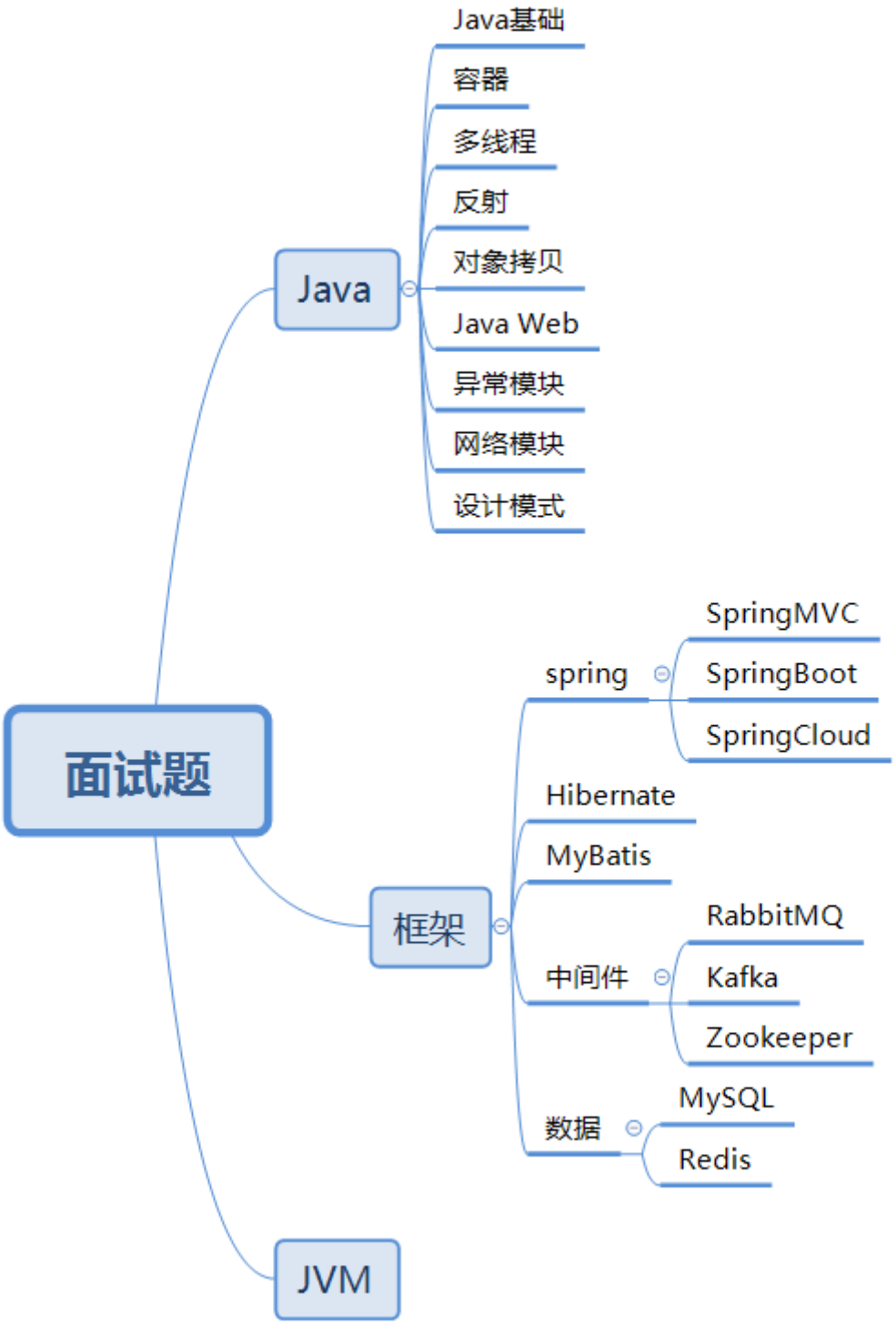
Java面试题

共分为十二个模块，分别是：Java基础、容器、多线程、反射、对象拷贝、Java Web、异常、网络、设计模式、Spring/SPringMVC、SpringBoot/SpringCloud、Hibernate、Mybatis、RabbitMQ、kafka、Zookeeper、MySQL、Redis、JVM



一. Java 基础模块

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("Java");
Cat c2 = new Cat("Java");
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("精彩猿笔记");
String s2 = new String("精彩猿笔记");
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，只是很多类重写了 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 = "精彩";
String str2 = "笔记";
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“精彩”和“笔记”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。round()是四舍五入，注意负数5是舍的，例如：Math.round(1.5)值是2，Math.round(-1.5)值是-1。

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**。 三者区别：

   - StringBuffer和StringBuilder都继承自抽象类AbstractStringBuilder。
   - 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 存储数据的字符数组没有被final修饰，说明值可以改变，抽象类AbstractStringBuilder内部都提供了一个自动扩容机制，当发现长度不够的时候(初始默认长度是16)，会自动进行扩容工作，扩展为原数组长度的2倍加2，创建一个新的数组，并将数组的数据复制到新数组，所以对于拼接字符串效率要比String要高。
   - 线程安全性：StringBuffer由于很多方法都被 synchronized 修饰了所以线程安全，但是当多线程访问时，加锁和释放锁的过程很平凡，所以效率相比StringBuilder要低。StringBuilder相反执行效率高，但是线程不安全。所以单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

   执行速度**:StringBuilder > StringBuffer > String**。

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，如果常量池中有"i”，就返回"i"的地址，如果没有就创建"i"，然后返回"i"的地址；而 String str=new String(“i”) 则会被分到堆内存中新开辟一块空间。

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；但是包含一个抽象方法的类一定是抽象类。 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。 抽象类是不能被实例化的，就是不能用new调出构造方法创建对象，普通类可以直接实例化。 如果一个类继承于抽象类，则该子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为abstract类。

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。

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。
- 实现数量：类可以实现很多个接口；但只能继承一个抽象类【java只支持单继承】。
- 访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的抽象方法可以使用Public和Protected修饰，如果抽象方法修饰符为Private，则报错：The abstract method 方法名 in type Test can only set a visibility modifier, one of public or protected。

