面试宝典

目录

基础.....4
1.说一下 ArrayList 底层实现方式？.....4
2.说一下 LinkedList 底层实现方式？.....4
3.说一下 HashMap 底层实现方式？.....5
4.HashMap 和 Hashtable 的区别？.....5
5.关于 HashMap 面试题.....5
6.ArrayList 集合加入 1 万条数据，应该怎么提高效率？.....8
7.io 和 nio 的区别？.....8
8.多线程的安全问题如何保证？.....10
9.懒汉式和饿汉式，哪些是线程安全的？以及懒汉式和饿汉式的区别？.....10
10.反射怎么理解？说一下反射经典的应用.....12
Web.....13
1.session 和 cookie 我们一般都用来做什么？.....13
2.Http 协议中有那些请求方式？.....13
3.get 与 post 请求区别？.....14
4 常见 Http 协议状态？.....14
5.Servlet 的执行流程。doGet 和 doPost 的区别？.....15
6.Jsp 的重定向和转发的流程有什么区别？.....15
7.Jsp 的九大内置对象，三大指令，七大动作的具体功能？.....15
8.request ，response，session 和 application 是怎么用的？.....16
数据库.....16
1.说一下对多线程死锁的理解，举个例子，说一下什么情况下会出现死锁？.....16
2.简要描述一下数据库的事务.....17
3.乐观锁和悲观锁的解释及其应用场景.....17
框架.....18
1.hibernate 的 get（）和 load（）方法的区别？.....18
2.描述 spring 中的编程式事务处理和声明式事务处理.....18
3.spring 设置为单例，线程安全问题怎么解决？.....18
4.struts 可以是单例的吗？ 为什么？.....19
5.SpringMVC 工程流程.....19
6.Hibernate 中 get 和 load 有什么不同之处?.....20
7.hibernate 的懒加载有几种禁用方法？.....20
8.Hibernate 工作原理及为什么要用？.....20
实际开发.....21
1.简要描述一下数据库的事务.....21
2.如果项目已经上线了你，但是出现了问题。主要是怎么解决，或者你们怎样找出问题所在的（日志方向），这块你接触过吗？.....21
3.购物车如何做的？.....21
4. 是否了解工作流？.....23
5. 使用 SVN 时发生冲突，如何解决？.....23
6.Redis 有哪些数据结构? set 结构的应用场景.....23

7.未登录时购物车信息，存在 cookie 中？还有哪些方式？ 24
8. redis 的 hash 类型在项目中的使用场景..... 24

基础

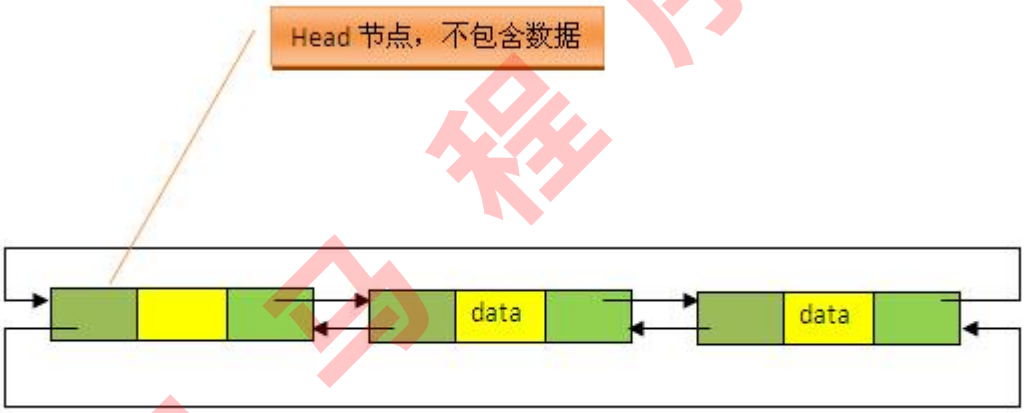
1.说一下 ArrayList 底层实现方式？

①ArrayList 通过数组实现，一旦我们实例化 ArrayList 无参数构造函数默认为数组初始化长度为 10

②add 方法底层实现如果增加的元素个数超过了 10 个，那么 ArrayList 底层会新生成一个数组，长度为原数组的 1.5 倍+1，然后将原数组的内容复制到新数组当中，并且后续增加的内容都会放到新数组当中。当新数组无法容纳增加的元素时，重复该过程。是一旦数组超出长度，就开始扩容数组。扩容数组调用的方法 Arrays.copyOf(objArr, objArr.length + 1);

2.说一下 LinkedList 底层实现方式？

LinkedList 底层的数据结构是基于双向循环链表的，且头结点中不存放数据，如下：



既然是双向链表，那么必定存在一种数据结构——我们可以称之为节点，节点实例保存业务数据，前一个节点的位置信息和后一个节点位置信息，如下图所示：

3.说一下 HashMap 底层实现方式？

HashMap 是由数组+链表组成

put 方法底层实现:

通过 key 的 hash 值%Entry[].length 得到该存储的下标位置，如果多个 key 的 hash 值%Entry[].length 值相同话就就会存储到该链表的后面。

4.HashMap 和 Hashtable 的区别？

总结：

hashmap	线程不安全	允许有 null 的键和值	效率高一点、	方法不是 Synchronize 的要提供外同步	有 containsvalue 和 containsKey 方法	HashMap 是 Javal.2 引进的 Map interface 的一个实现	HashMap 是 Hashtable 的轻量级实现
hashtable	线程安全	不允许有 null 的键和值	效率稍低、	方法是是 Synchronize 的	有 contains 方法方法	、 Hashtable 继承于 Dictionary 类	Hashtable 比 HashMap 要旧

5.关于 HashMap 面试题

HashMap 的工作原理是近年来常见的 Java 面试题。几乎每个 Java 程序员都知道 HashMap，都知道哪里要用 HashMap，知道 Hashtable 和 HashMap 之间的区别，那么为何这道面试题如此特殊呢？是因为这道题考察的深度很深。这题经常出现在高级或中高级面试中。投资银行更喜欢问这个问题，甚至会要求你实现 HashMap 来考察你的编程能力。ConcurrentHashMap 和其它同步集合的引入让这道题变得更加复杂。让我们开始探索的旅程吧！】

