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。

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。

Enumeration(列举)
public interface Enumeration<E>实现 Enumeration 接口的对象，它生成一系列元素，一次生成一个。连续调用 nextElement 方法将返回一系列的连续元素。

例如，要输出 Vector<E> v 的所有元素，可使用以下方法:
```
for (Enumeration<E> e = v.elements();
e.hasMoreElements();)
    System.out.println(e.nextElement());
```

这些方法主要通过向量的元素、哈希表的键以及哈希表中的值进行枚举。枚举也用于将输入流指定到 SequenceInputStream 中。

```
public static ArrayList getSingle(ArrayList list) {
 ArrayList newList = new ArrayList<>();     //1,创建新集合
 Iterator it = list.iterator();      //2,根据传入的集合(老集合)获取迭代器

 while(it.hasNext()) {        //3,遍历老集合
  Object obj = it.next();      //记录住每一个元素
  if(!newList.contains(obj)) {      //如果新集合中不包含老集合中的元素
   newList.add(obj);       //将该元素添加
  }
 }
 return newList;
}
```

迭代器：可以用于集合的遍历

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？

List<String> strList = new ArrayList<>(); // 使用 for-each 循环 for(String obj : strList){ System.out.println(obj); } //using iterator Iterator<String> it = strList.iterator(); while(it.hasNext()){ String obj = it.next(); System.out.println(obj); }

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。

hasCode()找到对应的索引，equals()找到对应的值。

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

```
public class Demo2_Iterator {
 public static void main(String[] args) {
  Map<String, Integer> map = new HashMap<>();map.put("张三", 23);map.put("李四", 24);map.put("王五", 25);map.put("赵六", 26);//  Integer i =
map.get("张三");     //根据键获取值/  System.out.println(i);//获取所有的键/*Set<String> keySet = map.keySet();  //获取所有键的集合Iterator<String> it
= keySet.iterator(); //获取迭代器while(it.hasNext()) {      //判断集合中是否有元素String key = it.next();      //获取每一个键Integer value = map.get
(key);   //根据键获取值System.out.println(key + "=" + value);
 }*///使用增强for循环遍历 for(String key : map.keySet()) {   //map.keySet()是所有键的集合 System.out.println(key + "=" + map.get(key));}}}
```

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 中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。

50 道 Java 线程面试题

下面是 Java 线程相关的热门面试题，你可以用它来好好准备面试。

1) 什么是线程？

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要 100 毫秒，那么用十个线程完成改任务只需 10 毫秒。Java 在语言层面对多线程提供了卓越的支持，它也是一个很好的卖点。

2) 线程和进程有什么区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。

3) 如何在 Java 中实现线程？

在语言层面有两种方式。java.lang.Thread 类的实例就是一个线程但是它需要调用 java.lang.Runnable 接口来执行，由于线程类本身就是调用的 Runnable 接口所以你可以继承 java.lang.Thread 类或者直接调用 Runnable 接口来重写 run()方法实现线程。

4) 用 Runnable 还是 Thread？

这个问题是上题的后续，大家都知道我们可以通过继承 Thread 类或者调用 Runnable 接口来实现线程，问题是，那个方法更好呢？什么情况下使用它？这个问题很容易回答，如果你知道 Java 不支持类的多重继承，但允许你调用多个接口。所以如果你要继承其他类，当然是调用 Runnable 接口好了。

6) Thread 类中的 start() 和 run() 方法有什么区别？

这个问题经常被问到，但还是能从此区分出面试者对 Java 线程模型的理解程度。start()方法被用来启动新创建的线程，而且 start()内部调用了 run()方法，这和直接调用 run()方法的效果不一样。当你调用 run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

7) Java 中 Runnable 和 Callable 有什么不同？

Runnable 和 Callable 都代表那些要在不同的线程中执行的任务。Runnable 从 JDK1.0 开始就有了，Callable 是在 JDK1.5 增加的。它们的主要区别是 Callable 的 call() 方法可以返回值和抛出异常，而 Runnable 的 run()方法没有这些功能。Callable 可以返回装载有计算结果的 Future 对象。我的博客有更详细的说明。（http://java67.blogspot.com/2013/01/difference-between-callable-and-runnable-java.html）

8) Java 中 CyclicBarrier 和 CountDownLatch 有什么不同？

CyclicBarrier 和 CountDownLatch 都可以用来让一组线程等待其它线程。与 CyclicBarrier 不同的是，CountdownLatch 不能重新使用。点此查看更多信息和示例代码。（http://javarevisited.blogspot.com/2012/07/cyclicbarrier-example-java-5-concurrency-tutorial.html）

9) Java 内存模型是什么？

Java 内存模型规定和指引 Java 程序在不同的内存架构、CPU 和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java 内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。这个关系定义了一些规则让程序员在并发编程时思路更清晰。比如，先行发生关系确保了：

线程内的代码能够按先后顺序执行，这被称为程序次序规则。

对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。

前一个对 volatile 的写操作在后一个 volatile 的读操作之前，也叫 volatile 变量规则。

一个线程内的任何操作必需在这个线程的 start()调用之后，也叫作线程启动规则。

一个线程的所有操作都会在线程终止之前，线程终止规则。

一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。

可传递性

我强烈建议大家阅读《Java 并发编程实践》第十六章来加深对 Java 内存模型的理解。

10) Java 中的 volatile 变量是什么？

volatile 是一个特殊的修饰符，只有成员变量才能使用它。在 Java 并发程序缺少同步类的情况下，多线程对成员变量的操作对其它线程是透明的。volatile 变量可以保证下一个读取操作会在前一个写操作之后发生，就是上一题的 volatile 变量规则。查看更多 volatile 的相关内容。（http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html）

11) 什么是线程安全？Vector 是一个线程安全类吗？

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况

下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector 是用同步方法来实现线程安全的，而和它相似的 ArrayList 不是线程安全的。

12) Java 中什么是竞态条件？ 举个例子说明。

竞态条件会导致程序在并发情况下出现一些 bugs。多线程对一些资源的竞争的时候就会产生竞态条件，如果首先要执行的程序竞争失败排到后面执行了，那么整个程序就会出现一些不确定的 bugs。这种 bugs 很难发现而且会重复出现，因为线程间的随机竞争。一个例子就是无序处理，详见答案。（http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html）

13) Java 中如何停止一个线程？

Java 提供了很丰富的 API 但没有为停止线程提供 API。JDK 1.0 本来有一些像 stop(), suspend() 和 resume()的控制方法但是由于潜在的死锁威胁因此在后续的 JDK 版本中他们被弃用了，之后 Java API 的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当 run() 或者 call() 方法执行完的时候线程会自动结束,如果要手动结束一个线程，你可以用 volatile 布尔变量来退出 run()方法的循环或者是取消任务来中断线程。点击这里查看示例代码。（http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html）

14) 一个线程运行时发生异常会怎样？

这是我在一次面试中遇到的一个很刁钻的 Java 面试题，简单的说，如果异常没有被捕获该线程将会停止执行。Thread.UncaughtExceptionHandler 是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候 JVM 会使用 Thread.getUncaughtExceptionHandler()来查询线程的 UncaughtExceptionHandler 并将线程和异常作为参数传递给 handler 的 uncaughtException()方法进行处理。

15） 如何在两个线程间共享数据？

你可以通过共享对象来实现这个目的，或者是使用像阻塞队列这样并发的数据结构。这篇教程《Java 线程间通信》（http://javarevisited.blogspot.sg/2013/12/inter-thread-communication-in-java-wait-notify-example.html）(涉及到在两个线程间共享对象)用 wait 和 notify 方法实现了生产者消费者模型。

16) Java 中 notify 和 notifyAll 有什么区别？

这又是一个刁钻的问题，因为多线程可以等待单监控锁，Java API 的设计人员提供了一些方法当等待条件改变的时候通知它们，但是这些方法没有完全实现。notify()方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而 notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。

17) 为什么 wait, notify 和 notifyAll 这些方法不在 thread 类里面？

这是个设计相关的问题，它考察的是面试者对现有系统和一些普遍存在但看起来不合理的事物的看法。回答这些问题的时候，你要说明为什么把这些方法放在 Object 类里是有意义的，还有不把它放在 Thread 类里的原因。一个很明显的原因是 JAVA 提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调用对象中的 wait()方法就有意义了。如果 wait()方法定义在 Thread 类中，线程正在等待的是哪个锁就不明显了。简单的说，由于 wait，notify 和 notifyAll 都是锁级别的操作，所以把他们定义在 Object 类中因为锁属于对象。

18) 什么是 ThreadLocal 变量？

ThreadLocal 是 Java 里一种特殊的变量。每个线程都有一个 ThreadLocal 就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用 ThreadLocal 让 SimpleDateFormat 变成线程安全的，因为那个类创建代价高昂且每次调用都需要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。线程局部变量的另一个不错的例子是 ThreadLocalRandom 类，它在多线程环境中减少了创建代价高昂的 Random 对象的个数。

19) 什么是 FutureTask？

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

20) Java 中 interrupted 和 isInterruptedd 方法的区别？

interrupted() 和 isInterrupted()的主要区别是前者会将中断状态清除而后者不会。Java 多线程的中断机制是用内部标识来实现的，调用 Thread.interrupt()来中断一个线程就会设置中断标识为 true。当中断线程调用静态方法 Thread.interrupted()来检查中断状态时，中断状态会被清零。而非静态方法 isInterrupted()用来查询其它线程的中断状态且不会改变中断状态标识。简单的说就是任何抛出 InterruptedException 异常的方法都会将中断状态清零。无论如何，一个线程的中断状态有有可能被其它线程调用中断来改变。

21) 为什么 wait 和 notify 方法要在同步块中调用？

主要是因为 Java API 强制要求这样做，如果你不这么做，你的代码会抛出 IllegalMonitorStateException 异常。还有一个原因是为了避免 wait 和 notify 之间产生竞态条件。

22) 为什么你应该在循环中检查等待条件?

