**C++基础语法**

**1.new、delete、malloc、free关系**

delete会调用对象的析构函数,和new对应free只会释放内存，new调用构造函数。malloc与free是C++/C语言的标准库函数，new/delete是C++的运算符。它们都可用于申请动态内存和释放内存。对于非内部数据类型的对象而言，光用maloc/free无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于malloc/free是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于malloc/free。因此C++语言需要一个能完成动态内存分配和初始化工作的运算符new，以及一个能完成清理与释放内存工作的运算符delete。注意new/delete不是库函数。

**2.delete与 delete []区别**

delete只会调用一次析构函数，而delete[]会调用每一个成员的析构函数。在More Effective C++中有更为详细的解释：“当delete操作符用于数组时，它为每个数组元素调用析构函数，然后调用operator delete来释放内存。”delete与new配套，delete []与new []配套

MemTest \*mTest1=new MemTest[10];

MemTest \*mTest2=new MemTest;

Int \*pInt1=new int [10];

Int \*pInt2=new int;

delete[]pInt1; //-1-

delete[]pInt2; //-2-

delete[]mTest1;//-3-

delete[]mTest2;//-4-

在-4-处报错。

这就说明：对于内建简单数据类型，delete和delete[]功能是相同的。对于自定义的复杂数据类型，delete和delete[]不能互用。delete[]删除一个数组，delete删除一个指针。简单来说，用new分配的内存用delete删除；用new[]分配的内存用delete[]删除。delete[]会调用数组元素的析构函数。内部数据类型没有析构函数，所以问题不大。如果你在用delete时没用括号，delete就会认为指向的是单个对象，否则，它就会认为指向的是一个数组。

**3.子类析构时要调用父类的析构函数吗？**

析构函数调用的次序是先派生类的析构后基类的析构，也就是说在基类的的析构调用的时候,派生类的信息已经全部销毁了。定义一个对象时先调用基类的构造函数、然后调用派生类的构造函数；析构的时候恰好相反：先调用派生类的析构函数、然后调用基类的析构函数。

**4.简单介绍：多态，虚函数，纯虚函数**

多态：是对于不同对象接收相同消息时产生不同的动作。C++的多态性具体体现在运行和编译两个方面：在程序运行时的多态性通过继承和虚函数来体现；

在程序编译时多态性体现在函数和运算符的重载上；

虚函数：在基类中冠以关键字 virtual 的成员函数。 它提供了一种接口界面。允许在派生类中对基类的虚函数重新定义。

纯虚函数的作用：在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。作为接口而存在 纯虚函数不具备函数的功能，一般不能直接被调用。

从基类继承来的纯虚函数，在派生类中仍是虚函数。如果一个类中至少有一个纯虚函数，那么这个类被称为抽象类（abstract class）。

抽象类中不仅包括纯虚函数，也可包括虚函数。抽象类必须用作派生其他类的基类，而不能用于直接创建对象实例。但仍可使用指向抽象类的指针支持运行时多态性。

**5.什么是“引用”？声明和使用“引用”要注意哪些问题？**

答：引用就是某个目标变量的“别名”(alias)，对应用的操作与对变量直接操作效果完全相同。声明一个引用的时候，切记要对其进行初始化。引用声明完毕后，相当于目标变量名有两个名称，即该目标原名称和引用名，不能再把该引用名作为其他变量名的别名。声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，它本身不是一种数据类型，因此引用本身不占存储单元，系统也不给引用分配存储单元。不能建立数组的引用。

**6.将“引用”作为函数参数有哪些特点？**

（1）传递引用给函数与传递指针的效果是一样的。这时，被调函数的形参就成为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

（2）使用引用传递函数的参数，在内存中并没有产生实参的副本，它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时，需要给形参分配存储单元，形参变量是实参变量的副本；如果传递的是对象，还将调用拷贝构造函数。因此，当参数传递的数据较大时，用引用比用一般变量传递参数的效率和所占空间都好。

