JNI

1.What 什么是JNI

Java Native Interface java本地接口
Native 本地 当前系统用什么语言开发的那么 这种语言对于这个系统而言就是本地语言
android底层是linux linux是c/c++开发的 所以对于android来说 c/c++就是本地语言

JNI 可以看做是一个翻译 实现JAVA语言和本地语言之间的相互调用

2.Why 为什么要用JNI

① java代码 不可以直接操作硬件 硬件的驱动基本都是C代码写的 如果想操作硬件必须得让java代码可以调用C
这个时候JNI就发挥作用了 可以扩展JAVA代码的性能
② java代码安全性比较差 java->,class->. C安全性相对高一些 C编译之后直接生成机器码 机器码可以反汇编 汇编代码可以猜出C的源代码 伪代码 有些时候跟源代码有区别 比如登录先关 跟钱打交道的应用 可以通过JNI把加密的业务逻辑放到C里面实现

③ Java特点 跨平台 一处编译到处运行 通过虚拟机来实现的 java是解释型语言 效率相对比较低 在要求效率的地方 java性能会差一些 大型3d游戏 c/c++ 音视频解码 JNI可以提升java的效率 复式投注 需要效率的时候可以通过JNI来调用C/C++来实现

④ c1972年 c++ java 1995年 ffmpeg音视频解码 opencv 图像处理 人脸识别 使用JNI可以让java代码调用 c/c++写的优秀开源项目
学习JNI目的 满足自己项目的需求

3.How怎么用JNI

Java
Native c/c++ 学习C的基础语法 能看懂C代码 会调用C的函数
Interface 接口 jni的规范 android的开发方式

C的基础语法
1 C的helloworld
2 C的基本数据类型 没有boolean byte String没有 c 用0 非0表示true false

java的基本数据类型

boolean 1

byte 1

short 2 short 2

char 2 char 1个字节 **

int 4 int 4

long 8 long 4 (有的编译器是4个字节 有的是8个)

