Java 面试宝典

JavaSE 阶段

1. 遍历 map 的几种方法？

答:

```
public static void main(String[] args) {
 Map<String, String> map = new HashMap<String, String>();
 map.put("1", "value1");
 map.put("2", "value2");
 map.put("3", "value3");
 //第一种：普遍使用，二次取值(3 分)
 System.out.println("通过 Map.keySet 遍历 key 和 value：");
 for (String key : map.keySet()) {
  System.out.println("key= "+ key + " and value= " + map.get(key));
 }
 //第二种(3 分)
 System.out.println("通过 Map.entrySet 使用 iterator 遍历 key 和 value：");
 Iterator<Map.Entry<String, String>> it = map.entrySet().iterator();
 while (it.hasNext()) {
  Map.Entry<String, String> entry = it.next();
  System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());
 }
 //第三种：推荐，尤其是容量大时(2 分)
 System.out.println("通过 Map.entrySet 遍历 key 和 value");
 for (Map.Entry<String, String> entry : map.entrySet()) {
  System.out.println("key= " + entry.getKey() + " and value= " + entry.getValue());
 }

 //第四种(2 分)
 System.out.println("通过 Map.values()遍历所有的 value，但不能遍历 key");
 for (String v : map.values()) {
  System.out.println("value= " + v);
 }
}
```

2. hashmap 的特性？

答:

a、HashMap 是可以序列化的。是线程不安全的。(3 分)
b、HashMap 的底层主要是基于数组和链表实现的，它之所以有相当快的查询速度主要是因为它是通过计算散列码来决定存储位置的。(2 分)
c、HashMap 中主要是通过 key 的 hashCode 来计算 hash 值，然后通过 hash 值选择不同的数组来存储。只要 hashCode 相同，计算出来的 hash 值就一样，如果存储对象多了，就有可能不同的对象计算出来的 hash 值是相同的，这就出现了所谓的 hash 冲突，HashMap 的底层是通过链表来解决 hash 冲突的。具体就是把相同 hash 值的 HashMap，通过链表的形式进行存储，相当于存储的数组就是哈希表，数组的每个元素都是一个单链表的头结点，链表是用来解决 hash 冲突的，如果不同的 key 映射到了数组的同一位置，那么就将其放入单链表中。(3 分)
d、HashMap 的遍历方式比较单一，一般分两步走：(2 分)
1）. 获得 Entry 或 key 或 value 的 Set 集合；
2）. 通过 Iterator 迭代器遍历此 Set 集合。

3. Java 虚拟机中的内存模型？

答:

如下图所示，与物理机中 cpu 访问内存的结构类似，JVM 内存模型中，所有变量都存储在主内存(Main Memory)

每条线程有自己的工作内存（ Working Memory)，变量在不同线程间的传递需要通过主内存。

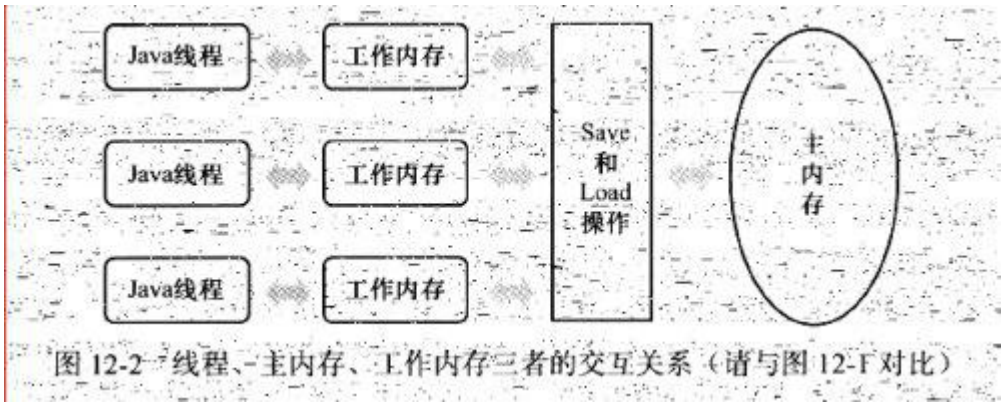


图 12-2 线程、主内存、工作内存三者的交互关系（请与图 12-1 对比）

如上所述，变量在线程间的传递需要工作内存和主内存间的交互。这种交互要遵循一定的一致性协议。

这里简单称为“交互协议”，交互协议具体体现为 8 中原子操作和一些操作规则，如下：

>8 种原子操作(5’)

lock：作用于主内存的变量，把一个变量标识为一条线程独占的状态；

1. unlock：作用于主内存的变量，把一个变量从一条线程独占的状态释放出来，释放后的变量才可以被其他线程上锁；

2. read：作用于主内存的变量，把一个变量从主内存传输到线程的工作内存，这一步仅仅是传输动作；

3. load：作用于工作内存的变量，把 read 中传输过来的变量放入工作内存的副本中；

4. use：作用于工作内存的变量，把工作内存中的一个变量传递给执行引擎，即线程在读取变量的值；

5. assign：作用于工作内存的变量，把一个从执行引擎接收到的值赋给工作内存的变量，即线程在写变量的值；

6. store：作用于工作内存的变量，把一个变量的值传输到主内存，这一步仅仅是传输动作；

7. write：作用于主内存的变量，把 store 过来的变量值放入主内存的变量中。

>操作规则(5’)

1. 不允许 read 和 load，store 和 write 操作之一单独出现；

2. 不允许一个线程丢弃它的最近的 assign 操作，即变量在工作内存中改变了之后必须把该变量同步到主内存中；

3. 不允许一个线程无原因地（没有发生过 assign 操作）把数据从线程的工作内存同步到主内存中；

4. 一个新的变量只能在主内存中“诞生”，不允许在工作内存中直接使用一个未被初始化的变量，也就是说变量在 use，store 之前必须先执行了 assign 和 load 操作；

5. 一个变量在同一个时刻只允许一条线程对其进行 lock 操作，但 lock 操作可以被同一条线程重复执行多次，而且，多次 lock 后，必须执行相同次数的 unlock 才能解锁变量；

6. 如果对一个变量执行 lock 操作，那将会清空工作内存中此变量的值，在执行引擎使用变革变量前，需要重新执行 load 或 assign 操作初始化变量；

7. 如果一个变量事先没有被 lock 操作锁定，那就不允许对它执行 unlock 操作，也不允许去 unlock 一个被其他线程锁定住的变量。

8. 对一个变量执行 unlock 操作前，必须先把此变量同步回主内存中（执行 store, write 操作）。

此外，volatile 关键字比较特殊，需要单独说明：

volatile 关键字具备两种语义特性：

一是保证变量对所有线程的“可见性”，即对于 volatile 变量，当一条线程修改了这个变量的值，新值对于其他线程来说是可以立即得知的。

二是禁止指令重排序优化，

并发环境下访问内存是否安全。

4. 简单说说 Java 中的异常处理机制的简单原理和应用。

答:

Java 中的异常处理机制的简单原理和应用 ：

java 中 Throwable 这个类可以被作为异常抛出的类,继承它的分为异常 Exception 和错误 Error.

Exception 表示程序需要捕捉和处理的的异常;(1 分)

Error 表示系统级别的错误和程序无需处理的。(1 分)

我们所需要关心的是 Exception. Execption 可以分为 java 标准定义的异常和程序员自定义异常 2 种.(2 分)

（1）一种是当程序违反了 java 语规则的时候,JAVA 虚拟机就会将发生的错误表示为一个异常.这里语法规则指的是 JAVA 类库内置的语义检查。

（2）另一种情况就是 JAVA 允许程序员扩展这种语义检查，程序员可以创建自己的异常，并自由选择在何时用 throw 关键字引发异常。所有的异常都是 Thowable 的子类。

异常处理是与程序执行是并行的.

```
Try{
    //可能发现异常的语句块(2 分)
}catch(异常类型,e){
   //发生异常时候的执行语句块(2 分)
} finnally{
  //不管是否发生异常都执行的语句块(2 分)
}
```

以下是一个自定义异常测试类:

```java
package code;
class MyException extends Exception
{
   public void f()
   {
       System.out.println("this is my Exception!!");
    }
}
public class ExceptionTestTwo {
    private int i = 0;
     private int j;
      ExceptionTestTwo(int x) throws MyException
     {
        f2();
         j = x / i;
       }
      public void f2() throws MyException
     {
         System.out.println("this is My first Exception!!");
         throw new MyException();
      }
```

```
22    public static void main(String[] args)
23   {
24       try {
25               new ExceptionTestTwo(9);
26           } catch (MyException e2) {
27                e2.f();
28            } catch (Exception e) {
29                e.printStackTrace();
30            } finally {
31                 System.out.println("Finally is first Exception!!");
32             }
33      try {
34          throw new MyException();
35        } catch (MyException e1) {
36            e1.f();
37         } finally {
38            System.out.println("Finally is second Exception!!");
39        }
40    }
41}
```

5. 创建线程的几种方式？

答:

编写多线程程序是为了实现多任务的并发执行，从而能够更好地与用户交互。一般有四种方法，Thread,Runnable,Callable,使用 Executor 框架来创建线程池。

Runnable 和 Callable 的区别是，
(1)Callable 规定的方法是 call(),Runnable 规定的方法是 run().
(2)Callable 的任务执行后可返回值，而 Runnable 的任务是不能返回值得
(3)call 方法可以抛出异常，run 方法不可以
(4)运行 Callable 任务可以拿到一个 Future 对象，表示异步计算的结果。它提供了检查计算是否完成的方法，以等待计算的完成，并检索计算的结果。通过 Future 对象可以了解任务执行情况，可取消任务的执行，还可获取执行结果。

1、通过实现 Runnable 接口来创建 Thread 线程：(3 分)

步骤 1：创建实现 Runnable 接口的类：

```
class SomeRunnable implements Runnable
{
    public void run()
    {
     //do something here
    }
}
```

步骤 2：创建一个类对象：

```
Runnable oneRunnable = new SomeRunnable();
```

步骤 3：由 Runnable 创建一个 Thread 对象：

```
Thread oneThread = new Thread(oneRunnable);
```

步骤 4：启动线程：

```
oneThread.start();
```

至此，一个线程就创建完成了。

注释：线程的执行流程很简单，当执行代码 oneThread.start();时，就会执行 oneRunnable 对象中的 void run();方法，

该方法执行完成后，线程就消亡了。

与方法 1 类似，通过实现 Callable 接口来创建 Thread 线程：(3 分)

其中，Callable 接口（也只有一个方法）定义如下：

```
public interface Callable<V>
{
    V call() throws Exception;
}
```

步骤 1：创建实现 Callable 接口的类 SomeCallable<Integer>（略）；

步骤 2：创建一个类对象：

```
Callable<Integer> oneCallable = new SomeCallable<Integer>();
```

步骤 3：由 Callable<Integer>创建一个 FutureTask<Integer>对象：

```
FutureTask<Integer> oneTask = new FutureTask<Integer>(oneCallable);
```

注释：FutureTask<Integer>是一个包装器，它通过接受 Callable<Integer>来创建，它同时实现了 Future 和 Runnable 接口。

步骤 4：由 FutureTask<Integer>创建一个 Thread 对象：

```
Thread oneThread = new Thread(oneTask);
```

步骤 5：启动线程：

```
oneThread.start();
```

至此，一个线程就创建完成了。

3、通过继承 Thread 类来创建一个线程：(2 分)

步骤 1：定义一个继承 Thread 类的子类：

```
class SomeThead extends Thraad
{
    public void run()
    {
     //do something here
    }
}
```

步骤 2：构造子类的一个对象：

```
SomeThread oneThread = new SomeThread();
```

步骤 3：启动线程：

oneThread.start();

至此，一个线程就创建完成了。

注释：这种创建线程的方法不够好，主要是因为其涉及运行机制问题，影响程序性能。

4.使用 Executor 框架来创建线程池(2 分)

