面试大纲

一、c语言技术点

1、描述一下gcc的编译过程?

gcc编译过程分为4个阶段：预处理、编译、汇编、链接。

预处理：头文件包含、宏替换、条件编译、删除注释

编译：主要进行词法、语法、语义分析等，检查无误后将预处理好的文件编译成汇编文件。

汇编：将汇编文件转换成 二进制目标文件

链接：将项目中的各个二进制文件+所需的库+启动代码链接成可执行文件

2、内存的最小存储单位以及内存的最小计量单位分别是?

内存的最小存储单位为 二进制位， 内存的最小计量单位 字节

3、#include<> 与#include ""的区别?

include<>到系统指定目录寻找头文件，#include ""先到项目所在目录寻找头文件，如果没有找再到系统指定的目录下寻找

4、描述一下变量的命名规则

变量名有字母、数值、下划线组成，但不能以数值开头

5、变量的声明与定义有啥区别?

声明变量 不需要 建立存储空间， 变量的定义需要建立存储空间

6、谈谈c语言中有符号和无符号的区别?

有符号：数据的最高位为符号位，0表示正数，1表示负数

无符号：数据的最高位不是符号位，而是数据的一部分

7、谈谈计算机中补码的意义

统一了零的编码

将符号位与其他位统一处理

将减法运算转换成加法运算

8、谈谈数组的特点

同一个数组所有的成员都是相同的数据类型，同时所有的成员在内存中的地址是连续的

9、数组的分类

数组的分类主要是：静态数组、动态数组两类。

静态数组：类似int arr[5];在程序运行就确定了数组的大小，运行过程不能更改数组的大小。

动态数组：主要是在堆区申请的空间，数组的大小是在程序运行过程中确定，可以更改数组的大小。

10、描述一下一维数组的不初始化、部分初始化、完全初始化的不同点

不初始化：如果是局部数组 数组元素的内容随机 如果是全局数组，数组的元素内容为0

部分初始化：未被初始化的部分自动补0

完全初始化：如果一个数组全部初始化 可以省略元素的个数 数组的大小由初始化的个数确定

11、谈谈数组名作为类型、作为地址、对数组名取地址的区别?

数组名作为类型：代表的是整个数组的大小

数组名作为地址：代表的是数组首元素的地址

对数组名取地址：代表的是数组的首地址

12、谈谈你对二维数组在物理上以及逻辑上的数组维度理解

二维数组在逻辑上是二维的，在物理上是一维的

13、描述一下函数的定义与函数的声明的区别

函数定义：是指对函数功能的确立，包括指定函数名、函数类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。

函数的声明：是把函数的名字、函数类型以及形参的个数、类型和顺序(注意，不包括函数体)通知编译系统，以便在对包含函数调用的语句进行编译时，据此对其进行对照检查（例如函数名是否正确，实参与形参的类型和个数是否一致）

14、描述一下指针与指针变量的区别

指针：没存中每一个字节都会分配一个32位或64位的编号，这个编号就是地址, 而指针就是内存单元的编号。

指针变量：本质是变量 只是该变量存放的是空间的地址编号

15、描述一下32位或64位平台下指针的大小

32位平台：任意类型的指针大小为4字节

64位平台：任意类型的指针大小为8字节

16、描述一下指针数组的概念

指针数组本质是数组，只是数组的每个元素是一个指针（地址）

17、描述一下普通局部变量、普通全局变量、静态局部变量、静态全局变量的区别

普通局部变量：

存在栈区、不初始化内容随机、只在定义所在的复合语句中有效、符合语句结束变量空间释放

普通全局变量

存在全局区、不初始化内容为0、进程结束空间才被释放，能被当前源文件或其他源文件使用，只是其他源文件使用的时候，记得使用extern修饰

静态局部变量：

存在全局区、不初始化内容为0、整个进程结束空间才被释放，只能在定义所在的复合语句中有效

静态全局变量

存在全局区、不初始化内容为0、整个进程结束空间才被释放，只能被当前源文件使用

18、描述一下内存分区

程序在运行前：分为代码区、BSS段（未初始化数据区）、data段(初始化数据区)

程序在运行后：堆区、栈区、全局区（静态区）、文字常量区、代码区

19、在使用realloc给已分配的堆区空间追加空间时需要注意啥?

记得用指针变量保存realloc的返回值

20、结构体与共用体的区别是啥

结构体中的成员拥有独立的空间，共用体的成员共享同一块空间，但是每个共用体成员能访问共用区的空间大小是由成员自身的类型决定

21、谈谈文件的分类

文件分为二进制和文本文件

二进制文件基于值编码，需要根据具体的应用才能知道某个值具体的含义

文本文件基于字符编码，一个字节一个意思，可以通过记事本打开

22、文件缓冲区刷新方式有几种

行刷新、满刷新、强制刷新、关闭刷新

23、哪些情况下会出现野指针

指针变量未初始化、指针释放后未为置空、指针操作超越变量作用域

24、如何理解指针作为函数参数的输入和输出特性

输入特性：主调函数分配空间 背调函数使用该空间

输出特性：被调用分配空间 主调函数使用该空间

25、如何理解结构体的浅拷贝与深拷贝

当结构体中有指针成员的时候容易出现浅拷贝与深拷贝的问题。

浅拷贝就是，两个结构体变量的指针成员指向同一块堆区空间，在各个结构体变量释放的时候会出现多次释放同一段堆区空间

深拷贝就是，让两个结构体变量的指针成员分别指向不同的堆区空间，只是空间内容拷贝一份，这样在各个结构体变量释放的时候就不会出现多次释放同一段堆区空间的问题

26、描述一下结构体对齐规则

1. 数组成员对齐规则。第一个数组成员应该放在offset为0的地方，以后每个数组成员应该放在offset为min（当前成员的大小，#pargama pack(n)）整数倍的地方开始（比如int在32位机器为 4 字节，#pargama pack(2)，那么从2的倍数地方开始存储）。
2. 结构体总的大小，也就是sizeof的结果，必须是min（结构体内部最大成员，#pargama pack(n)）的整数倍，不足要补齐。
3. 结构体做为成员的对齐规则。如果一个结构体B里嵌套另一个结构体A,还是以最大成员类型的大小对齐，但是结构体A的起点为A内部最大成员的整数倍的地方。（struct B里存有struct A，A里有char，int，double等成员，那A应该从8的整数倍开始存储。），结构体A中的成员的对齐规则仍满足原则1、原则2。

27、啥叫宏函数以及作用

在项目中，经常把一些短小而又频繁使用的函数写成宏函数，这是由于宏函数没有普通函数参数压栈、跳转、返回等的开销，可以调高程序的效率。 宏通过使用参数，可以创建外形和作用都与函数类似地类函数宏(function-like macro). 宏的参数也用圆括号括起来，来保证宏函数的完整性。

28、如何理解库函数

库是已经写好的、成熟的、可复用的代码。每个程序都需要依赖很多底层库，不可能每个人的代码从零开始编写代码，因此库的存在具有非常重要的意义。 在我们的开发的应用中经常有一些公共代码是需要反复使用的，就把这些代码编译为库文件。 库可以简单看成一组目标文件的集合，将这些目标文件经过压缩打包之后形成的一个文件。像在Windows这样的平台上，最常用的c语言库是由集成按开发环境所附带的运行库，这些库一般由编译厂商提供

二、基础及系统编程技术点

1、shell是什么

Shell 是“命令解释器”（人和内核打交道的翻译） Shell 作用：对用户输入到终端的命令进行解析，调用对应的执行程序

2、常见的 shell 及查看 shell 的方法

unix默认的shell为sh linux默认的shell为bash

查看shell的方法：cat /etc/shells 或 echo $SHELL

3、简单描述一下linux下的目录结构

/bin 存放系统可执行程序（大部分系统命令） /sbin 存放 root 用户的系统可执行程序 /boot 存放内核和启动程序的相关文件 /lib 库目录，存放系统最基本的动态库 /media 默认挂载设备媒体的目录，例如 U 盘、光驱 /mut 推荐挂载设备媒体的目录 /usr 用于存放庞大而复杂的目录(unix system resource，用于安装软件的目录) /proc 系统内存的映射（随着系统的运行，时长变化的） /ect 系统软件的启动和配置目录 /dev 用于存放设备文件 /home 家目录，所用用户的根目录（当前用户的根目录是 /home/user）

