IO面试题：http://blog.csdn.net/t0404/article/details/51893168

1、NIO

2、IO流

包含：

（1）字节流（InputStream、OutputStream）

InputStreamReader会包含一个InputStream，从而可以将该输入字节流转换成字符流，代码例子：

```
01 InputStream inputStream = new FileInputStream("c:\\data\\input.txt");
02
03 Reader reader = new InputStreamReader(inputStream);
04
05 int data = reader.read();
06
07 while(data != -1){
08
09     char theChar = (char) data;
10
11     data = reader.read();
12
13 }
14
15 reader.close();
```

（2）字符流（Reader、Writer）

（3）转换流（InputStreamReader、OutputStreamWriter）

（4）缓冲流（BufferedReader）

BufferedReader能为字符输入流提供缓冲区，可以提高许多IO处理的速度。你可以一次读取一大块的数据，而不需要每次从网络或者磁盘中一次读取一个字节。特别是在访问大量磁盘数据时，缓冲通常会让IO快上许多。

BufferedReader和BufferedInputStream的主要区别在于，BufferedReader操作字符，而BufferedInputStream操作原始字节。只需要把Reader包装到BufferedReader中，就可以为Reader添加缓冲区(译者注：默认缓冲区大小为8192字节，即8KB)。代码如下：

```
1 Reader input = new BufferedReader(new FileReader("c:\\data\\input-file.txt"));
```

你也可以通过传递构造函数的第二个参数，指定缓冲区大小，代码如下：

```
1 Reader input = new BufferedReader(new FileReader("c:\\data\\input-file.txt"), 8 * 1024);
```

这个例子设置了8KB的缓冲区。最好把缓冲区大小设置成1024字节的整数倍，这样能更高效地利用内置缓冲区的磁盘。

除了能够为输入流提供缓冲区以外，其余方面BufferedReader基本与Reader类似。BufferedReader还有一个额外readLine()方法，可以方便地一次性读取一整行字符。

(5)打印流（PrintStream）

IO流的同步和异步的区别

1.同步与异步

同步和异步关注的是消息通信机制 (synchronous communication/ asynchronous communication)

同步，就是在发出一个*调用*时，在没有得到结果之前，该*调用*就不返回。但是一旦调用返回，就得到返回值了。
换句话说，就是由*调用者*主动等待这个*调用*的结果。

异步，*调用*在发出之后，这个调用就直接返回了，所以没有返回结果。换句话说，当一个异步过程调用发出后，调用者不会立刻得到结果。而是在*调用*发出后，*被调用者*通过状态、通知来通知调用者，或通过回调函数处理这个调用。
典型的异步编程模型比如Node.js

举个通俗的例子：
你打电话问书店老板有没有《分布式系统》这本书，如果是同步通信机制，书店老板会说，你稍等，" 我查一下"，然后开始查啊查，等查好了（可能是5秒，也可能是一天）告诉你结果（返回结果）。
而异步通信机制，书店老板直接告诉你我查一下啊，查好了打电话给你，然后直接挂电话了（不返回结果）。然后查好了，他会主动打电话给你。在这里老板通过“回电”这种方式来

回调。

阻塞与非阻塞

阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态.

阻塞调用是指调用结果返回之前，当前线程会被挂起。调用线程只有在得到结果之后才会返回。

非阻塞调用指在不能立刻得到结果之前，该调用不会阻塞当前线程。

还是上面的例子，
你打电话问书店老板有没有《分布式系统》这本书，你如果是阻塞式调用，你会一直把自己“挂起”，直到得到这本书有没有的结果，如果是非阻塞式调用，你不管老板有没有告诉你，你自己先一边去玩了， 当然你也要偶尔过几分钟check一下老板有没有返回结果。
在这里阻塞与非阻塞与是否同步异步无关。跟老板通过什么方式回答你结果无关。
归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

同步异步 阻塞非阻塞
老张爱喝茶，废话不说，煮开水。
出场人物：老张，水壶两把（普通水壶，简称水壶；会响的水壶，简称响水壶）。
1 老张把水壶放到火上，立等水开。（同步阻塞）
老张觉得自己有点傻
2 老张把水壶放到火上，去客厅看电视，时不时去厨房看看水开没有。（同步非阻塞）
老张还是觉得自己有点傻，于是变高端了，买了把会响笛的那种水壶。水开之后，能大声发出嘀~~~~的噪音。
3 老张把响水壶放到火上，立等水开。（异步阻塞）
老张觉得这样傻等意义不大
4 老张把响水壶放到火上，去客厅看电视，水壶响之前不再去看它了，响了再去拿壶。（异步非阻塞）
老张觉得自己聪明了。

所谓同步异步，只是对于水壶而言。
普通水壶，同步；响水壶，异步。
虽然都能干活，但响水壶可以在自己完工之后，提示老张水开了。这是普通水壶所不能及的。
同步只能让调用者去轮询自己（情况2中），造成老张效率的低下。
所谓阻塞非阻塞，仅仅对于老张而言。
立等的老张，阻塞；看电视的老张，非阻塞。
情况1和情况3中老张就是阻塞的，媳妇喊他都不知道。虽然3中响水壶是异步的，可对于立等的老张没有太大的意义。所以一般异步是配合非阻塞使用的，这样才能发挥异步的效用。

