C++面经大纲

C++基础知识

基础语法

const的作用

> 1. 修饰变量，说明该变量不可以被改变；
> 2. 修饰指针，分为指向常量的指针和指针常量；
> 3. 常量引用，经常用于形参类型，即避免了拷贝，又避免了函数对值的修改；
> 4. 修饰成员函数，说明该成员函数内不能修改成员变量

static的作用:

> 1. 修饰普通变量，修改变量的存储区域和生命周期，使变量存储在静态区，在main函数运行前就分配了空间，如果有初始值就用初始值初始化它，如果没有初始值系统用默认值初始化它。
> 2. 修饰普通函数，表明函数的作用范围，仅在定义该函数的文件内才能使用。在多人开发项目时，为了防止与他人命令函数重名，可以将函数定位为static。
> 3. 修饰成员变量，修饰成员变量使所有的对象只保存一个该变量，而且不需要生成对象就可以访问该成员。
> 4. 修饰成员函数，修饰成员函数使得不需要生成对象就可以访问该函数，但是在static函数内不能访问非静态成员

#pragmapack(n)

> 设定结构体、联合以及类成员变量以n字节方式对齐
>
> ```
> #pragmapack(push)//保存对齐状态
> #pragmapack(4)//设定为4字节对齐
> structtest
> {
> charm1;
> doublem4;
> intm3;
> };
> #pragmapack(pop)//恢复对齐状态
> ```

volatile

> - volatile关键字是一种类型修饰符，用它声明的类型变量表示可以被某些编译器未知的因素（操作系统、硬件、其它线程等）更改。所以使用volatile

> 告诉编译器不应对这样的对象进行优化。
- volatile关键字声明的变量，每次访问时都必须从内存中取出值（没有被volatile修饰的变量，可能由于编译器的优化，从CPU寄存器中取值）
- const可以是volatile（如只读的状态寄存器）
- 指针可以是volatile

extern"C"

> - 被extern限定的函数或变量是extern类型的
> - 被`extern"C"`修饰的变量和函数是按照C语言方式编译和连接的
>
> `extern"C"`的作用是让C++编译器将`extern"C"`声明的代码当作C语言代码处理，可以避免C++因符号修饰导致代码不能和C语言库中的符号进行链接的问题。
>
> ```
> #ifdef__cplusplus
> extern"C"{
> #endif
> void*memset(void*,int,size_t);
> #ifdef__cplusplus
> }
> #endif
> ```

C++struct和class的区别

> 总的来说，struct更适合看成是一个数据结构的实现体，class更适合看成是一个对象的实现体。
>
> 最本质的一个区别就是默认的访问控制
>
> 1. 默认的继承访问权限。struct是public的，class是private的。
> 2. struct作为数据结构的实现体，它默认的数据访问控制是public的，而class作为对象的实现体，它默认的成员变量访问控制是private的。

expliciit关键字

> - explicit修饰构造函数时，可以防止隐式转换和复制初始化
> - explicit修饰转换函数时，可以防止隐式转换，但按语境转换除外

空类占用内存大小

> 1个字节

智能指针：

> auto_ptr：
>
> shared_ptr：
>
> unique_ptr：
>
> weak_ptr：

---

weak_ptr如何转shared_ptr?

>

C++的四种类型转换

> 使用C风格的类型转换可以把想要的任何东西转换成我们需要的类型，但是这种类型转换太过松散，对于这种松散的情况，C++提供了更严格的类型转换，可以提供更好的控制转换过程，并添加4个类型转换运算符，使转换过程更规范：static_cast、dynamic_cast、const_cast、reinterpret_cast。
>
> static_cast静态转换
>
> 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
>
> o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
>
> o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
>
> 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
>
> dynamic_cast动态转换
>
> dynamic_cast主要用于类层次间的上行转换和下行转换
>
> 在类层次间进行上行转换时，dynamic_cast和static_cast的效果是一样的
>
> 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
>
> const_cast常量转换
>
> 该运算符用来修改类型的const属性
>
> 常量指针被转化成非常量指针，并且仍然指向原来的对象
>
> 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
>
> reinterpret_cast重新解释转换

> 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

C++中的内存对齐

> 主要就是struct的那一块。
>
> 对于32位来说默认四字节对齐
>
> 对于64位来说采用八字节对齐

简述一下C++从代码到可执行二进制文件的过程

> 标准回答
>
> C++从代码到可执行二进制文件经过四个过程，分别是：预编译、编译、汇编、链接。
>
> 1. 预编译，主要的处理操作：
>
>   a.将所有的#define删除，并且展开所有的宏定义（宏替换）
>
>   b.处理所有的条件预编译指令，如#if、#ifdef
>
>   c.处理#include预编译指令，将被包含的文件插入到该预编译指令的位置
>
>   d.删除所有的注释
>
>   e.添加行号和文件名标识
> 2. 编译：将预处理之后的代码转换成特定的汇编代码，主要包括词法分析、语法分析、语义分析、优化代码等操作
> 3. 汇编：将汇编代码汇编成机器指令
> 4. 链接：将不同源文件生成的目标代码以及其它目标代码、库文件组合起来，从而形成可执行程序
>
> 加分回答
>
> 链接分为静态链接和动态链接。
>
> 1. 静态链接：静态链接是由链接器在链接时将库的内容加入到可执行程序中，将一个或多个库或目标文件（先前由编译器或汇编器生成）链接到一块生成可执行程序。
> 2. 动态链接：动态链接在链接后动态库仍然与可执行文件分离，直到运行时才动态加载。