4、linux下的权限分为哪3个组

用户权限、同组用户权限、其他用户权限

5、linux下文件的类型

普通文件 d 目录文件 c 字符设备文件 b 块设备文件 l (软)连接文件 p 管道文件 s 本地套接字 // 网络编程中介绍

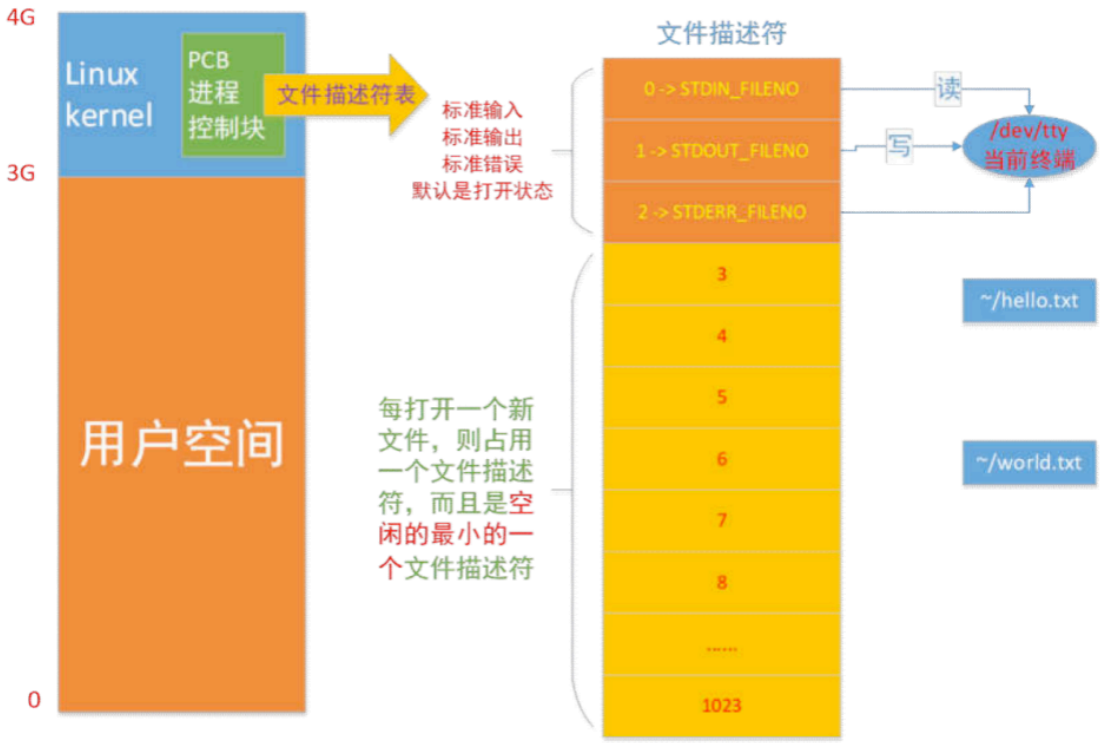
6、描述一下软连接和硬链接的区别

软链接：ln -s 源文件 目标文件 硬链接：ln 源文件 目标文件 源文件：即你要对谁建立链接

1，软链接可以理解成快捷方式。它和windows下的快捷方式的作用是一样的。 2，硬链接等于cp -p 加同步更新。

7、谈谈你对文件描述表的理解

内核区-->进程管理 --> PCB 进程控制块（超大结构体）--> 文件描述符表（用于寻磁盘文件） 一个进程会对应一个文件描述符表，每打开一个文件会占用一个位置 一个文件描述表本质上是一个数组，最多可以容纳 1024（编号：0-1023）个文件描述符 前 3 个（0-2）默认是打开状态的（被占用），分别的标准输入、标准输出、标准错误



8、谈谈你对程序和进程的理解

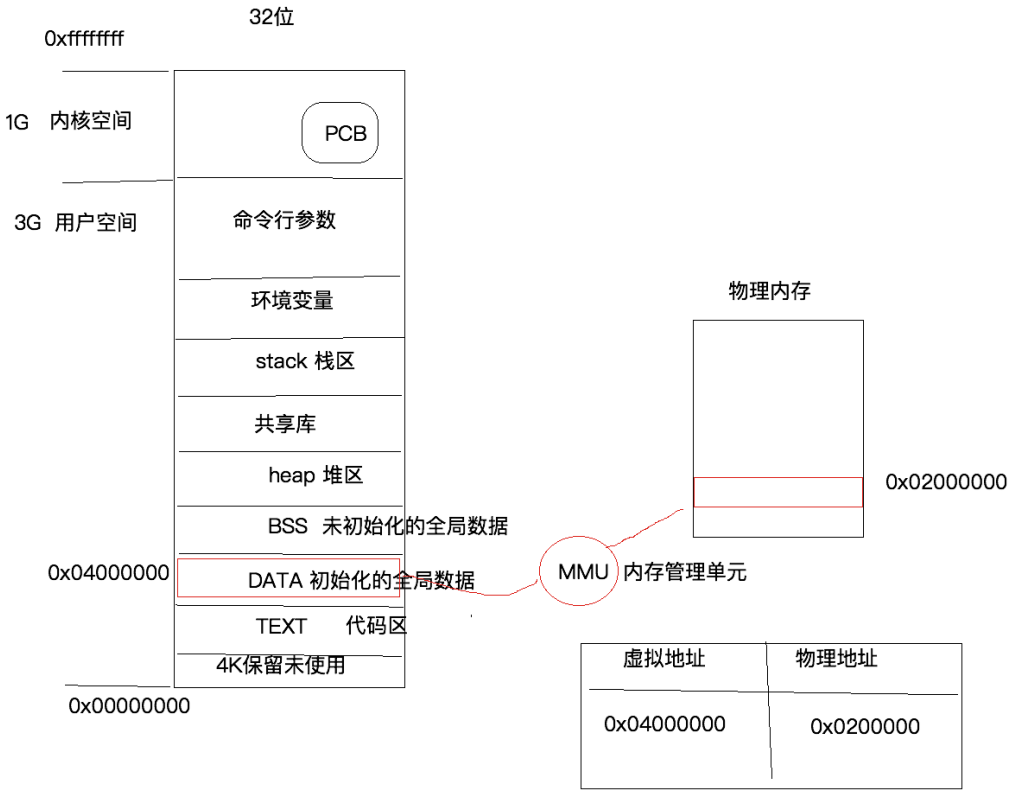
程序： 本质：二进制文件，可以运行，但还没有运行 占用磁盘空间，不占用CPU 和内存（系统资源）
进程： 本质：正在执行的程序 占用CPU 和内存等更多的系统资源，一般不占用磁盘空间（I/O 操作可能会占用磁盘空间）

站在程序员的角度：进程是一系列指令的执行过程 站在操作系统的角度：进程是分配系统的资源的最小单位 关系： 一个程序可以对应多个进程，但一个进程只能对应一个程序

9、谈谈你对并行和并发的理解

并发： 1、CPU 将 1s 分成若干个时间碎片，每个时间碎片CPU 只能执行一个进程的一小部分 2、以时间碎片为单位，若干个进程循环占有CPU，并执行对应进程的一小部分 3、经过 n 次循环占有 CPU，每个进程才能执行完毕 即：多个进程以时间碎片为单位，循环占有CPU，以完成同时执行的现象称为并发注意：CPU 处理是纳秒级，肉眼识别是毫秒级的，虽然每个进程的执行是间断的，但肉眼感官上是连续的。 并行： 并发发生在单核（只能有一个并发链） 并行发生在多核（可以有多个并发链

