1. 实践中如何优化MySQL

2. 什么情况下设置了索引但无法使用

3. SQL语句的优化

4. 索引的底层实现原理和优化

5. 锁的优化策略

6.区别（epoll相对select优点）主要有三：

7.构造函数不能声明为虚函数的原因是:

8.TCP UDP端口扫描的实现方式

9、进程（线程）通信有哪些手段？

10、文件描述符知道吧？同一个进程打开同一个文件两次，会发生什么？具体说说文件的底层数据结构

1. 实践中如何优化MySQL

第一条影响最大，后面越来越小。

① SQL语句及索引的优化

② 数据库表结构的优化

③ 系统配置的优化

④ 硬件的优化

2. 什么情况下设置了索引但无法使用

① 以“%”开头的LIKE语句，模糊匹配

② OR语句前后没有同时使用索引

③ 数据类型出现隐式转化（如varchar不加单引号的话可能会自动转换为int型）

对于多列索引，不是使用的第一部分，则不会使用索引

如果mysql估计使用全表扫描要比使用索引快,则不使用索引

3. SQL语句的优化

order by要怎么处理

alter尽量将多次合并为一次

insert和delete也需要合并

4. 索引的底层实现原理和优化

B+树，经过优化的B+树

主要是在所有的叶子结点中增加了指向下一个叶子节点的指针，因此InnoDB建议为大部分表使用默认自增的主键作为主索引。

5. 锁的优化策略

① 读写分离

② 分段加锁

③ 减少锁持有的时间

④ 多个线程尽量以相同的顺序去获取资源

等等，这些都不是绝对原则，都要根据情况，比如不能将锁的粒度过于细化，不然可能会出现线程的加锁和释放次数过多，反而效率不如一次加一把大锁。

6.区别（epoll相对select优点）主要有三：

①.select的句柄数目受限，在linux/posix\_types.h头文件有这样的声明：#define \_\_FD\_SETSIZE 1024 表示select最多同时监听1024个fd。而epoll没有，它的限制是最大的打开文件句柄数目。

②.epoll的最大好处是不会随着FD的数目增长而降低效率，在selec中采用轮询处理，其中的数据结构类似一个数组的数据结构，而epoll是维护一个队列，直接看队列是不是空就可以了。epoll只会对"活跃"的socket进行操作---这是因为在内核实现中epoll是根据每个fd上面的callback函数实现的。那么，只有"活跃"的socket才会主动的去调用 callback函数（把这个句柄加入队列），其他idle状态句柄则不会，在这点上，epoll实现了一个"伪"AIO。但是如果绝大部分的I/O都是“活跃的”，每个I/O端口使用率很高的话，epoll效率不一定比select高（可能是要维护队列复杂）。

③.使用mmap加速内核与用户空间的消息传递。无论是select,poll还是epoll都需要内核把FD消息通知给用户空间，如何避免不必要的内存拷贝就很重要，在这点上，epoll是通过内核于用户空间mmap同一块内存实现的

7. **构造函数不能声明为虚函数的原因是:** 构造一个对象的时候，必须知道对象的实际类型，而虚函数行为是在运行期间确定实际类型的。而在构造一个对象时，由于对象还未构造成功。编译器无法知道对象 的实际类型，是该类本身，还是该类的一个派生类，或是更深层次的派生类。无法确定。。。

虚函数的执行依赖于虚函数表。而虚函数表在构造函数中进行初始化工作，即初始化vptr，让他指向正确的虚函数表。而在构造对象期间，虚函数表还没有被初 始化，将无法进行。

8.TCP UDP端口扫描的实现方式

[**linux下端口扫描的实现(TCP connect、TCP SYN、TCP FIN、UDP四种方式)**](http://blog.csdn.net/tuantuanlin/article/details/17422705)

.connect扫描

优点： 编程简单，是需要一个API connect(),比较可靠，因为TCP是可靠协议，当丢包的时候，会重传SYN帧。

缺点：正因为TCP的可靠性，所以当端口不存在的时候，源主机会不断尝试发SYN帧企图得到ack的应答，多次尝试后才会放弃，因此造成了扫描的时间较长。并且，connect的扫描方式可能较容易被目标主机发现。

