

14. Class.forName（String className）这个方法的作用

答：通过类的全名获得该类的类对象

15. 什么是 AOP 和 OOP，IOC 和 DI 有什么不同?

答:

1）面向对象编程（Object Oriented Programming，OOP，面向对象程序设计）是一种计算机编程架构。AOP 是 OOP 的延续，是 Aspect Oriented Programming 的缩写，意思是面向方面编程。 将通用需求功能从不相关类之中分离出来；同时，能够使得很多类共享一个行为，一旦行为发生变化，不必修改很多类，只要修改这个行为就可以。AOP 就是这种实现分散关注的编程方法，它将"关注"封装在"方面"中

2）控制反转 IOC(Inversion of Control) 控制指的就是程序相关类之间的依赖关系.传统观念设计中,

通常由调用者来创建被调用者的实例, 在 Spring 里,创建被调用者的工作不再由调用者来完成,而是由 Spring 容器完成，依赖关系被反转了，称为控制反转，目的是为了获得更好的扩展性和良好的可维护性。依赖注入(Dependency injection)创建被调用者的工作由 Spring 容器完成，然后注入调用者，因此也称依赖注入。控制反转和依赖注入是同一个概念。

16. 判断下列语句是否正确，如果有错误，请指出错误所在？

```
interface A{

int add(final A a);

}

class B implements A{

long add(final A a){

return this.hashCode() + a.hashCode();
```

```
}
}
```

答：返回值不是 long 类型

关键字

1. ”static” 关键字是什么意思？Java 中是否可以覆盖(override) 一个 private 或者是 static 的方法？

“static” 关键字表明一个成员变量或者是成员方法可以在没有所属的类的实例变量的情况下被访问。

Java 中 static 方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而 static 方法是编译时静态绑定的。static 方法跟类的任何实例都不相关，所以概念上不适用。

2. 是否可以在 static 环境中访问非 static 变量？

static 变量在 Java 中是属于类的，它在所有的实例中的值是一样的。当类被 Java 虚拟机载入的时候，会对 static 变量进行初始化。如果你的代码尝试不用实例来访问非 static 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟任何实例关联上。

3. 访问修饰符 public, private, protected, 以及不写（默认）时的区别？

修饰符	当前类	同 包	子 类	
public	√	√	√	
protected	√	√	√	
default	√	√	×	
private	√	×	×	

类的成员不写访问修饰时默认为 default 。默认对于同一个包中的其他类相当于公开（public），对于不是同一个包中的其他类相当于私有（private）。受保护（protected）对子类相当于公开，对不是同一包中的没有父子关系的类相当于私有。Java 中，外部类的修饰符只能是 public 或默认，类的成员（包括内部类）的修饰符可以是以上四种。

4. volatile 关键字是否能保证线程安全？

答案：不能

解析：volatile 关键字用在多线程同步中，可保证读取的可见性，JVM 只是保证从主内存加载到线程工作内存的值是最新的读取值，而非 cache 中。但多个线程对 volatile 的写操作，无法保证线程安全。例如假如线程 1，线程 2 在进行 read,load 操作中，发现主内存中 count 的值都是 5，那么都会加载这个最新的值，在线程 1 堆 count 进行修改之后，会 write 到主内存中，主内存中的 count 变量就会变为 6；线程 2 由于已经进行 read,load 操作，在进行运算之后，也会更新主内存 count 的变量值为 6；导致两个线程及时用 volatile 关键字修改之后，还是会存在并发的情况。

5. Java 有没有 goto?

答：goto 是 Java 中的保留字，在目前版本的 Java 中没有使用。（根据 James Gosling（Java 之父）编写的《The Java Programming Language》一书的附录中给出了一个 Java 关键字列表，其中有 goto 和 const，但是这两个是目前无法使用的关键字，因此有些地方将其称之为保留字，其实保留字这个词应该有更广泛的意义，因为熟悉 C 语言的程序员都知道，在系统类库中使用过的有特殊意义的单词或单词的组合都被视为保留字）

6. Java 中的 final 关键字有哪些用法？

答：

(1)修饰类：表示该类不能被继承；

(2)修饰方法：表示方法不能被重写；

(3)修饰变量：表示变量只能一次赋值以后值不能被修改（常量）。

7. 什么时候用 assert？

答：assertion(断言)在软件开发中是一种常用的调试方式，很多开发语言中都支持这种机制。一般来说，assertion 用于保证程序最基本、关键的正确性。assertion 检查通常在开发和测试时开启。为了提高性能，在软件发布后， assertion 检查通常是关闭的。在实现中，断言是一个包含布尔表达式的语句，在执行这个语句时假定该表达式为 true；如果表达式计算为 false，那么系统会报告一个 AssertionError。

断言用于调试目的：

```
assert(a > 0); // throws an AssertionError if a <= 0
```

断言可以有两种形式：

assert Expression1;

assert Expression1 ：Expression2 ；

Expression1 应该总是产生一个布尔值。

Expression2 可以是得出一个值的任意表达式；这个值用于生成显示更多调试信息的字符串消息。

断言在默认情况下是禁用的，要在编译时启用断言，需使用 source 1.4 标记：

```
javac -source 1.4 Test.java
```

要在运行时启用断言，可使用 -enableassertions 或者 -ea 标记。

要在运行时选择禁用断言，可使用 -da 或者 -disableassertions 标记。

8. final, finally, finalize 的区别?

答：

final：修饰符（关键字）有三种用法：如果一个类被声明为 final，意味着它不能再派生出新的子类，即不能被继承，因此它和 abstract 是反义词。将变量声明为 final，可以保证它们在使用中不被改变，被声明为 final 的变量必须在声明时给定初值，而在以后的引用中只能读取不可修改。被声明为 final 的方法也同样只能使用，不能在子类中被重写。

finally：通常放在 try…catch 的后面构造总是执行代码块，这就意味着程序无论正常执行还是发生异常，这里的代码只要 JVM 不关闭都能执行，可以将释放外部资源的代码写在 finally 块中。

finalize：Object 类中定义的方法，Java 中允许使用 finalize() 方法在垃圾收集器将对象从内存中清除出去之前做必要的清理工作。这个方法是由垃圾收集器在销毁对象时调用的，通过重写 finalize() 方法可以整理系统资源或者执行其他清理工作。

基本类型与运算

1. 说说 & 和 && 的区别。

& 和 && 都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。

&& 还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式，例如，对于 `if(str != null&& !str.equals(“”))` 表达式，当 str 为 null 时，后面的表达式不会执行，所以不会出现 NullPointerException 如果将 && 改为 & ，则会抛出

NullPointerException 异常。`If(x==33 & ++y>0) y` 会增长， `If(x==33 && ++y>0)` 不会增长。
& 还可以用作位运算符，当 & 操作符两边的表达式不是 boolean 类型时，& 表示按位与操作，我们通常使用 0x0f 来与一个整数进行 & 运算，来获取该整数的最低 4 个 bit 位，例如，0x31 & 0x0f 的结果为 0x01。

备注：这道题先说两者的共同点，再说出 && 和 & 的特殊之处，并列举一些经典的例子来表明自己理解透彻深入、实际经验丰富。

2. 用最有效率的方法算出 2 乘以 8 等於几?

2 << 3，因为将一个数左移 n 位，就相当于乘以了 2 的 n 次方，那么，一个数乘以 8 只要将其左移 3 位即可，而位运算 cpu 直接支持的，效率最高，所以，2 乘以 8 等於几的最效率的方法是 2 << 3 。

3. 存在使 i + 1 < i 的数吗?

答案：存在

解析：如果 i 为 int 型，那么当 i 为 int 能表示的最大整数时，`i+1` 就溢出变成负数了，此时不就 <i 了吗。
扩展：存在使 `i > j || i <= j` 不成立的数吗?
答案：存在

解析：比如 Double.NaN 或 Float.NaN 。

4. 0.6332 的数据类型是（）

A. float

B. double

C. Float

D. Double
答案：B

解析：默认为 double 型，如果为 float 型需要加上 f 显示说明，即 0.6332f

5. System.out.println("5" + 2);的输出结果应该是（）。

A. 52

B. 7

C. 2

D. 5

答案：A

解析：没啥好说的，Java 会自动将 2 转换为字符串。

6. 下面的方法，当输入为 2 的时候返回值是多少?

```
public static int getValue(int i) {
    int result = 0;
    switch (i) {
    case 1:
        result = result + i;
    case 2:
        result = result + i * 2;
    case 3:
        result = result + i * 3;
    }
    return result;
}
```

A. 0

B. 2

C. 4

D. 10

答案：D

解析：注意这里 case 后面没有加 break，所以从 case 2 开始一直往下运行。

7. float f=3.4;是否正确?

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）属于下转型（down-casting，也称为窄化）会造成精度损失，因此需要强制类型转换 float f =(float)3.4; 或者写成 float f =3.4F;。

8. int 和 Integer 有什么区别?

答：Java 是一个近乎纯洁的面向对象编程语言，但是为了编程的方便还是引入不是对象的基本数据类型，但是为了能够将这些基本数据类型当成对象操作，Java 为每一个基本数据类型都引入了对应的包装类型（wrapper class），int 的包装类就是 Integer，从 JDK 1.5 开始引入了自动装箱/拆箱机制，使得二者可以相互转换。

Java 为每个原始类型提供了包装类型：

原始类型: boolean，char，byte，short，int，long，float，double

包装类型：Boolean，Character，Byte，Short，Integer，Long，Float，Double

```
package com.lovo;

public class AutoUnboxingTest {

    public static void main(String[] args) {
        Integer a = new Integer(3);
        Integer b = 3;              // 将 3 自动装箱成 Integer 类型
        int c = 3;
        System.out.println(a == b); // false 两个引用没有引用同一对象
        System.out.println(a == c); // true a 自动拆箱成 int 类型再和 c 比较
    }
}
```

9. char 型变量中能不能存贮一个中文汉字?为什么?

答：char 类型可以存储一个中文汉字，因为 Java 中使用的编码是 Unicode（不选择任何特定的编码，直接使用字符在字符集中的编号，这是统一的唯一方法），一个 char 类型占 2 个字节（16bit），所以放一个中文是没问题的。

补充：使用 Unicode 意味着字符在 JVM 内部和外部有不同的表现形式，在 JVM 内部都是 Unicode，当这个字符被从 JVM 内部转移到外部时（例如存入文件系统中），需要进行编码转换。所以 Java 中有字节流和字符流，以及在字符流和字节流之间进行转换的转换流，如 InputStreamReader 和 OutputStreamReader，这两个类是字节流和字符流之间的适配器类，承担了编码转换的任务；对于 C 程序员来说，要完成这样的编码转换恐怕要依赖于 union（联合体/共用体）共享内存的特征来实现了。

10. Math.round(11.5) 等于多少? Math.round(-11.5)等于多少?

答：Math.round(11.5)==12 Math.round(-11.5)==-11 round 方法返回与参数 最接近的长整数，参数加 1/2 后求其 floor

字符串与数组

1. 下面程序的运行结果是（）

```
String str1 = "hello";
String str2 = "he" + new String("llo");
```

```
System.err.println(str1 == str2);
```

答案：false

解析：因为 str2 中的 llo 是新申请的内存块，而 == 判断的是对象的地址而非值，所以不一样。如果是 `String str2 = str1`，那么就是 true 了。

2. 下面代码的运行结果为?

```
import java.io.*;
import java.util.*;

public class foo{

    public static void main (String[] args){

        String s;

        System.out.println("s=" + s);

    }

}
```

A. 代码得到编译，并输出“s=”

B. 代码得到编译，并输出“s=null”

C. 由于 String s 没有初始化，代码不能编译通过

D. 代码得到编译，但捕获到 NullPointException 异常

答案：C

解析：开始以为会输出 null 什么的，运行后才发现 Java 中所有定义的基本类型或对象都必须初始化才能输出值。

3. String 是最基本的数据类型吗?

答：不是。Java 中的基本数据类型只有 8 个：byte、short、int、long、float、double、char、boolean；除了基本类型（primitive type）和枚举类型（enumeration type），剩下的都是引用类型（reference type）。

4. 数组有没有 length() 方法? String 有没有 length() 方法?

答：数组没有 length()方法，有 length 的属性。String 有 length()方法。JavaScript 中，获得字符串的长度是通过 length 属性得到的，这一点容易和 Java 混淆。

5. 是否可以继承 String 类?