（3）使用指针作为函数的参数虽然也能达到与使用引用的效果，但是，在被调函数中同样要给形参分配存储单元，且需要重复使用"\*指针变量名"的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处，必须用变量的地址作为实参。而引用更容易使用，更清晰。

**7.在什么时候需要使用“常引用”？**

如果既要利用引用提高程序的效率，又要保护传递给函数的数据不在函数中被改变，就应使用常引用。常引用声明方式：const 类型标识符 &引用名=目标变量名；

例1

int a ;

const int &ra=a;

ra=1; //错误

a=1; //正确

例2

string foo( );

void bar(string & s);

那么下面的表达式将是非法的：

bar(foo( ));

bar("hello world");

原因在于foo( )和"hello world"串都会产生一个临时对象，而在C++中，这些临时对象都是const类型的。因此上面的表达式就是试图将一个const类型的对象转换为非const类型，这是非法的。引用型参数应该在能被定义为const的情况下，尽量定义为const 。

**8.将“引用”作为函数返回值类型的格式、好处和需要遵守的规则?**

格式：类型标识符 &函数名（形参列表及类型说明）{ //函数体 }

好处：在内存中不产生被返回值的副本；（注意：正是因为这点原因，所以返回一个局部变量的引用是不可取的。因为随着该局部变量生存期的结束，相应的引用也会失效，产生runtime error!

注意事项：

（1）不能返回局部变量的引用。这条可以参照Effective C++[1]的Item 31。主要原因是局部变量会在函数返回后被销毁，因此被返回的引用就成为了"无所指"的引用，程序会进入未知状态。

（2）不能返回函数内部new分配的内存的引用。这条可以参照Effective C++[1]的Item 31。虽然不存在局部变量的被动销毁问题，可对于这种情况（返回函数内部new分配内存的引用），又面临其它尴尬局面。例如，被函数返回的引用只是作为一个临时变量出现，而没有被赋予一个实际的变量，那么这个引用所指向的空间（由new分配）就无法释放，造成memory leak。

（3）可以返回类成员的引用，但最好是const。这条原则可以参照Effective C++[1]的Item 30。主要原因是当对象的属性是与某种业务规则（business rule）相关联的时候，其赋值常常与某些其它属性或者对象的状态有关，因此有必要将赋值操作封装在一个业务规则当中。如果其它对象可以获得该属性的非常量引用（或指针），那么对该属性的单纯赋值就会破坏业务规则的完整性。

（4）流操作符重载返回值申明为“引用”的作用：

流操作符<<和>>，这两个操作符常常希望被连续使用，例如：cout << "hello" << endl;　因此这两个操作符的返回值应该是一个仍然支持这两个操作符的流引用。可选的其它方案包括：返回一个流对象和返回一个流对象指针。但是对于返回一个流对象，程序必须重新（拷贝）构造一个新的流对象，也就是说，连续的两个<<操作符实际上是针对不同对象的！这无法让人接受。对于返回一个流指针则不能连续使用<<操作符。因此，返回一个流对象引用是惟一选择。这个唯一选择很关键，它说明了引用的重要性以及无可替代性，也许这就是C++语言中引入引用这个概念的原因吧。

赋值操作符=。这个操作符象流操作符一样，是可以连续使用的，例如：x = j = 10;或者(x=10)=100;赋值操作符的返回值必须是一个左值，以便可以被继续赋值。因此引用成了这个操作符的惟一返回值选择。

＃include<iostream.h>

int &put(int n);

int vals[10];

int error=-1;

void main()

{

put(0)=10; //以put(0)函数值作为左值，等价于vals[0]=10;

put(9)=20; //以put(9)函数值作为左值，等价于vals[9]=20;

cout<<vals[0];

cout<<vals[9];

}

int &put(int n)