.SYN扫描

SYN扫描的原理就是想目标端口发送SUN数据帧，如果源主机收到SYN+ACK数据包，寿命此端口开放，如果收到RST说明此端口关闭。由于SYN扫描并不会完成TCP三次握手过程，所以又叫半开放扫描。

优点： 速度快；如果不被防火墙过滤的话，基本都能收到应答包。

缺点： 扫描行为容易被发现；因为是自己攒包发，是在ip层的，因此不可靠，可能会丢包；实现起来比connect稍复杂。

.FIN扫描

根据上述四次挥手过程，主动结束的一方会发送FIN帧。当我们发送FIN帧给一个非监听的端口时，会有RST应答，反之，发给一个正在监听的端口时，不会有任何回应。

优点： 隐蔽性好；速度快。

缺点： 只能用于linux系统，windows系统下无效，在windows下，无论端口是否监听，都将回应RST帧，造成无法判断；不可靠，当收不到应答包时，不确定是端口在监听，还是丢包了。

、UDP

 常见的方式有UDP recvfrom扫描，UDP ICMP端口不可达扫描，后者给一个端口发送UDP报文，如果端口是开放的，则没有响应，如果端口是关闭的，对方会回复一个ICMP 端口不可达报文（对应ICMP首部前两个字段：类型3 代码3，ICMP详见[ping那篇文章](http://blog.csdn.net/tuantuanlin/article/details/17006867" \t "_blank)），

优点：linux windows都能用

缺点：也是不可靠的，因为返回的是错误信息，所以速度相对于TCP的FIN,SYN扫描要慢一些，如果发送的UDP包太快了，回应的ICMP包会出现大量丢失的现象。

9、TCP close\_wait状态

[**TCP/IP详解--连接状态变迁图CLOSE\_WAIT**](http://blog.csdn.net/yusiguyuan/article/details/21445945)

CLOSED：表示初始状态。对服务端和C客户端双方都一样。  
        LISTEN：表示监听状态。服务端调用了listen函数，可以开始accept连接了。  
        SYN\_SENT：表示客户端已经发送了SYN报文。当客户端调用connect函数发起连接时，首先发SYN给服务端，然后自己进入SYN\_SENT状态，并等待服务端发送ACK+SYN。  
        SYN\_RCVD：表示服务端收到客户端发送SYN报文。服务端收到这个报文后，进入SYN\_RCVD状态，然后发送ACK+SYN给客户端。  
        ESTABLISHED：表示连接已经建立成功了。服务端发送完ACK+SYN后进入该状态，客户端收到ACK后也进入该状态。  
        FIN\_WAIT\_1：表示主动关闭连接。无论哪方调用close函数发送FIN报文都会进入这个这个状态。  
        FIN\_WAIT\_2：表示被动关闭方同意关闭连接。主动关闭连接方收到被动关闭方返回的ACK后，会进入该状态。  
        TIME\_WAIT：表示收到对方的FIN报文并发送了ACK报文，就等2MSL后即可回到CLOSED状态了。如果FIN\_WAIT\_1状态下，收到对 方同时带FIN标志和ACK标志的报文时，可以直接进入TIME\_WAIT状态，而无须经过FIN\_WAIT\_2状态。  
        CLOSING：表示双方同时关闭连接。如果双方几乎同时调用close函数，那么会出现双方同时发送FIN报文的情况，此时就会出现CLOSING状态，表示双方都在关闭连接。  
        CLOSE\_WAIT：表示被动关闭方等待关闭。当收到对方调用close函数发送的FIN报文时，回应对方ACK报文，此时进入CLOSE\_WAIT状态。  
        LAST\_ACK：表示被动关闭方发送FIN报文后，等待对方的ACK报文状态，当收到ACK后进入CLOSED状态。

7. TCP流量控制手段，拥塞控制，TCP怎么判断发生了网络拥塞

**.TCP的流量控制**

利用滑动窗口实现流量控制；必须考虑传输速率；

**.拥塞控制方法**

慢开始( slow-start )、拥塞避免( congestion avoidance )、快重传( fast retransmit )和快恢复( fast recovery )。