15. **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。

16. **BIO、NIO、AIO 有什么区别？**

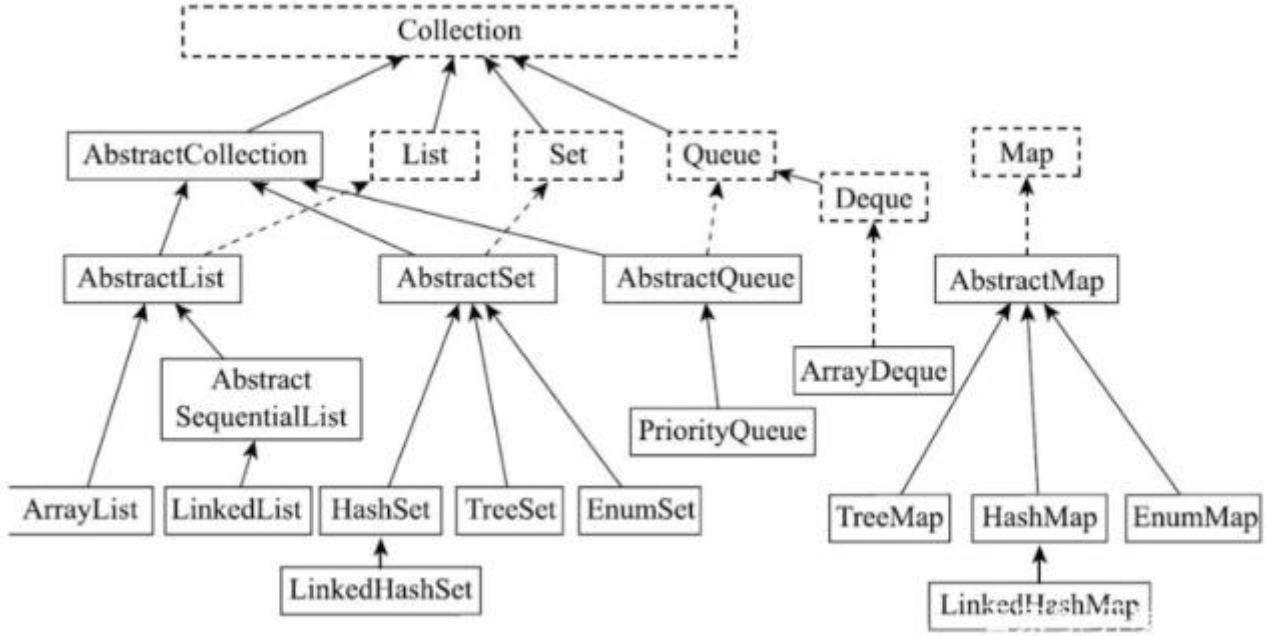
- BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。
- NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。
- AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。

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。 …

二. Java 容器模块

18. **Java 容器都有哪些？** Java 容器分为 Collection 和 Map 两大类，其下又有很多子类，如下所示：



Collection ---List(ArrayList,LinkedList) Vector Stack Set(HashSet,LinkedHashSet,TreeSet) Map(HashMap<---LinkedHashMap,TreeMap,ConcurrentHashMap) Hashtable

19. **Collection 和 Collections 有什么区别？** Collection 是一个集合接口，它提供了对集合对象进行基本操作的通用接口方法，所有集合都是它的子类，比如 List、Set 等。 Collections 是一个包装类，包含了很多静态方法，不能被实例化，就像一个工具类，比如提供的排序方法：Collections. sort(list)。

20. **List、Set、Map 之间的区别是什么？** List、Set、Map 的区别主要体现在两个方面：元素是否有序、是否允许元素重复。 三者之间的区别，如下表：

21. **HashMap 和 Hashtable 有什么区别？**

- HashMap是继承自AbstractMap类，而HashTable是继承自Dictionary类。不过它们都实现了同时实现了map、Cloneable（可复制）、Serializable（可序列化）这三个接口。
- Hashtable比HashMap多提供了elments() 和contains() 两个方法。
- HashMap的key-value支持key-value，null-null，key-null，null-value四种。而Hashtable只支持key-value一种（即key和value都不为null这种形式）。既然HashMap支持带有null的形式，那么在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，因为使用get的时候，当返回null时，你无法判断到底是不存在这个key，还是这个key就是null，还是key存在但value是null。
- 线程安全性不同：HashMap的方法都没有使用synchronized关键字修饰，都是非线程安全的，而Hashtable的方法几乎都是被synchronized关键字修饰的。但是，当我们需要HashMap是线程安全的时，怎么办呢？我们可以通过Collections.synchronizedMap(hashMap)来进行处理，亦或者我们使用线程安全

的ConcurrentHashMap。ConcurrentHashMap虽然也是线程安全的，但是它的效率比Hashtable要高好多倍。因为ConcurrentHashMap使用了分段锁，并不对整个数据进行锁定。

- 初始容量大小和每次扩充容量大小的不同：Hashtable默认的初始大小为11，之后每次扩充，容量变为原来的2n+1。HashMap默认的初始化大小为16。之后每次扩充，容量变为原来的2倍。
- 计算hash值的方法不同：为了得到元素的位置，首先需要根据元素的 KEY计算出一个hash值，然后再用这个hash值来计算得到最终的位置。Hashtable直接使用对象的hashCode。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。然后再使用除留余数发来获得最终的位置。

22. **如何决定使用 HashMap 还是 TreeMap？** 对于在 Map 中插入、删除、定位一个元素这类操作，HashMap 是最好的选择，因为相对而言 HashMap 的插入会更快，但如果你要对一个 key 集合进行有序的遍历，那 TreeMap 是更好的选择。

23. **说一下 HashMap 的实现原理？** HashMap 基于 Hash 算法实现的，我们通过 put(key,value)存储，get(key)来获取。当传入 key 时，HashMap 会根据 key. hashCode() 计算出 hash 值，根据 hash 值将 value 保存在 bucket 里。当计算出的 hash 值相同时，我们称之为 hash 冲突，HashMap 的做法是用链表和红黑树存储相同 hash 值的 value。当 hash 冲突的个数比较少时，使用链表否则使用红黑树。

24. **说一下 HashSet 的实现原理？** HashSet 是基于 HashMap 实现的，HashSet 底层使用 HashMap 来保存所有元素，因此 HashSet 的实现比较简单，相关 HashSet 的操作，基本上都是直接调用底层 HashMap 的相关方法来完成，HashSet 不允许重复的值。

25. **ArrayList 和 LinkedList 的区别是什么？** 数据结构实现：ArrayList 是动态数组的数据结构实现，而 LinkedList 是双向链表的数据结构实现。 随机访问效率：ArrayList 比 LinkedList 在随机访问的时候效率要高，因为 LinkedList 是线性的数据存储方式，所以需要移动指针从前往后依次查找。 增加和删除效率：在非首尾的增加和删除操作，LinkedList 要比 ArrayList 效率要高，因为 ArrayList 增删操作要影响数组内的其他数据的下标。 综合来说，在需要频繁读取集合中的元素时，更推荐使用 ArrayList，而在插入和删除操作较多时，更推荐使用 LinkedList。

26. **如何实现数组和 List 之间的转换？** 数组转 List：使用 Arrays. asList(array) 进行转换。 List 转数组：使用 List 自带的 toArray() 方法。 代码示例：

```
// list to array
List<String> list = new ArrayList<String>();
list. add("☞精◈彩◈猿◈笔◈记☜");
list. add("的博客");
list. toArray();
// array to list
String[] array = new String[]{"☞精◈彩◈猿◈笔◈记☜","的博客"};
Arrays. asList(array);
```

27. **ArrayList 和 Vector 的区别是什么？** 线程安全：Vector 使用了 Synchronized 来实现线程同步，是线程安全的，而 ArrayList 是非线程安全的。 性能：ArrayList 在性能方面要优于 Vector。 扩容：ArrayList 和 Vector 都会根据实际的需要动态的调整容量，只不过在 Vector 扩容每次会增加 1 倍，而 ArrayList 只会增加 50%。

28. **Array 和 ArrayList 有何区别？** Array 可以存储基本数据类型和对象，ArrayList 只能存储对象。 Array 是指定固定大小的，而 ArrayList 大小是自动扩展的。 Array 内置方法没有 ArrayList 多，比如 addAll、

removeAll、iteration 等方法只有 ArrayList 有。

29. 在 **Queue** 中 **poll()**和 **remove()**有什么区别？ 相同点：都是返回第一个元素，并在队列中删除返回的对象。 不同点：如果没有元素 remove()会直接抛出NoSuchElementException 异常，而 poll()会返回 null。 代码示例：

```
Queue<String> queue = new LinkedList<String>();
queue. offer("string"); // add
System. out. println(queue. poll());
System. out. println(queue. remove());
System. out. println(queue. size());
```

