[什么是线程安全](https://www.cnblogs.com/lixinjie/p/a-answer-about-thread-safety-in-a-interview.html)

【面试】如果你这样回答“什么是线程安全”，面试官都会对你刮目相看

有读者跟我说，喜欢看我的文章，说很容易读，我确实在易读性上花费的心思不亚于在内容上。因为我不喜欢一上来就堆很多东西，而且把简单的东西搞得复杂人人都会，但是把复杂的东西讲的简单，确实需要非常多的思考。

不是线程的安全

面试官问：“什么是线程安全”，如果你不能很好的回答，那就请往下看吧。

论语中有句话叫“学而优则仕”，相信很多人都觉得是“学习好了可以做官”。然而，这样理解却是错的。切记望文生义。

同理，“线程安全”也不是指线程的安全，而是指内存的安全。为什么如此说呢？这和操作系统有关。

目前主流操作系统都是多任务的，即多个进程同时运行。为了保证安全，每个进程只能访问分配给自己的内存空间，而不能访问别的进程的，这是由操作系统保障的。

在每个进程的内存空间中都会有一块特殊的公共区域，通常称为堆（内存）。进程内的所有线程都可以访问到该区域，这就是造成问题的潜在原因。

假设某个线程把数据处理到一半，觉得很累，就去休息了一会，回来准备接着处理，却发现数据已经被修改了，不是自己离开时的样子了。可能被其它线程修改了。

比如把你住的小区看作一个进程，小区里的道路/绿化等就属于公共区域。你拿1万块钱往地上一扔，就回家睡觉去了。睡醒后你打算去把它捡回来，发现钱已经不见了。可能被别人拿走了。

因为公共区域人来人往，你放的东西在没有看管措施时，一定是不安全的。内存中的情况亦然如此。

所以线程安全指的是，在堆内存中的数据由于可以被任何线程访问到，在没有限制的情况下存在被意外修改的风险。

即堆内存空间在没有保护机制的情况下，对多线程来说是不安全的地方，因为你放进去的数据，可能被别的线程“破坏”。

那我们该怎么办呢？解决问题的过程其实就是一个取舍的过程，不同的解决方案有不同的侧重点。

\*\*[同步/异步/阻塞/非阻塞/BIO/NIO/AIO](https://www.cnblogs.com/lixinjie/p/a-post-about-io-clearly.html)

网上有很多讲同步/异步/阻塞/非阻塞/BIO/NIO/AIO的文章，但是都没有达到我的心里预期，于是自己写一篇出来。

常规的误区

假设有一个展示用户详情的需求，分两步，先调用一个HTTP接口拿到详情数据，然后使用适合的视图展示详情数据。

如果网速很慢，代码发起一个HTTP请求后，就卡住不动了，直到十几秒后才拿到HTTP响应，然后继续往下执行。

这个时候你问别人，刚刚代码发起的这个请求是不是一个同步请求，对方一定回答是。这是对的，它确实是。

但你要问它为什么是呢？对方一定是这样回答的，“因为发起请求后，代码就卡住不动了，直到拿到响应后才可以继续往下执行”。

我相信很多人也都是这样认为的，其实这是不对的，是把因果关系搞反了：

不是因为代码卡住不动了才叫同步请求，而是因为它是同步请求所以代码才卡住不动了。

至于为什么能卡住不动，这是由操作系统和CPU决定的：

因为内核空间里的对应函数会卡住不动，造成用户空间发起的系统调用卡住不动，继而使程序里的用户代码卡住不动了。

因此卡住不动了只是同步请求的一个副作用，并不能用它来定义同步请求，那该如何定义呢？

两两组合

所谓同步/异步，关注的是能不能同时开工。

所谓阻塞/非阻塞，关注的是能不能动。

通过推理进行组合：

同步阻塞，不能同时开工，也不能动。只有一条小道，一次只能过一辆车，可悲的是还TMD的堵上了。

同步非阻塞，不能同时开工，但可以动。只有一条小道，一次只能过一辆车，幸运的是可以正常通行。

异步阻塞，可以同时开工，但不可以动。有多条路，每条路都可以跑车，可气的是全都TMD的堵上了。

异步非阻塞，可以工时开工，也可以动。有多条路，每条路都可以跑车，很爽的是全都可以正常通行。

是不是很容易理解啊。其实它们的关注点是不同的，只要搞明白了这点，组合起来也不是事儿。

回到程序里，把它们和线程关联起来：

同步阻塞，相当于一个线程在等待。

同步非阻塞，相当于一个线程在正常运行。

异步阻塞，相当于多个线程都在等待。

异步非阻塞，相当于多个线程都在正常运行。

redis也是多路复用，但它只有一个线程在执行select操作，处理就绪的连接，整个是串行化的，所以天然不存在并发问题。只能把它归为同步阻塞了。

BIO是阻塞IO，可以是同步阻塞，也可以是异步阻塞。AIO是异步IO，只有异步非阻塞这一种。因此没有同步非阻塞这种说法，因为同步一定是阻塞的。

## [适用场景：](https://blog.csdn.net/shipfei_csdn/article/details/104499525/)

BIO，适用于连接较少，对服务器资源消耗很大，但是编程简单。是同步阻塞的。

举例：你到餐馆点餐，然后在那儿等着，什么也做不了，只要饭还没有好，就要必须等着

NIO，使用于连接数量比较多且连接时间比较短的架构，比如聊天服务器，编程比较复杂。是同步非阻塞的

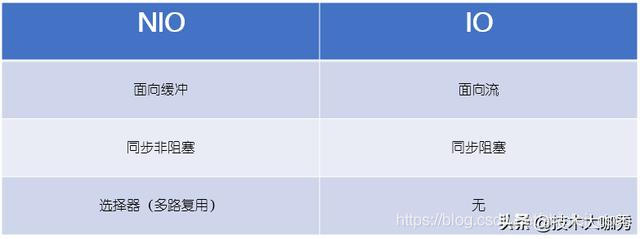
举例：你到餐馆点完餐，然后就可以去玩儿了，玩一会儿就回饭馆问一声，饭好了没。

AIO，适用于连接数量多而且连接时间长的架构，比如相册服务器，充分调用OS参与并发操作，编程比较复杂。是异步非阻塞的。

举例：饭馆打电话给你说，我们知道你的位置，待会儿给您送来，你安心的玩儿就可以了。类似于外卖。

二、NIO 和 IO 的区别

NIO的三个主要特点：面向缓冲、同步非阻塞和多路复用。



1、面向缓冲

传统IO是面向流的，NIO是面向缓冲的。传统IO是每次从流中读一个或多个字节，直到读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。NIO是将数据读取到一个她稍后处理的缓冲区，需要时可在缓冲区中前后移动，增加了处理过程中的灵活性。

2、同步非阻塞

传统IO的流是阻塞的，当一个线程调用read() 或 write()时，该线程被阻塞，直到数据完成读取或写入完成，此期间该线程不能再干任何事情了。NIO的非阻塞模式，使一个线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取。而不是保持线程阻塞，所以直至数据变得可以读取之前，该线程可以继续做其他的事情。 非阻塞写也是如此。

3、选择器（多路复用）

多路复用是指使用单线程也可以通过轮询监控的方式实现多线程类似的效果。简单的说就是，通过选择机制，使用一个单独的线程很容易来管理多个通道。