10、谈谈你对MMU(内存管理单元)的理解

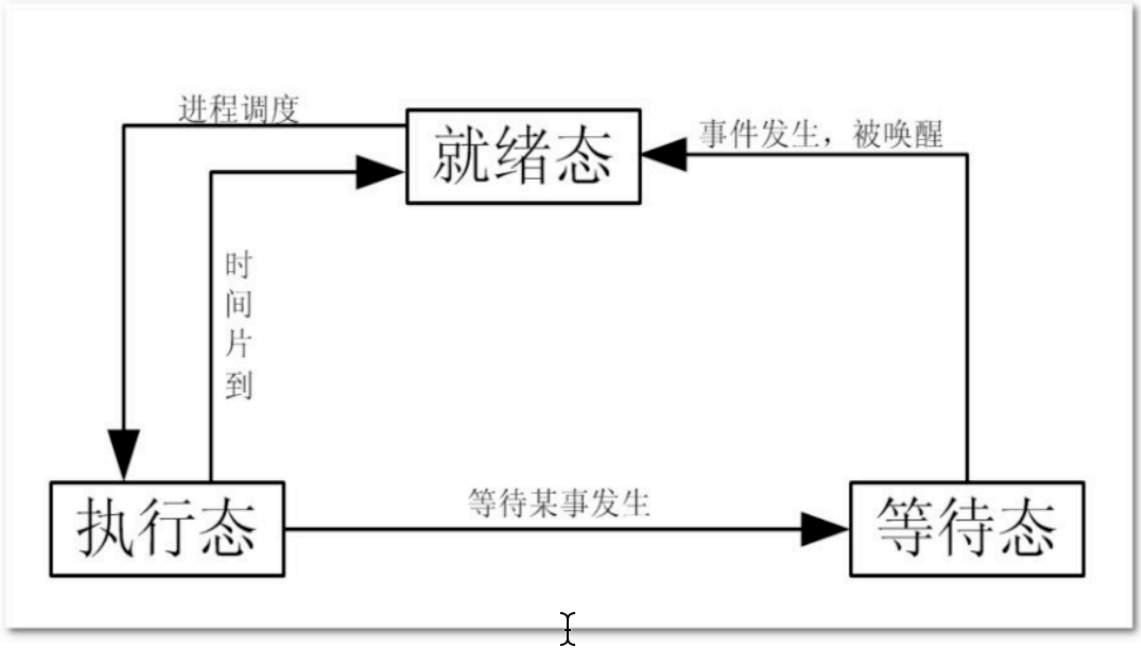


11、谈谈你对PCB（进程控制块）的理解

PCB 本质是一个超大的结构体 PCB 结构体中重要属性： 进程 pid ： 进程的唯一编号，类型 pid_t，无符号整形 进程状态：就绪、运行、挂起、停止 CPU 寄存器：进程只能短时间的占有 CPU，它用于进程切换时候保存和恢复进程的执行进度 描述虚拟地址空间的信息：每启动一个进程，就会对应一个虚拟地址空间（理论值 0 - 4G） 描述控制终端的信息 当前工作目录 umask 掩码：默认 0002，每一进程都有自己的 umask 掩码 文件描述符表 和信号相关的信息 用户 id 和组 id 进程组和会话：多个进程构成一个进程组，多个进程组构成一个会话 进程可以使用的资源上限：比如文件描述符的最大个数、管道 和 栈 的默认缓存区上限等

12、进程的状态有几种

进程的状态：等待态 执行态 就绪态 就绪态：一切条件都具备 只差cpu时间片到来 等待态：缺少执行条件 执行态：cpu时间片到来 执行



进程三种状态的转换关系

13、谈谈你对fork创建的进程的数据共享模式

fork() 后，其实是拷贝父进程内容给子进程（只有进程pid 不同） 数据：刚创建子进程时，父子用户区数据完全相同，但父子后续各自进行不同操作，互不影响，各个进程的数据是完全独立的 原理剖析：进程运行时候，所有的计算都是在物理内存中的，父子进程“虚拟地址空间 用户区中的数据”都会通过 MMU 映射到物理内存 问题 1：父子进程的数据都会映射到内存中，此时在内存中，会有几份数据?

答：一份（早期 Linux 系统设计是两份，但那种设计太浪费内存空间） 问题 2：那如何实现父子进程的数据独立的?

答：读时共享，写时复制（父子有一个进程对数据进行修改操作，都是先复制出一份，然后修改复制数据） 思考：父子进程能否通过全局变量进行通信？ 答：不能，因为是父子的全局变量是相互独立的 进程访问的都是各自的经过复制的变量，内存不能共享，无法通信 PCB 中的文件描述符表、内存映射区：是共享的 例如： 父进程先打开一个文件，此时父进程文件描述符表第 3 号位被占用，指向文件，此时fork，子进程会拷贝父进程的文件描述符表，子进程第 3 号也指向该文件，意味着父子进程可以操作同一个文件 这种设计的原因：是父子进程通过 pipe、fifo、内存映射区、套接字 等实现父子进程通信的基础

14、请说出啥叫僵尸进程以及孤儿进程

孤儿进程：不是没有爹！（会被领养） 过程：父进程先结束，子进程还运行，子进程称为孤儿进程 注意：孤儿会被 init 进程领养（init 进程是所有孤儿进程的父进程）

设计领养目的:为了子进程结束后，可以回收子进程占有的系统资源 （进程结束后，子进程自己能够释放用户区空间，但无法释放 PCB，只能由父亲释放） 僵尸进程：是一个死的进程 过程：子进程结束，父进程没有回收子进程的 PCB，此时，子进程称为僵尸进程

15、进程间的通信方式有几种？它们之间的特点?

无名管道、有名管道、信号、消息队列、共享内存、socket

管道：最简单、数据只能读取一次半双工、匿名管道只能是有血缘的关系间通信：

命名管道：用于没有血缘关系之间的进程间通信

共享内存：效率高、不需要太多次的数据拷贝，可以直接进行读写，缺点是不能保证数据同步，只能借助信号量保证同步

信号：简单、携带的信息量少，使用在特定的场景，优先级高。建议不要使用信号进行进程间通信，因为信号的优先级高会打破原有进程的执行过程

socket：主要用于网络中的进程间通信，通信过程以及数据复杂，但安全可靠。

16、谈谈守护进程的特点以及创建过程

特点：后台服务程序、不受终端控制、周期执行某个任务、不受用户登陆注销影响、一般以d结尾的名字

创建:

1、创建子进程，父进程退出

2、子进程创建新会话（该子进程既是会长也是组长：一个进程构成一个进程组，构成一个会话）

3、改变当前工作目录 int chdir(const char *path) //避免因为原工作目录由于某种原因失效，影响进程

4、重设文件掩码 umask：由于第一步的 fork，子进程会继承父进程的掩码，默认 0002 umask(0)，创建文件指定什么权限就是什么权限

5、关闭文件描述符：因为已经脱离终端，所以标准输入、输出、错误没有用了

6、执行核心工作

17、多进程和多线程的区别

进程是最小的系统资源分配单位，线程是最小CPU调度的执行单位

线程依赖于进程

进程始终共享的资源：代码段.txt，文件描述符，内存映射区

多线程始终共享的资源：堆、全局比变量 使用多线程好处：可以更简单的实现通信；而且可以节省资源（多线程共享虚拟地址空间）； 更合理的利用CPU（如果有两个 CPU，对于一个进程，只能占用一个，但创建一个线程，两个线程就都可以跑，在内核看来，占用同样的地址空间，但有 2 个进程在运作）

18、能够用exit退出某个线程?

不能! exit 是用来退出进程的，因为父子线程共享同一地址空间，如果在某线程中使用，会使所有线程全退出 因此需要使用别的函数 pthread_exit （注意：子线程使用 return NULL 也可以，如上面例子）单个线程退出函数：void pthread_exit(void *retval); -- 使某个线程退出，而不影响其他线程。

19、谈谈你对线程分离的理解

一般主控线程会回收子线程的资源，但是有时候需要子线程自己回收自己回收 PCB，因此调用该函数之后，不需要调用 pthread_join，设置了线程分离，该线程就是不可回收的，使用 pthread_join 反而会报错

20、谈谈你对同步互斥的理解

互斥：是指某一资源同时只允许一个访问者对其进行访问，具有唯一性和排它性。但互斥无法限制访问者对资源的访问顺序，即访问是无序的

同步:是指在互斥的基础上（大多数情况），通过其它机制实现访问者对资源的有序访问。在大多数情况下，同步已经实现了互斥，特别是所有写入资源的情况必定是互斥的。少数情况是指可以允许多个访问者同时访问资源.

总之：同步是一种更为复杂的互斥，而互斥是一种特殊的同步。

22、同步互斥的方式有哪些?

互斥锁、读写锁、条件变量、信号量

三、网络编程

1、及如何看待协议这个概念?

从应用的角度出发，协议可理解为“规则”，是数据传输和数据解释的规则 数据的发送方和接收方要严格遵照这些规则（这些规则肯定是之前就定好的） 例如：第一次发送文件名，第二次传输文件大小，第三次发送文件内容 ---- FTP 协议的雏形 试想：如果双方不遵照这个规则，就会出现数据混乱

2、网络程序的设计模式有哪些?

