技术问题：

1. 二叉树最大子树和

2. 判断一棵树是不是对称二叉树

3. RST 在 TCP 协议中的作用

4. TCP/IP 三次握手，四次挥手，（握手）协议的漏洞

TCP 三次握手：

① 主机 A 通过向主机 B 发送一个含有同步序列号的标志位的数据段给主机 B ，向主机 B 请求建立连接。

主机 A 告诉主机 B 两件事:我想要和你通信;你可以用哪个序列号作为起始数据段来回应我.

② 主机 B 收到主机 A 的请求后,用一个带有确认应答(ACK)和同步序列号(SYN)标志位的数据段响应主机 A。

主机 B 告诉主机 A 两件事:我已经收到你的请求了,你可以传输数据了;你要用哪伅序列号作为起始数据段来回应我

③ 主机 A 收到这个数据段后,再发送一个确认应答,确认已收到主机 B 的数据段。

三次握手的特点：没有应用层的数据；SYN 标志位只有在 TCP 建立连接时才会被置为 1，握手完成后 SYN 标志位被置 0。

三次握手缺陷：SYNF LOOD 攻击（一种常见的 DDos 攻击，拒绝服务攻击。通过网络服务所在的端口发送大量伪造原地址的攻击报文，发送到服务端，造成服务端上的半开连接队列被占满，从而阻止其他用户进行访。**它的数据报特征是大量 syn 包，并且缺少最后一步的 ACK 回复。**)。

原理：攻击者首先伪造地址，对服务器发起 syn 请求，服务器回应 syn+ACK，而真实的 IP 会认为我没有发送请求，不做回应，而服务端没有收到回应，服务器就不知道是否发送成功，默认情况下重试 5 次 syn_retries，这样的话，对于服务器内存和带宽有很大的消耗。

解决办法：1.无效连接监控 2.延缓 TCB 方法 3.SYN Cookie

【more】

TCP 四次挥手：

① 当主机 A 完成数据传输后,将控制位 FIN 置 1,提出停止 TCP 连接的请求

② 主机 B 收到 FIN 后对其作出响应,确认这一方向上的 TCP 连接将关闭,将 ACK 置 1

③ 由 B 端再提出反方向的关闭请求，将 FIN 置 1

④ 主机 A 对主机 B 的请求进行确认，将 ACK 置 1，双方向的关闭结束

名词	含义
ACK	TCP 报头的控制位之一，对数据进行确认.确认由目的端发出，用它来告诉发送端这个序列号之前的数据段都收到了.
SYN	同步序列号，TCP 建立连接时将这个位置 1
FIN	发送端完成发送任务位，当 TCP 完成数据传输需要断开时，提出断开连接的一方将这位置 1

5. 求一个字符串最大对称子串；求最长不重复子串

6. Http 协议解析（URL 地址组成）

URL(Uniform Resource Locator) 地址用于描述一个网络上的资源，基本格式如下

```
schema://host[:port#]/path/.../[?query-string][#anchor]
```

scheme	指定低层使用的协议(例如：http, https, ftp)
host	HTTP服务器的IP地址或者域名
port#	HTTP服务器的默认端口是80，这种情况下端口号可以省略。如果使用了别的端口，必须指明，例如 http://www.cnblogs.com:8080/
path	访问资源的路径
query-string	发送给http服务器的数据
anchor-	锚

http 协议是无状态的，同一个客户端的这次请求和上次请求是没有对应关系，对 http 服务器来说，它并不知道这两个请求来自同一个客户端。为了解决这个问题，Web 程序引入了 Cookie 机制来维护状态。【more】

7. RPC 过程

8. 几种排序算法代码以及稳定性和复杂度,堆排序过程，快排和堆排哪个更好

9. 集合的结构（哪些是线程安全的）

10. 线程安全的锁的类型

11. 线程池的原理，里面的几个参数。如何自己实现线程池，设计一个动态大小的线程池，如何设计，应该有哪些方法？调度的线程池怎么对线程进行的回收？如果线程池的线程不够会出现什么问题，shcedule 的线程池实现是怎么处理这个问题的。

12.事务 ACID 的意思

ACID: 原子性(Atomicity)、一致性(Consistency)、隔离性或独立性(Isolation)和持久性(Durabilily)。

1. **原子性**。一个事务要么全部执行，要么不执行。
2. **一致性**。事务的运行不改变数据库中数据的一致性,即数据路的完整性约束没有被破坏。
3. **独立性**。两个以上的事务不会出现交错执行的状态，因为这可能会导致数据不一致。一个事务所做的修改在最终提交以前，对其他事务是不可见的。
4. **持久性**。事务运行成功后，就系统的更新是永久的，不会无缘无故回滚。

【Detail】

13.Mysql 索引，Myisam 与 innodb 哪个是锁表，哪个是锁行

14.GC 算法，什么时候会发生 Full GC

15.GC 机制，串行还是并行，CMS 收集器过程

16.一个 10 万条数据的商品表，100 万条数据的订单表，如何在订单表中查出商品表中存在的商品的 top10，内存 256M

17.求数组第二大的数，数组中最长递减子数组长度，把数组中 0 元素全部移到数组末尾

18.并行和并发的区别；线程与进程的区别；同步与异步的区别

并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔发生

一个程序运行至少一个进程，一个进程里面至少包含一个线程，线程是进程的组成部分。【more】

同步交互：指发送一个请求,需要等待返回,然后才能够发送下一个请求，有个等待过程；

异步交互：指发送一个请求,不需要等待返回,随时可以再发送下一个请求，即不需要等待。

19.创建线程的几种方式；线程同步有哪些方式

创建线程的三种方式

① 继承 Thread 类创建线程类

② 通过 Runnable 接口创建线程类

③ 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

线程同步的方法

① 同步方法

② 同步代码块

③ wait + notify

④ 使用特殊域变量（volatile）

⑤ 使用重入锁

ReentrantLock 类是可重入、互斥、实现了 Lock 接口的锁，它与使用 synchronized 方法和快具有相同的基本行为和语义，并且扩展了其能力。

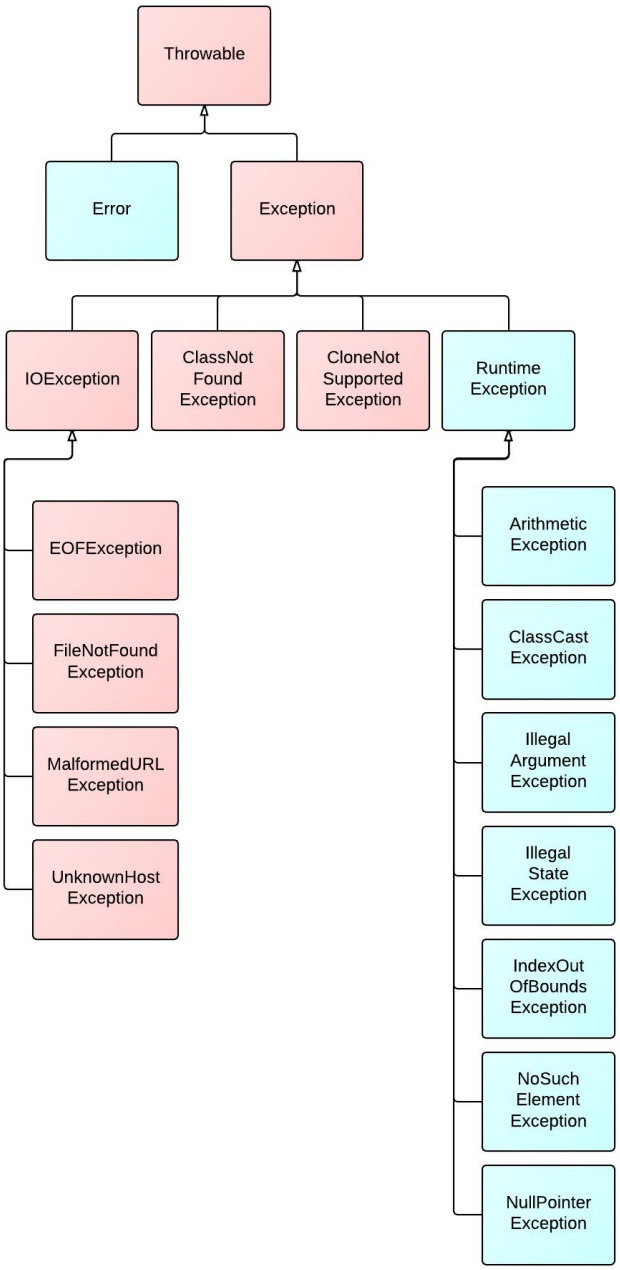
⑥ 使用局部变量

使用 ThreadLocal 管理变量，则每一个使用该变量的线程都获得该变量的副本，副本之间相互独立，这样每一个线程都可以随意修改自己的变量副本，而不会对其他线程产生影响。

⑦ 使用阻塞队列

BlockingQueue<E>

20. 异常的结构 ； try，catch，finally，finally 是不是一定会执行，如果 try 中有 system.exit(0)呢？



在后台线程中，finally 不一定会执行。

//thread.setDaemon(true); 将线程作为后台线程执行

//当最后一个非后台线程终止时，后台线程会“突然”终止

System.exit(0)会导致 VM 退出，finally 就不会执行了。

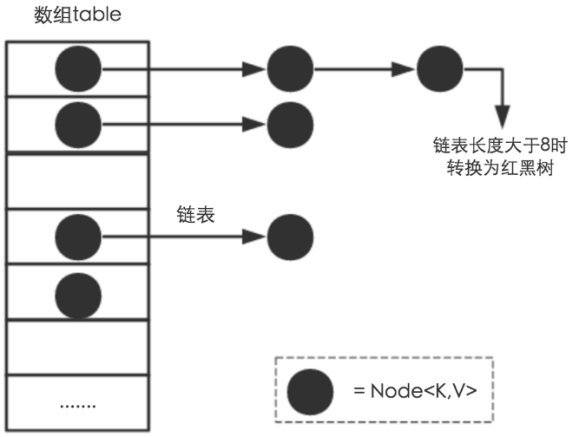
21. 服务器怎么记住客户端，session 原理

22. Linkedlist 与 ArrayList 区别，有 200 万个字符串，查找某个字符串，用什么？

23. Sysnchronized，以及它的几种用法。A 和 B 方法都用 sysnchronized 修饰，C 和 D 方法是锁住.class，两个线程持有同一个对象，访问这些方法有什么影响。Synchronized 与 Lock 的区别。
24. 线程池作用，maxpoolsize，coolpoolsize，keepalivetime 的含义
25. I/O 流。有一个文件，每行一个字符串，读出全是数字的一行，写程序实现
26. 数据库的范式，举例说明
27. Java 内存模型，stackoverflow 怎么造成的
28. 求一个字符串的全排列
29. Mysql 索引，聚集索引与非聚集索引的区别
30. 一个 SQL 语句查询很慢怎么办？