30. 哪些集合类是线程安全的？ Vector、Hashtable、Stack 都是线程安全的，而像 HashMap 则是非线程安全的，不过在 JDK 1.5 之后随着 Java. util. concurrent 并发包的出现，它们也有了自己对应的线程安全类，比如 HashMap 对应的线程安全类就是 ConcurrentHashMap。

31. 迭代器 **Iterator** 是什么？ Iterator 接口提供遍历任何 Collection 的接口。我们可以从一个 Collection 中使用迭代器方法来获取迭代器实例。迭代器取代了 Java 集合框架中的 Enumeration，迭代器允许调用者在迭代过程中移除元素。

32. **Iterator** 怎么使用？有什么特点？ Iterator 使用代码如下：

```
List<String> list = new ArrayList<>();
Iterator<String> it = list. iterator();
while(it. hasNext()){
String obj = it. next();
System. out. println(obj);
}
```

Iterator 的特点是更加安全，因为它可以确保，在当前遍历的集合元素被更改的时候，就会抛出 ConcurrentModificationException 异常。

33. **Iterator** 和 **ListIterator** 有什么区别？ Iterator 可以遍历 Set 和 List 集合，而 ListIterator 只能遍历 List。 Iterator 只能单向遍历，而 ListIterator 可以双向遍历（向前/后遍历）。 ListIterator 从 Iterator 接口继承，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

34. 怎么确保一个集合不能被修改？ 可以使用 Collections. unmodifiableCollection(Collection c) 方法来创建一个只读集合，这样改变集合的任何操作都会抛出 Java. lang. UnsupportedOperationException 异常。 示例代码如下：

```
List<String> list = new ArrayList<>();
list. add("x");
Collection<String> clist = Collections. unmodifiableCollection(list);
clist. add("y"); // 运行时此行报错
System. out. println(list. size());
```

三. Java 多线程模块

35. **并行和并发有什么区别？** 并行：多个处理器或多核处理器同时处理多个任务。 并发：多个任务在同一个 CPU 核上，按细分的时间片轮流(交替)执行，从逻辑上来看那些任务是同时执行。 如下图：【并发 = 两个队列和一台咖啡机】【并行 = 两个队列和两台咖啡机】

36. **线程和进程的区别？** 一个程序下至少有一个进程，一个进程下至少有一个线程，一个进程下也可以有多个线程来增加程序的执行速度。

37. **守护线程是什么？** 守护线程是运行在后台的一种特殊进程。它独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件。在 Java 中垃圾回收线程就是特殊的守护线程。

38. **多线程有几种实现方式？** 有4种，分别是： 继承Thread类 实现Runnable接口 实现Callable接口通过FutureTask包装器来创建Thread线程 通过线程池创建线程，使用线程池接口ExecutorService结合Callable、Future实现有返回结果的多线程。 前面两种【无返回值】原因：通过重写run方法，run方法的返回值是void，所以没有办法返回结果。 后面两种【有返回值】原因：通过Callable接口，就要实现call方法，这个方法的返回值是Object，所以返回的结果可以放在Object对象中。

39. **说一下 Runnable和 Callable有什么区别？** Runnable没有返回值，Callable可以拿到有返回值，Callable可以看作是 Runnable的补充。

40. **线程有哪些状态？** 线程的6种状态： 初始(NEW)：新创建了一个线程对象，但还没有调用start()方法。 运行(RUNNABLE)：Java线程中将就绪（ready）和运行中（running）两种状态笼统的称为"运行"。线程对象创建后，其他线程(比如main线程）调用了该对象的start()方法。该状态的线程位于可运行线程池中，等待被线程调度选中，获取CPU的使用权，此时处于就绪状态（ready）。就绪状态的线程在获得CPU时间片后变为运行中状态（running）。 阻塞(BLOCKED)：表示线程阻塞于锁。 等待(WAITING)：进入该状态的线程需要等待其他线程做出一些特定动作（通知或中断）。 超时等待(TIMED_WAITING)：该状态不同于WAITING，它可以在指定的时间后自行返回。 终止(TERMINATED)：表示该线程已经执行完毕。

41. **sleep() 和 wait() 有什么区别？** 类的不同：sleep() 来自 Thread，wait() 来自 Object。 释放锁：sleep() 不释放锁；wait() 释放锁。 用法不同：sleep() 时间到会自动恢复；wait() 可以使用 notify()/notifyAll()直接唤醒。

42. **notify()和 notifyAll()有什么区别？** notifyAll()会唤醒所有的线程，notify()之后唤醒一个线程。 notifyAll() 调用后，会将全部线程由等待池移到锁池，然后参与锁的竞争，竞争成功则继续执行，如果不成功则留在锁池等待锁被释放后再次参与竞争。而 notify()只会唤醒一个线程，具体唤醒哪一个线程由虚拟机控制。

43. **线程的 run() 和 start() 有什么区别？** start() 方法用于启动线程，run() 方法用于执行线程的运行时代码。run() 可以重复调用，而 start() 只能调用一次。

44. **创建线程池有哪几种方式？** 线程池创建有七种方式，最核心的是最后一种： newSingleThreadExecutor()：它的特点在于工作线程数目被限制为 1，操作一个无界的工作队列，所以它保证了所有任务的都是被顺序执行，最多会有一个任务处于活动状态，并且不允许使用者改动线程池实例，因此可以避免其改变线程数目； newCachedThreadPool()：它是一种用来处理大量短时间工作任务的线程池，具有几个鲜明特点：它会试图缓存线程并重用，当无缓存线程可用时，就会创建新的工作线程；如果线程闲置的时间超过 60 秒，则被终止并移出缓存；长时间闲置时，这种线程池，不会消耗什么资源。其内部使用 SynchronousQueue 作为工作队列； newFixedThreadPool(int nThreads)：重用指定数目（nThreads）的线程，其背后使用的是无界的工作队列，任何时候最多有 nThreads 个工作线程是活

动的。这意味着，如果任务数量超过了活动队列数目，将在工作队列中等待空闲线程出现；如果有工作线程退出，将会有新的工作线程被创建，以补足指定的数目 nThreads； newSingleThreadScheduledExecutor()：创建单线程池，返回 ScheduledExecutorService，可以进行定时或周期性的工作调度； newScheduledThreadPool(int corePoolSize)：和newSingleThreadScheduledExecutor()类似，创建的是个 ScheduledExecutorService，可以进行定时或周期性的工作调度，区别在于单一工作线程还是多个工作线程； newWorkStealingPool(int parallelism)：这是一个经常被人忽略的线程池，Java 8 才加入这个创建方法，其内部会构建ForkJoinPool，利用Work-Stealing算法，并行地处理任务，不保证处理顺序； ThreadPoolExecutor()：是最原始的线程池创建，上面1-3创建方式都是对ThreadPoolExecutor的封装。

45. 线程池都有哪些状态？ RUNNING：这是最正常的状态，接受新的任务，处理等待队列中的任务。 SHUTDOWN：不接受新的任务提交，但是会继续处理等待队列中的任务。 STOP：不接受新的任务提交，不再处理等待队列中的任务，中断正在执行任务的线程。 TIDYING：所有的任务都销毁了，workCount 为 0，线程池的状态在转换为 TIDYING 状态时，会执行钩子方法 terminated()。 TERMINATED：terminated()方法结束后，线程池的状态就会变成这个。