面向对象

简述一下C++中的多态

> 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：
>
>   o要有继承关系
>
>   o要有虚函数重写（被virtual声明的函数叫虚函数）
>
>   o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。

说一说C++中哪些不能是虚函数

> C++中，普通函数（非成员函数）、构造函数、友元函数、静态成员函数、内联成员函数这些不能是虚函数。

> 1.普通函数（非成员函数）
>
> 2.构造函数
>
> 3.友元函数
>
> 4.静态成员函数
>
> 5.内联成员函数

虚函数表运行时加载在虚拟地址空间的哪里?

> .rodata数据段

STL

STL常用容器

> STL中容器分为顺序容器、关联式容器、容器适配器三种类型，三种类型容器特性分别如下：
>
> 1、顺序容器
>
> 容器并非排序的，元素的插入位置同元素的值无关，包含vector、deque、list。
>
> ovector：动态数组
>
> 元素在内存连续存放。随机存取任何元素都能在常数时间完成。在尾端增删元素具有较佳的性能。
>
> odeque：双向队列
>
> 元素在内存连续存放。随机存取任何元素都能在常数时间完成（仅次于vector）。在两端增删元素具有较佳的性能（大部分情况下是常数时间）。
>
> olist：双向链表
>
> 元素在内存不连续存放。在任何位置增删元素都能在常数时间完成。不支持随机存取。
>
> 2、关联式容器
>
> 元素是排序的；插入任何元素，都按相应的排序规则来确定其位置；在查找时具有非常好的性能；通常以平衡二叉树的方式实现，包含set、multiset、map、multimap。
>
> oset/multiset

> set中不允许相同元素，multiset中允许存在相同元素。
>
> omap/multimap
>
> map与set的不同在于map中存放的元素有且仅有两个成员变，一个名为first，另一个名为second，map根据first值对元素从小到大排序，并可快速地根据first来检索元素。map和multimap的不同在于是否允许相同first值的元素。
>
> 3、容器适配器
>
> 封装了一些基本的容器，使之具备了新的函数功能，包含stack、queue、priority_queue。
>
> ostack：栈
>
> 栈是项的有限序列，并满足序列中被删除、检索和修改的项只能是最进插入序列的项（栈顶的项），后进先出。
>
> oqueue：队列
>
> 插入只可以在尾部进行，删除、检索和修改只允许从头部进行，先进先出。
>
> opriority_queue：优先级队列
>
> 内部维持某种有序，然后确保优先级最高的元素总是位于头部，最高优先级元素总是第一个出列。

C++11

sharedptr的引用计数对象在哪里？栈上还是堆上

> 堆

unique-ptr和scoped-ptr什么区别?

> 就比uniqueptr少了那个带右值引用的拷贝构造和重载赋值。

实现getSharedPtr让使技术引用为2

```
#include<iostream>
#include<memory>
usingnamespacestd;
/*代码*/
classA:publicenable_shared_from_this<A>{
public:
shared_ptr<A>getSharedPtr()
{
returnshared_from_this();
}
```

```
};
intmain()
{
shared_ptr<A>ptr1(newA());
shared_ptr<A>ptr2=ptr1->getSharedPtr();
cout<<ptr1.use_count()<<endl;
cout<<ptr2.use_count()<<endl;
return0;
}
```

简述一下C++11中auto的用法

> 标准回答
>
> 1. 实现自动类型推断，要求进行显示初始化，让编译器能够将变量的类型设置为初始值的类型：
>
> > autoa=12;//a为int类型
> >
> > autopt=&a;//pt为int*类型
> >
> > doublefm(doublea,intb){
> >
> > returna+b;
> >
> > }
> >
> > autopf=fm;//pf为double类型
>
> 2. 简化模板声明
>
> > for(std::initializer_list::iteratorp=il.begin();p!=il.end();p++)
> >
> > for(autop=il.begin();p!=il.end();p++)

linux

进程间通信的方式?

> 1、无名管道
>
> 2、有名管道
>
> 3、消息队列
>
> 4、信号量数组
>
> 5、共享内存

> 6、socket通信

- IO多路复用

  - 什么是IO多路复用?

