复习资料

c/c++

面向对象的基本思想

封装、继承、多态

封装：面向对象程序设计最基本的特性，把数据（属性）和函数（操作）合成一个整体，这在计算机世界中是用类与对象实现的。

继承：是面向对象程序设计使代码可以复用的最重要的手段，它允许程序员在保持原有类特性的基础上进行扩展，增加功能。这样产生新的类，称派生类。继承呈现了面向对象程序设计的层次结构.体现了由简单到复杂的认识过程。

派生反映了事物之间的联系，事物的共性与个性之间的关系。派生与独立设计若干相关的类，前者工作量少，重复的部分可以从基类继承来，不需要单独编程。

(1) **公有继承(public)**
公有继承的特点是基类的公有成员和保护成员作为派生类的成员时，它们都保持原有的状态，而基类的私有成员仍然是私有的，不能被这个派生类的子类所访问。

(2) **私有继承(private)** ----------默认的继承方式（如果缺省，默认为private继承)

私有继承的特点是基类的公有成员和保护成员都作为派生类的私有成员，并且不能被这个派生类的子类所访问。

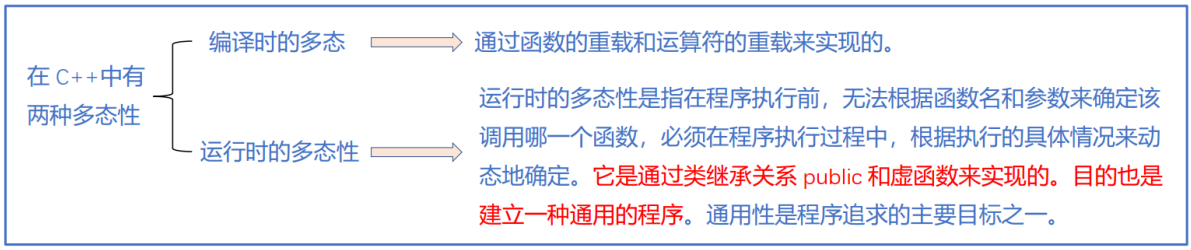
子类也不能转换成相应的基类，如果转换，会报错：“不允许对不可访问的基类进行转换”。

(3) **保护继承(protected)**
保护继承的特点是基类的所有公有成员和保护成员都成为派生类的保护成员，并且只能被它的派生类成员函数或友元访问，基类的私有成员仍然是私有的。

下面列出三种不同的继承方式的基类特性和派生类特性。

	public	**protected**	**private**
公有继承	public	protected	不可见
私有继承	private	private	不可见
保护继承	protected	protected	不可见

多态：分为编译时多态和运行时多态



虚函数，虚表

虚函数的作用主要是为了实现多态机制。多态，简单来说，是指在继承层次中，父类的指针可以具有多种形态——当它指向某个子类对象时，通过它能够调用到子类的函数，而非父类的函数。

虚函数是一个类的成员函数，定义格式如下:

virtual 返回类型 函数名（参数列表);

关键字virtual指明该成员函数为虚函数。virtual仅用于类定义中，如虚函数在类外定义，不可加virtual。

所有的类会维护一个虚表，当它被继承时，虚表也会被继承，如果派生类修改了父类的虚函数，则会在虚表中替换虚函数地址为修改后的函数地址

当某一个类的一个类成员函数被定义为虚函数，则由该类派生出来的所有派生类中，该函数始终保持虚函数的特征。

成员函数应尽可能地设置为虚函数，但必须注意以下几条:

> 1．派生类中定义虚函数必须与基类中的虚函数同名外，还必须同参数表，同返回类型。否则被认为是重载，而不是虚函数。如基类中返回基类指针，派生类中返回派生类指针是允许的，这是一个例外。
>
> 2．只有类的成员函数才能说明为虚函数。这是因为虚函数仅适用于有继承关系的类对象。
>
> 3．静态成员函数，是所有同一类对象共有，不受限于某个对象，不能作为虚函数。
>
> 4．实现动态多态性时，必须使用基类类型的指针变量或引用，使该指针指向该基类的不同派生类的对象，并通过该指针指向虚函数，才能实现动态的多态性。
>
> 5.内联函数每个对象一个拷贝，无映射关系，不能作为虚函数。
>
> 6.析构函数可定义为虚函数，构造函数不能定义虚函数，因为在调用构造函数时对象还没有完成实例化。在基类中及其派生类中都动态分配的内存空间时，必须把析构函数定义为虚函数，实现撤消对象时的多态性。
>
> 7．函数执行速度要稍慢一些。为了实现多态性，每一个派生类中均要保存相应虚函数的入口地址表，函数的调用机制也是间接实现。所以多态性总是要付出一定代价，但通用性是一个更高的目标。
>
> 8．如果定义放在类外，virtual 只能加在函数声明前面，不能(再）加在函数定义前面。正确的定义必须不包括virtual

指针和引用的区别

> **引用**就是某一变量的一个别名，对引用的操作与对变量直接操作完全一样。 **引用的本质是指针，而且是常量指针，占用4个字节的空间**

> **指针**利用地址，它的值直接指向存在电脑存储器中另一个地方的值。由于通过地址能找到所需的变量单元，可以说，地址指向该变量单元。

> 1.指针有自己的一块空间，而引用是一个别名
>
> 2.一个指针的大小是4或者8，而引用的大小是对象的大小
>
> 3.指针可以被初始化为nullptr但是引用必须被初始化为一个已有对象的引用
>
> 4.可以有const指针但是没有const引用
>
> 5.指针通过解引用才能对对象操作，对引用的改变会直接的改变引用所指的对象
>
> 6.指针可以有多级引用，引用只有一级
>
> 7.指针和引用使用++的含义不一样
>
> 8.返回动态内存的对象必须使用指针，使用引用可能对导致内存泄漏
>
> 9.指针只能指向一个对象，但是引用只能是一个对象的引用

