题库（尽可能结合项目讲解！！！！！！！！！！！！！）

指针和引用的区别

1定义和性质不同。指针是一种数据类型，用于保存地址类型的数据，而引用可以堪称是变量的别名。指针定义格式为：数据类型\*；而引用的定义格式为：数据类型&；

2引用不可以为空，当被创建的时候必须初始化，而指针变量可以是空值，在任何时候初始化；

3指针可以有多级，但引用只能是一级

4引用使用时无需解引用(\*)，指针需要解引用；

5指针变量的值可以是NULL，而引用的值不可以为NULL；

6指针的值在初始化后可以改变，即指向其他的存储单元，而引用在进行初始化后就不会再改变了

7sizeof引用得到的是所指向的变量的大小，而sizeof指针得到的是指针变量本身的大小

8指针作为函数参数传递时传递的是指针变量的值，而引用作为函数参数传递时传递的是实参本身，而不是拷贝副本

9引用和指针进行++运算的意义不一样

重载，重写，重定义的区别

重载：函数名相同，函数的参数个数，参数类型或参数顺序三者必须至少有一种不同。函数返回值的类型可以相同，也可以不同。发生在一个类内部，不能跨作用域。

重定义：也叫做隐藏，子类重新定义父类中有相同名称的非虚函数，指派生类的函数屏蔽了与其同名的基类函数。可以理解成发生在继承中的重载。

重写：也叫做覆盖，一般发生在子类和父类继承关系之间。子类重新定义父类中有相同名称和参数的虚函数

TCP和UDP的区别

1TCP是面向对象链接；而UDP是无面向对象连接

2TCP确保所传输的数据一定会到达目的地，但时间却不能保证；

而UDP的使用场景则是对实时性要求较高。

3TCP的使用场景是不在意传输数据速度，但在意传输数据准确；而UDP的使用场景则是对实用性要求较高。

4TCP的首部开销大，占用20字节而UDP的首部开销小，只占用8字节

5TCP只支持点对点的数据传输；而UDP支持一对一，一对多，多对多等各种场景

6TCP有拥塞控制和流量控制来确保数据的安全性，但如果网络拥塞的话，速度会比较慢；

UDP即使在网络拥塞的时候，传输速度也不会慢

关于static用途说法正确的是？

1声明静态全局变量2声明静态函数3声明静态局部变量

TCP三次握手四次挥手

标准回答：第一、二次分别包含数据通讯初始序号。第三次是必须的，为了防止已经失效的连接请求报文突然又被传送给了服务器端，然后产生错误。TCP是全双工通讯，客户端和服务器端都需要释放连接和接受确认，所以必须是四次挥手。 标准回答 三次握手过程：第一次握手：客户端向服务器端发送连接请求报文段，包含自身数据通讯初始序号，进入SYN-SENT状态。第二次握手：服务器端收到连接请求报文段后，如果同意，发送应答，包含自身数据通讯初始序号，进入SYN-RECEIVED状态。第三次握手：客户端收到应答，最后向服务器端发送确认报文，进入ESTABLISHED状态，此时成功建立长连接。 四次挥手过程：首先第一次挥手：客户端认为数据发送完毕，需要向服务器端发送连接释放请求。第二次挥手：服务器收到连接释放请求，告诉应用层释放TCP连接。然后发送ACK包，进入CLOSE-WT状态，此时表明客户端到服务器端的连接已经释放，不再接受客户端的数据。因为TCP是全双工的，所以服务器仍可以发送数据。第三次挥手：当服务器端数据发送完毕，向客户端发送连接释放请求，进入LAST-ACK状态。第四次挥手：客户端收到连接释放请求，向服务器端发送确认应答报文，此时客户端进入TIME-WT状态，持续2倍的MSL（最长报文段寿命），若期间没有收到服务器端的数据报文，进入CLOSED状态。服务器端收到确认应答后，也进入CLOSED状态。 加分回答 以下是客户端向服务器端发起TCP连接的详细过程： 1. 客户端和服务器端刚开始都是处于CLOSED（关闭）状态。 2. 要注意的是客户端主动打开连接，而服务器端是被动打开连接的。 