1. 操作系统
    a) 进程与线程
    b) 处理机的调度与死锁
    c) 内核态与用户态
    d) 调度算法
    e) Linux 操作指令：查看进程、内存、磁盘空间
    f) 存储器管理、页面置换算法、物理地址与虚地址
    g) 文件系统
    h) I/O 控制方式、直接存储器访问
2. 计算机网络
    a) TCP 三次握手、四次挥手、滑动窗口、拥塞控制、TCP 头
    b) UDP 用 UDP 实现 Http
    c) IP
    d) HTTP、DNS
    e) 对称加密、非对称加密
    f) 网络数据包寻址方式
3. 数据库
    a) Mysql 的操作：创建、增删改查
    b) B 树
4. C++基础
    a) 引用与指针
    b) Const 与 Static 的区别
    c) 智能指针
    d) Vector、List、Map、Set、Deque
    e) Volatile
    f) 继承
    g) 红黑树和 B+树
5. Python 基础知识
6. 算法

操作系统

1. 线程与进程

定义：

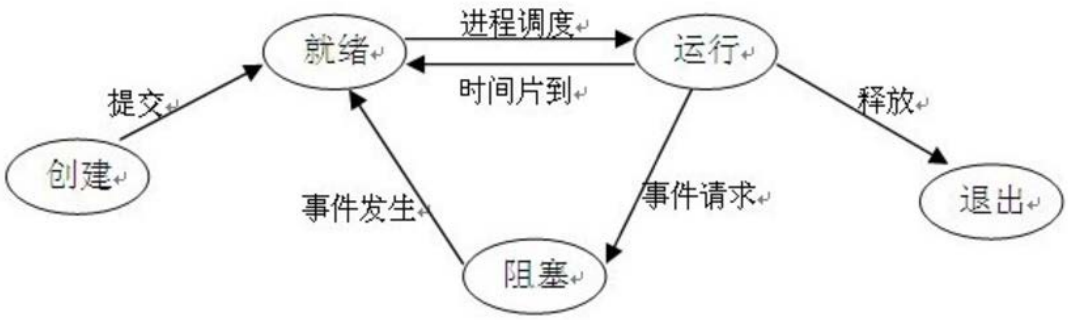
1） 进程是程序的一次执行

2） 进程是一个程序及其数据在处理机上顺序执行时所发生的活动

3） 进程是程序在一个数据集合上运行的过程，是系统进行资源分配与调度的一个独立单位。

进程的**状态：**

就绪、执行、阻塞、挂起、创建、终止



进程控制块（PCB）：

进程标识符、处理机状态、进程调度信息、进程控制信息

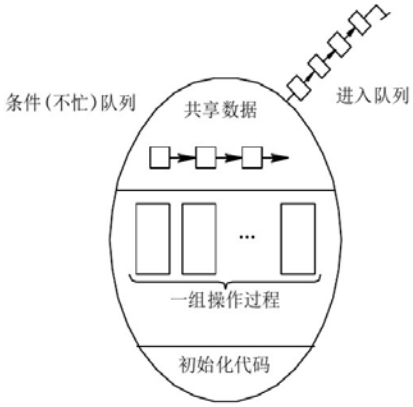
同步机制的规则：

空闲让进、忙则等待、有限等待、让权等待

进程同步：

信号量：整型信号量、记录型信号量、AND 型信号量、信号量集

管程机制：利用共享数据结构抽象地表示系统中的共享资源，而把对该共享数据结构实施的操作定义为一组操作，进程对共享资源的申请、释放和其他操作，都是通过这组过程对共享数据结构的操作来实现的。



经典的进程同步问题：

生产者-消费者，哲学家进餐问题、读写者问题

进程通信：

共享存储器系统（基于共享数据结构的、基于共享存储区的）、消息传递系统、管道通信（连接一个读进程与一个写进程的一个共享文件）

消息传递系统：

直接通信方式（Send(Receiver, message), Receive(Sender, message)）、间接通信方式(Send(mailbox, message), Receiver(mailbox, message))