处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就

会在没有满足结束条件的情况下退出。因此，当一个等待线程醒来时，不能认为它原来的等待状态仍然是有效的，在 notify()方法调用之后和等待线程醒来之前这段时间它可能会改变。这就是在循环中使用 wait()方法效果更好的原因，你可以在 Eclipse 中创建模板调用 wait 和 notify 试一试。如果你想了解更多关于这个问题的内容，我推荐你阅读《Effective Java》这本书中的线程和同步章节。

23) Java 中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在 Java1.5 之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5 介绍了并发集合像 ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。

24） Java 中堆和栈有什么不同？

为什么把这个问题归类在多线程和并发面试题里？因为栈是一块和线程紧密相关的内存区域。每个线程都有自己的栈内存，用于存储本地变量，方法参数和栈调用，一个线程中存储的变量对其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，如果多个线程使用该变量就可能引发问题，这时 volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

易变性变量

25） 什么是线程池？ 为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从 JDK1.5 开始，Java API 提供了 Executor 框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

26） 如何写代码来解决生产者消费者问题？

在现实中你解决的许多线程问题都属于生产者消费者模型，就是一个线程生产任务供其它线程进行消费，你必须知道怎么进行线程间通信来解决这个问题。比较低级的办法是用 wait 和 notify 来解决这个问题，比较赞的办法是用 Semaphore 或者 BlockingQueue 来实现生产者消费者模型。

27） 如何避免死锁？

Java 多线程中的死锁

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因为死锁会让你的程序挂起无法完成任务，死锁的发生必须满足以下四个条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源，在末使用完之前，不能强行剥夺。

循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

避免死锁最简单的方法就是阻止循环等待条件，将系统中所有的资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或降序）做操作来避免死锁。

28) Java 中活锁和死锁有什么区别？

这是上题的扩展，活锁和死锁类似，不同之处在于处于活锁的线程或进程的状态是不断改变的，活锁可以认为是一种特殊的饥饿。一个现实的活锁例子是两个人在狭小的走廊碰到，两个人都试着避让对方好让彼此通过，但是因为避让的方向都一样导致最后谁都不能通过走廊。简单的说就是，活锁和死锁的主要区别是前者进程的状态可以改变但是却不能继续执行。

29） 怎么检测一个线程是否拥有锁？

我一直不知道我们竟然可以检测一个线程是否拥有锁，直到我参加了一次电话面试。在 java.lang.Thread 中有一个方法叫 holdsLock()，它返回 true 如果当且仅当当前线程拥有某个具体对象的锁。

30) 你如何在 Java 中获取线程堆栈？

对于不同的操作系统，有多种方法来获得 Java 进程的线程堆栈。当你获取线程堆栈时，JVM 会把所有线程的状态存到日志文件或者输出到控制台。在 Windows 你可以使用 Ctrl + Break 组合键来获取线程堆栈，Linux 下用 kill -3 命令。你也可以用 jstack 这个工具来获取，它对线程 id 进行操作，你可以用 jps 这个工具找到 id。

31) JVM 中哪个参数是用来控制线程的栈堆栈小的

这个问题很简单， -Xss 参数用来控制线程的堆栈大小。你可以查看 JVM 配置列表来了解这个参数的更多信息。

32） Java 中 synchronized 和 ReentrantLock 有什么不同？

Java 在过去很长一段时间只能通过 synchronized 关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java 5 通过 Lock 接口提供了更复杂的控制来解决这些问题。 ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。

33） 有三个线程 T1，T2，T3，怎么确保它们按顺序执行？

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的 join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3 调用 T2，T2 调用 T1)，这样 T1 就会先完成而 T3 最后完成。

34) Thread 类中的 yield 方法有什么作用？

Yield 方法可以暂停当前正在执行的线程对象，让其它有相同优先级的线程执行。它是一个静态方法而且只保证当前线程放弃 CPU 占用而不能保证使其它线程一定能占用 CPU，执行 yield()的线程有可能在进入到暂停状态后马上又被执行。

35） Java 中 ConcurrentHashMap 的并发度是什么？

ConcurrentHashMap 把实际 map 划分成若干部分来实现它的可扩展性和线程安全。这种划分是使用并发度获得的，它是 ConcurrentHashMap 类构造函数的一个可选参数，默认值为 16，这样在多线程情况下就能避免争用。欲了解更多并发度和内部大小调整请阅读我的文章 How ConcurrentHashMap works in Java。

36） Java 中 Semaphore 是什么？

Java 中的 Semaphore 是一种新的同步类，它是一个计数信号。从概念上讲，信号量维护了一个许可集合。如有必要，在许可可用前会阻塞每一个 acquire()，然后再获取该许可。每个 release()添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象，Semaphore 只对可用许可的号码进行计数，并采取相应的行动。信号量常常用于多线程的代码中，比如数据库连接池。

37）如果你提交任务时，线程池队列已满。会时发会生什么？

这个问题问得很狡猾，许多程序员会认为该任务会阻塞直到线程池队列有空位。事实上如果一个任务不能被调度执行那么 ThreadPoolExecutor's submit()方法将会抛出一个 RejectedExecutionException 异常。

38) Java 线程池中 submit() 和 execute()方法有什么区别？

两个方法都可以向线程池提交任务，execute()方法的返回类型是 void，它定义在 Executor 接口中，而 submit()方法可以返回持有计算结果的 Future 对象，它定义在 ExecutorService 接口中，它扩展了 Executor 接口，其它线程池类像 ThreadPoolExecutor 和 ScheduledThreadPoolExecutor 都有这些方法。

39) 什么是阻塞式方法？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket 的 accept()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。

40) Swing 是线程安全的吗？ 为什么？

你可以很肯定的给出回答，Swing 不是线程安全的，但是你应该解释这么回答的原因即便面试官没有问你为什么。当我们说 swing 不是线程安全的常常提到它的组件，这些组件不能在多线程中进行修改，所有对 GUI 组件的更新都要在 AWT 线程中完成，而 Swing 提供了同步和异步两种回调方法来进行更新。

41） Java 中 invokeAndWait 和 invokeLater 有什么区别？

这两个方法是 Swing API 提供给 Java 开发者用来从当前线程而不是事件派发线程更新 GUI 组件用的。InvokeAndWait()同步更新 GUI 组件，比如一个进度条，一旦进度更新了，进度条也要做出相应改变。如果进度被多个线程跟踪，那么就调用 invokeAndWait()方法请求事件派发线程对组件进行相应更新。而 invokeLater()方法是异步调用更新组件的。

42) Swing API 中那些方法是线程安全的？

这个问题又提到了 swing 和线程安全，虽然组件不是线程安全的但是有一些方法是可以被多线程安全调用的，比如 repaint(), revalidate()。 JTextComponent 的 setText()方法和 JTextArea 的 insert() 和 append() 方法也是线程安全的。

43) 如何在 Java 中创建 Immutable 对象？

这个问题看起来和多线程没什么关系， 但不变性有助于简化已经很复杂的并发程序。Immutable 对象可以在没有同步的情况下共享，降低了对该对象进行并发访问时的同步化开销。可是 Java 没有@Immutable 这个注解符，要创建不可变类，要实现下面几个步骤：通过构造方法初始化所有成员、对变量不要提供 setter 方法、将所有的成员声明为私有的，这样就不允许直接访问这些成员、在 getter 方法中，不要直接返回对象本身，而是克隆对象，并返回对象的拷贝。我的文章 how to make an object Immutable in Java 有详细的教程，看完你可以充满自信。

44） Java 中的 ReadWriteLock 是什么？

一般而言，读写锁是用来提升并发程序性能的锁分离技术的成果。Java 中的 ReadWriteLock 是 Java 5 中新增的一个接口，一个 ReadWriteLock 维护一对关联的锁，一个用于只读操作一个用于写。在没有写线程的情况下一个读锁可能会同时被多个读线程持有。写锁是独占的，你可以使用 JDK 中的 ReentrantReadWriteLock 来实现这个规则，它最多支持 65535 个写锁和 65535 个读锁。

45) 多线程中的忙循环是什么?

忙循环就是程序员用循环让一个线程等待，不像传统方法 wait(), sleep() 或 yield() 它们都放弃了 CPU 控制，而忙循环不会放弃 CPU，它就是在运行一个空循环。这么做的目的是为了保留 CPU 缓存，在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样

会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。

46）volatile 变量和 atomic 变量有什么不同？

这是个有趣的问题。首先，volatile 变量和 atomic 变量看起来很像，但功能却不一样。Volatile 变量可以确保先行关系，即写操作会发生在后续的读操作之前，但它并不能保证原子性。例如用 volatile 修饰 count 变量那么 count++ 操作就不是原子性的。而 AtomicInteger 类提供的 atomic 方法可以让这种操作具有原子性如 getAndIncrement()方法会原子性的进行增量操作把当前值加一，其它数据类型和引用变量也可以进行相似操作。

47) 如果同步块内的线程抛出异常会发生什么？

这个问题坑了很多 Java 程序员，若你能想到锁是否释放这条线索来回答还有点希望答对。无论你的同步块是正常还是异常退出的，里面的线程都会释放锁，所以对比锁接口我更喜欢同步块，因为它不用我花费精力去释放锁，该功能可以在 finally block 里释放锁实现。

48） 单例模式的双检锁是什么？

这个问题在 Java 面试中经常被问到，但是面试官对回答此问题的满意度仅为 50%。一半的人写不出双检锁还有一半的人说不出它的隐患和 Java1.5 是如何对它修正的。它其实是一个用来创建线程安全的单例的老方法，当单例实例第一次被创建时它试图用单个锁进行性能优化，但是由于太过于复杂在 JDK1.4 中它是失败的，我个人也不喜欢它。无论如何，即便你也不喜欢它但是还是要了解一下，因为它经常被问到。你可以查看 how double checked locking on Singleton works 这篇文章获得更多信息。