31. HashMap 的实现原理；HashMap 的加载因子怎么考虑的；向 HashMap 中添加 100 个元素，HashMap 应初始化为多少；HashMap 与 Hashtable 区别

存储结构：数组+链表+红黑树



HashMap 是用哈希表来存储的，用了链地址法来解决哈希冲突（*开放地址法也可以解决该问题*）。

链地址法，简单来说，就是数组加链表的结合。在每个数组元素上都一个链表结构，当数据被 Hash 后，得到数组下标，把数据放在对应下标元素的链表上。

减小 Hash 碰撞的概率：扩容机制和好的 Hash 算法。

默认构造函数初始化字段：

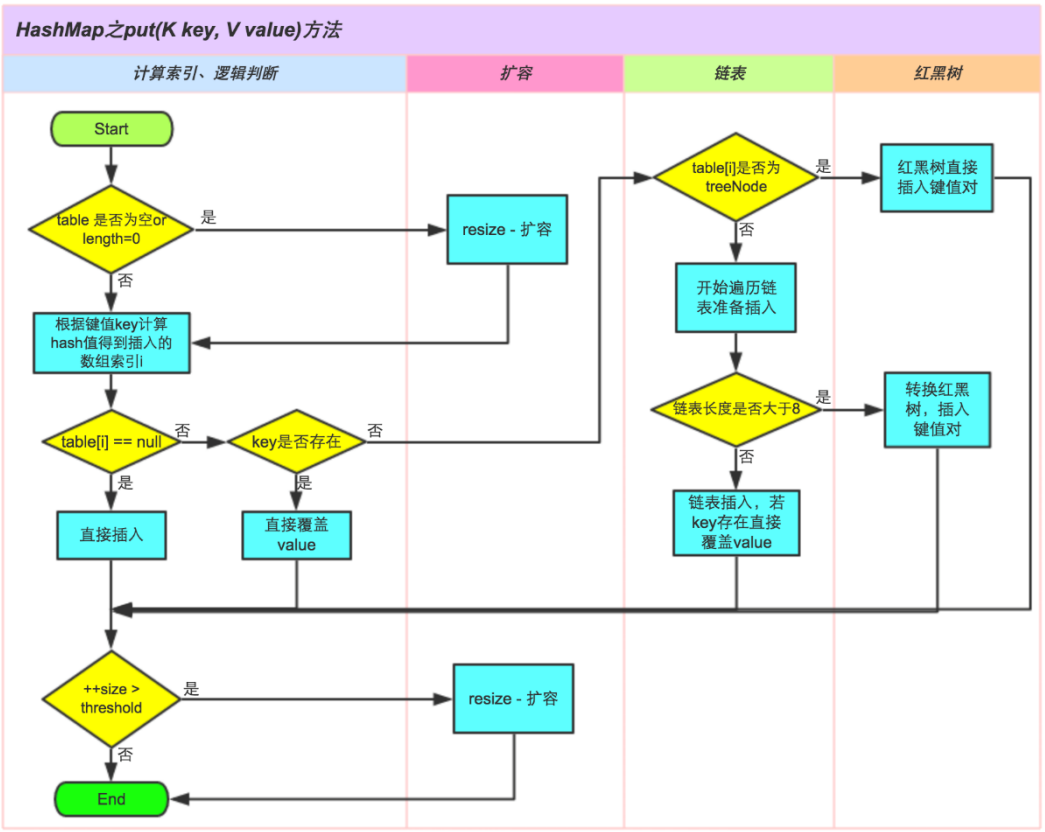
```
1.  int threshold;              // 所能容纳的 key-value 对极限
2.  final float loadFactor;     // 负载因子,可以大于 1
3.  int modCount;
```

```
4.  int size;
```

Node[] table 的初始化长度 length(默认值是 16)，Load factor 为负载因子(默认值是 0.75)，threshold 是 HashMap 所能容纳的最大数据量的 Node(键值对)个数，超过这个数据就重新扩容,容量是之前的两倍。threshold = length * Load factor。也就是说，在数组定义好长度之后，负载因子越大，所能容纳的键值对个数越多。

HashMap 中，哈希桶数组 table 的长度 length 大小必须为 2 的 n 次方（一定是合数）。

HashMap 的 put 方法：



扩容机制：

Java7 扩容时，遍历每个节点，并重新 hash 获得当前数组的位置并添加到链表中；Java8 进一步做了优化，将元素的 hash 和旧数组的大小（大小为 2 次幂）做与运算，为 0 则表示数组位置不变，不为 0 则表示需要移位，新位置为原先位置+旧数组的小大（新数组大小为旧数组翻倍），并将当前链表拆分为两个链表，一个链表放到原先位置，一个链路放到新位置，效率比 Java7 高。

其实在 Java 中，无论你的 HashMap(x)中的 x 设置为多少，HashMap 的大小都是 2^n（这么做是为了降低哈希碰撞的概率）。如果 x=100，那么 HashMap 的初

始大小应该是 100/0.75=133.333，即哈希桶大小 134，再次回到哈希碰撞问题，HashMap 应该设置的大小为 256。【参考来源】

加载因子的考虑：

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，它衡量的是一个散列表的空间的使用程度，负载因子越大表示散列表的装填程度越高，反之愈小。

对于使用链表法的散列表来说，查找一个元素的平均时间是 O(1+a)，因此如果负载因子越大，对空间的利用更充分，然而后果是查找效率的降低；如果负载因子太小，那么散列表的数据将过于稀疏，对空间造成严重浪费。系统默认负载因子为 0.75，一般情况下我们是无需修改的。

为什么是 0.75：从源码注释来看，在理想情况下,使用随机哈希码,节点出现的频率在 hash 桶中遵循**泊松分布**，同时给出了桶中元素个数和概率的对照表。从表中可以看到当桶中元素到达 8 个的时候，概率已经变得非常小，也就是说用 0.75 作为加载因子，每个碰撞位置的链表长度超过 8 个是几乎不可能的。

HashMap 与 Hashtable 的区别：

	HashMap	Hashtable
直系父类	AbstractMap	Dictionary
是否同步	否	是
K,V 可否为 null	是	否

HashMap 是 Hashtable 的轻量级实现，效率要高于 Hashtable。

HashMap 的迭代器(Iterator)是 fail-fast 迭代器，而 Hashtable 的 enumerator 迭代器不是 fail-fast 的。(当有其它线程改变了 HashMap 的结构（增加或者移除元素），将会抛出 ConcurrentModificationException，但迭代器本身的 remove()方法移除元素则不会抛出 ConcurrentModificationException 异常。但这并不是一个一定发生的行为，要看 JVM。这条同样也是 Enumeration 和 Iterator 的区别。)

HashMap 不能保证随着时间的推移 Map 中的元素次序是不变的。

HashMap 可以通过以下语句实现同步：

```
1.  Map m = Collections.synchronizeMap(hashMap);
```

【more】

32.类与类之间的关系

① **继承关系**

一个类（称为子类、子接口）继承另外的一个类（称为父类、父接口）

② **实现关系**

一个 class 类实现 interface 接口（可以是多个）

③ **依赖关系**

一个类 A 使用到了另一个类 B,主要表现形式为 B 为 A 构造器或方法中的局部变量、方法或构造器的参数、方法的返回值。

④ **关联关系**

是类与类之间的联接，它使一个类知道另一个类的属性和方法。关联可以是双向的，也可以是单向的。一般使用成员变量来实现。

⑤ **聚合关系**

关联关系的一种特例，它体现的是整体与部分的关系，即 has-a 的关系。两个雷处在不平等层次上，一个代表整体，一个代表部分。

⑥ **组合关系**

关联关系的一种特例，它体现的是一种 contains-a 的关系，这种关系比聚合更强，也称为强聚合。也表示类之间整体和部分的关系，但是组合关系中部分和整体具有统一的生存期。一旦整体对象不存在，部分对象也将不存在。部分对象与整体对象之间具有共生死的关系。

#通俗的例子#

你和你的心脏之间是 composition 关系(心脏只属于自己)

你和你买的书之间是 aggregation 关系（书可能是别人的）

你和你的朋友之间是 association 关系

【UML 参考】

33.设计原则，常用设计模式的类图

34.抽象类与接口的区别

抽象类是用来捕捉子类的通用特性的 。它不能被实例化，只能被用作子类的超类，比如 JDK 中的 GenericServlet。

接口是抽象方法的集合。接口像是契约模式，如果实现了接口，就必须确保实现这些方法。

抽象类和接口的对比

参数	抽象类	接口
默认的方法实现	它可以有默认的方法实现	接口完全是抽象的。它根本不存在方法的实现
实现	子类使用**extends**关键字来继承抽象类。如果子类不是抽象类的话，它需要提供抽象类中所有声明的方法的实现。	子类使用关键字**implements**来实现接口。它需要提供接口中所有声明的方法的实现
构造器	抽象类可以有构造器	接口不能有构造器
与正常Java类的区别	除了你不能实例化抽象类之外，它和普通Java类没有任何区别	接口是完全不同的类型
访问修饰符	抽象方法可以有**public**、**protected**和**default**这些修饰符	接口方法默认修饰符是**public**。你不可以使用其它修饰符。
main方法	抽象方法可以有main方法并且我们可以运行它	接口没有main方法，因此我们不能运行它。
多继承	抽象方法可以继承一个类和实现多个接口	接口只可以继承一个或多个其它接口
速度	它比接口速度要快	接口是稍微有点慢的，因为它需要时间去寻找在类中实现的方法。
添加新方法	如果你往抽象类中添加新的方法，你可以给它提供默认的实现。因此你不需要改变你现在的代码。	如果你往接口中添加方法，那么你必须改变实现该接口的类。