在 Java 5 之后，并发编程引入了一堆新的启动、调度和管理线程的 API。Executor 框架便是 Java 5 中引入的，其内部使用了线程池机制，它在 java.util.cocurrent 包下，通过该框架来控制线程的启动、执行和关闭，可以简化并发编程的操作。因此，在 Java 5 之后，通过 Executor 来启动线程比使用 Thread 的 start 方法更好，除了更易管理，效率更好（用线程池实现，节约开销）外，还有关键的一点：有助于避免 this 逃逸问题——如果我们在构造器中启动一个线程，因为另一个任务可能会在构造器结束之前开始执行，此时可能会访问到初始化了一半的对象用 Executor 在构造器中。

Executor 框架包括：线程池，Executor，Executors，ExecutorService，CompletionService，Future，Callable 等。 Executor 接口中之定义了一个方法 execute（Runnable command），该方法接收一个 Runable 实例，它用来执行一个任务，任务即一个实现了 Runnable 接口的类。ExecutorService 接口继承自 Executor 接口，它提供了更丰富的实现多线程的方法，比如，ExecutorService 提供了关闭自己的方法，以及可为跟踪一个或多个异步任务执行状况而生成 Future 的方法。 可以调用 ExecutorService 的 shutdown（）方法来平滑地关闭 ExecutorService，调用该方法后，将导致 ExecutorService 停止接受任何新的任务且等待已经提交的任务执行完成(已经提交的任务会分两类：一类是已经在执行的，另一类是还没有开始执行的)，当所有已经提交的任务执行完毕后将会关闭 ExecutorService。因此我们一般用该接口来实现和管理多线程。ExecutorService 的生命周期包括三种状态：运行、关闭、终止。创建后便进入运行状态，当调用了 shutdown（）方法时，便进入关闭状态，此时意味着 ExecutorService 不再接受新的任务，但它还在执行已经提交了的任务，当素有已经提交了的任务执行完后，便到达终止状态。如果不调用 shutdown（）方法，ExecutorService 会一直处在运行状态，不断接收新的任务，执行新的任务，服务器端一般不需要关闭它，保持一直运行即可。Executors 提供了一系列工厂方法用于创先线程池，返回的线程池都实现了 ExecutorService 接口。

public static ExecutorService newFixedThreadPool(int nThreads)

创建固定数目线程的线程池。

public static ExecutorService newCachedThreadPool()

创建一个可缓存的线程池，调用 execute 将重用以前构造的线程（如果线程可用）。如果现有线程没有可用的，则创建一个新线 程并添加到池中。终止并从缓存中移除那些已有 60 秒钟未被使用的线程。

public static ExecutorService newSingleThreadExecutor()

创建一个单线程化的 Executor。

public static ScheduledExecutorService newScheduledThreadPool(int corePoolSize)

创建一个支持定时及周期性的任务执行的线程池，多数情况下可用来替代 Timer 类。

6. 谈谈你对垃圾回收机制的了解？

答:

垃圾回收机制，顾名思义，所以该机制可以使得 Java 编程自动释放内存空间，减轻了编程的负担，同时，这也是 Java 语言安全性策略的一个重要部份。当然，凡是有利必有弊，由于 Java 虚拟机必须追踪运行程序中有用的对象，最终释放没用的对象。这一过程需要花费处理器的时间，同时也因为垃圾回收算法的不完备性，使得垃圾回收机制的开销影响了程序的性能。不过，由于机器的进步和算法的改进，这些问题会慢慢地被解决的。

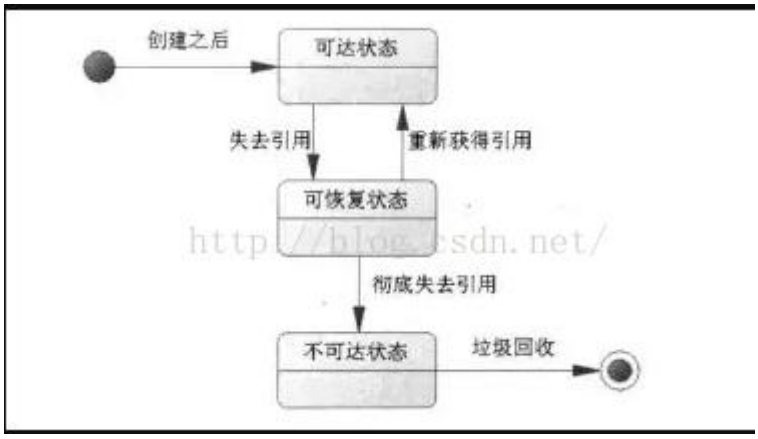
一.垃圾回收机制具有以下的特点：(3 分)

1、 垃圾回收机制只负责回收堆内存，不会回收任何物理资源

2、 程序无法精确控制垃圾回收的进行，会在合适的时候进行

3、 在垃圾回收机制回收的任何对象之前，总会先调用它的 finalize()方法需要强调的一点:垃圾回收回收的是无任何引用的对象占据的内存空间而不是对象本身

二、对象在内存中的状态(2 分)



三、垃圾回收机制中的算法(2 分)

Java 并没有明确地说明 JVM 使用哪种垃圾回收算法，但是任何一种垃圾回收算法一般要做 2 件基本的事情：(1)发现无用信息对象；(2)回收被无用对象占用的内存空间，使该空间可被程序再次使用。垃圾回收算法有很多种，如引用计数算法、可达性分析算法……我对 JVM 也没有过多的学习，所以这里也没法进行深入的探讨。

四、强制垃圾回收(2 分)

强制系统垃圾回收有两种方式：

1、 调用 System 类的 gc()静态方法：System.gc()

2、 调用 Runtime 对象的 gc()实例：Runtime.getRuntime().gc()

强调：所谓强制回收，其实也仅仅是一个建议。JVM 接受这个消息后，并不是立即做垃圾回收，而只是对几个垃圾回收算法做了加权，使垃圾回收操作容易发生，或提早发生，或回收较多而已。

finalize 方法(1 分)

为什么要使用这个方法呢？这是一个默认机制，当要回收某个内存之前，通常需要调用适当的方法来清理资源，而这个方法就是 finalize 方法。它的原型为：protected void finalize() throws Throwable。在 finalize()方法返回之后，对象消失，垃圾收集开始执行。原型中的 throws Throwable 表示它可以抛出任何类型的异常。

7. 说说 HashCode()、equals()的区别？

答:

在 JAVA 语言中，判断两个对象是否相等，一般有两种方法，一种是 hashcode()，另一种是 equals(),这两个方法在判断准确性和效率上有很大的区别，下面章节详细说明：

hashCode()方法和 equals()方法的作用其实一样，在 Java 里都是用来对比两个对象是否相等一致，那么 equals()既然已经能实现对比的功能了，为什么还要 hashCode()呢？

因为重写的 equals( )里一般比较全面比较复杂，这样效率就比较低，而利用 hashCode()进行对比，则只要生成一个 hash 值进行比较就可以了，效率很高，那么 hashCode()既然效率这么高为什么还要 equals()呢？

因为 hashCode()并不是完全可靠，有时候不同的对象他们生成的 hashcode 也会一样（生成 hash 值得公式可能存在的问题），所以 hashCode()只能说是大部分时候可靠，并不是绝对可靠，所以我们可以得出：(5 分)

equals()相等的两个对象他们的 hashCode()肯定相等，也就是用 equal()对比是绝对可靠的。

2.hashCode()相等的两个对象他们的 equals()不一定相等，也就是 hashCode()不是绝对可靠的。

所有对于需要大量并且快速的对比的话如果都用 equal()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用 hashCode()去对比，如果 hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用 equals()去再对比了）,如果 hashCode()相同，此时再对比他们的 equals()，如果 equals()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

这种大量的并且快速的对象对比一般使用的 hash 容器中，比如 hashset,hashmap,hashtable 等等，比如 hashset 里要求对象不能重复，则他内部必然要对添加进去的每个对象进行对比，而他的对比规则就是像

上面说的那样，先 hashCode()，如果 hashCode()相同，再用 equals()验证，如果 hashCode()都不同，则肯定不同，这样对比的效率就很高了。(5 分)

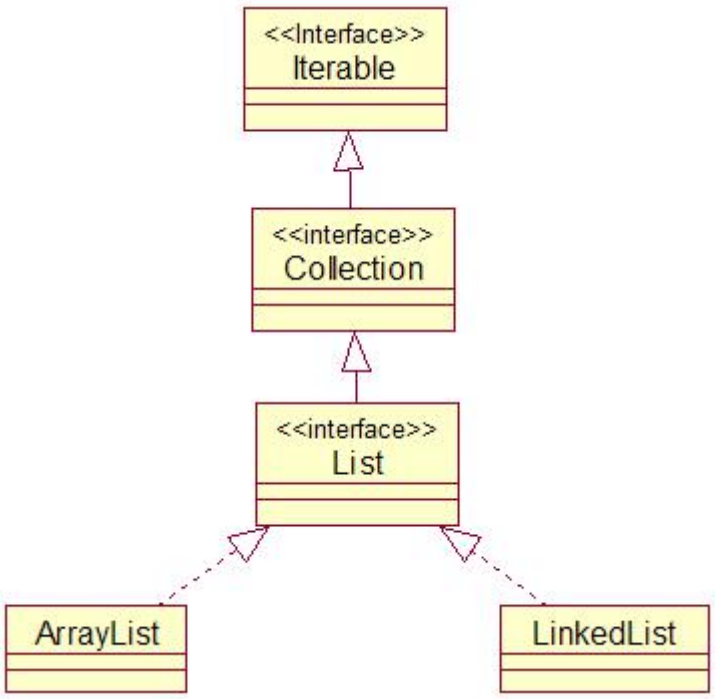
8. LinkedList 和 ArrayList 区别？

答:
ArrayList 和 LinkedList

共性：ArrayList 与 LinkedList 都是 List 接口的实现类，因此都实现了 List 的所有未实现的方法，只是实现的方式有所不同。(3 分)

区别：List 接口的实现方式不同(7 分)

ArrayList 实现了 List 接口，以数组的方式来实现的，因此对于快速的随机取得对象的需求，使用 ArrayList 实现执行效率上会比较好。LinkedList 是采用链表的方式来实现 List 接口的，因此在进行 insert 和 remove 动作时效率要比 ArrayList 高。适合用来实现 Stack(堆栈)与 Queue(队列)。



9. String，StringBuilder，StringBuffer 三者的区别？

答:

1、String，StingBuffer，StringBuilder 是字符串变量还是常量？(3 分)

String ----> 字符串常量

StringBuffer ----> 字符串变量（线程安全的）

StringBuilder ----> 字符串变量（非线程安全的）

String 是字符串常量，对于这个叫法大家都可能有一个疑问，String 字符串不是变量吗？怎么叫常量？(3 分)

我看首先看一下官方的解释：

Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared

官方解释说：String 是一个常量，他的值在创建之后不能改变，字符串缓冲区支持修改，因为字符串对象是不可变的，所以支持共享。

举个例子：

```
String s = "abc";

s = s + 1;

System.out.println(s);
```

代码解析：首先我们定义一个变量 s，给他赋值“abc”，然后在创建一个新的对象 s 用来执行 s=s+1;也就是说我们之前的 s 并没有发生变化。这也说明了 String 类型是一个不可改变的对象，每次操作 String 字符串，实际上是不断的创建新的对象，而原来的对象变成了垃圾被 GC 回收，从而 String 字符串的效率也是最低的。

2、String，StingBuffer，StringBuilder 效率如何呢？(2 分)

从高到底的顺序依次是：StringBuilder > StingBuffer > String

由上知道了 String 是字符串常量，所以他的效率自然而然是最低的。对于 StringBuffer 和 StringBuilder 它们属于变量，是可以改变的对象，每次对字符串的操作，实际上实在一个对象上操作，所以效率更高一些。StringBuffer 是线程安全的，考虑到安全问题，相对他的性能会更低一点。由此知道了从效率的角度看，StringBuilder 最高，其次是 StringBuffer，最后是 String 字符串常量。

总结：当多个线程使用字符串缓冲区时，使用 StringBuffer 保证正确的操作，如果是单线程的，建议使用 StringBuiler 效率更高一些。(2 分)

10. 是否可以从一个 static 方法内部发出对非 static 方法的调用？

答:

不可以。(2 分)

因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，(2 分)

而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法关联到那个对象上的呢？这个逻辑无法成立，所以，一个 static 方法内部不能发出对非 static 方法的调用。(2 分)

static 方法是静态方法，是属于类的方法,(1’)非 static 方法是属于对象的方法(1’)，所以在 static 方法中想要调用非 static 方法，要先新创建一个对象，再有这个对象来调用非 static 方法。(2 分)

