1.Java 集合框架是什么？说出一些集合框架的优点？

每种编程语言中都有集合，最初的 Java 版本包含几种集合类：Vector、Stack、HashTable 和 Array。随着集合的广泛使用，Java1.2 提出了囊括所有集合接口、实现和算法的集合框架。在保证线程安全的情况下使用泛型和并发集合类，Java 已经经历了很久。它还包括在 Java 并发包中，阻塞接口以及它们的实现。集合框架的部分优点如下：

（1）使用核心集合类降低开发成本，而非实现我们自己的集合类。

（2）随着使用经过严格测试的集合框架类，代码质量会得到提高。

（3）通过使用 JDK 附带的集合类，可以降低代码维护成本。

（4）复用性和可操作性。

2.集合框架中的泛型有什么优点？

Java1.5 引入了泛型，所有的集合接口和实现都大量地使用它。泛型允许我们为集合提供一个可以容纳的对象类型，因此，如果你添加其它类型的任何元素，它会在编译时报错。这避免了在运行时出现 ClassCastException，因为你将会在编译时得到报错信息。泛型也使得代码整洁，我们不需要使用显式转换和 instanceOf 操作符。它也给运行时带来好处，因为不会产生类型检查的字节码指令，如果不明白可以学习动力 节点发布的 java 基础视频。

3.Java 集合框架的基础接口有哪些？

Collection 为集合层级的根接口。一个集合代表一组对象，这些对象即为它的元素。Java 平台不提供这个接口任何直接的实现。

Set 是一个不能包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。

List 是一个有序集合，可以包含重复元素。你可以通过它的索引来访问任何元素。List 更像长度动态变换的数组。

Map 是一个将 key 映射到 value 的对象.一个 Map 不能包含重复的 key：每个 key 最多只能映射一个 value。

一些其它的接口有 Queue、Dequeue、SortedSet、SortedMap 和 ListIterator。

4.为何 Collection 不从 Cloneable 和 Serializable 接口继承？

Collection 接口指定一组对象，对象即为它的元素。如何维护这些元素由 Collection 的具体实现决定。例如，一些如 List 的 Collection 实现允许重复的元素，而其它的如 Set 就不允许。很多 Collection 实现有一个公有的 clone 方法。然而，把它放到集合的所有实现中也是没有意义的。这是因为 Collection 是一个抽象表现。重要的是实现。

当与具体实现打交道的时候，克隆或序列化的语义和含义才发挥作用。所以，具体实现应该决定如何对它进行克隆或序列化，或它是否可以被克隆或序列化。

在所有的实现中授权克隆和序列化，最终导致更少的灵活性和更多的限制。特定的实现应该决定它是否可以被克隆和序列化。

5.为何 Map 接口不继承 Collection 接口？

尽管 Map 接口和它的实现也是集合框架的一部分，但 Map 不是集合，集合也不是 Map。因此，Map 继承 Collection 毫无意义，反之亦然。

如果 Map 继承 Collection 接口，那么元素去哪儿？Map 包含 key-value 对，它提供抽取 key 或 value 列表集合的方法，但是它不适合“一组对象”规范。

6.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。

7.Enumeration 和 Iterator 接口的区别？

Enumeration 的速度是 Iterator 的两倍，也使用更少的内存。Enumeration 是非常基础的，也满足了基础的需要。但是，与 Enumeration 相比，Iterator 更加安全，因为当一个集合正在被遍历的时候，它会阻止其它线程去修改集合。

迭代器取代了 Java 集合框架中的 Enumeration。迭代器允许调用者从集合中移除元素，而 Enumeration 不能做到。为了使它的功能更加清晰，迭代器方法名已经经过改善。

8.为何没有像 Iterator.add()这样的方法，向集合中添加元素？

语义不明，已知的是，Iterator 的协议不能确保迭代的次序。然而要注意，ListIterator 没有提供一个 add 操作，它要确保迭代的顺序。

9.为何迭代器没有一个方法可以直接获取下一个元素，而不需要移动游标？

它可以在当前 Iterator 的顶层实现，但是它用得很少，如果将它加到接口中，每个继承都要去实现它，这没有意义。

10.Iterater 和 ListIterator 之间有什么区别？

（1）我们可以使用 Iterator 来遍历 Set 和 List 集合，而 ListIterator 只能遍历 List。

（2）Iterator 只可以向前遍历，而 LIstIterator 可以双向遍历。

（3）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。

11.遍历一个 List 有哪些不同的方式？

```
1   List<String> strList = new ArrayList<>();
2   //使用 for-each 循环
3   for(String obj : strList){
4       System.out.println(obj);
5   }
6   //using iterator
7   Iterator<String> it = strList.iterator();
8   while(it.hasNext()){
9       String obj = it.next();
10      System.out.println(obj);
```

```
11  }
```

