Java 最常见面试题

1 Java 基础

1. JDK 和 JRE 有什么区别？

JDK：Java Development Kit 的简称，java 开发工具包，提供了 java 的开发环境和运行环境。
JRE：Java Runtime Environment 的简称，java 运行环境，为 java 的运行提供了所需环境。
具体来说 JDK 其实包含了 JRE，同时还包含了编译 java 源码的编译器 javac，还包含了很多 java 程序调试和分析的工具。简单来说：如果你需要运行 java 程序，只需安装 JRE 就可以了，如果你需要编写 java 程序，需要安装 JDK。

2. == 和 equals 的区别是什么？

== 解读
对于基本类型和引用类型 == 的作用效果是不同的，如下所示：
基本类型：比较的是值是否相同；
引用类型：比较的是引用是否相同；

代码示例：

```
String x = "string";
String y = "string";
String z = new String("string");
System.out.println(x==y); // true
System.out.println(x==z); // false
System.out.println(x.equals(y)); // true
System.out.println(x.equals(z)); // true
```

代码解读：因为 x 和 y 指向的是同一个引用，所以 == 也是 true，而 new String()方法则重写开辟了内存空间，所以 == 结果为 false，而 equals 比较的一直是值，所以结果都为 true。
equals 解读
equals 本质上就是 ==，只不过 String 和 Integer 等重写了 equals 方法，把它变成了值比较。看下面的代码就明白了。
首先来看默认情况下 equals 比较一个有相同值的对象，代码如下：

```
class Cat {
    public Cat(String name) {
        this.name = name;
```

```
    }
    private String name;
    public String getName() {
           return name;
     }
    public void setName(String name) {
          this.name = name;
    }
}
Cat c1 = new Cat("王磊");Cat c2 = new Cat("王磊");
    System.out.println(c1.equals(c2)); // false
```

输出结果出乎我们的意料，竟然是 false？这是怎么回事，看了 equals 源码就知道了，源码如下：

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

原来 equals 本质上就是 ==。

那问题来了，两个相同值的 String 对象，为什么返回的是 true？代码如下：

```
String s1 = new String("老王");
String s2 = new String("老王");
System.out.println(s1.equals(s2)); // true
```

同样的，当我们进入 String 的 equals 方法，找到了答案，代码如下：

```
public boolean equals(Object anObject) {
   if (this == anObject) {
        return true;
 }
  if (anObject instanceof String) {
       String anotherString = (String)anObject;
       int n = value.length;
       if (n == anotherString.value.length) {
          char v1[] = value;
          char v2[] = anotherString.value;
          int i = 0;
           while (n-- != 0) {
               if (v1[i] != v2[i])
                   return false;
               i++;
           }
           return true;
         }
```

```
    }
  return false;
  }
```

原来是 String 重写了 Object 的 equals 方法，把引用比较改成了值比较。

总结：== 对于基本类型来说是值比较，对于引用类型来说是比较的是引用；而 equals 默认情况下是引用比较，只是很多类重新了 equals 方法，比如 String、Integer 等把它变成了值比较，所以一般情况下 equals 比较的是值是否相等。

3. 两个对象的 hashCode()相同，则 equals()也一定为 true，对吗？

不对，两个对象的 hashCode()相同，equals()不一定 true。
代码示例：

```
String str1 = "通话";
String str2 = "重地";
System.out.println(String.format("str1：%d | str2：%d",  str1.hashCode(),str2.hashCode()));
System.out.println(str1.equals(str2));
```

执行的结果：
str1：1179395 | str2：1179395
false

代码解读：很显然“通话”和“重地”的 hashCode() 相同，然而 equals() 则为 false，因为在散列表中，hashCode()相等即两个键值对的哈希值相等，然而哈希值相等，并不一定能得出键值对相等。

4. final 在 java 中有什么作用？

final 修饰的类叫最终类，该类不能被继承。
final 修饰的方法不能被重写。
final 修饰的变量叫常量，常量必须初始化，初始化之后值就不能被修改。

5. java 中的 Math.round(-1.5) 等于多少？

等于 -1，因为在数轴上取值时，中间值（0.5）向右取整，所以正 0.5 是往上取整，负 0.5 是直接舍弃。

6. String 属于基础的数据类型吗？

String 不属于基础类型，基础类型有 8 种：byte、boolean、char、short、int、float、long、double，而 String 属于对象。

7. java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。
String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。
StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

8. String str="i"与 String str=new String("i")一样吗？

不一样，因为内存的分配方式不一样。String str="i"的方式，java 虚拟机会将其分配到常量池中；而 String str=new String("i") 则会被分到堆内存中。

9. 如何将字符串反转？

使用 StringBuilder 或者 stringBuffer 的 reverse() 方法。
示例代码：

```
// StringBuffer reverse
StringBuffer stringBuffer = new StringBuffer();
stringBuffer.append("abcdefg");
System.out.println(stringBuffer.reverse()); // gfedcba
// StringBuilder reverse
StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("abcdefg");
System.out.println(stringBuilder.reverse()); // gfedcba
```

10. String 类的常用方法都有那些？

indexOf()：返回指定字符的索引。
charAt()：返回指定索引处的字符。
replace()：字符串替换。
trim()：去除字符串两端空白。
split()：分割字符串，返回一个分割后的字符串数组。
getBytes()：返回字符串的 byte 类型数组。
length()：返回字符串长度。
toLowerCase()：将字符串转成小写字母。
toUpperCase()：将字符串转成大写字符。
substring()：截取字符串。
equals()：字符串比较。

11. 抽象类必须要有抽象方法吗？

不需要，抽象类不一定非要有抽象方法。
示例代码：

```
abstract class Cat {
   public static void sayHi() {
        System.out.println("hi~");
   }
}
```

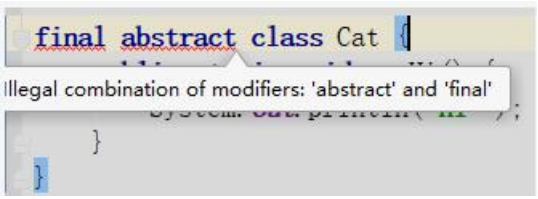
上面代码，抽象类并没有抽象方法但完全可以正常运行。

12. 普通类和抽象类有哪些区别？

普通类不能包含抽象方法，抽象类可以包含抽象方法。
抽象类不能直接实例化，普通类可以直接实例化。

13. 抽象类能使用 final 修饰吗？

不能，定义抽象类就是让其他类继承的，如果定义为 final 该类就不能被继承，这样彼此就会产生矛盾，所以 final 不能修饰抽象类，如下图所示，编辑器也会提示错误信息：



14. 接口和抽象类有什么区别？

实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。
构造函数：抽象类可以有构造函数；接口不能有。
main 方法：抽象类可以有 main 方法，并且我们能运行它；接口不能有 main 方法。
实现数量：类可以实现很多个接口；但是只能继承一个抽象类。
访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。

15. public、private 和 protected 的区别

1、public：public 表明该数据成员、成员函数是对所有用户开放的，所有用户都可以直接进行调用
2、private：private 表示私有，私有的意思就是除了 class 自己之外，任何人都不可以直接使用，私有财产神圣不可侵犯嘛，即便是子女，朋友，都不可以使用。
3、protected：protected 对于子女、朋友来说，就是 public 的，可以自由使用，没有任何限制，而对于其他的外部 class，protected 就变成 private。

作用域	当前类	同一 package	子孙类	其他 package
public	√	√	√	√
protected	√	√	√	×
default	√	√	×	×
Private	√	×	×	×

16. java 中 IO 流分为几种？

按功能来分：输入流（input）、输出流（output）。
按类型来分：字节流和字符流。
字节流和字符流的区别是：字节流按 8 位传输以字节为单位输入输出数据，字符流按 16 位传输以字符为单位输入输出数据。

字节流：继承自 InputStream 和 OutputStream

字符流：继承自 InputStreamReader 和 OutputStreamWriter

17.字节流有字符流的区别？

字节流操作的最基本的单元是字节，字符流操作的基本单位是字符，也就是 Unicode 码元。

字节流默认不使用缓冲区，字符流使用缓冲区。

在硬盘上的所有文件都是以字节形式存在的（图片，声音，视频），而字符只在内存中才会形成。

17. BIO、NIO、AIO 有什么区别？

BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

17. Files 的常用方法都有哪些？

Files.exists()：检测文件路径是否存在。
Files.createFile()：创建文件。
Files.createDirectory()：创建文件夹。
Files.delete()：删除一个文件或目录。
Files.copy()：复制文件。
Files.move()：移动文件。
Files.size()：查看文件个数。
Files.read()：读取文件。
Files.write()：写入文件。

2 容器

18.Collection 和 Collections 有什么区别？

Collection 是集合类的上级接口，继承于它的接口主要有 Set 和 List。
Collections 是针对集合类的一个帮助类，它提供了一系列静态方法实现了对各种集合的排序，搜索和线程安全等操作,而且这个类不能被实例化

19.List、Set、Map 之间的区别是什么？

比较	List	Set	Map
继承接口	Collection	Collection	
常见实现类	AbstractList(其常用子类有ArrayList、LinkedList、Vector)	AbstractSet(其常用子类有HashSet、LinkedHashSet、TreeSet)	HashMap、HashTable
常见方法	add( )、remove( )、clear( )、get( )、contains( )、size( )	add( )、remove( )、clear( )、contains( )、size( )	put( )、get( )、remove( )、clear( )、containsKey( )、containsValue( )、keySet( )、values( )、size( )
元素	可重复	不可重复(用 equals() 判断)	不可重复
顺序	有序	无序(实际上由HashCode决定)	
线程安全	Vector线程安全		Hashtable线程安全

21.HashMap 和 Hashtable 有什么区别？

（1）继承父类不同：Hashtable 继承 Dictionary 类，而 HashMap 继承 AbstractMap 类，但是二者都继承了 Map 接口

（2）线程安全性不同：Hashtable 是线程安全的，因为它的每个方法都加入了 Synchronize，对整个 Table 加了锁。

HashMap 是线程不安全的。HashMap 底层是一个 Entry 数组，当发生 hash 冲突的时候，hashmap 是采用链表的方式来解决的，在对应的数组位置存放链表的头结点。对链表而言，新加入的节点会从头结点加入。

我们来分析一下多线程访问：

1）在 hashmap 做 put 操作的时候会调用 addEntry 方法：

现在假如 A 线程和 B 线程同时对同一个桶调用 addEntry，两个线程会同时得到现在的头结点，然后 A 写入新的头结点之后，B 也写入新的头结点，那 B 的写入操作就会覆盖 A 的写入操作造成 A 的写入操作丢失。

2）**调用删除键值对方法**removeEntryForKey(Object key)

多个线程对同一个桶操作时，同时得到数组中的头结点，并发访问和修改，会造成修改覆盖

3）put 新的键值对后，若键值对 总数量 超过门限值的时候会调用一个 resize 操作：

这个操作会新生成一个新的容量的数组，然后对原数组的所有键值对重新进行计算和写入新的数组，之后指向新生成的数组。

当多个线程同时检测到总数量超过门限值的时候就会同时调用 resize 操作，各自生成新的数组并 rehash 后赋给该 map 底层的数组 table，结果最终只有最后一个线程生成的新数组被赋给 table 变量，其他线程的均会丢失。而且当某些线程已经完成赋值而其他线程刚开始的时候，就会用已经被赋值的 table 作为原始数组，这样也会有问题。

（3）HashMap 把 Hashtable 的 contains 方法去掉了，改成 containsValue 和 containsKey。Hashtable 则保留了 contains，containsValue 和 containsKey 三个方法，其中 contains 和 containsValue 功能相同。

（4）HashMap 中，null 可以作为键，这样的键只有一个；可以有一个或多个键所对应的值为 null。当 get()方法返回 null 值时，可能是 HashMap 中没有该键，也可能使该键所对应的值为 null。因此，在 HashMap 中不能由 get()方法来判断 HashMap 中是否存在某个键， 而应该用 containsKey()方法来判断。

Hashtable 中，key 和 value 都不允许出现 null 值。

（5）两个遍历方式的内部实现上不同
HashMap 使用 Iterator。
Hashtable 使用 Iterator，还使用了 Enumeration 的方式 。

（6）hash 值不同
HashTable 直接使用对象的 hashCode。而 HashMap 重新计算 hash 值。
hashCode 是 jdk 根据对象的地址或者字符串或者数字算出来的 int 类型的数值。

HashMap 重新计算了 key 的 hash 值，HashMap 在求位置索引时，则用与运算，即 hash&(length-1)

Hashtable 计算 hash 值：直接用 key 的 hashCode()，Hashtable 在求 hash 值对应的位置索引时，用取模运算， 且这里一般先用 hash&0x7FFFFFFF 后，再对 length 取模，&0x7FFFFFFF 的目的是为了将负的 hash 值转化为正值，因为 hash 值有可能为负数，而&0x7FFFFFFF 后，只有符号外改变，而后面的位都不变。

（7）内部实现使用的数组初始化和扩容方式不同
HashTable 初始默认容量为 11，Hashtable 不要求 底层数组的容量一定要为 2 的整数次幂，Hashtable 扩容时，将容量变为原来的 2 倍加 1，
而 HashMap 初始默认容量为为 16，而 HashMap 则要求一定为 2 的整数次幂，而 HashMap 扩容时，将容量变为原来的 2 倍。