49） 如何在 Java 中创建线程安全的 Singleton？

这是上面那个问题的后续，如果你不喜欢双检锁而面试官问了创建 Singleton 类的替代方法，你可以利用 JVM 的类加载和静态变量初始化特征来创建 Singleton 实例，或者是利用枚举类型来创建 Singleton，我很喜欢用这种方法。

50) 写出 3 条你遵循的多线程最佳实践

这种问题我最喜欢了，我相信你在写并发代码来提升性能的时候也会遵循某些最佳实践。以下三条最佳实践我觉得大多数 Java 程序员都应该遵循：

给你的线程起个有意义的名字。
这样可以方便找 bug 或追踪。OrderProcessor, QuoteProcessor or TradeProcessor 这种名字比 Thread-1. Thread-2 and Thread-3 好多了，给线程起一个和它要完成的任务相关的名字，所有的主要框架甚至 JDK 都遵循这个最佳实践。

避免锁定和缩小同步的范围
锁花费的代价高昂且上下文切换更耗费时间空间，试试最低限度的使用同步和锁，缩小临界区。因此相对于同步方法我更喜欢同步块，它给我拥有对锁的绝对控制权。

多用同步类少用 wait 和 notify
首先，CountDownLatch, Semaphore, CyclicBarrier 和 Exchanger 这些同步类简化了编码操作，而用 wait 和 notify 很难实现对复杂控制流的控制。其次，这些类是由最好的企业编写和维护在后续的 JDK 中它们还会不断优化和完善，使用这些更高等级的同步工具你的程序可以不费吹灰之力获得优化。

多用并发集合少用同步集合
这是另外一个容易遵循且受益巨大的最佳实践，并发集合比同步集合的可扩展性更好，所以在并发编程时使用并发集合效果更好。如果下一次你需要用到 map，你应该首先想到用 ConcurrentHashMap。我的文章 Java 并发集合有更详细的说明。

51) 如何强制启动一个线程？

这个问题就像是如何强制进行 Java 垃圾回收，目前还没有觉得方法，虽然你可以使用 System.gc()来进行垃圾回收，但是不保证能成功。在 Java 里面没有办法强制启动一个线程，它是被线程调度器控制着且 Java 没有公布相关的 API。

52) Java 中的 fork join 框架是什么？

fork join 框架是 JDK7 中出现的一款高效的工具，Java 开发人员可以通过它充分利用现代服务器上的多处理器。它是专门为了那些可以递归划分成许多子模块设计的，目的是将所有可用的处理能力用来提升程序的性能。fork join 框架一个巨大的优势是它使用了工作窃取算法，可以完成更多任务的工作线程可以从其它线程中窃取任务来执行。

53） Java 多线程中调用 wait() 和 sleep()方法有什么不同？

Java 程序中 wait 和 sleep 都会造成某种形式的暂停，它们可以满足不同的需要。wait()方法用于线程间通信，如果等待条件为真且其它线程被唤醒时它会释放锁，而 sleep()方法仅仅释放 CPU 资源或者让当前线程停止执行一段时间，但不会释放锁。

SpringMVC 工作原理？
1、用户发送请求至前端控制器 DispatcherServlet
2、DispatcherServlet 收到请求调用 HandlerMapping 处理器映射器。
3、处理器映射器找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给 DispatcherServlet。
4、DispatcherServlet 调用 HandlerAdapter 处理器适配器
5、HandlerAdapter 经过适配调用具体的处理器(Controller，也叫后端控制器)。
6、Controller 执行完成返回 ModelAndView
7、HandlerAdapter 将 controller 执行结果 ModelAndView 返回给 DispatcherServlet
8、DispatcherServlet 将 ModelAndView 传给 ViewReslover 视图解析器
9、ViewReslover 解析后返回具体 View
10、DispatcherServlet 根据 View 进行渲染视图（即将模型数据填充至视图中）。
11、DispatcherServlet 响应用户

2.Struts2 的工作原理？
1 客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求
2 这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器，这个过滤器对于 Struts2 和其他框架的集成很有帮助，例如：SiteMesh Plugin）
3 接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请是否需要调用某个 Action
4 如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy
5 ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类
6 ActionProxy 创建一个 ActionInvocation 的实例。
7 ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。
8 一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。在这个过程中需要涉及到 ActionMapper
3.Hibernate 的工作原理?
1.通过 Configuration().configure();读取并解析 hibernate.cfg.xml 配置文件
2.由 hibernate.cfg.xml 中的<mapping resource="com/xx/User.hbm.xml"/>读取并解析映射信息
3.通过 config.buildSessionFactory();//创建 SessionFactory
4.sessionFactory.openSession();//打开 Sesssion
5.session.beginTransaction();//创建事务 Transation
6.persistent operate 持久化操作
7.session.getTransaction().commit();//提交事务
8.关闭 Session
9.关闭 SesstionFactory

4.mybatis 的工作原理？
MyBatis 应用程序根据 XML 配置文件创建 SqlSessionFactory，SqlSessionFactory 在根据配置，

配置来源于两个地方，一处是配置文件，一处是 Java 代码的注解，获取一个 SqlSession。SqlSession 包含了执行 sql 所需要的所有方法，可以通过 SqlSession 实例直接运行映射的 sql 语句，完成对数据的增删改查和事务提交等，用完之后关闭 SqlSession。

一、struts2 工作流程

1、Struts 2 框架本身大致可以分为 3 个部分：核心控制器 FilterDispatcher、业务控制器 Action 和用户实现的企业业务逻辑组件。

1）核心控制器 FilterDispatcher 是 Struts 2 框架的基础，包含了框架内部的控制流程和处理机制。

2）业务控制器 Action 和业务逻辑组件是需要用户来自己实现的。用户在开发 Action 和业务逻辑组件的同时，还需要编写相关的配置文件，供核心控制器 FilterDispatcher 来使用。

Struts 2 的工作流程相对于 Struts 1 要简单，与 WebWork 框架基本相同，所以说 Struts 2 是 WebWork 的升级版本。

2、基本简要流程如下：

1）客户端初始化一个指向 Servlet 容器的请求；

2）这个请求经过一系列的过滤器（Filter）

（这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器，这个过滤器对于 Struts2 和其他框架的集成很有帮助，例如：SiteMesh Plugin）

3）接着 FilterDispatcher 被调用，

FilterDispatcher 询问 ActionMapper 来决定这个请是否需要调用某个 Action

4）如果 ActionMapper 决定需要调用某个 Action，

FilterDispatcher 把请求的处理交给 ActionProxy

5）ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类

6）ActionProxy 创建一个 ActionInvocation 的实例。

7）ActionInvocation 实例使用命名模式来调用，

在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。

8）一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。 在这个过程中需要涉及到 ActionMapper

9）响应的返回是通过我们在 web.xml 中配置的过滤器

10）如果 ActionContextCleanUp 是当前使用的，则 FilterDispatecher 将不会清理 sreadlocal ActionContext;如果 ActionContextCleanUp 不使用，则将会去清理 sreadlocals。

二、说下 Struts 的设计模式

MVC 模式:

1、web 应用程序启动时就会加载并初始化 ActionServler。

2、用户提交表单时，一个配置好的 ActionForm 对象被创建，并被填入表单相应的数据，ActionServler 根据 Struts-config.xml 文件配置好的设置决定是否需要表单验证，如果需要就调用 ActionForm 的 Validate（）验证后选择将请求发送到哪个 Action，如果 Action 不存在，ActionServlet 会先创建这个对象，然后调用 Action 的 execute（）方法.

3、Execute（）从 ActionForm 对象中获取数据，完成业务逻辑，返回一个 ActionForward 对象，ActionServlet 再把客户请求转发给 ActionForward 对象指定的 jsp 组件，ActionForward 对象指定的 jsp 生成动态的网页，返回给客户。

三、拦截器和过滤器的区别

1、拦截器是基于 Java 反射机制的，而过滤器是基于函数回调的。

2、过滤器依赖于 servlet 容器，而拦截器不依赖于 servlet 容器。
3、拦截器只能对 Action 请求起作用，而过滤器则可以对几乎所有请求起作用。
4、拦截器可以访问 Action 上下文、值栈里的对象，而过滤器不能。
5、在 Action 的生命周期中，拦截器可以多次调用，而过滤器只能在容器初始化时被调用一次。

四、struts1 于 struts2 的比较
1、Action 类:
Struts1 要求 Action 类继承一个抽象基类。Struts1 的一个普遍问题是使用抽象类编程而不是接口。
Struts 2 Action 类可以实现一个 Action 接口，也可实现其他接口，使可选和定制的服务成为可能。Struts2 提供一个 ActionSupport 基类去 实现常用的接口。Action 接口不是必须的，任何有 execute 标识的 POJO 对象都可以用作 Struts2 的 Action 对象。
2、线程模式:
Struts1 Action 是单例模式并且必须是线程安全的，因为仅有 Action 的一个实例来处理所有的请求。单例策略限制了 Struts1 Action 能作的事，并且要在开发时特别小心。Action 资源必须是线程安全的或同步的。
Struts2 Action 对象为每一个请求产生一个实例，因此没有线程安全问题。（实际上，servlet 容器给每个请求产生许多可丢弃的对象，并且不会导致性能和垃圾回收问题）
3、Servlet 依赖:
Struts1 Action 依赖于 Servlet API ，因为当一个 Action 被调用时 HttpServletRequest 和 HttpServletResponse 被传递给 execute 方法。
Struts 2 Action 不依赖于容器，允许 Action 脱离容器单独被测试。如果需要，Struts2 Action 仍然可以访问初始的 request 和 response。但是，其他的元素减少或者消除了直接访问 HttpServetRequest 和 HttpServletResponse 的必要性。
4、可测性:
测试 Struts1 Action 的一个主要问题是 execute 方法暴露了 servlet API（这使得测试要依赖于容器）。一个第三方扩展——Struts TestCase——提供了一套 Struts1 的模拟对象（来进行测试）。
Struts 2 Action 可以通过初始化、设置属性、调用方法来测试，“依赖注入”支持也使测试更容易。
5、捕获输入:
Struts1 使用 ActionForm 对象捕获输入。所有的 ActionForm 必须继承一个基类。因为其他 JavaBean 不能用作 ActionForm，开发者经 常创建多余的类捕获输入。动态 Bean（DynaBeans）可以作为创建传统 ActionForm 的选择，但是，开发者可能是在重新描述(创建)已经存 在的 JavaBean（仍然会导致有冗余的 javabean）。
Struts 2 直接使用 Action 属性作为输入属性，消除了对第二个输入对象的需求。输入属性可能是有自己(子)属性的 rich 对象类型。Action 属性能够通过 web 页面上的 taglibs 访问。Struts2 也支持 ActionForm 模式。rich 对象类型，包括业务对象，能够用作输入/输出对象。这种 ModelDriven 特性简化了 taglib 对 POJO 输入对象的引用。
6、表达式语言：
Struts1 整合了 JSTL，因此使用 JSTL EL。这种 EL 有基本对象图遍历，但是对集合和索引属性的支持很弱。