11. java 中 sleep 和 wait 的区别？

答:

首先，要记住这个差别，“sleep 是 Thread 类的方法,wait 是 Object 类中定义的方法”。尽管这两个方法都会影响线程的执行行为，但是本质上是有区别的。(2 分)

Thread.sleep 不会导致锁行为的改变，如果当前线程是拥有锁的，那么 Thread.sleep 不会让线程释放锁。如果能够帮助你记忆的话，可以简单认为和锁相关的方法都定义在 Object 类中，因此调用 Thread.sleep 是不会影响锁的相关行为。(2 分)

Thread.sleep 和 Object.wait 都会暂停当前的线程，对于 CPU 资源来说，不管是哪种方式暂停的线程，都表示它暂时不再需要 CPU 的执行时间。OS 会将执行时间分配给其它线程。区别是，调用 wait 后，需要别的线程执行 notify/notifyAll 才能够重新获得 CPU 执行时间。(2 分)

线程的状态参考 Thread.State 的定义。新创建的但是没有执行 (还没有调用 start())的线程处于“就绪”，或者说 Thread.State.NEW 状态。(2 分)

Thread.State.BLOCKED（阻塞）表示线程正在获取锁时，因为锁不能获取到而被迫暂停执行下面的指令，一直等到这个锁被别的线程释放。BLOCKED 状态下线程，OS 调度机制需要决定下一个能够获取锁的线程是哪个，这种情况下，就是产生锁的争用，无论如何这都是很耗时的操作。(2 分)

12. java 中 sleep 和 wait 的区别？

其实从字面上就可以理解了 ,sleep 指定一段时间后 ,自己就会继续执行 ,但是 wait 会在使用 notify 后(其他地方通知), 才会继续执行。

详细一点：

对于 sleep()方法，我们首先要知道该方法是属于 Thread 类中的。而 wait()方法，则是属于 Object 类中的。

sleep()方法导致了程序暂停执行指定的时间，让出 cpu 该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。

在调用 sleep()方法的过程中，线程不会释放对象锁。

而当调用 wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用 notify()方法后本线程才进入对象锁定池准备获取对象锁进入运行状态。

13. 实现一个线程有哪几种方式,各有什么优缺点,比较常用的是那种？

1.继承 Thread 类

2.实现 Runnable 接口

3.实现 Callable 接口

4.线程池方式

优缺点：

1.继承 Thread 类

优点：代码简单 。 缺点：该类无法集成别的类。

2.实现 Runnable 接口

优点：继承其他类。 同一实现该接口的实例可以共享资源。

缺点：代码复杂

3.实现 Callable

优点：可以获得异步任务的返回值

4. 线程池：实现自动化装配，易于管理，循环利用资源。

14. String s = new String("xyz");创建了几个 StringObject？是否可以继承 String 类？

两个或一个都有可能，”xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量”xyz”不管出现多少遍，都是缓冲区中的那一个。NewString 每写一遍，就创建一个新的对象，它使用常量”xyz”对象的内容来创建出一个新 String 对象。如果以前就用过’xyz’，那么这里就不会创建”xyz”了，直接从缓冲区拿，这时创建了一个 StringObject；但如果以前没有用过"xyz"，那么此时就会创建一个对象并放入缓冲区，这种情况它创建两个对象。至于 String 类是否继承，答案是否定的，因为 String 默认 final 修饰，是不可继承的。

15. String s = "Hello";s = s + "world!";这两行代码执行后，原始的 String 对象中的内容到底变了没有？

答案是肯定的。因为 String 被设计为不可变（immutable）类，所以它的所有对象都是不可变的对象。在这段代码中，s 原先指向一个 String 对象，内容是 hello，然后我们对 s 进行+操作，那么 s 所指向的那个对象是否发生了变化呢？答案是没有，这时，s 不再指向原来的那个对象了，而指向另一个 String 对象，内容为 hello world ，原来的那个对象还存在于内存中，只是 s 这个引用变量不再指向它了。

16. 动态代理 的 2 种方式以及区别？

一般而言，动态代理分为两种，一种是 JDK 反射机制提供的代理，另一种是 CGLIB 代理。在 JDK 代理，必须提供接口，而 CGLIB 则不需要提供接口，在 Mybatis 里两种动态代理技术都已经使用了，在 Mybatis 中通常在延迟加载的时候才会用到 CGLIB 动态代理。

17. 多线程解决同步问题的方式？

wait()/notify()方法

await()/signal()方法

BlockingQueue 阻塞队列方法

18. Hashtable 与 HashMap 有什么不同之处？

a.Hashtable 是继承自陈旧的 Dictionary 类的，HashMap 继承自 AbstractMap 类同时是 Java 1.2 引进的 Map 接口的一个实现。

b.也许最重要的不同是 Hashtable 的方法是同步的，而 HashMap 的方法不是。这就意味着，然你可以不用采取任何特殊的行为就可以在一个 多线程的应用程序中用一个 Hashtable ,但你必须同样地为一个 HashMap 提供外同步。一个方便的方法就是利用 Collections 类的静态的 synchronizedMap()方法，它创建一个线程安全的 Map 对象，并把它作为一个封装的对象来返回。这个对象的方法可以让你同步访问潜在的 HashMap。这么做的结果就是当你不需要同步时，你不能切断 Hashtable 中的同步（比如在一个单线程的应用程序中），而且同步增加了很多处理费用。

c.第三点不同是，只有 HashMap 可以让你将空值作为一个表的条目的 key 或 value。HashMap 中只有一条记录可以是一个空的 key，但任意数量的条目可以是空的 value。这就是说，如果在表中没有发现搜索键 ,或者如果发现了搜索键 ,但它是一个空的值 ,那么 get()将返回 null。如果有必要 ,用 containKey()方法来区别这两种情况。

d.HashMap 去掉了 Hashtable 的 contains 方法，保留了 containsValue 和 containsKey 方法

19. 单例中的懒汉和恶汉模式的区别？

1、饿汉式：在程序启动或单件模式类被加载的时候，单件模式实例就已经被创建。

2、懒汉式：当程序第一次访问单件模式实例时才进行创建。 如何选择：如果单件模式实例在系统中经常会被用到，饿汉式是一个不错的选择。

20. 类加载机制有了解嘛？

加载是类加载过程中的一个阶段，这个阶段会在内存中生成一个代表这个类的 java.lang.Class 对象，作为方法区这个类的各种数据的入口。注意这里不一定非得要从一个 Class 文件获取，这里既可以从 ZIP 包中读取（比如从 jar 包和 war 包中读取），也可以在运行时计算生成（动态代理），也可以由其它文件生成（比如将 JSP 文件转换成对应的 Class 类）。

这一阶段的主要目的是为了确保 Class 文件的字节流中包含的信息是否符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。

准备阶段是正式为类变量分配内存并设置类变量的初始值阶段，即在方法区中分配这些变量所使用的内存空间。

javaWEB 阶段

1、Jquery 常用选择器都有哪些？

分数标注：答出一个得 2 分，10 分满分

基本选择器：

ID 选择器：$(“#id”)、

类选择器：$(“.class”)、

元素选择器： $(“元素”)、

通配符选择器： $(“*”)、

并列选择器: $(“#id,.class”)

层级选择器：

某个元素下的所有子孙节点 （空格）: $("form input")选择所有的 form 元素中的 input 元素 ；

表单选择器:

$(":input") 选择所有的表单输入元素，包括 input, textarea, select 和 button

$(":text") 选择所有的 text input 元素

$(":checkbox") 选择所有的 checkbox input 元素

基本过滤选择器：

: eq(index) 选择所有的 td 元素中序号为 index 值的那个 td 元素

等等...

2、ajax 书写方式及内部主要参数都有哪些？

分数标注：答出一项得 2 分，10 分满分

书写方式 :(常用 jquery 的 ajax 书写方式)

```
$.ajax({
    url:"http://www.microsoft.com",//请求的 url 地址（2 分）
    dataType:"json",          //返回 json、xml、html、jsonp 等数据格式（2 分）
    async:true,                    //请求是否异步，默认为异步（2 分）
    data:{"id":"value"},          //参数值（2 分）
    type:"GET",                    //请求方式 get 或者 post（2 分）
    success:function(req){
        //请求成功时处理、回调函数（2 分）
    },
    error:function(){
        //请求出错处理、回调函数（2 分）
    }
});
```

3、ajax 提交请求 默认是 异步还是同步 怎么改成同步？

分数标注：答出参数及值，得满分，否则 0 分。

Ajax 请求默认的都是异步的

如果想同步 async 参数设置为 false 就可以（默认是 true）

async 参数，要求为 Boolean 类型的参数，默认设置为 true，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。注意，同步请求将锁住浏览器，用户其他操作必须等待请求完成才可以执行。

4、一次完整的 http 请求是什么样的?

分数标注：满分 10 分

HTTP 通信机制是在一次完整的 HTTP 通信过程中，Web 浏览器与 Web 服务器之间将完成下列 7 个步骤：(1 分)

4.1. 建立 TCP 连接（1 分）

在 HTTP 工作开始之前，Web 浏览器首先要通过网络与 Web 服务器建立连接，该连接是通过 TCP 来完成的，该协议与 IP 协议共同构建 Internet，即著名的 TCP/IP 协议族，因此 Internet 又被称作是 TCP/IP 网络。HTTP 是比 TCP 更高层次的应用层协议，根据规则，只有低层协议建立之后才能进行更高层协议的连接，因此，首先要建立 TCP 连接，一般 TCP 连接的端口号是 80。

4.2. Web 浏览器向 Web 服务器发送请求命令（1 分）

一旦建立了 TCP 连接，Web 浏览器就会向 Web 服务器发送请求命令。例如：GET/sample/hello.jsp HTTP/1.1。

4.3. Web 浏览器发送请求头信息（1 分）

浏览器发送其请求命令之后，还要以头信息的形式向 Web 服务器发送一些别的信息，之后浏览器发送了一空白行来通知服务器，它已经结束了该头信息的发送。

4.4. Web 服务器应答（1 分）

客户机向服务器发出请求后，服务器会客户机回送应答， HTTP/1.1 200 OK ，应答的第一部分是协议的版本号和应答状态码。

4.5. Web 服务器发送应答头信息（1 分）

正如客户端会随同请求发送关于自身的信息一样，服务器也会随同应答向用户发送关于它自己的数据及被请求的文档。

4.6. Web 服务器向浏览器发送数据（1 分）

Web 服务器向浏览器发送头信息后，它会发送一个空白行来表示头信息的发送到此为结束，接着，它就以 Content-Type 应答头信息所描述的格式发送用户所请求的实际数据。

4.7. Web 服务器关闭 TCP 连接（1 分）

一般情况下，一旦 Web 服务器向浏览器发送了请求数据，它就要关闭 TCP 连接，然后如果浏览器或者服务器在其头信息加入了这行代码：Connection:keep-alive

TCP 连接在发送后将仍然保持打开状态，于是，浏览器可以继续通过相同的连接发送请求。保持连接节省了为每个请求建立新连接所需的时间，还节约了网络带宽。(1 分)

4.8、表达流畅(1 分)

5、分别说出 http,https,ftp,talnet 的默认端口

分数标注：满分 10 分

（1）、HTTP 协议代理服务器常用端口号：80 （2.5 分）

（2）、HTTPS 服务器，默认端口号为 443/tcp，443/udp （2.5 分）

（3）、FTP（文件传输）协议代理服务器常用端口号：21 （2.5 分）

（4）、Telnet（远程登录）协议代理服务器常用端口：23 （2.5 分）

6、常见的 http 返回状态码都有哪些?