答：String 类是 final 类，不可以被继承。

补充：继承 String 本身就是一个错误的行为，对 String 类型最好的重用方式是关联（HAS-A）而不是继承（IS-A）。

6. String 和 StringBuilder、StringBuffer 的区别?

答：Java 平台提供了两种类型的字符串：String 和 StringBuffer / StringBuilder，它们可以储存和操作字符串。其中 String 是只读字符串，也就意味着 String 引用的字符串内容是不能被改变的。而 StringBuffer 和 StringBuilder 类表示的字符串对象可以直接进行修改。StringBuilder 是 JDK 1.5 中引入的，它和 StringBuffer 的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被 synchronized 修饰，因此它的效率也比 StringBuffer 略高。

有一个面试题问：有没有哪种情况用 + 做字符串连接比调用 StringBuffer / StringBuilder 对象的 append 方法性能更好？如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于 StringBuffer / StringBuilder 的 append 方法的。

7. String s=new String(“xyz”);创建了几个字符串对象？

答：两个对象，一个是静态存储区的"xyz",一个是用 new 创建在堆上的对象。

8. 将字符 “12345” 转换成 long 型

解答： String s=”12345”;

long num=Long.valueOf(s).longValue();

9. 为了显示 myStr = 23 这样的结果，写出在控制台输入的命令

```
public class MyClass {

public static void main(String args[]) {

String s1 = args[0];

String s2 = args[1];

String myStr = args[2];

System.out.printin(“myStr =” + s2 + myStr);

}
```

```
}
```

答：java MyClass 1 2 3 4

10. `String s = "Hello";s = s + " world!”;` 这两行代码执行后，原始的 String 对象中的内容到底变了没有？

没有。因为 String 被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了+操作，那么 s 所指向的那个对象是否发生了改变呢？答案是没有。这时， s 不指向原来那个对象了，而指向了另一个 String 对象，内容为 "Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用 String 来代表字符串的话会引起很大的内存开销。因为 String 对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个 String 对象来表示。这时，应该考虑使用 StringBuffer 类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都 new 一个 String。例如我们要在构造器中对一个名叫 s 的 String 引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo {
private String s;
...
public Demo {
s = "Initial Value";
}
...
}
```

而非 `s = new String("Initial Value”);`

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为 String 对象不可改变，所以对于内容相同的字符串，只要一个 String 对象来表示就可以了。也就说，多次调用上面的构造器创建多个对象，他们的 String 类型属性 s 都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java 认为它们代表同一个 String 对象。而用关键字 new 调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把 String 类设计成不可变类，是它的用途决定的。其实不只 String，很多 Java 标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以 Java 标准类库还提供了一个可变版本，即 StringBuffer。

11. 如何把一段逗号分割的字符串转换成一个数组?

如果不查 jdk api，我很难写出来！我可以说说我的思路：

1 用正则表达式，代码大概为：`String [] result = orgStr.split(",");`

2 用 StingTokenizer ,代码为：

```
StringTokenizer tokener = StringTokenizer(orgStr,",");
String [] result =new String[tokener .countTokens()];
Int i=0;
while(tokener.hasNext(){result[i++]=toker.nextToken();}
```

12. 下面这条语句一共创建了多少个对象：`String s="a"+"b"+"c"+"d";`

答：对于如下代码：

```
String s1 = "a";
String s2 = s1 + "b";
String s3 = "a" + "b";
System.out.println(s2 == "ab");
System.out.println(s3 == "ab");
```

第一条语句打印的结果为 false ，第二条语句打印的结果为 true，这说明 Javac 编译可以对字符串常量直接相加的表达式进行优化，不必要等到运行期去进行加法运算处理，而是在编译时去掉其中的加号，直接将其编译成一个这些常量相连的结果。

题目中的第一行代码被编译器在编译时优化后，相当于直接定义了一个"abcd"的字符串，所以，上面的代码应该只创建了一个 String 对象。写如下两行代码，

```
String s ="a" + "b" + "c" + "d";

System.out.println(s== "abcd");
```

最终打印的结果应该为 true。

13. String 和 StringBuffer 的区别?

答：JAVA 平台提供了两个类：String 和 StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符数据。这个 String 类提供了数值不可改变的字符串。而这个 StringBuffer 类提供的字符串进行修改。当你知道字符数据要改变 的时候你就可以使用 StringBuffer。典型地，你可以使用 StringBuffers 来动态构造字符数据。

14. String, StringBuffer StringBuilder 的区别。

答：

String 的长度是不可变的；

StringBuffer 的长度是可变的，如果你对字符串中的内容经常进行操作，特别是内容要修改时，那么使用 StringBuffer，如果最后需要 String，那么使用 StringBuffer 的 toString() 方法；线程安全；

StringBuilder 是从 JDK 5 开始，为 StringBuffer 该类补充了一个单个线程使用的等价类；通常应该优先使用 StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。

输入输出流

1. 下面哪个流类属于面向字符的输入流?

A. BufferedWriter
B. FileInputStream
C. ObjectInputStream
D. InputStreamReader
答案：D

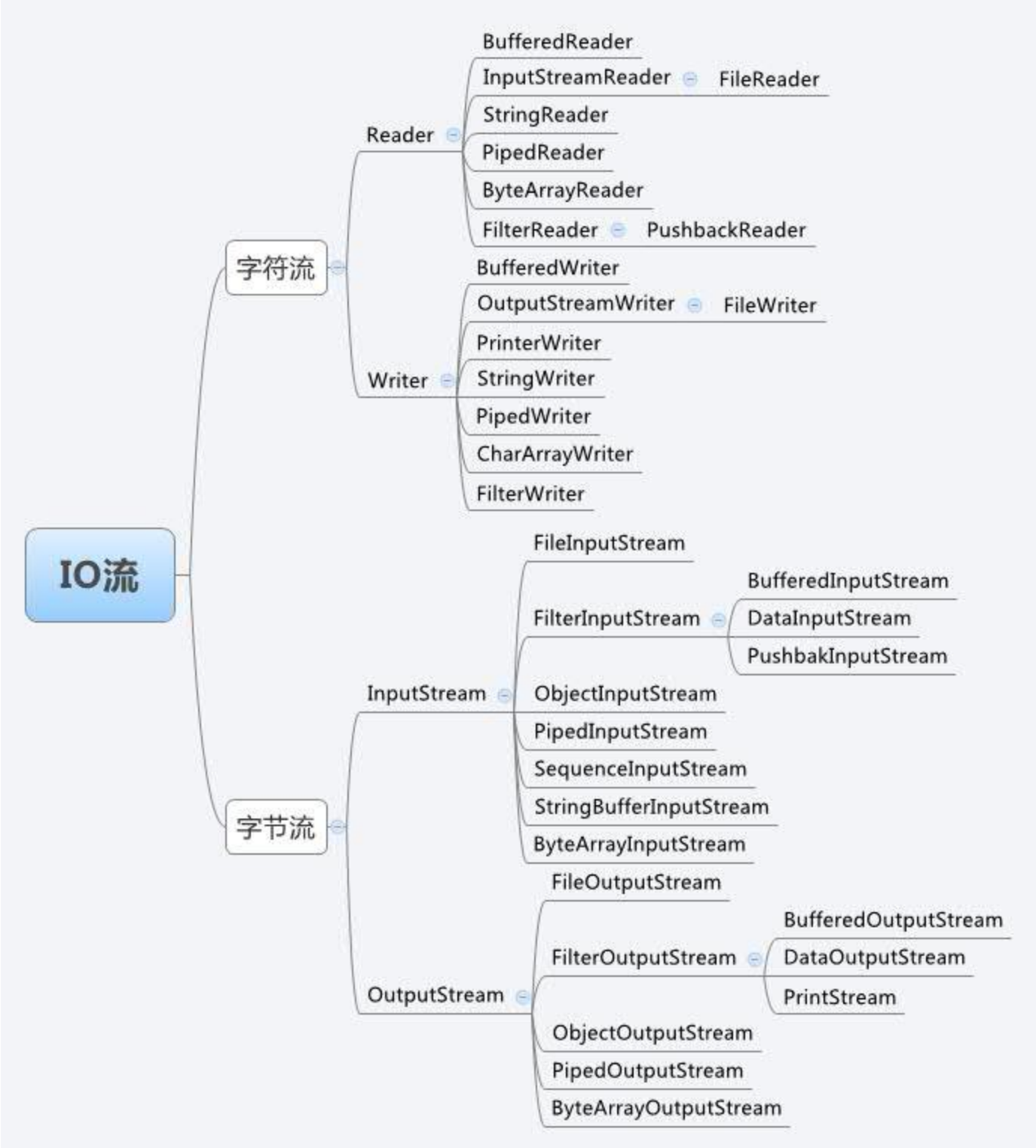
解析：Java 的 IO 操作中有面向字节(Byte)和面向字符(Character)两种方式。

面向字节的操作为以 8 位为单位对二进制的数据进行操作，对数据不进行转换，这些类都是 InputStream 和 OutputStream 的子类。

面向字符的操作为以字符为单位对数据进行操作，在读的时候将二进制数据转为字符，在写的时候将字符转为二进制数据，这些类都是 Reader 和 Writer 的子类。

总结：以 InputStream（输入）/OutputStream（输出）为后缀的是字节流；以 Reader（输入）/Writer（输出）为后缀的是字符流。

扩展：Java 流类图结构，一目了然，解决大部分选择题：



2. 阅读 **Shape** 和 **Circle** 两个类的定义。在序列化一个 **Circle** 的对象 **circle** 到文件时，下面哪个字段会被保存到文件中？