  IO多路复用可以同时监听多个文件描述符，能够提高程序的性能，linux下实现I/O多路复用的系统调用主要有：select、poll、epoll。

  - IO模型：
    - 阻塞等待（BIO）：

      基本的read、write，采用多线程或者多进程解决

      缺点：1、线程或者进程会消耗资源；2、线程或进程调度消耗CPU资源

    - 非阻塞，忙轮询（NIO模型）：

      提高了程序的执行效率，采用IO多路复用解决。

      缺点：需要占用更多的CPU和系统资源。

    - 多路复用IO
      - select：

        主旨思想：

        1. 首先要构造一个关于文件描述符的列表，将要监听的文件描述符添加到该列表中。
        2. 调用一个系统函数，监听该列表中的文件描述符，直到这些描述符中的一个或者多个进行I/O操作时，该函数才返回。

           a.这个函数是阻塞

           b.函数对文件描述符的检测的操作是由内核完成的
        3. 在返回时，它会告诉进程有多少（哪些）描述符要进行I/O操作。

        缺点：

        - 每次要将fdset从用户态拷贝到内核态，当fd比较多这个开销会很大。
        - 内核需要遍历所有文件描述符，在文件描述符很多的时候开销比较大。
        - select支持的文件描述符数量太少，默认是1024。
        - fds集合不能重用每次都要重置。

- poll

```
#include<poll.h>
structpollfd{
intfd;/*委托内核检测的文件描述符*/
shortevents;/*委托内核检测文件描述符的什么事件*/
shortrevents;/*文件描述符实际发生的事件*/
};
structpollfdmyfd;
myfd.fd=5;
myfd.events=POLLIN|POLLOUT;
intpoll(structpollfd*fds,nfds_tnfds,inttimeou
t);
-参数：
    -fds:是一个structpollfd结构体数组，这是一个需要检测的文件描述符的集合
    -nfds:这个是第一个参数数组中最后一个有效元素的下标+1
    -timeout:阻塞时长
        0:不阻塞
        -1:阻塞，当检测到需要检测的文件描述符有变化，解除阻塞
        >0:阻塞的时长
-返回值：
    -1:失败
    >0（n）:成功,n表示检测到集合中有n个文件描述符发生变化
```

- epoll

1、采用红黑树

2、epoll底层在内核中，减少了用户态到内核态拷贝的动作。

```
include<sys/epoll.h>
//创建一个新的epoll实例。在内核中创建了一个数据，这个数据中有两个比较重要的数据，一个是需要检
测的文件描述符的信息（红黑树），还有一个是就绪列表，存放检测到数据发送改变的文件描述符信息（双向链表）。
intepoll_create(intsize);
    -参数：
        size:目前没有意义了。随便写一个数，必须大于0
    -返回值：
```

-1:失败
>0:文件描述符，操作epoll实例的

```
typedef  union epoll_data{
 void*ptr;
 intfd;
 uint32_tu32;
 uint64_tu64;
}epoll_data_t;
struct epoll_event{
uint32 _tevents;/*Epollevents*/
epoll_data _tdata;/*Userdatavariable*/
};
```

常见的Epoll检测事件：
-EPOLLIN
-EPOLLOUT
-EPOLLERR

```
//对epoll实例进行管理：添加文件描述符信息，删除信息，修改信息
intepoll_ctl(intepfd,intop,intfd,structepoll_event*event);
```

-参数：
-epfd:epoll实例对应的文件描述符
-op:要进行什么操作
EPOLL_CTL_ADD:添加
EPOLL_CTL_MOD:修改
EPOLL_CTL_DEL:删除
-fd:要检测的文件描述符
-event:检测文件描述符什么事情

```
//检测函数
int epoll_wait(int epfd,struct epoll_event*events,int maxevents,int timeout);
```

-参数：
-epfd:epoll实例对应的文件描述符
-events:传出参数，保存了发送了变化的文件描述符的信息
-maxevents:第二个参数结构体数组的大小
-timeout:阻塞时间
-0:不阻塞
-1:阻塞，直到检测到fd数据发生变化，解除阻塞
->0:阻塞的时长（毫秒）
-返回值：
-成功，返回发送变化的文件描述符的个数>0
-失败-1

- 水平触发LT:

- 边沿触发ET：

计算机网络

面经链接：https://www.nowcoder.com/discuss/839894

https://www.cnblogs.com/winlsr/p/15094958.html

网络基础

OSI七层模型

> OSI模型（OpenSystemInterconnectionModel）是一个由ISO提出得到概念模型，试图提供一个使各种不同的的计算机和网络在世界范围内实现互联的标准框架。

网络层次	常用协议	传输方式	功能
应用层	HTTP、SMTP、FTP、TELNET		这一层为操作系统或网络应用程序提供访问网络服务的接口。
表示层			使通信的应用程序能够解释交换数据的含义，说图像、视频编码解，数据加密。
会话层	SSL、TLS		管理主机之间的会话进程，即负责建立、管理、终止进程之间的会话
运输层	TCP、UDP	数据段	
网络层	IP、ARP、ICMP、IGMP	数据包	
链路层		将bit流封装成frame帧	不可靠的物理介质上提供可靠的传输。
物理层		bit流	确保原始的数据可在各种物理媒体上传输

网络层

> IP协议
>
> IP地址划分
>
> DHCP协议
>
> ICMP协议
>
>

> - 功能：ICMP经常被认为是IP层的一个组成部分。它传递差错报文以及其他需要注意的信息。
> - 类型：
>   - ICMP差错报文
>   - ICMP询问报文
> - 以下情况不会产生ICMP报文
>   - ICMP差错报文
>   - 目的地址是广播地址
>   - 作为链路层广播的数据报
>   - 不是IP分片的第一片
>   - 原地址不是单个主机的数据报。

