Java 面试题库

牛客网出品

Java 工程师校招面试题库导读

一、学习说明

本题库均来自海量真实校招面试题目大数据进行的整理，后续也会不断更新，可免费在线观看，如需下载，也可在页面 https://www.nowcoder.com/interview/center 直接进行**下载**（下载需要用牛币兑换，一次兑换可享受永久下载权限，因为后续会更新）

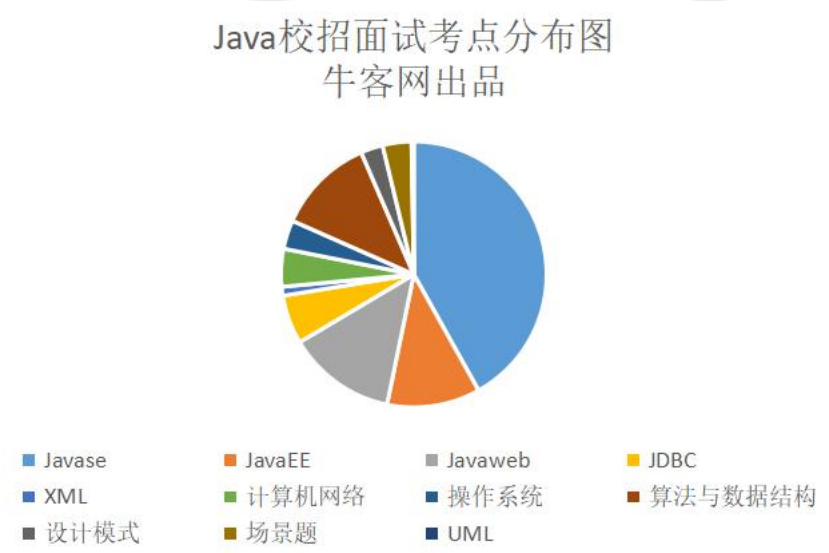
需要严肃说明的是：面试题库作为帮助同学准备面试的辅助资料，但是绝对不能作为备考唯一途径，因为面试是一个考察真实水平的，不是背会了答案就可以的，需要你透彻理解的，否则追问问题答不出来反而减分，毕竟技术面试中面试官最痛恨的就是背答案这个事情了。

学完这个题库，把此题库都理解透彻应对各家企业面试完全没有问题。（当然，要加上好的项目以及透彻掌握）

另外，此面试题库中不包括面试中问到的项目，hr 面以及个人技术发展类。

- 项目是比较个性化的，没办法作为一个题库来给大家参考，但是如果你有一个非常有含金量的项目的话，是非常加分的，而且你的项目可能也会被问的多一些；
- hr 面的话一般来说技术面通过的话个人没有太大的和公司不符合的问题都能通过；
- 技术发展类的话这个就完全看自己啦，主要考察的会是你对技术的热爱和学习能力，比如会问一些你是如何学习 xxx 技术的，或者能表达出你对技术的热爱的地方等等。此处不做赘述。

那么抛开这些，Java 工程师中技术面中考察的占比如下：



需要注意的是：此图不绝对，因为实际面试中面试官会根据你的简历去问，比如你的项目多

可能就问的项目多一些，或者你说哪里精通可能面试官就多去问你这些。而且此图是根据题库数据整理出来，并不是根据实际单场面试整理，比如基础部分不会考那么多，会从中抽着考

但是面试中必考的点且占比非常大的有 Java 基础和**算法**。
决定你是否能拿 sp offer（高薪 offer）以及是否进名企的是项目和**算法**。

可以看出，算法除了是面试必过门槛以外，更是决定你是否能进名企或高薪 offer 的决定性因素。

另外关于算法部分，想要系统的学习算法思想，实现高频面试题最优解等详细讲解的话可以报名算法名企校招冲刺班或算法高薪校招冲刺班，你将能学到更先进的算法思想以及又一套系统的校招高频题目的解题套路和方法论。

多出来的服务如下：

如果有什么问题，也可以加 qq 咨询 1440073724，如果是早鸟的话，还可以领取早鸟优惠哦

二、面试技巧

面试一般分为技术面和 hr 面，形式的话很少有群面，少部分企业可能会有一个交叉面，不过总的来说，技术面基本就是考察你的专业技术水平的，hr 面的话主要是看这个人的综合素质以及家庭情况符不符合公司要求，一般来讲，技术的话只要通过了技术面 hr 面基本上是没有问题（也有少数企业 hr 面会刷很多人）

那我们主要来说技术面，技术面的话主要是考察专业技术知识和水平，我们是可以有一定的技巧的，但是一定是基于有一定的能力水平的。

所以也慎重的告诉大家，技巧不是投机取巧，是起到辅助效果的，技术面最主要的还是要有实力，这里是基于实力水平之上的技巧。

这里告诉大家面试中的几个技巧：

1、简历上做一个引导：
在词汇上做好区分，比如熟悉 Java，了解 python，精通 c 语言

这样的话对自己的掌握程度有个区分，也好让面试官有个着重去问，python 本来写的也只是了解，自然就不会多问你深入的一些东西了。

2、在面试过程中做一个引导：
面试过程中尽量引导到自己熟知的一个领域，比如问到你说一下 DNS 寻址，然后你简单回答（甚至这步也可以省略）之后，可以说一句，自己对这块可能不是特别熟悉，对计算机网络中的运输层比较熟悉，如果有具体的，甚至可以再加一句，比如 TCP 和 UDP

这样的话你可以把整个面试过程往你熟知的地方引导，也能更倾向于体现出你的优势而不是劣势，但是此方法仅限于掌握合适的度，比如有的知识点是必会的而你想往别处引就有点说不过去了，比如让你说几个 Java 的关键字，你一个也说不上来，那可能就真的没辙了。

3、在自我介绍中做一个引导：
一般面试的开头都会有一个自我介绍，在这个位置你也可以尽情的为自己的优势方面去引导。

4、面试过程中展示出自信：
面试过程中的态度也要掌握好，不要自卑，也不要傲娇，自信的回答出每个问题，尤其遇到不会的问题，要么做一些引导，实在不能引导也可以先打打擦边球，和面试官交流一下问题，看起来像是没听懂题意，这个过程也可以再自己思考一下，如果觉得这个过程可以免了的话也直接表明一下这个地方不太熟悉或者还没有掌握好，千万不要强行回答。

面试前的准备：

最重要的肯定是系统的学习了，有一个知识的框架，基础知识的牢靠程度等。
其中算法尤其重要，越来越多公司还会让你现场或者视频面试中手写代码；
另一大重要的和加分项就是项目，在面试前，一定要练习回答自己项目的三个问题：

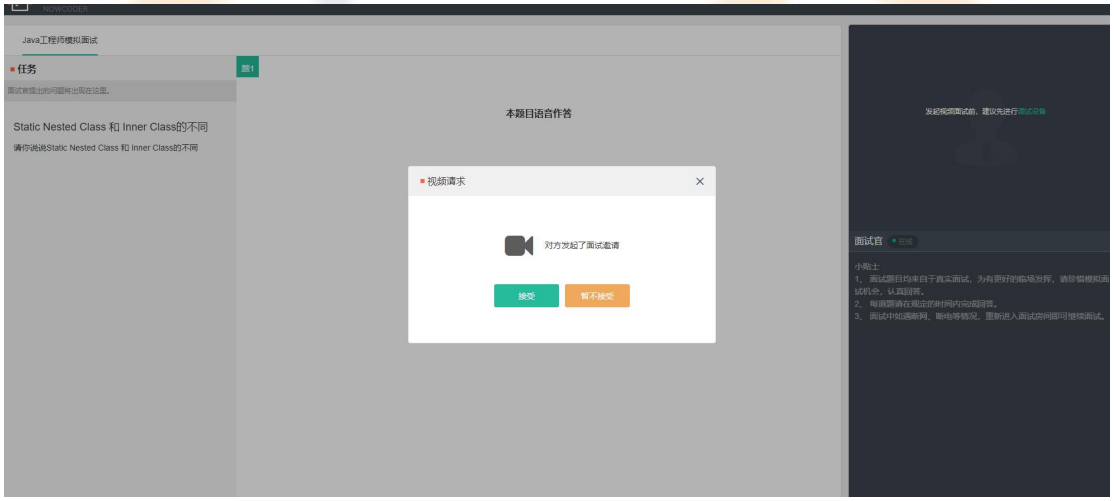
- 这是一个怎样的项目
- 用到了什么技术，为什么用这项技术（以及每项技术很细的点以及扩展）
- 过程中遇到了什么问题，怎么解决的。

那么话说回来，这个的前提是你要有一个好的项目，牛客网 CEO 叶向宇有带大家做项目，感兴趣的可以去了解一下

- 竞争力超过 70%求职者的项目：https://www.nowcoder.com/courses/semester/medium
  （专属优惠码：DjPgy3x，每期限量前 100 个）
- 竞争力超过 80%求职者的项目：https://www.nowcoder.com/courses/semester/senior
  （专属优惠码：DMVSexJ，每期限量前 100 个）

知识都掌握好后，剩下的就是一个心态和模拟练习啦，因为你面试的少的话现场难免紧张，而且没在那个环境下可能永远不知道自己回答的怎么样。

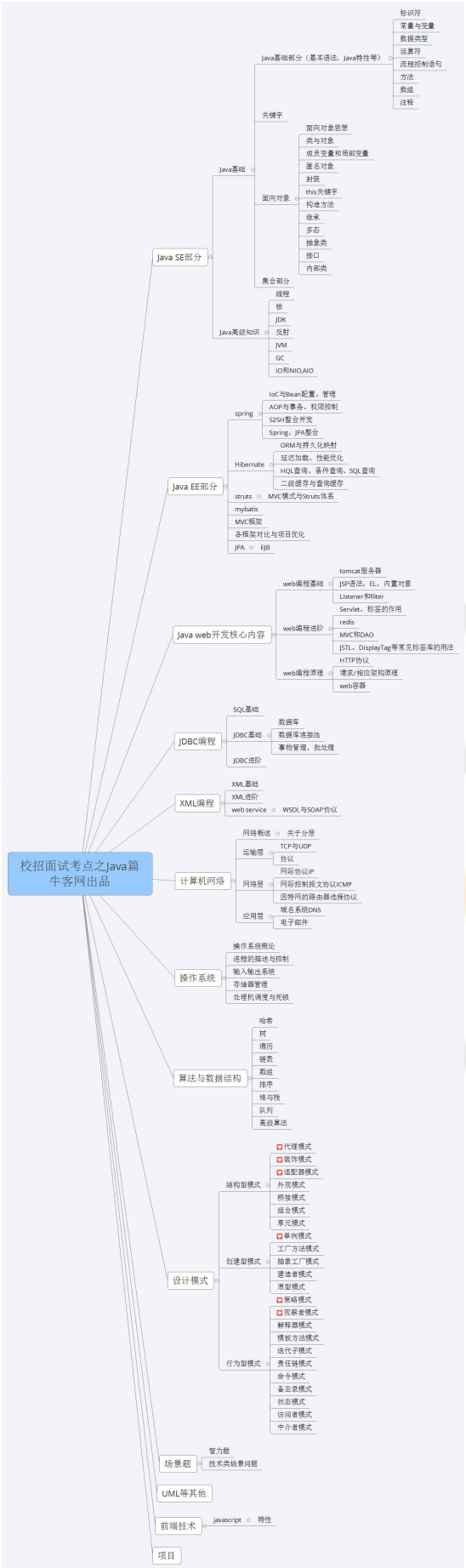
因为哪怕当你都会了的情况下，你的表达和心态就显得更重要了，会了但是没有表达的很清晰就很吃亏了，牛客网这边有 AI 模拟面试，完全模拟了真实面试环境，正好大家可以真正的去练习一下，还能收获一份面试报告：
https://www.nowcoder.com/interview/ai/index



面试后需要做的：
面试完了的话就不用太在意结果了，有限的时间就应该做事半功倍的事情，当然，要保持电话邮箱畅通，不然别给你发 offer 你都不知道。

抛开这些，我们需要做的是及时将面试中的问题记录下来，尤其是自己回答的不够好的问题，一定要花时间去研究，并解决这些问题，下次面试再遇到相同的问题就能很好的解决，当然，即使不遇到，你这个习惯坚持住，后面也可以作为一个经历去跟面试官说，能表现出你对技术的喜爱和钻研的一个态度，同时，每次面试后你会发现自己的不足，查缺补漏的好机会，及时调整，在不断的调整和查缺补漏的过程中，你会越来越好。

三、面试考点导图


校招面试考点之Java篇
牛客网出品
Java SE部分
Java基础
Java基础部分（基本语法，Java特性等）
标识符
常量与变量
数据类型
运算符
流程控制语句
方法
数组
注释
关键字
面向对象
面向对象思想
类与对象
成员变量和局部变量
匿名对象
封装
this关键字
构造方法
继承
多态
抽象类
接口
内部类
集合部分
Java高级知识
线程
锁
JDK
反射
JVM
GC
IO和NIO,AIO
Java EE部分
spring
IoC与Bean配置、管理
AOP与事务、权限控制
S2SH整合开发
Spring、JPA整合
Hibernate
ORM与持久化映射
延迟加载、性能优化
HQL查询、条件查询、SQL查询
二级缓存与查询缓存
struts
MVC模式与Struts体系
mybatis
MVC框架
各框架对比与项目优化
JPA
EJB
Java web开发核心内容
web编程基础
tomcat服务器
JSP语法，EL，内置对象
Listener和filter
Servlet、标签的作用
web编程进阶
redis
MVC和DAO
JSTL、DisplayTag等常见标签库的用法
web编程原理
HTTP协议
请求/相应架构原理
web容器
JDBC编程
SQL基础
JDBC基础
数据库
数据库连接池
事物管理，批处理
JDBC进阶
XML编程
XML基础
XML进阶
web service
WSDL与SOAP协议
计算机网络
网络概述
关于分层
运输层
TCP与UDP
协议
网络层
网际协议IP
网际控制报文协议ICMP
因特网的路由器选择协议
应用层
域名系统DNS
电子邮件
操作系统
操作系统概论
进程的描述与控制
输入输出系统
存储器管理
处理机调度与死锁
算法与数据结构
哈希
树
遍历
链表
数组
排序
堆与栈
队列
高级算法
设计模式
结构型模式
代理模式
装饰模式
适配器模式
外观模式
桥接模式
组合模式
享元模式
创建型模式
单例模式
工厂方法模式
抽象工厂模式
建造者模式
原型模式
行为型模式
策略模式
观察者模式
解释器模式
模板方法模式
迭代子模式
责任链模式
命令模式
备忘录模式
状态模式
访问者模式
中介者模式
场景题
智力题
技术类场景问题
UML等其他
前端技术
javascript
特性
项目

四、一对一答疑讲解戳这里

如果你对校招求职或者职业发展很困惑，欢迎与牛客网专业老师沟通，老师会帮你一对一讲解答疑哦（可以扫下方二维码或者添加微信号：niukewang985）

目录

②JSP 语法，EL，内置对象
③Listener 和 filter

2、Web 编程进阶
①Servlet、标签的作用
②redis
③MVC 和 DAO
④JSTL、DisplayTag 等常见标签库的用法

3、Web 编程原理
① HTTP 协议
②请求/相应架构原理
③web 容器

四、JDBC 编程
1、SQL 基础
2、JDBC 基础
①数据库
②数据库连接池
③事物管理，批处理
3、JDBC 进阶

五、XML 编程
1、XML 基础
2、XML 进阶
3、Web service
①WSDL 与 SOAP 协议

六、计算机网络
1、网络概述
①关于分层

2、运输层
①TCP 与 UDP
②协议

3、网络层
①网际协议 IP
②网际控制报文协议 ICMP
③因特网的路由器选择协议

4、应用层
①域名系统 DNS
②电子邮件

七、操作系统
1、操作系统概论
2、进程的描述与控制
3、输入输出系统
4、存储器管理
5、处理机调度与死锁

八、算法与数据结构
1、哈希
2、树
3、遍历
4、链表
5、数组
6、排序
7、堆与栈
8、队列
9、高级算法

九、设计模式
1、结构型模式
①代理模式
②装饰模式
③适配器模式

2、创建型模式
①单例模式

3、行为型模式
①策略模式
②观察者模式

4、所有模式汇总

十、场景题
十一、UML

一、JavaSE 部分

1、Java 基础

①Java 基础部分（基本语法，Java 特性等）

1、为什么重写 equals 还要重写 hashcode？

考点： java 基础

参考回答：

HashMap 中，如果要比较 key 是否相等，要同时使用这两个函数！因为自定义的类的 hashcode()方法继承于 Object 类，其 hashcode 码为默认的内存地址，这样即便有相同含义的两个对象，比较也是不相等的。HashMap 中的比较 key 是这样的，先求出 key 的 hashcode()，比较其值是否相等，若相等再比较 equals()，若相等则认为他们是相等的。若 equals()不相等则认为他们不相等。如果只重写 hashcode()不重写 equals()方法，当比较 equals()时只是看他们是否为同一对象（即进行内存地址的比较），所以必定要两个方法一起重写。HashMap 用来判断 key 是否相等的方法，其实是调用了 HashSet 判断加入元素 是否相等。重载 hashCode()是为了对同一个 key，能得到相同的 Hash Code，这样 HashMap 就可以定位到我们指定的 key 上。重载 equals()是为了向 HashMap 表明当前对象和 key 上所保存的对象是相等的，这样我们才真正地获得了这个 key 所对应的这个键值对。

2、说一下 map 的分类和常见的情况

考点： java 基础

参考回答：

java 为数据结构中的映射定义了一个接口 java.util.Map;它有四个实现类，分别是 HashMap Hashtable LinkedHashMap 和 TreeMap.

Map 主要用于存储健值对，根据键得到值，因此不允许键重复(重复了覆盖了)，但允许值重复。

Hashmap 是一个最常用的 Map，它根据键的 HashCode 值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。 HashMap 最多只允许一条记录的键为 Null;允许多条记录的值为 Null;HashMap 不支持线程的同步，即任一时刻可以有多个线程同时写 HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections 的 synchronizedMap 方法使 HashMap 具有同步的能力，或者使用 ConcurrentHashMap。

Hashtable 与 HashMap 类似，它继承自 Dictionary 类，不同的是:它不允许记录的键或者值为空;它支持线程的同步，即任一时刻只有一个线程能写 Hashtable，因此也导致了 Hashtable 在写入时会比较慢。

LinkedHashMap 是 HashMap 的一个子类，保存了记录的插入顺序，在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比 HashMap 慢，不过有种情况例外，当 HashMap 容量很大，实际数据较少时，遍历起来可能会比 LinkedHashMap 慢，因为 LinkedHashMap 的遍历速度只和实际数据有关，和容量无关，而 HashMap 的遍历速度和他的容量有关。

TreeMap 实现 SortMap 接口，能够把它保存的记录根据键排序,默认是按键值的升序排序，也可以指定排序的比较器，当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。

一般情况下，我们用的最多的是 HashMap,在 Map 中插入、删除和定位元素，HashMap 是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么 TreeMap 会更好。如果需要输出的顺序和输入的相同,那么用 LinkedHashMap 可以实现,它还可以按读取顺序来排列.

HashMap 是一个最常用的 Map，它根据键的 hashCode 值存储数据，根据键可以直接获取它的值，具有很快的访问速度。HashMap 最多只允许一条记录的键为 NULL，允许多条记录的值为 NULL。

HashMap 不支持线程同步，即任一时刻可以有多个线程同时写 HashMap，可能会导致数据的不一致性。如果需要同步，可以用 Collections 的 synchronizedMap 方法使 HashMap 具有同步的能力。

Hashtable 与 HashMap 类似，不同的是：它不允许记录的键或者值为空；它支持线程的同步，即任一时刻只有一个线程能写 Hashtable，因此也导致了 Hashtable 在写入时会比较慢。

LinkedHashMap 保存了记录的插入顺序，在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的。

在遍历的时候会比 HashMap 慢 TreeMap 能够把它保存的记录根据键排序,默认是按升序排序，也可以指定排序的比较器。当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。

3、Object 若不重写 hashCode()的话，hashCode()如何计算出来的？

考点：基础

参考回答：

Object 的 hashcode 方法是本地方法，也就是用 c 语言或 c++ 实现的，该方法直接返回对象的 内存地址。

4、==比较的是什么？

考点：基础

参考回答：

“==”对比两个对象基于内存引用，如果两个对象的引用完全相同（指向同一个对象）时，“==”操作将返回 true，否则返回 false。“==”如果两边是基本类型，就是比较数值是否相等。

5、若对一个类不重写，它的 equals()方法是如何比较的？

考点：基础

参考回答：

比较是对象的地址。

6、java8 新特性

考察点：java8

参考回答：

Lambda 表达式 - Lambda 允许把函数作为一个方法的参数（函数作为参数传递进方法中。

方法引用 - 方法引用提供了非常有用的语法，可以直接引用已有 Java 类或对象（实例）的方法或构造器。与 lambda 联合使用，方法引用可以使语言的构造更紧凑简洁，减少冗余代码。

默认方法 - 默认方法就是一个在接口里面有了一个实现的方法。

新工具 - 新的编译工具，如：Nashorn 引擎 jjs、 类依赖分析器 jdeps。

Stream API -新添加的 Stream API（java.util.stream） 把真正的函数式编程风格引入到 Java 中。

Date Time API - 加强对日期与时间的处理。

Optional 类 - Optional 类已经成为 Java 8 类库的一部分，用来解决空指针异常。

Nashorn, JavaScript 引擎 - Java 8 提供了一个新的 Nashorn javascript 引擎，它允许我们在 JVM 上运行特定的 javascript 应用。

7、说说 Lamda 表达式的优缺点。

考察点：Java 基础

参考回答：

优点：1. 简洁。2. 非常容易并行计算。3. 可能代表未来的编程趋势。

缺点：1. 若不用并行计算，很多时候计算速度没有比传统的 for 循环快。（并行计算有时需要预热才显示出效率优势）2. 不容易调试。3. 若其他程序员没有学过 lambda 表达式，代码不容易让其他语言的程序员看懂。

8、一个十进制的数在内存中是怎么存的？

考察点：计算机基础

参考回答：

补码的形式。

9、为啥有时会出现 4.0-3.6=0.40000001 这种现象？

考察点：计算机基础

参考回答：

原因简单来说是这样：2 进制的小数无法精确的表达 10 进制小数，计算机在计算 10 进制小数的过程中要先转换为 2 进制进行计算，这个过程中出现了误差。

10、Java 支持的数据类型有哪些？什么是自动拆装箱？

考察点：JAVA 数据类型

参考回答：

Java 语言支持的 8 种基本数据类型是：
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。

11、什么是值传递和引用传递？

考察点：JAVA 引用传递

参考回答：

值传递是对基本型变量而言的,传递的是该变量的一个副本,改变副本不影响原变量.
引用传递一般是对于对象型变量而言的,传递的是该对象地址的一个副本，并不是原对象本身 。所以对引用对象进行操作会同时改变原对象.
一般认为,java 内的传递都是值传递.

12、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？

考察点：Array

参考回答：

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。

13、你了解大 0 符号(big-0 notation)么？你能给出不同数据结构的例子么？

考察点：JAVA notation

参考回答：

大 0 符号描述了当数据结构里面的元素增加的时候,算法的规模或者是性能在最坏的场景下有多么好。
大 0 符号也可用来描述其他的行为，比如：内存消耗。因为集合类实际上是数据结构，我们一般使用大 0 符号基于时间，内存和性能来选择最好的实现。大 0 符号可以对大量数据的性能给出一个很好的说明。

同时，大 0 符号表示一个程序运行时所需要的渐进时间复杂度上界。

其函数表示是：

对于函数 f(n),g(n),如果存在一个常数 c，使得 f(n)<=c*g(n),则 f(n)=0(g(n))；

大 0 描述当数据结构中的元素增加时，算法的规模和性能在最坏情景下有多好。

大 0 还可以描述其它行为，比如内存消耗。因为集合类实际上是数据结构，因此我们一般使用大 0 符号基于时间，内存，性能选择最好的实现。大 0 符号可以对大量数据性能给予一个很好的说明。

14、String 是最基本的数据类型吗?

考察点：数据类型

参考回答：

基本数据类型包括 byte、int、char、long、float、double、boolean 和 short。
java.lang.String 类是 final 类型的，因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用 StringBuffer 类。

15、int 和 Integer 有什么区别

考察点：数据类型

参考回答：

Java 提供两种不同的类型：引用类型和原始类型（或内置类型）。Int 是 java 的原始数据类型，Integer 是 java 为 int 提供的封装类。
Java 为每个原始类型提供了封装类。
原始类型封装类
booleanBoolean
charCharacter
byteByte
shortShort
intInteger
longLong
floatFloat
doubleDouble
引用类型和原始类型的行为完全不同，并且它们具有不同的语义。引用类型和原始类型具有不同的特征和用法，它们包括：大小和速
度问题，这种类型以哪种类型的数据结构存储，当引用类型和原始类型用作某个类的实例数据时所指定的缺省值。对象引用实例变量
的缺省值为 null，而原始类型实例变量的缺省值与它们的类型有关。

16、String 和 StringBuffer 的区别

考察点：数据类型

参考回答：

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。

17、我们在 web 应用开发过程中经常遇到输出某种编码的字符，如 iso8859-1 等，如何输出一个某种编码的字符串？

考察点：数据类型

参考回答：

```
Public String translate (String str) {
String tempStr = “”;
try {
tempStr = new String(str.getBytes(“ISO-8859-1″), “GBK”);
tempStr = tempStr.trim();
}
catch (Exception e) {
```

```
System.err.println(e.getMessage());
}
return tempStr;
}
```

18、int 和 Integer 有什么区别？

考察点：数据类型

参考回答：

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了基本数据类型，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 Java 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

如：

```
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
    }
}
```

19、&和&&的区别？

考察点：运算符

参考回答：

&运算符有两种用法：(1)按位与；(2)逻辑与。&&运算符是短路与运算。逻辑与跟短路与的差别是非常巨大的，虽然二者都要求运算符左右两端的布尔值都是 true 整个表达式的值才是

true。&&之所以称为短路运算是因为，如果&&左边的表达式的值是 false，右边的表达式会被直接短路掉，不会进行运算。很多时候我们可能都需要用&&而不是&，例如在验证用户登录时判定用户名不是 null 而且不是空字符串，应当写为：username != null &&!username.equals(""),二者的顺序不能交换，更不能用&运算符，因为第一个条件如果不成立，根本不能进行字符串的 equals 比较，否则会产生 NullPointerException 异常。

20、在 Java 中，如何跳出当前的多重嵌套循环？

考察点：循环

参考回答：

在最外层循环前加一个标记如 A，然后用 break A;可以跳出多重循环。（Java 中支持带标签的 break 和 continue 语句,作用有点类似于 C 和 C++中的 goto 语句,但是就像要避免使用 goto 一样，应该避免使用带标签的 break 和 continue，因为它不会让你的程序变得更优雅，很多时候甚至有相反的作用，所以这种语法其实不知道更好）

21、你能比较一下 Java 和 JavaSciprt 吗？

考察：Java&JavaScript

参考回答：

JavaScript 与 Java 是两个公司开发的不同的两个产品。Java 是原 Sun Microsystems 公司推出的面向对象的程序设计语言，特别适合于互联网应用程序开发；而 JavaScript 是 Netscape 公司的产品，为了扩展 Netscape 浏览器的功能而开发的一种可以嵌入 Web 页面中运行的基于对象和事件驱动的解释性语言。JavaScript 的前身是 LiveScript；而 Java 的前身是 Oak 语言。
下面对两种语言间的异同作如下比较：
- 基于对象和面向对象：Java 是一种真正的面向对象的语言，即使是开发简单的程序，必须设计对象；JavaScript 是种脚本语言，它可以用来制作与网络无关的，与用户交互作用的复杂软件。它是一种基于对象（Object-Based）和事件驱动（Event-Driven）的编程语言，因而它本身提供了非常丰富的内部对象供设计人员使用。
- 解释和编译：Java 的源代码在执行之前，必须经过编译。JavaScript 是一种解释性编程语言，其源代码不需经过编译，由浏览器解释执行。（目前的浏览器几乎都使用了 JIT（即时编译）技术来提升 JavaScript 的运行效率）
- 强类型变量和类型弱变量：Java 采用强类型变量检查，即所有变量在编译之前必须作声明；JavaScript 中变量是弱类型的，甚至在使用变量前可以不作声明，JavaScript 的解释器在运行时检查推断其数据类型。
- 代码格式不一样。

22、简述正则表达式及其用途。

考察点：正则表达式

参考回答：

在编写处理字符串的程序时，经常会有查找符合某些复杂规则的字符串的需要。正则表达式就是用于描述这些规则的工具。换句话说，正则表达式就是记录文本规则的代码。计算机处理的信息更多的时候不是数值而是字符串，正则表达式就是在进行字符串匹配和处理的时候最为强大的工具，绝大多数语言都提供了对正则表达式的支持。

23、Java 中是如何支持正则表达式操作的？

考察点：正则表达式

参考回答：

Java 中的 String 类提供了支持正则表达式操作的方法，包括：matches()、replaceAll()、replaceFirst()、split()。此外，Java 中可以用 Pattern 类表示正则表达式对象，它提供了丰富的 API 进行各种正则表达式操作，如：

```
    import java.util.regex.Matcher;
import java.util.regex.Pattern;
class RegExpTest {
       public static void main(String[] args) {
               String str = "成都市(成华区)(武侯区)(高新区)";
               Pattern p = Pattern.compile(".*?(?=\\()");
               Matcher m = p.matcher(str);
               if(m.find()) {
                       System.out.println(m.group());
               }
       }
}
```

24、请你说说 Java 和 PHP 的区别？

考察点：Java 特性

参考回答：

PHP 暂时还不支持像 Java 那样 JIT 运行时编译热点代码,但是 PHP 具有 opcache 机制,能够把脚本对应的 opcode 缓存在内存,PHP7 中还支持配置 opcache.file_cache 导出 opcode 到文件.第三方的 Facebook HHVM 也支持 JIT.另外 PHP 官方基于 LLVM 围绕 opcache 机制构建的 Zend JIT 分支也正在开发测试中.在 php-src/Zend/bench.php 测试显示,PHP JIT 分支速度是 PHP 5.4 的 10 倍.
PHP 的库函数用 C 实现,而 Java 核心运行时类库(jdk/jre/lib/rt.jar,大于 60MB)用 Java 编写(jdk/src.zip)，所以 Java 应用运行的时候,用户编写的代码以及引用的类库和框架都要在 JVM 上解释执行. Java 的 HotSpot 机制,直到有方法被执行 10000 次才会触发 JIT 编译，在此之前运行在解释模式下,以避免出现 JIT 编译花费的时间比方法解释执行消耗的时间还要多的情况.

PHP 内置模板引擎,自身就是模板语言.而 Java Web 需要使用 JSP 容器如 Tomcat 或第三方模板引擎.

PHP 也可以运行在多线程模式下,比如 Apache 的 event MPM 和 Facebook 的 HHVM 都是多线程架构.不管是多进程还是多线程的 PHP Web 运行模式,都不需要 PHP 开发者关心和控制,也就是说 PHP 开发者不需要写代码参与进程和线程的管理,这些都由 PHP-FPM/HHVM/Apache 实现.PHP-FPM 进程管理和并发实现并不需要 PHP 开发者关心,而 Java 多线程编程需要 Java 开发者编码参与.PHP 一个 worker 进程崩溃,master 进程会自动新建一个新的 worker 进程,并不会导致 PHP 服务崩溃.而 Java 多线程编程稍有不慎(比如没有捕获异常)就会导致 JVM 崩溃退出.对于 PHP-FPM 和 Apache MOD_PHP 来说,服务进程常驻内存,但一次请求释放一次资源,这种内存释放非常彻底.PHP 基于引用计数的 GC 甚至都还没发挥作用程序就已经结束了。

②关键字

1、介绍一下 Syncronized 锁，如果用这个关键字修饰一个静态方法，锁住了什么？如果修饰成员方法，锁住了什么？

考点：java 关键字

参考回答：

synchronized 修饰静态方法以及同步代码块的 synchronized (类.class)用法锁的是类，线程想要执行对应同步代码，需要获得类锁。

synchronized 修饰成员方法，线程获取的是当前调用该方法的对象实例的对象锁。

2、介绍一下 volatile?

考察点：java 关键字

参考回答：

volatile 关键字是用来保证有序性和可见性的。这跟 Java 内存模型有关。比如我们所写的代码，不一定是按照我们自己书写的顺序来执行的，编译器会做重排序，CPU 也会做重排序的，这样的重排序是为了减少流水线的阻塞的，引起流水阻塞，比如数据相关性，提高 CPU 的执行效率。需要有一定的顺序和规则来保证，不然程序员自己写的代码都不知带对不对了，所以有 happens-before 规则，其中有条就是 volatile 变量规则：对一个变量的写操作先行发生于后面对这个变量的读操作；有序性实现的是通过插入内存屏障来保证的。可见性：首先 Java 内存模型分为，主内存，工作内存。比如线程 A 从主内存把变量从主内存读到了自己的工作内存中，做了加 1 的操作，但是此时没有将 i 的最新值刷新会主内存中，线程 B 此时读到的还是 i 的旧值。加了 volatile 关键字的代码生成的汇编代码发现,会多出一个 lock 前缀指令。Lock 指令对 Intel 平台的 CPU，早期是锁总线，这样代价太高了，后面提出了缓存一致性协议，MESI，来保证了多核之间数据不一致性问题。

3、锁有了解嘛，说一下 Synchronized 和 lock

考察点：java 关键字

参考回答：

synchronized 是 Java 的关键字，当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。JDK1.5 以后引入了自旋锁、锁粗化、轻量级锁，偏向锁来有优化关键字的性能。

Lock 是一个接口，而 synchronized 是 Java 中的关键字，synchronized 是内置的语言实现；synchronized 在发生异常时，会自动释放线程占有的锁，因此不会导致死锁现象发生；而 Lock 在发生异常时，如果没有主动通过 unLock()去释放锁，则很可能造成死锁现象，因此使用 Lock 时需要在 finally 块中释放锁；Lock 可以让等待锁的线程响应中断，而 synchronized 却不行，使用 synchronized 时，等待的线程会一直等待下去，不能够响应中断；通过 Lock 可以知道有没有成功获取锁，而 synchronized 却无法办到。

4、讲一讲 Java 里面的 final 关键字怎么用的？

考察点：关键字

参考回答：

当用 final 修饰一个类时，表明这个类不能被继承。也就是说，如果一个类你永远不会让他被继承，就可以用 final 进行修饰。final 类中的成员变量可以根据需要设为 final，但是要注意 final 类中的所有成员方法都会被隐式地指定为 final 方法。

“使用 final 方法的原因有两个。第一个原因是把方法锁定，以防任何继承类修改它的含义；第二个原因是效率。在早期的 Java 实现版本中，会将 final 方法转为内嵌调用。但是如果方法过于庞大，可能看不到内嵌调用带来的任何性能提升。在最近的 Java 版本中，不需要使用 final 方法进行这些优化了。“

对于一个 final 变量，如果是基本数据类型的变量，则其数值一旦在初始化之后便不能更改；如果是引用类型的变量，则在对其初始化之后便不能再让其指向另一个对象。

③面向对象

1、wait 方法底层原理

考察点：基础

参考回答：

ObjectSynchronizer::wait 方法通过 object 的对象中找到 ObjectMonitor 对象调用方法 void ObjectMonitor::wait(jlong millis, bool interruptible, TRAPS)

通过 ObjectMonitor::AddWaiter 调用把新建立的 ObjectWaiter 对象放入到 _WaitSet 的队列的末尾中然后在 ObjectMonitor::exit 释放锁，接着 thread_ParkEvent->park 也就是 wait。

2、Java 有哪些特性，举个多态的例子。

考察点：语言特性

参考回答：

封装、继承、多态。多态：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

3、String 为啥不可变？

考察点：面向对象

参考回答：

不可变对象是指一个对象的状态在对象被创建之后就不再变化。不可改变的意思就是说：不能改变对象内的成员变量，包括基本数据类型的值不能改变，引用类型的变量不能指向其他的对象，引用类型指向的对象的状态也不能改变。

String 不可变是因为在 JDK 中 String 类被声明为一个 final 类,且类内部的 value 字节数组也是 final 的，只有当字符串是不可变时字符串池才有可能实现，字符串池的实现可以在运行时节约很多 heap 空间，因为不同的字符串变量都指向池中的同一个字符串；如果字符串是可变的则会引起很严重的安全问题,譬如数据库的用户名密码都是以字符串的形式传入来获得数据库的连接，或者在 socket 编程中主机名和端口都是以字符串的形式传入，因为字符串是不可变的，所以它的值是不可改变的，否则黑客们可以钻到空子改变字符串指向的对象的值造成安全漏洞；因为字符串是不可变的，所以是多线程安全的，同一个字符串实例可以被多个线程共享，这样便不用因为线程安全问题而使用同步，字符串自己便是线程安全的；因为字符串是不可变的所以在它创建的时候 hashcode 就被缓存了，不变性也保证了 hash 码的唯一性，不需要重新计算，这就使得字符串很适合作为 Map 的键，字符串的处理速度要快过其它的键对象，这就是 HashMap 中的键往往都使用字符串的原因。

4、类和对象的区别

考察点：面向对象

参考回答：

1.类是对某一类事物的描述，是抽象的；而对象是一个实实在在的个体，是类的一个实例。

比如：“人”是一个类，而“教师”则是“人”的一个实例。

2.对象是函数、变量的集合体；而类是一组函数和变量的集合体，即类是一组具有相同属性的对象集合体。

5、请列举你所知道的 Object 类的方法。

考察点：面向对象

参考回答：

Object()默认构造方法。clone() 创建并返回此对象的一个副本。equals(Object obj) 指示某个其他对象是否与此对象“相等”。finalize()当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。getClass()返回一个对象的运行时类。hashCode()返回该对象的哈希码值。 notify()唤醒在此对象监视器上等待的单个线程。 notifyAll()唤醒在此对象监视器上等待的所有线程。toString()返回该对象的字符串表示。wait()导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法。wait(long timeout)导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量。wait(long timeout, int nanos) 导致当前的线程等待，直到其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量。

6、重载和重写的区别？相同参数不同返回值能重载吗？

考察点：重载

参考回答：

重载(Overloading)

（1） 方法重载是让类以统一的方式处理不同类型数据的一种手段。多个同名函数同时存在，具有不同的参数个数/类型。

重载 Overloading 是一个类中多态性的一种表现。

（2） Java 的方法重载，就是在类中可以创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。

调用方法时通过传递给它们的不同参数个数和参数类型来决定具体使用哪个方法，这就是多态性。

（3） 重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。

重写（Overriding）

（1） 父类与子类之间的多态性，对父类的函数进行重新定义。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。在 Java 中，子类可继承父类中的方法，而不需要重新编写相同的方法。

但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。

方法重写又称方法覆盖。

（2）若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。

如需父类中原有的方法，可使用 super 关键字，该关键字引用了当前类的父类。

（3）子类函数的访问修饰权限不能少于父类的。

7、”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？

考察点：static 变量

参考回答：

“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。

8、String 能继承吗？

考察点：String

参考回答：

不能，char 数组用 final 修饰的。

9、StringBuffer 和 StringBuilder 有什么区别，底层实现上呢？

考察点：类

参考回答：

StringBuffer 线程安全，StringBuilder 线程不安全，底层实现上的话，StringBuffer 其实就是比 StringBuilder 多了 Synchronized 修饰符。

10、类加载机制，双亲委派模型，好处是什么？

考察点：类

参考回答：

某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。

使用双亲委派模型的好处在于 Java 类随着它的类加载器一起具备了一种带有优先级的层次关系。例如类 java.lang.Object，它存在在 rt.jar 中，无论哪一个类加载器要加载这个类，最终都是委派给处于模型最顶端的 Bootstrap ClassLoader 进行加载，因此 Object 类在程序的各种类加载器环境中都是同一个类。相反，如果没有双亲委派模型而是由各个类加载器自行加载的话，如果用户编写了一个 java.lang.Object 的同名类并放在 ClassPath 中，那系统中将会出现

多个不同的 Object 类，程序将混乱。因此，如果开发者尝试编写一个与 rt.jar 类库中重名的 Java 类，可以正常编译，但是永远无法被加载运行。

11、静态变量存在哪?