思考(可以问面试官)：引用的本质是指针，那么引用的大小到底的指针的大小还是引用的对象的大小，visual studio 2019上测出的是引用的对象的大小，其他编译器没测。

c++中的类型转换

const_cast(a)常被用于除去a变量或者表达式的const属性，a的类型就是T。**强制转换类型必须是指针、引用或者指向对象成员的指针**

static_cast：更安全的类型强转，可以在内置的数据类型之间互相转化，对于类只能在有联系的指针类型间进行转换。**不进行类型检查来确保转换的安全性**

reinterpret_cast 类似于C的强转，任何类型都可以相互转换，不安全的转换

dynamic_cast 通常用于基类和派生类之间的转换，在执行转换时会检查能否转换，如果成功则转换之，如果失败，返回一个nullptr。

动态内存管理

内存管理的基本要求是“**不重不漏**”，不重复的delete也不漏掉delete。重载new是有争议的。

3G用户空间和1G内核空间

静态区域：

text segment(代码段):包括只读存储区和文本区，其中只读存储区存储字符串常量，文本区存储程序的机器代码。

data segment(数据段)：存储程序中已初始化的全局变量和静态变量

bss segment：存储未初始化的全局变量和静态变量（局部+全局），以及所有被初始化为0的全局变量和静态变量，对于未初始化的全局变量和静态变量，程序运行main之前时会统一清零。即未初始化的全局变量编译器会初始化为0

动态区域：

heap（堆）： 当进程未调用malloc时是没有堆段的，只有调用malloc时采用分配一个堆，并且在程序运行过程中可以动态增加堆大小(移动break指针)，从低地址向高地址增长。分配小内存时使用该区域。堆的起始地址由mm_struct 结构体中的start_brk标识，结束地址由brk标识。

memory mapping segment(映射区):存储动态链接库等文件映射、申请大内存（malloc时调用mmap函数）

stack（栈）：使用栈空间存储函数的返回地址、参数、局部变量、返回值，从高地址向低地址增长。在创建进程时会有一个最大栈大小，Linux可以通过ulimit命令指定。

new/delete和malloc/free的区别

new/delete是c++中的关键字，在申请空间的同时会调用构造函数进行初始化，同样delete也会调用析构函数。

而malloc/free是c中的一个函数，必须指明申请空间的大小，不会对空间进行构造和析构。

static关键字

可以修饰全局变量，局部变量，函数，类的成员，类的成员函数。

静态全局变量：若未被初始化则自动初始化为0，除声明之外的其他文件不可见，作用域从定义开始，到程序结束。

静态局部变量：若未被初始化则自动初始化为0，作用域在定义它的函数，函数结束时，作用域不会消失，只会不能访问，当再次访问该函数时被访问，值不变。

静态函数：若函数被声明为静态函数，则除声明之外的其他文件不可见。

类的静态成员：静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。对多个对象来说，静态数据成员只存储一处，供所有对象共用。

类的静态函数：静态成员函数和静态数据成员一样，它们都属于类的静态成员，它们都不是对象成员。因此，对静态成员的引用不需要用对象名。

const关键字

const可以修饰变量，函数，函数参数，函数返回值。

const修饰变量：变量的值不可改变

const修饰函数：类中将成员函数修饰为const表明在该函数体内， **不能修改对象的数据成员而且不能调用非const函数。**

const修饰参数：防止传入的参数代表的内容在函数体内被改变，但仅对指针和引用有意义。因为如果是按值传递，传给参数的仅仅是实参的副本，即使在函数体内改变了形参，实参也不会得到影响。

const修饰函数返回值：也是用const来修饰返回的指针或引用，保护指针指向的内容或引用的内容不被修改，也常用于运算符重载。

智能指针

智能指针的作用是管理一个指针，因为存在以下这种情况：申请的空间在函数结束时忘记释放，造成内存泄漏。使用智能指针可以很大程度上的避免这个问题，因为智能指针就是一个类，当超出了类的作用域是，类会自动调用析构函数，析构函数会自动释放资源。所以智能指针的作用原理就是在函数结束时自动释放内存空间，不需要手动释放内存空间。

auto_ptr	**c++98的方案，c++11已经废弃，没有拥有权的概念，不知道delete的时机**
unique_ptr	独占式拥有或严格拥有概念，保证同一时间内只有一个智能指针可以指向该对象。
shared_ptr	共享式拥有概念。多个智能指针可以指向相同对象，该对象和其相关资源会在“最后一个引用被销毁”时候释放。
weak_ptr	配合shared_ptr使用的一个指针，不增加引用计数的值，防止共享型智能指针相互引用，无法析构。

shared_ptr如何做到共享的?