21HashMap 和 ConcurrentHashMap 的区别
HashMap 是线程不安全的，ConcurrentHashMap 是线程安全的，他和 hashTable 不同，hashTable 是每个方法都加了 synchronized，而 ConcurrenthashMap 引入了分段锁的概念，就是把 Map 分成了 N 个 Segment，put 和 get 的时候，都是现根据 key.hashCode()算出放到哪个 Segment 中。

Segments 的初始化长度是 16

20. 解决 Hash 碰撞冲突方法

开放地址法
开放地执法有一个公式:Hi=(H(key)+di) MOD m i=1,2,…,k(k<=m-1)
其中，m 为哈希表的表长。di 是产生冲突的时候的增量序列。如果 di 值可能为 1,2,3,…m-1，称线性探测再散列。
如果 di 取 1，则每次冲突之后，向后移动 1 个位置.如果 di 取值可能为 1,-1,2,-2,4,-4,9,-9,16,-16,…k*k,-k*k(k<=m/2)，称二次探测再散列。
如果 di 取值可能为伪随机数列。称伪随机探测再散列。

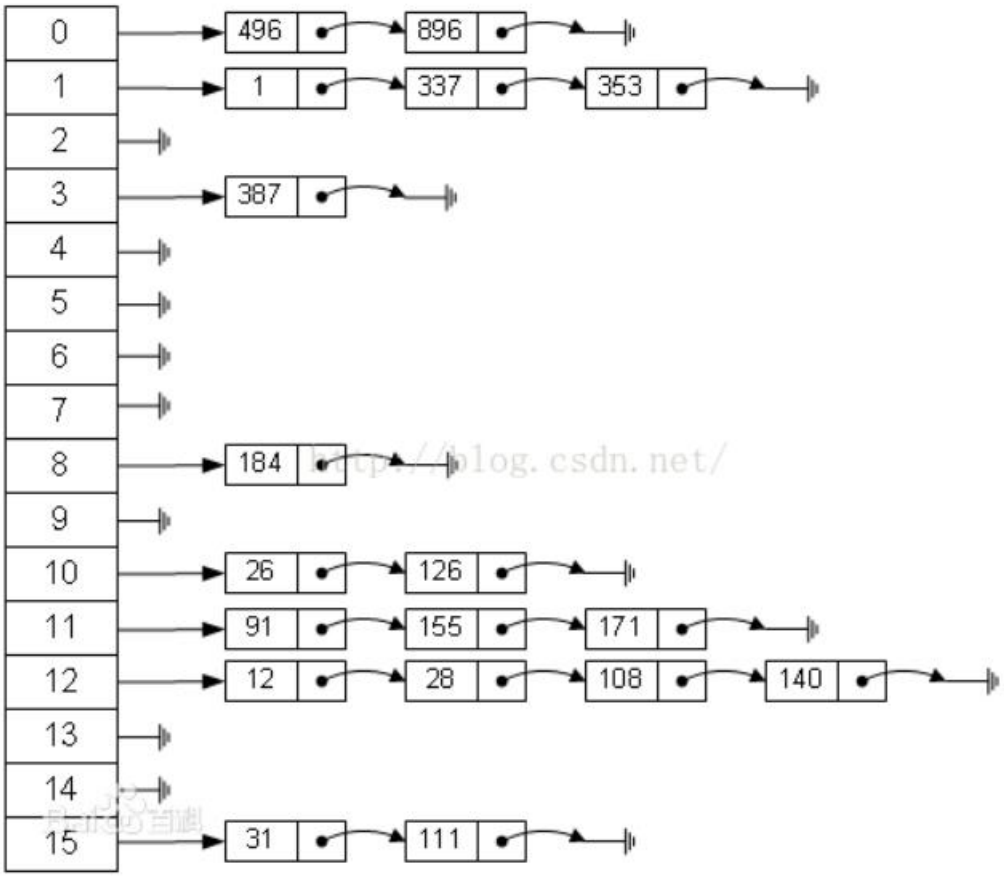
再哈希法

当发生冲突时，使用第二个、第三个、哈希函数计算地址，直到无冲突时。缺点：计算时间增加。

比如上面第一次按照姓首字母进行哈希，如果产生冲突可以按照姓字母首字母第二位进行哈希，再冲突，第三位，直到不冲突为止

链地址法（拉链法）

将所有关键字为同义词的记录存储在同一线性链表中。如下：



因此这种方法，可以近似的认为是筒子里面套筒子

建立一个公共溢出区

假设哈希函数的值域为[0,m-1],则设向量 HashTable[0..m-1]为基本表，另外设立存储空间向量 OverTable[0..v]用以存储发生冲突的记录。

拉链法的优缺点：

优点：

①拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

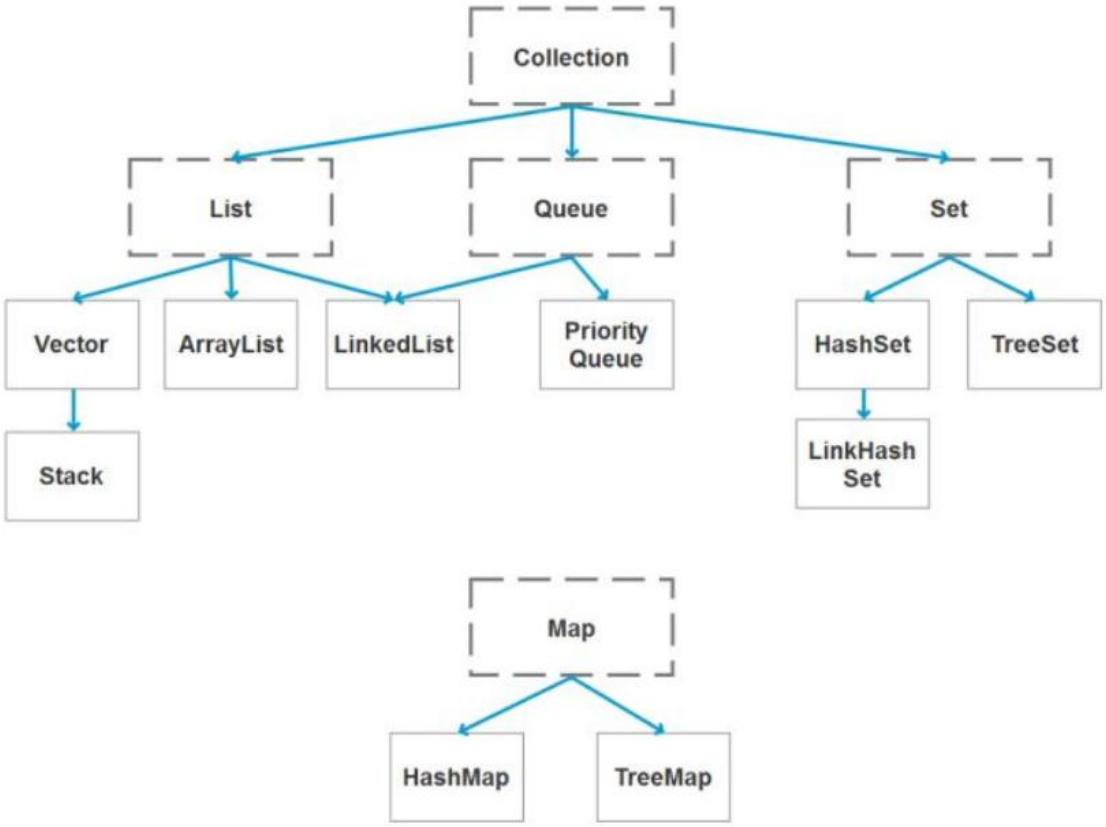
②由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

③开放定址法为减少冲突，要求装填因子α较小，故当结点规模较大时会浪费很多空间。而拉链法中可取α≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间；
④在用拉链法构造的散列表中，删除结点的操作易于实现。只要简单地删去链表上相应的结点即可。而对开放地址法构造的散列表，删除结点不能简单地将被删结 点的空间置为空，否则将截断在它之后填人散列表的同义词结点的查找路径。这是因为各种开放地址法中，空地址单元(即开放地址)都是查找失败的条件。因此在 用开放地址法处理冲突的散列表上执行删除操作，只能在被删结点上做删除标记，而不能真正删除结点。
缺点：
指针需要额外的空间，故当结点规模较小时，开放定址法较为节省空间，而若将节省的指针空间用来扩大散列表的规模，可使装填因子变小，这又减少了开放定址法中的冲突，从而提高平均查找速度。

21. java 容器都有哪些？

JAVA 的容器包括如下：
List,Map,Set ， Collection ， List ， LinkedList ， ArrayList ， Vector ， Stack ， Set
Map ， Hashtable ， HashMap
数据容器主要分为了两类：
Collection: 存放独立元素的序列。
Map：存放 key-value 型的元素对。（这对于需要利用 key 查找 value 的程序十分的重要！）
从类体系图中可以看出，Collection 定义了 Collection 类型数据的最基本、最共性的功能接口，而 List 对该接口进行了拓展。

最为常用的四个容器：
LinkedList ：其数据结构采用的是链表，此种结构的优势是删除和添加的效率很高，但随机访问元素时效率较 ArrayList 类低。
ArrayList：其数据结构采用的是线性表，此种结构的优势是访问和查询十分方便，但添加和删除的时候效率很低。
HashSet: Set 类不允许其中存在重复的元素（集），无法添加一个重复的元素（Set 中已经存在）。HashSet 利用 Hash 函数进行了查询效率上的优化，其 contain（）方法经常被使用，以用于判断相关元素是否已经被添加过。
HashMap: 提供了 key-value 的键值对数据存储机制，可以十分方便的通过键值查找相应的元素，而且通过 Hash 散列机制，查找十分的方便。



23.如何决定使用 HashMap 还是 TreeMap？

对于在 Map 中插入、删除和定位元素这类操作，HashMap 是最好的选择。然而，假如你需要对一个有序的 key 集合进行遍历，TreeMap 是更好的选择。基于你的 collection 的大小，也许向 HashMap 中添加元素会更快，将 map 换为 TreeMap 进行有序 key 的遍历。

23.说一下 HashMap 的实现原理？

HashMap 概述： HashMap 是基于哈希表的 Map 接口的非同步实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

HashMap 的数据结构： 在 java 编程语言中，最基本的结构就是两种，一个是数组，另外一个是模拟指针（引用），所有的数据结构都可以用这两个基本结构来构造的，HashMap 也不例外。HashMap 实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

当我们往 Hashmap 中 put 元素时,首先根据 key 的 hashcode 重新计算 hash 值,根绝 hash 值得到这个元素在数组中的位置(下标),如果该数组在该位置上已经存放了其他元素,那么在这个位置上的元素将以链表的形式存放,新加入的放在链头,最先加入的放入链尾.如果数组中该位置没有元素,就直接将该元素放到数组的该位置上。

需要注意 Jdk 1.8 中对 HashMap 的实现做了优化,当链表中的节点数据超过八个之后,该链表会转为红黑树来提高查询效率,从原来的 O(n)到 O(logn)

24.说一下 HashSet 的实现原理？

HashSet 底层由 HashMap 实现
HashSet 的值存放于 HashMap 的 key 上
HashMap 的 value 统一为 PRESENT

25 HashSet 和 TreeSet 的区别

1. HashSet 是通过 HashMap 实现的,TreeSet 是通过 TreeMap 实现的,只不过 Set 用的只是 Map 的 key
2. Map 的 key 和 Set 都有一个共同的特性就是集合的唯一性.TreeMap 更是多了一个排序的功能.
3. hashCode 和 equal()是 HashMap 用的, 因为无需排序所以只需要关注定位和唯一性即可.
    a. hashCode 是用来计算 hash 值的,hash 值是用来确定 hash 表索引的.
    b. hash 表中的一个索引处存放的是一张链表, 所以还要通过 equal 方法循环比较链上的每一个对象 才可以真正定位到键值对应的 Entry.
    c. put 时,如果 hash 表中没定位到,就在链表前加一个 Entry,如果定位到了,则更换 Entry 中的 value,并返回旧 value
4. 由于 TreeMap 需要排序,所以需要一个 Comparator 为键值进行大小比较.当然也是用 Comparator 定位的.
    a. Comparator 可以在创建 TreeMap 时指定
    b. 如果创建时没有确定,那么就会使用 key.com

25.ArrayList 和 LinkedList 的区别是什么？

区别是 ArrrayList 底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构是双向循环链表，不支持随机访问。使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。

26.如何实现数组和 List 之间的转换？

List 转换成为数组：调用 ArrayList 的 toArray 方法。
数组转换成为 List：调用 Arrays 的 asList 方法

27.ArrayList 和 Vector 的区别是什么？

Vector 是同步的，而 ArrayList 不是。然而，如果你寻求在迭代的时候对列表进行改变，你应

该使用 CopyOnWriteArrayList。
ArrayList 比 Vector 快，它因为有同步，不会过载。
ArrayList 更加通用，因为我们可以使用 Collections 工具类轻易地获取同步列表和只读列表。

28. Array 和 ArrayList 有何区别？

Array 可以容纳基本类型和对象，而 ArrayList 只能容纳对象。
Array 是指定大小的，而 ArrayList 大小是固定的。
Array 没有提供 ArrayList 那么多功能，比如 addAll、removeAll 和 iterator 等。

29. 在 Queue 中 poll()和 remove()有什么区别？

poll() 和 remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

30. 哪些集合类是线程安全的？

vector：就比 arraylist 多了个同步化机制（线程安全），因为效率较低，现在已经不太建议使用。在 web 应用中，特别是前台页面，往往效率（页面响应速度）是优先考虑的。
statck：堆栈类，先进后出。
hashtable：就比 hashmap 多了个线程安全。
enumeration：枚举，相当于迭代器。