```
class Shape {
```

```
        public String name;
}
class Circle extends Shape implements Serializable{
        private float radius;
        transient int color;
        public static String type = "Circle";
}
```

A. name B. radius C. color D. type

答案：B

解析：这里有详细的解释：http://www.cnblogs.com/lanxuezaipiao/p/3369962.html

3. 什么是 Java 序列化，如何实现 Java 序列化？

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行流化。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。序列化是为了解决在对对象流进行读写操作时所引发的问题。

序列化的实现：将需要被序列化的类实现 Serializable 接口，该接口没有需要实现的方法，implements Serializable 只是为了标注该对象是可被序列化的，然后使用一个输出流(如：FileOutputStream)来构造一个 ObjectOutputStream(对象流)对象，接着，使用 ObjectOutputStream 对象的 writeObject(Object obj)方法就可以将参数为 obj 的对象写出(即保存其状态)，要恢复的话则用输入流

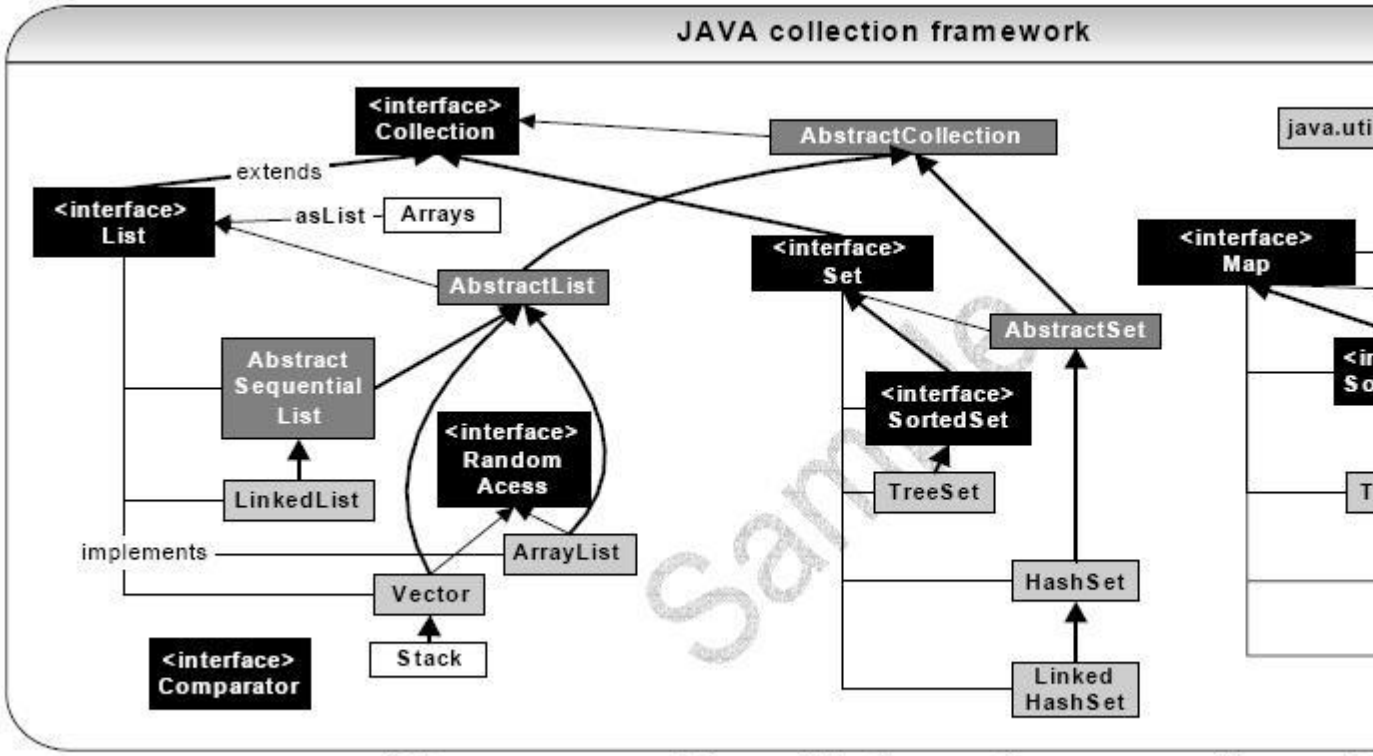
集合类

1. 下列说法正确的是（）

A. LinkedList 继承自 List B. AbstractSet 继承自 Set C. HashSet 继承自 AbstractSet D. WeakMap 继承自 HashMap

答案：AC

解析：下面是一张下载的 Java 中的集合类型的继承关系图，一目了然。



(Diagram sourced from: http://www.wilsonmar.com/1arrays.h

2. ArrayList list = new ArrayList(20);中的 list 扩充几次?

A. 0

B. 1

C. 2

D. 3

答案：A

解析：这里有点迷惑人，大家都知道默认 ArrayList 的长度是 10 个，所以如果你要往 list 里添加 20 个元素肯定要扩充一次（扩充为原来的 1.5 倍），但是这里显示指明了需要多少空间，所以就一次性为你分配这么多空间，也就是不需要扩充了。

3. Java 集合类框架的基本接口有哪些?

Java 集合类提供了一套设计良好的支持对一组对象进行操作的接口和类。Java 集合类里面最基本的接口有：

Collection：代表一组对象，每一个对象都是它的子元素。

Set：不包含重复元素的 Collection。

List：有顺序的 collection，并且可以包含重复元素。

Map：可以把键(key)映射到值(value)的对象，键不能重复。

4. 为什么集合类没有实现 Cloneable 和 Serializable 接口？

集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以它自己的方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。

5. 什么是迭代器(Iterator)？

Iterator 接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实例的 迭代方法。迭代器可以在迭代的过程中删除底层集合的元素。

克隆(cloning)或者是序列化(serialization)的语义和含义是跟具体的实现相关的。因此，应该由集合类的具体实现来决定如何被克隆或者是序列化。

6. Iterator 和 ListIterator 的区别是什么？

下面列出了他们的区别：

Iterator 可用来遍历 Set 和 List 集合，但是 ListIterator 只能用来遍历 List 。

Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。

ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引，等等。

7. 快速失败(fail-fast)和安全失败(fail-safe)的区别是什么？

Iterator 的安全失败是基于对底层集合做拷贝，因此，它不受源集合上修改的影响。java.util 包下面的所有的集合类都是快速失败的，而 java.util.concurrent 包下面的所有的类都是安全失败的。快速失败的迭代器会抛出 ConcurrentModificationException 异常，而安全失败的迭代器永远不会抛出这样的异常。

8. Java 中的 HashMap 的工作原理是什么？

Java 中的 HashMap 是以键值对(key-value)的形式存储元素的。HashMap 需要一个 hash 函数，它使用 hashCode()和 equals()方法来向集合/从集合添加和检索元素。当调用 put()方法的时候，HashMap 会计算 key 的 hash 值，然后把键值对存储在集合中合适的索引上。如果 key 已经存在了，value 会被更新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子(load factor)和扩容极限(threshold resizing)。

9. hashCode() 和 equals() 方法的重要性体现在什么地方？

Java 中的 HashMap 使用 hashCode() 和 equals() 方法来确定键值对的索引，当根据键获取值的时候也会用到这两个方法。如果没有正确的实现这两个方法，两个不同的键可能会有相同的 hash 值，因此，可能会被集合认为是相等的。而且，这两个方法也用来发现重复元素。所以这两个方法的实现对 HashMap 的精确性和正确性是至关重要的。

10. HashMap 和 Hashtable 有什么区别？

HashMap 和 Hashtable 都实现了 Map 接口，因此很多特性非常相似。但是，他们有以下不同点： HashMap 允许键和值是 null，而 Hashtable 不允许键或者值是 null。

Hashtable 是同步的，而 HashMap 不是。因此， HashMap 更适合于单线程环境，而 Hashtable 适合于多线程环境。

HashMap 提供了可供应用迭代的键的集合，因此，HashMap 是快速失败的。另一方面，Hashtable 提供了对键的列举(Enumeration)。

一般认为 Hashtable 是一个遗留的类。

11. 数组(Array)和列表(ArrayList)有什么区别？什么时候应该使用 Array 而不是 ArrayList？

下面列出了 Array 和 ArrayList 的不同点：

Array 可以包含基本类型和对象类型，ArrayList 只能包含对象类型。

Array 大小是固定的，ArrayList 的大小是动态变化的。

ArrayList 提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等等。 对于基本类型数据，集合使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式相对比较慢。

12. ArrayList 和 LinkedList 有什么区别？

ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点：

ArrayList 是基于索引的数据接口，它的底层是数组。它可以以 O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是 O(n)。

相对于 ArrayList，LinkedList 的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。

LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

也可以参考 ArrayList vs. LinkedList。

13. Comparable 和 Comparator 接口是干什么的？列出它们的区别。

Java 提供了只包含一个 compareTo() 方法的 Comparable 接口。这个方法可以个给两个对象排序。具体来说，它返回负数，0，正数来表明输入对象小于，等于，大于已经存在的对象。

Java 提供了包含 compare() 和 equals() 两个方法的 Comparator 接口。compare() 方法用来给两个输入参数排序，返回负数，0，正数表明第一个参数是小于，等于，大于第二个参数。equals() 方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。

14. Java 集合类框架的最佳实践有哪些？

根据应用的需要正确选择要使用的集合的类型对性能非常重要，比如：假如元素的大小是固定的，而且能事先知道，我们就应该用 Array 而不是 ArrayList。 有些集合类允许指定初始容量。因此，如果我们能估计出存储的元素的数目，我们可以设置初始容量来避免重新计算 hash 值或者是扩容。

为了类型安全，可读性和健壮性的原因总是要使用泛型。同时，使用泛型还可以避免运行时的 ClassCastException。

使用 JDK 提供的不变类(immutable class)作为 Map 的键可以避免为我们自己的类实现 hashCode() 和 equals() 方法。

编程的时候接口优于实现。

底层的集合实际上是空的情况下，返回长度是 0 的集合或者是数组，不要返回 null。

15. Enumeration 接口和 Iterator 接口的区别有哪些？

Enumeration 速度是 Iterator 的 2 倍，同时占用更少的内存。但是，Iterator 远远比 Enumeration 安全，因为其他线程不能够修改正在被 iterator 遍历的集合里面的对象。同时，Iterator 允许调用者删除底层集合里面的元素，这对 Enumeration 来说是不可能的。

16. HashSet 和 TreeSet 有什么区别？

HashSet 是由一个 hash 表来实现的，因此，它的元素是无序的。add()，remove()，contains()方法的时间复杂度是 O(1)。

另一方面，TreeSet 是由一个树形的结构来实现的，它里面的元素是有序的。因此，add()，remove()，contains() 方法的时间复杂度是 O(logn)。

17. List、Set、Map 是否继承自 Collection 接口？

答：List、Set 是，Map 不是。Map 是键值对映射容器，与 List 和 Set 有明显的区别，而 Set 存储的零散的元素且不允许有重复元素（数学中的集合也是如此），List 是线性结构的容器，适用于按数值索引访问元素的情形。

18. 说出 ArrayList、Vector、LinkedList 的存储性能和特性？

答：ArrayList 和 Vector 都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector 由于使用了 synchronized 方法（线程安全），通常性能上较 ArrayList 差，而 LinkedList 使用双向链表实现存储（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，其实对内存的利用率更高），按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

Vector 属于遗留容器（早期的 JDK 中使用的容器，除此之外 Hashtable、Dictionary、BitSet、Stack、Properties 都是遗留容器），现在已经不推荐使用，但是由于 ArrayList 和 LinkedListed 都是非线程安全的，如果需要多个线程操作同一个容器，那么可以通过工具类 Collections 中的 synchronizedList 方法将其转换成线程安全的容器后再使用（这其实是装潢模式最好的例子，将已有对象传入另一个类的构造器中创建新的对象来增加新功能）。

19. List、Map、Set 三个接口存储元素时各有什么特点？

答：

1）List 是有序的 Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在 List 中的位置，类似于数组下标）来访问 List 中的元素，这类似于 Java 的数组。

2）Set 是一种不包含重复的元素的 Collection，即任意的两个元素 e1 和 e2 都有 e1.equals(e2)=false，Set 最多有一个 null 元素。

3) Map 接口 ：请注意，Map 没有继承 Collection 接口，Map 提供 key 到 value 的映射

20. 判断下列语句是否正确，如果有错误，请指出错误所在？

List a = new ArrayList();

a.add(5);

答：错误,默认封装 int 类型。

21. 你是怎么理解 Java 泛型的？

答： 在 Java SE 1.5 之前，没有泛型的情况的下，通过对类型 Object 的引用来实现参数的“任意化”，“任意化”带来的缺点是要做显式的强制类型转换，而这种转换是要求开发者对实际参数类型可以预知的情况下进行的。对于强制类型转换错误的情况，编译器可能不提示错误，在运行的时候才出现异常，这是一个安全隐患。

泛型是 Java SE 1.5 的新特性，泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。

泛型的好处是在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，提高代码的重用率。

异常处理

1. 下面关于 java.lang.Exception 类的说法正确的是（）

A. 继承自 Throwable

B. Serialable
CD 不记得，反正不正确
答案：A

解析：Java 异常的基类为 java.lang.Throwable，java.lang.Error 和 java.lang.Exception 继承 Throwable，RuntimeException 和其它的 Exception 等继承 Exception，具体的 RuntimeException 继承 RuntimeException。

2. 扩展：错误和异常的区别(Error vs Exception)

1) java.lang.Error: Throwable 的子类，用于标记严重错误。合理的应用程序不应该去 try/catch 这种错误。绝大多数的错误都是非正常的，就根本不该出现的。

java.lang.Exception: Throwable 的子类，用于指示一种合理的程序想去 catch 的条件。即它仅仅是一种程序运行条件，而非严重错误，并且鼓励用户程序去 catch 它。

2) Error 和 RuntimeException 及其子类都是未检查的异常（unchecked exceptions），而所有其他的 Exception 类都是检查了的异常（checked exceptions）

- checked exceptions: 通常是从一个可以恢复的程序中抛出来的，并且最好能够从这种异常中使用程序恢复。比如 FileNotFoundException, ParseException 等。检查了的异常发生在编译阶段，必须要使用 try…catch（或者 throws ）否则编译不通过。
- unchecked exceptions: 通常是如果一切正常的话本不该发生的异常，但是的确发生了。发生在运行期，具有不确定性，主要是由于程序的逻辑问题所引起的。比如 ArrayIndexOutOfBoundException, ClassCastException 等。从语言本身的角度讲，程序不该去 catch 这类异常，虽然能够从诸如 RuntimeException 这样的异常中 catch 并恢复，但是并不鼓励终端程序员这么做，因为完全没要必要。因为这类错误本身就是 bug，应该被修复，出现此类错误时程序就应该立即停止执行。 因此，面对 Errors 和 unchecked exceptions 应该让程序自动终止执行，程序员不该做诸如 try/catch 这样的事情，而是应该查明原因，修改代码逻辑。
- RuntimeException：RuntimeException 体系包括错误的类型转换、数组越界访问和试图访问空指针等等。处理 RuntimeException 的原则是：如果出现 RuntimeException，那么一定是程序员的错误。例如，可以通过检查数组下标和数组边界来避免数组越界访问异常。其他（IOException 等等）checked 异常一般是外部错误，例如试图从文件尾后读取数据等，这并不是程序本身的错误，而是在应用环境中出现的外部错误。

3. getCustomerInfo() 方法如下，try 中可以捕获三种类型的异常，如果在该方法运行中产生了一个 IOException，将会输出什么结果?

```
public void getCustomerInfo() {
     try {
         // do something that may cause an Exception
     } catch (java.io.FileNotFoundException ex) {

         System.out.print("FileNotFoundException!");

     } catch (java.io.IOException ex) {

         System.out.print("IOException!");

     } catch (java.lang.Exception ex) {

         System.out.print("Exception!");

     }
```

```
    }
```

A. IOException!

B. IOException!Exception!

C. FileNotFoundException!IOException!

D. FileNotFoundException!IOException!Exception!

答案：A

解析：考察多个 catch 语句块的执行顺序。当用多个 catch 语句时，catch 语句块在次序上有先后之分。从最前面的 catch 语句块依次先后进行异常类型匹配，这样如果父异常在子异常类之前，那么首先匹配的将是父异常类，子异常类将不会获得匹配的机会，也即子异常类型所在的 catch 语句块将是不可到达的语句。所以，一般将父类异常类即 Exception 老大放在 catch 语句块的最后一个。

4. try{} 里有一个 return 语句，那么紧跟在这个 try 后的 finally{} 里的 code 会不会被执行，什么时候被执行，在 return 前还是后?

答：会执行，在方法返回调用者前执行。Java 允许在 finally 中改变返回值的做法是不好的，因为如果存在 finally 代码块，try 中的 return 语句不会立马返回调用者，而是记录下返回值待 finally 代码块执行完毕之后再向调用者返回其值，然后如果在 finally 中修改了返回值，这会对程序造成很大的困扰，C# 中就从语法上规定不能做这样的事。

5. Java 语言如何进行异常处理，关键字：throws、throw、try、catch、finally 分别如何使用？

答：Java 通过面向对象的方法进行异常处理，把各种不同的异常进行分类，并提供了良好的接口。在 Java 中，每个异常都是一个对象，它是 Throwable 类或其子类的实例。当一个方法出现异常后便抛出一个异常对象，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 和 finally。一般情况下是用 try 来执行一段程序，如果出现异常，系统会抛出（throw）一个异常，这时候你可以通过它的类型来捕捉（catch）它，或最后（finally）由缺省处理器来处理；try 用来指定一块预防所有“异常”的程序；catch 子句紧跟在 try 块后面，用来指定你想要捕捉的“异常”的类型；throw 语句用来明确地抛出一个“异常”；throws 用来标明一个成员函数可能抛出的各种“异常”；finally 为确保一段代码不管发生什么“异常”都被执行一段代码；可以在一个成员函数调用的外面写一个 try 语句，在这个成员函数内部写

另一个 try 语句保护其他代码。每当遇到一个 try 语句，“异常”的框架就放到栈上面，直到所有的 try 语句都完成。如果下一级的 try 语句没有对某种“异常”进行处理，栈就会展开，直到遇到有处理这种“异常”的 try 语句。

6. 运行时异常与受检异常有何异同？

答：异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误，只要程序设计得没有问题通常就不会发生。受检异常跟程序运行的上下文环境有关，即使程序设计无误，仍然可能因使用的问题而引发。Java 编译器要求方法必须声明抛出可能发生的受检异常，但是并不要求必须声明抛出未被捕获的运行时异常。异常和继承一样，是面向对象程序设计中经常被滥用的东西，神作《Effective Java》中对异常的使用给出了以下指导原则：

- 不要将异常处理用于正常的控制流（设计良好的 API 不应该强迫它的调用者为了正常的控制流而使用异常）
- 对可以恢复的情况使用受检异常，对编程错误使用运行时异常
- 避免不必要的使用受检异常（可以通过一些状态检测手段来避免异常的发生）
- 优先使用标准的异常
- 每个方法抛出的异常都要有文档
- 保持异常的原子性
- 不要在 catch 中忽略掉捕获到的异常

7. 请写出 5 种常见到的 runtime exception。

答：

NullPointerException：当操作一个空引用时会出现此错误。

NumberFormatException：数据格式转换出现问题时出现此异常。

ClassCastException：强制类型转换类型不匹配时出现此异常。

ArrayIndexOutOfBoundsException：数组下标越界，当使用一个不存在的数组下标时出现此异常。

ArithmeticException：数学运行错误时出现此异常

8. error 和 exception 有什么区别?

答：

error 表示系统级的错误和程序不必处理的异常，是恢复不是不可能但很困难的情况下的一种严重问题；比如内存溢出，不可能指望程序能处理这样的情况； exception 表示需要捕捉或

者需要程序进行处理的异常，是一种设计或实现问题；也就是说，它表示如果程序运行正常，从不会发生的情况。

Java 平台与内存管理

1. GC 线程是否为守护线程？（）

答案：是

解析：线程分为守护线程和非守护线程（即用户线程）。

只要当前 JVM 实例中尚存在任何一个非守护线程没有结束，守护线程就全部工作；只有当最后一个非守护线程结束时，守护线程随着 JVM 一同结束工作。 守护线程最典型的应用就是 GC (垃圾回收器)

2. 解释内存中的栈（stack）、堆(heap)和静态存储区的用法。

答：通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用内存中的栈空间；而通过 new 关键字和构造器创建的对象放在堆空间；程序中的字面量（literal）如直接书写的 100、“hello”和常量都是放在静态存储区中。栈空间操作最快但是也很小，通常大量的对象都是放在堆空间，整个内存包括硬盘上的虚拟内存都可以被当成堆空间来使用。