分数标注：答对一个得两分，满分 10 分。

1XX	Informational	信息性状态码，表示接受的请求正在处理
2XX	Success	成功状态码，表示请求正常处理完毕
3XX	Redirection	重定向状态码，表示需要客户端需要进行附加操作
4XX	Client Error	客户端错误状态码，表示服务器无法处理请求
5XX	Server Error	服务器错误状态码，表示服务器处理请求出错

举例：

200 OK：

响应成功；

301 Movied Permanently：

永久性重定向；

302 Found:

临时性重定向；

400 Bad Request：

表示该请求报文中存在语法错误，导致服务器无法理解该请求。

403 Forbidden：

该状态码表明对请求资源的访问被服务器拒绝了。未获得文件系统的访问权限，访问权限出现某些问题，从未授权的发送源 IP 地址试图访问等情况都可能发生 403 响应。

404 Not Found：

:该状态码表明服务器上无法找到指定的资源。

500 Internal Server Error：

该状态码表明服务器端在执行请求时发生了错误。

503 Service Unavailable：

该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

7、XML 的解析方式有哪些？

分数标注：答对一个得两分，满分 10 分。

Java 原生的 XML 解析方法：

一、DOM 解析：一次性将文档加载到内存.形成树形结构。（2 分）

（1）.（2 分）原理是：首先在内存中创建一个 Document 对象，然后把 XML 文档读取进来赋值给这

个 dom 对象。由于 dom 对象是基于树结构的，所以对 dom 对象进行遍历即可。对内存中的 dom 对象可以进行查询、修改、删除操作，还可以写回原 XML 文档保存修改。

（2）、优缺点（1 分）

优点：

a、由于整棵树在内存中，因此可以对 xml 文档随机访问

b、可以对 xml 文档进行修改操作

缺点：

a、整个文档必须一次性解析完

b、由于整个文档都需要载入内存，对于大文档成本高

二、SAX 解析：事件驱动的方式，边读边解析（2 分）

（1）、（2 分）原理：通过 parse(file,listener)函数用一个 listener 对 xml 文件进行查找，按顺序读取文档，遍历每个标签，当发现目标标签时，读取标签的属性、结点值等信息并返回。

（2）、优缺点（1 分）

优点：

a、无需将整个 xml 文档载入内存，因此消耗内存少

b、可以继承 ContentHandler 创建多个执行不同查询的 listener 进行解析操作

缺点：

a、不能随机的访问 xml 中的节点

b、不能修改文档

c、查询依次就要对 XML 文档从头到尾遍历一次

8、jsp 和 servlet 有哪些相同点和不同点，他们之间的联系是什么？

分数标注：答对一个得两分，满分 10 分。

8.1、JSP 本身就是一个 Servlet.（2 分）

8.2、JSP 的执行原理:JSP 也会被翻译成 Servlet.将 Servlet 编译成一个 class 文件。（2 分）

(jsp 本质就是 servlet,jvm 只能识别 java 的类 ,不能识别 jsp 代码 ,web 容器将 jsp 的代码编译成 jvm 能够识别的 java 类)

8.3、jsp 更擅长表现于页面显示，servlet 更擅长于逻辑控制（2 分）

8.4、JSP = JSP 自身东西 + HTML + Java 代码.（2 分）

（jsp 是 servlet 的一种简化，使用 jsp 只需要完成程序员需用输出到客户端的内容，jsp 中的 java 脚本如何镶嵌到一个类中，由 jsp 容器完成，而 servlet 则是个完整的 java 类，这个类的 service 方法用于生成对客户端的响应

）

8.5、setvlet 中没有内置对象，jsp 中的内置对象都是必须通过 HttpServletRequest 对象，HttpServletResponse 对象及 HttpServlet 对象得到（2 分）

9、JSP 的九大内置对象及作用分别是什么？

分数标注：满分 10 分。

JSP 中一共预先定义了 9 个这样的对象，分别为：request、response、session、application、out、pagecontext、config、page、exception（1 分）

9.1、request 对象（1 分）

request 对象是 javax.servlet.httpServletRequest 类型的对象。 该对象代表了客户端的请求信息，主要用于接受通过 HTTP 协议传送到服务器的数据。（包括头信息、系统信息、请求方式以及请求参数等）。request 对象的作用域为一次请求。

9.2、response 对象（1 分）

response 代表的是对客户端的响应，主要是将 JSP 容器处理过的对象传回到客户端。response 对象也具有作用域，它只在 JSP 页面内有效。

9.3、session 对象（1 分）

session 对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个 session 对象，用于保存该用户的信息，跟踪用户的操作状态。session 对象内部使用 Map 类来保存数据，因此保存数据的格式为 “Key/value”。 session 对象的 value 可以使复杂的对象类型，而不仅仅局限于字符串类型。

9.4、application 对象（1 分）

application 对象可将信息保存在服务器中，直到服务器关闭，否则 application 对象中保存的信息会在整个应用中都有效。与 session 对象相比，application 对象生命周期更长，类似于系统的“全局变量”。

9.5、out 对象（1 分）

out 对象用于在 Web 浏览器内输出信息，并且管理应用服务器上的输出缓冲区。在使用 out 对象输出数据时，可以对数据缓冲区进行操作，及时清除缓冲区中的残余数据，为其他的输出让出缓冲空间。待数据输出完毕后，要及时关闭输出流。

9.6、pageContext 对象（1 分）

pageContext 对象的作用是取得任何范围的参数，通过它可以获取 JSP 页面的 out、request、reponse、session、application 等对象。pageContext 对象的创建和初始化都是由容器来完成的，在 JSP 页面中可以直接使用 pageContext 对象。

9.7、config 对象（1 分）

config 对象的主要作用是取得服务器的配置信息。通过 pageConext 对象的 getServletConfig() 方法可以获取一个 config 对象。当一个 Servlet 初始化时，容器把某些信息通过 config 对象传递给这个 Servlet。 开发者可以在 web.xml 文件中为应用程序环境中的 Servlet 程序和 JSP 页面提供初始化参数。

9.8、page 对象（1 分）

page 对象代表 JSP 本身，只有在 JSP 页面内才是合法的。 page 隐含对象本质上包含当前 Servlet 接口引用的变量，类似于 Java 编程中的 this 指针。

9.9、exception 对象（1 分）

exception 对象的作用是显示异常信息，只有在包含 isErrorPage="true" 的页面中才可以被使用，在一般的 JSP 页面中使用该对象将无法编译 JSP 文件。excepation 对象和 Java 的所有对象一样，都具有系统提供的继承结构。exception 对象几乎定义了所有异常情况。在 Java 程序中，可以使用 try/catch 关键字来处理异常情况；如果在 JSP 页面中出现没有捕获到的异常，就会生成 exception 对象，并把 exception 对象传送到在 page 指令中设定的错误页面中，然后在错误页面中处理相应的 exception 对象。

10、JSP 常用的动作标签及作用

作用：简化代码的编写,尽量要少在 JSP 中使用<%%>，利用 JSP 动作可以动态地插入文件、重用 JavaBean 组件、把用户重定向到另外的页面、为 Java 插件生成 HTML 代码。（2 分）

动作元素只有一种语法，它符合 XML 标准：

```
<jsp:action_name attribute="value" />
```

以下标签，讲出一个给 2 分：

常用动作标签：

jsp:include：在页面被请求的时候引入一个文件。（2 分）

jsp:useBean：寻找或者实例化一个 JavaBean。（2 分）

jsp:setProperty：设置 JavaBean 的属性。（2 分）

jsp:getProperty：输出某个 JavaBean 的属性。（2 分）

jsp:forward：把请求转到一个新的页面。

jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记。

jsp:element：定义动态 XML 元素

jsp:attribute：设置动态定义的 XML 元素属性。

jsp:body：设置动态定义的 XML 元素内容。

jsp:text：在 JSP 页面和文档中使用写入文本的模板

11、JSP 四大作用域及请求范围

分数标注：满分 10 分,答出一个酌情给 2-3 分。

a、page 是代表一个页面相关的对象和属性。一个页面由一个编译好的 java servlet 类（可以带有 include 指令，但不可以带有 include 动作）表示。这既包括 servlet 又包括编译成 servlet 的 jsp 页面。

b、request 是代表与 web 客户机发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 web 组件（由于 forware 指令和 include 动作的关系）

c、session 是代表与用于某个 web 客户机的一个用户体验相关的对象和属性。一个 web 回话也可以经常跨域多个客户机请求。

d、application 是代表与整个 web 应用程序相关的对象和属性。这实质上是跨域整个 web 应用程序，包括多个页面、请求和回话的一个全局作用域。

12、servlet 的生命周期及常用方法？

分数标注：满分 10 分,答出生命周期 2 分，每个方法两分。

servlet 有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。

这个生存期由 javax.servlet.servlet 接口中的 init、service、destroy 方法表达。

web 容器加载 servlet，生命周期开始。通过调用 servlet 的 init()方法进行 servlet 的初始化。通过调用 service()方法实现

根据请求的不同调用不同的 do**()方法。结束服务，web 容器调用 servlet 的 destroy()方法。

13、在 HTTP 请求中，什么情况下我们会选择 post 方式而非 get？反之也是如何考虑的？

分数标注：满分 10 分,答出一条 5 分。

1、get 方式的安全性较 Post 方式要差些，包含机密信息的话，建议用 Post 数据提　交方式；

2、在做数据查询时，建议用 Get 方式；而在做数据添加、修改或删除时，建议用　　Post 方式；

14、cookie 和 session 的区别与联系

分数标注：满分 10 分,答出一条 2 分。

1、cookie 数据存放在客户的浏览器上，session 数据放在服务器上。

2、cookie 不是很安全，别人可以分析存放在本地的 cookie 并进行 cookie 欺骗，考虑到安全应当使用 session。

3、session 会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能，考虑到减轻服务器性能方面，应当使用 cookie。

4、单个 cookie 保存的数据不能超过 4K，很多浏览器都限制一个站点最多保存 20 个 cookie。

5、可以考虑将登陆信息等重要信息存放为 session，其他信息如果需要保留，可以放在 cookie 中。

15、转发和重定向区别？

分数标注：满分 10 分,答出一条 2 分。

1.重定向的执行过程 :Web 服务器向浏览器发送一个 http 响应--》浏览器接受此响应后再发送一个新的 http 请求到服务器--》服务器根据此请求寻找资源并发送给浏览器。它可以重定向到任意 URL，不能共享 request 范围内的数据。

2.重定向是在客户端发挥作用，通过新的地址实现页面转向。

3.重定向是通过浏览器重新请求地址，在地址栏中可以显示转向后的地址。

4.转发过程：Web 服务器调用内部方法在容器内部完成请求和转发动作--》将目标资源发送给浏览器，它只能在同一个 Web 应用中使用，可以共享 request 范围内的数据。

5.转发是在服务器端发挥作用，通过 forward()方法将提交信息在多个页面间进行传递。

6.转发是在服务器内部控制权的转移，客户端浏览器的地址栏不会显示出转向后的地址。

16、拦截器和过滤器区别

分数标注：满分 10 分,答出一条 2 分。

1.拦截器是基于 java 的反射机制的，而过滤器是基于函数回调。

2.拦截器不依赖与 servlet 容器，过滤器依赖与 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的请求起作用。

4.拦截器可以访问 action 上下文、值栈里的对象，而过滤器不能访问。

5.在 action 的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时被调用一次。

6.拦截器可以获取 IOC 容器中的各个 bean，而过滤器就不行，这点很重要，在拦截器里注入一个 service，可以调用业务逻辑。

7.过滤器包裹住 servlet，servlet 包裹住拦截器。

17、你的项目中使用过那些 JSTL 标签？

分数标注：满分 10 分,答出一条 1 分。

<c:out>　　用于在 JSP 中显示数据，就像<%= ... >

<c:set> 用于保存数据
<c:remove> 用于删除数据
<c:catch> 用来处理产生错误的异常状况，并且将错误信息储存起来
<c:if> 与我们在一般程序中用的 if 一样
<c:choose> 本身只当做<c:when>和<c:otherwise>的父标签
<c:when> <c:choose>的子标签，用来判断条件是否成立
<c:otherwise><c:choose>的子标签，接在<c:when>标签后，当<c:when>标签判 断为 false 时被执行
<c:import> 检索一个绝对或相对 URL，然后将其内容暴露给页面
<c:forEach> 基础迭代标签，接受多种集合类型
<c:forTokens> 根据指定的分隔符来分隔内容并迭代输出
<c:param> 用来给包含或重定向的页面传递参数
<c:redirect> 重定向至一个新的 URL
<c:url> 使用可选的查询参数来创造一个 URL