Java8 新增了接口的默认方法和类方法。

35. 重载与重写的区别

重写与重载之间的区别

区别点	重载方法	重写方法
参数列表	必须修改	一定不能修改
返回类型	可以修改	一定不能修改
异常	可以修改	可以减少或删除，一定不能抛出新的或者更广的异常
访问	可以修改	一定不能做更严格的限制（可以降低限制）

重写 Overriding 是实现多态必须的步骤；由于方法调用的参数是编译器确定

的，因此重载 Overloading 发生在编译器。

36. JVM 内存模型，memory analyer，jstack
37. 如何查看内存泄漏
38. ClassNotFoundException 与 NoClassDefFounderror 区别
39. AOP 实现原理
40. Maven 了解，主要用来干什么
41. Mybatis 原理
42. 为什么用 XML，不用 json
43. MySQL 索引的数据结构，有什么特性
44. Structs2 工作流程；Structs2 和 Structs1 的区别
45. Spring IOC 启动过程
46. TCP Time_wait 发生在什么时候
47. ConcurrentHashMap 源码，原理
48. 当向 ConcurrentHashMap 的一个 segment 中添加的 HashEntry 太多时怎么办
49. Java 的四种引用

① **强引用（使用最普遍的引用）**

如果一个对象具有强引用（类似 Object obj=new Object();String s="He";），垃圾回收器绝不会回收它。当内存空间不足，Java 虚拟机宁愿抛出 OutOfMemoryError 错误，使程序异常终止，也不会靠随意回收具有强引用的对象来解决内存不足的问题。

② **软引用 （Soft Reference）**

用于描述一些还有用但并非必需的对象，当堆将发生 OOM 时则会回收软引用所指向的内存地址，若回收后依然不足才会抛出 OOM。一般用于实现内存敏感的高速缓存。

③ **弱引用（Weak Reference）**

也用于描述非必需对象，发生 GC 时必定回收弱引用指向的内存空间。

④ **虚引用（Phantom Reference）**

虚引用既不会影响对象的生命周期，也无法通过虚引用来获取对象实例，仅用于发生 GC 时接收一个系统通知。虚引用必须和引用队列（java.lang.ref.ReferenceQueue）使用。

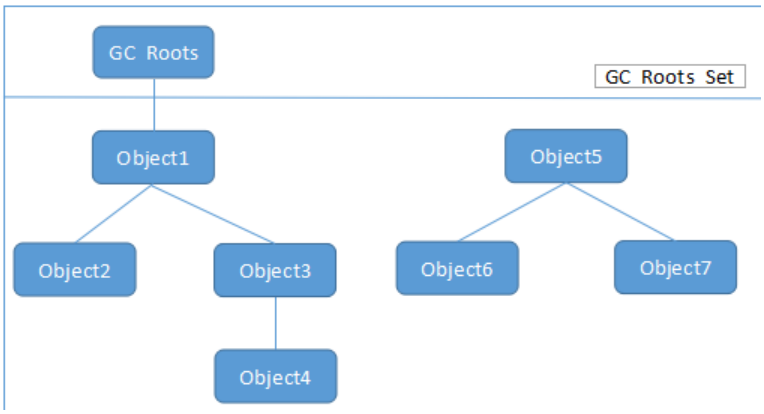
##如果一个对象引用类型有多个，怎么判断可达性##

① 单条引用链的可达性以最弱的一个引用类型来决定
② 多条引用链的可达性以最强的一个引用类型来决定

【实例参考】
【内容参考】

50. 可达性分析算法

Java 使用可达性分析算法判断对象是否存活。
基本思路 :通过一系列名为"GC Roots"的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链(Reference Chain)，当一个对象到 GC Roots 没有任何引用链相连时，则证明此对象是不可用的。



##可作为 GC Roots 的对象##

a. 虚拟机栈(栈桢中的本地变量表)中的引用的对象

b. 方法区中的类静态属性引用的对象

c. 方法区中的常量引用的对象

d. 本地方法栈中 JNI 的引用的对象

51. 一致性哈希算法的原理

52. 反射

53. 序列化与反序列化

如何实现序列化与反序列化：

序列化是一种处理对象流的机制，所谓对象流也就是将对象的内容进行流化。

要实现序列化，需要让一个类实现 Serializable 接口，该接口是一个标识性

接口，标注该类对象是可被序列化的，然后使用一个输出流来构造一个对象输出流并通过 writeObject(Object)方法就可以将实现对象写出（即保存其状态）；如果需要反序列化则可以用一个输入流建立对象输入流，然后通过 readObject 方法从流中读取对象。

序列化的意义：

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。序列化是为了解决对象流读写操作时可能引发的问题（如果不进行序列化可能会存在数据乱序的问题）。

序列化除了能够实现对象的持久化之外，还能够用于对象的深度克隆。

54. String 可不可变；自己实现 String 的 split 函数

String 不可变。（final 修饰,且底部是 final char[]）

***不可变对象**：如果一个对象，在它创建完成之后，不能再改变它的状态，那么这个对象就是不可变的。不能改变状态的意思是，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。*

好处：字符串常量池；缓存哈希值；促进其他对象的使用（比如 HashSet）；安全性（例如 网络连接地址 URL,文件路径 path,还有反射机制所需要的 String 参数等, 假若 String 不是固定不变的,将会引起各种安全隐患。）；线程安全。

重写 split 函数：【参考】

55. 链表的倒数第 k 个节点；双链表的倒数第 k 个节点

56. 写 SQL 语句查找成绩第一名的学生，如果不能用排序呢？如果不能用 max 函数呢？

57. 比如现在有一张学生信息表，如何查询总分前三的学生。

58. int 与 Integer 的区别，我讲到了 Integer 会对-128-127 之间的数进行缓存，然后他就问到了哪些包装类型会进行缓存

59. LinkedBlockingQueue 源码；生产者消费者问题

60. 红黑树的性质，如何保持平衡；红黑树什么时候经过三次旋转保持平衡

61. 写一个脚本统计出现次数前 20 的 IP

Shell 脚本如下：(test.txt 含 IP 地址)

cat test.txt | awk '{print $2}' | sort | uniq -c | sort -n -r | head -n 1

62.树中两个节点的最低公共祖先（思路）

63.策略模式与工厂模式的异同点

64.线程安全的单例模式的几种实现方式

① 饿汉式单例。在方法在调用前，实例就已经创建好了。

② 懒汉式单例。在方法调用时才创建实例。要保证线程安全，需要使用同步机制。

⑴ 方法中声明 synchronized 关键字。这种方法可以解决问题，但是运行效率会很低。

⑵ 同步代码块实现。效率同样不高。

⑶ 针对某些重要的代码来进行单独的同步（可能非线程安全）

⑷ Double Check Locking（双检查锁机制），需要使用 volatile 关键字保证对象的可见性。*（否则不能保证线程安全）*【更多参考】

③ 使用静态内置类实现单例模式。

④ 序列化与反序列化的单例模式实现。解决办法线程安全的办法就是在反序列化的过程中使用 readResolve()方法。

⑤ 使用 static 代码块实现单例。

⑥ 使用枚举数据类型实现单例模式。

【参考】

65.ListIterator 的作用

Iterator 与 ListIterator 的联系和区别：

① ListIterator 有 add 方法，可以向 List 中添加对象，而 Iterator 不能

② ListIterator 和 Iterator 都有 hasNext 和 next 方法，可以实现顺序向后遍历。但是 ListIterator 有 hasPrevious 和 previous 方法，可以实现逆向遍历。Iterator 就不可以。

③ ListIterator 可以定位当前的索引位置，nextIndex()和 previousIndex()可以实现。Iterator 没有此功能。

④ 都可实现删除对象，但是 ListIterator 可以实现对象的修改，set()方法可以实现。Iterator 仅能遍历，不能修改。

Iterator 模式是用于遍历集合类的标准访问方法。它可以把访问逻辑从不同类型的集合类中抽象出来，从而避免向客户端暴露集合的内部结构。

66.数据库主键和外键的区别

主键、外键、索引的区别：

	主键	外键（FK）	索引
定义	唯一标识一条记录，不能有重复，不能为空	表的外键是另一个表的主键，外键可以有重复，可以为空值	该字段没有重复值，但可以有一个空值
作用	保证数据完整性	用来和其他表建立联系	提高查询排序的速度
个数	只能有一个	可以有多个	一个表可以有多个唯一索引

主键选取策略：1.自动增长型；2.手动增长型；3.使用 UniqueIdentifier SQL Server 提供一个 UniqueIdentifier 数据类型（16 字节），并提供一个生成函数 NEWID()，生成一个唯一的 UniqueIdentifier；34.使用 COMB 类型