考察点：类

参考回答：

方法区

12、讲讲什么是泛型？

考察点：JAVA 泛型

参考回答：

泛型，即“参数化类型”。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？顾名思义，就是将类型由原来的具体的类型参数化，类似于方法中的变量参数，此时类型也定义成参数形式（可以称之为类型形参），然后在使用/调用时传入具体的类型（类型实参）。

```
public class GenericTest {


      public static void main(String[] args) {
          /*
          List list = new ArrayList();
          list.add("qqyumidi");
          list.add("corn");
          list.add(100);
          */


         List<String> list = new ArrayList<String>();
         list.add("qqyumidi");
         list.add("corn");
         //list.add(100);   // 1  提示编译错误
```

```
            for (int i = 0; i < list.size(); i++) {
                String name = list.get(i); // 2
                System.out.println("name:" + name);
            }
        }
    }
```

采用泛型写法后，在//1 处想加入一个 Integer 类型的对象时会出现编译错误，通过 List<String>，直接限定了 list 集合中只能含有 String 类型的元素，从而在//2 处无须进行强制类型转换，因为此时，集合能够记住元素的类型信息，编译器已经能够确认它是 String 类型了。

13、解释 extends 和 super 泛型限定符-上界不存下界不取

考察点：JAVA 泛型

参考回答：

（1）泛型中上界和下界的定义

上界 <? extend Fruit>

下界 <? super Apple>

（2）上界和下界的特点

上界的 list 只能 get，不能 add（确切地说不能 add 出除 null 之外的对象，包括 Object）

下界的 list 只能 add，不能 get

（3）示例代码

```
import java.util.ArrayList;
import java.util.List;

class Fruit {}
class Apple extends Fruit {}
```

```
class Jonathan extends Apple {}

class Orange extends Fruit {}


public class CovariantArrays {

  public static void main(String[] args) {

//上界

    List<? extends Fruit> flistTop = new ArrayList<Apple>();

    flistTop.add(null);

    //add Fruit 对象会报错

    //flist.add(new Fruit());

    Fruit fruit1 = flistTop.get(0);


//下界

    List<? super Apple> flistBottem = new ArrayList<Apple>();

    flistBottem.add(new Apple());

    flistBottem.add(new Jonathan());

    //get Apple 对象会报错

    //Apple apple = flistBottem.get(0);

  }

}
```

（4）上界 <? extend Fruit> ，表示所有继承 Fruit 的子类，但是具体是哪个子类，无法确定，所以调用 add 的时候，要 add 什么类型，谁也不知道。但是 get 的时候，不管是什么子类，不管追溯多少辈，肯定有个父类是 Fruit，所以，我都可以用最大的父类 Fruit 接着，也就是把所有的子类向上转型为 Fruit。

下界 <? super Apple>,表示 Apple 的所有父类,包括 Fruit,一直可以追溯到老祖宗 Object 。那么当我 add 的时候，我不能 add Apple 的父类，因为不能确定 List 里面存放的到底是哪个父类。但是我可以 add Apple 及其子类。因为不管我的子类是什么类型,它都可以向上转型为 Apple 及其所有的父类甚至转型为 Object 。但是当我 get 的时候，Apple 的父类这么多，我用什么接着呢，除了 Object，其他的都接不住。

所以，归根结底可以用一句话表示，那就是编译器可以支持向上转型，但不支持向下转型。具体来讲，我可以把 Apple 对象赋值给 Fruit 的引用，但是如果把 Fruit 对象赋值给 Apple 的引用就必须得用 cast。

14、是否可以在 static 环境中访问非 static 变量？

考察点：static 变量

参考回答：

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。

15、谈谈如何通过反射创建对象？

考察点：类

参考回答：

- 方法 1：通过类对象调用 newInstance()方法，例如：String.class.newInstance()
- 方法 2：通过类对象的 getConstructor()或 getDeclaredConstructor()方法获得构造器（Constructor）对象并调用其 newInstance()方法创建对象，例如：String.class.getConstructor(String.class).newInstance("Hello");

16、Java 支持多继承么？

考察点：JAVA 多继承

参考回答：

Java 中类不支持多继承，只支持单继承（即一个类只有一个父类）。 但是 java 中的接口支持多继承，，即一个子接口可以有多个父接口。（接口的作用是用来扩展对象的功能，一个子接口继承多个父接口，说明子接口扩展了多个功能，当类实现接口时，类就扩展了相应的功能）。

17、接口和抽象类的区别是什么？

考察点：抽象类

参考回答：

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
类可以不实现抽象类和接口声明的所有方法，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

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

考察点：comparable 接口

参考回答：

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。

19、面向对象的特征有哪些方面

考察点：JAVA 特征

参考回答：

(1)抽象：
抽象就是忽略一个主题中与当前目标无关的那些方面，以便更充分地注意与当前目标有关的方面。抽象并不打算了解全部问题，而只
是选择其中的一部分，暂时不用部分细节。抽象包括两个方面，一是过程抽象，二是数据抽象。
(2)继承：
继承是一种联结类的层次模型，并且允许和鼓励类的重用，它提供了一种明确表述共性的方法。对象的一个新类可以从现有的类中派
生，这个过程称为类继承。新类继承了原始类的特性，新类称为原始类的派生类（子类），而原始类称为新类的基类（父类）。派生
类可以从它的基类那里继承方法和实例变量，并且类可以修改或增加新的方法使之更适合特殊的需要。
(3)封装：
封装是把过程和数据包围起来，对数据的访问只能通过已定义的界面。面向对象计算始于这个基本概念，即现实世界可以被描绘成一
系列完全自治、封装的对象，这些对象通过一个受保护的接口访问其他对象。
(4) 多态性：
多态性是指允许不同类的对象对同一消息作出响应。多态性包括参数化多态性和包含多态性。多

态性语言具有灵活、抽象、行为共享
、代码共享的优势，很好的解决了应用程序函数同名问题。

20、final，finally，finalize 的区别。

考察点：声明

参考回答：

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。
finally 是异常处理语句结构的一部分，表示总是执行。
finalize 是 Object 类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源
回收，例如关闭文件等。

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

考察点：JAVA 多态

参考回答：

方法的重写 Overriding 和重载 Overloading 是 Java 多态性的不同表现。重写 Overriding 是父类与子类之间多态性的一种表现，重载
Overloading 是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写
(Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被”屏蔽”了。如果在一个类中定义
了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载
(Overloading)。Overloaded 的方法是可以改
变返回值的类型。

22、abstract class 和 interface 有什么区别?

考察点：抽象类

参考回答：

声明方法的存在而不去实现它的类被叫做抽象类（abstract class），它用于要创建一个体现某些基本行为的类，并为该类声明方法
，但不能在该类中实现该类的情况。不能创建 abstract 类的实例。然而可以创建一个变量，其类型是一个抽象类，并让它指向具体
子类的一个实例。不能有抽象构造函数或抽象静态方法。Abstract 类的子类为它们父类中的所有抽象方法提供实现，否则它们也是
抽象类为。取而代之，在子类中实现该方法。知道其行为的其它类可以在类中实现这些方法。
接口（interface）是抽象类的变体。在接口中，所有方法都是抽象的。多继承性可通过实现这样的接口而获得。接口中的所有方法

都是抽象的，没有一个有程序体。接口只可以定义 static final 成员变量。接口的实现与子类相似，除了该实现类不能从接口定义中
继承行为。当类实现特殊接口时，它定义（即将程序体给予）所有这种接口的方法。然后，它可以在实现了该接口的类的任何对象上
调用接口的方法。由于有抽象类，它允许使用接口名作为引用变量的类型。通常的动态联编将生效。引用可以转换到接口类型或从接
口类型转换，instanceof 运算符可以用来决定某对象的类是否实现了接口。

23、Static Nested Class 和 Inner Class 的不同

考察点：声明

参考回答：

Static Nested Class 是被声明为静态（static）的内部类，它可以不依赖于外部类实例被实例化。而通常的内部类需要在外部类实
例化后才能实例化。Static-Nested Class 的成员，既可以定义为静态的(static)，也可以定义为动态的(instance).Nested Class 的静态成员(Method)只能对 Outer Class 的静态成员(static memebr)进行操作(ACCESS)，而不能 Access Outer Class 的动态成员(instance member).而 Nested Class 的动态成员(instance method) 却可以 Access Outer Class 的所有成员，这个概念很重要，许多人对这个概念模糊.
有一个普通的原则，因为静态方法(static method) 总是跟 CLASS 相关联(bind CLASS)，而 动态方法((instance method) 总是跟 instance object 相关联，所以,静态方法(static method)永远不可以 Access 跟 object 相关的动态成员(instance member),反过来就可以，一个 CLASS 的 instance object 可以 Access 这个 Class 的任何成员，包括静态成员(static member).

24、当一个对象被当作参数传递到一个方法后，此方法可改变这个对象的属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递?

考察点：对象

参考回答：

是值传递。Java 编程语言只有值传递参数。当一个对象实例作为一个参数被传递到方法中时，参数的值就是对该对象的引用。对象
的内容可以在被调用的方法中改变，但对象的引用是永远不会改变的。

25、Java 的接口和 C++的虚类的相同和不同处。

考察点：接口

参考回答：

由于 Java 不支持多继承，而有可能某个类或对象要使用分别在几个类或对象里面的方法或属性，现有的单继承机制就不能满足要求。
与继承相比，接口有更高的灵活性，因为接口中没有任何实现代码。当一个类实现了接口以后，该类要实现接口里面所有的方法和属

性，并且接口里面的属性在默认状态下面都是 public static,所有方法默认情况下是 public. 一个类可以实现多个接口。

26、JAVA 语言如何进行异常处理，关键字：throws,throw,try,catch,finally 分别代表什么意义？在 try 块中可以抛出异常吗？

考察点：异常

参考回答：

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 类或其它子类的实例。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（throws）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。用 try 来指定一块预防所有”异常”的程序。紧跟在 try 程序后面，应包含一个 catch 子句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“的框架就放到堆栈上面，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，堆栈就会展开，直到遇到有处理这种”异常”的 try 语句。

27、内部类可以引用他包含类的成员吗？有没有什么限制？

考察点：类

参考回答：

一个内部类对象可以访问创建它的外部类对象的内容，内部类如果不是 static 的，那么它可以访问创建它的外部类对象的所有属性内部类如果是 sattic 的，即为 nested class，那么它只可以访问创建它的外部类对象的所有 static 属性一般普通类只有 public 或 package 的访问修饰，而内部类可以实现 static，protected，private 等访问修饰。当从外部类继承的时候，内部类是不会被覆盖的，它们是完全独立的实体，每个都在自己的命名空间内，如果从内部类中明确地继承，就可以覆盖原来内部类的方法。

28、两个对象值相同(x.equals(y) == true)，但却可有不同的 hash code 说法是否正确？

考察点：对象

参考回答：

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）。

29、重载（Overload）和重写（Override）的区别。重载的方法能否根据返回类型进行区分？

考察点：java 重载

参考回答：

方法的重载和重写都是实现多态的方式，区别在于前者实现的是编译时的多态性，而后者实现的是运行时的多态性。重载发生在一个类中，同名的方法如果有不同的参数列表（参数类型不同、参数个数不同或者二者都不同）则视为重载；重写发生在子类与父类之间，重写要求子类被重写方法与父类被重写方法有相同的返回类型，比父类被重写方法更好访问，不能比父类被重写方法声明更多的异常（里氏代换原则）。重载对返回类型没有特殊的要求。

30、如何通过反射获取和设置对象私有字段的值？

考察点：类

参考回答：

可以通过类对象的 getDeclaredField()方法字段（Field）对象，然后再通过字段对象的 setAccessible(true)将其设置为可以访问，接下来就可以通过 get/set 方法来获取/设置字段的值了。下面的代码实现了一个反射的工具类，其中的两个静态方法分别用于获取和设置私有字段的值，字段可以是基本类型也可以是对象类型且支持多级对象操作，例如 ReflectionUtil.get(dog, "owner.car.engine.id");可以获得 dog 对象的主人的汽车的引擎的 ID 号。

```
    import java.lang.reflect.Method;
class MethodInvokeTest {

        public static void main(String[] args) throws Exception {
                String str = "hello";
         Method m = str.getClass().getMethod("toUpperCase");
                System.out.println(m.invoke(str));   // HELLO
        }
}
```

31、谈一下面向对象的"六原则一法则"。

考察点：Java 对象

参考回答：

- 单一职责原则：一个类只做它该做的事情。（单一职责原则想表达的就是"高内聚"，写代码最终极的原则只有六个字"高内聚、低耦合"，所谓的高内聚就是一个代码模块只完成一项功能，在面向对象中，如果只让一个类完成它该做的事，而不涉及与它无关的领域就是践行了高内聚的原则，这个类就只有单一职责。另一个是模块化，好的自行车是组装车，从减震叉、刹车到变速器，所有的部件都是可以拆卸和重新组装的，好的乒乓球拍也不是成品拍，一定是底板和胶皮可以拆分和自行组装的，一个好的软件系统，它里面的每个功能模块也应该是可以轻易的拿到其他系统中使用的，这样才能实现软件复用的目标。）- 开闭原则：软件实体应当对扩展开放，对修改关闭。（在理想的状态下，当我们需要为一个软件系统增加新功能时，只需要从原来的系统派生出一些新类就可以，不需要修改原来的任何一行代码。要做到开闭有两个要点：①抽象是关键，一个系统中如果没有抽象类或接口系统就没有扩展点；②封装可变性，将系统中的各种可变因素封装到一个继承结构中，如果多个可变因素混杂在一起，系统将变得复杂而换乱，如果不清楚如何封装可变性，可以参考《设计模式精解》一书中对桥梁模式的讲解的章节。）- 依赖倒转原则：面向接口编程。（该原则说得直白和具体一些就是声明方法的参数类型、方法的返回类型、变量的引用类型时，尽可能使用抽象类型而不用具体类型，因为抽象类型可以被它的任何一个子类型所替代，请参考下面的里氏替换原则。）里氏替换原则：任何时候都可以用子类型替换掉父类型。（关于里氏替换原则的描述，Barbara Liskov 女士的描述比这个要复杂得多，但简单的说就是能用父类型的地方就一定能使用子类型。里氏替换原则可以检查继承关系是否合理，如果一个继承关系违背了里氏替换原则，那么这个继承关系一定是错误的，需要对代码进行重构。例如让猫继承狗，或者狗继承猫，又或者让正方形继承长方形都是错误的继承关系，因为你很容易找到违反里氏替换原则的场景。需要注意的是：子类一定是增加父类的能力而不是减少父类的能力，因为子类比父类的能力更多，把能力多的对象当成能力少的对象来用当然没有任何问题。）- 接口隔离原则：接口要小而专，绝不能大而全。（臃肿的接口是对接口的污染，既然接口表示能力，那么一个接口只应该描述一种能力，接口也应该是高度内聚的。例如，琴棋书画就应该分别设计为四个接口，而不应设计成一个接口中的四个方法，因为如果设计成一个接口中的四个方法，那么这个接口很难用，毕竟琴棋书画四样都精通的人还是少数，而如果设计成四个接口，会几项就实现几个接口，这样的话每个接口被复用的可能性是很高的。Java 中的接口代表能力、代表约定、代表角色，能否正确的使用接口一定是编程水平高低的重要标识。）- 合成聚合复用原则：优先使用聚合或合成关系复用代码。（通过继承来复用代码是面向对象程序设计中被滥用得最多的东西，因为所有的教科书都无一例外的对继承进行了鼓吹从而误导了初学者，类与类之间简单的说有三种关系，Is-A 关系、Has-A 关系、Use-A 关系，分别代表继承、关联和依赖。其中，关联关系根据其关联的强度又可以进一步划分为关联、聚合和合成，但说白了都是 Has-A 关系，合成聚合复用原则想表达的是优先考虑 Has-A 关系而不是 Is-A 关系复用代码，原因嘛可以自己从百度上找到一万个理由，需要说明的是，即使在 Java 的 API 中也有不少滥用继承的例子，例如 Properties 类继承了 Hashtable 类，Stack 类继承了 Vector 类，这些继承明显就是错误的，更好的做法是在 Properties 类中放置一个 Hashtable 类型的成员并且将其键和值都设置为字符串来存储数据，而 Stack 类的设计也应该是在 Stack 类中放一个 Vector 对象来存储数据。记住：任何时候都不要继承工具类，工具是可以拥有并可以使用的，而不是拿来继承的。）- 迪米特法则：迪米特法则又叫最少知识原则，一个对象应当对其他对象有尽可能少的了解。再复杂的系统都可以为用户提供一个简单的门面，Java Web 开发中作为前端控制器的 Servlet 或 Filter 不就是一个门面吗，浏览器对服务器的运作方式一无所知，但是通过前端控制器就能够根据你的请求得到相应的服务。调停者模式也可以举一个简单的例子来说明，例如一台计算机，CPU、内存、硬盘、显卡、声卡各种设备需要相互配合才能很好的工作，但是如果这些东西都直接连接到一起，计算机的布线将异常复杂，在这种情况下，主板作为一个调停者的身份出现，它将各个设备连接在一起而不需要每个设备之间直接交换数据，这样就减小了系统的耦合度和复杂度。

32、请问 Query 接口的 list 方法和 iterate 方法有什么区别？

考察点：接口

参考回答：

①list()方法无法利用一级缓存和二级缓存（对缓存只写不读），它只能在开启查询缓存的前提下使用查询缓存；iterate()方法可以充分利用缓存，如果目标数据只读或者读取频繁，使用 iterate()方法可以减少性能开销。
② list()方法不会引起 N+1 查询问题，而 iterate()方法可能引起 N+1 查询问题

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

考察点：方法

参考回答：

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。

34、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？

考察点：JAVA 构造函数

参考回答：

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。

35、hashCode()和 equals()方法有什么联系？

考点：基础

参考回答：

Java 对象的 eqauls 方法和 hashCode 方法是这样规定的：

① 相等（相同）的对象必须具有相等的哈希码（或者散列码）。

② 如果两个对象的 hashCode 相同，它们并不一定相同。

④集合部分

1、Map 和 ConcurrentHashMap 的区别？

考点：集合

参考回答：

hashmap 是线程不安全的，put 时在多线程情况下，会形成环从而导致死循环。CoucurrentHashMap 是线程安全的，采用分段锁机制，减少锁的粒度。

2、hashMap 内部具体如何实现的？

考点：集合

参考回答：

Hashmap 基于数组实现的，通过对 key 的 hashcode & 数组的长度得到在数组中位置，如当前数组有元素，则数组当前元素 next 指向要插入的元素，这样来解决 hash 冲突的，形成了拉链式的结构。put 时在多线程情况下，会形成环从而导致死循环。数组长度一般是 2n，从 0 开始编号，所以 hashcode & （2n-1），（2n-1）每一位都是 1，这样会让散列均匀。需要注意的是，HashMap 在 JDK1.8 的版本中引入了红黑树结构做优化，当链表元素个数大于等于 8 时，链表转换成树结构；若桶中链表元素个数小于等于 6 时，树结构还原成链表。因为红黑树的平均查找长度是 log(n)，长度为 8 的时候，平均查找长度为 3，如果继续使用链表，平均查找长度为 8/2=4，这才有转换为树的必要。链表长度如果是小于等于 6，6/2=3，虽然速度也很快的，但是转化为树结构和生成树的时间并不会太短。还有选择 6 和 8，中间有个差值 7 可以有效防止链表和树频繁转换。假设一下，如果设计成链表个数超过 8 则链表转换成树结构，链表个数小于 8 则树结构转换成链表，如果一个 HashMap 不停的插入、删除元素，链表个数在 8 左右徘徊，就会频繁的发生树转链表、链表转树，效率会很低。

3、如果 hashMap 的 key 是一个自定义的类，怎么办？

考点：集合

参考回答：

使用 HashMap，如果 key 是自定义的类，就必须重写 hashcode()和 equals()。

4、ArrayList 和 LinkedList 的区别，如果一直在 list 的尾部添加元素，用哪个效率高？

考点：集合

参考回答：

ArrayList 采用数组数组实现的，查找效率比 LinkedList 高。LinkedList 采用双向链表实现的，插入和删除的效率比 ArrayList 要高。一直在 list 的尾部添加元素，LinkedList 效率要高。

5、HashMap 底层，负载因子，为啥是 2^n？

考点：集合

参考回答：

负载因子默认是 0.75， 2^n 是为了让散列更加均匀，例如出现极端情况都散列在数组中的一个下标，那么 hashmap 会由 0（1）复杂退化为 0（n）的。

6、ConcurrentHashMap 锁加在了哪些地方？

考点：集合

参考回答：

加在每个 Segment 上面。

7、TreeMap 底层，红黑树原理？

考点：集合

参考回答：

TreeMap 的实现就是红黑树数据结构，也就说是一棵自平衡的排序二叉树，这样就可以保证当需要快速检索指定节点。

红黑树的插入、删除、遍历时间复杂度都为 O(lgN)，所以性能上低于哈希表。但是哈希表无法提供键值对的有序输出，红黑树因为是排序插入的，可以按照键的值的大小有序输出。红黑树性质：

性质 1：每个节点要么是红色，要么是黑色。

性质 2：根节点永远是黑色的。

性质 3：所有的叶节点都是空节点（即 null），并且是黑色的。

性质 4：每个红色节点的两个子节点都是黑色。（从每个叶子到根的路径上不会有两个连续的红色节点）

性质 5：从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点。

8、concurrenthashmap 有啥优势，1.7，1.8 区别？

考点：集合

参考回答：

Concurrenthashmap 线程安全的，1.7 是在 jdk1.7 中采用 Segment + HashEntry 的方式进行实现的，lock 加在 Segment 上面。1.7size 计算是先采用不加锁的方式，连续计算元素的个数，最多计算 3 次：1、如果前后两次计算结果相同，则说明计算出来的元素个数是准确的；2、如果前后两次计算结果都不同，则给每个 Segment 进行加锁，再计算一次元素的个数；

1.8 中放弃了 Segment 臃肿的设计，取而代之的是采用 Node + CAS + Synchronized 来保证并发安全进行实现，1.8 中使用一个 volatile 类型的变量 baseCount 记录元素的个数，当插入新数据或则删除数据时，会通过 addCount()方法更新 baseCount，通过累加 baseCount 和 CounterCell 数组中的数量，即可得到元素的总个数；

9、ArrayList 是否会越界？

考点：集合

参考回答：

ArrayList 是实现了基于动态数组的数据结构，而 LinkedList 是基于链表的数据结构 2. 对于随机访问 get 和 set，ArrayList 要优于 LinkedList，因为 LinkedList 要移动指针；ArrayList 并发 add()可能出现数组下标越界异常

10、什么是 TreeMap?

考察点：key-value 集合

参考回答：

TreeMap 是一个有序的 key-value 集合，基于红黑树（Red-Black tree）的 NavigableMap 实现。该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的 Comparator 进行排序，具体取决于使用的构造方法。

TreeMap 的特性：

根节点是黑色

每个节点都只能是红色或者黑色

每个叶节点（NIL 节点，空节点）是黑色的。

如果一个节点是红色的，则它两个子节点都是黑色的，也就是说在一条路径上不能出现两个红色的节点。

从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

11、ConcurrentHashMap 的原理是什么？

考察点：JAVA 内存模型

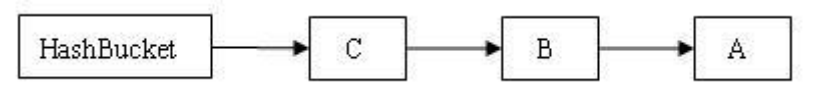
参考回答：

ConcurrentHashMap 类中包含两个静态内部类 HashEntry 和 Segment。HashEntry 用来封装映射表的键 / 值对；Segment 用来充当锁的角色，每个 Segment 对象守护整个散列映射表的若干个桶。每个桶是由若干个 HashEntry 对象链接起来的链表。一个 ConcurrentHashMap 实例中包含由若干个 Segment 对象组成的数组。HashEntry 用来封装散列映射表中的键值对。在 HashEntry 类中，key,hash 和 next 域都被声明为 final 型，value 域被声明为 volatile 型。

```
static final class HashEntry<K,V> {

                final K key;                                        // 声明
key 为 final 型

                final int hash;                                     // 声明 hash
值为 final 型

                volatile V value;                                   // 声明 value 为
volatile 型

                final HashEntry<K,V> next;          // 声明 next 为 final 型


                HashEntry(K key, int hash, HashEntry<K,V> next, V value) {

                        this.key = key;

                        this.hash = hash;

                        this.next = next;

                        this.value = value;

                }

}
```

在 ConcurrentHashMap 中，在散列时如果产生“碰撞”，将采用“分离链接法”来处理“碰撞”：把“碰撞”的 HashEntry 对象链接成一个链表。由于 HashEntry 的 next 域为 final 型，所以新节点只能在链表的表头处插入。 下图是在一个空桶中依次插入 A，B，C 三个 HashEntry 对象后的结构图：

图 1. 插入三个节点后桶的结构示意图：



注意：由于只能在表头插入，所以链表中节点的顺序和插入的顺序相反。

Segment 类继承于 ReentrantLock 类，从而使得 Segment 对象能充当锁的角色。每个 Segment 对象用来守护其（成员对象 table 中）包含的若干个桶。

12、Java 集合类框架的基本接口有哪些？

考察点：JAVA 集合

参考回答：

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。
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。

13、为什么集合类没有实现 Cloneable 和 Serializable 接口？

考察点：JAVA 集合

参考回答：

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。

实现 Serializable 序列化的作用：将对象的状态保存在存储媒体中以便可以在以后重写创建出完全相同的副本；按值将对象从一个从一个应用程序域发向另一个应用程序域。

实现 Serializable 接口的作用就是可以把对象存到字节流，然后可以恢复。所以你想如果你的对象没有序列化，怎么才能进行网络传输呢？要网络传输就得转为字节流，所以在分布式应用中，你就得实现序列化。如果你不需要分布式应用，那就没必要实现实现序列化

14、什么是迭代器？

考察点：JAVA 迭代器

参考回答：

Iterator 提供了统一遍历操作集合元素的统一接口，Collection 接口实现 Iterable 接口，

每个集合都通过实现 Iterable 接口中 iterator()方法返回 Iterator 接口的实例，然后对集合的元素进行迭代操作.
有一点需要注意的是：在迭代元素的时候不能通过集合的方法删除元素，否则会抛出 ConcurrentModificationException
异常. 但是可以通过 Iterator 接口中的 remove()方法进行删除.

15、Iterator 和 ListIterator 的区别是什么？

考察点：迭代器

参考回答：

Iterator 和 ListIterator 的区别是：
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。

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

考察点：集合

参考回答：

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。

17、HashMap 和 Hashtable 有什么区别？

考察点：集合

参考回答：

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。

18、ArrayList 和 LinkedList 有什么区别？

考察点：ArrayList

参考回答：

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。

19、ArrayList,Vector,LinkedList 的存储性能和特性是什么？

考察点：ArrayList

参考回答：

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，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。

20、Collection 和 Collections 的区别。

考察点：集合

参考回答：

Collection 是集合类的上级接口，继承与他的接口主要有 Set 和 List.
Collections 是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

21、你所知道的集合类都有哪些？主要方法？

考察点：集合

参考回答：

最常用的集合类是 List 和 Map。List 的具体实现包括 ArrayList 和 Vector，它们是可变大小的列表，比较适合构建、存储和操
作任何类型对象的元素列表。 List 适用于按数值索引访问元素的情形。
Map 提供了一个更通用的元素存储方法。 Map 集合类用于存储元素对（称作"键"和"值"），其中每个键映射到一个值。

22、List、Set、Map 是否继承自 Collection 接口？

考察点：collection 接口

参考回答：

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。

23、阐述 ArrayList、Vector、LinkedList 的存储性能和特性

考察点：ArrayList

参考回答：

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 中的方法由于添加了 synchronized 修饰，因此 Vector 是线程安全的容器，但性能上较 ArrayList 差，因此已经是 Java 中的遗留容器。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，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（Java 早期的版本中提供的容器，除此之外，Hashtable、Dictionary、BitSet、Stack、Properties 都是遗留容器），已经不推荐使用，但是由于 ArrayList 和 LinkedListed 都是非线程安全的，如果遇到多个线程操作同一个容器的场景，则可以通过工具类 Collections 中的 synchronizedList 方法将其转换成线程安全的容器后再使用（这是对装潢模式的应用，将已有对象传入另一个类的构造器中创建新的对象来增强实现）。

24、List、Map、Set 三个接口存取元素时，各有什么特点？

考察点：List

参考回答：

List 以特定索引来存取元素，可以有重复元素。Set 不能存放重复元素（用对象的 equals()方法来区分元素是否重复）。Map 保存键值对（key-value pair）映射，映射关系可以是一对一或多对一。Set 和 Map 容器都有基于哈希存储和排序树的两种实现版本，基于哈希存储的版本理论存取时间复杂度为 O(1)，而基于排序树版本的实现在插入或删除元素时会按照元素或元素的键（key）构成排序树从而达到排序和去重的效果。

2、Java 高级知识

①线程

1、多线程中的 i++线程安全吗？为什么？

考察点：多线程

参考回答：

不安全。i++不是原子性操作。i++分为读取 i 值，对 i 值加一，再赋值给 i++，执行期中任何一步都是有可能被其他线程抢占的。

2、如何线程安全的实现一个计数器？

考察点：多线程

参考回答：

可以使用加锁，比如 synchronized 或者 lock。也可以使用 Concurrent 包下的原子类。

3、多线程同步的方法

考察点：多线程

参考回答：

可以使用 synchronized、lock、volatile 和 ThreadLocal 来实现同步。

4、介绍一下生产者消费者模式？

考察点：线程

参考回答：

生产者消费者问题是线程模型中的经典问题:生产者和消费者在同一时间段内共用同一存储空间，生产者向空间里生产数据，而消费者取走数据。

优点：支持并发、解耦。

5、线程，进程，然后线程创建有很大开销，怎么优化？

考察点：多线程

参考回答：

可以使用线程池。

6、线程池运行流程，参数，策略

考察点：线程池

参考回答：

线程池主要就是指定线程池核心线程数大小，最大线程数，存储的队列，拒绝策略，空闲线程存活时长。当需要任务大于核心线程数时候，就开始把任务往存储任务的队列里，当存储队列满了的话，就开始增加线程池创建的线程数量，如果当线程数量也达到了最大，就开始执行拒绝策略，比如说记录日志，直接丢弃，或者丢弃最老的任务。

7、讲一下 AQS 吧。

考察点：多线程

参考回答：

AQS 其实就是一个可以给我们实现锁的框架

内部实现的关键是：先进先出的队列、state 状态

定义了内部类 ConditionObject

拥有两种线程模式独占模式和共享模式。

在 LOCK 包中的相关锁(常用的有 ReentrantLock、 ReadWriteLock)都是基于 AQS 来构建，一般我们叫 AQS 为同步器。

8、创建线程的方法，哪个更好，为什么？

考察点：线程

参考回答：

需要从 Java.lang.Thread 类派生一个新的线程类，重载它的 run()方法；

实现 Runnalbe 接口，重载 Runnalbe 接口中的 run()方法。

实现 Runnalbe 接口更好，使用实现 Runnable 接口的方式创建的线程可以处理同一资源，从而实现资源的共享.

9、Java 中有几种方式启动一个线程？

考察点：线程

参考回答：

1. 继承自 Thread 类

2. 实现 Runnable 接口

3.即实现 Runnable 接口，也继承 Thread 类，并重写 run 方法

10、Java 中有几种线程池？

考察点：线程池

参考回答：

1、newFixedThreadPool 创建一个指定工作线程数量的线程池。每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。

2、newCachedThreadPool 创建一个可缓存的线程池。这种类型的线程池特点是：

1).工作线程的创建数量几乎没有限制(其实也有限制的,数目为 Interger. MAX_VALUE),这样可灵活的往线程池中添加线程。

2).如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为 1 分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

3、newSingleThreadExecutor 创建一个单线程化的 Executor，即只创建唯一的工作者线程来执行任务，如果这个线程异常结束，会有另一个取代它，保证顺序执行(我觉得这点是它的特色)。单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的 。

4、newScheduleThreadPool 创建一个定长的线程池，而且支持定时的以及周期性的任务执行，类似于 Timer。(这种线程池原理暂还没完全了解透彻)

11、线程池有什么好处？

考察点：线程池

参考回答：

第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能执行。

第三：提高线程的可管理性，线程是稀缺资源，如果无限制地创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。

12、cyclicbarrier 和 countdownlatch 的区别

考察点：线程

参考回答：

CountDownLatch 和 CyclicBarrier 都能够实现线程之间的等待，只不过它们侧重点不同：

CountDownLatch 一般用于某个线程 A 等待若干个其他线程执行完任务之后，它才执行；

而 CyclicBarrier 一般用于一组线程互相等待至某个状态，然后这一组线程再同时执行；

另外，CountDownLatch 是不能够重用的，而 CyclicBarrier 是可以重用的。

13、启动线程有哪几种方式，线程池有哪几种？

考察点：线程池

参考回答：

①启动线程有如下三种方式：

一、继承 Thread 类创建线程类

（1）定义 Thread 类的子类，并重写该类的 run 方法，该 run 方法的方法体就代表了线程要完成的任务。因此把 run()方法称为执行体。

（2）创建 Thread 子类的实例，即创建了线程对象。

（3）调用线程对象的 start()方法来启动该线程。

代码：

```
package com.thread;

public class FirstThreadTest extends Thread{

int i = 0;

//重写 run 方法，run 方法的方法体就是现场执行体

public void run()

{

    for(;i<100;i++){

    System.out.println(getName()+"  "+i);


    }

}

public static void main(String[] args)

{
```

```
        for(int i = 0;i< 100;i++)
        {
            System.out.println(Thread.currentThread().getName()+"  : "+i);
            if(i==20)
            {
                new FirstThreadTest().start();
                new FirstThreadTest().start();
            }
        }
    }


    }
```

上述代码中 Thread.currentThread()方法返回当前正在执行的线程对象。GetName()方法返回调用该方法的线程的名字。

二、通过 Runnable 接口创建线程类

（1）定义 runnable 接口的实现类，并重写该接口的 run()方法，该 run()方法的方法体同样是该线程的线程执行体。

（2）创建 Runnable 实现类的实例，并依此实例作为 Thread 的 target 来创建 Thread 对象，该 Thread 对象才是真正的线程对象。

（3）调用线程对象的 start()方法来启动该线程。

代码：

```
package com.thread;


public class RunnableThreadTest implements Runnable
{


private int i;
public void run()
{
```

```
        for(i = 0;i <100;i++)
        {
            System.out.println(Thread.currentThread().getName()+" "+i);
        }
    }
    public static void main(String[] args)
    {
        for(int i = 0;i < 100;i++)
        {
            System.out.println(Thread.currentThread().getName()+" "+i);
            if(i==20)
            {
                RunnableThreadTest rtt = new RunnableThreadTest();
                new Thread(rtt,"新线程 1").start();
                new Thread(rtt,"新线程 2").start();
            }
        }

    }

    }
```

三、通过 Callable 和 Future 创建线程

（1）创建 Callable 接口的实现类，并实现 call()方法，该 call()方法将作为线程执行体，并且有返回值。

（2)创建 Callable 实现类的实例，使用 FutureTask 类来包装 Callable 对象，该 FutureTask 对象封装了该 Callable 对象的 call()方法的返回值。

（3）使用 FutureTask 对象作为 Thread 对象的 target 创建并启动新线程。

（4）调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值

代码：

```
package com.thread;

import java.util.concurrent.Callable;

import java.util.concurrent.ExecutionException;

import java.util.concurrent.FutureTask;


public class CallableThreadTest implements Callable<Integer>

{


public static void main(String[] args)

{

    CallableThreadTest ctt = new CallableThreadTest();

    FutureTask<Integer> ft = new FutureTask<>(ctt);

    for(int i = 0;i < 100;i++)

    {

        System.out.println(Thread.currentThread().getName()+" 的循环变量 i 的值
"+i);

        if(i==20)

        {

            new Thread(ft,"有返回值的线程").start();

        }

    }

    try

    {

        System.out.println("子线程的返回值："+ft.get());

    } catch (InterruptedException e)

    {

        e.printStackTrace();
```

```
        } catch (ExecutionException e)
        {
            e.printStackTrace();
        }

    }

    @Override
    public Integer call() throws Exception
    {
        int i = 0;
        for(;i<100;i++)
        {
            System.out.println(Thread.currentThread().getName()+" "+i);
        }
        return i;
    }

    }
```

②线程池的种类有：

Java 通过 Executors 提供四种线程池，分别为：
newCachedThreadPool 创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。
newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

14、如何理解 Java 多线程回调方法？

考察点：JAVA 多线程

参考回答：

所谓回调，就是客户程序 C 调用服务程序 S 中的某个方法 A，然后 S 又在某个时候反过来调用 C 中的某个方法 B，对于 C 来说，这个 B 便叫做回调方法。

15、创建线程有几种不同的方式？你喜欢哪一种？为什么？

考察点：JAVA 线程

参考回答：

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。

16、概括的解释下线程的几种可用状态。

考察点：JAVA 线程状态

参考回答：

1. 新建(new)：新创建了一个线程对象。
2. 可运行(runnable)：线程对象创建后，其他线程(比如 main 线程）调用了该对象 的 start ()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获 取 cpu 的使用权 。
3. 运行(running)：可运行状态(runnable)的线程获得了 cpu 时间片 (timeslice) ，执行程序代码。
4. 阻塞(block)：阻塞状态是指线程因为某种原因放弃了 cpu 使用权，也即让出了 cpu timeslice ，暂时停止运行。直到线程进入可运行(runnable)状态，才有 机会再次获得 cpu timeslice 转到运行(running)状态。阻塞的情况分三种：
(一). 等待阻塞：运行(running)的线程执行 o . wait ()方法， JVM 会把该线程放 入等待队列(waitting queue)中。
(二). 同步阻塞：运行(running)的线程在获取对象的同步锁时，若该同步锁 被别的线程占用，则 JVM 会把该线程放入锁池(lock pool)中。
(三). 其他阻塞：运行(running)的线程执行 Thread . sleep (long ms)或 t . join () 方法，或者发出了 I/O 请求时， JVM 会把该线程置为阻塞状态。 当 sleep ()状态超时、 join ()等待线程终止或者超时、或者 I/O 处理完毕时，线程重新转入可运行(runnable)状态。
5. 死亡(dead)：线程 run ()、 main () 方法执行结束，或者因异常退出了 run ()方法，则该线程结束生命周期。死亡的线程不可再次复生。

17、同步方法和同步代码块的区别是什么？

考察点：JAVA 代码块同步

参考回答：

区别：
同步方法默认用 this 或者当前类 class 对象作为锁；

同步代码块可以选择以什么来加锁，比同步方法要更细颗粒度，我们可以选择只同步会发生同步问题的部分代码而不是整个方法；

18、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？

考察点：JAVA 线程同步

参考回答：

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。

19、sleep() 和 wait() 有什么区别？

考察点：线程

参考回答：

sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，把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。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，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运行状态。

20、同步和异步有何异同，在什么情况下分别使用他们？举例说明。

考察点：线程同步

参考回答：

如果数据将在线程间共享。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，那么这些数据就是共享数据，必须进行同步存取。
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。

21、设计 4 个线程，其中两个线程每次对 j 增加 1，另外两个线程对 j 每次减少 1。使用内部类实现线程，对 j 增减的时候没有考虑顺序问题。

考察点：JAVA 线程

参考回答：

```
public class ThreadTest1{
private int j;
public static void main(String args[]){
ThreadTest1 tt=new ThreadTest1();
Inc inc=tt.new Inc();
```

```
Dec dec=tt.new Dec();
for(int i=0;i<2;i++){
Thread t=new Thread(inc);
t.start();
t=new Thread(dec);
t.start();
}
}
private synchronized void inc(){
j++;
System.out.println(Thread.currentThread().getName()+"-inc:"+j);
}
private synchronized void dec(){
j--;
System.out.println(Thread.currentThread().getName()+"-dec:"+j);
}
class Inc implements Runnable{
public void run(){
for(int i=0;i<100;i++){
inc();
}
}
}
class Dec implements Runnable{
public void run(){
for(int i=0;i<100;i++){
dec();
}
}
}
}
```

22、启动一个线程是用 run()还是 start()?