IP转发

ARP协议

> - 功能：从逻辑Internet地址到对应的物理硬件地址需要进行翻译。
> - 请求协议格式：
>
>   。。。
> - 对不存在的主机发送一个ARP请求：
>
>   我们通过telnet测试会发现我们会看到多次arp的请求这是tcp协议重传机制造成的默认时长设置为30s，但实际上有些会设置为75秒。arp-a发现不存在主机返回iocomplete
> - ARP高速缓存
>   - 一般来说arp缓存设置超时为20分钟。
>   - ARP高效运行的关键是由于每个主机上都有一个ARP高速缓存。这个高速缓存存放了最近Internet地址到硬件地址之间的映射记录。
>   - 可以采用arp-a来查看浏览器缓存
> - ARP代理
>
>> 如果ARP请求是从一个网络的主机发往另一个网络上的主机，那么连接这两个网络的路由器就可以回答该请求，这个过程称作委托ARP或ARP代理(ProxyARP)。这样可以欺骗发起ARP请求的发送端，使它误以为路由器就是目的主机，而事实上目的主机是在路由器的“另一边”。路由器的功能相当于目的主机的代理，把分组从其他主机转发给它。

传输层

运输层的功能?

> 运输层协议会运行在不同主机上的应用进程提供逻辑通信的功能*/

UDP协议介绍:

> UDP数据报由20的IP首部(伪首部)+8的UDP首部（源端口16|目的端口16|UDP长度（**指的是首部+数据总长度最小为8个字节**）16|UDP检验和16）+数据
>
> **由于IP层已经把IP数据报分配给TCP或UDP（根据IP首部中协议字段值），因此TCP端口号由TCP来查看，而UDP端口号由UDP来查看。TCP端口号与UDP端口号是相互独立的。**
>
> UDP检验和：**如果发送端没有计算检验和而接收端检测到检验和有差错，那么UDP数据报就要被悄悄地丢弃。不产生任何差错报文（当IP层检测到IP首部检验和有差错时也这样做）。**
>
> 如何知道是否打开了UDP检验和：可以采用tcpdump

TCP协议介绍:

> TCP向应用层提供一种可靠的面向连接的字节流服务。

TCP如何保证可靠性?

> - 应用数据被分割成TCP认为最适合发送的数据块。
> - 当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。
> - 当TCP收到发自TCP连接另一端的数据，它将发送一个确认。
> - TCP将保持它首部和数据的检验和。
> - 如果必要，TCP将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。
> - 既然IP数据报会发生重复，TCP的接收端必须丢弃重复的数据。
> - TCP还能提供流量控制。

TCP和UDP的区别

> 1、从首部来看：UDP只占8个字节而TCP占20个字节
>
> 2、从连接来看：TCP是面向连接的而UDP无连接
>
> 3、从发送数据类型来看：UDP发送的是数据报，而TCP发送的是字节流
>
> 4、从数据安全来看：TCP保证数据正确性，UDP可能丢包
>
> 5、TCP保证数据顺序，UDP不保证。

三次握手：

> - 第一次：客户端发含SYN位，SEQ_NUM=S的包到服务器。（客->SYN_SEND）
> - 第二次：服务器发含ACK，SYN位且ACK_NUM=S+1，SEQ_NUM=P的包到客户机。（服->SYN_RECV）
> - 第三次：客户机发送含ACK位，ACK_NUM=P+1的包到服务器。（客->ESTABLISH，服->ESTABLISH）

序列号如何确定：

> 发送第一个SYN的一端将执行主动打开（active open）。接收这个SYN并发回下一个SYN的另一端执行被动打开（passive open）（在18.8节我们将介绍双方如何都执行主动打开）。
>
> 当一端为建立连接而发送它的 SYN时，它为连接选择一个初始序号。 ISN随时间而变化，因此每个连接都将具有不同的ISN。RFC 793 [Postel 1981c]指出ISN可看作是一个32比特的计数器，每4ms加1。这样选择序号的目的在于防止在网络中被延迟的分组在以后又被传送，而导致某个连接的一方对它作错误的解释。
>
> > 如何进行序号选择？在4.4BSD（和多数的伯克利的实现版）中，系统初始化时初始的发送序号被初始化为1。这种方法违背了Host Requirements RFC（在这个代码中的一个注释确认这是一个错误）。这个变量每0.5秒增加64000，并每隔9.5小时又回到0（对应这个计数器每8 ms加1，而不是每4 ms加1）。另外，每次建立一个连接后，这个变量将增加64000。

四次挥手：

> - 第一次：客户机发含FIN位，SEQ=Q的包到服务器。（客->FIN_WAIT_1）
> - 第二次：服务器发送含ACK且ACK_NUM=Q+1的包到服务器。（服->CLOSE_WAIT，客->FIN_WAIT_2）此处有等待
> - 第三次：服务器发送含FIN且SEQ_NUM=R的包到客户机。（服->LAST_ACK，客->TIME_WAIT）此处有等待
> - 第四次：客户机发送最后一个含有ACK位且ACK_NUM=R+1的包到客户机。（服->CLOSED）

为什么握手是三次，挥手是四次?