外键主要是用来控制数据库中的数据完整性的，当对一个表的数据进行操作时，和他有关联的一个表或多个表的数据能够同时发生改变。

67.HashMap 和 concurrenthashmap 源码

68.Atomic 类为什么比 Synchronized 效率高

69.HashCode 设计，以及目的

重写 equals()方法时，有一个原则：那就是必须覆写 hashCode()方法，让 equals 方法和 hashCode 方法始终在逻辑上保持一致性。

在 java 中，hashCode()方法的主要作用是基于散列的集合正确存取，比如 HashSet、HashMap\HashTable 等。

hashCode 契约：

① 在一个运行的进程中，相等的对象必须要有相同的哈希码

hashCode 并不保证在不同的应用执行中得到相同的结果。

② 不相等的对象不一定有着不同的哈希码

③ 有同一个哈希值的对象不一定相等

70. 数组和链表的区别

数组分配在一块连续的内存空间，在编译阶段就要确定空间大小。

优点：可以利用偏移地址访问元素，效率高；已排序数组还可以折半查找，效率极高。

缺点：空间连续，存储率低；插入/删除效率低，且很麻烦。

链表是动态申请的内存空间。

优点：插入和删除元素不需要移动其余元素，效率高。

缺点：查找和搜索元素效率低。

链表由于存放的地方在内存中是分散的，因此 cpu 的基地址寄存器等等必须重新赋值，而数组这一过程是不要的。

71.平时怎么学习编程？会怎么学习一门新的编程语言？

72. B+树原理？Bloom Filter 什么原理？BloomFilter 有什么缺陷？

73. Java 多线程作用？使用中会有什么问题？

74. 死锁是什么？怎么解决？

75. Java float 精确么？为什么？

小数的二进制表示有时是不可能精确的（比如 0.9）。

float 型在内存中的存储如下

4 bytes	31	30	29-23	22-0
表示意义	实数位符号	指数位符号	指数位	有效数位

其中符号位 0 表示正，1 表示负。有效位数位 24 位，其中一位是实数符号位。

76. 为什么要 8:1:1 设定

77. I/O 流。文件流是什么，有没有缓冲区；FileInputStream 无缓冲区，BufferedInputStream 才有。I/O 流不关闭会出现什么问题

78. 文件到 CPU 的过程

79. 两个服务器，A 向 B 传递数据，预估时间

80.什么是互联网+？

81. 写 socket 通信代码，代码中每个函数对应到 TCP 各个状态

82. Servlet 生命周期

① 初始化阶段 调用 init()方法，仅会被执行一次

##在以下时刻 Servlet 容器装载 Servlet##

I Servlet 容器启动时自动装载某些 Servlet,只需要在 web.xml 中注册

II 在 Servlet 容器启动后，客户首次向 Servlet 发送请求

III Servlet 类文件被更新后，重新装载 Servlet

② 响应客户请求 调用 service()方法

③ 终止阶段 调用 destroy()方法

当 WEB 应用被终止，或 Servlet 容器终止运行，或 Servlet 容器重新装载 Servlet 新实例时，Servlet 容器会先调用 Servlet 的 destroy()方法，在 destroy()方法中可以释放掉 Servlet 所占用的资源。

WEB 应用启动时机：1. 当 Servlet 容器启动时，所有 WEB 应用都会被启动

2. 控制器启动 WEB 应用

83. Spring 依赖注入原理；SpringMVC 的工作流程（Spring 需看源码）

84. Mysql 和 Oracle 的区别；数据库优化

(1) Oracle 是大型数据库，Mysql 是中小型数据库

(2) Oracle 支持大并发，大访问量，是 OLTP 最好的工具

(3) Oracle 安装所使用空间大一些

(4) 操作上也有所区别：

1. Mysql 一般使用自动增长类型；Oracle 没有，主键一般使用序列
2. Mysql 可以用双引号包起字符串，Oracle 只使用单引号
3. Mysql 翻页：limit；Oracle 使用 ROWNUM
4. MYSQL 的非空字段也有空的内容，ORACLE 里定义了非空字段就不容许有空的内容
5. MYSQL 里用 字段名 like '%字符串%',ORACLE 里也可以用 字段名 like '%字符串%' 但这种方法不能使用索引，速度不快。
6. Oracle 实现了 ANSII SQL 大部分功能，而 Mysql 在这方面还比较弱。
7. 在长字符串处理上也有所区别。

【Detail】

数据库优化（Mysql）：

1. 选取最适用的字段属性

   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 NOT NULL，这样在将来执行查询的时候，数据库不用去比较 NULL 值。

2. 使用连接（join）来代替子查询（Sub-Queries）
3. 使用联合（union）来代替手动创建的临时表
4. 事务（有独占性，又是会影响性能，尤其是在很大的应用系统中）
5. 锁定表（可以维护数据完整性，但却不能保证数据的关联性）
6. 使用外键
7. 使用索引

   一般说来，索引应建立在那些将用于 JOIN,WHERE 判断和 ORDERBY 排序的字段上。尽量不要对数据库中某个含有大量重复的值的字段建立索引。对于一个 ENUM 类型的字段来说，出现大量重复值是很有可能的情况。

8. 优化的查询语句
   a) 最好是在相同类型的字段间进行比较操作
   b) 在建有索引的字段上尽量不要使用函数进行操作
   c) 在搜索字符型字段时，我们有时会使用 LIKE 关键字和通配符，这种做法虽然简单，但却也是以牺牲系统性能为代价的。

【传送门】

85.负载均衡有哪几种

86.ThreadLocal 原理，用它实现统计 WEB 服务器每一层的调用时间；ThreadLocalMap 和 HashMap 的区别？使用时需要注意什么？

87.Tomcat 的体系架构和请求处理流程

88.Spring 的启动初始化流程（定位、载入、注册）和依赖注入的流程；要是相互依赖怎么办？

89.如何实现数据库的行锁和表锁

90.服务器是如何保持 HTTP 长连接的，web 服务器是如何控制和生成 session 的

91.对公司以及公司产品的了解

92.Java 多态有什么特性

多态：指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的

方法调用在编程时并不确定，而是在程序运行期间才确定。

实现条件：继承、重写、向上转型

实现形式：基于继承实现、基于接口实现

遵循的原则概括：当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法，但是它仍然要根据继承链中方法调用的优先级来确认方法，该优先级为：this.show(O)、super.show(O)、this.show((super)O)、super.show((super)O)。

93. 如何查找一个 Key 对应的 Value

94.25 匹马 5 个赛道，没有计时器，最少多少次赛马就能找出最快的 3 匹马

95. 有一个 500 万用户的黑名单，一个 500 万用户的白名单，如何识别一个用户是不是在一个黑名单里面

96. 一个字符串里面找 acdecd 一个子串出现的次数

97. 单向链表的排序最快要怎么做

98. 输出一个字符串数组里面，表示数字的字符串中，最长的那个串

99. 输入美团的网址，后端都做了什么

100. tcp 协议为什么比 udp 可靠，dns 会用到 tcp 实现吗，有这个应用场景吗？

101. 怎么统计整个成都的公交车数量

102. 内存管理的页面地址转换怎么做

103. 二叉树的非递归遍历

104. 有哪些机制可以实现并发？若果不使用锁如何实现？

105. 哪些情况会出现死锁

106. TCP 与 UDP 区别

TCP 是面向连接的协议，UDP 是一个非连接协议

TCP 对系统资源的要求较多

UDP 程序结构更为简单

TCP 是流模式，UDP 是基于数据报模式（UDP 是面向报文的）

TCP 保证数据正确性，UDP 可能丢包；TCP 保证数据顺序，UDP 不保证

107. HashMap 的 Key 有什么要求（其实就是重新 HashCode 的问

题）

这个 Key 既可以是基本数据类型对象，如 Integer，Float，同时也可以是自己编写的对象，甚至支持 null（最多只允许一条记录的键为 null）。

可变对象是指创建后自身状态能改变的对象。换句话说，可变对象是该对象在创建后它的哈希值（由类的 hashCode（）方法可以得出哈希值）可能被改变。

Key 不能为可变对象。

108. Get 和 Post 的区别

① Get 是用来从服务器上获得数据，而 Post 是用来向服务器上传数据。

② Get 将表单中数据的按照 variable=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 form 的数据体中，按照变量和值相对应的方式，传递到 action 所指向 URL。

③ Get 是不安全的，因为在传输过程，数据被放在请求的 URL 中，而如今现有的很多服务器、代理服务器或者用户代理都会将请求 URL 记录到日志文件中，然后放在某个地方，这样就可能会有一些隐私的信息被第三方看到。另外，用户也可以在浏览器上直接看到提交的数据，一些系统内部消息将会一同显示在用户面前。Post 的所有操作对用户来说都是不可见的。

④ Get 传输的数据量小，这主要是因为受 URL 长度限制；而 Post 可以传输大量的数据，所以在上传文件只能使用 Post（当然还有一个原因，将在后面的提到）。

⑤ Get 限制 Form 表单的数据集的值必须为 ASCII 字符；而 Post 支持整个 ISO10646 字符集。默认是用 ISO-8859-1 编码

⑥ Get 是 Form 的默认方法。

109. Session 与 Cookie 的原理以及区别

110. NIO，IO，AIO 的区别和基本原理

111. 泛型基本原理

泛型的本质是参数化类型，即所操作的数据类型被指定为一个参数。

Java 的泛型是伪泛型。为什么说 Java 的泛型是伪泛型呢？因为，在编译期间，所有的泛型信息都会被擦除掉。

泛型的**实现方法：类型擦除**

Java 中的泛型基本上都是在编译器这个层次来实现的。在生成的 Java 字节码中是不包含泛型中的类型信息的。使用泛型的时候加上的类型参数，会在编译器在编译的时候去掉。这个过程就称为类型擦除。

类型擦除引起的问题以及解决办法：

1. 先检查，再编译，以及检查编译的对象和引用传递的问题

   java 编译器通过先检查代码中泛型的类型，然后再进行类型擦除，再进行编译。

   类型检查就是针对引用的，谁是一个引用，用这个引用调用泛型方法，就会对这个引用调用的方法进行类型检测，而无关它真正引用的对象。

2. 自动类型转换
3. 类型擦除与多态的冲突和解决办法

   桥方法
4. 泛型类型变量不能是基本数据类型
5. 运行时类型查询

```
1. if(arrayList instanceof ArrayList<String>) //错误
2. //java 限定了这种类型查询的方式
3. if(arrayList instanceof ArrayList<?>)
```

6. 异常中使用泛型的问题

   ⑴ 不能抛出也不能捕获泛型类的对象

   事实上，泛型类扩展 Throwable 都不合法。

   ⑵ 不能在 catch 子句中使用泛型变量
7. 数组（这个不属于类型擦除引起的问题）

   不能声明参数化类型的数组，不能建立一个泛型数组，但是，可以用反射构造泛型对象和数组。
8. 泛型类型的实例化

   不能实例化泛型类型，如 first=new T();//error
9. 类型擦除后的冲突

   1. 当泛型类型被擦除后，创建条件不能产生冲突。
   2. 要支持擦除的转换，需要强行制一个类或者类型变量不能同时成为两个接口的子类，而这两个子类是同一接品的不同参数化。
10. 泛型在静态方法和静态类中的问题

    泛型类中的静态方法和静态变量不可以使用泛型类所声明的泛型类型参数

泛型的好处：

① 类型安全

通过知道使用泛型定义的变量的类型限制，编译器可以在一个高得多的程度上验证类型假设。

② 消除强制类型转换

消除源代码中许多强制类型转换，使得代码更加可读，并减少了出错的机会。

③ 潜在的性能收益

在泛型的初始实现中，编译器将强制类型转换（没有泛型的话，程序员会指定这些强制类型转换）插入生成的字节码中。但是更多类型信息可用于编译器这一事实，为未来版本的 JVM 的优化带来可能。由于泛型的实现方式，支持泛型（几乎）不需要 JVM 或类文件更改。

