

操作系统面试题

1、**进程和线程的区别**？

解析：（1）进程是资源的分配和调度的一个独立单元，而线程是CPU调度的基本单元

（2）同一个进程中可以包括多个线程，并且线程共享整个进程的资源（寄存器、堆栈、上下文），一个进行至少包括一个线程。

（3）进程的创建调用fork或者vfork，而线程的创建调用pthread\_create，进程结束后它拥有的所有线程都将销毁，而线程的结束不会影响同个进程中的其他线程的结束

（4）线程是轻两级的进程，它的创建和销毁所需要的时间比进程小很多，所有操作系统中的执行功能都是创建线程去完成的

（5）线程中执行时一般都要进行同步和互斥，因为他们共享同一进程的所有资源

（6）线程有自己的私有属性TCB，线程id，寄存器、硬件上下文，而进程也有自己的私有属性进程控制块PCB，这些私有属性是不被共享的，用来标示一个进程或一个线程的标志

2、**死锁？死锁产生的原因？死锁的必要条件？怎么处理死锁？**

解析：·相互等待资源而产生的一种僵持状态，如果没有外力的干预将一直持续这个状态

·系统资源不足、相互竞争资源、请求资源顺序不当

·互斥、不可抢占、循环等待、请求与保持

·因为互斥是不可改变的，所以只能破坏其他三个条件中的一个来解除死锁，方法：剥夺资源、杀死其中一个线程

**3、Windows内存管理方式：段存储、页存储、段页存储**

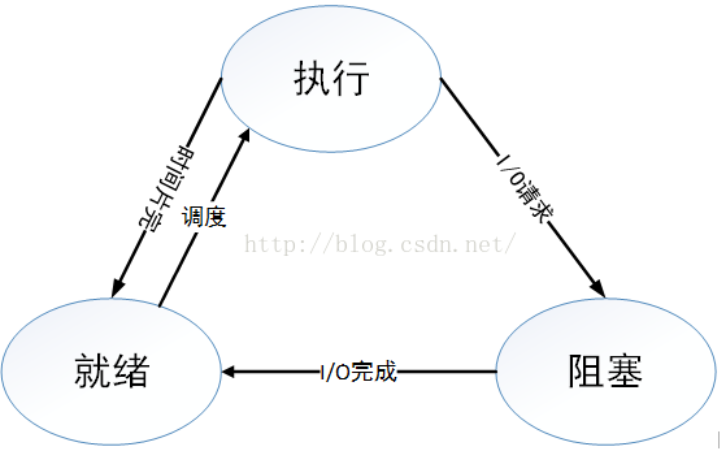
**4、进程的几种状态？**

解析：进程状态：一般来说，进程有三个状态，即就绪状态，运行状态，阻塞状态。

·运行态：进程占用CPU，并在CPU上运行；

·就绪态：进程已经具备运行条件，但是CPU还没有分配过来；

·阻塞态：进程因等待某件事发生而暂时不能运行；



（1）run（运行状态）：正在运行的进程或在等待队列中对待的进程，等待的进程只要以得到cpu就可以运行

（2）Sleep（可中断休眠状态）：相当于阻塞或在等待的状态

（3）D（不可中断休眠状态）：在磁盘上的进程

（4）T（停止状态）：这中状态无法直观的看见，因为是进程停止后就释放了资源，所以不会留在linux中

（5）Z（僵尸状态）：子进程先与父进程结束，但父进程没有调用wait或waitpid来回收子进程的资源，所以子进程就成了僵尸进程，如果父进程结束后任然没有回收子进程的资源，那么1号进程将回收