> 对于握手：握手只需要确认双方通信时的初始化序号，保证通信不会乱序。（第三次握手必要性：假设服务端的确认丢失，连接并未断开，客户机超时发连接请求，这样服务器会对同一个客户机保持多个连接，造成资源浪费。）
>
> 对于挥手：TCP是双工的，所以发送方和接收方都需要FIN和ACK。只不过有一方是被动的，所以看上去就成了4次挥手。同时当一方发送FIN后，另外一方可能还有数据要发送，只有等到另外一方也把数据发送完毕的情况下才能发送ACK

TCP连接状态?

> - CLOSED：初始状态。
> - LISTEN：服务器处于监听状态。

- SYN_SEND：客户端socket执行CONNECT连接，发送SYN包，进入此状态。
- SYN_RECV：服务端收到SYN包并发送服务端SYN包，进入此状态。
- ESTABLISH：表示连接建立。客户端发送了最后一个ACK包后进入此状态，服务端接收到ACK包后进入此状态。
- FIN_WAIT_1：终止连接的一方（通常是客户机）发送了FIN报文后进入。等待对方FIN。
- CLOSE_WAIT：（假设服务器）接收到客户机FIN包之后等待关闭的阶段。在接收到对方的FIN包之后，自然是需要立即回复ACK包的，表示已经知道断开请求。但是本方是否立即断开连接（发送FIN包）取决于是否还有数据需要发送给客户端，若有，则在发送FIN包之前均为此状态。
- FIN_WAIT_2：此时是半连接状态，即有一方要求关闭连接，等待另一方关闭。客户端接收到服务器的ACK包，但并没有立即接收到服务端的FIN包，进入FIN_WAIT_2状态。
- LAST_ACK：服务端发动最后的FIN包，等待最后的客户端ACK响应，进入此状态。
- TIME_WAIT：客户端收到服务端的FIN包，并立即发出ACK包做最后的确认，在此之后的2MSL时间称为TIME_WAIT状态。

解释FIN_WAIT_2，CLOSE_WAIT状态和TIME_WAIT状态

- FIN_WAIT_2：
  - 半关闭状态。
  - 发送断开请求一方还有接收数据能力，但已经没有发送数据能力。
- CLOSE_WAIT状态：
  - 被动关闭连接一方接收到FIN包会立即回应ACK包表示已接收到断开请求。
  - 被动关闭连接一方如果还有剩余数据要发送就会进入CLOSED_WAIT状态。
- TIME_WAIT状态：
  - 又叫2MSL等待状态。
  - 如果客户端直接进入CLOSED状态，如果服务端没有接收到最后一次ACK包会在超时之后□新再发FIN包，此时因为客户端已经CLOSED，所以服务端就不会收到ACK而是收到RST。所以TIME_WAIT状态目的是防止最后一次握手数据没有到达对方而触发□传FIN准备的。
  - 在2MSL时间内，同一个socket不能再被使用，否则有可能会和旧连接数据混淆（如果新连接和旧连接的socket相同的话）。

拥塞控制原理

> 拥塞控制目的是防止数据被过多注网络中导致网络资源（路由器、交换机等）过载。因为拥塞控制涉及网络链路全局，所以属于全局控制。控制拥塞使用拥塞窗口。
>
> TCP拥塞控制算法：
>
> - 慢开始&拥塞避免：先试探网络拥塞程度再逐渐增大拥塞窗口。每次收到确认后拥塞窗口翻倍，直到达到阀值ssthresh，这部分是慢开始过程。达到阀值后每次以一个MSS为单位增长拥塞窗口大小，当发生拥塞（超时未收到确认），将阀值减为原先一半，继续执行线性增加，这个过程为拥塞避免。
> - 快速重传&快速恢复：略。
> - 最终拥塞窗口会收敛于稳定值。

拥塞控制

>

流量控制

>

TCP如何提供可靠数据传输的

> 建立连接（标志位）：通信前确认通信实体存在。
>
> 序号机制（序号、确认号）：确保了数据是按序、完整到达。
>
> 数据校验（校验和）：CRC校验全部数据。超时重传（定时器）：保证因链路故障未能到达数据能够被多次重发。
>
> 窗口机制（窗口）：提供流量控制，避免过量发送。
>
> 拥塞控制：同上

应用层

协议：

> # DNS:
>
> **作用**
>
> - 域名向IP地址的翻译
> - 主机别名
> - 邮件服务器别名
> - 负载均衡

为什么不使用集中式的DNS

- 单点失败问题
- 距离问题
- 流量问题
- 维护性问题

DNS采用分层次是数据库

本地域名服务器

- 作为代理将查询转发给层级式域名解析服务器系统

DNS的两种方式

1. 迭代查询：本地域名服务器想根域名服务器，然后返回顶级域名，如何查询顶级域名。。。一次查询
2. 递归查询：本地域名询问根域名服务器，然后跟域名查询顶级域名，依次查询，最后得到结果

DNS采用什么协议

DNS名字服务器使用的熟知端口号无论对UDP还是TCP都是53。这意味着DNS均支持UDP和TCP访问，但我们使用tcpdump观察的所有例子都是采用UDP。

什么情况下用tcp?