3. 服务器端的进程先创建TCB（传输控制块）准备接受客户端的连接请求。 4. 客户端的进程也是先创建TCB（传输控制块），然后向服务器端发出连接请求报文段，这个报文段中的同步位SYN置为1，同时选择一个初始序号seq=x。TCP协议规定了SYN=1的报文段不可以携带数据，但是要消耗掉一个序号。这个时候客户端进入SYN-SENT状态。 5. 服务器端收到连接请求报文之后，如果同意连接，就给客户端发送确认响应。在确认报文中应该将同步位SYN和ACK都置为1，而确认号是ACK+1。这时候服务器端也需要给自己选一个初始序号seq=y。值得注意的是这个确认报文也不能携带数据，同样要消耗掉一个序号。这时服务器端进入SYN-RECEIVED状态。 6. 客户端进程收到服务器端的确认报文，最后还要向服务器端给出确认。确认报文段的ACK置为1，确认号是y+1，而自己的序号seq=x+1。TCP标准规定，ACK报文段可以携带数据，但是如果不携带数据就不消耗序号。在这个情况下，下一个数据报文的序号仍然是seq=x+1。到这时，TCP连接已经成功建立，A进入ESTABLISHED（已建立连接）状态。 到此TCP连接三次握手的过程就全部结束了。但是为什么一定要三次握手而不是两次，为什么客户端最后还需要发送一次确认报文呢？其实主要是为了防止已经失效的连接请求报文突然又被传送给了服务器端，然后产生错误。假设现在有一种情况，客户端发出的第一个连接请求报文段并没有丢失而是在某些网络节点上被滞留了，直到客户端和服务器端的新连接已经释放后的某个时间点，第一个连接请求报文段才到了服务器端，这时候服务器端以为客户端又发起了一次请求，于是服务器端向客户端发起了确认连接报文段，同意连接。假设不采用三次握手，这时候连接已经建立了，但是客户端并不知道这个情况，服务器端会一直等待客户端的数据报文，这样服务器端的资源就会被浪费，占用大量的资源。所以采用三次握手可以防止这种现象，保护网络和系统资源。 TCP连接释放的过程比较复杂，客户端和服务器端都可以主动释放连接。下面是从客户端主动释放连接为例讲解四次挥手的详细过程： 1. 客户端的应用进程先向TCP发出一个连接释放报文段，然后停止发送数据报，主动关闭TCP连接。客户端需要将连接释放报文段首部的终止控制FIN置为1，序号设置为u，u相当于前面传输的数据报文段的最后一个字节的序号加1。这时候客户端进入FIN-WT-1（终止等待1）状态，等待服务器端的确认。需要注意的是，FIN报文段也是即使不携带数据，它也消耗一个序号。 2. 服务器在收到客户端发来的连接释放报文段请求之后就发出确认，确认号ack=u+1，这个报文段自己的序号是v，v相当于之前已经传送出去的最后一个报文段的序号加1。这时候服务器端进入CLOSE-WT（关闭等待）状态，这时候服务器端的TCP进程就要通知应用进程，客户端到服务器端的连接已经关闭了。需要注意的是，这个时候的TCP连接就处于一个半关闭（half-colse）的状态，尽管客户端已经没有数据要发送了，但是服务器端还是可以向客户端发送数据的，服务器端到客户端的连接并没有被释放掉。 3. 如果服务器端也没有数据要发送给客户端了，那么应用进程就通知TCP释放连接。这时候服务器端发出的连接释放报文段请求的终止指令FIN也置为1。这时候服务器端的序号已经是w了，因为在半关闭状态服务器端可能又发送了一些数据，服务器也必须重复上次已经发送过的确认号ack=u+1。这时候服务器端进入LAST-ACK（最后确认）状态，等待客户端的确认。 4. 客户端收到服务器端的连接释放请求报文段之后，必须发出确认。在确认报文段中把ACK置为1，确认号ack=w+1，而自己的序号是seq=u+1（根据TCP标准，FIN消耗了一个序号），然后进入TIME-WT（时间等待）状态，这时候连接并没有释放掉，必须等到2倍的MSL（最长报文段寿命）之后，连接才会释放。