先来些简单的问题

“你用过 HashMap 吗？”“什么是 HashMap？你为什么用到它？”

几乎每个人都会回答“是的”，然后回答 HashMap 的一些特性，譬如 HashMap 可以接受 null 键值和值，而 Hashtable 则不能；HashMap 是非 synchronized;HashMap 很快；以及 HashMap 储存的是键值对等等。这显示出你已经用过 HashMap，而且对它相当的熟悉。但是面试官来个急转直下，从此刻开始问出一些刁钻的问题，关于 HashMap 的更多基础的细节。面试官可能会问出下面的问题：

“你知道 HashMap 的工作原理吗？”“你知道 HashMap 的 get()方法的工作原理吗？”

你也许会回答“我没有详查标准的 Java API，你可以看看 Java 源代码或者 Open JDK。”“我可以用 Google 找到答案。”

但一些面试者可能可以给出答案，“HashMap 是基于 hashing 的原理，我们使用 put(key, value)存储对象到 HashMap 中，使用 get(key)从 HashMap 中获取对象。当我们给 put()方法传递键和值时，我们先对键调用 hashCode()方法，返回的 hashCode 用于找到 bucket 位置来储存 Entry 对象。”这里关键点在于指出，HashMap 是在 bucket 中储存键对象和值对象，作为 Map.Entry。这一点有助于理解获取对象的逻辑。如果你没有意识到这一点，或者错误的认为仅仅只在 bucket 中存储值的话，你将不会回答如何从 HashMap 中获取对象的逻辑。这个答案相当的正确，也显示出面试者确实知道 hashing 以及 HashMap 的工作原理。但是这仅仅是故事的开始，当面试官加入一些 Java 程序员每天要碰到的实际场景的时候，错误的答案频现。下个问题可能是关于 HashMap 中的碰撞探测(collision detection)以及碰撞的解决方法：

“当两个对象的 hashcode 相同会发生什么？”

从这里开始，真正的困惑开始了，一些面试者会回答因为 hashcode 相同，所以两个对象是相等的，HashMap 将会抛出异常，或者不会存储它们。然后面试官可能会提醒他们有 equals()和 hashCode()两个方法，并告诉他们两个对象就算 hashcode 相同，但是它们可能并不相等。一些面试者可能就此放弃，而另外一些还能继续挺进，他们回答“因为 hashcode 相同，所以它们的 bucket 位置相同，‘碰撞’会发生。因为 HashMap 使用链表存储对象，这个 Entry(包含有键值对的 Map.Entry 对象)会存储在链表中。”这个答案非常的合理，虽然有很多种处理碰撞的方法，这种方法是最简单的，也正是 HashMap 的处理方法。但故事还没有完结，面试官会继续问：

“如果两个键的 hashcode 相同，你如何获取值对象？”

面试者会回答：当我们调用 get()方法，HashMap 会使用键对象的 hashcode 找到 bucket 位置，然后获取值对象。面试官提醒他如果有两个值对象储存在同一个 bucket，他给出答案:将会遍历链表直到找到值对象。面试官会问因为你并没有值对象去比较，你是如何确定确定找到值对象的？除非面试者直到 HashMap 在链表中存储的是键值对，否则他们不可能回答出这一题。
其中一些记得这个重要知识点的面试者会说，找到 bucket 位置之后，会调用 keys.equals()方法去找到链表中正确的节点，最终找到要找的值对象。完美的答案！
许多情况下，面试者会在这个环节中出错，因为他们混淆了 hashCode()和 equals()方法。因为在此之前 hashCode()屡屡出现，而 equals()方法仅仅在获取值对象的时候才出现。一些优秀的开发者会指出使用不可变的、声明作 final 的对象，并且采用合适的 equals()和 hashCode()方法的话，将会减少碰撞的发生，提高效率。不可变性使得能够缓存不同键的 hashcode，这将提高整个获取对象的速度，使用 String，Interger 这样的 wrapper 类作为键是非常好的选择。

如果你认为到这里已经完结了，那么听到下面这个问题的时候，你会大吃一惊。

“如果 HashMap 的大小超过了负载因子(load factor)定义的容量，怎么办？”

除非你真正知道 HashMap 的工作原理，否则你将回答不出这道题。默认的负载因子大小为 0.75，也就是说，当一个 map 填满了 75%的 bucket 时候，和其它集合类(如 ArrayList 等)一样，将会创建原来 HashMap 大小的两倍的 bucket 数组，来重新调整 map 的大小，并将原来的对象放入新的 bucket 数组中。这个过程叫作 rehashing，因为它调用 hash 方法找到新的 bucket 位置。
如果你能够回答这道问题，下面的问题来了：

“你了解重新调整 HashMap 大小存在什么问题吗？”

你可能回答不上来，这时面试官会提醒你当多线程的情况下，可能产生条件竞争(race condition)。
当重新调整 HashMap 大小的时候，确实存在条件竞争，因为如果两个线程都发现 HashMap 需要重新调整大小了，它们会同时试着调整大小。在调整大小的过程中，存储在链表中的元素的次序会反过来，因为移动到新的 bucket 位置的时候，HashMap 并不会将元素放在链表的尾部，而是放在头部，这是为了避免尾部遍历(tail traversing)。如果条件竞争发生了，那么就死循环了。这个时候，你可以质问面试官，为什么这么奇怪，要在多线程的环境下使用 HashMap 呢？：）
热心的读者贡献了更多的关于 HashMap 的问题：

为什么 String, Interger 这样的 wrapper 类适合作为键？

String, Interger 这样的 wrapper 类作为 HashMap 的键是再适合不过了，而且 String 最为常用。因为 String 是不可变的，也是 final 的，而且已经重写了 equals()和 hashCode()方法了。其他的 wrapper 类也有这个特点。不可变性是必要的，因为为了要计算 hashCode()，就要防止键值改变，如果键值在放入时和获取时返回不同的 hashcode 的话，那么就不能从 HashMap 中找到你想要的对象。不可变性还有其他的优点如线程安全。如果你可以仅仅通过将某个 field 声明成 final 就能保证 hashCode 是不变的，那么请这么做吧。因为获取对象的时候要用到 equals()和 hashCode()方法，那么键对象正确的重写这两个方法是非常重要的。如果两个不相等的对象返回不同的 hashcode 的话，那么碰撞的几率就会小些，这样就能提高 HashMap 的性能。

