模拟面试Day17

> Written By CTENET 所有面试内容来自于腾讯Tencent，Nowcoder以及个人经历整理...

1、HTTP常见的请求方法?

客户端发送的 **请求报文** 第一行为请求行，包含了方法字段。

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

序号	方法	描述
1	GET	请求指定的页面信息，并返回实体主体。
2	HEAD	类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头
3	POST	向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
4	PUT	从客户端向服务器传送的数据取代指定的文档的内容。
5	DELETE	请求服务器删除指定的页面。
6	CONNECT	HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
7	OPTIONS	允许客户端查看服务器的性能。
8	TRACE	回显服务器收到的请求，主要用于测试或诊断。
9	PATCH	是对 PUT 方法的补充，用来对已知资源进行局部更新 。

2、GET和POST的区别?

1. get是获取数据，post是修改数据
2. get把请求的数据放在url上， 以?分割URL和传输数据，参数之间以&相连，所以get不太安全。而post把数据放在HTTP的包体内（request body 相对安全）
3. get提交的数据最大是2k（ 限制实际上取决于浏览器）， post理论上没有限制。
4. GET产生一个TCP数据包，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); POST产生两个TCP数据包，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)。

5. GET请求会被浏览器主动缓存，而POST不会，除非手动设置。
6. 本质区别：GET是幂等的，而POST不是幂等的

> 这里的幂等性：幂等性是指一次和多次请求某一个资源应该具有同样的副作用。简单来说意味着对同一URL的多个请求应该返回同样的结果。

正因为它们有这样的区别，所以不应该且**不能用GET请求做数据的增删改这些有副作用的操作。**因为get请求是幂等的，**在网络不好的隧道中会尝试重试。**如果用get请求增数据，会有**重复操作**的风险，而这种重复操作可能会导致副作用（浏览器和操作系统并不知道你会用get请求去做增操作）。

3、HTTP长连接和短连接的区别?

在HTTP/1.0中默认使用短连接。也就是说，客户端和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。

而从HTTP/1.1起，默认使用长连接，用以保持连接特性。

4、OSI七层模型及各自功能?

简要概括

- 物理层：底层数据传输，如网线；网卡标准。
- 数据链路层：定义数据的基本格式，如何传输，如何标识；如网卡MAC地址。
- 网络层：定义IP编址，定义路由功能；如不同设备的数据转发。
- 传输层：端到端传输数据的基本功能；如 TCP、UDP。
- 会话层：控制应用程序之间会话能力；如不同软件数据分发给不同软件。
- 表示层：数据格式标识，基本压缩加密功能。
- 应用层：各种应用软件，包括 Web 应用。

说明

- 在四层，既传输层数据被称作**tcp报文段或udp用户数据报**（Segments）；
- 三层网络层数据被称做**包**（Packages）；

- 二层数据链路层时数据被称为**帧**（Frames）；
- 一层物理层时数据被称为**比特流**（Bits）。

总结

- 网络七层模型是一个标准，而非实现。
- 网络四层模型是一个实现的应用模型。
- 网络四层模型由七层模型简化合并而来。

5、线程与进程的比较或者说区别？

第一种回答

1、线程启动速度快，轻量级

2、线程的系统开销小

3、线程使用有一定难度，需要处理数据一致性问题

4、同一线程共享的有堆、全局变量、静态变量、指针，引用、文件等，而独自占有栈

第二种回答

- 调度：线程是调度的基本单位（PC，状态码，通用寄存器，线程栈及栈指针）；进程是拥有资源的基本单位（打开文件，堆，静态区，代码段等）。
- 并发性：一个进程内多个线程可以并发（最好和CPU核数相等）；多个进程可以并发。
- 拥有资源：线程不拥有系统资源，但一个进程的多个线程可以共享隶属进程的资源；进程是拥有资源的独立单位。
- 系统开销：线程创建销毁只需要处理PC值，状态码，通用寄存器值，线程栈及栈指针即可；进程创建和销毁需要重新分配及销毁task_struct结构。

6、堆和栈的区别？

申请方式不同

- 栈由系统自动分配。
- 堆是自己申请和释放的。

申请大小限制不同

- 栈顶和栈底是之前预设好的，栈是向栈底扩展，大小固定，可以通过ulimit -a查看，由ulimit -s修改。
- 堆向高地址扩展，是不连续的内存区域，大小可以灵活调整。

申请效率不同

- 栈由系统分配，速度快，不会有碎片。
- 堆由程序员分配，速度慢，且会有碎片。

栈空间默认是4M, 堆区一般是 1G - 4G

	堆	栈
管理方式	堆中资源由程序员控制（容易产生memory leak）	栈资源由编译器自动管理，无需手工控制
内存管理机制	系统有一个记录空闲内存地址的链表，当系统收到程序申请时，遍历该链表，寻找第一个空间大于申请空间的堆结点，删 除空闲结点链表中的该结点，并将该结点空间分配给程序（大多数系统会在这块内存空间首地址记录本次分配的大小，这样delete才能正确释放本内存空间，另外系统会将多余的部分重新放入空闲链表中）	只要栈的剩余空间大于所申请空间，系统为程序提供内存，否则报异常提示栈溢出。（这一块理解一下链表和队列的区别，不连续空间和连续空间的区别，应该就比较好理解这两种机制的区别了）
空间大小	堆是不连续的内存区域（因为系统是用链表来存储空闲内存地址，自然不是连续的），堆大小受限于计算机系统中有效的虚拟内存（32bit 系统理论上是4G），所以堆的空间比较灵活，比较大	栈是一块连续的内存区域，大小是操作系统预定好的，windows下栈大小是2M（也有是1M，在 编译时确定，VC中可设置）
碎片问题	对于堆，频繁的new/delete会造成大量碎片，使程序效率降低	对于栈，它是有点类似于数据结构上的一个先进后出的栈，进出一一对应，不会产生碎片。（看到这里我突然明白了为什么面试官在问我堆和栈的区别之前先问了我栈和队列的区别）
生长方向	堆向上，向高地址方向增长。	栈向下，向低地址方向增长。