简单回答：tcp的三次挥手：的作用是保证 通信双方都能够正常的收发信息；三次握手的发生阶段是在客户端连接服务器的connect阶段开始的，首先客户端发送一个syn=1，ack给服务器，服务器接收到之后将ack+1，syn=1，ACK包给客户端，这时候，客户端在将ACK+1，ack +1包发送给服务器，这样服务器收到之后就能够，服务器客户端就能够双方都知道可以相互通信，三次握手完成可以开始传输数据。 tcp的四次挥手：的作用是将服务器和客户端的连接安全的断开，四次挥手是发生在客户端或者服务器断开连接的时候，客户端向服务器发送一个fin=1，ack包，服务器收到之后将ack+1包发送给客户端，这时候，服务器会讲传输的数据传输完，之后再发一个fin的包给客户端，客户端收到之后发送一个ACK=1，ack+1包给服务器，这样四次挥手就完成了，服务器和客户端双方都断开了链接。

文件传输思路

将文件拆分成若干个小片，并且记录每个小片的下标，大小，装入容器，

发出n次请求不断向服务器提交文件片，服务端不断追加文件内容

当已上传文件大小达到文件总大小时，上传结束

服务器按照小片下标进行拼合

静态绑定和动态绑定

对象的静态类型：

对象在声明时候的类型，是在编译时期确定的。

对象的动态类型：

目标所指向的对象，是在运行期决定的。对象的动态类型可以更改，但是静态类型无法更改。

静态绑定：

绑定的是静态类型，比如函数依赖于对象的静态类型，发生在编译期。

动态绑定：

绑定的是动态类型，比如函数依赖于对象的动态类型，发生在运行期。

信号和槽原理

谓信号与槽，实际就是观察者模式(发布-订阅模式)。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

一 、QT connect 第5个参数一般不填，为默认值。

1、Qt::AutoConnection： 默认值，使用这个值则连接类型会在信号发送时决定。如果接收者和发送者在同一个线程，则自动使用Qt::DirectConnection类型。如果接收者和发送者不在一个线程，则自动使用Qt::QueuedConnection类型。

2、Qt::DirectConnection：槽函数会在信号发送的时候直接被调用，槽函数和信号发送者在同一线程。效果看上去就像是直接在信号发送位置调用了槽函数，效果上看起来像函数调用，同步执行。

emit语句后面的代码将在与信号关联的所有槽函数执行完毕后才被执行。

无论槽函数所属对象在哪个线程，槽函数都在发射信号的线程内执行。

3、Qt::QueuedConnection：信号发出后，信号会暂时被放到一个消息队列中，需等到接收对象所属线程的事件循环取得控制权时才取得该信号，然后执行和信号关联的槽函数，这种方式既可以在同一线程内传递消息也可以跨线程操作。

emit语句后的代码将在发出信号后立即被执行，无需等待槽函数执行完毕

槽函数在接收者所依附线程执行。

4、Qt::BlockingQueuedConnection：槽函数的调用时机与Qt::QueuedConnection一致，不过发送完信号后发送者所在线程会阻塞，直到槽函数运行完。而且接收者和发送者绝对不能在一个线程，否则程序会死锁。在多线程间需要同步的场合可能需要这个。

5、Qt::UniqueConnection：这个flag可以通过按位或（|）与以上四个结合在一起使用。当这个flag设置时，当某个信号和槽已经连接时，再进行重复的连接就会失败。也就是为了避免重复连接。

一般qt connect的第五个参数会在多线程中运用到,需要注意的是:

QThread是用来管理线程的，QThread对象所依附的线程和所管理的线程并不是同一个概念。QThread所依附的线程，就是创建QThread对象的线程;QThread 所管理的线程，就是run启动的线程，也就是新建线程。