```
String str = new String(“hello”);
```

上面的语句中 str 放在栈上，用 new 创建出来的字符串对象放在堆上，而“hello”这个字面量放在静态存储区。

3. Java 中会存在内存泄漏吗，请简单描述。

答：理论上 Java 因为有垃圾回收机制（GC）不会存在内存泄露问题（这也是 Java 被广泛使用于服务器端编程的一个重要原因）；然而在实际开发中，可能会存在无用但可达的对象，这些对象不能被 GC 回收也会发生内存泄露。一个例子就是 Hibernate 的 Session（一级缓存）中的对象属于持久态，垃圾回收器是不会回收这些对象的，然而这些对象中可能存在无用的垃圾对象。下面的例子也展示了 Java 中发生内存泄露的情况：

```
package com.lovo;

import java.util.Arrays;
import java.util.EmptyStackException;

public class MyStack<T> {
    private T[] elements;
```

```
    private int size = 0;

    private static final int INIT_CAPACITY = 16;

    public MyStack() {
        elements = (T[]) new Object[INIT_CAPACITY];
    }

    public void push(T elem) {
        ensureCapacity();
        elements[size++] = elem;
    }

    public T pop() {
        if(size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }

    private void ensureCapacity() {
        if(elements.length == size) {
            elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }
}
```

上面的代码实现了一个栈（先进后出（FILO））结构，乍看之下似乎没有什么明显的问题，它甚至可以通过你编写的各种单元测试。然而其中的 pop 却存在内存泄露的问题，当我们用 pop 方法弹出栈中的对象时，该对象不会被当作垃圾回收，即使使用栈的程序不再引用这些对象，因为栈内部维护着对这些对象的过期引用（obsolete reference）。在支持垃圾回收的语言中，内存泄露是很隐蔽的，这种内存泄露其实就是无意识的对象保持。如果一个对象引用被无意识的保留起来了，那么垃圾回收器不会处理这个对象，也不会处理该对象引用的其他对象，即使这样的对象只有少数几个，也可能会导致很多的对象被排除在垃圾回收之外，从而对性能造成重大影响，极端情况下会引发 Disk Paging（物理内存与硬盘的虚拟内存交换数据），甚至造成 OutOfMemoryError。

4. GC 是什么？为什么要有 GC？

答：GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。Java 程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：`System.gc()` 或 `Runtime.getRuntime().gc()` ，但 JVM 可以屏蔽掉显示的垃圾回收调用。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在 Java 诞生初期，垃圾回收是 Java 最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今 Java 的垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得 iOS 的系统比 Android 系统有更好的用户体验，其中一个深层次的原因就在于 Android 系统中垃圾回收的不可预知性。

5. 第 3 行中生成的 object 在第几行执行后成为 garbage collection 的对象？

```
1.public class MyClass {
2. public StringBuffer aMethod() {
3. StringBuffer sf = new StringBuffer(“Hello”);
4. StringBuffer[] sf_arr = new StringBuffer[1];
5. sf_arr[0] = sf;
6. sf = null;
7. sf_arr[0] = null;
8. return sf;
9. }
10.}
```

答：第 7 行

6. 描述一下 JVM 加载 class 文件的原理机制?

答：JVM 中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java 中的类加载器是一个重要的 Java 运行时系统组件，它负责在运行时查找和装入类文件中的类。

1. 由于 Java 的跨平台性，经过编译的 Java 源程序并不是一个可执行程序，而是一个或多个类文件。当 Java 程序需要使用某个类时，JVM 会确保这个类已经被加载、连接(验证、准备和解析)和初始化。类的加载是指把类的 .class 文件中的数据读入到内存中，通常是创建一个字节数组读入 .class 文件，然后产生与所加载类对应的 Class 对象。加载完成后，Class 对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备(为静态变量分配内存并设置默认的初始值)和解析(将符号引用替换为直接引用)三个步骤。最

后 JVM 对类进行初始化，包括：1. 如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2. 如果类中存在初始化语句，就依次执行这些初始化语句。

2. 类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader 的子类）。从 JDK 1.2 开始，类加载过程采取了父亲委托机制(PDM)。PDM 更好的保证了 Java 平台的安全性，在该机制中，JVM 自带的 Bootstrap 是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM 不会向 Java 程序提供对 Bootstrap 的引用。下面是关于几个类加载器的说明：

   a)Bootstrap：一般用本地代码实现，负责加载 JVM 基础核心类库（rt.jar）；

   b)Extension：从 `java.ext.dirs` 系统属性所指定的目录中加载类库，它的父加载器是 Bootstrap；

   c)System：又叫应用类加载器，其父类是 Extension。它是应用最广泛的类加载器。它从环境变量 classpath 或者系统属性 java.class.path 所指定的目录中记载类，是用户自定义加载器的默认父加载器。

XML

1. XML 包括哪些解释技术，区别是什么？

包括：DOM（Document Object Modal）文档对象模型，SAX（Simple API for XML）。DOM 是一次性将整个文档读入内存操作，如果是文档比较小，读入内存，可以极大提高操作的速度，但如果文档比较大，那么这个就吃力了。所以此时 SAX 应用而生，它不是一次性的将整个文档读入内存，这对于处理大型文档就比较就力了

2. XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方式？

答：XML 文档定义分为 DTD 和 Schema 两种形式；其本质区别在于 Schema 本身也是一个 XML 文件，可以被 XML 解析器解析。对 XML 的解析主要有 DOM（文档对象模型）、SAX、StAX（JDK 1.6 中引入的新的解析 XML 的方式，Streaming API for XML）等，其中 DOM 处理大型文件时其性能下降的非常厉害，这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问（典型的用空间换取时间的策略）；SAX 是事件驱动型的 XML 解析方式，它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问；如其名称所暗示的那样，StAX 把重点放在流上。实际上，StAX 与其他方法的区别就在于应用程序能够把 XML 作为一个事件流来处理。将 XML 作

为一组事件来处理的想法并不新颖（事实上 SAX 已经提出来了），但不同之处在于 StAX 允许应用程序代码把这些事件逐个拉出来，而不用提供在解析器方便时从解析器中接收事件的处理程序。

3. 你在项目中哪些地方用到了 XML？

答:XML 的主要作用有两个方面：数据交换（曾经被称为业界数据交换的事实标准，现在此项功能在很多时候都被 JSON 取代）和信息配置。在做数据交换时，XML 将数据用标签组装成起来，然后压缩打包加密后通过网络传送给接收者，接收解密与解压缩后再从 XML 文件中还原相关信息进行处理。目前很多软件都使用 XML 来存储配置信息，很多项目中我们通常也会将作为配置的硬代码（hard code）写在 XML 文件中，Java 的很多框架也是这么做的。

4. 谈谈对 XML 的理解？说明 Web 应用中 Web.xml 文件的作用？

解答：XML（Extensible Markup Language）即可扩展标记语言，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language,标准通用标记语言)。Xml 是 Internet 环境中跨平台的，依赖于内容的技术，是当前处理结构化文档信息的有力工具。扩展标记语言 XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立，虽然 XML 占用的空间比二进制数据要占用更多的空间，但 XML 极其简单易于掌握和使用。

web.xml 的作用是配置欢迎页，servlet，filter，listener 等的。

5. XML 是一种元语言, 可以用它来描述其他语言。

A．正确

B．错误

解答：B

XML（Extensible Markup Language）即可扩展标记语言，它与 HTML 一样，都是 SGML(Standard Generalized Markup Language,标准通用标记语言)。Xml 是 Internet 环境中跨平台的，依赖于内容的技术，是当前处理结构化文档信息的有力工具。扩展标记语言 XML 是一种简单的数据存储语言，使用一系列简单的标记描述数据，而这些标记可以用方便的方式建立，虽然 XML 占用的空间比二进制数据要占用更多的空间，但 XML 极其简单易于掌握和使用。

6. 在 XML 中用于注释的符号是。(选择 1 项)

A．<!– –>

B . <?– –?>

C . <% %>

D . <!– –!>

解答 : A

7. DTD 与 XML Schema 都是 XML 文档。(选择 1 项)

A . 正确

B. 不正确

解答：DTD 不是 XML 文件， schema 是 XML 文档

Java 多线程

1. 下面哪些是 Thread 类的方法?

A start() B run() C exit() D getPriority()

答案 : ABD

解析 : 看 Java API docs 吧 : http://docs.oracle.com/javase/7/docs/api/ ， `exit()` 是 System 类的方法 , 如 System.exit(0)。

2. 下面程序的运行结果?