类的成员内部有一个引用计数，这个引用计数是一个指针。如果有多个指针指向同一对象，引用计数的值为指针的数量，引用计数为零时调用析构函数，销毁对象。

析构函数为什么是虚函数

如果析构函数不是虚函数的话，那么当用基类指针操作派生类的对象的话，析构的时候会调用基类的析构函数，从而导致派生类无法被正确析构。用虚函数则可以避免这种情况，无论指针类型是什么，总能够找到适合对象类型的析构函数。

class和struct的区别

class的成员属性默认是private(私有)

struct的成员属性默认是public(共有)

操作系统

进程和线程

进程是对运行时程序的封装，是系统进行资源调度和分配的的基本单位，实现了操作系统的并发；

线程是进程的子任务，是CPU调度和分派的基本单位，用于保证程序的实时性，实现进程内部的并发；线程是操作系统可识别的最小执行和调度单位。

区别:

> 1.一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。线程依赖于进程而存在。

> 2.进程在执行过程中拥有独立的内存单元，而多个线程共享进程的内存。（资源分配给进程，同一进程的所有线程共享该进程的所有资源。同一进程中的多个线程共享代码段（代码和常量），数据段（全局变量和静态变量），扩展段（堆存储）。但是每个线程拥有自己的栈段，栈段又叫运行时段，用来存放所有局部变量和临时变量。
>
> 3.进程是资源分配的最小单位，线程是CPU调度的最小单位；
>
> 4.系统开销： 由于在创建或撤消进程时，系统都要为之分配或回收资源，如内存空间、I / o设备等。因此，操作系统所付出的开销将显著地大于在创建或撤消线程时的开销。类似地，在进行进程切换时，涉及到整个当前进程CPU环境的保存以及新被调度运行的进程的CPU环境的设置。而线程切换只须保存和设置少量寄存器的内容，并不涉及存储器管理方面的操作。可见，进程切换的开销也远大于线程切换的开销。
>
> 5.通信：由于同一进程中的多个线程具有相同的地址空间，致使它们之间的同步和通信的实现，也变得比较容易。进程间通信IPC，线程间可以直接读写进程数据段（如全局变量）来进行通信——需要进程同步和互斥手段的辅助，以保证数据的一致性。在有的系统中，线程的切换、同步和通信都无须操作系统内核的干预
>
> 6.进程编程调试简单可靠性高，但是创建销毁开销大；线程正相反，开销小，切换速度快，但是编程调试相对复杂。
>
> 7.进程间不会相互影响 ； 线程一个线程挂掉将导致整个进程挂掉
>
> 8.进程适应于多核、多机分布；线程适用于多核

进程/线程间通讯方式

进程间通讯方式：**管道、消息队列、信号、信号量、共享内存、套接字socket**

共享内存是最快的ipc，是进程对内存直接存取

线程间通讯方式：**临界区、互斥量、信号量、事件**

临界区：通过多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问；

互斥量Synchronized/Lock：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问

信号量Semphare：为控制具有有限数量的用户资源而设计的，它允许多个线程在同一时刻去访问同一个资源，但一般需要限制同一时刻访问此资源的最大线程数目。

事件(信号)，Wait/Notify：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作

Linux的虚拟地址空间

为了防止不同进程同一时刻在物理内存中运行而对物理内存的争夺和践踏，采用了虚拟内存。

虚拟内存技术使得不同进程在运行过程中，它所看到的是自己独自占有了当前系统的4G内存。所有进程共享同一物理内存，每个进程只把自己目前需要的虚拟内存空间映射并存储到物理内存上。 事实上，在每个进程创建加载时，内核只是为进程“创建”了虚拟内存的布局，具体就是初始化进程控制表中内存相关的链表，实际上并不立即就把虚拟内存对应位置的程序数据和代码（比如.text .data段）拷贝到物理内存中，只是建立好虚拟内存和磁盘文件之间的映射就好（叫做存储器映射），等到运行到对应的程序时，才会通过缺页异常，来拷贝数据。还有进程运行过程中，要动态分配内存，比如malloc时，也只是分配了虚拟内存，即为这块虚拟内存对应的页表项做相应设置，当进程真正访问到此数据时，才引发缺页异常。

请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。

虚拟内存的好处：

> 1.扩大地址空间；
>
> 2.内存保护：每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方。虚存还对特定的内存地址提供写保护，可以防止代码或数据被恶意篡改。
>
> 3.公平内存分配。采用了虚存之后，每个进程都相当于有同样大小的虚存空间。
>
> 4.当进程通信时，可采用虚存共享的方式实现。
>
> 5.当不同的进程使用同样的代码时，比如库文件中的代码，物理内存中可以只存储一份这样的代码，不同的进程只需要把自己的虚拟内存映射过去就可以了，节省内存
>
> 6.虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。在内存中可以保留多个进程，系统并发度提高
>
> 7.在程序需要分配连续的内存空间的时候，只需要在虚拟内存空间分配连续空间，而不需要实际物理内存的连续空间，可以利用碎片

虚拟内存的代价：

1.虚存的管理需要建立很多数据结构，这些数据结构要占用额外的内存

2.虚拟地址到物理地址的转换，增加了指令的执行时间。

3.页面的换入换出需要磁盘I/O，这是很耗时的

4.如果一页中只有一部分数据，会浪费内存。

谈谈操作系统中的缺页中断

malloc()和mmap()等内存分配函数，在分配时只是建立了进程虚拟地址空间，并没有分配虚拟内存对应的物理内存。当进程访问这些没有建立映射关系的虚拟内存时，处理器自动触发一个缺页异常。

缺页中断：在请求分页系统中，可以通过查询页表中的状态位来确定所要访问的页面是否存在于内存中。每当所要访问的页面不在内存是，会产生一次缺页中断，此时操作系统会根据页表中的外存地址在外存中找到所缺的一页，将其调入内存。

缺页本身是一种中断，与一般的中断一样，需要经过4个处理步骤：

1、保护CPU现场

2、分析中断原因

3、转入缺页中断处理程序进行处理

4、恢复CPU现场，继续执行

但是缺页中断是由于所要访问的页面不存在于内存时，由硬件所产生的一种特殊的中断，因此，与一般的中断存在区别：

1、在指令执行期间产生和处理缺页中断信号

2、一条指令在执行期间，可能产生多次缺页中断

3、缺页中断返回是，执行产生中断的一条指令，而一般的中断返回是，执行下一条指令。

fork与写时拷贝

#include <sys/types.h>

#include <unistd.h>

pid_t fork(void);