- C/S 模式： 传统的网络应用设计模式，客户机(client) / 服务器(server)模式。需要在通讯两端各自部署客户机和服务器来完成数据通信 优点：协议可以自定义（灵活） 数据可以提前缓存到本机上，后续运行快 缺点：客户端安装在主机电脑上，对用户的安全有一定威胁 需要分别开发客户端和服务器，而且需要联合调试，工作量大 使用场景：数据量访问比较大，要求稳定性较高
- B/S 模式： 浏览器(browser) / 服务器(server)模式。只需在一端部署服务器，而另外一端使用每台 PC 都默认配置的浏览器即可完成数据的传输。 优点：相对安全、工作量小、跨平台 缺点：协议是定死的，不能提前数据缓存使用场景：数据访问量较小

3、谈谈网络的分层结构

OSI/RM(理论上的标准)	TCP/IP(事实上的标准)
应用层	应用层
表示层	
会话层	
传输层	传输层
网络层	网络层
数据链路层	链路层
物理层	

七层模型：

- 物理层：不是指具体的物理设备 指的是物理设备的标准制定 （网线 光纤的接口类型 网卡的电流强弱） 比特流
- 数据链路层：负责完整的帧数据收发 (帧数据可以独立在网络传输的数据) mac地址封装和解封装 交换器就工作在这一层
- 网络层：负责IP地址封装和解封装 逻辑主机的识别 路由器工作在这一层 数据包
- 传输层：负责的是协议端口的封装和解封装 识别不同的进程通信 数据段 分组和重组
- 会话层：通过传输层 建立通信线路 发送或接受请求
- 表示层：对数据进行加密解密 压缩解压缩 将计算机能识别的信息转换成人能识别的信息
- 应用层：具体的网络通信app(QQ FQ Weixin 浏览器)

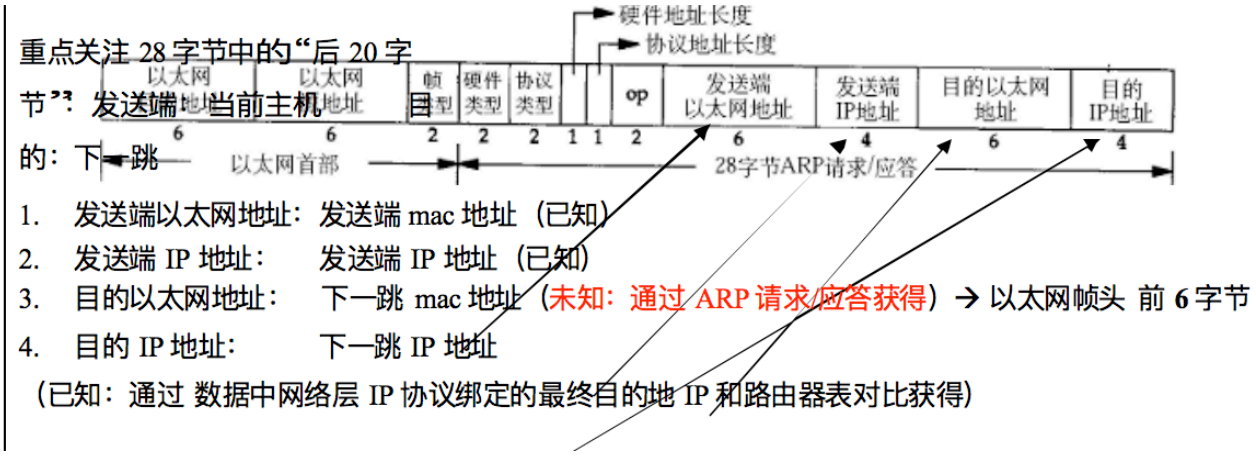
四层模型：

- 应用层：FTP 文件传送协议 telnet远程登录协议 http超文本传送协议
- 传输层：不同进程识别（端口） TCP 传输控制协议 UDP用户数据报协议
- 网络层：不同主机识别（IP） IP 网际协议 ICMP网络控制报文协议（ping命令）
- 数据链路层：不同物理设备（网卡）的识别（MAC地址） arp 地址解析协议（ip—>mac） rarp逆地址解析协议(mac———>ip)

4、我们是如何知道要前往的下一块网卡的mac地址（目的mac地址）？

通过arp表以及arp协议。链路层先查看路由表中是否记录了下一块网卡的mac地址，如果没有就调用arp协议获得下一块网卡的mac地址。

继续追问：能够描述一下arp协议的格式



5、如果判断一个网段的广播地址以及网段地址?

主机ID全为1就是网段的广播地址，主机ID全为0就是该网段的网段地址。

6、分别描述一下IP地址以及端口的作用

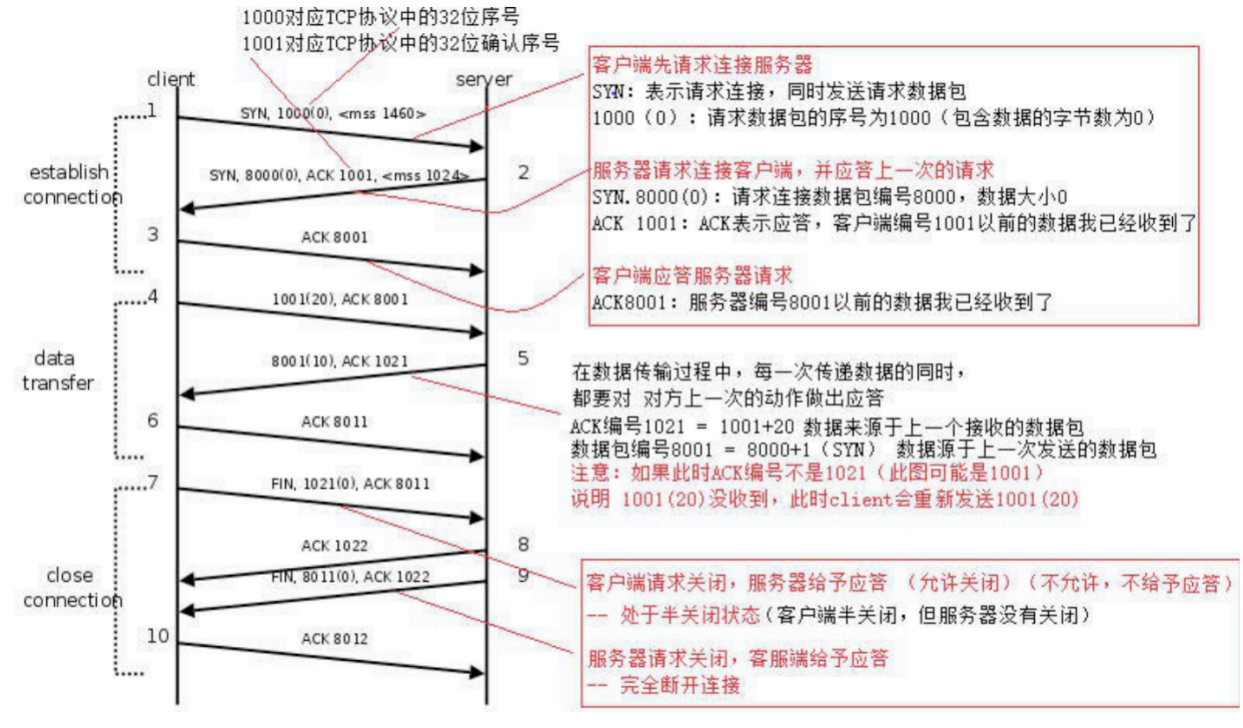
IP地址在网络环境中表示唯一的主机，端口可以在主机中表示唯一的进程。

网络层定义了源和目的 IP，传输层定义了源和目的的端口，这样就可以指定由“源主机的某进程”到“目的主机的某进程”的通信

7、谈谈你对字节序的理解

字节序就是主机存放数据的顺序，一般在异构计算机以及多字节的时候才讨论字节序。字节序的分为本地字节序，网络字节序。字节序的存储方式有大端存储以及小端存储，而网络字节序就是大端存储。

8、描述一下三次握手以及四次挥手的过程



9、描述一下TCP与UDP的区别

TCP与UDP都是传输层协议，进程与进程间的通信。

TCP传输控制协议：面向连接编程，需要建立连接—>使用链接—>关闭链接，失败重传、排序检错，面向数据传输，不支持广播、多播。

UDP用户报文协议：面向无连接编程，不排序，不重重传，不检错，简单的应答服务，支持广播、多播。

10、谈谈select的优缺点

优点：一个线程就可以支持多个客户端（多路 IO 都有这个优点！），可以跨平台（select 独有） 缺点：

- 同时监听文件描述符的上限是 1024 个 注意：不是因为打开文件的上限是 1024（上限数可以修改），而是因为在 select 底层实现时候，fd_set使用了宏 FD_SETSIZE = 1024
- 返回值是一个数量，需要循环遍历判断到底谁符合条件，因此高并发少访问的时候，效率低 监听集合和满足监听条件的集合是同一个集合，select 后会改变原监听集合，使其无法再次使用，因此，select 前需要将原监听集合保存

11、谈谈poll相较于select的优点

select 的监听和返回的集合是不分离的，是通过传入传出的方式实现而 poll 通过“参数 1”的结构体实现请求和返回分离 poll 相较于 select 的劣势：不能跨平台，只能在linux 下使用，导致 poll 地位比较尴尬