```
public static void main(String args[]) {
     Thread t = new Thread() {
         public void run() {
             pong();
         }
     };
     t.run();
     System.out.print("ping");
 }
 static void pong() {
     System.out.print("pong");
 }
```

A. pingpong

B. pongping

C. pingpong 和 pongping 都有可能

D. 都不输出

答案：B

解析：这里考的是 Thread 类中 start() 和 run() 方法的区别了。start() 用来启动一个线程，当调用 start 方法后，系统才会开启一个新的线程，进而调用 run() 方法来执行任务，而单独的调用 run() 就跟调用普通方法是一样的，已经失去线程的特性了。因此在启动一个线程的时候一定要使用 start() 而不是 run()。

3. 进程和线程的区别是什么？

进程是执行着的应用程序，而线程是进程内部的一个执行序列。一个进程可以有多个线程。线程又叫做轻量级进程。

4. 创建线程有几种不同的方式？你喜欢哪一种？为什么？

有三种方式可以用来创建线程：

- 继承 Thread 类
- 实现 Runnable 接口
- 应用程序可以使用 Executor 框架来创建线程池

实现 Runnable 接口这种方式更受欢迎，因为这不需要继承 Thread 类。在应用设计中已经继承了别的对象的情况下，这需要多继承（而 Java 不支持多继承），只能实现接口。同时，线程池也是非常高效的，很容易实现和使用。

5. 概括的解释下线程的几种可用状态。

线程在执行过程中，可以处于下面几种状态：

- 就绪(Runnable):线程准备运行，不一定立马就能开始执行。
- 运行中(Running)：进程正在执行线程的代码。
- 等待中(Waiting):线程处于阻塞的状态，等待外部的处理结束。
- 睡眠中(Sleeping)：线程被强制睡眠。
- I/O 阻塞(Blocked on I/O)：等待 I/O 操作完成。
  同步阻塞(Blocked on Synchronization)：等待获取锁。
- 死亡(Dead)：线程完成了执行。

6. 同步方法和同步代码块的区别是什么？

在 Java 语言中，每一个对象有一把锁。线程可以使用 synchronized 关键字来获取对象上的锁。 synchronized 关键字可应用在方法级别(粗粒度锁)或者是代码块级别(细粒度锁)。

7. 在监视器(Monitor)内部，是如何做线程同步的？程序应该做哪种级别的同步？

监视器和锁在 Java 虚拟机中是一块使用的。监视器监视一块同步代码块，确保一次只有一个线程执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码。

8. 什么是死锁(deadlock)？

两个进程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是两个进程都陷入了无限的等待中。

9. 如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？

使用多线程的时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了

10. sleep() 和 wait() 有什么区别?

答：sleep()方法是线程类（Thread）的静态方法，导致此线程暂停执行指定时间，将执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复（线程回到就绪（ready）状态），因为调用 sleep 不会释放对象锁。wait() 是 Object 类的方法，对此对象调用 wait() 方法导致本线程放弃对象锁(线程暂停执行)，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后本线程才进入对象锁定池准备获得对象锁进入就绪状态。

补充：这里似乎漏掉了一个作为先决条件的问题，就是什么是进程，什么是线程？为什么需要多线程编程？答案如下所示：

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，是操作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，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它占用了更多的 CPU 资源。

11. sleep() 和 yield() 有什么区别?

答:

① sleep() 方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield() 方法只会给相同优先级或更高优先级的线程以运行的机会；

② 线程执行 sleep() 方法后转入阻塞（blocked）状态，而执行 yield() 方法后转入就绪（ready）状态；

③ sleep() 方法声明抛出 InterruptedException，而 yield() 方法没有声明任何异常；

④ sleep() 方法比 yield() 方法（跟操作系统相关）具有更好的可移植性。

12. 当一个线程进入一个对象的 synchronized 方法 A 之后，其它线程是否可进入此对象的 synchronized 方法？

答：不能。其它线程只能访问该对象的非同步方法，同步方法则不能进入。

13. 请说出与线程同步相关的方法。

答：

- wait():使一个线程处于等待（阻塞）状态，并且释放所持有的对象的锁；
- sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉 InterruptedException 异常；
- notify():唤醒一个处于等待状态的线程，当然在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且与优先级无关；
- notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争；

JDK 1.5 通过 Lock 接口提供了显式(explicit)的锁机制，增强了灵活性以及对线程的协调。Lock 接口中定义了加锁（lock()）和解锁(unlock())的方法，同时还提供了 newCondition() 方法来产生用于线程之间通信的 Condition 对象；

JDK 1.5 还提供了信号量(semaphore)机制，信号量可以用来限制对某个共享资源进行访问的线程的数量。在对资源进行访问之前，线程必须得到信号量的许可（调用 Semaphore 对象的 acquire()方法）；在完成对资源的访问后，线程必须向信号量归还许可（调用 Semaphore 对象的 release() 方法）。

14. synchronized 关键字的用法？

答：synchronized 关键字可以将对象或者方法标记为同步，以实现对对象和方法的互斥访问，可以用 synchronized(对象) { … }定义同步代码块，或者在声明方法时将 synchronized 作为方法的修饰符。在第 60 题的例子中已经展示了 synchronized 关键字的用法。

15. 举例说明同步和异步。

答：如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（数据库操作中的悲观锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

16. 启动一个线程是用 run() 还是 start() 方法?

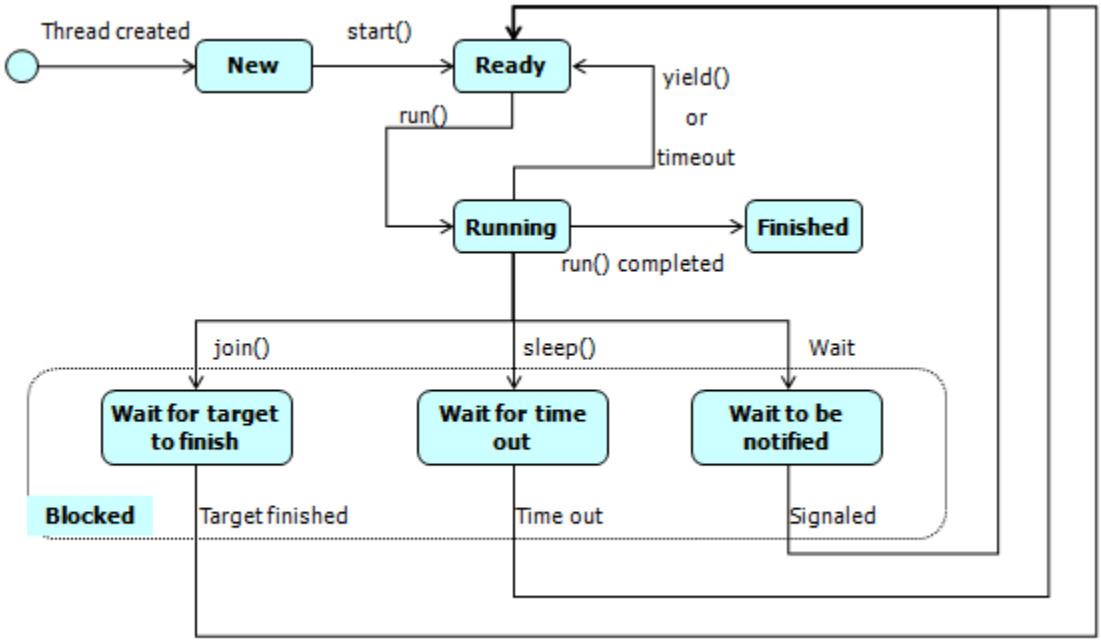
答：启动一个线程是调用 start() 方法，使线程所代表的虚拟处理机处于可运行状态，这意味着它可以由 JVM 调度并执行，这并不意味着线程就会立即运行。run()方法是线程启动后要进行回调（callback）的方法。

17. 什么是线程池（thread pool）？

答：在面向对象编程中，创建和销毁对象是很费时间的，因为创建一个对象要获取内存资源或者其它更多资源。在 Java 中更是如此，虚拟机将试图跟踪每一个对象，以便能够在对象销毁后进行垃圾回收。所以提高服务程序效率的一个手段就是尽可能减少创建和销毁对象的次数，特别是一些很耗资源的对象创建和销毁，这就是"池化资源"技术产生的原因。线程池顾名思义就是事先创建若干个可执行的线程放入一个池（容器）中，需要的时候从池中获取线程不用自行创建，使用完毕不需要销毁线程而是放回池中，从而减少创建和销毁线程对象的开销。

18. 线程的基本状态以及状态之间的关系？

答：



线程的生命周期图

除去起始（new）状态和结束（finished）状态，线程有三种状态，分别是：就绪（ready）、运行（running）和阻塞（blocked）。其中就绪状态代表线程具备了运行的所有条件，只等待 CPU 调度（万事俱备，只欠东风）；处于运行状态的线程可能因为 CPU 调度（时间片用完了）的原因回到就绪状态，也有可能因为调用了线程的 yield 方法回到就绪状态，此时线程不会释放它占有的资源的锁，坐等 CPU 以继续执行；运行状态的线程可能因为 I/O 中断、线程休眠、调用了对象的 wait 方法而进入阻塞状态（有的地方也称之为等待状态）；而进入阻塞状态的线程会因为休眠结束、调用了对象的 notify 方法或 notifyAll 方法或其他线程执行结束而进入就绪状态。注意：调用 wait 方法会让线程进入等待池中等待被唤醒， notify 方法或 notifyAll 方法会让等待锁中的线程从等待池进入等锁池，在没有得到对象的锁之前，线程仍然无法获得 CPU 的调度和执行。

19. 死锁的必要条件？怎么克服？

答：产生死锁的四个必要条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺。

循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之一不满足，就不会发生死锁。

死锁的解决方法:

a 撤消陷于死锁的全部进程;

b 逐个撤消陷于死锁的进程，直到死锁不存在;

c 从陷于死锁的进程中逐个强迫放弃所占用的资源，直至死锁消失。

d 从另外一些进程那里强行剥夺足够数量的资源分配给死锁进程，以解除死锁状态

JDBC 与数据库

1. 下列属于关系型数据库的是（）

A. Oracle

B. MySql

C. IMS

D. MongoDB

答案：AB

解答：IMS（Information Management System ）数据库是 IBM 公司开发的两种数据库类型之一;

一种是关系数据库，典型代表产品：DB2； 另一种则是层次数据库，代表产品：IMS 层次数据库。

非关系型数据库有 MongoDB、memcachedb、Redis 等。

2. 在进行数据库编程时，连接池有什么作用？

答：由于创建连接和释放连接都有很大的开销（尤其是数据库服务器不在本地时，每次建立连接都需要进行 TCP 的三次握手，再加上网络延迟，造成的开销是不可忽视的），为了提升系统访问数据库的性能，可以事先创建若干连接置于连接池中，需要时直接从连接池获取，使用结束时归还连接池而不必关闭连接，从而避免频繁创建和释放连接所造成的开销，这是典型的用空间换取时间的策略（浪费了空间存储连接，但节省了创建和释放连接的时间）。池化技术在 Java 开发中是很常见的，在使用线程时创建线程池的道理与此相同。基于 Java 的开源数据库连接池主要有： C3P0、Proxool、DBCP、BoneCP、Druid 等。

【补充】在计算机系统中时间和空间是不可调和的矛盾，理解这一点对设计满足性能要求的算法是至关重要的。大型网站性能优化的一个关键就是使用缓存，而缓存跟上面讲的连接池道理非常类似，也是使用空间换时间的策略。可以将热点数据置于缓存中，当用户查询这些数据时可以直接从缓存中得到，这无论如何也快过去数据库中查询。当然，缓存的置换策略等也会对系统性能产生重要影响，对于这个问题的讨论已经超出了这里要阐述的范围。

3. 什么是 DAO 模式？

答：DAO（DataAccess Object）顾名思义是一个为数据库或其他持久化机制提供了抽象接口的对象，在不暴露数据库实现细节的前提下提供了各种数据操作。为了建立一个健壮的 Java EE 应用，应该将所有对数据源的访问操作进行抽象化后封装在一个公共 API 中。

用程序设计语言来说，就是建立一个接口，接口中定义了此应用程序中将会用到的所有事务方法。在这个应用程序中，当需要和数据源进行交互的时候则使用这个接口，并且编写一个单独的类来实现这个接口，在逻辑上该类对应一个特定的数据存储。DAO 模式实际上包含了两个模式，一是 Data Accessor（数据访问器），二是 Data Object（数据对象），前者要解决如何访问数据的问题，而后者要解决的是如何用对象封装数据。

4. 什么是 ORM？

答：对象关系映射（Object-Relational Mapping，简称 ORM）是一种为了解决程序的面向对象模型与数据库的关系模型互不匹配问题的技术；简单的说，ORM 是通过使用描述对象和数据库之间映射的元数据（可以用 XML 或者是注解），将 Java 程序中的对象自动持久化到关系数据库中或者将关系数据库表中的行转换成 Java 对象，其本质上就是将数据从一种形式转换到另外一种形式。

5. JDBC 中如何进行事务处理？

答：Connection 提供了事务处理的方法，通过调用 setAutoCommit(false)可以设置手动提交事务；当事务完成后用 commit()显式提交事务；如果在事务处理过程中发生异常则通过 rollback() 进行事务回滚。除此之外，较新的 JDBC 标准还引入了 Savepoint（保存点）的概念，允许事务回滚到指定的保存点。

6. 事务的 ACID 是指什么？

答：

1)原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；

2)一致性(Consistent)：事务结束后系统状态是一致的；

3)隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；

4)持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

【补充】关于事务，在面试中被问到的概率是很高的，可以问的问题也是很多的。首先需要知道的是，只有存在并发数据访问时才需要事务。

7. 使用 JDBC 操作数据库时，如何提升读取数据的性能？如何提升更新数据的性能？

答：要提升读取数据的性能，可以指定通过结果集（ResultSet）对象指定每次抓取数据的大小（fetch size）；要提升更新数据的性能可以使用 PreparedStatement 语句构建批处理（batch）。

8. 存储过程和函数的区别

答：

从参数的返回情况来看：

如果返回多个参数值最好使用存储过程，如果只有一个返回值的话可以使用函数。

从调用情况来看：

如果在 SQL 语句（DML 或 SELECT）中调用的话一定是存储函数或存储的封装函数不可以是存储过程，但调用存储函数的时候还有好多限制以及函数的纯度等级的问题，如果是在过程化语句中调用的话，就要看你要实现什么样的功能。函数一般情况下是用来计算并返回一个计算结果而存储过程一般是用来完成特定的数据操作（比如修改、插入数据库表或执行某些 DDL 语句等等），所以虽然他们的语法上很相似但用户在使用他们的时候所需要完成的功能大部分情况下是不同的。

9. 你认为在表上建立索引可以提高数据库系统的效率吗，为什么？

答：不一定

建立太多的索引将会影响更新和插入的速度，因为它需要同样更新每个索引文件。对于一个经常需要更新和插入的表格，就没有必要为一个很少使用的 where 子句单独建立索引了，对于比较小的表，排序的开销不会很大，也没有必要建立另外的索引。

10. 什么是数据库的参照完整性？

答：数据库的参照完整性是指表与表之间的一种对应关系，通常情况下可以通过设置两表之间的主键、外键关系，或者编写两表的触发器来实现。 有对应参照完整性的两张表格，在对他们进行数据插入、更新、删除的过程中，系统都会将被修改表格与另一张对应表格进行对照，从而阻止一些不正确的数据的操作。

11. 如何优化数据库，如何提高数据库的性能?

答:

1）硬件调整性能 最有可能影响性能的是磁盘和网络吞吐量,解决办法扩大虚拟内存，并保证有足够可以扩充的空间；把数据库服务器上的不必要服务关闭掉；把数据库服务器和主域服务器分开；把 SQL 数据库服务器的吞吐量调为最大；在具有一个以上处理器的机器上运行 SQL 。

2）调整数据库

若对该表的查询频率比较高，则建立索引；建立索引时，想尽对该表的所有查询搜索操作，按照 where 选择条件建立索引，尽量为整型键建立为有且只有一个簇集索引，数据在物理上按顺序在数据页上，缩短查找范围，为在查询经常使用的全部列建立非簇集索引，能最大地覆盖查询；但是索引不可太多，执行 UPDATE DELETE INSERT 语句需要用于维护这些索引的开销量急剧增加；避免在索引中有太多的索引键；避免使用大型数据类型的列为索引；保证每个索引键值有少数行。

3）使用存储过程

应用程序的实现过程中，能够采用存储过程实现的对数据库的操作尽量通过存储过程来实现，因为存储过程是存放在数据库服务器上的一次性被设计、编码、测试，并被再次使用，需要执行该任务的应用可以简单地执行存储过程，并且只返回结果集或者数值，这样不仅可以使程序模块化，同时提高响应速度，减少网络流量，并且通过输入参数接受输入，使得在应用中完成逻辑的一致性实现。

4）应用程序结构和算法

建立查询条件索引仅仅是提高速度的前提条件，响应速度的提高还依赖于对索引的使用。因为人们在

使用 SQL 时往往会陷入一个误区，即太关注于所得的结果是否正确，特别是对数据量不是特别大的数据库操作时，是否建立索引和使用索引的好坏对程序的响应速度并不大，因此程序员

在书写程序时就忽略了不同的实现方法之间可能存在的性能差异，这种性能差异在数据量特别大时或者大型的或是复杂的数据库环境中（如联机事务处理 OLTP 或决策支持系统 DSS ）中表现得尤为明显。在工作实践中发现，不良的 SQL 往往来自于不恰当的索引设计、不充份的连接条件和不可优化的 where 子句。在对它们进行适当的优化后，其运行速度有了明显地提高！

Servlet 与 JSP

1. JSP 有哪些内置对象和动作？它们的作用分别是什么？

JSP 共有以下 9 种基本内置组件：

- request 用户端请求，此请求会包含来自 GET/POST 请求的参数
- response 网页传回用户端的回应
- pageContext 网页的属性是在这里管理
- session 与请求有关的会话期
- application servlet 正在执行的内容
- out 用来传送回应的输出
- config servlet 的构架部件
- page JSP 网页本身
- exception 针对错误网页，未捕捉的例外

常用的组件：request、response、out、session、application、exception

2. 描述 JSP 和 Servlet 的区别、共同点、各自应用的范围

答：JSP 在本质上就是 SERVLET,但是两者的创建方式不一样.Servlet 完全是 JAVA 程序代码构成，擅长于流程控制和事务处理，通过 Servlet 来生成动态网页很不直观.JSP 由 HTML 代码和 JSP 标签构成，可以方便地编写动态网页.因此在实际应用中采用 Servlet 来控制业务流程，而采用 JSP 来生成动态网页.

3. 从以下哪一个选项中可以获得 Servlet 的 初始化参数?

A.Servlet B.ServletContext C.ServletConfig D.GenericServlet

解答：C

servlet 的生命周期的方法中有一个 init 方法，其中一个重载的 init 方法的参数为 ServletConfig 可以获取初始化参数。

4. 哪一个对象可以用于获得浏览器发送的请求?

A.HttpServletRequest B.HttpServletResponse C.HttpServlet D.Http

解答：A

HttpServletRequest 中有一些方法可以获取浏览器发送的请求信息。

5. 运行 jsp 需要安装___Web 服务器。

A．Apache

B．tomcat

C．WebLogic

D．IIS

答：BC

Apache 是 PHP 程序运行的服务器，IIS 是.net 程序运行的服务器。

6. 在服务器的网络编程中，解决会话跟踪的方法有：

A. 使用 Cookie。

B. 使用 URL 重写。

C. 使用隐藏的表单域。

D. 以上方法都不能单独使用。

答：ABC

URL 重写就是首先获得一个进入的 URL 请求然后把它重新写成网站可以处理的另一个 URL 的过程

隐藏域是在页面级保存信息。与其他用户标准控件的区别是，隐藏域不被呈现在页面中。当页面提交的时候，隐藏域中的值将被一同发送给服务端。

Cookie 是以文本存储于计算机中，使用 name-value 匹配。一般用户存储标识用户信息

7. 与 HttpSessionListener 接口有关的方法是。

A.sessionInitialized()

B.sessionCreated()

C.sessionFinialized()

D.sessionDestroyed()

答：BD

8. 关于 JSP 生命周期的叙述，下列哪些为真？

A.JSP 会先解释成 Servlet 源文件，然后编译成 Servlet 类文件

B.每当用户端运行 JSP 时，jspInit()方法都会运行一次

C.每当用户端运行 JSP 时，_jspService()方法都会运行一次

D.每当用户端运行 JSP 时，jspDestroy()方法都会运行一次

解答：AC

9. 以下声明正确的是?

A．

B．<?xml-stylesheet type='txt/css' href='abc.css'?>

C．<?xml-stylesheet type="txt/css" href="abc.css"?>

D．<%xml-stylesheet type="txt/css" href="abc.css"%>

答: BC

单引号，双引号都可以使用在属性上。

10. 下列哪个为 JSP 的隐含对象?

A.env

B.page

C.jspinfo

D.context

解答：B

JSP 有九个隐含对象

- request 对象：保存了很多客户端请求的信息。
- response 对象：生成服务器端响应，然后将响应结果发送到客户端
- out 对象：表示输出流，此输出流将作为请求发送到客户端
- session 对象：我们写个对象放在这个 session 对象中,这个对象就在我们的会话中都存在。
- application 对象:我们写个对象放在这个 application 对象中，这个对象就在整个应用程序中都存在
- pageContext 对象相当于当前页面的容器，可以访问当前页面的所有对象。
- paget 对象:一般我们使用 Page 指令来替代使用这个对象

- exception 对象：用来处理异常的
- config 对象：一样的我们在页面中是使用很少的，一般会在 Servlet 中使用这个

11. 下面的那一个不属于 MVC 模式中的对象？

A. Model

B. View

C. Collection

D. Controller

答：C

MVC 是三个单词的缩写,分别为： 模型(Model),视图(View)和控制(Controller)。 MVC 模式的目的就是实现 Web 系统的职能分工。 Model 层实现系统中的业务逻辑，通常可以用 JavaBean 或 EJB 来实现。View 层用于与用户的交互，通常用 JSP 来实现。 Controller 层是 Model 与 View 之间沟通的桥梁，

它可以分派用户的请求并选择恰当的视图以用于显示，同时它也可以解释用户的输入并将它们映射为模型层可执行的操作。

12. 在 Servlet 处理请求的方式为。(选择 1 项)

A、以进程的方式

B、以程序的方式

C、以线程的方式

D、以响应的方式

答：C

Servlet 采用多线程来处理多个请求同时访问，Servlet 容器维护了一个线程池来服务请求。

13. javax.Servlet 的包中，属于类的是。(选择 1 项)

A、Servlet

B、GenericServlet

C、ServletRequest

D、ServletContext

解答：B

ServletContext 和 ServletRequest 是该包下的接口。

14. Http 缺省的请求方法是。(选择 1 项)

A.PUT

B.GET

C.POST

D.TRACE

答：B

15. 实现现下列哪一种接口的对象，并不需要在 web.xml 文件内进行额外的设定，Servlet 容器就能够回应该对象加入 HTTP 会话所发生的事件？(选择 1 项)

A．ServletContextListener

B．HttpSessionListener

C．HttpSessionAttributeListener

D．HttpSessionBindingListener

解答：D

HttpSessionListener 只需要设置到 web.xml 中就可以监听整个应用中的所有 session 。

HttpSessionBindingListener 必须实例化后放入某一个 session 中，才可以进行监听

16. 下列哪个为 JSP 的小脚本的标签？(选择 1 项)

A．<% %>

B．<@ %>

C．<%! %>

D．<%– %>

解答：A

17. 以下不属于 JSP 的标准指令的是。(选择 1 项)

A.Taglib

B.Include

C.Import

D.Page

解答：C

import 是 page 指令的一个属性。

18. 对于每一个网站访问用户都要访问的变量，应该将它设为_____变量。(选择 1 项)

A. Session

B. Reques

C. Response

D. Application

解答：D

Application 应用程序级变量

19. 查看下列 JSP 内容