消息缓冲区、发送原语、接收原语，先获取资源信号量，再获取互斥信号量

管道：

写进程向管道中写数据，读进程从管道中读进程，注意互斥、同步确定对方是否存在

线程：

线程是能独立运行的基本单位，是独立调度和分派的基本单位，线程切换非常迅速且开销小。

进程与线程的区别：

调度性：线程是调度和分派的基本单位，进程是资源拥有的基本单位，线程基本上不拥有资源，一个进程内的线程切换不会引起进程的切换，进程间的线程切换会引起进程切换。

并发性：多个进程可以并发，一个进程内的多个线程也可以并发，使系统有更好的并发性，有效地提高系统的利用率与系统的吞吐量。

拥有资源：进程是系统中拥有资源的一个基本单位，一般线程不拥有自己的资源（也有一点不可少的资源），但它可以访问隶属进程的资源。

系统开销：在创建或撤销进程时，系统都要为之创建和回首进程控制块，分配或回收资源，系统付出的开销大于线程的创建和撤销的开销。进程的切换涉及当前进程 CPU 环境的保存以及新被调度运行进程的 CPU 环境设置，而线程的切换只需保存和设置少量的寄存器内容，不涉及存储器管理方面的操作，进程的切换代价高于线程。同一个进程内的多个线程拥有相同的地址空间,在同步与通信方面线程也比进程容易。一些操作系统中,线程的切换、同步、通信都无须操作系统内核的干预。

线程间的同步与通信：

互斥锁（开锁、关锁）、条件变量、信号量机制

线程实现方式：

内核支持线程（所有进程的操作都是利用系统调用而进入内核）、用户级线程（无需内核的支持，采用）、组合方式

2. 处理机的调度与死锁

调度算法：

作业周转时间：指从作业被提交给系统开始，到作业完成为止的这段时间间隔。

平均周转时间：$T = \frac{1}{n}[\sum_{i=1}^{n} Ti]$

带权周转时间：$W = \frac{Ti}{Ts}$，　$T = \frac{1}{n}[\sum_{i=1}^{n} \frac{Ti}{Ts}]$, Ts 为系统为其提供服务的时间

先来先服务调度算法（FCFS）：有利于长作业，不利于短作业，有利于 CPU 繁忙型作业，不利于 I/O 繁忙型作业。

短作业（进程）优先调度算法（SJ(P)F）：有利于短作业，不利于长作业，可能导致长作业不被调用。不能改保证紧迫性作业会被及时处理。不一定能真正做到短作业优先。

最高优先权优先调度算法：非抢占式优先权算法、抢占式优先权算法，静态优先权、动态优先权

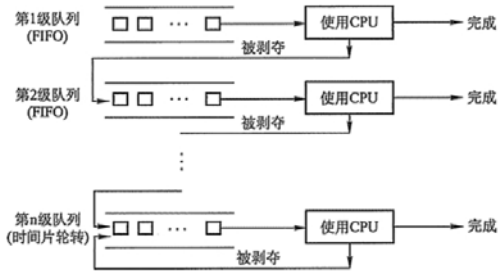
高响应比优先调度算法：动态优先权，优先权 $= \frac{等待时间+要求服务时间}{要求服务时间} = \frac{响应时间}{要求服务时间} = Rp$，

增加了系统开销

时间片轮转法：就绪队列，时间片的选择，太小会频繁发生中断、进程上下文的切换，增加系统开销，太长时间片算法退化为 FCFS 算法，无法满足交互式用户的需求。

多级反馈队列调度算法：

1） 设置多个就绪队列，并为各个队列赋予不同的优先级。第一个队列的有限级最高，第二队列次之，其余各队列优先级递减。各个队列的执行时间片，优先级越高，执行时间片越短。



2） 当一个新进程进入内存后，先加入到第一个队列的末尾，按照 FCFS 算法排队等待调度，当轮到该进程执行时，如果它能在该时间片内完成，便可准备撤离系统，如果未完成，调度程序将该进程转入第二个队列的末尾，再按 FCFS 原则等待调度执行，如此下去，当一个长作业（进程）从第一队列一次降到第 n 队列后，在第 n 队列中采用按时间片轮转的方式运行。

