技术面试过程中回答问题应该注意的事项

1.当面试官提问问题后，不要着急作答，应该适当停一下，整理一下逻辑思路

2.对于简单问题的回答，尽量不要照本宣科，找准问题回答的角度/层次，争取简单问题回答的比较有亮点

3.对于相对复杂的问题，比较难以阐述的问题，思考上要花一些时间，整理好逻辑思路，以及问题大致描述的顺序。如果是现常面试（视频面试），最好用纸笔边画边讲；如果是电话面试，回答问题的过程中，需要和面试官经常沟通，不要自顾自的滔滔不绝。

4.对于面试中，被提问到自己不知道的内容，被问到区块链？人工智能，机器人，大数据离线分析，用户行为分析，智能推荐

5.你还有什么问题

问题：C++this指针干什么用的?

问题：C++的new和delete，什么时候用new[]申请，可以用delete释放?

问题：C++的static关键字的作用(我从elf结构，链接过程来回答)?

问题：C++的继承?

类和类　继承a kind of...还有组合a part of...

1. 代码的复用
2. 通过继承，在基类里面给所有派生类可以保留统一的纯虚函数接口，等待派生类进行重写，通过使用多态，可以通过基类的指针访问不同派生类对象的同名覆盖方法。

问题：C++的继承多态，空间配置器，vector 和list的区别，map，多重map?

多态：静（编译时期）多态：函数重载和模板　　和动（运行时期）多态：虚函数 指针/引用指向派生类对象

空间配置器allocator：给容器使用的，主要作用把内存开辟和对象构造分开，把对象析构和内存释放分开

vector和list的区别：数组和链表，随机访问多（优先级队列基于vector）　　，list适合增加删除多

map（不允许key重复的）：映射表[key-value]，底层实现红黑树，multimap（允许key重复的）

红黑树：5个性质，插入3种情况（最多旋转2次），删除（最多旋转3次）4种情况

问题：C++如何防止内存泄露？智能指针详述?

内存泄漏：分配的堆内存（没有名字，只能用指针来指向）没有释放，也再没有机会释放了

int *p = new int[10000];　　unique_ptr pre(new int[100000]);

auto_ptr/scoped_ptr/unique_ptr　shared_ptr/weak_ptr

```
if(xxx)
  return;  // 运行抛异常了
delete []p;
```

问题：C++如何调用C语言函数接口?

C和C++生成符号的方式不同，C和C++语言之间的API接口是无法直接调用的，C语言的函数声明必须扩在extern "C"{}

```
#ifdef __cpluscplus  // C生成的函数接口，可以在C和C++环境下直接使用
extern "C"
{
#endif
    int sum(int, int);  // sum   sum_int_int
#ifdef __cpluscplus
}
#endif
```

问题：C++什么时候会出现访问越界?

1.访问数组元素越界了

2.vector容器访问 vector< int > vec; vec[2];

3.string str; str[2]

4.array

5.字符串处理，没有添加'\0'字符，导致访问字符串的时候越界了

6.使用类型强转，让一个大类型（派生类）的指针指向一块小内存（基类对象）了，然后指针解引用，访问的内存就越界了!

问题：C++中类的初始化列表?

可以指定对象成员变量的初始化方式，尤其是指定成员对象的构造方式;

问题：C和C++的区别? C和C++的内存分布有什么区别?

1.引用

2.函数重载

3.new/delete malloc/free

4.const, inline,带默认值参数的函数

5.模板

6.类和对象 OOP =》 设计模式了

7.STL

8.异常 智能指针 运算符重载

9.user space reserve .text .rodata .data .bss heap stack 命令行参数和环境变量 kernal space ZONE_DMA ZONE_NORMAL(.text .rodata .data. bss. heap stack) ZONE_HIGHMEM

问题：int* const p和const int *p区别?

问题：malloc和new区别?

1.malloc按字节开辟内存 new底层也是通过malloc开辟内存，但是还可以提供初始化

2.malloc开辟内存失败 nullptr new开辟失败，抛出bad_alloc类型的异常

3.malloc C的库函数　operator new

4.malloc 单个 数组　new int(10);　new int[20] ();

问题：map&set容器的实现原理?

set集合，只存储key；map映射表，存储[key,value]键值对，底层数据结构都是红黑树

红黑树

问题：shared_ptr引用计数存在哪里?