**5、IPC通信方式？**

解析：（1）管道（匿名管道（pipe亲缘关系的进程通信）、命名管道（mkfifo/mknod））

（2）消息队列：是基于消息的、用无亲缘关系的进程间通信，主要函数：msgget、msgsend、msgrecv、msgctl

（3）信号量：相当于一把互斥锁，通过p、v操作，主要函数：semget、semop、semctl

（4）共享内存：是进程间通信速度最快的，所以用经常是集合信号量或互斥锁来实现同步，shmget、shmat、shmdt、shmctl

**6、什么是虚拟内存？**

解析：是将进程部分装入内存中，从而能实现一个很大的程序能在一个比它小的内存中运行，它的主要实现是靠程序的换进换出来实现的，因为内存中0~3G是用户使用，3~4G才是内存使用，通过映射来实现来进行逻辑地址到物理地址的映射

**7、虚拟地址、逻辑地址、线性地址、物理地址的区别？**

解析： 分段机制把一个逻辑地址转换为线性地址；接着，分页机制把一个线性地址转换为物理地址。

（1）虚拟地址：虚拟内存映射出来的地址

（2）逻辑地址：程序的段加偏移量形成的，C/C++程序中取地址求出来的地址就是逻辑地址

（3）线性地址：是逻辑地址到物理地址的中间层，只有启动分页机制的时候才有线性地址，如果没有分页机制，那么线性地址就是物理地址

（4）物理地址：是内存中实实在在存在的硬件地址，

逻辑地址（启动分段）--》线性地址（启动分页）--》物理地址

**如何选择数据结构**

**--------------------------------------------------------------------------------------**

Array (T[])

•当元素的**数量是固定**的，并且需要使用**下标**时。

Linked list (LinkedList<T>)

•当元素需要**能够在列表的两端添加**时。否则使用 List<T>。

Resizable array list (List<T>)

•当元素的**数量不是固定**的，并且需要使用**下标**时。

Stack (Stack<T>)

•当需要实现 LIFO（Last In First Out）**后进先出**时。

Queue (Queue<T>)

•当需要实现 FIFO（First In First Out）**先进先出**时。

Hash table (Dictionary<K,T>)

•当需要使用**键值对**（Key-Value）**来快速添加和查找**，并且元素**没有特定的顺序**时。

Tree-based dictionary (SortedDictionary<K,T>)

•当需要使用价值对（Key-Value）来快速添加和查找，并且元素根据 **Key 来排序**时。

Hash table based set (HashSet<T>)

•当需要保存一组唯一的值，并且元素没有特定顺序时。

Tree based set (SortedSet<T>)

•当需要**保存一组唯一**的值，并且元素需**要排序**时。

ArrayList、Vector、LinkedList的区别及其优缺点？

**--------------------------------------------------------------------------------------**

ArrayList,LinkedList,Vestor这三个类都实现了java.util.List接口，但它们有各自不同的特性，主要如下：

一、同步性

ArrayList,LinkedList是不同步的，**而Vector是同步的**。所以如果不**要求线程安全**的话，可以使用ArrayList或 LinkedList，可以节省为同步而耗费的开销。但在**多线程的情况下**，有时候就不得不使用Vector了。

二、数据增长

从内部实现机制来讲**ArrayList和Vector都是**使用Objec的**数组形式来存储**的。当你向这两种类型中增加元素的时候，如果元素的数目超出了内部数组目前的长度它们都需要扩展内部数组的长度，**Vector缺省情况下自动增长原来一倍的数组长度**，**ArrayList是原来的50%**,所以最后你获得的这个集合所占的空间总是比你实际需要的要大。所以**如果你要在集合中保存大量的数据那么使用Vector有一些优势**，因为你可以通过设置集合的初始化大小来避免不必要的资源开销。

三、检索、插入、删除对象的效率

**ArrayList和Vector检索数据快，插入和删除慢。LinkedList插入和删除数据快，检索慢。**

ArrayList和Vector中，从指定的位置（用index）**检索**一个对象，或在**集合的末尾插入、删除**一个对象的时间是一样的，可表示为**O(1)**。 但是，如果在集合的其他位置**增加或移除元素**那么花费的时间会呈线形增长：**O(n-i)**，其中n代表集合中元素的个数，i代表元素增加或移除元素的索引位置。以为在进行上述操作的时候集合中第i和第i个元素之后的所有元素都要执行(n-i)个对象的**位移操作**。