考察点：JAVA 线程

参考回答：

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()方法可以产生必须退出的标志来停止一个线程。

23、请说出你所知道的线程同步的方法

考察点：线程同步

参考回答：

wait():使一个线程处于等待状态，并且释放所持有的对象的 lock。
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，注意的是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，而且不是按优先级。
Allnotity():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。

24、多线程有几种实现方法,都是什么?同步有几种实现方法,都是什么?

考察点：线程

参考回答：

多线程有两种实现方法,分别是继承 Thread 类与实现 Runnable 接口同步的实现方面有两种，分别是 synchronized,wait 与 notify。

25、java 中有几种方法可以实现一个线程？用什么关键字修饰同步方法? stop()和 suspend()方法为何不推荐使用？

考察点：线程

参考回答：

有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口用 synchronized 关键字修饰同步方法，反对使用 stop()，是因为它不安全。它会解除由线程获取的所有锁定，而且如果对象处于一种不连贯状态，那么其他线程能在那种状态下检查和修改它们。结果很难检查出真正的问题所在。suspend()方法容易发生死锁。调用 suspend()的时候，目标线程会停下来，但却仍然持有在这之前获得的锁定。此时，其他任何线程都不能访问锁定的资源，除非被”挂起”的线程恢复运行。对任何线程来说，如果它们想恢复目标线程，同时又试图使用任何一个锁定的资源，就会造成死锁。所以不应该使用 suspend()，而应在自己的 Thread 类中置入一个标志，指出线程应该活动还是挂起。若标志指出线程应该挂起，便用 wait()命其进入等待状态。若标志指出线程应当恢复，则用一个 notify()重新启动线程。

26、线程的 sleep()方法和 yield()方法有什么区别？

考察点：线程

参考回答：

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 CPU 调度相关）具有更好的可移植性。

27、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 B?

考察点：线程

参考回答：

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。因为非静态方法上的 synchronized 修饰符要求执行方法时要获得对象的锁，如果已经进入 A 方法说明对象锁已经被取走，那么试图进入 B 方法的线程就只能在等锁池（注意不是等待池哦）中等待对象的锁。

28、请说出与线程同步以及线程调度相关的方法。

考察点：线程同步

参考回答：

- 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，调用此方法要处理 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()：唤醒所有处于等待状态的线程，该方法并不是将对象的锁给所有线程，而是让它们竞争，只有获得锁的线程才能进入就绪状态；

通过 Lock 接口提供了显式的锁机制（explicit lock），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；此外，Java 5 还提供了信号量机制（semaphore），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）。

29、举例说明同步和异步

考察点：线程

参考回答：

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（数据库操作中的排他锁就是最好的例子）。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。

30、什么是线程池（thread pool）？

考察点：线程池

参考回答：

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。
Java 5+中的 Executor 接口定义一个执行线程的工具。它的子类型即线程池接口是

ExecutorService。要配置一个线程池是比较复杂的，尤其是对于线程池的原理不是很清楚的情况下，因此在工具类 Executors 面提供了一些静态工厂方法，生成一些常用的线程池，如下所示：
- newSingleThreadExecutor：创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。
- newFixedThreadPool：创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。
- newCachedThreadPool：创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60 秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说 JVM）能够创建的最大线程大小。
- newScheduledThreadPool：创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。
- newSingleThreadExecutor：创建一个单线程的线程池。此线程池支持定时以及周期性执行任务的需求。

31、说说线程的基本状态以及状态之间的关系？

考察点：线程

参考回答：

其中 Running 表示运行状态，Runnable 表示就绪状态（万事俱备，只欠 CPU），Blocked 表示阻塞状态，阻塞状态又有多种情况，可能是因为调用 wait()方法进入等待池，也可能是执行同步方法或同步代码块进入等锁池，或者是调用了 sleep()方法或 join()方法等待休眠或其他线程结束，或是因为发生了 I/O 中断。

32、如何保证线程安全？

考察点：线程

参考回答：

通过合理的时间调度，避开共享资源的存取冲突。另外，在并行任务设计上可以通过适当的策略，保证任务与任务之间不存在共享资源，设计一个规则来保证一个客户的计算工作和数据访问只会被一个线程或一台工作机完成，而不是把一个客户的计算工作分配给多个线程去完成。

②锁

1、讲一下非公平锁和公平锁在 reetrantlock 里的实现。

考察点：锁

参考回答：

如果一个锁是公平的，那么锁的获取顺序就应该符合请求的绝对时间顺序，FIFO。对于非公平锁，只要 CAS 设置同步状态成功，则表示当前线程获取了锁，而公平锁还需要判断当前节点是否有前驱节点，如果有，则表示有线程比当前线程更早请求获取锁，因此需要等待前驱线程获取并释放锁之后才能继续获取锁。

2、讲一下 synchronized，可重入怎么实现。

考察点：锁

参考回答：

每个锁关联一个线程持有者和一个计数器。当计数器为 0 时表示该锁没有被任何线程持有，那么任何线程都都可能获得该锁而调用相应方法。当一个线程请求成功后，JVM 会记下持有锁的线程，并将计数器计为 1。此时其他线程请求该锁，则必须等待。而该持有锁的线程如果再次请求这个锁，就可以再次拿到这个锁，同时计数器会递增。当线程退出一个 synchronized 方法/块时，计数器会递减，如果计数器为 0 则释放该锁。

3、锁和同步的区别。

考察点：锁

参考回答：

用法上的不同：

synchronized 既可以加在方法上，也可以加载特定代码块上，而 lock 需要显示地指定起始位置和终止位置。

synchronized 是托管给 JVM 执行的，lock 的锁定是通过代码实现的，它有比 synchronized 更精确的线程语义。

性能上的不同：

lock 接口的实现类 ReentrantLock，不仅具有和 synchronized 相同的并发性和内存语义，还多了超时的获取锁、定时锁、等候和中断锁等。

在竞争不是很激烈的情况下，synchronized 的性能优于 ReentrantLock，竞争激烈的情况下 synchronized 的性能会下降的非常快，而 ReentrantLock 则基本不变。

锁机制不同：

synchronized 获取锁和释放锁的方式都是在块结构中，当获取多个锁时，必须以相反的顺序释放，并且是自动解锁。而 Lock 则需要开发人员手动释放，并且必须在 finally 中释放，否则会引起死锁。

4、什么是死锁(deadlock)？

考察点：线程死锁

参考回答：

两个线程或两个以上线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。结果就是这些线程都陷入了无限的等待中。

例如，如果线程 1 锁住了 A，然后尝试对 B 进行加锁，同时线程 2 已经锁住了 B，接着尝试对 A 进行加锁，这时死锁就发生了。线程 1 永远得不到 B，线程 2 也永远得不到 A，并且它们永

远也不会知道发生了这样的事情。为了得到彼此的对象（A 和 B），它们将永远阻塞下去。这种情况就是一个死锁。

5、如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？

考察点：死锁

参考回答：

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。

预防死锁，预先破坏产生死锁的四个条件。互斥不可能破坏，所以有如下三种方法：

破坏请求和保持条件，进程必须等所有要请求的资源都空闲时才能申请资源，这种方法会使资源浪费严重(有些资源可能仅在运行初期或结束时才使用,甚至根本不使用). 允许进程获取初期所需资源后，便开始运行，运行过程中再逐步释放自己占有的资源，比如有一个进程的任务是把数据复制到磁盘中再打印，前期只需获得磁盘资源而不需要获得打印机资源，待复制完毕后再释放掉磁盘资源。这种方法比第一种方法好，会使资源利用率上升。

2.破坏不可抢占条件，这种方法代价大，实现复杂。

3.破坏循坏等待条件，对各进程请求资源的顺序做一个规定，避免相互等待。这种方法对资源的利用率比前两种都高，但是前期要为设备指定序号，新设备加入会有一个问题，其次对用户编程也有限制。

6、请你简述 synchronized 和 java.util.concurrent.locks.Lock 的异同？

考察点：锁机制

参考回答：

主要相同点：Lock 能完成 synchronized 所实现的所有功能
主要不同点：Lock 有比 synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而 Lock 一定要求程序员手工释放
，并且必须在 finally 从句中释放。

③JDK

1、Java 中的 LongAdder 和 AtomicLong 的区别

考点：JDK

参考回答：

JDK1.8 引入了 LongAdder 类。CAS 机制就是，在一个死循环内，不断尝试修改目标值，直到修改成功。如果竞争不激烈，那么修改成功的概率就很高，否则，修改失败的的概率就很高，在大量修改失败时，这些原子操作就会进行多次循环尝试，因此性能就会受到影响。 结合 ConcurrentHashMap 的实现思想,应该可以想到对一种传统 AtomicInteger 等原子类的改进思路。虽然 CAS 操作没有锁，但是像减少粒度这种分离热点的思想依然可以使用。将 AtomicInteger

的内部核心数据 value 分离成一个数组，每个线程访问时，通过哈希等算法映射到其中一个数字进行计数，而最终的计数结果，则为这个数组的求和累加。热点数据 value 被分离成多个单元 cell，每个 cell 独自维护内部的值，当前对象的实际值由所有的 cell 累计合成，这样热点就进行了有效的分离，提高了并行度。

2、JDK 和 JRE 的区别是什么？

考察点：JDK

参考回答：

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。

④反射

1、反射的实现与作用

考察点：反射

参考回答；

JAVA 语言编译之后会生成一个.class 文件，反射就是通过字节码文件找到某一个类、类中的方法以及属性等。反射的实现主要借助以下四个类：Class：类的对象，Constructor：类的构造方法，Field：类中的属性对象，Method：类中的方法对象。

作用：反射机制指的是程序在运行时能够获取自身的信息。在 JAVA 中，只要给定类的名字，那么就可以通过反射机制来获取类的所有信息。

⑤JVM

1、JVM 回收算法和回收器，CMS 采用哪种回收算法，怎么解决内存碎片问题？

考察点：JVM

参考回答：

垃圾回收算法

标记清除

标记-清除算法将垃圾回收分为两个阶段：标记阶段和清除阶段。在标记阶段首先通过根节点，标记所有从根节点开始的对象，未被标记的对象就是未被引用的垃圾对象。然后，在清除阶段，清除所有未被标记的对象。标记清除算法带来的一个问题是会存在大量的空间碎片，因为回收后的空间是不连续的，这样给大对象分配内存的时候可能会提前触发 full gc。

复制算法

将现有的内存空间分为两快，每次只使用其中一块，在垃圾回收时将正在使用的内存中的存活对象复制到未被使用的内存块中，之后，清除正在使用的内存块中的所有对象，交换两个内存的角色，完成垃圾回收。

现在的商业虚拟机都采用这种收集算法来回收新生代，IBM 研究表明新生代中的对象 98%是朝夕生死的，所以并不需要按照 1:1 的比例划分内存空间，而是将内存分为一块较大的 Eden 空间和两块较小的 Survivor 空间，每次使用 Eden 和其中的一块 Survivor。当回收时，将 Eden 和 Survivor 中还存活着的对象一次性地拷贝到另外一个 Survivor 空间上，最后清理掉 Eden 和刚才用过的 Survivor 的空间。HotSpot 虚拟机默认 Eden 和 Survivor 的大小比例是 8:1(可以通过-SurvivorRattio 来配置)，也就是每次新生代中可用内存空间为整个新生代容量的 90%，只有 10%的内存会被“浪费”。当然，98%的对象可回收只是一般场景下的数据，我们没有办法保证回收都只有不多于 10%的对象存活，当 Survivor 空间不够用时，需要依赖其他内存（这里指老年代）进行分配担保。

标记整理

复制算法的高效性是建立在存活对象少、垃圾对象多的前提下的。这种情况在新生代经常发生，但是在老年代更常见的情况是大部分对象都是存活对象。如果依然使用复制算法，由于存活的对象较多，复制的成本也将很高。

标记-压缩算法是一种老年代的回收算法，它在标记-清除算法的基础上做了一些优化。首先也需要从根节点开始对所有可达对象做一次标记，但之后，它并不简单地清理未标记的对象，而是将所有的存活对象压缩到内存的一端。之后，清理边界外所有的空间。这种方法既避免了碎片的产生，又不需要两块相同的内存空间，因此，其性价比比较高。

增量算法

增量算法的基本思想是，如果一次性将所有的垃圾进行处理，需要造成系统长时间的停顿，那么就可以让垃圾收集线程和应用程序线程交替执行。每次，垃圾收集线程只收集一小片区域的内存空间，接着切换到应用程序线程。依次反复，直到垃圾收集完成。使用这种方式，由于在垃圾回收过程中，间断性地还执行了应用程序代码，所以能减少系统的停顿时间。但是，因为线程切换和上下文转换的消耗，会使得垃圾回收的总体成本上升，造成系统吞吐量的下降。

垃圾回收器

Serial 收集器

Serial 收集器是最古老的收集器，它的缺点是当 Serial 收集器想进行垃圾回收的时候，必须暂停用户的所有进程，即 stop the world。到现在为止，它依然是虚拟机运行在 client 模式下的默认新生代收集器，与其他收集器相比，对于限定在单个 CPU 的运行环境来说，Serial 收集器由于没有线程交互的开销，专心做垃圾回收自然可以获得最高的单线程收集效率。

Serial Old 是 Serial 收集器的老年代版本，它同样是一个单线程收集器，使用”标记－整理“算法。这个收集器的主要意义也是被 Client 模式下的虚拟机使用。在 Server 模式下，它主要还有两大用途：一个是在 JDK1.5 及以前的版本中与 Parallel Scanvenge 收集器搭配使用，另外一个就是作为 CMS 收集器的后备预案，在并发收集发生 Concurrent Mode Failure 的时候使用。

通过指定-UseSerialGC 参数，使用 Serial + Serial Old 的串行收集器组合进行内存回收。

ParNew 收集器

ParNew 收集器是 Serial 收集器新生代的多线程实现，注意在进行垃圾回收的时候依然会 stop the world，只是相比较 Serial 收集器而言它会运行多条进程进行垃圾回收。

ParNew 收集器在单 CPU 的环境中绝对不会有比 Serial 收集器更好的效果，甚至由于存在线程交互的开销，该收集器在通过超线程技术实现的两个 CPU 的环境中都不能百分之百的保证能超越 Serial 收集器。当然，随着可以使用的 CPU 的数量增加，它对于 GC 时系统资源的利用还是很有好处的。它默认开启的收集线程数与 CPU 的数量相同，在 CPU 非常多（譬如 32 个，现在 CPU 动辄 4 核加超线程，服务器超过 32 个逻辑 CPU 的情况越来越多了）的环境下，可以使用 -XX:ParallelGCThreads 参数来限制垃圾收集的线程数。

-UseParNewGC: 打开此开关后，使用 ParNew + Serial Old 的收集器组合进行内存回收，这样新生代使用并行收集器，老年代使用串行收集器。

Parallel Scavenge 收集器

Parallel 是采用复制算法的多线程新生代垃圾回收器，似乎和 ParNew 收集器有很多的相似的地方。但是 Parallel Scanvenge 收集器的一个特点是它所关注的目标是吞吐量(Throughput)。所谓吞吐量就是 CPU 用于运行用户代码的时间与 CPU 总消耗时间的比值，即吞吐量=运行用户代码时间 /（运行用户代码时间 + 垃圾收集时间）。停顿时间越短就越适合需要与用户交互的程序，良好的响应速度能够提升用户的体验；而高吞吐量则可以最高效率地利用 CPU 时间，尽快地完成程序的运算任务，主要适合在后台运算而不需要太多交互的任务。

Parallel Old 收集器是 Parallel Scavenge 收集器的老年代版本，采用多线程和”标记－整理”算法。这个收集器是在 jdk1.6 中才开始提供的，在此之前，新生代的 Parallel Scavenge 收集器一直处于比较尴尬的状态。原因是如果新生代 Parallel Scavenge 收集器，那么老年代除了 Serial Old(PS MarkSweep)收集器外别无选择。由于单线程的老年代 Serial Old 收集器在服务端应用性能上的”拖累“，即使使用了 Parallel Scavenge 收集器也未必能在整体应用上获得吞吐量最大化的效果，又因为老年代收集中无法充分利用服务器多 CPU 的处理能力，在老年代很大而且硬件比较高级的环境中，这种组合的吞吐量甚至还不一定有 ParNew 加 CMS 的组合”给力“。直到 Parallel Old 收集器出现后，”吞吐量优先“收集器终于有了比较名副其实的应用祝贺，在注重吞吐量及 CPU 资源敏感的场合，都可以优先考虑 Parallel Scavenge 加 Parallel Old 收集器。

-UseParallelGC: 虚拟机运行在 Server 模式下的默认值，打开此开关后，使用 Parallel Scavenge + Serial Old 的收集器组合进行内存回收。-UseParallelOldGC: 打开此开关后，使用 Parallel Scavenge + Parallel Old 的收集器组合进行垃圾回收

CMS 收集器

CMS(Concurrent Mark Swep)收集器是一个比较重要的回收器，现在应用非常广泛，我们重点来看一下，CMS 一种获取最短回收停顿时间为目标的收集器，这使得它很适合用于和用户交互的业务。从名字(Mark Swep)就可以看出，CMS 收集器是基于标记清除算法实现的。它的收集过程分为四个步骤：

初始标记(initial mark)

并发标记(concurrent mark)

重新标记(remark)

并发清除(concurrent sweep)

注意初始标记和重新标记还是会 stop the world，但是在耗费时间更长的并发标记和并发清除两个阶段都可以和用户进程同时工作。

G1 收集器

G1 收集器是一款面向服务端应用的垃圾收集器。HotSpot 团队赋予它的使命是在未来替换掉 JDK1.5 中发布的 CMS 收集器。与其他 GC 收集器相比，G1 具备如下特点：

并行与并发：G1 能更充分的利用 CPU，多核环境下的硬件优势来缩短 stop the world 的停顿时间。

分代收集：和其他收集器一样，分代的概念在 G1 中依然存在，不过 G1 不需要其他的垃圾回收器的配合就可以独自管理整个 GC 堆。

空间整合：G1 收集器有利于程序长时间运行，分配大对象时不会无法得到连续的空间而提前触发一次 GC。

可预测的非停顿：这是 G1 相对于 CMS 的另一大优势，降低停顿时间是 G1 和 CMS 共同的关注点，能让使用者明确指定在一个长度为 M 毫秒的时间片段内，消耗在垃圾收集上的时间不得超过 N 毫秒。

CMS：采用标记清除算法

解决这个问题的办法就是可以让 CMS 在进行一定次数的 Full GC（标记清除）的时候进行一次标记整理算法，CMS 提供了以下参数来控制：

-XX:UseCMSCompactAtFullCollection -XX:CMSFullGCBeforeCompaction=5

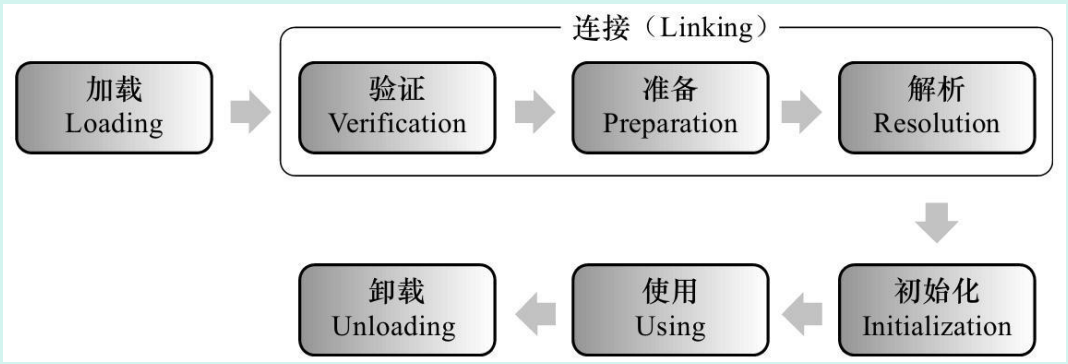
也就是 CMS 在进行 5 次 Full GC（标记清除）之后进行一次标记整理算法，从而可以控制老年带的碎片在一定的数量以内，甚至可以配置 CMS 在每次 Full GC 的时候都进行内存的整理。

2、类加载过程

考察点：JVM

参考回答：

如下图所示，JVM 类加载机制分为五个部分：加载，验证，准备，解析，初始化，下面我们就分别来看一下这五个过程。



加载

加载是类加载过程中的一个阶段，这个阶段会在内存中生成一个代表这个类的 java.lang.Class 对象，作为方法区这个类的各种数据的入口。注意这里不一定非得要从一个 Class 文件获取，这里既可以从 ZIP 包中读取（比如从 jar 包和 war 包中读取），也可以在运行时计算生成（动态代理），也可以由其它文件生成（比如将 JSP 文件转换成对应的 Class 类）。

验证

这一阶段的主要目的是为了确保 Class 文件的字节流中包含的信息是否符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

准备

准备阶段是正式为类变量分配内存并设置类变量的初始值阶段，即在方法区中分配这些变量所使用的内存空间。注意这里所说的初始值概念，比如一个类变量定义为：

```
public static int v = 8080;
```

实际上变量 v 在准备阶段过后的初始值为 0 而不是 8080，将 v 赋值为 8080 的 putstatic 指令是程序被编译后，存放于类构造器<client>方法之中，这里我们后面会解释。
但是注意如果声明为：

```
public static final int v = 8080;
```

在编译阶段会为 v 生成 ConstantValue 属性，在准备阶段虚拟机会根据 ConstantValue 属性将 v 赋值为 8080。

解析

解析阶段是指虚拟机将常量池中的符号引用替换为直接引用的过程。符号引用就是 class 文件中的：

CONSTANT_Class_info

CONSTANT_Field_info

CONSTANT_Method_info

等类型的常量。

下面我们解释一下符号引用和直接引用的概念：

符号引用与虚拟机实现的布局无关，引用的目标并不一定要已经加载到内存中。各种虚拟机实现的内存布局可以各不相同，但是它们能接受的符号引用必须是一致的，因为符号引用的字面量形式明确定义在 Java 虚拟机规范的 Class 文件格式中。

直接引用可以是指向目标的指针，相对偏移量或是一个能间接定位到目标的句柄。如果有了直接引用，那引用的目标必定已经在内存中存在。

初始化

初始化阶段是类加载最后一个阶段，前面的类加载阶段之后，除了在加载阶段可以自定义类加载器以外，其它操作都由 JVM 主导。到了初始阶段，才开始真正执行类中定义的 Java 程序代码。

初始化阶段是执行类构造器<client>方法的过程。<client>方法是由编译器自动收集类中的类变量的赋值操作和静态语句块中的语句合并而成的。虚拟机会保证<client>方法执行之前，父类的<client>方法已经执行完毕。p.s: 如果一个类中没有对静态变量赋值也没有静态语句块，那么编译器可以不为这个类生成<client>()方法。

注意以下几种情况不会执行类初始化：

通过子类引用父类的静态字段，只会触发父类的初始化，而不会触发子类的初始化。

定义对象数组，不会触发该类的初始化。

常量在编译期间会存入调用类的常量池中，本质上并没有直接引用定义常量的类，不会触发定义常量所在的类。

通过类名获取 Class 对象，不会触发类的初始化。

通过 Class.forName 加载指定类时，如果指定参数 initialize 为 false 时，也不会触发类初始化，其实这个参数是告诉虚拟机，是否要对类进行初始化。

通过 ClassLoader 默认的 loadClass 方法，也不会触发初始化动作。

类加载器

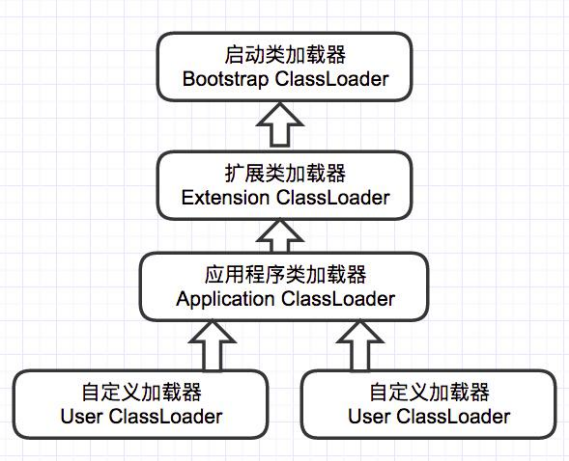
虚拟机设计团队把加载动作放到 JVM 外部实现，以便让应用程序决定如何获取所需的类，JVM 提供了 3 种类加载器：

启动类加载器(Bootstrap ClassLoader)：负责加载 JAVA_HOME\lib 目录中的，或通过 -Xbootclasspath 参数指定路径中的，且被虚拟机认可（按文件名识别，如 rt.jar）的类。

扩展类加载器(Extension ClassLoader)：负责加载 JAVA_HOME\lib\ext 目录中的，或通过 java.ext.dirs 系统变量指定路径中的类库。

应用程序类加载器(Application ClassLoader)：负责加载用户路径（classpath）上的类库。

JVM 通过双亲委派模型进行类的加载，当然我们也可以通过继承 java.lang.ClassLoader 实现自定义的类加载器。

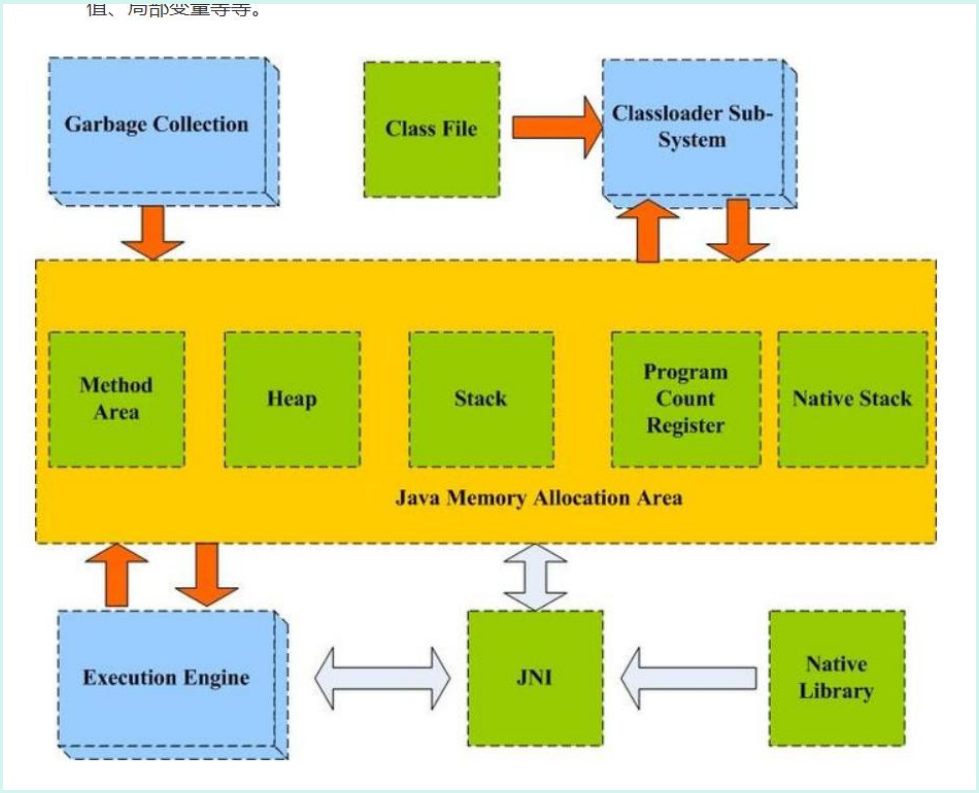


当一个类加载器收到类加载任务，会先交给其父类加载器去完成，因此最终加载任务都会传递到顶层的启动类加载器，只有当父类加载器无法完成加载任务时，才会尝试执行加载任务。采用双亲委派的一个好处是比如加载位于 rt.jar 包中的类 java.lang.Object，不管是哪个加载器加载这个类，最终都是委托给顶层的启动类加载器进行加载，这样就保证了使用不同的类加载器最终得到的都是同样一个 Object 对象。

3、JVM 分区

考察点：JVM

参考回答：



java 内存通常被划分为 5 个区域：程序计数器（Program Count Register）、本地方法栈（Native Stack）、方法区（Methon Area）、栈（Stack）、堆（Heap）。

4、eden 区，survial 区?

考察点：JVM

参考回答：

目前主流的虚拟机实现都采用了分代收集的思想，把整个堆区划分为新生代和老年代；新生代又被划分成 Eden 空间、 From Survivor 和 To Survivor 三块区域。

我们把 Eden：From Survivor：To Survivor 空间大小设成 8：1：1 ，对象总是在 Eden 区出生， From Survivor 保存当前的幸存对象， To Survivor 为空。一次 gc 发生后： 1）Eden 区活着的对象 ＋ From Survivor 存储的对象被复制到 To Survivor ；
2）清空 Eden 和 From Survivor ； 3）颠倒 From Survivor 和 To Survivor 的逻辑关系：From 变 To ， To 变 From 。可以看出，只有在 Eden 空间快满的时候才会触发 Minor GC 。而 Eden 空间占新生代的绝大部分，所以 Minor GC 的频率得以降低。当然，使用两个 Survivor 这种方式我们也付出了一定的代价，如 10% 的空间浪费、复制对象的开销等。

5、JAVA 虚拟机的作用?

考察点：java 虚拟机

参考回答：

解释运行字节码程序 消除平台相关性。

jvm 将 java 字节码解释为具体平台的具体指令。一般的高级语言如要在不同的平台上运行，至少需要编译成不同的目标代码。而引入 JVM 后，Java 语言在不同平台上运行时不需要重新编译。Java 语言使用模式 Java 虚拟机屏蔽了与具体平台相关的信息，使得 Java 语言编译程序只需生成在 Java 虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java 虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。

假设一个场景，要求 stop the world 时间非常短，你会怎么设计垃圾回收机制？

绝大多数新创建的对象分配在 Eden 区。

在 Eden 区发生一次 GC 后，存活的对象移到其中一个 Survivor 区。

在 Eden 区发生一次 GC 后，对象是存放到 Survivor 区，这个 Survivor 区已经存在其他存活的对象。

一旦一个 Survivor 区已满，存活的对象移动到另外一个 Survivor 区。然后之前那个空间已满 Survivor 区将置为空，没有任何数据。

经过重复多次这样的步骤后依旧存活的对象将被移到老年代。

6、GC 中如何判断对象需要被回收？

考察点：JAVA 虚拟机

参考回答：

即使在可达性分析算法中不可达的对象，也并非是“非回收不可”的，这时候它们暂时处于“等待”阶段，要真正宣告一个对象回收，至少要经历两次标记过程：如果对象在进行可达性分析后发现没有与 GC Roots 相连接的引用链，那它将会被第一次标记并且进行一次筛选，筛选的条件是此对象是否有必要执行 finalize()方法。当对象没有覆盖 finalize()方法，或者 finalize()方法已经被虚拟机调用过，虚拟机将这两种情况都视为“没有必要执行”。(即意味着直接回收)

如果这个对象被判定为有必要执行 finalize()方法，那么这个对象将会放置在一个叫做 F-Queue 的队列之中，并在稍后由一个由虚拟机自动建立的、低优先级的 Finalizer 线程去执行它。这里所谓的“执行”是指虚拟机会触发这个方法，但并不承诺会等待它运行结束，这样做的原因是，如果一个对象在 finalize()方法中执行缓慢，或者发生了死循环(更极端的情况)，将很可能会导致 F-Queue 队列中其他对象永久处于等待，甚至导致整个内存回收系统崩溃。

finalize()方法是对象逃脱回收的最后一次机会，稍后 GC 将对 F-Queue 中的对象进行第二次小规模的标记，如果对象要在 finalize()中跳出回收——只要重新与引用链上的任何一个对象建立关联即可，譬如把自己(this 关键字)赋值给某个类变量或者对象的成员变量，那在第二次标记时它将被移除出“即将回收”的集合；如果对象这时候还没有逃脱，那基本上它就真的被回收了。

7、JAVA 虚拟机中，哪些可作为 ROOT 对象？

考察点：JAVA 虚拟机

参考回答：

虚拟机栈中的引用对象

方法区中类静态属性引用的对象

方法区中常量引用对象

本地方法栈中 JNI 引用对象

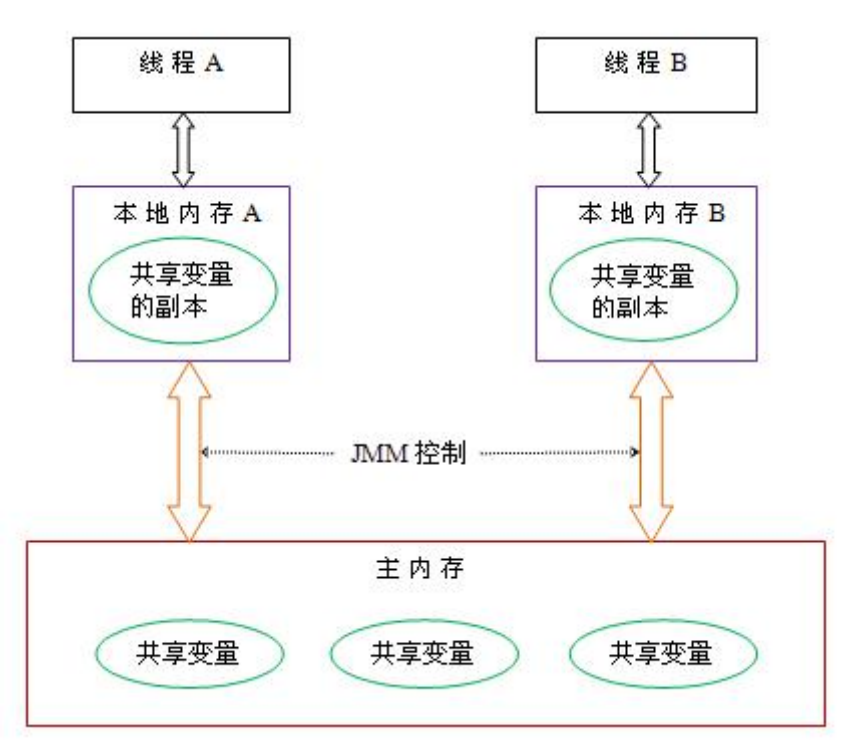
8、JVM 内存模型是什么？

考察点：JVM 内存模型

参考回答：

Java 内存模型(简称 JMM)，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（main memory）中，每个线程都有一个私有的本地内存（local memory），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。其关系模型图如下图所示：



9、jvm 是如何实现线程？

考察点：JVM

参考回答：

线程是比进程更轻量级的调度执行单位。线程可以把一个进程的资源分配和执行调度分开。一个进程里可以启动多条线程，各个线程可共享该进程的资源(内存地址，文件 IO 等)，又可以独立调度。线程是 CPU 调度的基本单位。

主流 OS 都提供线程实现。Java 语言提供对线程操作的同一 API，每个已经执行 start()，且还未结束的 java.lang.Thread 类的实例，代表了一个线程。

Thread 类的关键方法，都声明为 Native。这意味着这个方法无法或没有使用平台无关的手段来实现，也可能是为了执行效率。

实现线程的方式

A.使用内核线程实现内核线程(Kernel-Level Thread, KLT)就是直接由操作系统内核支持的线程。

内核来完成线程切换

内核通过调度器 Scheduler 调度线程，并将线程的任务映射到各个 CPU 上

程序使用内核线程的高级接口，轻量级进程(Light Weight Process,LWP)

用户态和内核态切换消耗内核资源

使用用户线程实现

系统内核不能感知线程存在的实现

用户线程的建立、同步、销毁和调度完全在用户态中完成

所有线程操作需要用户程序自己处理，复杂度高

用户线程加轻量级进程混合实现

轻量级进程作为用户线程和内核线程之间的桥梁

10、jvm 最大内存限制多少

考察点：JVM

参考回答：

(1)堆内存分配

JVM 初始分配的内存由-Xms 指定，默认是物理内存的 1/64；JVM 最大分配的内存由-Xmx 指定，默认是物理内存的 1/4。默认空余堆内存小 于 40%时，JVM 就会增大堆直到-Xmx 的最大限制；空余堆内存大于 70%时，JVM 会减少堆直到-Xms 的最小限制。因此服务器一般设置-Xms、 -Xmx 相等以避免在每次 GC 后调整堆的大小。

(2)非堆内存分配

JVM 使用-XX:PermSize 设置非堆内存初始值，默认是物理内存的 1/64；由 XX:MaxPermSize 设置最大非堆内存的大小，默认是物理内存的 1/4。

(3)VM 最大内存

首先 JVM 内存限制于实际的最大物理内存，假设物理内存无限大的话，JVM 内存的最大值跟操作系统有很大的关系。简单的说就 32 位处理器虽 然可控内存空间有 4GB,但是具体的操作系统会给一个限制，这个限制一般是 2GB-3GB（一般来说 Windows 系统下为 1.5G-2G，Linux 系 统下为 2G-3G），而 64bit 以上的处理器就不会有限制了。

(3)下面是当前比较流行的几个不同公司不同版本 JVM 最大堆内存:

公司JVM版本	最大堆内存(兆) client	最大堆内存 (兆) server
Sun 1.5.x	1492	1520
Sun 1.5.5(Linux)	2634	2660
Sun 1.4.2	1564	1564
Sun 1.4.2(Linux)	2047	N/A
BEA JRockit	1909	1902
Sun 1.6.0	1442	N/A

11、什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？

考察点：JVM

参考回答：

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。

12、描述一下 JVM 加载 class 文件的原理机制?