我们可以使用自定义的对象作为键吗？

这是前一个问题的延伸。当然你可能使用任何对象作为键，只要它遵守了 equals()和 hashCode()方法的定义规则，并且当对象插入到 Map 中之后将不会再改变了。如果这个自定义对象时不可变的，那么它已经满足了作为键的条件，因为当它创建之后就已经不能改变了。

我们可以使用 CocurrentHashMap 来代替 Hashtable 吗？

这是另外一个很热门的面试题，因为 ConcurrentHashMap 越来越多人用了。我们知道 Hashtable 是 synchronized 的，但是 ConcurrentHashMap 同步性能更好，因为它仅仅根据同步级别对 map 的一部分进行上锁。ConcurrentHashMap 当然可以代替 HashTable，但是 HashTable 提供更强的线程安全性。看看这篇博客查看 Hashtable 和 ConcurrentHashMap 的区别。

我个人很喜欢这个问题，因为这个问题的深度和广度，也不直接的涉及到不同的概念。让我们再来看看这些问题设计哪些知识点：

hashing 的概念

HashMap 中解决碰撞的方法

equals()和 hashCode()的应用，以及它们在 HashMap 中的重要性

不可变对象的好处

HashMap 多线程的条件竞争

重新调整 HashMap 的大小

总结

HashMap 的工作原理

HashMap 基于 hashing 原理，我们通过 put()和 get()方法储存和获取对象。当我们将键值对传递给 put()方法时，它调用键对象的 hashCode()方法来计算 hashcode，让后找到 bucket 位置来储存值对象。当获取对象时，通过键对象的 equals()方法找到正确的键值对，然后返回值对象。HashMap 使用链表来解决碰撞问题，当发生碰撞了，对象将会储存在链表的下一个节点中。 HashMap 在每个链表节点中储存键值对对象。

当两个不同的键对象的 hashcode 相同时会发生什么？

它们会储存在同一个 bucket 位置的链表中。键对象的 equals()方法用来找到键值对。

因为 HashMap 的好处非常多，我曾经在电子商务的应用中使用 HashMap 作为缓存。因为金融领域非常多的运用 Java，也出于性能的考虑，我们会经常用到 HashMap 和 ConcurrentHashMap。你可以查看更多的关于 HashMap 的文章:

6.ArrayList 集合加入 1 万条数据，应该怎么提高效率？

因为 ArrayList 的底层是数组实现,并且数组的默认值是 10,如果插入 10000 条要不断的扩容,耗费时间,所以我们调用 ArrayList 的指定容量的构造器方法 ArrayList(int size) 就可以实现不扩容,就提高了性能

7.io 和 nio 的区别？

传统的 socket IO 中，需要为每个连接创建一个线程，当并发的连接数量非常巨大时，线程所占用的栈内存和 CPU 线程切换的开销将非常巨大。使用

NIO，不再需要为每个线程创建单独的线程，可以用一个含有限数量线程的线程池，甚至一个线程来为任意数量的连接服务。由于线程数量小于连接数量，所以每个线程进行 IO 操作时就不能阻塞，如果阻塞的话，有些连接就得不到处理，NIO 提供了这种非阻塞的能力。

少量的线程如何同时为大量连接服务呢，答案就是就绪选择。这就好比到餐厅吃饭，每来一桌客人，都有一个服务员专门为你服务，从你到餐厅到结帐走人，这样方式的好处是服务质量好，一对一的服务，VIP 啊，可是缺点也很明显，成本高，如果餐厅生意好，同时来 100 桌客人，就需要 100 个服务员，那老板发工资的时候得心痛死了，这就是传统的一个连接一个线程的方式。

老板是什么人啊，精着呢。这老板就得捉摸怎么能用 10 个服务员同时为 100 桌客人服务呢，老板就发现，服务员在为客人服务的过程中并不是一直都忙着，客人点完菜，上完菜，吃着的这段时间，服务员就闲下来了，可是这个服务员还是被这桌客人占用着，不能为别的客人服务，用华为领导的话说，就是工作不饱满。那怎么把这段闲着的时间利用起来呢。这餐厅老板就想了一个办法，让一个服务员（前台）专门负责收集客人的需求，登记下来，比如有客人进来了、客人点菜了，客人要结帐了，都先记录下来按顺序排好。每个服务员到这里领一个需求，比如点菜，就拿着菜单帮客人点菜去了。点好菜以后，服务员马上回来，领取下一个需求，继续为别人客人服务去了。这种方式服务质量就不如一对一的服务了，当客人数据很多的时候可能需要等待。但好处也很明显，由于在客人正吃饭着的时候服务员不用闲着了，服务员这个时间内可以为其他客人服务了，原来 10 个服务员最多同时为 10 桌客人服务，现在可能为 50 桌，60 客人服务了。

这种服务方式跟传统的区别有两个：
1、增加了一个角色，要有一个专门负责收集客人需求的人。NIO 里对应的就是 Selector。
2、由阻塞服务方式改为非阻塞服务了，客人吃着的时候服务员不用一直侯在客人旁边了。传统的 IO 操作，比如 read()，当没有数据可读的时候，线程一直阻塞被占用，直到数据到来。NIO 中没有数据可读时，read()会立即返回 0，线程不会阻塞。
NIO 中，客户端创建一个连接后，先要将连接注册到 Selector，相当于客人进入餐厅后，告诉前台你要用餐，前台会告诉你你的桌号是几号，然后你就可能到那张桌子坐下了，SelectionKey 就是桌号。当某一桌需要服务时，前台就记录哪一桌需要什么服务，比如 1 号桌要点菜，2 号桌要结帐，服务员从前台取

一条记录，根据记录提供服务，完了再来取下一条。这样服务的时间就被最有效的利用起来了。

8.多线程的安全问题如何保证？

我们在考虑过线程安全问题的时候，要避免不必要的同步；过多的同步会造成死锁以及昂贵的锁竞争代价。除此以外，程序设计的时候，如果不得已面临线程安全的情况，要采取对应的措施；能保证线程的机制是锁机制、AQS 机制，对象头补齐机制等；根据环境和特定的业务挑选最合适的方法。