8.发送一个RST会发生什么？

http://my.oschina.net/costaxu/blog/127394

.端口未打开

.请求超时

.提前关闭

.在一个已关闭的socket上收到数据

9、进程（线程）通信有哪些手段？

**几种进程间的通信方式**

（1） 管道（pipe）：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有血缘关系的进程间使用。进程的血缘关系通常指父子进程关系。

（2）有名管道（named pipe）：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间通信。

（3）信号量（semophore）：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它通常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

（4）消息队列（message queue）：消息队列是由消息组成的链表，存放在内核中 并由消息队列标识符标识。消息队列克服了信号传递信息少，管道只能承载无格式字节流以及缓冲区大小受限等缺点。

（5）信号（signal）：信号是一种比较复杂的通信方式，用于通知接收进程某一事件已经发生。

（6）共享内存（shared memory）：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问，共享内存是最快的IPC方式，它是 针对其他进程间的通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量配合使用，来实现进程间的同步和通信。

（7）套接字（socket）：套接口也是一种进程间的通信机制，与其他通信机制不同的是它可以用于不同及其间的进程通信。

**几种线程间的通信机制**

、锁机制

     1.1 互斥锁：提供了以排它方式阻止数据结构被并发修改的方法。

     1.2 读写锁：允许多个线程同时读共享数据，而对写操作互斥。

     1.3 条件变量：可以以原子的方式阻塞进程，直到某个特定条件为真为止。对条件测试是在互斥锁的保护下进行的。条件变量始终与互斥锁一起使用。

、信号量机制：包括无名线程信号量与有名线程信号量

、信号机制：类似于进程间的信号处理。

线程间通信的主要目的是用于线程同步，所以线程没有象进程通信中用于数据交换的通信机制。

.**区别**  
进程和线程的主要差别在于它们是不同的操作系统资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。  
1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.  
2) 线程的划分尺度小于进程，使得多线程程序的并发性高。  
3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。  
4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。但是线程不能够独立执行，必须依存在应用程序中，由应用程序提供多个线程执行控制。  
5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。这就是进程和线程的重要区别。

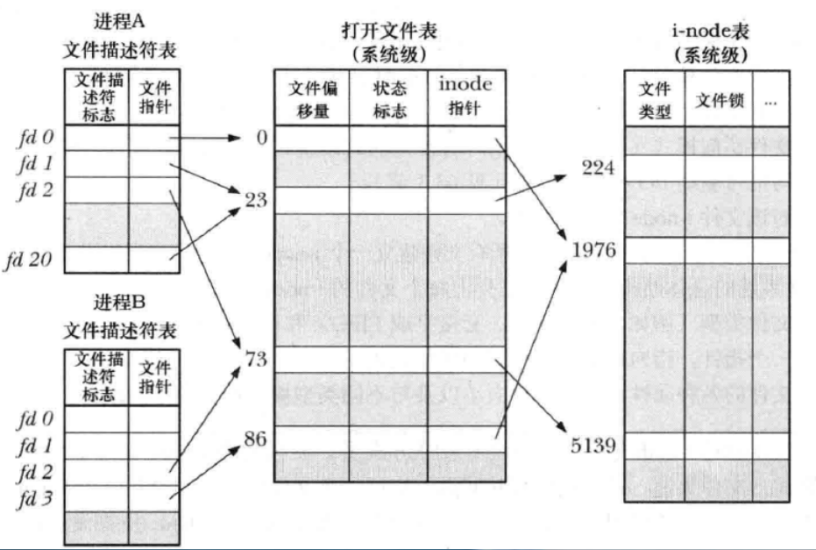
10、文件描述符知道吧？同一个进程打开同一个文件两次，会发生什么？具体说说文件的底层数据结构

所有执行I/O操作的系统调用都以文件描述符，即一个非负整数来指代所打开的文件。文件描述符可以用来表示所有类型的已打开文件。同时，多个文件描述符可以指向同一个打开文件，因为有在不同进程中打开同一个文件的需求。

系统是如何维护硬盘文件与文件描述符之间的联系？

查看由内核维护的3个数据结构：

* 进程级的文件描述符
* 系统级的打开文件表
* 文件系统的i-node表



考查11种模式

单例模式、简单工厂模式、工厂模式、抽象工厂模式、策略模式、观察者模式、组合模式、适配器模式、装饰模式、代理模式、外观模式