31. 迭代器 Iterator 是什么？

迭代器是一种设计模式，它是一个对象，它可以遍历并选择序列中的对象，而开发人员不需要了解该序列的底层结构。迭代器通常被称为“轻量级”对象，因为创建它的代价小

32. Iterator 怎么使用？有什么特点？

Java 中的 Iterator 功能比较简单，并且只能单向移动：
(1) 使用方法 iterator()要求容器返回一个 Iterator。第一次调用 Iterator 的 next()方法时，它返回序列的第一个元素。注意：iterator()方法是 java.lang.Iterable 接口,被 Collection 继承。
(2) 使用 next()获得序列中的下一个元素。
(3) 使用 hasNext()检查序列中是否还有元素。
(4) 使用 remove()将迭代器新返回的元素删除。
Iterator 是 Java 迭代器最简单的实现，为 List 设计的 ListIterator 具有更多的功能，它可以从两个方向遍历 List，也可以从 List 中插入和删除元素。

33. Iterator 和 ListIterator 有什么区别？

Iterator 可用来遍历 Set 和 List 集合，但是 ListIterator 只能用来遍历 List。
Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。
ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和后一个元素的索引

34.怎么确保一个集合不能被修改？

3 多线程

35. 并行和并发有什么区别？

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。
并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。
在一台处理器上“同时”处理多个任务，在多台处理器上同时处理多个任务。如 hadoop 分布式集群。
所以并发编程的目标是充分的利用处理器的每一个核，以达到最高的处理性能。

36. 线程和进程的区别？

线程：是程序执行流的最小单元，是系统独立调度和分配 CPU（独立运行）的基本单位
进程：是资源分配的基本单位。一个进程包括多个线程
区别：地址空间、资源拥有
（1）线程与资源分配无关，它属于某一个进程，并与进程内的其他线程一起共享进程的资源
（2）每个进程都有自己一套独立的资源（数据），供其内的所有线程共享
（3）不论是大小，开销线程要更“轻量级”
（4）一个进程内的线程通信比进程之间的通信更快速，有效。（因为共享变量）

37. 守护线程是什么？

守护线程（即 daemon thread），是个服务线程，准确地来说就是服务其他的线程。
举例：垃圾回收线程

38. 创建线程有哪几种方式？

①. 继承 Thread 类创建线程类
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。
创建 Thread 子类的实例，即创建了线程对象。
调用线程对象的 start()方法来启动该线程。
②. 通过 Runnable 接口创建线程类
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。
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。
调用线程对象的 start()方法来启动该线程。
③. 通过 Callable 和 Future 创建线程
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。
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。
使用 FutureTask 对象作为 Thread 对象的 target 创建并启动新线程。
调用 FutureTask 对象的 get()方法来获得子线程执行结束后的返回值。

39. 说一下 runnable 和 callable 有什么区别？

有点深的问题了，也看出一个 Java 程序员学习知识的广度。

Runnable 接口中的 run()方法的返回值是 void，它做的事情只是纯粹地去执行 run()方法中的代码而已；

Callable 接口中的 call()方法是有返回值的，是一个泛型，和 Future、FutureTask 配合可以用来获取异步执行的结果。

40. 线程有哪些状态？

线程通常都有五种状态，创建、就绪、运行、阻塞和死亡。

创建状态。在生成线程对象，并没有调用该对象的 start 方法，这是线程处于创建状态。

就绪状态。当调用了线程对象的 start 方法之后，该线程就进入了就绪状态，但是此时线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行之后，从等待或者睡眠中回来之后，也会处于就绪状态。

运行状态。线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行 run 函数当中的代码。

阻塞状态。线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend，wait 等方法都可以导致线程阻塞。

死亡状态。如果一个线程的 run 方法执行结束或者调用 stop 方法后，该线程就会死亡。对于已经死亡的线程，无法再使用 start 方法令其进入就绪

41. sleep() 和 wait() 有什么区别？

sleep()：方法是线程类（Thread）的静态方法，让调用线程进入睡眠状态，让出执行机会给其他线程，等到休眠时间结束后，线程进入就绪状态和其他线程一起竞争 cpu 的执行时间。因为 sleep() 是 static 静态的方法，他不能改变对象的机锁，当一个 synchronized 块中调用了 sleep() 方法，线程虽然进入休眠，但是对象的机锁没有被释放，其他线程依然无法访问这个对象。

wait()：wait()是 Object 类的方法，当一个线程执行到 wait 方法时，它就进入到一个和该对象相关的等待池，同时释放对象的机锁，使得其他线程能够访问，可以通过 notify，notifyAll 方法来唤醒等待的线程

42. notify()和 notifyAll()有什么区别？

如果线程调用了对象的 wait()方法，那么线程便会处于该对象的等待池中，等待池中的线程不会去竞争该对象的锁。

当有线程调用了对象的 notifyAll()方法（唤醒所有 wait 线程）或 notify()方法（只随机唤醒一个 wait 线程），被唤醒的的线程便会进入该对象的锁池中，锁池中的线程会去竞争该对象锁。也就是说，调用了 notify 后只要一个线程会由等待池进入锁池，而 notifyAll 会将该对象等待池内的所有线程移动到锁池中，等待锁竞争。

优先级高的线程竞争到对象锁的概率大，假若某线程没有竞争到该对象锁，它还会留在锁池中，唯有线程再次调用 wait()方法，它才会重新回到等待池中。而竞争到对象锁的线程则继续往下执行，直到执行完了 synchronized 代码块，它会释放掉该对象锁，这时锁池中的线程会继续竞争该对象锁。

43. 线程的 run()和 start()有什么区别？

每个线程都是通过某个特定 Thread 对象所对应的方法 run()来完成其操作的，方法 run()称为线程体,是线程任务的运行者。通过调用 Thread 类的 start()方法来启动一个线程。

start()方法来启动一个线程，真正实现了多线程运行。这时无需等待 run 方法体代码执行完毕，可以直接继续执行下面的代码； 这时此线程是处于就绪状态， 并没有运行。 然后通过此 Thread 类调用方法 run()来完成其运行状态， 这里方法 run()称为线程体，它包含了要执行的这个线程的内容， Run 方法运行结束， 此线程终止。然后 CPU 再调度其它线程。

run()方法是在本线程里的，只是线程里的一个函数,而不是多线程的。 如果直接调用 run(),其实就相当于是调用了一个普通函数而已，直接调用 run()方法必须等待 run()方法执行

完毕才能执行下面的代码，所以执行路径还是只有一条，根本就没有线程的特征，所以在多线程执行时要使用 start()方法而不是 run()方法。

44. 创建线程池有哪几种方式？

①. newFixedThreadPool(int nThreads)

创建一个固定长度的线程池，每当提交一个任务就创建一个线程，直到达到线程池的最大数量，这时线程规模将不再变化，当线程发生未预期的错误而结束时，线程池会补充一个新的线程。

②. newCachedThreadPool()

创建一个可缓存的线程池，如果线程池的规模超过了处理需求，将自动回收空闲线程，而当需求增加时，则可以自动添加新线程，线程池的规模不存在任何限制。

③. newSingleThreadExecutor()

这是一个单线程的 Executor，它创建单个工作线程来执行任务，如果这个线程异常结束，会创建一个新的来替代它；它的特点是能确保依照任务在队列中的顺序来串行执行。

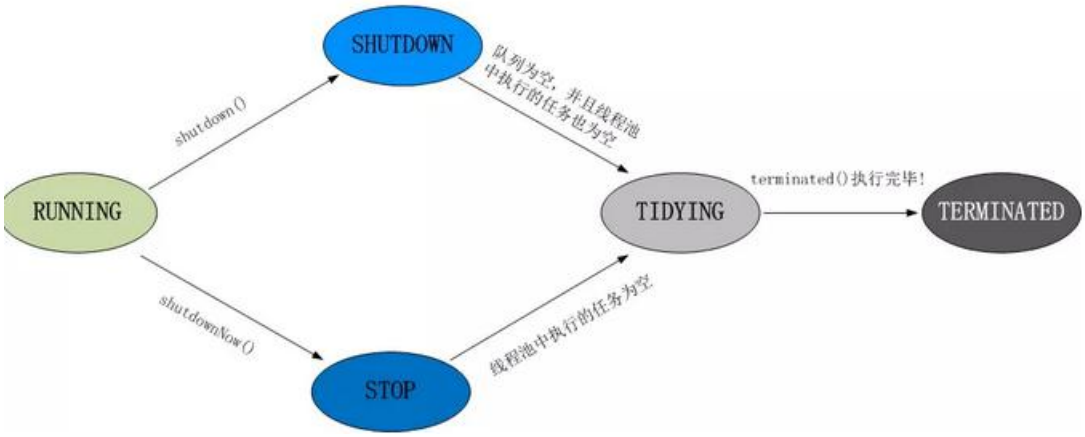
④. newScheduledThreadPool(int corePoolSize)

创建了一个固定长度的线程池，而且以延迟或定时的方式来执行任务，类似于 Timer。

45. 线程池都有哪些状态？

线程池有 5 种状态：Running、ShutDown、Stop、Tidying、Terminated。

线程池各个状态切换框架图：



46. 线程池中 submit()和 execute()方法有什么区别？

接收的参数不一样

submit 有返回值，而 execute 没有

submit 方便 Exception 处理

47. 在 java 程序中怎么保证多线程的运行安全？

线程安全在三个方面体现：

原子性：提供互斥访问，同一时刻只能有一个线程对数据进行操作，（atomic,synchronized）；

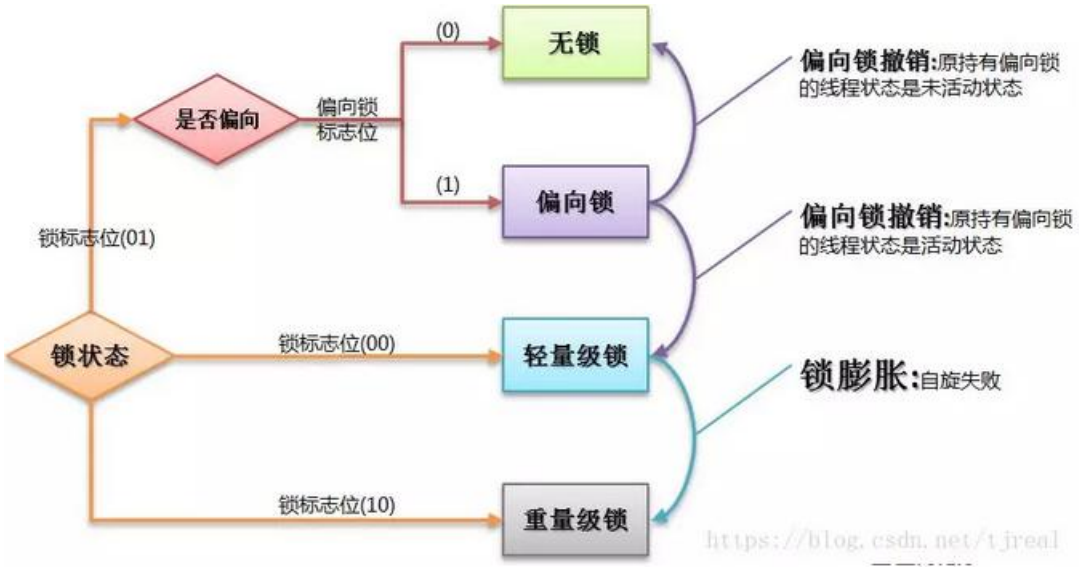
可见性：一个线程对主内存的修改可以及时地被其他线程看到，（synchronized,volatile）；

有序性：一个线程观察其他线程中的指令执行顺序，由于指令重排序，该观察结果一般杂乱无序，（happens-before 原则）。

48. 多线程锁的升级原理是什么？

在 Java 中，锁共有 4 种状态，级别从低到高依次为：无状态锁，偏向锁，轻量级锁和重量级锁状态，这几个状态会随着竞争情况逐渐升级。锁可以升级但不能降级。

锁升级的图示过程：



49. 什么是死锁？

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。是操作系统层面的一个错误，是进程死锁的简称，最早在 1965 年由 Dijkstra 在研究银行家算法时提出的，它是计算机操作系统乃至整个并发程序设计领域最难处理的问题之一。

50. 怎么防止死锁？

死锁的四个必要条件：

互斥条件：进程对所分配到的资源不允许其他进程进行访问，若其他进程访问该资源，只能等待，直至占有该资源的进程使用完成后释放该资源

请求和保持条件：进程获得一定的资源之后，又对其他资源发出请求，但是该资源可能被其他进程占有，此事请求阻塞，但又对自己获得的资源保持不放

不可剥夺条件：是指进程已获得的资源，在未完成使用之前，不可被剥夺，只能在使用完后自己释放

环路等待条件：是指进程发生死锁后，若干进程之间形成一种头尾相接的循环等待资源关系

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

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和解除死锁。

所以，在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确 定资源的合理分配算法，避免进程永久占据系统资源。

此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。

51. ThreadLocal 是什么？有哪些使用场景？

线程局部变量是局限于线程内部的变量，属于线程自身所有，不在多个线程间共享。Java 提供 ThreadLocal 类来支持线程局部变量，是一种实现线程安全的方式。但是在管理环境下（如 web 服务器）使用线程局部变量的时候要特别小心，在这种情况下，工作线程的生命周期比任何应用变量的生命周期都要长。任何线程局部变量一旦在工作完成后没有释放，Java 应用就存在内存泄露的风险。