我们常用的有三种，同步代码块，同步方法，同步锁

除了 sync 锁之外，Java 提供了一些线程安全容器，要活用这些容器。

对于 Java 而言，线程安全工具已经足够多了。基本上能满足需求。

（1）同步代码块：(代码见附件 8-1)

在代码块声明上 加上 synchronized

```
synchronized (锁对象) {
    可能会产生线程安全问题的代码
}
```

同步代码块中的锁对象可以是任意的对象；但多个线程时，要使用同一个锁对象才能够保证线程安全。

（2）同步方法：(代码见附件 8-2)

在方法声明上加上 synchronized

```
public synchronized void method(){
    可能会产生线程安全问题的代码
}
```

同步方法中的锁对象是 this

静态同步方法：在方法声明上加上 static synchronized

静态同步方法中的锁对象是 类名.class

（3）同步锁(代码见附件 8-3)

Lock 接口提供了与 synchronized 关键字类似的同步功能，但需要在使用时手动获取锁和释放锁。

9.懒汉式和饿汉式，哪些是线程安全的？以及懒汉式和饿汉式的区别？

第一种（懒汉，线程不安全）：(代码见附件 9-1)

这种写法 lazy loading 很明显，但是致命的是在多线程不能正常工作。

第二种（懒汉，线程安全）：(代码见附件 9-2)

这种写法能够在多线程中很好的工作，而且看起来它也具备很好的 lazy loading，但是，遗憾的是，效率很低，99%情况下不需要同步。

第三种（饿汉）：(代码见附件 9-3)

这种方式基于 classloder 机制避免了多线程的同步问题，不过，instance 在类装载时就实例化，虽然导致类装载的原因有很多种，在单例模式中大多数都是调用 getInstance 方法， 但是也不能确定有其他的方式（或者其他的静态方法）导致类装载，这时候初始化 instance 显然没有达到 lazy loading 的效果。

第四种（饿汉，变种）：(代码见附件 9-4)

表面上看起来差别挺大，其实更第三种方式差不多，都是在类初始化即实例化 instance。

第五种（静态内部类）：(代码见附件 9-5)

这种方式同样利用了 classloder 的机制来保证初始化 instance 时只有一个线程，它跟第三种和第四种方式不同的是（很细微的差别）：第三种和第四种方式是只要 Singleton 类被装载了，那么 instance 就会被实例化（没有达到 lazy loading 效果），而这种方式是 Singleton 类被装载了，instance 不一定被初始化。因为 SingletonHolder 类没有被主动使用，只有显示通过调用 getInstance 方法时，才会显示装载 SingletonHolder 类，从而实例化 instance。想象一下，如果实例化 instance 很消耗资源，我想让他延迟加载，另外一方面，我不希望在 Singleton 类加载时就实例化，因为我不能确保 Singleton 类还可能在其他的地方被主动使用从而被加载，那么这个时候实例化 instance 显然是不合适的。这个时候，这种方式相比第三和第四种方式就显得很合理。

第六种（枚举）：(代码见附件 9-6)

这种方式是 Effective Java 作者 Josh Bloch 提倡的方式，它不仅能避免多线程同步问题，而且还能防止反序列化重新创建新的对象，可谓是很坚强的壁垒

啊，不过，个人认为由于 1.5 中才加入 enum 特性，用这种方式写不免让人感觉生疏，在实际工作中，我也很少看见有人这么写过。

第七种（双重校验锁）：(代码见附件 9-7)

从加载时机区分
饿汉就是类一旦加载，就把单例初始化完成，保证 getInstance 的时候，单例是已经存在的了。
而懒汉比较懒，只有当调用 getInstance 的时候，才回去初始化这个单例。

从线程是否安全区分
饿汉式天生就是线程安全的，可以直接用于多线程而不会出现问题，
懒汉式本身是非线程安全的，为实现线程安全需要改变几种写法。

从资源和性能
饿汉式在类创建的同时就实例化一个静态对象出来，不管之后会不会使用这个单例，都会占据一定的内存，但是相应的，在第一次调用时速度也会更快，因为其资源已经初始化完成，而懒汉式顾名思义，会延迟加载，在第一次使用该单例的时候才会实例化对象出来，第一次调用时要做初始化，如果要做的工作比较多，性能上会有些延迟，之后就和饿汉式一样了。

10.反射怎么理解？说一下反射经典的应用

反射是什么呢？当我们的程序在运行时，需要动态的加载一些类这些类可能之前用不到所以不用加载到 jvm，而是在运行时根据需要才加载，这样的好处对于服务器来说不言而喻，举个例子我们的项目底层有时是用 mysql，有时用 oracle，需要动态地根据实际情况加载驱动类，这个时候反射就有用了，假设 com.java.dbtest.myqlConnection，com.java.dbtest.oracleConnection 这两个类我们要用，这时候我们的程序就写得比较动态化，通过 Class tc = Class.forName("com.java.dbtest.TestConnection");通过类的全类名让 jvm 在服务器中找到并加载这个类，而如果是 oracle 则传入的参数就变成另一个了。这时候就可以看到反射的好处了，这个动态性就体现出 java 的特性了！举个例子，使用 spring 中会发现当你配置各种各样的 bean 时，是以配置文件的形式配置的，你需要用到哪些 bean 就配哪些，spring 容器就会根据你的需求去动态加载，你的程序就能健壮地运行。