【more】

112. 动态规划和贪心算法的原理和区别

113. 100 亿数据，每条数数据有一些关键词，找出出现最多的 10 个，你如何做？（TOP 问题，很重要，经常出现）

114. JVM 调优

115. Mysql 建立索引应该遵循些什么原则

116. 线程池的状态？阻塞队列有哪些？scheduledthreadpoolexecutor 用到那种阻塞队列？

117. HashTable 有什么缺陷？Java 如何改进？（实际问线程安全问题，concurrenthashmap）

118. jvm 有哪些分区

119. 知道哪些加解密方式，MD5 一定是安全的吗

120. MySql 使用哪些引擎，它们的区别

121. 如何测试系统高并发的性能，如何模拟高并发

122. jdk/bin 目录下有哪些东西？

123. jconsole，jstack 怎么用，里面都有哪些信息

124. 如果发现 sql 语句执行很慢，你怎么找出这条语句，(查日志)

125. 堆的分区，新生代 GC 是如何触发的？

126. Java 垃圾回收时，哪些对象可以作为 GCROOT

127. 常用 jvm 参数，写个 OOM 例子

128. 假如堆内存最大只有 2G，现在发现电脑内存占用了 3G，你知道可能的原因吗？（直接内存导致）

129. Jdk1.8 新特性

130. TCP 如何保证可靠性？超时重传，校验机制，发送缓存和接受缓存，流量控制等方面回答

131. Json 怎么解析数据

132. 了解 https 吗，说下加密过程，客户端生成随机数之前难道不要做什么吗？(验证证书是否可信赖)

133. 面向对象和面向过程的区别（异同）

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了；面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为。

134. arraylist 和 vector 以及 linkedlist 的区别？

135. 一个数的集合，里面只有一个数不同，其他成对出现，怎么找出这个不同的数，如果有两个不同的数呢

136. 多线程间如何通信？

① 同步 （ex:synchronized）

本质上就是“共享内存”式的通信。多个线程需要访问同一个共享变量，谁拿到了锁（获得了访问权限），谁就可以执行。

② while 轮询 （线程 A 不断地改变条件，线程 B 不停地通过 while 语句检测条件是否成立，从而实现了线程间的通信。）

这种方式会浪费 CPU 资源，

③ wait/notify 机制

CPU 的利用率有所提高，但是会出现另一个问题：通知过早，会打乱程序的执行逻辑。

④ 管道通信

（使用 java.io.PipedInputStream 和 java.io.PipedOutputStream 进行通信）

管道通信更像消息传递机制，即通过管道，将一个线程中的消息发送给另一个。

137. 进程间通信有哪几种方式

① 管道

② 命名管道

③ 信号

④ 消息队列

⑤ 共享内存

⑥ 内存映射

⑦ 信号量

⑧ 套接字

【more】

138. 现在有很多 xml 布局文件里面要使用相同的布局，怎么实现复用?(include 使用)

139. 如何优化布局？使用过 ViewStub 和 merge 吗？

140. 现在有这样一种情形，数据的规模未知，数据的类型未知，请你选择一种合适的排序算法，并说明原因。

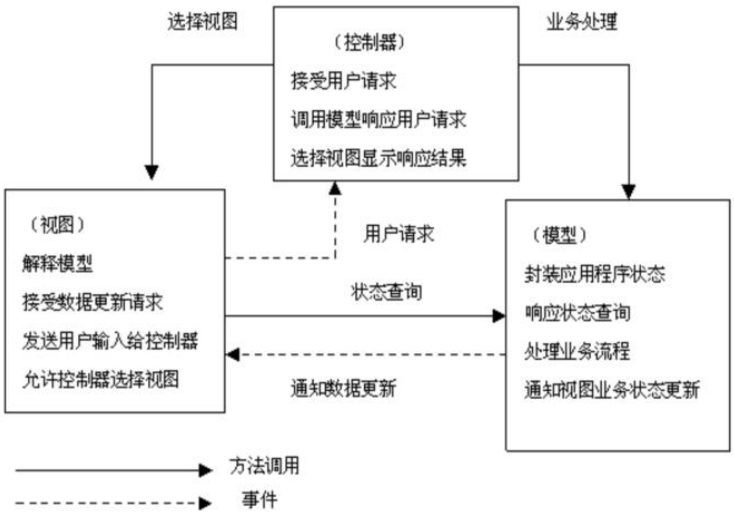
141. MVC 模式

MVC 模式（Model-View-Controller）是软件工程中的一种软件架构模式，把软件系统分为三个基本部分：模型（Model）、视图（View）和控制器（Controller）。

控制系统：对请求进行处理，负责请求转发

视图：界面设计人员进行图形界面设计

模型：程序编写程序应用的功能（实现算法等等）、数据库管理



自己总结的问题：

1. 创建数组的三种方式

① int[] arr=new int[6];

② int[] arr={1,2,3,4,5,6};

③ int[] arr=new int[]{1,2,3,4,5,6};

2. 复制数组的五种方法

① “=”

② for 循环

③ clone 方法 得到数组的值，而不是引用，灵活性不够

④ System.arraycopy(src, srcPos, dest, destPos,length) 效率最高

⑤ copyOf Arrays 的方法，本质上是调用 arraycopy 方法，效率就比不上 arraycopy 了

3. jsp 传参的几种方式

① form 表单

② request.setAttribute();和 request.getAttribute();

③ 超链接：name

④ <jsp:param>

4. Servlet 请求转发(RequestDispatcher)与重定向(sendRedirect)的区别

重定向功能(HttpServletResponse.sendRedirect)是让浏览器重新发生请求，但是将请求的是另外一个 Servlet;

请求转发(RequestDispatcher.forward)是将客户端的请求转发到另外一个 Servlet 或者 Jsp 页面

RequestDispatcher.forward 方法只能将请求转发给同一个 WEB 应用中的组件；而 HttpServletResponse.sendRedirect 方法不仅可以重定向到当前应用程序中的其他资源，还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对 URL 重定向到其他站点的资源。如果传递给 HttpServletResponse.sendRedirect 方法的相对 URL 以“/”开头，它是相对于整个 WEB 站点的根目录；如果创建 RequestDispatcher 对象时指定的相对 URL 以“/”开头，它是相对于当前 WEB 应用程序的根目录。

调用 HttpServletResponse.sendRedirect 方法重定向的访问过程结束后，浏览器地址栏中显示的 URL 会发生改变，由初始的 URL 地址变成重定向的目标 URL；而调用 RequestDispatcher.forward 方法的请求转发过程结束后，

浏览器地址栏保持初始的 URL 地址不变。

HttpServletResponse.sendRedirect 方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个 URL 的访问请求。RequestDispatcher.forward 方法在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为。

RequestDispatcher.forward 方法的调用者与被调用者之间共享相同的 request 对象和 response 对象，它们属于同一个访问请求和响应过程；而 HttpServletResponse.sendRedirect 方法调用者与被调用者使用各自的 request 对象和 response 对象，它们属于两个独立的访问请求和响应过程。对于同一个 WEB 应用程序的内部资源之间的跳转，特别是跳转之前要对请求进行一些前期预处理，并要使用 HttpServletRequest.setAttribute 方法传递预处理结果，那就应该使用 RequestDispatcher.forward 方法。不同 WEB 应用程序之间的重定向，特别是要重定向到另外一个 WEB 站点上的资源的情况，都应该使用 HttpServletResponse.sendRedirect 方法。

无论是 RequestDispatcher.forward 方法，还是 HttpServletResponse.sendRedirect 方法，在调用它们之前，都不能有内容已经被实际输出到了客户端。如果缓冲区中已经有了一些内容，这些内容将被从缓冲区中清除。

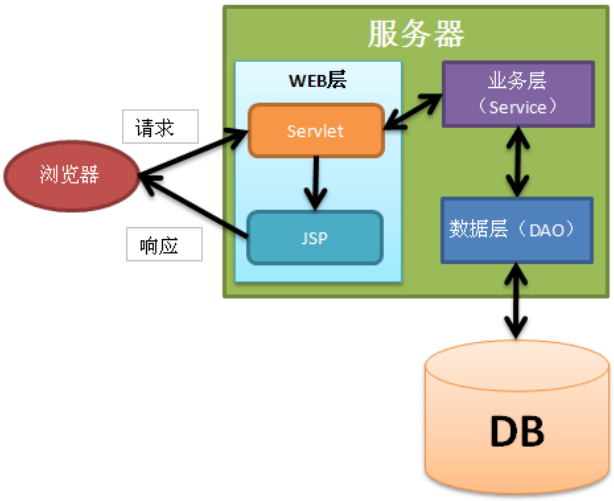
转发只发生一次 http 请求，而重定向发生了两次。

5. JavaWeb 经典三层框架

Web 层（表述层）：与 Web 相关，例如 jsp，servlet 都是 Web 层

Business 层（业务逻辑层）：封装业务逻辑，通常对应一个业务功能，例如登录，注册都是一个业务功能

Data 层（数据访问层）：封装对数据库的操作，通常对应一次对数据库的访问，例如添加、修改、删除、查询等



6. c3p0 数据库连接池使用，以及常用 Apache 包的使用

7. JDBC 操作数据库的步骤

```
1.  //加载驱动
2.  Class.forName("com.mysql.jdbc.Driver");
3.  //创建连接
4.  Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/my
    database", "scott", "tiger");
5.  //创建语句
6.  PreparedStatement ps = con.prepareStatement("select * from emp where sal bet
    ween ? and ?");
7.  ps.setInt(1,1000);
8.  ps.setInt(2,2000);
9.  //执行语句
10. ResultSet rs = ps.executeQuery();
11. //处理结果
12. while(rs.next()) {
13.     System.out.println(rs.getInt("empno") + " - " + rs.getString("ename"));

14. }
15. //关闭资源
16. finally {
17.     if(con != null) {
18.         try {
19.             con.close();
20.         } catch (SQLException e) {
21.             e.printStackTrace();
22.         }
```

```
23.     }
24. }
```