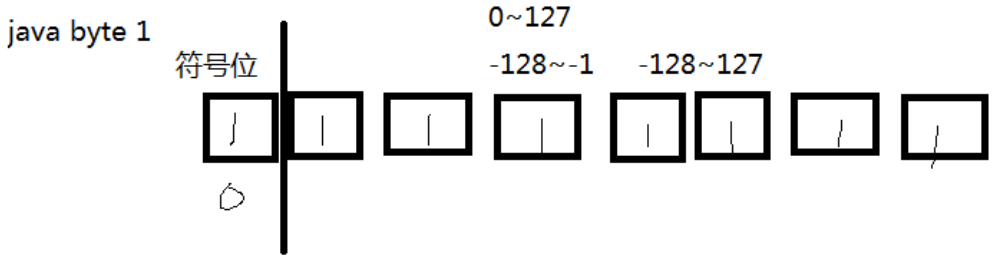
float 4 float 4

double 8 double 8

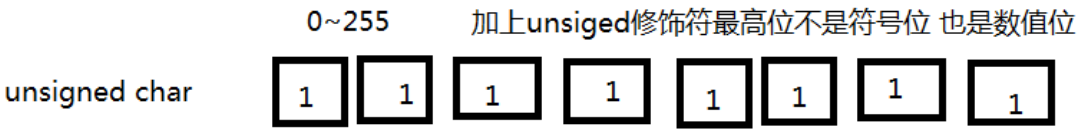
signed unsigned只能用来修饰整形变量

signed 有符号

unsigned 无符号



c char C 整形变量默认都是有符号的 最高位是符号位



signed 可以表示负数 表示的最大值相对小

unsigned 不可以表示负数 表示的最大值相对大

3.C的输出函数

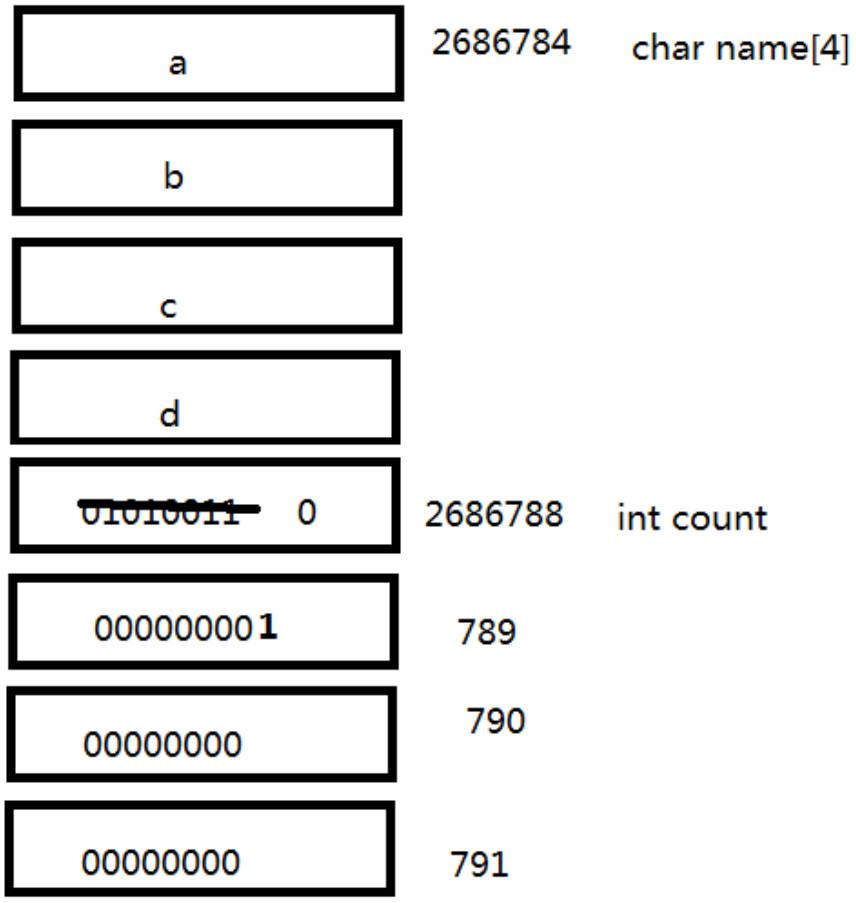
```
#include<stdio.h>
#include<stdlib.h>
/*
%d  -  int
%ld – long int
%lld - long long
%hd – 短整型 short
%c  - char
%f -  float
%lf – double
%u – 无符号数
%x – 十六进制输出 int 或者long int 或者short int
%o -  八进制输出
%s – 字符串
```

```
15.
16.  int 4个字节   12345678   10111100 01100001 01001110
17.                 24910            01100001 01001110
18.  */
19.  main(){
20.          int i = 12345678;
21.          short s = 1234;
22.          char c = 'a';
23.          long l = 1234567890;
24.          float f = 3.14;
25.          double d = 3.1415926;
26.          printf("i = %hd\n",i);
27.          printf("s = %hd\n",s);
28.          printf("c = %c\n",c);
29.          printf("l = %ld\n",l);
30.          printf("f = %.2f\n",f);//显示小数 默认是6位有效数字 要控制显示的位数加上.x
       %.2f 显示两位有效数字
31.          printf("d = %.7lf\n",d);
32.
33.          printf("i= %#x\n",i);  //%#X  加上#会添加0X前缀
34.          printf("i= %#o\n",i);  //%#o  加上#会添加0前缀
35.          //java 数组 和C数组声明方式区别:  C的数组[] 必须放到变量名后面
36.          char arr[] = {'a','b','c','d','\0'};//c的字符数组 需要手动声明一个结束符 '
     \0'
37.          char arr2[] = "你好";
38.          printf("arr = %s\n",arr);
39.         system("pause");
40.
41.         }
```