成功调用fork( )会创建一个新的进程，它几乎与调用fork( )的进程一模一样，这两个进程都会继续运行。**在子进程中，成功的fork( )调用会返回0。在父进程中fork( )返回子进程的pid。如果出现错误，fork( )返回一个负值。**

最常见的fork( )用法是创建一个新的进程，然后使用exec( )载入二进制映像，替换当前进程的映像。这种情况下，派生（fork）了新的进程，而这个子进程会执行一个新的二进制可执行文件的映像。这种"派生加执行"的方式是很常见的。

写时拷贝：Linux采用了写时复制的方法，以减少fork时对父进程空间进程整体复制带来的开销。

写时复制的主要好处在于：如果进程从来就不需要修改资源，则不需要进行复制。惰性算法的好处就在于它们尽量推迟代价高昂的操作，直到必要的时刻才会去执行。

请你说一说死锁发生的条件以及如何解决死锁

死锁是指两个或两个以上进程在执行过程中，因争夺资源而造成的下相互等待的现象。死锁发生的四个必要条件如下：

互斥条件：进程对所分配到的资源不允许其他进程访问，若其他进程访问该资源，只能等待，直至占有该资源的进程使用完成后释放该资源；

请求和保持条件：进程获得一定的资源后，又对其他资源发出请求，但是该资源可能被其他进程占有，此时请求阻塞，但该进程不会释放自己已经占有的资源

不可剥夺条件：进程已获得的资源，在未完成使用之前，不可被剥夺，只能在使用后自己释放

环路等待条件：进程发生死锁后，必然存在一个进程-资源之间的环形链

解决死锁的方法即破坏上述四个条件之一，主要方法如下：

资源一次性分配，从而剥夺请求和保持条件

可剥夺资源：即当进程新的资源未得到满足时，释放已占有的资源，从而破坏不可剥夺的条件

资源有序分配法：系统给每类资源赋予一个序号，每个进程按编号递增的请求资源，释放则相反，从而破坏环路等待的条件

请你讲述一下互斥锁（mutex）机制，以及互斥锁和读写锁的区别

1、互斥锁和读写锁区别：

互斥锁：mutex，用于保证在任何时刻，都只能有一个线程访问该对象。当获取锁操作失败时，线程会进入睡眠，等待锁释放时被唤醒。

读写锁：rwlock，分为读锁和写锁。处于读操作时，可以允许多个线程同时获得读操作。但是同一时刻只能有一个线程可以获得写锁。其它获取写锁失败的线程都会进入睡眠状态，直到写锁释放时被唤醒。注意：写锁会阻塞其它读写锁。当有一个线程获得写锁在写时，读锁也不能被其它线程获取；写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。适用于读取数据的频率远远大于写数据的频率的场合。

2、Linux的4种锁机制：

互斥锁：mutex，用于保证在任何时刻，都只能有一个线程访问该对象。当获取锁操作失败时，线程会进入睡眠，等待锁释放时被唤醒

读写锁：rwlock，分为读锁和写锁。处于读操作时，可以允许多个线程同时获得读操作。但是同一时刻只能有一个线程可以获得写锁。其它获取写锁失败的线程都会进入睡眠状态，直到写锁释放时被唤醒。注意：写锁会阻塞其它读写锁。当有一个线程获得写锁在写时，读锁也不能被其它线程获取；写者优先于读者（一旦有写者，则后续读者必须等待，唤醒时优先考虑写者）。适用于读取数据的频率远远大于写数据的频率的场合。

自旋锁：spinlock，在任何时刻同样只能有一个线程访问对象。但是当获取锁操作失败时，不会进入睡眠，而是会在原地自旋，直到锁被释放。这样节省了线程从睡眠状态到被唤醒期间的消耗，在加锁时间短暂的环境下会极大的提高效率。但如果加锁时间过长，则会非常浪费CPU资源。

RCU：即read-copy-update，在修改数据时，首先需要读取数据，然后生成一个副本，对副本进行修改。修改完成后，再将老数据update成新的数据。使用RCU时，读者几乎不需要同步开销，既不需要获得锁，也不使用原子指令，不会导致锁竞争，因此就不用考虑死锁问题了。而对于写者的同步开销较大，它需要复制被修改的数据，还必须使用锁机制同步并行其它写者的修改操作。在有大量读操作，少量写操作的情况下效率非常高。

大小端的概念和如何区别

大端是指低字节存储在高地址；小端存储是指低字节存储在低地址。

我们可以根据联合体来判断该系统是大端还是小端。因为联合体变量总是从低地址存储。

```c
union test
{
    int i;
    char c;
};
int main()
{
    test t;
    t.i = 1;

    //如果是大端，则t.c为0x00!=1,返回0；如果是小端，t.c为0x01==1，返回1;
    return t.c==1;
}
```

僵尸进程和孤儿进程

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵尸进程。

僵尸进程危害：

如果进程不调用wait的话， 那么保留的那段信息就不会释放，其进程号就会一直被占用，但是系统所能使用的进程号是有限的，如果大量的产生僵死进程，将因为没有可用的进程号而导致系统不能产生新的进程。

外部消灭：

通过kill发送SIGTERM或者SIGKILL信号消灭产生僵尸进程的进程，它产生的僵死进程就变成了孤儿进程，这些孤儿进程会被init进程接管，init进程会wait()这些孤儿进程，释放它们占用的系统进程表中的资源