QQ：413246753　堆上分配的

问题：STL、map底层、deque底层、vector里的empty()和size()的区别、函数对象?

标准容器 =》 顺序容器（vector,deque,list），容器适配器(stack,queue,priority_queue)，关联容器(有序和无序)

近容器　数组，string，bitset

迭代器

泛型算法

函数对象 operator()　test(); test.operator()(); sort find_if priority_queue set map

问题：STL中的迭代器失效的问题?

迭代器是不允许一边读一遍修改的

当通过迭代器插入一个元素，所有迭代器就都失效了

当通过迭代器删除一个元素，当前删除位置都后面所有的元素的迭代器就都失效了

当通过迭代器更新容器元素以后，要及时对迭代器进行更新，　insert / erase方法都会返回新位置的迭代器

问题：STL中哪些底层由红黑树实现?

set multiset map multimap

问题：struct和class的区别?

1.定义类的时候的区别

2.继承时，派生类 class B : A　struct B : A

3.struct 空结构体是0　struct 空类是1

4.C++11 struct Data { int ma, int mb}　Data data = {10, 20};

5.class在template< class T >还可以定义模板类型参数

问题：vector和list的区别，还有map的底层实现?

问题：vector和数组的区别，STL的容器分类，各容器底层实现?

直接使用数组，　vector是数组的一个面向对象的表示，把数组封装起来了

vector：　deque：　list：

stack（deque）：push pop　queue（deque）　priority_queue（大根堆，vector）

set/map

unordered_set/unordered_map：

问题：编译链接全过程?

预编译、编译、汇编 =》 二进制可重定向obj文件 *.o

链接：1.合并段，符号解析 2.符号的重定向 =》 可执行文件

问题：初始化全局变量和未初始化全局变量有什么区别?

.data(初始化，且初始值不为0) .bss（未初始化，初始化为0）

问题：堆和栈的区别?

堆内存的大小 >> 栈内存 malloc/new free/delete 函数的运行，函数的局部变量

低地址 =》 高地址

高地址 =》 低地址

问题：构造函数和析构函数可不可以是虚函数，为什么?

构造函数不能是虚函数

析构函数可以

Base *p = new Derive(); 把基类的析构函数是现成虚析构函数

delete p; // 对析构函数的调用进行动态绑定 ~Base() ~Derive()

问题：构造函数和析构函数中能不能抛出异常，为什么?

构造函数不能抛异常， 对象创建失败，就不会调用对象的析构函数了 把堆内存用智能指针来代替

init() 保证对象创建是成功的!

析构函数不能抛异常，后面的代码就无法得到执行了

问题：宏和内联函数的区别?

#define和inline

预编译阶段（字符串替换）

编译阶段（在函数调用点，通过函数的实参把函数代码直接展开调用，节省了函数的调用开销）

调试的，宏没有办法调试， inline函数可以调试（debug版本下inline就和普通函数一样，有标准的函数调用过程）

#define 可以定义常量，代码块，函数块 \

inline只是修饰函数

问题：局部变量存在哪里?

stack ebp指针偏移来访问的，不产生符号，.text

int a = 10; => mov dword ptr[ebp-4], 0Ah .data.bss malloc/new

问题：拷贝构造函数，为什么传引用而不传值?

class Test

{

public:

Test(const Test &t);

}

Test t1;

Test t2(t1); // t2.Test(t1) => const Test t(t1) => t.Test(t1) => 不行的，直接产生编译错误！！！

问题：内联函数和普通函数的区别（从反汇编角度来回答）？

核心：函数的调用开销！

push ebp

mov ebp, esp

sub esp, 4Ch

rep stos 0xCCCCCCCC(windows) GCC(gcc/g++)：分配完栈帧， 不做任何栈初始化动作

释放栈

mov esp, ebp

pop ebp

ret

问题：如何实现一个不可以被继承的类?