4.C的输入函数

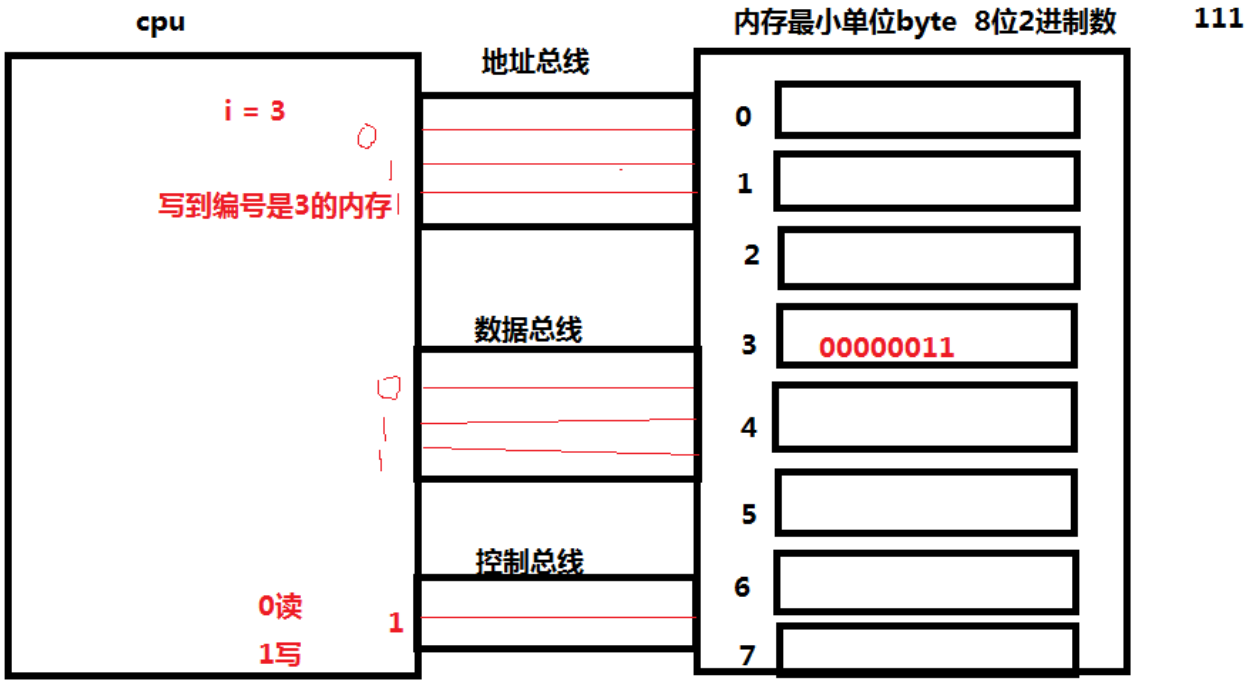
scanf

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  //c的数组不检测越界 使用时要加小心  声明数组的时候稍微申请的元素个数要大一些
4.  main(){
5.         printf("请输入班级的人数:\n");
6.         int count;
7.         printf("count的地址%d\n",&count);
8.         scanf("%d",&count);
9.         printf("班级人数为%d\n",count);
10.        char name[10];
11.        printf("name的地址%d\n",&name);
12.        printf("请输入班级名字:\n");
13.        scanf("%s",&name);
14.        printf("班级的名字%s,班级的人数是%d\n",name,count);
15.        system("pause");
16.
17.        }
```



5 内存地址的概念

cpu 想访问内存 内存必须有一个内存地址 没有内存地址的内存是无法使用的
内存地址实际上就是内存的编号 每一个byte的内存都会对应一个编号



32位操作系统最大支持多少内存 tip 32位操作系统能够驱动的地址总线是32位

2^32 32位操作系统 支持的最大的内存编号 可以支持2^32byte这么大的内存 4g
系统的一些硬件会占用内存 集成显卡会占用的内存比较多

通过内存地址 可以直接访问内存的内容 也可以对这块内存的内容进行修改

6 指针入门 **********

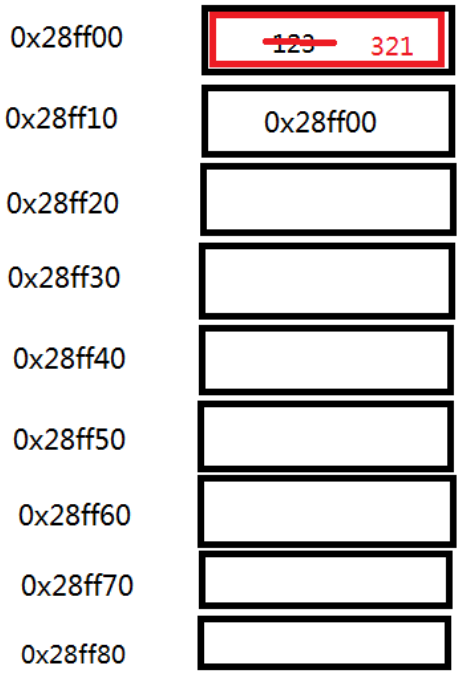
```
#include<stdio.h>
#include<stdlib.h>
main(){
	int i = 123;
	//变量名字+ "*" 可以声明一个指针变量  int* int类型的指针变量 可以保存一个int类型的变量地址
	int* pointer = &i;
	int * p;   int *p;   int* p;