> -当名字解析器发出一个查询请求，并且返回响应中的TC（删减标志）比特被设置为1时，
> -当一个域的辅助名字服务器在启动时，将从该域的主名字服务器执行区域传送。我们也说过辅助服务器将定时（通常是3小时）向主服务器进行查询以便了解主服务器数据是否发生变动。如果有变动，将执行一次区域传送。区域传送将使用TCP，因为这里传送的数据远比一个查询或响应多得多。

DNS缓存

- 一般来说跟域名服务器很少被使用

协议格式：查询-回复

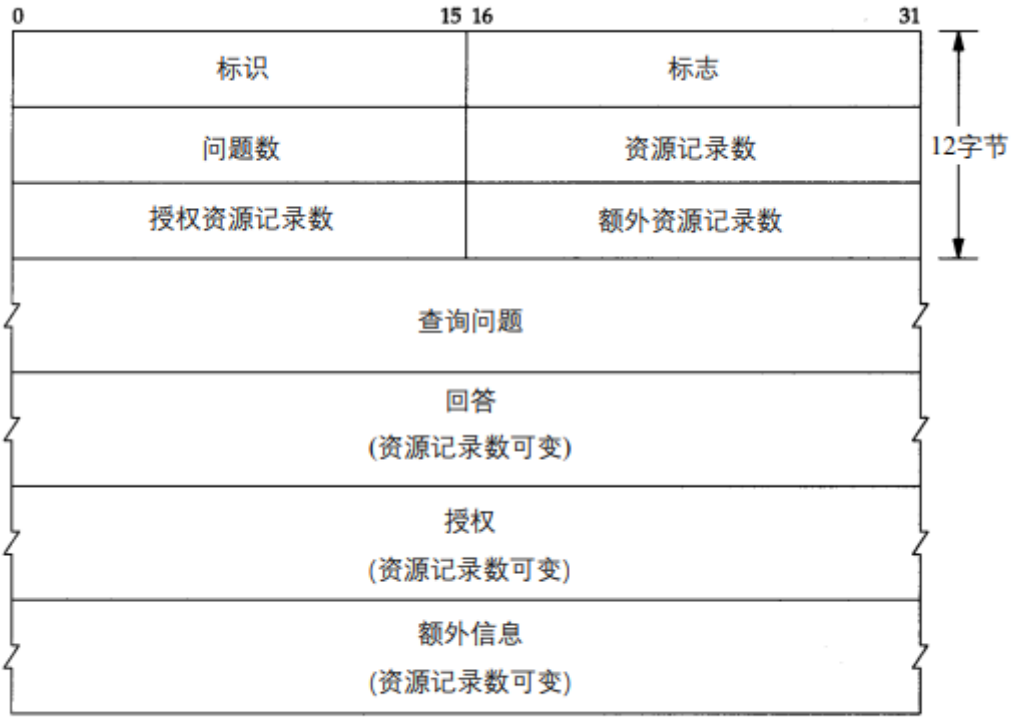

图14-3 DNS查询和响应的一般格式

HTTP:

http请求和响应

请求格式:

> 请求方法路径协议\n\r
>
> 请求头
>
> 请求体

```
▼ Request Headers      view source
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
  Accept-Encoding: gzip, deflate
  Accept-Language: zh-CN,zh;q=0.9
  Cache-Control: max-age=0
  Connection: keep-alive
  Cookie: PHPSESSID=4obtrhm8114nr93nuo6on7h23s
  Host: oj.ecustacm.cn
  Upgrade-Insecure-Requests: 1
  User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.100 Safari/537.36
```

> ### 响应格式:
>
> 协议状态码状态短语\n\r
>
> 响应头

> 响应体

```
▼Response Headers    view source
  Cache-Control: no-store, no-cache, must-revalidate
  Connection: keep-alive
  Content-Encoding: gzip
  Content-Type: text/html; charset=UTF-8
  Date: Wed, 16 Mar 2022 12:20:36 GMT
  Expires: Thu, 19 Nov 1981 08:52:00 GMT
  Pragma: no-cache
  Server: nginx/1.14.0 (Ubuntu)
  Transfer-Encoding: chunked
```

http请求方法

> GET主要用于请求获取资源请求参数一般要求不能超过256个字节

> POST主要用于传输实体的主题，没有长度限制，比GET请求安全一些

> PUT用于传输文件，但是因为在HTTP/1.1版本存在安全问题，因此一般web不使用该方法

> HEAD 获取报文首部信息，和GET请求一样

> DELETE主要用于删除文件与GET一样一般被禁用

> OPTIONS询问支持的方法

> CONNECT隧道协议连接代理采用SSL和TLS吧协议通信加密后传输

状态码

表 4-1：状态码的类别

	类别	原因短语
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

HTTP的缺点

> 1、无状态

> 2、明文传输

> 3、无法证明数据的完整性，容易被篡改

> 4、不验证通信身份，可能被伪装劫持

HTTPS

> HTTPS实质是http协议+ssl/stl

> 加密相关：

- 共享秘钥加密：加密方法公开，加密秘钥由通信双方共享，如果被攻击者获取，那加密失去了意义。(对称加密)
- 公开秘钥加密：有一个私钥和公钥，公钥随意发布，私钥只有自己知道，采用公钥加密私钥解密。(非对称加密)