内部解决：

1、子进程退出时向父进程发送SIGCHILD信号，父进程处理SIGCHILD信号。在信号处理函数中调用wait进行处理僵尸进程。

2、fork两次，原理是将子进程成为孤儿进程，从而其的父进程变为init进程，通过init进程可以处理僵尸进程。

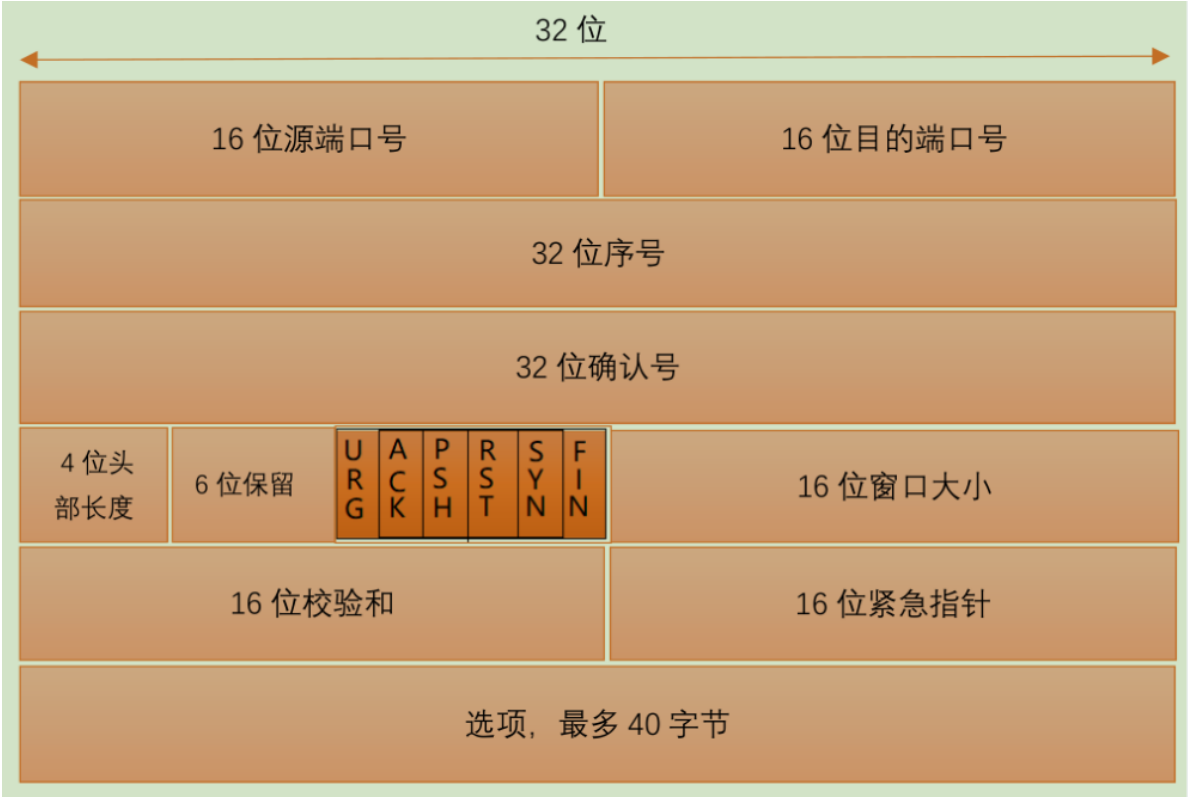
请你来介绍一下5种IO模型

1.**阻塞IO:**调用者调用了某个函数，等待这个函数返回，期间什么也不做，不停的去检查这个函数有没有返回，必须等这个函数返回才能进行下一步动作
2.**非阻塞IO:**非阻塞等待，每隔一段时间就去检测IO事件是否就绪。没有就绪就可以做其他事。
3.**信号驱动IO:**信号驱动IO:linux用套接口进行信号驱动IO，安装一个信号处理函数，进程继续运行并不阻塞，当IO时间就绪，进程收到SIGIO信号。然后处理IO事件。
4.**IO复用/多路转接IO:**linux用select/poll函数实现IO复用模型，这两个函数也会使进程阻塞，但是和阻塞IO所不同的是这两个函数可以同时阻塞多个IO操作。而且可以同时对多个读操作、写操作的IO函数进

行检测。知道有数据可读或可写时，才真正调用IO操作函数
5.**异步IO:**linux中，可以调用aio_read函数告诉内核描述字缓冲区指针和缓冲区的大小、文件偏移及通知的方式，然后立即返回，当内核将数据拷贝到缓冲区后，再通知应用程序。

网络

TCP报头



UDP报头

16位的源端的端口号	16位的目的端端口号
16位的UDP的报文段长度	16位的冗余校验码
UDP的数据部分	

HTTP报头

请求报头

应答报头：

HTTP的请求方法与状态码

请求方法	含　　义
GET	申请获取资源，而不对服务器产生任何其他影响
HEAD	和 GET 方法类似，不过仅要求服务器返回头部信息，而不需要传输任何实际内容
POST	客户端向服务器提交数据的方法。这种方法会影响服务器：服务器可能根据收到的数据动态创建新的资源，也可能更新原有的资源
PUT	上传某个资源
DELETE	删除某个资源
TRACE	要求目标服务器返回原始 HTTP 请求的内容。它可用来查看中间服务器（比如代理服务器）对 HTTP 请求的影响
OPTIONS	查看服务器对某个特定 URL 都支持哪些请求方法。也可以把 URL 设置为 *，从而获得服务器支持的所有请求方法
CONNECT	用于某些代理服务器，它们能把请求的连接转化为一个安全隧道
PATCH	对某个资源做部分修改