LinkedList中，在**插入、删除**集合中任何位置的元素所花费的时间都是一样的—**O(1)**，但它在**索引**一个元素的时候**比较慢，为O(i)**,其中i是索引的位置。

ArrayList和LinkedList的大致区别：

1**.ArrayList**是实现了基于**动态数组**的数据结构，**LinkedList**基于**链表**的数据结构。

2.对于**随机访问**get和set，**ArrayList优于LinkedList**，因为LinkedList要移动指针。

3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。

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

ArrayList 和Vector他们底层的实现都是一样的，都是使用**数组方式存储数据**，此数组元素数大于实际存储的数据以便增加和插入元素，它们都**允许直接按序号索引元素**，但是插入元素要涉及数组元素移动等内存操作，所以**索引数据快而插入数据慢**。

Vector中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但**性能上较ArrayList差**，因此已经是Java中的遗留容器。

LinkedList使用**双向链表实现存储**（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，**内存的利用率更高**），按序号**索引数据需要进行前向或后向遍历**，但是**插入数据时只需要记录本项的前后项即可**，所以**插入速度较快**。

网络七层协议

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

OSI，OSI是一个开放性的通信系统互连参考模型，他是一个定义得非常好的协议规范。OSI模型有7层结构，每层都可以有几个子层。

·应用层

示例：TELNET，HTTP，FTP，NFS，SMTP等。

·表示层

示例：加密，ASCII等。

·会话层

示例：RPC，SQL等。

·传输层

示例：TCP，UDP，SPX。

·网络层

示例：IP，IPX等。

·数据链路层

示例：ATM，FDDI等。

·物理层

示例：Rj45，802.3等。

-- http、TCP、UDP的区别和联系

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

TCP/IP

建立起一个TCP连接需要经过“三次握手”：

·第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；

·第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

·第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写 了，就是服务器和客户端交互，最终确定断开）

UDP与TCP的区别

TCP UDP

是否连接 面向连接 面向非连接

传输可靠性 可靠 不可靠

应用场合 传输大量数据 少量数据

速度 慢 快

1.http是要基于TCP连接基础上的，简单的说，TCP就是单纯建立连接，不涉及任何我们需要请求的实际数据，简单的传输。http是用来收发数据，即实际应用上来的。

2. TCP是底层通讯协议，定义的是数据传输和连接方式的规范

HTTP是应用层协议，定义的是传输数据的内容的规范

HTTP协议中的数据是利用TCP协议传输的，所以支持HTTP也就一定支持TCP

3. HTTP支持的是www服务

而TCP/IP是协议

它是Internet国际互联网络的基础。TCP/IP是网络中使用的基本的通信协议。

TCP/IP实际上是一组协议，它包括上百个各种功能的协议，如：远程登录、文件传输和电子邮件等，而TCP协议和IP协议是保证数据完整传输的两个基本的重要协议。通常说TCP/IP是Internet协议族，而不单单是TCP和IP。