46. 线程池中 **submit()** 和 **execute()** 方法有什么区别？ execute()：只能执行 Runnable 类型的任务。 submit()：可以执行 Runnable 和 Callable 类型的任务。 Callable 类型的任务可以获取执行的返回值，而 Runnable 执行无返回值。

47. 在 **Java** 程序中怎么保证多线程的运行安全？ 方法一：使用安全类，比如 Java. util. concurrent 下的类。 方法二：使用自动锁 synchronized。 方法三：使用手动锁 Lock。 手动锁 Java 示例代码如下：

```
Lock lock = new ReentrantLock();
lock. lock();
try {
System. out. println("获得锁");
} catch (Exception e) {
// TODO: handle exception
} finally {
System. out. println("释放锁");
lock. unlock();
}
```

48. 多线程中 **synchronized** 锁升级的原理是什么？ synchronized 锁升级原理：在锁对象的对象头里面有一个 threadid 字段，在第一次访问的时候 threadid 为空，jvm 让其持有偏向锁，并将 threadid 设置为其线程 id，再次进入的时候会先判断 threadid 是否与其线程 id 一致，如果一致则可以直接使用此对象，如果不一致，则升级偏向锁为轻量级锁，通过自旋循环一定次数来获取锁，执行一定次数之后，如果还没有正常获取到要使用的对象，此时就会把锁从轻量级升级为重量级锁，此过程就构成了 synchronized 锁的升级。 锁的升级的目的：锁升级是为了减低了锁带来的性能消耗。在 Java 6 之后优化 synchronized 的实现方式，使用了偏向锁升级为轻量级锁再升级到重量级锁的方式，从而减低了锁带来的性能消耗。

49. 什么是死锁？ 当线程 A 持有独占锁a，并尝试去获取独占锁 b 的同时，线程 B 持有独占锁 b，并尝试获取独占锁 a 的情况下，就会发生 AB 两个线程由于互相持有对方需要的锁，而发生的阻塞现象，我们称为死锁。

50. 怎么防止死锁？ 尽量使用 tryLock(long timeout, TimeUnit unit)的方法(ReentrantLock、ReentrantReadWriteLock)，设置超时时间，超时可以退出防止死锁。 尽量使用 Java. util. concurrent 并

发类代替自己手写锁。 尽量降低锁的使用粒度，尽量不要几个功能用同一把锁。 尽量减少同步的代码块。

51. **ThreadLocal 是什么？有哪些使用场景？** ThreadLocal 为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立地改变自己的副本，而不会影响其它线程所对应的副本。 ThreadLocal 的经典使用场景是数据库连接和 session 管理等。

52. **说一下 synchronized 底层实现原理？** synchronized 是由一对 monitorenter/monitorexit 指令实现的，monitor 对象是同步的基本实现单元。在 Java 6 之前，monitor 的实现完全是依靠操作系统内部的互斥锁，因为需要进行用户态到内核态的切换，所以同步操作是一个无差别的重量级操作，性能也很低。但在 Java 6 的时候，Java 虚拟机 对此进行了大刀阔斧地改进，提供了三种不同的 monitor 实现，也就是常说的三种不同的锁：偏向锁（Biased Locking）、轻量级锁和重量级锁，大大改进了其性能。

53. **synchronized 和 volatile 的区别是什么？** volatile 是变量修饰符；synchronized 是修饰类、方法、代码段。 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。

54. **synchronized 和 Lock 有什么区别？** synchronized 可以给类、方法、代码块加锁；而 lock 只能给代码块加锁。 synchronized 不需要手动获取锁和释放锁，使用简单，发生异常会自动释放锁，不会造成死锁；而 lock 需要自己加锁和释放锁，如果使用不当没有 unLock()去释放锁就会造成死锁。 通过 Lock 可以知道有没有成功获取锁，而 synchronized 却无法办到。

55. **synchronized 和 ReentrantLock 区别是什么？** synchronized 早期的实现比较低效，对比 ReentrantLock，大多数场景性能都相差较大，但是在 Java 6 中对 synchronized 进行了非常多的改进。主要区别如下： ReentrantLock 使用起来比较灵活，但是必须有释放锁的配合动作； ReentrantLock 必须手动获取与释放锁，而 synchronized 不需要手动释放和开启锁； ReentrantLock 只适用于代码块锁，而 synchronized 可用于修饰方法、代码块等。 ReentrantLock 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。 56.说一下 atomic 的原理？ atomic 主要利用 CAS (Compare And Swap) 和 volatile 和 native 方法来保证原子操作，从而避免 synchronized 的高开销，执行效率大为提升。

四. Java 反射模块

57. **什么是反射？** 反射是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为 Java 语言的反射机制。

58. **什么是 Java 序列化？什么情况下需要序列化？** Java 序列化是为了保存各种对象在内存中的状态，并且可以把保存的对象状态再读出来。 以下情况需要使用 Java 序列化： 想把的内存中的对象状态保存到一个文件中或者数据库中时候； 想用套接字在网络上传送对象的时候； 想通过RMI（远程方法调用）传输对象的时候。

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。 动态代理的应用有 spring aop、hibernate 数据查询、测试框架的后端 mock、rpc，Java注解对象获取等。

60. **怎么实现动态代理？** JDK 原生动态代理和 cglib 动态代理。JDK 原生动态代理是基于接口实现的，而 cglib 是基于继承当前类的子类实现的。

五. Java 对象拷贝模块

61. **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。

62. **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。

63. **深拷贝和浅拷贝区别是什么？** 浅拷贝：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。 深拷贝：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。

六. Java Web模块

64. **JSP 和 servlet 有什么区别？** JSP 是 servlet 技术的扩展，本质上就是 servlet 的简易方式。servlet 和 JSP 最主要的不同点在于，servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 html 里分离开来，而 JSP 的情况是 Java 和 html 可以组合成一个扩展名为 JSP 的文件。JSP 侧重于视图，servlet 主要用于控制逻辑。

65. **JSP 有哪些内置对象？作用分别是什么？** JSP 有 9 大内置对象： 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？** page：代表与一个页面相关的对象和属性。 request：代表与客户端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？** session：是一种将会话状态保存在服务器端的技术。 Cookie ：是在 HTTP 协议下， Web 服务器保存在用户浏览器（客户端）上的小文本文件，它可以包含有关用户的信息。无论何时用户链接到服务器，Web 站点都可以访问 Cookie 信息 。 存储位置不同：session 存储在服务器端；cookie 存储在浏览器端。 安全性不同：cookie 安全性一般，在浏览器存储，可以被伪造和修改。 容量和个数限制：cookie 有容量限制，每个站点下的 cookie 也有个数限制。 存储的多样性：session 可以存储在 Redis 中、数据库中、应用程序中；而 cookie 只能存储在浏览器中。

68. **说一下 session 的工作原理？** session 的工作原理是客户端登录完成之后，服务器会创建对应的 session，session 创建完之后，会把 session 的 id 发送给客户端，客户端再存储到浏览器中。这样客户端每次访问服务器时，都会带着 sessionid，服务器拿到 sessionid 之后，在内存找到与之对应的 session 这样就可以正常工作了。