	printf("i的地址%#x\n",pointer);
	printf("i = %d\n",i);
	printf("i = %d\n", *pointer);

	*pointer = 321;
	printf("i = %d\n",i);
	system("pause");

	}
```

pointer 保存了另外一个变量的内存地址
*pointer 拿着另外一个变量的内存地址 要对这个内存地址中对应的内容进行操作 如果*pointer在等号左边 是要做赋值的操作
	如果*pointer在等号的右边 就是要做取值的操作
&pointer pointer这个变量自己也对应着一块内存 这个内存也会有编号 &pointer取出自己的编号




```
main(){
    int i = 123;
    //变量名字+ "*" 可以声明一个指针变量  int* int类型的指针变量 可以保存一个int类型的变量地址
    int* pointer = &i;
    printf("i的地址%#x\n",pointer);
    printf("i = %d\n",i);
    printf("i = %d\n", *pointer);

    *pointer = 321;          pointer
    printf("i = %d\n",i);    *pointer
    system("pause");         &pointer

    }
```

==========

指针的常见错误

```c
#include<stdio.h>
#include<stdlib.h>
main(){
        //声明了一个指针变量并没有为它赋初始值就直接使用*p= 做赋值的操作 这个是不行的
        //这种没 赋初始值就直接使用的指针 叫野指针
        //指针变量的初始化 要用当前程序中声明的变量的地址 给指针赋值

        //int类型的指针变量 要保存一个int类型的变量地址 指针的类型 跟它保存的地址的变量类型要一一对应
        //如果不对应可能会有问题
        double i;
        printf("i = %lf\n",i);
        int* p = &i;
        *p = 123;
        printf("p=%#x\n",p);
        printf("i = %lf\n",i);
       system("pause");

       }
```

7指针的练习

7.1交换两个数的值

```c
#include<stdio.h>
```

```
#include<stdlib.h>
//值传递 直接传递是变量中保存的值
//引用传递  传递的也是值 但这个值比较特殊 它是一块内存的地址值
//如果想通过一个函数改变临时变量的值 必须使用引用传递 也就是说 必须把变量的地址传递给函数 函数拿到地址
//通过指针变量可以修改对应的值

swap(int* p ,int* q){
       int temp = *p;
        *p = *q;
        *q = temp;

         }
main(){
        int i = 123;
        int j = 456;
        //引用传递  传递的是变量的地址
       swap(&i,&j);
        printf("i = %d,j=%d\n",i,j);
       system("pause");

       }
```

7.2 返回多个值

```
#include<stdio.h>
#include<stdlib.h>
//* 作用 int* 声明指针变量
// a * b 乘法运算

function(int* p, int* q){
           *p *= 2;
           *q *= 2;
           }

main(){
   int i = 123;
   int j = 456;
   function(&i,&j);
   printf("i = %d,j=%d\n",i,j);
       system("pause");
       }
```

8数组和指针的关系

```
#include<stdio.h>
#include<stdlib.h>
//数组是一块连续的内存空间
//数组的名字的地址 就是数组第一个元素的地址 也就是数组的首地址
//可以通过一个指针变量 把数组的首地址保存起来 拿到了数组的首地址 就可以通过指针的+
```

```
1 +2 这种位移运算
//访问到数组中的任何一个变量 就像通过数组的下标操作一样
//指针位移运算的时候 指针+1 移动的字节数 跟指针的类型有关 如果是char* p+1 会移动一个字节
//如果是int* p  p+1会移动4个字节
main(){
        //char array[] = {'a','b','c','d','\0'};
        int array[] = {1,2,3,4,5};
        printf("array[0]的地址%#x\n",&array[0]);
        printf("array[1]的地址%#x\n",&array[1]);
        printf("array[2]的地址%#x\n",&array[2]);
        printf("array[3]的地址%#x\n",&array[3]);

        printf("array的地址%#x\n",&array);