例如:QThread对象依附在主线程中，QThread对象的slot函数会在主线程中执行，而不是次线程。除非QThread对象依附到次线程中(通过movetoThread)。

————————————————

什么是拷贝构造

CProp pr1;

CProp pr2(pr1); //拷贝构造的显式调用

CProp pr3 = pr1; //拷贝构造的隐式调用

CProp \*pPr = new CProp(pr1); //拷贝构造的显式调用

拷贝函数，它是一种特殊的[构造函数](https://so.csdn.net/so/search?q=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/y2629924166/article/details/_blank)，主要用来完成一些基于同一类的其他对象的构建及初始化，用自身这种类型的对象来构造自身。分为浅拷贝和深拷贝。

浅拷贝就是简单的赋值操作

深拷贝是给它一个新的空间然后其中的值赋值给它，并且它的改变不会影响原本的值

SQL语句中排序怎么做

Select 表字段from 表名order by 表字段 desc

其中desc表示倒叙，asc表示升序(默认情况下，我们的排序是“升序”，所有我们可以省略不写“asc”)

有没有了解过设计模式

https://blog.csdn.net/A1342772/article/details/91349142?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168050218116800211561441%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=168050218116800211561441&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_positive~default-1-91349142-null-null.142^v81^insert\_down38,201^v4^add\_ask,239^v2^insert\_chatgpt&utm\_term=%E8%AE%BE%E8%AE%A1%E6%A8%A1%E5%BC%8F&spm=1018.2226.3001.4187

STL容器原理

****1.STL共有六大组件****  
 1、容器。

 2、算法。

 3、迭代器。

 4、仿函数。

 6、适配器。

****2.STL容器的实现原理****

STL来管理数据十分方便,省去了我们自己构建数据结构的时间.其实,STL的实现也是基于我们常见的数据结构.

序列式容器：  
(1)vector-数组，元素不够时再重新分配内存，拷贝原来数组的元素到新分配的数组中。  
(2)list－单链表。  
(3)deque-分配中央控制器map(并非map容器)，map记录着一系列的固定长度的数组的地址.记住这个map仅仅保存的是数组的地址,真正的数据在数组中存放着.deque先从map中央的位置(因为双向队列，前后都可以插入元素)找到一个数组地址，向该数组中放入数据，数组不够时继续在map中找空闲的数组来存数据。当map也不够时重新分配内存当作新的map,把原来map中的内容copy的新map中。所以使用deque的复杂度要大于vector，尽量使用vector。  
(4)stack-基于deque。  
(5)queue-基于deque。  
    heap-完全二叉树，使用最大堆排序，以数组(vector)的形式存放。  
(6)priority\_queue-基于heap。  
(7)slist-双向链表。

关联式容器：  
set,map,multiset,multimap-基于红黑树(RB-tree)，一种加上了额外平衡条件的二叉搜索树。  
hash table-散列表。将待存数据的key经过映射函数变成一个数组(一般是vector)的索引，例如：数据的key%数组的大小＝数组的索引(一般文本通过算法也可以转换为数字)，然后将数据当作此索引的数组元素。有些数据的key经过算法的转换可能是同一个数组的索引值(碰撞问题，可以用线性探测，二次探测来解决)，STL是用开链的方法来解决的，每一个数组的元素维护一个list，他把相同索引值的数据存入一个list，这样当list比较短时执行删除，插入，搜索等算法比较快。  
hash\_map,hash\_set,hash\_multiset,hash\_multimap-基于hash table。