-- TCP和UDP各自的优势，知道哪些使用UDP协议的成功案例

-- TCP和UDP各用了底层什么协议

-- 单个UDP报文最大容量

-- TCP粘包怎么处理

如何解决拆包粘包

既然知道了tcp是无界的数据流，且协议本身无法避免粘包，拆包的发生，那我们只能在应用层数据协议上，加以控制。通常在制定传输数据时，可以使用如下方法：

1.使用带消息头的协议、消息头存储消息开始标识及消息长度信息，服务端获取消息头的时候解析出消息长度，然后向后读取该长度的内容。

2.设置定长消息，服务端每次读取既定长度的内容作为一条完整消息。

3.设置消息边界，服务端从网络流中按消息编辑分离出消息内容。

-- DDOS攻击了解吗

-- 用过那些数据结构

-- 进程和线程的区别和练习

-- Innodb和Myisam的区别

-- C 的内存分配

-- STL标准库中的数据结构占用的哪一部分的内存

即标准模板库，是一个具有工业强度的，高效的C++程序库。

-- map的实现用的什么数据结构，插入的时间复杂度是多少

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

如果插入单个元素且无暗示，时间复杂度为 O(logn)，其中 n 为容器的大小。

如果插入多个元素，时间复杂度为 O(nlogn)，“第一个” n 为插入元素数，“第二个” n 为插入元素数加容器大小。如果插入的范围中的元素已经按同样的排序规则排序，执行过程将被优化，时间复杂度甚至会降到 O(n)。

问项目在安全方面有什么考虑吗（并没有，被面试官鄙视了）

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

**·密码保存**

保存密码的第一准则是不能明文保存密码，之前CSDN密码泄露一事还记忆犹新。通常的做法是对密码进行不可逆加密，加密时不要使用MD5或者SHA系列的算法加密，现在对这两个算法的破解研究工作已经有了相当的进展。推荐使用bcrypt。

**·CI服务器的安全性**

CI服务器和Build服务器经常是攻击者最完美的跳板，这些服务器几乎拥有你系统的所有权限（代码库，各个生产环境的访问权限等等）。请确保其安全。

**·校验客户端安全证书的“指纹”**

当使用SSL时，我们需要验证安全证书的“指纹”来确认该证书是有目标网站颁发的证书，这可以有效防止“中间人攻击”漏洞。

**·随机生成密码**

不要对所有账户设置同一个密码，一锅端就不好了，使用密码管理软件为每一个账户生成一个独立的密码。当前比较推荐的是1Password，期待有更好的密码管理软件诞生。

**·使用安全系数更高的随机数生成类库**

要想得到真正的随机数是很困难的，尤其是要满足安全领域的随机数标准，语言自带的一般随机数生成算法很难满足需要，所以JDK还提供了SecuredRandom这样的类，Ruby中也有SecureRandom这样的模块。

**·总是使用HTTPS和HSTS**

如无值得信服的理由，所有的请求都应该使用加密协议

**·不要使用“安全问题”找回密码**

一个桶能装多少水是有最短的那块木板决定的，一个安全系统的安全性也是有最薄弱的环节决定的。不要让安全问题成为你的那块短木板。

**·不要限制密码长度**

通过密码长度来保证安全性的时代已经结束了，让用户使用"Pass Phrases"是一个不错的主意。

**·允许对密码或者密码短语使用更直观的复杂度控制**

使用更有效，直观的方法帮助用户增加密码或者密码短语的难度。而不是简单的必须有大小写，必须超过8位等这些生硬的条件。

**·避免缓冲溢出**

在弱类型语言中，最常见的安全性问题就是缓冲溢出，这也是最常见的一种远程攻击方法，一般有四种防范方法： \* 通过操作系统使得缓冲区不可执行，从而阻止攻击者植入攻击代码。 \* 写正确的代码 \* 利用编译器的边界检查来实现缓冲区的保护。 \* 在程序指针失效前进行完整性检查

**·避免注入攻击**

所有的用户输入必须经过验证和清理才能传递给下游，最常见的就是SQL注入攻击

**·避免跨站脚本攻击**

系统需要保证用户不能在页面中嵌入未经验证的信息

**·避免泄露信息给第三方**

目前很多系统倾向于泄露信息给第三方，比方说Google，Facebook，新浪微博等等。这些第三方会窃取很多用户信息。

数据库的行级锁和表级锁悲观锁和乐观锁思想

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

我们经常在开发中遇到数据库并发处理时，处理不一致的问题。需要对程序做并发控制。

典型的并发时出现的冲突有两种：

1、丢失更新：一个事务的更新覆盖了其它事务的更新结果，就是所谓的更新丢失。

例如X=0 A1把X=X+1 A2也设置X=X+1 如果A1和A2同时执行，可能出现最终X=1的情况。而我们需要得到的是X=2 。 这就需要update处理为串行化的。