使用迭代器更加线程安全，因为它可以确保，在当前遍历的集合元素被更改的时候，它会抛出 ConcurrentModificationException。

12.通过迭代器 fail-fast 属性，你明白了什么？

每次我们尝试获取下一个元素的时候，Iterator fail-fast 属性检查当前集合结构里的任何改动。如果发现任何改动，它抛出 ConcurrentModificationException。Collection 中所有 Iterator 的实现都是按 fail-fast 来设计的（ConcurrentHashMap 和 CopyOnWriteArrayList 这类并发集合类除外）。

13.fail-fast 与 fail-safe 有什么区别？

Iterator 的 fail-fast 属性与当前的集合共同起作用，因此它不会受到集合中任何改动的影响。Java.util 包中的所有集合类都被设计为 fail-fast 的，而 java.util.concurrent 中的集合类都为 fail-safe 的。Fail-fast 迭代器抛出 ConcurrentModificationException，而 fail-safe 迭代器从不抛出 ConcurrentModificationException。

14.在迭代一个集合的时候，如何避免 ConcurrentModificationException？

在遍历一个集合的时候，我们可以使用并发集合类来避免 ConcurrentModificationException，比如使用 CopyOnWriteArrayList，而不是 ArrayList。

15.为何 Iterator 接口没有具体的实现？

Iterator 接口定义了遍历集合的方法，但它的实现则是集合实现类的责任。每个能够返回用于遍历的 Iterator 的集合类都有它自己的 Iterator 实现内部类。

这就允许集合类去选择迭代器是 fail-fast 还是 fail-safe 的。比如，ArrayList 迭代器是 fail-fast 的，而 CopyOnWriteArrayList 迭代器是 fail-safe 的。

16.UnsupportedOperationException 是什么？

UnsupportedOperationException 是用于表明操作不支持的异常。在 JDK 类中已被大量运用，在集合框架 java.util.Collections.UnmodifiableCollection 将会在所有 add 和 remove 操作中抛出这个异常。

17.在 Java 中，HashMap 是如何工作的？

HashMap 在 Map.Entry 静态内部类实现中存储 key-value 对。HashMap 使用哈希算法，在 put 和 get 方法中，它使用 hashCode()和 equals()方法。当我们通过传递 key-value 对调用 put 方法的时候，HashMap 使用 Key hashCode()和哈希算法来找出存储 key-value 对的索引。Entry 存储在 LinkedList 中，所以如果存在 entry，它使用 equals()方法来检查传递的 key 是否已经存在，如果存在，它会覆盖 value，如果不存在，它会创建一个新的 entry 然后保存。当我们通过传递 key 调用 get 方法时，它再次使用 hashCode()来找到数组中的索引，然后使用 equals()方法找出正确的 Entry，然后返回它的值。下面的图片解释了详细内容。

其它关于 HashMap 比较重要的问题是容量、负荷系数和阀值调整。HashMap 默认的初始容量是 32，负荷系数是 0.75。阀值是为负荷系数乘以容量，无论何时我们尝试添加一个 entry，如果 map 的大小比阀值大的时候，HashMap 会对 map 的内容进行重新哈希，且使用更大的容量。容量总是 2 的幂，所以如果你知道你需要存储大量的 key-value 对，比如缓存从数据库里面拉取的数据，使用正确的容量和负荷系数对 HashMap 进行初始化是个不错的做法。

18.hashCode()和 equals()方法有何重要性？

HashMap 使用 Key 对象的 hashCode()和 equals()方法去决定 key-value 对的索引。当我们试着从 HashMap 中获取值的时候，这些方法也会被用到。如果这些方法没有被正确地实现，在这种情况下，两个不同 Key 也许会产生相同的 hashCode()和 equals()输出，HashMap 将会认为它们是相同的，然后覆盖它们，而非把它们存储到不同的地方。同样的，所有不允许存储重复数据的集合类都使用 hashCode()和 equals()去查找重复，所以正确实现它们非常重要。equals()和 hashCode()的实现应该遵循以下规则：