Struts2 可以使用 JSTL，但是也支持一个更强大和灵活的表达式语言－－ "Object Graph Notation Language " (OGNL).

7、绑定值到页面（view）:

Struts 1 使用标准 JSP 机制把对象绑定到页面中来访问。

Struts 2 使用 "ValueStack "技术，使 taglib 能够访问值而不需要把你的页面（view）和对象绑定起来。ValueStack 策略允许通过一系列名称相同但类型不同的属性重用页面（view）。

8、类型转换：

Struts 1 ActionForm 属性通常都是 String 类型。Struts1 使用 Commons-Beanutils 进行类型转换。每个类一个转换器，对每一个实例来说是不可配置的。

Struts2 使用 OGNL 进行类型转换。提供基本和常用对象的转换器。

9、校验：

Struts 1 支持在 ActionForm 的 validate 方法中手动校验，或者通过 Commons Validator 的扩展来校验。同一个类可以有不同的校验内容，但不能校验子对象。

Struts2 支持通过 validate 方法和 XWork 校验框架来进行校验。XWork 校验框架使用为属性类类型定义的校验和内容校验，来支持 chain 校验子属性

10、Action 执行的控制：

Struts1 支持每一个模块有单独的 Request Processors（生命周期），但是模块中的所有 Action 必须共享相同的生命周期。

Struts2 支持通过拦截器堆栈（Interceptor Stacks）为每一个 Action 创建不同的生命周期。堆栈能够根据需要和不同的 Action 一起使用。

五、为什么要使用 Struts2

Struts2 是一个相当强大的 Java Web 开源框架，是一个基于 POJO 的 Action 的 MVC Web 框架。它基于当年的 Webwork 和 XWork 框架，继承其优点，同时做了相当的改进。

1、Struts2 基于 MVC 架构，框架结构清晰，开发流程一目了然，开发人员可以很好的掌控开发的过程。

2、使用 OGNL 进行参数传递。

OGNL 提供了在 Struts2 里访问各种作用域中的数据的简单方式，你可以方便的获取 Request，Attribute，Application，Session，Parameters 中的数据。大大简化了开发人员在获取这些数据时的代码量。

3、强大的拦截器

Struts2 的拦截器是一个 Action 级别的 AOP，Struts2 中的许多特性都是通过拦截器来实现的，例如异常处理，文件上传，验证等。拦截器是可配置与重用的，可以将一些通用的功能如：登录验证，权限验证等置于拦截器中以完成一些 Java Web 项目中比较通用的功能。在我实现的的一 Web 项目中，就是使用 Struts2 的拦截器来完成了系统中的权限验证功能。

4、易于测试

Struts2 的 Action 都是简单的 POJO，这样可以方便的对 Struts2 的 Action 编写测试用例，大大方便了 5Java Web 项目的测试。

易于扩展的插件机制在 Struts2 添加扩展是一件愉快而轻松的事情，只需要将所需要的 Jar 包放到 WEB-INF/lib 文件夹中，在 struts.xml 中作一些简单的设置就可以实现扩展。

6、模块化管理

Struts2 已经把模块化作为了体系架构中的基本思想，可以通过三种方法来将应用程序模块

化：将配置信息拆分成多个文件把自包含的应用模块创建为插件创建新的框架特性，即将与特定应用无关的新功能组织成插件，以添加到多个应用中去。
7、全局结果与声明式异常
为应用程序添加全局的 Result，和在配置文件中对异常进行处理，这样当处理过程中出现指定异常时，可以跳转到特定页面。
他的如此之多的优点，是很多人比较的青睐，与 spring ,hibernate 进行结合，组成了现在比较流行的 ssh 框架，当然每个公司都要自己的框架，也是 ssh 变异的产品。

六、struts2 有哪些优点？
1、在软件设计上 Struts2 的应用可以不依赖于 Servlet API 和 struts API。 Struts2 的这种设计属于无侵入式设计；
2、拦截器，实现如参数拦截注入等功能；
3、类型转换器，可以把特殊的请求参数转换成需要的类型；
4、多种表现层技术，如：JSP、freeMarker、Velocity 等；
5、Struts2 的输入校验可以对指定某个方法进行校验；
6、提供了全局范围、包范围和 Action 范围的国际化资源文件管理实现

七、struts2 是如何启动的？

1、struts2 框架是通过 Filter 启动的，即 StrutsPrepareAndExecuteFilter，此过滤器为 struts2 的核心过滤器；
3、StrutsPrepareAndExecuteFilter 的 init()方法中将会读取类路径下默认的配置文件 struts.xml 完成初始化操作。struts2 读取到 struts.xml 的内容后，是将内容封装进 javabean 对象然后存放在内存中，以后用户的每次请求处理将使用内存中的数据，而不是每次请求都读取 struts.xml 文件。

八、struts2 框架的核心控制器是什么？它有什么作用？
1、Struts2 框架的核心控制器是 StrutsPrepareAndExecuteFilter。
2、作用：
负责拦截由<url-pattern>/*</url-pattern>指定的所有用户请求，当用户请求到达时，该 Filter 会过滤用户的请求。默认情况下，如果用户请求的路径
不带后缀或者后缀以.action 结尾，这时请求将被转入 struts2 框架处理，否则 struts2 框架将略过该请求的处理。
可以通过常量"struts.action.extension"修改 action 的后缀，如：
<constant name="struts.action.extension" value="do"/>
如果用户需要指定多个请求后缀，则多个后缀之间以英文逗号（,）隔开。
<constant name="struts.action.extension" value="do,Go"/>

九、struts2 配置文件的加载顺序？
struts.xml ——> struts.properties

常量可以在 struts.xml 或 struts.properties 中配置，如果在多个文件中配置了同一个常量，则后一个文件中配置的常量值会覆盖前面文件中配置的常量值.
struts.xml 文件的作用：通知 Struts2 框架加载对应的 Action 资源

十、struts2 常量的修改方式？
常量可以在 struts.xml 或 struts.properties 中配置，两种配置方式如下：
1、在 struts.xml 文件中配置常量
<constant name="struts.action.extension" value="do"/>
2、在 struts.properties 中配置常量（struts.properties 文件放置在 src 下）：
struts.action.extension=do

struts2 如何访问 HttpServletRequest、HttpSession、ServletContext 三个域对象？
方案一：
HttpServletRequest request =ServletActionContext.getRequest();
HttpServletResponse response =ServletActionContext.getResponse();
HttpSession session= request.getSession();
ServletContext servletContext=ServletActionContext.getServletContext();

方案二：
类 implements ServletRequestAware,ServletResponseAware ， SessionAware ， ServletContextAware
注意：框架自动传入对应的域对象

十二、struts2 是如何管理 action 的？这种管理方式有什么好处？
struts2 框架中使用包来管理 Action，包的作用和 java 中的类包是非常类似的。
主要用于管理一组业务功能相关的 action。在实际应用中，我们应该把一组业务功能相关的 Action 放在同一个包下。
struts2 中的默认包 struts-default 有什么作用？
1、struts-default 包是由 struts 内置的，它定义了 struts2 内部的众多拦截器和 Result 类型，而 Struts2 很多核心的功能都是通过这些内置的拦截器实现，如：从请求中
把请求参数封装到 action、文件上传和数据验证等等都是通过拦截器实现的。当包继承了 struts-default 包才能使用 struts2 为我们提供的这些功能。
2、struts-default 包是在 struts-default.xml 中定义，struts-default.xml 也是 Struts2 默认配置文件。 Struts2 每次都会自动加载 struts-default.xml 文件。
3、通常每个包都应该继承 struts-default 包。

十三、struts2 如何对指定的方法进行验证？
1）validate()方法会校验 action 中所有与 execute 方法签名相同的方法；

2）要校验指定的方法通过重写 validateXxx()方法实现， validateXxx()只会校验 action 中方法名为 Xxx 的方法。其中 Xxx 的第一个字母要大写；
3）当某个数据校验失败时，调用 addFieldError()方法往系统的 fieldErrors 添加校验失败信息（为了使用 addFieldError()方法，action 可以继承 ActionSupport）， 如果系统 的 fieldErrors 包含失败信息，struts2 会将请求转发到名为 input 的 result；
4）在 input 视图中可以通过<s:fielderror/>显示失败信息。
5）先执行 validateXxxx()->validate()->如果出错了，会转发<result name="input"/>所指定的页面，如果不出错，会直接进行 Action::execute()方法