Web

1.session 和 cookie 我们一般都用来做什么？

cookie 是一个非常具体的东西，指的就是浏览器里面能永久存储的一种数据。跟服务器没啥关系，仅仅是浏览器实现的一种数据存储功能。
session 从字面上讲，就是会话。这个就类似于你和一个人交谈，你怎么知道当前和你交谈的是张三而不是李四呢？对方肯定有某种特征（长相等）表明他就是张三。session 也是类似的道理，服务器要知道当前发请求给自己的是谁。为了做这种区分，服务器就要给每个客户端分配不同的“身份标识”，然后客户端每次向服务器发请求的时候，都带上这个“身份标识”，服务器就知道这个请求来自于谁了。至于客户端怎么保存这个“身份标识”，可以有很多种方式，对于浏览器客户端，大家都默认采用 cookie 的方式。
cookie 存于客户端。session 存于服务器端。服务器鉴别 session 需要至少从客户端传来一个 session_id，session_id 通常存于 cookie 中。所以在工程上 session 离了 cookie 基本没法用，但是 cookie 可以单独使用，不过 cookies 是明文存储，安全性很低，只使用 cookie 的话盗取了 cookie 基本就获取了用户所有权限。
面试官问问题的时候喜欢顺藤摸瓜，先问 cookie 和 session 的区别，然后可能问，用 cookie 和 session 存什么东西，所以在复习知识的时候，要学会自己将知识串联，要学会揣测面试会问什么
这时候面试官会问到，请说一下，在项目中 cookie 和 session 是怎么用的，这时候就可以将
项目中登录的模块结合 cookie 说一说，就能过渡到自己熟悉的模块了。
当会员需要登录的时候，首先进入的是“登录页面”，输入用户名、密码、验证码进行登录，进入单点登录系统（Controller）Service Mapper DB 提交登录页面，用户信息被提交到单点登录系统进行校验。如果登陆信息不正确，校验错误，将错误的登陆信息进行回显返回给用户，重新登录。验证通过，就将用户信息保存到 redis 中，并且生成 token 作为一个标识，并将 token 保存到 cookie 里面，发送给用户浏览器，下次请求的时候就会带回来，获取 cookie 里面的 token，根据 token 再去 redis 中查询对应的用户信息。登录成功后将重定向到登录之前的访问页面。

2.Http 协议中有那些请求方式？

GET： 用于请求访问已经被 URI（统一资源标识符）识别的资源，可以通过 URL 传参给服务器

POST：用于传输信息给服务器，主要功能与 GET 方法类似，但一般推荐使用 POST 方式。
PUT： 传输文件，报文主体中包含文件内容，保存到对应 URI 位置。
HEAD： 获得报文首部，与 GET 方法类似，只是不返回报文主体，一般用于验证 URI 是否有效。
DELETE：删除文件，与 PUT 方法相反，删除对应 URI 位置的文件。
OPTIONS：查询相应 URI 支持的 HTTP 方法。

3.get 与 post 请求区别？

区别一：
get 重点在从服务器上获取资源，post 重点在向服务器发送数据；
区别二：
get 传输数据是通过 URL 请求，以 field（字段）= value 的形式，置于 URL 后，并用"?"连接，多个请求数据间用"&"连接，如 http://127.0.0.1/Test/login.action?name=admin&password=admin，这个过程用户是可见的；
post 传输数据通过 Http 的 post 机制，将字段与对应值封存在请求实体中发送给服务器，这个过程对用户是不可见的；
区别三：
Get 传输的数据量小，因为受 URL 长度限制，但效率较高；
Post 可以传输大量数据，所以上传文件时只能用 Post 方式；
区别四：
get 是不安全的，因为 URL 是可见的，可能会泄露私密信息，如密码等；
post 较 get 安全性较高；
区别五：
get 方式只能支持 ASCII 字符，向服务器传的中文字符可能会乱码。
post 支持标准字符集，可以正确传递中文字符。

4 常见 Http 协议状态？

200：请求被正常处理
204：请求被受理但没有资源可以返回
206：客户端只是请求资源的一部分，服务器只对请求的部分资源执行 GET 方法，相应报文中通过 Content-Range 指定范围的资源。
301：永久性重定向
302：临时重定向
303：与 302 状态码有相似功能，只是它希望客户端在请求一个 URI 的时候，能通过 GET 方法重定向到另一个 URI 上
304：发送附带条件的请求时，条件不满足时返回，与重定向无关

307：临时重定向，与 302 类似，只是强制要求使用 POST 方法
400：请求报文语法有误，服务器无法识别
401：请求需要认证
403：请求的对应资源禁止被访问
404：服务器无法找到对应资源
500：服务器内部错误
503：服务器正忙

5.Servlet 的执行流程。doGet 和 doPost 的区别？

Servlet 的执行流程也就是 servlet 的生命周期，当服务器启动的时候生命周期开始，然后通过 init()《启动顺序根据 web.xml 里的 startup-on-load 来确定加载顺序》方法初始化 servlet，再根据不同请求调用 doGet 或 doPost 方法，最后再通过 destroy()方法进行销毁。
doGet 和 doPost 都是接受用户请求的方法，doGet 处理 get 请求，doPost 处理 post 请求，doGet 用于地址栏提交，doPost 用于表单提交，在页面提交数据时，get 的数据大小有限制 4k，post 没有限制，get 请求提交的数据会在地址栏显示，post 不显示，所以 post 比 get 安全.

6.Jsp 的重定向和转发的流程有什么区别？

重定向是客户端行为，转发是服务器端行为
重定向时服务器产生两次请求，转发产生一次请求，重定向时可以转发到项目以外的任何网址，转发只能在当前项目里转发
重定向会导致 request 对象信息丢失。转发则不会
转发的 url 不会变,request.getRequestDispatch（）.forward()
重定向的 url 会改变,response.getRedirect();

7.Jsp 的九大内置对象，三大指令，七大动作的具体功能？

JSP 九大内置对象:

pageContext ： 只对当前 jsp 页面有效，里面封装了基本的 request 和 session 的对象
Request ： 对当前请求进行封装
Session ： 浏览器会话对象，浏览器范围内有效
Application ： 应用程序对象，对整个 web 工程都有效
Out ： 页面打印对象，在 jsp 页面打印字符串
Response ： 返回服务器端信息给用户
Config ： 单个 servlet 的配置对象，相当于 servletConfig 对象
Page ： 当前页面对象，也就是 this

Exception ： 错误页面的 exception 对象，如果指定的是错误页面，这个就是异常对象

三大指令：

Page ： 指令是针对当前页面的指令

Include ： 用于指定如何包含另一个页面

Taglib ： 用于定义和指定自定义标签

七大动作：

Forward，执行页面跳转，将请求的处理转发到另一个页面

Param ： 用于传递参数

Include ： 用于动态引入一个 jsp 页面

Plugin ： 用于下载 javaBean 或 applet 到客户端执行

useBean ： 使用 javaBean

setProperty ： 修改 javaBean 实例的属性值

getProperty ： 获取 javaBean 实例的属性值

8.request ，response，session 和 application 是怎么用的？

Request 是客户端向服务端发送请求

Response 是服务端对客户端请求做出响应

Session 在 servlet 中不能直接使用，需要通过 getSession()创建，如果没有设定它的生命周期，或者通过 invildate()方法销毁，关闭浏览器 session 就会消失