考察点：JVM

参考回答：

JVM 中类的装载是由 ClassLoader 和它的子类来实现的,Java ClassLoader 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的类。

Java 中的所有类，都需要由类加载器装载到 JVM 中才能运行。类加载器本身也是一个类，而它的工作就是把 class 文件从硬盘读取到内存中。在写程序的时候，我们几乎不需要关心类的加载，因为这些都是隐式装载的，除非我们有特殊的用法，像是反射，就需要显式的加载所需要的类。

类装载方式，有两种
（1）隐式装载， 程序在运行过程中当碰到通过 new 等方式生成对象时，隐式调用类装载器加载对应的类到 jvm 中，
（2）显式装载， 通过 class.forname()等方法，显式加载需要的类 ， 隐式加载与显式加载的区别：两者本质是一样的。

Java 类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类(像是基类)完全加载到 jvm 中，至于其他类，则在需要的时候才加载。这当然就是为了节省内存开销。

⑥GC

1、java 中内存泄露是啥，什么时候出现内存泄露？

考察点：内存泄漏

参考回答：

Java 中的内存泄露，广义并通俗的说，就是：不再会被使用的对象的内存不能被回收，就是内存泄露。如果长生命周期的对象持有短生命周期的引用，就很可能会出现内存泄露。

2、minor gc 如果运行的很频繁，可能是什么原因引起的，minor gc 如果运行的很慢，可能是什么原因引起的?

考察点：GC

参考回答：

可能是堆内存太小。

3、阐述 GC 算法

考察点：JVM

参考回答：

①GC（GarbageCollection 垃圾收集），GC 的对象是堆空间和永久区

②GC 算法包含：引用计数法，标记清除，标记压缩，复制算法。

③引用计数器的实现很简单，对于一个对象 A，只要有任何一个对象引用了 A，则 A 的引用计数器就加 1，当引用失效时，引用计数器就减 1。只要对象 A 的引用计数器的值为 0，则对象 A 就不可能再被使用。

④标记-清除算法是现代垃圾回收算法的思想基础。标记-清除算法将垃圾回收分为两个阶段：标记阶段和清除阶段。一种可行的实现是，在标记阶段，首先通过根节点，标记所有从根节点开始的可达对象。因此，未被标记的对象就是未被引用的垃圾对象。然后，在清除阶段，清除所有未被标记的对象。与标记-清除算法相比，复制算法是一种相对高效的回收方法不适用于存活对象较多的场合如老年代将原有的内存空间分为两块，每次只使用其中一块，在垃圾回收时，将正

在使用的内存中的存活对象复制到未使用的内存块中，之后，清除正在使用的内存块中的所有对象，交换两个内存的角色，完成垃圾回收。

4、GC 是什么？为什么要有 GC?

考察点：回收

参考回答：

GC 是垃圾收集的意思（Gabage Collection）,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。

5、垃圾回收的优点和原理。并考虑 2 种回收机制

考察点：垃圾回收

参考回答：

Java 语言中一个显著的特点就是引入了垃圾回收机制，使 c++程序员最头疼的内存管理的问题迎刃而解，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有"作用域"的概念，只有对象的引用才有"作用域"。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级别的线程运行，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楚和回收，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。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

6、java 中会存在内存泄漏吗，请简单描述。

考察点：内存

参考回答：

Java 中的确存在 Java 的内存泄漏，并且事态可以变得相当严重

Java garbage collector 自动释放哪些内存里面程序不在需要的对象，以此避免大多数的其他程序上下文的内存泄漏. 但是 Java 应用程序依旧会有相当的内存泄漏. 查找原因会十分困难.
有两类主要的 Java 内存泄漏:
* 不再需要的对象引用
* 未释放的系统资源
2.2 非必要的对象引用
Java 代码常常保留对于不再需要的对象引用，并且这组织了内存的垃圾收集器的工作. Java 对象通常被其他对象包含引用，为此一个单一对象可以保持整个对象树在内存中，于是导致了如下问题:
* 在向数组添加对象以后遗漏了对于他们的处理
* 直到你再次使用对象的时候都不释放引用. 比如一个菜单指令可以插件一个对象实例引用并且不释放便于以后再次调用的时候使用，但是也许永远不会发生.
* 在其他引用依然需要旧有状态的时候贸然修改对象状态. 比如当你为了在一个文本文件里面保存一些属性而使用一个数组，诸如”字符个数”等字段在不再需要的时候依然保留在内存当中.

* 允许一个长久执行的线程所引用的对象. 设置引用为 NULL 也无济于事, 在线程退出和空闲之前, 对象不会被收集释放
2.3 未释放的系统资源
Java 方法可以定位 Java 实例意外的堆内存, 诸如针对视窗和位图的内存资源. Java 常常通过 JNI(Java Native Interface)调用 C/C++子程序定位这些资源.

7、垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？（垃圾回收）

考察点：垃圾回收

参考回答：

对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆
(heap)中的所有对象。通过这种方式确定哪些对象是”可达的”，哪些对象是”不可达的”。当 GC 确定一些对象为”不可达”时，GC 就有责
任回收这些内存空间。可以。程序员可以手动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

⑦ IO 和 NIO，AIO

1、怎么打印日志？

考察点：异常

参考回答：

cat /var/log/*.log

如果日志在更新，如何实时查看 tail -f /var/log/messages

还可以使用 watch -d -n 1 cat /var/log/messages

-d 表示高亮不同的地方，-n 表示多少秒刷新一次。

2、运行时异常与一般异常有何异同？

考察点：异常

参考回答：

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。java 编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

3、error 和 exception 有什么区别?

考察点：异常

参考回答：

error 表示恢复不是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。
exception 表示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。

4、给我一个你最常见到的 runtime exception

考察点：异常

参考回答：

```
ArithmeticException,

ArrayStoreException,

BufferOverflowException,

BufferUnderflowException,

CannotRedoException,
CannotUndoException,

ClassCastException,

CMMException,

ConcurrentModificationException,

DOMException,
EmptyStackException,

IllegalArgumentException,

IllegalMonitorStateException,

IllegalPathStateException,
IllegalStateException,

ImagingOpException,

IndexOutOfBoundsException,

MissingResourceException,
NegativeArraySizeException,

NoSuchElementException,

NullPointerException,

ProfileDataException,
```

```
    ProviderException,
RasterFormatException, SecurityException, SystemException,
UndeclaredThrowableException, UnmodifiableSetException,
UnsupportedOperationException
```

5、Java 中的异常处理机制的简单原理和应用。

考察点：异常

参考回答：

当 JAVA 程序违反了 JAVA 的语义规则时，JAVA 虚拟机就会将发生的错误表示为一个异常。违反语义规则包括 2 种情况。一种是 JAVA 类库内置的语义检查。例如数组下标越界,会引发 IndexOutOfBoundsException;访问 null 的对象时会引发 NullPointerException。另一种情况就是 JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用 throw 关键字引发异常。所有的异常都是 java.lang.Thowable 的子类。

6、java 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？

考察点：stream

参考回答：

字节流，字符流。字节流继承于 InputStream OutputStream，字符流继承于 InputStreamReader OutputStreamWriter。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

7、什么是 java 序列化，如何实现 java 序列化？

考察点：序列化

参考回答：

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。

8、运行时异常与受检异常有什么区别？

考察点：异常

参考回答：

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，在 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

二、JavaEE 部分

1、Spring

①IoC 与 Bean 配置、管理

1、说一下 IOC 和 AOP?

考察点：spring

参考回答：

依赖注入的三种方式：（1）接口注入（2）Construct 注入（3）Setter 注入

控制反转（IoC）与依赖注入（DI）是同一个概念，引入 IOC 的目的：（1）脱开、降低类之间的耦合；（2）倡导面向接口编程、实施依赖倒换原则；（3）提高系统可插入、可测试、可修改等特性。

具体做法：（1）将 bean 之间的依赖关系尽可能地抓换为关联关系；

（2）将对具体类的关联尽可能地转换为对 Java interface 的关联，而不是与具体的服务对象相关联；

（3）Bean 实例具体关联相关 Java interface 的哪个实现类的实例，在配置信息的元数据中描述；

（4）由 IoC 组件（或称容器）根据配置信息，实例化具体 bean 类、将 bean 之间的依赖关系注入进来。

AOP（Aspect Oriented Programming），即面向切面编程，可以说是 OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP 引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过 OOP 允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对

象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP 技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

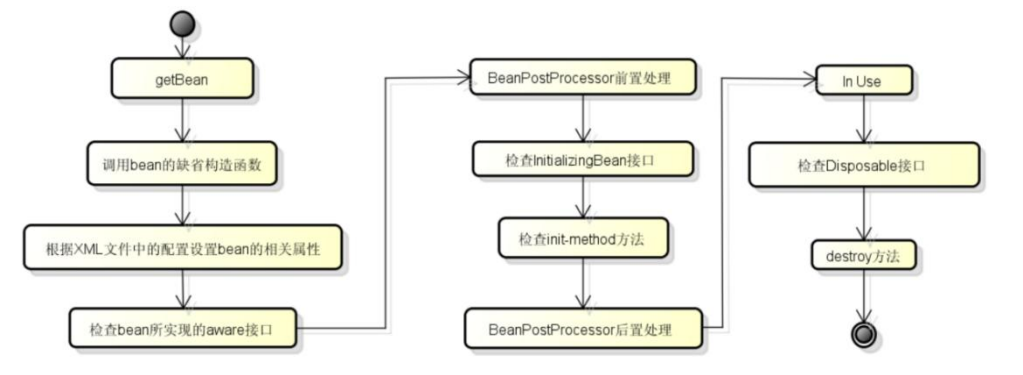
使用"横切"技术，AOP 把软件系统分为两个部分：核心关注点和横切关注点。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP 的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

2、介绍一下 bean 的生命周期

考察点：spring

参考回答：

Spring 生命周期流程图：



3、Spring 里面注解用过没有？autowired 和 resource 区别？

考察点：Spring

参考回答：

1、共同点

两者都可以写在字段和 setter 方法上。两者如果都写在字段上，那么就不需要再写 setter 方法。

2、不同点

（1）@Autowired

@Autowired 为 Spring 提供的注解，需要导入包 org.springframework.beans.factory.annotation.Autowired;只按照 byType 注入。

@Autowired 注解是按照类型（byType）装配依赖对象，默认情况下它要求依赖对象必须存在，如果允许 null 值，可以设置它的 required 属性为 false。如果我们想使用按照名称（byName）来装配，可以结合@Qualifier 注解一起使用。

（2）@Resource

@Resource 默认按照 ByName 自动注入，由 J2EE 提供，需要导入包 javax.annotation.Resource。@Resource 有两个重要的属性：name 和 type，而 Spring 将 @Resource 注解的 name 属性解析为 bean 的名字，而 type 属性则解析为 bean 的类型。所以，如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不制定 name 也不制定 type 属性，这时将通过反射机制使用 byName 自动注入策略。

4、@Controller 和@RestController 的区别？

考察点：spring

参考回答：

@RestController 注解相当于@ResponseBody ＋ @Controller 合在一起的作用

5、依赖注入的方式有几种，哪几种？

考察点：spring

参考回答：

1、Set 注入 2、构造器注入 3、接口注入

6、springIOC 原理？自己实现 IOC 要怎么做，哪些步骤？

考察点：spring

参考回答：

①IoC（Inversion of Control，控制倒转）。这是 spring 的核心，贯穿始终。所谓 IoC，对于 spring 框架来说，就是由 spring 来负责控制对象的生命周期和对象间的关系。

IoC 的一个重点是在系统运行中，动态的向某个对象提供它所需要的其他对象。这一点是通过 DI（Dependency Injection，依赖注入）来实现的。比如对象 A 需要操作数据库，以前我们总是要在 A 中自己编写代码来获得一个 Connection 对象，有了 spring 我们就只需要告诉 spring，A 中需要一个 Connection，至于这个 Connection 怎么构造，何时构造，A 不需要知道。在系统运行时，spring 会在适当的时候制造一个 Connection，然后像打针一样，注射到 A 当中，这样就完成了对各个对象之间关系的控制。A 需要依赖 Connection 才能正常运行，而这个 Connection 是由 spring 注入到 A 中的，依赖注入的名字就这么来的。那么 DI 是如何实现的呢？ Java 1.3 之后一个重要特征是反射（reflection），它允许程序在运行的时候动态的生成对象、执行对象的方法、改变对象的属性，spring 就是通过反射来实现注入的。

举个简单的例子，我们找女朋友常见的情况是，我们到处去看哪里有长得漂亮身材又好的女孩子，然后打听她们的兴趣爱好、qq 号、电话号、ip 号、iq 号……，想办法认识她们，投其所好送其所要，这个过程是复杂深奥的，我们必须自己设计和面对每个环节。传统的程序开发也是如此，在一个对象中，如果要使用另外的对象，就必须得到它（自己 new 一个，或者从 JNDI

中查询一个），使用完之后还要将对象销毁（比如 Connection 等），对象始终会和其他的接口或类藕合起来。

②实现 IOC 的步骤

定义用来描述 bean 的配置的 Java 类

解析 bean 的配置，将 bean 的配置信息转换为上面的 BeanDefinition 对象保存在内存中，spring 中采用 HashMap 进行对象存储，其中会用到一些 xml 解析技术

遍历存放 BeanDefinition 的 HashMap 对象，逐条取出 BeanDefinition 对象，获取 bean 的配置信息，利用 Java 的反射机制实例化对象，将实例化后的对象保存在另外一个 Map 中即可。

7、Spring 中 BeanFactory 和 ApplicationContext 的区别？、

考察点：spring 框架

参考回答：

概念：

BeanFactory：
BeanFactory 是 spring 中比较原始，比较古老的 Factory。因为比较古老，所以 BeanFactory 无法支持 spring 插件，例如：AOP、Web 应用等功能。

ApplicationContext
ApplicationContext 是 BeanFactory 的子类，因为古老的 BeanFactory 无法满足不断更新的 spring 的需求，于是 ApplicationContext 就基本上代替了 BeanFactory 的工作，以一种更面向框架的工作方式以及对上下文进行分层和实现继承，并在这个基础上对功能进行扩展：
<1>MessageSource，提供国际化的消息访问
<2>资源访问（如 URL 和文件）
<3>事件传递
<4>Bean 的自动装配
<5>各种不同应用层的 Context 实现

区别：

<1>如果使用 ApplicationContext，如果配置的 bean 是 singleton，那么不管你有没有或想不想用它，它都会被实例化。好处是可以预先加载，坏处是浪费内存。
<2>BeanFactory，当使用 BeanFactory 实例化对象时，配置的 bean 不会马上被实例化，而是等到你使用该 bean 的时候（getBean）才会被实例化。好处是节约内存，坏处是速度比较慢。多用于移动设备的开发。
<3>没有特殊要求的情况下，应该使用 ApplicationContext 完成。因为 BeanFactory 能完成的事情，ApplicationContext 都能完成，并且提供了更多接近现在开发的功能。

8、什么是 IoC 和 DI？DI 是如何实现的？

考察点：控制反转

参考回答：

IoC 叫控制反转，是 Inversion of Control 的缩写，DI（Dependency Injection）叫依赖注入，是对 IoC 更简单的诠释。控制反转是把传统上由程序代码直接操控的对象的调用权交给容

器，通过容器来实现对象组件的装配和管理。所谓的"控制反转"就是对组件对象控制权的转移，从程序代码本身转移到了外部容器，由容器来创建对象并管理对象之间的依赖关系。IoC 体现了好莱坞原则 - "Don't call me, we will call you"。依赖注入的基本原则是应用组件不应该负责查找资源或者其他依赖的协作对象。配置对象的工作应该由容器负责，查找资源的逻辑应该从应用组件的代码中抽取出来，交给容器来完成。DI 是对 IoC 更准确的描述，即组件之间的依赖关系由容器在运行期决定，形象的来说，即由容器动态的将某种依赖关系注入到组件之中。

一个类 A 需要用到接口 B 中的方法，那么就需要为类 A 和接口 B 建立关联或依赖关系，最原始的方法是在类 A 中创建一个接口 B 的实现类 C 的实例，但这种方法需要开发人员自行维护二者的依赖关系，也就是说当依赖关系发生变动的时候需要修改代码并重新构建整个系统。如果通过一个容器来管理这些对象以及对象的依赖关系，则只需要在类 A 中定义好用于关联接口 B 的方法（构造器或 setter 方法），将类 A 和接口 B 的实现类 C 放入容器中，通过对容器的配置来实现二者的关联。
依赖注入可以通过 setter 方法注入（设值注入）、构造器注入和接口注入三种方式来实现，Spring 支持 setter 注入和构造器注入，通常使用构造器注入来注入必须的依赖关系，对于可选的依赖关系，则 setter 注入是更好的选择，setter 注入需要类提供无参构造器或者无参的静态工厂方法来创建对象。

9、请问 Spring 中 Bean 的作用域有哪些？

考察点：框架

参考回答：

在 Spring 的早期版本中，仅有两个作用域：singleton 和 prototype，前者表示 Bean 以单例的方式存在；后者表示每次从容器中调用 Bean 时，都会返回一个新的实例，prototype 通常翻译为原型。

设计模式中的创建型模式中也有一个原型模式，原型模式也是一个常用的模式，例如做一个室内设计软件，所有的素材都在工具箱中，而每次从工具箱中取出的都是素材对象的一个原型，可以通过对象克隆来实现原型模式。Spring 2.x 中针对 WebApplicationContext 新增了 3 个作用域，分别是：request（每次 HTTP 请求都会创建一个新的 Bean）、session（同一个 HttpSession 共享同一个 Bean，不同的 HttpSession 使用不同的 Bean）和 globalSession（同一个全局 Session 共享一个 Bean）。

单例模式和原型模式都是重要的设计模式。一般情况下，无状态或状态不可变的类适合使用单例模式。在传统开发中，由于 DAO 持有 Connection 这个非线程安全对象因而没有使用单例模式；但在 Spring 环境下，所有 DAO 类对可以采用单例模式，因为 Spring 利用 AOP 和 Java API 中的 ThreadLocal 对非线程安全的对象进行了特殊处理。

10、谈谈 Spring 中自动装配的方式有哪些？

考察点：spring 框架

参考回答：

- no：不进行自动装配，手动设置 Bean 的依赖关系。
- byName：根据 Bean 的名字进行自动装配。
- byType：根据 Bean 的类型进行自动装配。
- constructor：类似于 byType，不过是应用于构造器的参数，如果正好有一个 Bean 与构造器的参数类型相同则可以自动装配，否则会导致错误。
- autodetect：如果有默认的构造器，则通过 constructor 的方式进行自动装配，否则使用 byType 的方式进行自动装配。

自动装配没有自定义装配方式那么精确，而且不能自动装配简单属性(基本类型、字符串等)，在使用时应注意。

②AOP 与事务、权限控制

1、aop 的应用场景？

考察点：spring AOP

参考回答：

Authentication 权限 ，Caching 缓存 ，Context passing 内容传递 ，Error handling 错误处理 ，Lazy loading 懒加载 ，Debugging 调试 ，logging, tracing, profiling and monitoring 记录跟踪 优化 校准，Performance optimization 性能优化 ，Persistence 持久化 ，Resource pooling 资源池 ，Synchronization 同步，Transactions 事务。

2、AOP 的原理是什么？

考察点：动态代理

参考回答：

AOP（Aspect Orient Programming），指面向方面（切面）编程，作为面向对象的一种补充，用于处理系统中分布于各个模块的横切关注点，比如事务管理、日志、缓存等等。AOP 实现的关键在于 AOP 框架自动创建的 AOP 代理，AOP 代理主要分为静态代理和动态代理，静态代理的代表为 AspectJ；而动态代理则以 Spring AOP 为代表。通常使用 AspectJ 的编译时增强实现 AOP，AspectJ 是静态代理的增强，所谓的静态代理就是 AOP 框架会在编译阶段生成 AOP 代理类，因此也称为编译时增强。

Spring AOP 中的动态代理主要有两种方式，JDK 动态代理和 CGLIB 动态代理。JDK 动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个接口。JDK 动态代理的核心是 InvocationHandler 接口和 Proxy 类。

如果目标类没有实现接口，那么 Spring AOP 会选择使用 CGLIB 来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB 是通过继承的方式做的动态代理，因此如果某个类被标记为 final，那么它是无法使用 CGLIB 做动态代理的。

3、你如何理解 AOP 中的连接点（Joinpoint）、切点（Pointcut）、增强（Advice）、引介（Introduction）、织入（Weaving）、切面（Aspect）这些概念？

考察点：AOP

参考回答：

a. 连接点（Joinpoint）：程序执行的某个特定位置（如：某个方法调用前、调用后，方法抛出异常后）。一个类或一段程序代码拥有一些具有边界性质的特定点，这些代码中的特定点就是连接点。Spring 仅支持方法的连接点。
b. 切点（Pointcut）：如果连接点相当于数据中的记录，那么切点相当于查询条件，一个切点可以匹配多个连接点。Spring AOP 的规则解析引擎负责解析切点所设定的查询条件，找到对应

的连接点。
c. 增强（Advice）：增强是织入到目标类连接点上的一段程序代码。Spring 提供的增强接口都是带方位名的，如：BeforeAdvice、AfterReturningAdvice、ThrowsAdvice 等。

d. 引介（Introduction）：引介是一种特殊的增强，它为类添加一些属性和方法。这样，即使一个业务类原本没有实现某个接口，通过引介功能，可以动态的未该业务类添加接口的实现逻辑，让业务类成为这个接口的实现类。
e. 织入（Weaving）：织入是将增强添加到目标类具体连接点上的过程，AOP 有三种织入方式：①编译期织入：需要特殊的 Java 编译期（例如 AspectJ 的 ajc）；②装载期织入：要求使用特殊的类加载器，在装载类的时候对类进行增强；③运行时织入：在运行时为目标类生成代理实现增强。Spring 采用了动态代理的方式实现了运行时织入，而 AspectJ 采用了编译期织入和装载期织入的方式。
f. 切面（Aspect）：切面是由切点和增强（引介）组成的，它包括了对横切关注功能的定义，也包括了对连接点的定义。

4、Spring 支持的事务管理类型有哪些？你在项目中使用哪种方式？

考察点：事物管理

参考回答：

Spring 支持编程式事务管理和声明式事务管理。许多 Spring 框架的用户选择声明式事务管理，因为这种方式和应用程序的关联较少，因此更加符合轻量级容器的概念。声明式事务管理要优于编程式事务管理，尽管在灵活性方面它弱于编程式事务管理，因为编程式事务允许你通过代码控制业务。

事务分为全局事务和局部事务。全局事务由应用服务器管理，需要底层服务器 JTA 支持（如 WebLogic、WildFly 等）。局部事务和底层采用的持久化方案有关，例如使用 JDBC 进行持久化时，需要使用 Connetion 对象来操作事务；而采用 Hibernate 进行持久化时，需要使用 Session 对象来操作事务。

这些事务的父接口都是 PlatformTransactionManager。Spring 的事务管理机制是一种典型的策略模式，PlatformTransactionManager 代表事务管理接口，该接口定义了三个方法，该接口并不知道底层如何管理事务，但是它的实现类必须提供 getTransaction()方法（开启事务）、commit()方法（提交事务）、rollback()方法（回滚事务）的多态实现，这样就可以用不同的实现类代表不同的事务管理策略。使用 JTA 全局事务策略时，需要底层应用服务器支持，而不同的应用服务器所提供的 JTA 全局事务可能存在细节上的差异，因此实际配置全局事务管理器是可能需要使用 JtaTransactionManager 的子类，如：WebLogicJtaTransactionManager（Oracle 的 WebLogic 服务器提供）、UowJtaTransactionManager（IBM 的 WebSphere 服务器提供）等。

③S2SH 整合开发

1、介绍一下 spring?

考察点：spring

参考回答：

Spring 是一个轻量级框架，可以一站式构建你的企业级应用。

Spring 的模块大概分为 6 个。分别是：

1、Core Container（Spring 的核心）【重要】

2、AOP（面向切面变成）【重要】

3、Messaging（消息发送的支持）

4、Data Access/Integration（数据访问和集成）

5、Web（主要是 SpringWeb 内容，包括 MVC）【重要】

6、Test（Spring 测试支持，包含 JUint 等测试单元的支持） 7、Instrumentation（设备支持，比如 Tomcat 的支持）

2、Struts 拦截器和 Spring AOP 区别？

考察点：框架

参考回答：

拦截器是 AOP 的一种实现，struts2 拦截器采用 xwork2 的 interceptor！而 spring 的 AOP 基于 IoC 基础,其底层采用动态代理与 CGLIB 代理两种方式结合的实现方式。

3、spring 框架的优点？

考察点：spring

参考回答：

Spring 是一个轻量级的 DI 和 AOP 容器框架，在项目的中的使用越来越广泛，它的优点主要有以下几点：

Spring 是一个非侵入式框架，其目标是使应用程序代码对框架的依赖最小化，应用代码可以在没有 Spring 或者其他容器的情况运行。

Spring 提供了一个一致的编程模型，使应用直接使用 POJO 开发，从而可以使运行环境隔离开来。

Spring 推动应用的设计风格向面向对象及面向接口编程转变，提高了代码的重用性和可测试性。

Spring 改进了结构体系的选择，虽然作为应用平台，Spring 可以帮助我们选择不同的技术实现，比如从 Hibernate 切换到其他的 ORM 工具，从 Struts 切换到 Spring MVC,尽管我们通常不会这么做，但是我们在技术方案上选择使用 Spring 作为应用平台，Spring 至少为我们提供了这种可能性的选择，从而降低了平台锁定风险。

4、选择使用 Spring 框架的原因(Spring 框架为企业级开发带来的好处有哪些)？

考察点：框架

参考回答：

- 非侵入式：支持基于 POJO 的编程模式，不强制性的要求实现 Spring 框架中的接口或继承 Spring 框架中的类。
- IoC 容器：IoC 容器帮助应用程序管理对象以及对象之间的依赖关系，对象之间的依赖关系如果发生了改变只需要修改配置文件而不是修改代码，因为代码的修改可能意味着项目的重新构建和完整的回归测试。有了 IoC 容器，程序员再也不需要自己编写工厂、单例，这一点特别符合 Spring 的精神"不要重复的发明轮子"。
- AOP（面向切面编程）：将所有的横切关注功能封装到切面（aspect）中，通过配置的方式将横切关注功能动态添加到目标代码上，进一步实现了业务逻辑和系统服务之间的分离。另一方面，有了 AOP 程序员可以省去很多自己写代理类的工作。
- MVC：Spring 的 MVC 框架为 Web 表示层提供了更好的解决方案。
- 事务管理：Spring 以宽广的胸怀接纳多种持久层技术，并且为其提供了声明式的事务管理，在不需要任何一行代码的情况下就能够完成事务管理。
- 其他：选择 Spring 框架的原因还远不止于此，Spring 为 Java 企业级开发提供了一站式选择，你可以在需要的时候使用它的部分和全部，更重要的是，甚至可以在感觉不到 Spring 存在的情况下，在你的项目中使用 Spring 提供的各种优秀的功能。

④Spring，JPA 整合

1、持久层设计要考虑的问题有哪些？你用过的持久层框架有哪些？

考察点：框架

参考回答：

所谓"持久"就是将数据保存到可掉电式存储设备中以便今后使用，简单的说，就是将内存中的数据保存到关系型数据库、文件系统、消息队列等提供持久化支持的设备中。持久层就是系统中专注于实现数据持久化的相对独立的层面。

持久层设计的目标包括：
- 数据存储逻辑的分离，提供抽象化的数据访问接口。
- 数据访问底层实现的分离，可以在不修改代码的情况下切换底层实现。
- 资源管理和调度的分离，在数据访问层实现统一的资源调度（如缓存机制）。
- 数据抽象，提供更面向对象的数据操作。

持久层框架有：
- Hibernate
- MyBatis
- TopLink
- Guzz
- jOOQ
- Spring Data
- ActiveJDBC

2、Hibernate

①ORM 与持久化映射

1、阐述实体对象的三种状态以及转换关系。

考察点：JAVA 实体

参考回答：

最新的 Hibernate 文档中为 Hibernate 对象定义了四种状态（原来是三种状态，面试的时候基本上问的也是三种状态），分别是：瞬时态（new, or transient）、持久态（managed, or persistent）、游状态（detached）和移除态（removed，以前 Hibernate 文档中定义的三种状态中没有移除态），如下图所示，就以前的 Hibernate 文档中移除态被视为是瞬时态。

瞬时态：当 new 一个实体对象后，这个对象处于瞬时态，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 JVM 的垃圾回收机制回收。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()、saveOrUpdate()、persist()、merge()方法把瞬时态对象与数据库关联，并把数据插入或者更新到数据库，这个对象才转换为持久态对象。
持久态：持久态对象的实例在数据库中有对应的记录，并拥有一个持久化标识（ID）。对持久态对象进行 delete 操作后，数据库中对应的记录将被删除，那么持久态对象与数据库记录不再存在对应关系，持久态对象变成移除态（可以视为瞬时态）。持久态对象被修改变更后，不会马上同步到数据库，直到数据库事务提交。
游离态：当 Session 进行了 close()、clear()、evict()或 flush()后，实体对象从持久态变成游离态，对象虽然拥有持久和与数据库对应记录一致的标识值，但是因为对象已经从会话中清除掉，对象不在持久化管理之内，所以处于游离态（也叫脱管态）。游离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。

②延迟加载、性能优化

1、Hibernate 中 SessionFactory 是线程安全的吗？Session 是线程安全的吗（两个线程能够共享同一个 Session 吗）？

考察点：session

参考回答：

SessionFactory 对应 Hibernate 的一个数据存储的概念，它是线程安全的，可以被多个线程并发访问。SessionFactory 一般只会在启动的时候构建。对于应用程序，最好将 SessionFactory 通过单例模式进行封装以便于访问。Session 是一个轻量级非线程安全的对象（线程间不能共享 session），它表示与数据库进行交互的一个工作单元。Session 是由 SessionFactory 创建的，在任务完成之后它会被关闭。Session 是持久层服务对外提供的主要接口。Session 会延迟获取数据库连接（也就是在需要的时候才会获取）。为了避免创建太多的 session，可以使用 ThreadLocal 将 session 和当前线程绑定在一起，这样可以让同一个线程获得的总是同一个 session。Hibernate 3 中 SessionFactory 的 getCurrentSession()方法就可以做到。

2、Hibernate 中 Session 的 load 和 get 方法的区别是什么？

考察点：请求方式

参考回答：

主要有以下三项区别：
① 如果没有找到符合条件的记录，get 方法返回 null，load 方法抛出异常。
② get 方法直接返回实体类对象，load 方法返回实体类对象的代理。
③ 在 Hibernate 3 之前，get 方法只在一级缓存中进行数据查找，如果没有找到对应的数据则越过二级缓存，直接发出 SQL 语句完成数据读取；load 方法则可以从二级缓存中获取数据；从 Hibernate 3 开始，get 方法不再是对二级缓存只写不读，它也是可以访问二级缓存的。
对于 load()方法 Hibernate 认为该数据在数据库中一定存在可以放心的使用代理来实现延迟加载，如果没有数据就抛出异常，而通过 get()方法获取的数据可以不存在。

3、如何理解 Hibernate 的延迟加载机制？在实际应用中，延迟加载与 Session 关闭的矛盾是如何处理的？

考察点：hibernate 框架

参考回答：

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。Hibernate 使用了虚拟代理机制实现延迟加载，我们使用 Session 的 load()方法加载数据或者一对多关联映射在使用延迟加载的情况下从一的一方加载多的一方，得到的都是虚拟代理，简单的说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 方法时才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。

延迟加载与 session 关闭的矛盾一般可以这样处理：
① 关闭延迟加载特性。这种方式操作起来比较简单，因为 Hibernate 的延迟加载特性是可以通过映射文件或者注解进行配置的，但这种解决方案存在明显的缺陷。首先，出现"no session or session was closed"通常说明系统中已经存在主外键关联，如果去掉延迟加载的话，每次查询的开销都会变得很大。
② 在 session 关闭之前先获取需要查询的数据，可以使用工具方法 Hibernate.isInitialized()判断对象是否被加载，如果没有被加载则可以使用 Hibernate.initialize()方法加载对象。
③ 使用拦截器或过滤器延长 Session 的生命周期直到视图获得数据。Spring 整合 Hibernate 提供的 OpenSessionInViewFilter 和 OpenSessionInViewInterceptor 就是这种做法。

4、简述 Hibernate 常见优化策略。

考察点：Hibernate

参考回答：

①制定合理的缓存策略（二级缓存、查询缓存）。
② 采用合理的 Session 管理机制。
③ 尽量使用延迟加载特性。
④ 设定合理的批处理参数。
⑤ 如果可以，选用 UUID 作为主键生成器。
⑥ 如果可以，选用基于版本号的乐观锁替代悲观锁。
⑦ 在开发过程中，开启 hibernate.show_sql 选项查看生成的 SQL，从而了解底层的状况；开发完成后关闭此选项。

⑧ 考虑数据库本身的优化，合理的索引、恰当的数据分区策略等都会对持久层的性能带来可观的提升，但这些需要专业的 DBA（数据库管理员）提供支持。

5、锁机制有什么用？简述 Hibernate 的悲观锁和乐观锁机制。

考察点：锁

参考回答：

有些业务逻辑在执行过程中要求对数据进行排他性的访问，于是需要通过一些机制保证在此过程中数据被锁住不会被外界修改，这就是所谓的锁机制。
Hibernate 支持悲观锁和乐观锁两种锁机制。悲观锁，顾名思义悲观的认为在数据处理过程中极有可能存在修改数据的并发事务（包括本系统的其他事务或来自外部系统的事务），于是将处理的数据设置为锁定状态。悲观锁必须依赖数据库本身的锁机制才能真正保证数据访问的排他性，乐观锁，顾名思义，对并发事务持乐观态度（认为对数据的并发操作不会经常性的发生），通过更加宽松的锁机制来解决由于悲观锁排他性的数据访问对系统性能造成的严重影响。最常见的乐观锁是通过数据版本标识来实现的，读取数据时获得数据的版本号，更新数据时将此版本号加 1，然后和数据库表对应记录的当前版本号进行比较，如果提交的数据版本号大于数据库中此记录的当前版本号则更新数据，否则认为是过期数据无法更新。Hibernate 中通过 Session 的 get()和 load()方法从数据库中加载对象时可以通过参数指定使用悲观锁；而乐观锁可以通过给实体类加整型的版本字段再通过 XML 或@Version 注解进行配置。

使用乐观锁会增加了一个版本字段，很明显这需要额外的空间来存储这个版本字段，浪费了空间，但是乐观锁会让系统具有更好的并发性，这是对时间的节省。因此乐观锁也是典型的空间换时间的策略。

③HQL 查询、条件查询、SQL 查询

1、Hibernate 如何实现分页查询？

考察点：框架

参考回答：

通过 Hibernate 实现分页查询，开发人员只需要提供 HQL 语句（调用 Session 的 createQuery()方法）或查询条件（调用 Session 的 createCriteria()方法）、设置查询起始行数（调用 Query 或 Criteria 接口的 setFirstResult()方法）和最大查询行数（调用 Query 或 Criteria 接口的 setMaxResults()方法），并调用 Query 或 Criteria 接口的 list()方法，Hibernate 会自动生成分页查询的 SQL 语句。

④二级缓存与查询缓存

1、谈一谈 Hibernate 的一级缓存、二级缓存和查询缓存。

考察点：缓存

参考回答：

Hibernate 的 Session 提供了一级缓存的功能，默认总是有效的，当应用程序保存持久化实体、修改持久化实体时，Session 并不会立即把这种改变提交到数据库，而是缓存在当前的 Session 中，除非显示调用了 Session 的 flush()方法或通过 close()方法关闭 Session。通过一级缓存，可以减少程序与数据库的交互，从而提高数据库访问性能。
SessionFactory 级别的二级缓存是全局性的，所有的 Session 可以共享这个二级缓存。不过二级缓存默认是关闭的，需要显示开启并指定需要使用哪种二级缓存实现类（可以使用第三方提供的实现）。一旦开启了二级缓存并设置了需要使用二级缓存的实体类，SessionFactory 就会缓存访问过的该实体类的每个对象，除非缓存的数据超出了指定的缓存空间。
一级缓存和二级缓存都是对整个实体进行缓存，不会缓存普通属性，如果希望对普通属性进行缓存，可以使用查询缓存。查询缓存是将 HQL 或 SQL 语句以及它们的查询结果作为键值对进行缓存，对于同样的查询可以直接从缓存中获取数据。查询缓存默认也是关闭的，需要显示开启。

3、Struts

①MVC 模式与 Struts 体系

1、说说 STRUTS 的应用

考察点：STRUTS 架构

参考回答：

Struts 是采用 Java Servlet/JavaServer Pages 技术，开发 Web 应用程序的开放源码的 framework。采用 Struts 能开发出基于 MVC(Model-View-Controller)设计模式的应用构架。Struts 有如下的主要功能：