18、如何防止表单重复提交问题

分数标注：满分 10 分,答出第一条，第二条各 1 分，答出第三条 2 分，第四条 6 分。

1.用 Javascript 禁掉提交按钮，表单提交之后，用 JS 置灰 submit 按钮，可防止心急用户重复点击提交按钮。

2.用 Redirect-After-Post 模式，用户提交表单之后，执行重定向 sendRedirect()，转到成功信息页面。除此之外，当用户提交表单，服务器端调用 forward()方法，转发到其他页面。

3.在数据库里添加唯一约束，在数据库里添加唯一约束或创建唯一索引，防止出现重复数据。这是最有效的防止重复提交数据的方法。

4.用 Session 防止表单重复提交

具体的做法：在服务器端生成一个唯一的随机标识号，专业术语称为 Token(令牌)，同时在当前用户的 Session 域中保存这个 Token。然后将 Token 发送到客户端的 Form 表单中，在 Form 表单中使用隐藏域来存储这个 Token，表单提交的时候连同这个 Token 一起提交到服务器端，然后在服务器端判断客户端提交上来的 Token 与服务器端生成的 Token 是否一致，如果不一致，那就是重复提交了，此时服务器端就可以不处理重复提交的表单。如果相同则处理表单提交，处理完后清除当前用户的 Session 域中存储的标识号。

在下列情况下，服务器程序将拒绝处理用户提交的表单请求：

（1）存储 Session 域中的 Token(令牌)与表单提交的 Token(令牌)不同。

（2）当前用户的 Session 中不存在 Token(令牌)。

（3）用户提交的表单数据中没有 Token(令牌)。

19、TCP 和 UDP 区别，你对 HTTP 协议的理解

分数标注：满分 10 分,酌情给分。

TCP 的优点： 可靠，稳定 TCP 的可靠体现在 TCP 在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源。 TCP 的缺点： 慢，效率低，占用系统资源高，易被攻击 TCP 在传递数据之前，要先建连接，这会消耗时间，

而且在数据传递时，确认机制、重传机制、拥塞控制机制等都会消耗大量的时间，而且要在每台设备上维护所有的传输连接，事实上，每个连接都会占用系统的 CPU、内存等硬件资源。 而且，因为 TCP 有确认机制、三次握手机制，这些也导致 TCP 容易被人利用，实现 DOS、DDOS、CC 等攻击。

UDP 的优点： 快，比 TCP 稍安全 UDP 没有 TCP 的握手、确认、窗口、重传、拥塞控制等机制，UDP 是一个无状态的传输协议，所以它在传递数据时非常快。没有 TCP 的这些机制，UDP 较 TCP 被攻击者利用的漏洞就要少一些。但 UDP 也是无法避免攻击的，比如：UDP Flood 攻击…… UDP 的缺点： 不可靠，不稳定 因为 UDP 没有 TCP 那些可靠的机制，在数据传递时，如果网络质量不好，就会很容易丢包。 基于上面的优缺点，那么： 什么时候应该使用 TCP： 当对网络通讯质量有要求的时候，比如：整个数据要准确无误的传递给对方，这往往用于一些要求可靠的应用，比如 HTTP、HTTPS、FTP 等传输文件的协议，POP、SMTP 等邮件传输的协议。 在日常生活中，常见使用 TCP 协议的应用如下： 浏览器，用的 HTTP FlashFXP，用的 FTP Outlook，用的 POP、SMTP Putty，用的 Telnet、SSH QQ 文件传输 …… 什么时候应该使用 UDP： 当对网络通讯质量要求不高的时候，要求网络通讯速度能尽量的快，这时就可以使用 UDP。 比如，日常生活中，常见使用 UDP 协议的应用如下： QQ 语音 QQ 视频 TFTP ……

HTTP 是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于 1990 年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在 WWW 中使用的是 HTTP/1.0 的第六版，HTTP/1.1 的规范化工作正在进行之中，而且 HTTP-NG(Next Generation of HTTP) 的建议已经提出。

HTTP 协议的主要特点可概括如下：

1.支持客户/服务器模式。

2.简单快速 :客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有 GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于 HTTP 协议简单，使得 HTTP 服务器的程序规模小，因而通信速度很快。

3.灵活：HTTP 允许传输任意类型的数据对象。正在传输的类型由 Content-Type 加以标记。

4.无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

5.无状态：HTTP 协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

20、json 数据的格式是什么?

分数标注：满分 10 分，答对满分，答错没分

1.最外层只能是[]或{}

2.[]表示 JSON 数组：[value_1,value_2,...,value_n]

3.{}表示 JSON 对象：{key_1:value_1,key_2:value_2,...,key_n:value_n}

4.key 数据类型：字符串

5.value 数据类型：

基本数据类型：数值，字符串，布尔

引用数据类型：数组和对象

SSM 框架阶段

1、Spring 是如何管理事务的，事务管理机制？以及隔离级别？

事务的概念（1 分）事务必须服从 ACID 原则。ACID 指的是原子性（atomicity）、一致性（consistency）、隔离性（isolation）和持久性（durability）。

通俗理解，事务其实就是一系列指令的集合。

特性：（4 分）

原子性：操作这些指令时，要么全部执行成功，要么全部不执行。只要其中一个指令执行失败，所有的指令都执行失败，数据进行回滚，回到执行指令前的数据状态。

一致性：事务的执行使数据从一个状态转换为另一个状态，但是对于整个数据的完整性保持稳定。

隔离性：在该事务执行的过程中，无论发生的任何数据的改变都应该只存在于该事务之中，对外界不存在任何影响。只有在事务确定正确提交之后，才会显示该事务对数据的改变。其他事务才能获取到这些改变后的数据。

持久性：当事务正确完成后，它对于数据的改变是永久性的。

编程式和声明式事务的区别：（2 分）

Spring 提供了对编程式事务和声明式事务的支持，编程式事务允许用户在代码中精确定义事务的边界，而声明式事务（基于 AOP）有助于用户将操作与事务规则进行解耦。

简单地说，编程式事务侵入到了业务代码里面，但是提供了更加详细的事务管理；而声明式事务由于基于 AOP，所以既能起到事务管理的作用，又可以不影响业务代码的具体实现。

事务管理机制：（1 分）

Spring 并不直接管理事务，而是提供了多种事务管理器，他们将事务管理的职责委托给 Hibernate 或者 JTA 等持久化机制所提供的相关平台框架的事务来实现。

Spring 事务管理器的接口是 org.springframework.transaction.PlatformTransactionManager，通过这个接口，Spring 为各个平台如 JDBC、Hibernate 等都提供了对应的事务管理器，但是具体的实现就是各个平台自己的事情了。

隔离级别：（2 分）

ISOLATION_DEFAULT：使用后端数据库默认的隔离级别；

ISOLATION_READ_UNCOMMITTED：最低的隔离级别，允许读取尚未提交的数据变更，可能会导致脏读、幻读或不可重复读；

ISOLATION_READ_COMMITTED：允许读取并发事务已经提交的数据，可以阻止脏读，但是幻读或不可重复读仍有可能发生；

ISOLATION_REPEATABLE_READ：对同一字段的多次读取结果都是一致的，除非数据是被本身事务自己所修改，可以阻止脏读和不可重复读，但幻读仍有可能发生；

ISOLATION_SERIALIZABLE：最高的隔离级别，完全服从 ACID 的隔离级别，确保阻止脏读、不可重复读以及幻读，也是最慢的事务隔离级别，因为它通常是通过完全锁定事务相关的数据库表来实现的。

2、Spring AOP 的实现原理？

AOP 的概念：面向切面编程（利用“横切”，解剖封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，这样就能减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。）（1 分）

OOP：面向对象编程（引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。但是，如果我们需要为部分对象引入公共部分的时候，OOP 就会引入大量重复的代码。）（1 分）

所以，在某种程度上也可以说 AOP 是对 OOP 的补充。

Spring 框架中的 AOP 体现了设计模式中的代理模式。（1 分）

CGLIB 和 JDK 动态代理区别:（2 分）

JDK:

目标类和代理类实现了共同的接口

拦截器必须实现 InvocationHandler 接口,而这个接口中 invoke 方法体的内容就是代理对象方法体的内容。

CGLIB:

目标类 是代理类的父类

拦截器必须实现 MethodInterceptor 接口,而接口中的 intercept 方法就是代理类的方法体,使用字节码增强机制创建代理对象的。

动态代理的特性：（2 分）

1） 继承了 Proxy 类，实现了代理的接口，由于 java 不能多继承，这里已经继承了 Proxy 类了，不能再继承其他的类，所以 JDK 的动态代理不支持对实现类的代理，只支持接口的代理。

2） 提供了一个使用 InvocationHandler 作为参数的构造方法。

3） 生成静态代码块来初始化接口中方法的 Method 对象，以及 Object 类的 equals、hashCode、toString 方法。

4）重写了 Object 类的 equals、hashCode、toString，它们都只是简单的调用了 InvocationHandler 的 invoke 方法，即可以对其进行特殊的操作，也就是说 JDK 的动态代理还可以代理上述三个方法。

动态代理和静态代理区别？（3 分）

静态代理需要手工编写代理类，代理类引用被代理对象（在代理对象和目标对象实现共同的接口，并且代理对象持有目标对象的引用。）

动态代理是在内存中构建的，不需要手动编写代理类（jdk1.3 后主要是实现 InvocationHandler，并且将目标对象注入到代理对象中，利用反射机制来执行目标对象的方法。）

代理的目的：是为了在原有的方法上进行增强。

3、IOC 和 DI 是什么？

IOC 是什么：控制反转，是一种设计思想。Ioc 意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。（3 分）

为什么是反转，那些地方反转了：传统应用程序是由我们自己在对象中主动控制去直接获取依赖对象，也就是正转；而反转则是由容器来帮忙创建及注入依赖对象；为何是反转？因为由容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反转；哪些方面反转了？依赖对象的获取被反转了。（3 分）

DI 是什么：依赖注入，是组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。（2 分）（以下 4 点共计 2 分）

谁依赖于谁：当然是应用程序依赖于 IOC 容器；

为什么需要依赖：应用程序需要 IOC 容器来提供对象需要的外部资源；

谁注入谁：很明显是 IOC 容器注入应用程序某个对象，应用程序依赖的对象；

注入了什么：就是注入某个对象所需要的外部资源（包括对象、资源、常量数据）。

4、Spring 中用到了那些设计模式？（工厂、代理、单例各 2 分，其他每多一项加 1 分）

简单工厂：实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类。

Spring 中的 BeanFactory 就是简单工厂模式的体现，根据传入一个唯一的标识来获得 Bean 对象，但是否是在传入参数后创建还是传入参数前创建这个要根据具体情况来定。

工厂方法：定义一个用于创建对象的接口，让子类决定实例化哪一个类。Factory Method 使一个类的实例化延迟到其子类。Spring 中的 FactoryBean 就是典型的工厂方法模式。

单例：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

Spring 中的单例模式完成了后半句话，即提供了全局的访问点 BeanFactory。但没有从构造器级别去控制单例，这是因为 Spring 管理的是是任意的 Java 对象。

适配器：将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的那些类可以一起工作。

由于 Advisor 链需要的是 MethodInterceptor（拦截器）对象，所以每一个 Advisor 中的 Advice 都要适配成对应的 MethodInterceptor 对象。

包装器：动态地给一个对象添加一些额外的职责。就增加功能来说，Decorator 模式相比生成子类更为灵活。

Spring 中用到的包装器模式在类名上有两种表现：一种是类名中含有 Wrapper，另一种是类名中含有 Decorator。基本上都是动态地给一个对象添加一些额外的职责。

代理：从结构上来看和 Decorator 模式类似，但 Proxy 是控制，更像是一种对功能的限制，而

Decorator 是增加职责。

Spring 的 Proxy 模式在 aop 中有体现，比如 JdkDynamicAopProxy 和 Cglib2AopProxy。

观察者：定义对象间的一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。

Spring 中 Observer 模式常用的地方是 listener 的实现。如 ApplicationListener。

策略：定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。本模式使得算法可独立于使用它的客户而变化。

Spring 中在实例化对象的时候用到 Strategy 模式。

模板方法：定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method 使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

回调对象中定义一个操纵 JdbcTemplate 中变量的方法，我们去实现这个方法，就把变化的东西集中到这里了。然后我们再传入这个回调对象到 JdbcTemplate，从而完成了调用。