69. **如果客户端禁止 cookie 能实现 session 还能用吗？** 可以用，session 只是依赖 cookie 存储 sessionid，如果 cookie 被禁用了，可以使用 url 中添加 sessionid 的方式保证 session 能正常使用。

70. **spring mvc 和 struts 的区别是什么？** 拦截级别：struts2 是类级别的拦截；spring mvc 是方法级别的拦截。 数据独立性：spring mvc 的方法之间基本上独立的，独享 request 和 response 数据，请求数据通过参数获取，处理结果通过 ModelMap 交回给框架，方法之间不共享变量；而 struts2 虽然方法之间也是独立的，但其所有 action 变量是共享的，这不会影响程序运行，却给我们编码和读程序时带来了一定的麻烦。 拦截机制：struts2 有以自己的 interceptor 机制，spring mvc 用的是独立的 aop 方式，这样导致struts2 的配置文件量比 spring mvc 大。 对 ajax 的支持：spring mvc 集成了ajax，所有 ajax 使用很

方便，只需要一个注解 @ResponseBody 就可以实现了；而 struts2 一般需要安装插件或者自己写代码才行。

71. **如何避免 SQL 注入？** 使用预处理 PreparedStatement。 使用正则表达式过滤掉字符中的特殊字符。

72. **什么是 XSS 攻击，如何避免？** XSS 攻击：即跨站脚本攻击，它是 Web 程序中常见的漏洞。原理是攻击者往 Web 页面里插入恶意的脚本代码（css 代码、Javascript 代码等），当用户浏览该页面时，嵌入其中的脚本代码会被执行，从而达到恶意攻击用户的目的，如盗取用户 cookie、破坏页面结构、重定向到其他网站等。 预防 XSS 的核心是必须对输入的数据做过滤处理。

73. **什么是 CSRF 攻击，如何避免？** CSRF：Cross-Site Request Forgery（中文：跨站请求伪造），可以理解为攻击者盗用了你的身份，以你的名义发送恶意请求，比如：以你名义发送邮件、发消息、购买商品，虚拟货币转账等。 防御手段： 验证请求来源地址； 关键操作添加验证码； 在请求地址添加 token 并验证。

七. Java 异常模块

74. **throw 和 throws 的区别？** throw：是真实抛出一个异常。 throws：是声明可能会抛出一个异常。

75. **final、finally、finalize 有什么区别？** final：是修饰符，如果修饰类，此类不能被继承；如果修饰方法和变量，则表示此方法和此变量不能在被改变，只能使用。 finally：是 try{} catch{} finally{} 最后一部分，表示不论发生任何情况都会执行，finally 部分可以省略，但如果 finally 部分存在，则一定会执行 finally 里面的代码。 finalize：是Object的protected方法，子类可以覆盖该方法以实现资源清理工作，GC在回收对象之前调用该方法。

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？** finally 一定会执行，即使是 catch 中 return 了，catch 中的 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

八. 网络模块

79. **http 响应码 301 和 302 代表的是什么？有什么区别？** 301：永久重定向；302：暂时重定向。 它们的区别是，301 对搜索引擎优化（SEO）更加有利；302 有被提示为网络拦截的风险。

80. **forward 和 redirect 的区别？** forward 是转发 和 redirect 是重定向： 地址栏 url 显示：foward url 不会发生改变，redirect url 会发生改变； 数据共享：forward 可以共享 request 里的数据，redirect 不能共享； 效率：forward 比 redirect 效率高。

81. **简述 tcp 和 udp的区别？** tcp 和 udp 是 OSI 模型中的运输层中的协议。tcp 提供可靠的通信传输，而 udp 则常被用于让广播和细节控制交给应用的通信传输。 两者的区别大致如下： tcp 面向连接，udp 面向非连接即发送数据前不需要建立链接； tcp 提供可靠的服务（数据传输），udp 无法保证； tcp 面向字节流，udp 面向报文； tcp 数据传输慢，udp 数据传输快；

82. **tcp 为什么要三次握手，两次不行吗？为什么？** 我们假设A和B是通信的双方。我理解的握手实际上就是通信，发一次信息就是进行一次握手。 第一次握手：A给B打电话说，你可以听到我说话吗？ 第二次握手：B收到了A的信息，然后对A说：我可以听得到你说话啊，你能听得到我说话吗？ 第三次握手：A收到了B的信息，然后说可以的，我要给你发信息啦！ 在三次握手之后，A和B都能确定这么一件事：我说的话，你能听到；你说的话，我也能听到。这样，就可以开始正常通信了。 注意：HTTP是基于TCP协议的，所以每次都是客户端发送请求，服务器应答，但是TCP还可以给其他应用层提供服务，即可能A、B在建立链接之后，谁都可能先开始通信。 如果采用两次握手，那么只要服务器发出确认数据包就会建立连接，但由于客户端此时并未响应服务器端的请求，那此时服务器端就会一直在等待客户端，这样服务器端就白白浪费了一定的资源。若采用三次握手，服务器端没有收到来自客户端的再此确认，则就会知道客户端并没有要求建立请求，就不会浪费服务器的资源。

83. **说一下 tcp 粘包是怎么产生的？** tcp 粘包可能发生在发送端或者接收端，分别来看两端各种产生粘包的原因： 发送端粘包：发送端需要等缓冲区满才发送出去，造成粘包； 接收方粘包：接收方不及时接收缓冲区的包，造成多个包接收。

84. **OSI 的七层模型都有哪些？** 物理层：利用传输介质为数据链路层提供物理连接，实现比特流的透明传输。 数据链路层：负责建立和管理节点间的链路。 网络层：通过路由选择算法，为报文或分组通过通信子网选择最适当的路径。 传输层：向用户提供可靠的端到端的差错和流量控制，保证报文的正确传输。 会话层：向两个实体的表示层提供建立和使用连接的方法。 表示层：处理用户信息的表示问题，如编码、数据格式转换和加密解密等。 应用层：直接向用户提供服务，完成用户希望在网络上完成的各种工作。

85. **get 和 post 请求有哪些区别？** get 请求会被浏览器主动缓存，而 post 不会。 get 传递参数有大小限制，而 post 没有。 post 参数传输更安全，get 的参数会明文限制在 url 上，post 不会。

86. **如何实现跨域？** 实现跨域有以下几种方案： 服务器端运行跨域 设置 CORS 等于 *； 在单个接口使用注解 @CrossOrigin 运行跨域； 使用 jsonp 跨域；

87. **说一下 JSONP 实现原理？** jsonp：JSON with Padding，它是利用script标签的 src 连接可以访问不同源的特性，加载远程返回的“JS 函数”来执行的。

九. 设计模式模块

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。

十. Spring/Spring MVC模块

90. **为什么要使用 spring？** spring 提供 ioc 技术，容器会帮你管理依赖的对象，从而不需要自己创建和管理依赖对象了，更轻松的实现了程序的解耦。 spring 提供了事务支持，使得事务操作变的更加方便。

spring 提供了面向切片编程，这样可以更方便的处理某一类的问题。 更方便的框架集成，spring 可以很方便的集成其他框架，比如 MyBatis、hibernate 等。

91. **解释一下什么是 aop？** aop 是面向切面编程，通过预编译方式和运行期动态代理实现程序功能的统一维护的一种技术。 简单来说就是统一处理某一“切面”（类）的问题的编程思想，比如统一处理日志、异常等。