```
<html><body>
<% for (int i=0;i<3;i++){ %>
out.print(i*2);
<% } %>
</body></html>
```

当这个 JSP 被运行时，其结果是什么？(选择 1 项)

A.此 JSP 因为语法错误，无法运行

B.显示出 0，2，4

C.显示出 0，2，4，6

D.显示出 out.print(i2) *out.print(i*2) out.print(i*2)

解答：D

20. 假设 A.jsp 内设定一个`<jsp:useBean>` 元素：

```
<jsp:useBean id=”bean1” class=”myBean” />
```

下列哪一个为真？(选择 1 项)

A.bean1 的存取范围（scope）默认为 application

B.在 HTTP 会话内可以存取 bean1

C.只有在 A.jsp 内可以存取 bean1

D.在 A.jsp 所属的 Web 应用程序内均可存取 bean1

解答：C

bean1 的存取范围（scope）默认为 page

(题有一点问题 javabean 的规则是要放在一个包中)

21. 在 MVC 设计模式中，JavaBean 的 作用是。(选择 1 项)

A、Controller

B、Model

C、业务数据的封装

D、View

解答：B

J2EE 与 EJB

1. J2EE 是什么？它包括哪些技术？

答：从整体上讲，J2EE 是使用 Java 技术开发企业级应用的工业标准，它是 Java 技术不断适应和促进企业级应用过程中的产物。适用于企业级应用的 J2EE，提供一个平台独立的、可移植的、多用户的、安全的和基于标准的企业级平台，从而简化企业应用的开发、管理和部署。J2EE 是一个标准，而不是一个现成的产品。

主要包括以下这些技术：

1) Servlet

Servlet 是 Java 平台上的 CGI 技术。Servlet 在服务器端运行，动态地生成 Web 页面。与传统的 CGI 和许多其它类似 CGI 的技术相比，Java Servlet 具有更高的效率并更容易使用。对于 Servlet，重复的请求不会导致同一程序的多次转载，它是依靠线程的方式来支持并发访问的。

2) JSP

JSP(Java Server Page)是一种实现普通静态 HTML 和动态页面输出混合编码的技术。从这一点来看，非常类似 Microsoft ASP、PHP 等技术。借助形式上的内容和外观表现的分离，Web 页面制作的任务可以比较方便地划分给页面设计人员和程序员，并方便地通过 JSP 来合成。在运行时态，JSP 将会被首先转换成 Servlet，并以 Servlet 的形态编译运行，因此它的效率和功能与 Servlet 相比没有差别，一样具有很高的效率。

3) EJB

EJB 定义了一组可重用的组件：Enterprise Beans。开发人员可以利用这些组件，像搭积木一样建立分布式应用。

4) JDBC

JDBC(Java Database Connectivity，Java 数据库连接) API 是一个标准 SQL(Structured Query Language，结构化查询语言)数据库访问接口，它使数据库开发人员能够用标准 Java API 编写数据库应用程序。JDBC API 主要用来连接数据库和直接调用 SQL 命令执行各种 SQL 语句。利用 JDBC API 可以执行一般的 SQL 语句、动态 SQL 语句及带 IN 和 OUT 参数的存储过程。Java 中的 JDBC 相当于 Microsoft 平台中的 ODBC (Open Database Connectivity)。

2. 描述 J2EE 框架的多层结构，并简要说明各层的作用。

答:

1) Presentation layer (表示层）

a. 表示逻辑 (生成界面代码）

b. 接收请求

c. 处理业务层抛出的异常

d. 负责规则验证 (数据格式，数据非空等）

e. 流程控制

2) Service layer (服务层/业务层）

a.封装业务逻辑处理，并且对外暴露接口

b.负责事务，安全等服务

3) Persistence layer (持久层）

a 封装数据访问的逻辑，暴露接口

b.提供方便的数据访问的方案 (查询语言，API，映射机制等）

4) Domain layer (域层）

a. 业务对象以及业务关系的表示

b. 处理简单的业务逻辑

c. 域层的对象可以穿越表示层，业务层，持久层

软件分层结构使得代码维护非常方便，设计明确，各层独立，专注自己擅长的领域。

3. EJB 包含哪 3 种 bean

答 : session bean (会话 bean) , entity bean (实体 bean) , message bean (消息 bean）

4. Tomcat 服务器的默认端口是多少？怎样修改 tomcat 的端口？

答 : 默认端口为 8080，可以通过 service.xml 的 Connector 元素的 port 属性来修改端口

5. EJB 的优点有哪些？(选择 2 项)

A、技术领先 B、价格低廉 C、性能优越 D、强大的容器支持

解答 : CD

6. 无状态会话 Bean、有状态会话 Bean、CMP 与 BMP 中，哪一种 Bean 不需要自己书写连接数据库的代码？

A、无状态会话 Bean B、有状态会话 Bean C、CMP D、BMP

答 : C

BMP 是在 Bean 中完成对数据库 JDBC 的各种调用

CMP 是由 EJB 容器自动完成对数据库的操作

会话 Bean 主要处理业务逻辑

7. 假设 web 应用的文档根目录为 MyApp，那么可以从哪里找到 database.jar 文件。

A. MyApp 目录下 B. MyApp\images 目录下 C. MyApp\WEB-INF 目录下 D. MyApp\WEB-INF\lib 目录下

答：D

Web 工程的 lib 是放置 .jar 文件的地方。

8.要创建一个 EJB，必须要至少编写哪些 Java 类和接口？

A. 定义远程(或业务)接口

B. 定义本地接口

C. 定义 Bean 接口

D. 编写 Bean 的实现

解答：ABC

9. EJB 类库存在于 Java 的哪个版本中？(选择 1 项)。

A. J2SE

B. J2EE

C. J2ME

D. J2NE

解答：B

Java 分为三个体系

JavaSE(Java2 Platform Standard Edition，java 平台标准版)，

JavaEE(Java 2 Platform,Enterprise Edition，java 平台企业版)，

JavaME(Java 2 Platform Micro Edition，java 平台微型版)。

EJB 属于 JavaEE 版本

10. 在 J2EE 中属于 Web 层的组件有(选择 1 项)

A. HTML

B. EJB

C. Applet

D. JSP

解答：D

11. EJB 的角色和三个对象

一个完整的基于 EJB 的分布式计算结构由六个角色组成，这六个角色可以由不同的开发商提供，每个角色所作的工作必须遵循 Sun 公司提供的 EJB 规范，以保证彼此之间的兼容性。这六个角色分别是 EJB 组件开发者（Enterprise Bean Provider）、应用组合者（Application Assembler）、部署者（Deployer）、EJB 服务器提供者（EJB Server Provider）、EJB 容器提供者（EJB Container Provider）、系统管理员（System Administrator）

三个对象是 Remote（Local）接口、Home（LocalHome）接口，Bean 类

12. EJB 的激活机制

以 Stateful Session Bean 为例：其 Cache 大小决定了内存中可以同时存在的 Bean 实例的数量，根据 MRU 或 NRU 算法，实例在激活和去激活状态之间迁移，激活机制是当客户端调用某个 EJB 实例业务方法时，如果对应 EJB Object 发现自己没有绑定对应的 Bean 实例则从其去激活 Bean 存储中（通过序列化机制存储实例）回复（激活）此实例。状态变迁前会调用对应的 ejbActive 和 ejbPassivate 方法。

13. EJB 的几种类型

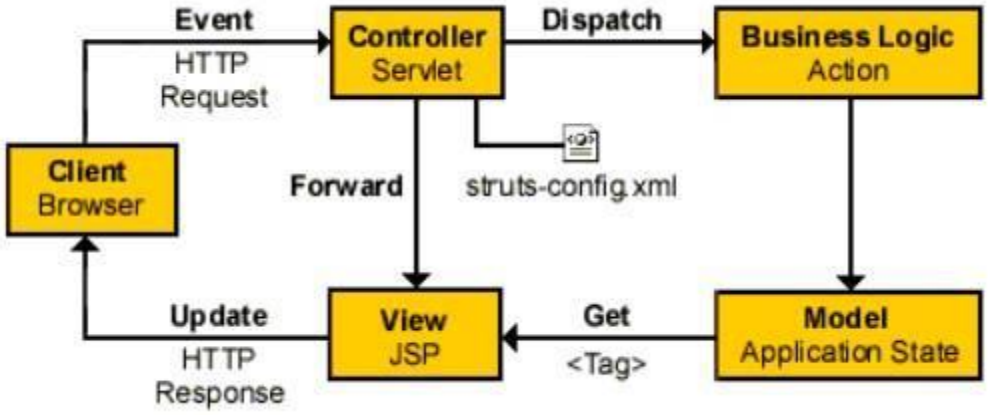
会话（Session）Bean，实体（Entity）Bean 消息驱动的（Message Driven）Bean 会话 Bean 又可分为有状态（Stateful）和无状态（Stateless）两种 实体 Bean 可分为 Bean 管理的持续性（BMP）和容器管理的持续性（CMP）两种

SSH 架构

1. 描述 Struts 体系结构？对应各个部分的开发工作主要包括哪些？

Struts 是 MVC 的一种实现，它将 Servlet 和 JSP 标记（属于 J2EE 规范）用作实现的一部分。Struts 继承了 MVC 的各项特性，并根据 J2EE 的特点，做了相应的变化与扩展。

Struts 的体系结构与工作原理如下图所示：



1）模型（Model）

在 Struts 的体系结构中，模型分为两个部分：系统的内部状态和可以改变状态的操作（事务逻辑）。内部状态通常由一组 Actinform Bean 表示。根据设计或应用程序复杂度的不同，这些 Bean 可以是自包含的并具有持续的状态，或只在需要时才获得数据（从某个数据库）。大型应用程序通常在方法内部封装事务逻辑（操作），这些方法可以被拥有状态信息的 bean 调用。比如购物车 bean，它拥有用户购买商品的信息，可能还有 checkOut()方法用来检查用户的信用卡，并向仓库发定货信息。小型程序中，操作可能会被内嵌在 Action 类，它是 struts 框架中控制器角色的一部分。当逻辑简单时这个方法很适合。建议用户将事务逻辑（要做什么）与 Action 类所扮演的角色（决定做什么）分开。

2）视图（View）　　视图主要由 JSP 建立，struts 包含扩展自定义标签库（TagLib），可以简化创建完全国际化用户界面的过程。目前的标签库包括：Bean Tags、HTML tags、Logic Tags、Nested Tags 以及 Template Tags 等。

3）控制器（Controller）

在 struts 中，基本的控制器组件是 ActionServlet 类中的实例 servelt，实际使用的 servlet 在配置文件中由一组映射（由 ActionMapping 类进行描述）进行定义。对于业务逻辑的操作则主要由 Action、ActionMapping、ActionForward 这几个组件协调完成的，其中 Action 扮演了真正的业务逻辑的实现者，ActionMapping 与 ActionForward 则指定了不同业务逻辑或流程的运行方向。struts-config.xml 文件配置控制器。

2. 简要描述如何结合 struts、hibernate、spring 开发 Web 应用？

答:

Struts 可以将 jsp 页面的表单关联起来,就是把 JSP 页面的表单数据封装成 javaBean,这样的话,在 action 中你再也不需要使用传统的 `request.getParameter("name")`;还有 struts 有一个控制器,你在 struts 编程中的控制器(XxxAction)都是继承总的 ActionServlet,它能集中处理请求,然后转到相关的页面。还有 struts 的表单验证组件,不用你写 js 验证了,只需要你配置一下文件就可以了。另外 struts 的令牌机制可以防表单重复提交。
Spring 是一个轻量级容器,非侵入性.包含依赖注入,AOP 等。它是为了解决企业应用程序开发复杂性而创建的。框架的主要优势之一就是其分层架构，分层架构允许您选择使用哪一个组件，同时为 J2EE 应用程序开发提供集成的框架。

Hibernate: 它可以让我们以 OO 的方式操作数据库，这让我们看到了 hibernate 的强大之处，体验到操作数据的方便。但 hibernate 最耀眼之处是 hibernate 的缓存机制，而不是以 OO 的方式操作数据库。Hibernate 的缓存机制不外乎是一级缓存 session，二级缓存 sessionFactory ，和第三方缓存如 ehcache 。也就是 hibernate 的最强大的地方是它的缓存，理解了这个才能真正的理解 hibernate,Hibernate 的命名查询/命名参数查询，就是将 hql 语句放在一个单独的 xml 文件之中，它仍然让人们以面向对象的方式去操纵数据，而不用在以 OO 的方式写着代码的同时，然后再转变思维，用面向关系的方式去写那些 sql 语句。但 hibernate 不仅做了这些，它的 native sql 查询方式，完全满足 sql 语句的偏爱者，它像 ibatis 一样，将 sql 语句放在配置文件之中。