8. Statement 和 PreparedStatement 有什么区别？哪个性能更好？

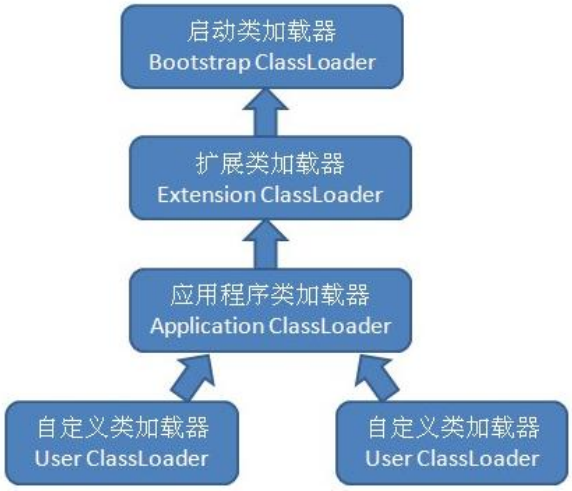
⑴ PreparedStatement 接口代表预编译的语句，它主要的优势在于可以减少 SQL 的编译错误并增加 SQL 的安全性（减少 SQL 注射攻击的可能性）。

⑵ PreparedStatement 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。

⑶ 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）。

9. 类加载器与双亲委派模型

在 Java 中，任意一个类都需要由加载它的类加载器和这个类本身一同确定其在 java 虚拟机中的唯一性，即比较两个类是否相等，只有在这两个类是由同一个类加载器加载的前提之下才有意义，否则，即使这两个类来源于同一个 Class 类文件，只要加载它的类加载器不相同，那么这两个类必定不相等(这里的相等包括代表类的 Class 对象的 equals()方法、isAssignableFrom()方法、isInstance()方法和 instanceof 关键字的结果)。



启动类加载器（针对 Hotspot,由 C++预言实现）；其他类加载器由 java 语言实现，继承自抽象类 ClassLoader。

① **BootStrap ClassLoader**:**启动类加载器**，负责将存放在<JAVA_HOME>\lib 目录或-Xbootclasspath 参数指定的路径中的类库加载到内存中。启动类加载器无法被 java 程序直接引用。

② **Extension ClassLoader**: **扩展类加载器**，负责加载%JAVA_HOME%\lib\ext 目录中的，或者被 java.ext.dirs 系统变量所指定的路径中的所有类库，开发者可以直接使用扩展类加载器。

③ **Application ClassLoader**: **应用程序类加载器**，负责加载用户类路径（classpath）上的指定类库，我们可以直接使用这个类加载器。一般情况，如果我们没有自定义类加载器默认就是用这个加载器。

加载过程如下：

① 如果一个类加载器收到了类加载请求，它首先不会自己去尝试加载这个类，而是把类加载请求委派给父类加载器去完成。

② 每一层的类加载器都把类加载请求委派给父类加载器，直到所有的类加载请求都应该传递给顶层的启动类加载器。

③ 如果顶层的启动类加载器无法完成加载请求，子类加载器尝试去加载，如果连最初发起类加载请求的类加载器也无法完成加载请求时，将会抛出 ClassNotFoundException，而不再调用其子类加载器去进行类加载。

双亲委派模型的**优点**：Java 类随着加载它的类加载器一起具备了一种带有优先级的层次关系。比如，Java 中的 Object 类，它存放在 rt.jar 之中,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启动类加载器进行加载，因此 Object 在各种类加载环境中都是同一个类。如果不采用双亲委派模型，那么由各个类加载器自己取加载的话，那么系统中会存在多种不同的 Object 类。

自定义类加载器，必须重写 findClass 方法。(*参考*)

10. Java 代理

代理模式特征是代理类与委托类有同样的接口，代理类主要负责为委托类预处理消息、过滤消息、把消息转发给委托类，以及事后处理消息等。

① 静态代理（程序运行前，代理类的 class 文件就已经存在了）

② JDK 动态代理（在程序运行时，运用反射机制动态创建而成）

③ cglib 动态代理（JDK 动态代理机制只能代理实现了接口的类，而不能实现接口的类就不能实现 JDK 的动态代理，cglib 是针对类来实现代理的，他的原理是对指定的目标类生成一个子类，并覆盖其中方法实现增强，但因为采用的是继承，所以不能对 final 类进行代理）

11. 类的加载机制

类的生命周期是从被加载到虚拟机内存中开始，到卸载出内存结束。过程共有七个阶段，其中初始化之前的都是属于类加载的部分。

加载

① 通过一个类的全限定名来获取定义此类的二进制字节流（并没有指明要从一个 Class 文件中获取，可以从其他渠道，譬如：网络、动态生成、数据库等）；

② 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构；

③ 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访问入口

加载阶段和连接阶段的部分内容（如一部分字节码文件格式验证动作）是交叉进行的，加载阶段尚未完成，连接阶段可能已经开始，但这些夹在加载阶段之中进行的动作，仍然属于连接阶段的内容，这两个阶段的开始时间仍然保持着固定的先后顺序。

验证（非常重要，但不是必须）

目的是为了确保 Class 文件的字节流中包含的信息符合当前虚拟机的要求，且不会危害虚拟机自身的安全。

文件格式验证 + 元数据验证 + 字节码验证 + 符号引用验证

准备

正式为类变量分配内存并设置类变量初始值，所使用的内存在方法区分配。

这里的初始值“通常情况”下是数据类型的零值。至于“特殊情况”是指：public static final int value=1;

解析

虚拟机将常量池内的符号引用替换为直接引用的过程

初始化（类加载过程的最后一步）

真正开始执行类中定义的 java 程序代码。初始化阶段是执行类构造器<clinit>()方法过程。对类的静态变量，静态代码块执行初始化操作

<clinit>()方法是由编译器自动收集类中所有类变量的赋值动作和静态语句块static{}中的语句合并产生的，收集顺序由语句在源文件中出现的顺序所决，静态语句块只能访问到定义

在静态语句块之前的变量，定义在它之后的变量，在前面的静态语句块可以赋值，但是不能访问。

【##注意点##】

① 子类引用父类的静态变量，不会导致子类初始化

② 通过数组定义引用类，不会触发此类的初始化

③ 引用常量时，不会触发该类的初始化

用 final 修饰某个类变量时，它的值在编译时就已经确定好放入常量池了，所以在访问该类变量时，等于直接从常量池中获取，并没有初始化该类。

④ 初始化时机

I 创建类实例的时候（1、使用 new 关键字创建实例；2、通过反射创建实例；3、通过反序列化方式创建实例。）

II 调用某个类的类方法（静态方法）

III 访问某个类或接口的类变量，或为该类变量赋值

IV 初始化某个类的子类。当初始化子类的时候，该子类的所有父类都会被初始化。

V 直接使用 java.exe 命令来运行某个主类。

12. javac 命令解释

javac [options] [sourcefiles] [@files] *(参数可按任意次序排列)*

options	命令行选项
sourcefiles	一个或多个要编译的源文件（例如 MyClass.java）
@files	一个或多个对源文件进行列表的文件

13. 生产者消费者问题

14. 四种常见的 post 提交数据方式

① application/x-www-form-urlencoded

② multipart/form-data

③ application/json

④ text/xml

【more】

15. 不用加减乘除做加法运算

```java
public static int add(int a,int b){
    int sum,carry;
    do{
        sum=a^b;
        carry=(a&b)<<1;
        a=sum;
        b=carry;
    }while(carry!=0);
    return sum;
}
```

16. 第一个出现一次的字符(字符串长度在 1-1000，全部由大小写字母组成)

```java
public static int FirstNotRepeatingChar(String str){
    List<Character> list=new ArrayList<>();
    for(int i=0;i<str.length();i++){
        Character temp=str.charAt(i);
        if(list.contains(temp)){
            list.remove(temp);
        }else{
            list.add(temp);
        }
    }
    return str.indexOf(list.get(0)+"");
}
```

17. 二叉搜索树转换成排序的双向链表
18. 实现一个保证迭代顺序的 HashMap
19. 双亲委派模型中有哪些方法？有没有可能父类加载器和子类加载器，加载同一个类？如果加载同一个类，该使用哪一个类？
20. HashMap 和 Concurrent HashMap 区别，Concurrent HashMap 线程安全吗， Concurrent HashMap 如何保证 线程安全？
21. JVM 分区，每个区的功能
22. JVM 如何 GC，新生代，老年代，持久代，都存储哪些东西？
23. 什么是二叉平衡树，如何插入节点，删除节点
24. 滑动窗口算法
25. 常用的 hash 算法

26. 数据库中索引的结构有哪些（B 树，B+树，全文倒排索引等）？什么情况下适合建索引？

27. Java 中 NIO，BIO，AIO 分别是什么

28. 调试工具？Jconsole?