{

if (n>=0 && n<=9 ) return vals[n];

else { cout<<"subscript error"; return error; }

}

（5）在另外的一些操作符中，却千万不能返回引用：+-\*/ 四则运算符。它们不能返回引用，Effective C++[1]的Item23详细的讨论了这个问题。主要原因是这四个操作符没有side effect，因此，它们必须构造一个对象作为返回值，可选的方案包括：返回一个对象、返回一个局部变量的引用，返回一个new分配的对象的引用、返回一个静态对象引用。根据前面提到的引用作为返回值的三个规则，2、3两个方案都被否决了。静态对象的引用又因为((a+b) == (c+d))会永远为true而导致错误。所以可选的只剩下返回一个对象了。

**9、结构与联合有和区别？**

(1). 结构和联合都是由多个不同的数据类型成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员（所有成员共用一块地址空间）, 而结构的所有成员都存在（不同成员的存放地址不同）。

(2). 对于联合的不同成员赋值, 将会对其它成员重写, 原来成员的值就不存在了, 而对于结构的不同成员赋值是互不影响的。

**10.重载（overload)和重写(overried，有的书也叫做“覆盖”）的区别？**

常考的题目。从定义上来说：

重载：是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。

重写：是指子类重新定义父类虚函数的方法。

从实现原理上来说：

重载：编译器根据函数不同的参数表，对同名函数的名称做修饰，然后这些同名函数就成了不同的函数（至少对于编译器来说是这样的）。如，有两个同名函数：function func(p:integer):integer;和function func(p:string):integer;。那么编译器做过修饰后的函数名称可能是这样的：int\_func、str\_func。对于这两个函数的调用，在编译器间就已经确定了，是静态的。也就是说，它们的地址在编译期就绑定了（早绑定），因此，重载和多态无关！

重写：和多态真正相关。当子类重新定义了父类的虚函数后，父类指针根据赋给它的不同的子类指针，动态的调用属于子类的该函数，这样的函数调用在编译期间是无法确定的（调用的子类的虚函数的地址无法给出）。因此，这样的函数地址是在运行期绑定的（晚绑定）。

**11.有哪几种情况只能用intialization list 而不能用assignment?**

答案：当类中含有const、reference 成员变量；基类的构造函数都需要初始化表。

**12. main 函数执行以前，还会执行什么代码？**

答案：全局对象的构造函数会在main 函数之前执行。

**13. 描述内存分配方式以及它们的区别?**

1） 从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2） 在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。

3） 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

**14.简述数组与指针的区别？**

数组要么在静态存储区被创建（如全局数组），要么在栈上被创建。指针可以随时指向任意类型的内存块。

(1)修改内容上的差别

char a[] = “hello”;

a[0] = ‘X’;

char \*p = “world”; // 注意p 指向常量字符串

p[0] = ‘X’; // 编译器不能发现该错误，运行时错误

(2) 用运算符sizeof 可以计算出数组的容量（字节数）。sizeof(p),p 为指针得到的是一个指针变量的字节数，而不是p 所指的内存容量。C++/C 语言没有办法知道指针所指的内存容量，除非在申请内存时记住它。注意当数组作为函数的参数进行传递时，该数组自动退化为同类型的指针。

char a[] = "hello world";

char \*p = a;

cout<< sizeof(a) << endl; // 12 字节

cout<< sizeof(p) << endl; // 4 字节

计算数组和指针的内存容量

void Func(char a[100])

{

cout<< sizeof(a) << endl; // 4 字节而不是100 字节

}

**15题：将程序跳转到指定内存地址**

要对绝对地址0x100000赋值，我们可以用(unsigned int\*)0x100000 = 1234;那么要是想让程序跳转到绝对地址是0x100000去执行，应该怎么做？

　　\*((void (\*)( ))0x100000 ) ( );  
　　首先要将0x100000强制转换成函数指针,即:  
　　(void (\*)())0x100000  
　　然后再调用它:  
　　\*((void (\*)())0x100000)();  
　　用typedef可以看得更直观些:  
　　typedef void(\*)() voidFuncPtr;  
　　\*((voidFuncPtr)0x100000)();