12、谈谈不同局域网的通信

数据从应用层到达传输层进行源/目的端口的封装，将数据传输到网络层。网络层封装IP数据报文，并判断目的IP是否是当前局域网，将封装好的IP数据包发送链路层，链路层先到arp表找下一块网卡的mac地址，如果存在，将封装源mac和目的mac，如果arp表中没有对应的mac就要调用arp协议广播得到下一块网卡的mac地址，然后在封装mac报文。网络数据到达路由器，路由器先查看数据包的目的ip是否与路由器的某个本地网卡是同一个网段，如果是就修改数据包的源mac和目的mac，发送出去，如果不是同一个网段，路由器将查看路由表，寻找下一跳，根据下一跳寻找当前路由器数据出去的设备，让后将修改源mac为出去设备的mac，目的mac为下一跳的mac，将数据发送到下一跳，这样重复这个动作将数据发送到目的主机。

13、谈谈集线器、交换机、路由器的作用

集线器：Hub 发送到集线器上的数据 会被集线器广播到连接改集线器的所有主机上共享带宽。 整形放大 工作在物理层

交换机：switchs 单播 独享带宽 有自主学习功能（mac）拓展网络接口工作在链路层（二层交换机）3层交换机（核心层在第二层 具备网段划分的功能 VLAN虚拟局域网）4层交换机（核心层在第二层 具备端口映射功能）

路由器：不同网段 通信的桥梁 实现不同网段的数据传输 工作在网络层。

四、c++技术点

1、谈谈你对命名空间的理解

namespace 是一个关键字：随着工程量的增加，变量命名上不可避免的会出现重名，防止名称冲突（在两个不同的命名空间中，即使2个变量名相同，也是2个不同的变量），在实际工作中，基本都使用标准命名空间。

命名空间只能全局范围内定义，不能定义在函数内部。

命名空间内，可以存放 变量、函数、结构体、类；也可以嵌套其他的命名空。

命名空间可以匿名(但一般不这样使用),类似静态全局变量。

命名空间是可以起别名的。

2、谈谈指针和引用的区别

- 引用是给变量起别名，内部实现是指针常量（int* const ref = &a），其可以简单的理解为本体指针存放的是变量的地址
- 引用的本质是指针常量，其指向不可修改，而指针可以改变指向
- 引用创建的同时必须初始化，指针创建的时候可以不必初始化
- 引用不能为空，指针可以为 NULL
- “引用变量 ref”的内存单元保存的是“被引用变量 a”的地址 sizeof(引用) = 指向变量的大小 sizeof(指针) = 指针本身的大小
- 引用使用的时候无需解引用，指针需要解引用
- 指针和引用“自增/自减运算”意义不一样
- 在同等需求下，函数传参，引用可以将一级指针

3、谈谈你对内联函数的理解

宏函数缺陷 1： 需要将实现“加 括号”，以保证优先级的完整性

宏函数缺陷 2：即使加了括号 有些情况 依然有缺陷

- 内联函数本身是一个真正的函数 // 但，宏函数不是函数
- 内联函数具有普通函数的所有行为唯一不同之处在于：内联函数会在适当的地方像定义宏一样展开，可以以空间换时间因此，内联函数既可以避免宏函数的缺陷，也可以避免普通函数入栈的时间浪费
- 在普通函数函数前面加上 inline 关键字使之成为内联函数
- 如果有函数声明，函数本身和声明必须同时加 inline 关键字，否则视为普通函数
- 任何在“类”内部定义的函数会自动成为内联函数
- 下列情况，普通函数即使指定为内联函数，编译器也可能考虑不按内联编译 1）存在任何形式的循环语句 2）存在过多的条件判断语句 3）函数体过于庞大 4）对函数进行取址操作
- 使用方式建议： 1）内联仅仅只是给编译器一个建议，编译器不一定会接受这种建议 2）如果你没有将函数声明为内联函数，那么编译器也可能将此函数做内联编译（一个好的编译器将会内联小的、简单的函数）因此，不用刻意使用内联函数，可以交给编译器去自行处理

4、谈谈函数的重载条件

函数重载：在C语言中，函数名必须是唯一的，程序中不允许出现同名的函数 // 在C++中是允许出现同名的函数，即在同一作用域内，具有相同函数名，不同参数列表的一组函数， 称为函数重载

函数重载实现的原理

编译器为了实现函数重载，也是默认为我们做了一些幕后的工作，编译器用不同的参数类型来修饰不同的函数名，比如void func(); 编译器可能会将函数名修饰成*func，当编译器碰到void func(int x),编译器可能将函数名修饰为*func_int,当编译器碰到void func(int x,char c),编译器可能会将函数名修饰为_func_int_char我这里使用”可能”这个字眼是因为编译器如何修饰重载的函数名称并没有一个统一的标准，所以不同的编译器可能会产生不同的内部名

函数重载实现的条件

同一个作用域、参数的个数不同、参数类型不同、参数的顺序不同

5、谈谈c与c++中struct的不同点

c语言中struct只有数据

c++中的struct不止有数据 还有函数

6、如何理解c++的封装性

封装特性包含两个方面，一个是属性和变量合成一个整体，一个是给属性和函数增加访问权限

7、谈谈你对c++构造与析构的理解

对象的初始化和清理也是两个非常重要的安全问题，一个对象或者变量没有初始时，对其使用后果是未知，同样的使用完一个变量，没有及时清理，也会造成一定的安全问题。c++为了给我们提供这种问题的解决方案，构造函数和析构函数，这两个函数将会被编译器自动调用，完成对象初始化和对象清理工作。

无论你是否喜欢，对象的初始化和清理工作是编译器强制我们要做的事情，即使你不提供初始化操作和清理操作，编译器也会给你增加默认的操作，只是这个默认初始化操作不会做任何事，所以编写类就应该顺便提供初始化函数。

构造函数主要作用在于创建对象时为对象的成员属性赋值，构造函数由编译器自动调用，无须手动调用。 主要用于对象销毁前系统自动调用，执行一些清理工作。

8、构造函数的分类

按参数类型：分为无参构造函数和有参构造函数 按类型分类：普通构造函数和拷贝构造函数(复制构造函数)

9、构造函数的调用规则

默认情况下，c++编译器至少为我们写的类增加3个函数 1．默认构造函数(无参，函数体为空) 2．默认析构函数(无参，函数体为空) 3．默认拷贝构造函数，对类中非静态成员属性简单值拷贝 如果用户定义拷贝构造函数，c++不会再提供任何默认构造函数 如果用户定义了普通构造(非拷贝)，c++不在提供默认无参构造，但是会提供默认拷贝构造

10、谈谈你对浅拷贝与深拷贝的区别

浅拷贝

同一类型的对象之间可以赋值，使得两个对象的成员变量的值相同，两个对象仍然是独立的两个对象，这种情况被称为浅拷贝. 一般情况下，浅拷贝没有任何副作用，但是当类中有指针，并且指针指向动态分配的内存空间，析构函数做了动态内存释放的处理，会导致内存问题。

深拷贝

当类中有指针，并且此指针有动态分配空间，析构函数做了释放处理，往往需要自定义拷贝构造函数，自行给指针动态分配空间，深拷贝

11、谈谈啥叫对象成员以及对象成员的构造函数调用方式

在类中定义的数据成员一般都是基本的数据类型。但是类中的成员也可以是对象，叫做对象成员。

C++中对对象的初始化是非常重要的操作，当创建一个对象的时候，c++编译器必须确保调用了所有子对象的构造函数。如果所有的子对象有默认构造函数，编译器可以自动调用他们。但是如果子对象没有默认的构造函数，或者想指定调用某个构造函数怎么办?

那么是否可以在类的构造函数直接调用子类的属性完成初始化呢？但是如果子类的成员属性是私有的，我们是没有办法访问并完成初始化的。

解决办法非常简单：对于子类调用构造函数，c++为此提供了专门的语法，即构造函数初始化列表。 当调用构造函数时，首先按各对象成员在类定义中的顺序（和参数列表的顺序无关）依次调用它们的构造函数，对这些对象初始化，最后再调用本身的函数体。也就是说，先调用对象成员的构造函数，再调用本身的构造函数。 析构函数和构造函数调用顺序相反，先构造，后析构。

12、谈谈你对explicit的理解

c++提供了关键字explicit，禁止通过构造函数进行的隐式转换。声明为explicit的构造函数不能在隐式转换中使用

13、谈谈c中malloc free 与 c++中的new delete有啥区别，或则c++在堆区申请对象时为啥推荐使用new delete

c++中的malloc free的问题：

1、程序员必须确定对象的长度

2、malloc返回一个void*指针，c++不允许将void*赋值给其他任何指针，必须强转

3、malloc可能申请内存失败，所以必须判断返回值来确保内存分配成功

4、用户在使用对象之前必须记住对他初始化，构造函数不能显示调用初始化(构造函数是由编译器调用)，用户有可能忘记调用初始化函数

总结：c的动态内存分配函数太复杂，容易令人混淆，是不可接受的，c++中我们推荐使用运算符new 和delete

new操作符能确定在调用构造函数初始化之前内存分配是成功的，所有不用显式确定调用是否成功

delete表达式先调用析构函数，然后释放内存

14、谈谈你对static静态成员变量的理解

在一个类中，若将一个成员变量声明为static，这种成员称为静态成员变量。与一般的数据成员不同，无论建立了多少个对象，都只有一个静态数据的拷贝。静态成员变量，属于某个类，所有对象共享。