3） 仅当第一队列空闲时，调度程序才调度第二队列中的进程运行，仅当第 1~i-1 队列均为空闲时，才调度第 i 队列中的进程运行。如果处理机在执行第 i 队列的某进程时，又有新进程进入优先级较高的队列（1~i-1）中的任何一个队列，则此时新进程将抢占正在执行的进程的处理机，即由调度进程把正在运行的进程放回到第 i 队列的末尾，把处理机分配给新到的高优先权进程。

实时调度算法：

http://blog.csdn.net/qq_28602957/article/details/53445188

实时调度的 **CPU 处理能力：**

单处理机：m：周期性硬实时任务数，Ci：每次处理时间，Pi：周期时间

$$\sum_{i=1}^{m} \frac{Ci}{Pi} \leq 1$$

多处理机：处理机数 N

$$\sum_{i=1}^{m} \frac{Ci}{Pi} \leq N$$

实时调度算法**分类：**

按实时任务性质（即对时间约束的强弱程度）

硬实时调度：必须满足任务截止期要求，错过后果严重。

软实时调度算法：期望满足任务截止期要求，错过可容忍。

按调度方式

非抢占式调度算法

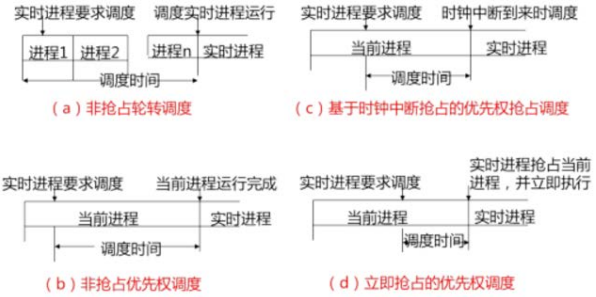
非抢占式轮转调度算法：用于工业生产的群控系统中。

非抢占式优先调度算法：用于有一定时间要求的实时控制系统之中。

抢占式调度算法 (按抢占发生的时间)

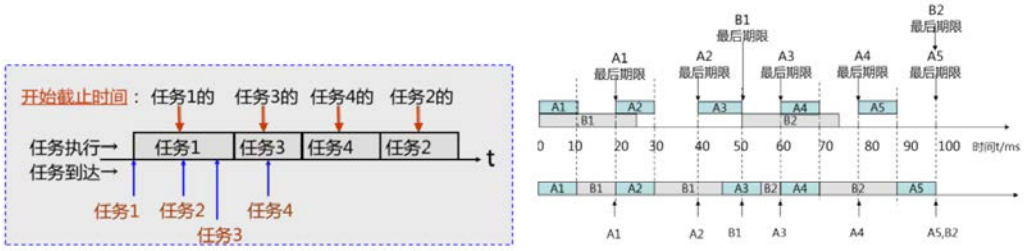
基于时钟中断抢占的优先权调度算法

立即抢占的优先权调度算法



就绪时间、开始截止时间和完成截止时间（周期性任务、非周期性任务）、处理时间、资源要求、优先级

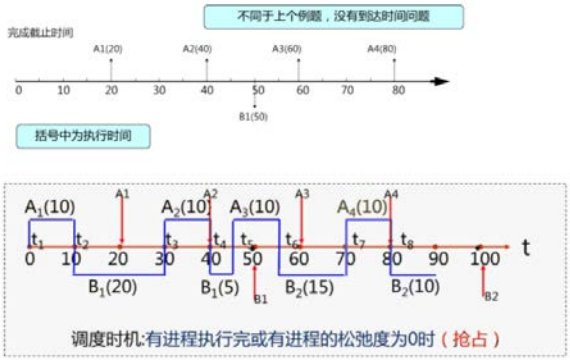
最早截至时间优先 EDF：根据开始截止时间确定优先级，开始截止时间越早，优先级越高，保持一个实时任务就绪队列，按各任务截止时间的早晚排序。可用于非抢占式（下图左），与抢占式（下图右，A 周期 20ms,处理时间 10ms，B 周期 50ms，处理时间 25ms）。