        //char* p = &array[0];
       // char* p = &array;
        int* p = &array;
       // printf("*p = %c\n",*p);
       // printf("*(p+1) = %c\n",*(p+1));
       // printf("*(p+2) = %c\n",*(p+2));
       // printf("*(p+3) = %c\n",*(p+3));

        printf("p+0 = %#x\n",p+0);
        printf("p+1 = %#x\n",p+1);
        printf("p+2 = %#x\n",p+2);
        printf("p+3 = %#x\n",p+3);

         printf("*p = %d\n",*p);
        printf("*(p+1) = %d\n",*(p+1));
        printf("*(p+2) = %d\n",*(p+2));
        printf("*(p+3) = %d\n",*(p+3));
       system("pause");

       }
```

c字符串

```
#include<stdio.h>
#include<stdlib.h>
//c定义字符串最常用的方式
//char* p = "hello";
main(){
        char str[] = {'a','b','c','d','\0'};
        char str1[] = "hello";
        //char str2[] = "hello";
        char* p = &str;
        printf("str = %#x\n",str);
        printf("str = %#x\n",&str);
        //c定义字符串最常用的方式
        char* p = "hello";
        printf("%s\n",p);
       system("pause");

```

```
        }
```

指针变量的长度

```
#include<stdio.h>
#include<stdlib.h>
//32位的环境(操作系统 编译器 32位) 指针变量长度就是4  跟具体类型没有关系
//64位环境 指针变量长度是8
main(){
        int* p;
        double* p1;
        printf("p的长度%d\n",sizeof p);
        printf("p1的长度%d\n",sizeof p1);
       system("pause");

        }
```

9多级指针

```
#include<stdio.h>
#include<stdlib.h>
//变量的地址 要用指针变量保存
//一级指针的地址 要用一个二级指针来保存
//二级指针的地址 要用一个三级指针来保存 以此类推
//虽说可以用一个Int类型变量保存内存地址 但是没什么意义 不要这样用
main(){
        int i = 123;
        int* p = &i;
        //int* q = &p;
       // int j = &q;
      //  *j = 456;
        int** p2 = &p;
        int*** p3 = &p2;

        ***p3 = 456;
        printf("i = %d\n",i);
       system("pause");
       }
```

多级指针练习 main函数获取子函数临时变量的地址

```
#include<stdio.h>
#include<stdlib.h>
function(double** p){
           double d = 123.456;
           *p = &d;
           printf("d的地址%#x\n",&d);
           }
main(){
        double* p;
        function(&p);
```

```
        printf("p的值%#x\n",p);
       system("pause");

       }
```

10 静态内存分配和动态内存分配

10.1静态内存分配

```
#include<stdio.h>
#include<stdlib.h>
//静态内存分配
//栈内存 由高地址向低地址扩展 栈内存是一块连续的内存空间  栈内存的大小是固定的 栈中内存空间比较小 不要创建大的数组
//由系统统一分配  统一释放
//当函数运行完 占用的栈内存会被立即回收 (注意回收的概念 不是内存被干掉 而是被标记可以被使用)
//stack overflow 栈溢出 stackoverflow.com
int* function(){
     int array[] = {1,2,3,4,5};
     printf("array的地址%#x\n",&array);
     return &array;
     }
int* function2(){
     int array2[] = {5,4,3,2,1};
     printf("array2的地址%#x\n",&array2);
     return &array2;
     }

main(){
        int* p = function();
        //function2();
        //第一次打印结果的时候 实际上function2占用的内存已经在回收 所以只能显示一次正确结果
        printf("%d,%d,%d\n",*p,*(p+1),*(p+2));
        printf("%d,%d,%d\n",*p,*(p+1),*(p+2));
        printf("%d,%d,%d\n",*p,*(p+1),*(p+2));
       system("pause");

       }
```

10.2动态内存分配

```
#include<stdio.h>
#include<stdlib.h>
//申请一个堆内存 就叫动态内存分配
//在java当中 使用new关键字来申请堆内存
//在c当中 malloc memory allocation 内存分配
//malloc(要申请的堆内存大小单位byte) 返回值 申请内存的首地址
//在c中 动态内存分配 就是通过malloc函数手动的申请一块堆内存
```