短连接和长连接：

> 短连接：一次http请求服务器应答之后就将这个连接断开 。
>
> 长连接：多个http请求可以共用同一个连接。

状态码：

状态类型	状态码和状态信息	含义
1xx 信息	100 Continue	服务器收到了客户端的请求行和头部信息，告诉客户端继续发送数据部分，客户端通常要先发送 Expect:100-continue 头部字段告诉服务器自己还有数据要发送
2xx 信息	200 OK	请求成功
3xx 重定向	301 Moved Permanently	资源被转移了，请求将被重定向
	302 Found	通知客户端资源能在其他地方找到，但是需要使用 GET 方法来获得它
	304 Not Modified	表示被申请的资源没有更新，和之前获得的相同

	307 Temporary	通知客户端资源能在其他地方找到，与 302 不同的是，客户端可以使用和原始请求相同的请求方法来访问目标资源
4xx 客户端错误	400 Bad Request	通用客户请求错误
	401 Unauthorized	请求需要认证信息
	403 Forbidden	访问被服务器禁止，通常是由于客户端没有权限访问该资源
	404 Not Found	资源没找到
	407 Proxy Authentication Required	客户端需要先获得代理服务器的认证
5xx 服务器错误	500 Internal Server Error	通用服务器错误
	503 Service Unavailable	暂时无法访问服务器

TCP和UDP的区别和各自适用的场景

TCP	UDP
面向连接的、可靠的、字节流服务	**无连接、不可靠、数据报服务**
点对点的两点间服务，即一条TCP连接只能有两个端点	支持一对一，一对多，多对一，多对多的交互通信
可靠交付：无差错，不丢失，不重复，按序到达	尽最大努力交付，不保证可靠交付
首部开销大，一次传输的东西少	首部开销小，一次传输的东西多

适用场景：

> TCP适用于对数据的准确性有要求，不在乎速率(重要的文件传输，状态的更新等)
>
> UDP适用于对数据的准确性没太大要求，但速率很快(视频传输，实时通信等)