IPC技术场景

****1. 管道：****只能用于具有亲缘关系的进行通信，使用面相对较窄，实际开发中较少使用；

****2. FIFO(命名管道)****：可以用于任意进程间的通信，对于大块数据的传输效率较高，可应用于单进程大量数据传递，和多个进程向一个进程传递数据；

****3. 信号****：无法传递数据，而且信号的种类有限，只适用于完成一些简单的事件通知任务，如配置跟新信号通知，一个服务通过信号告知另一个服务自身状态；

****4. 文件锁****：不能用来传递数据，用来对操作进行协调，利用文件锁实现多个进程对于某个资源的排队请求，或者多个进程对系统某个全局资源进行读写操作，可以通过文件锁实现进程间读写锁的功能；

****XSI IPC：****

****5. 共享内存****：最为高效的进程间通信方式，进程可以直接读写内存，不需要任何数据拷贝，适用于多个进程共享数据，或进程间频繁的进行大量的数据交互；****--建议使用mmap方式；****

****6. 消息队列****：进程间传递简单的命令和控制消息，如配置更新通知，多进程对多进程的通信等，可以简化代码逻辑；****--建议使用全双工管道替代；****

****7. 信号量****：某种资源数为N，多个进程都在使用该资源，为了进行进程间的互斥，可以使用初始值为N的信号量；****--建议使用记录锁替代；****

Epoll中LT和ET

 epoll 是一种更加高效的 IO 复用技术,epoll 的使用步骤及原理如下： 1. 调用 epoll\_create() 会在内核中创建一个 eventpoll 结构体数据,称之为 epoll 对象,在这个结构体中有 2 个比较重要的数据成员,一个是需要检测的文件描述符的信息 struct\_root rbr（红黑树）,还有一个是就绪列表struct list\_head rdlist,存放检测到数据发送改变的文件描述符信息（双向链表）； 2. 调用 epoll\_ctrl() 可以向 epoll 对象中添加、删除、修改要监听的文件描述符及事件； 3. 调用 epoll\_wt() 可以让内核去检测就绪的事件,并将就绪的事件放到就绪列表中并返回,通过返回的事件数组做进一步的事件处理。 epoll 的两种工作模式： 1. LT 模式（水平触发） LT（Level - Triggered）是缺省的工作方式,并且同时支持 Block 和 Nonblock Socket。在这种做法中,内核检测到一个文件描述符就绪了,然后可以对这个就绪的 fd 进行 IO 操作,如果不作任何操作,内核还是会继续通知。 2. ET 模式（边沿触发） ET（Edge - Triggered）是高速工作方式,只支持 Nonblock socket。在这种模式下,当描述符从未就绪变为就绪时,内核通过 epoll 检测到。然后它会假设你知道文件描述符已经就绪,并且不会再为那个文件描述符发送更多的就绪通知,直到你做了某些操作导致那个文件描述符不再为就绪状态了。但是请注意,如果一直不对这个 fd 进行 IO 操作（从而导致它再次变成未就绪）,内核不会发送更多的通知（only once）。 ET 模式在很大程度上减少了 epoll 事件被重复触发的次数,因此效率要比 LT 模式高。epoll 工作在 ET 模式的时候,必须使用非阻塞套接口,以避免由于一个文件描述符的阻塞读/阻塞写操作把处理多个文件描述符的任务饿死。