（1）如果 o1.equals(o2)，那么 o1.hashCode() == o2.hashCode()总是为 true 的。

（2）如果 o1.hashCode() == o2.hashCode()，并不意味着 o1.equals(o2)会为 true。

19.我们能否使用任何类作为 Map 的 key？

我们可以使用任何类作为 Map 的 key，然而在使用它们之前，需要考虑以下几点：

（1）如果类重写了 equals()方法，它也应该重写 hashCode()方法。

（2）类的所有实例需要遵循与 equals()和 hashCode()相关的规则。请参考之前提到的这些规则。

（3）如果一个类没有使用 equals()，你不应该在 hashCode()中使用它。

（4）用户自定义 key 类的最佳实践是使之为不可变的，这样，hashCode()值可以被缓存起来，拥有更好的性能。不可变的类也可以确保 hashCode()和 equals()在未来不会改变，这样就会解决与可变相关的问题了。

比如，我有一个类 MyKey，在 HashMap 中使用它。

```
//传递给 MyKey 的 name 参数被用于 equals()和 hashCode()中
MyKey key = new MyKey('Pankaj'); //assume hashCode=1234
myHashMap.put(key, 'Value');
// 以下的代码会改变 key 的 hashCode()和 equals()值
key.setName('Amit'); //assume new hashCode=7890
//下面会返回 null，因为 HashMap 会尝试查找存储同样索引的 key，而 key 已
被改变了，匹配失败，返回 null
myHashMap.get(new MyKey('Pankaj'));
```

那就是为何 String 和 Integer 被作为 HashMap 的 key 大量使用。

20.Map 接口提供了哪些不同的集合视图？

Map 接口提供三个集合视图：

（1）Set keyset()：返回 map 中包含的所有 key 的一个 Set 视图。集合是受 map 支持的，map 的变化会在集合中反映出来，反之亦然。当一个迭代器正在遍历一个集合时，若 map 被修改了（除迭代器自身的移除操作以外），迭代器的结果会变为未定义。集合支持通过 Iterator 的 Remove、Set.remove、removeAll、retainAll 和 clear 操作进行元素移除，从 map 中移除对应的映射。它不支持 add 和 addAll 操作。

（2）Collection values()：返回一个 map 中包含的所有 value 的一个 Collection 视图。这个 collection 受 map 支持的，map 的变化会在 collection 中反映出来，反之亦然。当一个迭代器正在遍历一个 collection 时，若 map 被修改了(除迭代器自身的移除操作以外），迭代器的结果会变为未定义。集合支持通过 Iterator 的 Remove、Set.remove、removeAll、retainAll 和 clear 操作进行元素移除，从 map 中移除对应的映射。它不支持 add 和 addAll 操作。

（3）Set<Map.Entry<K,V>> entrySet()：返回一个 map 钟包含的所有映射的一个集合视图。这个集合受 map 支持的，map 的变化会在 collection 中反映出来，反之亦然。当一个迭代器正在遍历一个集合时，若 map 被修改了（除迭代器自身的移除操作，以及对迭代器返回的 entry 进行 setValue 外），迭代器的结果会变为未定义。集合支持通过 Iterator 的 Remove、Set.remove、removeAll、retainAll 和 clear 操作进行元素移除，从 map 中移除对应的映射。它不支持 add 和 addAll 操作。

21.HashMap 和 HashTable 有何不同？

（1）HashMap 允许 key 和 value 为 null，而 HashTable 不允许。

（2）HashTable 是同步的，而 HashMap 不是。所以 HashMap 适合单线程环境，HashTable 适合多线程环境。

（3）在 Java1.4 中引入了 LinkedHashMap，HashMap 的一个子类，假如你想要遍历顺序，你很容易从 HashMap 转向 LinkedHashMap，但是 HashTable 不是这样的，它的顺序是不可预知的。

（4）HashMap 提供对 key 的 Set 进行遍历，因此它是 fail-fast 的，但 HashTable 提供对 key 的 Enumeration 进行遍历，它不支持 fail-fast。

（5）HashTable 被认为是个遗留的类，如果你寻求在迭代的时候修改 Map，你应该使用 CocurrentHashMap。

22.如何决定选用 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.ArrayList 和 Vector 有何异同点？