3. 说明反转控制（IOC）和面向方向编程（AOP）在 spring 中的应用

答：Spring 核心容器（Core）提供 Spring 框架的基本功能。核心容器的主要组件是 BeanFactory，它是工厂模式的实现。BeanFactory 使用控制反转（Ioc）模式将应用程序的配置和依赖性规范与实际的应用代码程序分开。Spring 的声明式事务基于 AOP 实现，却并不需要程序开发者成为 AOP 专家，亦可轻易使用 Spring 的声明式事务管理。

4. 简述基于 Struts 框架 Web 应用的工作流程

答：在 web 应用启动时就会加载初始化 ActionServlet,ActionServlet 从 struts-config.xml 文件中读取配置信息,把它们存放到各种配置对象中，当 ActionServlet 接收到一个客户请求时,将执行如下流程.

1) 检索和用户请求匹配的 ActionMapping 实例,如果不存在,就返回请求路径无效信息;

2) 如果 ActionForm 实例不存在,就创建一个 ActionForm 对象,把客户提交的表单数据保存到 ActionForm 对象中;

3) 根据配置信息决定是否需要表单验证.如果需要验证,就调用 ActionForm 的 validate() 方法;

4) 如果 ActionForm 的 validate() 方法返回 null 或返回一个不包含 ActionMessage 的 ActionErrors 对象, 就表示表单验证成功;

5)ActionServlet 根据 ActionMapping 所包含的映射信息决定将请求转发给哪个 Action, 如果相应的 Action 实例不存在,就先创建这个实例,然后调用 Action 的 execute() 方法;

6) Action 的 execute() 方法返回一个 ActionForward 对象,ActionServlet 在把客户请求转发给 ActionForward 对象指向的 JSP 组件;

7) ActionForward 对象指向 JSP 组件生成动态网页,返回给客户;

5. 在项目中用过 Spring 的 哪些方面？及用过哪些 Ajax 框架？

答：在项目使用过 Spring IOC ，AOP，DAO，ORM，还有上下文环境。

在项目使用过 Ext,Juery 等 Ajax 框架.

6. MVC 模式中 M，V，C 每个代表意义，并简述在 Struts 中 MVC 的表现方式。

答:

MVC 是 Model-View-Controller 的缩写，Model 代表的是应用的业务逻辑（通过 JavaBean，EJB 组件实现），View 是应用的表示层（由 JSP 页面产生）Controller 是通过应用的处理过程控制，（一般是一个 servlert）通过这种设计模型把应用逻辑，处理过程和显示逻辑分成不同的组件实现，这些组件可以进行交互和重用。

在 Struts 框架中 Controller 功能由 ActionServlet 和 ActionMapping 对象构成，核心是一个 Servlet 类型的对象 ActionServlet，它用来接收客户端的请求。ActionServlet 包括一组基于配置的 ActionMapping 对象，每个 ActionMapping 对象实现了一个请求到一个具体的 Model 部分的 Action 处理器对象之间的映射。

Model 部分由 Action 和 ActionForm 对象构成。所有的 Action 处理器对象都是开发者从 Struts 的 Action 类派生的子类。Action 处理器对象封装了具体的处理逻辑，调用业务逻辑

模块，并且把响应提交到合适的 View 组件以产生响应。Struts 提供的 ActionForm 组件对象可以通过定义属性描述客户端表单数据，开发者可以从它派生子类对象，并利用它和 Struts 提供的自定义标记库相结合，可以实现对客户端的表单数据的良好封装和支持，Action 处理器对象可以直接对它进行读写，而不再需要和 request、response 对象进行数据交互。通过 ActionForm 组件对象实现了对 View 和 Model 之间交互的支持（View 部分是通过 JSP 技术实现的）。Struts 提供了自定义的标记库，通过这些自定义标记库可以非常容易地和系统的 Model 部分交互，通过使用这些自定义标记库创建的

JSP 表单，可以实现对 Model 部分中的 ActionForm 的映射，完成对用户数据的封装。

7. Hibernate 中的 Java 对象有几种状态，其相互关系如何（区别和相互转换）。

答：在 Hibernate 中，对象有三种状态：临时状态、持久状态和游离状态。 临时状态：当 new 一个实体对象后，这个对象处于临时状态，即这个对象只是一个保存临时数据的内存区域，如果没有变量引用这个对象，则会被 jre 垃圾回收机制回收。这个对象所保存的数据与数据库没有任何关系，除非通过 Session 的 save 或者 SaveOrUpdate 把临时对象与数据库关联，并把数据插入或者更新到数据库，这个对象才转换为持久对象；

持久状态:持久化对象的实例在数据库中有对应的记录，并拥有一个持久化表示（ID）。对持久化对象进行 delete 操作后，数据库中对应的记录将被删除，那么持久化对象与数据库记录不再存在对应关系，持久化对象变成临时状态。

持久化对象被修改变更后，不会马上同步到数据库，直到数据库事务提交。在同步之前，持久化对象是脏的（Dirty）。

游离状态：当 Session 进行了 Close、Clear 或者 evict 后，持久化对象虽然拥有持久化标识符和与数据库对应记录一致的值，但是因为会话已经消失，对象不在持久化管理之内，所以处于游离状态（也 叫：脱管状态）。游离状态的对象与临时状态对象是十分相似的，只是它还含有持久化标识。

8. 对 Hibernate 的延迟加载如何理解，在实际应用中，延迟加载与 session 关闭的矛盾是如何处理的？

答：延迟加载就是并不是在读取的时候就把数据加载进来，而是等到使用时再加载。那么 Hibernate 是怎么知识用户在什么时候使用数据了呢？又是如何加载数据呢？其实很简单，它使用了代理机制。返回给用户的并不是实体本身，而是实体对象的代理。代理对象在用户调

用 getter 方法时就会去数据库加载数据。但加载数据就需要数据库连接。而当我们把会话关闭时，数据库连接就同时关闭了。这种情况就叫做未初始化的关系。

9. Struts1 中 actionform 和 action 属于 MVC 哪一层，为什么？

答：actionform 和 action 属于 MVC 的 Model 层，Action 用来处理业务逻辑，actionform 保存用户表单数据以便于在不同页面间传递。而 MVC 中的 model 层就是业务逻辑层，该层用于实现具体的业务逻辑、状态维护及管理。

10. struts2 中，Action 通过什么方式获得用户从页面输入的数据，又是通过什么方式把其自身的数据传给视图的？

答:

1）可以直接通过与表单元素相同名称的数据成员（需要存在符合命名规范 set 和 get 方法）获取页面表单数据。

2）会把处理好的数据成员放入值栈中，到页面可以使用 struts2 标签取值就可以了。

11. 说明什么是工厂模式？

工厂模式：工厂模式是一种经常被使用到的模式，根据工厂模式实现的类可以根据提供的数据生成一组类中某一个类的实例，通常这一组类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作。首先需要定义一个基类，该类的子类通过不同的方法实现了基类中的方法。然后需要定义一个工厂类，工厂类可以根据条件生成不同的子类实例。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

12. struts 中如何实现国际化，涉及哪些文件？

解答："国际化"是指一个应用程序在运行时能够根据客户端请求所来自的国家/地区、语言的不同而显示不同的用户界面。Struts 框架通过使用 标记，以及使用 java.util 数据包中定义的 Locale 和 ResourceBundle 类来支持国际化。java.text.MessageFormat 类定义的技术可以支持消息的格式。利用此功能，开发人员不需了解这些类的细节就可进行国际化和设置消息的格式。会涉及到资源文件，不需了解这些类的细节就可进行国际化和设置消息的格式。会涉及到资源文件，struts-config.xml 配置文件, web.xml 配置文件。

13. Struts 框架可以支持以下哪种程序开发语言？(选择 1 项)

A.C

B.C++

C.Java

D.C#

答：C

14. struts 是什么?

struts1 是基于 JSP 和 servlet 的一个开源的 Web 应用框架，使用的是 MVC 的设计模式 struts2 是基于 webwork 技术的框架，是 sun 和 webwork 公司联手开发的一个功能非常齐全的框架， struts2 和 struts1 没有任何关系，是一个全新的框架

15. spring 是什么?

spring 是一个集成了许多第三方框架的大杂烩，其核心技术是 IOC (控制反转，也称依赖注入)和 AOP(面向切面编程)

16. hibernate 是什么?

hibernate 是基于 ORM 对象关系映射(完成对象数据到关系数据映射的机制)实现的,做数据持久化的工具

17. 用自己的话简要阐述 struts2 的执行流程

Struts 2 框架本身大致可以分为 3 个部分：核心控制器 FilterDispatcher、业务控制器 Action 和用户实现的企业业务逻辑组件。核心控制器 FilterDispatcher 是 Struts2 框架的基础，包含了框架内部的控制流程和处理机制。业务控制器 Action 和业务逻辑组件是需要用户来自己实现的。用户在开发 Action 和业务逻辑组件的同时，还需要编写相关的配置文件，供核心控制器 FilterDispatcher 来使用。

Struts2 的工作流程相对于 Struts1 要简单，与 WebWork 框架基本相同，所以说 Struts2 是 WebWork 的升级版本。基本简要流程如下：

1、客户端浏览器发出 HTTP 请求。

2、根据 web.xml 配置，该请求被 FilterDispatcher 接收。

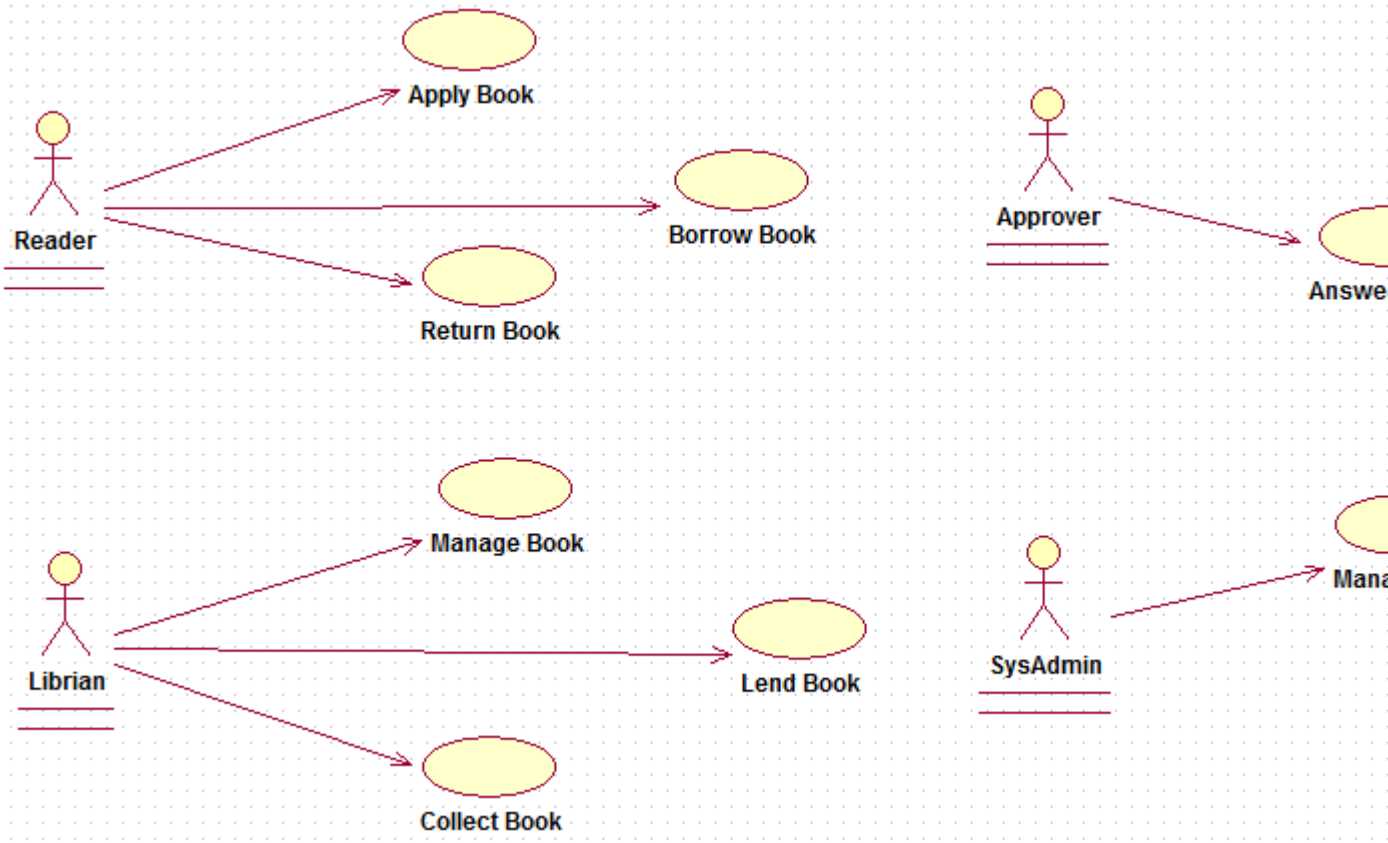
3、根据 struts.xml 配置，找到需要调用的 Action 类和方法， 并通过 IoC 方式，将值注入给 Aciton。 4、Action 调用业务逻辑组件处理业务逻辑，这一步包含表单验证。 5、Action 执行完毕，根据 struts.xml 中的配置找到对应的返回结果 result，并跳转到相应页面。 6、返回 HTTP 响应到客户端浏览器。

UML

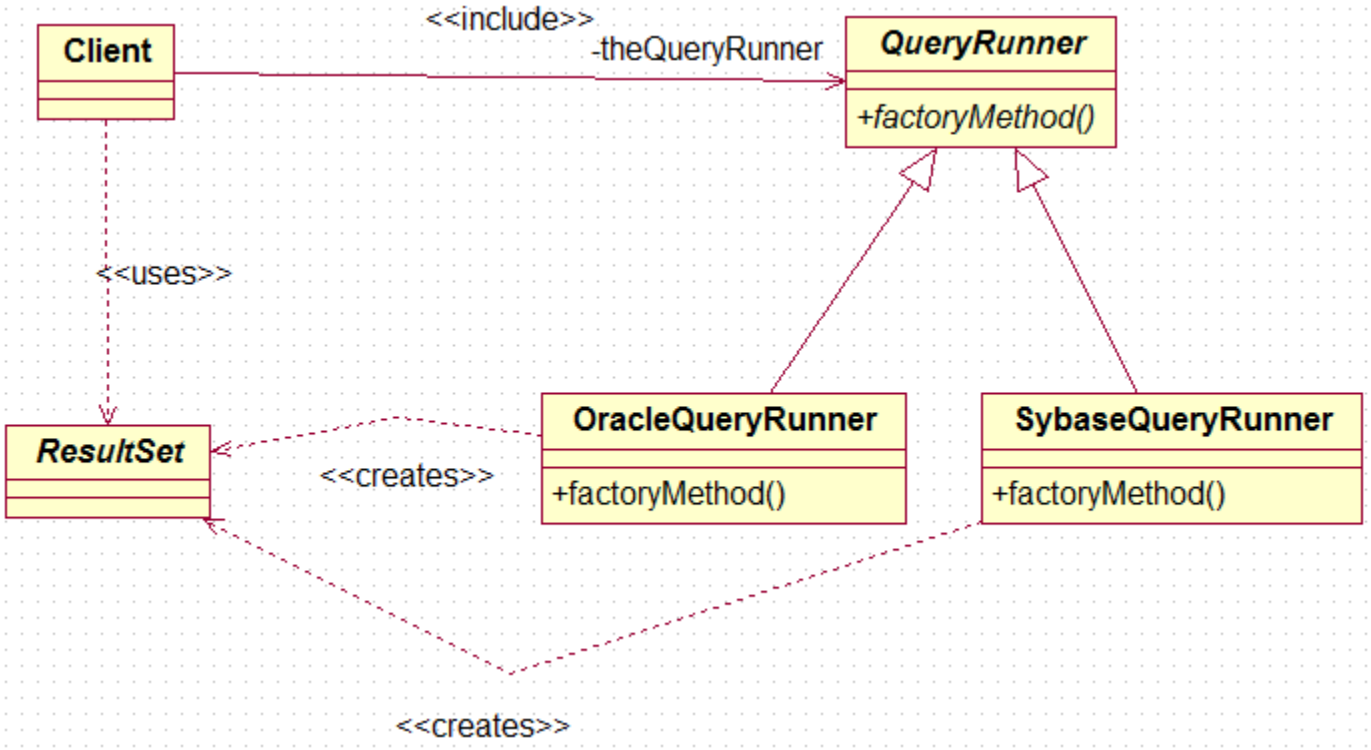
1. UML 是什么？UML 中有哪些图？

答：UML 是统一建模语言（Unified Modeling Language）的缩写，它发表于 1997 年，综合了当时已经存在的面向对象的建模语言、方法和过程，是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持。使用 UML 可以帮助沟通与交流，辅助应用设计和文档的生成，还能够阐释系统的结构和行为。UML 定义了多种图形化的符号来描述软件系统部分或全部的静态结构和动态结构，包括：用例图（use case diagram）、类图（class diagram）、时序图（sequence diagram）、协作图（collaboration diagram）、状态图（statechart diagram）、活动图（activity diagram）、构件图（component diagram）、部署图（deployment diagram）等。在这些图形化符号中，有三种图最为重要，分别是：用例图（用来捕获需求，描述系统的功能，通过该图可以迅速的了解系统的功能模块及其关系）、类图（描述类以及类与类之间的关系，通过该图可以快速了解系统）、时序图（描述执行特定任务时对象之间的交互关系以及执行顺序，通过该图可以了解对象能接收的消息也就是说对象能够向外界提供的服务）。

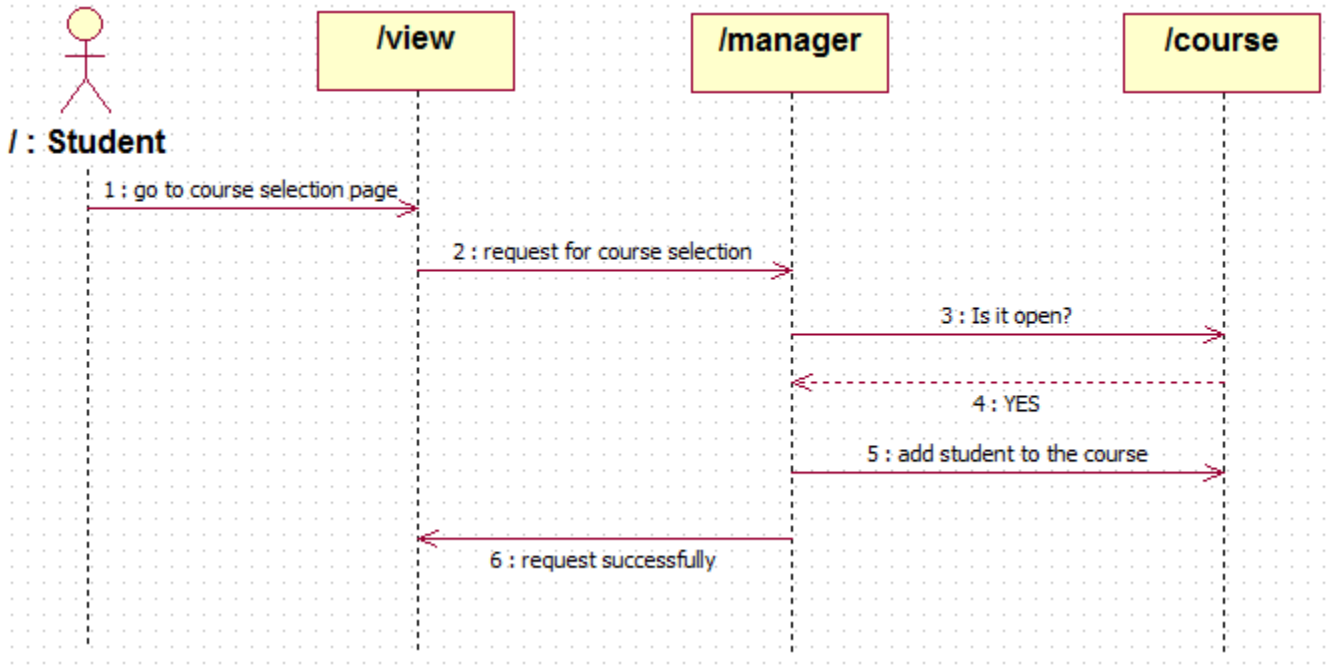
用例图：



类图：



时序图：



2. 类图用来表示系统中类和类与类之间的关系，它是对系统动态结构的描述。**(**选择 **1** 项**)**

A.正确

B.不正确

解答：B

类图是对系统静态结构的描述。

常见设计模式

1. 写一个单例类。

答：单例模式主要作用是保证在 Java 应用程序中，一个类只有一个实例存在。下面给出两种不同形式的单例：

第一种形式：饿汉式单例