92. **解释一下什么是 ioc？** ioc：Inversionof Control（中文：控制反转）是 spring 的核心，对于 spring 框架来说，就是由 spring 来负责控制对象的生命周期和对象间的关系。 简单来说，控制指的是当前对象对内部成员的控制权；控制反转指的是，这种控制权不由当前对象管理了，由其他（类,第三方容器）来管理。

93. **spring 有哪些主要模块？** 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。

94. **spring 常用的注入方式有哪些？** setter 属性注入 构造方法注入 注解方式注入

95. **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。

96. **spring 支持几种 bean 的作用域？** spring 支持 5 种作用域，如下： 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； 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。 注意：使用 prototype 作用域需要慎重的思考，因为频繁创建和销毁 bean 会带来很大的性能开销。

97. **spring 自动装配 bean 有哪些方式？** no：默认值，表示没有自动装配，应使用显式 bean 引用进行装配。 byName：它根据 bean 的名称注入对象依赖项。 byType：它根据类型注入对象依赖项。 构造函数：通过构造函数来注入依赖项，需要设置大量的参数。 autodetect：容器首先通过构造函数使用 autowire 装配，如果不能，则通过 byType 自动装配。

98. **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。

99. **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。

100. **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。

101. **spring mvc 有哪些组件？** 前置控制器 DispatcherServlet。 映射控制器 HandlerMapping。 处理器 Controller。 模型和视图 ModelAndView。 视图解析器 ViewResolver。

102. **@RequestMapping 的作用是什么？** 将 http 请求映射到相应的类/方法上。

103. **@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。

十一. Spring Boot/Spring Cloud模块

104. **什么是 spring boot？** spring boot 是为 spring 服务的，是用来简化新 spring 应用的初始搭建以及开发过程的。

105. **为什么要用 spring boot？** 配置简单 独立运行 自动装配 无代码生成和 xml 配置 提供应用监控 易上手 提升开发效率

106. **spring boot 核心配置文件是什么？** spring boot 核心的两个配置文件： bootstrap (. yml 或者 . properties)：boostrap 由父 ApplicationContext 加载的，比 applicaton 优先加载，且 boostrap 里面的属性不能被覆盖； application (. yml 或者 . properties)：用于 spring boot 项目的自动化配置。

107. **spring boot 配置文件有哪几种类型？它们有什么区别？** 配置文件有 . properties 格式和 . yml 格式，它们主要的区别是书法风格不同。 properties 配置如下： spring. RabbitMQ. port=5672 yml 配置如下：

```
spring:
    RabbitMQ:
        port: 5672
```

yml 格式不支持 @PropertySource 注解导入。

108. **spring boot 有哪些方式可以实现热部署？** 使用 devtools 启动热部署，添加 devtools 库，在配置文件中把 spring. devtools. restart. enabled 设置为 true； 使用 Intellij Idea 编辑器，勾上自动编译或手动重新编译。 109.jpa 和 hibernate 有什么区别？ jpa 全称 Java Persistence API，是 Java 持久化接口规范，hibernate 属于 jpa 的具体实现。

109. 什么是 **spring cloud**？ spring cloud 是一系列框架的有序集合。它利用 spring boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 spring boot 的开发风格做到一键启动和部署。

110. **spring cloud** 断路器的作用是什么？ 在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

111. **spring cloud** 的核心组件有哪些？ Eureka：服务注册于发现。 Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。 Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。 Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。 Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

十二. Hibernate模块

113. 为什么要使用 **hibernate**？ hibernate 是对 jdbc 的封装，大大简化了数据访问层的繁琐的重复性代码。hibernate 是一个优秀的 ORM 实现，很多程度上简化了 DAO 层的编码功能。 可以很方便的进行数据库的移植工作。 提供了缓存机制，是程序执行更改的高效。

114. 什么是 **ORM** 框架？ ORM（Object Relation Mapping）对象关系映射，是把数据库中的关系数据映射成为程序中的对象。 使用 ORM 的优点：提高了开发效率降低了开发成本、开发更简单更对象化、可移植更强。

115. **hibernate** 中如何在控制台查看打印的 **SQL** 语句？ 在 Config 里面把 hibernate. show_SQL 设置为 true 就可以。但不建议开启，开启之后会降低程序的运行效率。

116. **hibernate** 有几种查询方式？ 三种：hql、原生 SQL、条件查询 Criteria。

117. **hibernate** 实体类可以被定义为 **final** 吗？ 实体类可以定义为 final 类，但这样的话就不能使用 hibernate 代理模式下的延迟关联提供性能了，所以不建议定义实体类为 final。

118. 在 **hibernate** 中使用 **Integer** 和 **int** 做映射有什么区别？ Integer 类型为对象，它的值允许为 null，而 int 属于基础数据类型，值不能为 null。

119. **hibernate** 是如何工作的？ 读取并解析配置文件。 读取并解析映射文件，创建 SessionFactory。 打开 Session。 创建事务。 进行持久化操作。 提交事务。 关闭 Session。 关闭 SessionFactory。

120. **get()**和 **load()**的区别？ 数据查询时，没有 OID 指定的对象，get() 返回 null；load() 返回一个代理对象。 load()支持延迟加载；get() 不支持延迟加载。

121. 说一下 **hibernate** 的缓存机制？ hibernate 常用的缓存有一级缓存和二级缓存： 一级缓存：也叫 Session 缓存，只在 Session 作用范围内有效，不需要用户干涉，由 hibernate 自身维护，可以通过：evict(object)清除 object 的缓存；clear()清除一级缓存中的所有缓存；flush()刷出缓存； 二级缓存：应用级别的缓存，在所有 Session 中都有效，支持配置第三方的缓存，如：EhCache。

122. **hibernate** 对象有哪些状态？ 临时/瞬时状态：直接 new 出来的对象，该对象还没被持久化（没保存在数据库中），不受 Session 管理。 持久化状态：当调用 Session 的 save/saveOrupdate/get/load/list 等方法的时候，对象就是持久化状态。 游离状态：Session 关闭之后对象就是游离状态。

123. 在 **hibernate** 中 **getCurrentSession** 和 **openSession** 的区别是什么？ getCurrentSession 会绑定当前线程，而 openSession 则不会。 getCurrentSession 事务是 Spring 控制的，并且不需要手动关闭，而 openSession 需要我们自己手动开启和提交事务。

124. **hibernate** 实体类必须要有无参构造函数吗？为什么？ hibernate 中每个实体类必须提供一个无参构造函数，因为 hibernate 框架要使用 reflection api，通过调用 ClassnewInstance() 来创建实体类的实例，如果没有无参的构造函数就会抛出异常。

十三. MyBatis模块

125. **MyBatis** 中 **#{}**和 **${}**的区别是什么？ #{}是预编译处理，${}是字符替换。在使用 #{}时，MyBatis 会将 SQL 中的 #{}替换成"?"，配合 PreparedStatement 的 set 方法赋值，这样可以有效的防止 SQL 注入，保证程序的运行安全。

126. **MyBatis** 有几种分页方式？ 分页方式：逻辑分页和物理分页。 逻辑分页：使用 MyBatis 自带的 RowBounds 进行分页，它是一次性查询很多数据，然后在数据中再进行检索。 物理分页：自己手写 SQL 分页或使用分页插件 PageHelper，去数据库查询指定条数的分页数据的形式。