29. JVM 有一个线程挂起了，如何用工具查出原因

30. 线程同步与阻塞的关系？同步一定阻塞吗？阻塞一定同步吗？

31. 如何创建单例模式？如何高效的创建一个线程安全的单例

32. Concurrent 包，都用过哪些？

33. 如何判断一个单链表是否有环

34. 匿名内部类是什么？如何访问在其外面定义的变量

匿名内部类（没有名字，仅能被使用一次；继承一个类或者借口，不可兼得；不能定义构造函数；不能有任何静态成员/方法；属于局部内部类，即局部内部类限制对其都有效；不能是抽象的）

利用构造代码块能够达到为匿名内部类创建一个构造器的效果。

```
1.  new 父类构造器(参数列表)|实现接口()
2.      {
3.       //匿名内部类的类体部分
4.      }
```

给匿名内部类传递参数的时候，若该形参在内部类中需要被使用，那么该形参必须要为 final。【简单理解就是，拷贝引用，为了避免引用值发生改变，例如被外部类的方法修改等，而导致内部类得到的值不一致，于是用 final 来让该引用不可改变。】

35. sleep() 和 wait() 分别是哪个类的方法，有什么区别？synchronized 底层如何实现的？用在代码块和方法上有什么区别？

sleep()属于 Thread 类的方法；wait()属于 Object 类的方法。

区别：

① **最大本质的区别**：sleep()不释放同步锁，wait()释放同步锁

② sleep 的作用是使当前线程进入停滞状态（阻塞当前线程），让出 CPU 的使用、目的是不让当前线程独自霸占该进程所获的 CPU 资源，以留一定时间给其他线程执行的机会；

当一个线程执行到 wait 方法时，它就进入到一个和该对象相关的等待池中，同时失去（释放）了对象的机锁（暂时失去机锁，wait(long timeout)超时时间到后还需要返还对象锁）；可以调用里面的同步方法，其他线程可以访问。

wait()使用 notify 或者 notifyAlll 或者指定睡眠时间来唤醒当前等待池中的线程。wiat()必须放在 synchronized block 中，否则会在 program runtime 时扔出"java.lang.IllegalMonitorStateException"异常。

③ wait, notify 和 notifyAll 只能在同步控制方法或者同步控制块里面使用，而 sleep 可以在任何地方使用。

④ sleep 必须捕获异常，而 wait，notify 和 notifyAll 不需要捕获异常

Synchronized 实现原理：【more】

同步代码块是使用 monitorenter 和 monitorexit 指令实现的，同步方法依靠的是方法修饰符上的 ACC_SYNCHRONIZED 实现。

同步代码块：monitorenter 指令插入到同步代码块的开始位置，monitorexit 指令插入到同步代码块的结束位置，JVM 需要保证每一个 monitorenter 都有一个 monitorexit 与之相对应。任何对象都有一个 monitor 与之相关联，当且一个 monitor 被持有之后，他将处于锁定状态。线程执行到 monitorenter 指令时，将会尝试获取对象所对应的 monitor 所有权，即尝试获取对象的锁；

同步方法： synchronized 方法则会被翻译成普通的方法调用和返回指令如:invokevirtual、areturn 指令，在 VM 字节码层面并没有任何特别的指令来实现被 synchronized 修饰的方法，而是在 Class 文件的方法表中将该方法的 access_flags 字段中的 synchronized 标志位置 1，表示该方法是同步方法并使用调用该方法的对象或该方法所属的 Class 在 JVM 的内部对象表示 Class 做为锁对象。

Java 对象头和 monitor 是实现 synchronized 的基础。

Synchronized 加在不同地方，锁不一样：

① 普通同步方法，锁是当前实例对象

② 静态同步方法，锁是当前类的 class 对象

③ 同步方法块，锁是括号里面的对象

Synchronized **能够实现原子性和可见性**；在 Java 内存模型中，synchronized 规定，线程在加锁时，先清空工作内存→在主内存中拷贝最新变量的副本到工作

内存→执行完代码→将更改后的共享变量的值刷新到主内存中→释放互斥锁。

36.什么是一致性哈希？用来解决什么问题？

37.数据库中的分页语句怎么写？

```
1.  -- 取前 5 条 --
2.  select * from table_name limit 0,5;
3.  --或者--
4.  select * from table_name limit 5;
5.
6.  -- 查询第 11 到第 15 条 --
7.  select * from table_name limit 10,5;
8.
9.  -- limit 关键字用法:offset 指定要返回的第一行的偏移量，rows 第二个指定返回行的最大
    数目。初始行的偏移量是 0(不是 1)。--
10. limit [offset,] rows;
```

为了检索从某一个偏移量到记录集的结束所有的记录行，可以指定第二个参数为 -1。此外，limit n 等价于 limit 0,n。

38.数据库中什么是事务？事务的隔离级别？事务的四个特性？什么是脏读，幻读，不可重复读？

39.怎么查看 jvm 虚拟机里面堆，线程的信息，用过什么命令？

40.什么是死锁？JVM 线程死锁，你该如何判断是因为什么？如果用 VisualVM，dump 线程信息出来，会有哪些信息？

41.ConcurrentHashMap 的 get()，put()，又是如何实现的？ConcurrentHashMap 有哪些问题？ ConcurrentHashMap 的锁是读锁还是写锁？

42.Voliatle 关键字

前言：

要想并发程序正确地执行，必须要保证原子性、可见性以及有序性（指令重排序的影响，如以下代码）。

```
1.  //线程 1:
2.  context = loadContext();   //语句 1
3.  inited = true;             //语句 2
```

```
4.
5.  //线程 2:
6.  while(!inited ){
7.    sleep()
8.  }
9.  doSomethingwithconfig(context);
```

Java 内存模型具备一些先天的“有序性”，即不需要通过任何手段就能够得到保证的有序性，这个通常也称为 happens-before 原则。如果两个操作的执行次序无法从 happens-before 原则推导出来，那么它们就不能保证它们的有序性，虚拟机可以随意地对它们进行重排序。

happens-before 比较重要的四条原则（共 8 条）：

① 程序次序规则

一个线程内，按照代码顺序，书写在前面的操作先行发生于书写在后面的操作

② 锁定规则

一个 unLock 操作先行发生于后面对同一个锁的 lock 操作

③ 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

④ 传递规则

如果操作 A 先行发生于操作 B，而操作 B 又先行发生于操作 C，则可以得出操作 A 先行发生于操作 C

volatile 关键字含义：一旦一个共享变量（类的成员变量，类的静态成员变量）被 volatile 修饰，就具备了两层语义：

⑴ 保证了不同线程对该变量进行操作的可见性

⑵ 禁止进行指令重排序

volatile 关键字保证了操作的可见性，但是没办法保证对变量操作的原子性。

volatile 实现内存可见性：通过加入内存屏障和禁止重排序优化实现。

① 对 volatile 变量执行写操作时，会在写操作后加入一条 store 屏障指令

② 对 volatile 变量执行读操作时，会在读操作前加入一条 load 屏障指令

通俗地讲：volatile 变量在每次被线程访问时，都强迫从主内存中重读该变量的值，而当该变量发生变化时，又会强迫线程将最新的值刷新到主内存。这样任何时刻，不同的线程总能看到该变量的最新值。

volatile 关键字能禁止指令重排序，所以能在一定程度上保证有序性。

1）当程序执行到 volatile 变量的读操作或者写操作时，在其前面的操作的更改肯定全

部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行。

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

volatile 使用场景：

使用 volatile 必须具备以下 2 个条件：

⑴ 对变量的写操作不依赖于当前值

⑵ 该变量没有包含在具有其他变量的不变式中

Example：

① 状态标记量

② double check

43.CAS 以及缺点

CAS：Compare and Swap，java.util.concurrent 包建立在 CAS 之上

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。 如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值 。否则，处理器不做任何操作。无论哪种情况，它都会在 CAS 指令之前返回该 位置的值。（在 CAS 的一些特殊情况下将仅返回 CAS 是否成功，而不提取当前 值。）CAS 有效地说明了“我认为位置 V 应该包含值 A；如果包含该值，则将 B 放到这个位置；否则，不要更改该位置，只告诉我这个位置现在的值即可。

CAS 应用：利用 CPU 的 CAS 指令，同时借助 JNI 来完成 Java 的非阻塞算法（nonblocking algorithms，一个线程的失败或者挂起不应该影响其他线程的失败或挂起的算法）

CAS 存在的问题：

① ABA 问题

因为 CAS 需要在操作值的时候检查下值有没有发生变化，如果没有发生变化则更新，但是如果一个值原来是 A，变成了 B，又变成了 A，那么使用 CAS 进行检查时会发现它的值没有发生变化，但是实际上却变化了。ABA 问题的**解决思路就是使用版本号**。在变量前面追加上版本号，每次变量更新的时候把版本号加一，那么 A－B－A 就会变成 1A-2B－3A。

从 Java1.5 开始 JDK 的 atomic 包里提供了一个类 AtomicStampedReference 来解决 ABA 问题。这个类的 compareAndSet 方法作用是首先检查当前引用是否等于预期引用，并且当前标志是否等于预期标志，如果全部相等，则以原子方式将该引用和该标志的值设置为给定的更新值。