5、Spring 中 Bean 的作用域有哪些？（一项 2 分）

Singleton：使用该属性定义 Bean 时，IOC 容器仅创建一个 Bean 实例，IOC 容器每次返回的是同一个 Bean 实例；

Prototype：使用该属性定义 Bean 时，IOC 容器可以创建多个 Bean 实例，每次返回的都是一个新的实例；

Request：该属性仅对 HTTP 请求产生作用，使用该属性定义 Bean 时，每次 HTTP 请求都会创建一个新的 Bean，适用于 WebApplicationContext 环境；

Session：该属性仅用于 HTTP Session，同一个 Session 共享一个 Bean 实例。不同 Session 使用不同的实例；

global-session：该属性仅用于 HTTP Session，同 session 作用域不同的是，所有的 Session 共享一个 Bean 实例。

6、spring 框架实现实例化和依赖注入的方式分别是什么？

实例化 bean：也就是依赖对象实例化。（1 分）

创建方式：setter 方法、构造函数、静态工厂、实例工厂。（2 分）

依赖注入的方式：

使用最广泛的 @Autowired：自动装配：用于替代基于 XML 配置的自动装配；（2 分）

setter 方法注入：假设有一个 SpringAction，类中需要实例化一个 SpringDao 对象，那么就可以定义一个 private 的 SpringDao 成员变量，然后创建 SpringDao 的 set 方法（这是 ioc 的注入入口）；（2 分）

构造器注入：是指带有参数的构造函数注入，看下面的例子，我创建了两个成员变量 SpringDao 和 User，但是并未设置对象的 set 方法，所以就不能支持第一种注入方式，这里的注入方式是在 SpringAction 的构造函数中注入，也就是说在创建 SpringAction 对象时要将 SpringDao 和 User 两个参数值传进来；（1 分）

静态工厂的方法注入：就是通过调用静态工厂的方法来获取自己需要的对象，为了让 spring 管理所有对象，我们不能直接通过"工程类.静态方法()"来获取对象，而是依然通过 spring 注入的形式获取；（1 分）

实例工厂的方法注入：实例工厂的意思是获取对象实例的方法不是静态的，所以你需要首先 new 工厂类，再调用普通的实例方法。（1 分）

7、SpringMVC 的工作流程?（各 1 分）

（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）处理器映射器根据请求 url 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 通过 HandlerAdapter 处理器适配器调用处理器；

（5）执行处理器(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 对 View 进行渲染视图（即将模型数据填充至视图中）。DispatcherServlet 响应用户。

8、spring 以及 springMVC 常用注解有哪些？

Spring 常用注解：(各 1 分，最高 5 分)

@Configuration 把一个类作为一个 IoC 容器，它的某个方法头上如果注册了@Bean，就会作为这个 Spring 容器中的 Bean；

@Scope 注解 作用域；

@Lazy(true) 表示延迟初始化；

@Service 用于标注业务层组件；

@Controller 用于标注控制层组件（如 struts 中的 action）;

@Repository 用于标注数据访问组件，即 DAO 组件；

@Component 泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注；

@Scope 用于指定 scope 作用域的（用在类上）;

@Autowired 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。

SpringMVC 常用注解：(各 1 分，最高 5 分)

@Controller 用于标记在一个类上，使用它标记的类就是一个 SpringMVC Controller 对象；

@RequestMapping 是一个用来处理请求地址映射的注解，可用于类或方法上。用于类上，表示类中的所有响应请求的方法都是以该地址作为父路径；

@requestParam 主要用于在 SpringMVC 后台控制层获取参数；

@ResponseBody 用于将 Controller 的方法返回的对象，通过适当的 HttpMessageConverter 转换为指定格式后，写入到 Response 对象的 body 数据区；

@Resource 默认按照 ByName 自动注入，由 J2EE 提供，需要导入包 javax.annotation.Resource。@Resource 有两个重要的属性：name 和 type，而 Spring 将@Resource 注解的 name 属性解析为 bean

的名字，而 type 属性则解析为 bean 的类型。所以，如果使用 name 属性，则使用 byName 的自动注入策略，而使用 type 属性时则使用 byType 自动注入策略。如果既不制定 name 也不制定 type 属性，这时将通过反射机制使用 byName 自动注入策略；

@PathVariable 用于将请求 URL 中的模板变量映射到功能处理方法的参数上，即取出 uri 模板中的变量作为参数。

9、简单介绍下 springMVC 和 struts2 的区别有哪些？（任意一点 2 分）

一、框架机制

1、Struts2 采用 Filter（StrutsPrepareAndExecuteFilter）实现，SpringMVC（DispatcherServlet）则采用 Servlet 实现。

2、Filter 在容器启动之后即初始化；服务停止以后坠毁，晚于 Servlet。Servlet 在是在调用时初始化，先于 Filter 调用，服务停止后销毁。

二、拦截机制

1、Struts2

a、Struts2 框架是类级别的拦截，每次请求就会创建一个 Action，和 Spring 整合时 Struts2 的 ActionBean 注入作用域是原型模式 prototype（否则会出现线程并发问题），然后通过 setter，getter 吧 request 数据注入到属性。

b、Struts2 中，一个 Action 对应一个 request，response 上下文，在接收参数时，可以通过属性接收，这说明属性参数是让多个方法共享的。

c、Struts2 中 Action 的一个方法可以对应一个 url，而其类属性却被所有方法共享，这也就无法用注解或其他方式标识其所属方法了

2、SpringMVC

a、SpringMVC 是方法级别的拦截，一个方法对应一个 Request 上下文，所以方法直接基本上是独立的，独享 request，response 数据。而每个方法同时又何一个 url 对应，参数的传递是直接注入到方法中的，是方法所独有的。处理结果通过 ModeMap 返回给框架。

b、在 Spring 整合时，SpringMVC 的 Controller Bean 默认单例模式 Singleton，所以默认对所有的请求，只会创建一个 Controller，有应为没有共享的属性，所以是线程安全的，如果要改变默认的作用域，需要添加@Scope 注解修改。

三、性能方面

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

四、拦截机制

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

五、配置方面

spring MVC 和 Spring 是无缝的。从这个项目的管理和安全上也比 Struts2 高（当然 Struts2 也可以通过不同的目录结构和相关配置做到 SpringMVC 一样的效果，但是需要 xml 配置的地方不少）。

SpringMVC 可以认为已经 100%零配置。

六、设计思想

Struts2 更加符合 OOP 的编程思想， SpringMVC 就比较谨慎，在 servlet 上扩展。

10、springmvc 前端控制器是什么？处理器映射器是什么？

前端控制器：DispatcherServlet 是前端控制器设计模式的实现，提供 SpringMVC 的集中访问点，而且负责职责的分派，而且与 Spring IoC 容器无缝集成。（2 分）

配置流程：（3 分）

第一种:*.action。访问以.action 结尾由 DispatcherServlet 进行解析；

第二种:/，所有访问的地址都由 DispatcherServlet 进行解析，对于静态文件的解析，我们要配置不让 DispatcherServlet 进行解析。使用此种方法可以实现 RESTful 风格的 url；

第三种:/*，这样配置不对，使用这种配置，最终要转发到一个 jsp 页面时，仍然会由 DispatcherServlet 进行解析 jsp 地址，它不能根据 jsp 页面找到 Handler，会报错。

处理映射器：

SpringMVC 中处理器映射器 HandlerMapping 根据配置找到相应的 Handler，返回给前端控制器 DispatcherServlet，这个 Handler 可能包含 N 个 Interceptor 拦截器。（1 分）

非注解方式配置，也就是基于 xml 配置文件（2 分）

非注解处理器类 BeanNameUrlHandlerMapping，这个类的映射规则是将 bean 的 name 作为 url 进行查找，需要在配置 Handler 时指定 beanname，这时的 url 地址就会根据 bean 的 name 去查找，也就是示例中配置的 name 的值 test.do，且这个必须以 / 开头；

非注解处理器类 SimpleUrlHandlerMapping，它可以通过内部参数去配置请求的 url 和 handler 之间的映射关系，其中 property 属性可以配置 interceptors 拦截器，也可以配置 mapping（Handler 处理器和 url 的映射关系）；

非注解处理器类 ControllerClassNameHandlerMapping，它是将 Controller 控制器的名字作为映射的 url 地址，例如 TestController 控制器则映射的地址就为 /test*，如果是 test 控制器下的 test 方法，则 url 为 test/test.action。

注解配置，在类中做相应的注解即可（2 分）

第一种是在 springmvc.xml 中配置，声明相关的 bean 和实现即可；

第二种是使用 <mvc:annotation-driven/> 标签来配置，这个标签是一种简写方式，它会自动去注册处理器映射器。

（@Controller 是注解信息，表示该类是一个控制器类，可以被注解的处理器适配器找到，而 TestControllerTest 类中的 testurl 方法上有一个 @RequestMapping 注解信息，作用是指定一个 url 与该方法绑定。

这时为了让注解的处理器映射器能找到 Handler 控制器，需要在 springmvc.xml 做下配置，方式有两种：

在 springmcv.xml 中声明 bean 信息；

对某一个包下的所有类进行扫描，找出所有使用 @Controller 注解的 Handler 控制器类。）

11. springmvc 如何进行参数绑定?

1. 默认支持的参数绑定 （HttpServletRequest，HttpServletResponse，HttpSession，Model/ModelMap)(2.5 分)

2. 基本类型的参数绑定 （@RequestParam，@PathVariable，@ModelAttribute，特殊情况下可以用自定义类型转换器---实现 converter 接口)(2.5 分)

3. pojo 类型的绑定 (2.5 分)

4.高级参数绑定　与数组的参数绑定，与集合的参数绑定 (2.5 分)

12. springmvc 获取表单数据的几种方式?

1、直接把表单的参数写在 Controller 相应的方法的形参中 （2.5 分)

2、通过 HttpServletRequest 接收 (2.5 分)

3.、通过一个 bean 来接收，用这个 bean 来封装接收的参数 (2.5 分)

4、.利用 js 中 ajax 请求通过 json 数据接收 (2.5 分)

13、SSM 架构的整合流程是怎样的?

1. Maven 引入需要的 JAR 包　(1 分)

2. Spring 与 MyBatis 的整合

2.1 建立 JDBC 属性文件 jdbc.properties　(1 分)

2.2 建立 spring-mybatis.xml 配置文件 主要的就是自动扫描，自动注入，配置数据库，事务管理 (3 分)

2.3 配置 Log4j (1 分)

2.4 JUnit 测试

3. 整合 SpringMVC

3.1 配置 spring-mvc.xml　主要是自动扫描控制器，视图解析器，注解驱动的启动这三个 (2 分)

3.2 配置 web.xml 文件　引入 spring-mybatis.xml ,编码过滤器 ,Spring 监听器 ,配置的 spring-mvc 的 Servlet 前端控制器 （3 分)

4.整合完毕，进行测试

具体实现：https://blog.csdn.net/zhshulin/article/details/37956105/

14. mybatis 和 hibernate 之间的优缺点比较?