> HTTPS采用共享秘钥和公开秘钥进行混用的方式加密。

- 在交换秘钥环境采用了公开密钥加密的方式
- 建立通信后使用共享秘钥进行通信
- CA证书：利用这一证书的数字签名来保证网站的真实性，并防止公钥中途被替换

> HTTPS的通信过程：

> HTTPS的中间人攻击：

> **标准回答**

> 中间人攻击是指攻击者通过与客户端和客户端的目标服务器同时建立连接，作为客户端和服务器的桥梁，处理双方的数据，整个会话期间的内容几乎是完全被攻击者控制的。攻击者可以拦截双方的会话并且插入新的数据内容。

> **加分回答**

> 中间人攻击的过程：

1. 服务器向客户端发送公钥。
2. 攻击者截获公钥，保留在自己手上。
3. 然后攻击者自己生成一个伪造的公钥，发给客户端。
4. 客户端收到伪造的公钥后，生成加密哈希（此时加密内容是对称加解密秘钥）值发给服务器。
5. 攻击者获得加密哈希值，用自己的私钥解密获得真秘钥。
6. 同时生成假的加密哈希值，发给服务器。
7. 服务器用私钥解密获得假秘钥。
8. 服务器用假秘钥加密传输信息。

> ## SMTP:

1、tcp黏包问题··

>

2、当我们输入一个URL发生了什么

>

3、网站渲染

>

操作系统

什么是进程

> 1. 进程是指在系统中正在运行的一个应用程序，程序一旦运行就是进程；
> 2. 进程可以认为是程序执行的一个实例，进程是系统进行资源分配的最小单位，且每个进程拥有独立的地址空间；
> 3. 一个进程无法直接访问另一个进程的变量和数据结构，如果希望一个进程去访问另一个进程的资源，需要使用进程间的通信，比如：管道、消息队列等
> 4. 线程是进程的一个实体，是进程的一条执行路径；比进程更小的独立运行的基本单位，线程也被称为轻量级进程，一个程序至少有一个进程，一个进程至少有一个线程；

进程调度算法

> 先来先服务调度算法
>
> 短作业优先调度算法
>
> 非抢占式优先级调度算法
>
> 抢占式优先级调度算法
>
> 高响应比优先调度算法
>
> 时间片轮转法调度算法

进程与线程的区别

> 1. 同一进程的线程共享本进程的地址空间，而进程之间则是独立的地址空间
> 2. 同一进程内的线程共享本进程的资源，但是进程之间的资源是独立的

3. 一个进程崩溃后，在保护模式下不会对其他进程产生影响，但是一个线程崩溃整个进程崩溃，所以多进程比多线程健壮；
4. 进程切换，消耗的资源大。所以涉及到频繁的切换，使用线程要好于进程；
5. 两者均可并发执行
6. 每个独立的进程有一个程序的入口、程序出口。但是线程不能独立执行，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

死锁是什么？必要条件？如何解决

> 所谓死锁，是指多个进程循环等待它方占有的资源而无限期地僵持下去的局面。很显然，如果没有外力的作用，那麽死锁涉及到的各个进程都将永远处于封锁状态。当两个或两个以上的进程同时对多个互斥资源提出使用要求时，有可能导致死锁.
>
> 必要条件：
>
> 〈1〉 互斥条件。即某个资源在一段时间内只能由一个进程占有，不能同时被两个或两个以上的进程占有。这种独占资源如CD-ROM驱动器，打印机等等，必须在占有该资源的进程主动释放它之后，其它进程才能占有该资源。这是由资源本身的属性所决定的。如独木桥就是一种独占资源，两方的人不能同时过桥。
>
> 〈2〉 不可抢占条件。进程所获得的资源在未使用完毕之前，资源申请者不能强行地从资源占有者手中夺取资源，而只能由该资源的占有者进程自行释放。如过独木桥的人不能强迫对方后退，也不能非法地将对方推下桥，必须是桥上的人自己过桥后空出桥面（即主动释放占有资源），对方的人才能过桥。
>
> 〈3〉 占有且申请条件。进程至少已经占有一个资源，但又申请新的资源；由于该资源已被另外进程占有，此时该进程阻塞；但是，它在等待新资源之时，仍继续占用已占有的资源。还以过独木桥为例，甲乙两人在桥上相遇。甲走过一段桥面（即占有了一些资源），还需要走其余的桥面（申请新的资源），但那部分桥面被乙占有（乙走过一段桥面）。甲过不去，前进不能，又不后退；乙也处于同样的状况。
>
> 〈4〉 循环等待条件。存在一个进程等待序列{P1，P2，...，Pn}，其中P1等待P2所占有的某一资源，P2等待P3所占有的某一源，.....，而Pn等待P1所占有的的某一资源，形成一个进程循环等待环。就像前面的过独木桥问题，甲等待乙占有的桥面，而乙又等待甲占有的桥面，从而彼此循环等待。

什么时候用多进程？什么时候用多线程

1. 需要频繁创建销毁的优先用线程；
2. 需要进行大量计算的优先使用线程；
3. 强相关的处理用线程，弱相关的处理用进程；
4. 可能要扩展到多机分布的用进程，多核分布的用线程