包含一个 controller servlet，能将用户的请求发送到相应的 Action 对象。二. JSP 自由 tag 库，并且在 controller servlet 中提供关联支持，帮助开发员创建交互式表单应用。

提供了一系列实用对象：XML 处理、通过 Java reflection APIs 自动处理 JavaBeans 属性、国际化的提示和消息。

4、Mybatis

1、解释一下 MyBatis 中命名空间（namespace）的作用。

考察点：Mybatis

参考回答：

在大型项目中，可能存在大量的 SQL 语句，这时候为每个 SQL 语句起一个唯一的标识（ID）就变得并不容易了。为了解决这个问题，在 MyBatis 中，可以为每个映射文件起一个唯一的命名空间，这样定义在这个映射文件中的每个 SQL 语句就成了定义在这个命名空间中的一个 ID。只要我们能够保证每个命名空间中这个 ID 是唯一的，即使在不同映射文件中的语句 ID 相同，也不会再产生冲突了。

2、MyBatis 中的动态 SQL 是什么意思？

考察点：SQL

参考回答：

对于一些复杂的查询，我们可能会指定多个查询条件，但是这些条件可能存在也可能不存在，需要根据用户指定的条件动态生成 SQL 语句。如果不使用持久层框架我们可能需要自己拼装 SQL 语句，还好 MyBatis 提供了动态 SQL 的功能来解决这个问题。MyBatis 中用于实现动态 SQL 的元素主要有：

- if
- choose / when / otherwise
- trim
- where
- set
- foreach

5、MVC

1、Spring MVC 注解的优点

考察点：spring mvc

参考回答：

1、XML 配置起来有时候冗长，此时注解可能是更好的选择，如 jpa 的实体映射；注解在处理一些不变的元数据时有时候比 XML 方便的多，比如 springmvc 的数据绑定，如果用 xml 写的代码会多的多；

2、注解最大的好处就是简化了 XML 配置；其实大部分注解一定确定后很少会改变，所以在一些中小项目中使用注解反而提供了开发效率，所以没必要一头走到黑；

3、注解相对于 XML 的另一个好处是类型安全的，XML 只能在运行期才能发现问题。

2、springmvc 和 spring-boot 区别？

考察点：spring

参考回答：

总的来说，Spring 就像一个大家族，有众多衍生产品例如 Boot，Security，JPA 等等。但他们的基础都是 Spring 的 IOC 和 AOP，IOC 提供了依赖注入的容器，而 AOP 解决了面向切面的编程，然后在此两者的基础上实现了其他衍生产品的高级功能；因为 Spring 的配置非常复杂，各种 xml，properties 处理起来比较繁琐。于是为了简化开发者的使用，Spring 社区创造性地推出了 Spring Boot，它遵循约定优于配置，极大降低了 Spring 使用门槛，

但又不失 Spring 原本灵活强大的功能。

3、SpringMVC 的运行机制，运行机制的每一部分的相关知识？

考察点：spring

参考回答：

1、用户发送请求时会先从 DispathcherServler 的 doService 方法开始，在该方法中会将 ApplicationContext、localeResolver、themeResolver 等对象添加到 request 中，紧接着就是调用 doDispatch 方法。

2、进入该方法后首先会检查该请求是否是文件上传的请求(校验的规则是是否是 post 并且 contenttType 是否为 multipart/为前缀)即调用的是 checkMultipart 方法;如果是的将 request 包装成 MultipartHttpServletRequest。

3、然后调用 getHandler 方法来匹配每个 HandlerMapping 对象，如果匹配成功会返回这个 Handle 的处理链 HandlerExecutionChain 对象，在获取该对象的内部其实也获取我们自定定义的拦截器，并执行了其中的方法。

4、执行拦截器的 preHandle 方法，如果返回 false 执行 afterCompletion 方法并理解返回

5、通过上述获取到了 HandlerExecutionChain 对象，通过该对象的 getHandler()方法获得一个 object 通过 HandlerAdapter 进行封装得到 HandlerAdapter 对象。

6、该对象调用 handle 方法来执行 Controller 中的方法,该对象如果返回一个 ModelAndView 给 DispatcherServlet。

7、DispatcherServlet 借助 ViewResolver 完成逻辑试图名到真实视图对象的解析,得到 View 后 DispatcherServlet 使用这个 View 对 ModelAndView 中的模型数据进行视图渲染。

4、谈谈 Spring MVC 的工作原理是怎样的？

考察点：设计模式

参考回答：

①客户端的所有请求都交给前端控制器 DispatcherServlet 来处理,它会负责调用系统的其他模块来真正处理用户的请求。
② DispatcherServlet 收到请求后，将根据请求的信息（包括 URL、HTTP 协议方法、请求头、请求参数、Cookie 等）以及 HandlerMapping 的配置找到处理该请求的 Handler（任何一个对象都可以作为请求的 Handler）。
③在这个地方 Spring 会通过 HandlerAdapter 对该处理器进行封装。
④ HandlerAdapter 是一个适配器，它用统一的接口对各种 Handler 中的方法进行调用。
⑤ Handler 完成对用户请求的处理后，会返回一个 ModelAndView 对象给 DispatcherServlet，ModelAndView 顾名思义，包含了数据模型以及相应的视图的信息。
⑥ ModelAndView 的视图是逻辑视图，DispatcherServlet 还要借助 ViewResolver 完成从逻辑视图到真实视图对象的解析工作。
⑦ 当得到真正的视图对象后，DispatcherServlet 会利用视图对象对模型数据进行渲染。
⑧ 客户端得到响应，可能是一个普通的 HTML 页面，也可以是 XML 或 JSON 字符串，还可以是一张图片或者一个 PDF 文件。

6、各框架对比与项目优化

1、Mybatis 和 Hibernate 区别？

考察点：Spring 框架

参考回答：

1. 简介

Hibernate：Hibernate 是当前最流行的 ORM 框架之一，对 JDBC 提供了较为完整的封装。Hibernate 的 O/R Mapping 实现了 POJO 和数据库表之间的映射，以及 SQL 的自动生成和执行。

Mybatis：Mybatis 同样也是非常流行的 ORM 框架，主要着力点在于 POJO 与 SQL 之间的映射关系。然后通过映射配置文件，将 SQL 所需的参数，以及返回的结果字段映射到指定 POJO 。相对 Hibernate“O/R”而言，Mybatis 是一种“Sql Mapping”的 ORM 实现。

2、缓存机制对比

相同点

Hibernate 和 Mybatis 的二级缓存除了采用系统默认的缓存机制外，都可以通过实现你自己的缓存或为其他第三方缓存方案，创建适配器来完全覆盖缓存行为。

不同点

Hibernate 的二级缓存配置在 SessionFactory 生成的配置文件中进行详细配置，然后再在具体的表-对象映射中配置是那种缓存。

MyBatis 的二级缓存配置都是在每个具体的表-对象映射中进行详细配置，这样针对不同的表可以自定义不同的缓存机制。并且 Mybatis 可以在命名空间中共享相同的缓存配置和实例，通过 Cache-ref 来实现。

两者比较

因为 Hibernate 对查询对象有着良好的管理机制，用户无需关心 SQL。所以在使用二级缓存时如果出现脏数据，系统会报出错误并提示。而 MyBatis 在这一方面，使用二级缓存时需要特别小心。如果不能完全确定数据更新操作的波及范围，避免 Cache 的盲目使用。否则，脏数据的出现会给系统的正常运行带来很大的隐患。

Mybatis：小巧、方便、高效、简单、直接、半自动化

Hibernate：强大、方便、高效、复杂、间接、全自动化

2、介绍一下你了解的 Java 领域的 Web Service 框架。

考察点：框架

参考回答：

Java 领域的 Web Service 框架很多，包括 Axis2（Axis 的升级版本）、Jersey（RESTful 的 Web Service 框架）、CXF（XFire 的延续版本）、Hessian、Turmeric、JBoss SOA 等，其中绝大多数都是开源框架。

7、JPA

①EJB

1、EJB 是基于哪些技术实现的？并说出 SessionBean 和 EntityBean 的区别，StatefulBean 和 StatelessBean 的区别。

考察点：JAVA EJB

参考回答：

EJB 包括 Session Bean、Entity Bean、Message Driven Bean，基于 JNDI、RMI、JAT 等技术实现。
SessionBean 在 J2EE 应用程序中被用来完成一些服务器端的业务操作，例如访问数据库、调用其他 EJB 组件。EntityBean 被用来代表应用系统中用到的数据。
对于客户机，SessionBean 是一种非持久性对象，它实现某些在服务器上运行的业务逻辑。
对于客户机，EntityBean 是一种持久性对象，它代表一个存储在持久性存储器中的实体的对象视图，或是一个由现有企业应用程序实现的实体。
Session Bean 还可以再细分为 Stateful Session Bean 与 Stateless Session Bean ，这两种的 Session Bean 都可以将系统逻辑放在 method 之中执行，不同的是 Stateful Session Bean 可以记录呼叫者的状态，因此通常来说，一个使用者会有一个相对应的 Stateful Session Bean 的实体。Stateless Session Bean 虽然也是逻辑组件，但是他却不负责记录使用者状态，也就是说当使用者呼叫 Stateless Session Bean 的时候，EJB Container 并不会找寻特定的 Stateless Session Bean 的实体来执行这个 method。换言之，很可能数个使用者在执行某个 Stateless Session Bean 的 methods 时，会是同一个 Bean 的 Instance 在执行。从内存方面来看， Stateful Session Bean 与 Stateless Session Bean 比较， Stateful Session Bean 会消耗 J2EE Server 较多的内存，然而 Stateful Session Bean 的优势却在于他可以维持使用者的状态。

2、EJB 与 JAVA BEAN 的区别？

考察点：EJB

参考回答：

Java Bean 是可复用的组件，对 Java Bean 并没有严格的规范，理论上讲，任何一个 Java 类都可以是一个 Bean。但通常情况下，由于 Java Bean 是被容器所创建（如 Tomcat）的，所以 Java Bean 应具有一个无参的构造器，另外，通常 Java Bean 还要实现 Serializable 接口用于实现 Bean 的持久性。Java Bean 实际上相当于微软 COM 模型中的本地进程内 COM 组件，它是不能被跨进程访问的。EnterpriseJava Bean 相当于 DCOM，即分布式组件。它是基于 Java 的远程方法调用（RMI）技术的，所以 EJB 可以被远程访问（跨进程、跨计算机）。但 EJB 必须被布署在诸如 Webspere、WebLogic 这样的容器中，EJB 客户从不直接访问真正的 EJB 组件，而是通过其容器访问。EJB 容器是 EJB 组件的代理， EJB 组件由容器所创建和管理。客户通过容器来访问真正的 EJB 组件。

3、EJB 包括（SessionBean,EntityBean）说出他们的生命周期，及如何管理事务的？

考察点：JAVA EJB

参考回答：

SessionBean： Stateless Session Bean 的生命周期是由容器决定的，当客户机发出请求要建立一个 Bean 的实例时，EJB 容器不一定
要创建一个新的 Bean 的实例供客户机调用，而是随便找一个现有的实例提供给客户机。当客户机第一次调用一个 Stateful Session
Bean 时，容器必须立即在服务器中创建一个新的 Bean 实例，并关联到客户机上，以后此客户机调用 Stateful Session Bean 的方法
时容器会把调用分派到与此客户机相关联的 Bean 实例。
EntityBean：Entity Beans 能存活相对较长的时间，并且状态是持续的。只要数据库中的数据存在，Entity beans 就一直存活。而不
是按照应用程序或者服务进程来说的。即使 EJB 容器崩溃了，Entity beans 也是存活的。Entity Beans 生命周期能够被容器或者
Beans 自己管理。
EJB 通过以下技术管理实务：对象管理组织（OMG）的对象实务服务（OTS），Sun Microsystems 的 Transaction Service（JTS）、
Java Transaction API（JTA），开发组（X/Open）的 XA 接口。

4、EJB 的角色和三个对象是什么？

考察点：EJB

参考回答：

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

5、说说 EJB 规范规定 EJB 中禁止的操作有哪些？

考察点：EJB

参考回答：

1. 不能操作线程和线程 API(线程 API 指非线程对象的方法如 notify,wait 等)，

2. 不能操作 awt，

3. 不能实现服务器功能，

4. 不能对静态属生存取，

5. 不能使用 IO 操作直接存取文件系统，

6. 不能加载本地库.，

7. 不能将 this 作为变量和返回，

8.不能循环调用。

6、EJB 的激活机制是什么？

考察点：EJB

参考回答：

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。

7、EJB 的几种类型分别是什么

考察点：EJB

参考回答：

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。

8、EJB 需直接实现它的业务接口或 Home 接口吗，请简述理由。

考察点：EJB

参考回答：

在 EJB 中则至少要包括 10 个 class:
Bean 类，特定 App Server 的 Bean 实现类 Bean 的 remote 接口，特定 App Server 的 remote 接口实现类，特定 App Server 的 remote 接口的实现类的 stub 类和 skeleton 类。
Bean 的 home 接口，特定 App Server 的 home 接口实现类，特定 App Server 的 home 接口的实现类的 stub 类和 skeleton 类。
和 RMI 不同的是，EJB 中这 10 个 class 真正需要用户写的只有 3 个，Bean 类，remote 接口，home 接口，其它的 7 个究竟怎么生成，被打包在哪里，是否需要更多的类文件，否根据不同的 App Server 表现出较大的差异。
Weblogic：
home 接口和 remote 接口的 weblogic 的实现类的 stub 类和 skeleton 类是在 EJB 被部署到 weblogic 的时候，由 weblogic 动态生成 stub 类和 skeleton 类的字节码，所以看不到这 4 个类文件。
对于一次客户端远程调用 EJB,要经过两个远程对象的多次 RMI 循环。首先是通过 JNDI 查找 Home 接口，获得 Home 接口的实现类，这个过程其实相当复杂，首先是找到 Home 接口的 Weblogic 实现类，然后创建一个 Home 接口的 Weblogic 实现类的 stub 类的对象实例，将它序列化传送给客户端（注意 stub 类的实例是在第 1 次 RMI 循环中，由服务器动态发送给客户端的，因此不需要客户端保存 Home 接口的 Weblogic 实现类的 stub 类），最后客户端获得该 stub 类的对象实例（普通的 RMI 需要在客户端保存 stub 类，而 EJB 不需要，因为服务器会把 stub 类的对象实例发送给客户端）。
客户端拿到服务器给它的 Home 接口的 Weblogic 实现类的 stub 类对象实例以后，调用 stub 类的

create 方法， (在代码上就是 home.create()，但是后台要做很多事情),于是经过第 2 次 RMI 循环，在服务器端，Home 接口的 Weblogic 实现类的 skeleton 类收到 stub 类的调用信息后，由它再去调用 Home 接口的 Weblogic 实现类的 create 方法。
在服务端， Home 接口的 Weblogic 实现类的 create 方法再去调用 Bean 类的 Weblogic 实现类的 ejbCreate 方法，在服务端创建或者分配一个 EJB 实例，然后将这个 EJB 实例的远程接口的 Weblogic 实现类的 stub 类对象实例序列化发送给客户端。

三、Java web 编程

1、web 编程基础

①Tomcat 服务器

1、启动项目时如何实现不在链接里输入项目名就能启动?

考察点：tomcat

参考回答：

可在 taomcat 配置虚拟目录。

2、1 分钟之内只能处理 1000 个请求，你怎么实现，手撕代码?

考察点：tomcat

参考回答：

限流的几种方法：计数器，滑动窗口、漏桶法、令牌桶

3、什么时候用 assert

考察点：JAVA 调试

参考回答：

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。在实现中，assertion 就是在程序中的一条语句，它对一个 boolean 表达式进行检查，一个正确程序必须保证这个 boolean 表达式的值为 true；如果该值为 false，说明程序已经处于不正确的状态下，系统将给出警告或退出。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。

4、JAVA 应用服务器有那些?

考察点：服务器

参考回答：

BEA WebLogic Server，

IBM WebSphere Application Server，

Oracle9i Application Server

jBoss，

Tomcat

②JSP 语法，EL，内置对象

1、JSP 的内置对象及方法。

考察点：JAVA 对象

参考回答：

request 表示 HttpServletRequest 对象。它包含了有关浏览器请求的信息，并且提供了几个用于获取 cookie, header, 和 session 数据的有用的方法。
response 表示 HttpServletResponse 对象，并提供了几个用于设置送回浏览器的响应的方法（如 cookies,头信息等）
out 对象是 javax.jsp.JspWriter 的一个实例，并提供了几个方法使你能用于向浏览器回送输出结果。
pageContext 表示一个 javax.servlet.jsp.PageContext 对象。它是用于方便存取各种范围的名字空间、servlet 相关的对象的 API，并且包装了通用的 servlet 相关功能的方法。
session 表示一个请求的 javax.servlet.http.HttpSession 对象。Session 可以存贮用户的状态信息
applicaton 表示一个 javax.servle.ServletContext 对象。这有助于查找有关 servlet 引擎和 servlet 环境的信息
config 表示一个 javax.servlet.ServletConfig 对象。该对象用于存取 servlet 实例的初始化参数。
page 表示从该页面产生的一个 servlet 实例

2、JSP 和 Servlet 有哪些相同点和不同点，他们之间的联系是什么？（JSP）

考察点：JSP

参考回答：

JSP 是 Servlet 技术的扩展，本质上是 Servlet 的简易方式，更强调应用的外表表达。JSP 编译后是”类 servlet”。Servlet 和 JSP 最主要
的不同点在于，Servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 HTML 里分离开来。而 JSP 的情况是 Java 和 HTML 可以组合成
一个扩展名为.jsp 的文件。JSP 侧重于视图，Servlet 主要用于控制逻辑。

3、说一说四种会话跟踪技术

考察点：JSP

参考回答：

会话作用域 ServletsJSP 页面描述
page 否是代表与一个页面相关的对象和属性。一个页面由一个编译好的 Java servlet 类（可以带有任何的 include 指令，但是没有 include 动作）表示。这既包括 servlet 又包括被编译成 servlet 的 JSP 页面 request 是是代表与 Web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件（由于 forward 指令和 include 动作的关系）session 是是代表与用于某个 Web 客户机的一个用户体验相关的对象和属性。一个 Web 会话可以也经常会跨越多个客户机请求 application 是是代表与整个 Web 应用程序相关的对象和属性。这实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

4、讲讲 Request 对象的主要方法

考察点：Request

参考回答：

setAttribute(String name,Object)：设置名字为 name 的 request 的参数值
getAttribute(String name)：返回由 name 指定的属性值
getAttributeNames()：返回 request 对象所有属性的名字集合，结果是一个枚举的实例
getCookies()：返回客户端的所有 Cookie 对象，结果是一个 Cookie 数组
getCharacterEncoding()：返回请求中的字符编码方式
getContentLength()：返回请求的 Body 的长度
getHeader(String name)：获得 HTTP 协议定义的文件头信息
getHeaders(String name)：返回指定名字的 request Header 的所有值，结果是一个枚举的实例
getHeaderNames()：返回所以 request Header 的名字，结果是一个枚举的实例
getInputStream()：返回请求的输入流，用于获得请求中的数据
getMethod()：获得客户端向服务器端传送数据的方法
getParameter(String name)：获得客户端传送给服务器端的有 name 指定的参数值
getParameterNames()：获得客户端传送给服务器端的所有参数的名字，结果是一个枚举的实例
getParameterValues(String name)：获得有 name 指定的参数的所有值
getProtocol()：获取客户端向服务器端传送数据所依据的协议名称
getQueryString()：获得查询字符串
getRequestURI()：获取发出请求字符串的客户端地址
getRemoteAddr()：获取客户端的 IP 地址
getRemoteHost()：获取客户端的名字
getSession([Boolean create])：返回和请求相关 Session
getServerName()：获取服务器的名字
getServletPath()：获取客户端所请求的脚本文件的路径
getServerPort()：获取服务器的端口号
removeAttribute(String name)：删除请求中的一个属性

5、说说 weblogic 中一个 Domain 的缺省目录结构?比如要将一个简单的 helloWorld.jsp 放入何目录下,然后在浏览器上就可打入主机?

考察点：目录结构

参考回答：

端口号//helloword.jsp 就可以看到运行结果了？又比如这其中用到了一个自己写的 javaBean 该如何办?

Domain 目录服务器目录 applications，将应用目录放在此目录下将可以作为应用访问，如果是 Web 应用，应用目录需要满足 Web 应用
目录要求，jsp 文件可以直接放在应用目录中，Javabean 需要放在应用目录的 WEB-INF 目录的 classes 目录中，设置服务器的缺省应用
将可以实现在浏览器上无需输入应用名。

6、jsp 有哪些动作?作用分别是什么?

考察点：JSP

参考回答：

JSP 共有以下 6 种基本动作 jsp:include：在页面被请求的时候引入一个文件。
jsp:useBean：寻找或者实例化一个 JavaBean。
jsp:setProperty：设置 JavaBean 的属性。 jsp:getProperty：输出某个 JavaBean 的属性。
jsp:forward：把请求转到一个新的页面。

jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记。

7、请谈谈 JSP 有哪些内置对象？作用分别是什么？

考察点：JSP

参考回答：

JSP 有 9 个内置对象：
- request：封装客户端的请求，其中包含来自 GET 或 POST 请求的参数；
- response：封装服务器对客户端的响应；
- pageContext：通过该对象可以获取其他对象；
- session：封装用户会话的对象；
- application：封装服务器运行环境的对象；
- out：输出服务器响应的输出流对象；
- config：Web 应用的配置对象；
- page：JSP 页面本身（相当于 Java 程序中的 this）；
- exception：封装页面抛出异常的对象。

如果用 Servlet 来生成网页中的动态内容无疑是非常繁琐的工作，另一方面，所有的文本和 HTML 标签都是硬编码，即使做出微小的修改，都需要进行重新编译。JSP 解决了 Servlet 的这些问题，它是 Servlet 很好的补充，可以专门用作为用户呈现视图（View），而 Servlet 作为控制器（Controller）专门负责处理用户请求并转发或重定向到某个页面。基于 Java 的 Web 开发很多都同时使用了 Servlet 和 JSP。JSP 页面其实是一个 Servlet，能够运行 Servlet 的服务器（Servlet 容器)通常也是 JSP 容器,可以提供 JSP 页面的运行环境,Tomcat 就是一个 Servlet/JSP 容器。第一次请求一个 JSP 页面时，Servlet/JSP 容器首先将 JSP 页面转换成一个 JSP 页面的实现类,这是一个实现了 JspPage 接口或其子接口 HttpJspPage 的 Java 类。JspPage 接口是 Servlet 的子接口，因此每个 JSP 页面都是一个 Servlet。转换成功后，容器会编译 Servlet 类，之后容器加载和实例化 Java 字节码，并执行它通常对 Servlet 所做的生命周期操作。对同一个 JSP 页面的后续请求，容器会查看这个 JSP 页面是否被修改过，如果修改过就会重新转换并重新编译并执行。如果没有则执行内存中已经存在的 Servlet 实例。

8、说一下表达式语言（EL）的隐式对象及其作用

考察点：EL

参考回答：

EL 的隐式对象包括：pageContext、initParam（访问上下文参数）、param（访问请求参数）、paramValues、header（访问请求头）、headerValues、cookie（访问 cookie）、applicationScope（访问 application 作用域）、sessionScope（访问 session 作用域）、requestScope（访问 request 作用域）、pageScope（访问 page 作用域）。

9、JSP 中的静态包含和动态包含有什么区别？

考察点：JSP

参考回答：

静态包含是通过 JSP 的 include 指令包含页面，动态包含是通过 JSP 标准动作<jsp:forward>包含页面。静态包含是编译时包含，如果包含的页面不存在则会产生编译错误，而且两个页面的"contentType"属性应保持一致，因为两个页面会合二为一，只产生一个 class 文件，因此被包含页面发生的变动再包含它的页面更新前不会得到更新。动态包含是运行时包含，可以向被包含的页面传递参数，包含页面和被包含页面是独立的，会编译出两个 class 文件，如果被包含的页面不存在，不会产生编译错误，也不影响页面其他部分的执行。

例如：<%-- 静态包含 --%>

```
<%@ include file="..." %>
<%-- 动态包含 --%>
<jsp:include page="...">
      <jsp:param name="..." value="..." />
</jsp:include>
```

③Listener 和 Filter

1、过滤器有哪些作用和用法？

考察点：过滤器

参考回答：

Java Web 开发中的过滤器（filter）是从 Servlet 2.3 规范开始增加的功能，并在 Servlet 2.4 规范中得到增强。对 Web 应用来说，过滤器是一个驻留在服务器端的 Web 组件，它可以截取客户端和服务器之间的请求与响应信息，并对这些信息进行过滤。当 Web 容器接受到一个对资源的请求时，它将判断是否有过滤器与这个资源相关联。如果有，那么容器将把请求交给过滤器进行处理。在过滤器中，你可以改变请求的内容，或者重新设置请求的报头信息，然后再将请求发送给目标资源。当目标资源对请求作出响应时候，容器同样会将响应先转发给过滤器，在过滤器中你可以对响应的内容进行转换，然后再将响应发送到客户端。

常见的过滤器用途主要包括：对用户请求进行统一认证、对用户的访问请求进行记录和审核、对用户发送的数据进行过滤或替换、转换图象格式、对响应内容进行压缩以减少传输量、对请求或响应进行加解密处理、触发资源访问事件、对 XML 的输出应用 XSLT 等。
和过滤器相关的接口主要有：Filter、FilterConfig 和 FilterChain。

2、请谈谈你对 Javaweb 开发中的监听器的理解？

考察点：监听器

参考回答：

Java Web 开发中的监听器（listener）就是 application、session、request 三个对象创建、销毁或者往其中添加修改删除属性时自动执行代码的功能组件，如下所示：
①ServletContextListener：对 Servlet 上下文的创建和销毁进行监听。
②ServletContextAttributeListener：监听 Servlet 上下文属性的添加、删除和替换。
③HttpSessionListener：对 Session 的创建和销毁进行监听。

session 的销毁有两种情况：1). session 超时（可以在 web.xml 中通过<session-config>/<session-timeout>标签配置超时时间）；2). 通过调用 session 对象的 invalidate()方法使 session 失效。
④HttpSessionAttributeListener：对 Session 对象中属性的添加、删除和替换进行监听。
⑤ServletRequestListener：对请求对象的初始化和销毁进行监听。
⑥ServletRequestAttributeListener：对请求对象属性的添加、删除和替换进行监听。

3、说说 web.xml 文件中可以配置哪些内容？

考察点：xml 文件

参考回答：

web.xml 用于配置 Web 应用的相关信息,如:监听器(listener)、过滤器(filter)、Servlet、相关参数、会话超时时间、安全验证方式、错误页面等，下面是一些开发中常见的配置：

①配置 Spring 上下文加载监听器加载 Spring 配置文件并创建 IoC 容器：

```
<context-param>
       <param-name>contextConfigLocation</param-name>
      <param-value>classpath:applicationContext.xml</param-value>
 </context-param>

 <listener>
       <listener-class>
           org.springframework.web.context.ContextLoaderListener
       </listener-class>
 </listener>
```

②配置 Spring 的 OpenSessionInView 过滤器来解决延迟加载和 Hibernate 会话关闭的矛盾：

```
<filter>
         <filter-name>openSessionInView</filter-name>
         <filter-class>
               org.springframework.orm.hibernate3.support.OpenSessionInViewFilter
         </filter-class>
 </filter>

 <filter-mapping>
         <filter-name>openSessionInView</filter-name>
         <url-pattern>/*</url-pattern>
 </filter-mapping>
```

③配置会话超时时间为 10 分钟：

```
  <session-config>
          <session-timeout>10</session-timeout>
   </session-config>
```

④配置 404 和 Exception 的错误页面：

```
  <error-page>
          <error-code>404</error-code>
          <location>/error.jsp</location>
   </error-page>

   <error-page>
          <exception-type>java.lang.Exception</exception-type>
          <location>/error.jsp</location>
   </error-page>
```

⑤配置安全认证方式：

```
<security-constraint>
          <web-resource-collection>
                  <web-resource-name>ProtectedArea</web-resource-name>
                  <url-pattern>/admin/*</url-pattern>
                  <http-method>GET</http-method>
                  <http-method>POST</http-method>
          </web-resource-collection>
          <auth-constraint>
                  <role-name>admin</role-name>
          </auth-constraint>
   </security-constraint>

   <login-config>
          <auth-method>BASIC</auth-method>
   </login-config>

   <security-role>
          <role-name>admin</role-name>
   </security-role>
```

2、Web 编程进阶

①Servlet、标签的作用

1、forward 与 redirect 区别，说一下你知道的状态码，redirect 的状态码是多少？

考察点：Servlet

参考回答：

1.从地址栏显示来说

forward 是服务器请求资源,服务器直接访问目标地址的 URL,把那个 URL 的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址.

redirect 是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的 URL.

2.从数据共享来说

forward:转发页面和转发到的页面可以共享 request 里面的数据.

redirect:不能共享数据.

3.从运用地方来说

forward:一般用于用户登陆的时候,根据角色转发到相应的模块.

redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等.

4.从效率来说

forward:高.

redirect:低.

redirect 的状态码是 302

2、servlet 生命周期，是否单例，为什么是单例。

考察点：servlet

参考回答：

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

Servlet 通过调用 init () 方法进行初始化。

Servlet 调用 service() 方法来处理客户端的请求。

Servlet 通过调用 destroy() 方法终止（结束）。

最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

Servlet 单实例，减少了产生 servlet 的开销；

3、说出 Servlet 的生命周期，并说出 Servlet 和 CGI 的区别。

考察点：servlet

参考回答：

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，doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。
与 cgi 的区别在于 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，并且其实例一般不会销毁，而 CGI 对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于 servlet。

4、Servlet 执行时一般实现哪几个方法？

考察点：servlet

参考回答：

```
public void init(ServletConfig config)
public ServletConfig getServletConfig()
public String getServletInfo()
public void service(ServletRequest request,ServletResponse response)

public void destroy()
```

init ()方法在 servlet 的生命周期中仅执行一次，在服务器装载 servlet 时执行。缺省的 init()方法通常是符合要求的，不过也可以根据需要进行 override，比如管理服务器端资源，一次性装入 GIF 图像，初始化数据库连接等，缺省的 inti()方法设置了 servlet 的初始化参数，并用它的 ServeltConfig 对象参数来启动配置，所以覆盖 init()方法时，应调用 super.init()以确保仍然执行这些任务。service ()方法是 servlet 的核心，在调用 service()方法之前，应确保已完成 init()方法。对于 HttpServlet，每当客户请求一个 HttpServlet 对象，该对象的 service()方法就要被调用，HttpServlet 缺省的 service()方法的服务功能就是调用与 HTTP 请求的方法相应的 do 功能，doPost()和 doGet()，所以对于 HttpServlet，一般都是重写 doPost()和 doGet()方法。destroy()方法在 servlet 的生命周期中也仅执行一次，即在服务器停止卸载 servlet 时执行，把 servlet 作为服务器进程的一部分关闭。缺省的 destroy()方法通常是符合要求的，但也可以 override，比如在卸载 servlet 时将统计数字保存在文件中，或是关闭数据库连接 getServletConfig()方法返回一个 servletConfig 对象，该对象用来返回初始化参 servletContext。servletContext 接口提供有关 servlet 的环境信息。getServletInfo()方法提供有关 servlet 的信息，如作者，版本，版权。

5、阐述一下阐述 Servlet 和 CGI 的区别?

考察点：servlet

参考回答：

Servlet 与 CGI 的区别在于 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，并且其实例一般不会销毁，而 CGI 对每个请求都产生新的进程，服务完成后就销毁，所以效率上低于 Servlet。

6、说说 Servlet 接口中有哪些方法？

考察点：Servlet 接口

参考回答：

Servlet 接口定义了 5 个方法，其中前三个方法与 Servlet 生命周期相关：
- void init(ServletConfig config) throws ServletException
- void service(ServletRequest req, ServletResponse resp) throws ServletException, java.io.IOException
- void destory()
- java.lang.String getServletInfo()
- ServletConfig getServletConfig()

Web 容器加载 Servlet 并将其实例化后，Servlet 生命周期开始，容器运行其 init()方法进行 Servlet 的初始化；请求到达时调用 Servlet 的 service()方法，service()方法会根据需要调用与请求对应的 doGet 或 doPost 等方法；当服务器关闭或项目被卸载时服务

器会将 Servlet 实例销毁，此时会调用 Servlet 的 destroy()方法。

7、Servlet 3 中的异步处理指的是什么？

考察点：servlet

参考回答：

在 Servlet 3 中引入了一项新的技术可以让 Servlet 异步处理请求。有人可能会质疑，既然都有多线程了，还需要异步处理请求吗？答案是肯定的，因为如果一个任务处理时间相当长，那么 Servlet 或 Filter 会一直占用着请求处理线程直到任务结束，随着并发用户的增加，容器将会遭遇线程超出的风险，这这种情况下很多的请求将会被堆积起来而后续的请求可能会遭遇拒绝服务，直到有资源可以处理请求为止。异步特性可以帮助应用节省容器中的线程，特别适合执行时间长而且用户需要得到结果的任务，如果用户不需要得到结果则直接将一个 Runnable 对象交给 Executor 并立即返回即可。

例如：

```
import java.io.IOException;
import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(urlPatterns = {"/async"}, asyncSupported = true)
public class AsyncServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	@Override
	public void doGet(HttpServletRequest req, HttpServletResponse resp)
			throws ServletException, IOException {
		// 开启 Tomcat 异步 Servlet 支持
		req.setAttribute("org.apache.catalina.ASYNC_SUPPORTED", true);

		final AsyncContext ctx = req.startAsync();   // 启动异步处理的上下文
		// ctx.setTimeout(30000);
		ctx.start(new Runnable() {

			@Override
			public void run() {
				// 在此处添加异步处理的代码
```

```
                                ctx.complete();
                        }
                });
        }
}
```

8、如何在基于 Java 的 Web 项目中实现文件上传和下载？

考察点：文件传输

参考回答：

在 Sevlet 3 以前，Servlet API 中没有支持上传功能的 API，因此要实现上传功能需要引入第三方工具从 POST 请求中获得上传的附件或者通过自行处理输入流来获得上传的文件，我们推荐使用 Apache 的 commons-fileupload。
从 Servlet 3 开始，文件上传变得简单许多。

例如：

```
    package com.jackfrued.servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;
@WebServlet("/UploadServlet")
@MultipartConfig
public class UploadServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

        protected void doPost(HttpServletRequest request,
                        HttpServletResponse response) throws ServletException,
IOException {
                // 可以用 request.getPart()方法获得名为 photo 的上传附件
                // 也可以用 request.getParts()获得所有上传附件（多文件上传）
                // 然后通过循环分别处理每一个上传的文件
                Part part = request.getPart("photo");
                if (part != null && part.getSubmittedFileName().length() > 0) {
                        // 用 ServletContext 对象的 getRealPath()方法获得上传文件夹的
绝对路径
                        String savePath =
request.getServletContext().getRealPath("/upload");
                        // Servlet 3.1 规范中可以用 Part 对象的 getSubmittedFileName()
方法获得上传的文件名
                        // 更好的做法是为上传的文件进行重命名（避免同名文件的相互覆盖）
                        part.write(savePath + "/" + part.getSubmittedFileName());
                        request.setAttribute("hint", "Upload Successfully!");
                } else {
                        request.setAttribute("hint", "Upload failed!");
```

```
                }
                // 跳转回到上传页面
                request.getRequestDispatcher("index.jsp").forward(request, response);
        }
}
```

9、服务器收到用户提交的表单数据，到底是调用 Servlet 的 doGet()还是 doPost()方法？

考察点：servlet

参考回答：

HTML 的<form>元素有一个 method 属性，用来指定提交表单的方式，其值可以是 get 或 post。我们自定义的 Servlet 一般情况下会重写 doGet()或 doPost()两个方法之一或全部，如果是 GET 请求就调用 doGet()方法，如果是 POST 请求就调用 doPost()方法，那为什么为什么这样呢？我们自定义的 Servlet 通常继承自 HttpServlet，HttpServlet 继承自 GenericServlet 并重写了其中的 service()方法，这个方法是 Servlet 接口中定义的。HttpServlet 重写的 service()方法会先获取用户请求的方法，然后根据请求方法调用 doGet()、doPost()、doPut()、doDelete()等方法，如果在自定义 Servlet 中重写了这些方法，那么显然会调用重写过的（自定义的）方法，这显然是对模板方法模式的应用（如果不理解，请参考阎宏博士的《Java 与模式》一书的第 37 章）。当然，自定义 Servlet 中也可以直接重写 service()方法，那么不管是哪种方式的请求，都可以通过自己的代码进行处理，这对于不区分请求方法的场景比较合适。

10、Servlet 中如何获取用户提交的查询参数或表单数据？

考察点：servlet

参考回答：

可以通过请求对象（HttpServletRequest）的 getParameter()方法通过参数名获得参数值。如果有包含多个值的参数（例如复选框），可以通过请求对象的 getParameterValues()方法获得。当然也可以通过请求对象的 getParameterMap()获得一个参数名和参数值的映射（Map）。

11、Servlet 中如何获取用户配置的初始化参数以及服务器上下文参数？

考察点：初始化

参考回答：

可以通过重写 Servlet 接口的 init(ServletConfig)方法并通过 ServletConfig 对象的 getInitParameter()方法来获取 Servlet 的初始化参数。可以通过 ServletConfig 对象的 getServletContext()方法获取 ServletContext 对象，并通过该对象的 getInitParameter()方法来获取服务器上下文参数。当然，ServletContext 对象也在处理用户请求的方法（如 doGet()方法）中通过请求对象的 getServletContext()方法来获得。

②redis

1、讲一下 redis 的主从复制怎么做的？

考察点：

参考回答：

第一阶段：与 master 建立连接

第二阶段：向 master 发起同步请求（SYNC）

第三阶段：接受 master 发来的 RDB 数据

第四阶段：载入 RDB 文件

2、redis 为什么读写速率快性能好？

考察点：redis

参考回答：

Redis 是纯内存数据库，相对于读写磁盘，读写内存的速度就不是几倍几十倍了，一般，hash 查找可以达到每秒百万次的数量级。

多路复用 IO，“多路”指的是多个网络连接，“复用”指的是复用同一个线程。采用多路 I/O 复用技术可以让单个线程高效的处理多个连接请求（尽量减少网络 IO 的时间消耗）。可以直接理解为：单线程的原子操作，避免上下文切换的时间和性能消耗；加上对内存中数据的处理速度，很自然的提高 redis 的吞吐量。

3、redis 为什么是单线程？

考察点：

参考回答：

因为 CPU 不是 Redis 的瓶颈。Redis 的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且 CPU 不会成为瓶颈，那就顺理成章地采用单线程的方案了。缺点：服务器其他核闲置。