十四、struts2 默认能解决 get 和 post 提交方式的乱码问题吗？
不能。struts.i18n.encoding=UTF-8 属性值只能解析 POST 提交下的乱码问题。

十五、请你写出 struts2 中至少 5 个的默认拦截器？
fileUpload 提供文件上传功能
i18n 记录用户选择的 locale
cookies 使用配置的 name,value 来是指 cookies
checkbox 添加了 checkbox 自动处理代码，将没有选中的 checkbox 的内容设定为 false，而 html 默认情况下不提交没有选中的 checkbox。
chain 让前一个 Action 的属性可以被后一个 Action 访问，现在和 chain 类型的 result（）结合使用。
alias 在不同请求之间将请求参数在不同名字件转换，请求内容不变

十六、值栈 ValueStack 的原理与生命周期？
1、ValueStack 贯穿整个 Action 的生命周期，保存在 request 域中，所以 ValueStack 和 request 的生命周期一样。当 Struts2 接受一个请求时，会迅速创建 ActionContext，
ValueStack，action。然后把 action 存放进 ValueStack，所以 action 的实例变量可以被 OGNL 访问。 请求来的时候，action、ValueStack 的生命开始，请求结束，action、 ValueStack 的生命结束；
2、action 是多例的，和 Servlet 不一样，Servelt 是单例的；
3、每个 action 的都有一个对应的值栈，值栈存放的数据类型是该 action 的实例，以及该 action 中的实例变量，Action 对象默认保存在栈顶；
4、ValueStack 本质上就是一个 ArrayList；
5、关于 ContextMap，Struts 会把下面这些映射压入 ContextMap 中：
parameters : 该 Map 中包含当前请求的请求参数
request : 该 Map 中包含当前 request 对象中的所有属性 session :该 Map 中包含当前 session 对象中的所有属性
application :该 Map 中包含当前 application 对象中的所有属性
attr:该 Map 按如下顺序来检索某个属性: request, session, application
6、使用 OGNL 访问值栈的内容时，不需要#号，而访问 request、session、application、attr 时，需要加#号；
7、注意： Struts2 中，OGNL 表达式需要配合 Struts 标签才可以使用。如：<s:property value="name"/>

8、在 struts2 配置文件中引用 ognl 表达式 ,引用值栈的值 ，此时使用的"$"，而不是#或者%;

十七、ActionContext、ServletContext、pageContext 的区别？

1、ActionContext 是当前的 Action 的上下文环境，通过 ActionContext 可以获取到 request、session、ServletContext 等与 Action 有关的对象的引用；

2、ServletContext 是域对象，一个 web 应用中只有一个 ServletContext，生命周期伴随整个 web 应用；

3、pageContext 是 JSP 中的最重要的一个内置对象，可以通过 pageContext 获取其他域对象的应用，同时它是一个域对象，作用范围只针对当前页面，当前页面结束时，pageContext 销毁，

生命周期是 JSP 四个域对象中最小的。

十八、result 的 type 属性中有哪几种结果类型？

一共 10 种：

dispatcher

struts 默认的结果类型，把控制权转发给应用程序里的某个资源不能把控制权转发给一个外部资源，若需要把控制权重定向到一个外部资源, 应该使用

redirect 结果类型

redirect　　把响应重定向到另一个资源（包括一个外部资源）

redirectAction　　把响应重定向到另一个 Action

freemarker、velocity、chain、httpheader、xslt、plainText、stream

十九、拦截器的生命周期与工作过程？

1、每个拦截器都是实现了 Interceptor 接口的 Java 类；

2、init(): 该方法将在拦截器被创建后立即被调用, 它在拦截器的生命周期内只被调用一次. 可以在该方法中对相关资源进行必要的初始化；

3、intercept(ActionInvocation invocation): 每拦截一个动作请求, 该方法就会被调用一次；

4、destroy: 该方法将在拦截器被销毁之前被调用, 它在拦截器的生命周期内也只被调用一次；

5、struts2 中有内置了 18 个拦截器。

二十、struts2 如何完成文件的上传？

1、JSP 页面：

1）JSP 页面的上传文件的组件：<s: file name=”upload” />，如果需要一次上传多个文件, 就必须使用多个 file 标签, 但它们的名字必须是相同的，即：

name=“xxx”的值必须一样；

2）必须把表单的 enctype 属性设置为：multipart/form-data；

3）表单的方法必须为 post，因为 post 提交的数据在消息体中，而无大小限制。

2、对应的 action：

1）在 Action 中新添加 3 个和文件上传相关的属性；

2）如果是上传单个文件, uploadImage 属性的类型就是 java.io.File, 它代表被上传的文件, 第二个和第三个属性的类型是 String, 它们分别代表上传文
件的文件名和文件类型，定义方式是分别是：
jsp 页面 file 组件的名称+ContentType, jsp 页面 file 组件的名称+FileName
3）如果上上传多个文件, 可以使用数组或 List

二十一、struts 的工作原理
1、初始化，读取 struts-config.xml、web.xml 等配置文件（所有配置文件的初始化）
2、发送 HTTP 请求,客户端发送以.do 结尾的请求
3、填充 FormBean（实例化、复位、填充数据、校验、保存）
4、将请求转发到 Action（调用 Action 的 execute（）方法）
5、处理业务（可以调用后台类，返回 ActionForward 对象）
6、返回目标响应对象（从 Action 返回到 ActionServlet）
7、转换 Http 请求到目标响应对象（查找响应，根据返回的 Forward keyword）
8、Http 响应，返回到 Jsp 页面

二十二、用自己的话简要阐述 struts2 的执行流程。
Struts 2 框架本身大致可以分为 3 个部分：核心控制器 FilterDispatcher、业务控制器 Action 和用户实现的企业业务逻辑组件。
核心控制器 FilterDispatcher 是 Struts 2 框架的基础,包含了框架内部的控制流程和处理机制。
业务控制器 Action 和业务逻辑组件是需要用户来自己实现的。用户在开发 Action 和业务逻辑组件的同时，还需要编写相关的配置文件，供核心控制器 FilterDispatcher 来使用。
Struts 2 的工作流程相对于 Struts 1 要简单，与 WebWork 框架基本相同，所以说 Struts 2 是 WebWork 的升级版本。基本简要流程如下：
1、客户端浏览器发出 HTTP 请求。
2、根据 web.xml 配置，该请求被 FilterDispatcher 接收。
3、根据 struts.xml 配置，找到需要调用的 Action 类和方法， 并通过 IoC 方式，将值注入给 Aciton。
4、Action 调用业务逻辑组件处理业务逻辑，这一步包含表单验证。
5、Action 执行完毕，根据 struts.xml 中的配置找到对应的返回结果 result，并跳转到相应页面。
6、返回 HTTP 响应到客户端浏览器。
它是以 Webwork 的设计思想为核心，吸收 struts1 的优点，可以说 struts2 是 struts1 和 Webwork 结合的产物。

struts2 的工作原理图： 一个请求在 Struts2 框架中的处理分为以下几个步骤：

1.客户端发出一个指向 servlet 容器的请求(tomcat)；
2.这个请求会经过图中的几个过滤器，最后会到达 FilterDispatcher 过滤器。
3.过滤器 FilterDispatcher 是 struts2 框架的心脏，在处理用户请求时，它和请求一起相互配合访问 struts2 的底层框架结构。在 web 容器启动时，struts2 框架会自动加载配置文件里相关参数，并转换成相应的类。 如：ConfigurationManager、ActionMapper 和 ObjectFactory。

ConfigurationManager 存有配置文件的一 些基本信息，ActionMapper 存有 action 的配置信息。在请求过程中所有的对象（Action，Results， Interceptors，等）都是通过 ObjectFactory 来创建的。过滤器会通过询问 ActionMapper 类来查找请求中 需要用到的 Action。

4.如果找到需要调用的 Action，过滤器会把请求的处理交给 ActionProxy。ActionProxy 为 Action 的代理对象 。ActionProxy 通过 ConfigurationManager 询问框架的配置文件，找到需要调用的 Action 类。5.ActionProxy 创建一个 ActionInvocation 的实例。ActionInvocation 在 ActionProxy 层之下，它表示了 Action 的执行状态,或者说它控制的 Action 的执行步骤。它持有 Action 实例和所有的 Interceptor。 6.ActionInvocation 实例使用命名模式来调用，1） ActionInvocation 初始化时，根据配置，加载 Action 相 关的所有 Interceptor。2）通过 ActionInvocation.invoke 方法调用 Action 实现时，执行 Interceptor。在 调用 Action 的过程前后，涉及到相关拦截器(intercepetor)的调用。

7. 一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果 通常是（但不总是，也可能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表 示的过程中可以使用 Struts2 框架中继承的标签。

在 Java J2EE 方面进行面试时，常被问起的 Hibernate 面试问题，大多都是针对基于 Web 的企业级应用开发者的角色的。Hibernate 框架在 Java 界的成功和高度的可接受性使得它成为了 Java 技术栈中最受欢迎的对象关系影射（ORM）解决方案。Hibernate 将你从数据库相关的编码中解脱了出来，使你可以更加专注地利用强大的面向对象的设计原则来实现核心的业务逻辑。采用 Hibernate 后，你就能够相当容易地在不同的数据库间进行切换，而且你还可以利用 Hibernate 提供的开箱即用的二级缓存以及查询缓存功能。你也知道，大部分 Java 面试中所提的问题不仅仅会涉及 Java 的核心部分，而且还会涉及其它的 Java 框架，比如，根据项目的要求也有可能会问到 Spring 框架方面的问题或者 Struts 方面的问题。如果你要参加的项目使用了 Hibernate 作为 ORM 解决方案，你就应该同时准备好回答 Spring 和 Hibernate 这两个框架方面的问题。好好看看 JD 或者职位说明，如果其中的任何地方出现了 Hibernate 这个词，就要准备好怎样来面对 Hibernate 方面的问题。

本文给出了一个 Hibernate 面试问题列表，这些都是我从朋友以及同事那里搜集来的。Hibernate 是一个非常流行的对象关系影射框架，熟悉 Hibernate 的优势所在以及 Hibernate 的 Sesion API 是搞定 Hibernate 面试之关键所在。

Hibernate 中 get 和 load 有什么不同之处?