Application 不能直接创建，存在于服务器的内存中，由服务器创建和销毁

数据库

1.说一下对多线程死锁的理解，举个例子，说一下什么情况下会出现死锁？

举个生活中的例子：有一座桥很窄，一次只能通过一辆汽车，可现在有 A 和 B 两辆汽车分别从桥的两端已经驶进了桥，也就是两辆车现在分别占用了一部分桥的资源。A 要想过桥的话，需要 B 车让出桥面资源。同理 B 车想要过桥，也需要 A 车让出桥面资源。两边的车都不倒车，结果造成互相等待对方让出桥面，但是谁也不让路，就会无休止地等下去。这种现象就是死锁。如果把汽车比做进程，桥面作为资源，那上述问题就描述为：进程 A 占有资源 R1，等待进程 B 占有的资源 R2；进程 B 占有资源 R2，等待进程 A 占有的资源 R1。而且资

源 R1 和 R2 只允许一个进程占用，即：不允许两个进程同时占用。结果，两个进程都不能继续执行，若不采取其它措施，这种循环等待状况会无限期持续下去，就发生了进程死锁。
在多线程关于死锁的问题上，我们可以借助现实生活中的例子来辅助理解，但是我们在给面试官讲解知识点的时候，要将理解消化，尽量的多用术语。

2.简要描述一下数据库的事务

事务是并发控制的基本单位。所谓的事务，它的根本是一个操作序列，这些操作都执行，或者都不执行，它是一个无法分割的工作单位。
例如银行转账:从一个账户扣款并使另一个账户赠款，这两个操作要么都执行，要么都不执行，不能存在执行一半，否则会出现金额消失或者无中生有。所以我们能够把整个操作的过程，看成一个事务。

3.乐观锁和悲观锁的解释及其应用场景

悲观锁(Pessimistic Lock)，顾名思义，就是很悲观，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会 block 直到它拿到锁。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。

乐观锁(Optimistic Lock)，顾名思义，就是很乐观，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号等机制。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库如果提供类似于 write_condition 机制的其实都是提供的乐观锁。

使用场景
两种锁各有优缺点，不可认为一种好于另一种，像乐观锁适用于写比较少的情况下，即冲突真的很少发生的时候，这样可以省去了锁的开销，加大了系统的整个吞吐量。但如果经常产生冲突，上层应用会不断的进行 retry，这样反倒是降低了性能，所以这种情况下用悲观锁就比较合适。

问到悲观锁和乐观锁，面试官主要是想考查求职者对比使用数据库锁来保证多用户并发访问时数据库安全的问题。在回答这个问题的时候，可以从锁的名称来解析去两个锁的本质区别，然后面试官可能会问到两个锁那个更好，在回答框架或者某种技术那个更好的时候，我们要记住一点，就是没有最好的，只有最适合的。

框架

1.hibernate 的 get（）和 load（）方法的区别？

返回值：
get()返回的是查询出来的实体对象，而 load()查询出来的是一个目标实体的代理对象。
查询时机：
get()在调用的时候就立即发出 SQL 语句查询，而 load()在访问非 ID 属性的时候才会发出查询语句并且将被代理对象 target 填充上，但是如果这个动作发生在 Session 被关闭后的话就会抛出 LazyInitializationException。
查询结果为空时：
get()抛出 NullPointerException
load()抛出 ObjectNotFoundException

2.描述 spring 中的编程式事务处理和声明式事务处理

编程式事务需要你在代码中直接加入处理事务的逻辑,可能需要在代码中显式调用 beginTransaction()、commit()、rollback()等事务管理相关的方法,如在执行 a 方法时候需要事务处理,你需要在 a 方法开始时候开启事务,处理完后。在方法结束时候,关闭事务.

声明式的事务的做法是在 a 方法外围添加注解或者直接在配置文件中定义,a 方法需要事务处理,在 spring 中会通过配置文件在 a 方法前后拦截,并添加事务.

二者区别.编程式事务侵入性比较强，但处理粒度更细.

3.spring 设置为单例，线程安全问题怎么解决？

spring 的单例和我们设计模式中的单例还是有些区别的，设计模式中的单例是在整个应用中只存在一个单例，而 spring 中的单例是存在 IOC 容器中只有一个实例，spring 中的单例并不会影响到应用的并发访问，通常我们 spring 中的线程安全问题是指的在业务逻辑中的那个问题，ThreadLocal 则从另一个角度来解决多线程的并发访问。ThreadLocal 会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal 提供了

线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal。

4.struts 可以是单例的吗？ 为什么？

在 Struts2 中 action 必须为多例，主要原因在于请求数据和返回数据大部分封装到了 action 中，用于实例化变量，这样以来，如果将 action 设置成单例模式，则多个线程访问的时候，就会共享这些数据，从而引起数据混乱或者线程安全问题！

5.SpringMVC 工程流程

1. 用户向服务器发送请求，请求被 Spring 前端控制 ServeltDispatcherServlet 捕获；
2. DispatcherServlet 对请求 URL 进行解析，得到请求资源标识符（URI）。然后根据该 URI，调用 HandlerMapping 获得该 Handler 配置的所有相关的对象（包括 Handler 对象以及 Handler 对象对应的拦截器），最后以 HandlerExecutionChain 对象的形式返回；
3. DispatcherServlet 根据获得的 Handler，选择一个合适的 HandlerAdapter。（附注：如果成功获得 HandlerAdapter 后，此时将开始执行拦截器的 preHandler(...)方法）
4. 提取 Request 中的模型数据，填充 Handler 入参，开始执行 Handler（Controller)。 在填充 Handler 的入参过程中，根据你的配置，Spring 将帮你做一些额外的工作：
HttpMessageConveter：将请求消息（如 Json、xml 等数据）转换成一个对象，将对象转换为指定的响应信息
数据转换：对请求消息进行数据转换。如 String 转换成 Integer、Double 等
数据根式化：对请求消息进行数据格式化。 如将字符串转换成格式化数字或格式化日期等
数据验证： 验证数据的有效性（长度、格式等），验证结果存储到 BindingResult 或 Error 中
5. Handler 执行完成后，向 DispatcherServlet 返回一个 ModelAndView 对象；
6. 根据返回的 ModelAndView，选择一个适合的 ViewResolver（必须是已经注册到 Spring 容器中的 ViewResolver)返回给 DispatcherServlet ；
7. ViewResolver 结合 Model 和 View，来渲染视图
8. 将渲染结果返回给客户端