非抢占　　抢占式

最低松弛度优先 LLF：根据任务的松弛程度来确定任务的优先级，任务的紧急程度越高，优先级越高。系统中有一个按松弛度排序的实时任务就绪队列，松弛度最低的任务排在队列的最前面，调度程序总是选择就绪队列的队首任务执行。主要用于可抢占式调度方式中。进程一次执行结束调用队列队首任务或者松弛度为 0 时，抢占。

松弛度 = 必须完成时间 − 本身运行时间 − 当前时间



A 周期 20ms,处理时间 10ms，B 周期 50ms，处理时间 25ms。

优先级倒置的解决办法：Priority Ceiling、Priority Inheritance

产生死锁的原因：

竞争资源，多个进程竞争数目不足的共享资源

进程间推进顺序非法，请求释放资源的顺序不当

可剥夺性资源：CPU、主存

不可剥夺性资源：磁带机、打印机

临时性资源

死锁的四个必要条件：

互斥条件：指进程对所分配的资源进行排它性使用，即在一段时间内某资源只由一个进程占用。如果还有其他进程请求该资源，智能等待。

请求与保持条件：指进程保持了至少一个资源，但又提出了新的资源请求，而该资源又已被其他进程占用，此时请求进程阻塞，但又对自己已获得的其他资源保持不放。

不剥夺条件：指进程已获得的资源，在未使用完之前，不能被剥夺，只能在使用完时由自己释放。

环路等待条件：指在发生死锁时，必然存在一个进程——资源的环形链，即进程集合{P0,P1,…,Pn}中进程 P0 正在等待 P1 占用的资源，P1 正在等待 P2 占用的资源，…，Pn 正在等待 P0 占用的资源。

处理死锁的基本方法：

预防死锁：破坏死锁的四个必要条件，可能导致系统资源的利用率和系统吞吐量低。

摒弃“请求保持”条件：系统规定，在所有进程运行之前，都必须一次性地申请其在整个运行过程所需的全部资源。摒弃了请求的条件。缺点：浪费资源，进程延迟运行。

摒弃“不剥夺”条件：进程逐个地提出对资源的要求，当一个已经保持了某些资源的进程，再提出新的资源请求而不能立即得到满足时，必须释放它已经保持的所有资源，待以后重新申请。实现复杂，代价较大，延长了进程的周转时间，增加了系统开销，降低了系统吞吐量。

摒弃“环路等待”条件：系统将所有资源按类型进行线性排队，并赋予不同的序号，所有进程对资源的请求必须严格按照资源序号递增的次序提出。这种策略，总有一个进程占据了较高序号的资源，此后它继续申请的资源必然是空闲的，因而进程可以一直向前推进。影响扩展性、资源浪费、限制用户简单自主的编程。

避免死锁：在资源的动态分配过程中，采用某种方法防止系统进入不安全的状态，从而避免发生死锁。

安全状态：是指系统能够按照某种进程顺序（P1,P2,…,Pn）（称为安全序列），来为每个进程 Pi 分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利的完成。若果系统无法找到这样一个安全序列，则称系统处于不安全状态。

不安全状态

银行家算法数据结构

1）可利用资源向量 Available

是个含有 m 个元素的数组，其中的每一个元素代表一类可利用的资源数目。如果 Available[j]=K，则表示系统中现有 Rj 类资源 K 个。

2）最大需求矩阵 Max

这是一个 n×m 的矩阵，它定义了系统中 n 个进程中的每一个进程对 m 类资源的最大需求。如果 Max[i,j]=K，则表示进程 i 需要 Rj 类资源的最大数目为 K。

3）分配矩阵 Allocation

这也是一个 n×m 的矩阵，它定义了系统中每一类资源当前已分配给每一进程的资源数。如果 Allocation[i,j]=K，则表示进程 i 当前已分得 Rj 类资源的 数目为 K。
4）需求矩阵 Need。
这也是一个 n×m 的矩阵，用以表示每一个进程尚需的各类资源数。如果 Need[i,j]=K，则表示进程 i 还需要 Rj 类资源 K 个，方能完成其任务。
Need[i,j]=Max[i,j]-Allocation[i,j]