ArrayList 和 Vector 在很多时候都很类似。

（1）两者都是基于索引的，内部由一个数组支持。

（2）两者维护插入的顺序，我们可以根据插入顺序来获取元素。

（3）ArrayList 和 Vector 的迭代器实现都是 fail-fast 的。

（4）ArrayList 和 Vector 两者允许 null 值，也可以使用索引值对元素进行随机访问。

以下是 ArrayList 和 Vector 的不同点。

（1）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。

（2）ArrayList 比 Vector 快，它因为有同步，不会过载。

（3）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。

24.Array 和 ArrayList 有何区别？什么时候更适合用 Array？

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。尽管 ArrayList 明显是更好的选择，但也有些时候 Array 比较好用。

（1）如果列表的大小已经指定，大部分情况下是存储和遍历它们。

（2）对于遍历基本数据类型，尽管 Collections 使用自动装箱来减轻编码任务，在指定大小的基本类型的列表上工作也会变得很慢。

（3）如果你要使用多维数组，使用[][]比 List<List<>>更容易。

25.ArrayList 和 LinkedList 有何区别？

ArrayList 和 LinkedList 两者都实现了 List 接口，但是它们之间有些不同。

（1）ArrayList 是由 Array 所支持的基于一个索引的数据结构，所以它提供对元素的随机访问，复杂度为 O(1)，但 LinkedList 存储一系列的节点数据，每个节点都与前一个和下一个节点相连接。所以，尽管有使用索引获取元素的方法，内部实现是从起始点开始遍历，遍历到索引的节点然后返回元素，时间复杂度为 O(n)，比 ArrayList 要慢。

（2）与 ArrayList 相比，在 LinkedList 中插入、添加和删除一个元素会更快，因为在一个元素被插入到中间的时候，不会涉及改变数组的大小，或更新索引。

（3）LinkedList 比 ArrayList 消耗更多的内存，因为 LinkedList 中的每个节点存储了前后节点的引用。

26.哪些集合类提供对元素的随机访问？

ArrayList、HashMap、TreeMap 和 HashTable 类提供对元素的随机访问。

27.EnumSet 是什么？

java.util.EnumSet 是使用枚举类型的集合实现。当集合创建时，枚举集合中的所有元素必须来自单个指定的枚举类型，可以是显示的或隐示的。EnumSet 是不同步的，不允许值为 null 的元素。它也提供了一些有用的方法，比如 copyOf(Collection c)、of(E first,E...rest)和 complementOf(EnumSet s)。

28.哪些集合类是线程安全的？

Vector、HashTable、Properties 和 Stack 是同步类，所以它们是线程安全的，可以在多线程环境下使用。Java1.5 并发 API 包括一些集合类，允许迭代时修改，因为它们都工作在集合的克隆上，所以它们在多线程环境中是安全的。

29.并发集合类是什么？

Java1.5 并发包（java.util.concurrent）包含线程安全集合类，允许在迭代时修改集合。迭代器被设计为 fail-fast 的，会抛出 ConcurrentModificationException。一部分类为：CopyOnWriteArrayList、 ConcurrentHashMap、CopyOnWriteArraySet。

30.BlockingQueue 是什么？

Java.util.concurrent.BlockingQueue 是一个队列，在进行检索或移除一个元素的时候，它会等待队列变为非空；当在添加一个元素时，它会等待队列中的可用空间。BlockingQueue 接口是 Java 集合框架的一部分，主要用于实现生产者-消费者模式。我们不需要担心等待生产者有可用的空间，或消费者有可用的对象，因为它都在 BlockingQueue 的实现类中被处理了。Java 提供了集中 BlockingQueue 的实现 ,比如 ArrayBlockingQueue、LinkedBlockingQueue、PriorityBlockingQueue,、SynchronousQueue 等。

31.队列和栈是什么，列出它们的区别？

栈和队列两者都被用来预存储数据。java.util.Queue 是一个接口，它的实现类在 Java 并发包中。队列允许先进先出（FIFO）检索元素，但并非总是这样。Deque 接口允许从两端检索元素。