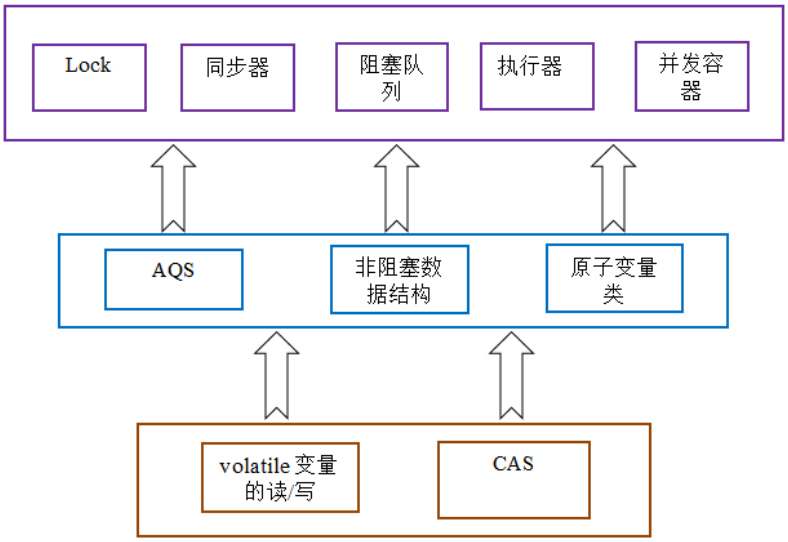
② 循环时间长开销大

自旋 CAS 如果长时间不成功，会给 CPU 带来非常大的执行开销。

③ 只能保证一个共享变量的原子操作

从 Java1.5 开始 JDK 提供了 AtomicReference 类来保证引用对象之间的原子性，你可以把多个变量放在一个对象里来进行 CAS 操作。

concurrent 包的实现：



【more】

44. HTTP/1.1 的五类状态码以及请求报文格式，响应报文格式

状态码由三位数字组成，第一个数字定义了响应的类别

1XX 提示信息 - 表示请求已被成功接收，继续处理

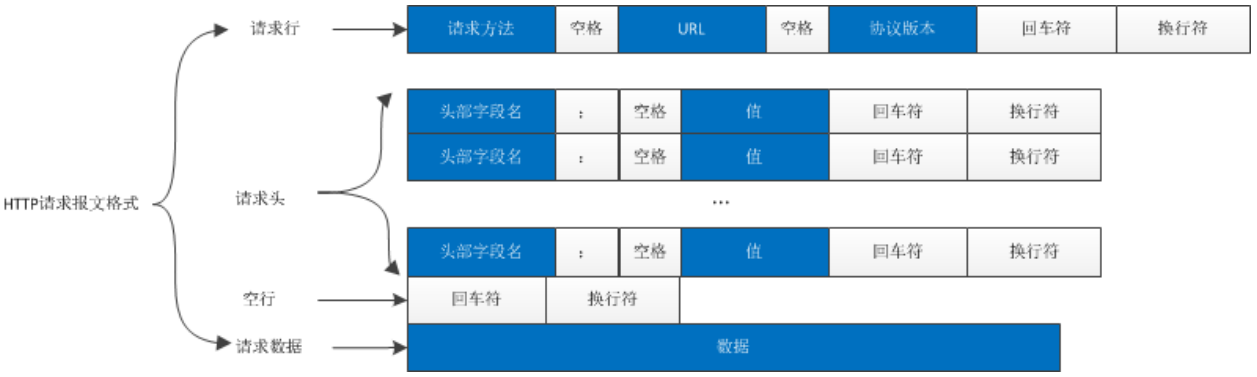
2XX 成功 - 表示请求已被成功接收，理解，接受

3XX 重定向 - 要完成请求必须进行更进一步的处理

4XX 客户端错误 - 请求有语法错误或请求无法实现

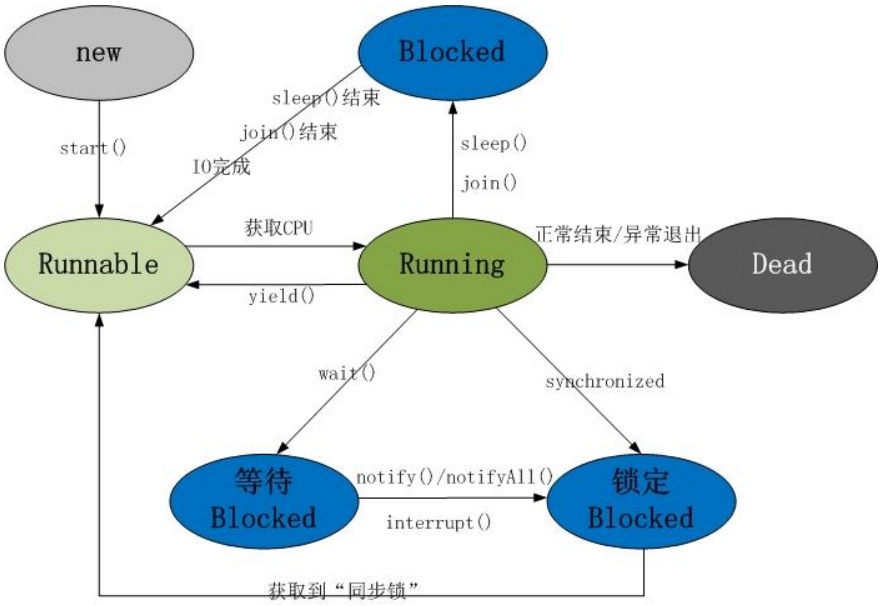
5XX 服务器端错误 - 服务器未能实现合法的请求

请求报文格式：



响应报文格式：

45.线程的生命周期



Java 线程的五种基本状态：

① 新建状态（New）：线程对象创建后，即进入该状态。

② 就绪状态（Runnable）：当调用了 start 方法即进入该状态，随时等待 CPU 调度执行。

③ 运行状态（Running）：CPU 开始调度处于就绪状态的线程，线程得到真正的执行，进入该状态。

④ 阻塞状态(Blocked)：处于运行状态的线程由于某种原因，**暂时放弃对 CPU 的使用权，停止执行，直到其进入就绪状态**，才有机会再被 CPU 调用进入搭配运行状态。

阻塞状态	解释
等待阻塞	运行中线程执行 wait 方法
同步阻塞	线程获取 synchronized 同步锁失败
其他阻塞	调用线程的 sleep 或 join 方法或发出 I/O 请求。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，线程重新转入就绪状态。

⑤ 死亡状态（Dead）：线程执行完了或因异常退出 run 方法，该线程结束生命周期

多线程的内存模型：main memory（主存）、working memory（线程栈），在处理数据时，线程会把值从主存 load 到本地栈，完成操作后再 save 回去(volatile 关键词的作用:每次针对该变量的操作都激发一次 load and save)。

46.线程的中断

Java 的中断是一种协作机制。也就是说调用线程对象的 interrupt 方法并不一定就中断了正在运行的线程，它只是要求线程自己在合适的时机中断自己，甚至可以不理会该请求，就像线程没有被中断一样。

中断是通过调用 Thread.interrupt()方法来做的。这个方法通过修改了被调用线程的中断状态来告知那个线程,说它被中断了。对于非阻塞中的线程， 只是改变了中断状态， 即 Thread.isInterrupted()将返回 true; 对于可取消的阻塞状态中的线程,比如等待在这些函数上的线程,Thread.sleep()， Object.wait(), Thread.join(),这个线程收到中断信号后,会抛出 InterruptedException， 同时会把中断状态置回为 true.但调用 Thread.interrupted()会对中断状态进行复位。

不是所有的阻塞方法收到中断后都可以取消阻塞状态,输入和输出流类会阻

塞等待 I/O 完成，但是它们不抛出 InterruptedException，而且在被中断的情况下也不会退出阻塞状态。

尝试获取一个内部锁的操作（进入一个 synchronized 块）是不能被中断的，但是 ReentrantLock 支持可中断的获取模式即 tryLock(long time, TimeUnit unit)。

中断的管理：

1. 如果遇到的是可中断的阻塞方法抛出 InterruptedException，可以继续向方法调用栈的上层抛出该异常，如果是检测到中断，则可清除中断状态并抛出 InterruptedException，使当前方法也成为一个可中断的方法。
2. 若有时候不太方便在方法上抛出 InterruptedException，比如要实现的某个接口中的方法签名上没有 throws InterruptedException，这时就可以捕获可中断方法的 InterruptedException 并通过 Thread.currentThread.interrupt()来重新设置中断状态。如果是检测并清除了中断状态，亦是如此。

总得来说，就是要让方法调用栈的上层获知中断的发生

中断的响应：

程序里发现中断后该怎么响应？这就得视实际情况而定了。有些程序可能一检测到中断就立马将线程终止，有些可能是退出当前执行的任务，继续执行下一个任务……作为一种协作机制，这要与中断方协商好，当调用 interrupt 会发生些什么都是事先知道的，如做一些事务回滚操作，一些清理工作，一些补偿操作等。若不确定调用某个线程的 interrupt 后该线程会做出什么样的响应，那就不应当中断该线程。

中断的使用：

通常，中断的使用场景有以下几个：

- 点击某个桌面应用中的取消按钮时；
- 某个操作超过了一定的执行时间限制需要中止时；
- 多个线程做相同的事情，只要一个线程成功其它线程都可以取消时；
- 一组线程中的一个或多个出现错误导致整组都无法继续时；
- 当一个应用或服务需要停止时。

一个线程在未正常结束之前，被强制终止是很危险的事情. 因为它可能带来完全预料不到的严重后果，比如会带着自己所持有的锁而永远的休眠，迟迟不归还锁等。

那么不能直接把一个线程搞挂掉，但有时候又有必要让一个线程死掉，或者让它结束某种等待的状态该怎么办呢？优雅的方法就是， ***使用等待/通知机制给那个线程一个中断信号，让它自己决定该怎么办。***

一些补充：此外，类库中的有些类的方法也可能会调用中断，如 FutureTask 中的 cancel

方法，如果传入的参数为 true，它将会在正在运行异步任务的线程上调用 interrupt 方法，如果正在执行的异步任务中的代码没有对中断做出响应，那么 cancel 方法中的参数将不会起到什么效果；又如 ThreadPoolExecutor 中的 shutdownNow 方法会遍历线程池中的工作线程并调用线程的 interrupt 方法来中断线程，所以如果工作线程中正在执行的任务没有对中断做出响应，任务将一直执行直到正常结束。

47. 一些网上面试题

【more】

48. AQS。。。

49. Mysql 事务隔离级别

50. Java 实现折半查找

```
1.  import java.util.Comparator;
2.
3.  public class MyUtil {
4.
5.     public static <T extends Comparable<T>> int binarySearch(T[] x, T key) {

6.        return binarySearch(x, 0, x.length- 1, key);
7.     }
8.
9.     // 使用循环实现的二分查找
10.    public static <T> int binarySearch(T[] x, T key, Comparator<T> comp) {
11.       int low = 0;
12.       int high = x.length - 1;
13.       while (low <= high) {
14.           int mid = (low + high) >>> 1;
15.           int cmp = comp.compare(x[mid], key);
16.           if (cmp < 0) {
17.             low= mid + 1;
18.           }
19.           else if (cmp > 0) {
20.             high= mid - 1;
21.           }
22.           else {
23.             return mid;
24.           }
25.       }
26.       return -1;
```

```
27.    }
28.
29.    // 使用递归实现的二分查找
30.    private static<T extends Comparable<T>> int binarySearch(T[] x, int low,
    int high, T key) {
31.        if(low <= high) {
32.          int mid = low + ((high -low) >> 1);
33.          if(key.compareTo(x[mid])== 0) {
34.             return mid;
35.          }
36.          else if(key.compareTo(x[mid])< 0) {
37.             return binarySearch(x,low, mid - 1, key);
38.          }
39.          else {
40.             return binarySearch(x,mid + 1, high, key);
41.          }
42.        }
43.        return -1;
44.    }
45. }
```

51. 222

52. 333

53. String 和 StringBuilder、StringBuffer 的区别

54. String s = new String("xyz");创建了几个字符串对象

两个对象，一个是静态区的"xyz"，一个是用 new 创建在堆上的对象。

55. 常见的运行时异常

- ArithmeticException（算术异常）
- ClassCastException （类转换异常）
- IllegalArgumentException （非法参数异常）
- IndexOutOfBoundsException （下标越界异常）
- NullPointerException （空指针异常）
- SecurityException （安全异常）

56. 用 java 的套接字实现一个多线程的回显（echo）服务器