52.说一下 synchronized 底层实现原理？

synchronized 可以保证方法或者代码块在运行时，同一时刻只有一个方法可以进入到临界区，同时它还可以保证共享变量的内存可见性。

Java 中每一个对象都可以作为锁，这是 synchronized 实现同步的基础：

普通同步方法，锁是当前实例对象

静态同步方法，锁是当前类的 class 对象

同步方法块，锁是括号里面的对象

53. synchronized 和 volatile 的区别是什么？

volatile 本质是在告诉 jvm 当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取； synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。

volatile 仅能使用在变量级别；synchronized 则可以使用在变量、方法、和类级别的。

volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修改可见性和原子性。

volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞。

volatile 标记的变量不会被编译器优化；synchronized 标记的变量可以被编译器优化。

54. synchronized 和 Lock 有什么区别？

首先 synchronized 是 java 内置关键字，在 jvm 层面，Lock 是个 java 类；

synchronized 无法判断是否获取锁的状态，Lock 可以判断是否获取到锁；

synchronized 会自动释放锁(a 线程执行完同步代码会释放锁 ；b 线程执行过程中发生异常会释放锁)，Lock 需在 finally 中手工释放锁（unlock()方法释放锁），否则容易造成线程死锁；

用 synchronized 关键字的两个线程 1 和线程 2，如果当前线程 1 获得锁，线程 2 线程等待。如果线程 1 阻塞，线程 2 则会一直等待下去，而 Lock 锁就不一定会等待下去，如果尝试获取不到锁，线程可以不用一直等待就结束了；

synchronized 的锁可重入、不可中断、非公平，而 Lock 锁可重入、可判断、可公平（两者皆可）；

Lock 锁适合大量同步的代码的同步问题，synchronized 锁适合代码少量的同步问题。

55. synchronized 和 ReentrantLock 区别是什么？

synchronized 是和 if、else、for、while 一样的关键字，ReentrantLock 是类，这是二者的本质区别。既然 ReentrantLock 是类，那么它就提供了比 synchronized 更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock 比 synchronized 的扩展性体现在几点上：

ReentrantLock 可以对获取锁的等待时间进行设置，这样就避免了死锁

ReentrantLock 可以获取各种锁的信息

ReentrantLock 可以灵活地实现多路通知

另外，二者的锁机制其实也是不一样的:ReentrantLock 底层调用的是 Unsafe 的 park 方法加锁，synchronized 操作的应该是对象头中 mark word。

56. 说一下 atomic 的原理？

Atomic 包中的类基本的特性就是在多线程环境下，当有多个线程同时对单个（包括基本类型及引用类型）变量进行操作时，具有排他性，即当多个线程同时对该变量的值进行更新时，仅有一个线程能成功，而未成功的线程可以向自旋锁一样，继续尝试，一直等到执行成功。

Atomic 系列的类中的核心方法都会调用 unsafe 类中的几个本地方法。我们需要先知道一个东西就是 Unsafe 类，全名为：sun.misc.Unsafe，这个类包含了大量的对 C 代码的操作，包括很多直接内存分配以及原子操作的调用，而它之所以标记为非安全的，是告诉你这个里面大量的方法调用都会存在安全隐患，需要小心使用，否则会导致严重的后果，例如在通过 unsafe 分配内存的时候，如果自己指定某些区域可能会导致一些类似 C++一样的指针越界到其他进程的问题。

4 反射

57. 什么是反射？

可以将一个程序（类）在运行的时候获得该程序（类）的信息的机制，也就是获得在编译期不可能获得的类的信息，因为这些信息是保存在 Class 对象中的，而这个 Class 对象是在程序运行时动态加载的

58. 什么是 java 序列化？什么情况下需要序列化？

序列化就是把 java 对象转换为字节序列的方法。

把对象的字节序列化到永久的保存到硬盘中

在网络上传递对象的字节序列

59. 动态代理是什么？有哪些应用？

动态代理是运行时动态生成代理类。

动态代理指的是可以任意控制任意对象的执行过程

本来应该自己做的事情，因为没有某种原因不能直接做，只能请别人代理做。被请的人就是代理

比如春节买票回家，由于没有时间，只能找票务中介来买，这就是代理模式

应用：Spring 的 AOP

60.怎么实现动态代理？

JDK 动态代理

cglib 动态代理

5 对象拷贝

60.为什么要使用克隆？

克隆的对象可能包含一些已经修改过的属性，而 new 出来的对象的属性都还是初始化时候的值，所以当需要一个新的对象来保存当前对象的“状态”就靠克隆方法了。

61.如何实现对象克隆？

实现 Cloneable 接口并重写 Object 类中的 clone() 方法。

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

62.深拷贝和浅拷贝区别是什么？

浅克隆：当对象被复制时只复制它本身和其中包含的值类型的成员变量，而引用类型的成员对象并没有复制。

深克隆：除了对象本身被复制外，对象所包含的所有成员变量也将复制。

6 Java Web 模块

64. jsp 和 servlet 有什么区别？

（1）jsp 经编译后就变成了 Servlet.（JSP 的本质就是 Servlet，JVM 只能识别 java 的类，不能识别 JSP 的代码，Web 容器将 JSP 的代码编译成 JVM 能够识别的 java 类）

（2）jsp 更擅长表现于页面显示，servlet 更擅长于逻辑控制。

（3）Servlet 中没有内置对象，Jsp 中的内置对象都是必须通过 HttpServletRequest 对象，HttpServletResponse 对象以及 HttpServlet 对象得到。
（4）Jsp 是 Servlet 的一种简化，使用 Jsp 只需要完成程序员需要输出到客户端的内容，Jsp 中的 Java 脚本如何镶嵌到一个类中，由 Jsp 容器完成。而 Servlet 则是个完整的 Java 类，这个类的 Service 方法用于生成对客户端的响应。

65. jsp 有哪些内置对象？作用分别是什么？

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。

66. 说一下 jsp 的 4 种作用域？

JSP 中的四种作用域包括 page、request、session 和 application，具体来说：
page 代表与一个页面相关的对象和属性。
request 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；需要在页面显示的临时数据可以置于此作用域。
session 代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的 session 中。
application 代表与整个 Web 应用程序相关的对象和属性，它实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

67. session 和 cookie 有什么区别？

（1）由于 HTTP 协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是 Session.典型的场景比如购物车，当你点击下单按钮时，由于 HTTP 协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的 Session，用用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。

这个 Session 是保存在服务端的，有一个唯一标识。在服务端保存 Session 的方法很多，内存、数据库、文件都有。集群的时候也要考虑 Session 的转移，在大型的网站，一般会有专门的 Session 服务器集群，用来保存用户会话，这个时候 Session 信息都是放在内存的，使用一些缓存服务比如 Memcached 之类的来放 Session。

（2）思考一下服务端如何识别特定的客户？这个时候 Cookie 就登场了。每次 HTTP 请求的时候，客户端都会发送相应的 Cookie 信息到服务端。实际上大多数的应用都是用 Cookie 来实现 Session 跟踪的，第一次创建 Session 的时候，服务端会在 HTTP 协议中告诉客户端，需要在 Cookie 里面记录一个 Session ID，以后每次请求把这个会话 ID 发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做 URL 重写的技术来进行会话跟踪，即每次 HTTP 交互，URL 后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。

（3）Cookie 其实还可以用在一些方便用户的场景下，设想你某次登陆过一个网站，下次登录的时候不想再次输入账号了，怎么办？这个信息可以写到 Cookie 里面，访问网站的时候，网站页面的脚本可以读取这个信息，就自动帮你把用户名给填了，能够方便一下用户。这也是 Cookie 名称的由来，给用户的一点甜头。所以，总结一下：Session 是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；Cookie 是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现 Session 的一种方式。

68. 说一下 session 的工作原理？

其实 session 是一个存在服务器上的类似于一个散列表格的文件。里面存有我们需要的信息，在我们需要用的时候可以从里面取出来。类似于一个大号的 map 吧，里面的键存储的是用户的 sessionid，用户向服务器发送请求的时候会带上这个 sessionid。这时就可以从中取出对应的值了。

69. 如果客户端禁止 cookie 能实现 session 还能用吗？

Cookie 与 Session，一般认为是两个独立的东西，Session 采用的是在服务器端保持状态的方案，而 Cookie 采用的是在客户端保持状态的方案。但为什么禁用 Cookie 就不能得到 Session 呢？因为 Session 是用 Session ID 来确定当前对话所对应的服务器 Session，而 Session ID 是通过 Cookie 来传递的，禁用 Cookie 相当于失去了 Session ID，也就得不到 Session 了。

假定用户关闭 Cookie 的情况下使用 Session，其实现途径有以下几种：
（1）设置 php.ini 配置文件中的“session.use_trans_sid = 1”，或者编译时打开打开了“--enable-trans-sid”选项，让 PHP 自动跨页传递 Session ID。
（2）手动通过 URL 传值、隐藏表单传递 Session ID。
（3）用文件、数据库等形式保存 Session ID，在跨页过程中手动调用。

70. spring mvc 和 struts 的区别是什么？

（1）拦截机制的不同

Struts2 是类级别的拦截，每次请求就会创建一个 Action，和 Spring 整合时 Struts2 的 ActionBean 注入作用域是原型模式 prototype，然后通过 setter，getter 吧 request 数据注入到属性。Struts2 中，一个 Action 对应一个 request，response 上下文，在接收参数时，可以通过属性接收，这说明属性参数是让多个方法共享的。Struts2 中 Action 的一个方法可以对应一个 url，而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了，只能设计为多例。

SpringMVC 是方法级别的拦截，一个方法对应一个 Request 上下文，所以方法直接基本上是独立的，独享 request，response 数据。而每个方法同时又何一个 url 对应，参数的传递是直接注入到方法中的，是方法所独有的。处理结果通过 ModeMap 返回给框架。在 Spring 整合时，SpringMVC 的 Controller Bean 默认单例模式 Singleton，所以默认对所有的请求，只会创建一个 Controller，有应为没有共享的属性，所以是线程安全的，如果要改变默认的作用域，需要添加@Scope 注解修改。

Struts2 有自己的拦截 Interceptor 机制，SpringMVC 这是用的是独立的 Aop 方式，这样导致 Struts2 的配置文件量还是比 SpringMVC 大。

（2）底层框架的不同

Struts2 采用 Filter（StrutsPrepareAndExecuteFilter）实现，SpringMVC（DispatcherServlet）则采用 Servlet 实现。Filter 在容器启动之后即初始化；服务停止以后坠毁，晚于 Servlet。Servlet 在是在调用时初始化，先于 Filter 调用，服务停止后销毁。

（3）性能方面

Struts2 是类级别的拦截，每次请求对应实例一个新的 Action，需要加载所有的属性值注入，SpringMVC 实现了零配置，由于 SpringMVC 基于方法的拦截，有加载一次单例模式 bean 注入。所以，SpringMVC 开发效率和性能高于 Struts2。

（4）配置方面

spring MVC 和 Spring 是无缝的。从这个项目的管理和安全上也比 Struts2 高。

71. 如何避免 sql 注入？

（1）PreparedStatement（简单又有效的方法）
（2）使用正则表达式过滤传入的参数
（3）字符串过滤
（4）JSP 中调用该函数检查是否包函非法字符
（5）JSP 页面判断代码

72. 什么是 XSS 攻击，如何避免？

XSS 攻击又称 CSS,全称 Cross Site Script （跨站脚本攻击），其原理是攻击者向有 XSS 漏洞的网站中输入恶意的 HTML 代码，当用户浏览该网站时，这段 HTML 代码会自动执行，从而达到攻击的目的。XSS 攻击类似于 SQL 注入攻击，SQL 注入攻击中以 SQL 语句作为用户输入，从而达到查询/修改/删除数据的目的，而在 xss 攻击中，通过插入恶意脚本，实现对用户游览器的控制，获取用户的一些信息。 XSS 是 Web 程序中常见的漏洞，XSS 属于被动式且用于客户端的攻击方式。
XSS 防范的总体思路是：对输入(和 URL 参数)进行过滤，对输出进行编码。

73. 什么是 CSRF 攻击，如何避免？

CSRF（Cross-site request forgery）也被称为 one-click attack 或者 session riding，中文全称是叫跨站请求伪造。一般来说，攻击者通过伪造用户的浏览器的请求，向访问一个用户自己曾经认证访问过的网站发送出去，使目标网站接收并误以为是用户的真实操作而去执行命令。常用于盗取账号、转账、发送虚假消息等。攻击者利用网站对请求的验证漏洞而实现这样的攻击行为，网站能够确认请求来源于用户的浏览器，却不能验证请求是否源于用户的真实意愿下的操作行为。

如何避免：
1. 验证 HTTP Referer 字段
HTTP 头中的 Referer 字段记录了该 HTTP 请求的来源地址。在通常情况下，访问一个安全受限页面的请求来自于同一个网站，而如果黑客要对其实施 CSRF
攻击，他一般只能在他自己的网站构造请求。因此，可以通过验证 Referer 值来防御 CSRF 攻击。