6.Hibernate 中 get 和 load 有什么不同之处?

load 找不到数据的话会抛出 org.hibernate.ObjectNotFoundException 异常。此时 hibernate 会使用延迟加载加载机制,get 找不到的话会返回 null。
如果查询不到数据，get 会返回 null,但是不会报错,load 如果查询不到数据,则报错 ObjectNotFoundException。
使用 get 去查询数据，(先到一级/二级)会立即向 db 发出查询请求(select...),如果你使用的是 load 查询数据，(先到一级、二级))即使查询到对象，返回的是一个代理对象,如果后面没有使用查询结果，它不会真的向数据库发 select,当程序员使用查询结果的时候才真的发出 select,这个现象我们称为懒加载(lazy)。

7.hibernate 的懒加载有几种禁用方法？

在 Hibernate 框架中，当我们要访问的数据量过大时，明显用缓存不太合适，因为内存容量有限 ，为了减少并发量，减少系统资源的消耗，这时 Hibernate 用懒加载机制来弥补这种缺陷，但是这只是弥补而不是用了懒加载总体性能就提高了。
我们所说的懒加载也被称为延迟加载，它在查询的时候不会立刻访问数据库，而是返回代理对象，当真正去使用对象的时候才会访问数据库。
1.使用代理对象：Hibernate.initialize("代理对象");
2.在需要禁用懒加载的映射文件中显示的加入 lazy = "false"
3.使用 openSessionInView【需要借助于过滤器】 需要在 web.xml 文件中配置

8.Hibernate 工作原理及为什么要用？

原理：
1)读取并解析配置文件
2)读取并解析映射信息
3)创建 SessionFactory
4)打开 Sesssion
5)创建事务 Transation
6)持久化操作
7)提交事务
8)关闭 Session
9)关闭 SesstionFactory
为什么要用：
1)对 JDBC 访问数据库的代码做了封装，大大简化了数据访问层繁琐的重复性代码。

2）Hibernate 是一个基于 JDBC 的主流持久化框架，是一个优秀的 ORM 实现，他很大程度的简化 DAO 层的编码工作。
3）hibernate 使用 Java 反射机制，而不是字节码增强程序来实现透明性。
4）hibernate 的性能非常好，因为它是个轻量级框架。映射的灵活性很出色。它支持各种关系数据库，从一对一到多对多的各种复杂关系。

实际开发

1.简要描述一下数据库的事务

事务是并发控制的基本单位。所谓的事务，它的根本是一个操作序列，这些操作都执行，或者都不执行，它是一个无法分割的工作单位。
例如银行转账:从一个账户扣款并使另一个账户赠款，这两个操作要么都执行，要么都不执行，不能存在执行一半，否则会出现金额消失或者无中生有。所以我们能够把整个操作的过程，看成一个事务。

2.如果项目已经上线了你，但是出现了问题。主要是怎么解决，或者你们怎样找出问题所在的（日志方向），这块你接触过吗？

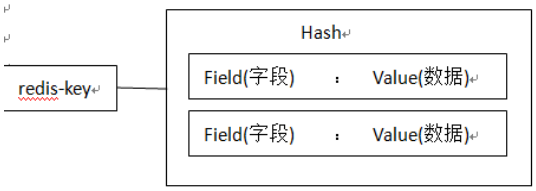
通过引用 log4j 日志包来查看，在每个类中引用相应的日志来记录，在服务器中留出专门的空间来存储日志，后缀为.log 的文件，出现的错误会存储到日志文件中，当我们需要查看的时候通过编译软件来查看。实际开发中，这一块一般都是运维人员负责的。

3.购物车如何做的？

购物车功能分两种状态，未登录状态购物车和已登录状态购物车。
未登录状态购物车:
利用 cookie 中的 value（value 可以是一个 uuid 来保证唯一性）再加上一个前缀比如 REDIS_CART 做为 redis 的 key,然后将购物车数据转成 json 存入到 redis 中。这样就达到了京东的未登录状态下加入商品到购物车的类似功能。但是这样存数据的话，如果购物车中有很多个商品，而之后要修改购物车中某个商品的数量，我们就需要将所有的商品取出来遍历、判断、修改然后再重新

转换成 json 再存入到 redis 中，这样就有点麻烦。因此就采用了 redis 中的 hash 结构的存储方式。

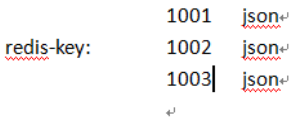
它的结构方式是：



那么我们怎么存储比较好呢？

redis-key 保持不变，然后可以将 itemId 作为字段，然后将数据存入到 value 中。

如：



如果需要修改商品数量的话直接可以通过 itemId 找到对应的商品修改而不需要全部取出了。

在购物车页面我们可以通过勾选前面的复选框来选择需要的商品进行下单，怎么实现呢，如果复选框被勾选了，在提交页面的时候将对应的商品 id 一起提交，如果全选，那就不用提交商品 id，那么在提交订单时我们只要进行判断，如果传递了商品 id，那么就指定商品下单，如果没有提交商品 id,那么就通过用户 id（在提交订单时用户肯定已经登录了）查询其下的所有订单进行全部下单。

登录状态购物车：

当我们在未登录状态下下单的时候，会跳转到登录页面，让用户先登录，这时我们就要考虑将未登录状态下的购物车数据在登录后合并到一起。

那么登录之后的合并数据的逻辑应该写在哪儿呢，由于登录功能是单独的一个工程，因此我利用了 MQ 的技术，在用户登录成功之后，发送消息，然后在购物车系统中监听到消息之后再处理数据合并。而消息内容一个是 userId，用来确定合并之后的数据保存在哪个用户的购物车中，另一个就是 cookie 中的 uuid?通过 uuid 查询到该用户在未登录状态下的购物车数据（之前购物车的优化，已经把未登录状态下的购物车数据存储到了 redis 中，而 redis 的 key 就是这个 uuid,如果用户登录之后携带的 cookie 中的 uuid 和 redis 的 key 一致，那是不是就可以合并了），然后进行数据的合并，那这个功能也就可以完成了。