栈与队列很相似，但它允许对元素进行后进先出（LIFO）进行检索。

Stack 是一个扩展自 Vector 的类，而 Queue 是一个接口。

32.Collections 类是什么？

Java.util.Collections 是一个工具类仅包含静态方法，它们操作或返回集合。它包含操作集合的多态算法，返回一个由指定集合支持的新集合和其它一些内容。这个类包含集合框架算法的方法，比如折半搜索、排序、混编和逆序等。

33.Comparable 和 Comparator 接口是什么？

如果我们想使用 Array 或 Collection 的排序方法时 ,需要在自定义类里实现 Java 提供 Comparable 接口。Comparable 接口有 compareTo(T OBJ)方法，它被排序方法所使用。我们应该重写这个方法，如果“this”对象比传递的对象参数更小、相等或更大时，它返回一个负整数、0 或正整数。但是，在大多数实际情况下，我们想根据不同参数进行排序。比如，作为一个 CEO，我想对雇员基于薪资进行排序，一个 HR 想基于年龄对他们进行排序。这就是我们需要使用 Comparator 接口的情景，因为 Comparable.compareTo(Object o)方法实现只能基于一个字段进行排序，我们不能根据对象排序的需要选择字段。Comparator 接口的 compare(Object o1, Object o2)方法的实现需要传递两个对象参数，若第一个参数比第二个小，返回负整数；若第一个等于第二个，返回 0；若第一个比第二个大，返回正整数。

34.Comparable 和 Comparator 接口有何区别？

Comparable 和 Comparator 接口被用来对对象集合或者数组进行排序。Comparable 接口被用来提供对象的自然排序，我们可以使用它来提供基于单个逻辑的排序。

Comparator 接口被用来提供不同的排序算法 ,我们可以选择需要使用的 Comparator 来对给定的对象集合进行排序。

35.我们如何对一组对象进行排序？

如果我们需要对一个对象数组进行排序，我们可以使用 Arrays.sort()方法。如果我们需要排序一个对象列表，我们可以使用 Collection.sort()方法。两个类都有用于自然排序(使用 Comparable)或基于标准的排序(使用 Comparator) 的重载方法 sort()。Collections 内部使用数组排序方法，所有它们两者都有相同的性能，只是 Collections 需要花时间将列表转换为数组。

36.当一个集合被作为参数传递给一个函数时，如何才可以确保函数不能修改它？

在作为参数传递之前，我们可以使用 Collections.unmodifiableCollection(Collection c)方法创建一个只读集合，这将确保改变集合的任何操作都会抛出 UnsupportedOperationException。

37.我们如何从给定集合那里创建一个 synchronized 的集合？

我们可以使用 Collections.synchronizedCollection(Collection c)根据指定集合来获取一个 synchronized(线程安全的) 集合。

38.集合框架里实现的通用算法有哪些？

Java 集合框架提供常用的算法实现，比如排序和搜索。Collections 类包含这些方法实现。大部分算法是操作 List 的，但一部分对所有类型的集合都是可用的。部分算法有排序、搜索、混编、最大最小值。

39.大写的 O 是什么？举几个例子？

大写的 O 描述的是，就数据结构中的一系列元素而言，一个算法的性能。Collection 类就是实际的数据结构，我们通常基于时间、内存和性能，使用大写的 O 来选择集合实现。比如：例子 1：ArrayList 的 get(index i)是一个常量时间操作，它不依赖 list 中元素的数量。所以它的性能是 O(1)。例子 2：一个对于数组或列表的线性搜索的性能是 O(n)，因为我们需要遍历所有的元素来查找需要的元素。

40.与 Java 集合框架相关的有哪些最好的实践？

（1）根据需要选择正确的集合类型。比如，如果指定了大小，我们会选用 Array 而非 ArrayList。如果我们想根据插入顺序遍历一个 Map，我们需要使用 TreeMap。如果我们不想重复，我们应该使用 Set。

（2）一些集合类允许指定初始容量，所以如果我们能够估计到存储元素的数量，我们可以使用它，就避免了重新哈希或大小调整。

（3）基于接口编程，而非基于实现编程，它允许我们后来轻易地改变实现。

（4）总是使用类型安全的泛型，避免在运行时出现 ClassCastException。

（5）使用 JDK 提供的不可变类作为 Map 的 key，可以避免自己实现 hashCode()和 equals()。

（6）尽可能使用 Collections 工具类，或者获取只读、同步或空的集合，而非编写自己的实现。它将会提供代码重用性，它有着更好的稳定性和可维护性。