2、脏读：读到的数据可能是旧的。 初使X=0 A1把X设置为1，A2读取时X是1，但A1后面又修改成了2，或回滚到了0，那么A2读取的数据就是脏数据。

A2读取的是未提交的数据。一般数据库不会设为读未提交，所以一般不会出现脏读。

**悲观锁：**需要使用数据库的锁机制，如数据库有表级排它锁，有行级排它锁。

**假定一切操作都可能发现并发冲突**，所以采取悲观态度。通过加锁，**屏蔽一切可能违反数据完整性的操作**

**乐观锁：**其实不是真实的去锁住记录不让访问，或者不让更新。

**假定操作很少发生冲突**，一般对于**读多写少**的情况。**只在提交操作时检查是否违反数据完整性**。

·乐观锁不能解决脏读的问题。

·可以通过版本号是否比上个版本号或者时间戳来实现。

·对于冲突检测后的处理，需要业务逻辑去处理。

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

**表级，**直接锁定整张表，在你锁定期间，其它进程无法对该表进行写操作。如果你是写锁，则其它进程则读也不允许

**行级,**仅对指定的记录进行加锁，这样其它进程还是可以对同一个表中的其它记录进行操作。

**页级，**表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，**一次锁定相邻的一组记录。**

上述三种锁的特性可大致归纳如下：

1） 表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

2） 行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。

3） 页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

InnoDB有两种模式的**行锁**：

1）共享锁：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。

( Select \* from table\_name where ......lock in share mode)

2）排他锁：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和 排他写锁。(select \* from table\_name where.....for update)

为了允许行锁和表锁共存，实现多粒度锁机制；同时还有两种内部使用的意向锁（都是表锁），分别为意向共享锁和意向排他锁。

JVM内存分区，内存泄露与内存溢出的区别

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

**内存泄漏**（memory leak）： 是指**程序在申请内存后，无法释放已申请的内存空间**，一次内存泄露危害可以忽略，但内存泄露堆积后果很严重，无论多少内存,迟早会被占光。

一般我们所说的内存泄漏是指**堆内存的泄漏**，堆内存是指程序从堆中分配的，大小任意的（内存块的大小可以在程序运行期决定），使用完成之后必须显示释放内存。应用程序一般使用malloc、realoc、new等函数从堆中分配到一块内存块，使用完成后，程序必须负责相应的释放。在C中使用free(),C++中delete、delete[]、free()。而Java中由于垃圾回收机制不用手动释放。

如果内存不能释放，这块内存就不能再次使用，我们就说这块内存泄漏了。

**内存溢出**（out of memory）：**程序要求的内存，超出了系统所能分配的范围**。如：我们用一个int型4字节的数据来装一个float型8字节的数据，就会产生内存溢出。不过我们在编程是可以强制类型转换int XX = (int)float;只取float 4字节数据给int.

遇到过项目频繁出现FULL GC的情况吗？怎么排查（项目规模小，并没有遇到过）

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

**GC机制**

要准确理解Java的垃圾回收机制，就要从：“什么时候”，“对什么东西”，“做了什么”三个方面来具体分析。

第一：“什么时候”即就是**GC触发的条件**。GC触发的条件有两种。（1）程序调用System.gc时可以触发；（2）系统自身来决定GC触发的时机。

系统判断GC触发的依据：根据Eden区和From Space区的内存大小来决定。当内存大小不足时，则会启动GC线程 并停止应用线程。

第二：“对什么东西”笼统的认为是**Java对象**并没有错。但是准确来讲，GC操作的对象分为：**通过可达性分析法无法搜索到的对象**和**可以搜索到的对象**。对于搜索不到的方法进行标记。