把 get 和 load 放到一起进行对比是 Hibernate 面试时最常问到的问题，这是因为只有正确理解 get()和 load()这二者后才有可能高效地使用 Hibernate。get 和 load 的最大区别是，如果在缓存中没有找到相应的对象，get 将会直接访问数据库并返回一个完全初始化好的对象，而这个过程有可能会涉及到多个数据库调用；而 load 方法在缓存中没有发现对象的情况下，只会返回一个代理对象，只有在对象 getId()之外的其它方法被调用时才会真正去访问数据库，这样就能在某些情况下大幅度提高性能。你也可以参考 Hibernate 中 get 和 load 的不同之处， 此链接给出了更多的不同之处并对该问题进行了更细致的讨论。

Hibernate 中 save、persist 和 saveOrUpdate 这三个方法的不同之处？

除了 get 和 load，这又是另外一个经常出现的 Hibernate 面试问题。 所有这三个方法，也就是 save()、saveOrUpdate()和 persist()都是用于将对象保存到数据库中的方法，但其中有些细微的差别。例如，save()只能 INSERT 记录，但是 saveOrUpdate()可以进行 记录的 INSERT 和 UPDATE。还有，save()的返回值是一个 Serializable 对象，而 persist()方法返回值为 void。你还可以访问 save、persist 以及 saveOrUpdate，找到它们所有的不同之处。

Hibernate 中的命名 SQL 查询指的是什么?

Hibernate 的这个面试问题同 Hibernate 提供的查询功能相关。命名查询指的是用<sql-query>标签在影射文档中定义的 SQL 查询，可以通过使用 Session.getNamedQuery()方法对它进行调用。命名查询使你可以使用你所指定的一个名字拿到某个特定的查询。 Hibernate 中的命名查询可以使用注解来定义，也可以使用我前面提到的 xml 影射问句来定义。在 Hibernate 中，@NameQuery 用来定义单个的命名查询，@NameQueries 用来定义多个命名查询。

Hibernate 中的 SessionFactory 有什么作用? SessionFactory 是线程安全的吗？

这也是 Hibernate 框架的常见面试问题。顾名思义，SessionFactory 就是一个用于创建 Hibernate 的 Session 对象的工厂。SessionFactory 通常是在应用启动时创建好的，应用程序中的代码用它来获得 Session 对象。作为一个单个的数据存储，它也是 线程安全的，所以多个线程可同时使用同一个 SessionFactory。Java JEE 应用一般只有一个 SessionFactory，服务于客户请求的各线程都通过这个工厂来获得 Hibernate 的 Session 实例，这也是为什么 SessionFactory 接口的实现必须是线程安全的原因。还有，SessionFactory 的内部状态包含着同对象关系影射有关的所有元数据，它是 不可变的，一旦创建好后就不能对其进行修改了。

Hibernate 中的 Session 指的是什么？可否将单个的 Session 在多个线程间进行共享？

前面的问题问完之后，通常就会接着再问这两个问题。问完 SessionFactory 的问题后就该轮到 Session 了。Session 代表着 Hibernate 所做的一小部分工作，它负责维护者同数据库的链接而且 不是线程安全的，也就是说，Hibernage 中的 Session 不能在多个线程间进行共享。虽然 Session 会以主动滞后的方式获得数据库连接，但是 Session 最好还是在用完之后立即将其关闭。

hibernate 中 sorted collection 和 ordered collection 有什么不同?

T 这个是你会碰到的所有 Hibernate 面试问题中比较容易的问题。sorted collection 是通过使用 Java 的 Comparator 在内存中进行排序的，ordered collection 中的排序用的是数据库的 order by 子句。对于比较大的数据集，为了避免在内存中对它们进行排序而出现 Java 中的 OutOfMemoryError，最好使用 ordered collection。

Hibernate 中 transient、persistent、detached 对象三者之间有什么区别？

在 Hibernate 中，对象具有三种状态：transient、persistent 和 detached。同 Hibernate 的 session 有关联的对象是 persistent 对象。对这种对象进行的所有修改都会按照事先设定的刷新策略，反映到数据库之中，也即，可以在对象的任何一个属性发生改变时自动刷新，也可以通过调用 Session.flush()方法显式地进行刷新。如果一个对象原来同 Session 有关联关系，但当下却没有关联关系了，这样的对象就是 detached 的对象。你可以通过调用任意一个 session 的 update()或者 saveOrUpdate()方法，重新将该 detached 对象同相应的 seesion 建立关联关系。Transient 对象指的是新建的持久化类的实例，它还从未同 Hibernate 的任何 Session 有过关联关系。同样的，你可以调用 persist()或者 save()方法，将 transient 对象变成 persistent 对象。可要记住，这里所说的 transient 指的可不是 Java 中的 transient 关键字，二者风马牛不相及。

Hibernate 中 Session 的 lock()方法有什么作用?

这是一个比较棘手的 Hibernate 面试问题，因为 Session 的 lock()方法重建了关联关系却并没有同数据库进行同步和更新。因此，你在使用 lock()方法时一定要多加小心。顺便说一下，在进行关联关系重建时，你可以随时使用 Session 的 update()方法同数据库进行同步。有时这个问题也可以这么来问：Session 的 lock()方法和 update()方法之间有什么区别？。这个小节中的关键点也可以拿来回答这个问题。

Hibernate 中二级缓存指的是什么？

这是同 Hibernate 的缓存机制相关的第一个面试问题，不出意外后面还会有更多这方面的问题。二级缓存是在 SessionFactory 这个级别维护的缓存，它能够通过节省几番数据库调用往返来提高性能。还有一点值得注意，二级缓存是针对整个应用而不是某个特定的 session 的。

Hibernate 中的查询缓存指的是什么？

这个问题有时是作为上个 Hibernate 面试问题的后继问题提出的。查询缓存实际上保存的是 sql 查询的结果，这样再进行相同的 sql 查询就可以之间从缓存中拿到结果了。为了改善性能，查询缓存可以同二级缓存一起来使用。Hibernate 支持用多种不同的开源缓存方案，比如 EhCache，来实现查询缓存。

为什么在 Hibernate 的实体类中要提供一个无参数的构造器这一点非常重要？

每个 Hibernate 实体类必须包含一个 无参数的构造器，这是因为 Hibernate 框架要使用 Reflection API，通过调用 Class.newInstance()来创建这些实体类的实例。如果在实体类中找不到无参数的构造器，这个方法就会抛出一个 InstantiationException 异常。

可不可以将 Hibernate 的实体类定义为 final 类?

是的，你可以将 Hibernate 的实体类定义为 final 类，但这种做法并不好。因为 Hibernate 会使用代理模式在延迟关联的情况下提高性能，如果你把实体类定义成 final 类之后，因为 Java 不允许对 final 类进行扩展，所以 Hibernate 就无法再使用代理了，如此一来就限制了使用可以提升性能的手段。不过，如果你的持久化类实现了一个接口而且在该接口中声明了所有定义于实体类中的所有 public 的方法轮到话，你就能够避免出现前面所说的不利后果。

Java 开发者的 Hibernate 面试问答列表就到此为止了。没人会对 Hibernate 作为 ORM 解决方案的受欢迎程度产生怀疑，如果你要申请的是 Java J2EE 方面的职位，你就等着人来问你 Hibernate 方面的面试问题吧。在 JEE 界，Spring 和 Hibernate 是两个最流行的 Java 框架。要是你被问到了其它也值得分享的

1.Mybatis 比 IBatis 比较大的几个改进是什么

a.有接口绑定,包括注解绑定 sql 和 xml 绑定 Sql ,

b.动态 sql 由原来的节点配置变成 OGNL 表达式,

c. 在一对一,一对多的时候引进了 association,在一对多的时候引入了 collection 节点,不过都是在 resultMap 里面配置

2.什么是 MyBatis 的接口绑定,有什么好处

接口映射就是在 IBatis 中任意定义接口,然后把接口里面的方法和 SQL 语句绑定,

我们直接调用接口方法就可以,这样比起原来了 SqlSession 提供的方法我们可以有更加灵活的选择和设置.

3.接口绑定有几种实现方式,分别是怎么实现的?

接口绑定有两种实现方式,一种是通过注解绑定,就是在接口的方法上面加上

@Select@Update 等注解里面包含 Sql 语句来绑定,另外一种就是通过 xml 里面写 SQL 来绑定,

在这种情况下,要指定 xml 映射文件里面的 namespace 必须为接口的全路径名.

4.什么情况下用注解绑定,什么情况下用 xml 绑定

当 Sql 语句比较简单时候,用注解绑定,

当 SQL 语句比较复杂时候,用 xml 绑定,一般用 xml 绑定的比较多

5.MyBatis 实现一对一有几种方式?具体怎么操作的

有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,

通过在 resultMap 里面配置 association 节点配置一对一的类就可以完成;

嵌套查询是先查一个表,根据这个表里面

的结果的外键 id,去再另外一个表里面查询数据,也是通过 association 配置,但另外一个表

的查询通过 select 属性配置

6.MyBatis 实现一对多有几种方式,怎么操作的

有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面配

置 collection 节点配置一对多的类就可以完成;

嵌套查询是先查一个表,根据这个表里面的

结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个

表的

查询通过 select 节点配置

7.MyBatis 里面的动态 Sql 是怎么设定的?用什么语法?