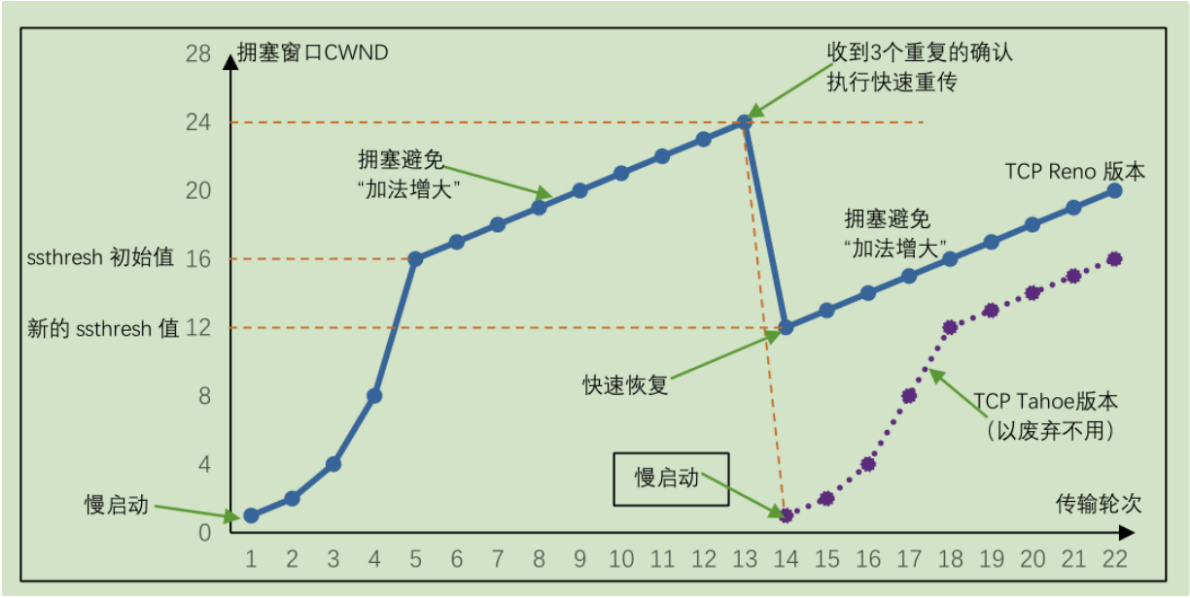
请你说一下TCP怎么保证可靠性

TCP保证可靠性：三次握手是保证可靠性的基础

> （1）序列号、确认应答、超时重传
>
> （2）窗口控制与高速重发控制/快速重传（重复确认应答）
>
> （3）拥塞控制

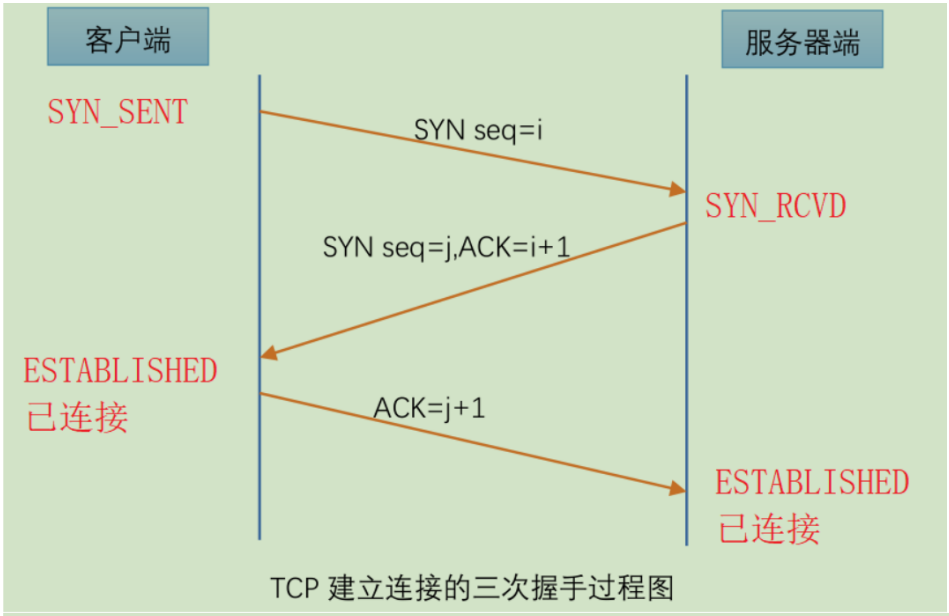
拥塞控制的四种方法：

> 慢启动、拥塞避免、快速重传、快速恢复

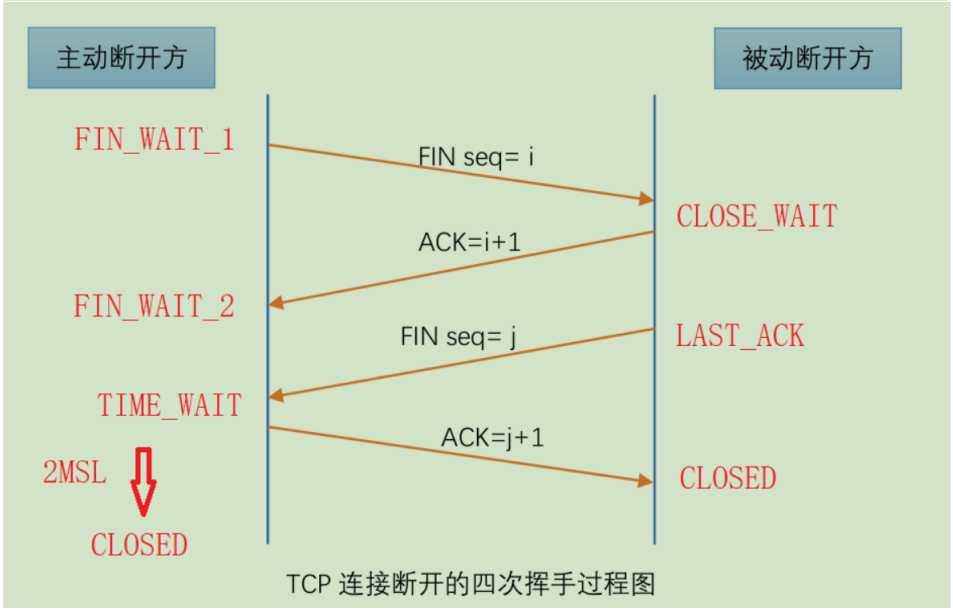


快速重传：接收方每次收到一个失序的报文段后就立即发出重复确认，发送方只要连续收到三个重复确认就立即重传（尽早重传未被确认的报文段）。

TCP的三次握手和四次挥手



TCP 建立连接的三次握手过程图



TCP 连接断开的四次挥手过程图

为什么是三次握手?

> 三次握手是为了防止，客户端的请求报文在网络滞留，客户端超时重传了请求报文，服务端建立连接，传输数据，释放连接之后，服务器又收到了客户端滞留的请求报文，建立连接一直等待客户端发送数据。
>
> 服务器对客户端的请求进行回应(第二次握手)后，就会理所当然的认为连接已建立，而如果客户端并没有收到服务器的回应呢？此时，客户端仍认为连接未建立，服务器会对已建立的连接保存必要的资源，如果大量的这种情况，服务器会崩溃。(SYN溢出攻击)

为什么是四次挥手?

> 因为tcp是全双工通讯，一端断开只能说明本端不再发送数据。

TIME_WAIT状态的含义，为什么还要等待2MSL才变成CLOSED状态

> 1. 保证迟来的数据能够被识别并丢弃。防止对后续的连接产生影响。
> 2. 保证最后的ACK能够到达对端，使得对端能够正常退出 。

MSL：最大报文段生存时间

请回答一下HTTP和HTTPS的区别，以及HTTPS有什么缺点?

HTTP协议和HTTPS协议区别如下：

1）HTTP协议是以明文的方式在网络中传输数据，而HTTPS协议传输的数据则是经过TLS加密后的，HTTPS具有更高的安全性

2）HTTPS在TCP三次握手阶段之后，还需要进行SSL 的handshake，协商加密使用的对称加密密钥

3）HTTPS协议需要服务端申请证书，浏览器端安装对应的根证书

4）HTTP协议端口是80，HTTPS协议端口是443

HTTPS优点：

HTTPS传输数据过程中使用密钥进行加密，所以安全性更高

HTTPS协议可以认证用户和服务器，确保数据发送到正确的用户和服务器

HTTPS缺点：

HTTPS握手阶段延时较高：由于在进行HTTP会话之前还需要进行SSL握手，因此HTTPS协议握手阶段延时增加

HTTPS部署成本高：一方面HTTPS协议需要使用证书来验证自身的安全性，所以需要购买CA证书；另一方面由于采用HTTPS协议需要进行加解密的计算，占用CPU资源较多，需要的服务器配置或数目高

浏览器中输入一个URL会发生什么?