**16题：引用与指针有什么区别？**

【参考答案】                           
1) 引用必须被初始化，指针不必。

2) 引用初始化以后不能被改变，指针可以改变所指的对象。

3) 不存在指向空值的引用，但是存在指向空值的指针。

**17题：const  与 #define 的比较 ，const有什么优点?**

 【参考答案】

（1） const 常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查。而对后者只进行字符替换，没有类型安全检查，并且在字符替换可能会产生意料不到的错误（边际效应） 。

（2）  有些集成化的调试工具可以对 const 常量进行调试，但是不能对宏常量进行调试。

**18题：内存的分配方式有几种?**

【参考答案】

一、从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量。

二、在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集中，效率很高，但是分配的内存容量有限。

三、从堆上分配，亦称动态内存分配。程序在运行的时候用malloc或new申请任意多少的内存，程序员自己负责在何时用free或delete释放内存。动态内存的生存期由我们决定，使用非常灵活，但问题也最多。

**19题：基类的析构函数不是虚函数，会带来什么问题？**

【参考答案】派生类的析构函数用不上，会造成资源的泄漏。

**20题：全局变量和局部变量有什么区别？是怎么实现的？操作系统和编译器是怎么知道的？**

【参考答案】

生命周期不同：

全局变量随主程序创建和创建，随主程序销毁而销毁；局部变量在局部函数内部，甚至局部循环体等内部存在，退出就不存在；

使用方式不同：通过声明后全局变量程序的各个部分都可以用到；局部变量只能在局部使用；分配在栈区。

操作系统和编译器通过内存分配的位置来知道的，全局变量分配在全局数据段并且在程序开始运行的时候被加载。局部变量则分配在堆栈里面 。

**21题：C++函数中值的传递方式有哪几种?**

答：三种传递方式为：值传递、指针传递和引用传递。

C++基础2

**1.为什么说，避免在构造函数和析构函数调用虚函数？**

虚函数仅仅“几乎”总是表现得虚拟：但在构造函数和析构函数中，他们并不虚拟。更糟糕的是，从构造函数或析构函数直接或者间接调用未实现的纯虚函数，会导致未定义的行为。

**2.在C++ 程序中调用被 C 编译器编译后的函数，为什么要加 extern “C”声明？**

解析：

函数和变量被C++编译后在符号库中的名字与C语言的不同，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。由于编译后的名字不同，C++程序不能直接调用C 函数。C++提供了一个C 连接交换指定符号extern“C”来解决这个问题。

**3、内联函数inline和宏定义一起使用的区别。**

解析：

内联函数是在编译的时候已经做好将对应的函数代码替换嵌入到对应的位置，适用于代码较少的函数。 宏定义是简单的替换变量，如果定义的是有参数的函数形式，参数不做类型校验。

**4、 关键字static的作用是什么？**

正确答案：

这个简单的问题很少有人能回答完全。在C语言中，关键字static有三个明显的作用：

1). 在函数体，一个被声明为静态的变量在这一函数被调用过程中维持其值不变。

2). 在模块内（但在函数体外），一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其它函数访问。它是一个本地的全局变量。

3). 在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用。 大多数应试者能正确回答第一部分，一部分能正确回答第二部分，同是很少的人能懂得第三部分。这是一个应试者的严重的缺点，因为他显然不懂得本地化数 据和代码范围的好处和重要性。