4、缓存的优点？

考察：redis

参考回答：

优点：1、减少了对数据库的读操作，数据库的压力降低　2、加快了响应速度

5、aof，rdb，优点，区别？

考察点：redis

参考回答：

RDB 持久化可以在指定的时间间隔内生成数据集的时间点快照（point-in-time snapshot）。

AOF 持久化记录服务器执行的所有写操作命令，并在服务器启动时，通过重新执行这些命令来还原数据集。 AOF 文件中的命令全部以 Redis 协议的格式来保存，新命令会被追加到文件的末尾。 Redis 还可以在后台对 AOF 文件进行重写（rewrite），使得 AOF 文件的体积不会超出保存数据集状态所需的实际大小。Redis 还可以同时使用 AOF 持久化和 RDB 持久化。 在这种情况下， 当 Redis 重启时， 它会优先使用 AOF 文件来还原数据集， 因为 AOF 文件保存的数据集通常比 RDB 文件所保存的数据集更完整。你甚至可以关闭持久化功能，让数据只在服务器运行时存在。

RDB 的优点:

RDB 是一个非常紧凑（compact）的文件，它保存了 Redis 在某个时间点上的数据集。 这种文件非常适合用于进行备份： 比如说，你可以在最近的 24 小时内，每小时备份一次 RDB 文件，并且在每个月的每一天，也备份一个 RDB 文件。 这样的话，即使遇上问题，也可以随时将数据集还原到不同的版本。RDB 非常适用于灾难恢复（disaster recovery）：它只有一个文件，并且内容都非常紧凑，可以（在加密后）将它传送到别的数据中心，或者亚马逊 S3 中。RDB 可以最大化 Redis 的性能：父进程在保存 RDB 文件时唯一要做的就是 fork 出一个子进程，然后这个子进程就会处理接下来的所有保存工作，父进程无须执行任何磁盘 I/O 操作。RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快。

RDB 的缺点:

如果你需要尽量避免在服务器故障时丢失数据，那么 RDB 不适合你。 虽然 Redis 允许你设置不同的保存点（save point）来控制保存 RDB 文件的频率， 但是， 因为 RDB 文件需要保存整个数据集的状态， 所以它并不是一个轻松的操作。 因此你可能会至少 5 分钟才保存一次 RDB 文件。在这种情况下，一旦发生故障停机，你就可能会丢失好几分钟的数据。每次保存 RDB 的时候，Redis 都要 fork() 出一个子进程，并由子进程来进行实际的持久化工作。 在数据集比较庞大时， fork() 可能会非常耗时，造成服务器在某某毫秒内停止处理客户端； 如果数据集非常巨大，并且 CPU 时间非常紧张的话，那么这种停止时间甚至可能会长达整整一秒。 虽然 AOF 重写也需要进行 fork() ，但无论 AOF 重写的执行间隔有多长，数据的耐久性都不会有任何损失。

AOF 的优点:

使用 AOF 持久化会让 Redis 变得非常耐久(much more durable)：你可以设置不同的 fsync 策略，比如无 fsync ，每秒钟一次 fsync ，或者每次执行写入命令时 fsync 。 AOF 的默认策略为每秒钟 fsync 一次，在这种配置下，Redis 仍然可以保持良好的性能，并且就算发生故障停机，也最多只会丢失一秒钟的数据（ fsync 会在后台线程执行，所以主线程可以继续努力地处理命令请求）。AOF 文件是一个只进行追加操作的日志文件（append only log）， 因此对 AOF 文件的写入不需要进行 seek ， 即使日志因为某些原因而包含了未写入完整的命令（比如写入时磁盘已满，写入中途停机，等等）， redis-check-aof 工具也可以轻易地修复这种问题。

Redis 可以在 AOF 文件体积变得过大时，自动地在后台对 AOF 进行重写：重写后的新 AOF 文件包含了恢复当前数据集所需的最小命令集合。 整个重写操作是绝对安全的，因为 Redis 在创建新 AOF 文件的过程中，会继续将命令追加到现有的 AOF 文件里面，即使重写过程中发生停机，现有的 AOF 文件也不会丢失。 而一旦新 AOF 文件创建完毕，Redis 就会从旧 AOF 文件切换到新 AOF 文件，并开始对新 AOF 文件进行追加操作。AOF 文件有序地保存了对数据库执行的所有写入操作， 这些写入操作以 Redis 协议的格式保存， 因此 AOF 文件的内容非常容易被人

读懂， 对文件进行分析（parse）也很轻松。 导出（export） AOF 文件也非常简单： 举个例子， 如果你不小心执行了 FLUSHALL 命令， 但只要 AOF 文件未被重写， 那么只要停止服务器， 移除 AOF 文件末尾的 FLUSHALL 命令， 并重启 Redis ， 就可以将数据集恢复到 FLUSHALL 执行之前的状态。

AOF 的缺点:

对于相同的数据集来说,AOF 文件的体积通常要大于 RDB 文件的体积。根据所使用的 fsync 策略，AOF 的速度可能会慢于 RDB 。 在一般情况下， 每秒 fsync 的性能依然非常高， 而关闭 fsync 可以让 AOF 的速度和 RDB 一样快， 即使在高负荷之下也是如此。 不过在处理巨大的写入载入时，RDB 可以提供更有保证的最大延迟时间（latency）。AOF 在过去曾经发生过这样的 bug ： 因为个别命令的原因，导致 AOF 文件在重新载入时，无法将数据集恢复成保存时的原样。 （举个例子，阻塞命令 BRPOPLPUSH 就曾经引起过这样的 bug 。） 测试套件里为这种情况添加了测试： 它们会自动生成随机的、复杂的数据集， 并通过重新载入这些数据来确保一切正常。 虽然这种 bug 在 AOF 文件中并不常见， 但是对比来说， RDB 几乎是不可能出现这种 bug 的。

6、redis 的 List 能用做什么场景？

考察点：redis

参考回答：

Redis 中 list 的数据结构实现是双向链表,所以可以非常便捷的应用于消息队列(生产者 / 消费者模型）。消息的生产者只需要通过 lpush 将消息放入 list，消费者便可以通过 rpop 取出该消息，并且可以保证消息的有序性。如果需要实现带有优先级的消息队列也可以选择 sorted set。而 pub/sub 功能也可以用作发布者 / 订阅者模型的消息。

③MVC 与 DAO

1、说说 MVC 的各个部分都有那些技术来实现?如何实现?

考察点：MVC

参考回答：

MVC 是 Model—View—Controller 的简写。”Model” 代表的是应用的业务逻辑（通过 JavaBean，EJB 组件实现）， “View” 是应用的表示面，用于与用户的交互（由 JSP 页面产生），”Controller” 是提供应用的处理过程控制（一般是一个 Servlet），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。model 层实现系统中的业务逻辑，view 层用于与用户的交互，controller 层是 model 与 view 之间沟通的桥梁，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。

2、什么是 DAO 模式?

考察点：对象

参考回答：

DAO（Data Access 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，在不暴露底层持久化方案实现细节的前提下提供了各种数据访问操作。在实际的开发中，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。

3、请问 Java Web 开发的 Model 1 和 Model 2 分别指的是什么？

考察点：Model

参考回答：

Model 1 是以页面为中心的 Java Web 开发，使用 JSP+JavaBean 技术将页面显示逻辑和业务逻辑处理分开，JSP 实现页面显示，JavaBean 对象用来保存数据和实现业务逻辑。Model 2 是基于 MVC（模型-视图-控制器，Model-View-Controller）架构模式的开发模型，实现了模型和视图的彻底分离，利于团队开发和代码复用。

④JSTL、DisplayTag 等常见标签库的用法

1、你的项目中使用过哪些 JSTL 标签？

考察点：JSTL

参考回答：

项目中主要使用了 JSTL 的核心标签库，包括<c:if>、<c:choose>、<c: when>、<c: otherwise>、<c:forEach>等，主要用于构造循环和分支结构以控制显示逻辑。

虽然 JSTL 标签库提供了 core、sql、fmt、xml 等标签库，但是实际开发中建议只使用核心标签库（core），而且最好只使用分支和循环标签并辅以表达式语言（EL），这样才能真正做到数据显示和业务逻辑的分离，这才是最佳实践。

2、使用标签库有什么好处？如何自定义 JSP 标签？（JSP 标签）

考察点：JSP 标签

参考回答：

使用标签库的好处包括以下几个方面：
- 分离 JSP 页面的内容和逻辑，简化了 Web 开发；
- 开发者可以创建自定义标签来封装业务逻辑和显示逻辑；
- 标签具有很好的可移植性、可维护性和可重用性；
- 避免了对 Scriptlet（小脚本）的使用（很多公司的项目开发都不允许在 JSP 中书写小脚本）

编写一个 Java 类实现实现 Tag/BodyTag/IterationTag 接口（开发中通常不直接实现这些接口而是继承 TagSupport/BodyTagSupport/SimpleTagSupport 类，这是对缺省适配模式的应用），重写 doStartTag()、doEndTag()等方法，定义标签要完成的功能：

- 编写扩展名为 tld 的标签描述文件对自定义标签进行部署，tld 文件通常放在 WEB-INF 文件夹下或其子目录中
- 在 JSP 页面中使用 taglib 指令引用该标签库

3、Web 编程原理

①HTTP 协议

1、get 和 post 区别？

考察点：HTTP 请求

参考回答：

（1）在客户端， Get 方式在通过 URL 提交数据，数据 在 URL 中可以看到；POST 方式，数据放置在 HTML HEADER 内提交。

（2）GET 方式提交的数据最多只能有 1024 字节，而 POST 则没有此限制。

（3）安全性问题。正如在（ 1 ）中提到，使用 Get 的时候，参数会显示在地址栏上，而 Post 不会。所以，如果这些数据是中文数据而且是非敏感数据，那么使用 get ；如果用户输入的数据不是中文字符而且包含敏感数据，那么还是使用 post 为好。

安全的和幂等的。所谓安全的意味着该操作用于获取信息而非修改信息。幂等的意味着对同一 URL 的多个请求应该返回同样的结果。完整的定义并不像看起来那样严格。换句话说，GET 请求一般不应产生副作用。从根本上讲，其目标是当用户打开一个链接时，她可以确信从自身的角度来看没有改变资源。比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。反之亦然。POST 请求就不那么轻松了。 POST 表示可能改变服务器上的资源的请求。仍然以新闻站点为例,读者对文章的注解应该通过 POST 请求实现,因为在注解提交之后站点已经不同了(比方说文章下面出现一条注解）。

2、请谈谈转发和重定向的区别？

考察点：重定向

参考回答：

forward 是容器中控制权的转向，是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。redirect 就是服务器端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，因此从浏览器的地址栏中可以看到跳转后的链接地址，很明显 redirect 无法访问到服务器保护起来资源，但是可以从一个网站 redirect 到其他网站。forward 更加高效，所以在满足需要时尽量使用 forward（通过调用 RequestDispatcher 对象的 forward()方法，该对象可以通过 ServletRequest 对象的 getRequestDispatcher()方法获得），并且这样也有助于隐藏实际的链接；在有些情况下，比如需要访问一个其它服务器上的资源，则必须使用重定向（通过 HttpServletResponse 对象调用其 sendRedirect()方法实现）。

3、说说你对 get 和 post 请求，并且说说它们之间的区别？

考察点：HTTP 请求

参考回答：

①get 请求用来从服务器上获得资源，而 post 是用来向服务器提交数据；
②get 将表单中数据按照 name=value 的形式，添加到 action 所指向的 URL 后面，并且两者使用"?"连接，而各个变量之间使用"&"连接；post 是将表单中的数据放在 HTTP 协议的请求头或消息体中，传递到 action 所指向 URL；
③get 传输的数据要受到 URL 长度限制（1024 字节）；而 post 可以传输大量的数据，上传文件通常要使用 post 方式；
④使用 get 时参数会显示在地址栏上，如果这些数据不是敏感数据，那么可以使用 get；对于敏感数据还是应用使用 post；
⑤get 使用 MIME 类型 application/x-www-form-urlencoded 的 URL 编码（也叫百分号编码）文本的格式传递参数，保证被传送的参数由遵循规范的文本组成，例如一个空格的编码是"%20"。

②请求/相应架构原理

1、cookie 和 session 的区别？

考察点：web 访问

参考回答：

1、cookie 数据存放在客户的浏览器上，session 数据放在服务器上。

2、cookie 不是很安全，别人可以分析存放在本地的 COOKIE 并进行 COOKIE 欺骗

考虑到安全应当使用 session。

3、session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用 COOKIE。

4、单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

2、forward 和 redirect 的区别

考察点：资源请求方式

参考回答：

forward 是服务器请求资源，服务器直接访问目标地址的 URL，把那个 URL 的响应内容读取过来，然后把这些内容再发给浏览器，浏览器根本不知道服务器发送的内容是从哪儿来的，所以它的地址栏中还是原来的地址。
redirect 就是服务端根据逻辑，发送一个状态码，告诉浏览器重新去请求那个地址，一般来说浏览器会用刚才请求的所有参数重新请求，所以 session，request 参数都可以获取。

3、BS 与 CS 的联系与区别。

考察点：客户端/服务器模式

参考回答：

C/S 是 Client/Server 的缩写。服务器通常采用高性能的 PC、工作站或小型机，并采用大型数据库系统，如 Oracle、Sybase、Informix
或 SQL Server。客户端需要安装专用的客户端软件。
B/S 是 Brower/Server 的缩写,客户机上只要安装一个浏览器(Browser),如 Netscape Navigator 或 Internet Explorer，服务器安
装 Oracle、Sybase、Informix 或 SQL Server 等数据库。在这种结构下，用户界面完全通过 WWW 浏览器实现，一部分事务逻辑在前端实
现，但是主要事务逻辑在服务器端实现。浏览器通过 Web Server 同数据库进行数据交互。
C/S 与 B/S 区别：

硬件环境不同:
C/S 一般建立在专用的网络上，小范围里的网络环境，局域网之间再通过专门服务器提供连接和数据交换服务.
B/S 建立在广域网之上的，不必是专门的网络硬件环境,例与电话上网，租用设备．信息自己管理．有比 C/S 更强的适应范围，一
般只要有操作系统和浏览器就行
２．对安全要求不同
C/S 一般面向相对固定的用户群，对信息安全的控制能力很强．一般高度机密的信息系统采用 C/S 结构适宜．可以通过 B/S 发布
部分可公开信息.
B/S 建立在广域网之上，对安全的控制能力相对弱，可能面向不可知的用户。
３．对程序架构不同
C/S 程序可以更加注重流程，可以对权限多层次校验，对系统运行速度可以较少考虑.
B/S 对安全以及访问速度的多重的考虑，建立在需要更加优化的基础之上．比 C/S 有更高的要求 B/S 结构的程序架构是发展的趋
势，从 MS 的.Net 系列的 BizTalk 2000 Exchange 2000 等，全面支持网络的构件搭建的系统．SUN 和 IBM 推的 JavaBean 构件技术等,使
B/S 更加成熟.
４．软件重用不同
C/S 程序可以不可避免的整体性考虑，构件的重用性不如在 B/S 要求下的构件的重用性好.
B/S 对的多重结构,要求构件相对独立的功能．能够相对较好的重用.就入买来的餐桌可以再利用,而不是做在墙上的石头桌子
５．系统维护不同
C/S 程序由于整体性，必须整体考察，处理出现的问题以及系统升级．升级难．可能是再做一个全新的系统
B/S 构件组成,方面构件个别的更换,实现系统的无缝升级．系统维护开销减到最小.用户从网上自己下载安装就可以实现升级.
６．处理问题不同
C/S 程序可以处理用户面固定，并且在相同区域，安全要求高需求，与操作系统相关．应该都是相同的系统
B/S 建立在广域网上，面向不同的用户群，分散地域，这是 C/S 无法作到的．与操作系统平台关系最小.
７．用户接口不同
C/S 多是建立的 Window 平台上,表现方法有限,对程序员普遍要求较高
B/S 建立在浏览器上，有更加丰富和生动的表现方式与用户交流．并且大部分难度减低,减低开发成本.
８．信息流不同
C/S 程序一般是典型的中央集权的机械式处理，交互性相对低
B/S 信息流向可变化，B-B B-C B-G 等信息、流向的变化，更像交易中心。

4、如何设置请求的编码以及响应内容的类型？

考察点：请求类型

参考回答：

通过请求对象（ServletRequest）的 setCharacterEncoding(String)方法可以设置请求的编码，其实要彻底解决乱码问题就应该让页面、服务器、请求和响应、Java 程序都使用统一的编码，最好的选择当然是 UTF-8；通过响应对象（ServletResponse）的 setContentType(String)方法可以设置响应内容的类型，当然也可以通过 HttpServletResponsed 对象的 setHeader(String, String)方法来设置。

5、什么是 Web Service（Web 服务）？

考察点：web service

参考回答：

从表面上看，Web Service 就是一个应用程序，它向外界暴露出一个能够通过 Web 进行调用的 API。这就是说，你能够用编程的方法透明的调用这个应用程序，不需要了解它的任何细节，跟你使用的编程语言也没有关系。例如可以创建一个提供天气预报的 Web Service，那么无论你用哪种编程语言开发的应用都可以通过调用它的 API 并传入城市信息来获得该城市的天气预报。之所以称之为 Web Service，是因为它基于 HTTP 协议传输数据，这使得运行在不同机器上的不同应用无须借助附加的、专门的第三方软件或硬件，就可相互交换数据或集成。

SOA（Service-Oriented Architecture，面向服务的架构），SOA 是一种思想，它将应用程序的不同功能单元通过中立的契约联系起来，独立于硬件平台、操作系统和编程语言，使得各种形式的功能单元能够更好的集成。显然，Web Service 是 SOA 的一种较好的解决方案，它更多的是一种标准，而不是一种具体的技术。

6、谈谈 Session 的 save()、update()、merge()、lock()、saveOrUpdate()和 persist()方法分别是做什么的？有什么区别？

考察点：session

参考回答：

Hibernate 的对象有三种状态：瞬时态（transient）、持久态（persistent）和游离态（detached），如第 135 题中的图所示。瞬时态的实例可以通过调用 save()、persist()或者 saveOrUpdate()方法变成持久态；游离态的实例可以通过调用 update()、saveOrUpdate()、lock()或者 replicate()变成持久态。save()和 persist()将会引发 SQL 的 INSERT 语句，而 update()或 merge()会引发 UPDATE 语句。save()和 update()的区别在于一个是将瞬时态对象变成持久态，一个是将游离态对象变为持久态。merge()方法可以完成 save()和 update()方法的功能，它的意图是将新的状态合并到已有的持久化对象上或创建新的持久化对象。对于 persist()方法，按照官方文档的说明：① persist()方法把一个瞬时态的实例持久化，但是并不保证标识符被立刻填入到持久化实例中，标识符的填入可能被推迟到 flush 的时间；② persist()方法保证当它在一个事务外部被调用的时候并不触发一个 INSERT 语句，当需要封装一个长会话流程的时候，persist()方法是很有必要的；③ save()方法不保证第②条，它要返回标识符，所以它会立即执行 INSERT 语句，不管是在事务内部还是外部。至于 lock()方法和 update()方法的区别，update()方法是把一个已经更改过的脱管状态的对象变成持久状态；lock()方法是把一个没有更改过的脱管状态的对象变成持久状态。

7、大型网站在架构上应当考虑哪些问题？

考察点：Java 架构

参考回答：

- 分层：分层是处理任何复杂系统最常见的手段之一，将系统横向切分成若干个层面，每个层面只承担单一的职责，然后通过下层为上层提供的基础设施和服务以及上层对下层的调用来形成一个完整的复杂的系统。计算机网络的开放系统互联参考模型(OSI/RM)和 Internet 的 TCP/IP 模型都是分层结构，大型网站的软件系统也可以使用分层的理念将其分为持久层（提供数据存储和访问服务）、业务层（处理业务逻辑，系统中最核心的部分）和表示层（系统交互、视图展示）。需要指出的是：（1）分层是逻辑上的划分，在物理上可以位于同一设备上也可以在不同的设备上部署不同的功能模块，这样可以使用更多的计算资源来应对用户的并发访问；（2）层与层之间应当有清晰的边界，这样分层才有意义，才更利于软件的开发和维护。
- 分割：分割是对软件的纵向切分。我们可以将大型网站的不同功能和服务分割开，形成高内聚低耦合的功能模块（单元）。在设计初期可以做一个粗粒度的分割，将网站分割为若干个功能模块，后期还可以进一步对每个模块进行细粒度的分割，这样一方面有助于软件的开发和维护，另一方面有助于分布式的部署，提供网站的并发处理能力和功能的扩展。
- 分布式：除了上面提到的内容，网站的静态资源（JavaScript、CSS、图片等）也可以采用独立分布式部署并采用独立的域名，这样可以减轻应用服务器的负载压力，也使得浏览器对资源的加载更快。数据的存取也应该是分布式的，传统的商业级关系型数据库产品基本上都支持分布式部署，而新生的 NoSQL 产品几乎都是分布式的。当然，网站后台的业务处理也要使用分布式技术，例如查询索引的构建、数据分析等，这些业务计算规模庞大，可以使用 Hadoop 以及 MapReduce 分布式计算框架来处理。
- 集群：集群使得有更多的服务器提供相同的服务，可以更好的提供对并发的支持。
- 缓存：所谓缓存就是用空间换取时间的技术，将数据尽可能放在距离计算最近的位置。使用缓存是网站优化的第一定律。我们通常说的 CDN、反向代理、热点数据都是对缓存技术的使用。
- 异步：异步是实现软件实体之间解耦合的又一重要手段。异步架构是典型的生产者消费者模式，二者之间没有直接的调用关系，只要保持数据结构不变，彼此功能实现可以随意变化而不互相影响，这对网站的扩展非常有利。使用异步处理还可以提高系统可用性，加快网站的响应速度（用 Ajax 加载数据就是一种异步技术），同时还可以起到削峰作用（应对瞬时高并发）。&quot；能推迟处理的都要推迟处理"是网站优化的第二定律，而异步是践行网站优化第二定律的重要手段。
- 冗余：各种服务器都要提供相应的冗余服务器以便在某台或某些服务器宕机时还能保证网站可以正常工作，同时也提供了灾难恢复的可能性。冗余是网站高可用性的重要保证。

③Web 容器

1、请对以下在 J2EE 中常用的名词进行解释(或简单描述)

考察点：J2EE

参考回答：

web 容器：给处于其中的应用程序组件（JSP，SERVLET）提供一个环境，使 JSP,SERVLET 直接和容器中的环境变量接接口互，不必关注其它系统问题。主要有 WEB 服务器来实现。例如：TOMCAT,WEBLOGIC,WEBSPHERE 等。该容器提供的接口严格遵守 J2EE 规范中的 WEBAPPLICATION 标准。我们把遵守以上标准的 WEB 服务器就叫做 J2EE 中的 WEB 容器。
Web container：实现 J2EE 体系结构中 Web 组件协议的容器。这个协议规定了一个 Web 组件运行时的环境，包括安全，一致性，生命周期管理，事务，配置和其它的服务。一个提供和 JSP 和 J2EE 平台 APIs 界面相同服务的容器。一个 Web container 由 Web 服务器或者 J2EE 服务器提供。
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。一个实现了 J2EE 体系结构中 EJB 组件规范的容器。这个规范指定了一个 Enterprise bean 的运行时环境，包括安全，一致性，生命周期，事务，配置，和其他的服务。
JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系统，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。RMI-IIOP 出现以前，只有 RMI 和 CORBA 两种选择来进行分布式程序设计。
RMI-IIOP 综合了 RMI 和 CORBA 的优点，克服了他们的缺点，使得程序员能更方便的编写分布式程序设计，实现分布式计算。首先，RMI-IIOP 综合了 RMI 的简单性和 CORBA 的多语言性（兼容性），其次 RMI-IIOP 克服了 RMI 只能用于 Java 的缺点和 CORBA 的复杂性。

四、JDBC 编程

1、SQL 基础

1、写 SQL：找出每个城市的最新一条记录。

考察点：sql 语句

参考回答：

id 城市 人口 信息 创建时间
1 北京 100 info1 时间戳
2 北京 100 info2 时间戳
3 上海 100 info3 时间戳
4 上海 100 info4 时间戳

2、一个学生表，一个课程成绩表，怎么找出学生课程的最高分数
3、有一组合索引（A,B,C），会出现哪几种查询方式？tag:sql 语句

考察点：sql 语句

参考回答：

优: select * from test where a=10 and b>50

差: select * from test where b = 50

优: select * from test order by a

差: select * from test order by b

差: select * from test order by c

优: select * from test where a=10 order by a

优: select * from test where a=10 order by b

差: select * from test where a=10 order by c

优: select * from test where a>10 order by a

差: select * from test where a>10 order by b

差: select * from test where a>10 order by c

优: select * from test where a=10 and b=10 order by a

优: select * from test where a=10 and b=10 order by b

优: select * from test where a=10 and b=10 order by c

优: select * from test where a=10 and b=10 order by a

优: select * from test where a=10 and b>10 order by b

差: select * from test where a=10 and b>10 order by c

2、JDBC 基础

①数据库

1、数据库水平切分，垂直切分

考察点：数据库

参考回答：

垂直拆分就是要把表按模块划分到不同数据库表中（当然原则还是不破坏第三范式），这种拆分在大型网站的演变过程中是很常见的。当一个网站还在很小的时候，只有小量的人来开发和维护，各模块和表都在一起，当网站不断丰富和壮大的时候，也会变成多个子系统来支撑，这时就有按模块和功能把表划分出来的需求。其实，相对于垂直切分更进一步的是服务化改造，说得简单就是要把原来强耦合的系统拆分成多个弱耦合的服务，通过服务间的调用来满足业务需求看，因此表拆出来后要通过服务的形式暴露出去，而不是直接调用不同模块的表，淘宝在架构不断演变过程，最重要的一环就是服务化改造，把用户、交易、店铺、宝贝这些核心的概念抽取成独立的服务，也非常有利于进行局部的优化和治理，保障核心模块的稳定性。

垂直拆分：单表大数据量依然存在性能瓶颈

水平拆分，上面谈到垂直切分只是把表按模块划分到不同数据库，但没有解决单表大数据量的问题，而水平切分就是要把一个表按照某种规则把数据划分到不同表或数据库里。例如像计费系统，通过按时间来划分表就比较合适，因为系统都是处理某一时间段的数据。而像 SaaS 应用，通过按用户维度来划分数据比较合适，因为用户与用户之间的隔离的，一般不存在处理多个用户数据的情况，简单的按 user_id 范围来水平切分。

通俗理解：水平拆分行，行数据拆分到不同表中， 垂直拆分列，表数据拆分到不同表中。

2、数据库索引介绍一下。介绍一下什么时候用 Innodb 什么时候用 MyISAM。

考察点：数据库

参考回答：

存储引擎

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。如果想按特定职员的姓来查找他或她，则与在表中搜索所有的行相比，索引有助于更快地获取信息。索引的一个主要目的就是加快检索表中数据的方法，亦即能协助信息搜索者尽快的找到符合限制条件的记录 ID 的辅助数据结构。InnoDB 主要面向在线事务处理（OLTP）的应用。MyISAM 主要面向一些 OLAP 的应用。

3、数据库两种引擎

考察点：数据库存储引擎

参考回答：

InnoDB 是聚集索引，支持事务，支持行级锁；MyISAM 是非聚集索引，不支持事务，只支持表级锁。

4、索引了解嘛，底层怎么实现的，什么时候会失效

考察点：数据库索引

参考回答：

B+树实现的。

没有遵循最左匹配原则。

一些关键字会导致索引失效，例如 or， ！= ， not in，is null ,is not unll

like 查询是以%开头

隐式转换会导致索引失效。

对索引应用内部函数，索引字段进行了运算。

5、问了数据库的隔离级别

考察点：事务的隔离级别

参考回答：

隔离级别	脏读（Dirty Read）	不可重复读（NonRepeatable Read）	幻读（Phantom Read）
未提交读（Read uncommitted）	可能	可能	可能
已提交读（Read committed）	不可能	可能	可能
可重复读（Repeatable read）	不可能	不可能	可能
可串行化（Serializable ）	不可能	不可能	不可能

未提交读(Read Uncommitted)：允许脏读，也就是可能读取到其他会话中未提交事务修改的数据。

提交读(Read Committed)：只能读取到已经提交的数据。Oracle 等多数数据库默认都是该级别 (不重复读)。

可重复读(Repeated Read)：可重复读。在同一个事务内的查询都是事务开始时刻一致的，InnoDB 默认级别。在 SQL 标准中，该隔离级别消除了不可重复读，但是还存在幻象读。

串行读(Serializable)：完全串行化的读，每次读都需要获得表级共享锁，读写相互都会阻塞。

6、数据库乐观锁和悲观锁

考察点：数据库

参考回答：

悲观锁

悲观锁（Pessimistic Lock），顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会 block 直到它拿到锁。悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作。

Java synchronized 就属于悲观锁的一种实现，每次线程要修改数据时都先获得锁，保证同一时刻只有一个线程能操作数据，其他线程则会被 block。

乐观锁

乐观锁（Optimistic Lock），顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在提交更新的时候会判断一下在此期间别人有没有去更新这个数据。乐观锁适用于读多写少的应用场景，这样可以提高吞吐量。

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。

乐观锁一般来说有以下 2 种方式：

使用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。何谓数据版本？即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的 “version” 字段来实现。当读取数据时，将 version 字段的值一同读出，数据每更新一次，对此 version 值加一。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的 version 值进行比对，如果数据库表当前版本号与第一次取出来的 version 值相等，则予以更新，否则认为是过期数据。

使用时间戳（timestamp）。乐观锁定的第二种实现方式和第一种差不多，同样是在需要乐观锁控制的 table 中增加一个字段，名称无所谓，字段类型使用时间戳（timestamp），和上面的 version 类似，也是在更新提交的时候检查当前数据库中数据的时间戳和自己更新前取到的时间戳进行对比，如果一致则 OK，否则就是版本冲突。

7、数据库的三范式？

考察点：数据库

参考回答：

第一范式（1NF）

强调的是列的原子性，即列不能够再分成其他几列。

第二范式（2NF）

首先是 1NF，另外包含两部分内容，一是表必须有一个主键；二是没有包含在主键中的列必须完全依赖于主键，而不能只依赖于主键的一部分。

在 1NF 基础上，任何非主属性不依赖于其它非主属性[在 2NF 基础上消除传递依赖]。

第三范式（3NF）

第三范式（3NF）是第二范式（2NF）的一个子集，即满足第三范式（3NF）必须满足第二范式（2NF）。

首先是 2NF，另外非主键列必须直接依赖于主键，不能存在传递依赖。即不能存在：非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况。

8、讲一下数据库 ACID 的特性？

考察点：数据库

参考回答：

原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。

一致性指事务前后数据的完整性必须保持一致。

隔离性指多个用户并发访问数据库时，一个用户的事务不能被其他用户的事务所干扰，多个并发事务之间数据要相互隔离。

持久性是指一个事务一旦提交，它对数据库中数据的改变就是永久性的，即便数据库发生故障也不应该对其有任何影响。

9、mysql 主从复制？

考察点：数据库

参考回答：

MySQL 主从复制是其最重要的功能之一。主从复制是指一台服务器充当主数据库服务器，另一台或多台服务器充当从数据库服务器，主服务器中的数据自动复制到从服务器之中。对于多级复制，数据库服务器即可充当主机，也可充当从机。MySQL 主从复制的基础是主服务器对数据库修改记录二进制日志，从服务器通过主服务器的二进制日志自动执行更新。

MySQL 主从复制的两种情况：同步复制和异步复制，实际复制架构中大部分为异步复制。

复制的基本过程如下：

Slave 上面的 IO 进程连接上 Master，并请求从指定日志文件的指定位置（或者从最开始的日志）之后的日志内容。

Master 接收到来自 Slave 的 IO 进程的请求后，负责复制的 IO 进程会根据请求信息读取日志指定位置之后的日志信息，返回给 Slave 的 IO 进程。返回信息中除了日志所包含的信息之外，还包括本次返回的信息已经到 Master 端的 bin-log 文件的名称以及 bin-log 的位置。

Slave 的 IO 进程接收到信息后，将接收到的日志内容依次添加到 Slave 端的 relay-log 文件的最末端，并将读取到的 Master 端的 bin-log 的文件名和位置记录到 master-info 文件中，以便在下一次读取的时候能够清楚的告诉 Master“我需要从某个 bin-log 的哪个位置开始往后的日志内容，请发给我”。

Slave 的 Sql 进程检测到 relay-log 中新增加了内容后，会马上解析 relay-log 的内容成为在 Master 端真实执行时候的那些可执行的内容，并在自身执行。

10、leftjoin 和 rightjoin 的区别？

考察点：表结构

参考回答：

left join(左联接) 返回包括左表中的所有记录和右表中联结字段相等的记录
right join(右联接) 返回包括右表中的所有记录和左表中联结字段相等的记录

比如：

表 A 记录如下：

aID	aNum
1	a20050111
2	a20050112
3	a20050113
4	a20050114
5	a20050115

表 B 记录如下:

bID	bName
1	2006032401
2	2006032402
3	2006032403
4	2006032404
8	2006032408

left join 是以 A 表的记录为基础的,A 可以看成左表,B 可以看成右表,left join 是以左表为准的.
换句话说,左表(A)的记录将会全部表示出来,而右表(B)只会显示符合搜索条件的记录(例子中为: A.aID = B.bID).
B 表记录不足的地方均为 NULL.

11、数据库优化方法

考察点：数据库

参考回答：

（1）选取最适用的字段属性

MySQL 可以很好的支持大数据量的存取，但是一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小。

例如，在定义邮政编码这个字段时，如果将其设置为 CHAR(255),显然给数据库增加了不必要的空间，甚至使用 VARCHAR 这种类型也是多余的，因为 CHAR(6)就可以很好的完成任务了。同样的，如果可以的话，我们应该使用 MEDIUMINT 而不是 BIGIN 来定义整型字段。

另外一个提高效率的方法是在可能的情况下，应该尽量把字段设置为 NOTNULL，这样在将来执行查询的时候，数据库不用去比较 NULL 值。
对于某些文本字段，例如“省份”或者“性别”，我们可以将它们定义为 ENUM 类型。因为在 MySQL 中，ENUM 类型被当作数值型数据来处理，而数值型数据被处理起来的速度要比文本类型快得多。这样，我们又可以提高数据库的性能。

（2）使用连接（JOIN）来代替子查询(Sub-Queries)

MySQL 从 4.1 开始支持 SQL 的子查询。这个技术可以使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。例如，我们要将客户基本信息表中没有任何订单的客户删除掉，就可以利用子查询先从销售信息表中将所有发出订单的客户 ID 取出来，然后将结果传递给主查询

（3）使用联合(UNION)来代替手动创建的临时表

MySQL 从 4.0 的版本开始支持 union 查询，它可以把需要使用临时表的两条或更多的 select 查询合并的一个查询中。在客户端的查询会话结束的时候，临时表会被自动删除，从而保证数据库整齐、高效。使用 union 来创建查询的时候，我们只需要用 UNION 作为关键字把多个 select 语句连接起来就可以了，要注意的是所有 select 语句中的字段数目要想同。下面的例子就演示了一个使用 UNION 的查询。

（4）事务

尽管我们可以使用子查询（Sub-Queries）、连接（JOIN）和联合（UNION）来创建各种各样的查询，但不是所有的数据库操作都可以只用一条或少数几条 SQL 语句就可以完成的。更多的时候是需要用到一系列的语句来完成某种工作。但是在这种情况下，当这个语句块中的某一条语句运行出错的时候，整个语句块的操作就会变得不确定起来。设想一下，要把某个数据同时插入两个相关联的表中，可能会出现这样的情况：第一个表中成功更新后，数据库突然出现意外状况，造成第二个表中的操作没有完成，这样，就会造成数据的不完整，甚至会破坏数据库中的数据。要避免这种情况，就应该使用事务，它的作用是：要么语句块中每条语句都操作成功，要么都失败。换句话说，就是可以保持数据库中数据的一致性和完整性。事物以 BEGIN 关键字开始，COMMIT 关键字结束。在这之间的一条 SQL 操作失败，那么，ROLLBACK 命令就可以把数据库恢复到 BEGIN 开始之前的状态。

12、谈一下你对继承映射的理解。

考察点：映射

参考回答：

继承关系的映射策略有三种：
① 每个继承结构一张表（table per class hierarchy），不管多少个子类都用一张表。
② 每个子类一张表（table per subclass），公共信息放一张表，特有信息放单独的表。
③ 每个具体类一张表（table per concrete class），有多少个子类就有多少张表。
第一种方式属于单表策略，其优点在于查询子类对象的时候无需表连接，查询速度快，适合多态查询；缺点是可能导致表很大。后两种方式属于多表策略，其优点在于数据存储紧凑，其缺点是需要进行连接查询，不适合多态查询。

②数据库连接池

1、说出数据连接池的工作机制是什么？

考察点：连接池

参考回答：

J2EE 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量由配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

③事物管理，批处理

1、事务的 ACID 是指什么？

考察点：数据库

参考回答：

- 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； - 一致性(Consistent)：事务结束后系统状态是一致的； - 隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态； - 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。

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。当多个事务访问同一数据时，可能会存在 5 类问题，包括 3 类数据读取问题（脏读、不可重复读和幻读）和 2 类数据更新问题（第 1 类丢失更新和第 2 类丢失更新）。

2、JDBC 中如何进行事务处理？

考察点：数据库

参考回答：

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，从 JDBC 3.0 中还引入了 Savepoint（保存点）的概念，允许通过代码设置保存点并让事务回滚到指定的保存点。

3、JDBC 进阶

1、JDBC 的反射，反射都是什么？

考察点：jdbc

参考回答：

通过反射 com.mysql.jdbc.Driver 类，实例化该类的时候会执行该类内部的静态代码块，该代码块会在 Java 实现的 DriverManager 类中注册自己,DriverManager 管理所有已经注册的驱动类，当调用 DriverManager.geConnection 方法时会遍历这些驱动类，并尝试去连接数据库，只要有一个能连接成功，就返回 Connection 对象，否则则报异常。

2、Jdo 是什么?