```
public class Singleton {
    private Singleton(){}
    private static Singleton instance = new Singleton();
    public static Singleton getInstance(){
        return instance;
    }
}
```

第二种形式：懒汉式单例

```
public class Singleton {
    private static Singleton instance = null;
    private Singleton() {}
    public static synchronized Singleton getInstance(){
        if (instance==null) instance = newSingleton();
        return instance;
    }
}
```

单例的特点：外界无法通过构造器来创建对象，该类必须提供一个静态方法向外界提供该类的唯一实例。

【补充】用 Java 进行服务器端编程时，使用单例模式的机会还是很多的，服务器上的资源都是很宝贵的，对于那些无状态的对象其实都可以单例化或者静态化（在内存中仅有唯一拷贝），如果使用了 Spring 这样的框架来进行对象托管，Spring 的 IoC 容器在默认情况下对所有托管对象都是进行了单例化处理的。

2. 说说你所熟悉或听说过的设计模式以及你对设计模式的看法。

答：在 GoF 的《Design Patterns: Elements of Reusable Object-Oriented Software》中给出了三类（创建型[对类的实例化过程的抽象化]、结构型[描述如何将类或对象结合在一起形成更大的结构]、行为型[对在不同的对象之间划分责任和算法的抽象化]）共 23 种设计模式，包括：Abstract Factory（抽象工厂模式），Builder（建造者模式），Factory Method（工厂方法模式），Prototype（原始模型模式），Singleton（单例模式）；Facade（门面模式），Adapter（适配器模式），Bridge（桥梁模式），Composite（合成模式），Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式）；Command（命令模式），Interpreter（解释器模式），Visitor（访问者模式），Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式），Observer（观察者模式），State（状态模式），Strategy（策略模式），Template Method（模板方法模式）， Chain Of Responsibility（责任链模式）。

所谓设计模式，就是一套被反复使用的代码设计经验的总结（情境中一个问题经过证实的一个解决方案）。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。设计模式使人们可以更加简单方便的复用成功的设计和体系结构。将已证实的技术表述成设计模式也会使新系统开发者更加容易理解其设计思路。

3. 你在开发中都用到了那些设计模式？用在什么场合？

答：面试被问到关于设计模式的知识时，可以拣最常用的作答，例如：

1)工厂模式：工厂类可以根据条件生成不同的子类实例，这些子类有一个公共的抽象父类并且实现了相同的方法，但是这些方法针对不同的数据进行了不同的操作（多态方法）。当得到子类的实例后，开发人员可以调用基类中的方法而不必考虑到底返回的是哪一个子类的实例。

2)代理模式：给一个对象提供一个代理对象，并由代理对象控制原对象的引用。实际开发中，按照使用目的的不同，代理可以分为：远程代理、虚拟代理、保护代理、Cache 代理、防火墙代理、同步化代理、智能引用代理。

3)适配器模式：把一个类的接口变换成客户端所期待的另一种接口，从而使原本因接口不匹配而无法在一起使用的类能够一起工作。

4)模板方法模式：提供一个抽象类，将部分逻辑以具体方法或构造器的形式实现，然后声明一些抽象方法来迫使子类实现剩余的逻辑。不同的子类可以以不同的方式实现这些抽象方法（多态实现），从而实现不同的业务逻辑。

除此之外，还可以讲讲上面提到的门面模式、桥梁模式、单例模式、装潢模式（Collections 工具类里面的 synchronizedXXX 方法把一个线程不安全的容器变成线程安全容器就是对装潢模式的应用，而 Java IO 里面的过滤流（有的翻译成处理流）也是应用装潢模式的经典例子）等，反正原则就是拣自己最熟悉的用得最多的作答，以免言多必失。

4. 编程题: 写一个 Singleton 出来

Singleton 模式主要作用是保证在 Java 应用程序中，一个类 Class 只有一个实例存在。 一般 Singleton 模式通常有几种种形式:

第一种形式: 定义一个类，它的构造函数为 private 的，它有一个 static 的 private 的该类变量，在类初始化时实例话，通过一个 public 的 getInstance 方法获取对它的引用,继而调用其中的方法。

```
public class Singleton {
private Singleton(){}
     //在自己内部定义自己一个实例，是不是很奇怪？
     //注意这是 private 只供内部调用
     private static Singleton instance = new Singleton();
     //这里提供了一个供外部访问本 class 的静态方法，可以直接访问
     public static Singleton getInstance() {
         return instance;
     }
}
第二种形式:
public class Singleton {
    private static Singleton instance = null;
    public static synchronized Singleton getInstance() {
    //这个方法比上面有所改进，不用每次都进行生成对象，只是第一次
    //使用时生成实例，提高了效率！
    if (instance==null)
        instance=new Singleton();
return instance;     }
}
```

其他形式:

定义一个类，它的构造函数为 private 的，所有方法为 static 的。

一般认为第一种形式要更加安全些