page 3 user: sakura

21.请描述一下C++的内存分区模型.

栈区(stack): 由编译器自动分配释放，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

堆区(heap): 一般由程序员分配释放，若程序员不释放，程序结束时可能由OS回收。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表，呵呵。

全局区(静态区)(static): 全局变量和静态变量的存储是放在一块的，初始化的全局变量和静态变量在一块区域，未初始化的全局变量和未初始化的静态变量在相邻的另一块区域。-程序结束后由系统释放。

常量区: 常量字符串就是放在这里的。程序结束后由系统释放

代码段: 存放函数体的二进制代码，直接的操作数也是存储在这个位置的。如int a=4;

22.C语言如何实现C++的面向对象特性?

封装 :C语言中是没有class类这个概念的，但是有struct结构体，我们可以考虑使用struct来模拟；使用函数指针把属性与方法封装到结构体中。

继承 :结构体嵌套

多态 :类与子类方法的函数指针不同

23.简述数据库的三大范式.

第一范式：1NF是对属性的原子性约束，要求字段具有原子性，不可再分解；

第二范式：2NF是在满足第一范式的前提下，非主键字段不能出现部分依赖主键；

第三范式：3NF是在满足第二范式的前提下，非主键字段不能出现传递依赖；

24.char和varchar有什么区别?

char 是一种**固定**长度的字符串类型，（如果数据类型不足固定长度的话，会自动用0补齐）

varchar 是一种可变长度的字符串类型，（如果数据类型不足长度的话，自动将长度缩放成我们所输入的数据类型的长度）

25.事务是什么？事务的四大特性是什么？

> **事务是一组原子性的 SQL 语句，或者说一个独立的工作单元。**

> **原子性**：不可分割的操作单元，事务中所有操作，要么全部成功；要么撤回到执行事务之前的状态； **一致性**：如果在执行事务之前数据库是一致的，那么在执行事务之后数据库也还是一致的； **隔离性**：事务操作之间彼此独立和透明互不影响。事务独立运行。这通常使用锁来实现。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。 **持久性**：事务一旦提交，其结果就是永久的。即便发生系统故障，也能恢复。

26.索引是什么？它是如何加快查询性能的？

> **索引**是对数据库表中一列或多列的值进行排序的一种**数据结构**，也就是说索引是一种数据结构。数据库在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中通过二分法等高效率算法一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询性能。类似新华字典的目录，如果没有目录的话，我们想要查找一个汉字的话，就必须检索整本字典，而正因为有了目录，我们只要知道我们所要查找的偏旁或者拼音首字母，就可以快速地定位到我们想要查找汉字的所在页码。

27.C++会为一个空类提供哪些函数？

> 当空类Empty_one定义一个对象时Empty_one pt;sizeof(pt)仍是为1，但编译器会生成6个成员函数：一个缺省的构造函数、一个拷贝构造函数、一个析构函数、一个赋值运算符、两个取址运算符(const限定和非const限定)。
>
> 但实际上,我们只需要了解前面四种即可, 即缺省的构造函数, 拷贝构造, 析构函数, 赋值运算符重载。

28.什么是内存对齐?为什么要内存对齐？

> 举个例子: 理论上，32位系统下，int占4byte，char占一个byte，那么将它们放到一个结构体中应该占4+1=5byte；但是实际上，通过运行程序得到的结果是8 byte，这就是内存对齐所导致的。 现代计算机中内存空间都是按照 byte 划分的，从理论上讲似乎对任何类型的变量的访问可以从任何地址开始，但是**实际的计算机系统对基本类型数据在内存中存放的位置有限制，它们会要求这些数据的首地址的值是某个数k（通常它为4或8）的倍数，这就是所谓的内存对齐。**

> 尽管内存是以字节为单位，但是大部分处理器并不是按字节块来存取内存的.它一般会以双字节,四字节,8字节,16字节甚至32字节为单位来存取内存，我们将上述这些存取单位称为**内存存取粒度**.
>
> 现在考虑4字节存取粒度的处理器取int类型变量（32位系统），该处理器只能从地址为4的倍数的内存开始读取数据。
>
> 假如没有内存对齐机制，数据可以任意存放，现在一个int变量存放在从地址1开始的联系四个字节地址中，该处理器去取数据时，要先从0地址开始读取第一个4字节块,剔除不想要的字节（0地址）,然后从地址4开始读取下一个4字节块,同样剔除不要的数据（5，6，7地址）,最后留下的两块数据合并放入寄存器.这需要做很多工作。

扩展内容：内存对齐规则

> 每个特定平台上的编译器都有自己的默认“对齐系数”（也叫对齐模数）。gcc中默认**#pragma pack(4)，**可以通过预编译命令**#pragma pack(n)**，n = 1,2,4,8,16来改变这一系数。

> **有效对齐值**：是给定值**#pragma pack(n)**和结构体中最长数据类型长度中较小的那个。有效对齐值也叫对齐单位。
>
> 了解了上面的概念后，我们现在可以来看看内存对齐需要遵循的**规则**：
>
> (1) 结构体第一个成员的偏移量（offset）为0，以后每个成员相对于结构体首地址的 offset 都是该成员大小与有效对齐值中较小那个的整数倍，如有需要编译器会在成员之间加上填充字节。
>
> (2) 结构体的总大小为 有效对齐值 的整数倍，如有需要编译器会在最末一个成员之后加上填充字节。

举例:

```
//32位系统
#include<stdio.h>
struct
{
    int i;    //4
    char c1;  //1
    char c2;  //1
}x1;
struct{
    char c1;  //1
    int i;    //4
    char c2;  //1
}x2;
struct{
    char c1;  //1
    char c2;  //1
```

```
    int i;    //4
}x3;
int main(){
    printf("%d\n",sizeof(x1));  // 输出8
    printf("%d\n",sizeof(x2));  // 输出12
    printf("%d\n",sizeof(x3));  // 输出8
    return 0;
}
```

**此部分转载自知乎(忆臻); 强烈建议看看这位大佬的文章, 受益匪浅! **

——文章地址:https://zhuanlan.zhihu.com/p/30007037

29.条件编译的作用?

> #if 如果条件为真，则执行相应操作 #elif 如果前面条件为假，而该条件为真，则执行相应操作 #else 如果前面条件均为假，则执行相应操作 #endif 结束相应的条件编译指令 #ifdef 如果该宏已定义，则执行相应操作 #ifndef 如果该宏没有定义，则执行相应操作

> 条件编译是指预处理器根据条件编译指令，有条件地选择源程序代码中的一部分代码作为输出，送给编译器进行编译。主要是为了有选择性地执行相应操作，防止宏替换内容（如文件等）的**重复包含**。

30.子网掩码的作用?

> 子网掩码(subnet mask)又叫网络掩码、地址掩码、子网络遮罩，它是一种用来指明一个IP地址的哪些位标识的是主机所在的子网，以及哪些位标识的是主机的位掩码。子网掩码不能单独存在，它必须结合IP地址一起使用。 它的主要作用有两个： 一是用于屏蔽IP地址的一部分以区别网络标识和主机标识，并说明该IP地址是在局域网上，还是在远程网上。二是用于将一个大的IP网络划分为若干小的子网络。 使用子网是为了**减少IP浪费**。可以提高网络应用的效率。 通过计算机的子网掩码判断两台计算机是否属于同一网段的方法是，将计算机十进制的IP地址和子网掩码转换为二进制的形式，然后进行二进制"与"(AND)计算（全1则得1，不全1则得0），如果得出的结果是相同的，那么这两台计算机就属于同一网段。