静态变量，是在编译阶段就分配空间，对象还没有创建时，就已经分配空间

注意：

1、静态成员变量必须在类中声明，在类外定义。 2、静态数据成员不属于某个对象，在为对象分配空间中不包括静态成员所占空间。 3、静态数据成员可以通过类名或者对象名来引用

15、谈谈你对static静态成员函数的理解

在类定义中，前面有static说明的成员函数称为静态成员函数。静态成员函数使用方式和静态变量一样，同样在对象没有创建前，即可通过类名调用。静态成员函数主要为了访问静态变量，但是，不能访问普通成员变量。

静态成员函数的意义，不在于信息共享，数据沟通，而在于管理静态数据成员，完成对静态数据成员的封装。

1、静态成员函数只能访问静态变量，不能访问普通成员变量 2、静态成员函数的使用和静态成员变量一样 3、静态成员函数也有访问权限 4、普通成员函数可访问静态成员变量、也可以访问非经常成员变量

16、谈谈你对this的理解

成员函数通过this指针即可知道操作的是那个对象的数据。This指针是一种隐含指针，它隐含于每个类的非静态成员函数中。This指针无需定义，直接使用即可。

注意：静态成员函数内部没有this指针，静态成员函数不能操作非静态成员变量

17、谈谈你对友元的理解

类的主要特点之一是数据隐藏，即类的私有成员无法在类的外部(作用域之外)访问。但是，有时候需要在类的外部访问类的私有成员，怎么办？ 解决方法是使用友元函数，友元函数是一种特权函数，c++允许这个特权函数访问私有成员

1、friend关键字只出现在声明处 2、其他类、类成员函数、全局函数都可声明为友元 3、友元函数不是类的成员，不带this指针 4、友元函数可访问对象任意成员属性，包括私有属性

友元的注意事项

1、友元关系不能被继承。 2、友元关系是单向的，类A是类B的朋友，但类B不一定是类A的朋友。

3、友元关系不具有传递性。类B是类A的朋友，类C是类B的朋友，但类C不一定是类A的朋友。

18、谈谈你对继承的理解

c++最重要的特征是代码重用，通过继承机制可以利用已有的数据类型来定义新的数据类型，新的类不仅拥有旧类的成员，还拥有新定义的成员。 一个B类继承于A类，或称从类A派生类B。这样的话，类A成为基类（父类）， 类B成为派生类（子类）。 派生类中的成员，包含两大部分： 1、一类是从基类继承过来的，一类是自己增加的成员。 2、从基类继承过过来的表现其共性，而新增的成员体现了其个性

19、谈谈继承中的构造与析构的顺序

1、子类对象在创建时会首先调用父类的构造函数 2、父类构造函数执行完毕后，才会调用子类的构造函数 3、当父类构造函数有参数时，需要在子类初始化列表(参数列表)中显示调用父类构造函数 析构函数调用顺序和构造函数相反

20、继承中同名成员的处理方法

1、当子类成员和父类成员同名时，子类依然从父类继承同名成员 2、如果子类有成员和父类同名，子类访问其成员默认访问子类的成员(本作用域，就近原则) 3、在子类通过作用域::进行同名成员区分(在派生类中使用基类的同名成员，显示使用类名限定符)

21、哪些函数是无法继承的

是所有的函数都能自动从基类继承到派生类中。构造函数和析构函数用来处理对象的创建和析构操作，构造和析构函数只知道对它们的特定层次的对象做什么，也就是说构造函数和析构函数不能被继承，必须为每一个特定的派生类分别创建。

另外operator=也不能被继承，因为它完成类似构造函数的行为。也就是说尽管我们知道如何用=右边的对象如何初始化=左边的对象的所有成员，但是这个并不意味着对其派生类依然有效。

在继承的过程中，如果没有创建这些函数，编译器会自动生成它们。

22、静态多态与动态多态的区别

静态多态和动态多态的区别就是函数地址是早绑定(静态联编)还是晚绑定(动态联编)。如果函数的调用，在编译阶段就可以确定函数的调用地址，并产生代码，就是静态多态(编译时多态)，就是说地址是早绑定的。而如果函数的调用地址不能编译不能在编译期间确定，而需要在运行时才能决定，这这就属于晚绑定(动态多态,运行时多态)

23、c++的动态捆绑机制是怎样的?

首先，我们看看编译器如何处理虚函数。当编译器发现我们的类中有虚函数的时候，编译器会创建一张虚函数表，把虚函数的函数入口地址放到虚函数表中，并且在类中秘密增加一个指针，这个指针就是vpointer(缩写vptr)，这个指针是指向对象的虚函数表。在多态调用的时候，根据vptr指针，找到虚函数表来实现动态绑定

24、多态成立的条件

1、有继承 2、子类重写父类虚函数函数 a) 返回值，函数名字，函数参数，必须和父类完全一致(析构函数除外) b) 子类中virtual关键字可写可不写，建议写 3、类型兼容，父类指针，父类引用 指向 子类对象

25、虚析构的作用

虚析构函数是为了解决基类的指针指向派生类对象，并用基类的指针删除派生类对象

26、纯虚析构与非纯虚析构的区别

纯虚析构函数在c++中是合法的，但是在使用的时候有一个额外的限制：必须为纯虚析构函数提供一个函数体。

纯虚析构函数和非纯析构函数之间唯一的不同之处在于纯虚析构函数使得基类是抽象类，不能创建基类的对象。

注意：如果类的目的不是为了实现多态，作为基类来使用，就不要声明虚析构函数，反之，则应该为类声明虚析构函数

27、谈谈重载、重写、重定义的概念

1、重载，同一作用域的同名函数 a、同一个作用域 b、参数个数，参数顺序，参数类型不同 c、和函数返回值，没有关系 d、const也可以作为重载条件 //do(const Teacher& t){} do(Teacher& t) 2、重定义（隐藏） a、有继承 b、子类（派生类）重新定义父类（基类）的同名成员（非virtual函数） 3、重写（覆盖） a、有继承 b、子类（派生类）重写父类（基类）的virtual函数 c、函数返回值，函数名字，函数参数，必须和基类中的虚函数一致

28、谈谈你对模板的理解

所谓函数模板，实际上是建立一个通用函数，其函数类型和形参类型不具体制定，用一个虚拟的类型来代表。这个通用函数就成为函数模板。

c++提供两种模板机制：函数模板和类模板

模板把函数或类要处理的数据类型参数化，表现为参数的多态性，成为类属.

模板用于表达逻辑结构相同，但具体数据元素类型不同的数据对象的通用行为

29、函数模板和普通函数的区别

函数模板不允许自动类型转换

普通函数可以自动实现类型转换

30、函数模板和普通函数同时出现时的调用机制

1、c++编译器优先考虑普通函数 2、可以通过空模板实参列表的语法限定编译器只能通过模板匹配 3、函数模板可以像普通函数那样可以被重载 4、如果函数模板可以产生一个更好的匹配，那么选择模板

31、谈谈静态转换(`static_cast`)、动态转换(`dynamic_cast`)、常量转换(`const_cast`)、重新解析转换(`reinterpret_cast`)的区别

静态转换(`static_cast`)

用于类层次结构中基类（父类）和派生类（子类）之间指针或引用的转换。 1、进行上行转换（把派生类的指针或引用转换成基类表示）是安全的； 2、进行下行转换（把基类指针或引用转换成派生类表示）时，由于没有动态类型检查，所以是不安全的。 3、用于基本数据类型之间的转换，如把int转换成char，把char转换成int。这种转换的安全性也要开发人员来保证

动态转换(`dynamic_cast`)

1、dynamic_cast主要用于类层次间的上行转换和下行转换； 2、在类层次间进行上行转换时，dynamic_cast和static_cast的效果是一样的； 3、在进行下行转换时，dynamic_cast具有类型检查的功能，比static_cast更安全

常量转换(`const_cast`)

1、该运算符用来修改类型的const属性。。 2、常量指针被转化成非常量指针，并且仍然指向原来的对象； 3、常量引用被转换成非常量引用，并且仍然指向原来的对象；

注意:不能直接对非指针和非引用的变量使用const_cast操作符去直接移除它的const.