浏览器要将URL解析为IP地址，解析域名就要用到DNS协议，首先主机会查询DNS的缓存，如果没有就给本地DNS发送查询请求。DNS查询分为两种方式，一种是递归查询，一种是迭代查询。如果是迭代查询，本地的DNS服务器，向根域名服务器发送查询请求，根域名服务器告知该域名的一级域名服务器，然后本地服务器给该一级域名服务器发送查询请求，然后依次类推直到查询到该域名的IP地址。DNS服务器是基于UDP的，因此会用到UDP协议。

得到IP地址后，浏览器就要与服务器建立一个http连接。因此要用到http协议，http协议报文格式上面已经提到。http生成一个get请求报文，将该报文传给TCP层处理，所以还会用到TCP协议。如果采用https还会使用https协议先对http数据进行加密。TCP层如果有需要先将HTTP数据包分片，分片依据路径MTU和MSS。TCP的数据包然后会发送给IP层，用到IP协议。IP层通过路由选路，一跳一跳发送到目的地址。当然在一个网段内的寻址是通过以太网协议实现(也可以是其他物理层协议，比如PPP，SLIP)，以太网协议需要直到目的IP地址的物理地址，有需要ARP协议。

GET和POST的区别

1、概括

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）

2、区别：

1、get参数通过url传递，post放在request body中。

2、get请求在url中传递的参数是有长度限制的，而post没有。

3、get比post更不安全，因为参数直接暴露在url中，所以不能用来传递敏感信息。

4、get请求只能进行url编码，而post支持多种编码方式。

5、get请求会浏览器主动cache，而post支持多种编码方式。

6、get请求参数会被完整保留在浏览历史记录里，而post中的参数不会被保留。

7、GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。

8、GET产生一个TCP数据包；POST产生两个TCP数据包。

请你说一下阻塞，非阻塞，同步，异步

阻塞和非阻塞：调用者在事件没有发生的时候，一直在等待事件发生，不能去处理别的任务这是阻塞。调用者在事件没有发生的时候，可以去处理别的任务这是非阻塞。

同步和异步：调用者必须循环自去查看事件有没有发生，这种情况是同步。调用者不用自己去查看事件有没有发生，而是等待着注册在事件上的回调函数通知自己，这种情况是异步

IO复用

IO复用的方式有三种，**select，poll，epoll(Linux独有)**。

select：

> 最多只能监听1024个文件描述符，而且文件描述符的值最大为1023
>
> 只能关注3种事件类型： 读事件、写事件、异常事件
>
> 内核会在线修改用户传递关注事件的文件描述符的集合的变量（修改fd_set结构）， 每次调用select之前都必须重新设置三个fd_set结构变量。
>
> select返回后，只是告诉用户程序有几个文件描述符就绪，但是并没有指定是哪几个文件描述符。用户程序就需要遍历所有的文件描述符，在探测哪些文件描述符就绪，所以时间复杂度为O(n)。
>
> 用户程序得自己维护所有的文件描述符。然后，每次调用select的时候，都需要将用户空间的fd_set集合传递给内核空间，select返回时，又需要将内核的fd_set传递给用户空间。这样调用select的时候，会存在两次数据的拷贝，效率不是很高。
>
> select内核采用的是轮询的方式去监测哪些文件描述符上的事件就绪。
>
> select的工作模式只能是LT模式

poll:

> poll所能关注的文件描述符的个数是没有限制的。文件描述符的取值范围也是没有限制的。
>
> poll支持的事件类型更多

> poll返回时，内核修改的数组元素中revents成员，与用户填充的关注的事件类型不冲突，所以每次调用poll之前，不需要重新设置这个数组。
>
> poll返回时，也仅仅是返回就绪文件描述符的个数，并没有指定是哪几个文件描述符就绪。所以用户程序监测就绪文件描述符的时间复杂度为O(n)
>
> poll每次调用时，也是需要将用户空间的数组拷贝给内核，poll返回时，由将内核的拷贝到用户空间。
>
> poll内核也是采用轮询的方式
>
> poll也只能工作在LT模式。

poll关注的事件类型：

POLLIN	**数据可读（包括普通数据和优先数据）**
POLLOUT	数据可写（包括普通数据和优先数据）
POLLRDHUP	TCP链接被对方关闭，对方关闭了写操作
POLLHUP	挂起，管道写端关闭，读端会受到该事件
POLLERR	错误
POLLNVAL	文件描述符没有打开

epoll：epoll是一组方法，epoll_create()、 epoll_ctl()、 epoll_wait();

> epoll不在是一个接口，而是一组接口
>
> 能够监听的文件描述符个数没有限制，值的范围也没有限制
>
> 关注的事件类型更多 EPOLLET EPOLLONESHOT
>
> epoll在内核中维护用户关注的文件描述符和事件类型，每次epoll_wait时，不需要传递用户空间的数据。epoll_wait返回时，仅仅返回就绪的文件
>
> 描述符和事件，相比于select和poll，效率更高。
>
> epoll_wait返回的仅仅是就绪的事件，所以用户程序监测就绪文件描述符的时间复杂度就为O(1)。
>
> epoll内核采用的是回调的方式监测事件就绪
>
> epoll不仅支持LT模式，也支持高校的ET模式

epoll的LT和ET模式：

> 区别： 当epoll_wait将就绪事件返回后，对于LT模式，如果用户程序没有处理该就绪事件，下一次epoll_wait还会通知这个就绪事件。对于ET模式，
> 如果用户程序没有处理该就绪事件，则下一次epoll_wait不会通知这个就绪事件。ET模式下，同一个就绪事件只会通知一次，相较于LT模式，效率会更高一些。