Hibernate 的优点(2.5 分)：

1、hibernate 是全自动，hibernate 完全可以通过对象关系模型实现对数据库的操作，拥有完整的 JavaBean 对象与数据库的映射结构来自动生成 sql。

2、功能强大，数据库无关性好，O/R 映射能力强，需要写的代码很少，开发速度很快。

3、有更好的二级缓存机制，可以使用第三方缓存。

4、数据库移植性良好。

5、hibernate 拥有完整的日志系统，hibernate 日志系统非常健全，涉及广泛，包括 sql 记录、关系异常、优化警告、缓存提示、脏数据警告等

Hibernate 的缺点(2.5 分)：

1、学习门槛高，精通门槛更高，程序员如何设计 O/R 映射，在性能和对象模型之间如何取得平衡，以及怎样用好 Hibernate 方面需要的经验和能力都很强才行

2、hibernate 的 sql 很多都是自动生成的，无法直接维护 sql；虽然有 hql 查询，但功能还是不及 sql 强大，见到报表等变态需求时，hql 查询要虚，也就是说 hql 查询是有局限的；hibernate 虽然也支持原生 sql 查询，但开发模式上却与 orm 不同，需要转换思维，因此使用上有些不方便。总之写 sql 的灵活度上 hibernate 不及 mybatis。

Mybatis 的优点(2.5 分)：

1、易于上手和掌握，提供了数据库查询的自动对象绑定功能，而且延续了很好的 SQL 使用经验，对于没有那么高的对象模型要求的项目来说，相当完美。

2、sql 写在 xml 里，便于统一管理和优化， 解除 sql 与程序代码的耦合。

3、提供映射标签，支持对象与数据库的 orm 字段关系映射

4、 提供对象关系映射标签，支持对象关系组建维护

5、提供 xml 标签，支持编写动态 sql。

6、速度相对于 Hibernate 的速度较快

Mybatis 的缺点(2.5 分)：

1、关联表多时，字段多的时候，sql 工作量很大。

2、sql 依赖于数据库，导致数据库移植性差。

3、由于 xml 里标签 id 必须唯一，导致 DAO 中方法不支持方法重载。

4、对象关系映射标签和字段映射标签仅仅是对映射关系的描述，具体实现仍然依赖于 sql。

5、DAO 层过于简单，对象组装的工作量较大。

6、不支持级联更新、级联删除。

7、Mybatis 的日志除了基本记录功能外，其它功能薄弱很多。

8、编写动态 sql 时，不方便调试，尤其逻辑复杂时。

9、提供的写动态 sql 的 xml 标签功能简单，编写动态 sql 仍然受限，且可读性低。

15. MyBatis 中使用#和$书写占位符有什么区别？

1.#{}表示一个占位符号，通过#{}可以实现 preparedStatement 向占位符中设置值，自动进行 java 类型和 jdbc 类型转换，#{}可以有效防止 sql 注入。 （2.5 分）

2.#{}可以接收简单类型值或 pojo 属性值，如果 parameterType 传输单个简单类型值，#{}括号中可以是 value 或其它名称 （2.5 分）

3.${}表示拼接 sql 串，通过${}可以将 parameterType 传入的内容拼接在 sql 中且不进行 jdbc 类型转换。 （2.5 分）

4..${}可以接收简单类型值或 pojo 属性值，如果 parameterType 传输单个简单类型值，${}括号中只能是 value。（2.5 分）

16. MyBatis 中的动态 SQL 是什么意思？

1.mybatis 提供使用 ognl 表达式动态生成 sql 的功能。（是什么） 5 分

2.传统 jdbc 方法中，在写组合的多表复杂 sql 语句时，需要去拼接 sql 语句，稍不注意少写一个空格或“”，就会导致报错,m ybatis 动态 sql 的功能，就拥有有效的解决了这个问题。（为什么） 2.5 分

3.例如：（怎么用）2.5 分

If 标签: 多个条件查询时候用到

Where 标签:可以自动处理第一个 and

Foreatch:应用场景 在多个 ID 查询时 传递的是集合或者是数组时候

17. Mybatis 中 Mapper 动态代理规范是什么？

1.Mapper.xml 配置文件中 namespace 要和接口的全路径一直 （2.5 分）
2.接口中的方法名要和 mapper.xml 中 statement 的 ID 要相同（2.5 分）
3.接口中的方法的参数类型要和 paremterType 的类型要一致 （2.5 分）
4.接口中的方法的返回值类型要和 mapper.xml 配置文件 resultType 一致 （2.5 分）

18. mybatis 的执行流程是什么？

1. 加载全局配置文件（mybatisConfig.xml），这个配置文件中通常是别名设置，拦截器的设置，（当 ssm 整合后，环境配置与 mapper 映射文件的注册会转移到 spring 配置文件中）（1 分）
2. xml 全局配置文件会产生一个构建者类，叫做 xmlConfigBuilder，这个类是用来通过 xml 配置文件来构建 COnfiguration 对象实例的，构建的过程就是解析（MBatistConfig。xml）配置文件调用 parse 产生 configuration 对象 （1 分）
3. 随后产生的就是 Mybatis 的配置类，（configuration），这个类可以作为项目的全局配置对象 （1 分）
4. 接下来便是 SqlSessionFactory（会话工厂）的构建者类，（SqlSessionFactoryBuilder），configuration 配置对象，就可以调用会话工厂构建者类中的 build 方法完成对会话工厂对象的构建 （1 分）
5. 产生 sqlsessionFactory（会话工厂），是用来生成会话的接口，有一个实现类（DefaultSqlSessionFactory），这个实现类是真正的会话 的工厂类，并且他是单利的。会一直存在到服务器关闭 （1 分）
6. 通过调用会话工厂的实现类中（DefaultSqlSessionFactory）的 openSession（ ）方法完成 SqlSession 对象的创建 （1 分）
7. 产生 sqlSession，该接口是会话，并且是非线程安全的，每一次对数据库的访问都需要创建一个 sqlSession，当得到结果后 SqlSession 就会被废弃。所以声明周期短 （1 分）
8.. 当然这当中还有一个 Executor 执行器接口，这才是内部真正对数据库进行操作的操作者，他才是真正的干事的 （1 分）
9.另外就是 StatementHandler 该类是 Statment 处理器，封装了对数据库各种操作方法，使用的时候，就调用其中的一些方法罢了 （1 分）
10，最后就是结果集处理器（ResultSetHandler），这个处理器的作用就是对结果进行处理并返回的。（当然这是在有结果返回的情况下，需要对结果集进行处理) （1 分）

19. 说说你比较熟悉的设计模式及应用场景？

单例

单例模式(Singleton)： 保证一个类仅有一个实例，并提供一个访问它的全局访问点。

应用场景：通常，我们可以让一个全局变量使得一个对象被访问，但它不能防止你实例化多个对象，一个最好的办法就是，让类自身负责保存它的唯一实例。这个类可以保证没有其他实例可以被创建，而且它可以提供一个访问该实例的方法。

应用：java.lang.Runtime；

hibernate 的 sessionFactory 是会话工厂，整个运行过程中只有一个 sessionFactory 实例即可

工厂模式

工厂模式（Factory）：定义一个用于创建对象的接口，让接口子类通过工厂方法决定实例化哪一个类。

应用场景：创建不同的产品对象，客户端应使用不同的具体工厂。

应用：jdk 中连接数据库的代码是典型的工厂模式，每一种数据库只需提供一个统一的接口：Driver（工厂类）

java.util.Collection 接口中定义了一个抽象的 iterator() 方法，该方法就是一个工厂方法。对于 iterator() 方法来说 Collection 就是一个工厂。

Spring 的 BeanFactory

代理模式(proxy)：代理模式给某一个对象提供一个代理对象，并由代理对象控制对源对象的引用。代理就是一人或一个机构代表另一个人或者一个机构采取行动。

应用场景：

a) 远程代理：为一个对象在不同的地址空间提供局部代表，这样可以隐藏一个对象存在于不同地址空间的事实。【WebService，客户端可以调用代理解决远程访问问题】

b) 虚拟代理：根据需要创建开销很大的对象，通过它来存放实例化需要很长时间地真实对象。【比如 Html 网页的图片，代理存储的是真实图片的路径和尺寸】

c) 安全代理：用来控制真实对象的访问权限。

d) 智能指引：当调用真实的对象时，代理处理另一些事。【如计算机真实对象的引用次数，代理在访问一个对象的时候回附加一些内务处理，检查对象是否被锁定、是否该释放、是否该装入内存等等】

应用：

1. java.lang.reflect 包中的 Proxy 类和 InvocationHandler 接口提供了生成动态代理类的能力

2.spring 的 aop

装饰模式：

装饰模式(Decorator) ：动态地给一个对象添加一些额外的职责，就增加功能来说，装饰模式比生成子类更灵活

场景：装饰模式是为了已有功能动态地添加更多功能的一种方式，当系统需要新功能的时候，是向旧类中添加新的代码，这些新的代码通常装饰了原有类的核心职责或主要行为。装饰着模式把每个要装饰的功能放在单独的类中，并让这个类包装它所要装饰的对象，当需要执行特殊行为时，客户代码就可以在运行时根据需要有选择的、按顺序地使用装饰功能包装对象。

应用：Java I/O 使用装饰模式设计,JDK 中还有很多类是使用装饰模式设计的,如:Reader 类、Writer 类、OutputStream 类等。

迭代器模式

迭代器模式(Iterator)：提供一种方法顺序访问一个聚合对象中各个元素，而又不暴露该对象的内部表示

场景：当需要对聚集有多种方式遍历时，可以考虑使用迭代器

应用：collection 容器使用了迭代器模式

说出任意一种得 5 分（单例模式需要会手写）

https://blog.csdn.net/u014282557/article/details/72823201

http://www.cnblogs.com/dassmeta/p/6985431.html

20. 动态代理 的 2 种方式以及区别？

1.JDK 动态代理：利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调用 InvokeHandler 来处理。（2.5 分）

2.CGlib 动态代理：利用 ASM（开源的 Java 字节码编辑库，操作字节码）开源包，将代理对象类的 class 文件加载进来，通过修改其字节码生成子类来处理。 （2.5 分）

3.区别：（5 分）

（1）JDK 动态代理只能对实现了接口的类生成代理，而不能针对类

（2）CGLIB 是针对类实现代理，主要是对指定的类生成一个子类，覆盖其中的方法

因为是继承，所以该类或方法最好不要声明成 final

（3）CGLIB 效率较高

电商项目问题（每小题 10 分）

1. 电商项目中有没有用到多线程，哪些地方要用多线程？（10 分）

答：

1.用户注册完成送大礼包/积分之类，单独启动一个线程调用积分接口，不必等待返回值。（2 分）

2.对大批量数据的复制操作，可以启动多个线程，给每个线程分配指定的记录数进行保存操作。（2 分）

3.多个请求的操作很耗时，比如三个线程一个线程执行一分钟，可以使用线程池，等待所有线程都执行完毕再返回结果，整个耗时一分钟而不是三分钟。（3 分）

4.springMVC 默认是单例（基于方法），所以每次请求访问就是一个线程

电商项目设计思路：降低应用之间耦合性，减轻服务器压力。会把应用服务之间进行拆分，使用了 ativeMQ（消息队列），解决了一部分多线程问题。（3 分）

2. 日志文件的管理，你们是怎么做的？(10 分)

答：在 linux 中配置日志文件每周转储一次,使用开源实时日志分析 ELK 平台对系统信息进行查找，ELK 由 ElasticSearch、Logstash 和 Kiabana 三个开源工具组成。(10 分)

3. 你觉得分布式开发的缺点是什么？(10 分)

答：

1.增加编码复杂程度，对开发人员技术要求比较高，增加维护成本和公司运营成本。(4 分)

2.环境搭建复杂,各种模块间数据交换容易出现问题,如:保密数据的访问.(3 分)

3.硬件成本高，系统之间交互需要使用远程通信，接口开发增加工作量。(3 分)

4. 支付接口是怎么做的？(10 分)

答：

1.支付接口是通过非对称加密，通过交换公钥使用各自的私钥验证请求参数，保证请求参数的安全性。请求支付接口，同步返回支付结果使用户及时看到支付成功，通过调用系统回调接口通知支付结果，该通知为多次通知，保证用户扣款成功系统订单也必须为已支付状态。(5 分)

2.接口参数，如商品名称、金额等，并通过约定的加密方式及密钥进行加密处理，将以 FORM 表单 POST 或 GET 的方式发送给支付公司提供的支付网关 URL。接口对接时，商城需要提供回调地址，由于接受支付返回的信息，要确保回调地址能正常接受通知。(5 分)

5. redis 为什么可以做缓存？(10 分)

答：

1.内存数据库, 读写速度快(4 分)

2. redis 是单线程非关系型内存数据库，redis 内部是一个 key-value 存储系统，不存在加锁问题，适用于多读少些的情况。(6 分)

6. solr 怎么设置搜索结果排名靠前？(10 分)

答：

1.利用 solr 自己的排序方法，可以在查询时指定按照哪一字段进行排序，比如按照时间的倒序。(4 分)

2.Solr 内改变打分规则有几种形式(6 分)

(1)配置 solr 的 solrconfig.xml 中 edismax，来改变 Boost 打分规则

(2)在 solr 的 schema 中增加一个字段，该字段专门用于排序

(3)自写一个 solr 的评分规则。

7. activeMQ 在项目中如何应用的？(10 分)