```
//通过malloc申请的堆内存 需要程序员手动释放 释放的时候 使用free函数
//free传入的是malloc的返回值  free之后 malloc申请的堆内存保存的数据就没有意义了
 //malloc申请
 //free释放
main(){
        int* p = malloc(sizeof(int)*4);
        int i;
        for(i = 0;i<4;i++){
                *(p+i) = i;
                }
        printf("*(p+0) = %d\n",*(p+0));
        printf("*(p+1) = %d\n",*(p+1));
          printf("*(p+0) = %d\n",*(p+0));
        printf("*(p+1) = %d\n",*(p+1));
          printf("*(p+0) = %d\n",*(p+0));
        printf("*(p+1) = %d\n",*(p+1));
        free(p);
         printf("*(p+0) = %d\n",*(p+0));
        printf("*(p+1) = %d\n",*(p+1));
       system("pause");
       }
```

11 结构体

*****

```
#include<stdio.h>
#include<stdlib.h>
//c中用来表示一个事务的实体 可以用结构体 类似java的class
//结构体 关键字 struct
//class Student
//{
//   char gender;
//    short age;
//   int score;
        //}
// Student stu = new Student();
//结构体占字节数  ① 大于或等于所有成员的大小总和 ② 是最大的那个成员大小的整数倍
//结构体的指针   struct Student*  类型+"*"
//通过结构体的指针 访问结构体中的变量 (*p).age
//通过间接引用运算符  一级指针+->   p->age
typedef struct Student{
       char gender; //1
       short age;   //2
       int score;   //4
       } stud;
main(){
   stud stu = {'f',19,100};
   printf("stu.age = %hd\n",stu.age);
   printf("stu.score = %d\n",stu.score);
   printf("stu.gender = %c\n",stu.gender);
```

```
   printf("struct Student的大小%d个字节\n",sizeof(stu));

   struct Student* p = &stu;
   struct Student** p2 = &p;
 //  (*p2)->age = 30;
   (**p2).age = 35;
   //(*p).age = 20;
   printf("stu.age = %hd\n",stu.age);
   p->score = 99;
   printf("stu.score = %d\n",stu.score);
       system("pause");

       }
```

12 联合体

```
#include<stdio.h>
#include<stdlib.h>
//联合体 又叫共用体  所有的变量占用同一块内存  所以联合体大小取决于所有变量中最大的那个变量占的字节数
//联合体由于所有的变量占用相同的内存 对不同变量先后赋值 只有最后一次是有意义的
union un{
      int i;
      short j;
      }

main(){
   union un u;
   printf("联合体u占%d个字节\n",sizeof(u));
   u.j = 1234;
   u.i = 12345678;
   printf("u.j = %hd\n",u.j);
       system("pause");
       }
```

13 枚举

```
#include<stdio.h>
#include<stdlib.h>
//enum枚举 定义一系列枚举值 枚举只能从枚举值中取值
//枚举值 默认从0开始 如果修改了就从修改的值开始 依次+1
enum weekday{
     SUN=2,MON,TUE,WENDS,THUS,FRI,STA
     }
main(){
        enum weekday day = MON;
        printf("day = %d\n",day);
       system("pause");

```

```
13.         }
```

14自定义类型

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  //typedef就是给已知的数据类型起别名
4.  typedef int i;
5.  main(){
6.       i j = 123;
7.       printf("j = %d\n",j);
8.         system("pause");
9.
10.        }
```

今日重点 指针和结构体
C的字符串的声明
char* arr;

区别
 指针->可以直接操作内存 内存的分配 和释放是程序员手动控制
java没有指针 不能直接操作内存 java只申请不用管释放

c函数 先声明再使用
c函数思路 丢进去变量的地址 运行之后 通过指针修改变量
java方法 运行之后 产生返回值 用变量接收返回值

面向对象
面向过程

开灯
Class light
int state
turnon
turnoff
light.turnon
light.turnoff
对象调用方法

struct light
{

state = 1;
}

turnon(light*)
turnon(light*)
函数处理数据