派生类的初始化过程：基类构造 =》 派生类构造 基类的构造函数私有化

问题：什么是纯虚函数？为什么要有纯虚函数？虚函数表放在哪里的?

virtual void func() = 0; 纯虚函数 =》 抽象类（不能实例化对象的，可以定义指针和引用）

一般定义在基类里面，基类不代表任务实体，它的主要作用之一就是给所有的派生类保留统一的纯虚函数接口，让

派生类进行重写，方便的使用多态机制。因为基类不需要实例化，它的方法也就不指导该怎么去实现！

虚函数表 在 编译阶段产生的！ 运行时，加载到.rodata段

问题：手写单例模式

问题：说一下C++中的const，const与static的区别?

const定义的叫常量，它的编译方式是：编译过程中，把出现常量名字的地方，用常量的值进行替换

```
int b = 10;
const int a = b;  // 常变量
int *p = (int*)&a;
*p = 20;
cout<<a<<" "<<*p<<endl;
```

const还可以定义常成员方法， Test *this => const Test *this 普通对象和常对象就都可以调用了！

const和static的区别

面向过程：

const：全局变量，局部变量，形参变量 static：全局变量，局部变量

const：不能修饰函数　static：可以修饰函数

面向对象：

const：常方法/成员变量　Test *this => const Test *this 依赖对象

static：静态方法/成员变量　Test *this => 没有了！ 不依赖于对象 通过类作用域访问

问题：四种强制类型转换?

const_cast

static_cast

reinterpret_cast: C风格的类型转换

dynamic_cast：支持RTTI信息识别的类型转换

问题：详细解释deque的底层原理

动态开辟的二维数组

#define MAP_SIZE 2

#define QUE_SIZE(T) 4096/sizeof(T)

一维数组的初始大小 MAP_SIZE (T*)

第二维数组默认开辟的大小就是QUE_SIZE(int) 1024

双端队列　两端都有队头和队尾 两端都可以插入删除O(1)

扩容：把第一维数组按照2倍的方式进行扩容　2-4-8-16。。。 扩容以后，会把原来的第二维的数组，从新一维数组的第oldsize / 2 开始存放　2 - 4　　4/2 = 2

=

=

=

=

stack queue =》 deque（内存利用率好， 刚开始就有一段内存可以供使用）0-1-2-4
priority_queue：vector

问题：虚函数，多态

一个类 =》 虚函数 =》 编译阶段 =》 该类产生一张虚函数表 =》 运行时，加载到.rodata

用指针或者引用 =》 调用 虚函数 =》 指针访问对象的头四个字节vfptr =》 vftable中取虚函数的地址，进行动态绑定调用

多态：设计函数接口的时候，可以都是用基类的指针或者引用来接收不同的派生类对象，功能增加，删除；设计模式

问题：虚析构函数、智能指针

Base *p = new Derive();

delete p; // 析构函数的调用，动态绑定

智能指针：管理资源的生命周期

问题：一个类，写了一个构造函数，还写了一个虚构造函数，可不可以，会发生什么？

在构造函数中，是不会进行动态绑定的！构造函数本身也不能实现成虚函数！

问题：异常机制怎么回事儿？

try

{

　可能会抛出异常的代码 throw

}

catch(const string &err)

{

捕获相应异常类型对象，进行处理，完成后，代码继续向下运行

}

异常的栈展开！ 可以把代码中所有的异常抛到统一的地方进行处理！ exit(0);

问题：早绑定和晚绑定？

早绑定（静态绑定）：普通函数的调用，用对象调用虚函数 call 编译阶段已经知道调用哪个函数

晚绑定（动态绑定）：用指针/引用调用虚函数的时候，都是动态绑定 p->vfptr->vftable->virtual addr => call eax

问题：指针和引用的区别(反汇编分析)

int a = 10;

int *p = &a;　lea eax, [a]　mov dword ptr[ebp-8], eax

int &b = a;　　lea eax, [a]　mov dword ptr[ebp-0Ch], eax

*p = 20;　mov eax, dword ptr[ebp-8]　 mov dword ptr[eax], 14H

b = 20;　 mov eax, dword ptr[ebp-0Ch]　 mov dword ptr[eax], 14H

问题：智能指针交叉引用问题怎么解决？

定义对象的时候用强智能指针shared_ptr，而引用对象的时候用弱智能指针weak_ptr，

当通过weak_ptr访问对象成员时，需要先调用weak_ptr的lock提升方法，把weak_ptr提升成shared_ptr强智能指针，再进行对象成员调用。

问题：重载的底层实现，虚函数的底层实现

重载，因为C++生成函数符号，是依赖函数名字+参数列表

编译到函数调用点时，根据函数名字和传入的实参（个数和类型），和某一个函数重载匹配的话，那么就直接调用相应的函数重载版本（静态的多态 都是在编译阶段处理的！）

虚函数 =》 对象　指针/引用 =》 vfptr => vftable => 虚函数的地址 call eax