考察点：JAVA API

参考回答：

JDO 是 Java 对象持久化的新的规范，为 java data object 的简称,也是一个用于存取某种数据仓库中的对象的标准化 API。JDO 提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如 JDBC API 的使用）。这些繁琐的例行工作已经转移到 JDO 产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO 很灵活，因为它可以在任何数据底层上运行。JDBC 只是面向关系数据库（RDBMS）JDO 更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML 以及对象数据库（ODBMS）等等，使得应用可移植性更强。

3、Statement 和 PreparedStatement 有什么区别？哪个性能更好？

考察点：Statement

参考回答：

与 Statement 相比，①PreparedStatement 接口代表预编译的语句，它主要的优势在于可以减少 SQL 的编译错误并增加 SQL 的安全性（减少 SQL 注射攻击的可能性）；②PreparedStatement 中的 SQL 语句是可以带参数的，避免了用字符串连接拼接 SQL 语句的麻烦和不安全；③当批量处理 SQL 或频繁执行相同的查询时，PreparedStatement 有明显的性能上的优势，由于数据库可以将编译优化后的 SQL 语句缓存起来，下次执行相同结构的语句时就会很快（不用再次编译和生成执行计划）。

为了提供对存储过程的调用，JDBC API 中还提供了 CallableStatement 接口。存储过程（Stored Procedure）是数据库中一组为了完成特定功能的 SQL 语句的集合，经编译后存储在数据库中，用户通过指定存储过程的名字并给出参数（如果该存储过程带有参数）来执行它。虽然调用存储过程会在网络开销、安全性、性能上获得很多好处，但是存在如果底层数据库发生迁移时就会有很多麻烦，因为每种数据库的存储过程在书写上存在不少的差别。

4、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？

考察点：JDBC 优化

参考回答：

要提升读取数据的性能，可以指定通过结果集（ResultSet）对象的 setFetchSize()方法指定每次抓取的记录数（典型的空间换时间策略）；要提升更新数据的性能可以使用 PreparedStatement 语句构建批处理，将若干 SQL 语句置于一个批处理中执行。

五、XML 编程

1、XML 基础

1、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？

考察点：XML

参考回答：

a: 两种形式 dtd schema
b: 本质区别:schema 本身是 xml 的，可以被 XML 解析器解析(这也是从 DTD 上发展 schema 的根本目的)
c:有 DOM,SAX,STAX 等
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
SAX:不现于 DOM,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
STAX:Streaming API for XML (StAX)
xml 文档有两种定义方法：
dtd：数据类型定义（data type definition），用以描述 XML 文档的文档结构，是早期的 XML 文档定义形式。
schema：其本身是基于 XML 语言编写的，在类型和语法上的限定能力比 dtd 强，处理也比较方便，因为此正逐渐代替 dtd 成为新的模式定义语言。

2、XML 进阶

3、Web service

①WSDL 与 SOAP 协议

1、WEB SERVICE 名词解释，JSWDL 开发包的介绍，JAXP、JAXM 的解释。SOAP、UDDI,WSDL 解释。

考察点：web service

参考回答：

Web ServiceWeb Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 WebService 能与其他兼容的组件进行互操作。JAXP(Java API for XML Parsing) 定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口。这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码。JAXM(Java API for XML Messaging) 是为 SOAP 通信提供访问方法和传输机制的 API。WSDL 是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。SOAP 即简单对象访问协议(Simple Object Access Protocol),它是用于交换 XML 编码信息的轻量级协议。UDDI 的目的是为电子商务建立标准;UDDI 是一套基于 Web 的、分布式的、为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准。soap 是 web service 最关键的技术，是 web service 中数据和方法调传输的介质。WSDL（web service definition language）描述了 web service 的接口和功能。

2、请你谈谈对 SOAP、WSDL、UDDI 的了解？

考察点：协议&语言

参考回答：

- SOAP：简单对象访问协议（Simple Object Access Protocol），是 Web Service 中交换数据的一种协议规范。
- WSDL：Web 服务描述语言（Web Service Description Language），它描述了 Web 服务的公共接口。这是一个基于 XML 的关于如何与 Web 服务通讯和使用的服务描述；也就是描述与目录中列出的 Web 服务进行交互时需要绑定的协议和信息格式。通常采用抽象语言描述该服务支持的操作和信息，使用的时候再将实际的网络协议和信息格式绑定给该服务。
- UDDI：统一描述、发现和集成（Universal Description, Discovery and Integration），它是一个基于 XML 的跨平台的描述规范,可以使世界范围内的企业在互联网上发布自己所提供的服务。简单的说，UDDI 是访问各种 WSDL 的一个门面（可以参考设计模式中的门面模式）。

3、谈谈 Java 规范中和 Web Service 相关的规范有哪些？

考察点：规范

参考回答：

Java 规范中和 Web Service 相关的有三个：
- JAX-WS(JSR 224)：这个规范是早期的基于 SOAP 的 Web Service 规范 JAX-RPC 的替代版本，它并不提供向下兼容性，因为 RPC 样式的 WSDL 以及相关的 API 已经在 Java EE5 中被移除了。WS-MetaData 是 JAX-WS 的依赖规范，提供了基于注解配置 Web Service 和 SOAP 消息的相关 API。
- JAXM(JSR 67)：定义了发送和接收消息所需的 API,相当于 Web Service 的服务器端。
- JAX-RS(JSR 311 & JSR 339 & JSR 370)：是 Java 针对 REST（Representation State Transfer）架构风格制定的一套 Web Service 规范。REST 是一种软件架构模式，是一种风格，它不像 SOAP 那样本身承载着一种消息协议，(两种风格的 Web Service 均采用了 HTTP 做传输协议,因为 HTTP 协议能穿越防火墙，Java 的远程方法调用（RMI）等是重量级协议，通常不能穿越防火墙），因此可以将 REST 视为基于 HTTP 协议的软件架构。REST 中最重要的两个概念是资源定位和资源操作，而 HTTP 协议恰好完整的提供了这两个点。HTTP 协议中的 URI 可以完成资源定位，而 GET、POST、OPTION、DELETE 方法可以完成资源操作。因此 REST 完全依赖 HTTP 协议就可以完成 Web

Service，而不像 SOAP 协议那样只利用了 HTTP 的传输特性，定位和操作都是由 SOAP 协议自身完成的，也正是由于 SOAP 消息的存在使得基于 SOAP 的 Web Service 显得笨重而逐渐被淘汰。

六、计算机网络

1、网络概述

①关于分层

1、TCP 协议在哪一层？IP 协议在那一层？HTTP 在哪一层？

考察点：网络七层模型

参考回答：

运输层，网络层，应用层。

2、运输层

①TCP 与 UDP

1、讲一下 TCP 的连接和释放连接。

考察点：网络基础

参考回答：

三次握手的过程

1）主机 A 向主机 B 发送 TCP 连接请求数据包，其中包含主机 A 的初始序列号 seq(A)=x。（其中报文中同步标志位 SYN=1，ACK=0，表示这是一个 TCP 连接请求数据报文；序号 seq=x，表明传输数据时的第一个数据字节的序号是 x）；

2）主机 B 收到请求后，会发回连接确认数据包。（其中确认报文段中，标识位 SYN=1，ACK=1，表示这是一个 TCP 连接响应数据报文，并含主机 B 的初始序列号 seq(B)=y，以及主机 B 对主机 A 初始序列号的确认号 ack(B)=seq(A)+1=x+1）

3）第三次，主机 A 收到主机 B 的确认报文后，还需作出确认，即发送一个序列号 seq(A)=x+1；确认号为 ack(A)=y+1 的报文；

四次挥手过程

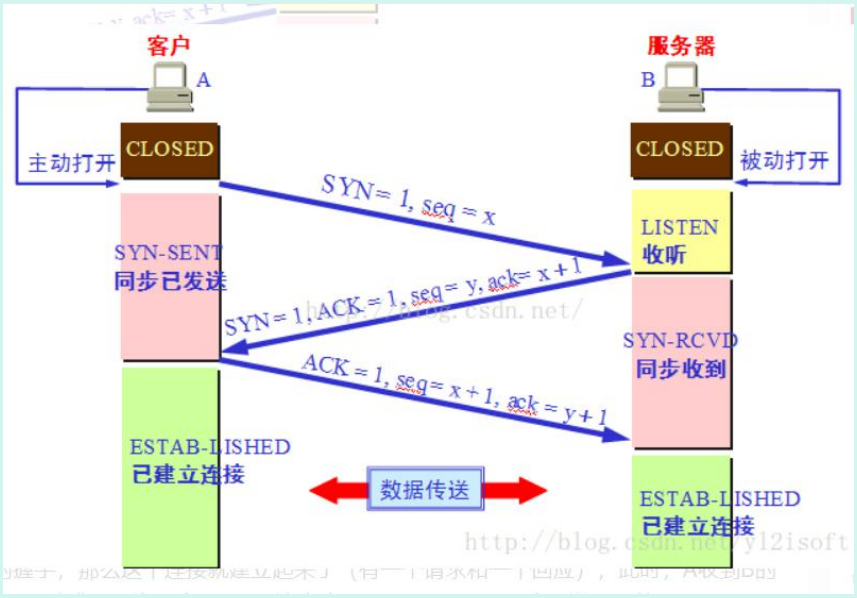
假设主机 A 为客户端，主机 B 为服务器，其释放 TCP 连接的过程如下：

1） 关闭客户端到服务器的连接：首先客户端 A 发送一个 FIN，用来关闭客户到服务器的数据传送，然后等待服务器的确认。其中终止标志位 FIN=1，序列号 seq=u。

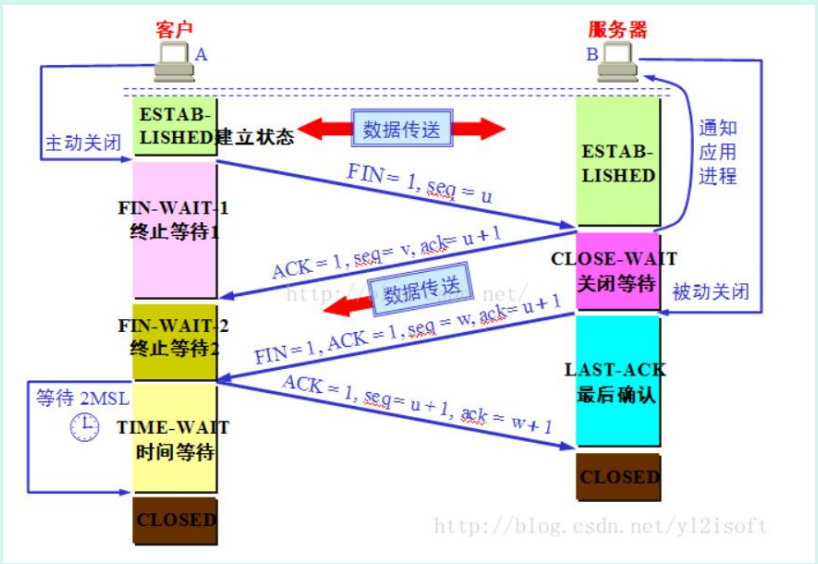
2） 服务器收到这个 FIN，它发回一个 ACK，确认号 ack 为收到的序号加 1。

3） 关闭服务器到客户端的连接：也是发送一个 FIN 给客户端。

4） 客户段收到 FIN 后，并发回一个 ACK 报文确认，并将确认序号 seq 设置为收到序号加 1。 首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。



三次握手



四次挥手

2、TCP 有哪些应用场景

考察点：TCP 协议

参考回答：

当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如 HTTP、HTTPS、FTP 等传输文件的协议，POP、SMTP 等邮件传输的协议

3、tcp 为什么可靠

考察点：TCP

参考回答：

三次握手，超时重传，滑动窗口，拥塞控制。

4、tcp 为什么要建立连接

考察点：TCP

参考回答：

保证可靠传输。

5、阐述 TCP 的 4 次挥手

考察点：TCP 协议

参考回答：

由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。这个原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接。收到一个 FIN 只意味着这一方向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。

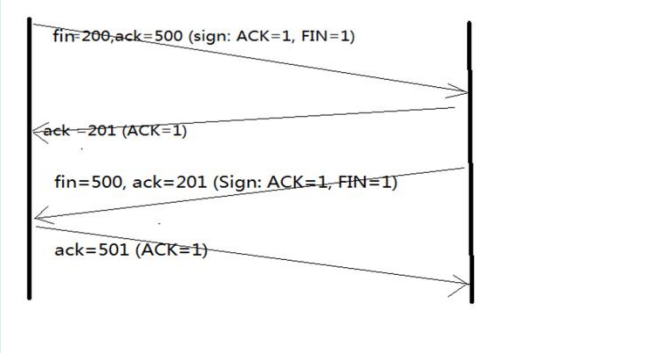
TCP 的连接的拆除需要发送四个包，因此称为四次挥手(four-way handshake)。客户端或服务器均可主动发起挥手动作，在 socket 编程中，任何一方执行 close()操作即可产生挥手操作。

（1）客户端 A 发送一个 FIN，用来关闭客户 A 到服务器 B 的数据传送。

（2）服务器 B 收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1。和 SYN 一样，一个 FIN 将占用一个序号。

（3）服务器 B 关闭与客户端 A 的连接，发送一个 FIN 给客户端 A。

（4）客户端 A 发回 ACK 报文确认，并将确认序号设置为收到序号加 1。



②协议

1、讲一下浏览器从接收到一个 URL 到最后展示出页面，经历了哪些过程。tag

考察点：http 协议

参考回答：

1.DNS 解析 2.TCP 连接 3.发送 HTTP 请求 4.服务器处理请求并返回 HTTP 报文

5.浏览器解析渲染页面

2、http 和 https 的区别

考察点：http 协议

参考回答；

https 协议要申请证书到 ca，需要一定经济成本；2） http 是明文传输，https 是加密的安全传输；3） 连接的端口不一样，http 是 80，https 是 443；4）http 连接很简单，没有状态；https 是 ssl 加密的传输，身份认证的网络协议，相对 http 传输比较安全。

3、http 的请求有哪些，应答码 502 和 504 有什么区别

考察点：http 协议

参考回答：

OPTIONS：返回服务器针对特定资源所支持的 HTTP 请求方法。也可以利用向 Web 服务器发送'*'的请求来测试服务器的功能性。

HEAD：向服务器索要与 GET 请求相一致的响应，只不过响应体将不会被返回。这一方法可以在不必传输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。

GET：向特定的资源发出请求。

POST：向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的创建和/或已有资源的修改。

PUT：向指定资源位置上传其最新内容。

DELETE：请求服务器删除 Request-URI 所标识的资源。

TRACE：回显服务器收到的请求，主要用于测试或诊断。

CONNECT：HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。

虽然 HTTP 的请求方式有 8 种，但是我们在实际应用中常用的也就是 get 和 post，其他请求方式也都可以通过这两种方式间接的来实现。

502：作为网关或者代理工作的服务器尝试执行请求时，从上游服务器接收到无效的响应。

504：作为网关或者代理工作的服务器尝试执行请求时，未能及时从上游服务器（URI 标识出的服务器，例如 HTTP、FTP、LDAP）或者辅助服务器（例如 DNS）收到响应。

4、http1.1 和 1.0 的区别

考察点：http

参考回答：

主要区别主要体现在：

缓存处理，在 HTTP1.0 中主要使用 header 里的 If-Modified-Since,Expires 来做为缓存判断的标准，HTTP1.1 则引入了更多的缓存控制策略例如 Entity tag，If-Unmodified-Since, If-Match, If-None-Match 等更多可供选择的缓存头来控制缓存策略。

带宽优化及网络连接的使用，HTTP1.0 中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了，并且不支持断点续传功能，HTTP1.1 则在请求头引入了 range 头域，它允许只请求资源的某个部分，即返回码是 206（Partial Content），这样就方便了开发者自由的选择以便于充分利用带宽和连接。

错误通知的管理，在 HTTP1.1 中新增了 24 个错误状态响应码，如 409（Conflict）表示请求的资源与资源的当前状态发生冲突；410（Gone）表示服务器上的某个资源被永久性的删除。

Host 头处理，在 HTTP1.0 中认为每台服务器都绑定一个唯一的 IP 地址，因此，请求消息中的 URL 并没有传递主机名（hostname）。但随着虚拟主机技术的发展，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个 IP 地址。HTTP1.1 的请求消息和响应消息都应支持 Host 头域，且请求消息中如果没有 Host 头域会报告一个错误（400 Bad Request）。

长连接，HTTP 1.1 支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟，在 HTTP1.1 中默认开启 Connection： keep-alive，一定程度上弥补了 HTTP1.0 每次请求都要创建连接的缺点。

5、说说 ssl 四次握手的过程

考察：HTTP 加密协议

参考回答：

1、 客户端发出请求

首先，客户端（通常是浏览器）先向服务器发出加密通信的请求，这被叫做 ClientHello 请求。

2、服务器回应

服务器收到客户端请求后，向客户端发出回应，这叫做 SeverHello。

3、客户端回应

客户端收到服务器回应以后，首先验证服务器证书。如果证书不是可信机构颁布、或者证书中的域名与实际域名不一致、或者证书已经过期，就会向访问者显示一个警告，由其选择是否还要继续通信。

4、服务器的最后回应

服务器收到客户端的第三个随机数 pre-master key 之后，计算生成本次会话所用的"会话密钥"。然后，向客户端最后发送下面信息。

（1）编码改变通知，表示随后的信息都将用双方商定的加密方法和密钥发送。

（2）服务器握手结束通知，表示服务器的握手阶段已经结束。这一项同时也是前面发送的所有内容的 hash 值，用来供客户端校验。

至此，整个握手阶段全部结束。接下来，客户端与服务器进入加密通信，就完全是使用普通的 HTTP 协议，只不过用"会话密钥"加密内容。

6、304 状态码有什么含义？

考察点：http

参考回答：

304(未修改)自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容。如果网页自请求者上次请求后再也没有更改过，您应将服务器配置为返回此响应(称为 If-Modified-Since HTTP 标头)。服务器可以告诉 Googlebot 自从上次抓取后网页没有变更，进而节省带宽和开销。

3、网络层

①网际协议 IP

1、arp 协议，arp 攻击

考察点：ARP 协议

参考回答：

地址解析协议。ARP 攻击的第一步就是 ARP 欺骗。由上述“ARP 协议的工作过程”我们知道，ARP 协议基本没有对网络的安全性做任何思考，当时人们考虑的重点是如何保证网络通信能够正确和快速的完成——ARP 协议工作的前提是默认了其所在的网络是一个善良的网络，每台主机在向网络中发送应答信号时都是使用的真实身份。不过后来，人们发现 ARP 应答中的 IP 地址和 MAC 地址中的信息是可以伪造的，并不一定是自己的真实 IP 地址和 MAC 地址，由此，ARP 欺骗就产生了。

②网际控制报文协议 ICMP

1、icmp 协议

考察点：ICMP 协议

参考回答：

它是 TCP/IP 协议族的一个子协议，用于在 IP 主机、路由器之间传递控制消息。控制消息是指网络通不通、主机是否可达、路由是否可用等网络本身的消息。这些控制消息虽然并不传输用户数据，但是对于用户数据的传递起着重要的作用。

③因特网的路由器选择协议

1、讲一下路由器和交换机的区别？

考察点：路由器

参考回答：

交换机用于同一网络内部数据的快速传输转发决策通过查看二层头部完成转发不需要修改数据帧工作在 TCP/IP 协议的二层 —— 数据链路层工作简单，直接使用硬件处理路由器用于不同网络间数据的跨网络传输转发决策通过查看三层头部完成转发需要修改 TTL ，IP 头部校验和需要重新计算，数据帧需要重新封装工作在 TCP/IP 协议的三层 —— 网络层工作复杂，使用软件处理。

4、应用层

①域名系统 DNS

1、DNS 寻址过程

考察点：DNS

参考回答：

1、在浏览器中输入 www.qq.com 域名，操作系统会先检查自己本地的 hosts 文件是否有这个网址映射关系，如果有，就先调用这个 IP 地址映射，完成域名解析。

2、如果 hosts 里没有这个域名的映射，则查找本地 DNS 解析器缓存，是否有这个网址映射关系，如果有，直接返回，完成域名解析。

3、如果 hosts 与本地 DNS 解析器缓存都没有相应的网址映射关系，首先会找 TCP/ip 参数中设置的首选 DNS 服务器，在此我们叫它本地 DNS 服务器，此服务器收到查询时，如果要查询的域名，包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析，此解析具有权威性。

4、如果要查询的域名，不由本地 DNS 服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个 IP 地址映射，完成域名解析，此解析不具有权威性。

5、如果本地 DNS 服务器本地区域文件与缓存解析都失效，则根据本地 DNS 服务器的设置（是否设置转发器）进行查询，如果未用转发模式，本地 DNS 就把请求发至 13 台根 DNS，根 DNS 服务器收到请求后会判断这个域名(.com)是谁来授权管理，并会返回一个负责该顶级域名服务器的一个 IP。本地 DNS 服务器收到 IP 信息后，将会联系负责.com 域的这台服务器。这台负责.com 域的服务器收到请求后，如果自己无法解析，它就会找一个管理.com 域的下一级 DNS 服务器地址(qq.com)给本地 DNS 服务器。当本地 DNS 服务器收到这个地址后，就会找 qq.com 域服务器，重复上面的动作，进行查询，直至找到 www.qq.com 主机。

6、如果用的是转发模式，此 DNS 服务器就会把请求转发至上一级 DNS 服务器，由上一级服务器进行解析，上一级服务器如果不能解析，或找根 DNS 或把转请求转至上上级，以此循环。不管是本地 DNS 服务器用是是转发，还是根提示，最后都是把结果返回给本地 DNS 服务器，由此 DNS 服务器再返回给客户机。

从客户端到本地 DNS 服务器是属于递归查询，而 DNS 服务器之间就是的交互查询就是迭代查询。

②电子邮件

1、负载均衡反向代理模式优点及缺点

考察点：反向代理

参考回答：

（1）反向代理（Reverse Proxy）方式是指以代理服务器来接受 internet 上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给 internet 上请求连接的客户端，此时代理服务器对外就表现为一个服务器。

（2）反向代理负载均衡技术是把将来自 internet 上的连接请求以反向代理的方式动态地转发给内部网络上的多台服务器进行处理，从而达到负载均衡的目的。

（3）反向代理负载均衡能以软件方式来实现，如 apache mod_proxy、netscape proxy 等，也可以在高速缓存器、负载均衡器等硬件设备上实现。反向代理负载均衡可以将优化的负载均衡策略和代理服务器的高速缓存技术结合在一起，提升静态网页的访问速度，提供有益的性能；由于网络外部用户不能直接访问真实的服务器，具备额外的安全性（同理，NAT 负载均衡技术也有此优点）。

（4）其缺点主要表现在以下两个方面

反向代理是处于 OSI 参考模型第七层应用的，所以就必须为每一种应用服务专门开发一个反向代理服务器，这样就限制了反向代理负载均衡技术的应用范围，现在一般都用于对 web 服务器的负载均衡。

针对每一次代理，代理服务器就必须打开两个连接，一个对外，一个对内，因此在并发连接请求数量非常大的时候，代理服务器的负载也就非常大了，在最后代理服务器本身会成为服务的瓶颈。

一般来讲，可以用它来对连接数量不是特别大，但每次连接都需要消耗大量处理资源的站点进行负载均衡，如 search 等。

七、操作系统

1、操作系统概论

1、CentOS 和 Linux 的关系？

考察点：操作系统

参考回答：

CentOS 是 Linux 众多得发行版本之一，linux 有三大发行版本（:Slackware、debian、redhat），而 Redhat 有收费的商业版和免费的开源版，商业版的业内称之为 RHEL 系列，CentOS 是来自于依照开放源代码规定而公布的源代码重新编译而成。可以用 CentOS 替代商业版的 RHEL 使用。两者的不同，CentOS 不包含封闭源代码软件，是免费的。

2、64 位和 32 位的区别？

考察点：

操作系统

参考回答：

操作系统只是硬件和应用软件中间的一个平台。32 位操作系统针对的 32 位的 CPU 设计。64 位操作系统针对的 64 位的 CPU 设计。

2、进程的描述与控制

1、怎么杀死进程？

考察点：进程

参考回答：

```
Kill pid
```

2、线程，进程区别

考察点：进程，线程

参考回答：

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。

3、系统线程数量上限是多少？

考察点：线程

参考回答：

Linux 系统中单个进程的最大线程数有其最大的限制 PTHREAD_THREADS_MAX。

这个限制可以在/usr/include/bits/local_lim.h 中查看 ，对 linuxthreads 这个值一般是 1024，对于 nptl 则没有硬性的限制，仅仅受限于系统的资源。

这个系统的资源主要就是线程的 stack 所占用的内存，用 ulimit -s 可以查看默认的线程栈大小，一般情况下，这个值是 8M=8192KB。

4、进程和线程的区别是什么？

考察点：JAVA 进程

参考回答：

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。

5、解释一下 LINUX 下线程，GDI 类。

考察点：线程

参考回答：

LINUX 实现的就是基于核心轻量级进程的”一对一”线程模型，一个线程实体对应一个核心轻量级进程，而线程之间的管理在核外函数库中实现。

GDI 类为图像设备编程接口类库。

3、输入输出系统

1、socket 编程，BIO，NIO，epoll？

考察点：I/O 多路复用

参考回答：

阻塞，非阻塞，io 多路复用，epoll 支持文件符数目没有限制，fd 集合只会从用户进程拷贝到内核一次，自己维护一个事件队列，不用每次遍历 fd 集合发现是否有就绪状态。

4、存储器管理

1、什么是页式存储？

考察点：页式存储

参考回答：

主存被等分成大小相等的片，称为主存块，又称为实页。

当一个用户程序装入内存时，以页面为单位进行分配。页面的大小是为 2n，通常为 1KB、2KB、2n KB 等

2、操作系统里的内存碎片你怎么理解，有什么解决办法？

考察点：内存碎片

参考回答：

内存碎片分为：内部碎片和外部碎片。

内部碎片就是已经被分配出去（能明确指出属于哪个进程）却不能被利用的内存空间；

内部碎片是处于区域内部或页面内部的存储块。占有这些区域或页面的进程并不使用这个存储块。而在进程占有这块存储块时，系统无法利用它。直到进程释放它，或进程结束时，系统才有可能利用这个存储块。

单道连续分配只有内部碎片。多道固定连续分配既有内部碎片，又有外部碎片。

外部碎片指的是还没有被分配出去（不属于任何进程），但由于太小了无法分配给申请内存空间的新进程的内存空闲区域。

外部碎片是出于任何已分配区域或页面外部的空闲存储块。这些存储块的总和可以满足当前申请的长度要求，但是由于它们的地址不连续或其他原因，使得系统无法满足当前申请。

使用伙伴系统算法。

5、处理机调度与死锁

1、什么情况下会发生死锁，解决策略有哪些？

考察点：死锁

参考回答：

（一）互斥条件：一个资源一次只能被一个进程访问。即某个资源在一段时间内只能由一个进程占有，不能同时被两个或两个以上的进程占 有。这种独占资源如 CD-ROM 驱动器，打印机等等，必须在占有该资源的进程主动释放它之后，其它进程才能占有该资源。这是由资源本身的属性所决定的。

（二）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。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外进程占有，此时该进程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。

（三）不剥夺条件：进程已经获得的资源，在未使用完之前不能强行剥夺，而只能由该资源的占有者进程自行释放。

（四）循环等待条件：若干资源形成一种头尾相接的循环等待资源关系。

解决方法：银行家算法

2、系统 CPU 比较高是什么原因？

考察点：处理机

参考回答：

1、首先查看是哪些进程的 CPU 占用率最高（如下可以看到详细的路径）

```
ps -aux --sort -pcpu | more

# 定位有问题的线程可以用如下命令

ps -mp pid -o THREAD,tid,time | more
```

2、查看 JAVA 进程的每个线程的 CPU 占用率

```
ps -Lp 5798 cu | more          # 5798 是查出来进程 PID
```

3、追踪线程，查看负载过高的原因，使用 JDK 下的一个工具

jstack 5798 # 5798 是 PID

jstack -J-d64 -m 5798 # -j-d64 指定 64 为系统

jstack 查出来的线程 ID 是 16 进制，可以把输出追加到文件，导出用记事本打开，再根据系统中的线程 ID 去搜索查看该 ID 的线程运行内容，可以和开发一起排查。

3、系统如何提高并发性？

考察：操作系统综合性

参考回答：

1、提高 CPU 并发计算能力

（1）多进程&多线程

（2）减少进程切换，使用线程，考虑进程绑定 CPU

（3）减少使用不必要的锁，考虑无锁编程

（4）考虑进程优先级

（5）关注系统负载

2、改进 I/O 模型

(1)DMA 技术

(2)异步 I/O

(3)改进多路 I/O 就绪通知策略，epoll

(4)Sendfile

(5)内存映射

(6)直接 I/O

八、算法与数据结构

1、哈希

1、hashset 存的数是有序的吗？

考察点：哈希

参考回答：

Hashset 是无序的。

2、Object 作为 HashMap 的 key 的话，对 Object 有什么要求吗？

考察点：哈希表

参考回答：

要求 Object 中 hashcode 不能变。

3、一致性哈希算法

考察点：哈希算法

参考回答：

先构造一个长度为 232 的整数环（这个环被称为一致性 Hash 环），根据节点名称的 Hash 值（其分布为[0, 232-1]）将服务器节点放置在这个 Hash 环上，然后根据数据的 Key 值计算得到其 Hash 值（其分布也为[0, 232-1]），接着在 Hash 环上顺时针查找距离这个 Key 值的 Hash 值最近的服务器节点，完成 Key 到服务器的映射查找。

这种算法解决了普通余数 Hash 算法伸缩性差的问题，可以保证在上线、下线服务器的情况下尽量有多的请求命中原来路由到的服务器。

4、什么是 hashmap?

考察点：哈希表

参考回答：

HashMap 是一个散列表，它存储的内容是键值对(key-value)映射。
HashMap 继承于 AbstractMap，实现了 Map、Cloneable、java.io.Serializable 接口。
HashMap 的实现不是同步的，这意味着它不是线程安全的。它的 key、value 都可以为 null。此外，HashMap 中的映射不是有序的。

HashMap 的实例有两个参数影响其性能：“初始容量” 和 “加载因子”。容量 是哈希表中桶的数量，初始容量 只是哈希表在创建时的容量。加载因子 是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行 rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。
通常，默认加载因子是 0.75, 这是在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数 HashMap 类的操作中，包括 get 和 put 操作，都反映了这一点）。在设置初始容量时应该考虑到映射中所需的条目数及其加载因子，以便最大限度地减少 rehash 操作次数。如果初始容量大于最大条目数除以加载因子，则不会发生 rehash 操作。

hashmap 共有 4 个构造函数：

// 默认构造函数。HashMap()

// 指定“容量大小”的构造函数

```
HashMap(int capacity)

// 指定“容量大小”和“加载因子”的构造函数

HashMap(int capacity, float loadFactor)

// 包含“子 Map”的构造函数

HashMap(Map<? extends K, ? extends V> map)
```

5、Java 中的 HashMap 的工作原理是什么？

考察点：JAVA 哈希表

参考回答：

HashMap 类有一个叫做 Entry 的内部类。这个 Entry 类包含了 key-value 作为实例变量。每当往 hashmap 里面存放 key-value 对的时候，都会为它们实例化一个 Entry 对象，这个 Entry 对象就会存储在前面提到的 Entry 数组 table 中。Entry 具体存在 table 的那个位置是 根据 key 的 hashcode()方法计算出来的 hash 值（来决定）。

6、hashCode()和 equals()方法的重要性体现在什么地方？

考察点：JAVA 哈希表

参考回答：

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。

2、树

1、说一下 B+树和 B-树？

考察点：树

参考回答：

b+树的中间节点不保存数据，所以磁盘页能容纳更多节点元素，更“矮胖”；

b+树查询必须查找到叶子节点，b 树只要匹配到即可不用管元素位置，因此 b+树查找更稳定（并不慢）；

对于范围查找来说，b+树只需遍历叶子节点链表即可，b 树却需要重复地中序遍历。

2、怎么求一个二叉树的深度?手撕代码?

考察点：二叉树

参考回答：

```
public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        int bigger = Math.max(left, right);
        return bigger + 1;
    }
```

3、算法题：二叉树层序遍历，进一步提问：要求每层打印出一个换行符

考察点：二叉树

参考回答：

```
public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        LinkedList<TreeNode> queue = new LinkedList<TreeNode>();
        if (root == null) {
            return res;
        }
        queue.offer(root);
        while (queue.size() != 0) {
            List<Integer> l = new ArrayList<Integer>();
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode temp = queue.poll();
                l.add(temp.val);
                if (temp.left != null) {
```

```
                    queue.offer(temp.left);
                }
                if (temp.right != null) {
                    queue.offer(temp.right);
                }
            }
            res.add(l);
        }
        return res;
    }
```

4、二叉树任意两个节点之间路径的最大长度

考察点：树

参考回答：

```
int maxDist(Tree root) {
    //如果树是空的，则返回 0
    if(root == NULL)
        return 0;
    if(root->left != NULL) {
        root->lm = maxDist(root->left) + 1;
    }
    if(root->right != NULL)
        root->rm = maxDist(root->right) + 1;
    //如果以该节点为根的子树中有最大的距离，那就更新最大距离
    int sum = root->rm + root->lm;
    if(sum > max) {
        max = sum;
```

```
        }


        return root->rm > root->lm ? root->rm : root->lm;
    }
```

5、如何实现二叉树的深度？

考察点：二叉树

参考回答：

实现二叉树的深度方式有两种，递归以及非递归。

①递归实现：

为了求树的深度，可以先求其左子树的深度和右子树的深度，可以用递归实现，递归的出口就是节点为空。返回值为 0；

②非递归实现：

利用层次遍历的算法，设置变量 level 记录当前节点所在的层数，设置变量 last 指向当前层的最后一个节点，当处理完当前层的最后一个节点，让 level 指向+1 操作。设置变量 cur 记录当前层已经访问的节点的个数，当 cur 等于 last 时，表示该层访问结束。

层次遍历在求树的宽度、输出某一层节点，某一层节点个数，每一层节点个数都可以采取类似的算法。

树的宽度：在树的深度算法基础上，加一个记录访问过的层节点个数最多的变量 max,在访问每层前 max 与 last 比较，如果 max 比较大，max 不变，如果 max 小于 last，把 last 赋值给 max;

代码解释：

```
import java.util.LinkedList;

public class Deep

{

//递归实现 1

  public int findDeep(BiTree root)

  {

  int deep = 0;

  if(root != null)

  {
```

```
            int lchilddeep = findDeep(root.left);
            int rchilddeep = findDeep(root.right);
            deep = lchilddeep > rchilddeep ? lchilddeep + 1 : rchilddeep + 1;
        }
        return deep;
        }



        //递归实现 2
        public int findDeep1(BiTree root)
        {

        if(root == null)
        {
            return 0;
        }
        else
        {
         int lchilddeep = findDeep1(root.left);//求左子树的深度
         int rchilddeep = findDeep1(root.left);//求右子树的深度
         return lchilddeep > rchilddeep ? lchilddeep + 1 : rchilddeep + 1;//左子树和
右子树深度较大的那个加一等于整个树的深度
        }
        }
        //非递归实现
        public int findDeep2(BiTree root)
        {
       if(root == null)
```

```
        return 0;


    BiTree current = null;
    LinkedList<BiTree> queue = new LinkedList<BiTree>();
    queue.offer(root);
    int cur,last;
    int level = 0;
    while(!queue.isEmpty())
    {
        cur = 0;//记录本层已经遍历的节点个数
        last = queue.size();//当遍历完当前层以后，队列里元素全是下一层的元素，队列的长度是这一层的节点的个数
        while(cur < last)//当还没有遍历到本层最后一个节点时循环
        {
            current = queue.poll();//出队一个元素
            cur++;
            //把当前节点的左右节点入队（如果存在的话）
            if(current.left != null)
            {
                queue.offer(current.left);
            }
            if(current.right != null)
            {
                queue.offer(current.right);
            }
        }
        level++;//每遍历完一层 level+1
    }
```

```
 return level;
  }
  public static void main(String[] args)
 {
BiTree root = BiTree.buildTree();
Deep deep = new Deep();
System.out.println(deep.findDeep(root));
System.out.println(deep.findDeep1(root));
System.out.println(deep.findDeep2(root));
 }
}
```

6、如何打印二叉树每层的节点？

考察点：二叉树

参考回答：

实现代码：

```
import java.util.ArrayList;
import java.util.Scanner;
public class Main {
// 定义节点
class Node{
    int val;
    Node left;
    Node right;
    public Node(int val) {
        this.val = val;
    }
}
```

```
public  ArrayList<Integer> gzy;   // 保存根左右的序列
public  ArrayList<Integer> zgy;   // 保存左跟右的序列
public  ArrayList<Node> pack;     // 保存已经排好的节点

public static void main(String[] args) {
    Main main = new Main();
    main.getResult();

}

public void getResult() {
    // init
    Scanner scanner = new Scanner(System.in);
    int count = scanner.nextInt();
    gzy = new ArrayList<>();
    zgy = new ArrayList<>();
    for(int i = 0; i < count; i++) {
        gzy.add(scanner.nextInt());
    }
    for(int j = 0; j < count; j++) {
        zgy.add(scanner.nextInt());
    }
    pack = new ArrayList<>();     // 已经还原的节点

    // exception
    if(count == 1) {
```

```
            System.out.println(gzy.get(0));
            return;
        }
        // 构造最左侧节点的二叉树
        Node node = new Node(gzy.get(0));
        pack.add(node);
        int index1 = 1;      // 根左右的下标
        Node tmp = node;
        while(gzy.get(index1) != zgy.get(0)) {        // 如果没访问到最左边的叶子节点,继续还原最左侧二叉树
            tmp.left = new Node(gzy.get(index1++));
            tmp = tmp.left;
            pack.add(tmp);
        }
        tmp.left = new Node(gzy.get(index1++));
        pack.add(tmp.left);

        // 加入剩余的节点完善二叉树
        for(int k = index1; k < gzy.size(); k++) {
            fillErCS(gzy.get(k));
        }

        // 层次遍历
        ArrayList<Node> res = new ArrayList<>();
        res.add(node);
        int num = 0;
        while(res.size() != num) {
            System.out.print(res.get(num).val + " ");
```

```
            if(res.get(num).left != null) {
                res.add(res.get(num).left);
            }
            if(res.get(num).right != null) {
                res.add(res.get(num).right);
            }
            num++;
        }
    }

    // 将值为 val 的节点加入二叉树
    private void fillErCS(int val) {
        int index = zgy.indexOf(val);
        // 每一个遍历的节点都是 val 节点的根或者在其左边
        for(int i = index-1; i >= 0; i--) {
            if(findNode(zgy.get(i)) != null) {    // 找到待插入节点的根节点或者其
左边的节点
                Node node = findNode(zgy.get(i));
                insert(node, val);
                break;
            }
        }
    }

    // 将节点 val 插入二叉树
    private void insert(Node node, int val) {
        if(zgy.indexOf(node.val) > zgy.indexOf(val)) {// node 在待插入节点的右边
            if(node.left == null) {
```

```
                node.left = new Node(val);
                pack.add(node.left);
                return;
            }
            insert(node.left, val);
        }else {      // node 在待插入节点的左边或是其根
            if(node.right == null) {
                node.right = new Node(val);
                pack.add(node.right);
                return;
            }
            insert(node.right, val);
        }
    }
    // 根据 val 找到 pack 里的节点
    private Node findNode(int val) {
        for(Node node : pack) {
            if(node.val == val) {
                return node;
            }
        }
        return null;
    }
    }
```