	堆	栈
分配方式	堆都是动态分配（没有静态分配的堆）	栈有静态分配和动态分配，静态分配由编译器完成（如局部变量分配），动态分配由alloca函数分配，但栈的动态分配的资源由编译器进行释放，无需程序员实现。
分配效率	堆由C/C++函数库提供，机制很复杂。所以堆的效率比栈低很多。	栈是其系统提供的数据结构，计算机在底层对栈提供支持，分配专门寄存器存放栈地址，栈操作有专门指令。

形象的比喻

栈就像我们去饭馆里吃饭，只管点菜（发出申请）、付钱、和吃（使用），吃饱了就走，不必理会切菜、洗菜等准备工作和洗碗、刷锅等扫尾工作，他的好处是快捷，但是自由度小。

堆就象是自己动手做喜欢吃的菜肴，比较麻烦，但是比较符合自己的口味，而且自由度大。

7、C++中NULL和nullptr的区别

算是为了与C语言进行兼容而定义的一个问题吧

NULL来自C语言，一般由宏定义实现，而 nullptr 则是C++11的新增关键字。

在C语言中，NULL被定义为(void*)0,而在C++语言中，NULL则被定义为整数0。编译器一般对其实际定义如下：

```
#ifdef __cplusplus
#define NULL 0
#else
#define NULL ((void *)0)
#endif
```

在C++中指针必须有明确的类型定义。但是将NULL定义为0带来的另一个问题是无法与整数的0区分。因为C++中允许有函数重载，所以可以试想如下函数定义情况：

```
#include <iostream>
using namespace std;

void fun(char* p) {
    cout << "char*" << endl;
}

void fun(int p) {
    cout << "int" << endl;
}

int main()
{
    fun(NULL);
    return 0;
}
//输出结果：int
```

那么**在传入NULL参数时，会把NULL当做整数0来看，如果我们想调用参数是指针的函数，该怎么办呢?。nullptr在C++11被引入用于解决这一问题，nullptr可以明确区分整型和指针类型，能够根据环境自动转换成相应的指针类型，但不会被转换为任何整型，所以不会造成参数传递错误。**

nullptr的一种实现方式如下：

```
const class nullptr_t{
public:
    template<class T>  inline operator T*() const{ return 0; }
    template<class C, class T> inline operator T C::*() const { return 0; }
private:
    void operator&() const;
} nullptr = {};
```

以上通过模板类和运算符重载的方式来对不同类型的指针进行实例化从而解决了(void*)指针带来参数类型不明的问题，**另外由于nullptr是明确的指针类型，所以不会与整形变量相混淆。**但nullptr仍然存在一定问题，例如：

```
#include <iostream>
using namespace std;

void fun(char* p)
{
    cout<< "char* p" <<endl;
}
void fun(int* p)
{
    cout<< "int* p" <<endl;
}

void fun(int p)
{
    cout<< "int p" <<endl;
}
int main()
{
    fun((char*)nullptr);//语句1
    fun(nullptr);//语句2
    fun(NULL);//语句3
    return 0;
}
//运行结果：
```

```
//语句1：char* p
//语句2:报错，有多个匹配
//3：int p
```

在这种情况下存在对不同指针类型的函数重载，此时如果传入nullptr指针则仍然存在无法区分应实际调用哪个函数，这种情况下必须显示的指明参数类型。

8、malloc申请的存储空间能用delete释放吗?

不能，malloc /free主要为了兼容C，new和delete 完全可以取代malloc /free的。

malloc /free的操作对象都是必须明确大小的，而且不能用在动态类上。

new 和delete会自动进行类型检查和大小，malloc/free不能执行构造函数与析构函数，所以动态对象它是不行的。

当然从理论上说使用malloc申请的内存是可以通过delete释放的。

不过一般不这样写的。而且也不能保证每个C++的运行时都能正常。

9、delete p、delete [] p、allocator都有什么作用?

动态数组管理new一个数组时，[]中必须是一个整数，但是不一定是常量整数，普通数组必须是一个常量整数；

2、 new动态数组返回的并不是数组类型，而是一个元素类型的指针；

3、 delete[]时，数组中的元素按逆序的顺序进行销毁；

4、 new在内存分配上面有一些局限性，new的机制是将内存分配和对象构造组合在一起，同样的，delete也是将对象析构和内存释放组合在一起的。allocator将这两部分分开进行，allocator申请一部分内存，不进行初始化对象，只有当需要的时候才进行初始化操作。

10、简述MVVM

什么是MVVM?

视图模型双向绑定，是Model-View-ViewModel的缩写，也就是把MVC中的Controller演变成ViewModel。

Model层代表数据模型，View代表UI组件，ViewModel是View和Model层的桥梁，数据会绑定到viewModel层并自动将数据渲染到页面中，视图变化的时候会通知viewModel层更新数据。

以前是操作DOM结构更新视图，现在是数据驱动视图。

MVVM的优点：

1. 低耦合。视图（View）可以独立于Model变化和修改，一个Model可以绑定到不同的View上，当View变化的时候Model可以不变化，当Model变化的时候View也可以不变；
2. 可重用性。你可以把一些视图逻辑放在一个Model里面，让很多View重用这段视图逻辑。
3. 独立开发。开发人员可以专注于业务逻辑和数据的开发(ViewModel)，设计人员可以专注于页面设计。
4. 可测试。