需要注意的是，我们合并数据之后是不是还要把 redis 中的购物车数据删除掉，如果不删除的话，下次登录是不是又要合并了。

当我们下单之后，应该要将购物车里的数据删除掉，如果不删除的话下一下单购物车里的数据就又会增加上去了。

4. 是否了解工作流？

以请假为例，现在大多数公司的请假流程是这样的
员工打电话（或网聊）向上级提出请假申请——上级口头同意——上级将请假记录下来——月底将请 假记录上交公司——公司将请假录入电脑
采用工作流技术的公司的请假流程是这样的
员工使用账户登录系统——点击请假——上级登录系统点击允许
就这样，一个请假流程就结束了

工作流系统，实现了工作流程的自动化，提高了企业运营效率、改善企业资源利用、提高企业运作的 灵活性和适应性、提高量化考核业务处理的效率、减少浪费
工作流可以参考 OA 系统，就是工作流程的执行过程。
面试官可能会顺着工作流问到 OA 系统，顺带会问到其他你了解的系统，比如 erp 系统，物流管理系统等，求职者要对这些市面上常见的系统有所了解。

5. 使用 SVN 时发生冲突，如何解决？

A、放弃自己的更新，使用 svn revert（回滚），然后提交。在这种方式下不需要使用 svn resolved（解决）

B、放弃自己的更新，使用别人的更新。使用最新获取的版本覆盖目标文件，执行 resolved filename 并提交(选择文件—右键—解决)。

C、手动解决：冲突发生时，通过和其他用户沟通之后，手动更新目标文件。然后执行 resolved filename 来解除冲突，最后提交。
解决冲突：首先应该从版本库更新版本，然后去解决冲突，冲突解决后要执行 svn resolved（解决），然后在签入到版本库。在冲突解决之后，需要使用 svn resolved（解决）来告诉 subversion 冲突解决，这样才能提交更新。
SVN 是开发岗位每天都要用的版本控制工具，这个问题也是开发中的常见问题。面试官在问你们公司用的是什么版本控制工具的时候，是 git 还是 svn，选择自己擅长的版本控制工具进行回答。

6.Redis 有哪些数据结构? set 结构的应用场景

String list set hash zset

Set 就是一个集合，集合的概念就是一堆不重复值的组合。利用 Redis 提供的 Set 数据结构，可以存储一些集合性的数据。比如在微博应用中，可以将一个用户所有的关注人存在一个集合中，将其所有粉丝存在一个集合。因为 Redis

非常人性化的为集合提供了求交集、并集、差集等操作，那么就可以非常方便的实现如共同关注、共同喜好、二度好友等功能。

7.未登录时购物车信息，存在 cookie 中？还有哪些方式？

未登录状态下,购物车信息就是存在 cookie 中的,如果禁用 cookie 的话,servlet 引入了一种补充会话的机制 ， 当用户发出下一次请求时 ， 如果请求信息中没有包含 Cookie 头字段 ， Servlet 引擎则认为客户端不支持 cookie,它将依据请求 url 参数中的会话标识号来实施会话跟踪.
这种技术称为 url 重写，tomcat 发送给客户端的会话标识号的 Cookie 名称为 JSESSIONID。url 重写就是将 JSESSIONID 关键字作为参数名和会话标识号作为参数附加到 URL 后面。如果浏览器不支持 Cookie 或者关闭 Cookie，就必须对所有可能被客户端访问的请求路径进行 URL 重写，如超链接，form 表单的 action 属性和重定向的 URL.

response.encodeURL() 使 URL 包含 session ID，如果你需要使用重定向，可以使用 response.encodeRedirectURL ()来对 URL 进行编码。encodeURL () 及 encodeRedirectedURL () 方法首先判断 cookie 是否被浏览器支持；如果支持，则参数 URL 被原样返回，session ID 将通过 cookies 来维持。

8. redis 的 hash 类型在项目中的使用场景

我们在开发中常用 redis 存储五种数据类型，
String,Hash,List,Set,Zset(sorted set)
关于 hash 在我们实际开发中的应用，我以存储用户信息为例。

第一种： key：userID； value：username；age；birthday

第一种将用户 ID 作为查找 key,把其他信息封装成一个对象以序列化的方式存储，这种方式的缺点是，增加了序列化/反序列化的开销，并且在需要修改其中一项信息时，需要把整个对象取回，并且修改操作需要对并发进行保护，避免产生脏数据。

第二种：key：userID+name　　value：name
　　key：userID+age　　value：age
　　key：userID+birthday　　value：birthday；

第二种方法是这个用户信息对象有多少成员就存成多少个 key-value 对儿，用用户 ID+对应属性的名称作为唯一标识来取得对应属性的值，虽然省去了序列

化开销和并发问题，但是用户 ID 为重复存储，如果存在大量这样的数据，内存浪费还是非常可观的。

我们采取 hashmap 方法存储解决了上述两个问题，在存储用户信息的时候，key 依旧是 userID，但是 value 值却是一个 map，map 中依旧存在 key 和 value；key 是姓名标签，对应的 value 是具体的姓名；如果需要修改姓名值时，只加用 userID+field（姓名）就可以修改姓名值了。这样操作不需要重复存储数据，也不需要考虑并发保护问题。

在企业面试中，面试官不喜欢问一些书本上的问题，譬如 redis 中的五种数据类型，求职者将五种基本数据类型回答出来，感觉像是在背书，面试官喜欢问某种技术在项目中应用场景，第一点是考察是否有真实的开发经验，第二点是面试官想听求职者对这个技术点有什么自己的理解，所以，我们在回答这个问题的时候，不能只空洞的说 redis 的五种数据类型和特点，应该联系我们实际开发中对该技术点的运用加以说明，便能达到面试官想要的效果，这个题目以 hash 类型举例，求职者应将五种数据类型都掌握，面试官可能会顺着问到其他的数据类型。