2. 使用验证码
关键操作页面加上验证码，后台收到请求后通过判断验证码可以防御 CSRF。但这种方法对用户不太友好。

3. 在请求地址中添加 token 并验证
CSRF 攻击之所以能够成功，是因为黑客可以完全伪造用户的请求，该请求中所有的用户验证信息都是存在于 cookie 中，因此黑客可以在不知道这些验证信息的情况下直接利用用户自己的 cookie 来通过安全验证。要抵御 CSRF，关键在于在请求中放入黑客所不能伪造的信息，并且该信息不存在于 cookie 之中。可以在 HTTP 请求中以参数的形式加入一个随机产生的 token，并在服务器端建立一个拦截器来验证这个 token，如果请求中没有 token 或者 token 内容不正确，则认为可能是 CSRF 攻击而拒绝该请求。这种方法要比检查 Referer 要安全一些，token 可以在用户登陆后产生并放于 session 之中，然后在每次请求时把 token 从 session 中拿出，与请求中的 token 进行比对，但这种方法的难点在于如何把 token 以参数的形式加入请求。
对于 GET 请求，token 将附在请求地址之后，这样 URL 就变成 http://url?csrftoken=tokenvalue。
而对于 POST 请求来说，要在 form 的最后加上 <input type="hidden" name="csrftoken" value="tokenvalue"/>，这样就把 token 以参数的形式加入请求了。

4. 在 HTTP 头中自定义属性并验证
这种方法也是使用 token 并进行验证，和上一种方法不同的是，这里并不是把 token 以参数的形式置于 HTTP 请求之中，而是把它放到 HTTP 头中自定义的属性里。通过 XMLHttpRequest 这个类，可以一次性给所有该类请求加上 csrftoken 这个 HTTP 头属性，并把 token 值放入其中。这样解决了上种方法在请求中加入 token 的不便，同时，通过 XMLHttpRequest 请求的地址不会被记录到浏览器的地址栏，也不用担心 token 会透过 Referer 泄露到其他网站中去。

7 异常模块

74. throw 和 throws 的区别？

throw 则是指抛出的一个具体异常类型
throws 是用来声明一个方法可能抛出的所有异常信息

75. final、finally、finalize 有什么区别？

final 是用来修饰类、方法、变量
finally 只能用在 try catch 语法中，表示这段语句最终一定会被执行

76. try-catch-finally 中哪个部分可以省略？

try-catch-finally 其中 catch 和 finally 都可以被省略，但是不能同时省略，也就是说有 try 的时候，必须后面跟一个 catch 或者 finally

77. try-catch-finally 中，如果 catch 中 return 了，finally 还会执行吗？

一定会， catch 中 return 会等 finally 中的代码执行完之后才会执行

78. 常见的异常类有哪些？

NullPointerException 空指针异常
ClassNotFoundException 指定类不存在
NumberFormatException 字符串转换为数字异常

IndexOutOfBoundsException 数组下标越界异常
ClassCastException 数据类型转换异常
FileNotFoundException 文件未找到异常
NoSuchMethodException 方法不存在异常
IOException IO 异常
SocketException Socket 异常

8 网络模块

79. http 响应码 301 和 302 代表的是什么？有什么区别

答：301，302 都是 HTTP 状态的编码，都代表着某个 URL 发生了转移。

区别：

301 redirect: 301 代表永久性转移(Permanently Moved)。
302 redirect: 302 代表暂时性转移(Temporarily Moved)。

80. forward 和 redirect 的区别？

Forward 和 Redirect 代表了两种请求转发方式：直接转发和间接转发。

直接转发方式（Forward），客户端和浏览器只发出一次请求，Servlet、HTML、JSP 或其它信息资源，由第二个信息资源响应该请求，在请求对象 request 中，保存的对象对于每个信息资源是共享的。

间接转发方式（Redirect）实际是两次 HTTP 请求，服务器端在响应第一次请求的时候，让浏览器再向另外一个 URL 发出请求，从而达到转发的目的。

举个通俗的例子：

直接转发就相当于：“A 找 B 借钱，B 说没有，B 去找 C 借，借到借不到都会把消息传递给 A”；

间接转发就相当于："A 找 B 借钱，B 说没有，让 A 去找 C 借"。

81. 简述 tcp 和 udp 的区别？

TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接。
TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付。
Tcp 通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。
UDP 具有较好的实时性，工作效率比 TCP 高，适用于对高速传输和实时性有较高的通信或广播通信。
每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信。
TCP 对系统资源要求较多，UDP 对系统资源要求较少。

82. 简述 tcp 三次握手和四次挥手？

三次握手
第一次握手：主机 A 发送同步报文段（SYN）请求建立连接。
第二次握手：主机 B 听到连接请求，就将该连接放入内核等待队列当中，并向主机 A 发送针对 SYN 的确认 ACK，同时主机 B 也发送自己的请求建立连接（SYN）。
第三次握手：主机 A 收到主机 B 发出的 SYN 给出确认应答 ACK。

四次挥手
第一次挥手：当主机 A 发送数据完毕后，发送 FIN 结束报文段。
第二次挥手：主机 B 收到 FIN 报文段后，向主机 A 发送一个确认序号 ACK（为了防止在这段时间内，对方重传 FIN 报文段）。
第三次挥手：主机 B 准备关闭连接，向主机 A 发送一个 FIN 结束报文段。
第四次挥手：主机 A 收到 FIN 结束报文段后，进入 TIME_WAIT 状态。并向主机 B 发送一个 ACK 表示连接彻底释放。

82. tcp 为什么要三次握手，两次不行吗？为什么？

如果是 2 次
①当客户端发出一个请求报文段并没有丢失，而是滞留在了某个网络节点，过了好长时间才发送到服务端，此时服务端建立连接。但是由于现在客户端并没有发出连接请求，因此不会理睬服务端的确认，而服务端以为新的连接产生，服务端的好多资源被白白浪费。
②当确认应答 ACK 总是丢失时，客户端以为服务端没有连接，它将会不断地重新请求连接，而服务端会连接大量的无效连接，给服务器增加维护成本，服务器很容易受到 SYN 洪水攻击。

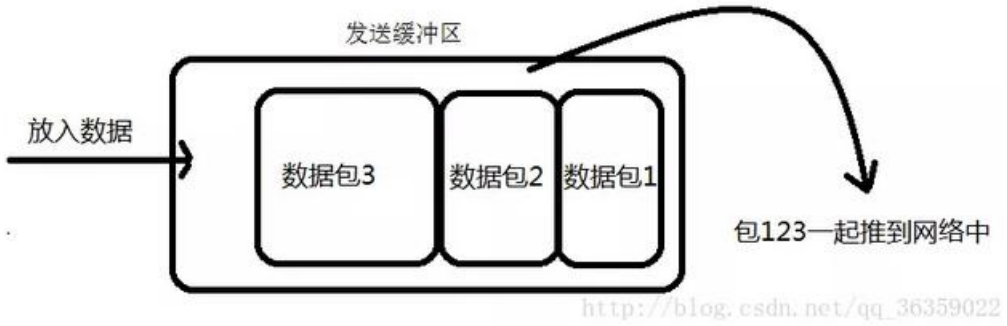
83. tcp 为什么是四次挥手？

TCP 是面向连接的，属于全双工，断开连接是双方的事情。所以是四次。
当客户端发送 FIN 结束报文段时，服务端并不会立即关闭 SOCKET，所以只能先发送一个 ACK。因为有可能此时服务端还没有发送完报文。

83. 说一下 tcp 粘包是怎么产生的？

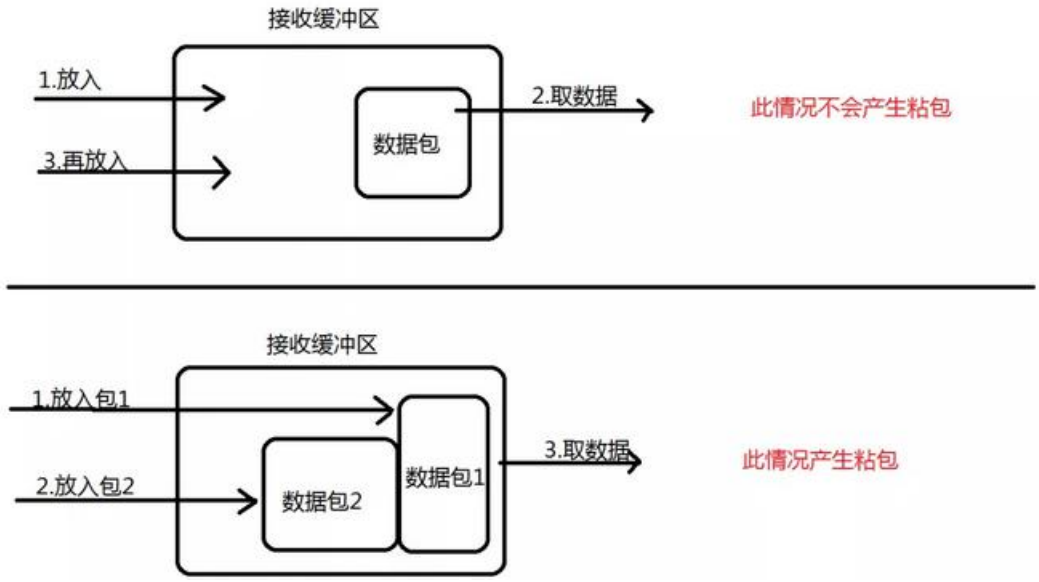
①. 发送方产生粘包

采用 TCP 协议传输数据的客户端与服务器经常是保持一个长连接的状态（一次连接发一次数据不存在粘包），双方在连接不断开的情况下，可以一直传输数据；但当发送的数据包过于的小时，那么 TCP 协议默认的会启用 Nagle 算法，将这些较小的数据包进行合并发送（缓冲区数据发送是一个堆压的过程）；这个合并过程就是在发送缓冲区中进行的，也就是说数据发送出来它已经是粘包的状态了。



②. 接收方产生粘包

接收方采用 TCP 协议接收数据时的过程是这样的：数据到底接收方，从网络模型的下方传递至传输层，传输层的 TCP 协议处理是将其放置接收缓冲区，然后由应用层来主动获取（C 语言用 recv、read 等函数）；这时会出现一个问题，就是我们在程序中调用的读取数据函数不能及时的把缓冲区中的数据拿出来，而下一个数据又到来并有一部分放入的缓冲区末尾，等我们读取数据时就是一个粘包。（放数据的速度 > 应用层拿数据速度）



84. OSI 的七层模型都有哪些？

应用层：网络服务与最终用户的一个接口。
表示层：数据的表示、安全、压缩。
会话层：建立、管理、终止会话。
传输层：定义传输数据的协议端口号，以及流控和差错校验。
网络层：进行逻辑地址寻址，实现不同网络之间的路径选择。
数据链路层：建立逻辑连接、进行硬件地址寻址、差错校验等功能。
物理层：建立、维护、断开物理连接。

85. get 和 post 请求有哪些区别？

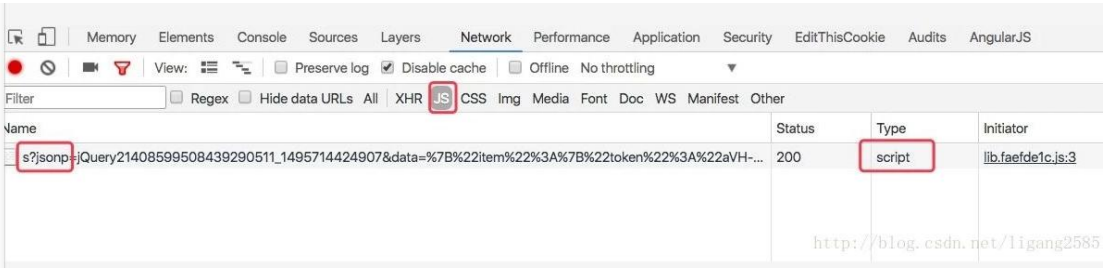
GET 在浏览器回退时是无害的，而 POST 会再次提交请求。
GET 产生的 URL 地址可以被 Bookmark，而 POST 不可以。
GET 请求会被浏览器主动 cache，而 POST 不会，除非手动设置。
GET 请求只能进行 url 编码，而 POST 支持多种编码方式。
GET 请求参数会被完整保留在浏览器历史记录里，而 POST 中的参数不会被保留。
GET 请求在 URL 中传送的参数是有长度限制的，而 POST 么有。
对参数的数据类型，GET 只接受 ASCII 字符，而 POST 没有限制
GET 比 POST 更不安全，因为参数直接暴露在 URL 上，所以不能用来传递敏感信息。
GET 参数通过 URL 传递，POST 放在 Request body 中。

86. 如何实现跨域？

方式一：图片 ping 或 script 标签跨域

图片 ping 常用于跟踪用户点击页面或动态广告曝光次数。
script 标签可以得到从其他来源数据，这也是 JSONP 依赖的根据。

方式二：JSONP 跨域
JSONP（JSON with Padding）是数据格式 JSON 的一种“使用模式”，可以让网页从别的网域要数据。根据 XmlHttpRequest 对象受到同源策略的影响，而利用 <script>元素的这个开放策略，网页可以得到从其他来源动态产生的 JSON 数据，而这种使用模式就是所谓的 JSONP。用 JSONP 抓到的数据并不是 JSON，而是任意的 JavaScript，用 JavaScript 解释器运行而不是用 JSON 解析器解析。所有，通过 Chrome 查看所有 JSONP 发送的 Get 请求都是 js 类型，而非 XHR。