重新解析转换(`reinterpret_cast`)

这是最不安全的一种转换机制，最有可能出问题。 主要用于将一种数据类型从一种类型转换为另一种类型。它可以将一个指针转换成一个整数，也可以将一个整数转换成一个指针

32、谈谈你对异常的理解

异常处理就是处理程序中的错误。所谓错误是指在程序运行的过程中发生的一些异常事件（如：除0溢出，数组下标越界，所要读取的文件不存在,空指针，内存不足等等）

33、谈谈c++的异常机制相比c语言的异常处理的优势

1、函数的返回值可以忽略，但异常不可忽略。如果程序出现异常，但是没有被捕获，程序就会终止，这多少会促使程序员开发出来的程序更健壮一点。而如果使用C语言的error宏或者函数返回值，调用者都有可能忘记检查，从而没有对错误进行处理，结果造成程序莫名其面的终止或出现错误的结果。

2、整型返回值没有任何语义信息。而异常却包含语义信息，有时你从类名就能够体现出来。

3、整型返回值缺乏相关的上下文信息。异常作为一个类，可以拥有自己的成员，这些成员就可以传递足够的信息。

4、异常处理可以在调用跳级。这是一个代码编写时的问题：假设在有多个函数的调用栈中出现了某个错误，使用整型返回码要求你在每一级函数中都要进行处理。而使用异常处理的栈展开机制，只需要在一处进行处理就可以了，不需要每级函数都处理。

五、STL标准模板库

1、谈谈STL的六大组件

容器：各种数据结构，如vector、list、deque、set、map等,用来存放数据，从实现角度来看，STL容器是一种class template。

算法：各种常用的算法，如sort、find、copy、for_each。从实现的角度来看，STL算法是一种function tempalte.

迭代器：扮演了容器与算法之间的胶合剂，共有五种类型，从实现角度来看，迭代器是一种将operator* , operator-> , operator++,operator--等指针相关操作予以重载的class template. 所有STL容器都附带有自己专属的迭代器，只有容器的设计者才知道如何遍历自己的元素。原生指针(native pointer)也是一种迭代器。

仿函数：行为类似函数，可作为算法的某种策略。从实现角度来看，仿函数是一种重载了operator()的class 或者class template

适配器：一种用来修饰容器或者仿函数或迭代器接口的东西。

空间配置：负责空间的配置与管理。从实现角度看，配置器是一个实现了动态空间配置、空间管理、空间释放的class tempalte.

2、谈谈STL六大主键的关系

STL六大组件的交互关系，容器通过空间配置器取得数据存储空间，算法通过迭代器存储容器中的内容，仿函数可以协助算法完成不同的策略的变化，适配器可以修饰仿函数

3、vector容器相比于普通的数组array有啥优点

vector的数据安排以及操作方式，与array非常相似，两者的唯一差别在于空间的运用的灵活性。Array是静态空间，一旦配置了就不能改变，要换大一点或者小一点的空间，可以，一切琐碎得由自己来，首先配置一块新的空间，然后将旧空间的数据搬往新空间，再释放原来的空间。Vector是动态空间，随着元素的加入，它的内部机制会自动扩充空间以容纳新元素。因此vector的运用对于内存的合理利用与运用的灵活性有很大的帮助，我们再也不必害怕空间不足而一开始就要求一个大块头的array了

Vector的实现技术，关键在于其对大小的控制以及重新配置时的数据移动效率，一旦vector旧空间满了，如果客户每新增一个元素，vector内部只是扩充一个元素的空间，实为不智，因为所谓的扩充空间(不论多大)，一如刚所说，是”配置新空间-数据移动-释放旧空间”的大工程,时间成本很高，应该加入某种未雨绸缪的考虑，稍后我们便可以看到vector的空间配置策略

4、deque容器相对于vector容器的区别

Deque容器和vector容器最大的差异

一、在于deque允许使用常数项时间对头端进行元素的插入和删除操作。

二、在于deque没有容量的概念，因为它是动态的以分段连续空间组合而成，随时可以增加一段新的空间并链接起来，换句话说，像vector那样，”旧空间不足而重新配置一块更大空间，然后复制元素，再释放旧空间”这样的事情在deque身上是不会发生的。也因此，deque没有必须要提供所谓的空间保留(reserve)功能

5、谈谈stack容器的概念

Stack所有元素的进出都必须符合”先进后出”的条件，只有stack顶端的元素，才有机会被外界取用。Stack不提供遍历功能，也不提供迭代器

6、谈谈queue容器的概念

Queue是一种先进先出(First In First Out,FIFO)的数据结构，它有两个出口，queue容器允许从一端新增元素，从另一端移除元素

Queue所有元素的进出都必须符合”先进先出”的条件，只有queue的顶端元素，才有机会被外界取用。Queue不提供遍历功能，也不提供迭代器

7、谈谈list容器的概念

相较于vector的连续线性空间，list就显得负责许多，它的好处是每次插入或者删除一个元素，就是配置或者释放一个元素的空间。因此，list对于空间的运用有绝对的精准，一点也不浪费。而且，对于任何位置的元素插入或元素的移除，list永远是常数时间

1、采用动态存储分配，不会造成内存浪费和溢出 2、链表执行插入和删除操作十分方便，修改指针即可，不需要移动大量元素 3、链表灵活，但是空间和时间额外耗费较大

8、谈谈set/multiset容器的概念

Set的特性是。所有元素都会根据元素的键值自动被排序。Set的元素不像map那样可以同时拥有实值和键值，set的元素即是键值又是实值。Set不允许两个元素有相同的键值

multiset特性及用法和set完全相同，唯一的差别在于它允许键值重复。set和multiset的底层实现是红黑树，红黑树为平衡二叉树的一种

9、谈谈map/multimap容器的概念

Map的特性是，所有元素都会根据元素的键值自动排序。Map所有的元素都是pair,同时拥有实值和键值，pair的第一元素被视为键值，第二元素被视为实值，map不允许两个元素有相同的键值

Multimap和map的操作类似，唯一区别multimap键值可重复。

Map和multimap都是以红黑树为底层实现机制

10、谈谈什么是函数对象

重载函数调用操作符的类，其对象常称为函数对象（function object），即它们是行为类似函数的对象，也叫仿函数(functor),其实就是重载“()”操作符，使得类对象可以像函数那样调用

注意：

1、函数对象(仿函数)是一个类，不是一个函数。 2、函数对象(仿函数)重载了”() ”操作符使得它可以像函数一样调用

总结：

1、函数对象通常不定义构造函数和析构函数，所以在构造和析构时不会发生任何问题，避免了函数调用的运行时问题。 2、函数对象超出普通函数的概念，函数对象可以有自己的状态 3、函数对象可内联编译，性能好。用函数指针几乎不可能 4、模版函数对象使函数对象具有通用性，这也是它的优势之一

11、谈谈什么是谓词

谓词是指普通函数或重载的operator()返回值是bool类型的函数对象(仿函数)。如果operator接受一个参数，那么叫做一元谓词,如果接受两个参数，那么叫做二元谓词，谓词可作为一个判断式

12、常用的遍历算法有哪些

```
//遍历容器元素
for_each(iterator beg, iterator end, _callback);
//将指定容器区间元素搬运到另一容器中
transform(iterator beg1, iterator end1, iterator beg2, _callbakc);
```

13、常用的查找算法有哪些

```
//查找元素
find(iterator beg, iterator end, value);
//条件查找
find_if(iterator beg, iterator end, _callback);
//查找相邻重复元素
adjacent_find(iterator beg, iterator end, _callback);
//二分查找法
bool binary_search(iterator beg, iterator end, value);
//统计元素出现的次数
count(iterator beg, iterator end, value);
```

14、常用的排序算法有哪些

```
//容器元素合并
merge(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator
dest);
//容器元素排序
sort(iterator beg, iterator end, _callback);
//对指定范围内的元素随机调整次序
random_shuffle(iterator beg, iterator end);
//反转指定范围的元素
reverse(iterator beg, iterator end);
```

15、常用拷贝和替换算法

```
//copy算法 将容器内指定范围的元素拷贝到另一容器中
copy(iterator beg, iterator end, iterator dest)

//replace算法 将容器内指定范围的旧元素修改为新元素
replace(iterator beg, iterator end, oldvalue, newvalue)

//replace_if算法 将容器内指定范围满足条件的元素替换为新元素
replace_if(iterator beg, iterator end, _callback, newvalue)

//swap算法 互换两个容器的元素
swap(container c1, container c2)
```

16、常用算法生成算法

```
//accumulate算法 计算容器元素累计总和
accumulate(iterator beg, iterator end, value)

//fill算法 向容器中添加元素
fill(iterator beg, iterator end, value)
```

17、常用集合算法

```
//set_intersection算法 求两个set集合的交集
set_intersection(iterator beg1, iterator end1, iterator beg2, iterator end2,
iterator dest)

//set_union算法 求两个set集合的并集
set_union(iterator beg1, iterator end1, iterator beg2, iterator end2, iterator
dest);

//set_difference算法 求两个set集合的差集
set_difference(iterator beg1, iterator end1, iterator beg2, iterator end2,
iterator dest)
```

六、Qt

1、谈谈信号和槽机制

信号槽是 Qt 框架引以为豪的机制之一。所谓信号槽，实际就是观察者模式。当某个事件发生之后，比如，按钮检测到自己被点击了一下，它就会发出一个信号（signal）。这种发出是没有目的的，类似广播。如果有对象对这个信号感兴趣，它就会使用连接（connect）函数，意思是，将想要处理的信号和自己的一个函数（称为槽（slot））绑定来处理这个信号。也就是说，当信号发出时，被连接的槽函数会自动被回调。这就类似观察者模式：当发生了感兴趣的事件，某一个操作就会被自动触发

七、数据库

1、谈谈你对数据库中事务的理解

事务(Transaction)可以使用 BEGIN TRANSACTION 命令或简单的 BEGIN 命令来启动。此类事务通常会持续执行下 去，直到遇到下一个 COMMIT 或 ROLLBACK 命令。不过在数据库关闭或发生错误时，事务处理也会回滚.