算法 的实现
一、初始化
由用户输入数据，分别对可利用资源向量矩阵 AVAILABLE 、 最大需求矩阵 MAX 、分配矩阵 ALLOCATION、 需求矩阵 NEED 赋值。
二、银行家算法
在该方法中把系统的状态分为**安全**状态和不安全状态，只要能使系统始终都处于安全状态，便可以避免发生死锁。
银行家算法的基本思想是分配资源之前, 判断系统是否是安全的; 若是, 才分配。
设进程 cusneed 提出请求 REQUEST [i] ，则银行家算法按如下规则进行判断。
(1) 如果 REQUEST [cusneed] [i]<= NEED[cusneed][i] ，则转(2) ；否则，出错。
(2) 如果 REQUEST [cusneed] [i]<= AVAILABLE[cusneed][i] ，则转(3) ；否则，出错。
(3) 系统试探分配资源，修改相关数据：
AVAILABLE[i]-=REQUEST[cusneed][i];
ALLOCATION[cusneed][i]+=REQUEST[cusneed][i];
NEED[cusneed][i]-=REQUEST[cusneed][i];
(4) 系统执行安全性检查，如安全，则分配成立；否则试探险性分配作废，系统**恢复** 原状，进程等待。
三、安全性检查算法
(1) 设置两个工作向量 Work=AVAILABLE;FINISH
(2) 从进程集合中找到一个满足下述条件的进程，
FINISH[i]==false;
NEED[i,j]<=Work[j];
如找到，执行(3) ； 否则，执行(4)
(3) 设进程获得资源，可顺利执行，直至完成，从而释放资源。
Work[j]+=ALLOCATION[i,j];
Finish[i]=true;
GOTO 2
(4) 如所有的进程 Finish= true ，则表示安全；否则系统不安全。
操作系统安全状态和不安全状态：
安全序列是指一个进程序列{P1，…，Pn}是安全的，如果对于每一个进程 Pi(1≤i≤n)，它以后尚需要的资源量不超过系统当前剩余资源量与所有进程 Pj (1<=j < i)当前占有资源
量之和。
如果存在一个由系统中所有进程构成的安全序列 P1，…，Pn，则系统处于安全状态。安全状态一定是没有死锁发生。
不存在一个安全序列。不安全状态不一定导致死锁。

检测死锁：资源分配图，进程节点 P，资源节点 R，边 E，e 一边是 p 一边是 r。在资源分配

图中找到一个既不阻塞又非独立的进程结点 Pi，消去 Pi 所有的请求边与分配边，使之成为孤立结点，一次简化各进程，若能消去图中所有的边，使所有进程结点成为孤立结点，则称该图是可完全简化的，否则是不可完全简化的。
死锁的充分条件：当且仅当 S 状态的资源分配图是不可完全简化的，s 为死锁状态。
解除死锁：剥夺资源（从其他进程剥夺足够的资源给死锁进程，以解除死锁）、撤销进程（使全部死锁进程夭折，按照某种顺序逐个撤销进程，直至有足够的资源可用）

内核态与用户态的区别

http://www.cnblogs.com/viviwind/archive/2012/09/22/2698450.html

内核态：当一个任务（进程）通过系统调用陷入内核代码中执行时，就称进程处于内核运行态，此时处理器处于**特权级**最高的（**0 级**）内核代码中执行。处于内核态的进程，使用当前进程的内核栈执行内核代码。处于内核态的进程，可以访问内存所有数据，包括外围设备，也可以从一个进程切换到另一个进程。

用户态：进程执行用户自己的代码时，则称其处于用户运行状态（用户态），此时处理器处于**特权级（3 级）**最低的用户代码中执行，当正在执行用户程序而突然被中断程序中断时，此时用户也可以象征性的称为处于进程的内核态，因为中断处理程序将使用当前进程的内核栈。处于用户态的进程只能受限地访问内存，且不允许访问外围设备，占用 CPU 的能力被剥夺，CPU 的资源可以被其他程序获取，运行在用户态下的程序不能直接访问操作系统内核数据结构和程序。