**5、C语言的volatile的含义是什么。使用时会对编译器有什么暗示。**

正确答案：

终于最后一题了，容易么……如果这个测试是一个关于嵌入式的，那么这道题非常重要！！从词面上讲，volatile的意思是易变的，也就是说，在程序运行过程中，有一些变量可能会被莫名其妙的改变，而优化器为了节约时间，有时候不会重读这个变量的真实值，而是去读在寄存器的备份，这样的话，这个变量的真实值反而被优化器给“优化”掉了，用时髦的词说就是被“和谐”了。如果使用了这个修饰词，就是通知编译器别犯懒，老老实实去重新读一遍！可能我说的太“通俗”了，那么我引用一下“大师”的标准解释： volatile的本意是“易变的” 。 由于访问寄存器的速度要快过RAM,所以编译器一般都会作减少存取外部RAM的优化，但有可能会读脏数据。当要求使用volatile 声明的变量的值的时候，系统总是重新从它所在的内存读取数据，即使它前面的指令刚刚从该处读取过数据。而且读取的数据立刻被保存。 精确地说就是，优化器在用到这个变量时必须每次都小心地重新读取这个变量的值，而不是使用保存在寄存器里的备份。 下面是volatile变量的几个例子：

1). 并行设备的硬件寄存器（如：状态寄存器）

2). 一个中断服务子程序中会访问到的非自动变量(Non-automatic variables)

3). 多线程应用中被几个任务共享的变量 嵌入式系统程序员经常同硬件、中断、RTOS等等打交道，所用这些都要求volatile变量。不懂得volatile内容将会带来灾难。

**6、类使用static成员的优点，如何访问？**

答：优点：

（1）static 成员的名字是在类的作用域中，因此可以避免与其他类的成员或全局对象名字冲突；

（2）可以实施封装。static 成员可以是私有成员，而全局对象不可以；

（3） static 成员是与特定类关联的，可清晰地显示程序员的意图。 static 数据成员必须在类定义体的外部定义(正好一次)，static 关键字只能用于类定义体内部的声明中，定义不能标示为static. 不像普通数据成员，static成员不是通过类构造函数进行初始化，也不能在类的声明中初始化，而是应该在定义时进行初始化.保证对象正好定义一次的最好办法，就是将static 数据成员的定义放在包含类非内联成员函数定义的文件中。 静态数据成员初始化的格式为： ＜数据类型＞＜类名＞::＜静态数据成员名＞=＜值＞ 类的静态数据成员有两种访问形式： ＜类对象名＞.＜静态数据成员名＞ 或 ＜类类型名＞::＜静态数据成员名＞

**7、static数据成员和static成员函数**

答：（1）static数据成员： static数据成员独立于该类的任意对象而存在；每个static数据成员是与类关联的对象，并不与该类的对象相关联。Static数据成员（const static数据成员除外）必须在类定义体的外部定义。不像普通数据成员，static成员不是通过类的构造函数进行初始化，而是应该在定义时进行初始化。 （2）static成员函数： Static成员函数没有this形参，它可以直接访问所属类的static成员，不能直接使用非static成员。因为static成员不是任何对象的组成部分，所以static成员不能被声明为const。同时，static成员函数也不能被声明为虚函数。

**8、static成员变量定义放在cpp文件中，不能放在初始化列表中。Const static成员可就地初始化。**

**9、如何引用一个已经定义过的全局变量？**

答：可以用引用头文件的方式，也可以用extern关键字，如果用引用头文件方式来引用某个在头文件中声明的全局变量，假定你将那个变量写错了，那么在编译期间会报错，如果你用extern方式引用时，假定你犯了同样的错误，那么在编译期间不会报错，而在连接期间报错。

**10、多态类中的虚函数表是 Compile-Time，还是 Run-Time时建立的?**

答案：虚拟函数表是在编译期就建立了,各个虚拟函数这时被组织成了一个虚拟函数的入口地址的数组。而对象的隐藏成员--虚拟函数表指针是在运行期--也就是构造函数被调用时进行初始化的，这是实现多态的关键。

**11、一个父类写了一个 virtual 函数，如果子类覆盖它的函数不加 virtual ,也能实现多态?**

在子类的空间里，有没有父类的这个函数，或者父类的私有变量? (华为笔试题） 答案：只要基类在定义成员函数时已经声明了 virtue关键字，在派生类实现的时候覆盖该函数时，virtue关键字可加可不加，不影响多态的实现。子类的空间里有父类的所有变量(static除外)。