缺点：
（1）只能使用 Get 请求
（2）不能注册 success、error 等事件监听函数，不能很容易的确定 JSONP 请求是否失败
（3）JSONP 是从其他域中加载代码执行，容易受到跨站请求伪造的攻击，其安全性无法确保
方式三：CORS

Cross-Origin Resource Sharing（CORS）跨域资源共享是一份浏览器技术的规范，提供了 Web 服务从不同域传来沙盒脚本的方法，以避开浏览器的同源策略，确保安全的跨域数据传输。现代浏览器使用 CORS 在 API 容器如 XMLHttpRequest 来减少 HTTP 请求的风险来源。与 JSONP 不同，CORS 除了 GET 要求方法以外也支持其他的 HTTP 要求。服务器一般需要增加如下响应头的一种或几种：

```
Access-Control-Allow-Origin:
*Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
Access-Control-Max-Age: 86400
```

跨域请求默认不会携带 Cookie 信息，如果需要携带，请配置下述参数：

```
"Access-Control-Allow-Credentials": true
// Ajax 设置
"withCredentials": true
```

方式四：window.name+iframe

window.name 通过在 iframe（一般动态创建 i）中加载跨域 HTML 文件来起作用。然后，HTML 文件将传递给请求者的字符串内容赋值给 window.name。然后，请求者可以检索 window.name 值作为响应。

（1）iframe 标签的跨域能力；

（2）window.name 属性值在文档刷新后依旧存在的能力（且最大允许 2M 左右）。

每个 iframe 都有包裹它的 window，而这个 window 是 top window 的子窗口。contentWindow 属性返回<iframe>元素的 Window 对象。你可以使用这个 Window 对象来访问 iframe 的文档及其内部 DOM。

```
<!--  下述用端口  10000 表示：domainA 10001 表示：domainB-->
<!-- localhost:10000 -->
<script>
    var iframe = document.createElement('iframe');
    iframe.style.display = 'none'; // 隐藏
    var state = 0; // 防止页面无限刷新
    iframe.onload = function() {
       if(state === 1) {
          console.log(JSON.parse(iframe.contentWindow.name));          // 清除创建的iframe
          iframe.contentWindow.document.write('');
         iframe.contentWindow.close();
         document.body.removeChild(iframe);
       } else if(state === 0) {
           state = 1;
           // 加载完成，指向当前域，防止错误(proxy.html 为空白页面)
    // Blocked a frame with origin "http://localhost:10000" from accessing a cross-origin frame.
iframe.contentWindow.location = 'http://localhost:10000/proxy.html';
       }
     };
  iframe.src = 'http://localhost:10001';
  document.body.appendChild(iframe);
    </script>
    <!-- localhost:10001 -->
    <!DOCTYPE html>
    ...
    <script>
      window.name = JSON.stringify({a: 1, b: 2});
    </script>
```

```
</html>
```

方式五：window.postMessage()

HTML5 新特性，可以用来向其他所有的 window 对象发送消息。需要注意的是我们必须要保证所有的脚本执行完才发送 MessageEvent，如果在函数执行的过程中调用了它，就会让后面的函数超时无法执行。

下述代码实现了跨域存储 localStorage

```
<!--  下述用端口  10000 表示：domainA 10001 表示：domainB-->
<!-- localhost:10000 -->
<iframe
src="http://localhost:10001/msg.html" name="myPostMessage" style="display:none;">
</iframe>
<script>
    function main() {
       LSsetItem('test', 'Test: ' + new Date());
       LSgetItem('test', function(value) {
           console.log('value: ' + value);
      });
       LSremoveItem('test');
}
 var callbacks = {};
 window.addEventListener('message', function(event) {
     if (event.source === frames['myPostMessage']) {
         console.log(event)
          var data = /^#localStorage#(\d+)(null)?#([\S\s]*)/.exec(event.data);
        if (data) {
           if (callbacks[data[1]]) {
                callbacks[data[1]](data[2] === 'null' ? null : data[3]);
          }
            delete callbacks[data[1]];
        }
    }
  }, false);
  var domain = '*';
 // 增加
function LSsetItem(key, value) {
    var obj = {
          setItem: key,
          value: value
   };
```

```
    frames['myPostMessage'].postMessage(JSON.stringify(obj), domain);
      }
     // 获取
    function LSgetItem(key, callback) {
       var identifier = new Date().getTime();
       var obj = {
           identifier: identifier,
             getItem: key
      };
     callbacks[identifier] = callback;
     frames['myPostMessage'].postMessage(JSON.stringify(obj), domain);
     }
    // 删除
     function LSremoveItem(key) {
       var obj = {
            removeItem: key
        };
     frames['myPostMessage'].postMessage(JSON.stringify(obj), domain);
     }
</script>
<!-- localhost:10001 -->
<script>
    window.addEventListener('message', function(event) {
       console.log('Receiver debugging', event);
     if (event.origin == 'http://localhost:10000') {
        var data = JSON.parse(event.data);
        if ('setItem' in data) {
           localStorage.setItem(data.setItem, data.value);
      } else if ('getItem' in data) {
         var gotItem = localStorage.getItem(data.getItem);
         event.source.postMessage(
           '#localStorage#' + data.identifier + (gotItem === null ? 'null#' : '#' + gotItem),
event.origin
    );
       } else if ('removeItem' in data) {
        localStorage.removeItem(data.removeItem);
         }
     }
 }, false);
</script>
```

注意 Safari 一下，会报错：

Blocked a frame with origin “http://localhost:10001” from accessing a frame with origin “http://localhost:10000 “. Protocols, domains, and ports must match.

避免该错误，可以在 Safari 浏览器中勾选开发菜单==>停用跨域限制。或者只能使用服务器端转存的方式实现，因为 Safari 浏览器默认只支持 CORS 跨域请求。

方式六：修改 document.domain 跨子域

前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用 document.domain 进行跨域，所以只能跨子域
在根域范围内，允许把 domain 属性的值设置为它的上一级域。例如，在”aaa.xxx.com”域内，可以把 domain 设置为 “xxx.com” 但不能设置为 “xxx.org” 或者”com”。
现在存在两个域名 aaa.xxx.com 和 bbb.xxx.com。在 aaa 下嵌入 bbb 的页面，由于其 document.name 不一致，无法在 aaa 下操作 bbb 的 js。可以在 aaa 和 bbb 下通过 js 将 document.name = 'xxx.com';设置一致，来达到互相访问的作用。

方式七：WebSocket

WebSocket protocol 是 HTML5 一种新的协议。它实现了浏览器与服务器全双工通信，同时允许跨域通讯，是 server push 技术的一种很棒的实现。相关文章，请查看：WebSocket、WebSocket-SockJS

需要注意：WebSocket 对象不支持 DOM 2 级事件侦听器，必须使用 DOM 0 级语法分别定义各个事件。

方式八：代理

同源策略是针对浏览器端进行的限制，可以通过服务器端来解决该问题
DomainA 客户端（浏览器） ==> DomainA 服务器 ==> DomainB 服务器 ==> DomainA 客户端（浏览器）
来源：blog.csdn.net/ligang2585116/article/details/73072868

87.说一下 JSONP 实现原理？

jsonp 即 json+padding，动态创建 script 标签，利用 script 标签的 src 属性可以获取任何域下的 js 脚本，通过这个特性(也可以说漏洞)，服务器端不在返货 json 格式，而是返回一段调用某个函数的 js 代码，在 src 中进行了调用，这样实现了跨域。

9 设计模式

88. 说一下你熟悉的设计模式？

单例模式：保证被创建一次，节省系统开销。
工厂模式（简单工厂、抽象工厂）：解耦代码。
观察者模式：定义了对象之间的一对多的依赖，这样一来，当一个对象改变时，它的所有的依赖者都会收到通知并自动更新。
外观模式：提供一个统一的接口，用来访问子系统中的一群接口，外观定义了一个高层的接口，让子系统更容易使用。
模版方法模式：定义了一个算法的骨架，而将一些步骤延迟到子类中，模版方法使得子类可以在不改变算法结构的情况下，重新定义算法的步骤。
状态模式：允许对象在内部状态改变时改变它的行为，对象看起来好像修改了它的类。

89. 简单工厂和抽象工厂有什么区别？

简单工厂：用来生产同一等级结构中的任意产品，对于增加新的产品，无能为力。
工厂方法：用来生产同一等级结构中的固定产品，支持增加任意产品。
抽象工厂：用来生产不同产品族的全部产品，对于增加新的产品，无能为力；支持增加产品族

10 Spring/Spring MVC

90. 为什么要使用 spring？

spring 是一个开源框架，是个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架

- 方便结构简化开发
- AOP 编码的支持
- 声明式事物的支持
- 方便程序的测试
- 方便集成各种优势框架
- 降低 Java EE API 的使用难度

91. 解释一下什么是 aop？

AOP 即面向切面编程，是 OOP 编程的有效补充。使用 AOP 技术，可以将一些系统性相关的编程工作，独立提取出来，独立实现，然后通过切面切入进系统。从而避免了在业务逻辑的代码中混入很多的系统相关的逻辑——比如权限管理，事物管理，日志记录等等。

AOP 分为静态 AOP 和动态 AOP：

- 静态 AOP 是指 AspectJ 实现的 AOP，他是将切面代码直接编译到 Java 类文件中。
- 动态 AOP 是指将切面代码进行动态织入实现的 AOP，JDK 动态代理。

91.解释一下什么是 ioc？

即“控制反转”，不是什么技术，而是一种设计思想。在 Java 开发中，Ioc 意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。

IoC 很好的体现了面向对象设计法则之一—— 好莱坞法则：“别找我们，我们找你”；即由 IoC 容器帮对象找相应的依赖对象并注入，而不是由对象主动去找。

92.spring 有哪些主要模块？

core 模块、aop 模块、data access 模块、web 模块、test 模块

spring core：框架的最基础部分，提供 ioc 和依赖注入特性。

spring context：构建于 core 封装包基础上的 context 封装包，提供了一种框架式的对象访问方法。

spring dao：Data Access Object 提供了 JDBC 的抽象层。

spring aop：提供了面向切面的编程实现，让你可以自定义拦截器、切点等。

spring Web：提供了针对 Web 开发的集成特性，例如文件上传，利用 servlet listeners 进行 ioc 容器初始化和针对 Web 的 ApplicationContext。

spring Web mvc：spring 中的 mvc 封装包提供了 Web 应用的 Model-View-Controller（MVC）的实现。

93.spring 常用的注入方式有哪些？

详细解析：https://blog.csdn.net/a909301740/article/details/78379720

构造方法注入

setter 注入

基于注解注入

94. spring 中的 bean 是线程安全的吗？

spring 中的 bean 默认是单例模式，spring 框架并没有对单例 bean 进行多线程的封装处理。

实际上大部分时候 spring bean 无状态的（比如 dao 类），所有某种程度上来说 bean 也是安全的，但如果 bean 有状态的话（比如 view model 对象），那就要开发者自己去保证线程安全了，最简单的就是改变 bean 的作用域，把“singleton”变更为“prototype”，这样请求 bean 相当于 new Bean()了，所以就可以保证线程安全了。

有状态就是有数据存储功能。

无状态就是不会保存数据。

95. spring 支持几种 bean 的作用域？

singleton、prototype、request、session、globalSession 五中作用域。

singleton：spring ioc 容器中只存在一个 bean 实例，bean 以单例模式存在，是系统默认值；

prototype：每次从容器调用 bean 时都会创建一个新的示例，既每次 getBean()相当于执行 new Bean()操作；

Web 环境下的作用域：

request：每次 http 请求都会创建一个 bean；

session：同一个 http session 共享一个 bean 实例；

global-session：用于 portlet 容器，因为每个 portlet 有单独的 session，globalsession 提供一个全局性的 http session

96. spring 自动装配 bean 有哪些方式？

可分为四种：

byName：按照 bean 的属性名称来匹配要装配的 bean

byType：按照 bean 的类型来匹配要装配的 bean

constructor：按照 bean 的构造器入参的类型来进行匹配

autodetect（自动检测）：先使用 constructor 进行装配，如果不成功就使用 byType 来装配

97. spring 事务实现方式有哪些？

- 声明式事务：声明式事务也有两种实现方式，基于 xml 配置文件的方式和注解方式（在类上添加 @Transaction 注解）。
- 编码方式：提供编码的形式管理和维护事务。

98. 说一下 spring 的事务隔离？

- spring 有五大隔离级别，默认值为 ISOLATION_DEFAULT（使用数据库的设置），其他四个隔离级别和数据库的隔离级别一致：
- ISOLATION_DEFAULT：用底层数据库的设置隔离级别，数据库设置的是什么我就用什么；
- ISOLATIONREADUNCOMMITTED：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）；
- ISOLATIONREADCOMMITTED：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读），SQL server 的默认级别；
- ISOLATIONREPEATABLEREAD：可重复读，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读），MySQL 的默认级别；
- ISOLATION_SERIALIZABLE：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。
- 脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。
- 不可重复读 ：是指在一个事务内，多次读同一数据。
- 幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了

99. 说一下 spring mvc 运行流程？

- spring mvc 先将请求发送给 DispatcherServlet。
- DispatcherServlet 查询一个或多个 HandlerMapping，找到处理请求的 Controller。
- DispatcherServlet 再把请求提交到对应的 Controller。
- Controller 进行业务逻辑处理后，会返回一个 ModelAndView。
- Dispathcher 查询一个或多个 ViewResolver 视图解析器，找到 ModelAndView 对象指定的视图对象。
- 视图对象负责渲染返回给客户端