MyBatis 里面的动态 Sql 一般是通过 if 节点来实现,通过 OGNL 语法来实现,但是如果要写的完

整,必须配合 where,trim 节点,where 节点是判断包含节点有内容就插入 where,否则不插

入,trim 节点是用来判断如果动态语句是以 and 或 or 开始,那么会自动把这个 and 或者 or 取

掉

8.IBatis 和 MyBatis 在核心处理类分别叫什么

IBatis 里面的核心处理类交 SqlMapClient,

MyBatis 里面的核心处理类叫做 SqlSession

9.IBatis 和 MyBatis 在细节上的不同有哪些

在 sql 里面变量命名有原来的#变量# 变成了#{变量}

原来的$变量$变成了${变量},

原来在 sql 节点里面的 class 都换名字交 type

原来的 queryForObject queryForList 变成了 selectOne selectList

原来的别名设置在映射文件里面放在了核心配置文件里

10.讲下 MyBatis 的缓存

MyBatis 的缓存分为一级缓存和二级缓存,

一级缓存放在 session 里面,默认就有,二级缓存放在它的命名空间里,默认是打开的,

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),可在它的映射文件中配置<cache/>

11.MyBatis(IBatis)的好处是什么

ibatis 把 sql 语句从 Java 源程序中独立出来，

放在单独的 XML 文件中编写，给程序的维护带来了很大便利。

ibatis 封装了底层 JDBC API 的调用细节，并能自动将结果集转换成 Java Bean 对象，

大大简化了 Java 数据库编程的重复工作。

因为 Ibatis 需要程序员自己去编写 sql 语句，

程序员可以结合数据库自身的特点灵活控制 sql 语句，

因此能够实现比 hibernate 等全自动 orm 框架更高的查询效率，能够完成复杂查询。

Spring 概述

1. 什么是 spring?

Spring 是个 java 企业级应用的开源开发框架。Spring 主要用来开发 Java 应用，但是有些扩展是针对构建 J2EE 平台的 web 应用。Spring 框架目标是简化 Java 企业级应用开发，并通过 POJO 为基础的编程模型促进良好的编程习惯。

2. 使用 Spring 框架的好处是什么？

轻量：Spring 是轻量的，基本的版本大约 2MB。
控制反转：Spring 通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。
面向切面的编程(AOP)：Spring 支持面向切面的编程，并且把应用业务逻辑和系统服务分开。
容器：Spring 包含并管理应用中对象的生命周期和配置。
MVC 框架：Spring 的 WEB 框架是个精心设计的框架，是 Web 框架的一个很好的替代品。
事务管理：Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。
异常处理：Spring 提供方便的 API 把具体技术相关的异常（比如由 JDBC，Hibernate or JDO 抛出的）转化为一致的 unchecked 异常。

3. Spring 由哪些模块组成?

以下是 Spring 框架的基本模块：

Core module
Bean module
Context module
Expression Language module
JDBC module
ORM module
OXM module
Java Messaging Service(JMS) module
Transaction module
Web module
Web-Servlet module
Web-Struts module
Web-Portlet module

4. 核心容器（应用上下文) 模块。

这是基本的 Spring 模块，提供 spring 框架的基础功能，BeanFactory 是 任何以 spring 为基础的应用的核心。Spring 框架建立在此模块之上，它使 Spring 成为一个容器。

5. BeanFactory – BeanFactory 实现举例。

Bean 工厂是工厂模式的一个实现，提供了控制反转功能，用来把应用的配置和依赖从正真的应用代码中分离。

最常用的 BeanFactory 实现是 XmlBeanFactory 类。

6. XMLBeanFactory

最常用的就是 org.springframework.beans.factory.xml.XmlBeanFactory ，它根据 XML 文件中的定义加载 beans。该容器从 XML 文件读取配置元数据并用它去创建一个完全配置的系统或应用。

7. 解释 AOP 模块

AOP 模块用于发给我们的 Spring 应用做面向切面的开发， 很多支持由 AOP 联盟提供，这样就确保了 Spring 和其他 AOP 框架的共通性。这个模块将元数据编程引入 Spring。

8. 解释 JDBC 抽象和 DAO 模块。

通过使用 JDBC 抽象和 DAO 模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用 Spring 的 AOP 模块给 Spring 应用中的对象提供事务管理服务。

9. 解释对象/关系映射集成模块。

Spring 通过提供 ORM 模块，支持我们在直接 JDBC 之上使用一个对象/关系映射映射(ORM)工具，Spring 支持集成主流的 ORM 框架，如 Hiberate,JDO 和 iBATIS SQL Maps。Spring 的事务管理同样支持以上所有 ORM 框架及 JDBC。

10. 解释 WEB 模块。

Spring 的 WEB 模块是构建在 application context 模块基础之上，提供一个适合 web 应用的上下文。这个模块也包括支持多种面向 web 的任务，如透明地处理多个文件上传请求和程序级请求参数的绑定到你的业务对象。它也有对 Jakarta Struts 的支持。

12. Spring 配置文件

Spring 配置文件是个 XML 文件，这个文件包含了类信息，描述了如何配置它们，以及如何相互调用。

13. 什么是 Spring IOC 容器？

Spring IOC 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。

14. IOC 的优点是什么？

IOC 或 依赖注入把应用的代码量降到最低。它使应用容易测试，单元测试不再需要单例和 JNDI 查找机制。最小的代价和最小的侵入性使松散耦合得以实现。IOC 容器支持加载服务时的饿汉式初始化和懒加载。

15. ApplicationContext 通常的实现是什么?

FileSystemXmlApplicationContext ：此容器从一个 XML 文件中加载 beans 的定义，XML Bean 配置文件的全路径名必须提供给它的构造函数。
ClassPathXmlApplicationContext：此容器也从一个 XML 文件中加载 beans 的定义，这里，你需要正确设置 classpath 因为这个容器将在 classpath 里找 bean 配置。
WebXmlApplicationContext：此容器加载一个 XML 文件，此文件定义了一个 WEB 应用的所有 bean。

16. Bean 工厂和 Application contexts 有什么区别？

Application contexts 提供一种方法处理文本消息，一个通常的做法是加载文件资源（比如镜像），它们可以向注册为监听器的 bean 发布事件。另外，在容器或容器内的对象上执行的那些不得不由 bean 工厂以程序化方式处理的操作，可以在 Application contexts 中以声明的方式处理。Application contexts 实现了 MessageSource 接口，该接口的实现以可插拔的方式提供获取本地化消息的方法。

17. 一个 Spring 的应用看起来象什么？

一个定义了一些功能的接口。
这实现包括属性，它的 Setter ， getter 方法和函数等。
Spring AOP。
Spring 的 XML 配置文件。
使用以上功能的客户端程序。
依赖注入

18. 什么是 Spring 的依赖注入？

依赖注入，是 IOC 的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC 容器）负责把他们组装起来。

19. 有哪些不同类型的 IOC（依赖注入）方式？

构造器依赖注入：构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。
Setter 方法注入：Setter 方法注入是容器通过调用无参构造器或无参 static 工厂 方法实例化 bean 之后，调用该 bean 的 setter 方法，即实现了基于 setter 的依赖注入。
20. 哪种依赖注入方式你建议使用，构造器注入，还是 Setter 方法注入？

你两种依赖方式都可以使用，构造器注入和 Setter 方法注入。最好的解决方案是用构造器参数实现强制依赖，setter 方法实现可选依赖。

Spring Beans

21.什么是 Spring beans?

Spring beans 是那些形成 Spring 应用的主干的 java 对象。它们被 Spring IOC 容器初始化，装配，和管理。这些 beans 通过容器中配置的元数据创建。比如，以 XML 文件中<bean/> 的形式定义。

Spring 框架定义的 beans 都是单件 beans。在 bean tag 中有个属性”singleton”，如果它被赋为 TRUE，bean 就是单件，否则就是一个 prototype bean。默认是 TRUE，所以所有在 Spring 框架中的 beans 缺省都是单件。

22. 一个 Spring Bean 定义 包含什么？

一个 Spring Bean 的定义包含容器必知的所有配置元数据，包括如何创建一个 bean，它的生命周期详情及它的依赖。

23. 如何给 Spring 容器提供配置元数据?

这里有三种重要的方法给 Spring 容器提供配置元数据。

XML 配置文件。

基于注解的配置。

基于 java 的配置。

24. 你怎样定义类的作用域?

当定义一个<bean> 在 Spring 里，我们还能给这个 bean 声明一个作用域。它可以通过 bean 定义中的 scope 属性来定义。如，当 Spring 要在需要的时候每次生产一个新的 bean 实例，bean 的 scope 属性被指定为 prototype。另一方面，一个 bean 每次使用的时候必须返回同一个实例，这个 bean 的 scope 属性 必须设为 singleton。

25. 解释 Spring 支持的几种 bean 的作用域。

Spring 框架支持以下五种 bean 的作用域：

singleton : bean 在每个 Spring ioc 容器中只有一个实例。
prototype：一个 bean 的定义可以有多个实例。
request：每次 http 请求都会创建一个 bean，该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
session：在一个 HTTP Session 中，一个 bean 定义对应一个实例。该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
global-session：在一个全局的 HTTP Session 中，一个 bean 定义对应一个实例。该作用域仅在基于 web 的 Spring ApplicationContext 情形下有效。
缺省的 Spring bean 的作用域是 Singleton.