2、谈谈你对数据库中联结表的理解

保存数据时往往不会将所有数据保存在一个表中，而是在多个表中存储，联结表就是从多个表中查询数据。

在一个表中不利于分解数据，也容易使相同数据出现多次，浪费存储空间;使用联结表查看各个数据更直观， 这使得在处理数据时更简单 `

3、谈谈你对数据库中触发器的理解

SQLite 的触发器是数据库的回调函数，它会在指定的数据库事件发生时自动执行调用

4、谈谈你对数据库中索引的理解

索引是对数据库表中一列或多列的值进行排序的一种结构，使用索引可快速访问数据库表中的特定信息。 索引就是恰当的排序，经过某种算法优化，使查找次数要少的多的多

缺点: 1、索引数据可能要占用大量的存储空间，因此并非所有数据都适合索引 2、索引改善检索操作的性能，但降低了数据插入、修改和删除的性能

5、谈谈你对数据库中主键、唯一约束、检查约束的理解

主键: 惟一的标识一行(一张表中只能有一个主键) 主键应当是对用户没有意义的(常用于索引) 永远不要更新主键，否则违反对用户没有意义原则 主键不应包含动态变化的数据，如时间戳、创建时间列、修改时间列等 主键应当有计算机自动生成(保证唯一性)

唯一约束: 用来保证一个列(或一组列)中数据唯一，类似于主键，但跟主键有区别 表可包含多个唯一约束，但只允许一个主键 唯一约束列可修改或更新

检查约束: 用来保证一个列(或一组列)中的数据满足一组指定的条件。 指定范围，检查最大或最小范围，通过 check 实现

八、BS开发

1、什么是B/S架构

Browser/Server(浏览器/服务器结构)，是随着 Internet 技术的兴起，是对 C/S 结构的一种变化或者改进的结 构。 随着 Windows98/Windows2000 将浏览器技术植入操作系统内部，这种结构更成为当今应用软件的首选体系结 构

2、B/S 架构 与 C/S 架构对比

C/S

专用网络 面向相对固定的用户群、信息安全的控制能力很强 更加注重流程、系统运行速度可较少考 虑升级难 处理问题集中 与操作系统关系密切 交互性低

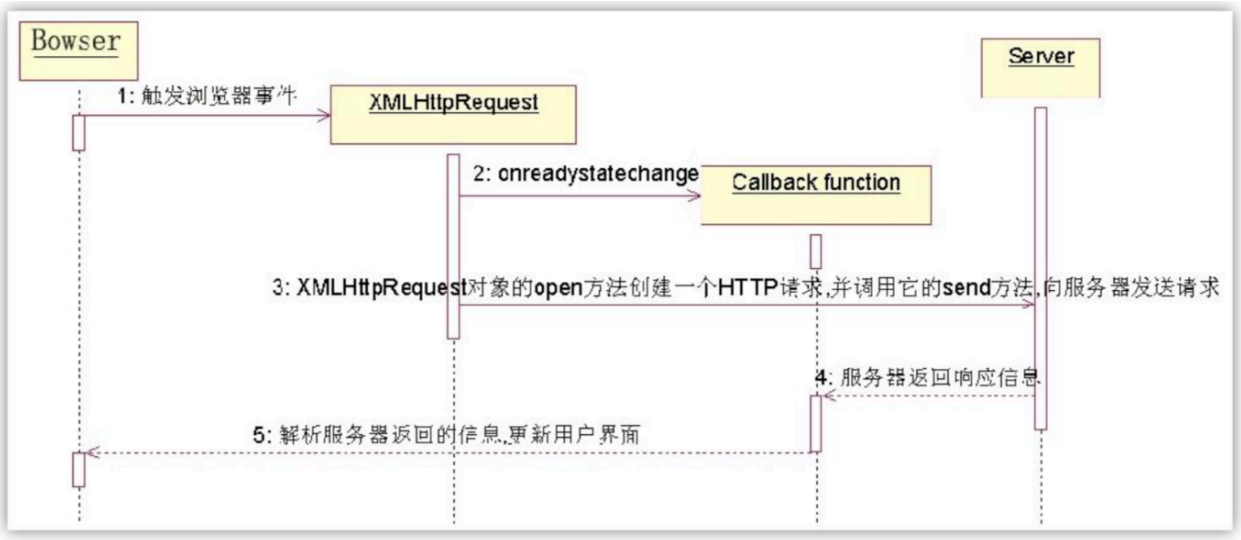
B/S

广域网 面向是不可知的用户群、对安全的控制能力相对弱 对安全以及访问速度要多重的考虑 B/S 结构的程序架构是发展的趋势 开销小、方便升级 处理问题分散 跨平台，与浏览器相关 交互密集

3、什么是Javascript

Javascript 是一种基于对象并具有安全性能的脚本语言，是由浏览器内解释器翻译成可执行格式后执行，在概 念和设计方面，Java 和 Javascript 是两种完全不同的语言。 Javascript 的四个特点:基于对象的语言、简单性、动态性、跨平台性

4、描述一下ajax工作流程



1、创建对象 2、设置回调函数 ， fun 函数 3、open 创建服务器请求 4、send 向服务器发送请求， 5、服务器有结果会自动调用 fun 回调函数 在回调函数里面根据服务器返回的响应信息 更新用户界面

九、物联网

1、RFID了解多少?

用过13.56M频段的，对国产的FM17550和恩智普的RC522都比较熟悉，能够实现对A类卡的读写、充值、扣款等操作以及密码的认证和修改，其它频段工作原理相似，主要参照读写器数据手册和《中国金融集成电路(IC)卡规范》对设备进行操作。

2、请问s50卡的默认密码是多少?

s50卡密码分为A和B两种，都是6个字节，默认都是0xFF,0xFF,0xFF,0xFF,0xFF,0xFF

3、请问zigbee是用的什么平台?

我们用的是TI公司出的CC2530平台，功耗和成本都比较低。

4、你们使用zigbee模块是基于AT指令模式，还是协议栈模式?

我们是基于TI提供的协议栈接口直接开发的。

5、zigbee的物理信道有几个?

zigbee可以工作在3个频段，总共27个物理信道上，分别是868.3M的1个信道，和906M的10个信道，还有2.4G的16个信道。

6、zigbee协议栈规定了模块的几种通信角色?

三种：全功能节点的协调器，全功能节点的路由器，半功能节点的终端。

7、MQTT是什么?

MQTT是IBM推出的一种针对移动终端设备的基于TCP/IP的发布/订阅协议，实现了对通信双方在时间、空间、同步这三种场合的解耦。

8、你们用的是MQTT那个版本?

我们目前用的是mosquitto开源软件，支持MQTTv3.1版本。

9、消息订阅发布可以使用通配符，请问具体怎么使用有什么需要注意的?

+代表单层过滤，#代表末尾多层过滤，但通配符只适用于订阅主题，而不适用于发布主题

10、NB-IOT是什么?

NB-IoT是指Narrow Band Internet of Things，聚焦于低功耗窄带广域物联网，工作在License频谱，能与现有移动通信网共存。

11、NB-IOT主要解决了什么问题?

主要解决了广域网通信，高资费、大功耗等问题。

12、你用的是什么NB-IOT模块，都支持哪些协议?

我们用过上海移远BC系列模块，还用过中移物联的m5310a模块，基本上都是全网通，并支持协议有：IPv4、IPV6、UDP、TCP、CoAP、LwM2M、Non-IP、DTLS、MQTTT等。

十、项目篇

1、描述一下你熟悉的一个项目