协程是什么

1. 是一种比线程更加轻□级的存在。正如一个进程可以拥有多个线程一样，一个线程可以拥有多个协程；协程不是被操作系统内核管理，而完全是由程序所控制。
2. 协程的开销远远小于线程；
3. 协程拥有自己寄存器上下文和栈。协程调度切换时，将寄存器上下文和栈保存到其他地方，在切换回来的时候，恢复先前保存的寄存器上下文和栈.
4. 每个协程表示一个执行单元，有自己的本地数据，与其他协程共享全局数据和其他资源。

5. 跨平台、跨体系架构、无需线程上下文切换的开销、方便切换控制流，简化编程模型；
6. 协程又称为微线程，协程的完成主要靠yeild关键字，协程执行过程中，在子程序内部可中断，然后转而执行别的子程序，在适当的时候再返回来接着执行；
7. 协程极高的执行效率，和多线程相比，线程数越多，协程的性能优势就越明显
8. 不需要多线程的锁机制

计算机组成原理

堆和栈的区别?

> 1、栈是从上往下生长的，而堆是从下往上生长的
>
> 2、栈由系统自动分配，而堆是人为申请开辟
>
> 3、栈是连续的空间，而堆是不一定连续。
>
> 4、栈获得的空间较小，而堆获得的空间较大

数据库

MySQL

面经链接：https://www.nowcoder.com/discuss/tiny/837435?channel=666&source_id=feed_index_nctrack&fromIframe=true

常见的存储引擎：MYISAM、InnoDB、MEMORY

MYISAM和InnoDB的区别

对比项	MyISAM	InnoDB
主外键	不支持	支持
事物	不支持	支持
行表锁	表锁，即使操作一条记录也会锁住整个表，不适合高并发	行锁，操作时锁住某一行，适合高并发
缓存	只缓存索引，不缓存真实数据	既缓存索引，又缓存真实数据
表空间	小	大
关注点	性能	事物

MYSQL事物

> 事物的特性：ACID
>
> 原子性：是指事务包含的所有操作要么全部成功，要么全部失败回滚。
>
> 一致性：是指一个事务执行之前和执行之后都必须处于一致性状态。
>
> 隔离性：跟隔离级别相关，如`readcommitted`，一个事务只能读到已经提交的修改。
>
> 持久性：是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

> 脏读：**脏读**是指在一个事务处理过程里读取了另一个未提交的事务中的数据。
>
> 不可重复度：**不可重复读**是指在对于数据库中的某行记录，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，另一个事务修改了数据并提交了。

> 幻读：**幻读**是当某个事务在读取某个范围内的记录时，另外一个事务又在该范围内插入了新的记录，当之前的事务再次读取该范围的记录时，会产生幻行，就像产生幻觉一样，这就是发生了幻读。

数据库的隔离级别：

> 读未提交：所有事务都可以看到其他未提交事务的执行结果。无法避免脏读、幻读、不可重复读
>
> 读已提交：一个事务只能看见已经提交事务所做的改变。可避免脏读的发生。
>
> 可重复读：MySQL的默认事务隔离级别，它确保同一事务的多个实例在并发读取数据时，会看到同样的数据行，解决了不可重复读的问题。
>
> 串行化：通过强制事务排序，使之不可能相互冲突，从而解决幻读问题。

性能下降慢SQL

> 查询语句写的烂
>
> 索引失效
>
> 关联查询太多
>
> 服务器调优及各个参数设置（缓存，线程数等）

索引

- 什么是索引

  > 索引是存储引擎用于提高数据库表的访问速度的一种**数据结构**。

- 索引的优缺点?

  > 优点：
  >
  > - **加快数据查找的速度**
  > - 为用来排序或者是分组的字段添加索引，可以加快分组和排序的速度
  > - 加快表与表之间的连接
  >
  > 缺点：
  >
  > - 建立索引需要**占用物理空间**
  > - 会降低表的增删改的效率，因为每次对表记录进行增删改，需要进行**动态维护索引**，导致增删改时间变长

- 索引的分类

  > 单值索引：即一个索引值包含单个例，一个表可以有多个单例索引
  >
  > 唯一索引：索引列的值必须唯一，但允许空值

> 复合索引：即一个索引包含多个列
>
> 基本语法：create[unique]indexindexNameONmytable(列名);

- 索引结构

> B+树索引：
>
> 全文索引：
>
> hash索引（MEMEROY）
>
> R-Tree索引：

- 什么情况下要建立索引

> 主键自动建立唯一索引
>
> 频繁作为查询条件的字段应该创建索引
>
> 查询中与其他表关联的字段，外键关系建立索引
>
> 频繁更新的字段不适合创建索引
>
> where条件里用不到的字段不创建索引
>
> 单键/组合索引的选择问题（高并发下倾向创建组合索引）
>
> 查询中排序的字段
>
> 查询中统计或分组字段

- 什么情况下不要建索引

> - 表记录太少
> - 经常增删改的表
> - 数据重复且分布平均的表字段

-

Redis

面经链接：https://baijiahao.baidu.com/s?id=1718316080906441023&wfr=spider&for=pc&searchword=每月反馈用redis做&qq-pf-to=pcqq.group

中间件