26. Spring 框架中的单例 bean 是线程安全的吗?

不，Spring 框架中的单例 bean 不是线程安全的。

27. 解释 Spring 框架中 bean 的生命周期。

Spring 容器 从 XML 文件中读取 bean 的定义，并实例化 bean。
Spring 根据 bean 的定义填充所有的属性。
如果 bean 实现了 BeanNameAware 接口，Spring 传递 bean 的 ID 到 setBeanName 方法。
如果 Bean 实现了 BeanFactoryAware 接口， Spring 传递 beanfactory 给 setBeanFactory 方法。
如果有任何与 bean 相关联的 BeanPostProcessors，Spring 会在 postProcesserBeforeInitialization() 方法内调用它们。
如果 bean 实现 IntializingBean 了，调用它的 afterPropertySet 方法，如果 bean 声明了初始化方法，调用此初始化方法。
如果有 BeanPostProcessors 和 bean 关联，这些 bean 的 postProcessAfterInitialization() 方法将被调用。
如果 bean 实现了 DisposableBean，它将调用 destroy()方法。

28. 哪些是重要的 bean 生命周期方法？ 你能重载它们吗？

有两个重要的 bean 生命周期方法，第一个是 setup ， 它是在容器加载 bean 的时候被调用。第二个方法是 teardown 它是在容器卸载类的时候被调用。

The bean 标签有两个重要的属性（init-method 和 destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct 和@PreDestroy）。

29. 什么是 Spring 的内部 bean？

当一个 bean 仅被用作另一个 bean 的属性时，它能被声明为一个内部 bean，为了定义 inner bean，在 Spring 的 基于 XML 的 配置元数据中，可以在 <property/>或 <constructor-arg/> 元

素内使用<bean/> 元素，内部 bean 通常是匿名的，它们的 Scope 一般是 prototype。

30. 在 Spring 中如何注入一个 java 集合？

Spring 提供以下几种集合的配置元素：

<list>类型用于注入一列值，允许有相同的值。
<set> 类型用于注入一组值，不允许有相同的值。
<map> 类型用于注入一组键值对，键和值都可以为任意类型。
<props>类型用于注入一组键值对，键和值都只能为 String 类型。

31. 什么是 bean 装配?

装配，或 bean 装配是指在 Spring 容器中把 bean 组装到一起，前提是容器需要知道 bean 的依赖关系，如何通过依赖注入来把它们装配到一起。

32. 什么是 bean 的自动装配？

Spring 容器能够自动装配相互合作的 bean，这意味着容器不需要<constructor-arg>和<property>配置，能通过 Bean 工厂自动处理 bean 之间的协作。

33. 解释不同方式的自动装配 。

有五种自动装配的方式，可以用来指导 Spring 容器用自动装配方式来进行依赖注入。

no：默认的方式是不进行自动装配，通过显式设置 ref 属性来进行装配。
byName：通过参数名 自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byname，之后容器试图匹配、装配和该 bean 的属性具有相同名字的 bean。
byType:：通过参数类型自动装配，Spring 容器在配置文件中发现 bean 的 autowire 属性被设置成 byType，之后容器试图匹配、装配和该 bean 的属性具有相同类型的 bean。如果有多个 bean 符合条件，则抛出错误。
constructor：这个方式类似于 byType， 但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
autodetect：首先尝试使用 constructor 来自动装配，如果无法工作，则使用 byType 方式。

34.自动装配有哪些局限性 ？

自动装配的局限性是：

重写： 你仍需用 <constructor-arg>和 <property> 配置来定义依赖，意味着总要重写自动装配。
基本数据类型：你不能自动装配简单的属性，如基本数据类型，String 字符串，和类。
模糊特性：自动装配不如显式装配精确，如果有可能，建议使用显式装配。

35. 你可以在 Spring 中注入一个 null 和一个空字符串吗？

可以。

Spring 注解

36. 什么是基于 Java 的 Spring 注解配置？给一些注解的例子.

基于 Java 的配置，允许你在少量的 Java 注解的帮助下，进行你的大部分 Spring 配置而非通过 XML 文件。

以@Configuration 注解为例，它用来标记类可以当做一个 bean 的定义，被 Spring IOC 容器使用。另一个例子是@Bean 注解，它表示此方法将要返回一个对象，作为一个 bean 注册进 Spring 应用上下文。

37. 什么是基于注解的容器配置?

相对于 XML 文件，注解型的配置依赖于通过字节码元数据装配组件，而非尖括号的声明。

开发者通过在相应的类，方法或属性上使用注解的方式，直接组件类中进行配置，而不是使用 xml 表述 bean 的装配关系。

38. 怎样开启注解装配？

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在 Spring 配置文件中配置 <context:annotation-config/>元素。

39. @Required 注解

这个注解表明 bean 的属性必须在配置的时候设置，通过一个 bean 定义的显式的属性值或通过自动装配，若@Required 注解的 bean 属性未被设置，容器将抛出 BeanInitializationException。

40. @Autowired 注解

@Autowired 注解提供了更细粒度的控制，包括在何处以及如何完成自动装配。它的用法和@Required 一样，修饰 setter 方法、构造器、属性或者具有任意名称和/或多个参数的 PN 方法。

41. @Qualifier 注解

当有多个相同类型的 bean 却只有一个需要自动装配时，将@Qualifier 注解和@Autowire 注解结合使用以消除这种混淆，指定需要装配的确切的 bean。

Spring 数据访问

42.在 Spring 框架中如何更有效地使用 JDBC?

使用 SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写 statements 和 queries 从数据存取数据，JDBC 也可以在 Spring 框架提供的模板类的帮助下更有效地被使用，这个模板叫 JdbcTemplate （例子见这里 here）

43. JdbcTemplate

JdbcTemplate 类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

44. Spring 对 DAO 的支持

Spring 对数据访问对象（DAO）的支持旨在简化它和数据访问技术如 JDBC，Hibernate or JDO 结合使用。这使我们可以方便切换持久层。编码时也不用担心会捕获每种技术特有的异常。

45. 使用 Spring 通过什么方式访问 Hibernate?

在 Spring 中有两种方式访问 Hibernate：

控制反转 Hibernate Template 和 Callback。
继承 HibernateDAOSupport 提供一个 AOP 拦截器。

46. Spring 支持的 ORM

Spring 支持以下 ORM：

Hibernate
iBatis
JPA (Java Persistence API)
TopLink
JDO (Java Data Objects)
OJB

47.如何通过 HibernateDaoSupport 将 Spring 和 Hibernate 结合起来？

用 Spring 的 SessionFactory 调用 LocalSessionFactory。集成过程分三步：

配置 the Hibernate SessionFactory。
继承 HibernateDaoSupport 实现一个 DAO。
在 AOP 支持的事务中装配。

48. Spring 支持的事务管理类型

Spring 支持两种类型的事务管理：

编程式事务管理：这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
声明式事务管理：这意味着你可以将业务代码和事务管理分离，你只需用注解和 XML 配置来管理事务。

49. Spring 框架的事务管理有哪些优点？

它为不同的事务 API　如 JTA，JDBC，Hibernate，JPA 和 JDO，提供一个不变的编程模式。
它为编程式事务管理提供了一套简单的 API 而不是一些复杂的事务 API 如
它支持声明式事务管理。
它和 Spring 各种数据访问抽象层很好得集成。

50. 你更倾向用那种事务管理类型？

大多数 Spring 框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此更符合一个无侵入的轻量级容器的思想。声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

Spring 面向切面编程（AOP）

51.　解释 AOP

面向切面的编程，或 AOP，　是一种编程技术，允许程序模块化横向切割关注点，或横切典型的责任划分，如日志和事务管理。

52. Aspect 切面

AOP 核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组 API 提供横切功能。比如，一个日志模块可以被称作日志的 AOP 切面。根据需求的不同，一个应用程序可以有若干切面。在 Spring AOP 中，切面通过带有@Aspect 注解的类实现。

52. 在 Spring AOP 中，关注点和横切关注的区别是什么？

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。
横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志，安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

54. 连接点

连接点代表一个应用程序的某个位置，在这个位置我们可以插入一个 AOP 切面，它实际上是个应用程序执行 Spring AOP 的位置。

55. 通知

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过 SpringAOP 框架触发的代码段。

Spring 切面可以应用五种类型的通知：

before：前置通知，在一个方法执行前被调用。
after: 在方法执行之后调用的通知，无论方法执行是否成功。
after-returning: 仅当方法成功完成后执行的通知。
after-throwing: 在方法抛出异常退出时执行的通知。
around: 在方法执行之前和之后调用的通知。

56. 切点

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

57. 什么是引入?

引入允许我们在已存在的类中增加新的方法和属性。

58. 什么是目标对象?

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

59. 什么是代理?

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

60. 有几种不同类型的自动代理？

BeanNameAutoProxyCreator

DefaultAdvisorAutoProxyCreator

Metadata autoproxying

61. 什么是织入。什么是织入应用的不同点？

织入是将切面和到其他应用类型或对象连接或创建一个被通知对象的过程。

织入可以在编译时，加载时，或运行时完成。

62. 解释基于 XML Schema 方式的切面实现。

在这种情况下，切面由常规类以及基于 XML 的配置实现。

63. 解释基于注解的切面实现

在这种情况下(基于@AspectJ 的实现)，涉及到的切面声明的风格与带有 java5 标注的普通 java 类一致。

Spring 的 MVC

64. 什么是 Spring 的 MVC 框架？

Spring 配备构建 Web 应用的全功能 MVC 框架。Spring 可以很便捷地和其他 MVC 框架集成，如 Struts，Spring 的 MVC 框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

65. DispatcherServlet

Spring 的 MVC 框架是围绕 DispatcherServlet 来设计的，它用来处理所有的 HTTP 请求和响应。

66. WebApplicationContext

WebApplicationContext 继承了 ApplicationContext 并增加了一些 WEB 应用必备的特有功能，它不同于一般的 ApplicationContext ，因为它能处理主题，并找到被关联的 servlet。

67. 什么是 Spring MVC 框架的控制器？

控制器提供一个访问应用程序的行为，此行为通常通过服务接口实现。控制器解析用户输入并将其转换为一个由视图呈现给用户的模型。Spring 用一个非常抽象的方式实现了一个控制层，允许用户创建多种用途的控制器。

68. @Controller 注解

该注解表明该类扮演控制器的角色，Spring 不需要你继承任何其他控制器基类或引用 Servlet API。

69. @RequestMapping 注解

该注解是用来映射一个 URL 到一个类或一个特定的方法上。