7、TreeMap 和 TreeSet 在排序时如何比较元素？Collections 工具类中的 sort()方法如何比较元素？

考察点：Tree

参考回答：

TreeSet 要求存放的对象所属的类必须实现 Comparable 接口，该接口提供了比较元素的 compareTo()方法，当插入元素时会回调该方法比较元素的大小。TreeMap 要求存放的键值对映射的键必须实现 Comparable 接口从而根据键对元素进行排序。Collections 工具类的 sort 方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现 Comparable 接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是通过接口注入比较元素大小的算法，也是对回调模式的应用（Java 中对函数式编程的支持）。

代码示例：

```
    public class Student implements Comparable<Student> {
       private String name;               // 姓名
       private int age;                        // 年龄

       public Student(String name, int age) {
               this.name = name;
               this.age = age;
       }
       @Override
       public String toString() {
               return "Student [name=" + name + ", age=" + age + "]";
       }

       @Override
       public int compareTo(Student o) {
               return this.age - o.age; // 比较年龄(年龄的升序)
       }
}
import java.util.Set;
import java.util.TreeSet;
class Test01 {
       public static void main(String[] args) {
               Set<Student> set = new TreeSet<>();         // Java 7 的钻石语法(构造器后面的尖括号中不需要写类型)
               set.add(new Student("Hao LUO", 33));
               set.add(new Student("XJ WANG", 32));
               set.add(new Student("Bruce LEE", 60));
               set.add(new Student("Bob YANG", 22));

               for(Student stu : set) {
                       System.out.println(stu);
               }
//           输出结果:
//           Student [name=Bob YANG, age=22]
//           Student [name=XJ WANG, age=32]
//           Student [name=Hao LUO, age=33]
//           Student [name=Bruce LEE, age=60]
       }
}
```

3、遍历

1、编程题：写一个函数，找到一个文件夹下所有文件，包括子文件夹

考察点：遍历

参考回答：

```
import java.io.File;


public class Counter2 {


public static void main(String[] args) {
    //取得目标目录
    File file = new File("D:");
    //获取目录下子文件及子文件夹
    File[] files = file.listFiles();
    readfile(files);


}


public static void readfile(File[] files) {
    if (files == null) {// 如果目录为空，直接退出
            return;
        }
    for(File f:files) {
        //如果是文件，直接输出名字
        if(f.isFile()) {
            System.out.println(f.getName());
        }
        //如果是文件夹，递归调用
```

```
            else if(f.isDirectory()) {
                readfile(f.listFiles());
            }
        }
    }
    }
```

2、二叉树 Z 字型遍历

考察点：遍历

参考回答：

```
public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<List<Integer>>();
        LinkedList<TreeNode> l = new LinkedList<TreeNode>();
        boolean flag = true;
        if (root == null) {
            return res;
        }
        l.add(root);
        while (l.size() != 0) {
            flag = !flag;
            int size = l.size();
            List<Integer> list = new ArrayList<Integer>();
            for (int i = 0; i < size; i++) {
                TreeNode temp = l.remove();
                list.add(temp.val);
                if (temp.left != null) l.add(temp.left);
                if (temp.right != null) l.add(temp.right);
            }
```

```
                if (flag) {
                    Collections.reverse(list);
                }
                res.add(list);
            }
            return res;
        }
```

4、链表

1、反转单链表

考察点：链表

参考回答：

```
ListNode reverseList(ListNode* head) {
        if(head == nullptr || head->next == nullptr)
            return head;
        ListNode* p;
        ListNode* q;
        ListNode* r;
        p = head;
        q = head->next;
        head->next = nullptr;//旧的头指针是新的尾指针 指向NULL
        while(q){
            r = q->next;//用来保存下一步要处理的指针
            q->next = p;//p q 交替处理 进行反转单链表
            p = q;
            q = r;
```

```
            }
            head = p;//最后的 q 必定指向 NULL，p 就成了新链表的头指针
            return head;
    }
```

2、随机链表的复制

考察点：链表

参考回答：

```
public RandomListNode copyRandomList(RandomListNode head) {

if (head == null)
    return null;

RandomListNode p = head;

// copy every node and insert to list
while (p != null) {
    RandomListNode copy = new RandomListNode(p.label);
    copy.next = p.next;
    p.next = copy;
    p = copy.next;
}

// copy random pointer for each new node
p = head;
while (p != null) {
    if (p.random != null)
        p.next.random = p.random.next;
```

```
        p = p.next.next;
    }

    // break list to two
    p = head;
    RandomListNode newHead = head.next;
    while (p != null) {
        RandomListNode temp = p.next;
        p.next = temp.next;
        if (temp.next != null)
            temp.next = temp.next.next;
        p = p.next;
    }

    return newHead;
    }
```

3、链表-奇数位升序偶数位降序-让链表变成升序

考察点：链表

参考回答：

```
public class OddIncreaseEvenDecrease {
    /**
     * 按照奇偶位拆分成两个链表
     * @param head
     * @return
     */
    public static Node[] getLists(Node head){
        Node head1 = null;
```

```
        Node head2 = null;


        Node cur1 = null;
        Node cur2 = null;
        int count = 1;//用来计数
        while(head != null){
            if(count % 2 == 1){
                if(cur1 != null){
                    cur1.next = head;
                    cur1 = cur1.next;
                }else{
                    cur1 = head;
                    head1 = cur1;
                }
            }else{
                if(cur2 != null){
                    cur2.next = head;
                    cur2 = cur2.next;
                }else{
                    cur2 = head;
                    head2 = cur2;
                }
            }
            head = head.next;
            count++;
        }
        //跳出循环，要让最后两个末尾元素的下一个都指向 null
```

```
        cur1.next = null;
        cur2.next = null;

        Node[] nodes = new Node[]{head1, head2};
        return nodes;
    }

    /**
     * 反转链表
     * @param head
     * @return
     */
    public static Node reverseList(Node head){
        Node pre = null;
        Node next = null;
        while(head != null){
            next = head.next;
            head.next = pre;
            pre = head;
            head = next;
        }
        return pre;
    }

    /**
     * 合并两个有序链表
     * @param head1
```

```
     * @param head2
     * @return
     */
    public static Node CombineList(Node head1, Node head2){
        if(head1 == null || head2 == null){
            return head1 != null ? head1 : head2;
        }
        Node head = head1.value < head2.value ? head1 : head2;
        Node cur1 = head == head1 ? head1 : head2;
        Node cur2 = head == head1 ? head2 : head1;
        Node pre = null;
        Node next = null;
        while(cur1 != null && cur2 != null){
            if(cur1.value <= cur2.value){//这里一定要有=，否则一旦 cur1 的 value
和 cur2 的 value 相等的话，下面的 pre.next 会出现空指针异常
                pre = cur1;
                cur1 = cur1.next;
            }else{
                next = cur2.next;
                pre.next = cur2;
                cur2.next = cur1;
                pre = cur2;
                cur2 = next;
            }
        }
        pre.next = cur1 == null ? cur2 : cur1;

        return head;
```

```
        }


    }
```

4、bucket 如果用链表存储，它的缺点是什么？

考察点：链表

参考回答：

①查找速度慢，因为查找时，需要循环链表访问

②如果进行频繁插入和删除操作，会导致速度很慢。

5、如何判断链表检测环

考察点：链表

参考回答：

单链表有环，是指单链表中某个节点的 next 指针域指向的是链表中在它之前的某一个节点，这样在链表的尾部形成一个环形结构。

// 链表的节点结构如下 typedef struct node { int data; struct node *next; } NODE;

最常用方法：定义两个指针，同时从链表的头节点出发，一个指针一次走一步，另一个指针一次走两步。如果走得快的指针追上了走得慢的指针，那么链表就是环形链表；如果走得快的指针走到了链表的末尾（next 指向 NULL）都没有追上第一个指针，那么链表就不是环形链表。

通过使用 STL 库中的 map 表进行映射。首先定义 map<NODE*, int> m; 将一个 NODE* 指针映射成数组的下标，并赋值为一个 int 类型的数值。然后从链表的头指针开始往后遍历，每次遇到一个指针 p，就判断 m[p] 是否为 0。如果为 0，则将 m[p]赋值为 1，表示该节点第一次访问；而如果 m[p]的值为 1，则说明这个节点已经被访问过一次了，于是就形成了环。

5、数组

1、寻找一数组中前 K 个最大的数

考察点：数组

参考回答：

```
public int findKthLargest(int[] nums, int k) {
if (k < 1 || nums == null) {
    return 0;
```

```
    }

    return getKth(nums.length - k +1, nums, 0, nums.length - 1);
    }

    public int getKth(int k, int[] nums, int start, int end) {

    int pivot = nums[end];

    int left = start;
    int right = end;

    while (true) {

        while (nums[left] < pivot && left < right) {
            left++;
        }

        while (nums[right] >= pivot && right > left) {
            right--;
        }

        if (left == right) {
            break;
        }

        swap(nums, left, right);
```

```
    }


    swap(nums, left, end);


    if (k == left + 1) {
        return pivot;
    } else if (k < left + 1) {
        return getKth(k, nums, start, left - 1);
    } else {
        return getKth(k, nums, left + 1, end);
    }
    }


    public void swap(int[] nums, int n1, int n2) {
    int tmp = nums[n1];
    nums[n1] = nums[n2];
    nums[n2] = tmp;
    }
```

2、求一个数组中连续子向量的最大和

考察点：数组

参考回答：

```
public int maxSubArray(int[] nums) {
        int sum = 0;
        int maxSum = Integer.MIN_VALUE;
        if (nums == null || nums.length == 0) {
            return sum;
        }
```

```
        for (int i = 0; i < nums.length; i++) {
            sum += nums[i];
            maxSum = Math.max(maxSum, sum);
            if (sum < 0) {
                sum = 0;
            }
        }
        return maxSum;
    }
```

3、找出数组中和为 S 的一对组合，找出一组就行

考察点：数组

参考回答：

```
public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        int[] a = new int[2];
        map.put(nums[0], 0);
        for (int i = 1; i < nums.length; i++) {
            if (map.containsKey(target - nums[i])) {
                a[0] = map.get(target - nums[i]);
                a[1] = i;
                return a;
            } else {
                map.put(nums[i], i);
            }
        }
        return a;
    }
```

4、一个数组，除一个元素外其它都是两两相等，求那个元素?

考察点：数组

参考回答：

```
  public static int find1From2(int[] a){
                int len = a.length, res = 0;
                for(int i = 0; i < len; i++){
                        res = res ^ a[i];
                }
                return res;
}
```

5、算法题：将一个二维数组顺时针旋转 90 度，说一下思路。

考察点：数组

参考回答：

```
public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n/2; i++) {
            for (int j = i; j < n-1-i; j++) {
                int temp = matrix[i][j];
                matrix[i][j] = matrix[n-1-j][i];
                matrix[n-1-j][i] = matrix[n-1-i][n-1-j];
                matrix[n-1-i][n-1-j] = matrix[j][n-1-i];
                matrix[j][n-1-i] = temp;
            }
        }
}
```

6、排序

1、排序算法知道哪些，时间复杂度是多少，解释一下快排？

考察点：快排

参考回答：

排序方式	时间复杂度			空间复杂度	稳定性	复杂性
	平均情况	最坏情况	最好情况			
插入排序	$O(n^2)$	$O(n^2)$	O(n)	O(1)	稳定	简单
希尔排序	$O(n^{1.3})$			O(1)	不稳定	较复杂
冒泡排序	$O(n^2)$	$O(n^2)$	O(n)	O(1)	稳定	简单
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	$O(\log_2 n)$	不稳定	较复杂
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定	简单
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	不稳定	较复杂
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(n)	稳定	较复杂
基数排序	O(d(n+r))	O(d(n+r))	O(d(n+r))	O(r)	稳定	较复杂

快排：快速排序有两个方向，左边的 i 下标一直往右走（当条件 a[i] <= a[center_index]时），其中 center_index 是中枢元素的数组下标，一般取为数组第 0 个元素。

而右边的 j 下标一直往左走（当 a[j] > a[center_index]时）。

如果 i 和 j 都走不动了，i <= j，交换 a[i]和 a[j],重复上面的过程，直到 i>j。交换 a[j]和 a[center_index]，完成一趟快速排序。

2、如何得到一个数据流中的中位数？

考察点：排序

参考回答：

数据是从一个数据流中读出来的，数据的数目随着时间的变化而增加。如果用一个数据容器来保存从流中读出来的数据，当有新的数据流中读出来时，这些数据就插入到数据容器中。

数组是最简单的容器。如果数组没有排序，可以用 Partition 函数找出数组中的中位数。在没有排序的数组中插入一个数字和找出中位数的时间复杂度是 O(1)和 O(n)。

我们还可以往数组里插入新数据时让数组保持排序，这是由于可能要移动 O(n)个数，因此需要 O(n)时间才能完成插入操作。在已经排好序的数组中找出中位数是一个简单的操作，只需要 O(1)时间即可完成。

排序的链表时另外一个选择。我们需要 O(n)时间才能在链表中找到合适的位置插入新的数据。如果定义两个指针指向链表的中间结点（如果链表的结点数目是奇数，那么这两个指针指向同一个结点），那么可以在 O（1）时间得出中位数。此时时间效率与及基于排序的数组的时间效率一样。

如果能够保证数据容器左边的数据都小于右边的数据，这样即使左、右两边内部的数据没有排序，也可以根据左边最大的数及右边最小的数得到中位数。如何快速从一个容器中找出最大

数？用最大堆实现这个数据容器，因为位于堆顶的就是最大的数据。同样，也可以快速从最小堆中找出最小数。

因此可以用如下思路来解决这个问题：用一个最大堆实现左边的数据容器，用最小堆实现右边的数据容器。往堆中插入一个数据的时间效率是 O(logn)。由于只需 O(1)时间就可以得到位于堆顶的数据，因此得到中位数的时间效率是 O(1)。

3、堆排序的原理是什么？

考察点：堆排序

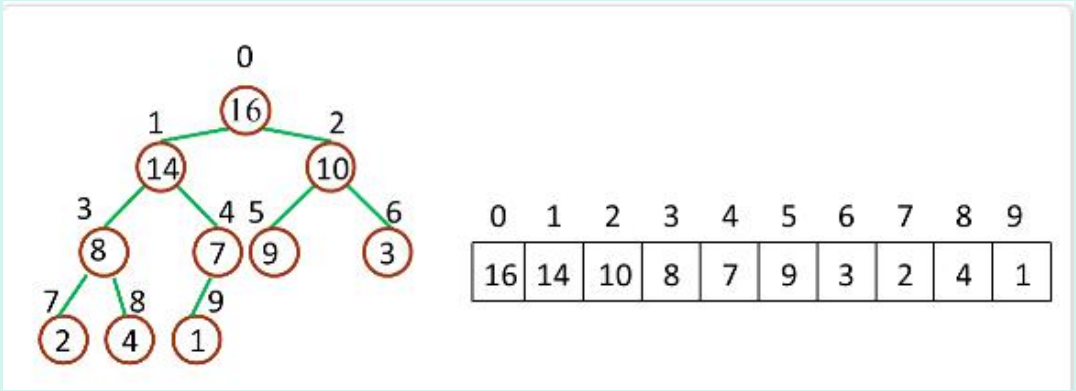
参考回答：

堆排序就是把最大堆堆顶的最大数取出，将剩余的堆继续调整为最大堆，再次将堆顶的最大数取出，这个过程持续到剩余数只有一个时结束。在堆中定义以下几种操作：

（1）最大堆调整（Max-Heapify）：将堆的末端子节点作调整，使得子节点永远小于父节点。

（2）创建最大堆（Build-Max-Heap）：将堆所有数据重新排序，使其成为最大堆。

（3）堆排序（Heap-Sort）：移除位在第一个数据的根节点，并做最大堆调整的递归运算



4、归并排序的原理是什么？

考察点：归并排序

参考回答：

（1）归并排序是建立在归并操作上的一种有效的排序算法。该算法是采用分治法（Divide and Conquer）的一个非常典型的应用。

（2）首先考虑下如何将将二个有序数列合并。这个非常简单，只要从比较二个数列的第一个数，谁小就先取谁，取了后就在对应数列中删除这个数。然后再进行比较，如果有数列为空，那直接将另一个数列的数据依次取出即可。

（3）解决了上面的合并有序数列问题，再来看归并排序，其的基本思路就是将数组分成二组A，B，如果这二组组内的数据都是有序的，那么就可以很方便的将这二组数据进行排序。如何让这二组组内数据有序了？

可以将 A，B 组各自再分成二组。依次类推，当分出来的小组只有一个数据时，可以认为这个小组组内已经达到了有序，然后再合并相邻的二个小组就可以了。这样通过先递归的分解数列，再合并数列就完成了归并排序。

5、排序都有哪几种方法？请列举出来。

考察点：排序

参考回答：

排序的方法有：插入排序（直接插入排序、希尔排序），交换排序（冒泡排序、快速排序），选择排序（直接选择排序、堆排序），
归并排序，分配排序（箱排序、基数排序）
快速排序的伪代码。
/ /使用快速排序方法对 a[0 :n- 1]排序
从 a[0 :n- 1]中选择一个元素作为 m i d d l e，该元素为支点
把余下的元素分割为两段 left 和 r i g h t，使得 l e f t 中的元素都小于等于支点，而 right 中的元素都大于等于支点
递归地使用快速排序方法对 left 进行排序
递归地使用快速排序方法对 right 进行排序
所得结果为 l e f t + m i d d l e + r i g h t

6、如何用 java 写一个冒泡排序？

考察点：冒泡排序

参考回答：

```
import java.util.Comparator;
/**
 * 排序器接口(策略模式: 将算法封装到具有共同接口的独立的类中使得它们可以相互替换)
 */
public interface Sorter {
    /**
     * 排序
     * @param list 待排序的数组
     */
    public <T extends Comparable<T>> void sort(T[] list);
    /**
     * 排序
     * @param list 待排序的数组
     * @param comp 比较两个对象的比较器
     */
    public <T> void sort(T[] list, Comparator<T> comp);
}
import java.util.Comparator;
/**
 * 冒泡排序
 *
 */
public class BubbleSorter implements Sorter {
```

```
        @Override
        public <T extends Comparable<T>> void sort(T[] list) {
                boolean swapped = true;
                for (int i = 1, len = list.length; i < len && swapped; ++i) {
                        swapped = false;
                        for (int j = 0; j < len - i; ++j) {
                                if (list[j].compareTo(list[j + 1]) > 0) {
                                        T temp = list[j];
                                        list[j] = list[j + 1];
                                        list[j + 1] = temp;
                                        swapped = true;
                                }
                        }
                }
        }

        @Override
        public <T> void sort(T[] list, Comparator<T> comp) {
                boolean swapped = true;
                for (int i = 1, len = list.length; i < len && swapped; ++i) {
                        swapped = false;
                        for (int j = 0; j < len - i; ++j) {
                                if (comp.compare(list[j], list[j + 1]) > 0) {
                                        T temp = list[j];
                                        list[j] = list[j + 1];
                                        list[j + 1] = temp;
                                        swapped = true;
                                }
                        }
                }
```

7、堆与栈

1、堆与栈的不同是什么？

考察点：堆，栈

参考回答：

（1）Java 的堆是一个运行时数据区，类的对象从中分配空间。通过比如：new 等指令建立，不需要代码显式的释放，由垃圾回收来负责。

优点：可以动态地分配内存大小，垃圾收集器会自动回收垃圾数据。

缺点：由于其优点，所以存取速度较慢。

（2）栈：

其数据项的插入和删除都只能在称为栈顶的一端完成，后进先出。栈中存放一些基本类型的变量 和 对象句柄。

优点：读取数度比堆要快，仅次于寄存器，栈数据可以共享。

缺点：比堆缺乏灵活性，存在栈中的数据大小与生存期必须是确定的。

举例：

String 是一个特殊的包装类数据。可以用：

```
String str = new String("csdn");
String str = "csdn";
```

两种的形式来创建，第一种是用 new()来新建对象的，它会在存放于堆中。每调用一次就会创建一个新的对象。而第二种是先在栈中创建一个对 String 类的对象引用变量 str，然后查找栈中有没有存放"csdn"，如果没有，则将"csdn"存放进栈，并令 str 指向”abc”，如果已经有”csdn” 则直接令 str 指向“csdn”。

2、heap 和 stack 有什么区别。

考察点：堆与栈

参考回答：

栈是一种线形集合，其添加和删除元素的操作应在同一段完成。栈按照后进先出的方式进行处理。堆是栈的一个组成元素。

3、解释内存中的栈(stack)、堆(heap)和静态区(static area)的用法。

考察点：堆栈

参考回答：

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区中。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，理论上整个内存没有被其他进程使用的空间甚至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");
```

上面的语句中变量 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区。

8、队列

1、什么是 Java 优先级队列(Priority Queue)？

考察点：队列

参考回答：

PriorityQueue 是一个基于优先级堆的无界队列，它的元素是按照自然顺序(natural order)排序的。在创建的时候，我们可以给它提供一个负责给元素排序的比较器。PriorityQueue 不允

许 null 值，因为他们没有自然顺序，或者说他们没有任何的相关联的比较器。最后，PriorityQueue 不是线程安全的，入队和出队的时间复杂度是 O(log(n))。

9、高级算法

1、题目：

Design and implement a data structure for Least Frequently Used (LFU) cache. It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least frequently used item before inserting a new item. For the purpose of this problem, when there is a tie (i.e., two or more keys that have the same frequency), the least recently used key would be evicted.

Could you do both operations in O(1) time complexity?

考察点：LFU Cache

参考回答：

```
public class LFUCache {

private class Node{

    int value;

    ArrayList<Integer> set;

    Node prev;

    Node next;

    public Node (int value ){

        this.value = value;

        this.set = new ArrayList<Integer> ();

        this.prev = null;
```

```
            this.next = null;
        }
    }
    private class item{
        int key;
        int value;
        Node parent ;
        public item(int key ,int value, Node parent){
             this.key = key ;
             this.value = value;
             this.parent  = parent;
        }
    }
     private HashMap<Integer, item> map;
    private  Node head,tail;
    private  int capacity;
    // @param capacity, an integer
    public LFUCache(int capacity) {
        // Write your code here
        this.capacity = capacity;
        this.map = new HashMap <Integer,item> ();
        this.head = new Node (0);
        this.tail = new Node(Integer.MAX_VALUE);
        head.next = tail;
        tail.prev = head;
```

```
    }


    // @param key, an integer
    // @param value, an integer
    // @return nothing
    public void set(int key, int value) {
        // Write your code here
       if (get(key) != -1 ) {
          map.get(key).value = value;
          return ;
       }
       if (map.size() == capacity ){
           getLFUitem();
       }

       Node newpar = head.next;
       if ( newpar.value != 1){
           newpar = getNewNode(1,head,newpar);
       }
        item curitem = new item(key,value,newpar);
        map.put(key,curitem);
        newpar.set.add(key);
        return;
    }


    public int get(int key) {
        // Write your code here
```

```
        if (!map.containsKey(key)){
            return -1;
        }
        item cur = map.get(key);
        Node curpar = cur.parent;
        if (curpar.next.value == curpar.value + 1){
            cur.parent= curpar.next;
            cur.parent.set.add(key);
        }else{
            Node newpar =getNewNode(curpar.value + 1,curpar,curpar.next);
            cur.parent = newpar;
            newpar.set.add(key);
        }
         curpar.set.remove(new Integer(key));
         if(curpar.set.isEmpty()){
             deleteNode(curpar);
         }
        return cur.value;
    }
    private Node getNewNode (int value ,Node prev , Node next){
        Node temp = new Node(value);
        temp.prev = prev;
        temp.next = next;
        prev.next = temp;
        next.prev = temp;
        return temp;
    }
```

```
    private void deleteNode(Node temp){
        temp.prev.next = temp.next;
        temp.next.prev = temp.prev;
        return ;
    }
    private void getLFUitem(){
        Node temp = head.next;
         int LFUkey = temp.set.get(0);
        temp.set.remove(0);
        map.remove(LFUkey);
        if (temp.set.isEmpty()){
            deleteNode(temp);
        }
        return;
    }
```

2、id 全局唯一且自增，如何实现？

考察点：SnowFlake 雪花算法

参考回答：

SnowFlake 雪花算法

雪花 ID 生成的是一个 64 位的二进制正整数，然后转换成 10 进制的数。64 位二进制数由如下部分组成：

snowflake id 生成规则

1 位标识符：始终是 0，由于 long 基本类型在 Java 中是带符号的，最高位是符号位，正数是 0，负数是 1，所以 id 一般是正数，最高位是 0。

41 位时间戳：41 位时间截不是存储当前时间的时间截，而是存储时间截的差值（当前时间截 - 开始时间截)得到的值，这里的的开始时间截，一般是我们的 id 生成器开始使用的时间，由我们程序来指定的。

10 位机器标识码：可以部署在 1024 个节点，如果机器分机房（IDC）部署，这 10 位可以由 5 位机房 ID + 5 位机器 ID 组成。

12 位序列：毫秒内的计数，12 位的计数顺序号支持每个节点每毫秒(同一机器，同一时间截)产生 4096 个 ID 序号

优点

简单高效，生成速度快。

时间戳在高位，自增序列在低位，整个 ID 是趋势递增的，按照时间有序递增。

灵活度高，可以根据业务需求，调整 bit 位的划分，满足不同的需求。

缺点

依赖机器的时钟，如果服务器时钟回拨，会导致重复 ID 生成。

在分布式环境上，每个服务器的时钟不可能完全同步，有时会出现不是全局递增的情况。

3、如何设计算法压缩一段 URL？

考察点：MD5 加密算法

参考回答：

该算法主要使用 MD5 算法对原始链接进行加密（这里使用的 MD5 加密后的字符串长度为 32 位），然后对加密后的字符串进行处理以得到短链接的地址。

4、为什么要设计后缀表达式，有什么好处？

考察点：逆波兰表达式

参考回答：

后缀表达式又叫逆波兰表达式，逆波兰记法不需要括号来标识操作符的优先级。

5、LRU 算法的实现原理？

考察点：LRU 算法

参考回答：

①LRU（Least recently used，最近最少使用）算法根据数据的历史访问记录来进行淘汰数据，其核心思想是“如果数据最近被访问过，那么将来被访问的几率也很高”，反过来说“如果数据最近这段时间一直都没有访问,那么将来被访问的概率也会很低”，两种理解是一样的；常用于页面置换算法，为虚拟页式存储管理服务。

②达到这样一种情形的算法是最理想的:每次调换出的页面是所有内存页面中最迟将被使用的；这可以最大限度的推迟页面调换，这种算法，被称为理想页面置换算法。可惜的是，这种算法是无法实现的。
为了尽量减少与理想算法的差距，产生了各种精妙的算法，最近最少使用页面置换算法便是其中一个。LRU 算法的提出，是基于这样一个事实：在前面几条指令中使用频繁的页面很可能在后面的几条指令中频繁使用。反过来说，已经很久没有使用的页面很可能在未来较长的一段时间内不

会被用到 。这个，就是著名的局部性原理——比内存速度还要快的 cache，也是基于同样的原理运行的。因此，我们只需要在每次调换时，找到最近最少使用的那个页面调出内存。

算法实现的关键

命中率：
当存在热点数据时，LRU 的效率很好，但偶发性的、周期性的批量操作会导致 LRU 命中率急剧下降，缓存污染情况比较严重。
复杂度：
实现起来较为简单。
存储成本：
几乎没有空间上浪费。
代价：
命中时需要遍历链表，找到命中的数据块索引，然后需要将数据移到头部。

九、设计模式

1、结构型模式

①代理模式

1、java 中有哪些代理模式？

考察点：代理模式

参考回答：

静态代理，动态代理，Cglib 代理。

2、如何实现动态代理

考察点：动态代理流程

参考回答：

Java 实现动态代理的大致步骤如下：

1.定义一个委托类和公共接口。

2.自己定义一个类（调用处理器类，即实现 InvocationHandler 接口），这个类的目的是指定运行时将生成的代理类需要完成的具体任务（包括 Preprocess 和 Postprocess），即代理类调用任何方法都会经过这个调用处理器类（在本文最后一节对此进行解释）。

3.生成代理对象（当然也会生成代理类），需要为他指定(1)委托对象(2)实现的一系列接口(3)调用处理器类的实例。因此可以看出一个代理对象对应一个委托对象，对应一个调用处理器实例。

4.Java 实现动态代理主要涉及以下几个类：

①java.lang.reflect.Proxy: 这是生成代理类的主类,通过 Proxy 类生成的代理类都继承了 Proxy 类，即 DynamicProxyClass extends Proxy。

②java.lang.reflect.InvocationHandler: 这里称他为"调用处理器"，他是一个接口，我们动态生成的代理类需要完成的具体内容需要自己定义一个类，而这个类必须实现 InvocationHandler 接口。

示例代码：

```
public final class $Proxy1 extends Proxy implements Subject{

private InvocationHandler h;

private $Proxy1(){}

public $Proxy1(InvocationHandler h){

this.h = h; }

public int request(int i){

Method method = Subject.class.getMethod("request", new Class[]{int.class}); //创建 method 对象

return (Integer)h.invoke(this, method, new Object[]{new Integer(i)}); //调用了 invoke 方法 } }
```

②适配器模式

1、IO 流熟悉吗，用的什么设计模式？

考察点：装饰模式，适配器模式

参考回答：

装饰模式和适配器模式

2、创建型模式

①单例模式

1、介绍一下单例模式？懒汉式的单例模式如何实现单例？

考察点：单例模式

参考回答：

定义：保证一个类仅有一个实例，并提供一个访问它的全局访问点。优点：单例类只有一个实例、共享资源，全局使用节省创建时间，提高性能。可以用静态内部实现，保证是懒加载就行了，就是使用才会创建实例对象。

3、行为型模式

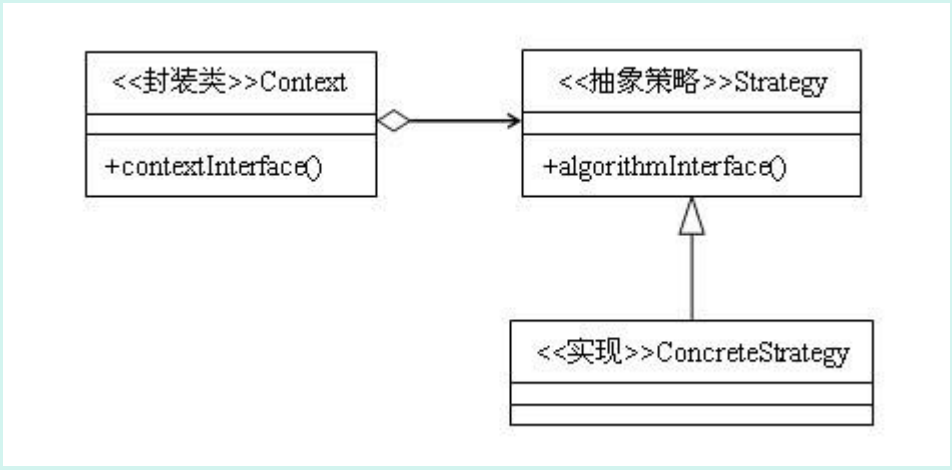
①策略模式

1、介绍一下策略模式？

考察点：策略模式

参考回答：

策略模式也叫政策模式，是一种行为型设计模式，是一种比较简单的设计模式。策略模式采用了面向对象的继承和多态机制。略模式适合使用在：1.多个类只有在算法或行为上稍有不同的场景。2.算法需要自由切换的场景。3.需要屏蔽算法规则的场景。 使用策略模式当然也有需要注意的地方，那么就是策略类不要太多，如果一个策略家族的具体策略数量超过 4 个，则需要考虑混合模式，解决策略类膨胀和对外暴露问题。在实际项目中，我们一般通过工厂方法模式来实现策略类的声明。



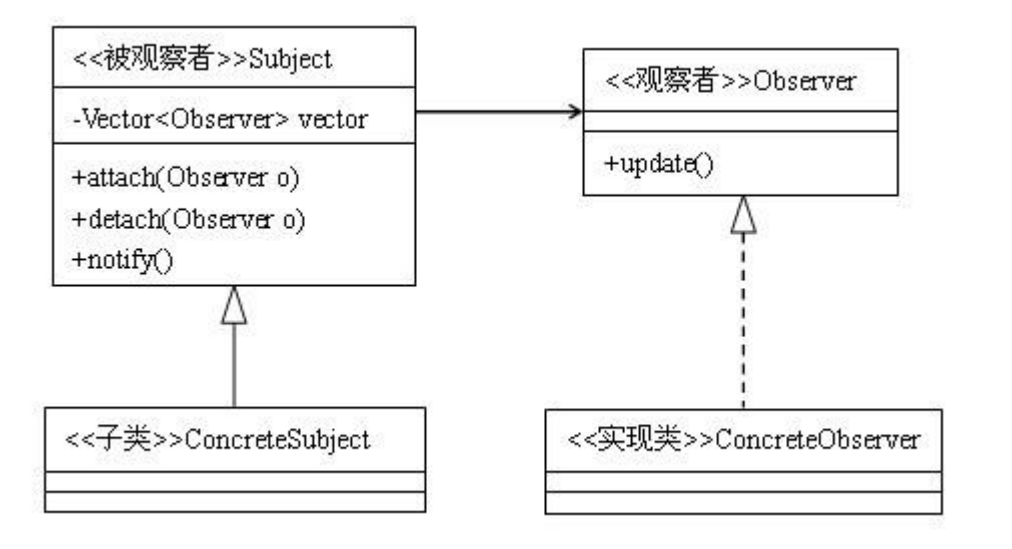
优点：算法可以自由切换。2.避免使用多重条件判断。3.扩展性良好。

②观察者模式

1、设计模式了解哪些，手写一下观察者模式？

考察点：观察者模式

参考回答：



观察者模式优点：

观察者模式在被观察者和观察者之间建立一个抽象的耦合。被观察者角色所知道的只是一个具体观察者列表，每一个具体观察者都符合一个抽象观察者的接口。被观察者并不认识任何一个具体观察者，它只知道它们都有一个共同的接口。由于被观察者和观察者没有紧密地耦合在一起，因此它们可以属于不同的抽象化层次。如果被观察者和观察者都被扔到一起，那么这个对象必然跨越抽象化和具体化层次。

观察者模式缺点：

如果一个被观察者对象有很多的直接和间接的观察者的话，将所有的观察者都通知到会花费很多时间。

如果在被观察者之间有循环依赖的话，被观察者会触发它们之间进行循环调用，导致系统崩溃。在使用观察者模式是要特别注意这一点。

如果对观察者的通知是通过另外的线程进行异步投递的话，系统必须保证投递是以自恰的方式进行的。

虽然观察者模式可以随时使观察者知道所观察的对象发生了变化，但是观察者模式没有相应的机制使观察者知道所观察的对象是怎么发生变化的。

4、模式汇总

1、说说你所熟悉或听说过的 j2ee 中的几种常用模式?及对设计模式的一些看法

考察点：J2EE 设计模式

参考回答：

Session Facade Pattern：使用 SessionBean 访问 EntityBean Message Facade Pattern：实现异步调用 EJB Command Pattern：使用 Command JavaBeans 取代 SessionBean，实现轻量级访问 Data Transfer Object Factory：通过 DTO Factory 简化 EntityBean 数据提供特性 Generic Attribute Access：通过 AttibuteAccess 接口简化 EntityBean 数据提供特性 Business

Interface：通过远程（本地）接口和 Bean 类实现相同接口规范业务逻辑一致性 E J B 架构的设计好坏将直接影响系统的性能、可扩展性、可维护性、组件可重用性及开发效率。项目越复杂，项目队伍越庞大则越能体现良好设计的重要性。

2、j2ee 常用的设计模式？说明工厂模式。

考察点：j2ee 设计模式

参考回答：

Java 中的 23 种设计模式：
Factory（工厂模式）， Builder（建造模式）， 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 Responsibleity（责任链模式）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。

3、开发中都用到了那些设计模式?用在什么场合?

考察点：设计模式

参考回答：

每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心。通过这种方式，你可以无数次地使用那些已有的解决方案，无需在重复相同的工作。主要用到了 MVC 的设计模式。用来开发 JSP/Servlet 或者 J2EE 的相关应用。简单工厂模式等。

4、简述一下你了解的 Java 设计模式

考察点：设计模式

参考回答：

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。
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）。

十、场景题

1、情景题：如果一个外卖配送单子要发布，现在有 200 个骑手都想要接这一单，如何保证只有一个骑手接到单子？
2、场景题：美团首页每天会从 10000 个商家里面推荐 50 个商家置顶，每个商家有一个权值，你如何来推荐？第二天怎么更新推荐的商家？
可以借鉴下 stackoverflow，视频网站等等的推荐算法。
3、场景题：微信抢红包问题
悲观锁，乐观锁，存储过程放在 mysql 数据库中。
4、场景题：1000 个任务，分给 10 个人做，你怎么分配，先在纸上写个最简单的版本，然后优化。
全局队列，把 1000 任务放在一个队列里面，然后每个人都是取，完成任务。
分为 10 个队列，每个人分别到自己对应的队列中去取务。
5、场景题：保证发送消息的有序性，消息处理的有序性。
6、如何把一个文件快速下发到 100w 个服务器
7、给每个组分配不同的 IP 段，怎么设计一种结构使的快速得知 IP 是哪个组的?
8、10 亿个数，找出最大的 10 个。
建议一个大小为 10 的小根堆。
9、有几台机器存储着几亿淘宝搜索日志，你只有一台 2g 的电脑，怎么选出搜索热度最高的十个搜索关键词？
10、分布式集群中如何保证线程安全？
11、给个淘宝场景，怎么设计一消息队列？
12、10 万个数，输出从小到大？
先划分成多个小文件，送进内存排序，然后再采用多路归并排序。
13、有十万个单词，找出重复次数最高十个？

十一、UML

1、请你谈一下 UML 中有哪些常用的图？

考察点：用例图

参考回答：

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）。

十二、惊喜福利

此面试题库将根据当下面试形式大数据随时更新，如果你已获得下载权限，那么你可以终身在牛币兑换中心里去兑换此面试题库的电子版，如果电子版有更新，会通过牛客站内信进行通知（前提是你已获得下载权限）。

牛币兑换中心：https://www.nowcoder.com/coin/index
还能兑换各种惊喜周边哦