Intel x86 架构的 CPU 来说一共有 0~3 四个特权级，0 级最高，3 级最低，Unix/Linux 只使用了 0 级特权级和 3 级特权级。Linux 进程的 4G 地址空间，3G-4G 是共享的内核态的地址空间，存放整个内核的代码和所有的内核模块，以及内核所维护的数据。用户运行一个程序，该程序所创建的进程开始是运 行在用户态的，如果要执行文件操作，网络数据发送等操作，必须通过 write，send 等系统调用，这些系统调用会调用内核中的代码来完成操作，这时，必须切换到 Ring0，然后进入 3GB-4GB 中的内核地址空间去执行这些代码完成操作，完成后，切换回 Ring3，回到用户态。这样，用户态的程序就不能 随意操作内核地址空间，具有一定的安全保护作用。

用户态和内核态的转换：

a. 系统调用

这是用户态进程主动要求切换到内核态的一种方式，用户态进程通过系统调用申请使用操作系统提供的服务程序完成工作。而系统调用的机制其核心还是使用了操作系统为用户特别开放的一个中断来实现，例如 Linux 的 int 80h 中断。

b. 异常

当 CPU 在执行运行在用户态下的程序时，发生了某些事先不可知的异常，这时会触发由当前运行进程切换到处理此异常的内核相关程序中，也就转到了内核态，比如缺页异常。

c. 外围设备的中断

当外围设备完成用户请求的操作后，会向 CPU 发出相应的中断信号，这时 CPU 会暂停执行下一条即将要执行的指令转而去执行与中断信号对应的处理程序，如果先前执行的指令是用户态下的程序，那么这个转换的过程自然也就发生了由用户态到内核态的切换。比如硬盘读写操作完成，系统会切换到硬盘读写的中断处理程序中执行后续操作等。

由用户态切换到内核态的步骤主要包括：

[1] 从当前进程的描述符中提取其内核栈的 ss0 及 esp0 信息。

[2] 使用 ss0 和 esp0 指向的内核栈将当前进程的 cs,eip,eflags,ss,esp 信息保存起来，这个过程也完成了由用户栈到内核栈的切换过程，同时保存了被暂停执行的程序的下一条指令。

[3] 将先前由中断向量检索得到的中断处理程序的 cs,eip 信息装入相应的寄存器，开始执行中断处理程序，这时就转到了内核态的程序执行了。

Linux 指令：

http://www.cnblogs.com/laov/p/3541414.html

多线程同步：

linux 看栈有多大，多进程、多线程、通信、调试
设计模式
Ulimit –a
Memory cache
Map Reduce
Memcpy

MySQL 数据库

参考网页：http://www.cnblogs.com/mr-wid/archive/2013/05/09/3068229.html
关系型数据库：SQLSever、MySQL、Oracle、PostgreSQL
非关系型数据库：NoSQL(MongoSQL)，更加灵活，可以是 JSON 文档、哈希表或其他

MySQL 服务的**启动、停止与卸载**
启动：net start MySQL
停止：net stop MySQL
卸载：sc delete MySQL
修改 root 密码：

mysqladmin -u root -p password 新密码

执行后提示输入旧密码完成密码修改，当旧密码为空时直接按回车键确认即可。

查看数据库用到的编码：

show variables like 'character%'

MySQL 中的**数据类型**(http://www.cnblogs.com/zbseoag/archive/2013/03/19/2970004.html)

- 整数类型
  - 整型：tinyint (1B), smallint (2B), mediumint (3B), int (4B), bigint(8B)
  - 浮点型：float(4B), double(8B), real, decimal
- 日期与时间：data, time, datetime, timestamp, year
- 字符串类型
  - 字符串：char(<=255), varchar(<=65535)(固定长度)
  - 文本：tinytext(<=255), text(<64K), mediumtext(<16M), longtext(<4G)（可变长度）
  - 二进制：tinyblob, blob, mediumblob, longblob