127. **RowBounds** 是一次性查询全部结果吗？为什么？ RowBounds 表面是在"所有"数据中检索数据，其实并非是一次性查询出所有数据，因为 MyBatis 是对 jdbc 的封装，在 jdbc 驱动中有一个 Fetch Size 的配置，它规定了每次最多从数据库查询多少条数据，假如你要查询更多数据，它会在你执行 next()的时候，去查询更多的数据。就好比你去自动取款机取 10000 元，但取款机每次最多能取 2500 元，所以你要取 4 次才能把钱取完。只是对于 jdbc 来说，当你调用 next()的时候会自动帮你完成查询工作。这样做的好处可以有效的防止内存溢出。 Fetch Size 官方相关文档：http://t. cn/EfSE2g3

128. **MyBatis** 逻辑分页和物理分页的区别是什么？ 逻辑分页是一次性查询很多数据，然后再在结果中检索分页的数据。这样做弊端是需要消耗大量的内存、有内存溢出的风险、对数据库压力较大。 物理分页是从数据库查询指定条数的数据，弥补了一次性全部查出的所有数据的种种缺点，比如需要大量的内存，对数据库查询压力较大等问题。

129. **MyBatis** 是否支持延迟加载？延迟加载的原理是什么？ MyBatis 支持延迟加载，设置 lazyLoadingEnabled=true 即可。 延迟加载的原理的是调用的时候触发加载，而不是在初始化的时候就加载信息。比如调用 a. getB(). getName()，这个时候发现 a. getB() 的值为 null，此时会单独触发事先保存好的关联 B 对象的 SQL，先查询出来 B，然后再调用 a. setB(b)，而这时候再调用 a. getB(). getName() 就有值了，这就是延迟加载的基本原理。

130. 说一下 **MyBatis** 的一级缓存和二级缓存？ 一级缓存：基于 PerpetualCache 的 HashMap 本地缓存，它的声明周期是和 SQLSession 一致的，有多个 SQLSession 或者分布式的环境中数据库操作，可能会出现脏数据。当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认一级缓存是开启的。 二级缓存：也是基于 PerpetualCache 的 HashMap 本地缓存，不同在于其存储作用域为 Mapper 级别的，如果多个SQLSession之间需要共享缓存，则需要使用到二级缓存，并且二级缓存可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态)。 开启二级缓存数据查询流程：二级缓存 -> 一级缓存 -> 数据库。 缓存更新机制：当某一个作用域(一级缓存 Session/二级缓存 Mapper)进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

131. **MyBatis** 和 **hibernate** 的区别有哪些？ 灵活性：MyBatis 更加灵活，自己可以写 SQL 语句，使用起来比较方便。 可移植性：MyBatis 有很多自己写的 SQL，因为每个数据库的 SQL 可以不相同，所以可移植

性比较差。 学习和使用门槛：MyBatis 入门比较简单，使用门槛也更低。 二级缓存：hibernate 拥有更好的二级缓存，它的二级缓存可以自行更换为第三方的二级缓存。

132. **MyBatis 有哪些执行器（Executor）？** MyBatis 有三种基本的Executor执行器： SimpleExecutor：每执行一次 update 或 select 就开启一个 Statement 对象，用完立刻关闭 Statement 对象； ReuseExecutor：执行 update 或 select，以 SQL 作为 key 查找 Statement 对象，存在就使用，不存在就创建，用完后不关闭 Statement 对象，而是放置于 Map 内供下一次使用。简言之，就是重复使用 Statement 对象； BatchExecutor：执行 update（没有 select，jdbc 批处理不支持 select），将所有 SQL 都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个 Statement 对象，每个 Statement 对象都是 addBatch()完毕后，等待逐一执行 executeBatch()批处理，与 jdbc 批处理相同。

133. **MyBatis 分页插件的实现原理是什么？** 分页插件的基本原理是使用 MyBatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 SQL，然后重写 SQL，根据 dialect 方言，添加对应的物理分页语句和物理分页参数。

134. **MyBatis 如何编写一个自定义插件？** 自定义插件实现原理： MyBatis 自定义插件针对 MyBatis 四大对象（Executor、StatementHandler、ParameterHandler、ResultSetHandler）进行拦截： Executor：拦截内部执行器，它负责调用 StatementHandler 操作数据库，并把结果集通过 ResultSetHandler 进行自动映射，另外它还处理了二级缓存的操作； StatementHandler：拦截 SQL 语法构建的处理，它是 MyBatis 直接和数据库执行 SQL 脚本的对象，另外它也实现了 MyBatis 的一级缓存； ParameterHandler：拦截参数的处理； ResultSetHandler：拦截结果集的处理。 自定义插件实现关键： MyBatis 插件要实现 Interceptor 接口，接口包含的方法，如下：

```
public interface Interceptor {
Object intercept(Invocation invocation) throws Throwable;
Object plugin(Object target);
void setProperties(Properties properties);
}
```

setProperties 方法是在 MyBatis 进行配置插件的时候可以配置自定义相关属性，即：接口实现对象的参数配置； plugin 方法是插件用于封装目标对象的，通过该方法我们可以返回目标对象本身，也可以返回一个它的代理，可以决定是否要进行拦截进而决定要返回一个什么样的目标对象，官方提供了示例：return Plugin.wrap(target, this)； intercept 方法就是要进行拦截的时候要执行的方法。 自定义插件实现示例： 官方插件实现：

```
@Intercepts({@Signature(type = Executor. class, method = "query",
args = {MappedStatement. class, Object. class, RowBounds. class, ResultHandler.
class})})
public class TestInterceptor implements Interceptor {
public Object intercept(Invocation invocation) throws Throwable {
Object target = invocation. getTarget(); //被代理对象
Method method = invocation. getMethod(); //代理方法
Object[] args = invocation. getArgs(); //方法参数
// do something . . . . . 方法拦截前执行代码块
Object result = invocation. proceed();
// do something . . . . . 方法拦截后执行代码块
return result;
}
```

```
public Object plugin(Object target) {
return Plugin. wrap(target, this);
}
}
```

十四. RabbitMQ模块

135. **RabbitMQ 的使用场景有哪些？** 抢购活动，削峰填谷，防止系统崩塌。 延迟信息处理，比如 10 分钟之后给下单未付款的用户发送邮件提醒。 解耦系统，对于新增的功能可以单独写模块扩展，比如用户确认评价之后，新增了给用户返积分的功能，这个时候不用在业务代码里添加新增积分的功能，只需要把新增积分的接口订阅确认评价的消息队列即可，后面再添加任何功能只需要订阅对应的消息队列即可。

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：每个 RabbitMQ 都能创建很多 vhost，我们称之为虚拟主机，每个虚拟主机其实都是 mini 版的RabbitMQ，它拥有自己的队列，交换器和绑定，拥有自己的权限机制。

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？** 把消息持久化磁盘，保证服务器重启消息不丢失。 每个集群中至少有一个物理磁盘，保证消息落入磁盘。

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？** direct（默认方式）：最基础最简单的模式，发送方把消息发送给订阅方，如果有多个订阅者，默认采取轮询的方式进行消息发送。 headers：与 direct 类似，只是性能很差，此类型几乎用不到。 fanout：分发模式，把消费分发给所有订阅者。 topic：匹配订阅模式，使用正则匹配到消息队列，能匹配到的都能接收到。