100. spring mvc 有哪些组件？

- 前端控制器 DispatcherServlet
- 映射控制器 HandlerMapping
- 处理器适配器 HandlerMapping
- 处理器 Controller
- 模型和视图 ModelAndView
- 视图解析器 ViewResolver

101. @RequestMapping 的作用是什么？

将 http 请求映射到相应的类/方法上

102. @Autowired 的作用是什么？

@Autowired 它可以对类成员变量、方法及构造函数进行标注，完成自动装配的工作，通过@Autowired 的使用来消除 set/get 方法

11 Spring Boot/Spring Cloud

104.什么是 spring boot？

105.为什么要用 spring boot？

106.spring boot 核心配置文件是什么？

107.spring boot 配置文件有哪几种类型？它们有什么区别？

108.spring boot 有哪些方式可以实现热部署？

109.jpa 和 hibernate 有什么区别？

110.什么是 spring cloud？

111.spring cloud 断路器的作用是什么？

112.spring cloud 的核心组件有哪些

12 Hibernate

113.为什么要使用 hibernate？

114.什么是 ORM 框架？

115.hibernate 中如何在控制台查看打印的 sql 语句？

116.hibernate 有几种查询方式？

117.hibernate 实体类可以被定义为 final 吗？

118.在 hibernate 中使用 Integer 和 int 做映射有什么区别？

119.hibernate 是如何工作的？

120.get()和 load()的区别？

121.说一下 hibernate 的缓存机制？

122.hibernate 对象有哪些状态？

123.在 hibernate 中 getCurrentSession 和 openSession 的区别是什么？

124.hibernate 实体类必须要有无参构造函数吗？为什么？

13 Mybatis

125.mybatis 中 #{}和 ${}的区别是什么？

126.mybatis 有几种分页方式？

127.RowBounds 是一次性查询全部结果吗？为什么？

128.mybatis 逻辑分页和物理分页的区别是什么？

129.mybatis 是否支持延迟加载？延迟加载的原理是什么？

130.说一下 mybatis 的一级缓存和二级缓存？

131.mybatis 和 hibernate 的区别有哪些？

132.mybatis 有哪些执行器（Executor）？

133.mybatis 分页插件的实现原理是什么？

134.mybatis 如何编写一个自定义插件？

14 RabbitMQ

135. rabbitmq 的使用场景有哪些？

①. 跨系统的异步通信，所有需要异步交互的地方都可以使用消息队列。就像我们除了打电话（同步）以外，还需要发短信，发电子邮件（异步）的通讯方式。

②. 多个应用之间的耦合，由于消息是平台无关和语言无关的，而且语义上也不再是函数调用，因此更适合作为多个应用之间的松耦合的接口。基于消息队列的耦合，不需要发送方和接收方同时在线。在企业应用集成（EAI）中，文件传输，共享数据库，消息队列，远程过程调用都可以作为集成的方法。

③. 应用内的同步变异步，比如订单处理，就可以由前端应用将订单信息放到队列，后端应用从队列里依次获得消息处理，高峰时的大量订单可以积压在队列里慢慢处理掉。由于同步通常意味着阻塞，而大量线程的阻塞会降低计算机的性能。

④. 消息驱动的架构（EDA），系统分解为消息队列，和消息制造者和消息消费者，一个处理流程可以根据需要拆成多个阶段（Stage），阶段之间用队列连接起来，前一个阶段处理的结果放入队列，后一个阶段从队列中获取消息继续处理。

⑤. 应用需要更灵活的耦合方式，如发布订阅，比如可以指定路由规则。

⑥. 跨局域网，甚至跨城市的通讯（CDN 行业），比如北京机房与广州机房的应用程序的通信。

136. rabbitmq 有哪些重要的角色？

RabbitMQ 中重要的角色有：生产者、消费者和代理：

生产者：消息的创建者，负责创建和推送数据到消息服务器；

消费者：消息的接收方，用于处理数据和确认消息；

代理：就是 RabbitMQ 本身，用于扮演“快递”的角色，本身不生产消息，只是扮演“快递”的角色。

137. rabbitmq 有哪些重要的组件？

ConnectionFactory（连接管理器）：应用程序与 Rabbit 之间建立连接的管理器，程序代码中使用。

Channel（信道）：消息推送使用的通道。

Exchange（交换器）：用于接受、分配消息。

Queue（队列）：用于存储生产者的消息。

RoutingKey（路由键）：用于把生成者的数据分配到交换器上。

BindingKey（绑定键）：用于把交换器的消息绑定到队列上。

138. rabbitmq 中 vhost 的作用是什么？

vhost 可以理解为虚拟 broker ，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

139. rabbitmq 的消息是怎么发送的？

首先客户端必须连接到 RabbitMQ 服务器才能发布和消费消息，客户端和 rabbit server 之间会创建一个 tcp 连接，一旦 tcp 打开并通过了认证（认证就是你发送给 rabbit 服务器的用户名和密码），你的客户端和 RabbitMQ 就创建了一条 amqp 信道（channel），信道是创建在“真实” tcp 上的虚拟连接，amqp 命令都是通过信道发送出去的，每个信道都会有一个唯一的 id，不论是发布消息，订阅队列都是通过这个信道完成的。

140. rabbitmq 怎么保证消息的稳定性？

提供了事务的功能

通过将 channel 设置为 confirm（确认）模式

141. rabbitmq 怎么避免消息丢失？

消息持久化

ACK 确认机制

设置集群镜像模式

消息补偿机制

142. 要保证消息持久化成功的条件有哪些？

声明队列必须设置持久化 durable 设置为 true.

消息推送投递模式必须设置持久化，deliveryMode 设置为 2（持久）。

消息已经到达持久化交换器。

消息已经到达持久化队列。

以上四个条件都满足才能保证消息持久化成功

143. rabbitmq 持久化有什么缺点？

持久化的缺地就是降低了服务器的吞吐量，因为使用的是磁盘而非内存存储，从而降低了吞吐量。可尽量使用 ssd 硬盘来缓解吞吐量的问题。

144. rabbitmq 有几种广播类型？

1, fanout: 所有 bind 到此 exchange 的 queue 都可以接收消息（纯广播，绑定到 RabbitMQ 的接受者都能收到消息）；

2,direct: 通过 routingKey 和 exchange 决定的那个唯一的 queue 可以接收消息；

3,topic:所有符合 routingKey(此时可以是一个表达式)的 routingKey 所

bind 的 queue 可以接收消息；

145. rabbitmq 怎么实现延迟消息队列？

通过消息过期后进入死信交换器，再由交换器转发到延迟消费队列，实现延迟功能；

使用 RabbitMQ-delayed-message-exchange 插件实现延迟功能。

146. rabbitmq 集群有什么用？

集群主要有以下两个用途：

高可用：某个服务器出现问题，整个 RabbitMQ 还可以继续使用；

高容量：集群可以承载更多的消息量。

147. rabbitmq 节点的类型有哪些？

磁盘节点：消息会存储到磁盘。

内存节点：消息都存储在内存中，重启服务器消息丢失，性能高于磁盘类型。

148. rabbitmq 集群搭建需要注意哪些问题？

各节点之间使用“--link”连接，此属性不能忽略。

各节点使用的 erlang cookie 值必须相同，此值相当于“秘钥”的功能，用于各节点的认证。

整个集群中必须包含一个磁盘节点。

149. rabbitmq 每个节点是其他节点的完整拷贝吗？为什么？

不是，原因有以下两个：

存储空间的考虑：如果每个节点都拥有所有队列的完全拷贝，这样新增节点不但没有新增存储空间，反而增加了更多的冗余数据；

性能的考虑：如果每条消息都需要完整拷贝到每一个集群节点，那新增节点并没有提升处理消息的能力，最多是保持和单节点相同的性能甚至是更糟。

150. rabbitmq 集群中唯一一个磁盘节点崩溃了会发生什么情况？

如果唯一磁盘的磁盘节点崩溃了，不能进行以下操作：

不能创建队列

不能创建交换器

不能创建绑定

不能添加用户

不能更改权限

不能添加和删除集群节点

唯一磁盘节点崩溃了，集群是可以保持运行的，但你不能更改任何东西。

151. rabbitmq 对集群节点停止顺序有要求吗？

RabbitMQ 对集群的停止的顺序是有要求的，应该先关闭内存节点，最后再关闭磁盘节点。如果顺序恰好相反的话，可能会造成消息的丢失。

15 Kafka

152. kafka 可以脱离 zookeeper 单独使用吗？为什么？

kafka 不能脱离 zookeeper 单独使用，因为 kafka 使用 zookeeper 管理和协调 kafka 的节点服务器。

153. kafka 有几种数据保留的策略？

kafka 有两种数据保存策略：按照过期时间保留和按照存储的消息大小保留。

154. kafka 同时设置了 7 天和 10G 清除数据，到第五天的时候消息达到了 10G，这个时候 kafka 将如何处理？

这个时候 kafka 会执行数据清除工作，时间和大小不论那个满足条件，都会清空数据。

155. 什么情况会导致 kafka 运行变慢？

cpu 性能瓶颈
磁盘读写瓶颈
网络瓶颈

156. 使用 kafka 集群需要注意什么？

集群的数量不是越多越好，最好不要超过 7 个，因为节点越多，消息复制需要的时间就越长，整个群组的吞吐量就越低。
集群数量最好是单数，因为超过一半故障集群就不能用了，设置为单数容错率更高。

16 Zookper

157. zookeeper 是什么？

zookeeper 是一个分布式的，开放源码的分布式应用程序协调服务，是 google chubby 的开源实现，是 hadoop 和 hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

158. zookeeper 都有哪些功能？

集群管理：监控节点存活状态、运行请求等。
主节点选举：主节点挂掉了之后可以从备用的节点开始新一轮选主，主节点选举说的就是这个选举的过程，使用 zookeeper 可以协助完成这个过程。
分布式锁：zookeeper 提供两种锁：独占锁、共享锁。独占锁即一次只能有一个线程使用资源，共享锁是读锁共享，读写互斥，即可以有多线线程同时读同一个资源，如果要使用写锁也只能有一个线程使用。

zookeeper 可以对分布式锁进行控制。

命名服务：在分布式系统中，通过使用命名服务，客户端应用能够根据指定名字来获取资源或服务的地址，提供者等信息。

159. zookeeper 有几种部署模式？

zookeeper 有三种部署模式：

单机部署：一台集群上运行；

集群部署：多台集群运行；

伪集群部署：一台集群启动多个 zookeeper 实例运行。

160. zookeeper 怎么保证主从节点的状态同步？

zookeeper 的核心是原子广播，这个机制保证了各个 server 之间的同步。实现这个机制的协议叫做 zab 协议。 zab 协议有两种模式，分别是恢复模式（选主）和广播模式（同步）。当服务启动或者在领导者崩溃后，zab 就进入了恢复模式，当领导者被选举出来，且大多数 server 完成了和 leader 的状态同步以后，恢复模式就结束了。状态同步保证了 leader 和 server 具有相同的系统状态。

161. 集群中为什么要有主节点？

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，

所以就需要主节点。

162. 集群中有 3 台服务器,其中一个节点宕机,这个时候 zookeeper 还可以使用吗?

可以继续使用，单数服务器只要没超过一半的服务器宕机就可以继续使用。

163. 说一下 zookeeper 的通知机制?

客户端端会对某个 znode 建立一个 watcher 事件，当该 znode 发生变化时，这些客户端会收到 zookeeper 的通知，然后客户端可以根据 znode 变化来做出业务上的改变。

17 MySQL

164. 数据库的三范式是什么?

- 第一范式：强调的是列的原子性，即数据库表的每一列都是不可分割的原子数据项。
- 第二范式：要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性。
- 第三范式：任何非主属性不依赖于其它非主属性。

165. 一张自增表里面总共有 7 条数据，删除了最后 2 条数据，重启 mysql 数据库，又插入了一条数据，此时 id 是几?

- 表类型如果是 MyISAM ，那 id 就是 8。
- 表类型如果是 InnoDB，那 id 就是 6。

166. 如何获取当前数据库版本?