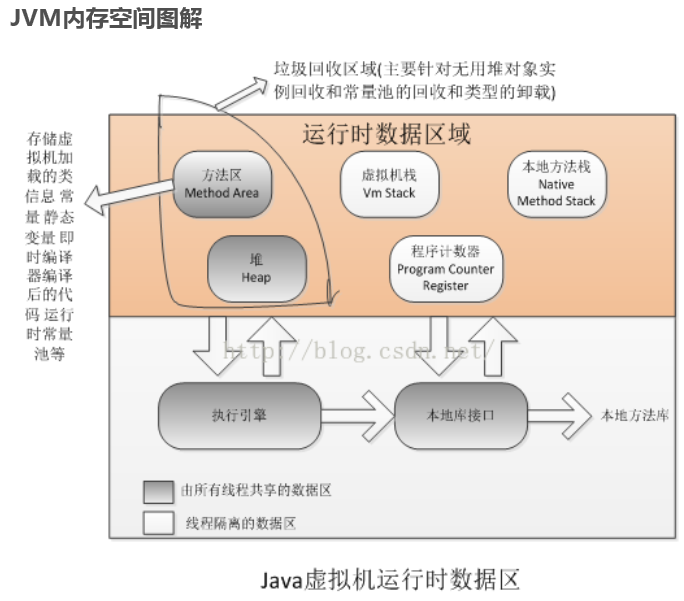
第三：“做了什么”最浅显的理解为**释放对象**。但是从GC的底层机制可以看出，对于可以搜索到的对象进行**复制操作**，对于搜索不到的对象，调用**finalize()方法进行释放**。

**具体过程：**当GC线程启动时，会通过可达性分析法把Eden区和From Space区的存活对象复制到To Space区，然后把Eden Space和From Space区的对象释放掉。当GC轮训扫描To Space区一定次数后，把依然存活的对象复制到老年代，然后释放To Space区的对象。

对于用可达性分析法搜索不到的对象，GC并不一定会回收该对象。要完全回收一个对象，至少需要经过两次标记的过程。

GC过程中用到的回收算法：

通过上面的GC过程不难看出，Java堆中的年轻代和老年代采用了不同的回收算法。**年轻代采用了复制法**；而**老年代采用了标记-整理法**



**程序计数器：**线程私有。是一块较小的内存，是当前线程所执行的字节码的行号指示器。是Java虚拟机规范中唯一没有规定OOM（OutOfMemoryError）的区域。

**Java栈：**线程私有。生命周期和线程相同。是Java方法执行的内存模型。执行每个方法都会创建一个栈帧，用于存储局部变量和操作数（对象引用）。局部变量所需要的内存空间大小在编译期间完成分配。所以栈帧的大小不会改变。存在两种异常情况：若线程请求深度大于栈的深度，抛StackOverflowError。若栈在动态扩展时无法请求足够内存，抛OOM。

**Java堆**：所有线程共享。虚拟机启动时创建。存放对象实力和数组。所占内存最大。分为新生代（Young区），老年代（Old区）。新生代分Eden区，Servior区。Servior区又分为From space区和To Space区。Eden区和Servior区的内存比为8:1。 当扩展内存大于可用内存，抛OOM。

**方法区：**所有线程共享。用于存储已被虚拟机加载的类信息、常量、静态变量等数据。又称为非堆（Non – Heap）。方法区又称“永久代”。GC很少在这个区域进行，但不代表不会回收。这个区域回收目标主要是针对常量池的回收和对类型的卸载。当内存申请大于实际可用内存，抛OOM。

**本地方法栈**：线程私有。与Java栈类似，但是不是为Java方法（字节码）服务，而是为本地非Java方法服务。也会抛StackOverflowError和OOM。

**Minor GC ，Full GC 触发条件**

Minor GC触发条件：当Eden区满时，触发Minor GC。

**Full GC触发条件：**

（1）调用System.gc时，系统建议执行Full GC，但是不必然执行

（2）老年代空间不足

（3）方法去空间不足

（4）通过Minor GC后进入老年代的平均大小大于老年代的可用内存

（5）由Eden区、From Space区向To Space区复制时，对象大小大于To Space可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小