服务器高并发处理

1)从客户端看 ：

尽量减少请求数量，比如：依靠客户端自身的缓存或处理能力；尽量减少对服务

端资源的不必要耗费，比如：重复使用某些资源，如连接池客户端处理的基本原则就是：能不访

问服务端就不要访问。

2)从服务端看：

增加资源供给，比如：更大的网络带宽，使用更高配置的服务器，使用高性能的

Web服务器，使用高性能的数据库；请求分流，比如：使用集群,分布式的系统架构；应用优化，

比如：使用更高效的编程语言,优化处理业务逻辑的算法,优化访问数据库的SQL。总之，基本原则

是分而治之，并提高单个请求的处理速度。

高并发处理的基本手段

客户端发出请求层面，常见的手段有：尽量利用浏览器的缓存功能，减少访问服务端，

比如：js、css、图片等；可以考虑使用压缩传输的功能，减少网络流量，也会提高传输速度；

考虑使用异步请求，分批获取数据。

前端接收客户端请求层面，常见的手段有：动静分离，部分静态资源可以直接从Nginx返回；

按请求的不同，分发到不同的后端进行处理，比如：负载均衡、业务拆分访问等；前面再加上一层

来做多个Nginx的负载均衡，比如：LVS、F5等；还可以在更前面使用CDN服务；还可以对动态内容

进行缓存，尽量减少访问后端服务。

Web服务器层面，常见的手段有：使用最新的JVM，并进行配置优化；对Web服务器进行配置优化，

比如：调整内存数量、线程数量等；提供多个能提供相同服务的Web服务器，以实现负载均衡；

仔细规划Web服务器上部署的应用规模；对Web服务器进行集群。

Web应用层面，常见的手段有：动态内容静态化、Java开发优化、优化处理业务逻辑的算法、合理

高效的利用缓存；优化访问数据库的Sql，可以考虑利用存储过程等数据库的能力；合理使用多线、

程，加快业务处理；部分业务可以考虑内存数据库，或者是进行纯内存处理；尽量避免远程调用、

大量I/O等耗时的操作；合理规划事务等较为耗资源的操作；合理使用异步处理；对部分业务考虑

采用预处理或者预计算的方式，减少实时计算量；内部系统间业务尽量直接调用、直接处理，

减少WebService、工作流等。

数据库层面，常见的手段有：合理选择数据库的引擎，比如Mysql的InnoDB与MyISAM引擎；进行配置优化；

可以考虑使用存储过程来处理复杂的数据逻辑；数据库集群，进行读写分离；合理设计数据库的表结构、

索引等；分库、分表，降低单库、单表的数据量。

链表的基本原理(单向)

链表每一个节点都是由数据域和指针域构成的，每一个指针域指向下一个节点，每个节点的数据域存储相应的数据，

两点除外

1）头节点，一般头节点不存储任何数据，但也可以用来存储表长等信息

2）最后一个节点的指针域指向NULL，即空，表示链表结束

特点：

1. 访问单链表中任何一个元素都必须通过头结点层层往下遍历，因此头节点是链表中最重要的节点，没有头节点就无法访问整个链表。这也是链表和顺序表最大的不同。

2. 链表则可以动态改变长度，加之指针的方式使得其遍历速度极为高效，其灵活性和可靠性使得链表成为线性表的主要实现方式，也是栈和队列等重要数据结构类型的基础。

备注： 顺序表是数组实现，访问节点以及返回表长都很容易实现，但是顺序表没有动态分配空间的功能，只能一开始初始化一个较大的空间以便以后利用，一旦填满，则顺序表不能再插入元素。

C++面向对象多态

得分点 静态多态、动态多态、多态的实现原理、虚函数、虚函数表 标准回答 在现实生活中，多态是同一个事物在不同场景下的多种形态。在面向对象中，多态是指通过基类的指针或者引用，在运行时动态调用实际绑定对象函数的行为，与之相对应的编译时绑定函数称为静态绑定。所以多态分为静态多态和动态多态。 1. 静态多态 静态多态是编译器在编译期间完成的，编译器会根据实参类型来选择调用合适的函数，如果有合适的函数就调用，没有的话就会发出警告或者报错。静态多态有函数重载、运算符重载、泛型编程等。 2. 动态多态 动态多态是在程序运行时根据基类的引用（指针）指向的对象来确定自己具体该调用哪一个类的虚函数。当父类指针（引用）指向 父类对象时，就调用父类中定义的虚函数；即当父类指针（引用）指向 子类对象时，就调用子类中定义的虚函数。 加分回答 1. 动态多态行为的表现效果为：同样的调用语句在实际运行时有多种不同的表现形态。 2. 实现动态多态的条件： - 要有继承关系 - 要有虚函数重写（被 virtual 声明的函数叫虚函数） - 要有父类指针（父类引用）指向子类对象 3. 动态多态的实现原理 当类中声明虚函数时，编译器会在类中生成一个虚函数表，虚函数表是一个存储类虚函数指针的数据结构， 虚函数表是由编译器自动生成与维护的。virtual 成员函数会被编译器放入虚函数表中，存在虚函数时，每个对象中都有一个指向虚函数表的指针（vptr 指针）。在多态调用时, vptr 指针就会根据这个对象在对应类的虚函数表中查找被调用的函数，从而找到函数的入口地址。