操作系统相关

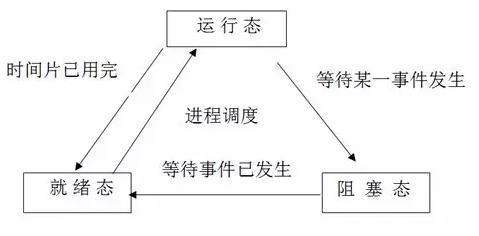
**1.进程与线程的区别**

（1）粒度性分析：线程的粒度小于进程。

（2）调度性分析：进程是资源拥有的基本单位，线程是独立调度与独立运行的基本单位，出了寄存器，程序计数器等必要的资源外基本不拥有其他资源。

（3）系统开销分析：由于线程基本不拥有系统资源，所以在进行切换时，线程切换的开销远远小于进程。

**2.进程的状态及其转换**



**3.进程间的通信方式有哪些？**

（1） 管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

（2）有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

（3）信号量( semophore ) ： 信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

（4） 消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

（5）信号 ( sinal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

（6）共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号两，配合使用，来实现进程间的同步和通信。c++学习资料分享群615968007

（7）套接字( socket ) ： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

**4、ICMP是什么协议,处于哪一层?**

正确答案：

Internet控制报文协议，处于网络层（IP层）

**5、在网络编程中涉及并发服务器，使用多进程与多线程的区别？**

答：（1）线程执行开销小，但不利于资源管理和保护；进程则相反，进程可跨越机器迁移。 （2）多进程时每个进程都有自己的内存空间，而多线程间共享内存空间；

（3）线程产生的速度快，线程间通信快、切换快；

（4）线程的资源利用率比较好；

（5）线程使用公共变量或者资源时需要同步机制。

**6、说一下TCP 3次握手、4次挥手的全过程。**

**7、TCP和UDP有什么区别。**

答： TCP——传输控制协议,提供的是面向连接、可靠的字节流服务。

当客户和服务器彼此交换数据前，必须先在双方之间建立一个TCP连接，之后才能传输数据。TCP提供超时重发，丢弃重复数据，检验数据，流量控制等功能，保证数据能从一端传到另一端。

UDP——用户数据报协议，是一个简单的面向数据报的传输层协议。UDP不提供可靠性，它只是把应用程序传给IP层的数据报发送出去，但是并不能保证它们能到达目的地。由于UDP在传输数据报前不用在客户和服务器之间建立一个连接，且没有超时重发等机制，故而传输速度很快.

TCP协议和UDP协议的一些特性区别如下：

1.TCP协议在传送数据段的时候要给段标号；UDP 协议不需要。

2.TCP协议可靠；UDP协议不可靠。

3.TCP协议是面向连接；UDP协议采用无连接。

4.TCP协议负载较高,采用虚电路；UDP协议低负载。

5.TCP协议的发送方要确认接受方是否收到数据段(3次握手协议)。

6.TCP协议采用窗口技术和流控制。

**8、经常要操作的内存分为那几个类别？**

答：（1）栈区：由编译器自动分配和释放，存放函数的参数值、局部变量的值等； （2）堆：一般由程序员分配和释放，存放动态分配的变量； （3）全局区（静态区）：全局变量和静态变量存放在这一块，初始化的和未初始化的分开放； （4）文字常量区：常量字符串就放在这里，程序结束自动释放； （5）程序代码区：参访函数体的二进制代码。

**9、请讲述堆和栈的区别。**

答：（1）申请方式不同。栈上有系统自动分配和释放；堆上有程序员自己申请并指明大小； （2）栈是向低地址扩展的数据结构，大小很有限；堆是向高地址扩展，是不连续的内存区域，空间相对大且灵活； （3）栈由系统分配和释放速度快；堆由程序员控制，一般较慢，且容易产生碎片；