使用 select version() 获取当前 MySQL 数据库版本。

167. 说一下 ACID 是什么？

- Atomicity（原子性）：一个事务（transaction）中的所有操作，或者全部完成，或者全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。即，事务不可分割、不可约简。
- Consistency（一致性）：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设约束、触发器、级联回滚等。
- Isolation（隔离性）：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。
- Durability（持久性）：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

168. char 和 varchar 的区别是什么？

- char(n) ：固定长度类型，比如订阅 char(10)，当你输入"abc"三个字符的时候，它们占的空间还是 10 个字节，其他 7 个是空字节。

char 优点：效率高；缺点：占用空间；适用场景：存储密码的 md5 值，固定长度的，使用 char 非常合适。

- varchar(n) ：可变长度，存储的值是每个值占用的字节再加上一个用来记录其长度的字节的长度。

所以，从空间上考虑 varcahr 比较合适；从效率上考虑 char 比较合适，二者使用需要权衡

169. float 和 double 的区别是什么？

- float 最多可以存储 8 位的十进制数，并在内存中占 4 字节。
- double 最可可以存储 16 位的十进制数，并在内存中占 8 字节。

170. mysql 的内连接、左连接、右连接有什么区别？

内连接关键字：inner join；左连接：left join；右连接：right join。 内连接是把匹配的关联数据显示出来；左连接是左边的表全部显示出来，右边的表显示出符合条件的数据；右连接正好相反

171. mysql 索引是怎么实现的？

索引是满足某种特定查找算法的数据结构，而这些数据结构会以某种方式指向数据，从而实现高效查找数据。 具体来说 MySQL 中的索引，不同的数据引擎实现有所不同，但目前主流的数据库引擎的索引都是 B+ 树实现的，B+ 树的搜索效率，可以到达二分法的性能，找到数据区域之后就找到了完整的数据结构了，所有索引的性能也是更好的

172. 怎么验证 mysql 的索引是否满足需求？

使用 explain 查看 SQL 是如何执行查询语句的，从而分析你的索引是否满足需求。
explain select * from table where type=1

173. 说一下数据库的事务隔离？

- READ-UNCOMMITTED：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）。
- READ-COMMITTED：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读）。
- REPEATABLE-READ：可重复读，默认级别，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读）。
- SERIALIZABLE：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。

脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。
不可重复读 ：是指在一个事务内，多次读同一数据。
幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了

174. 说一下 mysql 常用的引擎？

InnoDB 引擎：InnoDB 引擎提供了对数据库 acid 事务的支持，并且还提供了行级锁和外键的约束，它的设计的目标就是处理大数据容量的数据库系统。MySQL 运行的时候，InnoDB 会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎是不支持全文搜索，同时启动也比较的慢，它是不会保存表的行数的，所以当进行 select count(*) from table 指令的时候，需要进行扫描全表。由于锁的粒度小，写操作是不会锁定全表的,所以在并发度较高的场景下使用会提升效率
MyIASM 引擎：MySQL 的默认引擎，但不提供事务的支持，也不支持行级锁和外键。因此

当执行插入和更新语句时，即执行写操作的时候需要锁定这个表，所以会导致效率会降低。不过和 InnoDB 不同的是，MyIASM 引擎是保存了表的行数，于是当进行 select count(*) from table 语句时，可以直接的读取已经保存的值而不需要进行扫描全表。所以，如果表的读操作远远多于写操作时，并且不需要事务的支持的，可以将 MyIASM 作为数据库引擎的首选

175. 说一下 mysql 的行锁和表锁？

MyISAM 只支持表锁，InnoDB 支持表锁和行锁，默认为行锁
表级锁：开销小，加锁快，不会出现死锁。锁定粒度大，发生锁冲突的概率最高，并发量最低
行级锁：开销大，加锁慢，会出现死锁。锁力度小，发生锁冲突的概率小，并发度最高

176. 说一下乐观锁和悲观锁？

- 乐观锁：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
- 悲观锁：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阻止，直到这个锁被释放

数据库的乐观锁需要自己实现，在表里面添加一个 version 字段，每次修改成功值加 1，这样每次修改的时候先对比一下，自己拥有的 version 和数据库现在的 version 是否一致，如果不一致就不修改，这样就实现了乐观锁

177. mysql 问题排查都有哪些手段？

- 使用 show processlist 命令查看当前所有连接信息
- 使用 explain 命令查询 SQL 语句执行计划
- 开启慢查询日志，查看慢查询的 SQL

178. 如何做 mysql 的性能优化？

- 为搜索字段创建索引
- 避免使用 select *，列出需要查询的字段
- 垂直分割分表
- 选择正确的存储引擎

18 Redis

179.redis 是什么？都有哪些使用场景？

180.redis 有哪些功能？

181.redis 和 memecache 有什么区别？

182.redis 为什么是单线程的？

183.什么是缓存穿透？怎么解决？

184.redis 支持的数据类型有哪些？

185.redis 支持的 java 客户端都有哪些？

186.jedis 和 redisson 有哪些区别？

187.怎么保证缓存和数据库数据的一致性？

188.redis 持久化有几种方式？

189.redis 怎么实现分布式锁？

190.redis 分布式锁有什么缺陷？

191.redis 如何做内存优化？

192.redis 淘汰策略有哪些？

193.redis 常见的性能问题有哪些？该如何解决？

19 JVM

194. 说一下 jvm 的主要组成部分？及其作用？

- 类加载器（ClassLoader）
- 运行时数据区（Runtime Data Area）
- 执行引擎（Execution Engine）
- 本地库接口（Native Interface）

组件的作用： 首先通过类加载器（ClassLoader）会把 Java 代码转换成字节码，运行时数据区（Runtime Data Area）再把字节码加载到内存中，而字节码文件只是 JVM 的一套指令集规范，并不能直接交个底层操作系统去执行，因此需要特定的命令解析器执行引擎（Execution Engine），将字节码翻译成底层系统指令，再交由 CPU 去执行，而这个过程中需要调用其他语言的本地库接口（Native Interface）来实现整个程序的功能

195. 说一下 jvm 运行时数据区？

不同虚拟机的运行时数据区可能略微有所不同，但都会遵从 Java 虚拟机规范， Java 虚拟机规范规定的区域分为以下 5 个部分：

程序计数器（Program Counter Register）：当前线程所执行的字节码的行号指示器，字节码解析器的工作是通过改变这个计数器的值，来选取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能，都需要依赖这个计数器来完成

Java 虚拟机栈（Java Virtual Machine Stacks）：用于存储局部变量表、操作数栈、动态链接、方法出口等信息

本地方法栈（Native Method Stack）：与虚拟机栈的作用是一样的，只不过虚拟机栈是服务 Java 方法的，而本地方法栈是为虚拟机调用 Native 方法服务的

Java 堆（Java Heap）：Java 虚拟机中内存最大的一块，是被所有线程共享的，几乎所有的对象实例都在这里分配内存

方法区（Methed Area）：用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据

196. 说一下堆栈的区别？

- 功能方面：堆是用来存放对象的，栈是用来执行程序的
- 共享性：堆是线程共享的，栈是线程私有的
- 空间大小：堆大小远远大于栈

197. 队列和栈是什么？有什么区别？

- 队列和栈都是被用来预存储数据的。
- 队列允许先进先出检索元素，但也有例外的情况，Deque 接口允许从两端检索元素。
- 栈和队列很相似，但它运行对元素进行后进先出进行检索

198. 什么是双亲委派模型？

类加载器分类：

启动类加载器（Bootstrap ClassLoader），是虚拟机自身的一部分，用来加载 Java_HOME/lib/目录中的，或者被 -Xbootclasspath 参数所指定的路径中并且被虚拟机识别的类库

其他类加载器：

扩展类加载器（Extension ClassLoader）：负责加载\lib\ext 目录或 Java. ext. dirs 系统变量指定的路径中的所有类库

应用程序类加载器（Application ClassLoader）。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

双亲委派模型：如果一个类加载器收到了类加载的请求，它首先不会自己去加载这个类，而是把这个请求委派给父类加载器去完成，每一层的类加载器都是如此，这样所有的加载请求都会被传送到顶层的启动类加载器中，只有当父加载无法完成加载请求（它的搜索范围中没找到所需的类）时，子加载器才会尝试去加载类

199. 说一下类加载的执行过程？

类装载分为以下 5 个步骤：

加载：根据查找路径找到相应的 class 文件然后导入

检查：检查加载的 class 文件的正确性

准备：给类中的静态变量分配内存空间

解析：虚拟机将常量池中的符号引用替换成直接引用的过程。符号引用就理解为一个标示，而在直接引用直接指向内存中的地址

初始化：对静态变量和静态代码块执行初始化工作

200. 怎么判断对象是否可以被回收？

一般有两种方法来判断：

- 引用计数器：为每个对象创建一个引用计数，有对象引用时计数器 +1，引用被释放时计数 -1，当计数器为 0 时就可以被回收。它有一个缺点不能解决循环引用的问题
- 根搜索算法：从 GC Roots 开始向下搜索，搜索所走过的路径称为引用链。当一个对象到 GC Roots 没有任何引用链相连时，则证明此对象是可以被回收的

201. java 中都有哪些引用类型？

- 强引用：发生 gc 的时候不会被回收
- 软引用：有用但不是必须的对象，在发生内存溢出之前会被回收
- 弱引用：有用但不是必须的对象，在下一次 GC 时会被回收
- 虚引用（幽灵引用/幻影引用）：无法通过虚引用获得对象，用 PhantomReference 实现虚引用，虚引用的用途是在 gc 时返回一个通知

202. 说一下 jvm 有哪些垃圾回收算法？

- 标记-清除算法：标记无用对象，然后进行清除回收。缺点：效率不高，无法清除垃圾碎片
- 标记-整理算法：标记无用对象，让所有存活的对象都向一端移动，然后直接清除掉端边界以外的内存
- 复制算法：按照容量划分二个大小相等的内存区域，当一块用完的时候将活着的对象复制到另一块上，然后再把已使用的内存空间一次清理掉。缺点：内存使用率不高，只有原来的一半
- 分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法

203. 说一下 jvm 有哪些垃圾回收器？

- Serial：最早的单线程串行垃圾回收器
- Serial Old：Serial 垃圾回收器的老年版本，同样也是单线程的，可以作为 CMS 垃圾回收器的备选预案
- ParNew：是 Serial 的多线程版本
- Parallel 和 ParNew 收集器类似是多线程的，但 Parallel 是吞吐量优先的收集器，可以牺牲等待时间换取系统的吞吐量
- Parallel Old 是 Parallel 老生代版本，Parallel 使用的是复制的内存回收算法，Parallel Old 使用的是标记-整理的内存回收算法
- CMS：一种以获得最短停顿时间为目标的收集器，非常适用 B/S 系统
- G1：一种兼顾吞吐量和停顿时间的 GC 实现，是 JDK 9 以后的默认 GC 选项

204. 详细介绍一下 CMS 垃圾回收器？

CMS 是英文 Concurrent Mark-Sweep 的简称，是以牺牲吞吐量为代价来获得最短回收停顿时间的垃圾回收器。对于要求服务器响应速度的应用上，这种垃圾回收器非常适合。在启动 JVM 的参数加上“-XX:+UseConcMarkSweepGC”来指定使用 CMS 垃圾回收器
CMS 使用的是标记-清除的算法实现的，所以在 gc 的时候回产生大量的内存碎片，当剩余内存不能满足程序运行要求时，系统将会出现 Concurrent Mode Failure，临时 CMS 会采用 Serial Old 回收器进行垃圾清除，此时的性能将会被降低

205. 新生代垃圾回收器和老生代垃圾回收器都有哪些？有什么区别？

- 新生代回收器：Serial、ParNew、Parallel Scavenge
- 老年代回收器：Serial Old、Parallel Old、CMS
- 整堆回收器：G1

新生代垃圾回收器一般采用的是复制算法，复制算法的优点是效率高，缺点是内存利用率低；
老年代回收器一般采用的是标记-整理的算法进行垃圾回收

206. 简述分代垃圾回收器是怎么工作的？

分代回收器有两个分区：老生代和新生代，新生代默认的空间占比总空间的 1/3，老生代的默认占比是 2/3
新生代使用的是复制算法，新生代里有 3 个分区：Eden、To Survivor、From Survivor，它们的默认占比是 8:1:1，它的执行流程如下：
把 Eden + From Survivor 存活的对象放入 To Survivor 区
清空 Eden 和 From Survivor 分区
From Survivor 和 To Survivor 分区交换，From Survivor 变 To Survivor，To Survivor 变 From Survivor

每次在 From Survivor 到 To Survivor 移动时都存活的对象，年龄就 +1，当年龄到达 15（默认配置是 15）时，升级为老生代。大对象也会直接进入老生代。 老生代当空间占用到达某个值之后就会触发全局垃圾收回，一般使用标记整理的执行算法。以上这些循环往复就构成了整个分代垃圾回收的整体执行流程

207. 说一下 jvm 调优的工具？

JDK 自带了很多监控工具，都位于 JDK 的 bin 目录下，其中最常用的是 jconsole 和 jvisualvm 这两款视图监控工具

jconsole：用于对 JVM 中的内存、线程和类等进行监控；

jvisualvm：JDK 自带的全能分析工具，可以分析：内存快照、线程快照、程序死锁、监控内存的变化、gc 变化等

208. 常用的 jvm 调优的参数都有哪些？

-Xms2g：初始化推大小为 2g
-Xmx2g：堆最大内存为 2g
-XX:NewRatio=4：设置年轻的和老年代的内存比例为 1:4
-XX:SurvivorRatio=8：设置新生代 Eden 和 Survivor 比例为 8:2
–XX:+UseParNewGC：指定使用 ParNew + Serial Old 垃圾回收器组合
-XX:+UseParallelOldGC：指定使用 ParNew + ParNew Old 垃圾回收器组合
-XX:+UseConcMarkSweepGC：指定使用 CMS + Serial Old 垃圾回收器组合
-XX:+PrintGC：开启打印 gc 信息
-XX:+PrintGCDetails：打印 gc 详细信息

20 数据结构

209. 括号匹配

210. 计算逆波兰式

211. 实现一个 min 方法的栈

212. 使用栈，完成中缀表达式转后缀表达式

213. 约瑟夫环

214. 斐波那契数列

215. 用队列实现栈

216. 打印杨辉三角

217. 用两个栈实现一个队列

218. 翻转链表

219. 从尾到头打印链表

220. 合并两个有序链表

221. 查找单链表中的倒数第 K 个节点

222. 查找单链表的中间节点

223. 实现双向链表

224. 两个集合取交集

225. 查找不重复的数

226. 求一棵树的镜像

227. 使用非递归方式实现中序和后序遍历

228. 寻找两个节点的最近公共祖先

229. 分层打印二叉树

230. 输出指定层的节点个数