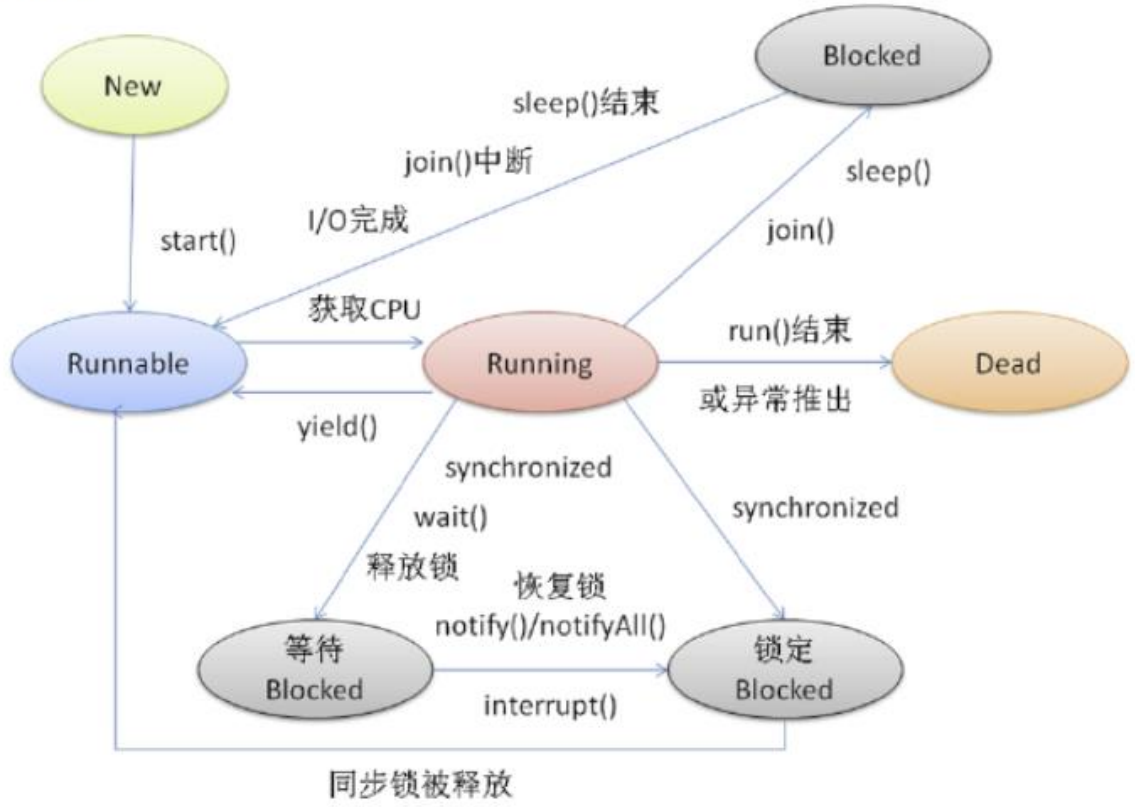
什么叫对象序列化(serializable)，什么是反序列化，实现对象序列化需要做哪些工作
对象序列化，将对象以二进制的形式保存在硬盘上
反序列化；将二进制的文件转化为对象读取
实现serializable接口

用什么把对象动态的写入磁盘中，写入要实现什么接口。
ObjectInputStream，需要实现Serializable接口

3、多线程

- 多线程：指的是这个程序（一个进程）运行时产生了不止一个线程
- 并行与并发：
  - 并行：多个cpu实例或者多台机器同时执行一段处理逻辑，是真正的同时。
  - 并发：通过cpu调度算法，让用户看上去同时执行，实际上从cpu操作层面不是真正的同时。并发往往在场景中有公用的资源，那么针对这个公用的资源往往产生瓶颈，我们会用TPS或者QPS来反应这个系统的处理能力。

线程状态



各种状态一目了然，值得一提的是"blocked"这个状态：
线程在Running的过程中可能会遇到阻塞(Blocked)情况

1. 调用join()和sleep()方法，sleep()时间结束或被打断，join()中断,IO完成都会回到Runnable状态，等待JVM的调度。
2. 调用wait()，使该线程处于等待池(wait blocked pool),直到notify()/notifyAll()，线程被唤醒被放到锁定池(lock blocked pool)，释放同步锁使线程回到可运行状态（Runnable）
3. 对Running状态的线程加同步锁(Synchronized)使其进入(lock blocked pool),同步锁被释放进入可运行状态(Runnable)。

此外，在runnable状态的线程是处于被调度的线程，此时的调度顺序是不一定的。Thread类中的yield方法可以让一个running状态的线程转入runnable。

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，它被包含在进程中，是进程的实际运作单位。程序员可以通过它进行多处理器编程，可以对运算进行提速。

2、线程和进程的区别

线程是进程的子集，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。每个线程都拥有单独的栈内存用来存储本地数据

3、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。

4、Java中堆和栈有什么不同?

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。

5、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。

6、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，在未使用完之前，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。

7、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。