**Java研发**

**阿里电面**

1. **Collections.sort()实现原理**

快速排序

1. **HashTable 和HashMap**

Hashtable和HashMap类似，都是散列表，存储键值对映射；Hashtable线程安全，HashMap非线程安全。。。

**3.ArrayList和LinkedList**

**4.线程安全的容器**

**Hashtable**：Hashtable的方法都是通过synchronized来进行方法层次的同步，以达到

线程安全的作用；

**ConcurrentHashMap**：是性能更好的散列表。在兼顾线程安全的同时，相对于Hashtable，在效率上有很大的提高。我们可以猜想，Hashtable的线程安全实现是对方法进行synchronized，很明显可以通过其他并发方式，如ReentrantLock进行优化。而ConcurrentHashMap正是采用了ReentrantLock。运用***锁分离技术***，即在代码块上加锁，而不是方法上加。同时ConcurrentHashMap的一个特色是*允许多个修改并发操作*。这就有意思了，我们知道一般写都是互斥的，为什么这个还能多个同时写呢？那是因为ConcurrentHashMap采用了内部使用段机制，将ConcurrentHashMap分成了很多小段。只要不在一个小段上写就可以并发写。

**CopyOnWriteArrayList**：Collection类的线程安全容器主要都是利用的ReentrantLock实现的线程安全，CopyOnWriteArrayList也不例外。在并发写的时候，需要获取lock。读的时候不需要进行lock。

**CopyOnWriteArraySet**：它的实现就是基于CopyOnWriteArrayList实现的，采用的装饰器进行实现。二者的区别和List和Set的区别一样。

**Vector**：一般我们都不用Vector了，不过它确实也是线程安全的。相对于其他容器，能够提供随机访问功能。

**StringBuffer：**是线程安全的，StringBuilder不是。从StringBuffer的源码可以看到，它采用的是对方法进行*synchronized*实现的同步。但是加了同步机制，肯定会对性能有一定影响。高并发情况下，对数据安全有需求，则用StringBuffer，否则用StringBuilder。

总结：

1.HashSet 非线程安全

2.TreeSet 非线程安全

3.LinkedHashSet 非线程安全

4.ArrayList 非线程安全

5.LinkedList 非线程安全

6.HashMap 非线程安全

7.TreeMap 非线程安全

8.LinkedHashMap 非线程安全

---------------------------

9.HashTable 线程安全

10.Vector 线程安全

**5.三次握手**

**6.HTTP是哪个层的（常见的协议位于那个层）**

**应用层**

**7.Get和Post的区别，其他请求方式**

**8.Volidate和sychronized的区别**

**9.Bio和nio的区别**

Java BIO: 同步并阻塞，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器端就需要启动一个线程进行处理，如果这个连接不做任何事情会造成不必要的线程开销，当然可以通过线程池机制改善。

Java NIO: 同步非阻塞，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有I/O请求时才启动一个线程进行处理。

<http://blog.csdn.net/wanglei_storage/article/details/50225779>

**10.4xx,3xx,5xx所具有的含义**

1XX类状态码信息表示：临时的响应。客户端在收到常规响应之前，应准备接收一个或多个1XX响应

　　2XX类状态码信息表示：服务器成功的接收了客户端请求

　　3XX类状态码信息表示：客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同页面，或者通过代理服务器重复该请求

　　4XX类状态码信息表示：发生错误，客户端似乎有问题。例如：客户端请求不存在的页面，客户端为提供有效的身份验证信息

　　5XX类状态码信息表示：服务器遇到错误而不能完成该请求

**11.Optional**

这是一个可以为null的容器对象。如果值存在则isPresent()方法会返回true，调用get()方法会返回该对象。

**12.CAS**

**13.ArrayList扩容**

ArrayList list = new ArrayList();初始容量10，每次扩容为原来的1.5倍,加1，即int newCapacity = (oldCapacity \* 3)/2 + 1;

14.B+树

**15.轻量级锁**

***轻量级锁能提高程序同步性能***的依据是“对于绝大部分的锁，在整个同步周期内都是不存在竞争的”，这是一个经验数据。如果没有竞争，轻量级锁使用CAS操作避免了使用互斥量的开销，但如果存在锁竞争，除了互斥量的开销外，还额外发生了CAS操作，因此在有竞争的情况下，轻量级锁会比传统的重量级锁更慢。

**16.偏向锁**

**优点：**消除数据在无竞争情况下的同步原语，提高性能。

*偏向锁与轻量级锁理念上的区别：*

**轻量级锁：**在无竞争的情况下使用CAS操作去消除同步使用的互斥量

**偏向锁：**在无竞争的情况下把整个同步都消除掉，连CAS操作都不做了

**意义：**锁偏向于第一个获得它的线程。如果在接下来的执行过程中，该锁没有被其他的线程获取，则持有偏向锁的线程将永远不需要再进行同步。