145. **RabbitMQ 怎么实现延迟消息队列？** 延迟队列的实现有两种方式： 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。

十五. 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。

十六. Zookeeper模块

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。

十七. MySQL模块

164. **数据库的三范式是什么？** 第一范式（1NF）：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（2NF）：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。（在1NF基础上消除非主属性对主键的部分函数依赖） 第三范式（3NF）：任何非主属性不依赖于其它非主属性。（在2NF基础上消除传递依赖）

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。 InnoDB 表只会把自增主键的最大 id 记录在内存中，所以重启之后会导致最大 id 丢失。

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。 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。

170. **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 语法：explain select * from table where type=1。

173. **说一下数据库的事务隔离？** MySQL 的事务隔离是在 MySQL. ini 配置文件里添加的，在文件的最后添加： transaction-isolation = REPEATABLE-READ 1 可用的配置值：READ-UNCOMMITTED、READ-COMMITTED、REPEATABLE-READ、SERIALIZABLE。 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的。*
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。 …

十八. Redis模块

179. **Redis 是什么？都有哪些使用场景？** Redis 是一个使用 C 语言开发的高速缓存数据库。 redis是一个key-value存储系统。和Memcached类似，它支持存储的value类型相对更多，包括string(字符串)、list(链表)、set(集合)、zset(sorted set --有序集合)和hash（哈希类型）。这些数据类型都支持push/pop、add/remove及取交集并集和差集及更丰富的操作，而且这些操作都是原子性的。在此基础上，redis支持各种不同方式的排序。与memcached一样，为了保证效率，数据都是缓存在内存中。区别的是redis会周期性的把更新的数据写入磁盘或者把修改操作写入追加的记录文件，并且在此基础上实现了master-slave主从同步【主从同步：数据可以从主服务器向任意数量的从服务器上同步】。 Redis 是一个高性能的key-value数据库。redis的出现，很大程度补偿了memcached这类key/value存储的不足，在部 分场合可以对关系数据库起到很好的补充作用。它提供了Java，C/C++，C#，PHP，JavaScript，Perl，Object-C，Python，Ruby，Erlang等客户端，使用很方便。 Redis 使用场景： 记录帖子点赞数、点击数、评论数； 缓存近期热帖； 缓存文章详情信息； 记录用户会话信息。

180. **Redis 有哪些功能？** 数据缓存功能 分布式锁的功能 支持数据持久化 支持事务 支持消息队列

181. **Redis 和 memcache 有什么区别？** 存储方式不同：memcache 把数据全部存在内存之中，断电后会挂掉，数据不能超过内存大小；Redis 有部份存在硬盘上，这样能保证数据的持久性。 数据支持类型：memcache 对数据类型支持相对简单；Redis 有复杂的数据类型。 使用底层模型不同：它们之间底层实现方式，以及与客户端之间通信的应用协议不一样，Redis 自己构建了 vm 机制，因为一般的系统调用系统函数的话，会浪费一定的时间去移动和请求。 value 值大小不同：Redis 最大可以达到 1gb；memcache 只有 1mb。

182. **Redis 为什么是单线程的？** 因为 cpu 不是 Redis 的瓶颈，Redis 的瓶颈最有可能是机器内存或者网络带宽。既然单线程容易实现，而且 cpu 又不会成为瓶颈，那就顺理成章地采用单线程的方案了。 关于 Redis 的性能，官方网站也有，普通笔记本轻松处理每秒几十万的请求。而且单线程并不代表就慢，nginx 和 node.js 也都是高性能单线程的代表。

183. **什么是缓存穿透？怎么解决？** 缓存穿透：指查询一个一定不存在的数据，由于缓存是不命中时需要从数据库查询，查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到数据库去查询，造成缓存穿透。 解决方案：最简单粗暴的方法如果一个查询返回的数据为空（不管是数据不存在，还是系统故障），我们就把这个空结果进行缓存，但它的过期时间会很短，最长不超过五分钟。

184. **Redis 支持的数据类型有哪些？** Redis 支持的数据类型：string（字符串）、list（列表）、hash（字典）、set（集合）、zset（有序集合）。

185. **Redis 支持的 Java 客户端都有哪些？** 支持的 Java 客户端有 Redisson、jedis、lettuce 等。

186. **jedis 和 Redisson 有哪些区别？** jedis：提供了比较全面的 Redis 命令的支持。 Redisson：实现了分布式和可扩展的 Java 数据结构，与 jedis 相比 Redisson 的功能相对简单，不支持排序、事务、管道、分区等 Redis 特性。

187. **怎么保证缓存和数据库数据的一致性？** 合理设置缓存的过期时间。 新增、更改、删除数据库操作时同步更新 Redis，可以使用事物机制来保证数据的一致性。

188. **Redis 持久化有几种方式？** Redis 的持久化有两种方式，或者说有两种策略： RDB（Redis Database）：指定的时间间隔能对你的数据进行快照存储。 AOF（Append Only File）：每一个收到的写命令都通过write函数追加到文件中。

189. **Redis 怎么实现分布式锁？** Redis 分布式锁其实就是在系统里面占一个“坑”，其他程序也要占“坑”的时候，占用成功了就可以继续执行，失败了就只能放弃或稍后重试。 占坑一般使用 setnx(set if not exists) 指令，只允许被一个程序占有，使用完调用 del 释放锁。

190. **Redis 分布式锁有什么缺陷？** Redis 分布式锁不能解决超时的问题，分布式锁有一个超时时间，程序的执行如果超出了锁的超时时间就会出现问题。

191. **Redis 如何做内存优化？** 尽量使用 Redis 的散列表，把相关的信息放到散列表里面存储，而不是把每个字段单独存储，这样可以有效的减少内存使用。比如将 Web 系统的用户对象，应该放到散列表里面再整体存储到 Redis，而不是把用户的姓名、年龄、密码、邮箱等字段分别设置 key 进行存储。

192. **Redis 淘汰策略有哪些？** volatile-lru：从已设置过期时间的数据集（server. db[i]. expires）中挑选最近最少使用的数据淘汰。 volatile-ttl：从已设置过期时间的数据集（server. db[i]. expires）中挑选将要过期的数据淘汰。 volatile-random：从已设置过期时间的数据集（server. db[i]. expires）中任意选择数据淘汰。 allkeys-lru：从数据集（server. db[i]. dict）中挑选最近最少使用的数据淘汰。 allkeys-random：从数据集（server. db[i]. dict）中任意选择数据淘汰。 no-enviction（驱逐）：禁止驱逐数据。 193.Redis 常见的性能问题有哪些？该如何解决？ 主服务器写内存快照，会阻塞主线程的工作，当快照比较大时对性能影响是非常大的，会间断性暂停服务，所以主服务器最好不要写内存快照。 Redis 主从复制的性能问题，为了主从复制的速度和连接的稳定性，主从库最好在同一个局域网内。 …

十九. JVM模块

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？** 在介绍双亲委派模型之前先说下类加载器。对于任意一个类，都需要由加载它的类加载器和这个类本身统一确立在 JVM 中的唯一性，每一个类加载器，都有一个独立的类名称空间。类加载器就是根据指定全限定名称将 class 文件加载到 JVM 内存，然后再转化为 class 对象。 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. **说一下类装载的执行过程？** 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； 可达性分析：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 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。