答：

1.分布式事物的 base 模型，就是通过消息中间件实现的。可以解决分布式系统之间的耦合情况。提供系统吞吐量，比如秒杀系统可以把用户秒杀成功消息放入 activeMQ 中，启用多线程处理用户成功消息，缓解数据库压力。(5 分)

2.用户的请求数据直接写入数据库，在高并发的情况下，会对数据库造成巨大的压力，同时也使得系统响应延迟加剧。在使用 ActiveMQ 后，用户的请求发给队列后立即返回（当然不能直接给用户提示订单提交成功，提示：您“您提交了订单，请等待系统确认”），再由消息队列的消费者进程从消息队列中获取数据，异步写入数据库。由于消息队列的服务处理速

度远快于数据库，因此用户的响应延迟可得到有效改善。(5 分)

8. activeMQ 如果数据提交不成功怎么办？(10 分)

答：

1.activeMQ 会重新发送消息，假如超过默认重试次数，会放入死信队列。(4 分)

2.会重新发送消息，并记录日志，一般重新发送三次，如果仍然失败，我们会邮件或短信通知相关人员进行处理。(6 分)

9. 单点登录系统是怎么做的？(10 分)

答：

1.单点登录是使用统一验证服务，需要登录时访问登录系统，登录成功之后返回 token，然后请求系统需要登录资源时调用统一验证接口即可。保证了用户一个账号可以登录一个公司的多个系统。(4 分)

2.当用户第一次访问应用系统 1 的时候,因为还没有登录,会被引导到认证系统中进行登录;根据用户提供的登录信息，认证系统进行身份效验，如果通过效验，应该返回给用户一个认证的凭据－－ticket；用户再访问别的应用的时候，就会将这个 ticket 带上，作为自己认证的凭据，应用系统接受到 请求之后会把 ticket 送到认证系统进行效验，检查 ticket 的合法性,如果通过效验，用户就可以在不用再次登录的情况下访问应用系统 2 和应用系统 3 了。(6 分)

(1)用户请求访问业务系统。

(2)业务系统在系统中查看是否有对应请求的有效令牌，若有，则读取对应的身份信息，允许其访问；若没有或令牌无效，则把用户重定向到统一身份认证平台，并携带业务系统地址，进入第③步。

(3)在统一身份认证平台提供的页面中，用户输入身份凭证信息，平台验证此身份凭证信息，若有效，则生成一个有效的令牌给用户，进入第④步；若无效，则继续进行认证，直到认证成功或退出为止。

(4)用户携带第③步获取的令牌，再次访问业务系统。

(5)业务系统获取用户携带的令牌，提交到认证平台进行有效性检查和身份信息获取。

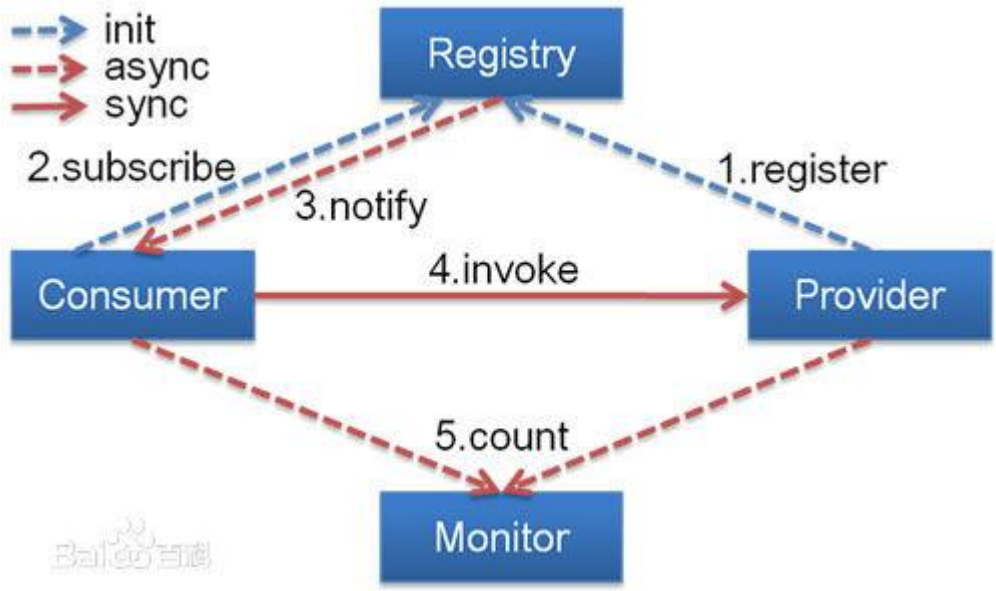
(6)若令牌通过有效性检查，则认证平台会把令牌对应的用户身份信息返回给业务系统，业务系统把身份信息和有效令牌写入会话状态中,允许用户以此身份信息进行业务系统的各种操作；若令牌未通过有效性检查，则会再次重定向到认证平台，返回第③步。通过统一身份认证平台获取的有效令牌，可以在各个业务系统之间实现应用漫游。

10. dubbo 服务开发流程，运行流程？(10 分)

答：

1.开发流程,创建 mavne 项目,导入 dubbo jar 包,编写提供者服务对外接口,配置 zokkeeper 等作为注册中心，注册为服务提供者。编写消费者，导入服务提供者接口工程，去注册中心找到指定的提供者。(4 分)

2.运行流程: 启动服务提供者，接口注入注册中心，启动消费者，消费者去注册中心寻找对应的服务提供者。(6 分)



11. solr 跟数据库的区别，你清楚么？

Solr 不仅限于搜索，Solr 也可以用于存储目的。（3）

像其他 NoSQL 数据库一样，它是一种非关系数据存储和处理技术。（3）

Solr 是一个可扩展的，可部署，存储引擎，优化搜索大量以文本为中心的数据。（4）

12. 广告数据是怎么用 redis 缓存的？

我们在项目使用的是 springDataRedis ,springDataRedis 是 spirng 家族的一部分 提供了在 spring 应用中通过简单的配置访问 redis 服务,对 redis 底层开发包(Jedis ， JRedis ）进行了高度封装，RedisTemplate 提供了 redis 各种操作、异常处理以及序列化，支持订阅发布。（4）

在广告这块其实就是把数据库中查到的数据放在 redis 中一份，我用的类型是 hash,把分类 ID 设置为 key 键，根据分类 ID 查询得到的广告集合放进 value 里，这样便于数据的管理和展示。前台先从缓存中查询，查到就直接返回给前台；如果缓存中查询不到，查询数据库中的数据，添加到 redis 中即可。（6）

13. 项目中权限是怎么做的？

在项目中使用的是 spring security 安全框架。

实现流程：

1．配置文件实现，只需要在配置文件中指定拦截的 url 所需要权限、配置 userDetailsService 指定用户名、密码、对应权限，就可以实现。（2）

2．实现 UserDetailsService，loadUserByUsername(String userName)方法，根据 userName 来实现自己的业务逻辑返回 UserDetails 的实现类，需要自定义 User 类实现 UserDetails，比较重要的方法是 getAuthorities()，用来返回该用户所拥有的权限。（2）

3．通过自定义 filter 重写 spring security 拦截器，实现动态过滤用户权限。（2）

4．通过自定义 filter 重写 spring security 拦截器，实现自定义参数来检验用户，并且过滤权限。（4）

14. 如何处理 activeMQ 消息丢失的问题?

1．首先，点对点模式默认方式支持消息发送失败进行持久，如果消费接受方宕机，那么消费服务器会将消费持久化到消费服务器中，接受方从新启动，消息服务器会将持久化消息从新发送给接受方，并将消

息服务器中持久化的这条消息清除。（3）

2．数据丢失一般是在 topic 订阅模式消费丢失，解决消息丢失需要另外设置。将消息发送端设置为：PERSISTENT,持久化消息发往 JMS 消息服务器之后 ,持久化数据。以保证消息服务器拓机造成的消息丢失。（7）

15. token 校验的过程?

1．客户端使用用户名跟密码请求登录 （1）

2．服务端收到请求，去验证用户名与密码 （1）

3．验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端 （1）

4．客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里或者 Local Storage 里（3）

5．客户端每次向服务端请求资源的时候需要带着服务端签发的 Token （3）

6．服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据 （1）

16. solr 和数据库怎么交互的？

我们这边主要做的是把数据库导入到 solr 索引库中，实现前台的数据展示功能。首先在 schema.xml 中配置对应数据库表的字段设置，在后台创建一个对应表结构的 pojo,比如我们在商品上架的时候，先查询要上架的商品数据，然后把查询到的数据以 hash 格式添加到 solr 索引库中。

17. redis 缓存与数据库同步，怎么做的？

主要是在修改数据库的时候，让 redis 中缓存与数据库修改后的数据保持一致。（1）

1．添加：由于已经写好了查询操作，所以添加数据后，根据 id 把相对应的缓冲清空。（3）

2．删除：由于广告轮播不止一条数据，所以，在遍历删除的时候根据 id 查询出对象，作为缓存删除使用，先删除数据库里真实数据，然后根据分类 id 清空 redis 中缓存数据。（3）

3．修改：首先明确，页面修改时也会进行广告分类修改，所以需要需要把修改前的缓存数据清空，还要清空新分类中的缓存，最后进行修改数据库操作。用户访问页面，请求在缓存中没有获取到数据，会查询数据库，并保存到缓存中一份。（3）

18. fastDFS 的执行流程，你清楚么？

FastDFS 是一个开源的轻量级分布式文件系统，它对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。（1）

FastDFS 服务端有两个角色：跟踪器（tracker）和存储节点（storage）。跟踪器主要做调度工作，在访问上起负载均衡的作用。存储节 点存储文件，完成文件管理的所有功能：存储、同步和提供存取接口，FastDFS 同时对文件的 metadata 进行管理。（3）

① 上传文件交互过程：

1. client 询问 tracker 上传到的 storage，不需要附加参数；
2. tracker 返回一台可用的 storage；
3. client 直接和 storage 通讯完成文件上传。（3）

② 下载文件交互过程：

1. client 询问 tracker 下载文件的 storage，参数为文件标识（卷名和文件名）；

2. tracker 返回一台可用的 storage；

3. client 直接和 storage 通讯完成文件下载。（3）

19. zookeeper 集群，全部都挂了，怎么办？

这个题目，主要还是想问的是 zookeeper 挂了怎么办。

启动 dubbo 时，消费者会从 zk 拉取注册的生产者的地址接口等数据，缓存在本地。每次调用时，按照本地存储的地址进行调用。但是在注册中心全部挂掉后增加新的提供者，则不能被消费者发现。（1）

1．监控中心宕掉不影响使用，只是丢失部分采样数据（1）

2．数据库宕掉后，注册中心仍能通过缓存提供服务列表查询，但不能注册新服务 （1）

3．注册中心对等集群，任意一台宕掉后，将自动切换到另一台（1）

4．注册中心全部宕掉后，服务提供者和服务消费者仍能通过本地缓存通讯（1）

5．服务提供者无状态，任意一台宕掉后，不影响使用（1）

6．服务提供者全部宕掉后，服务消费者应用将无法使用，并无限次重连等待服务提供者恢复，只能重启服务器。（4）

7．

20. 如何解决购物车内存大小的问题？

1．Session（Memcached）方式

优点：购物车信息保存在服务端，可以保存 1M 信息。

缺点：对于大型网站会占有过多的服务器内存资源，造成服务器压力过大。Session 保存的信息会在用户退出登录后丢失。用户下次登录，购物车中商品信息丢失，用户只能从新选择。（3）

2．Cookie 方式

优点：购物车信息存储在客户端，不占用服务器资源，基本可以到达持久化存储。

缺点：Cookie 有大小的限制，不能超过 4K，而且不够安全。如果是个人 PC 机，Cookie 能很好的保存购物车信息，但如果是公共办公环境，Cookie 保存的信息基本就失效了(会被其他人购物车信息覆盖)。对一个大型的电子商务网站，我们需要对用户的购买行为进行分析，需要对用户推荐用户感兴趣的商品，如果把购物车信息保存在 Cookie 中，则不能对用户购买行为分析统计。（3）

3．数据库存储

优点：持久化存储，可以分析用户购买行为。

缺点: 网站速度变慢，成本和维护增加。（4）