MySQL 关键字	含义
NULL	数据列可包含 NULL 值
NOT NULL	数据列不允许包含 NULL 值
DEFAULT	默认值
PRIMARY KEY	主键
AUTO_INCREMENT	自动递增，适用于整数类型
UNSIGNED	无符号
CHARACTER SET name	指定一个字符集

数据库操作：
登录到 MySQL
mysql –h 主机名 –u 用户名 -p

创建一个数据库

create database 数据库名 [其他选项];

例：create samp_db character set gbk;

成功后返回：Query OK 1 row affected(0.02 sec)

注意：MySQL 语句以分号作为语句的结束

查看创建了哪些数据库

show databases;

选择要操作的数据库

use samp_db;

或登录时，使用 mysql –D samp_db –h 主机名 –u 用户名 –p

创建数据库表

create table 表名称(列声明);

例：

```
create table students
(
    id int unsigned not null auto_increment primary key,
    name char(8) not null,
    sex char(4) not null,
    age tinyint unsigned not null,
    tel char(13) null default "-"
);
```

提示：可将 mysql 脚本语句写入.sql（createtable.sql）文件中，通过命令提示符下的文件重定向执行该脚本。mysql -D samp_db -u root -p < createtable.sql

使用 show tables;查看已创建的表，describe 表名;查看已创建表的详细信息

向表中插入数据

insert [into] 表名 [(列名 1,列名 2,…)] values (值 1,值 2,…);

例：insert into students values(NULL, "Jone", "男", 20, "18811112222");

插入部分数据：insert into students (name, sex, age) values("王五", "男", 22);

查询表中数据

select 列名称 from 表名称 [查询条件];

例：select name, age from student;

select 列名称 from 表名称 where 条件;

例：select * from students where name like "%王%";

更新表中数据

update 表名称 set 列名称=新值 where 更新条件;

例：update students set tel = default where id = 5;

删除表中数据

delete from 表名称 where 删除条件;

例：delete from students where id=2;

删除表中数据：delete from students;

创建表后修改表

alter table

添加列

alter table 表名 add 列名 列数据类型 [after 插入位置];

例：alter table students add address char(60) after tel;

修改列

alter table 表名称 change 列名称 新名称 新数据类型;

例：alter table students change tel telephone char(13) default “-”;

删除列

alter table 表名称 drop 列名称;

例：alter students drop address;

重命名表

alter table 表名称 rename 新名称;

删除整张表

drop table 表名称;

删除整个数据库

drop database 数据库名;

导入导出数据

http://www.runoob.com/mysql/mysql-database-import.html

聚集索引和非聚集索引

http://www.cnblogs.com/aspnethot/articles/1504082.html

聚集索引：索引的逻辑顺序决定了表中相应行的物理顺序。聚集索引确定表中数据的物理顺序，索引的叶结点就是数据结点。聚集索引对于经常要搜索的范围值的列特别有效，使用聚集索引找到包含第一个值的行后，便可以确保包含后续索引值的行在物理相邻。每个表只能有一个聚集索引。正文本身就是一个聚集索引。

非聚集索引：索引的逻辑顺序与磁盘上行的物理存储顺序不同。索引的叶结点任然是索引结点，只不过有一个指针指向对应的数据块。目录纯粹是目录，正文纯粹是正文。

何时使用聚集索引或非聚集索引（很重要）：

动作描述	使用聚集索引	使用非聚集索引
列经常被分组排序	应	应
返回某范围内的数据	应	不应
一个或极少不同值	不应	不应
小数目的不同值	应	不应
大数目的不同值	不应	应
频繁更新的列	不应	应
外键列	应	应
主键列	应	应
频繁修改索引列	不应	应

计算机网络

1. 分层模型
2. TCP

TCP 包头：

8bits	8bits	8bits	8bits
源端口		目的端口	
序列号			
确认序列号			
4 bits 头部长度, 6 bits 保留位, UAPRSF		窗口大小	
校验和		紧急指针	
选项			

URG 紧急指针，告诉接收 TCP 模块紧要指针域指着紧要数据。
ACK 置 1 时表示确认号（为合法，为 0 的时候表示数据段不包含确认信息，确认号被忽略。
PSH 置 1 时请求的数据段在接收方得到后就可直接送到应用程序，而不必等到缓冲区满时才传送。
RST 置 1 时重建连接。如果接收到 RST 位时候，通常发生了某些错误。
SYN 置 1 时用来发起一个连接。
FIN 置 1 时表示发端完成发送任务。用来释放连接，表明发送方已经没有数据发送了。

UDP 包头：

8 bits	8 bits	8 bits	8 bits
源端口		目的端口	
用户数据包长度		校验和	

IP 包头：

8 bits	8bits	8 bits	8bits
版本号	报头长度	服务类型	总长度
标识		3 标识	13bits 段偏移
TTL	协议	校验和	
源 IP 地址			
目的 IP 地址			
Option			

服务类型：
前 3 比特为优先权子字段（Precedence，现已被忽略），第 4 至第 7 比特分别代表延迟、吞吐量、可靠性和花费，第 8 比特保留未用。
总长度字段：占 16 比特。指明整个数据报的长度（以字节为单位）。最大长度为 65535 字节。
标志字段：占 16 比特。用来唯一地标识主机发送的每一份数据报。通常每发一份报文，它的值会加 1。
标志位字段：占 3 比特。标志一份数据报是否要求分段。
段偏移字段：占 13 比特。如果一份数据报要求分段的话，此字段指明该段偏移距原始数据报开始的位置。
协议字段：占 8 比特。指明 IP 层所封装的上层协议类型，如 ICMP（1）、IGMP（2） 、TCP

（6）、UDP（17）等
DNS 协议同时使用了 TCP 53 端口和 UDP 53 端口。DNS 协议在 UDP 的 53 端口提供域名解析服务，在 TCP 的 53 端口提供 DNS 区域文件传输服务

三次握手
四次挥手
为什么需要四次挥手？
为什么 TCP 能够提供可靠的连接？

因为 UDP 没有应答和重传过程，TCP 通过合理地截断数据包，超时重传，校验，失序重新排序，丢弃重复数据，流量控制保证可靠的连接。

1） 应用数据被分割成 TCP 认为最适合发送的数据块，而 UDP 中，应用程序产生的数据报长度保持不变。（将数据截断为合理的长度）
2） 当 TCP 发出一个段后，会启动一个定时器，等待目的端确认接收这个报文段，若果不能及时接收确认，将重发这个报文段。（超时重传）
3） 当 TCP 接收到另一端发送的数据后，会发送一个确认，这个确认包会有推迟。(对于收到的请求，给出确认响应，推迟是因为可能是要对包做完整校验)
4） TCP 将保持它头部和数据的校验和，这是一个端到端的校验和，目的是监测数据在传输过程中的任何变化。如果收到段的校验和有差错，TCP 将丢弃这个报文段和不确认接收此报文段，希望发送端超时重传。（校验包出错，丢弃报文段，不给响应，发送端超时重传）
5） TCP 报文段作为 IP 数据报来传输，IP 数据报的到达可能会失序，TCP 报文段的到达也会失序，如果有必要，TCP 会对收到的数据进行重新排序，将收到的数据以正确的顺序交给应用层。（对失序的数据进行重新排序后，才交给应用层）
6） IP 数据报会发生重复，TCP 接收端必须丢弃重复数据。（丢弃重复数据）
7） TCP 能提供流量控制，TCP 连接的每一方都有固定大小的缓冲空间（滑动窗口）。TCP 的接收端只允许另一端发送接收端缓冲区所能接纳的数据。防止较快主机导致较慢主机的缓冲区溢出。（TCP 可以进行流量控制，防止较快主机致使较慢主机的缓冲区溢出）

怎么通过 UDP 实现 HTTP 协议？

路由器与交换机

路由器属于网络层，通过 IP 地址转发数据包，交换机属于数据链路层，使用 MAC 地址转发

3. UDP
4. HTTP
5. IP
6. DNS

C++基础

红黑树：
http://www.cnblogs.com/skywang12345/p/3624177.html
http://www.imooc.com/article/11715

排序算法